МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Индустриальный институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования «Югорский государственный университет»
(ИндИ (филиал) ФГБОУ ВО «ЮГУ»)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Индустриальный институт (филиал)
Федерального государственного бюджетного образовательного учреждения
Высшего образования «Югорский государственный университет»
(ИндИ (филиал) ФГБОУ ВО «ЮГУ»)

Методические указания
по выполнению практических работ

ОП.04 Основы алгоритмизации и программирования
09.02.07 Информационные системы и программирование
(профессионалитет)

I курс, 2 семестр

Нефтеюганск
2024

РАССМОТРЕНО
Предметной цикловой
комиссией МиЕНД
Протокол № 5 от 18.01.2024г.
_____Е.С. Игнатенко

УТВЕРЖДЕНО
Заседанием методсовета
Протокол № 4 от 08.02.2024г.
Старший методист
_____Г.Р.Давлетбаева

Методические указания по выполнению практических работ по дисциплине ОП.04. Основы алгоритмизации и программирования для специальности 09.02.07 Информационные системы и программирование

Организация-разработчик: Индустриальный институт (филиал) федерального государственного бюджетного образовательного учреждения высшего образования «Югорский государственный университет»

Разработчик:
Игнатенко Е.С. – преподаватель ИндИ (филиала) ФГБОУ ВО «ЮГУ»

Содержание

Пояснительная записка

Методические указания по выполнению практических работ студентами составлены в соответствии с рабочей программой учебной дисциплины ОП.04. Основы алгоритмизации и программирования для специальности 09.02.07 Информационные системы и программирование.

Практические работы проводятся после изучения соответствующих разделов и тем учебной дисциплины в соответствии с учебным планом специальности. Целью практических работ является закрепление теоретических знаний и приобретение практических умений и навыков:

- использования базовых алгоритмических структур;
- правил построения алгоритмов;
- описания и использования типов и структур данных языка программирования;
- тестирования и отладки программ.

В результате выполнения практических работ, предусмотренных программой по дисциплине «Основы алгоритмизации и программирования», обучающийся должен:

Уметь:

- разрабатывать алгоритмы для конкретных задач;
- работать в среде программирования;
- реализовывать построенные алгоритмы в виде программ на конкретном языке программирования.
- выполнять оптимизацию и рефакторинг программного кода.
- работать с системой контроля версий.
- использовать выбранную систему контроля версий.
- использовать методы для получения кода с заданной функциональностью и степенью качества.
- анализировать проектную и техническую документацию.
- организовывать постобработку данных.
- приемы работы в системах контроля версий.
- выявлять ошибки в системных компонентах на основе спецификаций.

Знать:

- основные модели алгоритмов;
- методы построения алгоритмов;
- этапы решения задачи на компьютере;
- типы данных языка программирования;
- базовые конструкции изучаемых языков программирования;
- принципы структурного и модульного программирования;
- принципы объектно-ориентированного программирования.
- способы оптимизации и приемы рефакторинга.
- инструментальные средства анализа алгоритма.
- методы организации рефакторинга и оптимизации кода.
- принципы работы с системой контроля версий.
- модели процесса разработки программного обеспечения.
- основные принципы процесса разработки программного обеспечения.

- основные подходы к интегрированию программных модулей.
- основы верификации и аттестации программного обеспечения.
- стандарты качества программной документации.
- основы организации инспектирования и верификации.
- встроенные и основные специализированные инструменты анализа качества программных продуктов.
- методы организации работы в команде разработчиков.

При реализации содержания общеобразовательной учебной дисциплины «Информационные технологии» обязательная нагрузка обучающихся — 172 часа, включая практические занятия. Итоговой формой контроля является **экзамен**. Обучающиеся, которые не выполнили все практические работы до экзамена не допускаются.

Перечень практических работ

№	Наименование практических работ	Кол-во часов
1	**Практическая работа №1.** Построение блок – схем алгоритмов линейной структуры в программе DiagramDesigner	2
2	**Практическая работа №2.** Построение блок – схем алгоритмов разветвляющейся структуры в программе DiagramDesigner	2
3	**Практическая работа №3.** Построение блок – схем алгоритмов циклической структуры в программе DiagramDesigner	2
4	**Практическая работа №4.** Введение в язык программирования Python	2
5	**Практическая работа №5.** Целые числа, ввод-вывод в Python	2
6	**Практическая работа №6.** Математические операции в Python	2
7	**Практическая работа №7.** Работа со строками в Python	2
8	**Практическая работа №8.** Операции со строками в Python	2
9	**Практическая работа №9.** Логический тип данных и операции в Python	2
10	**Практическая работа №10.** Структура ветвление в Python. Условный оператор	2
11	**Практическая работа №11.** Структура ветвление в Python. Вложенный условный оператор и "иначе-если"	2
12	**Практическая работа №12.** Работа с циклами в Python. Цикл While	2
13	**Практическая работа №13.** Работа с циклами в Python. Цикл For	2
14	**Практическая работа №14.** Функции и процедуры в Python	2
15	**Практическая работа №15.** Использование функций и процедур в Python	2
16	**Практическая работа №16.** Работа со списками. Операции над списками в Python	2
17	**Практическая работа №17.** Кортежи в Python	2
18	**Практическая работа №18.** Сортировка. Сравнение списков и кортежей в Python	2
19	**Практическая работа №19.** Работа с массивами и множествами в Python	2
20	**Практическая работа №20.** Словари в Python	2
21	**Практическая работа №21.** Объектно-ориентированное программирование в Python.	2
22	**Практическая работа №22.** Решение комплексных задач в Python	2
Всего		44

Практическая работа №1. Построение блок – схем алгоритмов линейной структуры в программе DiagramDesigner

Цель: научиться составлять линейные алгоритмы.

Задание 1. Составить алгоритм запуска программы Paint в ОС Windows 7.

Решение:

Вспомним из курса информатики 5 класса порядок действий для запуска программы Paint.

1. Войти в меню «Пуск».
2. Войти в пункт «Все программы».
3. Войти в пункт «Стандартные».
4. Выбрать программу «Paint».

Данный алгоритм в виде блок-схемы имеет следующий вид:

Задание 2. Как убить Кощея?

Наверное, все помнят из детства сказку, в которой рассказывается о местонахождении смерти Кощея Бессмертного: «Смерть моя – на конце иглы, которая в яйце, яйцо – в утке, утка – в зайце, заяц в сундуке сидит, сундук на крепкий замок закрыт и закопан под самым большим дубом на острове Буяне, посреди моря-океяна …»

Предположим, вместо Ивана-царевича бороться с Кощеем был брошен Иван-дурак. Давайте поможем Василисе Премудрой составить такой алгоритм, чтобы даже Иван-дурак смог убить Кощея.

1. Конечно же, сначала необходимо разыскать остров Буян (на такие вещи, будем считать, Иван-дурак способен).
2. Поскольку сундук закопан под самым большим дубом, то сначала необходимо найти самый большой дуб на острове.
3. Затем нужно выкопать сам сундук.
4. Прежде чем доставать зайца, необходимо сломать крепкий замок.
5. Теперь уже можно достать зайца.
6. Из зайца нужно достать утку.
7. Из утки достать яйцо.
8. Разбить яйцо и достать иголку.
9. Иголку поломать.

Это тоже линейный алгоритм, хотя и более длинный, чем алгоритм запуска программы Paint.

Его блок-схема выглядит так:

Задание 3. Вычислить периметр треугольника со сторонами a, b, c.

Задание 4. Найдите площадь треугольника по формуле Герона. Здесь a, b, c – это длины сторон, S – площадь треугольника, P – периметр.

Задание 5. Вычислить значение функции y=ax+b

Задание 6. Дана величина A, выражающая объем информации в байтах. Перевести A в более крупные единицы измерения информации. Составьте блок-схему алгоритма решения поставленной задачи.

Задание 7. Даны длины двух катетов прямоугольного треугольника. Определить периметр этого треугольника.

Задание 8. Даны два действительных числа X и Y. Вычислить их сумму, разность, произведение и частное.

Задача 9. Вычислить значение выражения по формуле (каждая формула отдельный алгоритм)

$$R = 3t^2 + 3l^5 + 4.9$$

$$K = \ln(p^2 + y^3) + e^p$$

$$G = n(y + 3.5) + \sqrt{y}$$

$$S = \sqrt{cos4y^2 + 7.151}$$

$$N = 3y^2 + \sqrt{y + 1}$$

$$N = 3y^2 + \sqrt{y^3 + 1}$$

Задача 10. Дана длина L окружности. Найти ее радиус R и площадь S круга, ограниченного этой окружностью, учитывая, что $L = 2\pi R$, а $S = \pi R^2$. В качестве значения π использовать 3.14.

Задание 11. Составить линейный алгоритм вычисление площади квадрата со стороной a

Задние 12. Стороны прямоугольника a и b. Найдите периметр, площадь и диагональ прямоугольника. Для вычисления диагонали использовать формулу D=SQR(a2+b2). P:=2*(a+b), S := a*b

Практическая работа №2. Построение блок – схем алгоритмов разветвляющейся структуры в программе DiagramDesigner

Цель: научиться решать задачи на составление алгоритмов разветвляющей структуры

Разветвление (ветвление, развилка) – это такая структура организации действий в алгоритме, когда в зависимости от выполнения или невыполнения некоторого условия выполняется либо одна, либо другая последовательность действий.

Имеется две формы ветвлений – полная, имеющая две ветви и неполная, имеющая одну ветвь. В каждой из них указывается условие, которое надо проверять, и наборы действий, которые надо исполнять при выполнении или невыполнении условия. Ясно, что проверка условия должна быть допустимым действием исполнителя.

Задание 1. Вычислить выражение $y = \sqrt{x}$ для введенного x с клавиатуры. Если x >=0, тогда выполнить вычисления, в противном случае вывести сообщение «функция не определена».

Разберем решение задачи

На первом этапе определим, что нам известно и что нужно найти

Дано: x – аргумент функции

Вычислить y.

На втором этапе пропишем условие задачи

Если x >=0, тогда y:=SQRT(x) иначе вывести сообщение «функция не определена»

На третьем этапе составим блок – схему

Задание 2. Выбрать максимальное значение из 2x чисел a и b.

Задание 3. Вычислить значение выражений $\frac{a}{b} - \frac{c}{d}$. Если b >0, d>0

Задание 4. Составить блок-схему для решения задачи: дано число *X*. Увеличить его на 10, если оно положительное, во всех остальных случаях уменьшить его на 10.

Задание 5. Составить блок-схему для решения задачи. Дано число *X*. Увеличить его на 5, если оно положительное

Задание 6. Найти наименьшее из трех чисел.

Задание 7. Известны коэффициенты и с квадратного уравнения. Вычислить корни квадратного уравнения.

Входные данные: a, b, c.

Выходные данные: x_1, x_2.

Задание 8. Ввести число. Если оно больше 20, разделить его на 2, еслименьше или равно 20, то умножить на 5.

Задание 9. Ввести рост человека. Вывести на экран сообщение «ВЫСОКИЙ», если рост превышает 180 см, в противном случае вывести сообщение « НЕ ОЧЕНЬ ВЫСОКИЙ».

Задание 10. Вычислить значение выражений $\frac{2x-1}{3x}$. Если x >0

Задача 11. Вычислить значение функции $Y = \begin{cases} 2X, \text{если } X < 0 \\ 2XZ, \text{если } X = 0 \\ Z, \text{если } \sin X > 0 \end{cases}$ (у вас будет проверка условия три раза)

Практическая работа №3. Построение блок – схем алгоритмов циклической структуры в программе DiagramDesigner

Цель: научиться решать задачи на составление алгоритмов циклической структуры

Задание 1. Вывести все точки графика $y=x^2$ на отрезке для X от -5 до 5 с шагом 1 сантиметр.

Задание 2. Составить программу вычисляющие сумму чисел: S = 2 + 4 + 6 + 8 + … + 20

Задание 3. Вывести все точки графика $y=x^2$ на отрезке для X от -5 до 5 с шагом 1 сантиметр.

Задание 4. Составить блок-схему и программу вводящие с клавиатуры целые числа и суммирующие их, до тех пор пока не будет введен 0.

Задание 5. Вывести все точки графика $y=x^2$ на отрезке для X от -5 до 5 с шагом 1 сантиметр.

Задание 6. Рассчитать среднее арифметическое первых 10 чисел рада 3, 9, 15 и т.д.

Задание 7. С клавиатуры вводятся N натуральных чисел. Рассчитать, сумму четных и произведение нечетных чисел. Составить блок – схему цикла с параметром словесному описанию.

Формульно-словесный алгоритм

1) Описание переменных:
   a) *N* – количество вводимых чисел, количество итераций;
   b) *i=1; N; 1* – счетчик введенных чисел, параметр цикла;
   c) *x* – ячейка числа;
   d) *S* – переменная накопления суммы четных чисел;
   e) *P* - переменная накопления произведения нечетных чисел.
2) Для решения задачи на первом этапе с клавиатуры вводится N
3) Задаются все стартовые значения вычисляемых переменных:

S=0, P=1

4) Организуется цикл по параметру *i*
5) Для каждого значения счетчика *i* выполняется тело цикла:
   a) в ячейку *x* с клавиатуры вводится число;
   b) проверяется условие четности числа;
   c) если число четное, накапливается сумма;
   d) иначе - накапливается произведение.
6) Действия повторяются *N* раз
7) Выводятся накопленные значения *S* и *P*.

Задание 8. Подсчитать количество положительных чисел среди 10 чисел введенных с клавиатуры

Задание 9. Даны x и y. Рассчитать P по формуле:

$$P = x^y + \sqrt{2xy} + \sqrt[4]{4xy} + \cdots + \sqrt[10]{10xy}$$

Составить блок – схему цикла с параметром словесному описанию.

Формульно-словесный алгоритм

1) Описание переменных:

x, y – переменные;

i=2; 10; 2 – показатель корня и множитель под корнем, параметр цикла;

P - переменная накопления суммы.

2) Для решения задачи на первом этапе вводятся *x* и *y*
3) Проверяется условие: $x \cdot y \geq 0$

   Расчет может быть проведен, если подкоренное выражение положительно
4) Если условие выполняется, в ячейку накопления суммы записывается первое слагаемое: $P=x^y$
5) Организуется цикл по параметру *i*
6) Для каждого значения *i* выполняется тело цикла:
   a) в ячейку P накапливается сумма по формуле:

$$P = P + \sqrt[i]{i \cdot x \cdot y};$$

7) Действия повторяются
8) После выполнения цикла выводится значение P.
9) Иначе (условие не выполняется) выводится сообщение «Нет решений» и решение останавливается.

Практическая работа №4. Введение в язык программирования Python

Цель: познакомиться с языком программирования Python. Узнать об основных понятиях в программировании. Познакомиться с встроенной функцией printf. Научиться объявлять переменные в Python и работать с ними. Научиться работать с IDLE.

Теоретическая часть и терминология

- Python - *(пайтон)* высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода.
- Алгоритм - набор последовательных шагов для достижения цели.
- Программа - алгоритм, который выполняет компьютер.
- Язык программирования - язык, на котором пишут программы.
- Программирование - процесс создания компьютерных программ с помощью языков программирования.
- Индекс TIOBE - *(тиоб)* топ языков программирования по их популярности в интернете. Рассчитывается на основе поисковых запросов, частоты использования и многих других параметров.
- IDLE - *(айдл)* программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.
- Python Shell - *(пайтон шел)* это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Здесь мы можем писать команды и они будут сразу выполняться.
- Команда - это когда мы просим компьютер сделать что-то конкретное для нас.
- Встроенные функции - функции, уже существующие в Python.
- print() - встроенная функция, позволяющая выводить информацию на экран.
- Переменная - ячейка памяти, имеющая имя и хранящая какое-то значение.
- Тип данных - тип используемой информации в программе.
- Конкатенация - сложение строк.
- Алгоритм - набор последовательных шагов для достижения цели.
- Программа алгоритм, который выполняет компьютер, написанный на языке программирования. Программы бывают очень большие и очень маленькие. Вот сегодня мы с вами как раз напишем свою первую программу на языке программирования Python.
- Язык программирования - язык, на котором пишут программы. По своей сути это язык, на котором мы общаемся с машиной, даём ей команды.
- Программирование - это процесс создания компьютерных программ с помощью языков программирования. Т.е. это именно длительный процесс, когда человек сидит и пишет код на каком-то языке, чтобы потом машина прочитала его и выполнила данные инструкции. Отлично. Основные понятия повторили, теперь давайте перейдём непосредственно к языку Python.

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании. Например, Инстаграм для

того, чтобы показывать правильную рекламу своим пользователям, использует именно этот язык программирования. Он также используется в продуктах компании Microsoft, на нем создаются программы искусственного интеллекта, с его помощью может создаваться серверная часть сайтов.

Язык Python (по-русски можно произносить как Пайтон или Питон) появился в 1991 году и был разработан Гвидо ван Россумом. Язык назван в честь шоу "Летающий цирк Монти Пайтона". Одна из главных целей разработчиков – сделать язык забавным для использования.

Сейчас Питон регулярно входит в десятку наиболее популярных языков и хорошо подходит для решения широкого класса задач: обучение программированию, скрипты для обработки данных, машинное обучение, серверная веб-разработка и многое другое. Большинству из вас язык Питон потребуется для написания проектов и в ряде предметов третьего курса, а также в повседневном быту для автоматизации задач обработки данных.

Ход работы

Создайте в папке документы новую папку с вашей ФИО (сюда мы будем сохранять свои работы)

Открываем папку и внутри неё создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr4,5,6.py, обязательно поставьте расширение .py, это расширение файлов, написанных на языке python, как .html у HTML, или .js у JavaScript файлов (если у вас не показывается расширение файла, то найдите в проводнике (любая папка в windows) вкладку вид -> параметры -> вид -> скрывать расширения для зарегистрированных типов файлов -> снять галочку)

У вас должен появиться подобный файл 

Отлично, файл создали, теперь надо его запустить! Для этого щёлкаем по файлу правой кнопкой мыши и выбираем "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Здесь мы с вами и будем писать код на Python, который впоследствии запустим.

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python. С крупными проектами уже возникнут сложности, т.к. она не идеальна и для Python есть множество других программ, PyCharm, VS Code и многие другие. Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

А теперь давайте попробуем запустить программу! Ничего страшного, что она у нас пустая. Нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем

писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы, об этом чуть позже. 3 знака больше - это приглашение к написанию кода, именно здесь мы и будем давать команды.

А теперь давайте напишем первую команду на языке Python.

Команда - это когда мы просимьсомпьютер сделать что-то конкретное для нас. Попросим его распечатать нам текст "Привет, мир!". Это стандартное приветственное слово для программистов, при изучении любого языка программирования.

В программировании часто случаются ситуации, когда приходится выполнять одни и те же алгоритмы несколько раз. И чтобы не писать каждый раз весь алгоритм заново целиком, придумали функции, это набор алгоритмов. В Python, как и в других языках программирования существуют стандартные встроенные функции, которые мы можем вызывать.

Так вот, чтобы напечатать текст, существует встроенная функция print(). Она печатает текст в нашем Python Shell. Для этого надо написать print, поставить скобки, а в скобках написать сообщение в кавычках. Теперь нажмите enter.

Если вы всё сделали правильно, у вас получится следующий результат

```
>>> print("Привет, мир!")
Привет, мир!
>>>
```

Рисунок 4.1

Синим цветом нам распечаталось сообщение "Привет, мир!". Обратите внимание, что мы написали сообщение в кавычках, а вывелось оно нам без кавычек. Это всё потому, что если мы хотим напечатать слова с помощью функции print, нам нужно заключить их в кавычки.

Сейчас мы разберем с вами основные правила в Python.

Первое: в пайтоне нельзя начинать строку с пробела. И вообще крайне важно следить за отступами. В Python отступы играют решающую роль, с помощью них он отслеживает порядок выполнения действий, поэтому ставьте их очень аккуратно.

Второе: на одной строке может располагаться только одна команда.

Третье: в конце строки не ставится никаких знаков препинания. Т.е. нет точек с запятой, как в JavaScript.

При нарушении этих правил у вас появится красная синтаксическая ошибка, подробно эти ошибки мы с вами рассмотрели в лекции.

А теперь попробуйте решить самостоятельно несколько заданий в этом же файле.

Самостоятельное задание

1. Используя встроенную функцию print(), выведите своё имя и любой факт о себе в Python Shell.

Пример вывода:

Меня зовут Иван, мой любимый вид спорта - футбол.

2. В пустой файл IDLE (оболочка без знаков больше), напишите 2 функции print() таким образом, чтобы получился математический пример с вопросом, а сразу после примера, ответ на него. Затем нажмите Run -> Run module.

Пример вывода:

print("4+23 = ?")

print(4 + 23)

3. В IDLE напишите 3 команды для печати текста из трёх строк "лесенкой". Затем нажмите Run -> Run module.

Пример вывода:

Вот
    такая
        лесенка

Пример ввода:

```
print("Вот")
print("    такая")
print("        лесенка")
```

4. С помощью IDLE и функции print(), напечатайте в Shell такой рисунок.

Рисунок 4.2

При запуске программы ваши ответы будут дублироваться

Мы с вами научились выводить слова с помощью функции print. Однако эта функция может выводить не только слова, но и числа. Для этого нужно написать в скобках необходимое число.

Причём без кавычек. Почему без кавычек и в чём разница, мы с вами выяснили на лекции.

Дробные числа в пайтон пишутся с точкой, не через запятую. Попробуйте напечатать в оболочке следующие примеры

А ещё внутри функции print() можно проводить арифметические операции! Для этого достаточно написать 2 цифры и нужный вам оператор между ними. Арифметические операции мы с вами разобрали на лекции.

Однако, не очень удобно каждый раз печатать функцию print, чтобы что-то посчитать. Вы можете проводить арифметические вычисления прямо в оболочке через арифметические операторы.

Реализуйте следующие примеры

```
>>> print(45)
45
>>> print(-26)
-26
>>> print(3.7)
3.7
>>> print(25+4)
29
>>> print(45/5)
9.0
>>> 25+3
28
>>> 2+3*2
8
>>> (5+5)*2
20
```

Рисунок 4.3

Мы помним, что цифры мы пишем без кавычек, а слова в кавычках. Однако цифры мы тоже можем писать в кавычках, но складывать их уже не сможем. Если мы что-то с вами пишем в кавычках, то оно становится строкой. И при выводе с помощью print строка печатается полностью, без каких-либо арифметических операций.

Далее разберем работу с типами данных

Тип данных - тип используемой информации в программе. В пайтоне он определяется автоматически, в зависимости от значения, которое мы вписали в переменную. С типами данных в пайтоне мы познакомились с вами на лекции.

Вспомним некоторые из них:

- int - сокращение от integer, целое число, может быть как отрицательным, так и положительным, но обязательно целым
- str - сокращение от string, т.е. строка. Это как раз и есть наш тип данных в кавычках, в него можно помещать буквы, цифры, символы. Мы будем использовать его очень часто. При описании типа str нельзя открыть одинарную кавычку, а закрыть двойной. Однако можно писать кавычки внутри кавычек.

Строки как и числа можно складывать, точнее сказать - склеивать. Когда мы складываем 2 строки, они склеиваются и идут по порядку. Поэтому при сложении строк лучше заранее ставить пробел. Сложение строк называется конкатенация.

Реализуйте следующие примеры

```
>>> "Привет"+" мир"
'Привет мир'
>>> print("Меня"+"зовут"+"Иван")
МенязовутИван
>>>
```

Рисунок 4.4

Во втором примере мы не поставили пробелы, поэтому строки просто склеились в одно слово.

Самостоятельное задание

1. Задача 1:

"Артём решил накопить денег на новый смартфон, стоимость которого - 12 775 рублей. Сколько дней понадобится Артёму, чтобы собрать нужную сумму, если он

каждый день будет откладывать по 35 рублей?" Сначала просто посчитайте результат в Shell, а потом выведите ответ предложением через print().

Пример вывода:

Для того, чтобы накопить на новый смартфон, Артёму понадобится ▭ дней.

2. Задача 2:

Сколько дней понадобится Артёму, если у него уже накоплено 2100 рублей? Посчитайте в Python Shell и выведите ответ на экран с помощью print().

3. Задача 3:

На 230 день в магазине случилась акция, на смартфон скидка 20%. Сможет ли Артём его купить? Если нет, то сколько ему будет не хватать? Посчитайте в Python Shell и выведите ответ на экран с помощью print().

4. Задача 4:

С помощью функции print() выведите в Python Shell тип данных для 25, "49", -23, "Автомобиль".

Пример вывода:

25 – это тип данных int

А теперь давайте представим, что нам с вами нужно распечатать фразу "Привет, мир" много раз. Получается нам придётся её копировать и вставлять много раз, хотя фраза "Привет, мир!" ещё короткая, а если бы была длиннее? Решить эту проблему мы можем с помощью переменных.

Мы можем поместить туда эту фразу, а затем просто вызывать её по названию переменной.

Переменная - это ячейка памяти, имеющая имя и хранящая какое-либо значение. В Pascal, например, писали слово var, потом название переменной и т.д. В пайтоне этого делать не нужно, можно сразу написать название переменной, знак равно, т.е. оператор присваивания и её значение. Желательно также, чтобы имя переменной отражало то, что в ней находится.

Реализуйте следующий пример

```
>>> a="Привет, мир"
>>> print(a)
Привет, мир
```

Рисунок 4.5

Для переменных в Python есть свои правила написания, с которыми мы познакомились на лекции. Вам необходимо их запомнить, чтобы в будущем не совершать ошибок.

Реализуйте следующий пример

1. Объявите переменную d и присвойте ей значение 5. Затем вызовите её, чтобы узнать, что внутри и python вам распечатает эту информацию.
2. Тоже самое сделайте и с переменной b, только в неё поместите уже текст Python (текст должен быть в кавычках).
3. И у вас получится 2 переменные.
4. Объявите переменную c со значением 20
5. Сложите числовые переменные
6. В переменную v поместите слово Shell.

7. Создайте еще одну переменную f и поместите сложение переменных b и v. Выведите переменную f

Важное правило - вы не можете сложить переменные разных типов данных.

Также внутри функции print вы можете вызывать сразу много переменных, или другие данные, достаточно их разделить запятой.

Причём, если разделять их запятой, добавляется дополнительный пробел.

Если печатать через знак плюс, то пробела не будет и вам уже нужно позаботиться о нём самостоятельно.

Реализуйте данные примеры у себя в python shell.

```
>>> name="Иван"
>>> print("Добро пожаловать,", name)
Добро пожаловать, Иван
>>> print("Добро пожаловать,"+ name)
Добро пожаловать,Иван
```

Рисунок 4.6

Самостоятельное задание

1. Задача 1:

В зоопарке живут: 5 обезьян (monkeys), 3 льва (lions) и 2 медведя (bears). Сколько всего животных (animals) в зоопарке?

Решите задачу с помощью переменных.

Создайте для каждого животного отдельную переменную по его названию.

Создайте переменную animals для всех животных и внутри неё сложите переменные.

Результат выведите через print(), пользуясь переменной animals.

Пример вывода:

▭ животных живёт в зоопарке

2. Задача 2:

Создайте переменные с вашим именем (name) и возрастом (age) и выведите эти переменные на экран с помощью print, чтобы получилось одно предложение

Пример вывода:

Меня зовут Иван, мне 12 лет.

3. Задача 3:

Пират нашёл сундук с кладом в 300 золотых монет. 30% от него он сразу раздал на долги. Также он оставил сундук открытым без присмотра на некоторое время, прилетели 4 вороны и каждая утащила по 10 монет. Сколько монет осталось у пирата?

Решите через Python Shell, используя переменные, итоговый результат сохраните в переменную coins и выведите её на экран, чтобы получилось:

Ответ: у пирата осталось монет ▭

4. Задача 4:

Создайте в IDLE переменные с текущей датой:

day = 25

month = "января"

year = 2021

Выведите эти переменные через print() в Python Shell

Пример вывода:

Сегодня 25 января 2021 года

5. Задача 5:

Объявите 4 переменные в IDLE, заполните их своими данными.

Составьте предложение с этими переменными и выведите их все через один print() в Python Shell.

6. Задача 6:

Напишите программу с помощью IDLE, переменных и функции print(), напечатайте в Shell изображение одного пингвина размером 5×9 символов.

Для корректного вывода изображения сначала объеяляем в переменных построчно все символы (например: a1 = ' _~_ ', a2 = ' (o o) ' и т.п.), а потом выводим построчно все символы через print (например: print(a1), print(a2) и т.п.)

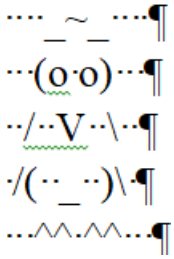


Рисунок 4.7

Примечание. Точки, которые вы видите на картинке – это количество пробелов (включен компонент «отобразить все знаки»)

7. Задача 7:

Придумайте свой рисунок. Напишите программу с помощью IDLE, переменных и функции print(), напечатайте в Shell изображение

Сохраните проекты с помощью горячей комбинации клавиш Ctrl+S.

Сохраните файл из IDLE (он сохраняется автоматически при запуске). У вас он сохранен с именем pr4

Сохраните файл из Shell (он несохраняется автоматически). Сохраните его в отдельный файл с именем pr_Shell 4

Практическая работа №5. Целые числа, ввод-вывод в Python

Цель: Познакомиться с языком программирования Python. Узнать об основных понятиях в программировании. Познакомиться с встроенной функцией printf. Научиться объявлять переменные в Python и работать с ними. Научиться работать с IDLE.

Теоретическая часть и терминология

- Python - *(пайтон)* высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода.
- Алгоритм - набор последовательных шагов для достижения цели.
- Программа - алгоритм, который выполняет компьютер.
- Язык программирования - язык, на котором пишут программы.
- Программирование - процесс создания компьютерных программ с помощью языков программирования.
- Индекс TIOBE - *(тиоб)* топ языков программирования по их популярности в интернете. Рассчитывается на основе поисковых запросов, частоты использования и многих других параметров.
- IDLE - *(айдл)* программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.
- Python Shell - *(пайтон шел)* это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Здесь мы можем писать команды и они будут сразу выполняться.
- Команда - это когда мы просим компьютер сделать что-то конкретное для нас.
- Встроенные функции - функции, уже существующие в Python.
- print() - встроенная функция, позволяющая выводить информацию на экран.
- Переменная - ячейка памяти, имеющая имя и хранящая какое-то значение.
- Тип данных - тип используемой информации в программе.
- Конкатенация - сложение строк.
- Алгоритм - набор последовательных шагов для достижения цели.
- Программа алгоритм, который выполняет компьютер, написанный на языке программирования. Программы бывают очень большие и очень маленькие. Вот сегодня мы с вами как раз напишем свою первую программу на языке программирования Python.
- Язык программирования - язык, на котором пишут программы. По своей сути это язык, на котором мы общаемся с машиной, даём ей команды.
- Программирование - это процесс создания компьютерных программ с помощью языков программирования. Т.е. это именно длительный процесс, когда человек сидит и пишет код на каком-то языке, чтобы потом машина прочитала его и выполнила данные инструкции. Отлично. Основные понятия повторили, теперь давайте перейдём непосредственно к языку Python.

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании. Например, Инстаграм для

того, чтобы показывать правильную рекламу своим пользователям, использует именно этот язык программирования. Он также используется в продуктах компании Microsoft, на нем создаются программы искусственного интеллекта, с его помощью может создаваться серверная часть сайтов.

Язык Python (по-русски можно произносить как Пайтон или Питон) появился в 1991 году и был разработан Гвидо ван Россумом. Язык назван в честь шоу "Летающий цирк Монти Пайтона". Одна из главных целей разработчиков – сделать язык забавным для использования.

Сейчас Питон регулярно входит в десятку наиболее популярных языков и хорошо подходит для решения широкого класса задач: обучение программированию, скрипты для обработки данных, машинное обучение, серверная веб-разработка и многое другое. Большинству из вас язык Питон потребуется для написания проектов и в ряде предметов третьего курса, а также в повседневном быту для автоматизации задач обработки данных.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Сохраните файл из Shell (он несохраняется автоматически). Сохраните его в отдельный файл с именем pr_Shell 5 (File – Save as).

Ход работы

Переменная - это именованный контейнер для заданного значения.

```
# Пример создания переменных
age = 19
name = 'Ivan'
isActive = True
name = 'Petr'
Name = 'Ivan'
# name и Name - это разные переменные
# Имена переменных не могут начинаться с цифры!
```

В Python есть 4 примитивных типа данных:

```
# int (целые числа)
age = 18
# float (дробные числа)
fraction = 2.5
# str (строки)
fruit = 'apple'
# bool (правда или ложь)
isReady = True # всего два значения: True и False
```

Мы можем преобразовывать один тип данных в другой с помощью одноименных функций. Например, число может стать строкой, строка - числом, дробное число - целым.

```
age = '22' # str -> '22'
age = int(age) # int -> 22
age = float(age) # float -> 22.0
age = bool(age) # bool -> True
```

Функция print() выводит данные на экран.

```
name = 'Alexey'
# В скобках записываются переданные в функцию параметры
print(name)
# Вывод
>> Alexey
```

Функция print() может принимать несколько входных параметров.

```
print(1, 2, 3)
# Вывод
>> 1 2 3
```

Каждый print() выводит данные на новой строке. По умолчанию завершающий символ строки равен символу новой строки (\n).

```
print('Hello')
print('world')
# Вывод
```

```
>> Hello
>> world
```

Завершающий символ строки в функции print() можно изменять.

```
print('Hello', end=' ')
print('world')
# Вывод
>> Hello world
```

Функция input() принимает пользовательский ввод данных.

```
name = input()¶print('Hello ' + name)
# После запуска скрипта Python будет ожидать ввода данных
```

Функция input может принимать всего лишь один аргумент - строку, которая выведется перед входной строкой.

```
name = input('Enter your name: ')
print('Hello, ' + name)
```

Данные, полученные с помощью функции input(), имеют строковый тип данных (str).

Строки можно складывать друг с другом, такое сложение называется их конкатенацией или объединением.

```
# Сумма двух строчных чисел
number1 = input('Введите число: ')
number2 = input('Введите число: ')
print(number1 + number2)
# Ввод:
>> 1
>> 2
# Вывод:
>> 12
```

Преобразуем строковый тип в целое число (str -> int).

```
# Исправленная сумма двух чисел
number1 = int(input('Введите число: '))
number2 = int(input('Введите число: '))
print(number1 + number2)
# Ввод:
>> 1
>> 2
# Вывод:
>> 3
```

Примеры

1. Произведение

```
# Произведение двух введенных чисел
a = int(input('Введите число: '))
b = int(input('Введите число: '))
```

```
print(a * b)
# Ввод:
>> 4
>> 3
# Вывод:
>> 12
```

2. Приветствие

```
# Приветствие пользователя по его имени
firstname = input('Введите свое имя: ') # здесь приводить к типу int не нужно
lastname = input('Введите свою фамилию: ')
print('Здравствуйте, ' + firstname + ' ' + lastname) # не забудьте про пробел между словами
# Ввод:
>> Иван
>> Иванов
# Вывод:
>> Здравствуйте, Иван Иванов
```

3. Остаток

```
# Операция % позволяет получить остаток от деления
print(10 % 2) # 0, так как 10 делится на 2 нацело
print(10 % 3) # 1, остаток равен 1
print(10 % 4) # 2, остаток равен 2
# Вывод:
>> 0
>> 1
>> 2
```

4. Деление нацело

```
# Операция // позволяет получить целую часть от деления
print(10 // 2) # 5
print(10 // 3) # 3
print(10 // 4) # 2
# Вывод:
>> 5
>> 3
>> 2
```

Решение задач

1. Сумма трех

Посчитайте сумму трех введенных целых чисел

2. Площадь

Пользователь вводит стороны прямоугольника, выведите его площадь

3. Периметр

Пользователь вводит стороны прямоугольника, выведите его периметр

4. Площадь круга

Пользователь вводит радиус круга, выведите площадь круга

Ввод:

>> 2

Вывод:

>> 12.56

5. Сумма дробных

Посчитайте сумму трех введенных дробных чисел.

Ввод:

>> 1.5

>> 2.5

>> 1.1

Вывод:

>> 5.1

6. Школьники и яблоки

n школьников делят k яблок поровну, неделящийся остаток остается в корзинке. Сколько яблок достанется каждому школьнику? Сколько яблок останется в корзинке?

Ввод:

>> 10

>> 3

Вывод:

>> 3 # каждому

>> 1 # останется

Решение задач

1. Одинаковая четность

Даны два целых числа: A, B. Проверить истинность высказывания: "Числа A и B имеют одинаковую четность".

Ввод:¶

>> 0¶

>> 1¶

Вывод:¶

>> False¶¶

Ввод:¶

>> 2¶

>> 10¶

Вывод:¶

>> True

2. Одно положительное

Даны три целых числа: A, B, C. Проверить истинность высказывания: "Хотя бы одно из чисел A, B, C положительное".

Ввод:¶

>> 0¶

>> -1¶
>> -10¶
Вывод:¶
>> False¶¶
Ввод:¶
>> -1¶
>> 1¶
>> 0¶
Вывод:¶
>> True

3. Последняя цифра

Дано натуральное число. Выведите его последнюю цифру.

Ввод:¶
>> 2345678¶
Вывод:¶
>> 8¶¶
Ввод:¶
>> 19¶
Вывод:¶
>> 9

4. Цифры двузначного

Дано двузначное число. Найдите сумму его цифр.

Ввод:¶
>> 22¶
Вывод:¶
>> 4¶¶
Ввод:¶
>> 99¶
Вывод:¶
>> 18

5. Цифры трехзначного

Дано трехзначное число. Найдите сумму его цифр.

Ввод:¶
>>123¶
Вывод:¶
>> 6¶¶
Ввод:¶
>> 332¶
Вывод:¶
>> 8

6. Разные цифры

Дано трехзначное число. Проверить истинность высказывания: "Все цифры данного числа различны".

Ввод:¶

>> 123¶

Вывод:¶

>> True¶¶

Ввод:¶

>> 332¶

Вывод:¶

>> False

7. Часы (финальный босс)

С начала суток прошло N секунд (N - целое). Найти количество часов, минут и секунд на электронных часах.

Ввод:¶

>> 1000¶

Вывод:¶

>> 0 16 40¶¶

Ввод:¶

>> 10000¶

Вывод:¶

>> 2 46 40¶¶

Ввод:¶

>> 85001¶

Вывод:¶

>> 23 36 41

Практическая работа №6. Математические операции в Python

Цель: Познакомиться с языком программирования Python. Узнать об основных понятиях в программировании. Познакомиться с встроенной функцией input. Научиться объявлять переменные в Python и работать с ними. Научиться работать с IDLE.

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании. Язык Python, благодаря наличию огромного количества библиотек для решения разного рода вычислительных задач, сегодня является конкурентом таким пакетам как Matlab и Octave. Запущенный в интерактивном режиме, он, фактически, превращается в мощный калькулятор. В этом уроке речь пойдет об арифметических операциях, доступных в данном языке.

Комментарий в языке Python

Комментарии используются для вставки в код пояснений к фрагментам программы. Такие пояснения обращены к человеку, читающему код, а не к обрабатывающей данный код ЭВМ.

Знак # превращает в комментарий все символы после него до конца текущей строки.

Пример использования комментария:

Действие 1 # Действие 2 не будет выполнено

Действие 3 # а Действие 3 – будет

Математические операции

Операция – это выполнение каких-либо действий над данными, которые в данном случае именуют **операндами**. Само действие выполняет **оператор** – специальный инструмент. Если бы вы выполняли операцию постройки стола, то вашими операндами были бы доска и гвоздь, а оператором – молоток.

Таблица 7.1

Операция	**Запись**	**Пример**
Сложение чисел x и y	x + y	5 + 2 # = 7
Вычитание числа x из числа y	x – y	5 – 2 # = 3
Умножение числа x на число y	x * y	5 * 2 # = 10
Деление числа x на число y	x / y	5 / 2 # = 2.5
Целочисленное деление (целая часть частного от деления)	x // y	5 // 2 # = 2
Остаток от деления (число, которое не поделилось нацело)	x % y	5 % 2 # = 1
Возведения числа x в степень числа y	x ** y	5 ** 2 # = 25
Смена знака числа x	-x	-5 # = -5

```
a = 2
b = 2.5
c = 5
temp0 = a + b  # складываем 2 и 2.5
temp1 = b - a  # вычитаем из 2.5 цифру 2
```

temp2 = a *b # умножаем a и b

temp3 = c / a # делим c на a

temp4 = c // a # осуществляем целочисленное деление c на a то есть получаем целую часть от деления, дробная отбрасывается автоматически

Бывает необходимо к старому значению прибавить какое-то число (например, счётчик, к старому значению прибавляется новое). Здесь очень помогает применение краткой записи.

Краткая форма Python

a = 1

b = 1

a = a + 1 #обычная запись

b += 1 #краткая форма записи

Кратко можно записывать любую математическую операцию.

При знакомстве с языком программирования Python мы столкнемся с тремя типами данных:

- целые числа (тип int) – положительные и отрицательные целые числа, а также 0 (например, 4, 687, -45, 0).
- числа с плавающей точкой (тип float) – дробные, они же вещественные, числа (например, 1.45, -3.789654, 0.00453). Примечание: для разделения целой и дробной частей здесь используется точка, а не запятая.
- строки (тип str) — набор символов, заключенных в кавычки (например, "ball", "What is your name?", 'dkfjUUv', '6589'). Примечание: кавычки в Python могут быть одинарными или двойными; одиночный символ в кавычках также является строкой, отдельного символьного типа в Питоне нет.

Так в математике и программировании символ плюса является оператором операции сложения по отношению к числам. В случае строк этот же оператор выполняет операцию *конкатенации*, то есть соединения.

Здесь следует для себя отметить, что то, что делает оператор в операции, зависит не только от него, но и от типов данных, которыми он оперирует. Молоток в случае нападения на вас крокодила перестанет играть роль строительного инструмента. Однако в большинстве случаев операторы не универсальны. Например, знак плюса неприменим, если операндами являются, с одной стороны, число, а с другой – строка.

Здесь в строке TypeError: unsupported operand type(s) for +: 'int' and 'str' интерпретатор сообщает, что произошла ошибка типа – неподдерживаемый операнд для типов int и str.

Вывод данных. Функция print()

Что такое функция в программировании, узнаем позже. Пока будем считать, что print() – это такая команда языка Python, которая выводит то, что в ее скобках на экран. С этой функцией мы познакомились ранее.

Ввод данных. Функция input()

За ввод в программу данных с клавиатуры в Python отвечает функция input. Когда вызывается эта функция, программа останавливает свое выполнение и ждет, когда пользователь введет текст. После этого, когда он нажмет Enter, функция input() заберет

введенный текст и передаст его программе, которая уже будет обрабатывать его согласно своим алгоритмам.

Если в интерактивном режиме ввести команду input(), то ничего интересного вы не увидите. Компьютер будет ждать, когда вы что-нибудь введете и нажмете Enter или просто нажмете Enter. Если вы что-то ввели, это сразу же отобразиться на экране.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Сохраните файл из Shell (он несохраняется автоматически). Сохраните его в отдельный файл с именем pr_Shell 7 (File – Save as).

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr7.py, обязательно поставьте расширение .py.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Запускаем программу Run -> Run Module, или просто F5.

Самостоятельное задание. Добавьте строку комментария перед каждым блоком заданий.

Арифметические операции

В программе на языке Python, как и на большинстве других языков, связь между данными и переменными устанавливается с помощью знака =. Такая операция называется присваивание (также говорят "присвоение"). Например, выражение sq = 4 означает, что на объект, представляющий собой число 4, находящееся в определенной области памяти, теперь ссылается переменная sq, и обращаться к этому объекту следует по имени sq.

Чтобы узнать значение, на которое ссылается переменная, находясь в режиме интерпретатора, достаточно ее вызвать, то есть написать имя и нажать Enter.

Реализуйте следующий код в программе

```
>>> sq=4
>>> sq
4
>>>
```

Рисунок 7.1

Реализуйте следующий код в программе

Напишите три переменные: apples, eat_day и day. Каждой из них присваивается свое значение. Выражение apples = apples - eat_day * day сложное. Сначала выполняется подвыражение, стоящее справа от знака равенства. После этого его результат присваивается переменной apples, в результате чего ее старое значение (100) теряется. В подвыражении apples - eat_day * day вместо имен переменных на самом деле используются их значения, то есть числа 100, 5 и 7.

```
>>> apples=100
>>> eat_day=5
>>> day=7
>>> apples = apples - eat_day * day
>>> apples
65
>>>
```

Рисунок 7.2

Реализуйте следующий код в программе

```
>>> a = 2
>>> b = 2.5
>>> c = 5
>>> temp0 = a + b
>>> print("temp0 = ", temp0)
temp0 =  4.5
>>>
```

Рисунок 7.3

Дополните код следующими вычислениями и выведите ответ аналогичным образом:

temp1 = b - a

temp2 = a *b

temp3 = c / a

temp4 = c // a #осуществляем целочисленное деление c на a

Бывает необходимо к старому значению прибавить какое-то число (например, счётчик, к старому значению прибавляется новое). Здесь очень помогает применение краткой записи.

Увеличьте ранее введенные значения переменных a и b как показано в примере.

```
>>> a = a + 1
>>> a
3
>>> b += 1
>>> b
3.5
```

Рисунок 7.4

Самостоятельное задание

1. Переменной var_int присвойте значение 10, var_float - значение 8.4, var_str - "No".
2. Значение, хранимое в переменной var_int, увеличьте в 3.5 раза. Полученный результат запишите в перменную var_big.
3. Измените значение, хранимое в переменной var_float, уменьшив его на единицу, результат свяжите с той же переменной.
4. Разделите var_int на var_float, а затем var_big на var_float. Результат данных выражений привяжите к переменным d1 и d2.
5. Измените значение переменной var_str на "NoNoYesYesYes". При формировании нового значения используйте операции конкатенации (+) и повторения строки (*) >>>var_str = var_str*2+"Yes"*3
6. Выведите значения всех переменных без использования print.

Ввод данных. Функция input()

На прошлом занятии мы познакомились с функцией print() – это такая команда языка Python, которая выводит то, что в ее скобках на экран.

За ввод в программу данных с клавиатуры в Python отвечает функция input. Когда вызывается эта функция, программа останавливает свое выполнение и ждет, когда пользователь введет текст. После этого, когда он нажмет Enter, функция input() заберет введенный текст и передаст его программе, которая уже будет обрабатывать его согласно своим алгоритмам.

Если в интерактивном режиме ввести команду input(), то ничего интересного вы не увидите. Компьютер будет ждать, когда вы что-нибудь введете и нажмете Enter или просто нажмете Enter. Если вы что-то ввели, это сразу же отобразиться на экране.

Реализуйте следующий код в программе

```
>>> input()
Yes
'Yes'
```

Рисунок 7.5

Пояснение. После ввода input() программа ожидает ввода с клавиатуры. Мы написали yes и программа далее нам вывела это слово.

Функция input() передает введенные данные в программу. Их можно присвоить переменной. В этом случае интерпретатор не выводит строку сразу же:

Реализуйте следующий код в программе

```
>>> answer = input()
No, it is not
```

Рисунок 7.6

В данном случае строка сохраняется в переменной *answer*, и при желании мы можем вывести ее значение на экран:

Реализуйте следующий код в программе

```
>>> answer
'No, it is not'
```

Рисунок 7.7

При использовании функции print() кавычки в выводе опускаются:

Реализуйте следующий код в программе

```
>>> print(answer)
No, it is not
```

Рисунок 7.8

Закрепление. Ввод и вывод данных. Функция print().Функция input()

Реализуйте все следующие коды в программе.

Работаем в Shell

```
>>> print("print() - это такая команда языка Python, которая выводит
то, что в ее скобках на экран")
print() - это такая команда языка Python, которая выводит то, что
в ее скобках на экран
```

Рисунок 7.9

```
print(1032)
print(2.34)
print("a:", 1)
one = 1
two = 2
three = 3
print(one, two, three)
```

В print() предусмотрены дополнительные параметры. Например, через параметр **sep** можно

указать отличный от пробела разделитель строк:

```
>>> print("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun", sep="-")
Mon-Tue-Wed-Thu-Fri-Sat-Sun
>>> print(1, 2, 3, sep="//")
1//2//3
```

Рисунок 7.10

Параметр end позволяет указывать, что делать, после вывода строки. По умолчанию происходит переход на новую строку. Однако это действие можно отменить, указав любой другой символ или строку:

```
>>> print(10, end="")
10
```

Рисунок 7.11

Обычно end используется не в интерактивном режиме, а в скриптах, когда несколько выводов подряд надо разделить не переходом на новую строку, а, скажем, запятыми. Сам переход на новую строку обозначается символом '\n'. Если присвоить это значение параметру end, то никаких изменений в работе функции print вы не увидите, так как это значение и так присвоено по умолчанию:

```
>>> print(10, end='\n')
10
```

Рисунок 7.12

Однако, если надо отступить на одну дополнительную строку после вывода, то можно сделать так:

```
>>> print(10, end='\n\n')
10


>>>
```

Рисунок 7.13

В функцию print нередко передаются так называемые форматированные строки, хотя по

смыслу их правильнее называть строки-шаблоны. Никакого отношения к самому print они не имеют. Когда такая строка находится в скобках print(), интерпретатор сначала согласно заданному в ней формату преобразует ее к обычной строке, после чего передает результат в print().

```
>>> pupil = "Ben"
>>> old = 16
>>> grade = 9.2
>>> print("It's %s, %d. Level: %f" % (pupil, old, grade))
It's Ben, 16. Level: 9.200000
```

Рисунок 7.14

Здесь вместо трех комбинаций символов %s, %d, %f подставляются значения переменных pupil, old, grade. Буквы s, d, f обозначают типы данных – строку, целое число, вещественное число.

Если бы требовалось подставить три строки, то во всех случаях использовалось бы сочетание %s.

Хотя в качестве значения переменной grade было указано число 9.2, на экран оно вывелось с дополнительными нулями. Чтобы указать, сколько требуется знаков после запятой, надо перед f поставить точку, после нее указать желаемое количество знаков в дробной части:

```
>>> print("It's %s, %d. Level: %.1f" % (pupil, old, grade))
It's Ben, 16. Level: 9.2
```

Рисунок 7.15

Далее работаем в IDLE

name_user = input()

city_user = input()

print(f'Вас зовут {name_user}. Ваш город {city_user}')

При запуске программы, компьютер ждет, когда будет введена сначала одна строка, потом вторая. Они будут присвоены переменным name_user и city_user. После этого значения этих переменных выводятся на экран с помощью форматированного вывода.

Вышеприведенный скрипт далек от совершенства. Откуда пользователю знать, что хочет от него программа? Чтобы не вводить человека в замешательство, для функции input предусмотрен специальный параметр-приглашение. Это приглашение выводится на экран при вызове input().

Усовершенствуем наш код:

name_user = input('Ваше имя: ')

city_user = input('Ваш город: ')

print(f'Вас зовут {name_user}. Ваш город {city_user}')

Обратите внимание, что в программу поступает строка. Даже если ввести число, функция input() все равно вернет его строковое представление. Но что делать, если надо получить число? Ответ: использовать функции преобразования типов.

Преобразование типов данных

Например, мы хотим сложить строку и число, если сразу это сделать, то программа нам выдаст ошибку. В этом случае нам помощь приходит преобразование типов.

Для изменения одних типов данных в другие в языке Python предусмотрен ряд встроенных в него функций. Поскольку мы пока работаем только с тремя типами (int, float и str), рассмотрим вызовы соответствующих им функций – int(), float(), str().

Преобразуем число 1 в строку и сложим его со строкой ‘a’

```
>>> str(1)+ 'a'
'1a'
```

Рисунок 7.16

преобразуем число, записанное в формате строки, и сложим его с другим числом

```
>>> int('3')+ 7
10
```

Рисунок 7.17

Выполним еще ряд примеров на преобразование типов

```
>>> float('3.2')+ int('2')
5.2
>>> str(4)+ str(1.2)
'41.2'
```

Рисунок 7.18

Эти функции преобразуют то, что помещается в их скобки соответственно в целое число, вещественное число или строку. Однако преобразовать можно не все:

```
>>> int('hi')
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    int('hi')
ValueError: invalid literal for int() with base 10: 'hi'
```

Рисунок 7.19

Здесь возникла ошибка значения (ValueError), так как передан литерал (в данном случае строка с буквенными символами), который нельзя преобразовать к числу с основанием 10.

Однако функция int не такая простая. Чтобы преобразовать дробное число в строковом представлении в целое число, сначала можно использовать функцию float(), затем int().

```
>>> int(float('15.76'))
15
```

Рисунок 7.20

В случае, когда указывается второй аргумент для функции int(), первый всегда должен быть строкой. С помощью второго аргумента сообщается, в какой системе счисления находится число, указанное в строке первого аргумента. Функция int() возвращает его значение в десятичной системе счисления.

```
>>> int('101', 2)
5
>>> int('F', 16)
15
>>> int('12', 8)
10
>>> int('13h', 20)
477
>>> int('111001', 2)
57
```

Рисунок 7.21

Данные могут называться **значениями**, а также **литералами**. Эти три понятия ("данные", "значение", "литерал") не обозначают одно и то же, но близки и нередко употребляются как синонимы. Чтобы понять различие между ними, места их употребления, надо изучить программирование глубже.

Дополним нашу программу в IDLE

```
name_user = input('Ваше имя: ')
city_user = input('Ваш город: ')
print(f'Вас зовут {name_user}. Ваш город {city_user}')
gty = input('Сколько апельсинов вы хотите заказать?')
price = input('Цена одного апельсина: ')
gty = int(gty)
price = float(price)
summa = gty * price
print('Заплатите',summa,'руб.')
```

В данном случае с помощью функций int() и float() строковые значения переменных qty и price преобразуются соответственно в целое число и вещественное число. После этого новые численные значения присваиваются тем же переменным.

Программный код можно сократить, если преобразование типов выполнить в тех же строках кода, где вызывается функция input(). Изменим наш код, а также сделаем так, чтобы ввод данных происходил с новой строки, для это будем использовать служебный символ «\n»:

```
name_user = input('Ваше имя: \n')
city_user = input('Ваш город: \n')
print(f'Вас зовут {name_user}. Ваш город {city_user}')
gty = int(input('Сколько апельсинов вы хотите заказать? \n'))
price = float(input('Цена одного апельсина: \n'))
summa = gty * price
print('Заплатите',summa,'руб.')
```

Решение математических задач в Shell

1. Какими будут значения переменных m и n после выполнения группы операторов? Результат выводим через print.

```
m=25
n=m+1
m=m-25
print(“Ответ =”, n)
```

print(“Ответ =”, m)

2. Каким будет значение n после выполнения группы операторов? Результат выводим через print по аналогии с предыдущим примером.

```
m=20
n=10
m=m/n
n=m*n
n=n+30
```

3. Каким будет значение m после выполнения группы операторов? Результат выводим через print по аналогии с предыдущим примером.

```
m=30
n=2
n=m/2
m=n
m=m+n
m=m
```

4. Какое число будет выведено в качестве ответа после выполнения группы операторов? Результат выводим через print по аналогии с предыдущим примером.

```
m=11
n=m*2
m=m+n
m=m
```

5. Какое значение будет находиться в ячейке a после выполнения группы операторов. Результат выводим через print по аналогии с предыдущим примером.

```
a=26
a=”Папа”
b=a
b=54
b=False
a=b
```

6. Какое значение будет находиться в ячейках c, x, d после выполнения группы операторов. Результат выводим через print по аналогии с предыдущим примером.

```
a=3.333
b=3.332
c=min(a,b)
x=round(c) # округление до целого
d=pow(x,3) # возведение в степень 3 числа x
```

Сохраните все файлы проекта

Сохраните файл из IDLE (он сохраняется автоматически при запуске). У вас он сохранен с именем pr7

Сохраните файл из Shell (он несохраняется автоматически). Сохраните его в отдельный файл с именем pr_Shell 7

Решение заданий в IDLE

1. Создайте новый файл cost.py. Откройте его и мы беуем работать в IDLE

Пусть есть два товара, первый из них стоит A рублей B копеек, а второй - C рублей D копеек. Сколько рублей и копеек стоят эти товары вместе.

В задачах где есть несколько размерностей величин (например, рубли и копейки, километры и метры, часы и минуты) следует переводить все в наименьшую единицу измерения, осуществлять необходимые действия, а затем переводить обратно к нужным единицам.

В нашей задаче наименьшей единицей являются копейки, поэтому все цены следует перевести в них, затем сложить их, а затем перевести результат обратно в рубли и копейки.

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
cost1 = a * 100 + b
cost2 = c * 100 + d
totalCost = cost1 + cost2
print(totalCost // 100, totalCost % 100)
```

Проверьте результат работы программы

2. Создайте новый файл math.py. Откройте его и мы будем работать в IDLE.

Вычислите значение арифметических выражений и выведите на экран результаты вычислений.

$y=\sin(x)$

$c=a!$

$d=b^8$

$s=\text{Log}_{10}t$

$h=x^2+t$

Решение:

```
from math import*
x=float(input('Введите значение x: '))
a=int(input('Введите значение a: '))
b=float(input('Введите значение b: '))
t=float(input('Введите значение t: '))
y=sin(x)
c=factorial(a)
d=pow(b,8)
s=log(t)
h=x**2+t
m=min(x,a,b,t)
print('y=',y)
```

```
print('c=',c)
print('d=',d)
print('s=',s)
print('h=',h)
print('m=',m)
```

Дополните пример следующими примерами:

k=ax+b

$r=8x^3+\cos^2 x$

$g=\sqrt{z}$

$e=Tg(x)*\sqrt{z^4}$

$$n=\sqrt{k-z}+\left(\frac{\sqrt{r}}{t}\right)$$

$$v=\pi r^2$$

$$u=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

При необходимости добавьте нужное количество переменных

3. Создайте новый файл fun.py. Откройте его и мы будем работать в IDLE.

$$y=9x^2+sin^2x\sqrt{a-b}$$

$$z=\sqrt[3]{x^t}\left(ax^3-\frac{x^2}{2!}\right)$$

Вычислите значение арифметических выражений и выведите на экран результаты вычислений.

Исходные данные (используем их для ввода при запуске системы):

x=1.4444

b=0.318

t=2.1

a=1.3

Решение:

```
from math import*
a=float(input('Введите значение a: '))
b=float(input('Введите значение b: '))
x=float(input('Введите значение x: '))
t=float(input('Введите значение t: '))
y=9*x*x+sin(x)*sin(x)*sqrt(a-b)
z1=log(pow(x,t))
z=pow(z1,(1/3))*((a*x*x*x-(x*x)/(1*2)))
print('y=',y)
print('z=',z)
exit(0)
```

Оператор exit(0) вызовет появление окна сообщения, в котором спрашивается о том, хотите ли вы завершить запущенный на выполнение процесс (программу). В случае

положительного ответа программа прекратит свою работу и, кроме того, закроется интерактивная сессия Python.

Проверьте результат работы программы

4. Создайте новый файл geron.py. Откройте его и мы будем работать в IDLE.

Разработайте алгоритм и программу, в которой вычисляется площадь треугольника по трем сторонам. Вычисление проводиться по формуле Герона.

$S = \sqrt{p(p-a)(p-b)(p-c)}$

где S - площадь треугольника;

p=(a+b+c)/2 - полупериметр;

a, b, c - длина стороны треугольника.

Решите задание по аналогии с предыдущим

Проверьте результат работы программы

5. Напишите программу (файл *user.py*), которая запрашивала бы у пользователя:

его имя (например, "What is your name?")

возраст ("How old are you?")

место жительства ("Where are you live?")

После этого выводила бы три строки:

"This is *имя*"

"It is *возраст*"

"(S)he live in *место_жительства*"

Вместо *имя, возраст, место_жительства* должны быть данные, введенные пользователем. Примечание: можно писать фразы на русском языке, но если вы планируете стать профессиональным программистом, привыкайте к английскому.

6. Напишите программу (файл *arithmetic.py*), которая предлагала бы пользователю решить пример 4 * 100 - 54. Потом выводила бы на экран правильный ответ и ответ пользователя.

Подумайте, нужно ли здесь преобразовывать строку в число.

7. Напишите программу (файл *chislo.py*), которая запросит у пользователя четыре числа. Отдельно сложите первые два и отдельно вторые два. Разделите первую сумму на вторую. Выведите результат на экран так, чтобы ответ содержал две цифры после запятой.

Практическая работа №7. Работа со строками в Python

Цель: Познакомиться с языком программирования Python. Узнать об основных понятиях в программировании. Изучить обработку исключений. Научиться работать со строками в Python и работать с ними.

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

Язык Python, благодаря наличию огромного количества библиотек для решения разного рода вычислительных задач, сегодня является конкурентом таким пакетам как Matlab и Octave. Запущенный в интерактивном режиме, он, фактически, превращается в мощный калькулятор. В этом уроке речь пойдет об арифметических операциях, доступных в данном языке.

PEP8 — это документ с рекомендациями, как писать код для языка Python.

Ключевая идея создателя языка, Гвидо Ван Россума, в том, что код программы читается намного больше раз, чем пишется. Рекомендации о стиле написания кода направлены на то, чтобы улучшить читаемость кода и сделать его согласованным между большим числом проектов. В идеале, весь код будет написан в едином стиле, и любой сможет легко его прочесть.

Мы будем давать выдержки из этих рекомендаций постепенно, по мере прохождения курса, для того, чтобы ваш код не только работал, но и соответствовал PEP8.

Одна из первых вещей, о которых говорит PEP8 — это использование пробелов. Избегайте использования пробелов в следующих ситуациях:

- Непосредственно внутри круглых скобок
- Непосредственно перед запятой
- Сразу перед открывающей скобкой, после которой начинается список аргументов при вызове функции
- Не используйте более одного пробела вокруг оператора присваивания (или любого другого) для того, чтобы выровнять его с другим. Рекомендуется всегда окружать операторы одним пробелом с каждой стороны.

Обработка исключений

Исключения (exceptions) происходят синхронно выполнению программы и возникают при появлении аварийной ситуации в ходе исполнения некоторой инструкции. Примерами исключений являются деление на нуль, переполнение, обращение к несуществующему файлу и т. д.

Обработчик ошибок в Python использует блок **try...except...finally.**

Блок **try...except** должен окружать ту часть кода, где может возникнуть исключительная ситуация.

Блок **Finally** всегда исполняется, поэтому в него помещают те инструкции, которые должны выполниться независимо от того, произошло ли исключение.

Программа может прервать свою работу по разным причинам, поэтому типов исключений существует довольно много.

Таблица 8.1

Тип исключения	Описание
IOError	Возникает при появлении ошибок, связанных с операциями ввода/вывода
Index Error	Возникает, если в последовательности не найден элемент с указанным индексом
NameError	Возникает, если не найдено имя переменной, имя функции
SyntaxError	Возникает в случае синтаксической ошибки в программе
TypeError	Возникает, если стандартная операция применяется к объекту несоответствующего типа
ValueError	Возникает, если стандартная операция применяется к объекту соответствующего типа, но с неподходящим значением
ZcroDivisionError	Возникает в случае выполнения операции деления на нуль

Работа со строками в Python

Строки (тип str) — набор символов, заключенных в кавычки (например, "ball", "What is your name?", 'dkfjUUv', '6589'). Примечание: кавычки в Python могут быть одинарными или двойными; одиночный символ в кавычках также является строкой, отдельного символьного типа в Питоне нет.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните

его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Сохраните файл из Shell (он несохраняется автоматически). Сохраните его в отдельный файл с именем pr_Shell 8 (File – Save as).

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr8.1.py, обязательно поставьте расширение .py.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Обработка исключений

Реализуем следующий пример, в комментариях подпишите какой обработчик исключений используется в примере

```
a=float(input('Введите первое число: '))
b=float(input('Введите второе число: '))
try:
        c=a/b
        print('\nЧастное от деления = ',c)
except ZeroDivisionError:
        print('Вы делите на нуль!')
finally:
        print('Давайте запустим программу еще раз или \nНажмите Enter для выхода')
exit(0)
```

Важно: Обратите внимание на отступы. При написании кода отступы ставятся автоматически, но в некоторых случаях отступ нужно убрать. Если будут стоять лишние отступы или вообще не стоять, то программа выдаст ошибку.

Усовершенствуем программу. Дело в том, что наша программа не совсем корректна: ведь пользователь по ошибке может вместо чисел ввести обычные символы, расположенные на клавиатуре. Исправим код чтобы он мог перехватит другую ошибку, связанную как раз с неверным форматом ввода.

Во-первых, оператор try разместим в том месте, где возможно возникновение подобной ошибки - это инструкции ввода данных.

Во-вторых, воспользуемся тем, что обработка нескольких исключений может быть перехвачена с помощью нескольких вложений конструкции except, и включим в наш код исключение ValueError, которое возникает, если стандартная операция применяется к объекту соответствующего типа, но с неподходящим значением.

```
try:
        a=float(input('Введите первое число: '))
        b=float(input('Введите второе число: '))
        c=a/b
        print('\nЧастное от деления = ',c)
except ZeroDivisionError:
```

```
        print('Вы делите на нуль!')
except ValueError:
        print('Вы ввели не числовое значение!')
finally:
        print('Давайте запустим программу еще раз')
        input('\nНажмите Enter для выхода')
exit(0)
```

Работа со строками в Python

Строка (str) — это набор символов Юникод. Для определения строки мы используем одинарные или двойные кавычки.

Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Строкой мы считаем все, что находится внутри кавычек: даже если это пробел, один символ или вообще отсутствие символов.

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr8.2.py, обязательно поставьте расширение .py.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Запускаем программу Run -> Run Module, или просто F5.

Реализуйте весь следующий код в Shell

```
>>> print("Dracarys!")
Dracarys!
>>>
```

```
>>> a='c'
>>> print(type(a))
<class 'str'>
>>> b='hello'
>>> print(type(b))
<class 'str'>
>>> c='hello python'
>>> print(type(c))
<class 'str'>
>>>
```

Рисунок 8.1

Литералы строк

Работа со строками в Python очень удобна. Существует несколько литералов строк, которые мы сейчас и рассмотрим.

Строки в апострофах и в кавычках

Теперь представьте, что вы хотите напечатать строчку *Dragon's mother*. Апостроф перед буквой **s** — это такой же символ, как одинарная кавычка. Попробуем:

```
>>> print('Dragon's mother')
...
SyntaxError: unterminated string literal (detected at line 1)
>>>
```

Рисунок 8.2

Такая программа не будет работать. С точки зрения Python строчка началась с одинарной кавычки, а потом закончилась после слова **dragon**. Дальше были символы s mother без кавычек — значит, это не строка. А потом была одна открывающая строку кавычка, которая так и не закрылась: '). Этот код содержит синтаксическую ошибку — это видно даже по тому, как подсвечен код.

Чтобы избежать этой ошибки, мы используем двойные кавычки. Такой вариант программы сработает верно:

```
>>> print("Dragon's mother")
...
    Dragon's mother
>>>
```

Рисунок 8.3

Теперь интерпретатор знает, что строка началась с двойной кавычки и закончиться должна тоже на двойной кавычке. А одинарная кавычка внутри стала частью строки.

Верно и обратное. Если внутри строки мы хотим использовать двойные кавычки, то саму строку надо делать в одинарных. Причем количество кавычек внутри самой строки неважно.

```
>>> S = 'spam"s'
>>> S
    'spam"s'
>>> S = "spam's"
>>> S
    "spam's"
>>>
```

Рисунок 8.4

Строки в апострофах и в кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы позволить вставлять в литералы строк символы кавычек или апострофов, не используя экранирование.

Экранированные последовательности - служебные символы

Экранированные последовательности позволяют вставить символы, которые сложно ввести с клавиатуры.

Таблица 8.2

Экранированная последовательность	**Назначение**
\n	Перевод строки
\a	Звонок
\b	Забой
\f	Перевод страницы
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\\	Выводит обратный слэш
\'	Выводит одиночную кавычку
\"	Выводит двойную кавычку

Теперь представим, что мы хотим создать такую строку:

Dragon's mother said "No"

В ней есть и одинарные, и двойные кавычки. Нам нужно каким-то образом указать интерпретатору, что кавычки — это один из символов внутри строки, а не начало или конец строки.

Для этого используют символ экранирования: \ — обратный слэш. Если мы поставим \ перед кавычкой (одинарной или двойной), то интерпретатор распознает кавычку как обычный символ внутри строки, а не начало или конец строки:

```
>>> print("Dragon's mother said \"No\"")# Экранируем кавычки вокруг No, чтобы интерпретатор
    распознал их как часть строки
...
    Dragon's mother said "No"
```

Рисунок 8.5

Обратите внимание, что в примере выше нам не пришлось экранировать одинарную кавычку (апостроф 's), потому что сама строка создана с двойными кавычками. Если бы строка создавалась с одинарными кавычками, то символ экранирования нужен был бы перед апострофом, но не перед двойными кавычками.

Если нужно вывести сам обратный слеш, то работает такое же правило. Как и любой другой специальный символ, его надо экранировать:

```
>>> print("\\")
...
    \
>>>
```

Рисунок 8.6

Экранированные последовательности

Мы хотим показать вот такой диалог:

- Are you hungry?

- Aaaarrrgh!

Попробуем вывести на экран строку с таким текстом:

```
>>> print("- Are you hungry?- Aaaarrrgh!")
...
    - Are you hungry?- Aaaarrrgh!
>>>
```

Рисунок 8.7

Как видите, результат получился не такой, как мы хотели. Строки расположились друг за другом, а не одна ниже другой. Нам нужно как-то сказать интерпретатору «нажать на Enter» — сделать перевод строки после вопросительного знака. Это можно сделать с помощью символа \n:

```
>>> print("- Are you hungry?\n- Aaaarrrgh!")
...
    - Are you hungry?
    - Aaaarrrgh!
>>>
```

Рисунок 8.8

\n — это пример **экранированной последовательности** (escape sequence). Такие последовательности еще называют управляющими конструкциями. Их нельзя увидеть в том же виде, в котором их набрали.

Набирая текст в Word, вы нажимаете на Enter в конце строчки. Редактор при этом ставит в конец строчки специальный невидимый символ, который называется LINE FEED (LF, перевод строчки).

Распознать такую управляющую конструкцию в тексте можно по символу \. Программисты часто используют перевод строки \n, чтобы правильно форматировать текст.

Например, напишем такой код:

print("Gregor Clegane\nDunsen\nPolliver\nChiswyck")

Тогда на экран выведется:

Gregor Clegane

Dunsen

Polliver

Chiswyck

Когда работаете с символом перевода, учитывайте следующие моменты:

1. *Не важно, что стоит перед или после \n: символ или пустая строка. Перевод обнаружится и выполнится в любом случае*
2. *Строка может содержать только \n:*

print('Gregor Clegane') # Строка с текстом

print("\n") # Строка с невидимыми символов перевода строки

print('Dunsen') # Строка с текстом

Программа выведет на экран:

Gregor Clegane

Dunsen

3. *В коде последовательность \n выглядит как два символа, но с точки зрения интерпретатора — это один специальный символ*
4. *Если нужно вывести \n как текст (два отдельных печатных символа), то можно воспользоваться экранированием — добавить еще один \ в начале. Последовательность \\n отобразится как символы \ и n, которые идут друг за другом:*

print("Joffrey loves using \\n")

 # => Joffrey loves using \n

Самостоятельное задание

Напишите программу, которая выводит на экран:

- Did Joffrey agree?

- He did. He also said "I love using \n".

При этом программа использует только один print(), но результат на экране должен выводиться в две строчки, как показано выше.

"Сырые" строки - подавляют экранирование

Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования отключается.

```
>>> S = r'C:\newt.txt'
>>> S
'C:\\newt.txt'
```

Рисунок 8.9

Но, несмотря на назначение, "сырая" строка не может заканчиваться символом обратного слэша. Пути решения:

```
>>> S = r'\n\n\\'[:-1]
>>> S
'\\n\\n\\'
>>> S = r'\n\n' + '\\'
>>> S
'\\n\\n\\'
>>> S = '\\n\\n'
>>> S
'\\n\\n'
>>>
```

Рисунок 8.10

Строки в тройных апострофах или кавычках

Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд.

```
>>> c = '''это очень большая
... строка, многострочный
... блок текста'''
>>> c
'это очень большая\nстрока, многострочный\nблок текста'
>>> print(c)
это очень большая
строка, многострочный
блок текста
>>>
```

Рисунок 8.11

Базовые операции

Конкатенация (сложение)

```
>>> S1 = 'spam'
>>> S2 = 'eggs'
>>> print(S1 + S2)
spameggs
>>> var_1 = 'Привет,'
>>> var_2 = 'Python!'
>>> print(var_1 + ' ' + var_2)
Привет, Python!
>>>
```

Рисунок 8.12

Дублирование строки

```
>>> print('spam' * 3)
spamspamspam
>>> var_1 = 'O_o '
>>> var_2 = 10
>>> print(var_1 * var_2)
O_o O_o O_o O_o O_o O_o O_o O_o O_o O_o
```

Рисунок 8.13

Длина строки (функция len)

Функция len() возвращает количество символов в строке.

```
>>> len('spam')
4
```

Рисунок 8.14

Функция ord() возвращает числовое значение символа, при чём, как для кодировки ASCII, так и для UNICODE.

```
>>> print('A', ord('A'))
A 65
>>> print('А русская', ord('А'))
А русская 1040
>>> print('~', ord('~'))
~ 126
>>> print('1', ord('1'))  # Обратите внимание, что здесь единица является именно строкой
1 49
```

Рисунок 8.15

Функция chr(n) возвращает символьное значение для данного целого числа, то есть выполняет действие обратное ord().

Функция str() возвращает строковое представление объекта. (Задание делаем в IDLE)

```
from math import inf, e, pi
print(str(...))
print(str(inf))
print(str(e))
print(str(pi))
print(str(10 + 11))
print(str(10 + 11j))
print(str(None))
print(str(1 == 1))
```

Рисунок 8.16

Доступ по индексу

```
>>> S = 'spam'
>>> S[0]
's'
>>> S[2]
'a'
>>> S[-2]
'a'
>>>
```

Как видно из примера, в Python возможен и доступ по отрицательному индексу, при этом отсчет идет от конца строки.

Извлечение среза

Оператор извлечения среза: [X:Y]. X – начало среза, а Y – окончание; символ с номером Y в срез не входит. По умолчанию первый индекс равен 0, а второй - длине строки.

```
>>> S = 'spam'
>>> S[0]
's'
>>> S[2]
'a'
>>> S[-2]
'a'
>>> s = 'spameggs'
>>> s[3:5]
'me'
>>> s[2:-2]
'ameg'
>>> s[:6]
'spameg'
>>> s[1:]
'pameggs'
>>> s[:]
'spameggs'
>>>
```

Рисунок 8.17

Кроме того, можно задать шаг, с которым нужно извлекать срез.

```
>>> s[::-1]
'sggemaps'
>>> s[3:5:-1]
''
>>> s[2::2]
'aeg'
>>>
```

Рисунок 8.18

Форматирование строки

В Python есть функция форматирования строки, которая официально названа литералом отформатированной строки, но обычно упоминается как **f-string.**

Главной особенностью этой функции является возможность подстановки значения переменной в строку.

Чтобы это сделать с помощью f-строки необходимо:

1. Указать f или F перед кавычками строки (что сообщит интерпретатору, что это f-строка).
2. В любом месте внутри строки вставить имя переменной в фигурных скобках ({ }).

```
>>> var = 10 * 254 // 77 % 5 ** 5
>>> var_2 = f'Вот что получилось {var}'
>>> print(var_2)
Вот что получилось 32
>>>
```

Рисунок 8.19

Изменение строк

Тип данных строка в Python относится к неизменяемым (immutable), но это почти не влияет на удобство их использования, ведь можно создать изменённую копию. Для этого есть два возможных пути:

Использовать перезапись значения переменной

```
>>> var = 'Что-то '
>>> var += 'и что-то ещё'
>>> print(var)
Что-то и что-то ещё
>>> var = var[:16] + 'новое'
>>> print(var)
Что-то и что-то новое
>>>
```

Рисунок 8.20

Использовать встроенный метод replace(x, y):

```
>>> var = 'Что-то '
>>> print(var.replace('то ', 'нибудь'))
Что-нибудь
>>> print(var)
Что-то
>>>
```

Рисунок 8.21

Метод replace(x, y) не меняет строку, а возвращает изменённую копию.

Изменение регистра строки

Если Вам надо изменить регистр строки, удобно использовать один из следующих методов (символы при использовании этих методов не изменяются):

– **capitalize()** переводит первую букву строки в верхний регистр, остальные в нижний.

– **lower()** преобразует все буквенные символы в строчные.
– **swapcase()** меняет регистр на противоположный.
– **title()** преобразует первые буквы всех слов в заглавные
– **upper()** преобразует все буквенные символы в заглавные.

```
>>> var = 'abracadabra'
>>> print(var)
abracadabra
>>> print(var.capitalize())
Abracadabra
>>> var = 'AbRaCAdAbRa'
>>> print(var.lower())
abracadabra
>>> print(var.swapcase())
aBrAcaDaBrA
>>> var = 'я пРосТо ПРИМЕР'
>>> print(var.title())
Я Просто Пример
>>> var = 'я @странный _ПРИМЕР'
>>> print(var.title())
Я @Странный _Пример
>>> print(var)
я @странный _ПРИМЕР
>>> print(var.upper())
Я @СТРАННЫЙ _ПРИМЕР
>>>
```

Рисунок 8.22

Практическая работа №8. Операции со строками в Python

Цель: Познакомиться с языком программирования Python. Узнать об основных понятиях в программировании. Научиться работать со строками в Python и работать с ними.

Теоретическая часть и терминология

Строки в Python - упорядоченные неизменяемые последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Работа со строками в Python

Строки (тип str) — набор символов, заключенных в кавычки (например, "ball", "What is your name?", 'dkfjUUv', '6589'). Примечание: кавычки в Python могут быть одинарными или двойными; одиночный символ в кавычках также является строкой, отдельного символьного типа в Питоне нет.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Сохраните файл из Shell (он несохраняется автоматически). Сохраните его в отдельный файл с именем pr_Shell 9 (File – Save as).

Последовательности в Python

Последовательность(Sequence Type) — итерируемый контейнер, к элементам которого есть эффективный доступ с использованием целочисленных индексов.

Последовательности могут быть как изменяемыми, так и неизменяемыми. Размерность и состав созданной однажды неизменяемой последовательности не может меняться, вместо этого обычно создаётся новая последовательность.

Примеры последовательностей в стандартной библиотеке Python:

1. Список (list) - изменяемая
2. Кортеж (tuple) - неизменяемая
3. Диапазон (range) - неизменяемая
4. Строка (str, unicode) - неизменяемая

Строки можно создать несколькими способами:

1. С помощью одинарных и двойных кавычек.

Например:

Python

```
first_string = 'Я текст в одинарных кавычках'
second_string = "Я текст в двойных кавычках"
```

Строки в одинарных и двойных кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы позволить вставлять в строки символы кавычек, не используя экранирование. Например вот так(обратите внимание на кавычки внутри строки):

Python

```
first_string = 'Слово "Python" обычно подразумевает змею'
second_string = "I'm learning Python"
```

2. С помощью тройных кавычек.

Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд. Пример:

Python

```
my_string = '''Это очень длинная
строка, ей нужно
много места'''
```

3. С помощью метода **str()**.

Как это работает:

Python

```
my_num = 12345
my_str = str(my_num)
```

В данном случае мы создали новую строку путем конвертации переменной другого типа(например, **int**).

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr9.py, обязательно поставьте расширение .py.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Теперь посмотрим, как каждая из них работает:

Python

```
# Обычная строка
>>> str = 'Моя строка вот такая'
>>> print(str)
Моя строка вот такая
# Добавим символ переноса строки
>>> str = 'Моя строка\n вот такая'
>>> print(str)
Моя строка
 вот такая
# А теперь добавим возврат каретки
>>> str = 'Моя строка\n вот\r такая'
>>> print(str)
Моя строка
 такая
# Горизонтальная табуляция(добавит отступ)
>>> str = '\tМоя строка вот такая'
>>> print(str)
    Моя строка вот такая
# Вертикальная табуляция(добавит пустую строку)
>>> str = '\vМоя строка вот такая'
>>> print(str)
Моя строка вот такая
# Добавим китайский иероглиф в строку
>>> str = 'Моя строка \u45b2 вот такая'
>>> print(str)
Моя строка 蚲 вот такая
```

Оператор +

Оператор + объединяет строки. Он возвращает строку, состоящую из операндов, соединенных вместе, как показано здесь:

```
>>> s = 'foo'
>>> t = 'bar'
>>> u = 'baz'
>>> s + t
'foobar'
>>> s + t + u
'foobarbaz'
>>> print('Go team' + '!!!')
Go team!!!
```

Оператор *

Оператор * создает несколько копий строки. Если s-строка, а n-целое число, то любое из следующих выражений возвращает строку, состоящую из n сцепленных копий s:

s * n

n * s

Вот примеры обеих форм:

```
>>> s = 'foo.'
>>> s * 4
'foo.foo.foo.foo.'
>>> 4 * s
'foo.foo.foo.foo.'
```

Операнд множителя n должен быть целым числом. Вы могли бы подумать, что это должно быть положительное целое число, но забавно, что оно может быть нулевым или отрицательным, и в этом случае результатом будет пустая строка:

```
>>> 'foo' * -8
''
```

Если бы вы создали строковую переменную и инициализировали ее пустой строкой, присвоив ей значение 'foo' * -8, любой справедливо подумал бы, что вы немного глупы. Но это сработает.

Оператор in

Python также предоставляет оператор членства, который можно использовать со строками. Оператор in возвращает True, если первый операнд содержится во втором, и False в противном случае:

```
>>> s = 'foo'
>>> s in 'That\'s food for thought.'
True
>>> s in 'That\'s good for now.'
False
```

Существует также оператор not in, который делает обратное:

```
>>> 'z' not in 'abc'
True
>>> 'z' not in 'xyz'
False
```

Встроенные строковые функции

Как вы видели в уроке по основным типам данных в Python, Python предоставляет множество функций, встроенных в интерпретатор и всегда доступных. Вот некоторые из них, которые работают со строками:

Таблица 9.1

Функция	Описание
chr()	Преобразует целое число в символ
ord()	Преобразует символ в целое число
len()	Возвращает длину строки
str()	Возвращает строковое представление объекта

ord(c)

Возвращает целочисленное значение для данного символа.

На самом базовом уровне компьютеры хранят всю информацию в виде чисел. Для представления символьных данных используется схема перевода, которая сопоставляет каждый символ с его репрезентативным номером.

Самая простая схема в обычном использовании называется ASCII. Он охватывает общие латинские символы, с которыми вы, вероятно, больше всего привыкли работать. Для этих символов ord(c) возвращает значение ASCII для символа c:

```
>>> ord('a')
97
>>> ord('#')
35
```

Символы ASCII[એ] встречаются довольно часто. Но в мире существует множество различных языков и бесчисленное множество символов и глифов, которые появляются в цифровых медиа. Полный набор символов, которые потенциально могут потребоваться для представления в компьютерном коде, намного превосходит обычные латинские буквы, цифры и символы, которые вы обычно видите.

Unicode — это амбициозный стандарт, который пытается предоставить числовой код для каждого возможного символа, на каждом возможном языке, на каждой возможной платформе. Python 3 широко поддерживает Юникод, включая разрешение символов Юникода в строках.

Пока вы остаетесь в области общих символов, существует небольшая практическая разница между ASCII и Unicode. Но функция ord() также возвращает числовые значения для символов Юникода:

```
>>> ord('€')
8364
>>> ord('∑')
8721
```

chr(n)

Возвращает символьное значение для данного целого числа.

chr() делает обратное ord(). При заданном числовом значении n chr(n) возвращает строку, представляющую символ, соответствующий n:

```
>>> chr(97)
'a'
>>> chr(35)
'#'
```

chr() также обрабатывает символы Юникода:

```
>>> chr(8364)
'€'
>>> chr(8721)
'∑'
```

len(s)

Возвращает длину строки.

С помощью функции len() вы можете проверить длину строки Python. len(s) возвращает количество символов в s:

```
>>> s = 'I am a string.'
>>> len(s)
14
```

str(obj)

Возвращает строковое представление объекта.

Практически любой объект в Python может быть представлен в виде строки. str(obj) возвращает строковое представление объекта obj:

```
>>> str(49.2)
'49.2'
>>> str(3+4j)
'(3+4j)'
>>> str(3 + 29)
'32'
>>> str('foo')
'foo'
```

Индексация Строк

Часто в языках программирования отдельные элементы в упорядоченном наборе данных могут быть доступны непосредственно с помощью числового индекса или ключевого значения. Этот процесс называется индексированием.

В Python строки-это упорядоченные последовательности символьных данных, и поэтому их можно индексировать таким образом. Доступ к отдельным символам в строке можно получить, указав имя строки, за которым следует число в квадратных скобках ([]).

Индексация строк в Python основана на нуле: первый символ в строке имеет индекс 0, следующий-индекс 1 и так далее. Индекс последнего символа будет равен длине строки минус единица.

Например, схематическое представление индексов строки 'foobar' будет выглядеть следующим образом:

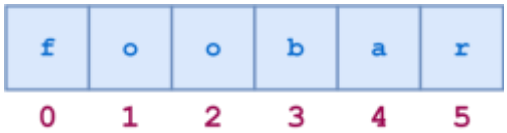


Рисунок 9.1

Отдельные символы могут быть доступны по индексу следующим образом:

```
>>> s = 'foobar'
>>> s[0]
'f'
>>> s[1]
'o'
>>> s[3]
'b'
>>> len(s)
6
>>> s[len(s)-1]
'r'
```

Попытка индексировать за пределами конца строки приводит к ошибке:

```
>>> s[6]
Traceback (most recent call last):
File " <pyshell#17>", line 1, in <module>
s[6]
IndexError: string index out of range
```

Строковые индексы также могут быть заданы отрицательными числами, и в этом случае индексация происходит от конца строки назад: -1 относится к последнему символу, -2-к предпоследнему символу и т. д. Вот та же самая диаграмма, показывающая как положительные, так и отрицательные индексы в строке 'foobar':

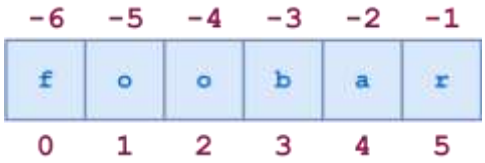


Рисунок 9.2

Вот несколько примеров негативной индексации:

```
>>> s = 'foobar'
>>> s[-1]
'r'
>>> s[-2]
'a'
>>> len(s)
6
>>> s[-len(s)]
'f'
```

Попытка индексирования с отрицательными числами за пределами начала строки приводит к ошибке:

```
>>> s[-7]
Traceback (most recent call last):
File "<pyshell#26>", line 1, in <module>
s[-7]
IndexError: string index out of range
```

Для любой непустой строки s, s[len(s)-1] и s[-1] оба возвращают последний символ. Нет никакого индекса, который имеет смысл для пустой строки.

Нарезка строк

Python также допускает форму синтаксиса индексирования, которая извлекает подстроки из строки, известную как нарезка строк. Если s является строкой, то выражение вида [m:n] возвращает часть s, начинающуюся с позиции m и вплоть до позиции n, но не включая ее:

```
>>> s = 'foobar'
>>> s[2:5]
'oba'
```

Помните: строковые индексы основаны на нуле. Первый символ в строке имеет индекс 0. Это относится как к стандартному индексированию, так и к нарезке.

Опять же, второй индекс указывает первый символ, который не включен в результат—символ 'r' (s[5]) в приведенном выше примере. Это может показаться немного неинтуитивным, но это приводит к такому результату, который имеет смысл: выражение s[m:n] вернет подстроку длиной n - m символов, в данном случае 5 - 2 = 3.

Если вы опустите первый индекс, срез начнется в начале строки. Таким образом, s[:m] и s[0:m] эквивалентны:

```
>>> s = 'foobar'
>>> s[:4]
'foob'
>>> s[0:4]
'foob'
```

Аналогично, если вы опустите второй индекс, как в s[n:], срез простирается от первого индекса до конца строки. Это хорошая, лаконичная альтернатива более громоздкому s[n:len(s)]:

```
>>> s = 'foobar'
>>> s[2:]
'obar'
>>> s[2:len(s)]
'obar'
```

Для любых строк и любого целого числа n (0 = n = len(s)), s[:n] + s[n:] будет равно s:

```
>>> s = 'foobar'
>>> s[:4] + s[4:]
'foobar'
>>> s[:4] + s[4:] == s
True
```

Опущение обоих индексов возвращает исходную строку целиком. Буквально. Это не копия, а ссылка на исходную строку:

```
>>> s = 'foobar'
>>> t = s[:]
>>> id(s)
59598496
>>> id(t)
59598496
>>> s is t
True
```

Если первый индекс в срезе больше или равен второму индексу, Python возвращает пустую строку. Это еще один запутанный способ сгенерировать пустую строку, если вы ее искали:

```
>>> s[2:2]
''
>>> s[4:2]
```

"

Отрицательные индексы также могут быть использованы при нарезке. -1 относится к последнему символу, -2-к предпоследнему и так далее, как и в случае простого индексирования. На приведенной ниже диаграмме показано, как срезать подстроку "oob"из строки "foobar", используя как положительные, так и отрицательные индексы:

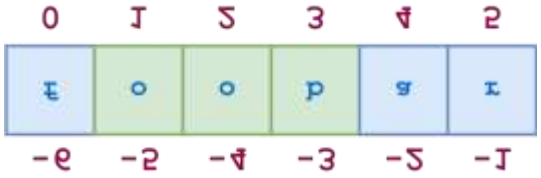


Рисунок 9.3

Вот соответствующий код Python:

```
>>> s = 'foobar'
>>> s[-5:-2]
'oob'
>>> s[1:4]
'oob'
>>> s[-5:-2] == s[1:4]
True
```

Указание шага в срезе строки

Существует еще один вариант синтаксиса нарезки, который следует обсудить. Добавление дополнительного : и третий индекс обозначает шаг, который указывает, сколько символов нужно перепрыгнуть после извлечения каждого символа в срезе.

Например, для строки 'foobar' фрагмент 0:6:2 начинается с первого символа и заканчивается последним символом (целой строкой), а каждый второй символ пропускается. Это показано на следующей диаграмме:

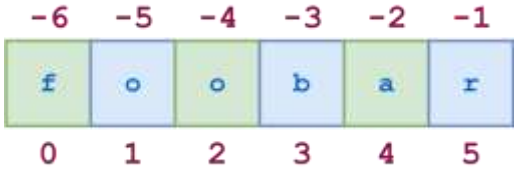


Рисунок 9.4

Аналогично, 1:6:2 задает срез, начинающийся со второго символа (индекс 1) и заканчивающийся последним символом, и снова значение шага 2 вызывает пропуск всех остальных символов:

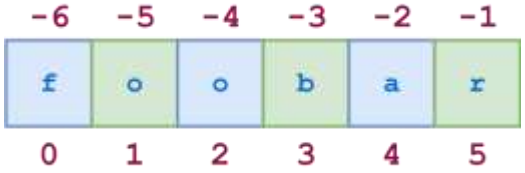


Рисунок 9.5

Иллюстративный REPL код показан здесь:

```
>>> s = 'foobar'
>>> s[0:6:2]
'foa'
>>> s[1:6:2]
```

'obr'

Как и при любом разрезании, первый и второй индексы могут быть опущены, а по умолчанию используются соответственно первый и последний символы:

```
>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::5]
'11111'
>>> s[4::5]
'55555'
```

Вы также можете указать отрицательное значение шага, и в этом случае Python делает шаг назад через строку. В этом случае начальный/первый индекс должен быть больше конечного/второго индекса:

```
>>> s = 'foobar'
>>> s[5:0:-2]
'rbo'
```

В приведенном выше примере 5:0:-2 означает “начать с последнего символа и отступить назад на 2, вплоть до первого символа, но не включая его.”

Когда вы отступаете назад, если первый и второй индексы опущены, значения по умолчанию меняются интуитивно: первый индекс по умолчанию находится в конце строки, а второй индекс по умолчанию-в начале. Вот вам пример:

```
>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::-5]
'55555'
```

Это обычная парадигма для обращения строки вспять:

```
>>> s = 'If Comrade Napoleon says it, it must be right.'
>>> s[::-1]
'.thgir eb tsum ti ,ti syas noelopaN edarmoC fI'
```

Интерполяция переменных в строку

В Python версии 3.6 был введен новый механизм форматирования строк. Эта функция формально называется форматированным строковым литералом, но чаще всего упоминается под псевдонимом **f-string**.

Одна простая функция вне строк, которую вы можете начать использовать сразу же,-это переменная интерполяция. Вы можете указать имя переменной непосредственно в литерале f-string, и Python заменит это имя соответствующим значением.

Например, предположим, что вы хотите отобразить результат арифметического вычисления. Это можно сделать с помощью простого оператора print(), разделяющего числовые значения и строковые литералы запятыми:

```
>>> n = 20
>>> m = 25
>>> prod = n * m
```

```
>>> print('The product of', n, 'and', m, 'is', prod)
The product of 20 and 25 is 500
```

Но это слишком громоздко. Чтобы сделать то же самое, используйте f-строку:

- Укажите либо нижний регистр f либо верхний регистр F непосредственно перед открывающей кавычкой строкового литерала. Это говорит Python, что это f-строка вместо стандартной строки.
- Укажите любые переменные, которые будут интерполированы в фигурных скобках ({}).

Переделанный с помощью f-строки, приведенный выше пример выглядит гораздо чище:

```
>>> n = 20
>>> m = 25
>>> prod = n * m
>>> print(f'The product of {n} and {m} is {prod}')
The product of 20 and 25 is 500
```

Любой из трех механизмов цитирования Python может быть использован для определения f-строки:

```
>>> var = 'Bark'
>>> print(f'A dog says {var}!')
A dog says Bark!
>>> print(f"A dog says {var}!")
A dog says Bark!
>>> print(f"'A dog says {var}!'")
A dog says Bark!
```

Изменение Строк

Короче говоря, вы не можете. Строки-это один из типов данных, которые Python считает неизменяемыми, то есть не подлежащими изменению. Фактически, все типы данных, которые вы видели до сих пор, неизменны. (Python предоставляет типы данных, которые являются изменяемыми, как вы скоро увидите.)

Подобное утверждение приведет к ошибке:

```
>>> s = 'foobar'
>>> s[3] = 'x'
Traceback (most recent call last):
File "<pyshell#40>", line 1, in <module>
s[3] = 'x'
TypeError: 'str' object does not support item assignment
```

По правде говоря, нет особой необходимости изменять строки. Обычно вы можете легко выполнить то, что хотите, создав копию исходной строки, которая имеет желаемое изменение на месте. В Python есть очень много способов сделать это. Вот одна из возможностей:

```
>>> s = s[:3] + 'x' + s[4:]
>>> s
'fooxar'
```

Для этого также существует встроенный строковый метод:

```
>>> s = 'foobar'
>>> s = s.replace('b', 'x')
>>> s
'fooxar'
```

Читайте дальше для получения дополнительной информации о встроенных строковых методах!

Встроенные строковые методы

В уроке по переменным в Python вы узнали, что Python-это высоко объектно-ориентированный язык. Каждый элемент данных в программе Python является объектом.

Вы также знакомы с функциями: вызываемыми процедурами, которые можно вызвать для выполнения определенных задач.

Методы подобны функциям. Метод - это специализированный тип вызываемой процедуры, тесно связанный с объектом. Как и функция, метод вызывается для выполнения отдельной задачи, но он вызывается для конкретного объекта и имеет знание о своем целевом объекте во время выполнения.

Синтаксис вызова метода для объекта выглядит следующим образом:

obj.foo(<args>)

Это вызывает метод .foo() на объекте obj.<args> указывает аргументы, передаваемые методу (если таковые имеются).

Вы узнаете гораздо больше об определении и вызове методов позже в обсуждении объектно-ориентированного программирования. На данный момент цель состоит в том, чтобы представить некоторые из наиболее часто используемых встроенных методов, поддерживаемых Python для работы со строковыми объектами.

В следующих определениях методов аргументы, указанные в квадратных скобках ([]), являются необязательными.

Преобразования Регистра

Методы этой группы выполняют преобразование регистра в целевой строке.

s.capitalize()

Возвращает целевую строку с заглавной первой буквой.

s.capitalize() возвращает копию s с первым символом, преобразованным в верхний регистр, и всеми остальными символами, преобразованными в нижний регистр:

```
>>> s = 'foO BaR BAZ quX'
>>> s.capitalize()
'Foo bar baz qux'
```

Неалфавитные символы остаются неизменными:

```
>>> s = 'foo123#BAR#.'
>>> s.capitalize()
'Foo123#bar#.'
```

s.lower()

Преобразует буквенные символы в строчные.

s.lower() возвращает копию s со всеми заглавными символами, преобразованными в нижний регистр:

>>> 'FOO Bar 123 baz qUX'.lower()

'foo bar 123 baz qux'

s.swapcase()

Меняет местами регистр буквенных символов.

s.swapcase() возвращает копию s с заглавными буквенными символами, преобразованными в строчные и наоборот:

>>> 'FOO Bar 123 baz qUX'.swapcase()

'foo bAR 123 BAZ Qux'

s.title()

Преобразует целевую строку в " регистр заголовка.”

s.title() возвращает копию s, в которой первая буква каждого слова преобразуется в верхний регистр, а остальные буквы-в нижний:

>>> 'the sun also rises'.title()

'The Sun Also Rises'

Этот метод использует довольно простой алгоритм. Он не пытается провести различие между важными и неважными словами, и он не обрабатывает апострофы, притяжательные или аббревиатуры изящно:

>>> "what's happened to ted's IBM stock?".title()

"What'S Happened To Ted'S Ibm Stock?"

s.upper()

Преобразует буквенные символы в заглавные.

s.upper() возвращает копию s со всеми буквенными символами, преобразованными в верхний регистр:

>>> 'FOO Bar 123 baz qUX'.upper()

'FOO BAR 123 BAZ QUX'

Найти и заменить

Эти методы предоставляют различные средства поиска целевой строки для указанной подстроки.

Каждый метод в этой группе поддерживает необязательные аргументы <start> и <end>. Они используются для нарезки строк: действие метода ограничено частью целевой строки, начинающейся с позиции символа <start> и продолжающейся до позиции символа <end>, но не включающей ее. Если < start> указан, а <end> нет, то метод применяется к части целевой строки от <start> до конца строки.

s.count(<sub>[, <start>[, <end>]])

Подсчитывает вхождения подстроки в целевую строку.

s.count(<sub>) возвращает количество неперекрывающихся вхождений подстроки <sub> в s:

>>> 'foo goo moo'.count('oo')

3

Подсчет ограничен количеством вхождений в подстроку, обозначенную <start> и <end>, если они указаны:

>>> 'foo goo moo'.count('oo', 0, 8)

2

s.endswith(<suffix>[, <start>[, <end>]])

Определяет, заканчивается ли целевая строка заданной подстрокой.

s.endswith(<suffix>) возвращает True, если s заканчивается указанным <suffix>, и False в противном случае:

>>> 'foobar'.endswith('bar')

True

>>> 'foobar'.endswith('baz')

False

Сравнение ограничивается подстрокой, обозначенной <start> и <end>, если они указаны:

>>> 'foobar'.endswith('oob', 0, 4)

True

>>> 'foobar'.endswith('oob', 2, 4)

False

s.find(<sub>[, <start>[, <end>]])

Выполняет поиск в целевой строке заданной подстроки.

Вы можете использовать .find(), чтобы увидеть, содержит ли строка определенную подстроку. s.find(<sub>) возвращает самый низкий индекс в s, где находится подстрока <sub> :

>>> 'foo bar foo baz foo qux'.find('foo')

0

Этот метод возвращает -1, если указанная подстрока не найдена:

>>> 'foo bar foo baz foo qux'.find('grault')

-1

Поиск ограничен подстрокой, указанной <start> и <end>, если они указаны:

>>> 'foo bar foo baz foo qux'.find('foo', 4)

8

>>> 'foo bar foo baz foo qux'.find('foo', 4, 7)

-1

s.index(<sub>[, <start>[, <end>]])

Выполняет поиск в целевой строке заданной подстроки.

Этот метод идентичен методу .find(), за исключением того, что он вызывает ошибку, если <sub> не найден, а не возвращает -1:

>>> 'foo bar foo baz foo qux'.index('grault')

Traceback (most recent call last):

File "<pyshell#0>", line 1, **in** <module>

'foo bar foo baz foo qux'.index('grault')

ValueError: substring not found

s.rfind(<sub>[, <start>[, <end>]])

Поиск в целевой строке заданной подстроки, начинающейся в конце.

s.rfind(<sub>) возвращает самый высокий индекс в s, где найдена подстрока <sub> :

>>> 'foo bar foo baz foo qux'.rfind('foo')

16

Как и в случае с .find(), если подстрока не найдена, возвращается -1:

>>> 'foo bar foo baz foo qux'.rfind('grault')

-1

Поиск ограничен подстрокой, указанной <start> и <end>, если они указаны:

>>> 'foo bar foo baz foo qux'.rfind('foo', 0, 14)

8

>>> 'foo bar foo baz foo qux'.rfind('foo', 10, 14)

-1

s.rindex(<sub>[, <start>[, <end>]])

Поиск в целевой строке заданной подстроки, начинающейся в конце.

Этот метод идентичен методу .rfind(), за исключением того, что он вызывает ошибку, если <sub> не найден, а не возвращает -1:

>>> 'foo bar foo baz foo qux'.rindex('grault')

Traceback (most recent call last):

File "<pyshell#1>", line 1, **in** <module>

'foo bar foo baz foo qux'.rindex('grault')

ValueError: substring not found

s.startswith(<prefix>[, <start>[, <end>]])

Определяет, начинается ли целевая строка с заданной подстроки.

При использовании метода .startswith() s.startswith(<suffix>) возвращает True, если s начинается с указанного <suffix> , и False в противном случае:

>>> 'foobar'.startswith('foo')

True

>>> 'foobar'.startswith('bar')

False

Сравнение ограничивается подстрокой, обозначенной <start> и <end>, если они указаны:

>>> 'foobar'.startswith('bar', 3)

True

>>> 'foobar'.startswith('bar', 3, 2)

False

Классификация Символов

Методы этой группы классифицируют строку на основе содержащихся в ней символов.

s.isalnum()

Определяет, состоит ли целевая строка из буквенно-цифровых символов.

s.isalnum() возвращает True, если s непусто и все его символы являются буквенно-цифровыми(либо буквой, либо цифрой), и False в противном случае:

>>> 'abc123'.isalnum()

True

>>> 'abc$123'.isalnum()

False

>>> ''.isalnum()

False

s.isalpha()

Определяет, состоит ли целевая строка из буквенных символов.

s.isalpha() возвращает True, если s непусто и все его символы алфавитны, и False в противном случае:

>>> 'ABCabc'.isalpha()

True

>>> 'abc123'.isalpha()

False

s.isdigit()

Определяет, состоит ли целевая строка из цифровых символов.

Вы можете использовать метод .isdigit(), чтобы проверить, состоит ли ваша строка только из цифр. s.isdigit() возвращает True, если s непусто и все его символы являются числовыми цифрами, и False в противном случае:

>>> '123'.isdigit()

True

>>> '123abc'.isdigit()

False

s.isidentifier()

Определяет, является ли целевая строка допустимым идентификатором Python.

s.isidentifier() возвращает True, если s является допустимым идентификатором Python в соответствии с определением языка, и False в противном случае:

>>> 'foo32'.isidentifier()

True

>>> '32foo'.isidentifier()

False

>>> 'foo$32'.isidentifier()

False

Примечание: .isidentifier() вернет True для строки, которая соответствует ключевому слову Python, даже если это на самом деле не является допустимым идентификатором:

>>> 'and'.isidentifier()

True

Вы можете проверить, соответствует ли строка ключевому слову Python, используя функцию iskeyword(), которая содержится в модуле keyword. Один из возможных способов сделать это показан ниже:

>>> **from** keyword **import** iskeyword

>>> iskeyword('and')

True

Если вы действительно хотите убедиться, что строка будет служить допустимым идентификатором Python, вы должны проверить, что .isidentifier() является истинным, а iskeyword() - ложным.

s.islower()

Определяет, являются ли буквенные символы целевой строки строчными.

s.islower() возвращает True, если s непусто и все содержащиеся в нем буквенные символы строчные, и False в противном случае. Неалфавитные символы игнорируются:

```
>>> 'abc'.islower()
True
>>> 'abc1$d'.islower()
True
>>> 'Abc1$D'.islower()
False
```

s.isprintable()

Определяет, состоит ли целевая строка полностью из печатаемых символов.

s.isprintable() возвращает True, если s пуст или все содержащиеся в нем буквенные символы доступны для печати. Он возвращает False, если s содержит хотя бы один непечатаемый символ. Неалфавитные символы игнорируются:

```
>>> 'a\tb'.isprintable()
False
>>> 'a b'.isprintable()
True
>>> ''.isprintable()
True
>>> 'a\nb'.isprintable()
False
```

Примечание: Только метод .isxxxx() возвращает True, если s является пустой строкой. Все остальные возвращают False для пустой строки.

s.isspace()

Определяет, состоит ли целевая строка из пробельных символов.

s.isspace() возвращает True, если s непусто и все символы являются пробелами, и False в противном случае.

Наиболее часто встречающимися пробелами являются пробел ' ', табуляция '\t' и новая строка '\n':

```
>>> ' \t \n '.isspace()
True
>>> ' a '.isspace()
False
```

Однако есть несколько других символов ASCII, которые квалифицируются как пробелы, и если вы учитываете символы Unicode, то их довольно много:

```
>>> '\f\u2005\r'.isspace()
True
```

s.istitle()

Определяет, является ли целевая строка заголовком.

s.istitle() возвращает True, если s непусто, первый буквенный символ каждого слова прописной, а все остальные буквенные символы в каждом слове строчные. В противном случае он возвращает False:

>>> 'This Is A Title'.istitle()

True

>>> 'This is a title'.istitle()

False

>>> 'Give Me The #$#@ Ball!'.istitle()

True

Примечание: вот как документация Python описывает .istitle(), если вы находите это более интуитивно понятным: “заглавные символы могут следовать только за несокращенными символами, а строчные-только за прописными.”

s.isupper()

Определяет, являются ли буквенные символы целевой строки заглавными.

s.isupper() возвращает True, если s непусто и все содержащиеся в нем буквенные символы заглавные, и False в противном случае. Неалфавитные символы игнорируются:

>>> 'ABC'.isupper()

True

>>> 'ABC1$D'.isupper()

True

>>> 'Abc1$D'.isupper()

False

Форматирование Строк

Методы этой группы изменяют или улучшают формат строки.

s.center(<width>[, <fill>])

Центрирует строку в поле.

s.center(<width>) возвращает строку, состоящую из s, центрированных в поле width <width>. По умолчанию заполнение состоит из символа пробела ASCII:

>>> 'foo'.center(10)

' foo '

Если указан необязательный аргумент <fill>, то он используется в качестве символа заполнения:

>>> 'bar'.center(10, '-')

'---bar----'

Если аргумент уже имеет длину не менее <width>, то он возвращается без изменений:

>>> 'foo'.center(2)

'foo'

s.expandtabs(tabsize=8)

Разворачивает столбцы в строку.

s.expandtabs() заменяет каждый символ табуляции ('\t') пробелами. По умолчанию пробелы заполняются при условии остановки табуляции в каждом восьмом столбце:

>>> 'a\tb\tc'.expandtabs()

'a b c'

>>> 'aaa\tbbb\tc'.expandtabs()

'aaa bbb c'

tabsize является необязательным параметром, определяющим остановку колонны:

```
>>> 'a\tb\tc'.expandtabs(4)
'a b c'
>>> 'aaa\tbbb\tc'.expandtabs(tabsize=4)
'aaa bbb c'
```

s.ljust(<width>[, <fill>])

Левостороннее выравнивание строки в поле.

s.ljust(<width>) возвращает строку, состоящую из s, выровненных по левому краю в поле ширина <width>. По умолчанию заполнение состоит из символа пробела ASCII:

```
>>> 'foo'.ljust(10)
'foo       '
```

Если указан необязательный аргумент <fill>, то он используется в качестве символа заполнения:

```
>>> 'foo'.ljust(10, '-')
'foo-------'
```

Если s уже достигает ширины такой как <width>, то он возвращается без изменений:

```
>>> 'foo'.ljust(2)
'foo'
```

s.lstrip([<chars>])

Обрезает ведущие символы из строки.

s.lstrip() возвращает копию s с любыми пробелами, удаленными с левого конца:

```
>>> '   foo bar baz   '.lstrip()
'foo bar baz   '
>>> '\t\nfoo\t\nbar\t\nbaz'.lstrip()
'foo\t\nbar\t\nbaz'
```

Если указан необязательный аргумент <chars>, то это строка, указывающая набор удаляемых символов:

```
>>> 'http://www.realpython.com'.lstrip('/:pth')
'www.realpython.com'
```

s.replace(<old>, <new>[, <count>])

Заменяет вхождения подстроки в строке.

В Python для удаления символа из строки можно использовать метод string.replace(). Метод s.replace(<old>, <new>) возвращает копию s со всеми вхождениями подстроки <old>, замененной на <new>.:

```
>>> 'foo bar foo baz foo qux'.replace('foo', 'grault')
'grault bar grault baz grault qux'
```

Если указан необязательный аргумент <count>, то выполняется максимум замен <count>, начиная с левого конца s:

```
>>> 'foo bar foo baz foo qux'.replace('foo', 'grault', 2)
'grault bar grault baz foo qux'
```

s.rjust(<width>[, <fill>])

Правостороннее выравнивание строки в поле.

s.rjust(<width>) возвращает строку, состоящую из s, выровненных по правому краю в поле ширины <width>. По умолчанию заполнение состоит из символа пробела ASCII:

```
>>> 'foo'.rjust(10)
' foo'
```

Если указан необязательный аргумент <fill>, то он используется в качестве символа заполнения:

```
>>> 'foo'.rjust(10, '-')
'-------foo'
```

Если s уже по крайней мере равен <width>, то он возвращается без изменений:

```
>>> 'foo'.rjust(2)
'foo'
```

s.rstrip([<chars>])

Обрезает конечные символы из строки.

s.rstrip() возвращает копию s с любыми пробелами, удаленными с правого конца:

```
>>> ' foo bar baz '.rstrip()
' foo bar baz'
>>> 'foo\t\nbar\t\nbaz\t\n'.rstrip()
'foo\t\nbar\t\nbaz'
```

Если указан необязательный аргумент <chars>, то это строка, указывающая набор удаляемых символов:

```
>>> 'foo.$$$;'.rstrip(';$.')
'foo'
```

s.strip([<chars>])

Удаляет символы с левого и правого концов строки.

s.strip() по существу эквивалентен вызову s.lstrip() и s.rstrip() последовательно. Без аргумента <chars> он удаляет начальные и конечные пробелы:

```
>>> s = ' foo bar baz\t\t\t'
>>> s = s.lstrip()
>>> s = s.rstrip()
>>> s
'foo bar baz'
```

Как и в случае с .lstrip() и .rstrip(), необязательный аргумент <chars> указывает набор удаляемых символов:

```
>>> 'www.realpython.com'.strip('w.moc')
'realpython'
```

Примечание: когда возвращаемое значение строкового метода является другой строкой, как это часто бывает, методы могут быть вызваны последовательно путем цепочки вызовов:

```
>>> ' foo bar baz\t\t\t'.lstrip().rstrip()
'foo bar baz'
>>> ' foo bar baz\t\t\t'.strip()
'foo bar baz'
>>> 'www.realpython.com'.lstrip('w.moc').rstrip('w.moc')
```

'realpython'

>>> 'www.realpython.com'.strip('w.moc')

'realpython'

s.zfill(<width>)

Подкладывает строку слева нулями.

s.zfill(<width>) возвращает копию s, заполненную слева символами '0' до указанной <width>:

>>> '42'.zfill(5)

'00042'

Если s содержит начальный знак, он остается на левом краю результирующей строки после вставки нулей:

>>> '+42'.zfill(8)

'+0000042'

>>> '-42'.zfill(8)

'-0000042'

Если s уже по крайней мере равен <width>, то он возвращается без изменений:

>>> '-42'.zfill(3)

'-42'

.zfill() наиболее полезен для строковых представлений чисел, но Python все равно будет обнулять строку, которая не является таковой.:

>>> 'foo'.zfill(6)

'000foo'

Преобразование между строками и списками

Методы в этой группе преобразуют между строкой и некоторым составным типом данных либо вставляя объекты вместе, чтобы сделать строку, либо разбивая строку на части.

Многие из этих методов возвращают либо список, либо кортеж. Это составные типы данных, которые являются прототипическими примерами итерируемые объекты в Python. Они описаны в следующем уроке, так что вы скоро узнаете о них! До тех пор просто думайте о них как о последовательностях значений. Список заключен в квадратные скобки ([]), а кортеж-в круглые скобки (()).

С этим введением давайте взглянем на эту последнюю группу строковых методов.

s.join(<iterable>)

Конкатенация строк из итерируемого объекта.

s.join(<iterable>) возвращает строку, полученную в результате объединения объектов в <iterable>, разделенных s.

Обратите внимание, что .join() вызывается на s, строке-разделителе. <iterable> также должна быть последовательностью строковых объектов.

Некоторые примеры кода должны помочь прояснить ситуацию. В следующем примере разделителями являются строки ',', а <iterable> - список строковых значений:

>>> ', '.join(['foo', 'bar', 'baz', 'qux'])

'foo, bar, baz, qux'

В результате получается одна строка, состоящая из объектов списка, разделенных запятыми.

В следующем примере <iterable> задается как одно строковое значение. Когда строковое значение используется в качестве итеративного, оно интерпретируется как список отдельных символов строки:

```
>>> list('corge')
['c', 'o', 'r', 'g', 'e']
>>> ':'.join('corge')
'c:o:r:g:e'
```

Таким образом, в результате ':'.join('corge') получается строка, состоящая из каждого символа в 'corge', разделенного ':'.

Этот пример терпит неудачу, потому что один из объектов в <iterable> не является строкой:

```
>>> '---'.join(['foo', 23, 'bar'])
Traceback (most recent call last):
File "<pyshell#0>", line 1, in <module>
'---'.join(['foo', 23, 'bar'])
TypeError: sequence item 1: expected str instance, int found
```

Это можно исправить с помощью:

```
>>> '---'.join(['foo', str(23), 'bar'])
'foo---23---bar'
```

Скоро увидите, многие составные объекты в Python могут быть истолкованы как итеративные, и .join() особенно полезен для создания строк из них.

s.partition(<sep>)

Делит строку на основе разделителя.

s.partition(<sep>) разбивает s при первом вхождении строки <sep>. Возвращаемое значение представляет собой кортеж из трех частей, состоящий из:

- Часть s, предшествующая <sep>
- Сам <sep>
- Часть s, следующая за <sep>

Вот несколько примеров .partition() в действии:

```
>>> 'foo.bar'.partition('.')
('foo', '.', 'bar')
>>> 'foo@@bar@@baz'.partition('@@')
('foo', '@@', 'bar@@baz')
```

Если <sep> не найден в s, возвращаемый кортеж содержит s, за которым следуют две пустые строки:

```
>>> 'foo.bar'.partition('@@')
('foo.bar', '', '')
```

s.rpartition(<sep>)

Делит строку на основе разделителя.

s.rpartition(<sep>) функционирует точно так же, как s.partition(<sep>), за исключением того, что s разделяется при последнем вхождении <sep> вместо первого вхождения:

```
>>> 'foo@@bar@@baz'.partition('@@')
('foo', '@@', 'bar@@baz')
>>> 'foo@@bar@@baz'.rpartition('@@')
('foo@@bar', '@@', 'baz')
```

s.rsplit(sep=None, maxsplit=-1)

Разбивает строку на список подстрок.

Без аргументов s.split() разбивает s на подстроки, разделенные любой последовательностью пробелов, и возвращает подстроки в виде списка:

```
>>> 'foo bar baz qux'.rsplit()
['foo', 'bar', 'baz', 'qux']
>>> 'foo\n\tbar baz\r\fqux'.rsplit()
['foo', 'bar', 'baz', 'qux']
```

Если указан <sep>, то он используется в качестве разделителя для разделения:

```
>>> 'foo.bar.baz.qux'.rsplit(sep='.')
['foo', 'bar', 'baz', 'qux']
```

(Если <sep> задано со значением None, строка разделяется пробелом, как если бы <sep> вообще не было задано.)

Когда <sep> явно задан в качестве разделителя, предполагается, что последовательные разделители в s разделяют пустые строки, которые будут возвращены:

```
>>> 'foo...bar'.rsplit(sep='.')
['foo', '', '', 'bar']
```

Однако это не тот случай, когда <sep> опущен. В этом случае последовательные пробелы объединяются в один разделитель, и результирующий список никогда не будет содержать пустых строк:

```
>>> 'foo\t\t\tbar'.rsplit()
['foo', 'bar']
```

Если указан необязательный параметр ключевого слова <maxsplit>, то выполняется максимум такое количество разбиений, начиная с правого конца s:

```
>>> 'www.realpython.com'.rsplit(sep='.', maxsplit=1)
['www.realpython', 'com']
```

Значение по умолчанию для <maxsplit> равно -1, что означает, что должны быть выполнены все возможные разбиения— так же, как если бы <maxsplit> был полностью опущен:

```
>>> 'www.realpython.com'.rsplit(sep='.', maxsplit=-1)
['www', 'realpython', 'com']
>>> 'www.realpython.com'.rsplit(sep='.')
['www', 'realpython', 'com']
```

s.split(sep=None, maxsplit=-1)

Разбивает строку на список подстрок.

s.split() ведет себя точно так же, как s.rsplit(), за исключением того, что если задано <maxsplit>, то разбиения отсчитываются с левого конца s, а не с правого:

```
>>> 'www.realpython.com'.split('.', maxsplit=1)
['www', 'realpython.com']
>>> 'www.realpython.com'.rsplit('.', maxsplit=1)
['www.realpython', 'com']
```

Если <maxsplit> не указан, то .split() и .rsplit() неразличимы.

s.splitlines([<keepends>])

Разрывает строку на границах линий.

s.splitlines() разбивает s на строки и возвращает их в виде спика. Любой из следующих символов или последовательностей символов считается границей линии:

Таблица 9.2

Escape-символ	**Описание**
\n	Новая строка
\r	Возврат к началу строки
\r\n	Возврат к началу строки + заполнение
\v или \x0b	Подведение Линии
\f или \x0c	Перевод на следующий лист(Form Feed)
\x1c	Разделение файлов
\x1d	разделитель групп
\x1e	разделитель записей
\x85	Следующая Строка (Контрольный Код C1)
\u2028	Разделитель Строк Unicode
\u2029	Разделитель Абзацев Unicode

Вот пример использования нескольких различных разделителей строк:

```
>>> 'foo\nbar\r\nbaz\fqux\u2028quux'.splitlines()
['foo', 'bar', 'baz', 'qux', 'quux']
```

Если в строке присутствуют последовательные символы границ строк, предполагается, что они разделяют пустые строки, которые появятся в списке результатов:

```
>>> 'foo\f\f\fbar'.splitlines()
['foo', '', '', 'bar']
```

Если необязательный аргумент <keepends> указан и является истинным, то границы строк сохраняются в результирующих строках:

```
>>> 'foo\nbar\nbaz\nqux'.splitlines(True)
['foo\n', 'bar\n', 'baz\n', 'qux']
>>> 'foo\nbar\nbaz\nqux'.splitlines(1)
['foo\n', 'bar\n', 'baz\n', 'qux']
```

Объекты bytes

Объект bytes является одним из основных встроенных типов для манипулирования двоичными данными. Объект bytes - это неизменяемая последовательность однобайтовых значений. Каждый элемент в объекте bytes представляет собой небольшое целое число в диапазоне от 0 до 255.

Определение литерала объекта bytes

Байтовый литерал определяется так же, как и строковый литерал с добавлением префикса 'b' :

```
>>> b = b'foo bar baz'
>>> b
b'foo bar baz'
>>> type(b)
<class 'bytes'>
```

Как и в случае со строками, вы можете использовать любой из механизмов одинарного, двойного или тройного цитирования:

```
>>> b'Contains embedded "double" quotes'
b'Contains embedded "double" quotes'
>>> b"Contains embedded 'single' quotes"
b"Contains embedded 'single' quotes"
>>> b'''Contains embedded "double" and 'single' quotes'''
b'Contains embedded "double" and \'single\' quotes'
>>> b"""Contains embedded "double" and 'single' quotes"""
b'Contains embedded "double" and \'single\' quotes'
```

В байтовом литерале допускаются только символы ASCII. Любое символьное значение больше 127 должно быть указано с помощью соответствующей escape-последовательности:

```
>>> b = b'foo\xddbar'
>>> b
b'foo\xddbar'
>>> b[3]
221
>>> int(0xdd)
221
```

Префикс 'r' может использоваться в байтовом литерале для отключения обработки escape-последовательностей, как и в случае со строками:

```
>>> b = rb'foo\xddbar'
>>> b
b'foo\\xddbar'
>>> b[3]
92
>>> chr(92)
'\\'
```

Определение объекта bytes с помощью встроенной функции bytes()

Функция bytes() также создает объект bytes. Какой тип байтов возвращается объекту, зависит от аргумента(ов), переданного функции. Возможные формы приведены ниже.

bytes(<s>, <encoding>)

Создает объект bytes из строки.

bytes(<s>, <encoding>) преобразует строку <s> в объект bytes, используя str.encode() в соответствии с заданной <encoding>:

```
>>> b = bytes('foo.bar', 'utf8')
>>> b
b'foo.bar'
>>> type(b)
<class 'bytes'>
```

Техническое Примечание: В этой форме функции bytes() требуется аргумент <encoding>. "Кодирование" относится к способу перевода символов в целочисленные значения. Значение "utf 8" указывает на формат преобразования Unicode UTF-8, который является кодировкой, способной обрабатывать все возможные символы Unicode. UTF-8 также можно указать, указав "UTF8", "utf-8" или "UTF-8" для <encoding>.

Дополнительные сведения см. В документации Unicode. До тех пор, пока вы имеете дело с обычными латинскими символами, UTF-8 будет хорошо служить вам.

bytes(<size>)

Создает объект bytes, состоящий из нулевых (0x00) байтов.

bytes(<size>) определяет объект bytes указанного <size>, который должен быть положительным целым числом. Результирующий объект bytes инициализируется нулевыми (0x00) байтами:

```
>>> b = bytes(8)
>>> b
b'\x00\x00\x00\x00\x00\x00\x00\x00'
>>> type(b)
<class 'bytes'>
```

bytes(<iterable>)

Создает объект bytes из итерируемого объекта.

bytes(<iterable>) определяет объект bytes из последовательности целых чисел, сгенерированных <iterable>. <iterable> должен быть итегрируемым, который генерирует последовательность целых чисел n в диапазоне $0 \leq n \leq 255$:

```
>>> b = bytes([100, 102, 104, 106, 108])
>>> b
b'dfhjl'
>>> type(b)
<class 'bytes'>
>>> b[2]
104
```

Операции с байтовыми объектами

Как и строки, объекты bytes поддерживают общие операции последовательности:

– Операторы in и not in:

```
>>> b = b'abcde'
>>> b'cd' in b
True
>>> b'foo' not in b
```

True

– Операторы конкатенации (+) и репликации (*) :

```
>>> b = b'abcde'
>>> b + b'fghi'
b'abcdefghi'
>>> b * 3
b'abcdeabcdeabcde'
```

– Индексирование и нарезка:

```
>>> b = b'abcde'
>>> b[2]
99
>>> b[1:3]
b'bc'
```

– Встроенные функции:

```
>>> len(b)
5
>>> min(b)
97
>>> max(b)
101
```

Многие методы, определенные для строковых объектов, допустимы и для байтовых объектов:

```
>>> b = b'foo,bar,foo,baz,foo,qux'
>>> b.count(b'foo')
3
>>> b.endswith(b'qux')
True
>>> b.find(b'baz')
12
>>> b.split(sep=b',')
[b'foo', b'bar', b'foo', b'baz', b'foo', b'qux']
>>> b.center(30, b'-')
b'---foo,bar,foo,baz,foo,qux----'
```

Обратите внимание, однако, что когда эти операторы и методы вызываются для объекта bytes, операнд и аргументы также должны быть объектами bytes:

```
>>> b = b'foo.bar'
>>> b + '.baz'
Traceback (most recent call last):
File "<pyshell#72>", line 1, in
b + '.baz'
TypeError: can't concat bytes to str
>>> b + b'.baz'
```

```
b'foo.bar.baz'
>>> b.split(sep='.')
Traceback (most recent call last):
File "<pyshell#74>", line 1, in
b.split(sep='.')
TypeError: a bytes-like object is required, not 'str'
>>> b.split(sep=b'.')
[b'foo', b'bar']
```

Хотя определение и представление байтового объекта основано на тексте ASCII, на самом деле он ведет себя как неизменяемая последовательность небольших целых чисел в диапазоне от 0 до 255 включительно. Вот почему один элемент из объекта bytes отображается как целое число:

```
>>> b = b'foo\xddbar'
>>> b[3]
221
>>> hex(b[3])
'0xdd'
>>> min(b)
97
>>> max(b)
221
```

Однако срез отображается как байтовый объект, даже если он имеет длину всего в один байт:

```
>>> b[2:3]
b'c'
```

Вы можете преобразовать объект bytes в список целых чисел с помощью встроенной функции list()

```
>>> list(b)
[97, 98, 99, 100, 101]
```

Шестнадцатеричные числа часто используются для указания двоичных данных, поскольку две шестнадцатеричные цифры непосредственно соответствуют одному байту. Класс bytes поддерживает два дополнительных метода, облегчающих преобразование в строку шестнадцатеричных цифр и обратно.

bytes.fromhex(<s>)

Возвращает объект bytes, построенный из строки шестнадцатеричных значений.

bytes.fromhex(<s>) возвращает объект bytes, полученный в результате преобразования каждой пары шестнадцатеричных цифр в <s> в соответствующее значение байта. Шестнадцатеричные пары цифр в <s> могут быть дополнительно разделены пробелами, которые игнорируются:

```
>>> b = bytes.fromhex(' aa 68 4682cc ')
>>> b
b'\xaahF\x82\xcc'
>>> list(b)
```

[170, 104, 70, 130, 204]

Примечание: этот метод является методом класса, а не методом объекта. Он привязан к классу bytes, а не к объекту bytes. В следующих уроках по объектно-ориентированному программированию вы гораздо глубже изучите различие между классами, объектами и соответствующими им методами. А пока просто обратите внимание, что этот метод вызывается в классе bytes, а не в объекте b.

b.hex()

Возвращает строку, содержащую шестнадцатеричное значение из объекта в байтах.

b.hex() возвращает результат преобразования байтов объекта b в строку шестнадцатеричных пар цифр. То есть он делает обратное .fromhex():

```
>>> b = bytes.fromhex(' aa 68 4682cc ')
>>> b
b'\xaahF\x82\xcc'
>>> b.hex()
'aa684682cc'
>>> type(b.hex())
<class 'str'>
```

Примечание: В отличие от .fromhex(), .hex() - это объектный метод, а не метод класса. Таким образом, он вызывается на объекте класса bytes, а не на самом классе.

Объекты bytearray

Python поддерживает другой тип двоичной последовательности, называемый bytearray. Объекты bytearray очень похожи на объекты bytes, несмотря на некоторые различия:

- В Python нет специального синтаксиса для определения литерала байтового массива, такого как префикс 'b', который может использоваться для определения объекта bytes. Объект bytearray всегда создается с помощью встроенной функции bytearray() :

```
>>> ba = bytearray('foo.bar.baz', 'UTF-8')
>>> ba
bytearray(b'foo.bar.baz')
>>> bytearray(6)
bytearray(b'\x00\x00\x00\x00\x00\x00')
>>> bytearray([100, 102, 104, 106, 108])
bytearray(b'dfhjl')
```

- Объекты bytearray изменчивы. Вы можете изменить содержимое объекта bytearray с помощью индексации и нарезки:

```
>>> ba = bytearray('foo.bar.baz', 'UTF-8')
>>> ba
bytearray(b'foo.bar.baz')
>>> ba[5] = 0xee
>>> ba
bytearray(b'foo.b\xeer.baz')
>>> ba[8:11] = b'qux'
```

```
>>> ba
bytearray(b'foo.b\xeer.qux')
```
Объект bytearray также может быть построен непосредственно из объекта bytes:
```
>>> ba = bytearray(b'foo')
>>> ba
bytearray(b'foo')
```
Самостоятельная работа

1. Используйте метод len для вывода длины строки.
   x = "Привет Мир"
   print( )
2. Получить первый символ строки txt.
   txt = "Привет Мир"
   x=
3. Получить символы из индекса 2 в индекс 4 (llo).
   txt = "Привет Мир"
   x=
4. Возвращает строку без пробелов в начале или конце.
   txt = " Привет Мир "
   x=
5. Преобразуйте значение txt в верхний регистр.
   txt = "Привет Мир"
   txt=
6. Преобразуйте значение txt в нижний регистр.
   txt = "Привет Мир"
   txt=
7. Замените символ П на К.
   txt = "Привет Мир"
   txt = txt. (,)
8. Вставьте правильный синтаксис, чтобы добавить заполнитель для параметра age
   age = 36
   txt = "Меня зовут Андрей, и мне. ……."
   print(txt.format(age))
9. Напишите программу, которая выводит на экран:
   - Did Joffrey agree?
   - He did. He also said "I love using \n".
10. При этом программа использует только один print(), но результат на экране должен выводиться в две строчки, как показано выше.
11. Напишите программу на Python для расчета длины строки.
12. Напишите программу на Python для подсчета количества символов (частоты символов) в строке.
    Пример строки: google.com '
    Ожидаемый результат: {'o': 3, 'g': 2, '.': 1, 'e': 1, 'l': 1, 'm': 1, 'c': 1}

13. Напишите программу на Python, чтобы получить строку из первых 2 и последних 2 символов из заданной строки. Если длина строки меньше 2, верните вместо пустой строки.
    Пример строки: «w3resource»
    Ожидаемый результат: 'w3ce'
    Пример строки: 'w3'
    Ожидаемый результат: 'w3w3'
    Пример строки: 'w'
    Ожидаемый результат: пустая строка
14. Напишите программу на Python, чтобы получить строку из заданной строки, в которой все вхождения ее первого символа были заменены на '$', кроме самого первого символа.
    Пример строки: «перезагрузка»
    Ожидаемый результат: «$ re $»
15. Напишите программу на Python, чтобы получить одну строку из двух заданных строк, разделенных пробелом, и поменять местами первые два символа каждой строки.
    Пример строки: «abc», «xyz»
    Ожидаемый результат: 'xyc abz'
16. Напишите программу на Python для добавления 'ing' в конец заданной строки (длина должна быть не менее 3). Если данная строка уже заканчивается на «ing», вместо этого добавьте «ly». Если длина строки данной строки меньше 3, оставьте ее без изменений.
    Пример строки: «abc»
    Ожидаемый результат: «abcing»
    Пример строки: «строка»
    Ожидаемый результат: «Строго»
17. Напишите программу на Python, чтобы найти первое появление подстроки «not» и «плохой» из заданной строки, если «not» следует за «плохой», замените всю подстроку «not» ... «плохой» на 'хорошо'. Вернуть полученную строку.
    Пример строки: «Текст не такой уж плохой!»
    «Текст плохой!»
    Ожидаемый результат: «Лирика хорошая!»
    «Текст плохой!»
18. Напишите функцию Python, которая берет список слов и возвращает длину самого длинного.
19. Напишите программу на Python для удаления n- го символа индекса из непустой строки.
20. Напишите программу на Python, чтобы заменить данную строку новой строкой, в которой были изменены первый и последний символы.
21. Напишите программу на Python для удаления символов, которые имеют нечетные значения индекса заданной строки.
22. Напишите программу на Python для подсчета вхождений каждого слова в данное предложение.

23. Напишите скрипт Python, который принимает ввод от пользователя и отображает его обратно в верхнем и нижнем регистре.
24. Напишите программу на языке Python, которая принимает последовательность слов, разделенных запятыми, в качестве входных данных и печатает уникальные слова в отсортированном виде (в алфавитном порядке).
    Примеры слов: красный, белый, черный, красный, зеленый, черный
    Ожидаемый результат: черный, зеленый, красный, белый, красный
25. Напишите функцию Python для создания строки HTML с тегами вокруг слова (ей).
    Пример функции и результат:
    add_tags ('i', 'Python') -> '<i> Python </ i>'
    add_tags ('b', 'Python Tutorial') -> '<b> Python Tutorial </ b>'
26. Напишите функцию Python для вставки строки в середину строки.
    Пример функции и результат:
    insert_sting_middle ('[[]] << >>', 'Python') -> [[Python]]
    insert_sting_middle ('{{}}', 'PHP') -> {{PHP}}
27. Напишите функцию Python, чтобы получить строку, составленную из 4 копий двух последних символов указанной строки (длина должна быть не менее 2).
    Пример функции и результат:
    insert_end ('Python') -> onononon
    insert_end ('Упражнения') -> eseseses
28. Напишите функцию Python, чтобы получить строку, состоящую из первых трех символов указанной строки. Если длина строки меньше 3, верните исходную строку.
    Пример функции и результат:
    first_three ('ipy') -> ipy
    first_three ('python') -> pyt
29. Напишите программу на Python, чтобы получить последнюю часть строки перед указанным символом.
    / питон-упражнение
    / питон
30. Напишите функцию Python, которая переворачивает строку, если ее длина кратна 4.
31. Напишите функцию Python для преобразования заданной строки в верхний регистр, если она содержит как минимум 2 заглавных символа в первых 4 символах.
32. Напишите программу на Python для лексикографической сортировки строк.
33. Напишите программу на Python для удаления новой строки в Python.
34. Напишите программу на Python, чтобы проверить, начинается ли строка с указанных символов.
35. Напишите программу на Python для создания шифрования Цезаря. Примечание. В криптографии шифр Цезаря, также известный как шифр Цезаря, шифр сдвига, код Цезаря или сдвиг Цезаря, является одним из самых простых и широко известных методов шифрования. Это тип шифра замещения, в котором каждая буква в открытом тексте заменяется буквой с фиксированным числом позиций

по алфавиту. Например, при сдвиге влево 3 D будет заменен на A, E станет B и так далее. Метод назван в честь Юлия Цезаря, который использовал его в своей личной переписке.

36. Напишите программу на Python для отображения форматированного текста (ширина = 50) в качестве вывода.
37. Напишите программу на Python для удаления существующего отступа из всех строк в данном тексте.
38. Напишите программу на Python, чтобы добавить текст префикса ко всем строкам в строке.
39. Напишите программу на Python для установки отступа первой строки.
40. Напишите программу на Python для печати следующих плавающих чисел с точностью до 2 десятичных знаков.
41. Напишите программу на Python для печати следующих плавающих чисел до 2 десятичных знаков со знаком.
42. Напишите программу на Python для печати следующих плавающих чисел без десятичных знаков.
43. Напишите программу на Python для печати следующих целых чисел с нулями слева от указанной ширины.
44. Напишите программу на Python, которая будет печатать следующие целые числа с '*' справа от указанной ширины.
45. Напишите программу на Python для отображения числа с запятой
46. Напишите программу на Python для форматирования числа в процентах.
47. Напишите программу на Python для отображения числа слева, справа и по центру ширины 10.
48. Напишите программу на Python для подсчета вхождений подстроки в строку.
49. Напишите программу на Python для обращения строки.
50. Напишите программу на Python для обращения слов в строку.
51. Напишите программу на Python для удаления набора символов из строки.
52. Напишите программу на Python для подсчета повторяющихся символов в строке.

    Пример строки: 'thequickbrownfoxjumpsoverthelazydog'

    Ожидаемый результат:

    o 4

    e 3

    ты 2

    ч 2

    р 2

    т 2
53. Напишите программу на языке Python для печати символа квадрата и куба в области прямоугольника и объема цилиндра.

    Образец вывода:

    Площадь прямоугольника составляет 1256,66 см $^{2.}$

    Объем цилиндра 1254,725 см ^{3}
54. Напишите программу на Python для печати индекса символа в строке.

    Пример строки: w3resource

Ожидаемый результат:
Текущий символ w позиция в 0
Текущий персонаж 3 позиция на 1
Текущий символ r позиция в 2
- -
Текущий персонаж c позиция на 8
Текущая позиция персонажа в 9

55. Напишите программу на Python, чтобы проверить, содержит ли строка все буквы алфавита.
56. Напишите программу на Python для преобразования строки в список.
57. Напишите программу на Python, которая должна содержать строчные первые n символов в строке.
58. Напишите программу на Python для замены запятой и точки в строке.
    Пример строки: «32.054,23»
    Ожидаемый результат: "32 054,23"
59. Напишите программу на Python для подсчета и отображения гласных текста.
60. Напишите программу на Python, чтобы разбить строку по последнему вхождению разделителя.
61. Напишите программу на Python, чтобы найти первый неповторяющийся символ в заданной строке.
62. Напишите программу на Python для печати всех перестановок с заданным числом повторений символов данной строки.
63. Напишите программу на Python, чтобы найти первый повторяющийся символ в заданной строке.
64. Напишите программу на Python, чтобы найти первый повторяющийся символ данной строки, где индекс первого вхождения является наименьшим.
65. Напишите программу на Python, чтобы найти первое повторяющееся слово в заданной строке.
66. Напишите программу на Python, чтобы найти второе наиболее повторяющееся слово в данной строке.
67. Напишите программу на Python для удаления пробелов из заданной строки.
68. Напишите программу на Python для перемещения пробелов в начало заданной строки.
69. Напишите программу на Python, чтобы найти максимальное число символов в данной строке.
70. Напишите программу на языке Python, которая будет использовать заглавные и первые буквы каждого слова данной строки.
71. Напишите программу на Python для удаления повторяющихся символов заданной строки.
72. Напишите программу на Python для вычисления суммы цифр заданной строки.
73. Напишите программу на Python для удаления начальных нулей с IP-адреса.
74. Напишите программу на Python, чтобы найти максимальную длину последовательных 0 в заданной двоичной строке.

75. Напишите программу на Python, чтобы найти все общие символы в лексикографическом порядке из двух заданных строчных букв. Если общих букв нет, выведите «Нет общих символов».
76. Напишите программу на Python для создания двух заданных строк (строчные, могут иметь или не иметь одинаковую длину) анаграмм, удаляющих любые символы из любой строки.
77. Напишите программу на Python для удаления всех последовательных дубликатов данной строки.
78. Напишите программу на Python для создания двух строк из заданной строки. Создайте первую строку, используя те символы, которые встречаются только один раз, и создайте вторую строку, состоящую из многократно встречающихся символов в указанной строке.
79. Напишите программу на Python, чтобы найти самую длинную общую подстроку из двух заданных строк.
80. Напишите программу на Python для создания строки из двух заданных строк, объединяющих необычные символы указанных строк.
81. Напишите программу на Python для перемещения всех пробелов впереди заданной строки за один проход.
82. Напишите программу на Python для удаления всех последовательных дубликатов из заданной строки.
83. Напишите программу на Python для подсчета заглавных, строчных букв, специальных символов и числовых значений в заданной строке.
84. Напишите программу на Python, чтобы найти минимальное окно в данной строке, которое будет содержать все символы другой данной строки.
    Пример 1
    Ввод: str1 = "PRWSOERIUSFK"
    str2 = "OSU"
    Вывод: минимальное окно "OERIUS"
85. Напишите программу на Python, чтобы найти наименьшее окно, которое содержит все символы данной строки.
86. Напишите программу на Python для подсчета количества подстрок из заданной строки строчных алфавитов с ровно k различными (заданными) символами.
87. Напишите программу на Python для подсчета количества непустых подстрок данной строки.
88. Напишите программу на Python для подсчета символов в той же позиции в заданной строке (строчные и прописные буквы), что и в английском алфавите.
89. Напишите программу на Python, чтобы найти наименьшее и наибольшее слово в данной строке.
90. Напишите программу на Python для подсчета количества подстрок с одинаковыми первым и последним символами данной строки.

Практическая работа №9. Логический тип данных и операции в Python

Цель: Познакомиться с языком программирования Python. Узнать об основных понятиях в программировании. Изучить работу с логическим типом данных и операциями сравнения.

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

Язык Python, благодаря наличию огромного количества библиотек для решения разного рода вычислительных задач, сегодня является конкурентом таким пакетам как Matlab и Octave. Запущенный в интерактивном режиме, он, фактически, превращается в мощный калькулятор. В этом уроке речь пойдет об арифметических операциях, доступных в данном языке.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Сохраните файл из Shell (он несохраняется автоматически). Сохраните его в отдельный файл с именем pr_Shell 10 (File – Save as).

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый

документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr9.py, обязательно поставьте расширение .py.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Реализуйте все примеры, которые есть в практической работе

Логический тип данных (bool) (или булевый тип) это примитивный тип данных, который принимает 2 значения — истина или ложь.

В Python имеется самостоятельный логический тип bool, с двумя предопределенными значениями:

– True — истина;

– False — ложь.

Переменные этого типа могут принимать одно из двух значений: True(Истина) или False(Ложь). Объект логического типа в Python обозначается как bool.

True и **False** пишутся с большой буквы. Если написать с маленькой **true**, интерпретатор выдаст ошибку: NameError: name 'true' is not defined

True и False являются экземплярами класса bool который в свою очередь является подклассом int Поэтому True и False в Python ведут себя как числа 1 и 0. Отличие только в том, как они выводятся на экран.

```
>>> True
True
>>> type(True)
<class 'bool'>
>>> isinstance(True,int)
True
>>> True == 1
True
>>> True + 4 #True это число 1
5
>>> False * 5 #False это число 0
0
>>>
```

Рисунок 10.1

Преобразования

Другие типы → bool

В Python для приведения других типов данных к булевому типу, используется функция bool() Работает эта функция по следующему соглашению:

Функция bool() вернет True:

– непустая строка (в том числе если это один или несколько пробелов);

– ненулевое число (в том числе меньшее единицы, например -5);

– непустой список/кортеж (даже если он содержит один пустой элемент, например пустой кортеж);

– функция.

Функция bool() вернет False:

– пустая строка;

– нулевое число;

– пустой список/кортеж.

```
>>> bool('python')
True
>>> bool('')
False
>>> bool(' ')
True
>>> bool(-5)
True
>>> bool((0,1))
True
>>> bool(0)
False
>>> bool(())
False
>>>
```

Рисунок 10.2

bool → str

Бывают ситуации, когда нам необходимо получить True и False в строковом представлении.

Для вывода на экран булевого значения, необходимо привести его к строке:

```
>>> print('my answer is ' + str(True))
my answer is True
>>>
```

Рисунок 10.3

bool → int

Встроенная функция int() преобразует логическое значение в 1 или 0.

```
>>> int(True)
1
>>> int(False)
0
>>>
```

Рисунок 10.4

Аналогичного результата можно добиться умножением логического типа на единицу:

```
>>> True * 1
1
>>> False * 1
0
>>>
```

Рисунок 10.5

Логический тип и операторы

Операторы — это своего рода функционал, представленный в виде символов (например + ==) или зарезервированных слов (например and not).

Говоря на естественном языке (например, русском) мы обозначаем сравнения словами "равно", "больше", "меньше". В языках программирования используются специальные знаки, подобные тем, которые используются в математике: > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), == (равно), != (не равно).

Не путайте операцию присваивания значения переменной, обозначаемую в языке Python одиночным знаком "равно", и операцию сравнения (два знака "равно"). Присваивание и сравнение – разные операции.

В Python используется несколько типов операторов. Мы же рассмотрим только **операторы сравнения** и **логические операторы**, т.к. результатом их выполнения являются True или False.

Операторы сравнения:

Операция	Запись в Python
>	>
<	<
=	==
≠	!=
≥	>=
≤	<=

Рисунок 10.6

```
>>> 1 == 5
False
>>> 1 != 5
True
>>> 1 > 5
False
>>> 1 < 5
True
>>> 1 >= 5
False
>>> 1 <= 5
True
>>>
```

Рисунок 10.7

Решим пример

```
>>> a=10
>>> b=5
>>> a+b>14
True
>>> a<14-b
False
>>> a<=b+5
True
>>> a==b
False
>>> c=a==b
>>> a,b,c
(10, 5, False)
>>>
```

Рисунок 10.8

В данном примере выражение c = a == b состоит из двух подвыражений. Сначала происходит сравнение (==) переменных *a* и *b*. После этого результат логической операции присваивается переменной *c*. Выражение a, b, c просто выводит значения переменных на экран.

Сложные логические выражения

Логические выражения типа kByte >= 1023 являются простыми, так как в них выполняется только одна логическая операция. Однако, на практике нередко возникает необходимость в более сложных выражениях. Может понадобиться получить ответа "Да" или "Нет" в зависимости от результата выполнения двух простых выражений. Например, "на улице идет снег или дождь", "переменная *news* больше 12 и меньше 20".

В таких случаях используются специальные операторы, объединяющие два и более простых логических выражения. Широко используются два оператора – так называемые логические И (and) и ИЛИ (or).

Чтобы получить **True при использовании оператора and**, необходимо, чтобы **результаты обоих простых выражений**, которые связывает данный оператор, **были истинными**. Если хотя бы в одном случае результатом будет False, то и все сложное выражение будет ложным.

Чтобы получить **True при использовании оператора or**, необходимо, чтобы **результат хотя бы одного простого выражения,** входящего в состав сложного, **был истинным**. В случае оператора or сложное выражение становится ложным лишь тогда, когда ложны оба составляющие его простые выражения.

Таблица 10.1

Операции	**Действие**	**Выражение**
and (конъюнкция)	Логическое И	A and B
or (дизъюнкция)	Логическое ИЛИ	A or B
not (отрицание)	Логическое НЕ	not A

Допустим, переменной *x* было присвоено значение 8 (x = 8), переменной *y* присвоили 13 (y = 13). Логическое выражение y < 15 and x > 8 будет выполняться следующим образом. Сначала выполнится выражение y < 15. Его результатом будет True. Затем выполнится выражение x > 8. Его результатом будет False. Далее выражение сведется к True and False, что вернет False.

```
>>> x=8
>>> y=13
>>> y<15 and x>8
False
>>>
```

Рисунок 10.9

Если бы мы записали выражение так: x > 8 and y < 15, то оно также вернуло бы False. Однако сравнение y < 15 не выполнялось бы интерпретатором, так как его незачем выполнять. Ведь первое простое логическое выражение (x > 8) уже вернуло ложь, которая, в случае оператора and, превращает все выражение в ложь.

В случае с оператором or второе простое выражение проверяется, если первое вернуло ложь, и не проверяется, если уже первое вернуло истину. Так как для истинности всего выражения достаточно единственного True, неважно по какую сторону от or оно стоит.

```
>>> x=8
>>> y=13
>>> y<15 or x>8
True
>>>
```

Рисунок 10.10

В языке Python есть еще унарный логический оператор not, то есть отрицание. Он превращает правду в ложь, а ложь в правду. Унарный он потому, что применяется к одному выражению, стоящему после него, а не справа и слева от него как в случае бинарных and и or.

```
>>> x=8
>>> y=13
>>> not y<15
False
>>>
```

Рисунок 10.11

Здесь y < 15 возвращает True. Отрицая это, мы получаем False.

```
>>> a=5
>>> b=0
>>> not a
False
>>> not b
True
>>>
```

Рисунок 10.12

Число 5 трактуется как истина, отрицание истины дает ложь. Ноль приравнивается к False. Отрицание False дает True.

Логические операторы:

```
>>> (1 + 1 == 2) or (2 * 2 == 5)
True
>>> (1 + 1 == 2) and (2 * 2 == 5)
False
>>> (1 + 1 == 2) and not (2 * 2 == 5)
True
>>>
```

Рисунок 10.13

Логические выражения, правила составления логических выражений

Логическая переменная – это простое высказывание, содержащее только одну мысль. Ее символическое обозначение – латинская буква (например, A, B, X, Y и т.д.). Значением логической переменной могут быть только константы ИСТИНА (1) и ЛОЖЬ (0). На основании простых высказываний могут быть построены составные высказывания.

Логическая функция - составное высказывание, которое содержит несколько простых мыслей, соединенных между собой с помощью логических операций.

Ее символическое обозначение – F (A, B, …).

Логические операции – логическое действие.

Решение логических выражений принято записывать в виде таблиц истинности – таблиц, в которых по действиям показано, какие значения принимает логическое выражение при всех возможных наборах его переменных.

При составлении таблицы истинности для логического выражения необходимо учитывать порядок выполнения логических операций, а именно:

– действия в скобках,

– ¬ (инверсия (отрицание (нет, not))),

– & (конъюнкция (и, and)),

– v (дизъюнкция (или, or)),

– => (импликация),

– <=> (эквивалентность).

Проанализируем составное высказывание "Если я куплю яблоки или абрикосы, то приготовлю фруктовый пирог". Обозначим буквой А высказывание: "Купить яблоки", буквой В - высказывание: "Купить абрикосы", буквой С - высказывание: "Испечь пирог". Запишем высказывание в виде логического выражения, высказывание "Если я куплю яблоки или абрикосы, то приготовлю фруктовый пирог" формализуется в виде формулы: F=(A v B)=>C.

```
>>> A='Купить яблоки'
>>> B='Купить абрикосы'
>>> C='Испечь пирог'
>>> F=(A or B)>=C
>>> F
True
```

Рисунок 10.14

Решим выражение F=(A v B) & (¬A v ¬B), используя таблицу истинности

Таблица 10.2

A	B
0	0
0	1
1	0
1	1

Сначала заводим переменные, далее решаем функцию.

```
>>> A=0
>>> B=0
>>> F=(A or B) and (not A or not B)
>>> F
0
>>> A=0
>>> B=1
>>> F=(A or B) and (not A or not B)
>>> F
True
>>> A=1
>>> B=0
>>> F=(A or B) and (not A or not B)
>>> F
True
>>> A=1
>>> B=1
>>> F=(A or B) and (not A or not B)
>>> F
False
>>>
```

Рисунок 10.15

Ответ у вас будет отображаться либо в числовом эквиваленте 0 или 1, либо True или False, что также соответствует 1 (True) или 0 (False)

Опишем две переменные f и k, переменной f присвоим значение 15, а переменной k присвоим значение 10

Составим сложные логические выражения, например

f>0 and k<20

f>k and f+k<20

f>k or f+k<20

f==0 or k == 0

Реализуем пример в программе

```
>>> f=15
>>> k=10
>>> f>0 and k<20
True
>>> f>k and f+k<20
False
>>> f>k or f+k<20
True
>>> f==0 or k == 0
False
>>>
```

Рисунок 10.16

Самостоятельное задание

1. Присвойте двум переменным любые числовые значения.
2. Используя переменные из п. 1, с помощью оператора and составьте два сложных логических выражения, одно из которых дает истину, другое – ложь.
3. Аналогично выполните п. 2, но уже с оператором or.
4. Решите выражения:
   F=A & ((¬ B) * (¬ C)), используя данные из таблицы истинности
   F= (A * (¬ B)) + (¬ C), используя данные из таблицы истинности
   F= (A * B) & (B*(¬ C)) v ((¬ A)+(¬ B)), используя данные из таблицы истинности

Таблица 10.3

A	B	C
0	0	0
0	0	1
0	1	0
1	0	0
0	1	1
1	0	1
1	1	0
1	1	1

5. Напишите программу, которая запрашивала бы у пользователя два числа и выводила бы True или False в зависимости от того, больше первое число второго или нет.

Практическая работа №10. Структура ветвление в Python. Условный оператор

Цель: Познакомиться с языком программирования Python. Изучить работу с условным оператором if..else и оператором вложенного цикла if.elif..else.

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

Ход выполнения программы может быть линейным, то есть таким, когда выражения выполняются друг за другом, начиная с первого и заканчивая последним. Ни одна строка кода программы не пропускается.

Однако чаще в программах бывает не так. При выполнении кода, в зависимости от тех или иных условий, некоторые его участки могут быть опущены, в то время как другие – выполнены.

Иными словами, в программе может присутствовать ветвление, которое реализуется условным оператором – особой конструкцией языка программирования.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr11_z1.py, обязательно поставьте расширение .py. Далее каждое задание делаем в новом файле и продолжаем нумерацию pr11_z2.py, pr11_z3.py и т.д.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Инструкция ветвления

Запись полной инструкции ветвления:

if логическое выражение:

P_1

…

P_n

else:

Q_1

…

Q_n

if (если), else (иначе) – зарезервированные слова, $P_1..P_n$, $Q_1..Q_n$ – операторы.

Оператор if - это блочный оператор, поэтому отступы в коде обязательны.

Вход в блок операторов осуществляется двоеточием.

Простой условный оператор работает по следующему алгоритму: сначала вычисляется логическое выражение. Если результат есть true (Истина), то выполняется оператор $P_1..P_n$, а оператор $Q_1..Q_n$ пропускается. Если результат есть false (Ложь), то выполняется оператор $Q_1..Q_n$, а оператор $P_1..P_n$ пропускается.

Напишем пример, в котором в зависимости от результата сравнения двух переменных выводится тот или иной ответ.

```
a=int(input('Введите значение a:\n'))
b=int(input('Введите значение b:\n'))
if a<b:
        print('Тест ветки 1')
else:
        print('Тест ветки 2')
```

Запись сокращенной инструкции ветвления:

if логическое выражение:

P_1

P_2

…

P_n

if (если) – зарезервированное слово, $P_1..P_n$ – операторы.

Решим следующий пример

```
a = int(input('Введите значение a:\n'))
n = int(input('Введите значение n:\n'))
if n < 100:
```

```
        b = n + a
        print(b)
```

Следует иметь в виду, что логическое выражение при if может выглядеть "нестандартно", то есть не так просто, как a > b и тому подобное. Там может стоять просто одна переменная, число, слово True или False, а также сложное логическое выражение, когда два простых соединяются через логически И (and) или ИЛИ (or).

```
a = ? #программа выдаст ошибку, этот код писать не нужно
if a: #программа выдаст ошибку, этот код писать не нужно
   a = 1  #программа выдаст ошибку, этот код писать не нужно
```

Если вместо знака вопроса будет стоять a=0, то с логической точки зрения это False, значит выражение в if не будет выполнено. Если *a* будет связано с любым другим числом, то оно будет расцениваться как True, и тело условного оператора выполнится.

Решим следующий пример:

```
a=0>0
if a:
        print(a)
```

Здесь *a* уже связана с булевым значением. В данном случае это True.

На будущее, если вы сомневаетесь в последовательности выполнения операторов, используйте скобки, например так: a = (5 > 0).

Решим следующий пример:

```
a=int(input('Введите a:\n'))
b=int(input('Введите b:\n'))
if a > 0 and a < b:
        print('c= ',b - a)
```

Тут, чтобы вложенный код выполнился, *a* должно быть больше нуля и одновременно меньше *b*. Также в Питоне, в отличие от других языков программирования, позволительна такая сокращенная запись сложного логического выражения: 0 < a < b

Решим задачу с использование блок схемы разобранной ранее

Перед выходным днем папа сказал своему сыну: «Давай спланируем свой завтрашний день. Если будет хорошая погода, то проведем день в лесу. Если же погода будет плохая, то сначала займемся уборкой квартиры, а во второй половине дня сходим в зоопарк». Что получится на выходе блок-схемы, если: а) погода хорошая; б) погода плохая? Для решения задачи нам нужно решить, что мы отнесем к плохой погоде, а что к хорошей. Будем считать, что если погода больше 10^0, то она хорошая, если меньше, то плохая.

Решение на Python

```
print('Прогулка или уборка')
answer = int(input('Введите сколько градусов на улице\n'))
if answer>10:
        print('Идем гулять в лес')
else:
```

pogoda=print('Всего '+str(answer)+' градусов - очень холодно. Займись уборкой')

print('Позже пойдем в зоопарк')

Самостоятельные задания:

1. Напишите программу, которая просит пользователя что-нибудь ввести с клавиатуры. Если он вводит какие-нибудь данные, то на экране должно выводиться сообщение "ОК". Если он не вводит данные, а просто нажимает Enter, то программа ничего не выводит на экран.
2. Напишите программу, которая запрашивает у пользователя число. Если оно больше нуля, то в ответ на экран выводится число 1. Если введенное число не является положительным, то на экран должно выводиться -1.
3. Напишите программу, которая вводит с клавиатуры число, затем прибавляет к этому числу 24, и проверяет полученный результат на условие. Если полученный результат больше 40, то вывести сообщение «Число {..} заданным критериям», иначе вывести сообщение об ошибке. Переменные для ввода и проверки условия, то разные переменные.
4. Найдите максимальное значение, введенное с клавиатуры, из двух чисел

Практическая работа №11. Структура ветвление в Python. Вложенный условный оператор и "иначе-если"

Цель: Познакомиться с языком программирования Python. Изучить работу с условным оператором if..else и оператором вложенного цикла if.elif..else.

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

Ход выполнения программы может быть линейным, то есть таким, когда выражения выполняются друг за другом, начиная с первого и заканчивая последним. Ни одна строка кода программы не пропускается.

Однако чаще в программах бывает не так. При выполнении кода, в зависимости от тех или иных условий, некоторые его участки могут быть опущены, в то время как другие – выполнены.

Иными словами, в программе может присутствовать ветвление, которое реализуется условным оператором – особой конструкцией языка программирования.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr12_z1.py, обязательно поставьте расширение .py. Далее каждое задание делаем в новом файле и продолжаем нумерацию pr12_z2.py, pr12_z3.py и т.д.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Сложные условия. Каскадные ветвления

Ранее мы рассмотрели работу условного оператора if. С помощью его расширенной версии if-else можно реализовать две отдельные ветви выполнения. Однако алгоритм программы может предполагать выбор больше, чем из двух путей, например, из трех, четырех или даже пяти. В данном случае следует говорить о необходимости множественного (каскадного) ветвления.

Запись каскадного ветвления:

if <условие 1>:
  <инструкции 1>
elif <условие 2>:
  <инструкции 3>
elif <условие 3>:
  <инструкции 4>
else:
  <инструкции 2>

Слово "elif" образовано от двух первых букв слова "else", к которым присоединено слово "if". Это можно перевести как "иначе если". В отличие от else, в заголовке elif обязательно должно быть логическое выражение также, как в заголовке if.

Обратите внимание, в конце, после всех elif, может использоваться одна ветка else для обработки случаев, не попавших в условия ветки if и всех elif. Блок-схему полной конструкции if-elif-…-elif-else можно изобразить так:

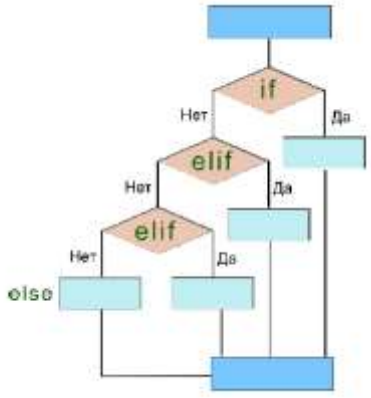


Рисунок 12.1

Рассмотрим конкретный пример. Допустим, в зависимости от возраста пользователя, ему рекомендуется определенный видеоконтент. При этом выделяют группы от 3 до 6 лет, от 6 до 12,от 12 до 16, 16+. Итого 4 диапазона.

old = int(input('Ваш возраст: '))

```
print('Рекомендовано:', end=' ')
if 3 <= old < 6:
        print('"Заяц в лабиринте"')
elif 6 <= old < 12:
        print('"Марсианин"')
elif 12 <= old < 16:
        print('"Загадочный остров"')
elif 16 <= old:
        print('"Поток сознания"')
else:
        print('Для вас нет рекомендаций')
```

Реализуем пример из лекции, который спрашивает вашу любимую игру, а затем печатает результат. Далее выводит количество часов для игры. Можете заполнить своими играми.

```
game=input("Введите вашу любимую игру:")
print ("Ваша любимая игра -",game)
if game =="CS:GO":
        print ("Вы наиграли в ней 420 часов")
elif game == "Mass Effect 2":
        print("Вы наиграли в ней 70 часов")
else:
        print("Странно, вы в нее не играли")
```

Дополните выше созданный пример еще 3 играми (для этого будем использовать elif).

Найдите наименьшее значение из трех чисел, используя блок – схему из лекции. Решение задачи представлено без использования elif, но при решении задач, лучше избегать такой способ решения.

Решение на Python

```
a=int(input('a=\n'))
b=int(input('b=\n'))
c=int(input('c=\n'))
min=a
if b<min:
        min=b
if c<min:
        min=c
print('Минимальное из трех чисел =',min)
```

Вычислите значение функции y:

$$y=\begin{cases} \mathrm{Sin}(x), & \text{если } x < 0 \\ \mathrm{Cos}(x), & \text{если } 0 \le x \le 1 \\ \mathrm{Tg}(x), & \text{если } x > 1 \end{cases}$$

Алгоритм решения

```
from math import*
```

```
x=float(input('Введите x:\n'))
if x<0:
        y=sin(x)
elif x>1:
        y=tan(x)
else:
        y=cos(x)
print('\nРезультат: ',y)
```

Самостоятельные задания:

1. Реализуйте задачу из лекции сравнения возраста трех человек
2. Найдите наименьшее значение из трех чисел, используя блок – схему из лекции. Решение задачи представьте с использования elif.
3. Напишите программу, которая запрашивает на ввод число. Если оно положительное, то на экран выводится цифра 1. Если число отрицательное, выводится -1. Если введенное число – это 0, то на экран выводится 0. Используйте в коде условный оператор множественного ветвления.

Дополнительные задания:

1. Зайдите в Znanium под свой учетной записью.
2. Найдите учебник С.Р. Гуриков Основы алгоритмизации и программирования на Python (https://znanium.com/catalog/document?id=390096)
3. Решите самостоятельные задания со страниц 56-57

Практическая работа №12. Работа с циклами в Python. Цикл While

Цель: Познакомиться с языком программирования Python. Изучить работу с циклом While.

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

Зачем нужны циклы

Прикладные программы помогают управлять сотрудниками, финансами и могут развлекать. Несмотря на различия, они выполняют заложенные в них алгоритмы, которые похожи. **Алгоритм** — это последовательность действий, которая приводит к ожидаемому результату.

Представим, что у нас есть книга, и мы хотим найти в ней конкретную фразу. Саму фразу мы помним, но не знаем, на какой она странице. Нам придется последовательно просматривать страницы до тех пор, пока не найдем нужную. Этот процесс и называется алгоритмом.

Алгоритм включает логические проверки и перебор страниц. Количество страниц, которое придется посмотреть, заранее неизвестно. Но сам процесс просмотра повторяется одинаковым образом. Чтобы выполнять повторяющиеся действия, нужны циклы. Каждый повтор называется **итерацией**.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить

оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

«While» переводится с английского как «пока». Но не в смысле «до свидания», а в смысле «пока имеем это, делаем то».

Можно сказать, while является универсальным циклом. Он присутствует во всех языках, поддерживающих структурное программирование, в том числе в Python. Его синтаксис обобщенно для всех языков можно выразить так:

while <логическое_выражение>:

<тело цикла>:

Это похоже на условный оператор if. Однако в случае циклических операторов их тела могут выполняться далеко не один раз. В случае if, если логическое выражение в заголовке возвращает истину, то тело выполняется единожды. После этого поток выполнения программы возвращается в основную ветку и выполняет следующие выражения, расположенные ниже всей конструкции условного оператора.

В случае while, после того как его тело выполнено, поток возвращается к заголовку цикла и снова проверяет условие. Если логическое выражение возвращает истину, то тело снова выполняется. Потом снова возвращаемся к заголовку и так далее.

Цикл завершает свою работу только тогда, когда логическое выражение в заголовке возвращает ложь, то есть условие выполнения цикла больше не соблюдается. После этого поток выполнения перемещается к выражениям, расположенным ниже всего цикла. Говорят, "происходит выход из цикла".

Рассмотрите блок-схему цикла while.

На ней ярко-голубыми прямоугольниками обозначена основная ветка программы, ромбом – заголовок цикла с логическим выражением, бирюзовым прямоугольником – тело цикла.

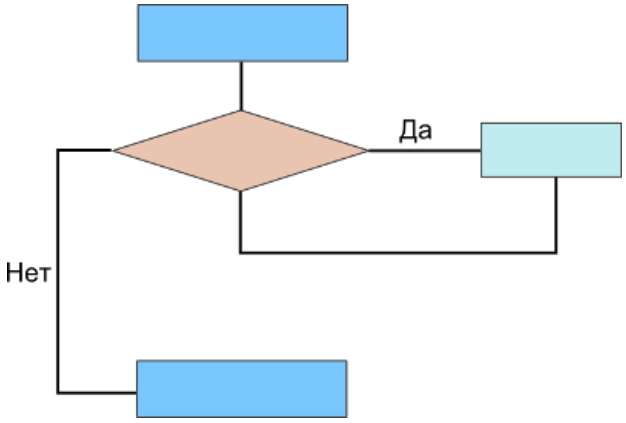


Рисунок 13.1

С циклом while возможны две исключительные ситуации:

- Если при первом заходе в цикл логическое выражение возвращает False, то тело цикла не выполняется ни разу. Эту ситуацию можно считать нормальной, так как при определенных условиях логика программы может предполагать отсутствие необходимости в выполнении выражений тела цикла.
- Если логическое выражение в заголовке while никогда не возвращает False, а всегда остается равным True, то цикл никогда не завершится, если только в его

теле нет оператора принудительного выхода из цикла (break) или вызовов функций выхода из программы – quit(), exit() в случае Python. Если цикл повторяется и повторяется бесконечное количество раз, то в программе происходит зацикливание. В это время она зависает и самостоятельно завершиться не может.

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr13_z1.py, обязательно поставьте расширение .py. Далее каждое задание делаем в новом файле и продолжаем нумерацию pr13_z2.py, pr13_z3.py и т.д.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Цикл while (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл while используется, когда невозможно определить точное значение количества проходов исполнения цикла.

Задача 1

```
i = 0
while i < 5:
                print(i)
                i += 1
```

Рисунок 13.2

Цикл while и цикл for (с ним познакомимся позже) имеют схожую структуру, но есть одно важное различие - цикл while может быть бесконечным.

Задача 2

```
i = 0
while True:
    print(i)
    i += 1
```

Рисунок 13.3

Код выше будет бесконечно печатать возрастающую последовательность чисел.

Цикл while можно сравнить с цикличным условным оператором.

Задача 3

```
text = 'Hello world'
i = 0
while i < len(text):
    print(text[i])
    i += 1
```

Рисунок 13.4

Код, приведенный выше, печатает строку посимвольно.

Цикл while, как и цикл for, можно остановить с помощью специальной управляющей конструкции break.

Задача 4

```
j = 0
while True:
    if j == 3:
        print('Выход из цикла')
        break
    print(j)
    j += 1
```

Рисунок 13.5

Конструкция break прерывает цикл. Она очень похожа на обычное условие после ключевого слова while.

Так же есть еще одна управляющая конструкция - continue. С ее помощью мы можем не выполнять текущую итерацию (повторение) цикла и перейти сразу к следующему.

Задача 5

```
j = 0
while j < 5:
    j += 1
    if j == 3:
        print('Пропускаем j == 3')
        continue
    print(j)
```

Рисунок 13.6

Задача 6

Для цикла while мы можем записать конструкцию if..else.

```
from random import randint
j = 0
element = randint(0, 15)
while j < 10:
    j += 1
    if j == element:
        print('Нашли element, он равен', element)
        break
else:
    print('Неудачная попытка')
```

Рисунок 13.7

В первой строке мы подключаем выбор случайного элемента randint из библиотеки random

Задача 7

Возведение числа в степень с помощью цикла while

```
n = int(input()) # число
k = int(input()) # степень
i = 1 # текущая степень
result = 1
while i <= k:
    result *= n
    i += 1
print(result)
```

Рисунок 13.8

Задача 8

Сумма последовательности с помощью цикла while

```
n = int(input())
result = 0
i = 0
while i <= n:
    result += i
    i += 1
print(result)
```

Рисунок 13.9

Задача 9

Ввод последовательности чисел

```
i = 0
while True:
    n = input()
    if n == 'end':
        print('Ввод закончен')
        print('Было введено', i, 'чисел')
        break
    n = int(n)
    i += 1
```

Рисунок 13.10

Задача 10

Сумма введенных чисел

```
i = 0
summa = 0
while True:
    n = input()
    if n == 'end':
        print('Ввод закончен')
        print('Было введено', i, 'чисел')
        print('Их сумма равна', summa)
        break
    n = int(n)
    summa += n
    i += 1
```

Рисунок 13.11

Вспомним наш пример из урока про исключения. Пользователь должен ввести целое число. Поскольку функция input() возвращает строку, то программный код должен преобразовать введенное к целочисленному типу с помощью функции int(). Однако, если были введены символы, не являющиеся цифрами, то возникает исключение ValueError, которое обрабатывается веткой except. На этом программа завершается.

Другими словами, если бы программа предполагала дальнейшие действия с числом (например, проверку на четность), а она его не получила, то единственное, что программа могла сделать, это закончить свою работу досрочно.

Но ведь можно просить и просить пользователя корректно вести число, пока он его не введет. Вот как может выглядеть реализующий это код:

Задача 11 (комментарии можно не печатать, хотя бы прочитайте их)

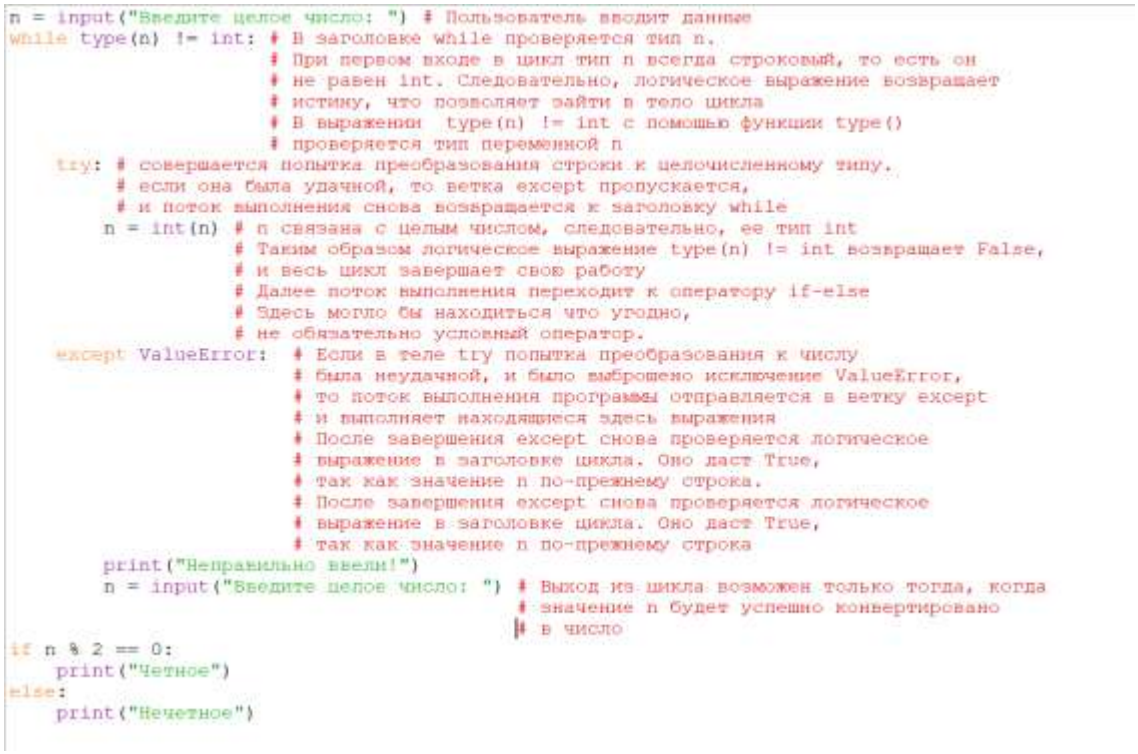


Рисунок 13.12

Примечание 1. Не забываем, в языке программирования Python в конце заголовков сложных инструкций ставится двоеточие.

Примечание 2. В выражении type(n) != int с помощью функции type() проверяется тип переменной *n*. Если он не равен **int**, то есть значение *n* не является целым числом, а является в данном случае строкой, то выражение возвращает истину. Если же тип *n* равен **int**, то данное логическое выражение возвращает ложь.

Примечание 3. Оператор **%** в языке Python используется для нахождения остатка от деления. Так, если число четное, то оно без остатка делится на 2, то есть остаток будет равен нулю. Если число нечетное, то остаток будет равен единице.

Задача 12

Разработаем цикл, который будет обрабатывать данные до тех пор, пока не будет введено слово студент

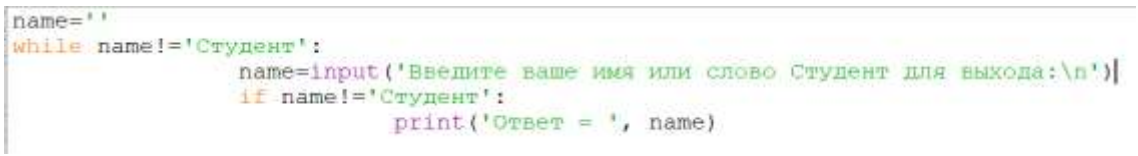


Рисунок 13.13

При запуске программы, даже если вы введете слово студент (но оно будет написано с маленькой буквы), то программа не закончит свою работу, т.к. мы указали, что слово студент должно писаться с большой буквы. Исправим это, добавив .capitalize() после строки input()

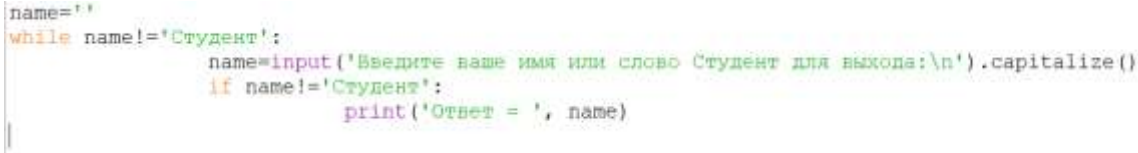
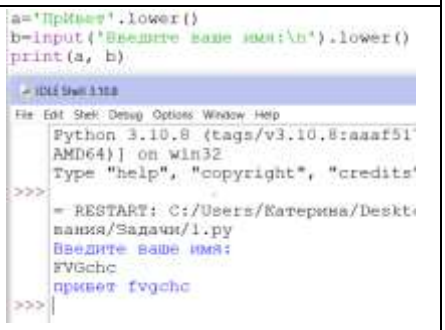
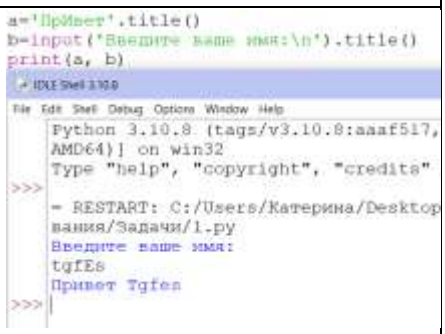
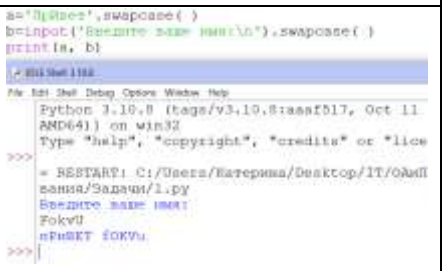


Рисунок 13.4

Таблица 13. 1

Строковый метод	Синтаксис	Описание	Пример
upper	строка.upper()	Преобразует все буквы строки(str) в буквы верхнего регистра. Возвращает - строку(str), копию оригинальной строки(str) состоящую из букв верхнего регистра.	
lower	строка.lower()	Преобразует все буквы строки(str) в буквы нижнего регистра. Возвращает - строку(str), копию оригинальной строки(str) состоящую из букв нижнего регистра.	
title	строка.title()	Преобразует первые буквы всех слов строки(str) в буквы верхнего регистра, все остальные буквы слов преобразует в буквы нижнего регистра. Возвращает – строку(str), копию оригинальной строки(str) в которой все слова начинаются с букв верхнего регистра, а все остальные буквы слов в нижнем регистре.	
swapcase	строка. swapcase()	Преобразуем все буквы верхнего регистра в буквы нижнего регистра, а буквы нижнего регистра преобразует в буквы нижнего регистра. Возвращает – строку(str), копию оригинальной строки(str) в которой все буквы верхнего регистра заменены буквами нижнего регистра, а буквы нижнего регистра заменены буквами верхнего регистра.	

capitalize	строка.capitalize()	Преобразует первую букву первого слова строки(str) в букву в верхнем регистре, все остальные буквы сроки(str) преобразуются в буквы в нижнем регистре. Возвращает – строку(str), копию оригинальной строки(str) у которой первая буква первого слова строки в верхнем регистре, все остальные буквы строки в нижнем регистре.	

Задача 13

Посчитаем сумму первых 50 чисел

Рассмотрим алгоритм решения задачи

Прежде всего, надо позаботиться о том, чтобы какая-нибудь переменная менялась в цикле от 1 до 50. Ведь такой величины, как параметр цикла, нет в конструкции while. В нашем примере такую роль будет играть переменная к. Задав в качестве условия выхода из цикла к!= 50 и применяя в цикле, оператор sum=sum+k, мы просуммируем все пятьдесят слагаемых и получим в ответе число 1275.

```
k=0
sum=0
while k!=50:
                    k=k+1
                    sum=sum+k
                    print('Сумма чисел от 1 до 50 =', sum)
```

Рисунок 13.5

Задача 14

Решим следующую задачу

Среди чисел $1{,}1+\frac{1}{2}, 1+\frac{1}{2}+\frac{1}{3}, \ldots$ найдите первое число, большее вводимого значения переменной a

Комментарий. Алгоритм решения данной задачи относится к алгоритмам вычисления членов бесконечных последовательностей. Очередной член бесконечной последовательности обозначим как Ь. Его номер, который совпадает со значением знаменателя дроби, добавляемой к предыдущему члену для получения значения очередного члена последовательности, обозначим как п. Тогда итерационная формула вычисления очередного члена последовательности примет вид: $b_n = b_{n-1} + \frac{1}{n}$

```
a=float(input('Введите число a: '))
b=1
n=1
print("\n n '+' b")
print()
while b<a:
```

```
n+=1
b=b+1/n
print(" ", n, "     ",b)
 print("Первое число, превышающее значение a:  %.4f " %(b), "Количество итераций", n)
```

Задача 15

Решим следующий пример

Используя цикл while, выведите на экран для числа 2 его степени от 0 до 20. Возведение в степень в Python обозначается как **.

```
i = 0
while i <= 20:
print("%7d" % 2**i)
i += 1
```

Благодаря форматированному выводу ("%7d") числа выравниваются по правому краю поля из семи знакомест. Это позволяет выводить единицы под единицами, десятки под десятками и т. д. Обратите внимание, что переменная i здесь используется и как счетчик для цикла, и как показатель степени.

Задача 16

Реализуйте следующие примеры, используя, представленные блок – схемы:

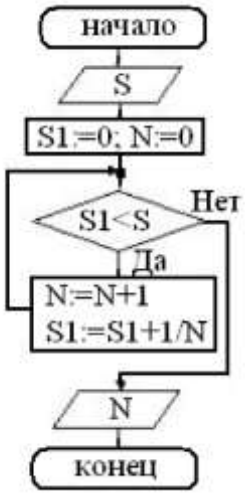


Рисунок 13.6

В вывод добавьте переменную S1 и укажите, чтобы программа выводила четыре знака после запятой.

Задача 17

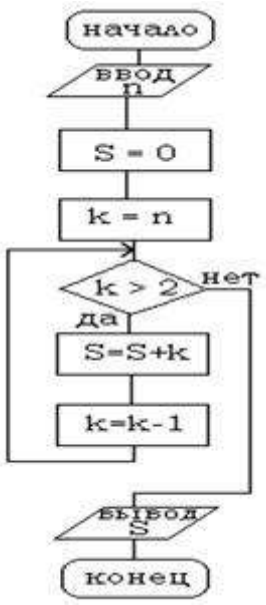


Рисунок 13.7

Вводимые числа должны быть целыми

Задача 18

Вводится последовательность вещественных чисел. Известно, что последний элемент последовательности равен пяти. Найдите количество положительных чисел и минимальное из них. Алгоритм решения представлен ниже.

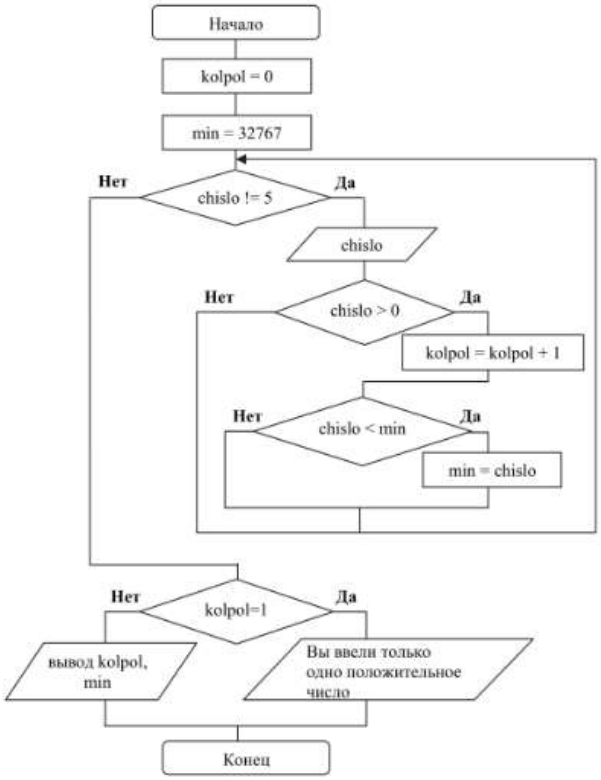


Рисунок 13.8

```
min=32767
chislo=0
while chislo!=5: # Пока число, введенное в цикле не равно 5, выполнять тело цикла
        chislo=float(input('Vvedite chislo: '))
```

```
            if chislo > 0: #Проверка на положительность очередного введенного числа
                kolpol = kolpol+1
                if chislo < min: # Реализация алгоритма поиска минимального
элемента
                    min = chislo
    if kolpol == 1:
        print('Вы ввели только одно число: 5, предназначенное для выхода из
программы')
    else:
        print('Количество положительных чисел = ', kolpol)
        print('Минимальное из них = ',min)
```

Задача 19

Найдите значение функции на y=sin(x) на отрезке [1;10] с шагом 1

```
from math import*
x=1 # Задаем первую точку в отрезке
while x<=10: # Проверяем отрезок от [1;10]
        y=sin(x) # Решаем уравнение
        x=x+1 # увеличиваем шаг на 1
        print("y= %.3f" % y)
```

Рисунок 13.9

Задача 20

Найдите значение функции на $y=5x^2-2x+1$ на отрезке [-5;5] с шагом 2

Дополнительные задания *

Задача 21

Вводится последовательность вещественных чисел. Известно, что последний элемент последовательности равен пяти. Определите, каких из них больше: положительных или отрицательных. Алгоритм решения представлен ниже.

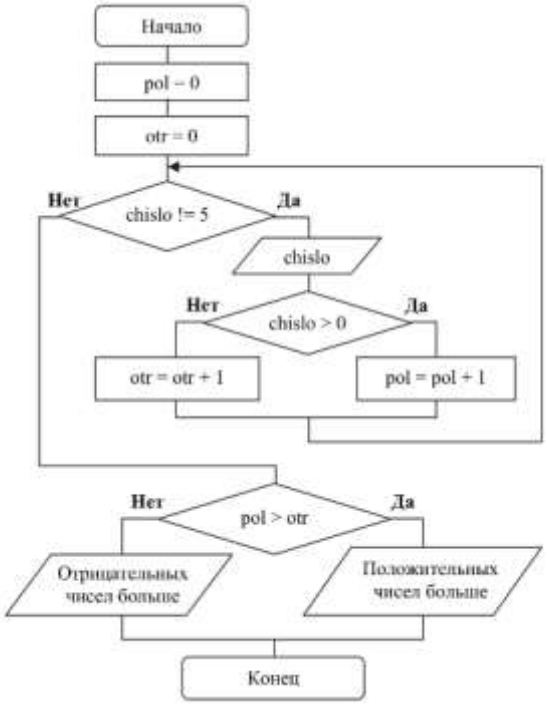


Рисунок 13.10

Задача 22

Вводится последовательность целых чисел, не равных нулю. Известно, что последний элемент последовательности равен нулю. Найдите среднее арифметическое этих чисел. Алгоритм решения представлен ниже.

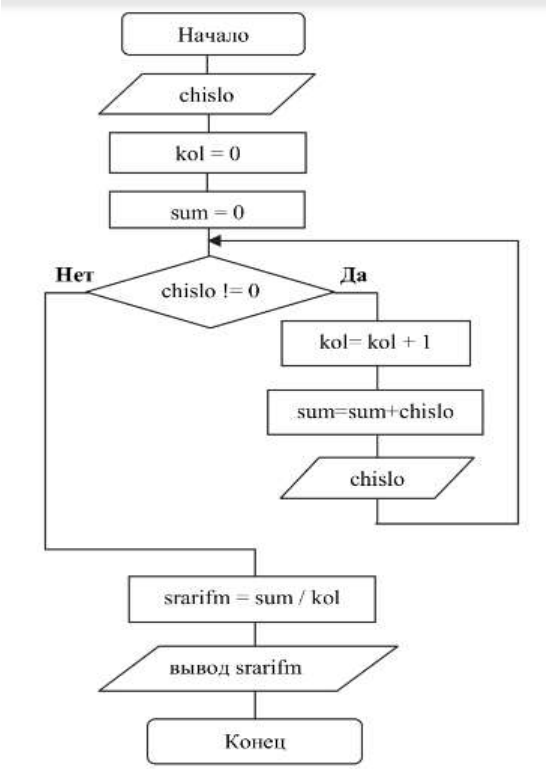


Рисунок 13.11

Задача 23

Дано положительное число N. Вывести все числа от 0 до N с помощью цикла while.

Задача 24

Дано положительное число N. Вывести все числа от N до 0 с помощью цикла while.
Пример:

Ввод: N = 10

Вывод: 10

9

8

7

6

...

0

Задача 25

Даны два положительных числа K и N (K < N). Вывести все числа от K до N с помощью цикла while.

Задача 26

Даны положительные числа A и B (A > B). На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Не используя

операции умножения и деления, найти длину незанятой части отрезка A (взятие остатка A % B)

Задача 27

Даны положительные числа A и B (A > B). На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Не используя операции умножения и деления, найти количество отрезков B, размещенных на отрезке A (деление нацело A // B)

Задача 28

Дано положительное число N. Найти сумму всех четных чисел от 0 до N с помощью цикла while.

Задача 29

Даны два положительных числа K и N (K < N). Найти сумму всех нечетных чисел от K до N с помощью цикла while.

Задача 30

Дано положительное число N. Найти факториал числа N. Факториалом числа называется произведение всех чисел от 1 до N. Например, факториал числа 5 равен 5! = 1*2*3*4*5 = 120, 2! = 1*2 = 2, 9! = 1*2*3*4*5*6*7*8*9 = 362880

Практическая работа №13. Работа с циклами в Python. Цикл For

Цель: Познакомиться с языком программирования Python. Изучить работу с циклом While.

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

Зачем нужны циклы

Прикладные программы помогают управлять сотрудниками, финансами и могут развлекать. Несмотря на различия, они выполняют заложенные в них алгоритмы, которые похожи. **Алгоритм** — это последовательность действий, которая приводит к ожидаемому результату.

Представим, что у нас есть книга, и мы хотим найти в ней конкретную фразу. Саму фразу мы помним, но не знаем, на какой она странице. Нам придется последовательно просматривать страницы до тех пор, пока не найдем нужную. Этот процесс и называется алгоритмом.

Алгоритм включает логические проверки и перебор страниц. Количество страниц, которое придется посмотреть, заранее неизвестно. Но сам процесс просмотра повторяется одинаковым образом. Чтобы выполнять повторяющиеся действия, нужны циклы. Каждый повтор называется **итерацией**.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить

оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Цикл for, также называемый циклом с параметром, в языке Питон богат возможностями. В цикле for указывается переменная и множество значений, по которому будет пробегать переменная. Множество значений может быть задано списком, кортежем, строкой или диапазоном.

Конструкция цикла for

for <перебор итерируемого объекта>:

<осуществление операций над каждым элементом>

else:

<обработается только при несрабатывании инструкции «**break**»>

Цикл **for** позволяет перебирать элементы по индексу или напрямую.

Оба вида циклических структур могут включать условные выражения и специальные **«прерыватели»**: **continue**, **break**.

Для успешного решения заданий необходимо понимать устройство циклов в Python, уметь использовать вложенные циклы, стремиться минимизировать количество итераций (при такой возможности).

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr13_z1.py, обязательно поставьте расширение .py. Далее каждое задание делаем в новом файле и продолжаем нумерацию pr13_z2.py, pr13_z3.py и т.д.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Цикл for в Python

Цикл в любом языке программирования - это многократное выполнение одного и то же действия.

В Python цикл for - это цикл перебора последовательности. Он состоит из двух компонент: переменной (переменных) цикла и последовательности.

Задание 1

```
for item in 'one', 'two', 'three':
print(item)
```

В приведенном выше примере переменная цикла item по очереди принимает каждое значение последовательности, которая записана после служебного слова in. На первом повторении переменная item равна строке 'one', на втором - строке 'two', на третьем - строке 'three'.

Вывод:

```
one
two
three
```

Задание 2

```
for element in '1', 'hello', 2, 1990, True, False:
    print(element)
```

Вывод:

```
1
hello
2
1990
True
False
```

Задание 3

```
for letter in 'Hello world':
    print(letter)
```

```
H
e
l
l
o
w
o
r
l
d
```

Функция range()

Функция range() возвращает диапазон (последовательность) целых чисел.

range() может принимать 1, 2 или 3 аргумента.

Задание 4

Примеры функции range() с одним аргументом аргументами (возвращает определенное количество чисел начиная с 0):

```
# range() с одним аргументом
print('range(3) -> 0, 1, 2')
for x in range(3): # range(3) -> 0, 1, 2
    print(x)
print('range(5) -> 0, 1, 2, 3, 4')
for x in range(5): # range(5) -> 0, 1, 2, 3, 4
    print(x)
# range(1) -> 0
# range(6) -> 0, 1, 2, 3, 4, 5
# range(10) -> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Вывод:

```
range(3) -> 0, 1, 2
0
1
```

2
range(5) -> 0, 1, 2, 3, 4
0
1
2
3
4
Уберите комментарии со следующих строк
range(1) -> 0
range(6) -> 0, 1, 2, 3, 4, 5
range(10) -> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Дополните код программы выводом этих строк
Задание 5
Примеры функции range() с двумя аргументами (возвращает числа в указанном диапазоне):

```
print('range(3, 5) -> 3, 4')
for x in range(3, 5): # range(3, 5) -> 3, 4
   print(x)
print('range(1, 5) -> 1, 2, 3, 4')
for x in range(1, 5): # range(1, 5) -> 1, 2, 3, 4
   print(x)
# range(2, 7) -> 2, 3, 4, 5, 6
# range(100, 105) -> 100, 101, 102, 103, 104
# range(50, 53) -> 50, 51, 52
```

Вывод:
range(3, 5) -> 3, 4
3
4
range(1, 5) -> 1, 2, 3, 4
1
2
3
4
Уберите комментарии со следующих строк
range(2, 7) -> 2, 3, 4, 5, 6
range(100, 105) -> 100, 101, 102, 103, 104
range(50, 53) -> 50, 51, 52
Дополните код программы выводом этих строк
Задание 6
Примеры функции range() с тремя аргументами аргументами (возвращает числа в указанном диапазоне с заданным шагом (шаг это 3 число)):

```
print('range(0, 10, 2) -> 0, 2, 4, 6, 8')
```

```
for x in range(0, 10, 2): # range(0, 10, 2) -> 0, 2, 4, 6, 8
    print(x)
print('range(1, 10, 2) -> 1, 3, 5, 7, 9')
for x in range(1, 10, 2): # range(1, 10, 2) -> 1, 3, 5, 7, 9
    print(x)
print('range(10, 20, 3) -> 10, 13, 16, 19')
for x in range(10, 20, 3): # range(10, 20, 3) -> 10, 13, 16, 19
    print(x)
# range(2, 7, 5) -> 2
# range(1, 8, 4) -> 1, 5
# range(4, 20, 5) -> 4, 9, 14, 19
```

Вывод

```
range(0, 10, 2) -> 0, 2, 4, 6, 8
0
2
4
6
8
range(1, 10, 2) -> 1, 3, 5, 7, 9
1
3
5
7
9
range(10, 20, 3) -> 10, 13, 16, 19
10
13
16
19
```

Уберите комментарии со следующих строк

```
# range(2, 7, 5) -> 2
# range(1, 8, 4) -> 1, 5
# range(4, 20, 5) -> 4, 9, 14, 19
```

Дополните код программы выводом этих строк

Задание 7

Перебор строк и функция len()

С помошью цикла for мы можем перебрать любую последовательность, например, строку

Евгению предоставили строку, состоящую из русских букв разных регистров, и попросили очистить ее от заглавных литер.

Как ему показалось, он написал верный код, но результат совсем не порадовал.

Ниже представлен пример работы «чистильщика строк», которому срочно требуется ваша помощь.

```
letters = 'ЫгВЫоЯСремДШНККАыкЩЙФа'
for letter in letters:
        if letter.upper() = letters:
                letters.replace(letter, '')
                print(letters)
```

Если запустить код Евгения, то получаем синтаксическую ошибку. Вот уж действительно горе-ученик.

Что ж, давайте думать, что нужно исправить:

- в условии **if** явно не хватает второго знака равно, так как мы не присваиваем переменную, а проверяем ее истинность;
- каждую букву в верхнем регистре нужно сравнивать не со всем набором символом, а с одним знаком;
- заменяя буквы на пустые символы мы создаем новую строку, которую никак не сохраняем. В итоге изначальный объект не поменяется. Нужно создать пустую строку и дополнять ее верными символами, пропуская ненужные. А раз требуется сохранить прежнее название строки, то мы переприсвоим исходной вновь полученную.

Исправленное решение:

```
letters = 'ЫгВЫоЯСремДШНККАыкЩЙФа'
clean_string = ''
for letter in letters:
   if not letter.isupper():
      clean_string += letter
letters = clean_string
print(letters)
```

Задача 8. Факториал

Факториалом числа n называется произведение 1 × 2 × … × n. Обозначение: n!.

По данному натуральному n вычислите значение n!.

Пользоваться математической библиотекой math в этой задаче запрещено.

```
res = 1
n = int(input('Введите число: '))
for i in range(1, n + 1):
   res *= i
print('res = ', res)
```

Задача 9. Сумма факториалов

По данному натуральном n вычислите сумму 1!+2!+3!+...+n!1!+2!+3!+…+n!. В решении этой задачи можно использовать только один цикл. Пользоваться математической библиотекой math в этой задаче запрещено.

```
n = int(input('Введите число: '))
partial_factorial = 1
partial_sum = 0
```

```
for i in range(1, n + 1):
    partial_factorial *= i
    partial_sum += partial_factorial
print('partial_sum = ', partial_sum)
```

Задача 10. Сумма кубов

По данному натуральному n вычислите сумму $1^3+2^3+3^3+\ldots+n^3$.

```
n = int(input('Введите число: ')
sum = 0
for i in range(1, n + 1):
    sum += i ** 3
print('sum =', sum)
```

Задача 11. Сумма 10 чисел

Дано 10 целых чисел. Вычислите их сумму.

Напишите программу, использующую наименьшее число переменных.

```
sum = 0
for i in range(10):
    number = int(input('Введите число: '))
    sum += number
print('sum =', sum)
```

Задача 12

Даны два целых числа A и B (при этом $A \leq B$). Выведите все числа от A до B включительно.

```
a = int(input('Введите число: '))
b = int(input('Введите число: '))
for i in range(a, b + 1):
    print('i =', i)
```

Задача 13

Посчитать сумму чисел от 0 до N

```
N = int(input('Введите число: '))
summa = 0
for i in range(N+1):
    # summa = summa + i
    summa += i
print('summa =', summa)
```

Задача 14

Посчитать сумму четных чисел от 0 до N

```
N = int(input('Введите число: '))
summa = 0
for i in range(N+1):
    if i % 2 == 0:
        # summa = summa + i
        summa += i
```

```
print('summa =', summa)
```

Задача 15

Разработайте приложение, которое осуществляет вывод на экран таблицы умножения для значений от 1 до 5

```
for i in range(1,6):
    for j in range (1,6):
        print(i, '*', j, '=', i*j, end='\t')
print()
```

Решение задач

Примечание. Все задачи решаем только через for и без применение не рассмотренных в процессе обучения функций и команд.

Задача 16

Выведите числа от 0 до N включительно. Число N задаем с клавиатуры

Задача 17

Выведите числа от K до N. Выведите все числа от K до N включительно. Числа K и N задаем с клавиатуры

Задача 18

Дано 10 целых чисел. Вычислите их произведение.

Задача 19

Посчитать сумму нечетных чисел от 0 до 15

Задача 20

Введите числа от K до N. Посчитайте произведение чисел от K до N включительно.

Задача 21

Введите числа K и N. Выведите сумму только четных чисел от K до N включительно.

Задача 22

Вывести числа, делящиеся на три без остатка, в диапазоне от 0 до n.

Задача 23

Найти произведение чисел от 1 до n, делящихся на 3

Задача 24

Введите число N. Найдите сумму чисел: 1 + 1.1 + 1.2 + 1.3 + ...

Задача 25

Разработайте приложение, которое осуществляет вывод на экран таблицы умножения

Задача 26

Дано целое число n. Найти сумму 1 + 1/2 + 1/3 + ... + 1/n

Задача 27

Дано целое число n. Найти сумму 1 + 2 + 4 + 8 + 16 + ... + 2**n где 2**n - это 2*2*2*...*2 раз. Таким образом, 2**4 = 2*2*2*2. Операция ** называется операцией возведения в степень.

Задача 28

Дано целое число n. Найти значение выражения 1.1 − 1.2 + 1.3 − ... (N слагаемых, знаки чередуются).

Задача 29

По заданной последовательности k_1, k_2, ……, k_n чисел вычислите сумму $k_1*k_2+k_2*k_3+\cdots+k_{n-1}*k_n$.

Задача 30

Вводятся вещественное число A и целое число N (> 0). Найти A в степени N

Задача 31

По выражению n ≤ 30 выведите лесенку из n ступенек, каждая ступенька состоит из чисел от 1 до j без пробелов. Для вывода используем end (print(j, end='')), которая позволяет выводить числа и другие символы в строку.

Задача 32

Дано целое число n. Найти сумму 1**1 + 2**2 + ... + n**n.

Задача 33

Даны два целых числа A и B. Выведите все числа от A до B включительно в порядке возрастания, если A<B, или в порядке убывания в противном случае.

Задача 34

Даны числа a, b, c, d. Выведите в порядке возрастания все целые числа от 00 до 10001 включительно, которые являются корнями уравнения a∗x3+b∗x2+c∗x+d=0.

Задача 35

Введите числа K и N. Выведите все числа от K до N включительно в порядке возрастания, если их произведение меньше 100, или в порядке убывания в противном случае

Дополнительные задания:

1. Зайдите в Znanium под свой учетной записью.
2. Найдите учебник С.Р. Гуриков Основы алгоритмизации и программирования на Python (https://znanium.com/catalog/document?id=390096)
3. Решите самостоятельные задания со страницы 85

Практическая работа №14. Функции и процедуры в Python

Цель: Познакомиться с языком программирования Python. Закрепитить навыки работы с числовыми данными

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Функция - это именованный блок кода, к которому можно обратиться из любого места программы. У функции есть имя и список входных параметров, а также возвращаемое значение.

Функция позволяет использовать в программе один и тот же фрагмент кода несколько раз.

Объявление функции в Python выглядит так:

```
def function_name(argument1, argument2, ...):
    # код функции
    # def - DEclare Function - "объявить функцию"
```

function_name - имя функции

(argument1, argument2, ...) - список аргументов, поступающих на вход функции при ее вызове

тело функции - это весь код, который идет после двоеточия

Объявление функции

В большинстве языков программирования, в том числе и в Python реализовано достаточное количество стандартных функций, например функции abs(x) и sqrt(x).

Кроме встроенных и библиотечных функций есть возможность писать собственные функции, предварительно описав их. Это даёт возможность выносить фрагменты кода, которые часто повторяются, в функции, что сокращает объём программы и делает её более понятной. Сначала рассмотрим пример использования стандартных функций:

```
x = int(input())
print(abs(x))
```

В этом примере используются четыре функции: input, int, abs и print.

Функции могут принимать какие-то значения в качестве параметров, но могут и не принимать никаких параметров. Например, функция input не получает входных значений.

Также функции могут возвращать какие-то значения, но могут ничего не возвращать. Например, функция print ничего не возвращает.

Описание функции в Python располагается в любом месте программы, но до первого её использования. После описания функции её можно использовать в выражениях наряду со стандартными функциями.

При описании функции указывается служебное слово **def**, затем после пробела имя функции, после чего в круглых скобках через запятую перечисляются её параметры. Если функция не принимает входных параметров, то в круглых скобках ничего не пишут. После круглых скобок ставится двоеточие, и затем на следующей строке с отступом задаётся тело функции так, как это делается при написании циклов.

Если функция должна вернуть какое-то значение, то для этого используется служебное слово **return**. Инструкция **return** завершает работу функции и возвращает значение соответствующей переменной (выражения). Инструкция **return** может встречаться в произвольном месте функции, её исполнение завершает работу функции и возвращает указанное значение в место вызова. Если функция не возвращает значения, то инструкция **return** используется без возвращаемого значения или инструкция **return** может отсутствовать.

Например, функция нахождения максимума из двух чисел будет выглядеть следующим образом:

```
def max(a, b):
    if a > b:
        return a
    else:
        return b
```

После того как мы написали такую функцию, мы можем ее использовать, например, вот так:

```
print(max(3, 8))
```

В результате эта строчка напечатает число 8.

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и

назовите pr16_z1.py, обязательно поставьте расширение .py. Далее каждое задание делаем в новом файле и продолжаем нумерацию pr16_z2.py, pr16_z3.py и т.д.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Объявление функции в Python

```
def function_name(argument1, argument2, ...):
    # код функции
    # def - DEclare Function - "объявить функцию"
# function_name - имя функции
# (argument1, argument2, ...) - список аргументов, поступающих на вход функции при ее вызове
# тело функции - это весь код, который идет после двоеточия
```

Объявим функцию по примеру

```
def hello(name):
    print('Hello, ' + name)
```

Вызовем функцию

```
hello('Max')
hello('Ivan')
hello('Alex')
hello('Kate')
# Вывод
>> Hello, Max
>> Hello, Ivan
>> Hello, Alex
>> Hello, Kate
```

Оператор return возвращает значение из функции.

Представьте себе обычный калькулятор. Вы вводите первое число, операцию и второе число. Калькулятор возвращает нам результат операции над этими числами. Всё просто, не так ли? Функции точно так же умеют возвращать значение с помощью специального оператора return.

```
# Объявление функции
def sum2(a, b):
    return a + b
# Вызовы функции
s1 = sum2(10, 2)
s2 = sum2(108, 100)
s3 = sum2(3, 1)
print(f's1 = {s1}')
print(f's2 = {s2}')
print(f's3 = {s3}')
>> s1 = 12
>> s2 = 208
>> s3 = 4
```

```
# Функция умножения двух чисел
def mult2(a, b):
    return a * b
# Вызовем нашу функцию
m1 = mult2(10, 2)
m2 = mult2(108, 100)
m3 = mult2(3, 1)
print(f'm1 = {m1}')
print(f'm2 = {m2}')
print(f'm3 = {m3}')
>> m1 = 20
>> m2 = 10800
>> m3 = 3
```

Для параметров функции можно указывать значения по умолчанию. Это дает возможность вызывать функцию с меньшим числом параметров.

```
# Аргумент name по умолчанию равен 'world'
def hello(name='world'):
    print('Hello, ' + name)
hello()
hello('Ivan')
>> Hello, world
>> Hello, Ivan
```

Мы можем использовать уже написанные функции для реализации других функций. Например, мы можем реализовать функцию max3, находящую максимум трёх чисел, следующим образом:

```
def max3(a, b, c):
    return max(max(a, b), c)
```

Функция max3 дважды вызывает функцию max для двух чисел: сначала чтобы найти максимум из a и b, потом чтобы найти максимум из этой величины и c. В программе вызов функции max3 может выглядеть так:

```
x = -2
y = 5
print(max3(2, -3 * x, y))
```

или так:

```
a = int(input())
b = int(input())
m = max3(172, a, b - c)
```

Для примера напишем функцию min_divisor, которая для заданного натурального числа n>1n>1 находит его минимальный делитель, больший единицы:

```
def min_divisor(n):
    for d in range(2, n + 1):
        if n % d == 0:
            return d
```

Эту же функцию можно реализовать с использованием цикла while следующим образом:

```
def min_divisor(n):
    d = 2
    while n % d != 0:
        d += 1
    return d
```

На основании этой функции можно написать функцию is_prime, которая проверяет число на простоту:

```
def is_prime(n):
    return n == min_divisor(n)
```

Если число простое, то его минимальный делитель равен самому числу, и в таком случае функция is_prime вернёт True. Если же число составное, то функция вернёт False.

Функцию is_prime можно использовать, например, вот таким образом:

```
n = int(input())
if is_prime(n):
    print("Простое")
else:
    print("Составное")
```

Решение задач

1. Квадрат

```
# Написать функцию square(), вычисляющую квадрат числа.
def square(number):
    return number * number # Возвращаем результат работы функции обратно в программу
a = square(2)
print(a)
>> 4
```

2. Периметр

```
# Напишите функцию perimetr, вычисляющую периметр прямоугольника со сторонами a и b.
def perimetr(a, b):
    return 2 * (a + b)
p = perimetr(4, 3)
print(p)
>> 14
```

3. Четное число

```
# Напишите функцию isEven, возвращающую True, если число четное, и False, если - нечетное.
def isEven(x):
    return x % 2 == 0
print(isEven(10))
print(isEven(11))
```

```
>> True
>> False
```

4. Сумма списка

```
# Напишите функцию amountList, которая возвращает сумму всех элементов списка.
def amountList(lst):
    amount = 0
    for x in lst:
        amount += x
    return amount
print(amountList([1, 2, 3]))
mylist = [1, 2, 4, 8, 16]
s = amountList(mylist)
print(f'Сумма списка {mylist} равна {s}')
>> 6
>> Сумма списка [1, 2, 4, 8, 16] равна 31
```

5. Фибоначчи

```
# Напишите функцию fib, которая возвращает n-ное число Фибоначчи.
# Последовательность Фибоначчи выглядит так: 1 1 2 3 5 8 13 21 34
def fib(n):
    a, b = 0, 1
    if n == 0: return 0
    for i in range(1, n):
        a, b = b, a + b
    return b
print(fib(2))
print(fib(3))
print(fib(4))
print(fib(5))
print(fib(10))
>> 1
>> 2
>> 3
>> 5
>> 55
```

6. Факториал

```
# Напишите функцию fact, вычисляющую значение факториала числа N.
# Факториал числа - это произведение всех чисел от 1 до N.
# Например, факториал числа 5 равен 120 (5! = 120).
def fact(n):
    result = 1
    while n > 1:
```

```
        result *= n
        n -= 1
    return result
print(fact(2))
print(fact(3))
print(fact(4))
print(fact(5))
>> 2
>> 6
>> 24
>> 120
```

7. Площадь круга

Напишите функцию, которая получает в качестве аргумента радиус круга и находит его площадь.

```
# Не забудьте написать функцию circle...
print(circle(4))
print(circle(1))
# Вывод:
>> 50.24
>> 3.14
```

8. На три

Напишите функцию, которая возвращает True, если число делится на 3, и False, если - нет.

```
# Не забудьте написать функцию three...
print(three(4))
print(three(3))
# Вывод:
>> False
>> True
```

9. Максимум в списке

Напишите функцию, которая возвращает максимальный элемент из переданного в нее списка.

```
# Напишите функцию maxList...
mylist = [1, 3, 2]
print(maxList(mylist))
# Вывод:
>> 3
```

10. Сколько четных

Напишите функцию, которая возвращает количество четных элементов в списке.

```
# Напишите функцию evenCounter...
mylist = [1, 10, 2, 4, 6]
evens = evenCounter(mylist)
```

```
print(even)
# Вывод:
>> 4
```

11. Уникальные

Напишите функцию, которая возвращает список с уникальными (неповторяющихся) элементам.

```
# Напишите функцию unique...
mylist = [1, 1, 2, 1, 3, 2, 3]
print(unique(mylist))
# Вывод:
>> [1, 2, 3]
```

12. Создайте функцию с именем my_function, которая выводит сообщение "Привет от функции"

Выполнить функцию с именем my_function.

13. Внутри функции с двумя параметрами выведите первый параметр.

```
def my_function(fname, lname):
  print(…..)
```

14. Пусть функция возвращает x параметр + 5.

```
def my_function(x):
    … … … …
```

15. Если вы не знаете количество аргументов, которые будут переданы в вашу функцию, есть префикс, который вы можете добавить в определение функции, какой префикс?

```
def my_function(…..kids):
  print("Самый младший ребенок - это " + kids[2])
```

16. Если вы не знаете количество аргументов *ключевого слова*, которые будут переданы в вашу функцию, есть префикс, который вы можете добавить в определение функции, какой префикс?

```
def my_function(…kid):
  print("Его фамилия " + kid["lname"])
```

17. Напишите функцию sum_range(start, end), которая суммирует все целые числа от значения «start» до величины «end» включительно.

Если пользователь задаст первое число большее чем второе, просто поменяйте их местами.

При решении удобно воспользоваться встроенными функциями **range()** и **sum()**.

```
def sum_range(start, end):
    if start > end:
        end, start = start, end
    return sum(range(start, end + 1))
# Тесты
print(sum_range(2, 12))
print(sum_range(-4, 4))
print(sum_range(3, 2))
```

Результат выполнения

77

0

5

18. Чтобы проверить понимание параметров и область их видимости, Николай создал 3 функции (представлены ниже).

Попытайтесь предугадать, как поведет себя каждая из них при запуске (возникнут ли ошибки, что возвратится).

```
def func1():
    param = 4

    def inner():
        param += 1
    return param
def func2():
    param = 4
    def inner(var):
        var += 1
    inner(param)
    return param
def func3():
    param = 4
    def inner(var):
        var += 1
        return var
    param = inner(param)
    return param
```

Как ни странно, никаких ошибок при вызове функций мы не увидели.

```
# Тесты
print(func1())
print(func2())
print(func3())
```

Результат выполнения

4

4

5

Тем не менее, попробуем разобраться в каждом представленном случае.

1) Первая функция

Во внутренней функции мы пытаемся воспользоваться внешней переменной. Но она не доступна. Ошибки не возникло по единственной причине: мы эту функцию не вызвали.

2) Вторая функция

Внутренняя функция увеличила значение переменной на 1, но сама она ничего не возвращает (кроме **None**), поэтому значение param не поменялось.

3) Третья функция

В данном случае вернулось 5, так как мы во внутренней функции увеличили внешнюю переменную и присвоили результат в **func3**.

19. Создайте функцию three_args(), которая принимает 1, 2 или 3 строго ключевых параметра. В результате ее работы на печать в консоль выводятся значения переданных переменных, но только если они не равны **None**. Получим, например, следующее сообщение: «Переданы аргументы: var1 = 2, var3 = 10». Для начала необходимо задать ограничение на тип переменных (в нашем случае предполагаются строго ключевые аргументы). Также, по условию сказано, что переменных может быть от одной до трех. Поэтому двум последним параметрам присваиваем дефолтное значение **None**. Проще всего решить задачу использовав функцию **locals()**, которая в виде словаря представит все внутренние аргументы функции.

```
def three_args(*, var1, var2=None, var3=None):
    arguments = ', '.join([f'{arg[0]} = {str(arg[1])}' for arg in locals().items() if arg[1] is not None])
    print(f'Переданы аргументы: {arguments}')
# Тесты
three_args(var1=21)
three_args(var1='Python', var3=3)
three_args(var1='Python', var2=3, var3=9)
```

Результат выполнения

```
Переданы аргументы: var1 = 21
Переданы аргументы: var1 = Python, var3 = 3
Переданы аргументы: var1 = Python, var2 = 3, var3 = 9
```

20. Антон попал в коллизию: его функция time_now() работает очень странно. Казалось бы, задача простая: показать текущее время с сообщением. Тем не менее, время не меняется. Код предоставлен ниже с примерами. Постарайтесь решить проблему незадачливого программиста.

```
from datetime import datetime
from time import sleep
def time_now(msg, *, dt=datetime.now()):
    print(msg, dt)
 # Тесты
time_now('Сейчас такое время: ')
sleep(1)
time_now('Прошла секунда: ')
sleep(1)
time_now('Ничего не понимаю... ')
```

Результат выполнения

```
Сейчас такое время:  2021-03-14 15:48:55.117455
Прошла секунда:  2021-03-14 15:48:55.117455
```

Ничего не понимаю... 2021-03-14 15:48:55.117455

Когда мы запускаем скрипт, он вычисляет некие глобальные переменные. Параметр по умолчанию **dt** был вычислен в момент создания функции. Так как мы его не меняли, то и заново пересчитывать значение Python не стал. Чтобы исправить ситуацию, потребуется вызов значений текущего времени при выполнении функции, а не при ее создании. Если нам все же нужен параметр **dt** по каким-то причинам (вдруг, мы захотим передать другую дату, а не текущую), то логичнее присвоить ему значение **None** изначально.

```
from datetime import datetime
from time import sleep
 def time_now(msg, *, dt=None):
    dt = dt or datetime.now()
    print(msg, dt)
# Тесты
time_now('Сейчас такое время: ')
sleep(1)
time_now('Прошла секунда: ')
sleep(1)
time_now('Задам-ка другую дату: ', dt='2020-01-11 11:00:10')
```

Результат выполнения

Сейчас такое время: 2021-03-14 16:01:53.331533

Прошла секунда: 2021-03-14 16:01:54.331609

Задам-ка другую дату: 2020-01-11 11:00:10

21. Чтобы лучше разобраться в типах параметров функций Инна создала inspect_**function**(), которая в качестве аргумента принимает другую функцию (главное, не встроенную, built-in). В результате работы она выводит следующие данные: название анализируемой функции, наименование всех принимаемых ею параметров и их типы (позиционные, ключевые и т.п.). Попробуйте повторить результат девушки. В данном случае на подмогу приходит модуль **inspect**. С его помощью можно реализовать задуманный функционал.

```
import inspect
import math
def inspect_function(some_func):
        print(f'Анализируем функцию {some_func.__name__}')
        for param in inspect.signature(some_func).parameters.values():
        print(param.name, param.kind, sep=': ')
# Некая функция для анализа
def my_func(a, b, /, c, d, *args, e, f, **kwargs):
        pass
# Тесты
inspect_function(my_func)
print('-' * 25)
inspect_function(math.sqrt)
```

Результат выполнения

```
Анализируем функцию my_func
a: POSITIONAL_ONLY
b: POSITIONAL_ONLY
c: POSITIONAL_OR_KEYWORD
d: POSITIONAL_OR_KEYWORD
args: VAR_POSITIONAL
e: KEYWORD_ONLY
f: KEYWORD_ONLY
kwargs: VAR_KEYWORD
------------------------
Анализируем функцию sqrt
x: POSITIONAL_ONLY
```

Практическая работа №15. Использование функций и процедур в Python

Цель: Познакомиться с языком программирования Python. Закрепитить навыки работы с числовыми данными

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Функция - это именованный блок кода, к которому можно обратиться из любого места программы. У функции есть имя и список входных параметров, а также возвращаемое значение.

Функция позволяет использовать в программе один и тот же фрагмент кода несколько раз.

В языке программирования Python функции определяются с помощью оператора **def**. Рассмотрим код:

```
def count_food():
    a = int(input())
```

```
b = int(input())
print("Всего", a+b, "шт.")
```

Это пример определения функции. Как и другие сложные инструкции вроде условного оператора и циклов функция состоит из заголовка и тела. Заголовок оканчивается двоеточием и переходом на новую строку. Тело имеет отступ.

Ключевое слово **def** сообщает интерпретатору, что перед ним определение функции. За **def** следует имя функции. Оно может быть любым, также как и всякий идентификатор, например, переменная. В программировании весьма желательно давать всему осмысленные имена. Так в данном случае функция названа "посчитать_еду" в переводе на русский.

После имени функции ставятся скобки. В приведенном примере они пустые. Это значит, что функция не принимает никакие данные из вызывающей ее программы. Однако она могла бы их принимать, и тогда в скобках были бы указаны так называемые параметры.

После двоеточия следует тело, содержащее инструкции, которые выполняются при вызове функции. Следует различать определение функции и ее вызов. В программном коде они не рядом и не вместе. Можно определить функцию, но ни разу ее не вызвать. Нельзя вызвать функцию, которая не была определена. Определив функцию, но ни разу не вызвав ее, вы никогда не выполните ее тела.

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr17_z1.py, обязательно поставьте расширение .py. Далее каждое задание делаем в новом файле и продолжаем нумерацию pr17_z2.py, pr17_z3.py и т.д.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Для объявления функции используется следующий синтаксис.

```
def ИмяФункции (Параметре ы)):
        Опера горы функции
        return возвращаемое значение
```

где:

- def (от англ, define - определять, устанавливать);
- ИмяФункнии - уникальное имя создаваемой функции. На имя функции распространяются общие правила написания идентификаторов,
- Параметры (аргументы) список необязательных аргументов, разделенных запятыми и используемых в данной функции;
- Операторы функции - блок операторов, который выполняет работу функции;
- return необязательный оператор, с помощью которого можно указать место, где в блоке кода функции требуется возвратить значение в вызывающую программу, и каково это возвращаемое значение. После выполнения данного оператора происходит выход из функции, и управление передается в то место программы, откуда эта функция была вызвана.

Вызов пользовательской функции происходит следующим образом: ИмяПеременной=ИмяФункции (Параметр(ы))

Еще раз напомним, чго при создании собственной функции следует учитывать. что у нее может не быть параметров, и она может не возвращать значение с помощью оператора return. В качестве примера напишем функцию, которая выводит сообщение об ошибке.

```
def func(): #Параметры функции отсутствуют
        printf("Ошибка")
func() # вызов функции
```

При вызове функции оператором func() на экран будет выведено слово «Ошибка».

Определение функции. Оператор def

В языке программирования Python функции определяются с помощью оператора **def**. Рассмотрим код:

```
def count_food():
   a = int(input())
   b = int(input())
   print("Всего", a+b, "шт.")
```

Это пример определения функции. Как и другие сложные инструкции вроде условного оператора и циклов функция состоит из заголовка и тела. Заголовок оканчивается двоеточием и переходом на новую строку. Тело имеет отступ.

Ключевое слово **def** сообщает интерпретатору, что перед ним определение функции. За **def** следует имя функции. Оно может быть любым, также как и всякий идентификатор, например, переменная. В программировании весьма желательно давать всему осмысленные имена. Так в данном случае функция названа "посчитать_еду" в переводе на русский.

После имени функции ставятся скобки. В приведенном примере они пустые. Это значит, что функция не принимает никакие данные из вызывающей ее программы. Однако она могла бы их принимать, и тогда в скобках были бы указаны так называемые параметры.

После двоеточия следует тело, содержащее инструкции, которые выполняются при вызове функции. Следует различать определение функции и ее вызов. В программном коде они не рядом и не вместе. Можно определить функцию, но ни разу ее не вызвать. Нельзя вызвать функцию, которая не была определена. Определив функцию, но ни разу не вызвав ее, вы никогда не выполните ее тела.

Вызов функции

Рассмотрим полную версию программы с функцией:

```
def count_food():
   a = int(input())
   b = int(input())
   print("Всего", a+b, "шт.")

print("Сколько бананов и ананасов для обезьян?")
count_food()
```

```
print("Сколько жуков и червей для ежей?")
count_food()

print("Сколько рыб и моллюсков для выдр?")
count_food()
```

После вывода на экран каждого информационного сообщения осуществляется вызов функции, который выглядит просто как упоминание ее имени со скобками. Поскольку в функцию мы ничего не передаем скобки опять же пустые. В приведенном коде функция вызывается три раза.

Когда функция вызывается, поток выполнения программы переходит к ее определению и начинает исполнять ее тело. После того, как тело функции исполнено, поток выполнения возвращается в основной код в то место, где функция вызывалась. Далее исполняется следующее за вызовом выражение.

В языке Python определение функции должно предшествовать ее вызовам. Это связано с тем, что интерпретатор читает код строка за строкой и о том, что находится ниже по течению, ему еще неизвестно. Поэтому если вызов функции предшествует ее определению, то возникает ошибка (выбрасывается исключение **NameError**):

```
print("Сколько бананов и ананасов для обезьян?")
count_food()
print("Сколько жуков и червей для ежей?")
count_food()
print("Сколько рыб и моллюсков для выдр?")
count_food()
def count_food():
    a = int(input())
    b = int(input())
    print("Всего", a+b, "шт.")
```

Результат:

```
Сколько бананов и ананасов для обезьян?
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    count_food()
NameError: name 'count_food' is not defined
```

Для многих компилируемых языков это не обязательное условие. Там можно определять и вызывать функцию в произвольных местах программы. Однако для удобочитаемости кода программисты даже в этом случае предпочитают соблюдать определенные правила.

Функции придают программе структуру

Польза функций не только в возможности многократного вызова одного и того же кода из разных мест программы. Не менее важно, что благодаря им программа обретает истинную структуру. Функции как бы разделяют ее на обособленные части, каждая из которых выполняет свою конкретную задачу.

Пусть надо написать программу, вычисляющую площади разных фигур. Пользователь указывает, площадь какой фигуры он хочет вычислить. После этого вводит исходные данные. Например, длину и ширину в случае прямоугольника. Чтобы разделить поток выполнения на несколько ветвей, следует использовать оператор **if-elif-else**:

```
figure = input("1-прямоугольник,
2-треугольник, 3-круг: ")
if figure == '1':
  a = float(input("Ширина: "))
  b = float(input("Высота: "))
  print("Площадь: %.2f" % (a*b))
elif figure == '2':
  a = float(input("Основание: "))
  h = float(input("Высота: "))
  print("Площадь: %.2f" % (0.5 * a * h))
elif figure == '3':
  r = float(input("Радиус: "))
  print("Площадь: %.2f" % (3.14 * r**2))
else:
  print("Ошибка ввода")
```

Здесь нет никаких функций, и все прекрасно. Но напишем вариант с функциями:

```
def rectangle():
    a = float(input("Ширина: "))
    b = float(input("Высота: "))
    print("Площадь: %.2f" % (a*b))
def triangle():
    a = float(input("Основание: "))
    h = float(input("Высота: "))
    print("Площадь: %.2f" % (0.5 * a * h))
def circle():
    r = float(input("Радиус: "))
    print("Площадь: %.2f" % (3.14 * r**2))
figure = input("1-прямоугольник,
2-треугольник, 3-круг: ")
if figure == '1':
  rectangle()
elif figure == '2':
  triangle()
elif figure == '3':
  circle()
else:
  print("Ошибка ввода")
```

Он кажется сложнее, а каждая из трех функций вызывается всего один раз. Однако из общей логики программы как бы убраны и обособлены инструкции для нахождения

площадей. Программа теперь состоит из отдельных "кирпичиков Лего". В основной ветке мы можем комбинировать их как угодно. Она играет роль управляющего механизма.

Если нам когда-нибудь захочется вычислять площадь треугольника по формуле Герона, а не через высоту, то не придется искать код во всей программе (представьте, что она состоит из тысяч строк кода как реальные программы). Мы пойдем к месту определения функций и изменим тело одной из них.

Если понадобиться использовать эти функции в какой-нибудь другой программе, то мы сможем импортировать их туда, сославшись на данный файл с кодом (как это делается в Python, будет рассмотрено позже).

Задача 1. Разработайте функцию, которая определяет наибольшее число из двух заданных. Ниже приведен код программы, отвечающий за решение задачи.

```
def maximum(a.b):
        if a>=b:
                max=a
        else:
                max=b
        return max
a=int(input("\nВведите первое число "))
b=int(input("\nВведите второе число "))
rez= maximum(a.b) #Вызов функции
print("\nМаксимальное из двух чисел =", rez)
```

Комментарий. Переменные a и b, указанные в затоловкс функции, называются позиционными параметрами. Такие параметры принимают значения только в том порядке, в котором они переданы. Прокомментируем механизм действия функции, изложенный в упрошенном виде, без учета динамической типизации свойственной Python.

После запуска программы на выполнение начинает свою работу основная часть программы, которая запрашивает у пользователя значения первого и второго чисел (a и b). Далее вызывается функция maximum(a,b) с указанием параметров a и b, которые передаются в переменные a и b,. указанные в заголовке функции: def maxinium(a,b). Далее над ними выполняются действия, записанные в теле функции, а именно, нахождение максимального значения из двух чисел и управление вновь перелается в основную часть программы на оператор rez=maximum(a,b). Результат нахождения максимального значения за счет выполнения оператора присваивания оказывается в ячейке rez и выводится на экран оператором print.

Добавим к вышесказанному, что между параметрами, объявленными в заголовке функции, и параметрами, указанными при вызове функции, должно быть соответствие, а именно, количество параметров должно быть одинаковым.

Задача 2. Разработайте функцию для вычисления суммы квадратов двух чи¬сел и вывода се на экран.

Комментарий. Рассмотрим на примере этой задачи несколько особенностей использования функций в языке Python. В коде, приведенном ниже используются значения, переданные по умолчанию. Таким образом, вызов функции square не содержит аргументов.

```
def square(x=5.y=5):
```

```
        c=x*x+y*y
        return c
kvadr=square()
print("\nСумма квадратов двух чисел=”, kvadr)
```

Разберем другой вариант решения этой же задачи. В примере используются именованные аргументы, которые позволяют передавать значения в любом порядке. Для того чтобы разобраться, как это происходит, мы добавили несколько операторов print в тело функции и будем выводить значения квадратов элементов в зависимости от порядка расположения аргументов.

```
def square(x.y):
        kv1=x*x
        print("\nКвадрат kv1=", kv1)
        kv2=y*y
        print("\nКвалрат kv2=", kv2)
        c=kv1+kv2
        return c
kvadr=square(x=5, y=6)
print("\nСумма квадратов двух чисел =", kvadr)
```

Задача 3. Вычислите значение функции y. если;

$$y = \begin{cases} \min(a1, a2, a3), \text{если } -1 < x < 1 \\ \max\{b1, b2, \min\{c1, c2\}\}, \text{если } x \geq 1 \\ 1, \text{если } x \leq 1 \end{cases}$$

Дополнительно выведите на жран номер ветви, по которой производятся вычисления в зависимости от введенных пользователем значении.

Комментарии. Разработанная функция znach содержит параметры, указанные в заголовке, а в переменной n будет находиться целое число, соответствующее номеру ветви. Обратим внимание на оператор return. Для того чтобы функция вернула два значения: значение y и значение n, мы указываем их через запятую в операторе return y, n.

```
def znach(a I ,a2,a3,b I ,Ь2.c I ,c2,x):
        if x>=l:
                min=cl
                if c2<min:
                        min=c2
                max=b1
                if b2>max:
                        max=b2
                if min>max:
                        max=min
                y=max
                n=2 #Фиксация номера ветви
        elif (-l<x) and (x< 1):
                min=al
                if a2<min:
```

```
            min=a2
        if a3<min:
            min=a3
        y=min
        n=1 #Фиксация номера ветви
    else:
        y= 1
        n=3 #Фиксация номера ветви
    return y,n
```

В основной части программы реализуем ввод данных. Теперь посмотрим, как осуществляется вызов функции. В отличие от предыдущих примеров. где функция возвращала одно значение, мы сейчас указали два значения rez и п. расположив их через запятую: rez,n=znach(al,a2,a3,b1,b2,cl,c2,x). Далее делаем вывод найденного значения функции и номера вегви про1раммы.

```
a1=int(input("Введите значение al "))
a2=int(input("Введите значение a2 "))
a3=int(input("Введите значение a3 "))
b1=int(input("Введите значение bl "))
b2=int(input("Введите значение b2 "))
b3=int(input("Введите значение b3 "))
c1=int(input("Введите значение cl "))
c2=int(input("Введите значение c2 "))
c3=int(input("Введите значение c3 "))
x = int(input("Введите значение x "))
rcz,n=znach(a1,a2,a3,bl,b2,cl,c2,x) #Вызов функции. Функция возвращает несколько значений
print("\nРезультат = ", rez)
print("\n Номер ветви = ", n)
```

Задача 4. Создайте программу для вычисления периметра и площади треугольника по заданным координатам трех его вершин.

Задано: xl, yl; x2, y2; x3, y3 координаты вершин. Требуется определить: P - периметр треугольника и S - площадь треугольника. Ограничения на значения исходных данных и их соотношения: A>0, B>0, C>0, A+B>C, A+C>B. B+C>A одновременно.

Комментарий. Для решения задачи существуют известные формулы:

$$P = a + b + c$$

$$s = \sqrt{P_P * (P_P - a) * (P_P - b) * (P_P - c)}$$

$$a = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

$$b = \sqrt{(X3 - X2)^2 + (Y3 - Y2)^2}$$

$$c = \sqrt{(X3 - X1)^2 + (Y3 - Y1)^2}$$

где Pp=p/2— полупериметр: a, b, c - стороны треугольника.

В соответствии с требованиями задания разобьем решение задачи на несколько отдельных задач и создадим соответствующие пользовательские функции:

– функцию def dl_otr (), вычисляющую длину отрезка по координатам двух точек:
– функцию def ps(), вычисляющую периметр и площадь треугольника.

```
from math import *
def dl_otr(x1 ,y1 ,x2,y2):
        f=sqrt(((x2-xl)*(x2-xl)) + ((y2-yl)*(y2-yl)))
        return f
def ps(xl,yl,x2,y2,x3,y3):
        a=dl_otr (xl. y 1. x2, y2) #Вызов функции вычисления длины отрезка
        b=dl_otr (x2, y2, x3, y3) #Вызов функции вычисления длины отрезка относительно других значений
        c=dl_otr (x3, y3, x1, y1)
        p=(a + b + c)
        Pp = p/2
        S=sqrt(Pp*(Pp-a)*(Pp-b)*(Pp-c))
        return s,p
x1 = int(input("Введите значение x1  "))
y1 = int(input("Введите значение y1  "))
x2 = int(input("Введите значение x2  "))
y2 = int(input("Введите значение y2  "))
x3 = int(input("Введите значение x3  "))
y3 = int(input("Введите значение y3  "))
s,p=ps(xl, yl, x2, y2, x3, y3) #Вызовфункции ps
print("\n Периметр треугольника =", p)
print("\n Площадь треугольника = ", s)
```

Решение задач

1. В программировании можно из одной функции вызывать другую. Для иллюстрации этой возможности напишите программу по следующему описанию.
2. Основная ветка программы, не считая заголовков функций, состоит из одной строки кода. Это вызов функции test(). В ней запрашивается на ввод целое число. Если оно положительное, то вызывается функция positive(), тело которой содержит команду вывода на экран слова "Положительное". Если число отрицательное, то вызывается функция negative(), ее тело содержит выражение вывода на экран слова "Отрицательное".

   Понятно, что вызов test() должен следовать после определения функций. Однако имеет ли значение порядок определения самих функций? То есть должны ли определения positive() и negative() предшествовать test() или могут следовать после него? Проверьте вашу гипотезу, поменяв объявления функций местами. Попробуйте объяснить результат.
3. Вводится последовательность вещественных чисел. Известно, что последний элемент последовательности равен 5. Разработайте функцию, подсчитывающую количество положительных чисел и минимальное из них.

4. Выполните табулирование (построение таблицы) функции y=sin(x) если известно начальное значение интервала, на котором изменяется функция, конечное значение интервала и шаг ее изменения. Требования к программе:
   - Вычисление функции y=sin(x) оформите в виде пользовательской функции.
   - Задачу табулирования функции также выполните в виде пользовательской функции. Вызовите из нее ранее написанную функцию для вычисления sin(x).

   Комментарий. В алгоритме решения задачи табулирования функции используется роулярная циклическая структура, которая в программе реализована оператором for. Предварительно в программе происходит вычисление числа повторений цикла по формуле:

$$n = \left[\frac{b-a}{h}\right] + 1$$

   где:

   a – начальное значение интервала;

   b – конечное значение интервала;

   h – шаг.

Практическая работа №16. Работа со списками. Операции над списками в Python

Цель: Познакомиться с языком программирования Python. Изучить особенности работы со списками и операции над ними

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Список в Python – это встроенный тип (класс) данных, представляющий собой одну из разновидностей структур данных. Структуру данных можно представить как сложную единицу, объединяющую в себе группу более простых. Каждая разновидность структур данных имеет свои особенности. Список – это изменяемая последовательность произвольных элементов.

В большинстве других языков программирования есть такой широко используемый тип данных как массив. В Питоне такого встроенного типа нет. Однако списки условно можно считать аналогом массивов за одним исключением. Составляющие массив

элементы должны принадлежать одному типу данных, для списков такого ограничения нет.

Например, массив может содержать только целые числа или только вещественные числа или только строки. Список также может содержать элементы только одного типа, что делает его внешне неотличимым от массива. Но вполне допустимо, чтобы в одном списке содержались как числа, так и строки, а также что-нибудь еще.

Список (**list**) - это упорядоченная изменяемая последовательность элементов.

Особенности:

– может содержать элементы разного типа;
– поддерживает операторы сравнения: при этом сравнивание производится поэлементно (и рекурсивно, при наличии вложенных элементов).

Ход работы

Список - это непрерывная динамическая коллекция элементов. Каждому элементу списка присваивается порядковый номер - его индекс. Первый индекс равен нулю, второй - единице и так далее. Основные операции для работы со списками - это индексирование, срезы, добавление и удаление элементов, а также проверка на наличие элемента в последовательности.

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr18_z1.py, обязательно поставьте расширение .py. Далее каждое задание делаем в новом файле и продолжаем нумерацию pr18_z2.py, pr18_z3.py и т.д.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Создание пустого списка выглядит так:

```
empty_list = []
```

Создадим список, состоящий из нескольких чисел:

```
numbers = [40, 20, 90, 11, 5]
```

Настало время строковых переменных:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
```

Не будем забывать и о дробях:

```
fractions = [3.14, 2.72, 1.41, 1.73, 17.9]
```

Мы можем создать список, состоящий из различных типов данных:

```
values = [3.14, 10, 'Hello world!', False, 'Python is the best']
```

И такое возможно (⊙_⊙)

```
list_of_lists = [[2, 4, 0], [11, 2, 10], [0, 19, 27]]
```

Создадим список простым перечисление элементов:

```
>>> a = [12, 3.85, "black", -4]
>>> a
[12, 3.85, 'black', -4]
```

Итак, у нас имеется список, присвоенный переменной *a*. В Python список определяется квадратными скобками. Он содержит четыре элемента. Если где-то в

программе нам понадобится весь этот список, мы получим доступ к нему, указав всего лишь одну переменную – *a*.

Элементы в списке упорядочены, имеет значение в каком порядке они расположены. Каждый элемент имеет свой индекс, или номер. Индексация начинается с нуля. В данном случае число 12 имеет индекс 0, строка "black" – индекс 2. Чтобы извлечь конкретный элемент, надо после имени переменной указать в квадратных скобках его индекс:

```
>>> a[0]
12
>>> a[3]
-4
```

В Python существует также индексация с конца. Она начинается с -1:

```
>>> a[-1]
-4
>>> a[-2]
'black'
>>> a[-3], a[-4]
(3.85, 12)
```

Часто требуется извлечь не один элемент, а так называемый срез – часть списка. В этом случае указывается индекс первого элемента среза и индекс следующего за последним элементом среза:

```
>>> a[0:2]
[12, 3.85]
```

В данном случае извлекаются первые два элемента с индексами 0 и 1. Элемент с индексом 2 в срез уже не входит. В таком случае возникает вопрос, как извлечь срез, включающий в себя последний элемент? Если какой-либо индекс не указан, то считается, что имеется в виду начало или конец:

```
>>> a[:3]
[12, 3.85, 'black']
>>> a[2:]
['black', -4]
>>> a[:]
[12, 3.85, 'black', -4]
```

Списки – изменяемые объекты. Это значит, что в них можно добавлять элементы, удалять их, изменять существующие. Проще всего изменить значение элемента. Для этого надо обратиться к нему по индексу и перезаписать значение в заданной позиции:

```
>>> a[1] = 4
>>> a
[12, 4, 'black', -4]
```

Добавлять и удалять лучше с помощью специальных встроенных методов списка:

```
>>> a.append('wood')
>>> a
```

```
[12, 4, 'black', -4, 'wood']
>>> a.insert(1, 'circle')
>>> a
[12, 'circle', 4, 'black', -4, 'wood']
>>> a.remove(4)
>>> a
[12, 'circle', 'black', -4, 'wood']
>>> a.pop()
'wood'
>>> a
[12, 'circle', 'black', -4]
>>> a.pop(2)
'black'
>>> a
[12, 'circle', -4]
```

Перечень всех методов списка можно узнать с помощью встроенной в Python функции dir(), передав в качестве аргумента переменную, связанную со списком, или название класса (в данном случае – **list**). В полученном из dir() списке надо смотреть имена без двойных подчеркиваний.

Для получения информации о конкретном методе следует воспользоваться встроенной функцией help(), передав ей в качестве аргумента имя метода, связанное с объектом или классом. Например, help(a.pop) или help(list.index). Выход из справки – q.

Можно изменять списки не используя методы, а с помощью взятия и объединения срезов:

```
>>> b = [1, 2, 3, 4, 5, 6]
>>> b = b[:2] + b[3:]
>>> b
[1, 2, 4, 5, 6]
```

Здесь берется срез из первых двух элементов и срез, начиная с четвертого элемента (индекс 3) и до конца. После чего срезы объединяются с помощью оператора "сложения".

Можно изменить не один элемент, а целый срез:

```
>>> mylist = ['ab','ra','ka','da','bra']
>>> mylist[0:2] = [10,20]
>>> mylist
[10, 20, 'ka', 'da', 'bra']
```

Пример создания пустого списка с последующим заполнением его в цикле случайными числами:

```
>>> import random
>>> c = []
>>> i = 0
>>> while i < 10:
...     c.append(random.randint(0,100))
...     i += 1
```

...

>>> c

[30, 44, 35, 77, 53, 44, 49, 17, 61, 82]

Индексирование

Что же такое индексирование? Это загадочное слово обозначает операцию обращения к элементу по его порядковому номеру ((･ω･)ア напоминаю, что нумерация начинается с нуля). Проиллюстрируем это на примере:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(fruits[0])
print(fruits[1])
print(fruits[4])
```

>>> Apple

>>> Grape

>>> Orange

Списки в Python являются изменяемым типом данных. Мы можем изменять содержимое каждой из ячеек:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
fruits[0] = 'Watermelon'
fruits[3] = 'Lemon'
print(fruits)
```

>>> ['Watermelon', 'Grape', 'Peach', 'Lemon', 'Orange']

Индексирование работает и в обратную сторону. Как такое возможно? Всё просто, мы обращаемся к элементу списка по отрицательному индексу. Индекс с номером -1 дает нам доступ к последнему элементу, -2 к предпоследнему и так далее.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(fruits[-1])
print(fruits[-2])
print(fruits[-3])
print(fruits[-4])
```

>>> Orange

>>> Banan

>>> Peach

>>> Grape

Создание списка с помощью list()

Переходим к способам создания списка. Самый простой из них был приведен выше. Еще раз для закрепления:

```
smiles = ['(ʘ_ʘ)', '(¯ ͟ʖ ¯)', '( ͡° ͜ʖ ͡°)', '(∩ °-°)⊃ ']
```

А есть еще способы? Да, есть. Один из них — создание списка с помощью функции list() В неё мы можем передать любой итерируемый объект (да-да, тот самый по которому можно запустить цикл (• ◡ •))

Рассмотрим несколько примеров:

```
letters = list('abcdef')
```

```
numbers = list(range(10))
even_numbers = list(range(0, 10, 2))
print(letters)
print(numbers)
print(even_numbers)
>>> ['a', 'b', 'c', 'd', 'e', 'f'
>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [0, 2, 4, 6, 8]
```

Длина списка

С созданием списка вроде разобрались. Следующий вопрос: как узнать длину списка? Можно, конечно, просто посчитать количество элементов... (⊙_⊙) Но есть способ получше! Функция len() возвращает длину любой итерируемой переменной, переменной, по которой можно запустить цикл. Рассмотрим пример:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(len(fruits))
>>> 5
numbers = [40, 20, 90]
print(len(numbers))
>>> 3
"...любой итерируемой", а это значит:
string = 'Hello world'
print(len(string))
# 11
>>> 11
print(len(range(10))
>>> 10
```

Срезы

В начале статьи что-то говорилось о "срезах". Давайте разберем подробнее, что это такое. Срезом называется некоторая подпоследовательность. Принцип действия срезов очень прост: мы "отрезаем" кусок от исходной последовательности элемента, не меняя её при этом. Я сказал "последовательность", а не "список", потому что срезы работают и с другими итерируемыми типами данных, например, со строками.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
part_of_fruits = fruits[0:3]
print(part_of_fruits)
>>> ['Apple', 'Grape', 'Peach']
```

Детально рассмотрим синтаксис срезов:

итерируемая_переменная[начальный_индекс:конечный_индекс - 1:длина_шага]

Обращаю ваше внимание, что мы делаем срез от начального индекса до конечного индекса - 1. То есть i = начальный_индекс и i < конечный индекс

Больше примеров!

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(fruits[0:1])
```

```
# Если начальный индекс равен 0, то его можно опустить
print(fruits[:2])
print(fruits[:3])
print(fruits[:4])
print(fruits[:5])
# Если конечный индекс равен длине списка, то его тоже можно опустить print(fruits[:len(fruits)])
print(fruits[::])
>>> ['Apple']
>>> ['Apple', 'Grape']
>>> ['Apple', 'Grape', 'Peach']
>>> ['Apple', 'Grape', 'Peach', 'Banan']
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
```

Самое время понять, что делает третий параметр среза - длина шага!

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(fruits[::2])
print(fruits[::3])
# Длина шага тоже может быть отрицательной!
print(fruits[::-1])
print(fruits[4:2:-1])
print(fruits[3:1:-1])
>>> ['Apple', 'Peach', 'Orange']
>>> ['Apple', 'Banan']
>>> ['Orange', 'Banan', 'Peach', 'Grape', 'Apple']
>>> ['Orange', 'Banan']
>>> ['Banan', 'Peach']
```

А теперь вспоминаем всё, что мы знаем о циклах. В Python их целых два! Цикл for и цикл while Нас интересует цикл for, с его помощью мы можем перебирать значения и индексы наших последовательностей. Начнем с перебора значений:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
for fruit in fruits:
   print(fruit, end=' ')
>>> Apple Grape Peach Banan Orange
```

Выглядит несложно, правда? В переменную fruit объявленную в цикле по очереди записываются значения всех элементов списка fruits

А что там с перебором индексов?

```
for index in range(len(fruits)):
   print(fruits[index], end=' ')
```

Этот пример гораздо интереснее предыдущего! Что же здесь происходит? Для начала разберемся, что делает функция range(len(fruits))

Мы с вами знаем, что функция len() возвращает длину списка, а range() генерирует диапазон целых чисел от 0 до len()-1.

Сложив 2+2, мы получим, что переменная index принимает значения в диапазоне от 0 до len()-1. Идем дальше, fruits[index] - это обращение по индексу к элементу с индексом index списка fruits. А так как переменная index принимает значения всех индексов списка fruits, то в цикле мы переберем значения всех элементов нашего списка!

Операция in

С помощью in мы можем проверить наличие элемента в списке, строке и любой другой итерируемой переменной.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
if 'Apple' in fruits:
   print('В списке есть элемент Apple')
>>> В списке есть элемент Apple
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
if 'Lemon' in fruits:
   print('В списке есть элемент Lemon')
else:'
   print('В списке НЕТ элемента Lemon')
>>> В списке НЕТ элемента Lemon
```

Приведу более сложный пример:

```
all_fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
my_favorite_fruits = ['Apple', 'Banan', 'Orange']
for item in all_fruits:
   if item in my_favorite_fruits:
      print(item + ' is my favorite fruit')
   else:
      print('I do not like ' + item)
>>> Apple is my favorite fruit
>>> I do not like Grape
>>> I do not like Peach
>>> Banan is my favorite fruit
>>> Orange is my favorite fruit
```

Методы для работы со списками

Начнем с метода append(), который добавляет элемент в конец списка:

```
# Создаем список, состоящий из четных чисел от 0 до 8 включительно
numbers = list(range(0,10,2))
# Добавляем число 200 в конец списка
numbers.append(200)
numbers.append(1)
numbers.append(2)
numbers.append(3)
print(numbers)
>>> [0, 2, 4, 6, 8, 200, 1, 2, 3]
```

Мы можем передавать методу append() абсолютно любые значения:

```
all_types = [10, 3.14, 'Python', ['I', 'am', 'list']]
all_types.append(1024)
all_types.append('Hello world!')
all_types.append([1, 2, 3])
print(all_types)
>>> [10, 3.14, 'Python', ['I', 'am', 'list'], 1024, 'Hello world!', [1, 2, 3]]
```

Метод append() отлично выполняет свою функцию. Но, что делать, если нам нужно добавить элемент в середину списка? Это умеет метод insert(). Он добавляет элемент в список на произвольную позицию. insert() принимает в качестве первого аргумента позицию, на которую нужно вставить элемент, а вторым — сам элемент.

```
# Создадим список чисел от 0 до 9
numbers = list(range(10))
# Добавление элемента 999 на позицию с индексом 0
numbers.insert(0, 999)
print(numbers) # первый print
numbers.insert(2, 1024)
print(numbers) # второй print
numbers.insert(5, 'Засланная строка-шпион')
print(numbers) # третий print
>>> [999, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # первый print
>>> [999, 0, 1024, 1, 2, 3, 4, 5, 6, 7, 8, 9] # второй print
>>> [999, 0, 1024, 1, 2, 'Засланная строка-шпион', 3, 4, 5, 6, 7, 8, 9] # третий print
```

Отлично! Добавлять элементы в список мы научились, осталось понять, как их из него удалять. Метод pop() удаляет элемент из списка по его индексу:

```
numbers = list(range(10))
print(numbers) # 1
# Удаляем первый элемент
numbers.pop(0)
print(numbers) # 2
numbers.pop(0)
print(numbers) # 3
numbers.pop(2)
print(numbers) # 4
# Чтобы удалить последний элемент, вызовем метод pop без аргументов
numbers.pop()
print(numbers) # 5
numbers.pop()
print(numbers) # 6
>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # 1
>>> [1, 2, 3, 4, 5, 6, 7, 8, 9] # 2
>>> [2, 3, 4, 5, 6, 7, 8, 9] # 3
>>> [2, 3, 5, 6, 7, 8, 9] # 4
```

```
>>> [2, 3, 5, 6, 7, 8] # 5
>>> [2, 3, 5, 6, 7] # 6
```

Теперь мы знаем, как удалять элемент из списка по его индексу. Но что, если мы не знаем индекса элемента, но знаем его значение? Для такого случая у нас есть метод remove(), который удаляет первый найденный по значению элемент в списке.

```
all_types = [10, 'Python', 10, 3.14, 'Python', ['I', 'am', 'list']]
all_types.remove(3.14)
print(all_types) # 1
all_types.remove(10)
print(all_types) # 2
all_types.remove('Python')
print(all_types) # 3
>>> [10, 'Python', 10, 'Python', ['I', 'am', 'list']] # 1
>>> ['Python', 10, 'Python', ['I', 'am', 'list']] # 2
>>> [10, 'Python', ['I', 'am', 'list']] # 3
```

А сейчас немного посчитаем, посчитаем элементы списка с помощью метода count()

```
numbers = [100, 100, 100, 200, 200, 500, 500, 500, 500, 500, 999]
print(numbers.count(100)) # 1
print(numbers.count(200)) # 2
print(numbers.count(500)) # 3
print(numbers.count(999)) # 4
>>> 3 # 1
>>> 2 # 2
>>> 5 # 3
>>> 1 # 4
```

В программировании, как и в жизни, проще работать с упорядоченными данными, в них легче ориентироваться и что-либо искать. Метод sort() сортирует список по возрастанию значений его элементов.

```
numbers = [100, 2, 11, 9, 3, 1024, 567, 78]
numbers.sort()
print(numbers) # 1
fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
fruits.sort()
print(fruits) # 2
>>> [2, 3, 9, 11, 78, 100, 567, 1024] # 1
>>> ['Apple', 'Banan', 'Grape', 'Orange', 'Peach'] # 2
```

Мы можем изменять порядок сортировки с помощью параметра reverse. По умолчанию этот параметр равен False

```
fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
fruits.sort()
print(fruits) # 1
fruits.sort(reverse=True)
```

```
print(fruits) # 2
>>> ['Apple', 'Banan', 'Grape', 'Orange', 'Peach'] # 1
>>> ['Peach', 'Orange', 'Grape', 'Banan', 'Apple'] # 2
```

Иногда нам нужно перевернуть список, не спрашивайте меня зачем... Для этого в самом лучшем языке программирования на этой планете JavaScr..Python есть метод reverse():

```
numbers = [100, 2, 11, 9, 3, 1024, 567, 78]
numbers.reverse()
print(numbers) # 1
fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
fruits.reverse()
print(fruits) # 2
>>> [78, 567, 1024, 3, 9, 11, 2, 100] # 1
>>> ['Apple', 'Banan', 'Peach', 'Grape', 'Orange'] # 2
```

Допустим, у нас есть два списка и нам нужно их объединить. Программисты на C++ сразу же кинулись писать циклы for, но мы пишем на python, а в python у списков есть полезный метод extend(). Этот метод вызывается для одного списка, а в качестве аргумента ему передается другой список, extend() записывает в конец первого из них начало второго:

```
fruits = ['Banana', 'Apple', 'Grape']
vegetables = ['Tomato', 'Cucumber', 'Potato', 'Carrot']
fruits.extend(vegetables)
print(fruits)
>>> ['Banana', 'Apple', 'Grape', 'Tomato', 'Cucumber', 'Potato', 'Carrot']
```

В природе существует специальный метод для очистки списка — clear()

```
fruits = ['Banana', 'Apple', 'Grape']
vegetables = ['Tomato', 'Cucumber', 'Potato', 'Carrot']
fruits.clear()
vegetables.clear()
print(fruits)
print(vegetables)
>>> []
>>> []
```

Осталось совсем чуть-чуть всего лишь пара методов, так что делаем последний рывок! Метод index() возвращает индекс элемента. Работает это так: вы передаете в качестве аргумента в index() значение элемента, а метод возвращает его индекс:

```
fruits = ['Banana', 'Apple', 'Grape']
print(fruits.index('Apple'))
print(fruits.index('Banana'))
print(fruits.index('Grape'))
>>> 1
>>> 0
>>> 2
```

Финишная прямая! Метод copy(), только не падайте, копирует список и возвращает его брата-близнеца. Вообще, копирование списков - это тема достаточно интересная, давайте рассмотрим её по-подробнее.

Во-первых, если мы просто присвоим уже существующий список новой переменной, то на первый взгляд всё выглядит неплохо:

```
fruits = ['Banana', 'Apple', 'Grape']
new_fruits = fruits
print(fruits)
print(new_fruits)
>>> ['Banana', 'Apple', 'Grape']
>>> ['Banana', 'Apple', 'Grape']
```

Но есть одно маленькое "НО":

```
fruits = ['Banana', 'Apple', 'Grape']
new_fruits = fruits
fruits.pop()
print(fruits)
print(new_fruits)
# Внезапно, из списка new_fruits исчез последний элемент
>>> ['Banana', 'Apple']
>>> ['Banana', 'Apple']
```

При прямом присваивании списков копирования не происходит. Обе переменные начинают ссылаться на один и тот же список! То есть если мы изменим один из них, то изменится и другой. Что же тогда делать? Пользоваться методом copy(), конечно:

```
fruits = ['Banana', 'Apple', 'Grape']
new_fruits = fruits.copy()
fruits.pop()
print(fruits)
print(new_fruits)
>>> ['Banana', 'Apple']
>>> ['Banana', 'Apple', 'Grape']
```

Отлично! Но что если у нас список в списке? Скопируется ли внутренний список с помощью метода copy() — нет:

```
fruits = ['Banana', 'Apple', 'Grape', ['Orange','Peach']]
new_fruits = fruits.copy()
fruits[-1].pop()
print(fruits) # 1
print(new_fruits) # 2
>>> ['Banana', 'Apple', 'Grape', ['Orange']] # 1
>>> ['Banana', 'Apple', 'Grape', ['Orange']] # 2
```

Решение задач

1. Распечатайте второй элемент в списке fruits.
   ```
   fruits = ["яблоко", "банан", "вишня"]
   ```

print( )

2. Измените значение с "яблоко" на "киви" в списке fruits.
   fruits = ["яблоко", "банан", "вишня"]
   =
3. Используйте метод append добавить "апельсин" в список fruits.
   fruits = ["яблоко", "банан", "вишня"]
4. Используйте метод insert добавить "лимон" в качестве второго пункта в списке fruits.
   fruits = ["яблоко", "банан", "вишня"]
   "лимон")
5. Используйте метод remove удалите "банан" из списка fruits.
   fruits = ["яблоко", "банан", "вишня"]
6. Используйте отрицательную индексацию для печати последнего элемента в списке.
   fruits = ["яблоко", "банан", "вишня"]
   print( )
7. Используйте диапазон индексов для печати третьего, четвертого и пятого элемента в списке.
   fruits = ["яблоко", "банан", "вишня", "апельсин", "киви", "дыня", "манго"]
   print(fruits[ ])
8. Используйте правильный синтаксис для вывода количества элементов в списке.
   fruits = ["яблоко", "банан", "вишня"]
   print( )
9. Напишите программу, которая запрашивает с ввода восемь чисел, добавляет их в список. На экран выводит их сумму, максимальное и минимальное из них. Для нахождения суммы, максимума и минимума воспользуйтесь встроенными в Python функциями sum(), max() и min().
10. Напишите программу, которая генерирует сто случайных вещественных чисел и заполняет ими список. Выводит получившийся список на экран по десять элементов в ряд. Далее сортирует список с помощью метода sort() и снова выводит его на экран по десять элементов в строке. Для вывода списка напишите отдельную функцию, в качестве аргумента она должна принимать список.
11. Создайте список из 10 четных чисел и выведите его с помощью цикла for
12. Создайте список из 5 элементов. Сделайте срез от второго индекса до четвертого
13. Создайте пустой список и добавьте в него 10 случайных чисел и выведите их. В данной задаче нужно использовать функцию randint.
    from random import randint
    n = randint(1, 10) # Случайное число от 1 до 10

14. Удалите все элементы из списка, созданного в задании 3
15. Создайте список из введенной пользователем строки и удалите из него символы 'a', 'e', 'o'
16. Даны два списка, удалите все элементы первого списка из второго
    a = [1, 3, 4, 5]
    b = [4, 5, 6, 7]
    # Вывод
    >>> [6, 7]
17. Создайте список из случайных чисел и найдите наибольший элемент в нем.
18. Найдите наименьший элемент в списке из задания 7
19. Найдите сумму элементов списка из задания 7
20. Найдите среднее арифметическое элементов списка из задания 7
21. Создайте список из случайных чисел. Найдите номер его последнего локального максимума (локальный максимум — это элемент, который больше любого из своих соседей).
22. Создайте список из случайных чисел. Найдите максимальное количество его одинаковых элементов.
23. Создайте список из случайных чисел. Найдите второй максимум.
    a = [1, 2, 3] # Первый максимум == 3, второй == 2
24. Создайте список из случайных чисел. Найдите количество различных элементов в нем.

Практическая работа №17. Кортежи в Python

Цель: Познакомиться с языком программирования Python. Изучить особенности работы с кортежей и операции над ними

Теоретическая часть и терминология

Python - высокоуровневый язык программирования, ориентированный на повышение производительности разработчика и читаемость кода. Это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для тех, кто только начинает свой путь в программировании.

ПРИМЕЧАНИЕ (обязательно к прочтению)

IDLE - программа для написания кода (интегрированная среда разработки, IDE), поставляется вместе с языком программирования Python. В ней удобно писать небольшие программы и учиться языку Python.

Сразу обратите внимание, что наши привычные горячие комбинации клавиш, вроде копировать/вставить - Ctrl+C, Ctrl+V, или отмена - Ctrl+Z в ней будут работать только на английской раскладке, поэтому когда ими пользуетесь, всегда проверяйте на какой вы раскладке.

Если в программе для вас мелкий шрифт изменить его можно нажав Options – Configyre IDLE.

Для запуска программы нажмите наверху Run -> Run Module, или просто F5. И у вас откроется Python Shell.

Python Shell - это интерактивный интерпретатор языка Python, так называемая "оболочка Python". Что-то вроде консоли, или терминала в нашей IDLE. Здесь мы можем писать команды и они будут сразу выполняться. Также он запускает и наши написанные программы.

>>> - это приглашение к написанию кода, именно здесь мы и будем давать команды.

Будем работать сразу одновременно в двух форматах и в IDLE и в Shell, поэтому во время работы не закрываем не одно из окон.

Задания выполняются только в тех окнах, в которых вам задано в задании.

Никакие строки кона не стираются

При сохранении работ будьте внимательны. Вам необходимо сохранить оба файла.

IDLE сохраняется автоматически после запуска, но лучше пересохраните его еще раз (Ctrl+S или File – Save as). Имя вы задаете в начале работы.

Кортежи (tuple) в Python – это те же списки за одним исключением. Кортежи неизменяемые структуры данных. Так же как списки они могут состоять из элементов разных типов, перечисленных через запятую. Кортежи заключаются в круглые, а не квадратные скобки.

```
>>> a = (10, 2.13, "square", 89, 'C')
>>> a
(10, 2.13, 'square', 89, 'C')
```

Кортеж (tuple) - это упорядоченная неизменяемая последовательность элементов.

Особенности: умеет все, что умеет список, за исключением операций, приводящих к изменению кортежа. Применяется в случаях, когда известно, что последовательность не будет меняться после создания.

class tuple([iterable])

Ход работы

В ранее созданной папке создаём текстовый документ. Для этого щёлкаем по пустому месту в папке правой кнопкой мыши и выбираем "создать" -> "текстовый документ". У вас появится текстовый документ, сотрите полностью его название и назовите pr19_z1.py, обязательно поставьте расширение .py. Далее каждое задание делаем в новом файле и продолжаем нумерацию pr19_z2.py, pr19_z3.py и т.д.

Открываем этот файл "Edit with IDLE" -> версию своей IDLE. Щёлкаем левой кнопкой мыши и у нас откроется пустой документ.

Создать кортеж можно несколькими способами

Создание кортежа

1. Пустой кортеж создается с помощью пустых круглых скобок или функции tuple()

```
>>> ()
()
>>> tuple()
()
```

2. Инициализировать кортеж элементами можно одним из следующих способов:

```
>>> 1,
(1,)
>>> 1, 2, "text"
(1, 2, 'text')
>>> s = tuple("text")
>>> s
('t', 'e', 'x', 't')
>>>
```

3. Т.к. структура является неизменяемой, изменение содержимого запрещено

```
>>> s[0] = "n"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Из кортежа можно извлекать элементы и брать срезы:

```
>>> a[3]
89
>>> a[1:3]
(2.13, 'square')
```

Однако изменять его элементы нельзя:

```
>>> a[0] = 11
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not
support item assignment
```

Также у типа **tuple** нет методов для добавления и удаления элементов.

Возникает резонный вопрос. Зачем в язык программирования был введен этот тип данных, по-сути представляющий собой неизменяемый список? Дело в том, что иногда надо защитить список от изменений. Преобразовать же кортеж в список, если это потребуется, как и выполнить обратную операцию легко с помощью встроенных в Python функций list() и tuple():

```
>>> a = (10, 2.13, "square", 89, 'C')
>>> b = [1, 2, 3]
>>> c = list(a)
>>> d = tuple(b)
>>> c
[10, 2.13, 'square', 89, 'C']
>>> d
(1, 2, 3)
```

Рассмотрим случай, когда уместно использовать кортежи. В Python изменяемые объекты передаются в функцию по ссылке. Это значит, что не создается копия объекта, а переменной-параметру присваивается ссылка на уже существующий объект. В итоге, если в теле функции объект изменяется, то эти изменения касаются глобального объекта.

```
def add_num(seq, num):
    for i in range(len(seq)):
        seq[i] += num
    return seq

origin = [3, 6, 2, 6]
changed = add_num(origin, 3)

print(origin)
print(changed)
```

Данная программа неправильная. Хотя никаких выбросов исключений не произойдет, она содержит логическую ошибку. На выводе получаем:

```
[6, 9, 5, 9]
[6, 9, 5, 9]
```

То есть исходный список был также изменен. Параметр *seq* содержал ссылку не на свой локальный список, а на список-оригинал. Таким образом, в операторе **return** здесь нет смысла. Если функция замысливалась как изменяющая глобальный список, то программа должна выглядеть так:

```
def add_num(seq, num):
    for i in range(len(seq)):
        seq[i] += num
```

```
origin = [3, 6, 2, 6]
add_num(origin, 3)
print(origin)
```

Что делать, если все же требуется не изменять исходный список, а сформировать по нему новый. Задачу можно решить несколькими способами. Во первых, в функции можно создать локальный список, после чего возвращать его:

```
def add_num(seq, num):
    new_seq = []
    for i in seq:
        new_seq.append(i + num)
    return new_seq
origin = [3, 6, 2, 6]
changed = add_num(origin, 3)
print(origin)
print(changed)
```

Результат:

```
[3, 6, 2, 6]
[6, 9, 5, 9]
```

Исходный список в функции не меняется. Его элементы лишь перебираются.

Второй способ защитить список-оригинал – использовать кортеж. Этот способ более надежный, так как в больших программах трудно отследить, что ни одна функция не содержит команд изменения глобальных данных.

Хотя преобразовывать к кортежу можно как при передаче в функцию, так и в самой функции, лучше сразу делать глобальный список кортежем. Поскольку неизменяемые объекты передаются по значению, а не по ссылке, то в функцию будет поступать копия структуры, а не оригинал. Даже если туда передается оригинал, изменить его невозможно. Можно лишь, как вариант, скопировать его и/или изменить тип, создав тем самым локальную структуру, и делать с ней все, что заблагорассудится.

```
def add_num(seq, num):
    seq = list(seq)
    for i in range(len(seq)):
        seq[i] += num
    return seq
origin = (3, 6, 2, 6)
changed = add_num(origin, 3)
print(origin)
print(changed)
```

Списки в кортежах

Кортежи могут содержать списки, также как списки быть вложенными в другие списки.

```
>>> nested = (1, "do", ["param", 10, 20])
```

Как вы думаете, можем ли мы изменить список ["param", 10, 20] вложенный в кортеж *nested*? Список изменяем, кортеж – нет. Если вам кажется, что нельзя, то вам кажется неправильно. На самом деле можно:

```
>>> nested[2][1] = 15
>>> nested
(1, 'do', ['param', 15, 20])
```

Примечание. Выражения типа nested[2][1] используются для обращения к вложенным объектам. Первый индекс указывает на позицию вложенного объекта, второй – индекс элемента внутри вложенного объекта. Так в данном случае сам список внутри кортежа имеет индекс 2, а элемент списка 10 – индекс 1 в списке.

Странная ситуация. Кортеж неизменяем, но мы все-таки можем изменить его. На самом деле кортеж остается неизменяемым. Просто в нем содержится не сам список, а ссылка на него. Ее изменить нельзя. Но менять сам список можно.

Чтобы было проще понять, перепишем кортеж так:

```
>>> l = ["param", 10, 20]
>>> t = (1, "do", l)
>>> t
(1, 'do', ['param', 10, 20])
```

Кортеж содержит переменную-ссылку. Поменять ее на другую ссылку нельзя. Но кортеж не содержит самого списка. Поэтому его можно менять как угодно:

```
>>> l.pop(0)
'param'
>>> t
(1, 'do', [10, 20])
```

Однако такой номер не проходит с неизменяемыми типами:

```
>>> a = "Kat"
>>> t = (a, l)
>>> t
('Kat', [10, 20])
>>> a = "Bat"
>>> t
('Kat', [10, 20])
```

Они передаются в кортеж как и в функцию – по значению. То есть их значение копируется в момент передачи.

Решение задач

1. Чтобы избежать изменения исходного списка, не обязательно использовать кортеж. Можно создать его копию с помощью метода списка copy() или взять срез от начала до конца [:]. Скопируйте список первым и вторым способом и убедитесь, что изменение копий никак не отражается на оригинале.
2. Заполните один кортеж десятью случайными целыми числами от 0 до 5 включительно. Также заполните второй кортеж числами от -5 до 0. Для заполнения кортежей числами напишите одну функцию. Объедините два кортежа с помощью оператора +, создав тем самым третий кортеж. С помощью

метода кортежа count() определите в нем количество нулей. Выведите на экран третий кортеж и количество нулей в нем.

Дополнительные задания:

3. Зайдите в Znanium под свой учетной записью.
4. Найдите учебник С.Р. Гуриков Основы алгоритмизации и программирования на Python (https://znanium.com/catalog/document?id=390096)
5. Повторите материал и выполните успражнения со страниц 112-145

Практическая работа №17. Сортировка. Сравнение списков и кортежей в Python

Цель: Познакомиться с языком программирования Python. Изучить особенности работы с кортежей и операции над ними

Теоретическая часть и терминология

Списки и кортежи — важные структуры данных в Python. Иногда вам может понадобиться отсортировать список кортежей. В этой статье мы рассмотрим, как использовать функцию sorted() и метод sort(), а также разберем различия между ними. Вы узнаете, как происходит сортировка кортежей в Python, и увидите, как дополнительные параметры key и reverse расширяют возможности сортировки.

Инициализация списков и кортежей

Во-первых, что такое список? Начинающему пользователю легко перепутать списки с массивами. Однако списки и массивы — это принципиально разные структуры данных. Массив — это структура данных фиксированной длины, в то время как список — это структура данных переменной длины, размер которой можно изменять.

В Python списки относятся к изменяемым типам данных. Они могут содержать любые данные, включая строки, кортежи и множества. В этой статье мы будем рассматривать исключительно списки кортежей.

Существует два способа инициализации пустого списка. Во-первых, вы можете указать пару квадратных скобок без каких-либо значений. Пример:

lst = []

Мы можем добавить несколько значений по умолчанию между нашими квадратными скобками. Списки Python позволяют нам иметь любое количество элементов разных типов:

jobs = ['Software Engineer','Data Analyst', 15, True]

Здесь мы объявили объект списка jobs с четырьмя начальными значениями: 'Software Engineer' и 'Data Analyst' (хранятся как строки), 15 (целое число) и булево значение True.

Еще один способ инициализировать пустой список в Python — использовать конструктор list(). Вот пример конструктора list() в действии:

lst = list()

Конструктор list() принимает один необязательный аргумент — итерируемый объект (англ. iterable). Если конструктору не передан параметр, он возвращает пустой список. Если передан параметр iterable, он возвращает список элементов iterable.

Оба подхода дают один и тот же результат: пустой список. Каких-либо рекомендаций по выбору того или иного подхода нет. Но в силу лаконичности обычно предпочтение отдается пустым квадратным скобкам [].

Теперь давайте разберемся, что такое кортежи в Python.

Кортежи, как и списки, представляют собой структуры данных с упорядоченным рядом из нуля или более записей. Основное различие между кортежами и списками заключается в том, что кортежи не могут быть изменены (они неизменяемы), а списки, как уже говорилось, изменяемы.

Для инициализации кортежа значениями используется последовательность значений, разделенных запятыми. Как и в случае со списками, есть два способа

инициализации пустого кортежа. Как видите, эти методы инициализации довольно похожи:

```
# Метод 1
tup = ()
# Метод 2
tup = tuple()
```

Важно помнить, что если вы хотите создать кортеж с одним значением, после значения нужно поставить запятую:

```
# Кортеж с 1 элементом
tup = (1, )
```

Функция sorted()

Теперь давайте рассмотрим сортировку в Python.

Сортировка списка в Python может быть легко выполнена с помощью функции sorted(). Она возвращает список с элементами, отсортированными в порядке возрастания или убывания. Синтаксис этой функции следующий:

```
sorted(list, key=..., reverse=...)
```

Обратите внимание, что key и reverse — два необязательных параметра:

- reverse: если этот параметр имеет значение True, то список будет сортироваться в обратном порядке (по убыванию)
- key: функция, которая действует как своего рода ключ сравнения.

Метод sort()

Метод sort() сортирует элементы списка по возрастанию (поведение по умолчанию) или по убыванию.

Синтаксис метода sort() следующий:

```
list.sort(key=..., reverse=...)
```

По умолчанию параметр reverse имеет значение False. При этом метод сортирует список в порядке возрастания.

Основное различие между sort() и sorted() заключается в том, что sort() сортирует список на месте, то есть изменяет сам список и возвращает значение None. Функция sorted() создает копию исходного списка, сортирует его и возвращает этот новый список, оставляя исходный список неизменным.

Теперь давайте перейдем к сортировке списков кортежей в Python.

Сортировка списка кортежей с помощью функции sorted()

Чтобы понять, как сортировать кортежи в Python, давайте рассмотрим пример. Допустим, у нас есть следующий список кортежей:

```
sample = [(2, 4, 3), (3, 5, 7), (1, 0, 1)]
```

Теперь применим функцию sorted():

```
print(sorted(sample))
# Output:
# [(1, 0, 1), (2, 4, 3), (3, 5, 7)]
```

Как видите, порядок кортежей в списке изменился.

По умолчанию метод sorted() сортирует кортежи, ориентируясь на их первые элементы. В результате первый кортеж отсортированного списка начинается с 0, второй — с 2, а третий — с 3.

Посмотрим, что произойдет, если первые элементы всех кортежей будут одинаковыми:

```
sample = [(1, 4, 3), (1, 5, 7), (1, 0, 1)]
print(sorted(sample))
# Output:
# [(1, 0, 1), (1, 4, 3), (1, 5, 7)]
```

Критерием сортировки становится следующие (вторые) элементы кортежей.

Если вторые элементы всех кортежей в списке одинаковы, то сортировка кортежей будет производиться по третьему элементу. Если первый, второй и третий элементы одинаковы, то используется четвертый элемент и так далее.

Сортировка списка кортежей с помощью метода sort()

Метод sort() также можно использовать для сортировки кортежей в Python. Начнем с исходного списка кортежей:

```
sample = [(2, 4, 3), (3, 5, 7), (1, 0, 1)]
sample.sort()
print(sample)
# Output:
# [(1, 0, 1), (2, 4, 3), (3, 5, 7)]
```

Обратите внимание, что вывод будет одинаковым независимо от того, используете ли вы sort() или sorted(). Однако sort() изменяет сам объект, а sorted() возвращает отсортированную копию списка.

Сортировка списка по второму элементу кортежа

Если вы хотите отсортировать список кортежей по заданному элементу, вы можете использовать метод sort() и указать лямбда-функцию в качестве ключа.

К сожалению, Python не позволяет напрямую указывать индекс элемента сортировки. Вместо этого внутри параметра key должна быть определена функция, которая поможет методу sort() выбрать нужный индекс. Функция лямбда позволяет указать в методе sort() пользовательскую функцию-компаратор.

Например:

```
sample = [('Jack', 76), ('Beneth', 78), ('Cirus', 77), ('Faiz', 79)]
sample.sort(key=lambda a: a[1])
print(sample)
```

Выполнение этого кода (который предписывает Python отсортировать список кортежей по второму элементу), даст следующий результат:

[(‘Jack’, 76), (‘Cirus’, 77), (‘Beneth’, 78), (‘Faiz’, 79)]

Сортировка списка кортежей в обратном порядке

Python позволяет указать порядок сортировки списка. Для сортировки списка кортежей по убыванию мы можем просто использовать тот же метод sort(). Сначала мы возьмем несортированный список кортежей, а затем вызовем метод sort(). Если передать ему в качестве аргумента reverse=True, это изменит порядок сортировки с возрастающего на убывающий.

Давайте рассмотрим это на примере:

```
sample = [(1, 0, 0), (4, 1, 2), (4, 2, 5), (1, 2, 1)]
sample.sort(reverse=True)
print(sample)
```

 # Output:

 # [(4, 2, 5), (4, 1, 2), (1, 2, 1), (1, 0, 0)]

Мы также можем отсортировать список кортежей по второму элементу в обратном порядке:

```
sample = [('f', 90), ('g', 84), ('d', 92), ('a', 96)]
sample.sort(key=lambda i:i[1], reverse=True)
print(sample)
# Output:
# [('a', 96), ('d', 92), ('f', 90), ('g', 84)]
```

Использование sorted() без лямбда-функции

В предыдущем примере мы передали лямбда-функцию в качестве необязательного аргумента key метода sort(). Но мы можем разработать обычную функцию, имитирующую встроенную функциональность лямбда-функции, и передать ее в качестве key.

Давайте определим функцию second_item(), которая принимает кортеж в качестве параметра и возвращает второй элемент кортежа.

```
def second_item(data):
return data[1]
```

А теперь вызовем функцию sorted() и передадим в качестве аргумента key нашу функцию second_item:

```
sample = [(1, 7, 3), (4, 9, 6), (7, 3, 9)]
sample.sort(key=select_item)
print(sample)
# Output:
# [(7, 3, 9), (1, 7, 3), (4, 9, 6)]
```

Результат правильный.

Сортировка списка кортежей с помощью itemgetter()

Альтернативой использованию лямбды или пользовательской функции в параметре key является использование функции itemgetter() из модуля operator. Прежде чем применить эту функцию, давайте рассмотрим, как она работает:

```
from operator import itemgetter
print(itemgetter(1)((2, 5, 3)))
# Output:
# 5
```

Функция itemgetter() возвращает элемент по переданному ей индексу (в данном случае 1). Теперь давайте воспользуемся функцией itemgetter() для сортировки нашего списка кортежей:

```
sample = [(1, 7, 3), (4, 9, 6), (7, 3, 9)]
print(sorted(sample, key=itemgetter(1)))
# Output:
# [(7, 3, 9), (1, 7, 3), (4, 9, 6)]
```

Здесь мы опять сортируем список кортежей по их вторым элементам.

Сортировка списка кортежей по двум элементам

До сих пор мы рассматривали сортировку кортежей по какому-нибудь одному элементу. Но для сравнения можно использовать и два, и более элементов.

Мы по-прежнему будем использовать необязательный аргумент key, а также лямбда-функцию. Допустим, мы хотим отсортировать список по второму и третьему элементам каждого кортежа.

Мы можем использовать лямбду для возврата кортежа из двух значений, которые определяют положение элементов, используемых для сортировки, при передаче кортежа в качестве входных данных.

```
sample = [(2, 4, 7), (3, 0, 1), (3, 0, 0)]
print(sorted(sample, key=lambda x: (x[1], x[2])))
# Output:
# [(3, 0, 0), (3, 0, 1), (2, 4, 7)]
```

Кортежи в списке сортируются по второму элементу, а затем по третьему. Если изменить порядок элементов, заданный в лямбде, то элементы будут отсортированы по второму элементу, а затем по третьему.

Сортировка списка кортежей может показаться сложной задачей, но в Python это легко. Язык предоставляет все необходимое для сортировки без необходимости написания собственных функций. Создавать собственную функцию придется разве что для передачи в параметре key.

Практическая работа №19. Работа с массивами и множествами в Python

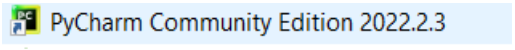
Цель: познакомиться с языком программирования Python. Изучить особенности работы с множествами на Python.

Теоретическая часть

Язык Python (по-русски можно произносить как Пайтон или Питон) появился в 1991 году и был разработан Гвидо ван Россумом. Язык назван в честь шоу "Летающий цирк Монти Пайтона". Одна из главных целей разработчиков – сделать язык забавным для использования.

Сейчас Питон регулярно входит в десятку наиболее популярных языков и хорошо подходит для решения широкого класса задач: обучение программированию, скрипты для обработки данных, машинное обучение, серверная веб-разработка и многое другое. Большинству из вас язык Питон потребуется для написания проектов и в ряде предметов третьего курса, а также в повседневном быту для автоматизации задач обработки данных.

Ход работы

Откройте программу PyCharm 

(через ярлык на рабочем столе или через пуск)

Перед нами окошко PyCharm Community Edition — это среда программирования, в которой мы будем работать.

Создайте проект под названием PR_21

Правой кнопкой нажимаем на название нашего проекта или File, New File. New — и нам нужен Python File.

Собственно, это файл с кодом, в котором будет находиться наша программа.

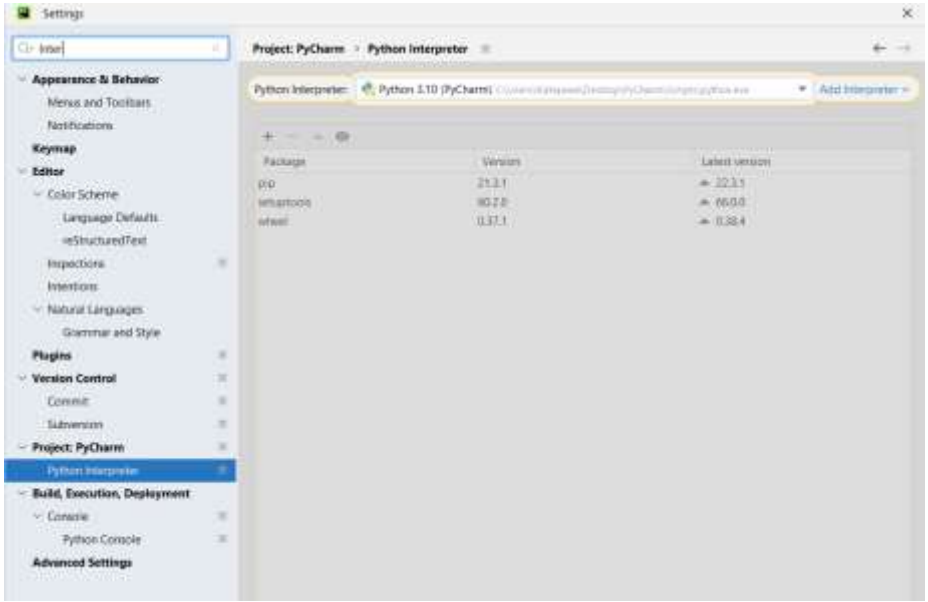
Даем ему название в соответствии с заданием и нажимаем Enter.

И все, теперь у нас есть текстовое окошко — можно считать, что это просто текстовый редактор, только специально приспособленный для написания программ на Питоне.

Примечание. Если вдруг опять слетела оболочка IDLE и не происходит запуск кода, тогда выполните следующие действия

Нажмите комбинацию клавиш Ctrl + Alt + S

В окне поиска наберите Interpreter



Если в строке Python Interpreter стоит, что интепретер отсутствует, то его необходимо добавить

Нажимаем Add Interpreter

Далее Add Local Interpreter

Переходим на вкладку System Interpreter

Нажимаем на три точки

В появившемся окне указываем путь к нашей оболочке

Для ваших ПК будет следующий путь (для домашних ПК путь будет подобным, т.е. будет меняться только имя пользователя):

C:\Users\Студент\AppData\Local\Programs\Python\Python310\python.exe

Все настройки закончены

Set в Python — это тип данных, встроенный в язык и предназначенный для работы с последовательностями, которые имеют свойства математических множеств. В качестве элементов набора данных выступают различные неизменяемые объекты. В python множество может состоять из не ограниченного количества элементов и они могут быть любых неизменяемых типов, таких как кортежи, числа, строки и т. д. Однако, множество не поддерживает mutable элементы, к примеру списки, словари, и прочие.

Множества в Python – это структура данных, которые содержат неупорядоченные элементы. Элементы также не является индексированным. Как и список, множество позволяет внесение и удаление элементов. Однако, есть ряд особенных характеристик, которые определяют и отделяют множество от других структур данных:

- Множество не содержит дубликаты элементов;
- Элементы множества являются неизменными (их нельзя менять), однако само по себе множество является изменяемым, и его можно менять;
- Так как элементы не индексируются, множества не поддерживают никаких операций **среза** и **индексирования**.

Как создать множество python

Перед тем как выполнять операции с множеством, необходимо его создать. Существует несколько способов создавать множества в Python. Можно это сделать, присвоив любой переменной коллекцию значений, окружив их фигурными скобками {}, а так же разделить все элементы запятыми.

1. Создание множества в Python выглядит следующим образом:

   my_set = {1, 2, 3, 4, 5}

   print(my_set)

   Это пример создания множества чисел.

2. Аналогичным образом создайте множество из строк (не менее 5 значений (слов)). Не забываем, что каждое слово пишется в кавычка.

Обратите внимание на то, что элементы в выводе находятся не в том порядке, в котором были добавлены в множество. Это происходит из-за того, что элементы множества находятся в случайном порядке. При повторном запуске того же кода, вывод может содержать элементы, которые каждый раз будут расположены в другом порядке.

3. Также можно создать множество с элементами различных типов:

```
my_set = {1, 2.0, «Ivan», True, (1, 2, 3)}
print(my_set)
```

Все элементы данного множества имеют различные типы данных. Обратите внимание, что в данном выводе нет элемента True. Скоро мы разберёмся в том, почему это произошло.

Существует другой путь, в котором также необходимо создавать множество из других коллекций. Это можно сделать, используя встроенную в Python3 функцию set(). Аргументом такой функции должна быть коллекция неких данных или строка с текстом.

4. Из списка:

```
my_set = set([1, 26, 4, 5, 6])
print(my_set)
```

5. Из кортежа:

```
my_set = set((1, 26, 4, 5, 6))
print(my_set)
```

6. Из словаря:

```
my_set = set({‘q’:1, ‘w’:2, ‘e’:3, ‘r’:4, ‘t’:5, ‘y’:6})
print(my_set)
```

Как уже говорилось, множества не содержат повторяющиеся элементы. Если последовательность включает дубликаты элементов, то из множества эти элементы будут автоматически удалены.

7. Множество повторяющихся значений

```
my_set = set([1, 3, 9, 27, 9, 3])
print(my_set)
```

Именно по этой причине в приведённом выше примере

my_set = {1, 2.0, «Ivan», True, (1, 2, 3)} в итоговом множестве отсутствует элемент True. Так как True и единица равны, Python справедливо посчитал эти элементы дубликатами и удалил один из них.

Создать пустое множество используя пустые фигурные скобки {} не получится, поскольку Python, встретив такую строку, создаст словарь, а не множество

```
example_dict = {}
print(type(example_dict))
```

Вывод:

<class ‘dict’>

Как показано в выводе, переменная — это словарь.

8. Для того, чтобы создать пустое множество, надо применить встроенную в язык и стандартную для этих случаев функцию set() без параметров:

```
my_set = set()
```

print(type(my_set))

Вывод показывает, что переменная my_set имеет тип множество (‘set’).

9. Создайте самостоятельно три множества. Перовое множество должно быть создано из списка и содержать пять ваших любимых напитков. Второе множество должно быть создано из кортежа и содержать пять ваших любимых фильмов. Третье множество должно быть создано из словаря и содержать пять ваших любимых книг.

Для создания множества можно воспользоваться генератором. Генератор позволяет заполнять кортежи, списки, а также другие наборы с учетом указанных условий. Заполнение проводится так же, как происходит генерация списков.

10. Генерация множества с использованием цикла for для нескольких чисел:

my_set = { _ for _ in [102, 21, 30, 51, 34, 72]}

print(my_set)

Добавляем элемент в множества python

Для изменения содержимого множеств в языке Пайтон присутствуют специальные методы, позволяющие добавлять и удалять различные элементы, а так же производить другие операции. Как и для создания set множества, существует несколько различных способов для добавления очередного элемента.

Добавление одного элемента в множество python

Добавить новый объект в уже существующее множество можно используя метод множества add(). Аргументом функции в данном случае является добавляемый элемент последовательности. Не стоит забывать, что добавлять можно только те данные, которые имеют неизменяемый тип.

11. Метод множества add()

my_set = set([1, 2, 9])

my_set.add(4) # добавляем элемент 4

print(my_set)

Если среди объектов, изначально составляющих set, данный элемент уже был, то ничего не произойдёт, и начальное множество останется неизменным.

my_set = set([1, 2, 9])

my_set.add(9) # пытаемся добавить элемент 9

print(my_set)

Вывод:

{1, 2, 9}

Добавление нескольких элементов в множество python

Добавление группы элементов в множество — это частный случай операции объединения множеств. Его отличие от стандартного объединения заключается в том, что добавление группы элементов изменяет первоначальное множество, а не создает новое.

Для добавления нескольких элементов необходимо использовать метод update, который принимает итерабельный объект (список, кортеж, генератор и т.п.) и добавляет все входящие в него элементы в исходное множество.

12. Добавление списка:

my_set = {1, 2, 3}

example_list = [‘q’, ‘w’, ‘e’]

my_set.update(example_list)

```
print(my_set)
```

13. Создайте два множества. В первом будут храниться дни рождения ваших близких, а во втором их имена. Путем добавления соедините эти множества в одно. Выполните запуск готового решения не менее пяти и раз и в комментарии к задаче напишите, почему у вас всегда получается разный результат.

Данный метод (update) можно использовать как альтернативу методу add и добавлять в множество всего один элемент с его помощью.

Удаляем элементы из множеств python

Метод remove удаляет элемент из коллекции set. В случае отсутствия такого элемента в наборе возникает исключение «KeyError «. В качестве входного параметра метода выступает элемент, который необходимо удалить.

14. Удаление элементов метод remove

```
# Объявляем множество
my_set = {1, 2, 3}
print(my_set)
# Удаляем элемент «1»
my_set.remove(1)
print(my_set)
# Удаляем элемент «5»
my_set.remove(5)
print(my_set)
# при попытке удалить 5 элемент будет ошибка KeyError: 5
```

15. Удаление элементов метод discard

Работает так же, как и remove(), но, в том случае если элемент отсутствует в коллекции, не выбрасывает исключение:

```
# Объявляем множество
my_set = {1, 2, 3}
print(my_set)
# Удаляем элемент «1»
my_set.discard(1)
print(my_set)
# Удаляем элемент «5»
my_set.discard (5)
print(my_set)
```

16. Удаление элементов метод pop

Метод pop удаляет тот элемент, который будет находиться в памяти первым. Хотелось бы отметить, что так как множества не упорядочены, то выглядит работа этого метода, как удаление случайного элемента. Этот метод не только удаляет один из элементов множества, но и возвращает его значение:

```
my_set = {«Ivan», «Rinat», «Olga», «Kira»}
print(my_set)
print(my_set.pop())
print(my_set)
```

17. Создайте три множества, содержащих по 5 элементов. Из первого удалите любой элемент используя метод remove, из второго удалите любой элемент используя метод discard, из третьего удалите элемент используя метод pop. Множества должны быть сначала выведены полностью, и только потом без элемента, который был удален.

Методы множеств python

Функция принадлежности членства

18. Оператор in используется для проверки принадлежности элемента набору:

```
my_set = {«Ivan», «Rinat», «Olga», «Kira»}
print({«Ivan» in my_set)
my_set = {«Ivan», «Rinat», «Olga», «Kira»}
print({«Anna» in my_set)
```

Разные функции

Кратко перечислим ещё несколько методов и функций, которыми можно пользоваться для работы с объектами множеств.

Когда необходимо полностью удалить все элементы, на помощь приходит метод clear(). Данный метод не принимает никаких аргументов. Он позволяет не удалять каждый элемент «вручную», а выполнить очистку всего множества одной командой.

19. Метод clear()

```
my_set = {102, 21, 330, 51, 34, 72}
my_set.clear() # Удаляем элементы множества my_set
print(my_set)
```

В множествах порядок следования элементов не учитывается. Поэтому бессмысленно говорить об их сортировке. Но, с другой стороны, дело обстоит немного иначе. Для быстрого поиска элементов, предпочтительно их записывать в память в упорядоченном виде. Это, естественно, понимали и создатели Пайтона. А что будет с элементами различных типов данных в одном множестве? Эти элементы, на сколько мы знаем, не должны сортироваться.

20. Порядок следования элементов

```
my_set = {1, 2.0, «Ivan», True, (1, 2, 3)}
print(my_set)
```

Как видим, у нас в выводе не отсортированные значения, при этом, если повторить запуск программы, то порядок, возможно, будет меняться. Но это только в том случае, если в множестве участвуют элементы разного типа.

21. Рассмотрим, что будет, если попытаться ввести только числа:

```
my_set = {6, 5, 4, 3, 2, 1}
print(my_set)
```

Все элементы выведены упорядоченно. Получается, что иногда объекты множества, если они одного типа, хранятся в памяти в строго упорядоченном виде. Но так бывает не всегда и лучше не стоит на это рассчитывать. Данное поведение зависит от внутренней логики обработки хеш-таблиц и может розница от версии к версии.

22. Если вам нужно получить отсортированную коллекцию из множества, лучше воспользоваться функцией sorted или встроенным методом списка sort. Элементы будут отсортированы и код будет понятен для других.

```
my_set = {i for i in range(3000, 0, -770)}
```

```
print(my_set)
my_list = list(my_set)
my_list.sort()
print(my_list)
```

copy() — возвращает не глубокую копию множества.

difference_update() — удаляет из заданного множества все элементы второго множества.

intersection_update() — множество обновляется пересечением со вторым множеством.

symmetric_difference_update() — обновляет set разницей между первым и вторым множеством и при этом разница симметричная.

Узнать количество элементов, из которых состоит set, можно при помощи функции len(), принимающей набор значений в качестве аргумента

Так же стоит отметить, что существует такой тип данных, как неизменяемое множество. Следовательно, в Python frozenset поддерживает все методы обычных множеств кроме тех, которые его изменяют.

Объединение множеств.

При использовании объединения множеств создаётся ещё одно множество, которое содержит все элементы, входящие в первоначальные два (без дубликатов). Операция объединения в языке Python выполняется несколькими способами: используя символ «|» или встроенный метод union().

23. Символ «|»

```
# используя символьный метод
set_a = {1, 9, 22, 4}
set_b = {3, 4, 5, 6}
set_c = set_a | set_b
print(set_c)
```

24. Метод union

```
# используя метод union
set_a = {1, 9, 22, 4}
set_b = {3, 4, 5, 6}
set_c = set_a.union(set_b)
print(set_c)
```

Пересечение множеств

При использовании пересечения множеств создаётся новый объект-множество, содержащий все элементы, общие для обоих (без дубликатов). Операция пересечения выполняется несколькими способами: используя символ & или метод intersection(). Методы дают одинаковый результат.

25. Символьный метод

```
# используя символьный метод
set_a = {1, 9, 22, 4}
set_b = {3, 4, 5, 6}
set_c = set_a & set_b
print(set_c)
```

26. Метод intersection

```
# используя метод intersection
set_a = {1, 9, 22, 4}
set_b = {3, 4, 5, 6}
set_c = set_a.intersection(set_b)
print(set_c)
```

Разность множеств

При использовании разности множеств Питон создаёт новый объект, включающий элементы, которые есть в первом, но не входят во второй (в данном случае — в множестве set_a). Операция разности выполняется несколькими способами: с помощью символа, а так же метода python set — difference().

27. Символьный метод

```
# используя символьный метод
set_a = {1, 9, 22, 4}
set_b = {3, 4, 5, 6}
set_c = set_a — set_b
print(set_c)
```

28. Метод difference

```
# используя метод difference
set_a = {1, 9, 22, 4}
set_b = {3, 4, 5, 6}
set_c = set_a.difference(set_b)
print(set_c)
```

Симметричная разность множеств

При применении симметричной разности на двух set-объектах создаётся новый объект, включающий все элементы, за исключением тех, которые есть в обоих. Выполняется симметричная разность несколькими способами: используя символ ^, или метод symmetric_difference().

29. Символьный метод

```
# используя символьный метод
set_a = {1, 9, 22, 4}
set_b = {3, 4, 5, 6}
set_c = set_a ^ set_b
print(set_c)
```

30. Метод symmetric_difference

```
# используя метод symmetric_difference
set_a = {1, 9, 22, 4}
set_b = {3, 4, 5, 6}
set_c = set_a.symmetric_difference(set_b)
print(set_c)
```

31. На основании 3 исходных множеств (передаются в качестве аргументов функции diff()) требуется написать функцию, которая будет возвращать либо симметричную разность, либо просто разность (если дополнительный аргумент функции symmetric имеет значение False) приведенных объектов в порядке: 1-ое множество, 2-ое множество, 3-е множество.

Разность двух множеств показывает уникальные элементы первого, которых нет во втором.

Симметричная разность выводит уникальные компоненты обоих контейнеров, исключая общие для них.

Решение можно провести как с помощью операторов, так и методов.

Вывод функции оформите самостоятельно, т.е. после создания функции вам нужно будет задать три множества set_1, set_2, set_3, и вывести через print результат выполнения функции

31.1. Вариант 1. При помощи операторов

```
def diff(set_1, set_2, set_3, symmetric=True):
    if symmetric:
        return set_1 ^ set_2 ^ set_3
    else:
        return set_1 - set_2 - set_3
```

Проверьте работу программы при разных значениях во всех set и при совпадающих значениях set.

Проверьте два вывода print с включением всех элементов и без включения в вывод symmetric=True

31.2. Вариант 2. При помощи методов

```
def diff(set_1, set_2, set_3, symmetric=True):
    if symmetric:
        return set_1.symmetric_difference(set_2).symmetric_difference(set_3)
    return set_1.difference(set_2, set_3)
```

Проверьте два вывода print с включением всех элементов и без включения в вывод symmetric. В выводе при включение symmetric задайте ему значение равное False.

32. Напишите функцию superset(), которая принимает 2 множества. Результат работы функции: вывод в консоль одного из сообщений в зависимости от ситуации:

1 - «Супермножество не обнаружено»

2 – «Объект {X} является чистым супермножеством»

3 – «Множества равны»

Для написания функции необходимо выявить подчиненность множеств, наличие чистого супермножества. При помощи print() вывести ответ по результатам оценки.

Расставьте правильно отступы самостоятельно.

Сначала реализуем функцию, только потом вывод

Вывод функции оформите самостоятельно (после создания функции) по следующему алгоритму:

```
# Тесты
set_1 = {1, 8, 3, 5} # множество должно содержать четыре любых числа
set_2 = {3, 5} # множество должно содержать два любых числа
set_3 = {5, 3, 8, 1} # множество должно содержать четыре числа, аналогичных set_1, но заданных в другом порядке
set_4 = {90, 100} # множество должно содержать два любых числа
superset(set_1, set_2) # вывод оформляем без print
```

Самостоятельно оформите вывод следующих условий (set_1, set_3), (set_2, set_3), (set_4, set_2)

```
# создание функции
def superset(set_1, set_2):
if set_1 > set_2:
print(f'Объект {set_1} является чистым супермножеством')
elif set_1 == set_2:
print(f'Множества равны')
elif set_1 < set_2:
print(f'Объект {set_2} является чистым супермножеством')
else:
print('Супермножество не обнаружено')
```

33. Соревнования по игре «Тетрис-онлайн»

Соревнования по игре «Тетрис-онлайн» проводятся по следующим правилам.

Каждый участник регистрируется на сайте игры под определённым игровым именем. Имена участников не повторяются.

Чемпионат проводится в течение определённого времени. В любой момент этого времени любой зарегистрированный участник может зайти на сайт чемпионата и начать зачётную игру. По окончании игры её результат (количество набранных очков) фиксируется и заносится в протокол.

Участники имеют право играть несколько раз. Количество попыток одного участника не ограничивается.

Окончательный результат участника определяется по одной игре, лучшей для данного участника.

Более высокое место в соревнованиях занимает участник, показавший лучший результат.

При равенстве результатов более высокое место занимает участник, раньше показавший лучший результат.

В ходе соревнований заполняется протокол, каждая строка которого описывает одну игру и содержит результат участника и его игровое имя. Протокол формируется в реальном времени по ходу проведения чемпионата, поэтому строки в нём расположены в порядке проведения игр: чем раньше встречается строка в протоколе, тем раньше закончилась соответствующая этой строке игра.

Напишите эффективную, в том числе по памяти, программу, которая по данным протокола определяет победителя и призёров. Гарантируется, что в чемпионате участвует не менее трёх игроков.

Перед текстом программы кратко опишите алгоритм решения задачи и укажите используемый язык программирования и его версию.

Описание входных данных

Первая строка содержит число N- общее количество строк протокола. Каждая из следующих N строк содержит записанные через пробел результат участника (целое неотрицательное число, не превышающее 100 миллионов) и игровое имя (имя не может содержать пробелов). Строки исходных данных соответствуют строкам протокола и расположены в том же порядке, что и в протоколе.

Гарантируется, что количество участников соревнований не меньше 3.

Описание выходных данных

Программа должна вывести имена и результаты трёх лучших игроков по форме, приведённой ниже в примере.

Самостоятельно подпишите в каждом input, что требуется ввести.

Пример входных данных:

```
9
69485 Jack
95715 qwerty
95715 Alex
83647 M
197128 qwerty
95715 Jack
93289 Alex
95715 Alex
95715 M
```

Пример выходных данных для приведённого выше примера входных данных:

```
1 место. qwerty (197128)
2 место. Alex (95715)
3 место. Jack (95715)
```

Решение:

```
score_table = {}
N = int(input())
for time in range(N):
    ball, name = input().split()
    ball = int(ball)
    if name in score_table:
        if ball > score_table[name][0]:
            score_table[name][0] = ball
            score_table[name][1] = time
    else:
        score_table[name] = [ball, time]
scores = list(score_table.items())
def score_key(a):
    return a[1][0]*100000000 - a[1][1]
scores.sort(key=score_key, reverse = True)
for winner_index in 0, 1, 2:
    print(winner_index + 1, 'место.', scores[winner_index][0], end =' ')
    print('(', scores[winner_index][1][0], ')', sep='')
```

34. Сдать багаж в камеру хранения.

На вход программе подаются сведения о пассажирах, желающих сдать свой багаж в камеру хранения на заранее известное время до полуночи. В первой строке сообщается число пассажиров N, которое не меньше 3, но не превосходит 1000; во второй строке – количество ячеек в камере хранения K, которое не меньше 10, но не превосходит 1000. Каждая из следующих N строк имеет следующий формат:

<Фамилия> <время сдачи багажа> <время освобождения ячейки>,

где <Фамилия> – строка, состоящая не более чем из 20 непробельных символов; <время сдачи багажа> – через двоеточие два целых числа, соответствующие часам (от 00 до 23 – ровно 2 символа) и минутам (от 00 до 59 – ровно 2 символа); <время освобождения ячейки> имеет тот же формат. <Фамилия> и <время сдачи багажа>, а также <время сдачи багажа> и <время освобождения ячейки> разделены одним пробелом. Время освобождения больше времени сдачи.

Сведения отсортированы в порядке времени сдачи багажа. Каждому из пассажиров в камере хранения выделяется свободная ячейка с минимальным номером. Если в момент сдачи багажа свободных ячеек нет, то пассажир уходит, не дожидаясь освобождения одной из них.

Требуется написать программу, которая будет выводить на экран для каждого пассажира номер ему предоставленной ячейки (можно сразу после ввода данных очередного пассажира). Если ячейка пассажиру не предоставлена, то его фамилия не печатается.

Самостоятельно подпишите в каждом input, что требуется ввести.

Пример входных данных:

3
10
Иванов 09:45 12:00
Петров 10:00 11:00
Сидоров 12:00 13:12

Результат работы программы на этих входных данных:

Иванов 1
Петров 2
Сидоров 1

Решение:

```
N = int(input())#число пассажиров
K = int(input()) #количество ячеек
time_outs = [None] * K #генератор списка
for i in range(N):
    name, time_in, time_out = input().split()
    for index, element in enumerate(time_outs):
        if not element or element < time_in:
            print(name, index + 1)
            time_outs[index] = time_out
            break
```

35. Распознавание чисел, записанных прописью.

Вам необходимо написать программу распознавания чисел, записанных прописью. Сначала на вход программе подается обучающий блок, состоящий из 27 строк. Первые 9 строк содержат слова «один», «два», …, «девять», следующие 9 строк – слова «одиннадцать», «двенадцать», … «девятнадцать», следующие 9 строк – слова «десять», «двадцать», …, «девяносто». Все слова записаны маленькими русскими буквами без лишних пробелов в начале и в конце строки.

Затем на вход программе подается значение N – количество записей, которые необходимо обработать. Следующие N строк содержат записанные словами числа.

Каждое число записано по-русски, маленькими буквами, без ошибок. Если число состоит из нескольких слов, между словами находится ровно один пробел, лишних пробелов в начале и в конце строк нет.

Напишите эффективную программу, которая определит сумму тех входных чисел, которые находятся в интервале от 1 до 99.

Размер памяти, которую использует Ваша программа, не должен зависеть от длины исходного списка.

Перед текстом программы кратко опишите используемый вами алгоритм решения задачи.

Самостоятельно подпишите в каждом input, что требуется ввести.

Пример входных данных (обучающий блок показан в примере с сокращениями):

один
два
три
четыре
пять
шесть
семь
восемь
девять
двенадцать
четырнадцать
пятнадцать
семнадцать
восемнадцать
девятнадцать
десять
двадцать
тридцать
сорок
пятьдесят
шестьдесят
семьдесят
восемьдесят
девяносто
5
двадцать восемь
два миллиона
четырнадцать
сто двадцать три
тысяча девятьсот восемьдесят четыре

Пример выходных данных для приведённого выше примера входных данных:

39

Решение:

Number = {}

```
for num in range(1,10):
    Number[input()] = num
for num in range(11,20):
    Number[input()] = num
for num in range(10, 100, 10):
    Number[input()] = num
N = int(input())
s = 0
for k in range(N):
    x = 0
    num = input().split()
    for written_num in num:
        if written_num in Number:
            x += Number[written_num]
        else:
            x = 0
            break
    s += x
print(s)
```

Практическая работа №20. Словари в Python

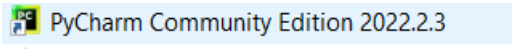
Цель: познакомиться с языком программирования Python. Изучить особенности работы со словарями на Python.

Теоретическая часть

Язык Python (по-русски можно произносить как Пайтон или Питон) появился в 1991 году и был разработан Гвидо ван Россумом. Язык назван в честь шоу "Летающий цирк Монти Пайтона". Одна из главных целей разработчиков – сделать язык забавным для использования.

Сейчас Питон регулярно входит в десятку наиболее популярных языков и хорошо подходит для решения широкого класса задач: обучение программированию, скрипты для обработки данных, машинное обучение, серверная веб-разработка и многое другое. Большинству из вас язык Питон потребуется для написания проектов и в ряде предметов третьего курса, а также в повседневном быту для автоматизации задач обработки данных.

Ход работы

Откройте программу PyCharm 

(через ярлык на рабочем столе или через пуск)

Перед нами окошко PyCharm Community Edition — это среда программирования, в которой мы будем работать.

Создайте проект

Правой кнопкой нажимаем на название нашего проекта или File, New File. New — и нам нужен Python File.

Собственно, это файл с кодом, в котором будет находиться наша программа.

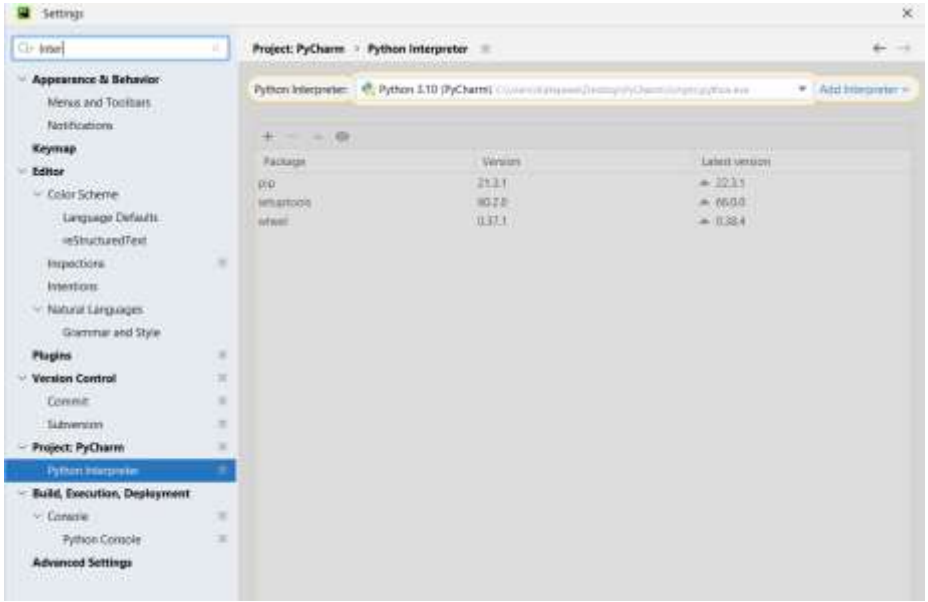
Даем ему название в соответствии с заданием и нажимаем Enter.

И все, теперь у нас есть текстовое окошко — можно считать, что это просто текстовый редактор, только специально приспособленный для написания программ на Питоне.

Примечание. Если вдруг опять слетела оболочка IDLE и не происходит запуск кода, тогда выполните следующие действия

Нажмите комбинацию клавиш Ctrl + Alt + S

В окне поиска наберите Interpreter



Если в строке Python Interpreter стоит, что интепретер отсутствует, то его необходимо добавить

Нажимаем Add Interpreter

Далее Add Local Interpreter

Переходим на вкладку System Interpreter

Нажимаем на три точки

В появившемся окне указываем путь к нашей оболочке

Для ваших ПК будет следующий путь (для домашних ПК путь будет подобным, т.е. будет меняться только имя пользователя):

C:\Users\Студент\AppData\Local\Programs\Python\Python310\python.exe

Все настройки закончены

Словарь — неупорядоченная структура данных, которая позволяет хранить пары «ключ — значение».

Создайте проект под названием Primer_20.

Решите все примеры из теоретического материала, исправляя ошибки в нем. Каждый код это новое задание. Для корректного решения вам понадобиться внимательно читать теорию и замечания к задачам.

Пример 1.

Вот пример словаря на Python:

```
dictionary = {'персона': 'человек',
              'марафон': 'гонка бегунов длиной около 26 миль',
              'противостоять': 'оставаться сильным, несмотря на давление',
              'бежать': 'двигаться со скоростью'}
```

Данный словарь использует строки в качестве ключей, однако ключом может являться в принципе любой неизменяемый тип данных. Значением же конкретного ключа может быть что угодно.

Пример 2.

Вот ещё один пример словаря, где ключами являются числа, а значениями — строки:

```
gender_dict = {0: 'муж',
               1: 'жен'}
```

Пример 3.

Важное уточнение: если вы попробуете использовать изменяемый тип данных в качестве ключа, то получите ошибку

```
dictionary = {(1, 2.0): 'кортежи могут быть ключами',
```

1: 'целые числа могут быть ключами',
'бежать': 'строки тоже',
['носок', 1, 2.0]: 'а списки не могут'}

Примечание. На самом деле проблема не с изменяемыми, а с нехэшируемыми типами данных, но обычно это одно и то же.

Получение данных из словаря

Пример 4.

Для решения задания скопируйте вначале Пример 1. Для получения значения конкретного ключа используются квадратные скобки []. Предположим, что в нашем словаре есть пара 'марафон': 26.

```
# берём значение с ключом "марафон"
dictionary['марафон']
```

Опять же, вы получите ошибку, если попытаетесь получить значение по несуществующему ключу. Для избежания подобных ошибок существуют методы, о которых мы сейчас поговорим.

Добавление и обновление ключей

Пример 5.

Для решения задания скопируйте вначале Пример 1.

Добавление новых пар в словарь происходит достаточно просто:

```
# Добавляем ключ "туфля" со значением "род обуви, закрывающей ногу не выше щиколотки"
dictionary['туфля'] = 'род обуви, закрывающей ногу не выше щиколотки'
```

Обновление существующих значений происходит абсолютно также:

```
# Обновляем ключ "туфля" и присваиваем ему значение "хорошая туфля"
dictionary['туфля'] = 'хорошая туфля'
```

Удаление ключей

Пример 6.

Для решения задания скопируйте вначале Пример 1.

Для удаления ключа и соответствующего значения из словаря можно использовать del

```
# Удаляем значение с ключом "противостоять" из словаря
del dictionary['противостоять']
```

Методы

Словари в Python имеют множество различных полезных методов, которые помогут вам в работе с ними.

Update

Пример 7.

Для решения задания скопируйте вначале Пример 1.

Метод update() пригодится, если нужно обновить несколько пар сразу. Метод принимает другой словарь в качестве аргумента.

```
# Добавляем две пары в словарь dictionary, используя метод update
dictionary.update({'бежал': 'бежать в прошедшем времени',
                   'туфли': 'туфля во множественном числе'})
 dictionary
{'марафон': 'гонка бегунов длиной около 26 миль',
```

'персона': 'человек',
'бежал': 'бежать в прошедшем времени',
'бежать': 'двигаться со скоростью',
'туфля': 'род обуви, закрывающей ногу не выше щиколотки',
'туфли': 'туфля во множественном числе'}

Если вас интересует, почему данные в словаре расположены не в том порядке, в котором они были внесены в него, то это потому что словари не упорядочены.

Get

Пример 8.

Для решения задания скопируйте вначале Пример 1.

```
# Допустим, у нас есть словарь story_count
story_count = {'сто': 100,
               'девяносто': 90,
               'двенадцать': 12,
               'пять': 5}
```

Метод get() возвращает значение по указанному ключу. Если указанного ключа не существует, метод вернёт None.

```
# Ключ "двенадцать" существует и метод get в данном случае вернёт 12
story_count.get('двенадцать')
```

Метод можно использовать для проверки наличия ключей в словаре:

```
story_count.get('два')
None
```

Также можно указать значение по умолчанию, которое будет возвращено вместо None, если ключа в словаре не окажется:

```
# Метод вернёт 0 в случае, если данного ключа не существует
story_count.get('два', 0)
```

Pop

Пример 9.

Для решения задания скопируйте вначале Пример 8.

Метод pop() удаляет ключ и возвращает соответствующее ему значение.

```
story_count.pop('девяносто')
90
story_count
{'двенадцать': 12, 'сто': 100, 'пять': 5}
```

Keys

Пример 10.

Для решения задания скопируйте вначале Пример 8.

Метод keys() возвращает коллекцию ключей в словаре.

```
story_count.keys()
['сто', 'пять', 'двенадцать']
```

Values

Пример 11.

Для решения задания скопируйте вначале Пример 8.

Метод values() возвращает коллекцию значений в словаре.

```
story_count.values()
```

[100, 12, 5]

Items

Пример 12.

Для решения задания скопируйте вначале Пример 1.

Метод items() возвращает пары «ключ — значение».

```
dictionary.items()
[('персона', 'человек'),
('бежать', 'двигаться со скоростью'),
('туфля', 'род обуви, закрывающей ногу не выше щиколотки'),
('бежал', 'бежать в прошедшем времени'),
('марафон', 'гонка бегунов длиной около 26 миль'),
('туфли', 'туфля во множественном числе')]
```

Итерация через словарь

Пример 13.

Для решения задания скопируйте вначале Пример 8.

Вы можете провести итерацию по каждому ключу в словаре.

```
for key in story_count:
  print(key)
```

Очевидно, вместо story_count можно использовать story_count.keys().

В примере кода ниже цикл for использует метод items() для получения пары «ключ — значение» на каждую итерацию.

```
for key, value in dictionary.items():
      print(key, value)
('персона', 'человек')
('бежать', 'двигаться со скоростью')
('туфля', 'род обуви, закрывающей ногу не выше щиколотки')
('бежал', 'бежать в прошедшем времени')
('марафон', 'гонка бегунов длиной около 26 миль')
('туфли', 'туфля во множественном числе')
```

Создайте проект под названием PR_20. Решите все задания.

Задание 20.1.

```
# Создание словаря с помощью литерала
student = {'name': 'Ivan',
           'age': 12}
print(student)
```

Задание 20.2.

Создайте словарь терминов по программированию (должно быть не менее 3 терминов)

Задание 20.3.

```
# Создание словаря с помощью функции dict()
credentials = dict(email='hacker1337@mail.ru',
                   password='123456')
print(credentials)
```

Задание 20.4.

Для получения значения конкретного ключа используются квадратные скобки [].

```
# Получаем значение с ключом 'name'
student = dict(name='Ivan', age=12)
print(student['name']) # -> Ivan
```

Задание 20.5.

Скопируйте код из задания 20.2. Отредактируйте его по примеру задания 20.4. Получите один из элементов созданных вами в задании 20.2

Задание 20.6

Обновление существующих значений происходит абсолютно также.

```
# Получаем значение с ключом 'name'
student = dict(name='Ivan', age=12)
student['name'] = 'Vasya'
print(student['name']) # -> Vasya
```

Задание 20.7

Скопируйте код из задания 20.2. Отредактируйте его по примеру задания 20.6. Обновите один из элементов, созданных в задании 20.2

Задание 20.8

Для удаления ключа и соответствующего значения из словаря можно использовать del

```
# Удаление ключа 'age'
student = dict(name='Ivan', age=12)
del student['age']
print(student) # -> {'name': 'Ivan'}
```

Задание 20.9

Скопируйте код из задания 20.2. Отредактируйте его по примеру задания 20.8. Удалите один из элементов, созданных в задании 20.2

Задание 20.10.

Метод get() возвращает значение по указанному ключу. Если указанного ключа не существует, метод вернёт None. Также можно указать значение по умолчанию, которое будет возвращено вместо None, если ключа в словаре не окажется.

```
# Использование метода get()
student = dict(name='Ivan', age=12)
print(student.get('name'))     # -> Ivan
print(student.get('lastname')) # -> None
print(student.get('lastname', 'No key')) # -> No key
```

Задание 20.11

Скопируйте код из задания 20.2. Отредактируйте его по примеру задания 20.10. Оформите возврат по указанному ключу для каждого из значений и добавьте несуществующий ключ созданных в задании 20.2

Задание 20.12

Метод pop() удаляет ключ и возвращает соответствующее ему значение.

```
# Использование метода pop()
sneakers = dict(brand='Adidas', price='9990 RUB', model='Nite Jogger')
model = sneakers.pop('model')
print(sneakers) # -> {'brand': 'Adidas', 'price': '9990 RUB'}
print(model)    # -> Nite Jogger
```

Метод keys() возвращает специальную коллекцию ключей в словаре.

Задание 20.13

```
# Использование метода keys()
sneakers = dict(brand='Adidas', price='9990 RUB', model='Nite Jogger')
print(sneakers.keys()) # -> dict_keys(['brand', 'price', 'model'])
# dict_keys - это неизменяемая коллекция элементов.
keys = list(sneakers.keys())
print(keys) # -> ['brand', 'price', 'model']
```

Задание 20.14

Метод values() возвращает специальную коллекцию значений в словаре.

```
# Использование метода values()
sneakers = dict(brand='Adidas', price='9990 RUB', model='Nite Jogger')
print(sneakers.values()) # -> dict_values(['Adidas', '9990 RUB', 'Nite Jogger'])
# dict_values - это неизменяемая коллекция элементов.
values = list(sneakers.values())
print(values) # -> ['Adidas', '9990 RUB', 'Nite Jogger']
```

Задание 20.15

Метод items() возвращает пары «ключ — значение» в формате кортежей.

```
# Использование метода items()
sneakers = dict(brand='Adidas', price='9990 RUB', model='Nite Jogger')
print(sneakers.items()) # -> dict_items([('brand', 'Adidas'), ('price', '9990 RUB'), ('model', 'Nite Jogger')])
# dict_items - это неизменяемая коллекция элементов.
items = list(sneakers.items())
print(items) # -> [('brand', 'Adidas'), ('price', '9990 RUB'), ('model', 'Nite Jogger')]
sneakers = dict(brand='Adidas', price='9990 RUB', model='Nite Jogger')
```

Задание 20.16

```
sneakers = dict(brand='Adidas', price='9990 RUB', model='Nite Jogger')
# Вывод ключей словаря с помощью цикла for
for key in sneakers:
    print(key)
# -> brand
# -> price
# -> model
```

Задание 20.17

Скопируйте код из задания 20.2. Отредактируйте его по примеру задания 20.16. Оформите вывод через for

Задание 20.18

```
# Вывод значений словаря с помощью цикла for
sneakers = dict(brand='Adidas', price='9990 RUB', model='Nite Jogger')
for value in sneakers.values():
    print(value)
# -> Adidas
# -> 9990 RUB
```

```
# -> Nite Jogger
```

Задание 20.19

```
# Вывод ключей и значений словаря с помощью цикла for
sneakers = dict(brand='Adidas', price='9990 RUB', model='Nite Jogger')
for key, value in sneakers.items():
    print(key, value)
# -> brand Adidas
# -> price 9990 RUB
# -> model Nite Jogger
```

Задание 20.20

Скопируйте задание 20.18. Создайте два аналогичных словаря и выведите их через for

Задание 20.21

Метод setdefault() возвращает значение ключа, но если его нет, создает ключ с указанным значением (по умолчанию None).

```
# Метод setdefault()
student = dict(name='Ivan', age=12)
student.setdefault('lastname', 'Ivanov')
print(student) # -> {'name': 'Ivan', 'age': 12, 'lastname': 'Ivanov'}
```

Задание 20.22. Пользователь

```
# Пользователь вводит имя, фамилия, возраст. Создайте словарь user и запишите данные пользователя в него.
firstname = input('Enter your firstname: ')
lastname = input('Enter your lastname: ')
user = dict(firstname=firstname, lastname=lastname)
print(user)
```

Добавьте ввод возраста в программу

Задание 20.23. Найти слово

```
# Выведите самое часто встречающееся слово в введенной строке.
list_of_words = ['hello', 'hello', 'hi']
words = {}
for word in list_of_words:
    words[word] = words.get(word, 0) + 1
# Функция max может получать функцию в качестве второго аргумента
freq_word = max(words, key=words.get)
print(freq_word)
```

Практическая работа №21. Объектно-ориентированное программирование в Python

Цель: познакомиться с языком программирования Python. Изучить основы объектно – ориентированного программирования в Python.

Теоретическая часть

Объект это конструкция, сочетающая в себе данные и процедуры, применимые к этим данным.

Объектно-ориентированное программирование это стиль программирования ориентированный на манипулирование объектами

- Процедурное программирование рассуждает в терминах данных и процедур их обработки
- Объектно-ориентированное программирование рассуждает в терминах объектов, их свойств и взаимосвязей

Создание иерархий объектов ведет к явному выделению уровней абстракции, более естественных чем в процедурном программировании

- В процедурном программировании выделяются процедуры которые делятся на более мелкие процедуры, и т.д.
- В объектно-ориентированном программировании модель реального мира представлена объектами которые делятся на более мелкие объекты, и т.д. Такой подход позволяет решать очень большие задачи.

Свойства объектно-ориентированного программирования

1. Инкапсуляция и абстракция
   - соединение данных и процедур их обработки в единую конструкцию
   - выделение интерфейса для взаимодействия с внешним миром
2. Наследование
   - создание новых типов данных на основе уже существующих с приобретением свойств "родительских" типов в дополнение к своим собственным
3. Полиморфизм
   - обработка объектов различных типов единым образом

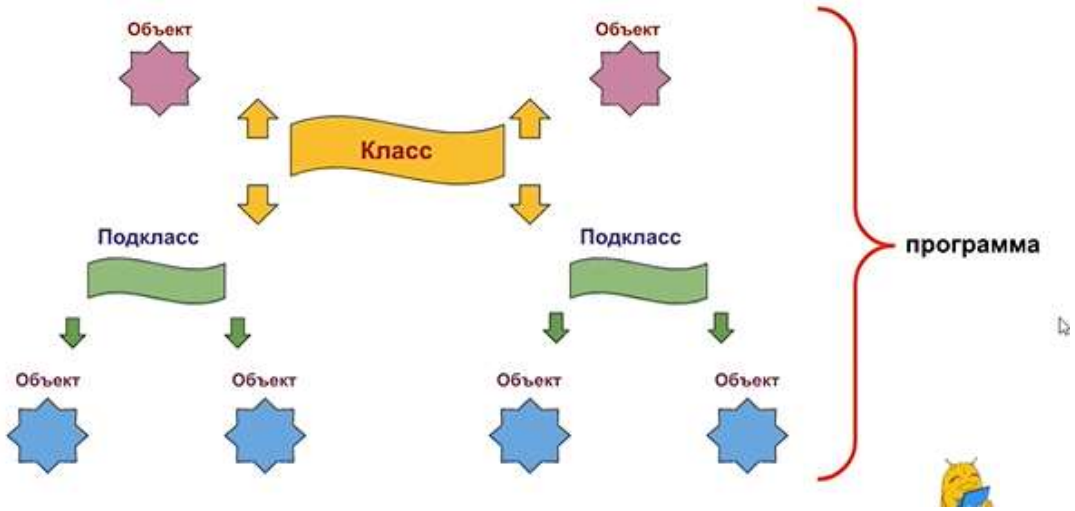
Задания

1. Изучите теоретический материал, выполняя задания внутри него
2. Изучите дополнительный теоретический материал в файле «ООП».
3. Зайдите в электронный университет и выполните тестирование по темам графика и объектно – ориентированное программирование.

Объектно - ориентированное программирование – способ программирования, использующий классы и объекты.

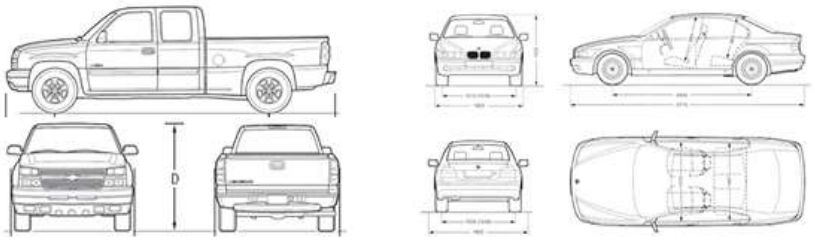
Объектно-ориентированное программирование (ООП) - это способ (методология) программирования, в котором программа представляет из себя совокупность объектов, объекты в свою очередь являются экземплярами классов, а классы образуют из себя систему наследования из классов и подклассов. Сейчас этот способ программирования самый популярный в мире и под него заточены множество языков программирования, в том числе и наш Python. Как же он выглядит? Ну, например у нас есть класс, из него

могут получаться объекты, а могут и подклассы из которых в свою очередь тоже будут получаться объекты и как-то взаимодействовать между собой, образуя таким образом программу. Т.е. проще говоря ООП - это создание программ с помощью классов и объектов. Ну и давайте остановимся на классах и объектах поподробнее, начнём с классов.



Класс – чертеж, по которому создаются объекты.

Звучит не очень официально, но по сути так оно и есть. Представьте себе автомобиль. Прежде чем создавать автомобиль, который будет ездить, сигналить и выполнять ещё множество функций, нам потребуется создать его чертёж. Ведь автомобиль очень сложный механизм. И только потом уже, на заводе его соберут в полноценный автомобиль, согласно нашему чертежу. Причём соберут не один, а сразу много штук, с разным цветом и некоторыми характеристиками. Также и в программировании. Нам в программе нужны объекты, которые будут взаимодействовать между собой. Но сами эти объекты внутри себя очень сложны, у них есть аргументы, различные методы и т.д. Поэтому сначала для каждого объекта пишется класс. Он может храниться в отдельных файлах (модулях). И когда мы с вами создаём объект, например холст на прошлом уроке, мы обращаемся к его классу и вписываем нужные аргументы.



Объект – готовый экземпляр класса, который имеет доступ ко всем его методам.

Ну а теперь, когда класс написан, можно начинать создавать по нему объекты, сколько угодно штук, а в программе уже они будут взаимодействовать между собой. Объект - это готовый экземпляр класса, который имеет доступ ко всем его аргументам и методам. Соответственно, как собранный по чертежу автомобиль. Ну или как canvas на прошлом уроке. Переменная canvas – это переменная, которую мы создавали по классу Canvas, передав три аргумента в числе которых ширина и высота. И к этому объекту

этому объекту мы уже вызывали его разные методы, например move или create_rectangle. Но класс Canvas написали создатели языка пайтон, когда делали модуль tkinter. А мы же сегодня научимся создавать свои классы, сделаем по ним объекты и узнаем, как они могут наследоваться между собой.

Итак, разбор таких понятий как класс и объекты мы с вами начали с автомобилей, поэтому давайте на их примере и продолжим. Создадим класс Car(), на основе которого будем создавать автомобили. Итак, при создании класса в первую очередь пишется ключевое слово class. Соответственно, оно сообщает языку Python, что будет создан класс. Далее пишется название класса. И вот здесь остановимся поподробнее. Дело в том, что классы в пайтон нужно по-особенному называть, так договорились программисты. Всегда с большой буквы и если несколько слов, то слитно, каждое слово с большой буквы. Таким образом, вы всегда отличите класс от функции или переменной. Наш класс называется Car, т.е. "автомобиль". Далее ставим круглые скобки, внутри которых указывается родитель класса. О них мы поговорим позже, единственное, что стоит отметить, что если у класса нет родителя, то круглые скобки не обязательны. Т.е. мы можем встретить написание класса без них. Ну мы все равно их будем ставить.

Далее в каждом классе принято указывать, для чего он написан. Т.е. что это за чертёж, что он создаёт. Пишется описание с помощью комментария, вы с ними уже знакомы. Давайте напишем комментарий, для этого вы должны поставить 3 кавычки в начале и конце строки. И в будущем, когда мы будем создавать экземпляр объекта и открывать круглые скобки, этот комментарий нам выведется и мы сможем его прочитать.

Далее в каждом классе требуется обязательно указывать его метод ___init___ (вокруг init ставится по два нижних подчеркивания).

Метод - это функция внутри класса.

Функции классов всегда называются методами, так что не путайтесь, по сути метод - это функция.

Метод __init__ используется непосредственно при создании объекта, он инициализирует его создание. Можете представить этот метод в виде завода, который создаёт автомобиль. Т.е. без него мы не сможем создать объект. Также обратите внимание на странный способ написания названия метода. 2 нижних подчёркивания спереди и сзади. Ну а далее в этом методе указываются аргументы которые будут присущи каждому объекту класса. Какие же аргументы возьмем для нашего класса Car?

```
class Car():
    """Общий класс для автомобилей"""
    def __init__
```

обязательный метод класса, вызывается один раз, при создании объекта

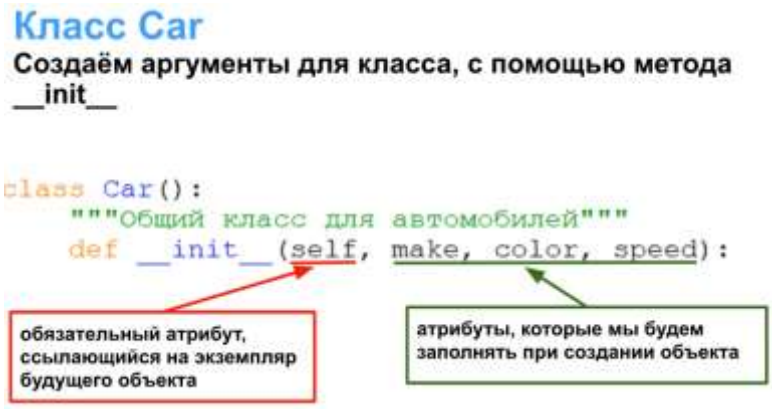
Мы конечно можем указать целую сотню разных аргументов, объем двигателя, радиус покрышек, вид топлива и т.д., но остановимся на трёх самых основных. Марка, цвет и максимальная скорость. Получается, при создании объекта по чертежу нашего класса Car, мы сможем ему указать эти 3 аргумента. Итак, давайте впишем их в метод _init _.

Класс Car

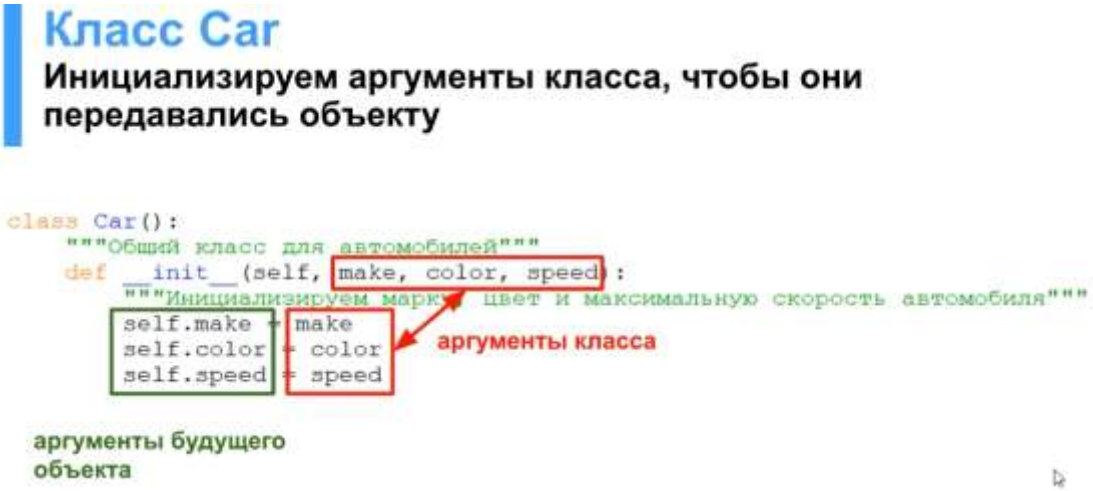
Основные характеристики объекта автомобиль

Марка: **BMW**
Цвет: **Чёрный**
Скорость: **250км/ч**

Однако не торопитесь писать аргументы нашего автомобиля, ведь в аргументах метода __init__, да и вообще всех остальных методах класса, должно обязательно стоять слово self - ("собственный") это обязательное ключевое слово, указываемое в классе и ссылающееся на конкретный будущий объект. Должно стоять первым в списке аргументов класса, а также в аргументах всех его методов. Затем, через запятую, мы указываем те самые аргументы нашего класса Car, которые мы только что придумали, make - марка, color - цвет, speed - максимальная скорость. После чего закрываем скобку и ставим двоеточие.



Метод __init__ инициализирует классовые аргументы объекту, т.е. он их привязывает к нему. В теории, когда мы создаём объект класса, он изначально пустой и не имеет никаких аргументов, если метод __init__ их ему не присвоит. И, как раз здесь, мы с вами и пишем инструкцию для присваивания аргументов будущему объекту. Сначала пишем self.make - это аргумент make будущего Объекта. Затем пишем равно и аргумент класса make, который у нас уже создан выше. У них одинаковые названия, это может вас запутать. Но слово self перед аргументом говорит, что это аргумент будущего объекта. А аргументы без self - это аргументы класса. Таким образом, при создании объекта однократно срабатывает метод __init__, который привязывает аргументы класса к объекту и собирает таким образом объект. Помните, я сравнил метод __init__ с заводом, который собирает автомобили. Вот он так и собирает их, присваивает объекту аргументы.



Но как нам проверить, какой объект у нас создался, применились ли к нему аргументы? Для этого в классах принято указывать метод description ("описание"), который печатает описание созданного объекта, если мы запросим. Напишите его ниже метода инит.

Класс Car

Добавим в наш класс метод description - описание

```
class Car():
    """Общий класс для автомобилей"""
    def __init__(self, make, color, speed):
        """Инициализируем марку, цвет и максимальную скорость автомобиля"""
        self.make = make
        self.color = color
        self.speed = speed

    def description(self):
        """Печатает описание автомобиля"""
        print("Марка автомобиля:", self.make, "\nЦвет:", self.color, "\nМакс. скорость:", self.speed)
```

Итак, мы с вами создали метод description, а обратили внимание на его аргумент? Аргумент self означает, что данный метод вызывается непосредственно к объекту, а self.make, self.color и self.speed пишутся через self, т.н. это именно аргументы объекта, которые мы укажем, а не аргументы класса.

Метод класса

Рассмотрим наш метод description внимательнее

```
def description(self):
    """Печатает описание автомобиля"""
    print("Марка автомобиля:", self.make, "\nЦвет:", self.color, "\nМакс. скорость:", self.speed
```

Итак, чертёж мы создали, теперь давайте запустим shell, Run -> Run module и попробуем создать объект.

Давайте создадим объект под названием car1. Это будет объект_класса Car(). Напишите Car и откройте скобку. И вы увидите подсказки в виде аргументов и того комментария, который мы оставили. Это очень удобно, обратите внимание в будущем, когда будете использовать готовые методы. Ну и какие же аргументы мы укажем? Создадим например вот такой автомобиль, это Toyota Camry 2020 года, но мы укажем только название марки, цвет и максимальную скорость.

Объект

Начинаем создавать объект класса, Python нам подсказывает аргументы и выводит комментарий

```
========== RESTART: C:\Users\APRT\Desktop\lesson7.py ==========
>>> car1 = Car(
          (make, color, speed)
          Общий класс для автомобилей
```

Марка: **Toyota**
Цвет: **Серый**
Скорость: **214 км/ч**

Итак, указываем и жмём на enter. Значит для того, чтобы создать объект, мы пишем его название, т.е. по сути создаём переменную, затем пишем равно и вызываем метод __init__ для создания экземпляра класса. Вызывается он просто по названию класса. Указываем аргументы и готово! Наш объект создан и сохранён в переменную carl. Давайте проверим!

Создание объекта

Указываем марку, цвет и максимальную скорость и жмём на enter

```
>>> car1 = Car("Toyota", "Grey", 214)
>>>
```

Объект car1

Название класса, по которому создаём объект

Аргументы класса

Обращаемся к объекту car1 и вызываем метод description(). И нам выводится краткое описание нашего объекта. Работает?

Объект создан

Вызываем метод description для объекта car1

```
>>> car1.description()
Марка автомобиля: Toyota
Цвет: Grey
Макс. скорость: 214
```

Но наш объект существует только в текущей сессии, если мы перезагрузим python shell, он исчезнет. Поэтому давайте создадим объект car1 непосредственно в файле.

Итак, обязательно создаём объект под классом, т.к. код в пайтон выполняется сверху вниз и если мы захотим создать объект класса Car до создания этого самого класса, то возникнет ошибка. Создайте объект BMW чёрного цвета с максимальной скоростью 250. Это BMW X5 2020 года. И вызываем сразу метод description. Тестируем, запускаем shell.

Создадим объект через IDLE

Закройте Shell и создайте объект car1 ниже класса, вызовите к нему метод description()

```
class Car():
    """Общий класс для автомобилей"""
    def __init__(self, make, color, speed):
        """Инициализируем марку, цвет и максимальную скорость автомобиля"""
        self.make = make
        self.color = color
        self.speed = speed

    def description(self):
        """Печатает описание автомобиля"""
        print("Марка автомобиля:", self.make, "\nЦвет:", self.color, "\nМакс. скорость:", self.spe

car1 = Car("BMW", "Чёрный", 250)
car1.description()
```

И в итоге, мы с вами создали объект car1 на основе класса Car() и воспользовались его методом description.

Тестируем

Запускаем Python Shell и видим описание автомобиля

```
========== RESTART: C:\Users\APRT\Desktop\lesson7.py ==========
Марка автомобиля: BMW
Цвет: Чёрный
Макс. скорость: 250
```

Самостоятельное задание

Добавьте метод ride() к классу Car. Примените метод к нашему объекту car1. Пример вывода:

```
Марка автомобиля: BMW
Цвет: Чёрный
Макс. скорость: 250
BMW Чёрный проехала 3 километра.
```

Подсказка:

```
"""Проехать на автомобиле 3 километра"""
print(self.make, self.color, "проехала 3 километра.")
```

Создайте еще один метод parking() в классе Car. Примените его к объекту car1. Пример:

```
Марка автомобиля: BMW
Цвет: Чёрный
Макс. скорость: 250
BMW Чёрный проехала 3 километра.
BMW Чёрный припаркован.
```

Создайте еще один объект класса Car и вызовите все методы класса Car. Пример:

Марка: **Hyundai**
Цвет: **Белый**
Скорость: **190 км/ч**

Теперь тестируем наш метод. У всех работает? Отлично, молодцы, вот так мы с вами и познакомились с классами и объектами. Это очень важно для понимания объектно-ориентированного программирования. А что ещё важно наследование классов, т.е. классы и подклассы.

Итак, для удобства давайте в файле разграничим зону, где мы пишем классы и зону, где мы создаём объекты. Для разграничения ставим однострочный комментарий с помощью символа решётка. Ранее мы с вами использовали многострочные комментарии. Затем пишем много знаков минус и где-нибудь в середине напишем MAIN, т.е. главная программа. Получается, что класс - это подготовительная часть, чертежи, а вот MAIN - это уже непосредственно сама программа, которая что-то выполняет на основе этих чертежей.

Наследование классов

Комментарием разграничим “зону” классов и главной программы

```
class Car():
    """Общий класс для автомобилей"""
    def __init__(self, make, color, speed):
        """Инициализируем марку, цвет и максимальную скорость автомобиля"""
        self.make = make
        self.color = color
        self.speed = speed

    def description(self):
        """Печатает описание автомобиля"""
        print("Марка автомобиля:", self.make, "\nЦвет:", self.color, "\nМакс. скорость:", self.speed)

    def ride(self):
        """Проехать на автомобиле 3 километра"""
        print(self.make, self.color, "проехала 3 километра.")

#-----------------------------------MAIN-----------------------------------------

car1 = Car("BMW", "черный", 250)
car1.description()
car1.ride()
```

А теперь давайте познакомимся с наследованием классов. Это очень важный принцип ООП. Мы с вами создали сейчас класс автомобиля. А теперь представим, что у нас с вами есть бензиновый автомобиль и электроавтомобиль. Это автомобили? Да, они оба автомобили. Но все-таки чем-то они отличаются, кто скажет, чем? Внутренним устройством и топливом, да. Хорошо, а теперь давайте рассмотрим на примере классов.

Наследование классов

Чем бензиновый автомобиль отличается от электроавтомобиля?

Взгляните на слайд. Итак, у нас с вами сейчас есть класс Car. Это просто чертёж автомобиля.

Если создаём по нему объект, то что он умеет, какие у него, методы? "Описание" и "Проехать 3 км", да.

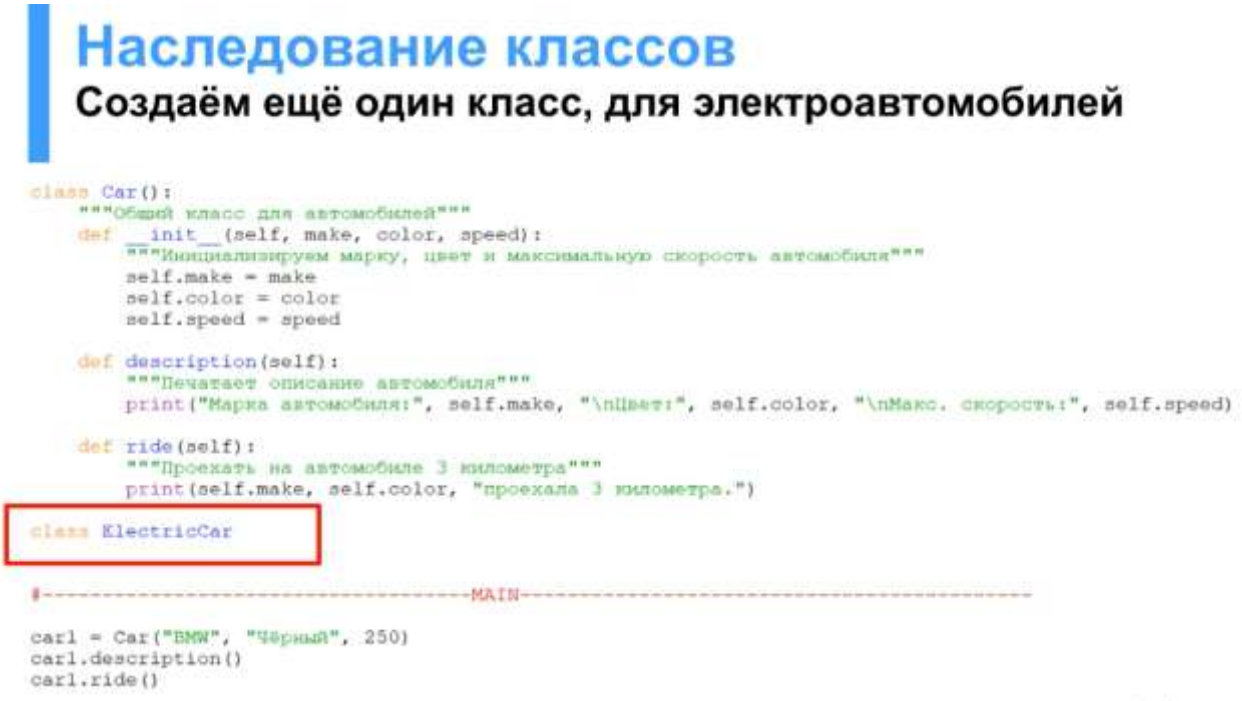
Умеет-ли в обычном мире это делать бензиновый автомобиль? Да, конечно. А электрический? Тоже да.

Значит, при создании их классов, мы с вами можем их наследовать от основного класса Car, чтобы заново не писать эти методы.

Наследование классов - это когда один класс наследует аргументы и методы другого класса (родителя). И добавляет что- то своё, если нужно. Получается, для бензинового автомобиля мы можем создать класс GasolineCar ("бензиновый автомобиль"), а для электроавтомобиля - ElectricCar. Указать в скобках родителя Car и тогда они станут его потомками, т.е. им будут доступны методы descriptionQ и ride(). Это очень удобно,

нам не придётся эти методы определять ещё раз. А что-же у них тогда будет своё, чем они отличаются от класса Car? Способом заправки конечно, у класса GasolineCar метод fill, т.е. "залить". А у ElectricCar метод charge_car, т.е. зарядить автомобиль. Итого, у нас получается 2 брата, отличающиеся друг от друга - ElectricCar и GasolineCar, и их отец - Car. Давайте-же создадим с вами класс ElectricCar, чтобы попрактиковаться в наследовании классов.

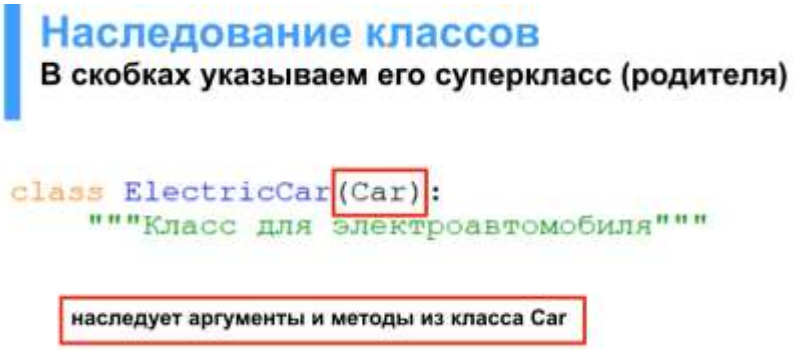
После класса Car делаем отступ в 2 строки и создаём класс ElectricCar, который в свою очередь будет создавать экземпляр именно электроавтомобиля. И у него также будет марка, цвет и максимальная скорость, как и у объекта класса Car. Плюс, он абсолютно также может проехать 3 километра и также может распечатать своё описание.



Ну а теперь нам нужно указать родителя класса. Кто он? Как указываем? Это Car, указываем в скобках.

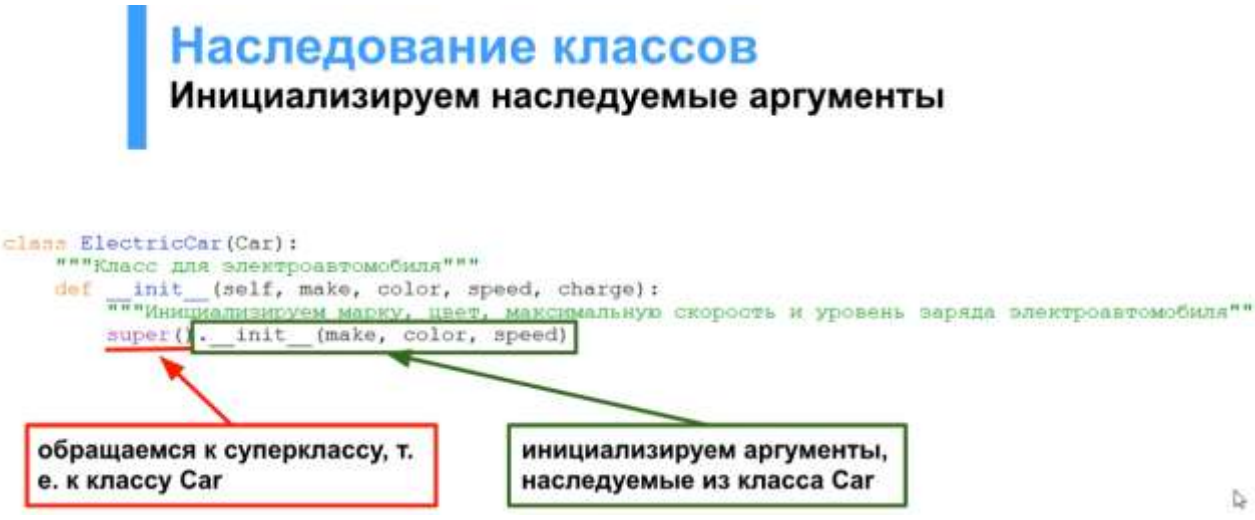
Итак, в скобках указываем ему родительский класс, т.е. Car. Таким образом, класс ElectricCar станет подклассом класса Car. Подкласс - это класс, у которого есть класс-

родитель. Теперь, класс ElectricCar сможет использовать методы descriptionQ и rideQ. А ниже прописываем комментарий для класса.



Ну а теперь нам необходимо вызвать обязательный метод __init__, который будет собирать нам объект класса ElectricCar. В скобках мы должны указать все аргументы, которые хотим применять, даже если они наследуемые. И указываем новый аргумент charge, т.е. заряд, он будет у нас отвечать за уровень заряда. Затем пишем комментарий, можете его скопировать с класса Car.

А теперь, внутри метода инит, мы должны инициализировать все аргументы, даже если они наследуемые. Наследуемые аргументы инициализируются по-особенному, сначала пишется слово superf), обозначающее, что мы обращаемся к суперклассу, т.е. к родителю нашего класса. Затем ставится точка и пишется ещё раз __init__, с нужными аргументами внутри скобок. Таким образом мы вызываем инит из класса Car.



Ну а собственный аргумент класса EletricCar - charge, мы инициализируем как положено, через self. Таким образом, мы написали метод init для нашего подкласса EletricCar.

Наследование классов

Инициализируем аргумент charge, принадлежащий только классу ElectricCar

```
class ElectricCar(Car):
    """Класс для электроавтомобиля"""
    def __init__(self, make, color, speed, charge):
        """Инициализируем марку, цвет, максимальную скорость и уровень заряда электроавтомобиля"""
        super().__init__(make, color, speed)
        self.charge = charge
```

Ну и конечно-же, давайте укажем особенный метод для класса EletricCar, который будет доступен только объектам этого класса. Что-же то за метод? Конечно же зарядка. Наш метод charge_car будет заряжать объект до 100%, т.е. менять значение аргумента charge. А потом выводить отчёт на экран.

Наследование классов

Создаём функцию для зарядки электроавтомобиля

```
class ElectricCar(Car):
    """Класс для электроавтомобиля"""
    def __init__(self, make, color, speed, charge):
        """Инициализируем марку, цвет, максимальную скорость и уровень заряда электроавтомобиля
        super().__init__(make, color, speed)
        self.charge = charge

    def charge_car(self):
        """Зарядить автомобиль до 100%"""
        charge = "100%"
        print(self.make, self.color, "- уровень заряда 100%")
```

Самостоятельное задание

Создайте новый объект car2 с классом ElectricCar. Вызовите методы description, ride и charge_car к этому объекту.

Марка: **Tesla**
Цвет: **Белый**
Скорость: **250 км/ч**
Заряд: **39%**

Создайте еще один класс GasolineCar, наследуйте его от Car, добавьте внего аргумент gas (“бензин”). Создайте в этом классе метод fill, который заправляет автомобиль и делает его уровень топлива 100%.

Создайте car3 с классом GasolineCar. Вызовите все классовые методы доступные этому объекту.

Теперь тестируем. Итого, мы с вами создали 2 объекта, один по классу Car, другой по классу ElectriCar. И вызвали к ним методы. Очень хорошо. Также мы могли бы начать сравнивать эти объекты между собой и основывать нашу программу на них, например, кто быстрее и т.д. Так и работает ООП.

А теперь давайте подведём итоги сегодняшнего занятия.

1. Итак, сегодня мы с вами познакомились с объектно - ориентированным программированием. Что это такое? Способ программирования, основанный на использовании классов и объектов.
2. Что такое класс? Это чертёж, по которому создаются объекты, в котором указаны разные методы.
3. Что такое объект? Экземпляр, созданный на основе класса.
4. Далее мы с вами научились создавать классы. Какое ключевое слово отвечает за создание класса? class.
5. А как нужно называть классы? С большой буквы, слитно, каждое следующее слово тоже с большой.
6. А какой метод должен быть обязательно у каждого класса? __init__.
7. Что он делает? Создаёт объект класса, инициализирует его аргументы.
8. А что делает ключевое слово self? Ссылается на будущий объект.
9. А как именно мы создавали новый объект? Писали название переменной, в которой объект будет находиться и вызывали инициализатор класса по его названию, вписывая в скобках нужные аргументы.
10. А ещё мы узнали, что классы могут наследоваться друг от друга. Чтобы создать подкласс какого-нибудь класса, что мы должны сделать? Указать родителя в скобках.
11. Как инициализируются аргументы родителя? Через ключевое слово super(), обозначающее супер класс, т.е. родителя.

Классы и объекты

Объектно-ориентированное программирование (ООП) – один из подходов к реализации программного кода для проецирования сущностей реального мира. Считается, что введение классов и объектов упрощает понимание кода человеком. В Питоне все является объектами.

Для решения заданий потребуется усвоение следующих тем:

1. Принципы ООП и их реализация в Python;
2. Свойства и методы классов;
3. Инициализация экземпляров классов;
4. Получение и задание свойств через декораторы;
5. Классовые и статические методы;
6. Магические (dunder или magic) методы классов;
7. Модификаторы доступа к методам и свойствам;

8. Дескрипторы.

1. Как связаны классы и объекты?

Классы – шаблоны (**blueprint**) конкретных объектов, т.е. на их основе создаются экземпляры, наследующие свойства и методы родителей.

ООП в языке Python базируется на следующей иерархии:

1. Имеется головной класс **object()**, являющийся основой для всех других (обычно, его явно не указывают);

2. Уровнем ниже расположены метаклассы , классы и подклассы (как самого Питона, так и пользовательские);

3. В результате получаем возможность создавать любое количество экземпляров классов, т.е. объектов.

А вообще говоря - все в Питоне является объектом (даже класс). Это просто нужно запомнить.

2. Для чего необходимо ключевое слово self в классах?

Так как на основе классов создаются конкретные объекты, необходима возможность получения доступа к каждому из них. Ключевое слово self обозначает текущий объект класса. Это некая договоренность (так как self никто не запрещает заменить на любое другое слово).

Слово self применяется в следующий случаях:

1. В качестве первого аргумента у методов экземпляра класса;
2. Для доступа к свойству объекта внутри класса.

Пример – IDE

```
class Hello:
____# self - указание на экземпляр класса
____def __init__(self, name):
_________# Свойство объекта
_________self.name = name
_________print(f'Привет, {self.name}')
greet_me = Hello('Дмитрий')
```

Результат выполнения

Привет, Дмитрий

3. Как создаются и для чего нужны статические методы?

Методы классов в Питоне делятся на 3 типа:

1. Методы экземпляров (наиболее часто используемые, в качестве первого аргумента всегда принимают self);
2. Классовые методы (здесь первым параметром передается cls. Привязаны к данному классу, а не к его экземплярам. Способны менять состояние класса, но не его экземпляров);
3. Статические методы (не требуют наличия особого первого аргумента. Фактически, не принадлежат никакому классу, а представляют собой независимую функцию, которую мы решили по причинам бизнес-логики включить в класс).

Для создания статического метода используют декоратор @staticmethod. Его можно вызывать как от имени класса, так и экземпляра. Главная причина использования – инкапсуляция (изоляция некоторой логики внутри класса). Также, код становится более читабельным и удобным при импорте (не нужно импортировать множество отдельных функций). Статические методы можно вызывать как от имени класса, так и объекта.

Пример – IDE

```
class Person:
____def __init__(self, name):
_________self.name = name
____@staticmethod
____def status(year_of_birth):
_________if 2021 - year_of_birth >= 18:
_____________print('Вы можете смотреть все страницы сайта')
 _________else:
_____________print('Часть страниц вам не доступна')
student = Person('Петр')
 # Тесты
student.status(1991)
Person.status(2011)
```

Результат выполнения

```
Вы можете смотреть все страницы сайта
Часть страниц вам не доступна
```

Фактические, метод status() никто не мешает нам вынести за рамки класса, но мы его тут разместили для изоляции и удобства.

4. Как реализуется наследование классов в Python?

Наследование – один из основных принципов ООП. Его суть выражается в следующем: на основании одних классов (базовых, суперклассов) можно создавать другие (подклассы), наследующие их свойства и методы.

Такой подход существенно снижает дублирование кода:

1. Нет нужды переписывать одни и те же методы и свойства у разных объектов;
2. Методы и свойства наследников не запрещено дополнять или модифицировать.

Как уже упоминалось выше, принцип наследования упрощает представление объектов как сущностей в реальном мире. Другими словами, человеческое сознание работает сходным образом: окружающие нас вещи мы легко может выстроить иерархически. Например, воробей относится к птицам, а птицы являются представителями животного царства. Аналогичным образом поступают и при создании классов: сначала определяют общий, а от него создаются частные, со своими особенностями.

Пример – IDE

```
class Bird:
____def __init__(self, age, fly_distance):
_________self.age = age
```

```
        self.fly_distance = fly_distance
    def fly(self):
        print(f"Птица может пролететь за раз километров: {self.fly_distance}")
    def human_age(self):
        print(f'Этой птице {self.age * 6} человеческих лет')
class Sparrow(Bird):
    def __init__(self, age, fly_distance, sound):
        super().__init__(age, fly_distance)
        self.sound = sound
    def human_age(self):
        print(f'Данному воробью {self.age * 25} человеческих лет')
    def sing(self):
        print(self.sound)
# Тесты
crow = Bird(11, 5)
crow.fly()
crow.human_age()
young_sparrow = Sparrow(1, 2, 'чик-чирик')
old_sparrow = Sparrow(3, 1, 'чирик-чирик')
 young_sparrow.fly()
young_sparrow.sing()
young_sparrow.human_age()
old_sparrow.human_age()
```

Результат выполнения

```
---
Птица пролетела километров: 5
Этой птице 66 человеческих лет
Птица пролетела километров: 2
чик-чирик
Данному воробью 25 человеческих лет
Данному воробью 75 человеческих лет
```

Функция super() позволяет ссылаться на родительский суперкласс. Класс Sparrow унаследовал от Bird метод fly(), затем дополнительно мы ему создали собственный метод sound() и изменили метод предка human_age().

5*. Что такое дескрипторы данных?

Очень часто переменные, инициализируемые в классе, являются однотипными. Например, есть класс Employee (сотрудник), принимающий параметры: имя, фамилия, отчество, должность. Все они являются строками. Следовательно, прежде чем создать экземпляр класса, нужно проверить, что пользователь ввел строки. А для этого потребуются сеттеры, проверяющие тип вводимых параметров. В итоге, мы 4 раза повторим код проверки. Нарушается принцип DRY (don't repeat yourself).

Для таких ситуаций удобно использовать дескрипторы (они, к слову, широко применяются во фреймворке Django при создании моделей).

Дескриптор - такой атрибут объекта, поведение которого переопределяется специальными методами. Проще говоря, доступ к какому-то свойству экземпляра можно

переопределить с учетом дополнительных проверок. Если делать эти верификации без дескрипторов, то один и тот же код начнет повторяться.

Существует 4 метода протокола дескрипторов:

1. __get__() - получить значение свойства;
2. __set__() - задать значение;
3. __delete__() - удалить атрибут;
4. __set_name__() - присвоить имя свойству (появился в Питоне версии 3.6).

Если применяется только метод __get__(), то мы имеем дело с дескриптором без данных, а если есть еще и __set__(), то речь будет идти о дескрипторе данных.

Покажем использование дескрипторов на вышеупомянутом примере.

Пример – IDE

```
# Создаем класс с протоколами дескриптора
class StringChecker:
    # Получаем доступ к свойству
    def __get__(self, instance, owner):
        if instance is None:
            return self
        return instance.__dict__[self.name]
    # Меняем свойство
    def __set__(self, instance, str_value):
        if not isinstance(str_value, str):
            raise ValueError('Нужно предоставить строку')
        elif len(str_value) < 2:
            raise ValueError('Необходимо минимум 2 буквы')
        instance.__dict__[self.name] = str_value
    # Задаем имя свойства
    def __set_name__(self, owner, name):
        self.name = name
class Employee:
    # Определяем атрибуты (их может быть любое количество)
    name = StringChecker()
    surname = StringChecker()
    patronymic = StringChecker()
    post = StringChecker()
    # Инициализируем свойства с учетом требуемых проверок
    def __init__(self, name, surname, patronymic, post):
        self.name = name
        self.surname = surname
        self.patronymic = patronymic
        self.post = post
# Тесты
director = Employee('Иван', 'Николаевич', 'Прогин', 'Директор')
print(director.__dict__)
director.name = 1
```

director.name = 'A'

Результат выполнения

{'name': 'Иван', 'surname': 'Николаевич', 'patronymic': 'Прогин', 'post': 'Директор'}

ValueError: Нужно предоставить строку

ValueError: Минимум две буквы в атрибуте требуется

Задание по теме «Классы и объекты»

Задача 1. Создание классов и объектов в Python. Конструктор класса

Класс содержит имя студента full_name, номер группы group_number и список полученных оценок progress. В программе вводится список студентов. Далее список сортируется по имени, потом выводятся студенты, имеющие неудовлетворительные оценки.

```
class Student:
    def __init__(self,full_name="", group_number=0, progress=[]): # конструктор
        self.full_name = full_name  # имя

        self.group_number = group_number # номер группы
        self.progress = progress   # оценки
    def __str__(self):   # печатаемое представление экземпляра класса
        txt = 'Студент: ' + self.full_name + ' Группа: ' + self.group_number
        txt += ' Оценки:'
        for x in self.progress:
            txt += ' ' + str(x) # добавляем список оценок
        return txt
#--------------------------------------------------------------------------------
def SortParam(st): # функция определяющая атрибут для сортировки
    return st.full_name
#--------------------------------------------------------------------------------

st_size = 5  # количество студенов

students = [] # создание пустого списка
for i in range(st_size): # цикл для ввода  st_size студентов
    print("Введите полное имя студента: ")
    full_name = input() # ввод фамилии
    print("Введите номер группы: ")
    group_number = input() # ввод группы
    n=5
    print('Введите ',n,' оценок в столбик: ') # у каждого студента n оценок
    progress = []
    for i in range(n):
        score = int(input()) # ввод оценок
        progress.append(score) # добавление оценок
    # создание экзепляра класса Student:
    st = Student(full_name, group_number, progress)
    students.append(st) # добавление экземпляра в список

print("Students list:")
for st in students: # вывод полного списка студентов
    print(st)
```

```
# сортировка по фамилии, ключ сортировки определяется функцией SortParam:
students = sorted(students, key=SortParam)

print("Sorted students:")
for st in students: #  вывод отсортированного списка
    print(st)

print("bad students:")
n=0 # счетчик количества неуспевающих
for st in students: #  вывод неуспевающих
   for val in st.progress:
      if val<3 : # есть плохая оценка
         print(st) # выводим студента с плохой оцекой
         n += 1
         break
if n == 0:
    print("no matches were found.")
```

Задача 2. Наследование. Множественное наследование

Класс ForeignPassport является производным от класса Passport. Метод PrintInfo существует в обоих классах. PassportList представляет собой список, содержащий объекты обоих классов.

```
class Passport():
    def __init__(self, first_name, last_name, country, date_of_birth,
numb_of_pasport):
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.country = country
        self.numb_of_pasport = numb_of_pasport

    def PrintInfo(self):
       print("\nFullname: ",self.first_name, " ",self.last_name)
       print("Date of birth: ",self.date_of_birth)
       print("County: ",self.country)
       print("Passport number: ",self.numb_of_pasport)

class ForeignPassport(Passport):
    def __init__(self, first_name, last_name, country, date_of_birth,
numb_of_pasport,visa):
        super().__init__(first_name, last_name, country, date_of_birth,
numb_of_pasport)
        self.visa = visa

    def PrintInfo(self):
        super().PrintInfo()
        print("Visa: ",self.visa)
```

```
PassportList=[]
request = ForeignPassport('Ivan', 'Ivanov', 'Russia', '12.03.1967', '123456789',
'USA')
PassportList.append(request)

request = Passport('Иван', 'Иванов', 'Россия', '12.03.1967', '45001432')
PassportList.append(request)

request = ForeignPassport('Peter', 'Smit', 'USA', '01.03.1990', '21435688',
'Germany')
PassportList.append(request)

for emp in PassportList:
   emp.PrintInfo()
```

Задача 3. Полиморфизм в Python

Классы Printer, Scaner и Xerox являются производными от класса Equipment. Метод str() перегружен только в классе Printer, для остальных используется метод из базового класса. Метод action() перегружен для всех производных классов. Вызов этих методов для каждого элемента списка демонстрирует их полиморфное поведение.

```
class Equipment:
    def __init__(self, name, make, year):
        self.name = name # производитель
        self.make = make # модель
        self.year = year # год выпуска
    def action(self):
        return 'Не определено'
    def __str__(self):
        return f'{self.name} {self.make} {self.year}'
#-----------------------------------------------
class Printer(Equipment):
    def __init__(self, series, name, make, year):
        super().__init__(name, make, year)
        self.series = series # серия
    def __str__(self):
        return f'{self.name} {self.series} {self.make} {self.year}'
    def action(self):
        return 'Печатает'

#-----------------------------------------------
class Scaner(Equipment):
    def __init__(self, name, make, year):
        super().__init__(name, make, year)
    def action(self):
        return 'Сканирует'
#-----------------------------------------------
class Xerox(Equipment):
    def __init__(self, name, make, year):
        super().__init__(name, make, year)
    def action(self):
        return 'Копирует'
#-----------------------------------------------
```

```
sklad = []
# создаем объект сканер и добавляем
scaner = Scaner('Mustek','BearPow 1200CU', 2010)
sklad.append(scaner)
# создаем объект ксерокс и добавляем
xerox = Xerox('Xerox','Phaser 3120', 2019)
sklad.append(xerox)
# создаем объект принтер и добавляем
printer = Printer("1200",'hp', 'Laser Jet', 2018)
sklad.append(printer)
# выводим склад
print("На складе имеются:")
for x in sklad:
   print(x,end=' ')
   print(x.action())
# забираем со склада все принтеры
for x in sklad:
   if isinstance(x,Printer):
      sklad.remove(x)
# выводим склад
print("\nНа складе осталось:")
for x in sklad:
   print(x,end=' ')
   print(x.action())
```

Задача 4. Примеры композиции классов

В классе Battle реализована композиция: он включает два объекта типа Soldier.

```
from random import randint
class Soldier: # класс описывающий одного солдата
    def __init__(self,name='Noname',health = 100): # конструктор
       self.name = name     # задаем имя воина
       self.health = health # задаем начальное здоровье
    def set_name(self, name):
       self.name = name     # есть возможность поменять имя
    def make_kick(self, enemy): # метод моделирующий атаку на солдата enemy
       enemy.health -= 20  # при атаке здоровье врага уменьшаем на 20
       if enemy.health<0 :
          enemy.health = 0
       self.health += 10   # а собственное здоровье увеличиваем на 10
       print(self.name, "бьет", enemy.name) # выводим кто кого бьет
       print('%s = %d' % (enemy.name, enemy.health)) # выводим состояние врага
#-------------------------------------------------------------------------------
```

```
class Battle:
    def __init__(self,u1,u2):   # конструктор
    # композиция: класс включает двух солдат u1 и u2
        self.u1 = u1
        self.u2 = u2
        self.result = "Сражения не было" # строка для хранения состояния сражени
    def battle(self): # метод моделирующий сражение
        while self.u1.health > 0 and self.u2.health > 0:
            n = randint(1, 2) # определяем, кто атакует
            if n == 1:
                self.u1.make_kick(self.u2) # если атакует первый
            else:
                self.u2.make_kick(self.u1) # если атакует второй
        if self.u1.health > self.u2.health:# определяем, кто победил
-------------------------------------------------------------------------------
            self.result = self.u1.name + " ПОБЕДИЛ"
        elif self.u2.health > self.u1.health:
            self.result = self.u2.name + " ПОБЕДИЛ"
    def who_win(self): # вывод результата
        print(self.result)
#-------------------------------------------------------------------------------
first = Soldier('Mr. First',140) # созаем 1 солдата с именем Mr. First и здоровьем
140
second = Soldier()  # созаем 2 солдата с паметрами по умолчанию
second.set_name('Mr. Second') # меняем имя 2 солдата
b = Battle(first,second) # создаем обьект Battle
b.battle()  # запускаем сражение
b.who_win()  # выводим итог
```

Задача 5. Создание и использование абстрактного метода

Пример класса с абстрактным методом.

```
import abc
class Class1(abc.ABC):
    def __init__(self, val):
        self.x = val
    @abc.abstractmethod # Абстрактный метод
    def func(self):
        raise NotImplementedError("Нельзя вызывать абстрактный метод")

class Class2(Class1): # Наследуем абстрактный метод
    def another_func(self): # Определяем другой метод
        print(-self.x)

class Class3(Class2):  # Наследуем два метода
    def func(self):  #  Переопределяем абстрактный метод метод
        print(self.x)

try: # Перехватываем исключения
    c = Class1(10) # Ошибка.  Метод func()  не переопределен
except TypeError as msg:
    print(msg) # вывод: Can't instantiate abstract class Class1 with abstract …

try: # Перехватываем исключения
    c = Class2(10) # Ошибка.  Метод func()  не переопределен
except TypeError as msg:
    print(msg) # вывод: Can't instantiate abstract class Class1 with abstract …

c = Class3(30)
c.func() # вывод: 30
c.another_func() # вывод: -30
```

Задание 6. Создание и использование статического метода

Класс, представляющий рациональную дробь (num – числитель, den – знаменатель). Класс содержит конструктор и перегруженные методы умножения и деления (дроби на дробь и дроби на целое число). Метод создания случайной дроби из заданного диапазона целых чисел объявлен как статический.

```
from math import gcd
from random import randint

class My_Fraction:
    def __init__(self, num, den):
        if num != 0 and den != 0:
            k = gcd(num, den) # находим НОД
            self.num = num // k  # числитель
            self.den = den // k  # знасенатель
        else:
            raise ValueError

    @staticmethod
    def generate(num_min, num_max, den_min, den_max):
        return My_Fraction(randint(num_min, num_max), randint(den_min, den_max)
```

```
    def __str__(self):           # Метод преобразования дроби в строку
        return f'{self.num}/{self.den}'

    def __mul__(self, other):    # Умножение дробей
        if isinstance(other,My_Fraction): # перегрузка умножения на дробь
            return My_Fraction(self.num * other.num, self.den * other.den)
        if isinstance(other,int):  # перегрузка умножения на целое число
            return My_Fraction(self.num * other, self.den)
        return self                # для остальных типов возвращаем значение самог
объекта

    def __truediv__(self, other): # Деление дробей
        if isinstance(other,My_Fraction):  # перегрузка деления на дробь
            return My_Fraction(self.num * other.den, self.den * other.num)
        if isinstance(other,int):  # перегрузка деления на целое число
            return My_Fraction(self.num, self.den*other)
        raise TypeError            # для остальных типов вызываем исключение
#-----------------------------------------------------------------------
# Список из 5 случайных дробей:
a = [My_Fraction.generate(1, 9, 1, 9) for i in range(5)]
for f in a:
    b = My_Fraction.generate(1, 9, 1, 9)  # дробь для правого операнда
    cm = f * b
    print(f'{f} * {b} = {cm}') # пример умножения на дробь
    cd = f / b
    print(f'{f} / {b} = {cd}') # пример деления на дробь
    n=randint(1, 9)  # число для правого операнда
    cm = f * n
    print(f'{f} * {n} = {cm}') # пример умножения на число
    cd = f / n
    print(f'{f} / {n} = {cd}') # пример деления на число
```

Задача 7. Создайте класс **Soda** (для определения типа газированной воды), принимающий 1 аргумент при инициализации (отвечающий за добавку к выбираемому лимонаду). В этом классе реализуйте метод **show_my_drink()**, выводящий на печать **Газировка и {ДОБАВКА}** в случае наличия добавки, а иначе отобразится следующая фраза: **Обычная газировка**.

При решении задания можно дополнительно проверить тип передаваемого аргумента: принимается только строка.

Решение - IDE

```
class Soda:
    def __init__(self, ingredient=None):
        if isinstance(ingredient, str):
            self.ingredient = ingredient
        else:
            self.ingredient = None
    def show_my_drink(self):
        if self.ingredient:
            print(f'Газировка и {self.ingredient}')
        else:
            print('Обычная газировка')
```

```
# Тесты
drink1 = Soda()
drink2 = Soda('малина')
drink3 = Soda(5)
drink1.show_my_drink()
drink2.show_my_drink()
drink3.show_my_drink()
```

Результат выполнения

Обычная газировка

Газировка и малина

Обычная газировка

Задача 8. Николаю требуется проверить, возможно ли из представленных отрезков условной длины сформировать треугольник. Для этого он решил создать класс **TriangleChecker**, принимающий только положительные числа. С помощью метода **is_triangle()** возвращаются следующие значения (в зависимости от ситуации):

– **Ура, можно построить треугольник!;**

– **С отрицательными числами ничего не выйдет!**;

– **Нужно вводить только числа!**;

– **Жаль, но из этого треугольник не сделать**.

Построить треугольник из отрезков можно лишь в одном случае: сумма длин двух любых сторон всегда больше третьей.

Решение - IDE

```
class TriangleChecker:
    def __init__(self, sides):
        self.sides = sides
    def is_triangle(self):
        if all(isinstance(side, (int, float)) for side in self.sides):
            if all(side > 0 for side in self.sides):
                sorted_sides = sorted(self.sides)
                if sorted_sides[0] + sorted_sides[1] > sorted_sides[2]:
                    return 'Ура, можно построить треугольник!'
                return 'Жаль, но из этого треугольник не сделать'
            return 'С отрицательными числами ничего не выйдет!'
        return 'Нужно вводить только числа!'
# Тесты
triangle1 = TriangleChecker([2, 3, 4])
print(triangle1.is_triangle())
triangle2 = TriangleChecker([77, 3, 4])
print(triangle2.is_triangle())
triangle3 = TriangleChecker([77, 3, 'Сторона3'])
print(triangle3.is_triangle())
triangle4 = TriangleChecker([77, -3, 4])
print(triangle4.is_triangle())
```

Результат выполнения

Ура, можно построить треугольник!

Жаль, но из этого треугольник не сделать

Нужно вводить только числа!

С отрицательными числами ничего не выйдет!

Задача 9. Евгения создала класс **KgToPounds** с параметром **kg**, куда передается определенное количество килограмм, а с помощью метода **to_pounds()** они переводятся в фунты. Чтобы закрыть доступ к переменной **kg** она реализовала методы **set_kg()** - для задания нового значения килограммов, **get_kg()** - для вывода текущего значения **кг**. Из-за этого возникло неудобство: нам нужно теперь использовать эти 2 метода для задания и вывода значений. Помогите ей переделать класс с использованием функции **property()** и свойств-декораторов. Код приведен ниже.

Пример – IDE

```
class KgToPounds:
    def __init__(self, kg):
        self.__kg = kg
    def to_pounds(self):
        return self.__kg * 2.205
    def set_kg(self, new_kg):
        if isinstance(new_kg, (int, float)):
            self.__kg = new_kg
        else:
            raise ValueError('Килограммы задаются только числами')
    def get_kg(self):
        return self.__kg
```

Чтобы не задавать новые значения или не получать к ним доступ через два метода, можно реализовать предложенный класс через функцию property() или свойства-декораторы.

Вариант решения 1. Функция property()

Решение - IDE

```
class KgToPounds:
    def __init__(self, kg):
        self.__kg = kg
    def to_pounds(self):
        return self.__kg * 2.205
    def __set_kg(self, new_kg):
        if isinstance(new_kg, (int, float)):
            self.__kg = new_kg
        else:
            raise ValueError('Килограммы задаются только числами')
    def __get_kg(self):
        return self.__kg
    kg = property(__get_kg, __set_kg)
```

Вариант решения 2. Свойства-декораторы

Решение - IDE

```
class KgToPounds:
```

```
    def __init__(self, kg):
        self.__kg = kg
    def to_pounds(self):
        return self.__kg * 2.205

    @property
    def kg(self):
        return self.__kg

    @kg.setter
    def kg(self, new_kg):
        if isinstance(new_kg, (int, float)):
            self.__kg = new_kg
        else:
            raise ValueError('Килограммы задаются только числами')
# Тесты
weight = KgToPounds(12)
print(weight.to_pounds())
print(weight.kg)
weight.kg = 41
print(weight.kg)
weight.kg = 'десять'
```

Результат выполнения

```
26.46
12
41
ValueError: Килограммы задаются только числами
```

Задача 10. Николай – оригинальный человек. Он решил создать класс **Nikola**, принимающий при инициализации 2 параметра: имя и возраст. Но на этом он не успокоился. Не важно, какое имя передаст пользователь при создании экземпляра, оно всегда будет содержать **Николая**. В частности - если пользователя на самом деле зовут Николаем, то с именем ничего не произойдет, а если его зовут, например, Максим, то оно преобразуется в **Я не Максим, а Николай**.

Более того, никаких других атрибутов и методов у экземпляра не может быть добавлено, даже если кто-то и вздумает так поступить (т.е. если некий пользователь решит прибавить к экземпляру свойство **отчество** или метод **приветствие**, то ничего у такого хитреца не получится).

Для ограничения количества наборов свойств и методов в экземпляре применяется специальный магический атрибут __slots__.

Решение - IDE

```
class Nikola:
    __slots__ = ['name', 'age']
    def __init__(self, name, age):
        if name == 'Николай':
```

```
            self.name = name
        else:
            self.name = f'Я не {name}, а Николай'
        self.age = age
# Тесты
person1 = Nikola('Иван', 31)
person2 = Nikola('Николай', 14)
print(person1.name)
print(person2.name)
person2.surname = 'Петров'
```

Результат выполнения

```
Я не Иван, а Николай
Николай
AttributeError: 'Nikola' object has no attribute 'surname'
```

Задача 11. Строки в Питоне сравниваются на основании значений символов. Т.е. если мы захотим выяснить, что больше: **Apple** или **Яблоко**, – то **Яблоко** окажется бОльшим. А все потому, что английская буква **A** имеет значение **65** (берется из таблицы кодировки), а русская буква **Я** – **1071** (с помощью функции **ord()** это можно выяснить). Такое положение дел не устроило Анну. Она считает, что строки нужно сравнивать по количеству входящих в них символов.

Для этого девушка создала класс **RealString** и реализовала озвученный инструментарий. Сравнивать между собой можно как объекты класса, так и обычные строки с экземплярами класса **RealString**. К слову, Анне понадобилось только 3 метода внутри класса (включая **__init__()**) для воплощения задуманного.

В общем случае для создания такого класса понадобится 4 метода, так как в Питоне реализованы «богатые» сравнения. Это значит, что если имеется сравнение «больше», то автоматом появится возможность осуществлять сравнение «меньше».

Решение - IDE

```
class RealString:
def __init__(self, some_str):
    self.some_str = str(some_str)
def __eq__(self, other):
    if not isinstance(other, RealString):
        other = RealString(other)
    return len(self.some_str) == len(other.some_str)
def __lt__(self, other):
    if not isinstance(other, RealString):
        other = RealString(other)
    return len(self.some_str) < len(other.some_str)
def __le__(self, other):
    return self == other or self < other
```

Чтобы повторить класс, придуманный Анной (с тремя методами), требуется воспользоваться декоратором @total_ordering из модуля functools (упрощает реализацию сравнений. Требует лишь 2 дополняющих варианта сравнения - например, больше и равно - чтобы автоматически "дописать" остальные).

Решение - IDE

```
from functools import total_ordering
@total_ordering
class RealString:
def __init__(self, some_str):
    self.some_str = str(some_str)
def __eq__(self, other):
    if not isinstance(other, RealString):
        other = RealString(other)
    return len(self.some_str) == len(other.some_str)
def __lt__(self, other):
    if not isinstance(other, RealString):
        other = RealString(other)
    return len(self.some_str) < len(other.some_str)
# Тесты
str1 = RealString('Молоко')
str2 = RealString('Абрикосы растут')
str3 = 'Золото'
str4 = [1, 2, 3]
print(str1 < str4)
print(str1 >= str2)
print(str1 == str3)
```

Результат выполнения

```
True
False
True
```

Задача 12. Создайте класс «Мебель» с полями «Марка», «Название», «Цена» и методом для вывода подробной информации о предмете. От класса «Мебель» необходимо унаследовать класс «Стол» с унаследованными полями класса «Мебель» и новыми полями «Спинка» (True/False), «Кол-во ножек» и методом для вывода подробной информации.

Задача 13. Создайте базовый класс «Транспортное средство» и производные классы «Автомобиль», «Велосипед», «Повозка». Подсчитайте время и стоимость перевозки пассажиров и грузов каждым транспортным средством.

Задача 14. Создайте базовый класс «Домашнее животное» и производные классы «Собака», «Кошка», «Попугай» и др. С помощью конструктора установите имя каждого животного и его характеристики.

Задача 15. Создайте абстрактный класс Shape для рисования плоских фигур. Необходимо построить производные классы Square (квадрат, который характеризуется координатами левого верхнего угла и длиной стороны), Circle (окружность с заданными координатами центра и радиусом), Ellipse (эллипс с заданными координатами вершин описанного вокруг него прямоугольника), позволяющие рисовать указанные фигуры, а также передвигать их на плоскости.

Задача 16. Создайте приложение, в котором необходимо разработать базовый класс Man. Объекты этого класса содержат справочную информацию о конкретном человеке (фамилию, инициалы, телефон, адрес, возраст). Создайте два производных от

него класса: Manager и Secretary. Объекты класса Manager дополнительно включают номер отдела и количество подчиненных. Объекты класса Manager дополнительно включают номер отдела и количество подчиненных. Объекты класса Secretary дополнительно включают фамилию начальника. Данные о менеджерах и секретарях введите с клавиатуры и выведите на экран дисплея.

Задача 17. Разработайте класс Book: Автор, Название, Издательство, Год, Количество страниц. Создайте массив объектов. Выведите:

а) список книг заданного автора;

б) список книг, выпущенных заданным издательством;

в) список книг, выпущенных после заданного года.

Задача 18. Разработайте класс Word: Слово, Номера страниц, на которых слово встречается (от 1 до 10), Число страниц. Создайте массив объектов. Выведите:

а) слова, которые встречаются более чем на N страницах;

б) слова в алфавитном порядке;

в) для заданного слова номера страниц, на которых оно встречается.

Задача 19. Разработайте класс Равнобочная трапеция, члены класса – координаты 4-х точек. Предусмотрите в классе конструктор и методы: проверка, является ли фигура равнобочной трапецией; вычисление и вывод сведений о фигуре: длины сторон, периметр, площадь.

Практическая работа №22. Решение комплексных задач в Python

Цель: познакомиться с языком программирования Python. Изучить особенности работы с графикой в Python.

Теоретическая часть

Язык Python (по-русски можно произносить как Пайтон или Питон) появился в 1991 году и был разработан Гвидо ван Россумом. Язык назван в честь шоу "Летающий цирк Монти Пайтона". Одна из главных целей разработчиков – сделать язык забавным для использования.

*С помощью **графики в Python** можно рисовать фигуры и изображения, создавать анимацию, визуализировать математические вычисления в Python.*

В программах python можно использовать элементы графики в компьютерных играх.

Графический модуль.

Модуль — это ряд связанных между собой операций. Модуль в Python — это файл, содержащий код языка программирования python, который вы хотите включить в проект. Модули – это, встроенные в язык программирования функции, которые доступны сразу. Чтобы их вызвать, не надо выполнять никаких дополнительных действий.

За время существования любого популярного языка, было написано много функций и классов, которые оказались востребованными многими программистами в разных областях. Включить весь этот код в сам язык если и возможно, то не целесообразно. Чтобы решить проблему доступа к различным возможностям языка в программировании стало общеизвестной практикой использовать так называемые модули, пакеты и библиотек. Каждый модуль содержит коллекцию функций и классов, предназначенных для решения задач из определённой области. Количество модулей для языка Python огромное. Это связано с популярностью языка. Часть модулей собрано в так называемую стандартную библиотеку. Стандартная она потому, что поставляется вместе с утановочным пакетом. Однако существуют сторонние библиотеки, они скачиваются и устанавливаются отдельно.

К стандартным модулям, как вы уже знаете относятся модуль math, random, sys и другие. Для доступа к функционалу модуля, его надо импортировать в программу. После импорта интерпретатор "знает" о существовании дополнительных классов и функций и позволяет ими пользоваться.

Графический модуль

- *в модуле **math** языка Python содержатся математические функции*
- *модуль **random** позволяет генерировать псевдослучайные числа*
- *модуль **sys** предоставляет доступ к системным переменным*

Для работы с графикой в Python нужно импортировать модуль graphics.py. Для использования модулей в программах Питон, необходимо скачать программу модуля в папку, где находится ваш проект.

Для использования модуля graphics скачайте папку с модулем graphics.

Для выполнения задания создайте папку PR 26. Из папки содержащую практические работы скопируете папку «graphics.py-5.0.1.post1» в свою папку PR 26. Зайдите в скопированную папку и нажмите setup. Выйдите из папки. Откройте оболочку IDLE.

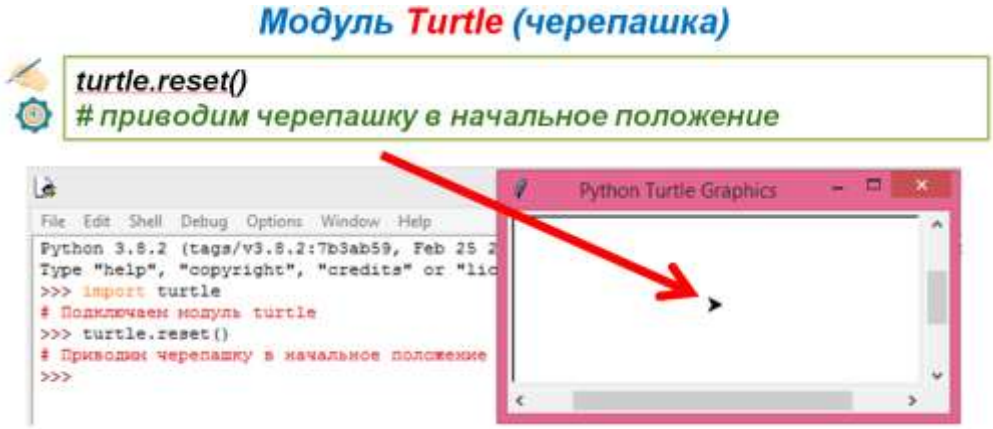
Перед выполнением практической работы изучите теоретический материал, чтобы понимать, что требуется выполнять при решении заданий.

Познакомимся с исполнителем Черепашка в Python.

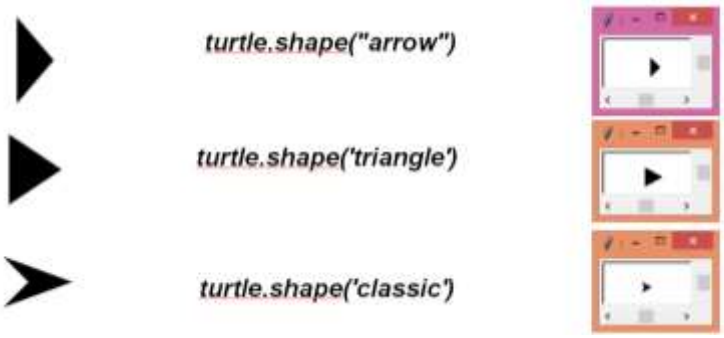
Модуль Turtle в переводе с английского означает черепашка. Для работы в графическом режиме необходимо подключение модуля Turtle при помощи команды import.

import turtle # подключаем модуль turtle

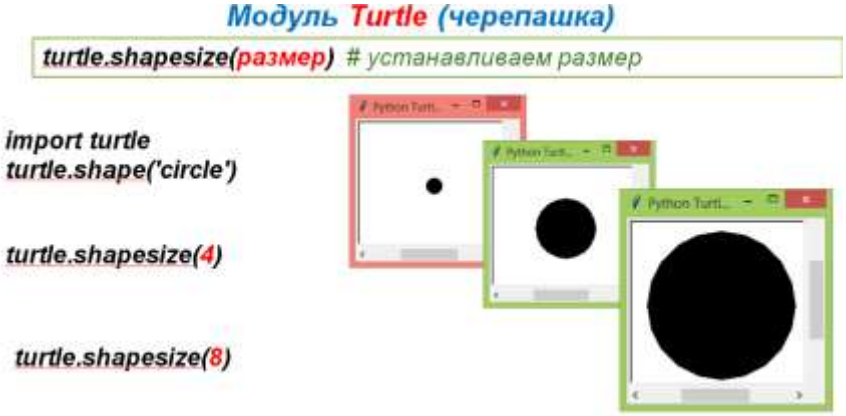
По умолчанию внешний вид исполнителя — это наконечник стрелы всегда направленный в сторону движения.



Команда turtle.shape меняет внешний вид черепашки. В круглых скобках мы указываем стиль внешнего вида: квадрат, черепашка, круг



После того как мы установили стиль модуля turtle, например круг, можно установить его размер, командой shapesize, а в скобочках указать размер в пикселях.



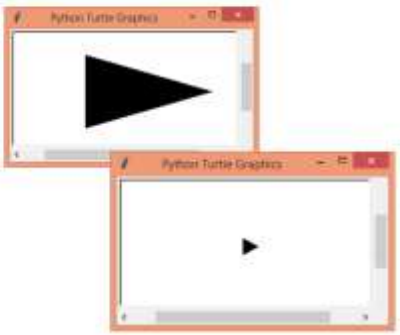
Для стиля квадрат размер создаётся также , как и в предыдущем случае, при этом изменения размера сторон квадрата происходит пропорционально

Команда shapesize позволяет устанавливать для черепашки высоту, ширину и контур. Если мы зададим разную ширину и высоту, то получим прямоугольник.

Для того, чтобы вернуть стиль черепашки, будем использовать команду reset, круглые скобки оставляем пустыми.

Меняем внешний вид:

```
import turtle
turtle.shape("arrow")
```



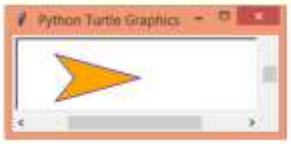
```
turtle.reset()
```

Нашей черепашке можно указать цвет заливки, цвет контура. Для это, до объявления размера, записываем команду turtle.color, а дальше в скобочках указываем цвет, например, синий – для контура и оранжевый для заливки. В команде turtle.shapesize (5,10,5)последняя величина в скобках – толщина контура в пикселях. В данном случае – пять пикселей. По умолчанию толщина контура равна одному пикселю.

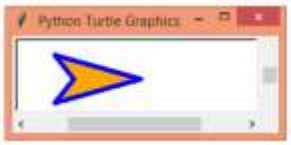
Меняем внешний вид:

```
turtle.color('цвет контура','цвет заливки')
# установка цветовой гаммы исполнителя
```

```
turtle.color('blue','orange')
turtle.shapesize(5,10)
```



```
turtle.shapesize(5,10,5)
```



Для того, чтобы задавать цвета, необходимо знать их названия на английском языке, например

Yellow — жёлтый
Green — зелёный
Blue — голубой, синий
Brown — коричневый
Red — красный и т.д.

Цветовая палитра

Yellow – жёлтый	Pink – розовый
Green – зелёный	Black – чёрный
Blue – голубой, синий	White – белый
Brown – коричневый	Gray – серый
Red – красный	Orange – оранжевый

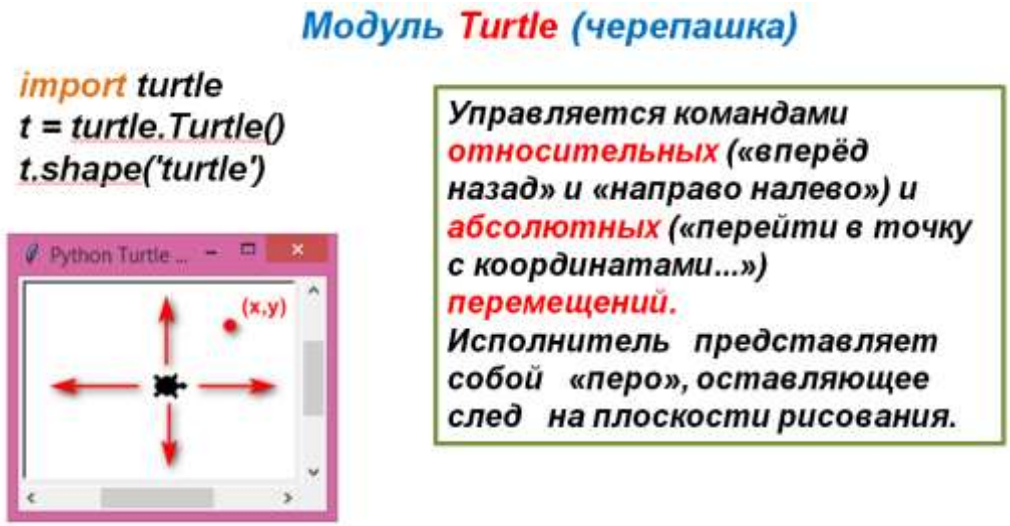
Команда turtle.stamp() оставляет отпечаток на холсте. Давайте рассмотрим пример: первая команда import turtle – подключает модуль черепашка, shape ('square') – устанавливает стиль – квадрат, turtle.color('red','green') – устанавливает цвет контура и цвет заливки, shapesize(5,5,5) – устанавливает размер черепашки и размер контура. turtle.stamp() – оставляет отпечаток на холсте как это показано на рисунке, а добавив команду turtle.shape('circle'), нарисует окружность, вписанную в квадрат

При помощи отпечатка на холсте можно создать определённый рисунок. Команда turtle.left – поворачивает черепашку влево на определённый угол. В скобках указывается угол поворота в градусах. Рассмотрим пример, сначала изобразим черепашку в виде прямоугольника красного цвета, размеры которого 5 на 10 пикселей. Командой turtle.stamp оставим на холсте отпечаток – этого прямоугольника, затем развернём черепашку влево на 90 градусов с установленными размерами

Давайте попробуем по управлять нашим исполнителем черепашка для построения изображений.

Модуль Turtle Управляется командами относительных («вперёд назад» и «направо налево») и абсолютных («перейти в точку с координатами...») перемещений.

Исполнитель представляет собой «перо», оставляющее след на плоскости рисования.



При работе в графическом режиме изображение на экране строится из точек, которые называются пикселями. Каждый пиксель (точка) имеет две координаты: x и y. По умолчанию, исполнитель черепаха установлен в центре холста и имеет координаты (0;0)

Задаем движение черепахи

forward(n) # вперед на n пикселей

backward(n) #назад на n пикселей

left(n) #влево на n градусов

right(n) #вправо на n градусов

circle(r) #начертить окружность радиуса r, с центром слева от курсора, если r>0, справа, если r<0

circle(r,n) #начертить дугу радиуса r, градусной мерой n против часовой стрелки, если r>0, по часовой стрелке, если r<0

goto(x,y) #переместить курсор в точку с координатами (x,y)

Команды рисования (управление пером)

down() #опустить курсор для рисования

up() #поднять курсор

width(n) #ширина следа курсора в n пикселей

color(s) #где s цвет рисования курсора

begin_fill(),end_fill() #рисует закрашенные области (начало и конец рисунка)

Сервисные команды:

reset() #очищается экран, возвращает курсор к центру

clear() #очистить экран

write(s) #вывести строку s в точке нахождения курсора

radians() #мера измерения углов в радианы

degrees() #мера измерения углов в градусах

mainloop() #задержка окна

tracer(f) #режим отладки

Основные команды:

Команда	Сокращение	Назначение команды
forward (n)	fd()	Проползти **вперёд** на n пикселей;
backward(n)	bk()	Проползти **назад** на n пикселей;
right(angle)	rt()	Повернуться **налево** на angle градусов;
left(angle)	lt()	Повернуться **направо** на angle градусов;
goto(x, y)		Переместить черепашку в точку с координатами (x,y);
circle(radius)		Нарисовать **окружность** радиуса |r|, центр которой находится слева от черепашки, если r>0 и справа, если r<1;
circle(r,angle)		Нарисовать **дугу** радиуса |r| и градусной мерой angle. Дуга рисуется против часовой стрелки, если r>0 и по часовой стрелке, если r<0.
circle(r, angle, n)		Нарисовать **дугу** радиусом r, с углом angle и числом шагов n. Чем больше число шагов, тем **плавнее** дуга. Например, нарисуем дугу радиусом 50 пикселей, с углом 180 градусов и числом шагов 100
circle(r, 360, n)		Нарисовать **многоугольник** с радиусом описанной окружности r и числом сторон n. Например, нарисуем шестиугольник с радиусом описанной окружности 100 пикселей
color(s, z)		Установить **цвет пера s и цвет заливки z**
pencolor (s)		Установить цвет **пера**
fillcolor(z)		Установить цвет **заливки**
dot(size, color)		Нарисовать **точку** диаметра size цвета color.

clear()		Очистка экрана.
write(s)		Вывести текстовую строку s в точке нахождения черепашки.
shape()		Изменить значок черепахи ("arrow", "turtle", "circle", "square", "triangle", "classic")
stamp()		Нарисовать копию черепахи в текущем месте
begin_fill()		Начать заливку. Необходимо вызвать перед рисованием фигуры, которую надо закрасить
end_fill()		Остановить заливку. Вызвать после окончания рисования фигуры (конец заливки)
home()		Вернуть черепашку домой — в точку, с координатами (0,0);
speed(n)		Установить скорость черепашки. speed должно быть от 1 (медленно) до 10 (быстро), или 0 (мгновенно);
width(n)		Установить ширину следа черепашки в n пикселей.
reset()		Возврат черепашки в исходное состояние: очищается экран
down()	pd()	Опустить перо.
up()	pu()	Поднять перо

Перед началом работы создайте папку PR 26, скопируйте в эту папку graphics.py-5.0.1.post1 и выполните установку модуля, нажав на имя файла setup.

Работать сего будем через IDLE.

Задание 1. Нарисуем командой circle синюю окружность.

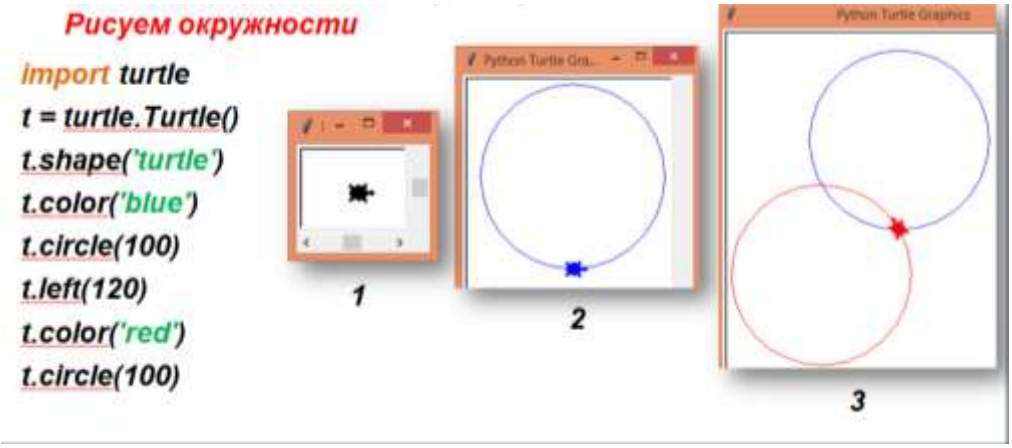
```
from turtle import *
color('blue')
circle(100)
```

Задание 2. Создайте круг и измените внешний вид исполнителя

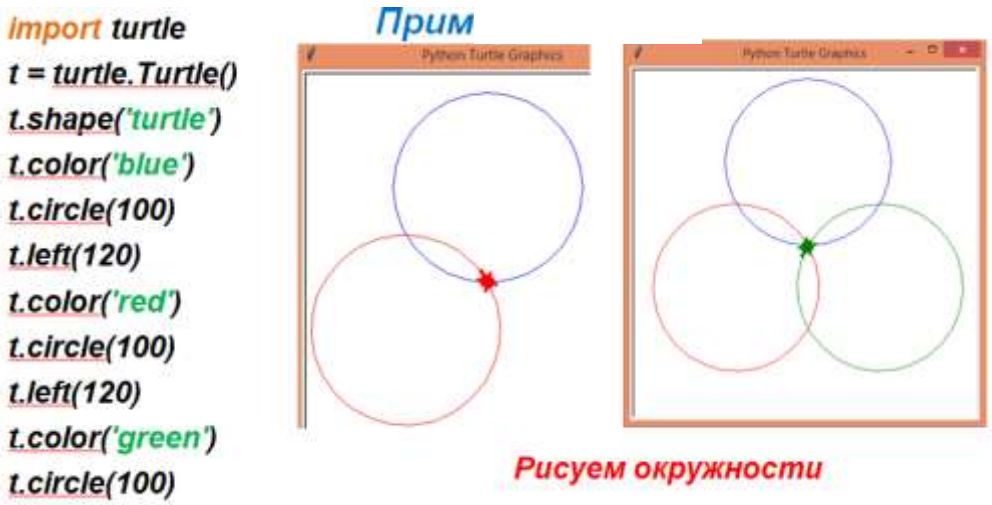
```
import turtle
turtle.color('red','blue')
turtle.shapesize(5,10,5)
from turtle import *
circle(100)
```

Задание 3. Создайте круг, измените внешний вид исполнителя на круг. Измените исполнителю контур и цвет.

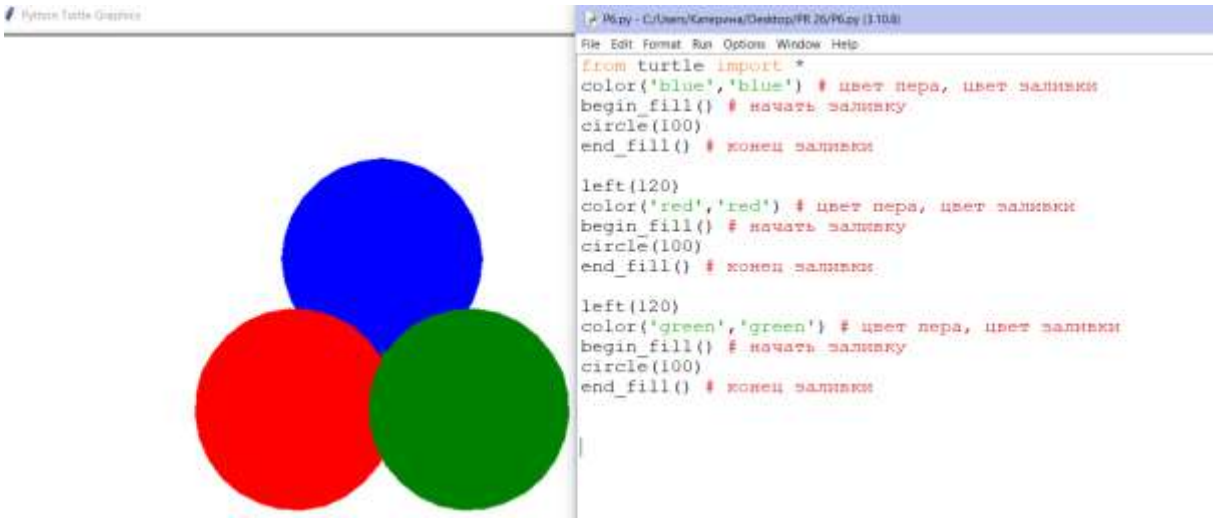
Задание 4. Повернём черепашку на 120 градусов влево и добавим красную окружность



Задание 5. Ещё раз повернём черепашку на 120 градусов влево и добавим зелёную окружность



Задние 6. Добавляем в color цвет заливки и команды: «начать заливку» и «остановить заливку».



Задание 7. Геометрические фигуры

Рисуем простые геометрические фигуры:

- Прямая: просто движение вперед
- Квадрат: вперед, поворот на 90 градусов и так 4 раза. Повторение команд – значит, можно выполнить их в цикле for!
- Пятиконечная звезда: вперед, поворот на 144 градусов и так 5 раз.

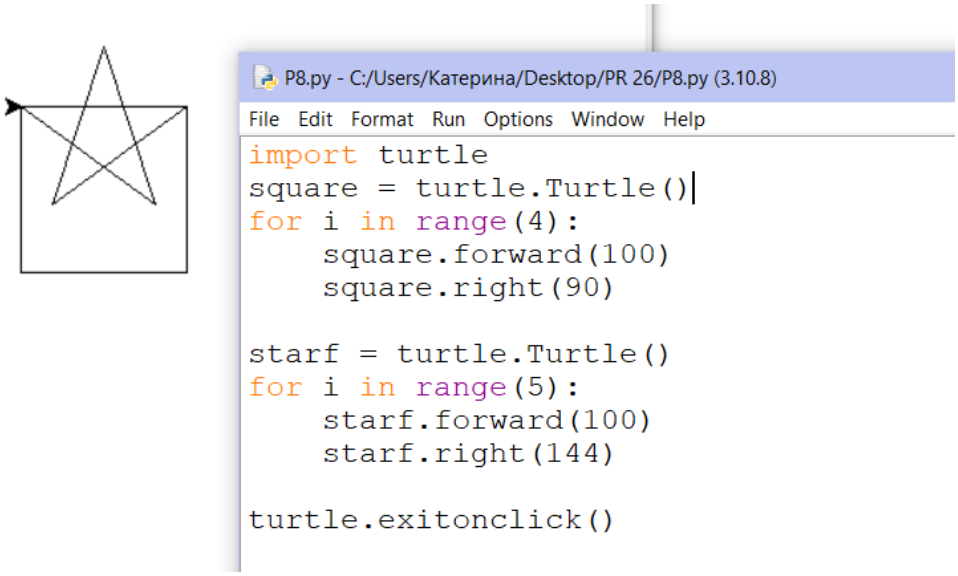
Если мы хотим выполнить инструкции n раз, мы пишем их в цикле

for i in range(n):

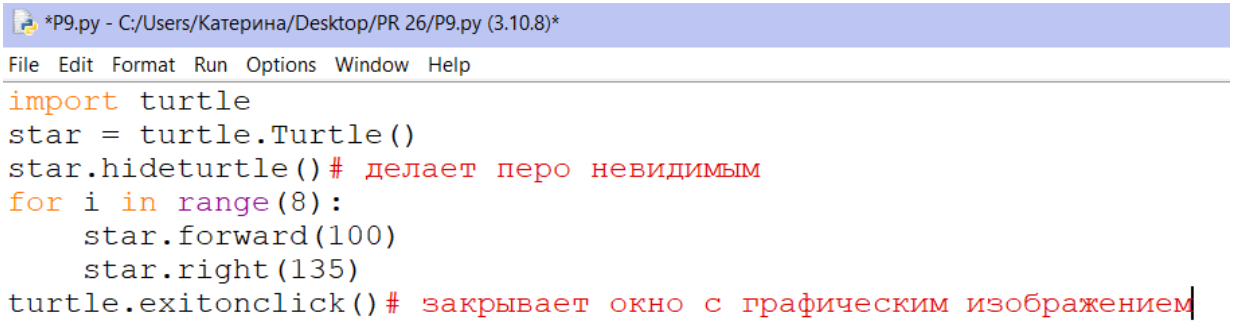
Далее идут инструкции с отступом в 4 пробела. Код с отступами – тело цикла. Когда цикл завершается, отступы больше не ставятся.

Рисуем квадрат:

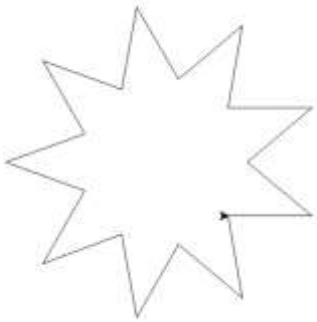
Задание 8. Квадрат и звезду в одной программе, на одном графическом поле, но с разными экземплярами класса Turtle.



Задание 9. Нарисуем восьмиконечную звезду (угол поворота 135 градусов).



Задание 10. Самостоятельно создайте девятиконечную звезду



Задание 11.

Изменяем параметры во время движения

При отрисовке простых фигур черепашка возвращалась в исходную точку, и программа останавливалась, ожидая, когда будет закрыто окно. Если в цикле продолжить рисовать по прежним инструкциям, фигура будет нарисована заново по уже нарисованным контурам. А если ввести дополнительный угол поворота?

Мы также добавили:

- color('red', 'green') определяет цвет линии и цвет заполнения. Черепашка теперь зеленая!
- begin_fill() и end_fill() обозначают начало и конец заполнения

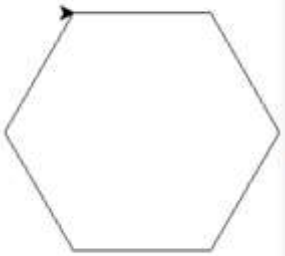
P11.py - C:/Users/Катерина/Desktop/PR 26/P11.py (3.10.8)

File Edit Format Run Options Window Help

```
import turtle
square = turtle.Turtle()
square.shape("turtle")
square.color('red', 'green')
square.begin_fill()
for j in range(3):
    square.left(20)
    for i in range(4):
        square.forward(100)
        square.left(90)

square.end_fill()
turtle.exitonclick()
```

Задание 12.

Напишем обобщенную программу рисования выпуклых равносторонних многоугольников. num_sides – количество граней, side_length – длина грани, angle – угол поворота.



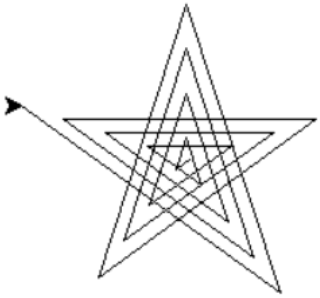
P12.py - C:/Users/Катерина/Desktop/PR 26/P12.py (3.10.8)

File Edit Format Run Options Window Help

```
import turtle

polygon = turtle.Turtle()
num_sides = 6
side_length = 100
angle = 360.0 / num_sides

for i in range(num_sides):
    polygon.forward(side_length)
    polygon.right(angle)

turtle.exitonclick()
```

Самостоятельно измените исполнителя, задайте цвет и контур фигуре

Задание 13. Что будет, если на каждом шаге увеличивать длину пути? В первый день 10 шагов, во второй – 20, далее 30, 40 и так до 200:



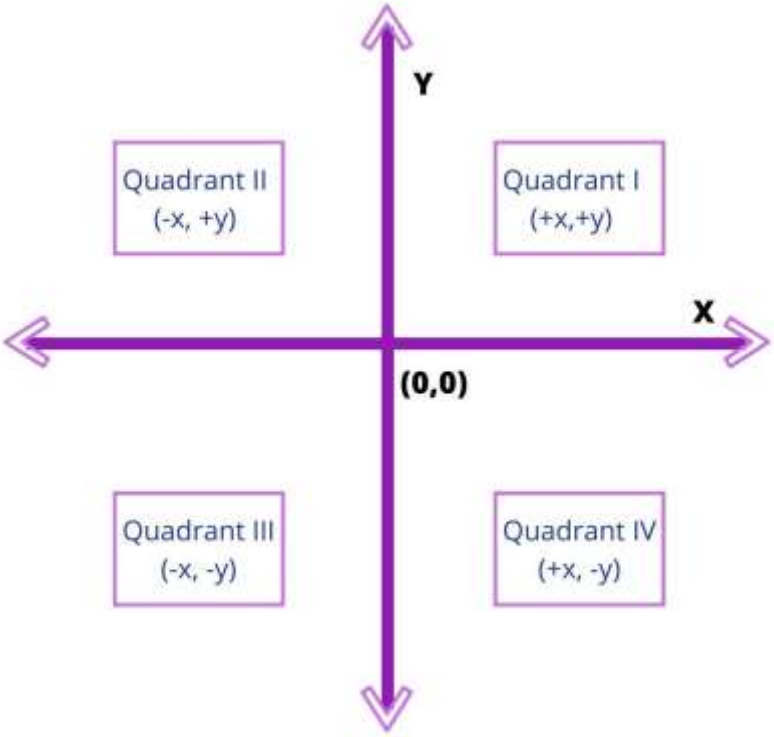
P13.py - C:/Users/Катерина/Desktop/PR 26/P13.py (3.10.8)

File Edit Format Run Options Window Help

```
import turtle

spiral = turtle.Turtle()
for i in range(20):
    spiral.forward(i * 10)
    spiral.right(144)
turtle.exitonclick()
```

Координаты на плоскости

Положение на плоскости определяется двумя числами, x и y:

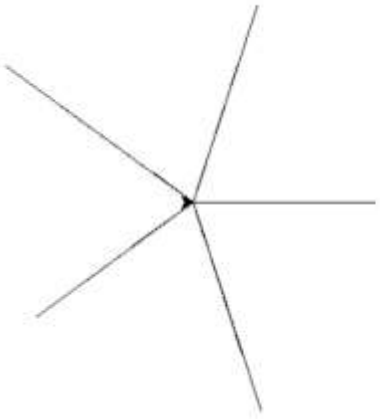


Черепашку в программе можно перемещать функцией goto(x, y).

x и y – числа, или переменные.

goto(0, 0) переместит черепашку в начало координат.

Задание 14. Рисуем линии из точки координат



```
P14.py - C:/Users/Катерина/Desktop/PR 26/P14.py (3.10.8)
File  Edit  Format  Run  Options  Window  Help
import turtle

spiral = turtle.Turtle()
for i in range(20):
    spiral.fd(i * 10)
    spiral.rt(144)
    spiral.goto(0,0)
turtle.exitonclick()
```

Вместо звезды-спирали мы получили 5 линий, расходящихся из точки начала координат.

Задание 15. Круг и точка

В предыдущем задании не хватает плавных изгибов? На помощь приходят функции dot() и circle():

```
P15.py - C:/Users/Катерина/Desktop/PR 26/P15.py (3.10.8)
File  Edit  Format  Run  Options  Window  Help
import turtle

turtle.title("Turtle Drawing")
circle = turtle.Turtle()
circle.shape("turtle")
circle.pensize(5)
circle.pencolor("cyan")

circle.dot(20)
circle.penup()
circle.goto(0, -100)
circle.pendown()
circle.circle(100)
turtle.exitonclick()
```

Дополнительно мы:

- изменили заголовок окна функцией title(),
- установили толщину линии – pensize(),
- установили цвет линии – pencolor(),
- Подняли черепашку перед перемещением – penup() и опустили после – pendown().

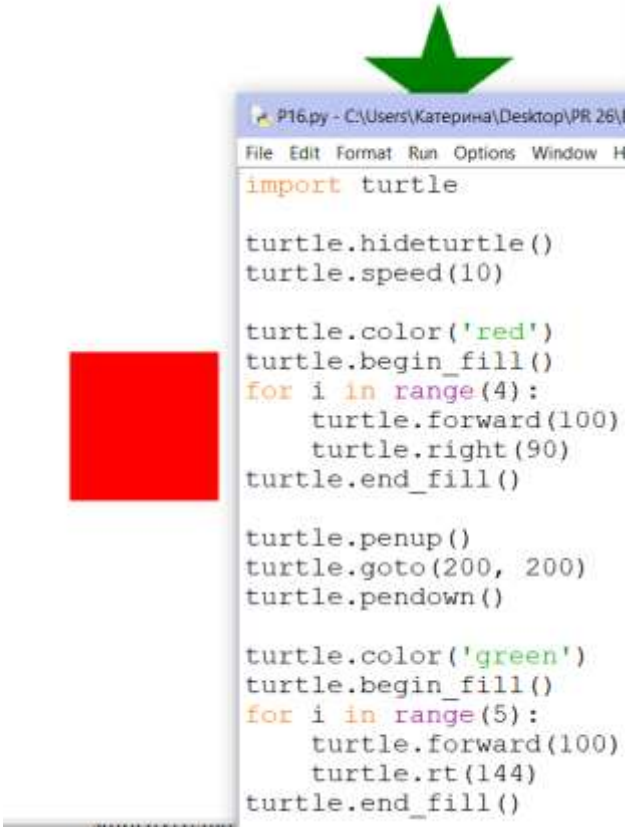
Задание 16.

Самостоятельно:

- Используя код из примеров и функцию goto(), нарисовать галерею из 5 или более многоугольников на одном поле. Использовать экземпляр класса turtle.Turtle().

– Нарисованные многоугольники закрасить разными цветами. Пробуйте стандартные цвета или их шестнадцатеричное представление. Не забудьте кавычки вокруг названия или кода цвета!

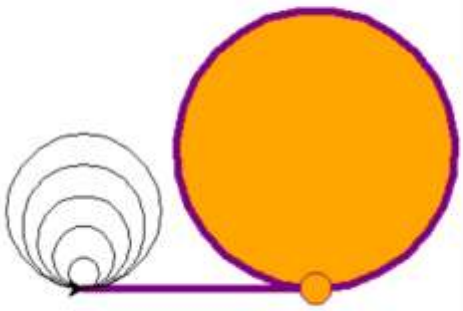
Пример, начала программы



Другие полезные функции:

– turtle.setup(800, 400) устанавливает размеры окна в 800 на 400 пикселей
– turtle.setworldcoordinates(0, 0, 800, 400) устанавливает начало координат в точку 800, 400
– turtle.tracer(0, 0) отключает анимацию
– setpos(x, y) устанавливает черепашку (курсор) в позицию с координатами (x, y)
– seth(x) устанавливает направление в градусах. 0 – горизонтально направо (на восток), 90 – вверх (на север) и так далее
– hideturtle() скрывает черепашку (или стрелку, курсор)
– speed(x) изменяет скорость рисования. Например, speed(11) – почти моментальная отрисовка простых фигур
– clear() очищает холст от нарисованного
– reset() очищает холст и возвращает курсор в начало координат

Задание 17. Пример двух рисунков – экземпляров класса Turtle() – на одном полотне



P17.py - C:/Users/Катерина/Desktop/PR 26/P17.py (3.10.8)

File Edit Format Run Options Window Help

```
import turtle
turtle.title("Turtle Circles")
circ = turtle.Turtle()
circ.pencolor("purple")
circ.fillcolor("orange")
circ.shape("circle")
circ.pensize(5)
circ.speed(10)
circ.fd(150)
circ.begin_fill()
circ.circle(90)
circ.end_fill()

n = 10
t = turtle.Turtle()
while n <= 50:
    t.circle(n)
    n += 10

turtle.exitonclick()
```

Что произошло:

1. Задали название окна,
2. создали экземпляр класса Turtle под именем circ. Все изменения сохраняются для класса circ;
3. цвет линии и заполняющий цвет,
4. форму и размер курсора,
5. установили 10-ю скорость
6. продвинулись на 150 пикселей вперед от старта,
7. начали заполнять фигуру цветом,
8. нарисовали круг
9. закончили заполнять цветом,

Затем:

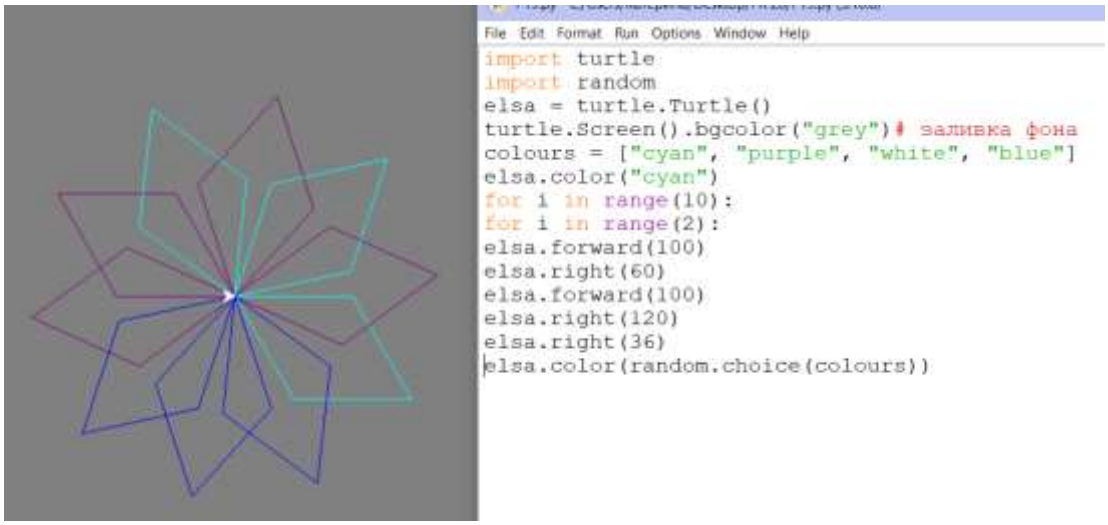
1. Объявили переменную n и присвоили ей значение 10,
2. создали новый экземпляр класса Turtle под именем t. **У него нет настроек экземпляра класса circ**
3. В цикле while: пока переменная n меньше или равна 50, рисовать круги радиусом n;
4. после нарисованного круга увеличить переменную n на 10.
5. Алгоритм рисования кругов прекратит рисовать круги после 4-го круга.

Задание 18. Рисуем дерево

```
import turtle
hr=turtle.Turtle()
hr.left(90)
hr.speed(150)

def tree(i):
    if i>10:
        hr.forward(i)
        hr.left(30)
        tree(3*i/4)
        hr.right(60)
        tree(3*i/4)
        hr.left(30)
        hr.backward(i)
```

Самостоятельно вызовите функцию tree, указав аргумент i, так чтобы у вас получилось дерево.

Задание 19. Графический цветок



Самостоятельно подумайте так расставить пробелы в программе так, чтобы получился подобный рисунок

Задание 20. Нарисуем прямоугольник и сделаем его кнопкой: при нажатии кнопка исчезает и появляется круг.

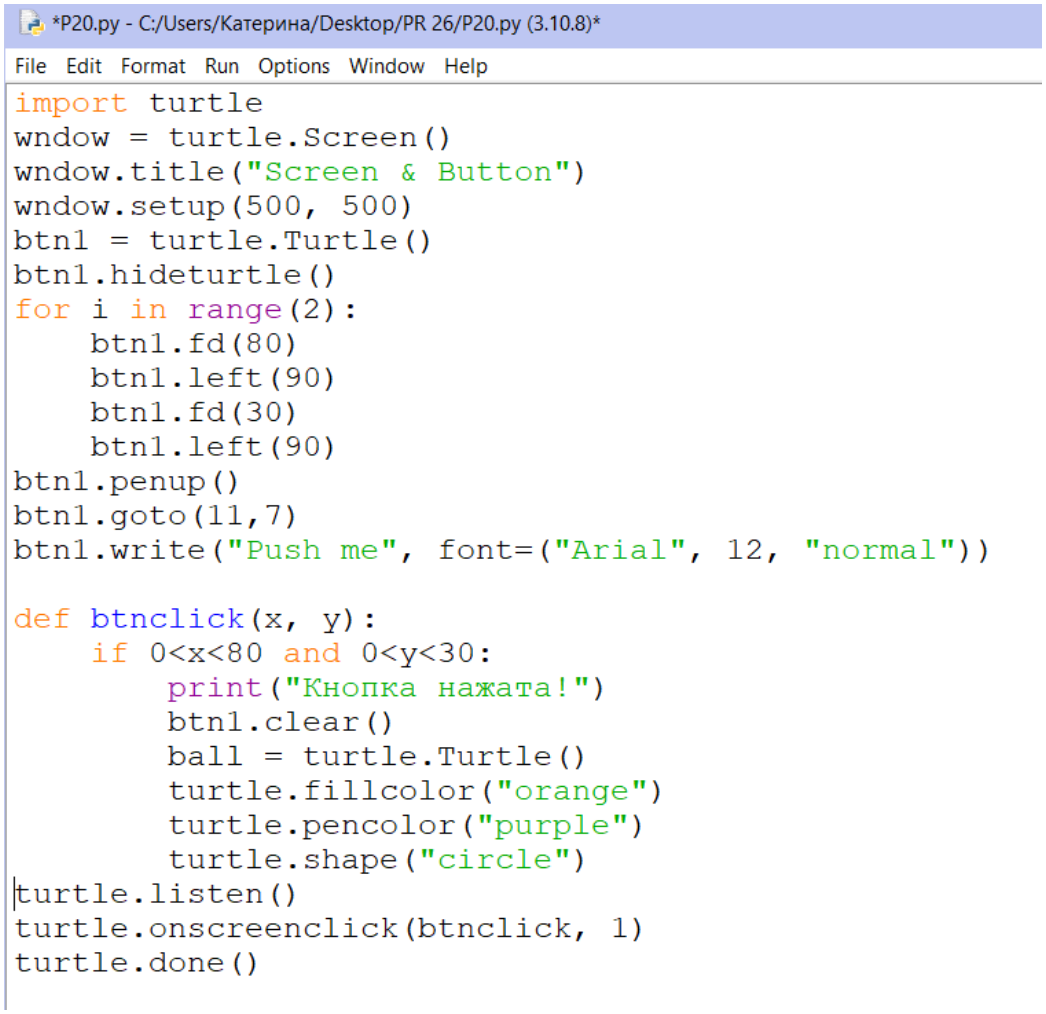
Описание работы кода и самостоятельное задание:

1. Задали название и размеры (500 на 500 пикселей) окна,
2. Создали экземпляр класса btn1 и спрятали курсор (черепашку),
3. Нарисовали **прямоугольник 80 на 30**;
4. подняли перо и перешли на координаты (11, 7);
5. написали **Push me** шрифтом Arial 12-го размера, нормальное начертание. **Попробуйте вместо normal ключевые слова bold (полужирный), italic (наклонный);**

Задаем поведение кнопки:

1. Функции turtle.listen() и turtle.onscreenclick() будут слушать (listen) и реагировать на клик по экрану (onscreenclick). Реакцией будет запуск функции btnclick(x, y)
2. Напишем btnclick(x, y). У нее 2 входных параметра – координаты точки, куда мы кликнули. Наша задача: если клик был по кнопке, спрятать ее и показать оранжевый круг

3. Мы помним: кнопка 80 на 30 пикселей от точки (0, 0). Значит, мы попали по кнопке, если x между 0 и 80 **и** y между 0 и 30. Условие попадания по кнопке: **if 0<x<80 and 0<y<30:**
4. 1) Убираем кнопку: btn1.clear(), 2) создаем экземпляр класса ball = turtle.Turtle(), 3) устанавливаем ему нужные свойства.



Самостоятельно:

1. Нарисовать вторую кнопку (не изменяя первую!), сделать обработчик нажатия на вторую фигуру, при котором рисуется произвольная фигура
2. Создайте третью кнопку, при нажатии на которую появляется случайная фигура: при рисовании фигуры использовать random:

Случайная фигура – это любая фигура, при рисовании которой используются случайные числа. Например:

from random **import** randrange
circle = turtle.Turtle()
circle.circle(randrange(36, 91))

Но есть и второй вариант: случайное число будет индексом списка и укажет на одну из заранее подготовленных неслучайных фигур:

from random **import** randrange
figures = ['circle', 'rectangle', 'triangle']
choice = figures[randrange(0, 3)] *# случайный индекс от 0 до 2 даст одно из трех слов списка*

Таким приемом можно случайно выбирать цвета фигур. Функция choice делает тоже самое изящнее:

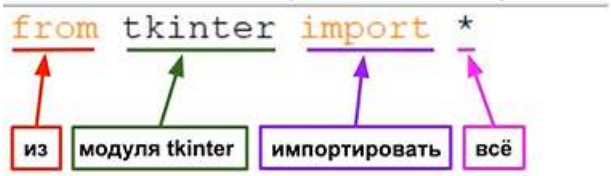
```
from random import randrange, choice
colors = ['red', 'green', 'blue']
color = colors[randrange(0, 3)]
another_color = choice(colors)
```

При работе с turtle кнопки выглядят плоскими. Далее познакомимся с еще одним модулем tkinter.

Модуль tkinter – один из стандартных модулей Python, отвечающий за графический пользовательский интерфейс (GUI).

Задание 21.

Перед началом работы модуль сначала необходимо импортировать в программу.



Далее создаем окно (window). На рисунке окно пустое.

```
window = Tk()
```

Добавим переменную с кнопкой

Далее отобразим кнопку на экране с помощью функции pack()

```
from tkinter import *

window = Tk()
my_button = Button(window, text="Нажать")
my_button.pack()
```

pack() - функция, включающая отображение элемента в окне Tk()

Далее запрограммируем кнопку. В самом начале файла определим функцию greeting(), которая приветствует человека.

```
def greeting():
    print("Добро пожаловать!")
from tkinter import *
```

Для программирования кнопки определим именованный аргумент command – функция, которая выполняется при нажатии на кнопку.

```
my_button = Button(window, text="Нажать", command=greeting)
my_button.pack()
```

при нажатии на кнопку, запустится функция greeting()

Запустите программу и посмотрите как работает кнопка.

Самостоятельно создайте кнопку «Я спать» и функцию, которая будет желать вам спокойной ночи при нажатии на кнопку.

Задание 22. Холст для рисования

Создадим холст для рисования - canvas

```
from tkinter import *

window = Tk()

canvas = Canvas(window, width=500, height=500)
```

окно, в котором создаётся холст

ширина в пикселях

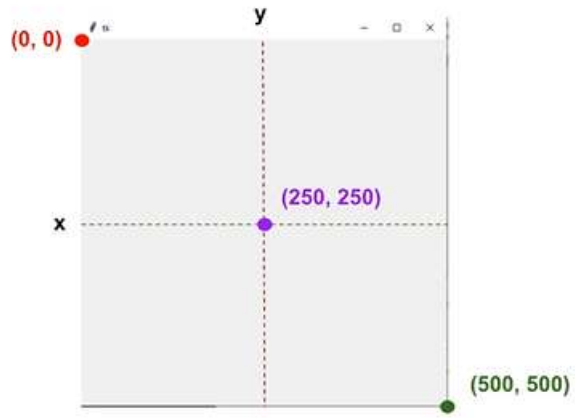
высота в пикселях

Отобразим холст при помощи функции pack()

```
canvas.pack()
```

У нас появится холст размером 500 на 500

Координаты холста



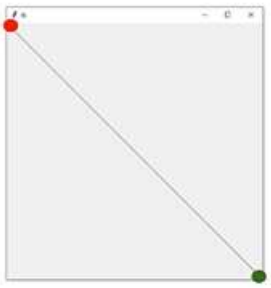
Нарисуем на холсте диагональную линию с помощью функции create_line

```
canvas.create_line(0, 0, 500, 500)
```

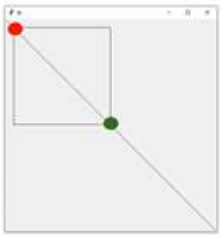
функция “создать линию”

начало линии, x и y

конец линии, x и y



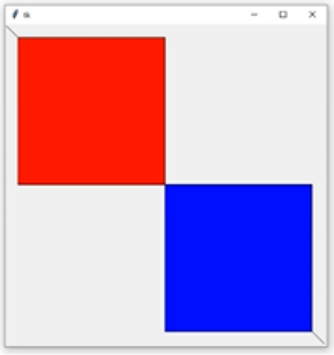
Далее нарисуем квадрат



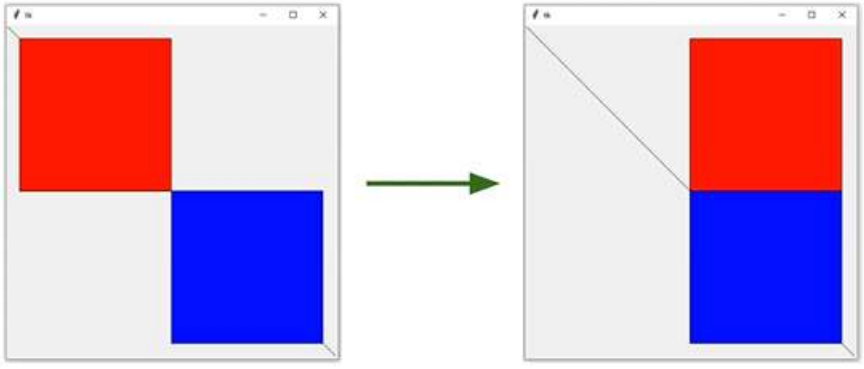
Раскрасим квадрат в красный цвет, при помощи аргумента fill – заливка.

```
canvas.create_rectangle(20, 20, 250, 250, fill="red")
```

Самостоятельно создайте синий квадрат, расположенный как показано на рисунке.



Далее попробуем командой передвинуть красный квадрат вправо



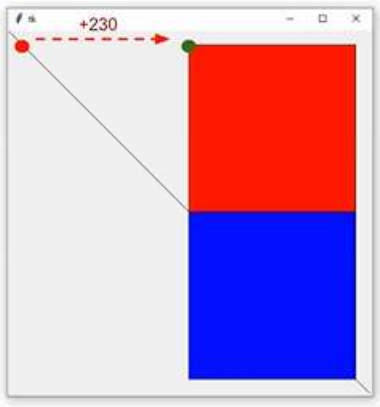
Сначала помещаем оба наших квадрата в переменные, чтобы к ним обращаться


```
red_rect = canvas.create_rectangle(20, 20, 250, 250, fill="red")
```

Синему квадрату задайте переменную самостоятельно

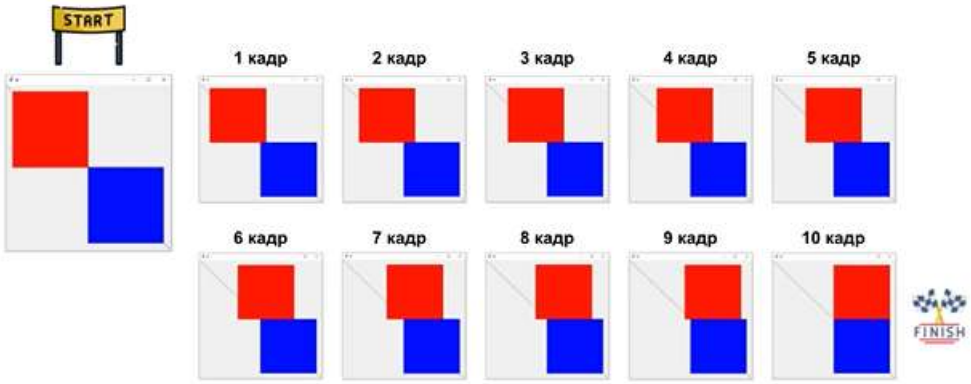
Двигать объекты нам поможет функция move

Запустим программу. Красный квадрат моментально сдвинулся на +230 пикселей вправо



Сдвигать объекты плавно нам поможет анимация.

Анимация – это плавная смена кадров, создающая иллюзию движения, +23 по оси x каждый кадр, 10 кадров.



Создадим цикл на 10 итераций (кадров), который будет двигать квадрат на +23 каждую итерацию

```
a = 10
while a > 0:
    canvas.move(red_rect, 23, 0)
    a = a - 1
```

Если запустить программу, то никакой анимации не будет. Для запуска анимации импортируем модуль time.

```
from tkinter import *
import time
```

Добавим функцию update() для обновления нариванного на экране и sleep(0.1) для паузы.

```
a = 10
while a > 0:
    canvas.move(red_rect, 23, 0)
    a = a - 1
    window.update()
    time.sleep(0.1)
```

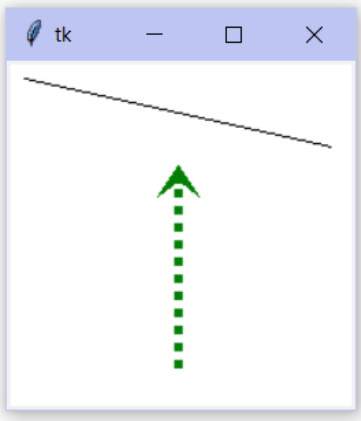
принудительно обновляет (перерисовывает) изображение на экране

задаёт 0.1 секундную паузу

Самостоятельно задайте анимацию для синего квадрата, чтобы он сдвигался влево сразу после красного квадрата.

Задание 23. В программе ниже создается холст. На нем с помощью метода **create_line** рисуются отрезки. Сначала указываются координаты начала (x_1, y_1), затем – конца (x_2, y_2).

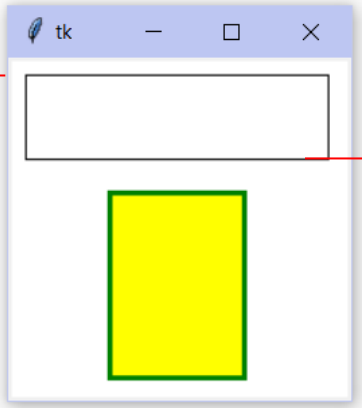
```
from tkinter import *
root = Tk()

c = Canvas(root, width=200, height=200, bg='white')
c.pack()

c.create_line(10, 10, 190, 50)

c.create_line(100, 180, 100, 60, fill='green',
              width=5, arrow=LAST, dash=(10,2),
              activefill='lightgreen',
              arrowshape="10 20 10")

root.mainloop()
```



Остальные свойства являются необязательными. Так **activefill** определяет цвет отрезка при наведении на него курсора мыши.

Задание 24. Создание прямоугольников методом create_rectangle:

```
from tkinter import *
root = Tk()

c = Canvas(root, width=200, height=200, bg='white')
c.pack()

c.create_rectangle(10, 10, 190, 60)

c.create_rectangle(60, 80, 140, 190,
                   fill='yellow',
                   outline='green',
                   width=3,
                   activedash=(5, 4))
```

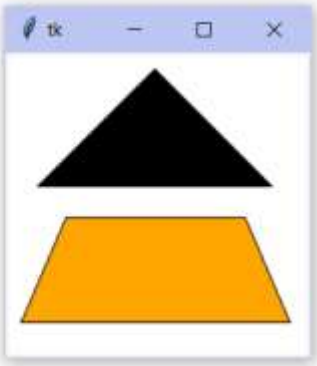


1

Первые координаты – верхний левый угол, вторые – правый нижний. В приведенном примере, когда на второй прямоугольник попадает курсор мыши, его рамка становится пунктирной, что определяется свойством **activedash**.

Задание 25. Методом **create_polygon** рисуется произвольный многоугольник путем задания координат каждой его точки

```
from tkinter import *
root = Tk()

c = Canvas(root, width=200, height=200, bg='white')
c.pack()

c.create_polygon(100, 10, 20, 90, 180, 90)

c.create_polygon(40, 110, 160, 110,
                 190, 180, 10, 180,
                 fill='orange', outline='black')
```

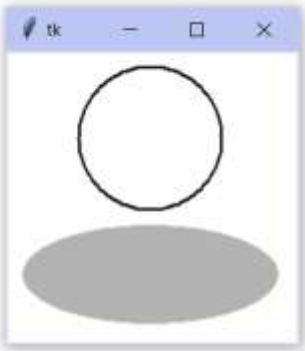


Для удобства координаты точек можно заключать в скобки:

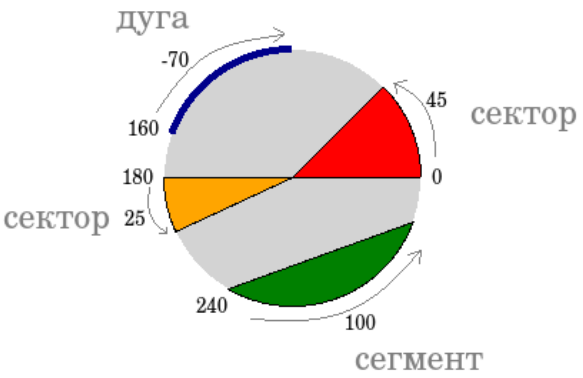
```
c.create_polygon((40, 110), (160, 110),
                 (190, 180), (10, 180),
                 fill='orange', outline='black')
```

Задание 26. Метод **create_oval** создает эллипсы. При этом задаются координаты гипотетического прямоугольника, описывающего эллипс. Если нужно получить круг, то соответственно описываемый прямоугольник должен быть квадратом.

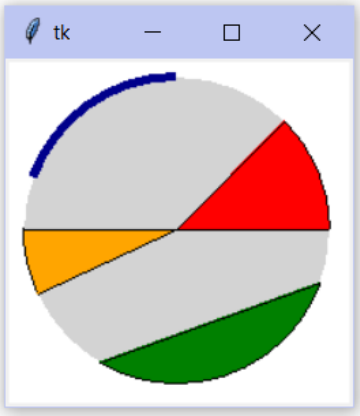
```
from tkinter import *
root = Tk()

c = Canvas(root, width=200, height=200, bg='white')
c.pack()

c.create_oval(50, 10, 150, 110, width=2)
c.create_oval(10, 120, 190, 190,
              fill='grey70', outline='white')
```



Задание 27. Более сложные для понимания фигуры получаются при использовании метода **create_arc**. В зависимости от значения опции **style** можно получить сектор (по умолчанию), сегмент (**CHORD**) или дугу (**ARC**). Также как в случае **create_oval** координаты задают прямоугольник, в который вписана окружность (или эллипс), из которой "вырезают" сектор, сегмент или дугу. Опции **start** присваивается градус начала фигуры, **extent** определяет угол поворота.




```
from tkinter import *
root = Tk()

c = Canvas(root, width=200, height=200, bg='white')
c.pack()

c.create_oval(10, 10, 190, 190,
              fill='lightgrey',
              outline='white')
c.create_arc(10, 10, 190, 190,
             start=0, extent=45,
             fill='red')
c.create_arc(10, 10, 190, 190,
             start=180, extent=25,
             fill='orange')
c.create_arc(10, 10, 190, 190,
             start=240, extent=100,
             style=CHORD, fill='green')
c.create_arc(10, 10, 190, 190,
             start=160, extent=-70,
             style=ARC, outline='darkblue',
             width=5)
```



В данном примере светло-серый круг используется исключительно для наглядности.

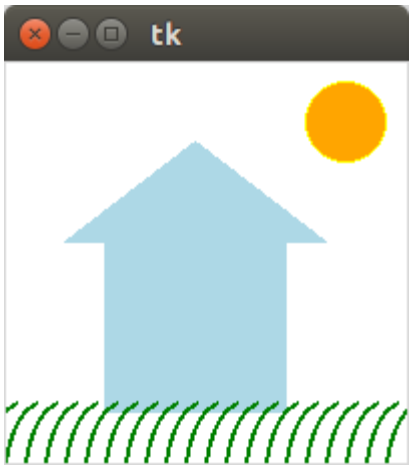
Задание 28. На холсте можно разместить текст. Делается это с помощью метода create_text

```
from tkinter import *
root = Tk()

c = Canvas(root, width=200, height=200, bg='white')
c.pack()

c.create_text(100, 100,
              text="Hello World,\nPython\nand Tk",
              justify=CENTER, font="Verdana 14")
c.create_text(200, 200, text="About this",
              anchor=SE, fill="grey")
```

По умолчанию в заданной координате располагается центр текстовой надписи. Чтобы изменить это и, например, разместить по указанной координате левую границу текста, используется якорь со значением **W** (от англ. west – запад). Другие значения: **N**, **NE**, **E**, **SE**, **S**, **SW**, **W**, **NW**. Если букв, задающих сторону привязки, две, то вторая определяет вертикальную привязку (вверх или вниз «уйдет» текст от заданной координаты). Свойство **justify** определяет лишь выравнивание текста относительно себя самого.

Задание 29. Разместите на холсте произвольный текст и кнопку, которая будет выводить произвольное сообщение.

Задание 30. Создайте на холсте подобное изображение:



Для создания травы используется цикл.

```
for i in range (5, 200, 10):
    c.create_arc(0 + i, 160, 23 + i, 290,
                 start=160, extent=-90,
                 style=ARC, outline='green',
                 width=2)
```

Координаты у вас могут отличаться

Задание 31. Рисование математических кривых

Рассмотрим задачу построения графика некоторой функции по вычисляемым точкам с помощью Tkinter.

Поскольку Tkinter позволяет работать с элементами GUI, создадим окно заданного размера, установим для него заголовок и цвет фона «холста», а также снабдим окно

программной «кнопкой». На «холсте» определим систему координат и нарисуем «косинусоиду».

```
import tkinter
import math
#
tk=tkinter.Tk()# создаем окно
tk.title("Sample") # задаем название окна
#
canvas=tkinter.Canvas(tk)
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack()
#
canvas.create_text(20,10,text="20,10")
canvas.create_text(460,350,text="460,350")
#
points=[]
ay=150
y0=150
x0=50
x1=470
dx=10
#
for n in range(x0,x1,dx):
        y=y0-ay*math.cos(n*dx)
        pp=(n,y)
        points.append(pp)
#
canvas.create_line(points,fill="blue",smooth=1)
#
y_axe=[]
yy=(x0,0)
y_axe.append(yy)
yy=(x0,y0+ay)
y_axe.append(yy)
canvas.create_line(y_axe,fill="black",width=2)
#
x_axe=[]
xx=(x0,y0)
x_axe.append(xx)
xx=(x1,y0)
x_axe.append(xx)
canvas.create_line(x_axe,fill="black",width=2)
```

```
#
tk.mainloop()
```

Для элемента canvas указываются высота, ширина, цвет фона и отступ от границ окна (таким образом, окно получается несколько больше, чем элемент canvas). Размеры окна автоматически подстраиваются так, чтобы обеспечить размещение всех элементов.

После размещение виджета canvas в окне исследуем систему координат. Поскольку размеры окна уже нами заданы, полезно определить, где находится точка с координатами (0,0). Как видно из попыток вывести значения координат с помощью функции canvas.create_text(), начало координат находится в верхнем левом углу «холста».

Теперь, определившись с координатами, можно выбрать масштабные коэффициенты и сдвиги и сформировать координаты точек для рисования кривой.

Для функции canvas.create_line() в качестве координат требуется список пар точек (кортежей) (x,y). Этот список формируется в цикле с шагом dx.

Для линии графика устанавливаются цвет и режим сглаживания. Это сглаживание обеспечивает некоторую «плавность» кривой. Если его убрать, линия будет состоять из отрезков прямых. Кроме того, для линий можно устанавливать толщину, как это показано для «осей координат».

Задание 32. Моделирование математических функций

Пусть требуется построить график функции, вид которой выбирается из списка. Здесь потребуется уже использование дополнительных интерфейсных элементов библиотеки Tkinter, а также создание собственных (пользовательских) процедур или функций для облегчения понимания кода.

Для выбора вида математической функции используется раскрывающийся список, после выбора вида функции и нажатия на кнопку «Нарисовать» на «холсте» схематически рисуется график этой функции. Кнопка «Закрыть» закрывает «приложение».

```
import tkinter
import math
#
# Пользовательские процедуры
def plot_x_axe(x0,y0,x1):
        x_axe=[]
        xx=(x0,y0)
        x_axe.append(xx)
        xx=(x1,y0)
        x_axe.append(xx)
        canvas.create_line(x_axe,fill="black",width=2)
#
def plot_y_axe(x0,y0,y1):
        y_axe=[]
        yy=(x0,y1)
        y_axe.append(yy)
        yy=(x0,y0)
        y_axe.append(yy)
        canvas.create_line(y_axe,fill="black",width=2)
#
```

```
def plot_func0(x0,x1,dx,y0,y1):
        x0i=int(x0)
        x1i=int(x1)
        y0i=int(y0)
        y1i=int(y1)
        a=y1
        b=(y0-y1)/(x1-x0)
        points=[]
        for x in range(x0i,x1i,dx):
                y=int(a+b*x)
                pp=(x,y)
                points.append(pp)
        #
        canvas.create_line(points,fill="blue",smooth=1)
        plot_y_axe(x0i,y0i,y1i)
        plot_x_axe(x0i,y0i,x1i)
#
def plot_func1(x0,x1,dx,y0,y1):
        x0i=int(x0)
        x1i=int(x1)
        y0i=int(y0)
        y1i=int(y1)
        a=y0
        b=y0-y1
        points=[]
        for x in range(x0i,x1i,dx):
                y=int(a-y1i*b/x)
                pp=(x,y)
                points.append(pp)
        #
        canvas.create_line(points,fill="blue",smooth=1)
        plot_y_axe(x0i,y0i,y1i)
        plot_x_axe(x0i,y0i,x1i)
#
def plot_func2(x0,x1,dx,y0,y1):
        x0i=int(x0)
        x1i=int(x1)
        y0i=int(y0)
        y1i=int(y1)
        a=(y0-y1)/(15*x1)
        b=1+((y0-y1)/(x1-x0))
        points=[]
        for x in range(x0i,x1i,dx):
                y=y0i-int(a*(x-x0i)**b)
                pp=(x,y)
```

```
                points.append(pp)
        #
        canvas.create_line(points,fill="blue",smooth=1)
        plot_y_axe(x0i,y0i,y1i)
        plot_x_axe(x0i,y0i,x1i)
#
def plot_func3(x0,x1,dx,y0,y1):
        x0i=int(x0)
        x1i=int(x1)
        y0i=int(y0)
        y1i=int(y1)
        ay=150
        y0i=150
        points=[]
        for x in range(x0i,x1i,dx):
                y=y0i-ay*math.cos(x*dx)
                pp=(x,y)
                points.append(pp)
        #
        canvas.create_line(points,fill="blue",smooth=1)
        plot_y_axe(x0i,0,y0i+ay)
        plot_x_axe(x0i,y0i,x1i)
#
def DrawGraph():
        fn=func.get()
        f=fn[0]
        x0=50.0
        y0=250.0
        x1=450.0
        y1=50.0
        dx=10
        #
        if f=="0":
                canvas.delete("all")
                plot_func0(x0,x1,dx,y0,y1)
        elif f=="1":
                canvas.delete("all")
                plot_func1(x0,x1,dx,y0,y1)
        elif f=="2":
                canvas.delete("all")
                plot_func2(x0,x1,dx,y0,y1)
        else:
                canvas.delete("all")
                plot_func3(x0,x1,dx,y0,y1)
#
```

```
# Основная часть
tk=tkinter.Tk()
tk.title("Sample")
# Верхняя часть окна со списком и кнопками
menuframe=tkinter.Frame(tk)
menuframe.pack({"side":"top","fill":"x"})
# Надпись для списка
lbl=tkinter.Label(menuframe)
lbl["text"]="Выбор:"
lbl.pack({"side":"left"})
# Инициализация и формирования списка
func=tkinter.StringVar(tk)
func.set('0 y=Ax+B')
#
fspis=tkinter.OptionMenu(menuframe,func,
                    '0 y=Ax+B',
                    '1 y=A+B/x',
                    '2 y=Ax^B',
                    '3 y=A*cos(Bx)')
fspis.pack({"side":"left"})
# Кнопка управления рисованием
btnOk=tkinter.Button(menuframe)
btnOk["text"]="Нарисовать"
btnOk["command"]=DrawGraph
btnOk.pack({"side":"left"})
# Кнопка закрытия приложения
button=tkinter.Button(menuframe)
button["text"]="Закрыть"
button["command"]=tk.quit
button.pack({"side":"right"})
# Область рисования (холст)
canvas=tkinter.Canvas(tk)
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack({"side":"bottom"})
tk.mainloop()
```

Основная часть программы (интерфейсная) начинается с момента создания корневого окна (инструкция tk=Tkinter.Tk()). В этом окне располагаются два интерфейсных элемента — рамка (Frame) и холст (canvas). Рамка является «контейнером» для остальных интерфейсных элементов — текстовой надписи (метки — Label), раскрывающегося списка вариантов (OptionMenu) и двух кнопок.

Как видно, кнопка закрытия стала объектом рамки, а не корневого окна, но попрежнему

закрывает всё окно. Для получения нужного расположения элементов метод pack() используется с указанием, как именно размещать элементы интерфейса (к какой стороне элементаконтейнера их нужно «прижимать»).

Есть некоторые тонкости в создании раскрывающегося списка. Для успешного выполнения этой операции нужно предварительно сформировать строку (а точнее, объект Tkinter.StringVar()) и определить для этого объекта значение по умолчанию (это значение показывается в только что запущенном приложении). Затем в элементе OptionMenu() список значений дополняется. При выборе элемента списка изменяется значение именно этой строки и для дальнейшей работы нужно его анализировать, что и делается в процедуре DrawGraph().

«Вычислительная» часть, а именно, все процедуры и функции, обеспечивающие вычисления координат точек и рисование линий, вынесена в начало текста программы.

Определение каждой пользовательской процедуры или функции обеспечивается составным оператором def. Поскольку функции занимаются только рисованием, они не возвращают никаких значений (т.е. результаты выполнения этих функций не присваиваются никаким переменным).

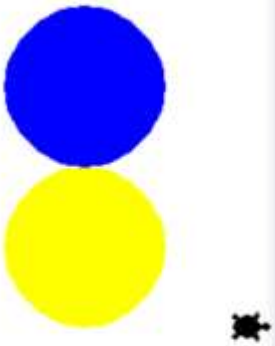
Процедура собственно рисования графика DrawGraph() вызывается при нажатии кнопки «Нарисовать», и имя этой функции является командой, которая сопоставляется кнопке. Эта процедура берёт значение из списка (метод get()), выбирает первый символ получившейся строки и в зависимости от этого символа вызывает другие процедуры и функции для построения конкретных графиков с установленными масштабными коэффициентами.

Перед рисованием следующего графика математической функции холст очищается командой canvas.delete("all"). Для построения графика каждой функции вычисляются собственные масштабные коэффициенты, поэтому их вычисление включено в код соответствующей процедуры. Кроме того, для графика нужны целые значения координат, поэтому в каждой процедуре выполняются соответствующие преобразования с помощью функции int().

Для каждого графика требуется нарисовать оси, и действия по рисованию осей также вынесены в отдельные процедуры. Таким образом, оказывается, что программу нужно читать «с конца», и писать тоже.

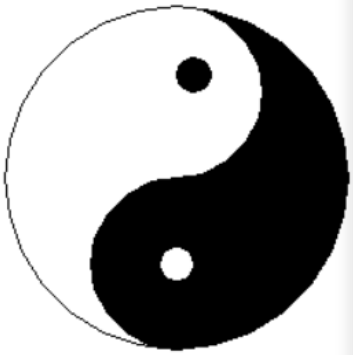
Задание 33. Рисование окружности при помощи turtle.

Вам дан фрагмент программы. Дополните программу так, чтобы вокруг желтого круга были круги разного цвета. Круги должны располагаться по аналогии с синим кругом. В конце черепашка должна стать черной.



```
File Edit Format Run Options Window Help
import turtle
turtle.shape('turtle')
turtle.color('yellow')
turtle.begin_fill()
turtle.circle(50)
turtle.end_fill()
turtle.penup()
turtle.goto(0, 100)
turtle.pendown()
turtle.color('blue')
turtle.begin_fill()
turtle.circle(50)
turtle.end_fill()
turtle.penup()
turtle.goto(100, 0)
turtle.pendown()
turtle.color('black')
```

Задание 34.



```
import turtle
turtle.fillcolor('black')
turtle.begin_fill()
turtle.circle(100,180)
turtle.circle(50,-180)
turtle.circle(-50, -180)
turtle.end_fill()
turtle.penup()
turtle.right(90)
turtle.fd(40)
turtle.pendown()
turtle.right(90)
turtle.fillcolor('white')
turtle.begin_fill()
turtle.circle(10, 360)
turtle.end_fill()
turtle.right(90)
turtle.fd(40)
turtle.right(90)
turtle.circle(-100, 180)
turtle.right(90)
turtle.penup()
turtle.fd(40)
turtle.fillcolor('black')
turtle.begin_fill()
turtle.pendown()
turtle.circle(10, 360)
turtle.end_fill()
turtle.hideturtle()
turtle.done()
```

Задание 35.

```
import turtle
import random
window = turtle.Screen()
arthur = turtle.Turtle()
window.colormode(255)
arthur.speed(0)
arthur.width(2)
window.bgcolor(50,0,70)
arthur.pencolor(255,255,0)

def shape(angle,side,limit):
                reverseDirection = 200
                arthur.forward(side)

                if side % (reverseDirection*2) == 0:
                                angle = angle + 2
                                print(side)
                elif side % reverseDirection == 0:
                                angle = angle - 2
                                print(side)

                arthur.right(angle)
                side = side + 2
                if side < limit:
                                shape(angle,side,limit)
angle = 119
side = 0
limit = 600
shape(angle, side, limit)
turtle.done()
```

Список литературы

Основные источники

1. Трофимов, В. В. Основы алгоритмизации и программирования : учебник для среднего профессионального образования / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — Москва : Издательство Юрайт, 2022. — 137 с. — (Профессиональное образование). — ISBN 978-5-534-07321-8. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/493261
2. Чернышев, С. А. Основы программирования на Python : учебное пособие для среднего профессионального образования / С. А. Чернышев. — Москва : Издательство Юрайт, 2022. — 286 с. — (Профессиональное образование). — ISBN 978-5-534-15160-2. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/496897

Дополнительные источники

1. Гуриков, С. Р. Основы алгоритмизации и программирования на Python : учебное пособие / С.Р. Гуриков. — Москва : ИНФРА-М, 2022. — 343 с. — (Среднее профессиональное образование). - ISBN 978-5-16-016906-4. - Текст : электронный. - URL: https://znanium.com/catalog/product/1356004
2. Колдаев, В. Д. Основы алгоритмизации и программирования : учебное пособие / В.Д. Колдаев ; под ред. проф. Л.Г. Гагариной. — Москва : ФОРУМ : ИНФРА-М, 2022. — 414 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0733-7. - Текст : электронный. - URL: https://znanium.com/catalog/product/1735805
3. Огнева, М. В. Программирование на языке C++: практический курс : учебное пособие для среднего профессионального образования / М. В. Огнева, Е. В. Кудрина. — Москва : Издательство Юрайт, 2022. — 335 с. — (Профессиональное образование). — ISBN 978-5-534-05780-5. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/493047
4. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для среднего профессионального образования / Д. Ю. Федоров. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2022. — 210 с. — (Профессиональное образование). — ISBN 978-5-534-12829-1. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/492921