Синтаксис Python

## **Отступы Python**

Отступы относятся к пробелам в начале строки кода.

Если в других языках программирования отступы в коде предназначены только для удобства чтения, отступы в Python очень важны.

Python использует отступ для обозначения блока кода.

### **Пример**

if 5 > 2:  
  print("Five is greater than two!")

Python выдаст вам ошибку, если вы пропустите отступ:

### **Пример**

Ошибка синтаксиса:

if 5 > 2:  
print("Five is greater than two!")

Количество пробелов зависит от вас как программиста, но оно должно быть не менее одного.

### **Пример**

if 5 > 2:  
 print("Five is greater than two!")   
if 5 > 2:  
        print("Five is greater than two!")

Вы должны использовать одинаковое количество пробелов в одном блоке кода, иначе Python выдаст ошибку:

### **Пример**

Ошибка синтаксиса:

if 5 > 2:  
 print("Five is greater than two!")  
        print("Five is greater than two!")

## **Встроенные типы данных**

В программировании тип данных - важное понятие.

Переменные могут хранить данные разных типов, и разные типы могут делать разные вещи.

Python по умолчанию имеет следующие типы данных, встроенные в эти категории:

|  |  |
| --- | --- |
| Тип текста: | str |
| Числовые типы: | int, float, complex |
| Типы последовательностей: | list, tuple, range |
| Тип отображения: | dict |
| Типы наборов: | set, frozenset |
| Логический тип: | bool |
| Бинарные типы: | bytes, bytearray, memoryview |

## **Получение типа данных**

Вы можете получить тип данных любого объекта, используя type()функцию:

### **Пример**

Выведите тип данных переменной x:

x = 5  
print(type(x))

## **Установка типа данных**

В Python тип данных устанавливается, когда вы присваиваете значение переменной:

|  |  |  |  |
| --- | --- | --- | --- |
| **Example** | | **Data Type** | **Try it** |
| x = "Hello World" | | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str) |
| x = 20 | | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int) |
| x = 20.5 | | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float) |
| x = 1j | | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex) |
| x = ["apple", "banana", "cherry"] | | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list) |
| x = ("apple", "banana", "cherry") | | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple) |
| x = range(6) | | range | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range) |
| x = {"name" : "John", "age" : 36} | | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict) |
| x = {"apple", "banana", "cherry"} | | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set) |
| x = frozenset({"apple", "banana", "cherry"}) | | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset) |
| x = True | bool | | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool) |
| x = b"Hello" | | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes) |
| x = bytearray(5) | | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray) |
| x = memoryview(bytes(5)) | | memoryview |  |

## **Числа Python**

В Python есть три числовых типа:

* int
* float
* complex

Переменные числового типа создаются, когда вы присваиваете им значение:

### **Пример**

x = 1    # int  
y = 2.8  # float  
z = 1j   # complex

Чтобы проверить тип любого объекта в Python, используйте type()функцию:

### **Пример**

print(type(x))  
print(type(y))  
print(type(z))

## **Плавать**

Число с плавающей запятой или «число с плавающей запятой» - это положительное или отрицательное число, содержащее один или несколько десятичных знаков.

### **Пример**

Плавает:

x = 1.10  
y = 1.0  
z = -35.59  
  
print(type(x))  
print(type(y))  
print(type(z))

Float также может быть научным числом с буквой "e", обозначающей степень 10.

### **Пример**

Плавает:

x = 35e3  
y = 12E4  
z = -87.7e100  
  
print(type(x))  
print(type(y))  
print(type(z))

## **Сложный**

Комплексные числа записываются с мнимой частью "j":

### **Пример**

Сложный:

x = 3+5j  
y = 5j  
z = -5j  
  
print(type(x))  
print(type(y))  
print(type(z))

## **Преобразование типов**

Вы можете конвертировать из одного типа в другой с int(), float()и complex()методы:

### **Пример**

Преобразование из одного типа в другой:

x = 1    # int  
y = 2.8  # float  
z = 1j   # complex  
  
#convert from int to float:  
a = float(x)  
  
#convert from float to int:  
b = int(y)  
  
#convert from int to complex:  
c = complex(x)  
  
print(a)  
print(b)  
print(c)  
  
print(type(a))  
print(type(b))  
print(type(c))

## **Случайное число**

В Python нет random()функции для создания случайных чисел, но в Python есть встроенный модуль, randomкоторый можно использовать для создания случайных чисел:

### **Пример**

Импортируйте случайный модуль и отобразите случайное число от 1 до 9:

import random  
  
print(random.randrange(1, 10))

# Кастинг на Python

## **Укажите тип переменной**

Бывают случаи, когда вы хотите указать тип переменной. Это можно сделать с помощью литья. Python - это объектно-ориентированный язык, и поэтому он использует классы для определения типов данных, включая его примитивные типы.

Поэтому преобразование в Python выполняется с использованием функций-конструкторов:

* int () - создает целое число из целочисленного литерала, литерала с плавающей запятой (путем удаления всех десятичных знаков) или строкового литерала (при условии, что строка представляет собой целое число)
* float () - создает число с плавающей запятой из целочисленного литерала, литерала с плавающей запятой или строкового литерала (при условии, что строка представляет собой число с плавающей запятой или целое число)
* str () - создает строку из самых разных типов данных, включая строки, целочисленные литералы и литералы с плавающей запятой.

### **Пример**

**Целые числа:**

x = int(1)   # x will be 1  
y = int(2.8) # y will be 2  
z = int("3") # z will be 3

**Плавает:**

x = float(1)     # x will be 1.0  
y = float(2.8)   # y will be 2.8  
z = float("3")   # z will be 3.0  
w = float("4.2") # w will be 4.2

**Струны:**

x = str("s1") # x will be 's1'  
y = str(2)    # y will be '2'  
z = str(3.0)  # z will be '3.0'

Строки Python

## **Струны**

Строки в Python заключаются в одинарные или двойные кавычки.

«привет» - это то же самое, что «привет» .

Вы можете отобразить строковый литерал с помощью print()функции:

### **Пример**

print("Hello")  
print('Hello')

## **Назначить строку переменной**

Присваивание строки переменной выполняется с именем переменной, за которым следует знак равенства и строка:

### **Пример**

a = "Hello"  
print(a)

## **Строки - это массивы**

Как и во многих других популярных языках программирования, строки в Python представляют собой массивы байтов, представляющих символы Юникода.

Однако в Python нет символьного типа данных, одиночный символ - это просто строка длиной 1.

Квадратные скобки могут использоваться для доступа к элементам строки.

### **Пример**

Получить символ в позиции 1 (помните, что первый символ имеет позицию 0):

a = "Hello, World!"  
print(a[1])

## **Цикл по строке**

Поскольку строки представляют собой массивы, мы можем перебирать символы в строке с помощью forцикла.

### **Пример**

Прокрутите буквы в слове «банан»:

for x in "banana":  
  print(x)

## **Проверить строку**

Чтобы проверить, присутствует ли в строке определенная фраза или символ, мы можем использовать ключевое слово in.

### **Пример**

Проверьте, присутствует ли слово «бесплатно» в следующем тексте:

txt = "The best things in life are free!"  
print("free" in txt)

Используйте это в if заявлении:

### **Пример**

Печатать только при наличии слова «бесплатно»:

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

## **Проверьте, НЕ**

Чтобы проверить, НЕ присутствует ли в строке определенная фраза или символ, мы можем использовать ключевое слово not in.

### **Пример**

Проверьте, НЕ присутствует ли "дорого" в следующем тексте:

txt = "The best things in life are free!"  
print("expensive" not in txt)

**Используйте это в ifзаявлении:**

### **Пример**

печатать, только если "дорого" НЕТ:

txt = "The best things in life are free!"  
if "expensive" not in txt:  
  print("No, 'expensive' is NOT present.")

# Python - нарезка строк

## **Нарезка**

Вы можете вернуть диапазон символов, используя синтаксис среза.

Укажите начальный и конечный индексы, разделенные двоеточием, чтобы вернуть часть строки.

### **Пример**

Получите символы из позиции 2 в позицию 5 (не входят в комплект):

b = "Hello, World!"  
print(b[2:5])

## **Отрицательное индексирование**

Используйте отрицательные индексы, чтобы начать срез с конца строки:

### **Пример**

Получите персонажей:

Откуда: "о" в "Мире!" (позиция -5)

К, но не входит: "d" в "Мира!" (позиция -2):

b = "Hello, World!"  
print(b[-5:-2])

Python - изменение строк

## **Верхний регистр**

### **Пример**

upper()Метод возвращает строку в верхнем регистре:

a = "Hello, World!"  
print(a.upper())

## **Нижний регистр**

### **Пример**

lower()Метод возвращает строку в нижнем регистре:

a = "Hello, World!"  
print(a.lower())

## **Удалить пробелы**

Пробел - это пробел до и / или после фактического текста, и очень часто вы хотите удалить это пространство.

### **Пример**

strip()Метод удаляет пробельные символы в начале или в конце:

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

## **Заменить строку**

### **Пример**

replace()Метод заменяет строку с другой строкой:

a = "Hello, World!"  
print(a.replace("H", "J"))

## **Разделить строку**

split()Метод возвращает список , в котором текст , расположенный между указанным разделителем становится элементы списка.

### **Пример**

split()Метод разбивает строку на подстроки , если он находит примеры сепаратора:

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

# Python - конкатенация строк

## **Конкатенация строк**

Чтобы соединить или объединить две строки, вы можете использовать оператор +.

### **Пример**

Объединить переменную aс переменной bв переменную c:

a = "Hello"  
b = "World"  
c = a + b  
print(c)

### **Пример**

Чтобы добавить пробел между ними, добавьте " ":

a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)

# Python - Формат – Строки

## **Струны**

Строки в Python заключаются в одинарные или двойные кавычки.

«привет» - это то же самое, что «привет» .

Вы можете отобразить строковый литерал с помощью print()функции:

### **Пример**

print("Hello")  
print('Hello')

## **Назначить строку переменной**

Присваивание строки переменной выполняется с именем переменной, за которым следует знак равенства и строка:

### **Пример**

a = "Hello"  
print(a)

## **Многострочные строки**

Вы можете присвоить переменной многострочную строку, используя три кавычки:

### **Пример**

Вы можете использовать три двойные кавычки:

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

Или три одинарные кавычки:

### **Пример**

a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)

## **Строки - это массивы**

Как и во многих других популярных языках программирования, строки в Python представляют собой массивы байтов, представляющих символы Юникода.

Однако в Python нет символьного типа данных, отдельный символ - это просто строка длиной 1.

Квадратные скобки могут исполь

### **Пример**

Получить символ в позиции 1 (помните, что первый символ имеет позицию 0):

a = "Hello, World!"  
print(a[1])

## **Цикл по строке**

Поскольку строки представляют собой массивы, мы можем перебирать символы в строке с помощью forцикла.

### **Пример**

Прокрутите буквы в слове «банан»:

for x in "banana":  
  print(x)

## **Длина строки**

Чтобы получить длину строки, используйте len()функцию.

### **Пример**

len()Функция возвращает длину строки:

a = "Hello, World!"  
print(len(a))

## **Проверить строку**

Чтобы проверить, присутствует ли в строке определенная фраза или символ, мы можем использовать ключевое слово in.

### **Пример**

Проверьте, присутствует ли слово «бесплатно» в следующем тексте:

txt = "The best things in life are free!"  
print("free" in txt)

Используйте это в ifзаявлении:

### **Пример**

Печатать только при наличии слова «бесплатно»:

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

## **Проверьте, НЕ**

Чтобы проверить, НЕ присутствует ли в строке определенная фраза или символ, мы можем использовать ключевое слово not in.

### **Пример**

Проверьте, НЕ присутствует ли "дорого" в следующем тексте:

txt = "The best things in life are free!"  
print("expensive" not in txt)

Используйте это в ifзаявлении:

### **Пример**

печатать, только если "дорого" НЕТ:

txt = "The best things in life are free!"  
if "expensive" not in txt:  
  print("No, 'expensive' is NOT present.")

# Python - нарезка строк

## **Нарезка**

Вы можете вернуть диапазон символов, используя синтаксис среза.

Укажите начальный и конечный индексы, разделенные двоеточием, чтобы вернуть часть строки.

### **Пример**

Получите символы из позиции 2 в позицию 5 (не входят в комплект):

b = "Hello, World!"  
print(b[2:5])

## **Нарезать с самого начала**

Если не указывать начальный индекс, диапазон будет начинаться с первого символа:

### **Пример**

Получите символы от начала до позиции 5 (не входит в комплект):

b = "Hello, World!"  
print(b[:5])

Hello

## **Отрицательное индексирование**

Используйте отрицательные индексы, чтобы начать срез с конца строки:

### **Пример**

Получите персонажей:

Откуда: "о" в "Мире!" (позиция -5)

К, но не входит: "d" в "Мира!" (позиция -2):

b = "Hello, World!"  
print(b[-5:-2])

orl

# Python - изменение строк

Python имеет набор встроенных методов, которые вы можете использовать для строк.

## **Верхний регистр**

### **Пример**

upper()Метод возвращает строку в верхнем регистре:

a = "Hello, World!"  
print(a.upper())

HELLO WORLD

## **Нижний регистр**

### **Пример**

lower()Метод возвращает строку в нижнем регистре:

a = "Hello, World!"  
print(a.lower()) Hello world

## **Удалить пробелы**

Пробел - это пробел до и / или после фактического текста, и очень часто вы хотите удалить это пространство.

### **Пример**

strip()Метод удаляет пробельные символы в начале или в конце:

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

## **Заменить строку**

### **Пример**

replace()Метод заменяет строку с другой строкой:

a = "Hello, World!"  
print(a.replace("H", "J"))

## **Разделить строку**

split()Метод возвращает список , в котором текст , расположенный между указанным разделителем становится элементы списка.

### **Пример**

split()Метод разбивает строку на подстроки , если он находит примеры сепаратора:

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

# Python - конкатенация строк

## **Конкатенация строк**

Чтобы соединить или объединить две строки, вы можете использовать оператор +.

### **Пример**

Объединить переменную aс переменной bв переменную c:

a = "Hello"  
b = "World"  
c = a + b  
print(c) Helloworld

### **Пример**

Чтобы добавить пробел между ними, добавьте " ":

a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)

# Python - Формат - Строки

## **Формат строки**

Как мы узнали в главе о переменных Python, мы не можем комбинировать строки и числа следующим образом:

### **Пример**

age = 36  
txt = "My name is John, I am " + age  
print(txt)

Но мы можем комбинировать строки и числа с помощью format()метода!

format()Метод принимает передаваемые аргументы, форматирует их, и помещают их в строке , где заполнители {}являются:

### **Пример**

Используйте format()метод для вставки чисел в строки:

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

Метод format () принимает неограниченное количество аргументов и помещается в соответствующие заполнители:

### **Пример**

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

Вы можете использовать номера индексов, {0}чтобы убедиться, что аргументы помещены в правильные заполнители:

### **Пример**

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))

Чтобы решить эту проблему, используйте escape-символ \":

### **Пример**

Эскейп-символ позволяет использовать двойные кавычки, когда обычно это не разрешено:

txt = "We are the so-called \"Vikings\" from the north."

# Логические значения Python

Логические значения представляют одно из двух значений: True или False.

## **Логические значения**

В программировании вам часто нужно знать, есть ли выражение Trueили False.

Вы можете оценить любое выражение в Python и получить один из двух ответов, Trueили False.

Когда вы сравниваете два значения, выражение оценивается, и Python возвращает логический ответ:

### **Пример**

print(10 > 9)  
print(10 == 9)  
print(10 < 9)

### **Пример**

Распечатайте сообщение в зависимости от того, является ли условие Trueили False:

a = 200  
b = 33  
  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

## **Оценить значения и переменные**

bool()Функция позволяет оценить любое значение, и дать вам Trueили False взамен,

### **Пример**

Вычислите строку и число:

print(bool("Hello"))  
print(bool(15))

## **Некоторые значения неверны**

На самом деле, не так много значения, оценивающей к False, за исключением пустых значений, такие как (), [], {}, "", количество 0, и значение None. И, конечно же, значение Falseоценивается в False.

### **Пример**

Следующее вернет False:

bool(False)  
bool(None)  
bool(0)  
bool("")  
bool(())  
bool([])  
bool({})

Еще одно значение или объект в этом случае оценивается как False, и это если у вас есть объект, созданный из класса с \_\_len\_\_функцией, которая возвращает 0или False:

### **Пример**

class myclass():  
  def \_\_len\_\_(self):  
    return 0  
  
myobj = myclass()  
print(bool(myobj))

Вы можете выполнить код на основе логического ответа функции:

### **Пример**

Выведите «ДА!» если функция возвращает True, иначе выведите «NO!»:

def myFunction() :  
  return True  
  
if myFunction():  
  print("YES!")  
else:  
  print("NO!")

Python также имеет множество встроенных функций, которые возвращают логическое значение, например, isinstance() функцию, которая может использоваться для определения того, имеет ли объект определенный тип данных:

### **Пример**

Проверьте, является ли объект целым числом или нет:

x = 200  
print(isinstance(x, int))

## **Длина кортежа**

Чтобы определить, сколько элементов в кортеже, используйте len()функцию:

### **Пример**

Выведите количество элементов в кортеже:

thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))

# Python - доступ к элементам набора

## **Доступ к элементам**

Вы не можете получить доступ к элементам в наборе, ссылаясь на индекс или ключ.

Но вы можете перебирать элементы набора с помощью for цикла или спрашивать, присутствует ли указанное значение в наборе, используя inключевое слово.

### **Пример**

Прокрутите набор и распечатайте значения:

thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
  print(x)

### **Пример**

Проверяем, присутствует ли в наборе «банан»:

thisset = {"apple", "banana", "cherry"}  
  
print("banana" in thisset)

## **Добавить элементы**

Чтобы добавить один элемент в набор, воспользуйтесь add() методом.

### **Пример**

Добавьте элемент в набор, используя add() метод:

thisset = {"apple", "banana", "cherry"}  
  
thisset.add("orange")  
  
print(thisset) orande, apple", "banana", "cherry"

## **Добавить наборы**

Чтобы добавить элементы из другого набора в текущий набор, используйте update() метод.

### **Пример**

Добавить элементы из tropicalв thisset:

thisset = {"apple", "banana", "cherry"}  
tropical = {"pineapple", "mango", "papaya"}  
  
thisset.update(tropical)  
  
print(thisset)

# Python - удаление элементов набора

## **Убрать предмет**

Чтобы удалить элемент из набора, используйте метод remove()или discard().

### **Пример**

Удалите «банан», используя remove() метод:

thisset = {"apple", "banana", "cherry"}  
  
thisset.remove("banana")  
  
print(thisset)

# Python - наборы петель

## **Элементы цикла**

Вы можете перемещаться по заданным элементам с помощью for цикла:

### **Пример**

Прокрутите набор и распечатайте значения:

thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
  print(x)

# Python - объединение множеств

## **Соединить два набора**

Есть несколько способов объединить два или более наборов в Python.

Вы можете использовать union()метод, который возвращает новый набор, содержащий все элементы из обоих наборов, или update()метод, который вставляет все элементы из одного набора в другой:

### **Пример**

union()Метод возвращает новый набор со всеми предметами из обоих наборов:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set3 = set1.union(set2)  
print(set3)

### **Пример**

update()Метод вставляет элементы в set2 в set1:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set1.update(set2)  
print(set1)

# Словари Python

## **Словарь**

Словари используются для хранения значений данных в парах ключ: значение.

Словарь - это упорядоченный \* сборник, изменяемый и не допускающий дублирования.

## Словари написаны в фигурных скобках и имеют ключи и значения:

### **Пример**

Создайте и распечатайте словарь:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict)

## **Элементы словаря**

Элементы словаря упорядочены, изменяемы и не допускают дублирования.

Элементы словаря представлены парами ключ: значение, и на них можно ссылаться, используя имя ключа.

### **Пример**

Выведите «фирменное» значение словаря:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict["brand"])

## **Получить ключи**

keys()Метод возвращает список всех ключей в словаре.

### **Пример**

Получите список ключей:

x = thisdict.keys()

## **Удаление предметов**

Есть несколько способов удалить элементы из словаря:

### **Пример**

pop()Метод удаляет элемент с указанным именем ключа:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

# Python If ... Else

## **Условия Python и операторы If**

Python поддерживает обычные логические условия из математики:

* Равно: a == b
* Не равно: a! = B
* Меньше чем: a <b
* Меньше или равно: a <= b
* Больше чем: a> b
* Больше или равно: a> = b

Эти условия можно использовать по-разному, чаще всего в «операторах if» и циклах.

Оператор if записывается с использованием ключевого слова if .

### **Пример**

Если заявление:

a = 33  
b = 200  
if b > a:  
  print("b is greater than a")

# Циклы Python While

## **Петли Python**

В Python есть две примитивные команды цикла:

* while петли
* for петель

## **Цикл while**

С в while петли можно выполнить множество операторов , как долго , как условие истинно.

### **Пример**

Выведите i, если i меньше 6:

i = 1  
while i < 6:  
  print(i)  
  i += 1

while петля требует соответствующих переменных , чтобы быть готовыми, в этом примере мы должны определить переменную индексирования, I , которые мы установили на 1.

## **Заявление о перерыве**

С помощью оператора break мы можем остановить цикл, даже если условие while истинно:

### **Пример**

Выйдите из цикла, когда мне будет 3:

i = 1  
while i < 6:  
  print(i)  
  if i == 3:  
    break  
  i += 1

## **Заявление continue**

## С помощью оператора continue мы можем остановить текущую итерацию и продолжить следующую:

### **Пример**

Если i равно 3, перейдите к следующей итерации:

i = 0  
while i < 6:  
  i += 1  
  if i == 3:  
    continue  
  print(i)

## **Заявление else**

С помощью оператора else мы можем запустить блок кода один раз, когда условие больше не выполняется:

### **Пример**

Если условие ложно, распечатайте сообщение:

i = 1  
while i < 6:  
  print(i)  
  i += 1  
else:  
  print("i is no longer less than 6")

# Python для циклов

## **Python для циклов**

Цикл for используется для перебора последовательности (то есть списка, кортежа, словаря, набора или строки).

Это меньше похоже for ключевое слово for в других языках программирования и больше похоже на метод итератора, как в других объектно-ориентированных языках программирования.

С помощью цикла for мы можем выполнить набор операторов один раз для каждого элемента в списке, кортеже, наборе и т. Д.

### **Пример**

Распечатайте каждый фрукт в списке фруктов:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)

Цикл for не требует предварительной установки индексирующей переменной.

## **Цикл по строке**

Даже строки являются повторяемыми объектами, они содержат последовательность символов:

### **Пример**

Прокрутите буквы в слове «банан»:

for x in "banana":  
  print(x)

## **Заявление о перерыве**

С помощью оператора break мы можем остановить цикл до того, как он пройдёт через все элементы:

### **Пример**

Выйти из цикла, когда xбудет «банан»:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)  
  if x == "banana":  
    break

### **Пример**

Выйдите из цикла, когда xбудет "банан", но на этот раз разрыв будет перед печатью:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    break  
  print(x)

## **Заявление continue**

С помощью оператора continue мы можем остановить текущую итерацию цикла и продолжить следующую:

### **Пример**

Не печатайте банан:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    continue  
  print(x)

## **Функция range ()**

Чтобы перебрать набор кода заданное количество раз, мы можем использовать функцию range () ,

Функция range () возвращает последовательность чисел, начиная с 0 по умолчанию и увеличивая на 1 (по умолчанию) и заканчивая указанным числом.

### **Пример**

Используя функцию range ():

for x in range(6):  
  print(x)

Функция range () по умолчанию имеет значение 0 в качестве начального значения, однако можно указать начальное значение, добавив параметр: range (2, 6) , что означает значения от 2 до 6 (но не включая 6):

### **Пример**

Используя параметр запуска:

for x in range(2, 6):  
  print(x)

По умолчанию функция range () увеличивает последовательность на 1, однако можно указать значение приращения, добавив третий параметр: range (2, 30, **3** ) :

### **Пример**

Увеличьте последовательность на 3 (по умолчанию 1):

for x in range(2, 30, 3):  
  print(x)

## **Остальное в цикле For**

elseКлючевое слово в forцикле определяет блок кода , который будет выполняться , когда цикл завершен:

### **Пример**

Распечатайте все числа от 0 до 5 и распечатайте сообщение, когда цикл закончится:

for x in range(6):  
  print(x)  
else:  
  print("Finally finished!")

**Примечание:**else блок НЕ будет выполняться , если цикл остановлен breakзаявлением.

### **Пример**

Разорвите цикл, когда xбудет 3, и посмотрите, что произойдет с elseблоком:

for x in range(6):  
  if x == 3: break  
  print(x)  
else:  
  print("Finally finished!")

# Функции Python

Функция - это блок кода, который запускается только при вызове.

Вы можете передавать данные, называемые параметрами, в функцию.

В результате функция может возвращать данные.

## **Создание функции**

В Python функция определяется с помощью ключевого слова def :

### **Пример**

def my\_function():  
  print("Hello from a function")

## **Вызов функции**

Чтобы вызвать функцию, используйте имя функции, за которым следует скобка:

### **Пример**

def my\_function():  
  print("Hello from a function")  
  
**my\_function()**

## **Аргументы**

Информация может быть передана в функции как аргументы.

Аргументы указываются после имени функции в круглых скобках. Вы можете добавить сколько угодно аргументов, просто разделив их запятыми.

В следующем примере есть функция с одним аргументом (fname). Когда функция вызывается, мы передаем имя, которое используется внутри функции для печати полного имени:

### **Пример**

def my\_function(**fname**):  
  print(fname + " Refsnes")  
  
my\_function(**"Emil"**)  
my\_function(**"Tobias"**)  
my\_function(**"Linus"**)

## **Параметры или аргументы?**

Термины, параметр и аргумент могут использоваться для одного и того же: информации, передаваемой в функцию.

С точки зрения функции:

Параметр - это переменная, указанная в скобках в определении функции.

Аргумент - это значение, которое отправляется функции при ее вызове.

## **Количество аргументов**

По умолчанию функция должна вызываться с правильным количеством аргументов. Это означает, что если ваша функция ожидает 2 аргумента, вы должны вызвать функцию с 2 аргументами, не больше и не меньше.

### **Пример**

Эта функция ожидает 2 аргумента и получает 2 аргумента:

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil", "Refsnes")

Если вы попытаетесь вызвать функцию с 1 или 3 аргументами, вы получите ошибку:

### **Пример**

Эта функция ожидает 2 аргумента, но получает только 1:

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil")

## **Произвольные аргументы, \* args**

Если вы не знаете, сколько аргументов будет передано вашей функции, добавьте \*перед именем параметра в определении функции.

Таким образом, функция получит кортеж аргументов и сможет соответственно обращаться к элементам:

### **Пример**

Если количество аргументов неизвестно, добавьте \*перед именем параметра:

def my\_function(\*kids):  
  print("The youngest child is " + kids[2])  
  
my\_function("Emil", "Tobias", "Linus")

## **Аргументы ключевых слов**

Вы также можете отправлять аргументы с синтаксисом ключ = значение .

Таким образом, порядок аргументов не имеет значения.

### **Пример**

def my\_function(child3, child2, child1):  
  print("The youngest child is " + child3)  
  
my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

## **Аргументы произвольных ключевых слов, \*\* kwargs**

Если вы не знаете, сколько аргументов ключевого слова будет передано в вашу функцию, добавьте две звездочки: \*\*перед именем параметра в определении функции.

Таким образом, функция получит словарь аргументов и сможет соответственно обращаться к элементам:

### **Пример**

Если количество аргументов ключевого слова неизвестно, добавьте двойное значение \*\*перед именем параметра:

def my\_function(\*\*kid):  
  print("His last name is " + kid["lname"])  
  
my\_function(fname = "Tobias", lname = "Refsnes")

## **Возвращаемые значения**

Чтобы позволить функции возвращать значение, используйте return инструкцию:

### **Пример**

def my\_function(x):  
  **return 5 \* x**  
print(my\_function(3))  
print(my\_function(5))  
print(my\_function(9))

## **Заявление о пропуске**

functionопределения не могут быть пустыми, но если по какой-то причине у вас есть functionопределение без содержимого, вставьте passоператор, чтобы избежать ошибки.

### **Пример**

def myfunction():  
  pass