Методические материалы по Python 3

(версия 3.7.0)

Актау, 2018г.

Оглавление

Почему именно PYTHON? 3
История PYTHON 3
Дзен PYTHON 4
IDE 5
Арифметические операции на PYTHON 6
Создание файла 7
Библиотека (модуль) math 8
Арифметические операторы: 9
Побитовые операторы: 9
Логические операторы 10
Операторы сравнения 11
Условный оператор IF 11
Цикл While 14
Множественное присваивание 15
Цикл For 15
Функции 17
Списки и кортежи 17
Списки 17
Кортежи 19
Множества 19
Словари 20
Практические задачи 22

Почему именно PYTHON?

- Язык программирования Python - язык высокого уровня, достаточно "молодой", но очень популярный, который уже сейчас широко используется на практике и сфера применения Python постоянно расширяется.
- Синтаксис языка Python минималистический и гибкий. На этом языке можно составлять простые и эффективные программы.
- Стандартная библиотека для этого языка содержит множество полезных функций, что значительно облегчает процесс создания программных кодов.
- Язык Python поддерживает несколько парадигм программирования, включая структурное, объектно-ориентированное и функциональное программирование. И это далеко не полный список.

Язык Python вполне удачный выбор для первого языка при обучении программированию.

История PYTHON

Разработка языка Python была начата в конце 1980-х годов сотрудником голландского института CWI Гвидо ван Россумом. Для распределённой ОС Amoeba требовался расширяемый скриптовый язык, и Гвидо начал писать Python на досуге, позаимствовав некоторые наработки для языка ABC (Гвидо участвовал в разработке этого языка, ориентированного на обучение программированию). В феврале 1991 года Гвидо опубликовал исходный текст в группе новостей alt.sources. С самого начала Python проектировался как объектно-ориентированный язык.

Название языка произошло вовсе не от вида пресмыкающихся. Автор назвал язык в честь популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона».

Впрочем, всё равно название языка чаще связывают именно со змеёй, нежели с передачей — пиктограммы файлов в KDE или в Microsoft Windows и даже эмблема на сайте python.org (до выхода версии 2.5) изображают змеиные головы.

Важная цель разработчиков Python — создавать его забавным для использования. Это отражено в его названии, которое пришло из Монти Пайтона. Также это отражено в иногда игривом подходе к обучающим программам и справочным материалам, таким как примеры использования, которые используют понятия ветчины (spam) и яиц вместо стандартных foo и bar.

«Серьёзное отношение к чему бы то ни было в этом мире является роковой ошибкой»
☺
Л. Кэрролл.
«Алиса в стране чудес»

Дзен PYTHON

Разработчики языка Python придерживаются определённой философии программирования, называемой «The Zen of Python» («Дзен Питона», или «Дзен Пайтона»).

Её текст выдаётся интерпретатором Python по команде import this (работает один раз за сессию). Автором этой философии считается Тим Петерс (Tim Peters).

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один — и, желательно, *только* один — очевидный способ сделать это.
- Сейчас лучше, чем никогда.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, *возможно*, хороша.
- Пространства имён — отличная вещь! Давайте будем делать их больше!

На сегодняшний день Python используется при разработке самых различных проектов, среди которых:

- разработка сценариев для работы с Web и Intemet-приложений;
- сетевое программирование;
- средства поддержка технологий HTML и XML;
- приложения для работы с электронной почтой и поддержки Internet-протоколов;

- приложения для обслуживания всевозможных баз данных;
- программы для научных расчетов;
- приложения с графическим интерфейсом;
- создание игр и компьютерной графики,
- и многое другое

1. **кроссплатформенный,** потому что Python работает почти на всех известных операционных системах, включая: Linux, Window, FreeBSD, Macintosh, Solaris и т.д.
2. **интерактивный,** потому что позволяет в режиме реального времени взаимодействовать с интерпретатором.
3. **интерпретируемый,** потому что не требует компиляций для выполнения кода

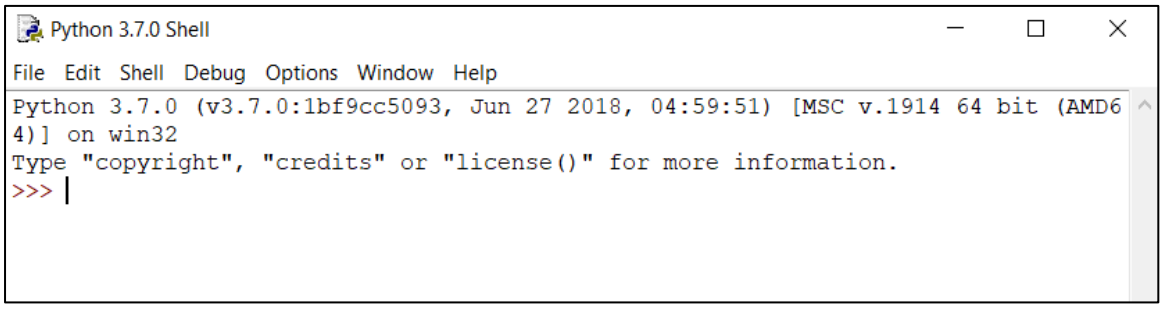
«Hello, World!»		
C++	Java	Python

# include < iostream> 	
using name space std ;
int main ( ) {
cout<< " Hello , world ! " <<endl ;
return 0;
}
	classMyClass{
	public static void main (String[ ] args){
	System.out.println (“Hello, World!”);
	}
	}
		Print(“Hello, World!”)

IDE

Как уже отмечалось выше, для выполнения программных кодов, написанных на Python, нам понадобится программа-интерпретатор. Но лучше всего воспользоваться какой-нибудь интегрированной средой разработки (сокращенно IDE от английского **Integrated Development Environment**). Среда разработки предоставляет пользователю не только интерпретатор, но и редактор кодов, равно как ряд других полезных утилит.

- Если мы говорим о программном обеспечении, то в первую очередь имеет смысл выйти на официальный сайт поддержки Python по адресу www.python.org.
- Затем нажать Downloads, и выбрать последнюю версию. Например 3.7.0



Синтаксис языка прост, понятен и нагляден. В некотором смысле его можно даже назвать по-спартански лаконичным.

Арифметические операции на PYTHON

```
>>> print (5+7) #напечатаемзначение суммы двух чисел
12
>>>
>>> print ("5"+"7") #если числа в кавычках, то "+" склеит данные в кавычках
57
```

```
>>> print(3 * 7, (17 - 2) * 8)
21 120
>>>
```

```
>>> print(2 ** 16)  # две звёздочки означают возведение в степень
65536
```

```
>>> print(37 / 3)  # один слэш — это деление с ответом-дробью
12.333333333333334
```

```
>>> print(37 // 3)# два слэша считают частное от деления нацело
                  # это как операция div в других языках

12
```

```
>>> print(37 % 3) #процент считает остаток от деления нацело
                  # это как операция mod в других языках

1
```

```
print('Как вас зовут?')
name = input()  # считываем строку и кладём её в переменную name
print('Здравствуйте, ' + name + '!')
```

```
1  a = input()
2  b = input()
3  s = a + b  # Переменной s присвоили значение суммы a+b
4  print(s)
```

Сумма переменных: (**неправильный** вариант)

```
a = input()
b = input()
s = a + b  # Переменной s присвоили значение суммы a+b
print(s)
```

```
========
5
9
59
>>>
```

Произошло не суммирование переменных, а склеивание двух строк

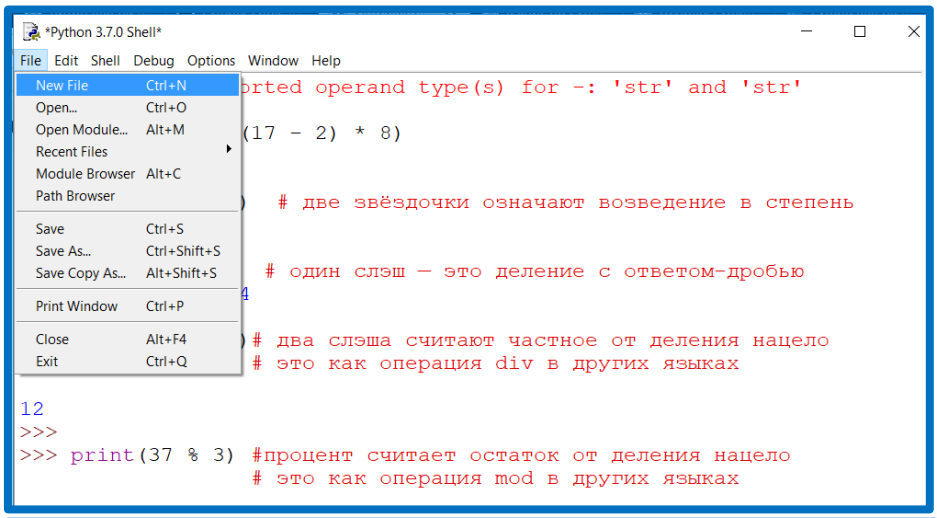
Сумма переменных: (**правильный** вариант)

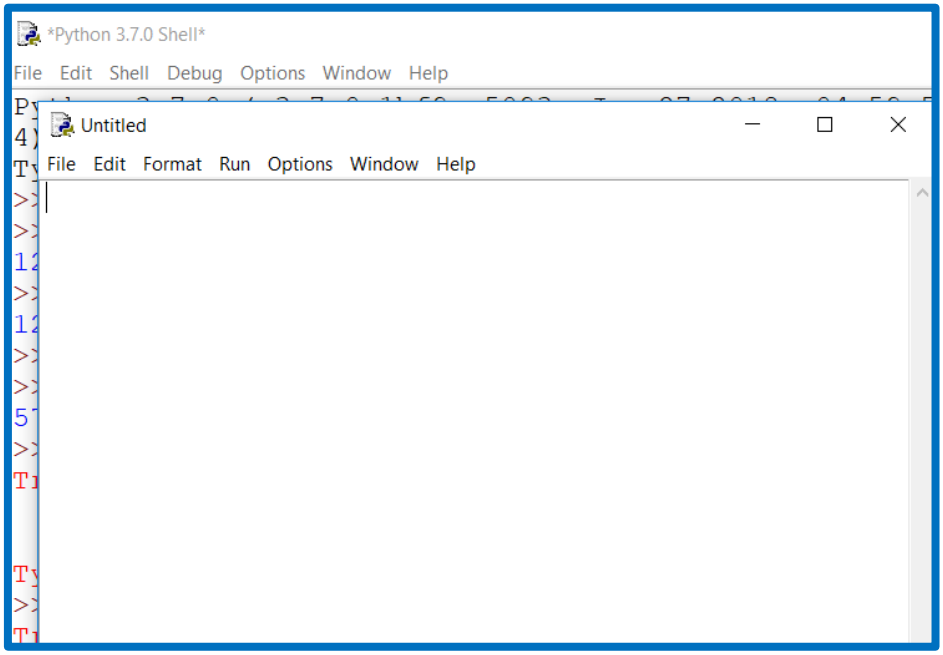
```
a = int(input())
b = int(input())
s = a + b
print(s)
```

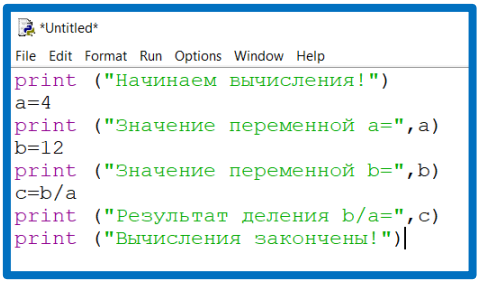
```
==============
5
9
14
>>>
```

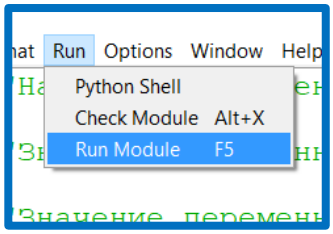
```
# Эта программа считывает три числа и выводит их сумму:
a = int(input())
b = int(input())
c = int(input())
print(a + b + c)
```

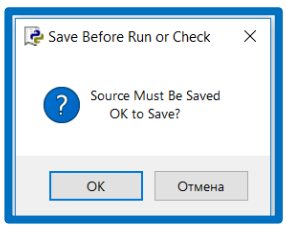
Создание файла

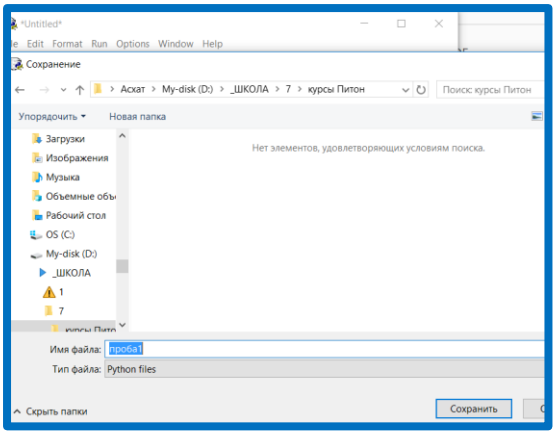


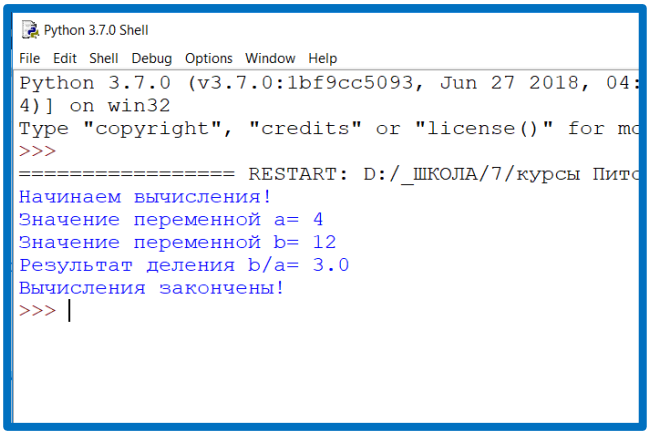












Библиотека (модуль) math

Для проведения вычислений с действительными числами язык Питон содержит много дополнительных функций, собранных в библиотеку (модуль), которая называется math.

Для использования этих функций в начале программы необходимо подключить математическую библиотеку, что делается командой

```
import math
```

Функция	Описание
Округление	
`int(x)`	Округляет число в сторону нуля. Это стандартная функция, для ее использования не нужно подключать модуль `math`.
`round(x)`	Округляет число до ближайшего целого. Если дробная часть числа равна 0.5, то число округляется до ближайшего четного числа.
`round(x, n)`	Округляет число `x` до `n` знаков после точки. Это стандартная функция, для ее использования не нужно подключать модуль `math`.
`floor(x)`	Округляет число вниз («пол»), при этом `floor(1.5) == 1`, `floor(-1.5) == -2`
`ceil(x)`	Округляет число вверх («потолок»), при этом `ceil(1.5) == 2`, `ceil(-1.5) == -1`
`abs(x)`	Модуль (абсолютная величина). Это — стандартная функция.

Корни, логарифмы	
`sqrt(x)`	Квадратный корень. Использование: `sqrt(x)`
`log(x)`	Натуральный логарифм. При вызове в виде `log(x, b)` возвращает логарифм по основанию `b`.
`e`	Основание натуральных логарифмов e = 2,71828...
Тригонометрия	
`sin(x)`	Синус угла, задаваемого в радианах
`cos(x)`	Косинус угла, задаваемого в радианах
`tan(x)`	Тангенс угла, задаваемого в радианах
`asin(x)`	Арксинус, возвращает значение в радианах
`acos(x)`	Арккосинус, возвращает значение в радианах
`atan(x)`	Арктангенс, возвращает значение в радианах
`atan2(y, x)`	Полярный угол (в радианах) точки с координатами (x, y).
`degrees(x)`	Преобразует угол, заданный в радианах, в градусы.
`radians(x)`	Преобразует угол, заданный в градусах, в радианы.
`pi`	Константа π = 3.1415...

Операторы:

Обычно выделяют четыре группы операторов:

- арифметические;
- побитовые;
- операторы сравнения;
- логические операторы.

Арифметические операторы:

Арифметические операторы

Оператор	Описание
+	Оператор сложения. Вычисляется сумма двух чисел
-	Оператор вычитания. Вычисляется разность двух чисел
*	Оператор умножения. Вычисляется произведение двух чисел
/	Оператор деления. Вычисляется отношение двух чисел
//	Оператор целочисленного деления. Вычисляется целая часть от деления одного числа на другое
%	Оператор вычисления остатка от целочисленного деления. Вычисляется остаток от деления одного числа на другое
**	Оператор возведения в степень. Результатом является число, получающееся возведением первого операнда в степень, определяемую вторым операндом

Побитовые операторы:

Побитовые операторы

Оператор	Описание
~	Побитовая инверсия (унарный оператор - у него один операнд). Результатом является число, получающееся заменой нулей на единицы и единиц на нули в побитовом представлении операнда (сам операнд при этом не меняется)
&	Побитовое И. При вычислении результата сравниваются побитовые представления операндов. Если на одной и той же позиции в операндах стоят единицы, то в числе-результате на этой же позиции будет единица. В противном случае (то есть если хотя бы в одной из двух позиций нуль) в числе-результате на соответствующей позиции будет нуль
|	Побитовое ИЛИ. Сравниваются побитовые представления операндов. Если на одной и той же позиции в операндах стоят нули, то в числе-результате на этой же позиции будет нуль. В противном случае (то есть если хотя бы в одной из двух позиций единица) в числе-результате на соответствующей позиции будет единица
^	Побитовое ИСКЮЧАЮЩЕЕ ИЛИ. Результат вычисляется сравнением побитовых представлений операндов. Если на одной и той же позиции в операндах стоят разные значения (у одного числа нуль, а у другого единица), то в числе-результате на этой же позиции будет единица. В противном случае (то есть если на соответствующих позициях в операндах стоят одинаковые числа) в числе-результате на этой позиции будет нуль
<<	Сдвиг влево. Результат вычисляется так: в побитовом представлении первого операнда выполняется сдвиг влево. Количество разрядов, на которые выполняется сдвиг, определяется вторым операндом. Младшие недостающие биты заполняются нулями
>>	Сдвиг вправо. Для вычисления результата в побитовом представлении первого операнда выполняется сдвиг вправо. Количество разрядов, на которые выполняется сдвиг, определяется вторым операндом. Биты слева заполняются значением самого старшего бита (для положительных чисел это нуль, а для отри-

Логические операторы

Логические операторы

Оператор	Описание
`or`	Бинарный оператор (у оператора два операнда). Логическое *ИЛИ*. В общем случае результатом выражения `x or y` является `True`, если значение хотя бы одного из операндов `x` или `y` равно `True`. Если значения обеих операндов `x` и `y` равны `False`, результатом выражения `x or y` будет `False`. В Python выражения на основе оператора `or` вычисляются по упрощенной схеме: если первый операнд `x` интерпретируется как `True`, то `x` возвращается в качестве результата (второй операнду при этом не вычисляется). Если первый операнд `x` интерпретируется как `False`, то в качестве результата возвращается второй операнд `y`
`and`	Бинарный оператор (у оператора два опера́нда). Логическое *И*. В общем случае результатом выражения `x and y` является значение `True`, если значения обеих операндов `x` и `y` равны `True`. Если значение хотя бы одного из операндов `x` или `y` равно `False`, результатом выражения `x and y` будет `False`. В Python выражения на основе оператора `and` вычисляются по упрощенной схеме: если первый операнд `x` интерпретируется как `False`, то `x` возвращается в качестве результата (второй операнду при этом не вычисляется). Если первый операнд `x` интерпретируется как `True`, то в качестве результата возвращается второй операнд `y`
`not`	Логическое отрицание. Унарный оператор (у оператора один операнд). Результатом выражения `not x` будет значение `True`, если у операнда `x` значение `False`. Результатом выражения `not x` будет значение `False`, если у операнда `x` значение `True`

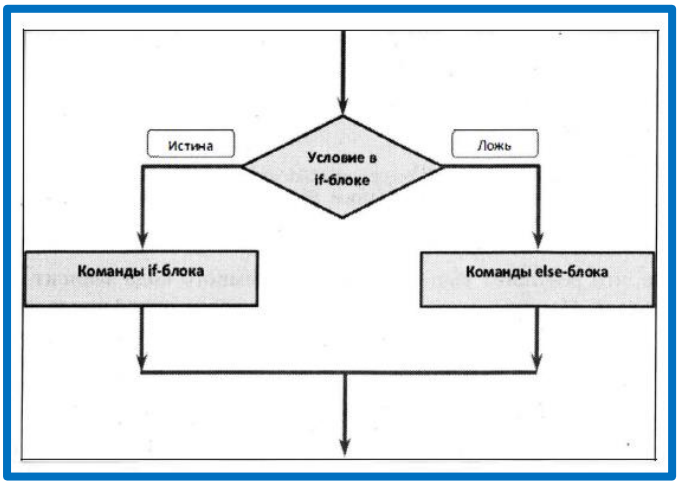
Операторы сравнения

Операторы сравнения

Оператор	Описание
<	Строго меньше. Результатом является True, если значение операнда слева от оператора *меньше* значения операнда справа от оператора. Иначе возвращается значение False
>	Строго больше. Результатом является True, если значение операнда слева от оператора *больше* значения операнда справа от оператора. Иначе возвращается значение False
<=	Меньше или равно. Результатом является True, если значение операнда слева от оператора *не больше* значения операнда справа от оператора. Иначе возвращается значение False
>=	Больше или равно. Результатом является True, если значение операнда слева от оператора *не меньше* значения операнда справа от оператора. Иначе возвращается значение False
==	Равно. Результатом является True, если значение операнда слева от оператора *равно* значению операнда справа от оператора. Иначе возвращается значение False
!=	Не равно. Результатом является True, если значение операнда слева от оператора *не равно* значению операнда справа от оператора. Иначе возвращается значение False
is	Оператор проверки идентичности объектов. В качестве результата возвращается значение True, если оба операнда ссылаются на один и тот же объект. В противном случае (то есть если операнды ссылаются на разные объекты) возвращается значение False
is not	Оператор проверки неидентичности объектов. В качестве результата возвращается значение True, если операнды ссылаются на разные объекты. В противном случае (то есть если операнды ссылаются на один и тот же объект) возвращается значение False

Условный оператор IF

Схема выполнения условного оператора



Оператор ветвления *if* позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования.

Итак, условная инструкция в Питоне имеет следующий синтаксис:

if Условие:

    Блок инструкций 1

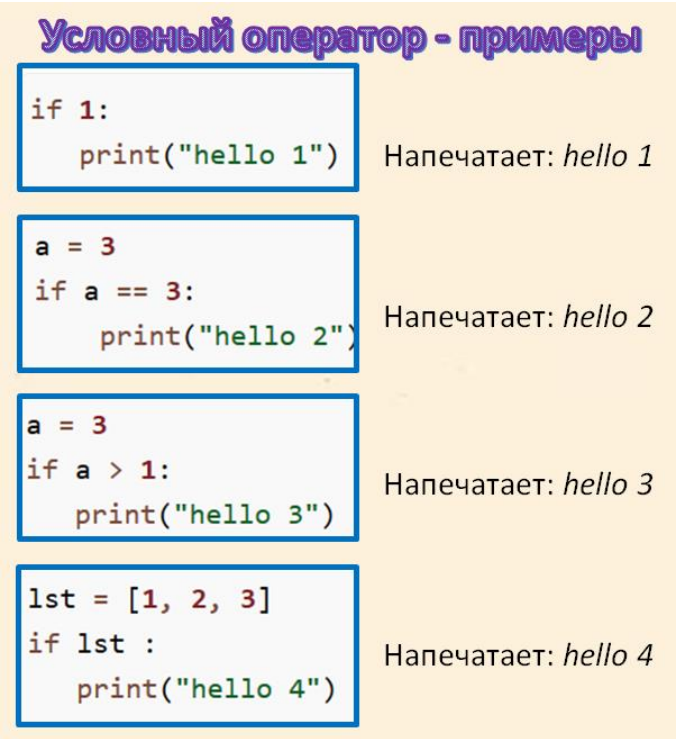
else:

    Блок инструкций 2

Блок инструкций 1 будет выполнен, если Условие истинно. Если Условие ложно, будет выполнен Блок инструкций 2.

В условной инструкции может отсутствовать слово else и последующий блок. Такая инструкция называется неполным ветвлением. Например, если дано число x и мы хотим заменить его на абсолютную величину x, то это можно сделать следующим образом:

```
x = int(input())
if x < 0:
    x = -x
print(x)
```



Для реализации выбора из нескольких альтернатив можно использовать конструкцию ***if – elif – else.***

```
if выражение_1:
    инструкции_(блок_1)
elif выражение_2:
    инструкции_(блок_2)
elif выражение_3:
    инструкции_(блок_3)
else:
    инструкции_(блок_4)
```

```
a = int(input("введите число:"))
if a < 0:
   print("Neg")
elif a == 0:
   print("Zero")
else:
   print("Pos")
```

Проверим, что хотя бы одно из чисел a или b оканчивается на 0:

```
a = int(input())
b = int(input())
if a % 10 == 0 or b % 10 == 0:
    print('YES')
else:
    print('NO')
```

Пример программы, определяющий четверть координатной плоскости

```
x = int(input())
y = int(input())
if x > 0 and y > 0:
    print("Первая четверть")
elif x > 0 and y < 0:
    print("Четвертая четверть")
elif y > 0:
    print("Вторая четверть")
else:
    print("Третья четверть")
```

```
# Пользователь вводит значение
res=eval(input("Введите что-нибудь: "))
# Тип значения запоминаем в переменной
resType=type(res)
# Используем условные операторы (упрощенная форма)
# для проверки типа введенного пользователем значения
if resType==int:
    # Если целое число
    print("Это целое число!")
if resType==float:
    # Если действительное число
    print("Это действительное число!")
if resType!=int and resType!=float:
    # Если не число
    print("Наверное, это текст!")
# После выполнения условных операторов
print("Работа завершена!")
```

```
# Пользователь вводит значение
res=eval(input("Введите что-нибудь: "))
# Тип значения запоминаем в переменной
resType=type(res)
# Используем условные операторы (упрощенная форма)
# для проверки типа введенного пользователем значения
if resType==int:
    # Если целое число
    print("Это целое число!")
elif resType==float:
    # Если действительное число
    print("Это действительное число!")
else:
    # Если не число
    print("Наверное, это текст!")
# После выполнения условных операторов
print("Работа завершена!")
```

```
Введите что-нибудь: 12
Это целое число!
Работа завершена!
```

```
Введите что-нибудь: "Изучаем Python"
Наверное, это текст!
Работа завершена!
```

Цикл While

Шаблон для оператора цикла:

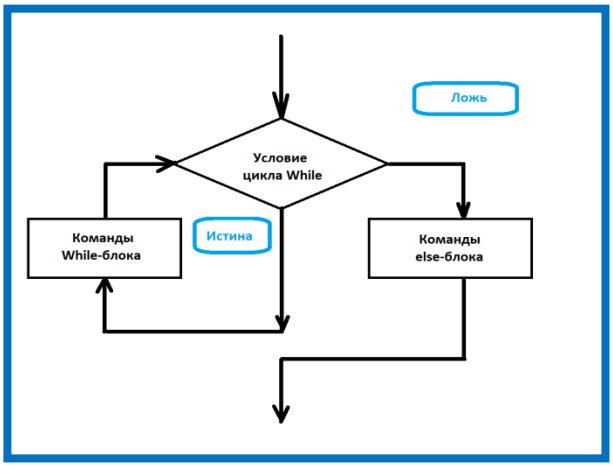
while условие:

команды_1

else:

команды_2

/else-необязательный блок/



В простейшем случае:

```
while условие:
    блок инструкций
```

```
#Выведем квадраты чисел от 1 до 10
i = 1 #Значение счетика= 1
while i <= 10:  # пока i <= 10
    print(i ** 2)
    i += 1
```

```
#Определим количество цифр натурального числа n
n = int(input())
length = 0
while n > 0:
    n //= 10  # это эквивалентно n = n // 10
    length += 1
print(length)
```

После тела цикла можно написать слово else: и после него блок операций, который будет выполнен *один раз* после окончания цикла, когда проверяемое условие станет неверно:

```
i = 1
while i <= 10:
    print(i)
    i += 1
else:
    print('Цикл окончен, i =', i)
```

Приведем пример программы, которая считывает числа до тех пор, пока не встретит отрицательное число. При появлении отрицательного числа программа завершается.

```
a = int(input())
while a != 0:
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
    a = int(input())
else:
    print('Ни одного отрицательного числа не встретилось')
```

Вместо команды `s=s+i` можно было использовать эквивалентную ей команду `s+=i`, а вместо команды `i=i+1` - соответственно, команду `i+=1`.

Листинг 2.5. Оператор цикла с break-инструкцией

```
print("Сумма натуральных чисел")
n=100 # Количество слагаемых
# Формируем текст для отображения результата
text="1+2+...+"+str(n)+" ="
# Итерационная переменная для оператора цикла
i=1
# Переменная для записи суммы
s=0
# Оператор цикла для вычисления суммы
while True:
    # Добавляем слагаемое к сумме
    s+=i
    # Изменяем итерационную переменную
    i+=1
    if i>n:
        break
# Отображаем результат
print(text,s)
```

Множественное присваивание

В Питоне можно за одну инструкцию присваивания изменять значение сразу нескольких переменных. Делается это так:

```
a, b = 0, 1
```

Множественное присваивание удобно использовать, когда нужно обменять значения двух переменных.

Такая программа

```
a = 17
b = 21
tmp = a
a = b
b = tmp
print(a, b)
# 21 17
```

в Питоне запишется так:

```
a = 17
b = 21
a, b = b, a
print(a, b)
# 21 17
```

Цикл For

Шаблон использования этого оператора такой (жирным шрифтом выделены ключевые элементы):

```
for элемент in последовательность:
    команды
```

Нередко (но не всегда) в качестве последовательности в операторе цикла `for` используются *списки*. Со списками мы еще не знакомы. Основные сведения о списках (равно как и об иных типах данных, содержащих коллекции значений и подпадающих под определение `последовательности`) представлены в следующих главах. Здесь нас списки интересуют только в контексте использования их в операторе цикла `for`. Важно знать, что список - это набор упорядоченных элементов (оформленных соответствующим образом). Причем элементы могут быть разного типа. Чтобы создать список, достаточно в квадратных скобках через запятую перечислить элементы списка. Например, конструкция `[1,2,3,4,5]` представляет собой список из пяти натуральных чисел.

В качестве `последовательности` можно использовать текстовые значения. В этом случае перебираются буквы в тексте.

```
i = 1
for color in 'red', 'orange', 'yellow', 'green', 'cyan', 'blue', 'violet':
    print('#', i, ' color of rainbow is ', color, sep = '')
    i += 1
```

```
#1 color of rainbow is red
#2 color of rainbow is orange
#3 color of rainbow is yellow
#4 color of rainbow is green
#5 color of rainbow is cyan
#6 color of rainbow is blue
#7 color of rainbow is violet
>>>
```

В списке значений могут быть выражения различных типов

```
for i in 1, 2, 3, 'one', 'two', 'three':
    print(i)
```

```
1
2
3
one
two
three
>>>
```

Функция range

```
for i in range(6):  # равносильно инструкции for i in 0, 1, 2, 3, 4, 5:
    # здесь можно выполнять циклические действия
    print(i)
    print(i ** 2)
# цикл закончился, поскольку закончился блок с отступом
print('Конец цикла')
```

```
=== RESTART: C:/
0
0
1
1
2
4
3
9
4
16
5
25
Конец цикла
```

```
sum = 0
n = 5
for i in range(1, n + 1):
    sum += i
print(sum)
```

```
=== RESTART:
15
>>>
```

```
print("Сумма натуральных чисел")
n=100 # Количество слагаемых
# Формируем текст для отображения результата
text="1+2+...+"+str(n)+" ="
# Переменная для записи суммы
s=0
# Оператор цикла для вычисления суммы
for i in range(1,n+1):
    # Добавляем слагаемое к сумме
    s=s+i
# Отображаем результат
print(text,s)
```

```
Сумма натуральных чисел
1+2+...+100 = 5050
```

Функции

Весь шаблон объявления функции выглядит следующим образом (жирным шрифтом выделены ключевые элементы):

def имя_ функции (аргументы) :

команды

- Для функции не указывается тип результата (хотя функция, разумеется , может возвращать результат).

```
# Функция без аргументов
def your_name():
    # Отображается сообщение
    print("Добрый день!")
    # Запоминается введенный пользователем текст
    name=input("Как Вас зовут? ")
    # Результат функции
    return name
# Функция с одним аргументом
def say_hello(txt):
    # Отображается сообщение
    print("Здравствуйте,",txt+"!")
# Вызываем функцию и результат записываем в переменную
my_name=your_name()
# Вызываем функцию с аргументом
say_hello(my_name)
```

Списки и кортежи

Списки

Большинство программ работает не с отдельными переменными, а с набором переменных.

Списки являются близким аналогом понятия «Массив» в других языках программирования.

Элементами списка могут быть объекты **разного типа**.

Список представляет собой последовательность элементов, пронумерованных от 0, как символы в строке. Список можно задать перечислением элементов списка в квадратных скобках, например, список можно задать так:

```
Primes = [2, 3, 5, 7, 11, 13]
Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']
```

В списке `Primes` — 6 элементов, а именно: `Primes[0] == 2`, `Primes[1] == 3`, `Primes[2] == 5`, `Primes[3] == 7`, `Primes[4] == 11`, `Primes[5] == 13`. Список `Rainbow` состоит из 7 элементов, каждый из которых является строкой.

Для списков целиком определены следующие операции: конкатенация списков (сложение списков, т. е. приписывание к одному списку другого) и повторение списков (умножение списка на число). Например:

```
a = [1, 2, 3]
b = [4, 5]
c = a + b
d = b * 3
print([7, 8] + [9])
print([0, 1] * 3)
```

В результате список `c` будет равен `[1, 2, 3, 4, 5]`, а список `d` будет равен `[4, 5, 4, 5, 4, 5]`. Это позволяет по-другому организовать процесс считывания списков: сначала считать размер списка и создать список из нужного числа элементов, затем организовать цикл по переменной `i` начиная с числа 0 и внутри цикла считывается `i`-й элемент списка:

```
a = [0] * int(input())
for i in range(len(a)):
    a[i] = int(input())
```

Приведем пример, демонстрирующий использование цикла `for` в ситуации, когда из строки надо выбрать все цифры и сложить их в массив как числа.

```
# дано: s = 'ab12c59p7dq'
# надо: извлечь цифры в список digits,
# чтобы стало так:
# digits == [1, 2, 5, 9, 7]

s = 'ab12c59p7dq'
digits = []
for symbol in s:
    if '1234567890'.find(symbol) != -1:
        digits.append(int(symbol))
print(digits)
```

Для создания списка, заполненного одинаковыми элементами, можно использовать оператор повторения списка (генератор), например:

```
n = 5
a = [0] * n
```

Кортеж

Кортеж представляет собой упорядоченный набор некоторых элементов.

Принципиальное отличие кортежей от списков состоит в том, что кортежи нельзя изменять. То есть после того, как кортеж создан, **внести в него изменения уже не получится**.

С точки зрения "типологии" языка Python кортеж относится к типу tuple. Поэтому нет ничего удивительного, что создать кортеж можно с помощью функции tuple ( ) . Если аргумента у функции нет, то будет создан пустой кортеж.

```
# Создаем пустой кортеж
a=tuple ( )
# Проверяем содержимое кортежа
print ( a )
# Создаем кортеж на основе списка
b=tuple ( [ l0 , 20 , 30 ] )
# Проверяем содержимое кортежа
print ( b )
# Создаем кортеж н а основе текста
c=tuple ( " Python " )
# Проверяем содержимое корт ежа
print ( c )
# Объединение кортежей
a=b+ ( 40 , ( 1 , 2 , 3 ] , 60 )
# Проверяем результат объединения кортежей
print ( a )
# Получение среза
print ( a [ 2 : 5 ] )
```

Множества

Множество в языке Питон — это структура данных, эквивалентная множествам в математике. Множество может состоять из различных элементов, порядок элементов в множестве неопределен. В множество можно добавлять и удалять элементы, можно перебирать элементы множества, можно выполнять операции над множествами (объединение, пересечение, разность). Можно проверять принадлежность элемента множеству.

В отличие от массивов, где элементы хранятся в виде последовательного списка, в множествах порядок хранения элементов неопределен (более того, элементы множества хранятся не подряд, как в списке, а при помощи хитрых алгоритмов). Это позволяет выполнять операции типа “проверить принадлежность элемента множеству” быстрее, чем просто перебирая все элементы множества.

Элементами множества может быть любой неизменяемый тип данных: числа, строки, кортежи.

Если функции `set` передать в качестве параметра список, строку или кортеж, то она вернёт множество, составленное из элементов списка, строки, кортежа. Например:

```
A = {1, 2, 3}
A = set('qwerty')
print(A)
```

выведет `{'e', 'q', 'r', 't', 'w', 'y'}`.

Каждый элемент может входить в множество только один раз, порядок задания элементов неважен. Например, программа:

```
A = {1, 2, 3}
B = {3, 2, 3, 1}
print(A == B)
```

выведет `True`, так как `A` и `B` — равные множества.

Каждый элемент может входить в множество только один раз. `set('Hello')` вернет множество из четырех элементов: `{'H', 'e', 'l', 'o'}`.

Словари

Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному, называется *словарем* или *ассоциативным массивом*. Соответствующая структура данных в языке Питон называется `dict`.

Рассмотрим простой пример использования словаря. Заведем словарь `Capitals`, где индексом является название страны, а значением — название столицы этой страны. Это позволит легко определять по строке с названием страны ее столицу.

```
# Список для формирования словаря
A=[["Пушкин А.С.","Капитанская дочка"],["Чехов А.П.","Вишневый
сад"],["Толстой Л.Н.","Война и мир"]]
# Создаем словарь на основе списка
writers=dict(A)
# Отображаем содержимое словаря
print("Словарь:")
print(writers)
# Обращение к элементу словаря по ключу
print("Чехов написал пьесу:",writers["Чехов А.П."])
# Изменяем значение элемента словаря

writers["Чехов А.П."]="Каштанка"
# Проверяем содержимое словаря
print("Словарь после изменения элемента:")
print(writers)
# Добавляем в словарь новый элемент
writers["Достоевский Ф.М."]="Преступление и наказание"
# Проверяем содержимое словаря
print("Словарь после добавления элемента:")
print(writers)
```

```
# Добавляем в словарь новый элемент
writers["Достоевский Ф.М."]="Преступление и наказание"
# Проверяем содержимое словаря
print("Словарь после добавления элемента:")
print(writers)
print()
# Перебор элементов словаря по ключу
print("Авторы и их произведения.")
for s in writers.keys():
    print("Автор:",s)
    print("Произведение:",writers[s])
    print()
# Создаем новый словарь
lights=dict(красный="движение запрещено",желтый="всем
внимание",зеленый="движение разрешено")
# Проверяем содержимое словаря
print("Новый словарь:")
print(lights)
# Значение ключа
color="зеленый"
# Обращение к элементу словаря по ключу
print("Если горит",color,"свет, то",lights[color]+"!")
print()
# Создаем еще один словарь
girls={(90,60,90):"Света",(85,65,89):"Юля",(92,58,91):"Нина"}
# Проверяем содержимое словаря
print("Еще один словарь:")
print(girls)
# Значение ключа
params=(90,60,90)
# Обращение к элементу словаря по ключу
print(girls[params]+":",params)
```

Результат:

```
Словарь:
{'Чехов А.П.': 'Вишневый сад', 'Пушкин А.С.': 'Капитанская
дочка', 'Толстой Л.Н.': 'Война и мир'}
Чехов написал пьесу: Вишневый сад
Словарь после изменения элемента:


{'Чехов А.П.': 'Каштанка', 'Пушкин А.С.': 'Капитанская дочка',
'Толстой Л.Н.': 'Война и мир'}
Словарь после добавления элемента:
{'Чехов А.П.': 'Каштанка', 'Достоевский Ф.М.': 'Преступление
и наказание', 'Пушкин А.С.': 'Капитанская дочка', 'Толстой
Л.Н.': 'Война и мир'}

Авторы и их произведения.
Автор: Чехов А.П.
Произведение: Каштанка

Автор: Достоевский Ф.М.
Произведение: Преступление и наказание

Автор: Пушкин А.С.
Произведение: Капитанская дочка

Автор: Толстой Л.Н.
Произведение: Война и мир

Новый словарь:
{'зеленый': 'движение разрешено', 'желтый': 'всем внимание',
'красный': 'движение запрещено'}
Если горит зеленый свет, то движение разрешено!

Еще один словарь:
{(92, 58, 91): 'Нина', (85, 65, 89): 'Юля', (90, 60, 90):
'Света'}
Света: (90, 60, 90)
```

Практические задачи

1). Напишите программу, вычисляющую площадь треугольника по переданным длинам трёх его сторон по формуле Герона: $S = \sqrt{p(p-a)(p-b)(p-c)}$ где $p = \frac{a+b+c}{2}$ - полупериметр треугольника. На вход программе подаются целые числа, выводом программы должно являться вещественное число, соответствующее площади треугольника.	**Sample Input:** 3 4 5 **Sample Output:** 6.0	a=int(input()) b=int(input()) c=int(input()) p=(a+b+c)/2 print((p*(p-a)*(p-b)*(p-c))**0.5)
2). Напишите программу, принимающую на вход целое число, которая выводит True, если переданное значение попадает в интервал $(-17,12] \cup (14,15) \cup [19,+\infty)$ и False в противном случае (регистр символов имеет значение). Обратите внимание на разные скобки, используемые для обозначения интервалов. В задании используются полуоткрытые и открытые интервалы.	**Sample Input 1:** 21 **Sample Output 1:** True **Sample Input 2:** -19 **Sample Output 2:** False	a=int(input()) print (-15<a<=12 or 14<a<17 or 19<=a)

3). Напишите простой калькулятор, который считывает с пользовательского ввода три строки: первое число, второе число и операцию, после чего применяет операцию к введённым числам ("первое число" "операция" "второе число") и выводит результат на экран. Поддерживаемые операции: +, -, /, *, mod, pow, div, где mod — это взятие остатка от деления, pow — возведение в степень, div — целочисленное деление. Если выполняется деление и второе число равно 0, необходимо выводить строку "Деление на 0!". Обратите внимание, что на вход программе приходят вещественные числа.	**Sample Input 1:** 5.0 0.0 mod **Sample Output 1:** Деление на 0! **Sample Input 2:** -12.0 -8.0 * **Sample Output 2:** 96.0	a=float(input()) b=float(input()) c=input() if c=='+': print (a + b) if c=='-': print (a-b) if c=='/' and b ==0.0: print("Деление на 0!") if c=='/' and b!=0.0 : print(a/b) if c=='*': print(a*b) if c=='pow': print(a**b) if c=='div' and b==0.0: print("Деление на 0!") if c=='div' and b!=0.0: print(a//b) if c=='mod' and b==0.0: print ("Деление на 0!") if c=='mod' and b!=0.0: print (a%b)

4). В программе для учета поголовья аистов необходимо правильно выставить окончания.
1 аиста, 2 аиста, 19 аистов. Известно, что поголовье насчитывает 1000 аистов.
Нужно программным кодом учесть все варианты окончаний. Проверьте, что ваша программа правильно обработает все случаи, как минимум до 1000 аистов (включительно)
	Sample Input 1:
	5
	Sample Output 1:
	5 аистов
		s=int(input())
		if s<0:
		  print("Ошибка! Введите положительное число!")
		elif s%10==1 and s%100!=11:
		  print(s,"аист")
		elif (2<s%10<=4 and (s%100<10 or s%100>20)) or 2<=s<=4:
		  print(s,"аиста")
		else: print(s,"аистов")

Автор методического пособия: учитель информатики КГУ «Школа-лицей №7 имени Марабаева Н.» Менгали Наталья Сергеевна. Актау, 2018 год.