

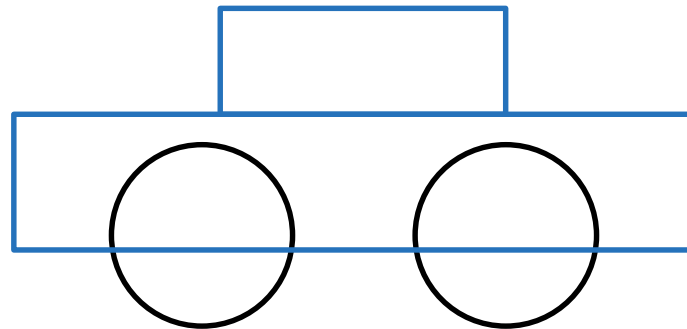
Лекция №2

Основы программирования на Python

- Функции и функциональный подход
- Введение в ООП
- Переменные, объекты и динамическая типизация
- Сборщик мусора
- Операторы Python
- Простые типы данных
- Операторы условия
- Циклы

Функции и функциональный подход

Задача поставлена следующим образом: написать алгоритм отрисовки автомобиля. ОС предлагает ограниченный API (application programming interface): `draw_pixel(x, y)`, где `x, y` – координаты закрашиваемого пикселя.



Функции и функциональный подход

Функция - это блок организованного, многократно используемого кода, который используется для выполнения конкретного задания. Функции обеспечивают лучшую модульность приложения и значительно повышают уровень повторного использования кода. **Функция определяется набором принимаемых аргументов и типом возвращаемого значения.**

```
# def - baseword для определения
# hello_world - название
def hello_world ():
    print 'Привет, Мир!' # блок, принадлежащий функции, print - тоже функция
# Конец функции
hello_world() # вызов функции
hello_world() # ещё один вызов функции
```

```
# a и b - параметры функций
>>> def test(a, b):
...     print "output: {}".format(a + b)
...
>>> test(1, 5) # 1 и 5 - это уже аргументы функций
output: 6
>>> test(2, 5)
output: 7
```

Функции и функциональный подход

есть аргументы со значением
по умолчанию

```
def test(a, b=3):
    print a, b
test(1, 4)
test(1)
```

именованные аргументы можно передавать
в любом порядке

```
def test(a, b=3, c=5):
    print a, b, c
test(1, c=10, b=8)
```

с помощью * в списке аргументов можно передать любое количество аргументов

```
def test(a, *args):
    print "a: {}".format(a)
    for arg in args:
        print arg
test(1, 2)
test(1, 2, 3, 4, 5)
my_list = [2, 3, 4]
test(1, *my_list)
```

с помощью ** в списке аргументов можно передать именованные аргументы

```
def test(a, **kwargs):
    print "a: {}".format(a)
    print "b: {}".format(kwargs['b'])
test(1, b=2)
test(1, b=2, c=3)
my_dict = {'b': 2}
test(1, **my_dict)
```

Функции и функциональный подход

есть аргументы со значением
по умолчанию

```
def test(a, b=3):
    print a, b
test(1, 4)
test(1)
```

именованные аргументы можно передавать
в любом порядке

```
def test(a, b=3, c=5):
    print a, b, c
test(1, c=10, b=8)
```

с помощью * в списке аргументов можно передать любое количество аргументов

```
def test(a, *args):
    print "a: {}".format(a)
    for arg in args:
        print arg
test(1, 2)
test(1, 2, 3, 4, 5)
my_list = [2, 3, 4]
test(1, *my_list)
```

с помощью ** в списке аргументов можно передать именованные аргументы

```
def test(a, **kwargs):
    print "a: {}".format(a)
    print "b: {}".format(kwargs['b'])
test(1, b=2)
test(1, b=2, c=3)
my_dict = {'b': 2}
test(1, **my_dict)
```

Функции и функциональный подход

```
# return - тут все понятно
def test():
    return 5
x = test()
print(x)
```

```
# pass - это заглушка
def test():
    pass
x = test()
print(x)
```

```
# lambda - анонимная функция
test = lambda x: print(x)
test(1)
```

```
# явное указание типа аргумента
# ничем не помогает, но становится понятней
def test(x: int):
    return x
```

Объектно-ориентированное программирование (ООП)

Задача посложнее: написать алгоритм работы приложения для заказа такси.

Принципы ООП:

1. Абстрагирование (создаем модель реального объекта, исключая его характеристики, не требуемые для решения поставленной задачи).
2. Инкапсуляция (модель – это «черный ящик»: пользователям известны допустимые входные данные, известно, что ожидать на выходе, но устройство и реализация их волновать не должны).
3. Наследование (если модели имеют одинаковые функциональности, выносим эти функциональности в общую родительскую модель).
4. Полиморфизм (алгоритм, умеющий работать с родительской моделью, будет работать и с дочерними).

и многие другие (см. SOLID – «single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion»).

Объектно-ориентированное программирование (ООП)

Как происходит проектирование программы с использованием принципов ООП?

1. Анализируем предметную область, выделяя объекты, т.е. сущности с определенным состоянием и поведением. Используя принцип абстрагирования, определяем важные для решения задачи аспекты объекта, выделяя их в классы, идем от частного к общему, по примеру определяем формулу.
2. По принципу инкапсуляции делаем классы максимально независимыми и изолированными друг от друга, ослабляем зависимости между классами (правило «разделяй и властвуй»).

Объектно-ориентированное программирование (ООП)

3. Из родственных классов, чтоб избежать дублирования в их содержании и поведении, организуем иерархию, применяя принцип наследования, например, для двух классов: автомобиль и мотоцикл, - вместо указания в описании каждого марки, модели, максимальной скорости переносим эти общие характеристики в третий класс – транспорт, объявляя его родительским для классов автомобиля и мотоцикла. Последние теперь будут содержать только исключительно им присущие характеристики.
4. Ну и наконец, для того, чтоб алгоритмы, работающие с данными классами, минимально зависели от конкретики этих классов, применяем принцип полиморфизма, позволяя программе уже в момент исполнения переходить от общего к частному. Т.е. алгоритм, написанный для работы с классом транспорт, сможет работать и с классами автомобиль и мотоцикл, и при этом, если необходимо, задействовать специфические функциональности этих классов.



Объектно-ориентированное программирование (ООП)

Рассмотрим, как это работает, на примере следующей задачи:

Надо написать алгоритм приготовления пиццы. Основные шаги приготовления следующие:

- подготовить тесто,
- добавить ингредиенты,
- испечь.

За порядок шагов и за сами шаги должны отвечать разные объекты – принцип единственной обязанности.

Пицца может быть разная: грибная, мясная, овощная. Тесто может готовиться одинаково, печься может одинаково, а вот ингредиенты могут добавляться и даже готовиться по-разному.



Объектно-ориентированное программирование (ООП)

Что такое класс или тип? Проведем аналогию с реальным миром. Если мы возьмем конкретный автомобиль, то это объект, но не класс. А вот общее представление об автомобилях, их назначении – это класс. Ему принадлежат все реальные объекты автомобилей, какими бы они ни были. Класс автомобилей дает общую характеристику всем автомобилям в мире, он их обобщает.

Рассмотрим целые числа в Python. Тип `int` – это класс целых чисел. Числа 5, 20344, -10 и т. д. – это конкретные объекты этого класса.

Любая программа работает с данными. Данные в языке Python представлены в форме объектов - встроенных, предоставляемых языком Python, или объектов, которые мы создаем с помощью других инструментов. По сути, объекты – это области памяти со значениями и ассоциированными с ними наборами операций (которые и отождествляются с типом объекта).

Переменные, объекты и динамическая типизация

Между программами и объектами выстроена целая иерархия (это справедливо как в Python, так и в других языках программирования):

- Программы делятся на модули.
- Модули содержат инструкции.
- Инструкции состоят из выражений.
- Выражения создают и обрабатывают объекты.

Переменные, объекты и динамическая типизация

Любой объект имеет тип. Поскольку Python использует динамическую типизацию, заранее объявлять тип переменной, ссылающейся на объект, не нужно. Механизм создания и определения переменных выглядит следующим образом:

- Переменная объявляется, как только ей присваивается некоторое значение
- Переменная не имеет типа, тип имеют объекты. А переменная - это просто ссылка, указывающая на конкретный объект в данный момент.
- Когда переменная используется, ее имя замещается объектом, на который она указывает.
- Неинициализированную переменную использовать нельзя.

Переменные, объекты и динамическая типизация

Если ввести такую инструкцию:

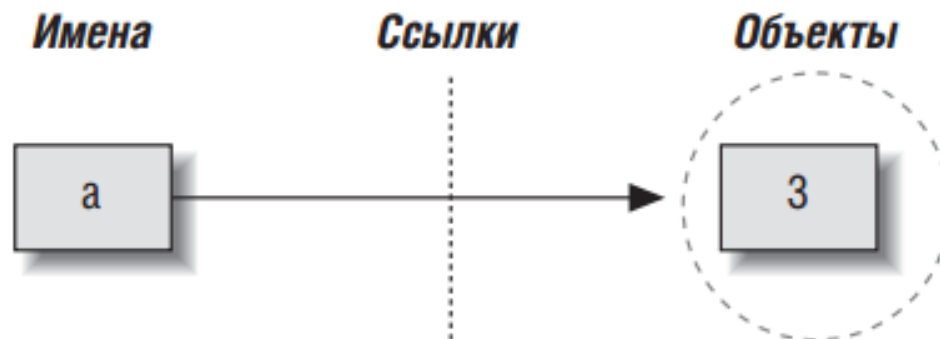
```
>>> a = 3
```

интерпретатор Python выполнит эту инструкцию в три этапа (по крайней мере, концептуально):

- Создается объект, представляющий число 3.
- Создается переменная *a*, если она еще отсутствует.
- В переменную *a* записывается ссылка (адрес в памяти) на вновь созданный объект, представляющий число 3.

Переменные, объекты и динамическая типизация

Переменные и объекты хранятся в разных частях памяти и связаны между собой ссылкой (ссылка на рисунке показана в виде стрелки). Переменные всегда ссылаются на объекты и никогда – на другие переменные, но крупные объекты могут ссылаться на другие объекты (например, объект списка содержит ссылки на объекты, которые включены в список). Когда бы ни использовалась переменная (то есть ссылка), интерпретатор Python автоматически переходит по ссылке от переменной к объекту.



Переменные, объекты и динамическая типизация

С точки зрения определений все это выглядит так:

Переменные — это записи в системной таблице, где предусмотрено место для хранения ссылок на объекты.

Объекты — это области памяти с объемом, достаточным для представления значений этих объектов.

Ссылки — это автоматически разыменовываемые указатели на объекты.

Python, создавая новое значение, всегда выделяет новый объект, то есть выделяет участок памяти. Но иногда Python хранит в кэше некоторые значения, оптимизируя все. Так, любая 1 или 97, используемая в программе - это один и тот же объект. А 99998 в одном месте и то же число в другом - не одно и то же.

Переменные, объекты и динамическая типизация

Нужно понимать, что внутри у стандартного Python сишный код. Так, например, в базовом варианте выглядит любой объект:

```
typedef struct _object
{
    Py_ssize_t ob_refcnt; // счетчик ссылок
    struct _typeobject *ob_type; // тип объекта
} PyObject;
```

Счётчик ссылок — это число, показывающее, сколько раз другие объекты ссылаются на данный объект, или сколько переменных хранят этот объект.

```
>>> a = b = c = object() # счетчик объекта увеличивается на три
```

Когда счетчик становится равен 0, сборщик мусора СМ удаляет объект и высвобождает память.

Важно помнить - в Python все и всегда передается по ссылке. Но неизменяемые (immutable) типы ведут себя так, как будто передаются по значению.

Переменные, объекты и динамическая типизация

Нужно понимать, что внутри у стандартного Python сишный код. Так, например, в базовом варианте выглядит любой объект:

```
typedef struct _object
{
    Py_ssize_t ob_refcnt; // счетчик ссылок
    struct _typeobject *ob_type; // тип объекта
} PyObject;
```

Счётчик ссылок — это число, показывающее, сколько раз другие объекты ссылаются на данный объект, или сколько переменных хранят этот объект.

```
>>> a = b = c = object() # счетчик объекта увеличивается на три
```

Когда счетчик становится равен 0, сборщик мусора СМ удаляет объект и высвобождает память.

Важно помнить - в Python все и всегда передается по ссылке. Но неизменяемые (immutable) типы ведут себя так, как будто передаются по значению.

Сборщик мусора

- Сборщик мусора имеет три поколения, при создании объекта он попадает в нулевое поколение. У каждого поколения есть счётчик (счетчик количества объектов) и порог. Работает эта пара так:
- При добавлении объекта в поколение счётчик увеличивается.
- При выбывании из поколения счётчик уменьшается.
- Когда счётчик превысит пороговое значение — по всем объектам из поколения пройдёт сборщик мусора. Кого найдёт — удалит.
- Все выжившие в поколении объекты перемещаются в следующее (из нулевого в первое, из первого во второе). Из второго поколения объекты никуда не попадают и остаются там навечно.
- Перемещённые в следующее поколение объекты меняют соответствующий счётчик, и операция может повториться уже для следующего поколения.
- Счётчик текущего поколения сбрасывается.

Арифметические операторы

Оператор	Название	Объяснение	Примеры
+	Сложение	Суммирует два объекта	3 + 5 даст 8; 'a' + 'b' даст 'ab'
-	Вычитание	Даёт разность двух чисел; если первый операнд отсутствует, он считается равным нулю	-5.2 даст отрицательное число, а 50 - 24 даст 26.
*	Умножение	Даёт произведение двух чисел или возвращает строку, повторённую заданное число раз.	2 * 3 даст 6. 'a' * 3 даст 'lalala'.
**	Возведение в степень	Возвращает число x, возведённое в степень y	3 ** 4 даст 81 (т.е. 3 * 3 * 3 * 3)
/	Деление	Возвращает частное от деления x на y	4 / 3 даст 1.3333333333333333.
//	Целочисленное деление	Возвращает неполное частное от деления	4 // 3 даст 1.
%	Деление по модулю	Возвращает остаток от деления	8 % 3 даст 2. -25.5 % 2.25 даст 1.5.

Побитовые операторы

Оператор	Название	Объяснение	Примеры
<<	Сдвиг влево	Сдвигает биты числа влево на заданное количество позиций.	$2 \ll 2$ даст 8. В двоичном виде 2 - 10. Сдвиг влево на 2 бита даёт 1000, в десятичном виде 8.
>>	Сдвиг вправо	Сдвигает биты числа вправо на заданное число позиций.	$11 \gg 1$ даст 5. В двоичном виде 11 - 1011, что будучи смещённым на 1 бит вправо, даёт 101, а это десятичное 5
&	Побитовое И	Побитовая операция И над числами	$5 \& 3$ даёт 1.
	Побитовое ИЛИ	Побитовая операция ИЛИ над числами	$5 3$ даёт 7
^	Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ	Побитовая операция ИСКЛЮЧАЮЩЕЕ ИЛИ	$5 \wedge 3$ даёт 6
~	Побитовое НЕ	Побитовая операция НЕ для числа x соответствует $-(x+1)$	~ 5 даёт -6.

Операторы сравнения

Оператор	Название	Объяснение	Примеры
<	Меньше	Определяет, верно ли, что x меньше y . Все операторы сравнения возвращают True или False.	<p>$5 < 3$ даст False,</p> <p>Можно составлять произвольные цепочки сравнений: $3 < 5 < 7$ даёт True.</p>
>	Больше	Определяет, верно ли, что x больше y	$5 > 3$ даёт True. Если оба операнда - числа, то оба преобразуются к одинаковому типу. Иначе оба возвращаются False.
<=	Меньше равно	Определяет, верно ли, что x меньше или равно y	$x = 3; y = 6; x <= y$ даёт True.
>=	Больше равно	Определяет, верно ли, что x больше или равно y	$x = 4; y = 3; x >= 3$ даёт True.
==	Равно	Равны ли объекты	$x = 2; y = 2; x == y$ даёт True. $x = 'str'; y = 'str'; x == y$ даёт True. $x = 2; y = '2'; x == y$ даёт False.

Логические операторы

Оператор	Название	Объяснение	Примеры
not	Логическое НЕ	Если <code>x == True</code> , вернётся <code>False</code> . Если <code>x</code> равно <code>False</code> , получим <code>True</code> .	<code>x = True</code> ; <code>not x</code> даёт <code>False</code> .
and	Логическое И	<code>x and y</code> даёт <code>False</code> , если <code>x</code> равно <code>False</code> , в противном случае возвращает значение <code>y</code>	<code>x = 0</code> ; <code>y = 1</code> ; <code>x and y</code> возвращает <code>0</code> , поскольку <code>x</code> равно <code>0</code> .
or	Логическое ИЛИ	Если <code>x</code> равно <code>True</code> , в результате получим <code>True</code> , в противном случае получим значение <code>y</code>	<code>x = True</code> ; <code>y = False</code> ; <code>x or y</code> даёт <code>True</code> . Здесь также может производиться укороченная оценка выражений.

Операторы идентичности и включения

Оператор	Название	Объяснение	Примеры
is	Идентичность	x is y даёт True, если x и y - это один и тот же объект, а не просто имеют одинаковые значения	x = y = [1,2,3]; x is y вернет True, при x = [1,2,3], y = [1,2,3] будет False.
is not	Неидентичность	x is not y даёт False, если x и y - это один и тот же объект.	x = y = [1,2,3]; x is not y вернет False, при x = [1,2,3], y = [1,2,3] будет True.
in	Проверяет наличие	x in y даёт True, если x присутствует внутри y	x = 1; y = [1,2,3,4,5,6,7,8,9,10]; x in y вернет True
not in	Проверяет отсутствие	x not in y даёт True, если x отсутствует внутри y	x = 1; y = [1,2,3,4,5,6,7,8,9,10]; x not in y вернет False

Типы данных

В Python всего 2 простых типа данных:

- число (256),
- строка (“256”).

Остальные типы: кортеж, список, множество, словарь, - являются составными, т.е. представляют собой набор значений простых типов, упорядоченных определенным образом.

Также Python включает в себя специальные типы: None (для обозначения неинициализированных переменных), NotImplemented (для информирования об определенных ошибках) и Ellipsis (то же, что ‘...’ - в срезах массивов, в качестве заглушек в функциях).

Еще есть тип File для работы с файлами и все пользовательские типы, обозначаемые ключевым словом Class – классы.

Число (Number) - Immutable

- Обычные целые числа: `int('5')`, 1, 5
- Восьмеричные числа(тоже `int`): 020
- Шестнадцатеричные числа: 0xA, 0xa
- Числа с плавающей точкой: `float('1.4')`, 1.4, 1e7
- Комплексные числа: `complex(1, 2)`, 5j, 5J, 3+4j, z.real, z.imag
- Логический (булев) тип: True, False

Логический тип в Python не выделяется в отдельный тип, т.к. считается одним из представлений типа Число. Логические выражения, имеющие значение True, считаются равными 1. Логические выражения, имеющие значение False, считаются равными 0.

Ноль, пустая строка (''), специальное значение 'None' считаются Ложью (False). Все остальное – Истина (True).

Функции для работы с числами

С числами связаны следующие операции, кроме перечисленных ранее:

`int(x [,base])` - преобразует `x` к целочисленному типу, `base` - основание системы счисления, если `x` - строка.

`float(x)` - преобразует `x` к числу с плавающей точкой (вещественному числу).

`complex(real [,imag])` – создает комплексное число

`ord(x)` – преобразует отдельный символ к целому числу согласно таблице ASCII.

`hex(x)` – преобразует число к 16-ричному виду в строковом представлении

`oct(x)` – преобразует число к 8-ричному виду в строковом представлении

`round(n[, ndigits])` – округляет вещественное число до указанного разряда после запятой (по умолчанию до нулевого, т.е. до целой части).

Примеры работы с числами

Примеры для практики:

```
>>> print(2 ** 150)
1427247692705959881058285969449495136382746624
>>> print(int(5.89), round(5.89), round(5.89, 1))
(5 6.0 5.9)
>>> import math
>>> math.pi
3.1415926535897931
>>> print math.sin(math.pi/2)
1.0
```

Строка (String) - Immutable

Фактически строка состоит из отдельных символов, но в Python нет понятия символа (это тоже строка только из одного элемента).

Доступ к элементам строки или подстрокам осуществляется по индексам или путем указания среза:

```
s1 = 'Hello World!'
s2 = "Python Programming!"
print "s1[0]: ", s1[0]
print "s2[1:5]: ", s2[1:5]
```

```
s1[0]: H
s2[1:5]: ytho
```

Строка является неизменяемой (immutable) последовательностью. Т.е. нельзя, к примеру, заменить какой-либо элемент в строке:

```
>>> s1[1] = 'a'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```


Работа со строками

Зато можно создать новую строку с использованием старой:

```
>>> s3 = s1[0] + 'a' + s1[2:len(s1)]  
>>> print s3  
Hallo World
```

Обычные строки хранятся из расчета 8 бит на символ. Это достаточно для кодирования всех символов таблицы ASCII. Для использования спецсимволов и различных языков мира это недостаточно. Для этого были добавлены unicode строки, в которых под символ отводится 16 бит.

Unicode строка задается следующим образом:

```
us = u'Hello World'
```

Преобразование обычной строки в Unicode строку выполняется так:

```
us1 = unicode(s1)
```

Преобразование из другого типа в строку выполняется так:

```
>>> print str(us1)  
Hello World  
>>> print str(5)+str(2)  
52
```

Строковые операторы

```
a = 'Hello'
b = 'Python'
```

Оператор	Описание	Пример
+	Конкатенация. Соединяет две строки в третью	a + b даст HelloPython
*	Повторение. Новая строка получается повторением исходной заданное количество раз.	a*2 даст HelloHello
[]	Индекс. Возвращает символ строки по индексу (начиная с 0).	a[1] даст e
[:]	Срез. Позволяет получить подстроку с начального индекса по конечный (можно указать шаг).	a[1:4] даст ell
in	Включение. Возвращает Истину, если указанный символ присутствует в строке.	H in a даст 1
not in	Не включение. Возвращает истину, если указанный символ отсутствует в строке.	M not in a даст 1
r/R	«Сырая» строка. Подавляет Escape-символы. Обозначается добавлением r или R перед открывающей кавычкой.	print r'\n' выводит \n and print R'\n' выводит \n
%	Форматирование. Выполняет форматирование строки.	

Встроенные методы строк и функции для работы со строками

Методы	Описание
<code>capitalize()</code>	Делает заглавной первую букву строки.
<code>center(width, fillchar)</code>	Центрирует строку по указанной ширине, заполняя оставшиеся места пробелами (по умолчанию), либо символом <code>fillchar</code> .
<code>count(str, beg=0, end=len(string))</code>	Считает, сколько раз подстрока <code>str</code> встречается в строке, начиная с индекса <code>beg</code> по индекс <code>end</code> .
<code>decode(encoding='UTF-8', errors='strict')</code>	Преобразует строку из указанной кодировки в кодировку по умолчанию.
<code>encode(encoding='UTF-8', errors='strict')</code>	Преобразует строку из кодировки по умолчанию в указанную кодировку.
<code>endswith(suffix, beg=0, end=len(string))</code>	Определяет заканчивается ли строка (или подстрока, если заданы <code>beg</code> и <code>end</code>) заданным суффиксом.
<code>expandtabs(tabsize=8)</code>	Заменяет символы табуляции на указанное число пробелов.
<code>find(str, beg=0 end=len(string))</code>	Определяет, встречается ли подстрока <code>str</code> в строке (или подстроке, если заданы <code>beg</code> и <code>end</code>); возвращает начальный индекс найденной подстроки, либо <code>-1</code> , если ничего не найдено.
<code>index(str, beg=0, end=len(string))</code>	Так же как <code>find()</code> , но выбрасывает исключение, если <code>str</code> не найдена.
<code>isalnum()</code>	Возвращает <code>True</code> , если строка содержит по крайней мере один символ, и все символы в строке - либо цифры, либо буквы. <code>False</code> в противном случае.

Встроенные методы строк и функции для работы со строками

Методы	Описание
isalpha()	Возвращает True, если строка содержит по крайней мере один символ, и все символы в строке - буквы. False в противном случае.
isdigit()	Возвращает True, если все символы в строке - цифры. False в противном случае.
islower()	Возвращает True, если строка содержит по крайней мере один символ, и все символы в строке строчные (в нижнем регистре). False в противном случае.
isnumeric()	Возвращает True, если все символы в unicode-строке - численные. False в противном случае.
isspace()	Возвращает True, если строка содержит только пробелы. False в противном случае.
istitle()	Возвращает True, если все слова в строке начинаются с заглавной буквы. False в противном случае.
isupper()	Возвращает True, если строка содержит по крайней мере один символ, который можно привести к верхнему и нижнему регистру, и все символы в строке прописные (в верхнем регистре). False в противном случае.
join(seq)	Формирует строку из элементов последовательности seq, разделителем является строка, чей метод вызван.
len(string)	Возвращает длину строки.
ljust(width[, fillchar])	Выравнивает строку по левому краю, до ширины width заполняя оставшиеся места пробелами (по умолчанию), либо символом fillchar.
lower()	Преобразует все прописные буквы в строке в строчные.
lstrip()	Удаляет все пробелы из начала строки.
maketrans()	Возвращает таблицу преобразования для использования в функции translate преобразования.
max(str)	Возвращает максимальный по алфавитному порядку символ в строке.

Встроенные методы строк и функции для работы со строками

Методы	Описание
<code>min(str)</code>	Возвращает минимальный по алфавитному порядку символ в строке.
<code>replace(old, new [, max])</code>	Замещает все (либо не больше <code>max</code> , если <code>max</code> задан) подстроки <code>old</code> в строке на подстроку <code>new</code> .
<code>rfind(str, beg=0, end=len(string))</code>	Так же как <code>find()</code> , только при поиске проходит строку в обратном направлении.
<code>rindex(str, beg=0, end=len(string))</code>	Также как <code>index()</code> , только при поиске проходит строку в обратном направлении.
<code>rjust(width[, fillchar])</code>	Выравнивает строку по правому краю, до ширины <code>width</code> заполняя оставшиеся места пробелами (по умолчанию), либо символом <code>fillchar</code> .
<code>rstrip()</code>	Удаляет все пробелы в конце строки.
<code>split(str="", num=string.count(str))</code>	Разделяет строку по символу <code>str</code> (по умолчанию, пробел) и возвращает список подстрок. Если указан <code>num</code> , то разделяет не больше, чем на <code>num</code> подстрок.
<code>splitlines(num=string.count("\n"))</code>	Разделяет строку по символу перевода строки (<code>\n</code>) и возвращает список подстрок (уже без <code>\n</code>). Если указан <code>num</code> , то разделяет не больше, чем на <code>num</code> подстрок.
<code>startswith(str, beg=0, end=len(string))</code>	Определяет начинается ли строка (или подстрока, если заданы <code>beg</code> и <code>end</code>) с подстроки <code>str</code> .
<code>strip([chars])</code>	Выполняет сразу и <code>lstrip()</code> , и <code>rstrip()</code> над строкой.

Встроенные методы строк и функции для работы со строками

Методы	Описание
<code>swapcase()</code>	Инвертирует регистр для всех букв в строке.
<code>title()</code>	Возвращает строку, в которой все слова начинаются с заглавной буквы.
<code>translate(table, deletechars="")</code>	Преобразует строку в соответствии с таблицей преобразования, удаляя символы в списке <code>deletechars</code> .
<code>upper()</code>	Преобразует строчные буквы в строке в заглавные.
<code>zfill (width)</code>	Заполняет строку нулями с начала до ширины <code>width</code> .
<code>isdecimal()</code>	Возвращает <code>True</code> , если все символы в <code>unicode</code> -строке - цифры. <code>False</code> в противном случае.

Оператор условия IF

```
if <логическое выражение>:  
    <блок кода>  
elif <логическое выражение>:  
    <блок кода>  
...  
else:  
    <блок кода, который должен выполняться, если остальные блоки не выполнялись>
```

```
if <логическое выражение>:  
    <блок кода>  
else:  
    <блок кода, который должен выполняться, если остальные блоки не выполнялись>
```

```
if <логическое выражение>:  
    <блок кода>  
elif <логическое выражение>:  
    <блок кода>
```

```
if <логическое выражение>:  
    <блок кода>
```



Оператор условия IF

```
number = 23
guess = int(input('Введите целое число : '))

if guess == number:
    print('Поздравляю, вы угадали') # Здесь начинается и заканчивается новый блок
elif guess < number:
    print('Нет, загаданное число немного больше этого.') # Ещё один блок
else:
    print('Нет, загаданное число немного меньше этого.')
    # чтобы попасть в else, guess должно быть больше, чем number
print('Завершено')
# Это последнее выражение выполняется всегда после выполнения оператора if
```

```
$ python if.py
Введите целое число : 50
Нет, загаданное число немного меньше этого.
Завершено
$ python if.py
Введите целое число : 22
Нет, загаданное число немного больше этого.
Завершено
$ python if.py
Введите целое число : 23
Поздравляю, вы угадали
Завершено
```

Оператор цикла WHILE

```
while <логическое выражение>:
    <тело цикла>
else <логическое выражение>:
    <блок кода, который должен выполняться, если тело цикла не
    выполнилось>
```

```
while <логическое выражение>:
    <тело цикла>
```

```
>>> while i != 5:
...     i += 1
...
>>> print(i)
5
```

```
>>> i = 0
>>> while i != 5:
...     i += 1
...     if i == 3:
...         break
...
>>> print(i)
3
```

Оператор цикла WHILE

```
>>> i = 0
>>> while i != 5:
...     i += 1
... else:
...     print('Цикл дошел до конца!')
```

Цикл дошел до конца!

```
>>> print(i)
5
```

```
>>> i = 0
>>> while i != 5:
...     i += 1
...     if i % 2:
...         print(i)
...     else:
...         continue
... 
```

```
1
3
5
>>> print(i)
5
```

Оператор цикла FOR

```
for <элемент> in <последовательность>:
    <тело цикла>
```

```
>>> for i in range(1, 5):
...     print(i)
...
1
2
3
4
```

```
>>> l = [1,2,3,4]
>>> for i in l:
...     print(i)
...
1
2
3
4
```

```
>>> for i in [1,2,3,4]:
...     print(i * 2)
...
2
4
6
8
```