****

**Федеральное государственное образовательное бюджетное учреждение высшего образования**

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ**

**ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

(Финансовый университет)

Департамент анализа данных, принятия решений и финансовых технологий

**И.В. Миронова**

**ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PYTHON**

Учебно-методическое пособие

по дисциплине «Компьютерный практикум»

для студентов, обучающихся по направлению подготовки

38.03.05 "Бизнес-информатика"

(очная и заочная формы обучения)

Москва, 2018

****

**Федеральное государственное образовательное бюджетное учреждение высшего образования**

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ**

**ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

(Финансовый университет)

Департамент анализа данных, принятия решений и финансовых технологий

**И.В. Миронова**

**ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PYTHON**

Учебно-методическое пособие

по дисциплине «Компьютерный практикум»

для студентов, обучающихся по направлению подготовки

38.03.05 "Бизнес-информатика"

(очная и заочная формы обучения)

*Рассмотрено и одобрено на заседании Департамента анализа данных, принятия решений и финансовых технологий*

*(протокол № от «» 2018 г.)*

Москва, 2018

УДК 004 (072)

ББК 32.973.2

М64

**Автор**: Миронова И.В., канд. физ.-мат. наук, доцент, доцент департамента анализа данных, принятия решений и финансовых технологий Финансового университета при Правительстве Российской Федерации

**Рецензент**: Утакаева И.Х., канд. физ.-мат. наук, доцент департамента анализа данных, принятия решений и финансовых технологий Финансового университета при Правительстве Российской Федерации

**Программирование на языке Python**: Учебно-методическое пособие по дисциплине «Компьютерный практикум» для студентов, обучающихся по направлению подготовки 38.03.05 "Бизнес-информатика" (очная и заочная формы обучения). - М.: Финансовый университет, департамент анализа данных, принятия решений и финансовых технологий, 2018. -68 с.

Дисциплина «Компьютерный практикум» является обязательной дисциплиной базовой части профессионального цикла ООП по направлению 38.03.05 "Бизнес-информатика" (модуль математики и информатики Б.1.1.2.3.). Изучение данной дисциплины нацелено на формирование у студентов практических навыков по использованию возможностей программирования для решения прикладных задач профессиональной деятельности. В учебном пособии рассматриваются основные типы данных и операторы языка Python. По каждой теме приведены задания для самостоятельной работы.

УДК 004 (072)

ББК 32.973.2

**Учебное издание**

*Миронова Ирина Васильевна*

**Программирование на языке Python**

Учебно-методическое пособие

по дисциплине «Компьютерный практикум»

Компьютерный набор, верстка И.В. Миронова

Формат 60x90/16. Гарнитура Times New Roman

Усл. п.л. 4,2. Изд. № - 2018.

Заказ № \_\_\_\_\_\_

Электронное издание

© ФГОБУ ВО «Финансовый университет при Правительстве Российской Федерации», 2018.

© Департамент анализа данных, принятия решений и финансовых технологий, 2018.

© Миронова Ирина Васильевна, 2018.

**Содержание**

[1. Структура программы на Python. Ввод и вывод данных 5](#_Toc518548275)

[2. Операторы для работы с числовыми объектами. Встроенные функции для работы с числами и функции модуля math 11](#_Toc518548276)

[3. Операторы сравнения и логические операторы. Инструкция ветвления if...else 15](#_Toc518548277)

[4. Цикл while 20](#_Toc518548278)

[5. Цикл for. Функция range 24](#_Toc518548279)

[6. Списки 27](#_Toc518548280)

[7. Строки 39](#_Toc518548281)

[8. Словари 49](#_Toc518548282)

[9. Множества 61](#_Toc518548283)

# 1. Структура программы на Python. Ввод и вывод данных

Программа на Python состоит из инструкций. Каждая инструкция располагается на отдельной строке. Если инструкция не является вложенной, то она должна начинаться с начала строки (пробелы перед инструкцией недопустимы), иначе будет выдано сообщение об ошибке.

Если нужно в одной строке расположить несколько инструкций, то их следует разделять символом точка с запятой.

Слишком длинную инструкцию можно перенести на следующую строку, поставив в конце символ продолжения строки «\» (перед Enter). Этот символ должен быть последним в строке. Однако, любая часть программы, заключенная в скобки (круглые, квадратные или фигурные) или тройные кавычки может располагаться на нескольких строках без использования символа продолжения строки. Например,



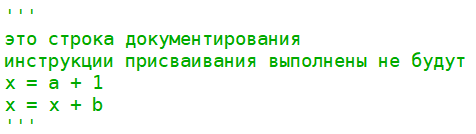
или



Комментарии в Python являются однострочными. Комментарий начинается с символа «#» и заканчивается в конце строки. Если символ «#» находится внутри строки в кавычках, то он не считается символом начала комментария.



Если нужно закомментировать несколько строк программы, то можно разместить их внутри тройных кавычек или апострофов. Получится строка документирования. Данный фрагмент программы выполняться не будет, хотя он и не является комментарием. Например,



Данные в программе являются объектами. Каждый объект имеет значение и тип. Например, когда в программе используется инструкция x = 22, создается объект типа int со значением 22. Тип (или класс) объекта определяет его внутреннее представление и операции, которые над ним можно выполнять.

*Таблица 1. Основные типы данных в языке Python*

|  |  |  |
| --- | --- | --- |
| ***Тип*** | ***Описание*** | ***Пример объекта*** |
| int | целые числа | 123 -12 |
| float | вещественные числа | 12.345 |
| str | строки символов | 'строка' |
| bool | логические значения | True False |
| list | списки | [1, 22, 333] |
| tuple | кортежи | (1, 2, 45, 1) |
| dict | словари | {'i':0, 'j':1} |
| set | множества | {5, 15, 25} |

Для доступа к объектам используются переменные.

Имя переменной (идентификатор) состоит из букв, цифр и символов подчеркивания. Имя не может начинаться с цифры. В качестве имен нельзя использовать ключевые слова языка. Следует избегать совпадения имен с уже имеющимися в системе, например, с именами функций, так как это может привести к ошибкам. В именах нужно учитывать регистр букв.

В языке отсутствуют инструкции объявления переменных. Переменная создается, когда в программе ей впервые присваивается значение. Значением переменной является ссылка на объект. Переменные не имеют информации о типе, тип имеют объекты. Поэтому во время выполнения программы переменной можно присваивать значения разных типов, ошибка не возникнет. Это называется динамической типизацией. Когда переменная участвует в выражении, ее имя заменяется на объект, на который она ссылается.

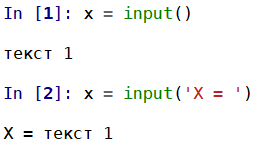
Значение переменной присваивается с помощью оператора «=». В простейшем случае это выглядит так:



Но могут использоваться и более сложные конструкции:



Для ввода данных в языке Python используется функция input(). Функция имеет необязательный параметр сообщение. При работе функции сообщение, если оно задано, будет выведено на экран, и система будет ожидать ввода с клавиатуры. Нужно набрать необходимый текст и нажать клавишу Enter. Значением функции будет введенная строка символов. Ее можно сохранить в переменной или использовать в каком-либо выражении для выполнения последующих действий. Проще всего понять работу функции на примере:



В обоих случаях вводился один и тот же текст «текст 1». Он и будет значением переменной x, но во втором случае понятно, что система ожидает ввода данных, а вот в первом случае нет. Чтобы не попасть в ситуацию, когда система ожидает ввода данных, а вы этого не видите, всегда используйте сообщение-подсказку при вызове функции input().

Значением функции input() является строка символов (объект типа str), даже если вы ввели число. Чтобы с введенным значением можно было выполнять арифметические операции, его нужно преобразовать к типу int или float. Для этого используются функции int() или float() соответственно. Например,

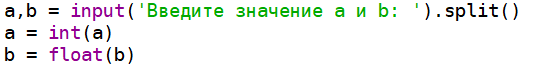


Или, используя вложенные функции:



Учтите, что разделителем между целой и дробной частью числа при вводе является точка. Если введенную строку нельзя преобразовать к нужному типу, возникнет ошибка.

Можно сразу в одной строке ввести несколько значений, разделив их, например, пробелами. В этом случае удобно воспользоваться методом split():



Если вы введете строку «12 23.45», то после выполнения ввода переменная a будет иметь значение «12», а b – значение «23.45». Значения имеют тип str. Подробнее метод split() будет рассмотрен позднее при изучении списков. Количество значений во введенной строке и количество переменных в левой части оператора присваивания должно совпадать. Такое присваивание называется позиционным.

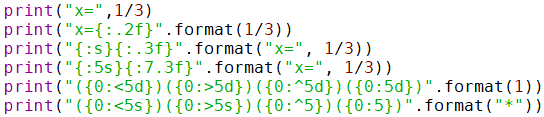
Для вывода результатов на экран используется функция print(). В скобках через запятую указываются значения (объекты), которые нужно вывести. Функция преобразует каждый объект в строку и выводит его на экран. Между полученными строками автоматически вставляется значение, указанное в параметре sep. По умолчанию sep = ' ' (строка, состоящая из одного пробела), поэтому вставляется пробел. Если нужно, чтобы значения выводились, например, через запятую и пробел, задайте sep = ', '. В конце выводится значение, заданное в параметре end. По умолчанию end = '\n' (символ перевода строки), поэтому курсор после выполнения функции переходит на новую строку. Если нужно, чтобы следующая функция print() продолжила печать в той же строке, то задайте значение параметра end = '' (пустая строка, две кавычки или два апострофа рядом без пробела). Примеры использования функции:

|  |  |
| --- | --- |
| Инструкции | Результат выполнения |
|  |  |

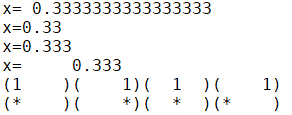
Если при печати нужно вывести числовое значение с определенным количеством знаков после запятой или нужно, чтобы значение занимало на экране определенное число позиций (например, при печати таблиц), то следует использовать форматирование. Для этого в Python можно использовать метод format(). Строка, для которой вызывается метод, может содержать любые символы, которые выводятся без преобразований, и специальные значения внутри фигурных скобок (спецификаторы формата), которые при печати заменяются на строку, созданную в соответствии с указанным форматом.

Внутри фигурных скобок до двоеточия указывается номер аргумента метода format(), который будет выводиться на месте данной фигурной скобки. Аргументы метода нумеруются слева направо, начиная с нуля. Если номер не указан, то значения подставляются по порядку. После двоеточия чаще всего указывают выравнивание, ширину поля, точность и преобразование. Выравнивание задается с помощью следующих символов: < – по левому краю (по умолчанию для строк), > – по правому краю (по умолчанию для чисел), ^ – по центру. Ширина поля определяет сколько символов должно быть выведено на печать. Если реальная строка длиннее, то выводятся все символы, если короче, то до нужной длины добавляются пробелы в соответствии с выравниванием. Если ширина не задана, то выводится столько символов, сколько есть в строке. Точность указывается после символа точка и определяет количество знаков после запятой, которое выводится для десятичных чисел. В качестве значения параметра преобразование чаще всего используются: d – десятичное целое значение, f – вещественное число, s – строка (по умолчанию). Любой из указанных параметров может быть опущен. Здесь описаны не все возможности для указания формата, более подробное описание можно найти в справочной литературе.

Рассмотрим несколько примеров:



Ниже приведены результаты выполнения этих инструкций:



***Задания для самостоятельной работы***

1. Значения переменных А и В ввести с клавиатуры и вывести на экран. После этого значения меняются местами, т.е. A нужно присвоить значение B, а B – значение A, и вновь значения переменных вывести на экран.
2. Введите строку, состоящую из 2 цифр. Преобразуйте ее в целое и вещественное число. Выведите полученные 3 значения (строку, целое число, вещественное число) на экран в одной строке через запятую, затем пропустите строку и вновь выведите значения по одному на строке. Перед каждым значением выведите его тип.

# 2. Операторы для работы с числовыми объектами. Встроенные функции для работы с числами и функции модуля math

*Таблица 2. Математические операторы Python*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Оператор*** | ***Описание*** | ***Пример*** | ***Результат*** |
| \*\* | возведение в степень | 2\*\*3  3.0\*\*-2 | 8  0.111111111 |
| -, + | унарный (одноместный) минус и плюс | x=5  x=-x | x имеет значение -5 |
| / | деление | 5/2 | 2.5 |
| // | деление с округлением вниз | 5//2  5.0//2  -5//2 | 2  2.0  -3 |
| \* | умножение | 10\*1.5 | 15 |
| % | остаток от деления | 5%2  5.0%2  -5%2 | 1  1.0  1 |
| + | сложение | 10+1.5 | 11.5 |
| - | вычитание | 10-1.5 | 8.5 |

Приоритет операторов в порядке убывания:

* возведение в степень (\*\*);
* унарный минус (-) и плюс (+);
* деление (/, //), умножение (\*), остаток от деления (%);
* сложение (+) и вычитание (-).

Порядок вычисления выражения можно, как обычно, изменить с помощью скобок.

Для работы с числами в Python используются следующие встроенные функции:

**int()** – преобразование к целому числу. Пример:



**float(**) – преобразование к числу с плавающей точкой. Пример:



**round()** – округление до N знаков после запятой. Пример:





**abs()** – возвращает абсолютную величину (модуль числа). Пример:



**pow()** – ( x \*\* y ) % r. Пример:



**max(), min()** – находят максимальное или минимальное значение. Пример:



О других встроенных функциях можно прочитать в литературе.

Модуль math содержит дополнительные функции для работы с числами, а также стандартные математические константы. Ниже приводятся некоторые полезные функции модуля math:

**math.pi, math.e** – числа π и e. Пример:



**math.sin(), math.cos(), math.tan()** – тригонометрические функции синус, косинус, тангенс. Аргумент указывается в радианах. Пример:



**math.asin(), math.acos(), math.atan()** – обратные тригонометрические функции (арксинус, арккосинус, арктангенс). Результат возвращается в радианах. Пример:



**math.degrees(), math.radians()** – функции преобразования радиан в градусы и градусов в радианы. Пример:



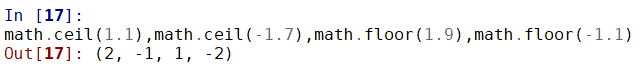
**math.exp()** – экспонента.

**math.log()** – логарифм по заданному основанию. Если основание не указано, вычисляется натуральный логарифм.

**math.sqrt()** – квадратный корень. Пример:



**math.ceil(), math.floor()** – округление до ближайшего большего и меньшего целого числа. Пример:



**math.pow()** – возведение числа в указанную степень. Пример:

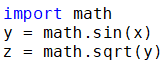


**math.fabs()** – модуль числа.

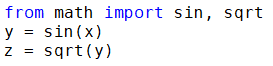
**math.fmod()** – остаток от деления X на Y.

**math.factorial()** – факториал числа.

Чтобы в программе можно было использовать функции модуля math, нужно предварительно выполнить инструкцию import:



Можно импортировать только отдельные функции, как показано ниже. В этом случае при вызове функции имя модуля указывать не нужно:



***Задания для самостоятельной работы***

1. Значение x вводится с клавиатуры. Вычислите . Выведите на экран значения x и y с 3 знаками после запятой.
2. Вычислите сумму цифр трехзначного числа.
3. Для заданного трехзначного числа выведите число, у которого цифры идут в обратном порядке, например, для числа 123 ответ 321.
4. Ввести координаты 2 точек: (x1, y1) и (x2, y2). Вычислите расстояние между этими точками. Результат выведите с 5 знаками после запятой.
5. Треугольник задан координатами своих вершин: (x1, y1), (x2, y2), (x3, y3). Значения переменных x1, y1, x2, y2, x3, y3 определите с помощью присваивания. Они могут быть нецелыми. Найти периметр и площадь треугольника (2 способами). (Указания: для вычисления площади можно использовать формулу Герона S=, где p – полупериметр, a, b, c – длины сторон, или формулу, выражающую площадь через координаты вершин, S=).
6. Заданы 3 корня кубического уравнения, например, 0.5, -1.2, 1. Найти коэффициенты этого уравнения и вывести уравнение на экран дважды в виде:

(x-(0.5))(x-(-1.2))(x-(1))=0

x^3+(-1.7)\*x^2+(0.1)\*x+(0.6)=0

Случаи, когда коэффициент равен нулю или единице или корень равен нулю отдельно не рассматривать.

# 3. Операторы сравнения и логические операторы. Инструкция ветвления if...else

В языке определены следующие операторы сравнения:

– меньше чем;

– больше чем;

– равно;

– не равно;

– больше или равно;

– меньше или равно;

in – проверка на вхождение (будет рассмотрен позднее).

Результатом операции является значение True, если выражение истинно, и False, если выражение ложно.

Операторы можно объединять в последовательности, например, x < y < z < v. Такая последовательность эквивалентна выражению x < y and  y < z and z < v.

Из имеющихся выражений можно составить более сложные конструкции, используя логические операторы not, and, or и скобки.

Оператор not (логическое отрицание) меняет значение на обратное (True на False и наоборот). Например,



Оператор and (логическое И) имеет значение True, если оба операнда имеют значение True, и False – в противном случае. Например,



Оператор or (логическое ИЛИ) имеет значение True, если хотя бы один операнд имеет значение True. Например,



Приоритет операторов в порядке убывания:

* <, >, ==, !=, <=, >=, in;
* not;
* and;
* or.

Инструкция ветвления в общем случае имеет вид:

**if** *выражение1***:**

*инструкции1*

**elif** *выражение2*:

*инструкции2*

**elif** *выражение3*:

*инструкции3*

…

**else**:

*инструкции*

Части elif и else могут отсутствовать. Инструкция ветвления выполняется следующим образом. Вычисляется *выражение1*. Если оно истинно, то выполняются *инструкции1* и выполнение ветвления завершается. Если *выражение1* ложно, то вычисляется следующее выражение и т.д. Если все выражения ложны, то выполняются инструкции, указанные после else. Если else отсутствует, то ничего не делается.

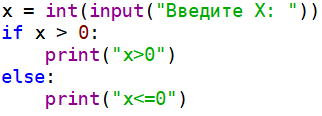
Вложенные инструкции внутри инструкции ветвления выделяются отступами. Для Python очень важно, чтобы эти отступы были одинаковыми (обычно это 4 пробела, но лучше использовать табуляцию). Слова if, elif, else должны находиться в одной позиции, так Python определяет, что они относятся к одной инструкции. Инструкции, которые следуют за инструкцией ветвления, тоже должны начинаться в этой же позиции. Эти правила являются общими для всех синтаксических конструкций, имеющих вложенные инструкции: циклов, функций и т.д. Их нарушение приведет к ошибкам.

Выражения, используемые в инструкции ветвления, часто в литературе называют условиями, а саму инструкцию ветвления – условным оператором.

Если при выполнении условия никакие действия не должны выполняться, то нельзя на месте инструкций оставить пустую строку. В этом случае нужно использовать инструкцию pass, которая является инструкцией, но ничего не делает.

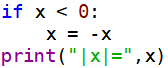
Пример 3.1

Напишем программу, в которой выводится текст «x>0», если введено положительное целое число, и текст «x<=0» в противном случае.



Пример 3.2

Вычислим абсолютное значение числа и выведем его на печать.



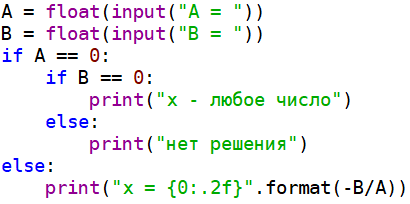
В данном случае нам потребовался вариант инструкции, в котором нет частей elif и else. Обратите внимание, что print расположен под if, поэтому он выполняется для любого x независимо от выполнения условия.

Эту программу можно записать короче:



Пример 3.3

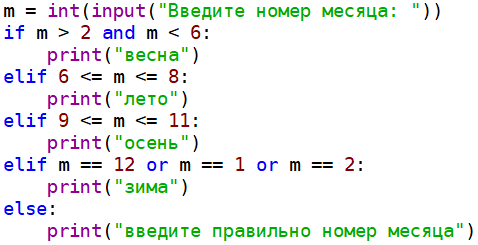
Напишем программу для решения уравнения A\*x+B=0. Коэффициенты A и B вводятся. Программа должна выдавать ответ для любых значений A и B.



Это пример программы, в которой используются вложенные инструкции ветвления.

Пример 3.4

Данная программа определяет время года по номеру месяца. Обратите внимание на более сложные условия и использование частей elif и else.



***Задания для самостоятельной работы***

1. Напишите программу, которая при попадании значения в диапазон печатает «Yes», в противном случае – «No». Границы диапазона и значение вводятся.
2. Определить минимальное значение среди чисел *a, b, c, d* (не использовать стандартные функции max и min).
3. Напишите программу, которая переводит оценку из 100-балльной системы в пятибалльную по правилам, принятым в университете.
4. Вычислить значение F: F=1, если цифра 7 входит в запись заданного трехзначного числа, и 0 – в противном случае.
5. Даны два отрезка [*a;b*] и [*c;d*]. Найдите их пересечение. Если отрезки не пересекаются, то выдайте сообщение.
6. По номеру года определите, является ли данный год високосным. (год является високосным, если его номер кратен 4, но не кратен 100, а также если он кратен 400).
7. Для отрезков длины *a, b, c* определить, можно ли из них составить треугольник и является ли этот треугольник прямоугольным.
8. Напишите программу для решения уравнения *ax2+bx+c=0*. Коэффициенты *a, b, c* могут быть любыми числами.
9. Вводится целое число. Выведите его на экран и допишите к нему слова «рубль», «рубля» или «рублей» в зависимости от значения. Алгоритм:

• исключение: если число оканчивается на 11, 12, 13 или 14, добавляем слово «рублей»;

• если число оканчивается на 1, добавляем слово «рубль»;

• если число оканчивается на 2, 3 или 4, добавляем слово «рубля»;

• если число оканчивается на цифры 5, 6, 7, 8, 9 или 0, добавляем слово «рублей».

# 4. Цикл while

Циклы позволяют выполнять инструкции многократно. В языке имеется два вида циклов. Рассмотрим цикл while. В общем случае он имеет вид:

**while** *выражение*:

*инструкции*

**else**:

*инструкции, выполняемые, если не было break*

Часть else является необязательной и чаще всего отсутствует.

Выражение, указанное в первой строке, должно иметь значение True или False. Его называют условием цикла, а циклы такого вида часто называют циклами с условием. Первую строку принято называть заголовком цикла. Инструкции, указанные после заголовка, называют телом цикла. Это вложенные инструкции, поэтому для них должны выполняться те же правила для отступов, которые мы рассматривали выше в инструкции ветвления. Выполнение инструкций тела цикла один раз называют итерацией.

Цикл выполняется следующим образом. Вычисляется значение выражения в заголовке цикла. Если оно имеет значение True (условие выполняется), то выполняются инструкции тела цикла. После этого вновь возвращаемся к заголовку цикла и проверяем условие. Если условие в заголовке цикла имеет значение False, то цикл завершает работу. Далее будут выполнены инструкции, указанные после else, если они есть.

Использование инструкции break в теле цикла приводит к немедленному прекращению цикла, при этом не выполняется ветка else. Следующей будет выполняться инструкция, следующая сразу за циклом.

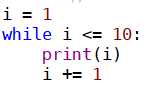
Использование инструкции continue в теле цикла приводит к тому, что все оставшиеся инструкции тела цикла пропускаются, происходит переход на строку заголовка и проверка условия цикла. Далее все выполняется как обычно.

Цикл завершает работу, когда выражение в заголовке цикла будет иметь значение False. Следовательно, цикл может не выполниться ни разу, если при первой же проверке условие ложно. Это не является ошибкой. Если часть else имеется, то инструкции в ней будут выполнены.

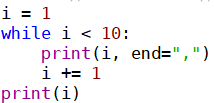
Следует очень внимательно проверять условие завершения цикла. Обычно оно зависит от каких-то переменных, которые изменяются в теле цикла. Если это не так, то в теле цикла обязательно должна быть инструкция break, которая и завершит цикл. В противном случае вы можете получить бесконечный цикл.

Пример 4.1

Выведем числа от 1 до 10 на экран.

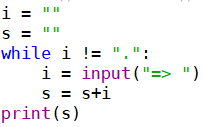


При таком решении числа будут выведены друг под другом. Ниже приведено другое решение, когда числа выводятся в одной строке через запятую.



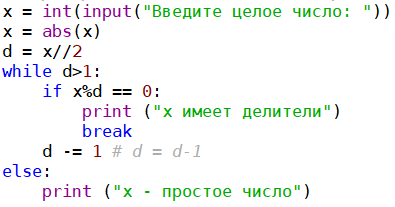
В первом случае цикл выполняется 10 раз, во втором – 9, так как последнее число выводится на экран не так, как остальные. Переменная i называется счетчиком. Она позволяет контролировать, сколько раз выполнился цикл (количество итераций).

Пример 4.2



В данной программе на каждой итерации цикла вводится строка символов, которая добавляется к символам, введенным ранее. Количество итераций цикла заранее неизвестно. Цикл завершает работу, если будет введена строка, состоящая из одного символа точка.

Пример 4.3



Программа проверяет, является ли введенное целое число простым. В данном случае обратите внимание на использование break и else.

***Задания для самостоятельной работы***

1. Напишите программу, которая выводит на экран значения функции sin(x) на заданном интервале [a,b] с шагом 0.1 в виде таблицы, состоящей из двух столбцов. В первом столбце выводится значение x, во втором – значение sin(x). Все значения выводятся с 2 знаками после запятой.
2. Напишите программу, которая вводит с клавиатуры числа до тех пор, пока не будет введено число 0. В конце работы программы на экран выводятся количество чисел (0 не считаем), минимальное и максимальное из введенных чисел, их сумму и среднее значение.
3. Напишите программу, которая вычисляет минимальное целое положительное число среди введенных значений. Признаком конца ввода является символ «.».
4. Напишите программу, которая находит наибольший общий делитель двух чисел, используя модифицированный алгоритм Евклида: нужно заменять большее число на остаток от деления большего на меньшее до тех пор, пока этот остаток не станет равен нулю; тогда второе и есть НОД.

# 5. Цикл for. Функция range

В общем случае цикл for имеет вид:

**for** *переменная* **in** *объект*:

*инструкции*

**else**:

*инструкции, выполняемые, если не было break*

Как и для цикла while часть else не является обязательной. Термины заголовок цикла, тело цикла и итерация имеют тот же смысл. Однако заголовок цикла здесь выглядит иначе.

Переменная в заголовке цикла for называется переменной цикла. На каждой итерации она получает новое значение из объекта, указанного после слова in. Цикл завершает свою работу, когда все значения просмотрены. Учтите, что если до выполнения цикла у вас уже была переменная с таким же именем как у переменной цикла, то ее значение будет испорчено. После завершения цикла вы можете использовать переменную цикла. Она будет равна последнему значению, полученному в цикле. Значение переменной цикла не обязательно является числом. Оно может быть объектом любого типа.

В качестве объекта в заголовке цикла for могут использоваться объекты самых разных типов, поддерживающих итерацию. По мере изучения языка мы увидим примеры для строк, списков, кортежей, словарей, множеств и других объектов.

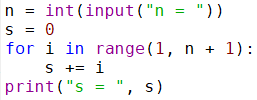
Инструкции break и continue в циклах for используются так же, как в циклах while.

Функция range() позволяет выполнить цикл для последовательности целых чисел из заданного диапазона. В общем случае у функции 3 аргумента, но 2 из них могут быть опущены, поэтому рассмотрим каждый вариант функции отдельно:

|  |  |
| --- | --- |
| for i in range(n): | Значение i меняется от 0 до n-1. Если значение n равно нулю или отрицательное, то тело цикла не выполнится ни разу. |
| for i in range(a, b): | Значение i меняется от a до b-1. Если b≤a, то цикл не выполнится ни разу. |
| for i in range(a, b, c): | Переменная i принимает значения a, a+c, a+2\*c и т.д., пока i<b (для c>0) или пока i>b (для c<0). Для с>0 должно выполняться условие a<b, для с<0 – условие a>b, иначе цикл не выполнится ни разу |

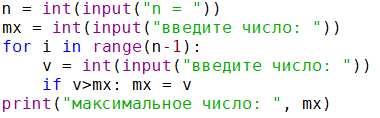
Пример 5.1

В данном примере вычисляется сумма чисел от 1 до n. Значение n вводится.



Пример 5.2

В данном примере ищется максимальное значение среди n введенных чисел. Как и в предыдущем случае значение n вводится.



Обратите внимание, что переменная цикла нам нужна только для подсчета количества итераций. В теле цикла она не используется. Первое введенное число обрабатывается не в цикле, оно используется для начального присваивания, поэтому нам нужно, чтобы цикл выполнился n-1 раз. Функция range() сгенерирует числа от 0 до n-2, то есть n-1 число.

***Задания для самостоятельной работы***

1. Напишите программу, которая для введенного n вычисляет значение n! (не использовать функции Python).
2. Напишите программу, которая вводит с клавиатуры 5 чисел и вычисляет их сумму, произведение и среднее значение.
3. Имеется 2 банка. Первый банк ежемесячно начисляет 1.1% от первоначально вложенной суммы (простые проценты). Второй банк ежемесячно начисляет 1% от накопленной к этому моменту суммы (сложные проценты). В каждый из банков в месяц 0 (начальный момент) положили 100 000 руб.

Напишите программу, которая выводит на экран в виде таблицы накопленные в этих банках суммы для 24 месяцев. Определите в программе и выведите на экран в какой банк выгоднее вложить деньги на 1 год и на 2 года.

1. С клавиатуры вводятся n целых чисел. Найти среди них наименьшее нечетное число.
2. С клавиатуры вводятся n целых чисел. Найти минимальное положительное и максимальное отрицательное числа.
3. Напишите программу, которая вычисляет значение функции sin(x), используя ее разложение в ряд:

Чтобы избежать вложенных циклов и сделать вычисления более эффективными, представим ряд в виде:

Очевидно, что . Для остальных мы получили рекуррентное соотношение, которое позволяет вычислить текущее значение через предыдущее.

У нас ряд бесконечный. На практике вычисления прекращают, когда очередное слагаемое по модулю меньше заданного маленького числа, например, .

Выведите на экран полученное значение и для сравнения выведите значение, которое возвращает стандартная функция math.sin(x).

1. Уточнить корень уравнения на отрезке [1;2] с точностью , используя метод деления отрезка пополам.
2. Вычислите значение интеграла для функции по формуле прямоугольников:

, где .

Значение n может меняться следующим образом. Сначала задайте n равным любому числу, например, n=10. Вычислите для заданного n значение y. Увеличьте значение n в 2 раза и вновь вычислите y. Если полученное значение по модулю отличается от предыдущего меньше чем на заданное маленькое число (), то вычисления можно прекратить. Если нет, то вновь увеличиваем n и сравниваем.

# 6. Списки

Большинство программ работает не с отдельными элементами данных, а с наборами данных. Для хранения таких данных в Питоне можно использовать структуру данных список.

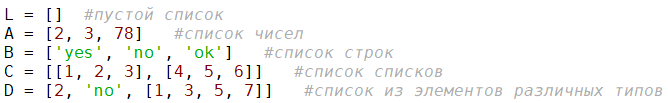
Список – упорядоченный по местоположению набор объектов произвольных типов (последовательность объектов). Позиция элемента в списке задается индексом. Первый элемент списка имеет индекс, равный нулю.

Список – изменяемый тип данных. Поэтому можно не только получить элемент по индексу, но и изменить его.

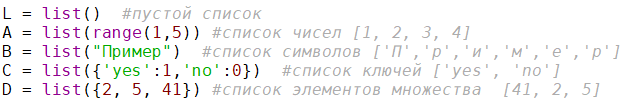
***Создание списков***

Создать список можно несколькими способами:

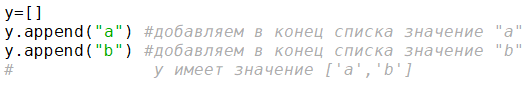
* явно указав все элементы списка в тексте. Элементы списка перечисляются в квадратных скобках через запятую. Они могут быть разных типов:



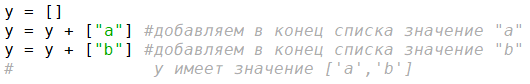
* с помощью функции list(). Функция позволяет преобразовать в список данные других типов:



* заполнив список поэлементно:

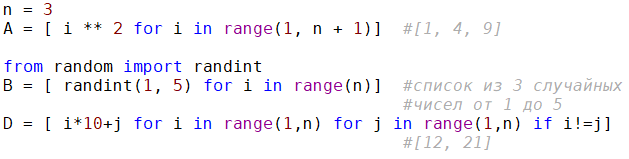


Метод append() добавляет в конец списка указанное значение. Метод изменяет текущий список.

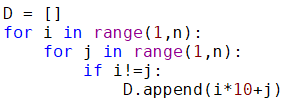


Оператор + объединяет два списка в новый список. В результате получается новый список.

* с помощью генератора списка:



Вычисление списка D с помощью генератора эквивалентно следующему коду:



Использование генератора позволяет не только написать более компактную программу, но и работать она будет быстрее, чем эквивалентные циклы for.

***Операции над списками***

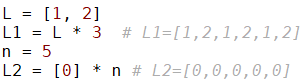
* конкатенация ( + )

Операция позволяет объединить два списка в один. Результатом будет новый список:



* повторение ( \* )

Повторяет список указанное количество раз:



Как видно из примера, оператор удобно использовать для создания списков нужной длины. Учтите, что при этом выполняется поверхностное копирование элементов, то есть копируются только ссылки на значения.

* проверка на вхождение ( in, not in )

Оператор in проверяет наличие значения в списке, оператор not in – отсутствие значения. Возвращают значения True или False:



* доступ по индексу ( [] )

Выражение L[i] позволяет получить значение c номером i из списка L. Элементы списка нумеруются с нуля. Можно использовать отрицательные индексы: последний элемент имеет индекс -1, предпоследний – индекс -2 и так далее.

Списки – изменяемый тип данных. Поэтому присвоить значение элементу списка L с индексом i можно с помощью оператора L[i] = x.

* получение среза

Операция позволяет выделить из списка фрагмент.

В общем случае срез задается тремя целыми числами:

«начало» : «конец» : «шаг»

В фрагмент входят элементы, чьи индексы j удовлетворяют условию «начало» ≤ j < «конец», то есть символ с номером «конец» не входит в фрагмент.

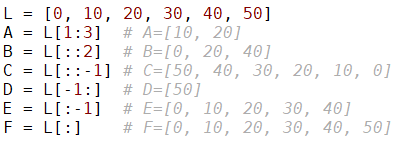
Все параметры являются необязательными и могут быть опущены.

Параметр «шаг» по умолчанию равен 1. Если «шаг» не указывается, то второе двоеточие тоже можно не писать. Если «шаг» ≠ 1, то из указанного диапазона индексов выбираются индексы с номерами «начало», «начало» + «шаг», «начало» + 2\*«шаг» и т.д. Шаг может быть отрицательным, тогда «начало» ≥ j > «конец».

Если параметр «начало» не указан, то используется индекс первого элемента для положительного шага, и индекс последнего элемента – для отрицательного.

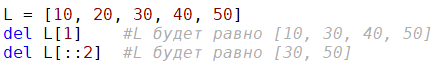
Если параметр «конец» не указан, то возвращается фрагмент до конца списка.

Примеры срезов:



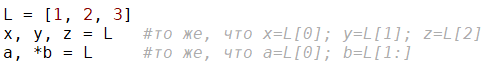
* удаление ( del )

Оператор удалит из списка элемент с заданным индексом или все элементы, попавшие в срез:



* позиционное присваивание

С помощью позиционного присваивания можно присвоить значения элементов списка каким-либо переменным. Количество элементов слева и справа от оператора присваивания должно совпадать. Перед одной переменной слева от = можно указать звездочку. В этой переменной будет сохранен список из всех «лишних» элементов:

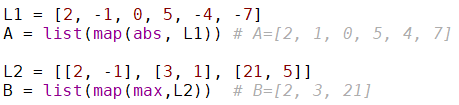


***Функции для работы со списками***

**list()** – преобразует данные различных типов в список (примеры см. в разделе создание списков);

**len()** – возвращает длину списка (количество элементов);

**map()** – позволяет применить заданную функцию к каждому элементу списка. Возвращает объект, который можно преобразовать в список, но не список.



***Преобразование списка в строку***

Для преобразования списка в строку можно использовать функцию str() и метод join().

Функция **str()** возвращает строку, которая соответствует стандартному представлению списка: элементы внутри квадратных скобок через запятую:



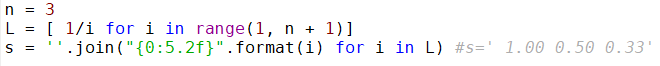
Метод **join()** создает строку, в которой элементы исходного списка размещены через указанный разделитель. Элементы исходного списка должны быть строками, иначе возникнет ошибка.



Если элементы исходного списка не являются строками, то их сначала нужно преобразовать в строку. Можно использовать функцию map():



Ниже показано как можно создать строку числовых значений с использованием форматирования:



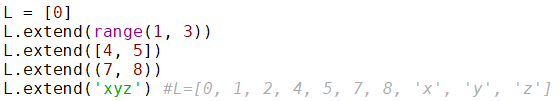
***Методы списков***

Здесь рассматриваются только некоторые методы списков. Описание всех методов следует смотреть в справочной системе.

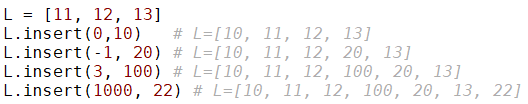
**append( )** – добавляет один объект в конец списка. Метод изменяет текущий список и ничего не возвращает. Пример:



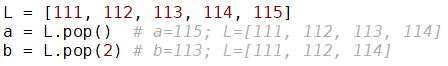
**extend( )** – добавляет элементы последовательности в конец списка. Последовательностями являются списки, кортежи, строки и некоторые другие объекты. Метод изменяет текущий список и ничего не возвращает. Пример:



**insert( )** – добавляет один элемент перед элементов в указанной позиции. Метод изменяет текущий список и ничего не возвращает. Пример:



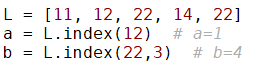
**pop( )** – удаляет элемент с указанным индексом и возвращает его. Если индекс не указан, то удаляет и возвращает последний элемент. Если элемента с заданным индексом нет или список пустой, то возникнет ошибка. Пример:



**remove( )** – удаляет первый элемент с указанным значением. Метод изменяет текущий список и ничего не возвращает. Если элемента нет в списке, то возникнет ошибка. Пример:



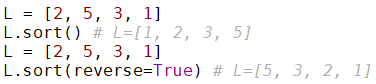
**index( )** – возвращает индекс элемента, имеющего указанное значение. Второй и третий параметры определяют часть списка, в которой выполняется поиск. Если значение не найдено, возникнет ошибка. Пример:



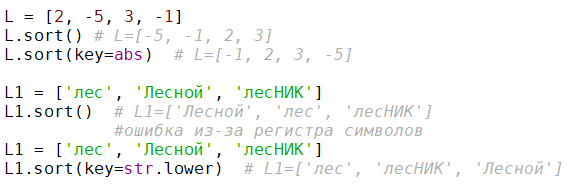
**count( )** – возвращает количество элементов с указанным значением. Пример:



**sort( )** – сортирует список. Метод изменяет текущий список и ничего не возвращает. Имеет 2 необязательных параметра. Параметр reverse определяет порядок сортировки (по возрастанию или по убыванию). По умолчанию список сортируется по возрастанию, так как reverse = False. Если установить reverse=True, то список будет сортироваться по убыванию. Пример:



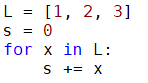
В параметре key указывается имя функции, которая будет применена к каждому элементу списка перед сортировкой. Пример:



***Перебор элементов списка***

Для обработки всех элементов списка обычно используется цикл for.

Если вам не нужно изменять элементы списка, например, при вычислении суммы элементов, то самым удобным будет вариант цикла по значениям списка:



Переменная x в данном случае по очереди будет равна каждому элементу списка. После завершения цикла s будет содержать значение суммы.

Используем такой же цикл, чтобы увеличить все элементы списка L на единицу:



Если напечатать значение L, то выяснится, что его элементы не изменились. Дело в том, что мы меняли x, а не список L. Изменить элемент списка можно только, используя его индекс. Поэтому следует использовать другой вариант цикла (по индексам):



***Задания для самостоятельной работы***

1. Сформировать и вывести на экран список из 5 случайных целых чисел от -10 до 10. Для созданного списка:

* вычислить среднее арифметическое значение элементов списка;
* переставить элементы списка в обратном порядке;
* создать новый список из отрицательных элементов исходного списка. В исходном списке удалить отрицательные элементы. Порядок элементов должен быть сохранен.

1. Имеются 2 списка целых чисел, упорядоченные по возрастанию. Получите новый список, содержащий все элементы исходных списков, в котором элементы также упорядочены в порядке возрастания, не используя сортировку.
2. Треугольник задан в виде списка координат вершин (вершина представляется в виде списка из двух чисел). Проверьте, действительно ли заданные точки определяют треугольник, вычислите длины сторон, площадь и периметр треугольника.

***Многомерные списки***

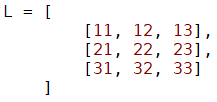
Элементом списка может быть любой объект: число, строка, список, кортеж, словарь, множество и т. д. Списки, которые являются элементами других списков, называют вложенными. Их элементами в свою очередь могут быть списки. Уровень вложенности может быть любым. Такие структуры данных часто используются для хранения данных таблиц или матриц.

Никаких специальных средств языка для таких списков нет, но есть некоторые особенности и приемы работы, которые лучше изучать на примерах.

Создать вложенный список можно как обычно:



Но гораздо нагляднее задать этот список в программе так:

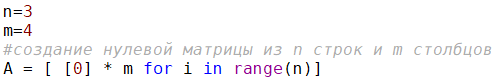


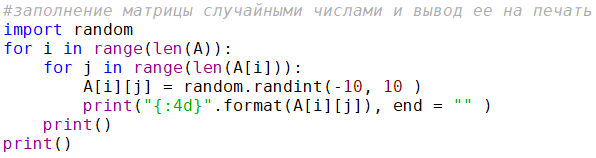
Это возможно благодаря тому, что выражения внутри скобок (любых, в том числе квадратных) может располагаться на нескольких строках.

Чтобы получить доступ к элементу вложенного списка, нужно указать в нашем случае два индекса. Например, к значению 23 мы можем получить доступ с помощью выражения L [1] [2], так как элементы списков нумеруются с нуля. Если мы сохранили каждую строку матрицы в виде списка, то первый индекс определяет номер строки, а второй – номер столбца матрицы (нумерация с нуля, а не как привыкли в математике с единицы). Каждый индекс заключается в свои скобки. Имеет смысл и выражение L[1]. Это элемент списка L, который является списком, и в котором мы храним строку матрицы. Поэтому, если нам нужно что-то выполнить со строкой (добавить строку, удалить строку), то мы работаем с индексами списка L. Если же нужно что-то сделать с элементами строки i, то работаем с индексами списка L[i]. В общем случае количество индексов может быть любым.

При обработке многомерных списков часто приходится использовать вложенные циклы, то есть циклы, которые выполняются в теле другого цикла.

Примеры типовых действий для вложенных списков:













***Задания для самостоятельной работы***

1. Создать матрицу из n строк и m столбцов (n=5, m=6). Заполнить матрицу случайными целыми числами от -10 до 10. Вывести матрицу на экран. Для созданной матрицы:

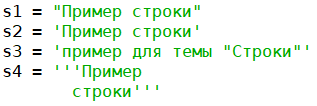
* вычислить количество строк, содержащих нули;
* проверить, есть ли в исходной матрице одинаковые строки;
* вычислить транспонированную матрицу;
* удалить строку и столбец с заданными номерами. Номера строки и столбца вводятся;
* вставить строку и столбец с заданными номерами. Номера вводятся. Строку и столбец заполнить нулями.

1. Результат сессии, состоящей из 3 экзаменов (История, Алгебра, Информатика), для студента задается в виде списка, содержащего фамилию студента и 3 оценки по пятибалльной системе (0-неявка, 2-неудовл., 3-удовл., 4-хорошо, 5-отлично). Результаты группы сохраняются в виде списка списков указанного вида. Для группы выведите на экран:

* таблицу с результатами экзаменов;
* фамилии студентов, имеющих задолженности, и названия несданных ими предметов;
* средний балл по каждой дисциплин;
* количество неявок, неудовлетворительных, удовлетворительных, хороших и отличных оценок по дисциплине Информатика.

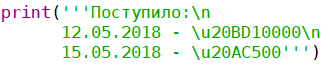
# 7. Строки

Строка (объект класса str) – это последовательность символов. В тексте программы этот набор символов нужно заключить в кавычки, апострофы или тройные кавычки (''' или """). Например,

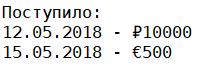


Последовательности символов в кавычках или апострофах должны располагаться в одной строке программы (нельзя переносить). Последовательности в тройных кавычках могут занимать любое количество строк. Никакой другой разницы между строками с разными типами кавычек нет, но тип кавычек в начале и конце должен совпадать. Длина строки может быть любой.

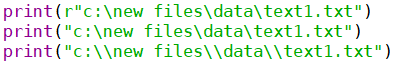
В строках символ обратного слэша «\» используется для задания специальных символов, которые сложно ввести с клавиатуры (управляющие символы, непечатаемые символы, псевдографические символы, символы национальных алфавитов и т.п.). Такие специальные символы называют экранированными последовательностями. Наиболее часто используются последовательности: \n – перевод строки, \t – табуляция, \\ - обратный слэш, \' – апостроф, \" – кавычки, \uxxxx – 16-битный символ Unicode. Например, в результате выполнения инструкции



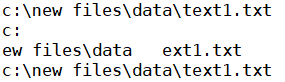
на экран будет выведено:



Если перед открывающей кавычкой стоит символ «r» или «R:», то механизм экранирования отключается, то есть символ «\» в такой строке не имеет специального смысла. Пример:



На экран будет выведено



Строки являются неизменяемыми объектами. Поэтому нельзя изменить часть существующей строки, но можно, используя существующую строку, создать новую, в которой оставить все, что нужно, из исходной строки и изменить все, что не нужно. Для этого используются операции над строками и функции.

Строки, как и списки, являются последовательностями. Поэтому многие операции у них похожи.

***Операции над строками***

* конкатенация ( + )

Операция позволяет объединить две строки в одну. Результатом будет новая строка:



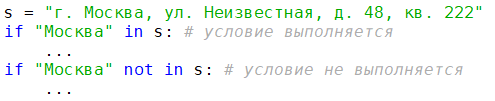
* повторение ( \* )

Повторяет строку указанное количество раз:



* проверка на вхождение ( in, not in )

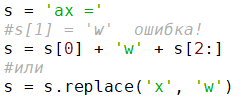
Оператор in проверяет наличие значения в строке, оператор not in – отсутствие значения. Возвращают значения True или False:



* доступ по индексу ( [] )

Выражение S[i] позволяет получить i-й символ строки. Как и в случае списков символы строки нумеруются с нуля. Возможна отрицательная нумерация с конца строки.

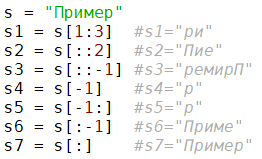
Строки – неизменяемый тип данных, поэтому изменить значение символа строки с помощью оператора присваивания, например, s[1] = "w", нельзя. Это ошибка. Вместо этого следует использовать метод replace() или комбинацию операций конкатенации и срезов:



В обоих случаях создается новая строка, которая присваивается той же переменной s.

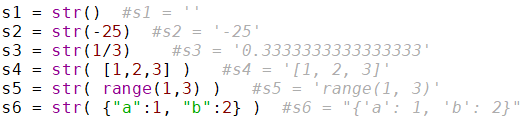
* получение среза

Операция позволяет выделить из строки фрагмент. Срез определяется точно так же, как для списков. Примеры:



***Функции для работы со строками***

**str()** – создает строку из объекта другого типа:



Обратите внимание на значение строки s5. Функция range() создает не список, как многие думают, а специальный объект, который в виде строки выглядит так. Если нужен именно список, то полученное значение нужно дополнительно преобразовать в список.

**list(), int(), float()** – преобразуют строку в объект соответствующего типа (список символов, целое или вещественное число);

**len()** – возвращает длину строки (количество элементов);

**chr()** – возвращает строку, состоящую из символа с заданным кодом;



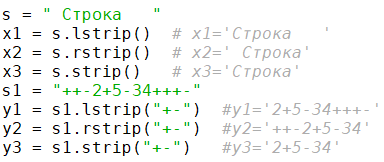
**ord()** – для заданного символа возвращает его код.



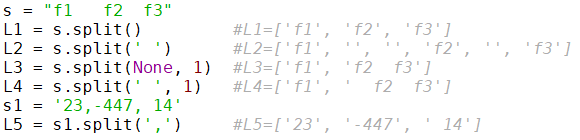
***Методы строк***

Здесь рассматриваются только основные методы строк. Описание всех методов следует смотреть в справочной системе.

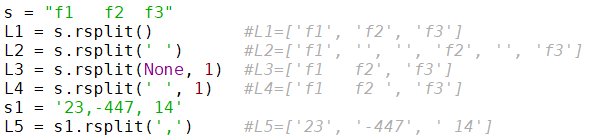
**lstrip( ), rstrip( ), strip( )** – создают копию строки, в которой удалены пробельные или указанные символы в начале, в конце или и в начале и в конце строки соответственно. Пробельными символами считаются: пробел, символ перевода строки (\n), символ возврата каретки (\r), символы горизонтальной (\t) и вертикальной (\v) табуляции.



**split()** – разделяет строку на подстроки по указанному разделителю и возвращает созданный из них список. Если разделитель не указан или имеет значение None, то в качестве разделителя используется пробел, причем несколько подряд идущих пробелов считаются за один пробел. Если во втором параметре указано число, то в списке будет указанное количество подстрок. Если подстрок больше, то будет еще один элемент, содержащий остаток строки. Примеры:



**rsplit()** аналогичен split(), но поиск символа-разделителя выполняется справа налево.



**upper()** – делает все символы строки прописными. Пример:



**lower()** – делает все символы строки строчными. Пример:



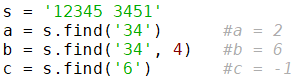
**title()** – делает первую букву каждого слова прописной. Пример:



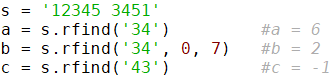
**capitalize()** –делает первую букву строки прописной. Пример:



**find()** – поиск строки-образца в исходной строке. Образец для поиска указывается в первом параметре функции. Необязательные второй и третий параметры определяют срез, в котором производится поиск. Метод возвращает позицию первого вхождения образца в строку. Если образец в строку не входит, то возвращает -1. Пример:



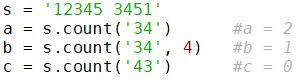
**rfind()** – поиск строки-образца в строке. Аналог find(), но возвращает позицию последнего вхождения образца или -1. Пример:



**index()** – метод аналогичен методу find(), но если образец в строку не входит, то возникает ошибка.

**rindex()** – метод аналогичен методу rfind(), но если образец в строку не входит, то возникает ошибка.

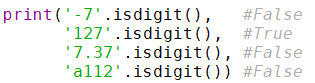
**count()** – возвращает число вхождений строки-образца в исходную строку. Если образец в строку не входит, то возвращает 0. Второй и третий параметр определяют срез для поиска (необязательные параметры). Пример:



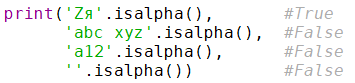
**replace()** – создает новую строку, в которой фрагмент исходной строки, указанный в первом аргументе, заменяется на строку, указанную во втором аргументе. Третий аргумент определяет количество замен. По умолчанию заменяются все вхождения.



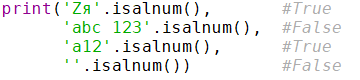
**isdigit()** – возвращает True, если строка содержит только цифры, в противном случае – False:



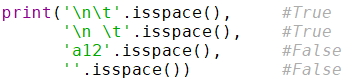
**isalpha()** – возвращает True, если строка содержит только буквы, в противном случае – False:



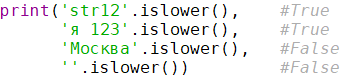
**isalnum()** – возвращает True, если строка содержит только буквы и (или) цифры, в противном случае – False:



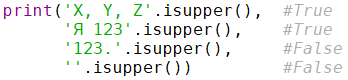
**isspace()** – возвращает True, если строка содержит только пробельные символы, в противном случае – False:



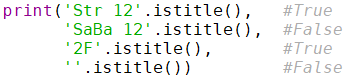
**islower()** – возвращает True, если строка содержит буквы, и все они в нижнем регистре (могут быть и другие символы), в противном случае – False:



**isupper()** – возвращает True, если строка содержит буквы, и все они в верхнем регистре (могут быть и другие символы), в противном случае – False:



**istitle()** – возвращает True, если строка содержит буквы, и первые буквы всех слов являются заглавными, в противном случае – False:

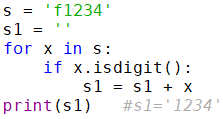


***Перебор элементов строки***

Для перебора всех символов строки удобно использовать цикл for. Рассмотрим использование этого цикла на примере.

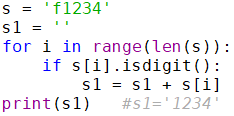
Пример 7.1

Для имеющейся строки необходимо получить строку, содержащую все цифры, которые встречаются в этой строке.



Переменная x содержит по очереди значения всех символов строки. Индексы в данном случае не используются.

С использованием индексов эту же задачу можно решить так:



Предпочтительнее использовать первый вариант, так как он компактнее и эффективнее.

***Задания для самостоятельной работы***

1. Для введенной строки выведите (на отдельной строке):

* второй символ этой строки;
* предпоследний символ этой строки;
* первые 3 символа этой строки;
* всю строку, кроме последних двух символов;
* все символы с четными индексами (считая, что индексация начинается с 0, поэтому символы выводятся начиная с первого);
* все символы с нечетными индексами, то есть начиная со второго символа строки;
* все символы в обратном порядке;
* все символы строки через один в обратном порядке, начиная с последнего;
* длину данной строки.

1. Не используя метод count, для заданной строки выполните:

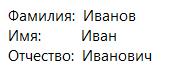
* если символ \* в данной строке отсутствует, выведите текст «нет символа»;
* если символ \* встречается в строке только один раз, выведите его индекс;
* если символ \* встречается два и более раз, выведите индекс его первого, второго и последнего вхождения, удалите первый и последний символ \* из строки.

1. Дана строка, состоящая из слов, разделенных пробелами. В этой строке:

* удалите все лишние пробелы (в начале, в конце, между словами оставить ровно один пробел);
* поменяйте регистр символов (строчные сделать прописными, прописные – строчными);
* определите, сколько в ней слов.

1. Строка содержит фамилию, имя и отчество, записанные через пробелы. Например «   Иванов Иван Иванович». Для этой строки:

* выведите информацию на экран в виде



* получите строки вида «Иванов И.И.» и «И.И. Иванов»

1. Выведите значение заданного целого числа от 0 до 999 прописью. Например, «сто девяносто один» для числа 191, «одиннадцать» для числа 11.
2. Вычислите значение выражения, которое состоит из целых чисел и знаков «+» и «-». Выражение вводится как символьная строка.

# 8. Словари

Словарь – набор объектов, каждый из которых является парой вида ***ключ : значение***. Ключи используются для доступа к значениям элементов словаря так же, как индексы для доступа к элементам списка. В качестве ключа чаще всего используется строка, но можно использовать и любые другие неизменяемые объекты, например, числа или кортежи. Значения могут быть любого типа. Все ключи в словаре уникальные. Порядок элементов в словаре не определен, поэтому они не являются последовательностями. В терминологии Python словари являются отображениями.

Словари удобны для хранения объектов, имеющих именованные поля.

Главное достоинство словарей – быстрый поиск среди неупорядоченных данных.

***Создание словаря***

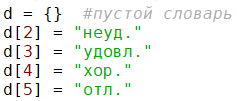
Создать словарь можно различными способами:

* явно указав все элементы словаря в тексте. Элементы словаря перечисляются в фигурных скобках через запятую. Для каждого элемента указываются ключ и значение, которые друг от друга отделяются двоеточием.





* заполнив словарь поэлементно:

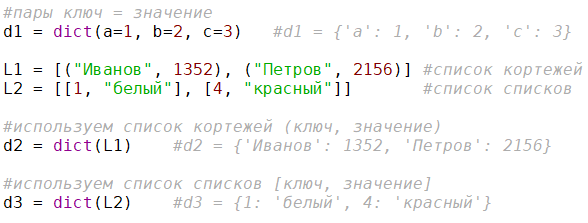


В правой части оператора присваивания указывается значение элемента. В левой – имя словаря и в квадратных скобках ключ. В данном случае в качестве ключей используются числа 2, 3, 4, 5. Словарь d будет иметь значение:



* с помощью функции dict().

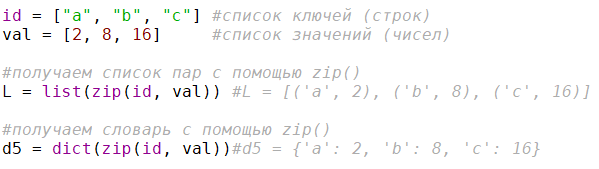
Функция обычно используется для создания словаря из данных других типов. В этом случае данные, как правило, являются последовательностью пар значений. Один элемент пары становится ключом, другой – значением элемента словаря:



Функция может применяться и для создания поверхностной копии словаря.



Если данных в виде пар нет, то их можно создать, используя функцию zip(). Вообще-то zip() умеет создавать не только пары, но и тройки, четверки и т.д. значений, но в данном случае нам нужны именно пары. Функция zip() возвращает последовательность, которую можно преобразовать в список или словарь:

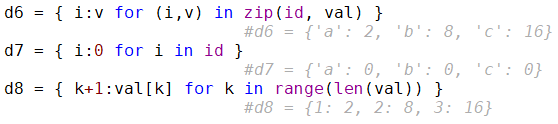


* с помощью генератора.

Генераторы словарей отличаются от генераторов списков тем, что:

* выражение заключается в фигурные скобки, а не в квадратные;
* внутри фигурных скобок перед for указываются два значения через двоеточие (слева от двоеточия ключ, справа – значение).

Примеры:



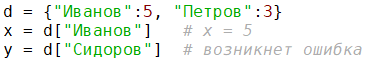
Как и в случае списков генераторы словарей могут содержать несколько вложенных циклов for и инструкцию ветвления if после циклов:



***Операции над словарями***

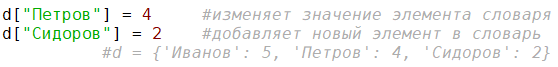
* доступ по ключу ( [] )

Для получения значения элемента словаря с заданным ключом нужно указать имя словаря и значение ключа в квадратных скобках. Если в словаре нет элемента с таким ключом, возникнет ошибка:



Чтобы подобные ошибки не возникали, можно либо вначале проверять наличие элемента в словаре, либо использовать метод get(), либо перехватывать эти ошибки (обрабатывать исключение KeyError).

Словарь – изменяемый тип данных, поэтому можно не только получить значение по ключу, но и изменить его. Для этого имя словаря и ключ должны быть указаны в левой части оператора присваивания. В правой части указывается новое значение элемента. Если элемента с нужным ключом в словаре нет, то он создается. Этим часто пользуются, чтобы добавлять элементы в словарь. Пример:



* проверка наличия элемента с заданным ключом ( in )

Оператор in проверяет наличие элемента с указанным ключом в словаре. Возвращает значения True или False:



* удаление ( del )

Удаляет из словаря элемент с заданным ключом. Если элемент с таким ключом отсутствует, то возникает ошибка (возбуждается исключение KeyError). Пример:



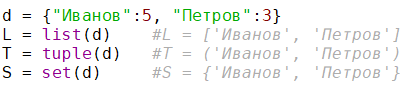
Словари не являются последовательностями, поэтому операции получения среза, конкатенации, повторения для них недопустимы.

Многие действия над словарями выполняются с помощью функций и методов.

***Функции для работы со словарями***

**dict()** – создает словарь (примеры приведены выше при описании способов создания словарей);

**list(), tuple(), set()** – преобразуют словарь в объект соответствующего типа (список, кортеж или множество ключей словаря):



**len()** – возвращает количество элементов в словаре.

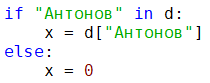
***Методы словарей***

Здесь рассматриваются только основные методы словарей. Полное описание следует смотреть в справочной литературе.

**get()** – возвращает значение элемента словаря. Ключ указывается в первом параметре метода. Если в словаре нет такого ключа, то возвращается значение второго (необязательного) параметра. По умолчанию значение второго параметра равно None. В отличие от операции [] здесь ошибка не возникает даже при отсутствии элемента в словаре. Например, инструкция



делает то же самое, что



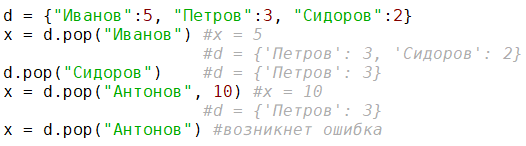
Другие примеры:



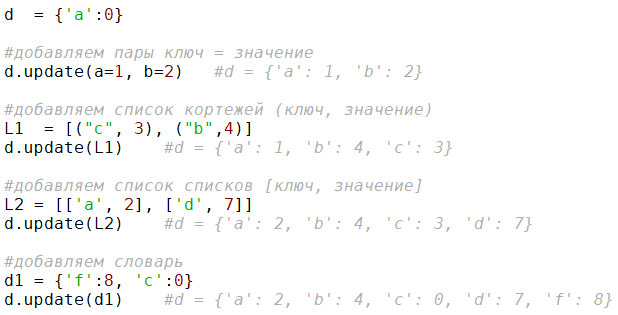
**clear()** – удаляет все элементы словаря. Метод ничего не возвращает:



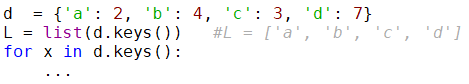
**pop()** – удаляет элемент с ключом, указанном в первом параметре, и возвращает значение, соответствующее этому ключу. Если ключ отсутствует, то возвращается значение второго (необязательного) параметра. Если второй параметр не задан, то возникнет ошибка (исключение KeyError):



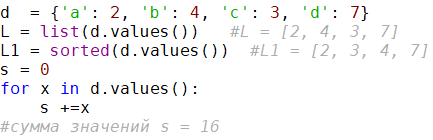
**update()** – добавляет элементы в словарь. Метод изменяет текущий словарь и ничего не возвращает. Параметры могут быть такими же, как у функции dict(). Если элемент с указанным ключом уже есть в словаре, то его значение заменяется на новое:



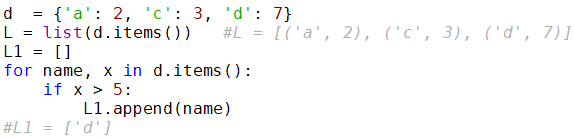
**keys()** – возвращает объект, содержащий все ключи словаря. Этот объект поддерживает итерации (можно использовать в цикле for) и операции над множествами:



**values()** – возвращает объект, содержащий все значения словаря. Объект поддерживает итерации (можно использовать в цикле for):



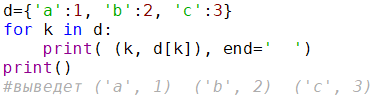
**items()** – возвращает объект, содержащий все ключи и значения в виде кортежей. Объект поддерживает итерации (можно использовать в цикле for):



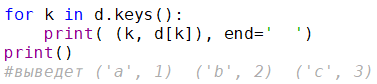
***Перебор элементов словаря***

Перебор всех элементов словаря выполняется с помощью цикла for.

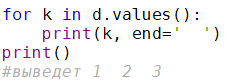
Если в заголовке цикла указать просто имя словаря, то переменная цикла на каждой итерации будет равна ключу очередного элемента словаря. Доступ к значению при необходимости выполняется стандартным образом. Учтите, что порядок элементов в словарях не определен, поэтому ключи могут выдаваться совсем не в том порядке, в котором элементы добавлялись в словарь:



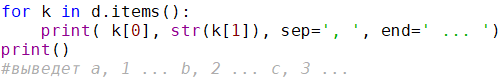
Если использовать в заголовке цикла метод keys(), то результат будет точно такой же:



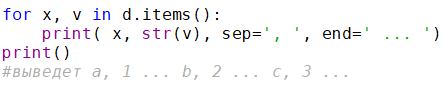
Если использовать в заголовке цикла метод values(), то переменная цикла на каждой итерации будет равна значению очередного элемента словаря. Ключи в таком варианте цикла будут недоступны. Пример:



Если использовать в заголовке цикла метод items(), то значением переменной цикла k будет кортеж из двух элементов. Тогда k[0] – это ключ очередного элемента словаря, а k[1] – значение этого элемента:



Чтобы не использовать индексы кортежа, удобно его распаковать:

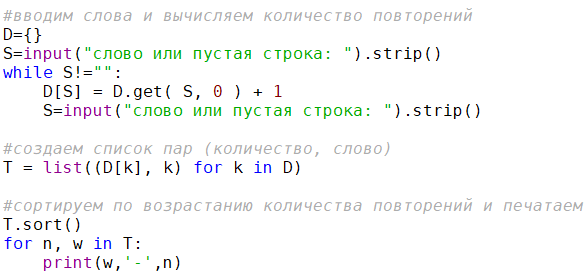


***Пример использования словаря***

Для введенных слов (каждое слово вводится отдельно, признак конца ввода – пустая строка) вычислить, сколько раз они вводились. Вывести слова и полученные значения на экран в порядке возрастания частоты встречаемости слов. Для простоты под словом будем понимать любую последовательность символов.

Для хранения информации используем словарь, в котором ключами будут введенные слова, а значениями – количество повторений слова.

Чтобы вывести информацию на экран, словарь сначала преобразуем в список пар (кортежей из двух элементов). Список отсортируем, так как словари сортировать нельзя. Используя полученный отсортированный список, выведем на экран информацию в нужном виде. Обратите внимание на элементы списка: ключ – второй элемент кортежа, значение – первый элемент кортежа. При сортировке пары сортируются сначала по первому элементу, т.е. по числу вхождений слова.



***Задания для самостоятельной работы***

1. Дана строка, состоящая из слов, разделенных пробелами. Выведите все слова, встречающиеся в этой строке, на экран. Рядом выведите, сколько раз встречается слово в исходной строке. Слова должны быть отсортированы по убыванию их количества появления в тексте.
2. Имеется строка с названиями товаров вида «яблоки, груши, яблоки, киви, сливы, киви». Товары перечислены через запятую, товары могут повторяться. Выведите название товара, который встречается в этой строке чаще всего. Если таких товаров несколько, то названия всех товаров.
3. Число от 2 до 55, которое может содержать только цифры 2, 3, 4, 5 записано прописью, например, «тридцать три». Вычислите квадратный корень из этого числа.
4. Имеется список названий месяцев: [‘января’, ‘февраля’, ‘марта’, ‘апреля’, ’мая’, ‘июня’, ‘июля’, ‘августа’, ‘сентября’, ‘октября’, ‘ноября’, ‘декабря’]. Создайте по этому списку словарь, в котором название месяца будет ключом, а номер месяца (от 1 до 12) – значением.

Используя полученный словарь преобразуйте строку с датой вида «1 января 2016» в строку «1.01.2016»

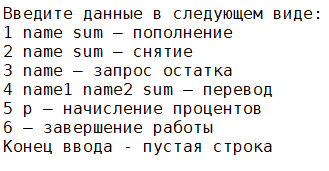
1. При работе со счетами клиентов поддерживаются следующие операции:

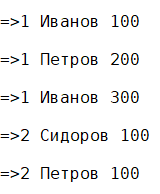
* 1 – пополнение счета клиента;
* 2 – снятие денег со счета;
* 3 – запрос остатка средств на счете;
* 4 – перевод денег между счетами клиентов;
* 5 – начисление процентов всем клиентам;
* 6 – завершение работы с программой.

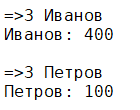
Во время работы программы информация об операциях вводится в виде:

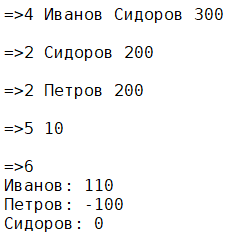
* 1 name sum – зачислить сумму sum на счет клиента name. Если у клиента нет счета, то счет создается.
* 2 name sum – снять сумму sum со счета клиента name. Если у клиента нет счета, то счет создается.
* 3 name – узнать остаток средств на счету клиента name.
* 4 name1 name2 sum - перевести сумму sum со счета клиента name1 на счет клиента name2. Если у какого-либо клиента нет счета, то ему создается счет.
* 5 p – начислить всем клиентам, у которых открыты счета, p% от суммы счета. Проценты начисляются только клиентам с положительным остатком на счете. Если у клиента остаток отрицательный, то его счет не меняется. После начисления процентов сумма на счету остается целой, то есть начисляется только целое число денежных единиц. Дробная часть начисленных процентов отбрасывается.
* 6 – завершить работу программы. Вывести на экран информацию о всех счетах.

Если символ => использовать как приглашение для ввода в функции input(), то результат выполнения программы должен выглядеть примерно так:







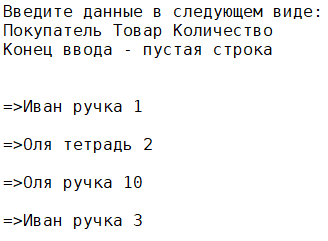


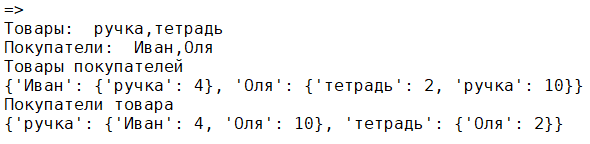
1. Информация о продажах вводится в виде строки «Покупатель Товар Количество», где Покупатель — имя покупателя (строка без пробелов), Товар — название товара (строка без пробелов), Количество — количество приобретенных единиц товара (целое число). Конец ввода – пустая строка.

Задание:

* выведите на экран списки покупателей и товаров;
* для каждого покупателя подсчитайте количество приобретенных им единиц каждого вида товаров;
* для каждого товара получите информацию о том, кто и сколько его купил.

Результат выполнения программы может выглядеть примерно так (как и в предыдущем задании => – приглашение для ввода):





# 9. Множества

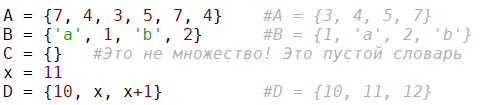
Множество (set) – неупорядоченный набор уникальных элементов.

Множество может содержать только элементы неизменяемых типов (числа, строки, кортежи). Элементы могут быть разных типов.

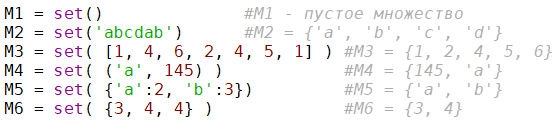
***Создание множества***

Создать множество можно несколькими способами:

* явно указав все элементы в программе. Элементы множества перечисляются в фигурных скобках через запятую. Можно использовать переменные. При перечислении значения могут повторяться, но в множестве будет сохранен только один экземпляр:



* с помощью функции set(). Функция позволяет создать пустое множество или преобразовать в множество данные других типов:



Учтите, что если в исходных данных имелись совпадающие значения, то в множестве всегда остаются только уникальные элементы. Этим эффектом можно воспользоваться для удаления повторяющихся значений из списка или кортежа:



* с помощью генератора множества.

Синтаксис генераторов множеств похож на синтаксис генераторов списков, но выражение заключается в фигурные скобки. Повторяющиеся элементы в множестве не сохраняются. Пример:

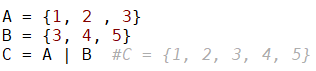


* из других множеств, используя операции над множествами, а также функции и методы множеств.

***Операции над множествами***

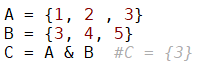
* объединение множеств ( | )

Результатом A | B является множество, содержащее все элементы A и все элементы В:



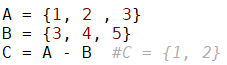
* пересечение множеств ( & )

Результатом A & B является множество, содержащее все элементы, которые одновременно входят в A и В:



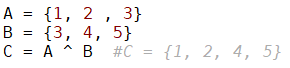
* разность множеств ( - )

Результатом A - B является множество, содержащее все элементы A, которые не входят в В:

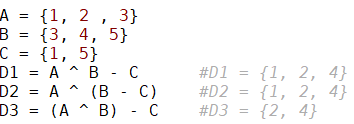


* симметричная разность множеств ( ^ )

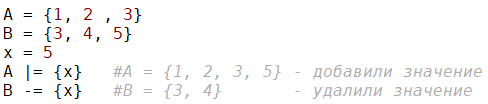
Результатом A ^ B является множество, содержащее те элементы A и В, которые не входят в пересечение этих множеств:



Используя эти операции, из множеств можно составлять выражения. Приоритет операций в порядке убывания: -, &, ^, |. При необходимости, как обычно, воспользуйтесь скобками:



Можно использовать комбинированные операторы присваивания: -=, &=, ^=, |=. Пример:



Следующая группа операций предназначена для выполнения различных проверок. Значением выражения являются значения True или False в зависимости от результатов проверки. Обычно соответствующие выражения используются в инструкциях if или while. В приведенных ниже примерах предполагается, что A = {2, 3, 4, 5}.

* проверка на наличие элемента в множестве ( in, not in ):



* проверка на равенство ( ==, != ):



* проверка, является ли одно множество подмножеством другого ( <, <=, >, >= ):





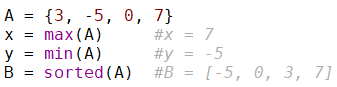
***Функции для работы с множествами***

**set()** – преобразует данные различных типов в множество (примеры см. в разделе создание множеств);

**len()** – возвращает количество элементов множества;

**max(), min()** – находят максимальное и минимальное значение элементов множества;

**sorted( )** – сортирует элементы множества. Возвращает список с отсортированными значениями. Метода sort() у множеств нет, так как множество это не последовательность и порядок элементов в множестве не определен. Примеры:



***Методы множеств***

Ниже приведены только основные методы множеств. Описание всех методов следует смотреть в справочной литературе. В частности, методы, соответствующие операциям -, &, ^, |, -=, &=, ^=, |= , здесь не описаны.

**add( )** – добавляет объект, указанный в качестве параметра, в множество. Метод изменяет текущее множество и ничего не возвращает. Пример:



**remove( )** – удаляет объект, указанный в качестве параметра, из множества. Метод изменяет текущее множество и ничего не возвращает. Если объект не найден, то возникнет ошибка (исключение KeyError). Пример:



**discard( )** – удаляет объект, указанный в качестве параметра, из множества. Метод изменяет текущее множество и ничего не возвращает. Если объект не найден, то ничего не делает. Пример:



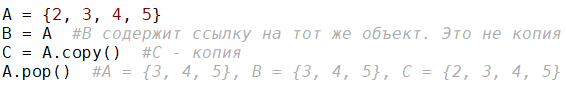
**pop( )** – удаляет произвольный элемент из множества и возвращает его. Если элементов нет, то возникнет ошибка (исключение KeyError). Пример:



**clear( )** – удаляет все элементы из множества. Метод изменяет текущее множество и ничего не возвращает:

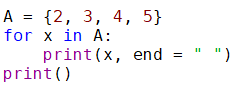


**copy( )** – создает копию множества. Пример:



***Перебор элементов множества***

Для обработки всех элементов множества используется цикл for:



***Задания для самостоятельной работы***

1. Создайте два множества A и B, добавив в A 10 случайных целых чисел от 1 до 20, а в B – 5 таких же чисел. Для созданных множеств:

* найдите их объединение, пересечение, разность, множество элементов, которые не входят в пересечение;
* проверьте, является ли A подмножеством B и наоборот.

1. Создайте 2 списка из 5 случайных целых чисел от 1 до 10.

* определите, сколько различных чисел содержат списки;
* определите, сколько различных чисел содержится одновременно как в первом списке, так и во втором,
* выведите все числа, которые входят как в первый, так и во второй список в порядке возрастания,
* удалите из первого списка числа, входящие во второй список

1. Для заданного натурального n≥2 найти все простые числа, которые меньше или равны n, используя алгоритм, который называют «решето Эратосфена»:

* пусть R – строимое множество простых чисел и D – множество, называемое решетом, в начале работы R – пустое множество, D – множество всех целых чисел от 2 до n;
* на каждом шаге алгоритма наименьший элемент D помещается в R, а из D удаляются все числа, кратные этому элементу;
* алгоритм заканчивает работу при пустом D.

1. Участники олимпиады решали 3 задачи. Известны фамилии тех, кто решил первую, вторую и третью задачи (для каждой задачи отдельный список). Найдите и выведите на экран фамилии тех, кто

* решил хотя бы одну задачу (любую);
* решил все задачи;
* решил ровно1 задачу (любую);
* решил ровно 2 задачи (любые);
* решил не больше 2 задач (любых).

1. Каждый участник международной конференции указал, какими языками он владеет (для хранения этой информации используйте словарь). Предполагается, что конференция проводится на русском языке, и выполняется синхронный перевод на английский. Определите:

* есть ли язык, на котором разговаривают все участники;
* фамилии участников, которые не владеют русским языком;
* на какие языки еще нужен перевод (язык учитывается, если участник не знает русского и английского), и сколько человек знают этот язык.

**Программирование на языке Python**

Учебно-методическое пособие

по дисциплине «Компьютерный практикум»

для студентов, обучающихся по направлению подготовки

38.03.05 "Бизнес-информатика"

(программа подготовки бакалавров

очная и заочная формы обучения)

**Автор:**

**Миронова Ирина Васильевна**, кандидат физ.-мат. наук, доцент, доцент департамента анализа данных, принятия решений и финансовых технологий

|  |  |  |
| --- | --- | --- |
| Компьютерный набор, верстка | И.В. Миронова | |
| Формат 60x90/16. Гарнитура Times New Roman | |
| 4,2 п.л. 2018 г. Электронное издание | |

Вычитка и корректура выполнены автором

© ФГОБУ ВО «Финансовый университет при Правительстве Российской Федерации», 2018.

© Департамент анализа данных, принятия решений и финансовых технологий, 2018.

© Миронова Ирина Васильевна, 2018.