## Основы синтаксиса Python

Синтаксис языка Python во многом похож на синтаксис таких языков, как Perl, C и Java, но вместе с этим имеет ряд отличий от этих языков программирования. В этой статье мы рассмотрим необходимые основы этого языка программирования.

## Идентификаторы в Python:

Идентификаторы в Python это имена используемые для обозначения переменной, функции, класса, модуля или другого объекта. Идентификатор должен начинаться с буквы (от a до Z) или со знака подчеркивания (\_), после которых может идти произвольное количество букв, знаков подчеркивания и чисел (от 0 до 9).

В Python недопустимо использование знаков препинания или специальных символов, таких как @, $ или % в качестве идентификаторов. Кроме того, Python чуствителен к регистру, то есть cat и Cat это два разных имени.

В Python существует следующая договоренность для названия идентификаторов:

* Имена классов начинаются с большой буквы, все остальные идентификаторы - с маленькой.
* Использования знака подчеркивания в качестве первого символа идентификатора означает, что данный идентификатор является частным (закрытым от использования вне класса).
* Если идентификатор начинается и заканчивается двумя знаками подчеркивания (например, \_\_init\_\_ ) это означает, что он является специальным именем, определенным внутри языка.

## Зарезервированые (ключевые) слова в Python:

В данной таблице собраны все ключевые слова Python.

|  |  |  |  |
| --- | --- | --- | --- |
| and | elif | if | print |
| as | else | import | raise |
| assert | except | in | return |
| break | exec | is | try |
| class | finally | lambda | while |
| continue | for | not | which |
| def | from | or | yield |
| del | global | pass |   |

Эти зарезервированные слова нельзя использовать в качестве имени переменной или любого другого идентификатора. Все ключевые слова Python состоят только из букв в нижнем регистре. Получить список ключевых слов возможно в интерпретаторе командой

help("keywords")

## Строки и отступы:

Одна из первых особенностей Python, которая бросается в глаза программистам, начинающим изучать этот язык программирования, это то, что в нем не используются скобки для обозначения отдельных блоков кода. Вместо них в Python используются двоеточия и отступы.

Количество пробелов в отступах произвольно и выбирается каждым на свое усмотрение, однако по договоренности равняется четырем пробелам.  При этом отступ всего блока должен быть одинаковым.

Например, этот блок кода будет работать (хотя так писать не стоит):

if True:

 print "Hi"

else:

 print "Bye"

 А этот уже вызовет ошибку:

if True:

 print "Hi"

 print "Bye"

Таким образом, в Python несколько строк кода с одинаковым отступом будут формировать отдельный блок кода. Благодаря такой системе значительно повышается читаемость кода и прививается привычка писать понятно и структурировано.

## Многострочные выражения:

Выражения в Python, как правило, заканчиваются новой строкой. Однако, в этом языке программирования существует специальный символ переноса строки (\), показывающий, что с окончанием строки не заканчивается код. Например:

total = item1 + \

 item2 + \

 item3

Выражения, которые находятся внутри скобок: квадратных ( [ ] ), фигурных ( { } ) или круглых ( ( ) ) не нуждаются в символе переноса строки. Например:

days = ["Sunday", "Monday", "Tuesday",

 "Wednesday", "Thursday", "Friday",

 "Saturday"]

## Кавычки в Python:

В Python можно использовать одинарные ( ' ), двойные (") и тройные (''' или """) кавычки чтобы обозначить строчный тип данных, при этом начинаться и заканчиваться строка должна одинаковыми кавычками. Строка занимающая несколько строк кода должна быть обрамлена тройными кавычками. Например:

name = 'wasd'

description = "Some text"

biography = """ Some long text for few

 lines of code """

## Комментирование в Python:

Символ решетки (#) в Python обозначает начало комментария. Любые символы после решетки и до конца строки считаются комментариями и игнорируются интерпретатором.

Например следующий код:

# First line comment

print "Hello, Python" # second comment

Выведет только Hello, Python в консоль.

## Ввод нескольких инструкций на одной строке:

Точка с запятой ( ; ) позволяет вводить несколько инструкций на одной строке. Например:

import random; x = random.randint(); print x

**Что такое оператор?**

Говоря простым языком, в выражении 2 + 3, числа "2" и "3" называются операндами, знак "+" оператором. В языке программирования Python существуют следующие типы операторов:

* [Арифметические операторы](http://pythonicway.com/python-operators#arithmetic)
* [Операторы сравнения (реляционные)](http://pythonicway.com/python-operators#compare)
* [Операторы присваивания](http://pythonicway.com/python-operators#assign)
* [Побитовые операторы](http://pythonicway.com/python-operators#bitwise)
* [Логические операторы](http://pythonicway.com/python-operators#logic)
* [Операторы членства (Membership operators)](http://pythonicway.com/python-operators#membership)
* [Операторы тождественности (Identity operators)](http://pythonicway.com/python-operators#identity)

Рассмотрим их по порядку.

**Арифметические операторы в Python:**

Оператор Описание Примеры

|  |  |  |
| --- | --- | --- |
| + | Сложение - Суммирует значения слева и справа от оператора | 15 + 5 в результате будет 2020 + -3 в результате будет 1713.4 + 7 в результате будет 20.4 |
| - | Вычитание - Вычитает правый операнд из левого | 15 - 5 в результате будет 1020 - -3 в результате будет 2313.4 - 7 в результате будет 6.4 |
| \* | Умножение - Перемножает операнды | 5 \* 5 в результате будет 257 \* 3.2 в результате будет 22.4-3 \* 12 в результате будет -36 |
| / | Деление - Делит левый операнд на правый | 15 / 5 в результате будет 35 / 2 в результате будет 2 (В Python 2.x версии при делении двух целых чисел результат будет целое число)5.0 / 2 в результате будет 2.5 (Чтобы получить "правильный" результат хотя бы один операнд должен быть float) |
| % | Деление по модулю - Делит левый операнд на правый и возвращает остаток. | 6 % 2 в результате будет 07 % 2 в результате будет 113.2 % 5 в результате 3.2 |
| \*\* | Возведение в степень - возводит левый операнд в степень правого | 5 \*\* 2 в результате будет 252 \*\* 3 в результате будет 8-3 \*\* 2 в результате будет -9 |
| // | Целочисленное деление - Деление в котором возвращается только целая часть результата. Часть после запятой отбрасывается. | 12 // 5 в результате будет 24 // 3 в результате будет 125 // 6 в результате будет 4 |

**Операторы сравнения в Python:**

|  |  |  |
| --- | --- | --- |
| == | Проверяет равны ли оба операнда. Если да, то условие становится истинным. | 5 == 5 в результате будет TrueTrue == False в результате будет False"hello" == "hello" в результате будет True |
| != | Проверяет равны ли оба операнда. Если нет, то условие становится истинным. | 12 != 5 в результате будет TrueFalse != False в результате будет False"hi" != "Hi" в результате будет True |
| <> | Проверяет равны ли оба операнда. Если нет, то условие становится истинным. | 12 <> 5 в результате будет True. Похоже на оператор != |
| > | Проверяет больше ли значение левого операнда, чем значение правого. Если да, то условие становится истинным. | 5 > 2 в результате будет True.True > False в результате будет True."A" > "B" в результате будет False. |
| < | Проверяет меньше ли значение левого операнда, чем значение правого. Если да, то условие становится истинным. | 3 < 5 в результате будет True.True < False в результате будет False."A" < "B" в результате будет True. |
| >= | Проверяет больше или равно значение левого операнда, чем значение правого. Если да, то условие становится истинным. | 1 >= 1 в результате будет True.23 >= 3.2 в результате будет True."C" >= "D" в результате будет False. |
| <= | Проверяет меньше или равно значение левого операнда, чем значение правого. Если да, то условие становится истинным. | 4 <= 5 в результате будет True.0 <= 0.0 в результате будет True.-0.001 <= -36 в результате будет False. |

**Операторы присваивания в Python:**

|  |  |  |
| --- | --- | --- |
| = | Присваивает значение правого операнда левому. | c = 23 присвоит переменной с значение 23 |
| += | Прибавит значение правого операнда к левому и присвоит эту сумму левому операнду. | с = 5а = 2 с += а равносильно: с = с + а. с будет равно 7 |
| -= | Отнимает значение правого операнда от левого и присваивает результат левому операнду. | с = 5а = 2 с -= а равносильно: с = с - а. с будет равно 3 |
| \*= | Умножает правый операнд с левым и присваивает результат левому операнду. | с = 5а = 2 с \*= а равносильно: с = с \* а. c будет равно 10 |
| /= | Делит левый операнд на правый и присваивает результат левому операнду. | с = 10а = 2 с /= а равносильно: с = с / а. c будет равно 5 |
| %= | Делит по модулю операнды и присваивает результат левому. | с = 5а = 2 с %= а равносильно: с = с % а. c будет равно 1 |
| \*\*= | Возводит в левый операнд в степень правого и присваивает результат левому операнду. | с = 3а = 2 с \*\*= а равносильно: с = с \*\* а. c будет равно 9 |
| //= | Производит целочисленное деление левого операнда на правый и присваивает результат левому операнду. | с = 11а = 2 с //= а равносильно: с = с // а. c будет равно 5 |

**Побитовые операторы в Python:**

Они есть, но рассматривать их не будем

**Логические операторы в Python:**

|  |  |  |
| --- | --- | --- |
| and | Логический оператор "И". Условие будет истинным если оба операнда истина. | True and True равно True.True and False равно False.False and True равно False.False and False равно False. |
| or | Логический оператор "ИЛИ". Если хотя бы один из операндов истинный, то и все выражение будет истинным. | True or True равно True.True or False равно True.False or True равно True.False or False равно False. |
| not | Логический оператор "НЕ". Изменяет логическое значение операнда на противоположное. | not True равно False.not False равно True. |

**Операторы членства в Python:**

В добавок к перечисленным операторам, в Python присутствуют, так называмые, операторы членства, предназначенные для проверки на наличие элемента в составных типах данных, таких, как [строки, списки, кортежи или словари](http://pythonicway.com/index.php/python-data-types):

|  |  |  |
| --- | --- | --- |
| in | Возвращает истину, если элемент присутствует в последовательности, иначе возвращает ложь. | "cad" in "cadillac" вернет True.1 in [2,3,1,6] вернет True."hi" in {"hi":2,"bye":1} вернет True. 2 in {"hi":2,"bye":1} вернет False (в словарях проверяется наличие в ключах, а не в значениях). |
| not in | Возвращает истину если элемента нет в последовательности. | Результаты противоположны результатам оператора in. |

**Приоритет операторов в Python**

В следующей таблице описан приоритет выполнения операторов в Python от наивысшего (выполняется в первую очередь) до наинизшего.

|  |  |
| --- | --- |
| \*\* | Возведение в степень |
| ~ + - | Комплиментарный оператор |
| \* / % // | Умножение, деление, деление по модулю, целочисленное деление. |
| + - | Сложение и вычитание. |
| >> << | Побитовый сдвиг вправо и побитовый сдвиг влево. |
| & | Бинарный "И". |
| ^ | | Бинарный  "Исключительное ИЛИ" и бинарный "ИЛИ" |
| <= < > >= | Операторы сравнения |
| <> == != | Операторы равенства |
| = %= /= //= -= += \*= \*\*= | Операторы присваивания |
| is is not | Тождественные операторы |
| in not in | Операторы членства |
| not or and | Логические операторы |

**Переменные в Python:**

Переменная в языке программирования это название для зарезервированного места в памяти компьютера, предназначенное для хранения значений. Это означает, что когда вы создаете переменную, вы на самом деле резервируете определенное место в памяти компьютера.

Основываясь на типе данных переменной, интерпретатор выделяет необходимое количество памяти и решает, что может находится в зарезервированной области памяти.

Для понимания, можете думать о переменной как о коробке, в которую можно положить любую вещь, но только определенного размера. Размер в данном примере будет типом переменной. Это не совсем верное определение, но оно дает общее представление о картине в целом.

**Присвоение значения переменной:**

В Python вам не нужно объявлять тип переменной вручную (как, например в С++). Объявление происходит автоматически (это называется динамическая типизация), когда вы присваиваете значение переменной. Знак равенства ( = ) используется для присвоения значения переменной.

Операнд по левую сторону от знака равно ( = ) это имя переменной, операнд по правую сторону - значение присвоенное этой переменной.

Например:

country = "Swiss" # Присвоить значение Swiss переменной под названием country

age = 23 # Присвоение значения 23 переменной age

print country

print age

При выполнении, данный код выведет:

Swiss

23

**Множественное присвоение значений:**

В Python возможно присваивать одно значение нескольким переменным сразу. Например:

a = b = c = 1

В данном создается объект со значением 1, и все 3 переменные указывают на область в памяти, в которой он находится.

**Встроенные типы данных в Python:**

Информация, сохраненная в памяти может быть разных типов данных. Например, возраст человека может быть числовым значением, а его адрес - буквенно-числовым. В Python существует множество стандартных типов данных, которые используются для хранения определенных значений и обладают своими специфическими методами.

К стандартным типам данных в Python относят:

* [Числа (Numbers)](http://pythonicway.com/python-data-types#num)
* [Строка (String)](http://pythonicway.com/python-data-types#str)
* [Список (List)](http://pythonicway.com/python-data-types#list)
* [Кортеж (Tuple)](http://pythonicway.com/python-data-types#tuple)
* [Словарь (Dictionary)](http://pythonicway.com/python-data-types#dict)
* [Сет (Set)](http://pythonicway.com/python-data-types#set)

**Числовой тип данных в Python:**

Числовой тип данных в Python предназначен для хранения числовых значений. Это неизменяемый тип данных, что означает, что изменение значения числового типа данных приведет к созданию нового объекта в памяти (и удалению старого)

Числовые объекты создаются, когда вы присваиваете им значение. Например:

num1 = 23

num2 = 42

Также вы можете удалять числовой объект при помощи ключевого слова del. Синтаксис команды del следующий:

del num1 # удаляет переменную num1

del num2, num3 # удаляет обе переменные num2 за num3 за раз

 В Python есть четыре вида числового типа данных:

* int (целое число)
* long (длинное целое число [может быть представлено в восьмеричной или шестнадцатеричной системе исчисления])
* float (число с плавающей точкой: -0.2, 0.0, 3.14159265 и т.д.)
* complex (комплексное число)

**Примеры видов числового типа данных:**

Int long float complex

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 51924361L | 0.0 | 3.14j |
| 102 | -0x19323L | 15.20 | 45.j |
| -786 | 0122L | -21.9 | 9.322e-36j |
| 0 | 0xDEFABCECBDAECBFBAEl | 32.3+e18 | .876j |
| 0b10 | 535633629843L | -90. | -.6545+0J |
| -0x260 | -052318172735L | -32.54e100 | 3e+26J |
| 0x69 | -4721885298529L | 70.2-E12 | 4.53e-7j |

**Строки в Python:**

Под строками в Python подразумевается набор символов между кавычками. В Python можно использовать пары одинарных либо двойных кавычек. Из строк можно взять подстроку используя оператор нарезки ( [ ] и [ : ] ) с индексами от нуля для первого символа строки и до последнего. Так же можно использовать обратную индексацию от -1 для последнего символа до начала.

Оператор плюс ( + ) для строк соединяет две строки в одну, звездочка ( \* ) оператор повторения. Например:

text = "Hello, Python!"

print text[0] # Выводит первый символ

print text[0:5] # Выводит подстроку text от 0 символа до 5 (включительно с нулевым, исключая пятый)

print text[4:10] # Выведет строку от 4 символа до 10 (включая четвертый, исключая 10)

print text[0:14] # Выведет всю строку

print text[7:] # Выведет строку с 7 символа до конца

print text[:5] # Выведет строку с начала до 5 символа. Аналогично print text[0:5]

print text[:] # Выведет всю строку

print text[-1] # Выводит последний символ

print text[-1:-14] # Не сработает, выведет пустую строку

print text[::2] # Третий аргумент - шаг. Выведет каждый второй символ

print text[::-1] # Шаг отрицательный. Выведет фразу наоборот

print text + "Nice to code you" # Выведет новую строку

print text[-1] \* 10 # Выведет 10 восклицательных знаков

В результате вы увидите следующее

string slicing python, нарезка строк в python

**Списки в Python:**

Списки, пожалуй, самый универсальный составной тип данных в Python. Список состоит из элементов, разделенных запятыми, находящихся между квадратными скобками ( [ ] ). В определенной мере, списки подобны массивам в C. Единственной разницей является то, что элементы одного списка могут иметь разные типы данных.

Получить доступ к элементам, сохраненным в списке можно, точно так же, как и в строках, при помощи оператора нарезки ( [ ] и [:] ) и индексов, начиная с нуля и до конца. Знак плюс ( + ) объединяет два списка, а звездочка ( \* ) - оператор повторения для списка. Например:

my\_list =[True, 786, 3.14, 'text', 70.2]

second\_list =[123, 'text']

print my\_list # Напечатает весь список

print my\_list[0] # Напечатает первый элемент списка

print my\_list[1:3] # Напечатает элементы списка со второго по третий

print my\_list[2:] # Напечатает элементы списка начиная с третьего

print second\_list \*2 # Напечатает удвоенный список

print my\_list + second\_list # Напечатает объединенные списки

В результате вы увидите:

[True, 786, 3.14, 'text', 70.2]

True

[786, 3.14]

[3.14, 'text', 70.2]

[123, 'text', 123, 'text']

[True, 786, 3.14, 'text', 70.2, 123, 'text']

**Кортежи в Python:**

Кортеж это еще один составной тип данных, похожий на список. Кортеж состоит из ряда значений, разделенных запятыми, заключенными в круглые скобки ( ( ) ). Основным различием между списками и кортежами является то, что элементы кортежей не могут быть изменены. То есть, кортежи можно рассматривать как списки доступные только для чтения.

Если у вас нет необходимости изменять элементы списка, то для экономии места в памяти лучше использовать тип данных кортеж.

my\_tuple =(True, 786, 3.14, 'text', 70.2)

second\_tuple =(123, 'text')

print my\_tuple # Печатает весь кортеж

rint my\_tuple[0] # Печатает первый элемент

print second\_tuple \*2 # Печатает удвоенный кортеж

print my\_tuple + second\_tuple # Печатает объединенные кортежи

В результате вы получите:

(True, 786, 3.14, 'text', 70.2)

True

(123, 'text', 123, 'text')

(True, 786, 3.14, 'text', 70.2, 123, 'text')

При этом, следующие действия доступны для списков и недоступны для кортежей:

my\_list = ["Rome", 23, ["cat","dog"], True, 3.14]

my\_tuple = ("Rome", 23, ["cat","dog"], True, 3.14)

my\_list[0] = "Paris" # Замена значения первого элемента сработает для списка

my\_tuple[0] = "Paris" # Та же операция для кортежа вызовет ошибку

**Словари в Python:**

Словари в Python это неотсортированная колекция элементов, доступ к которым осуществляется по ключу. То есть, каждому ключу словаря соответствует определенное значение. Ключом может быть любой неизменяемый тип данных (число, строка, кортеж), значением - любой тип данных.

Пары ключ, значение словаря заключаются в фигурные скобки ( { } ). Есть несколько способов создания словарей:

my\_dict = { } # Создаем пустой словарь

my\_dict["country"] = "Mexico" # Присваиваем ключу country значение Mexico

print my\_dict["country"] # Выведет Mexico

# Заполнение словаря при инициализации

another\_dict = {"number":23, 2: True, "my\_list":[1,2,3]}

print another\_dict.keys() # Напечатает список всех ключей

print another\_dict.values() # Напечатает список всех значений

Данный код выведет следующее:

python dictionaries, словари в python

Обратите внимание, что ключи и значения выводятся не в том порядке, в котором мы их задавали.

**Сеты в Python:**

Сет в Python это еще один изменяемый, коллекционный тип данных, отличительной чертой которого является то, что он хранит только уникальные значания.

Создать сеты можно следующими способами:

# Создание пустого сета

s = set()

# Создание сета инициализацией

s = {"hi", "bye"}

Для добавление элемента в сет используется метод add, для удаления - pop или remove. Добавление в сет уже существующего элемента не повлияет на сет. Сеты обладают множеством методов для работы с уникальными элементами, например difference - возвращает элементы сета отсутствующие в другом сете, intersection - наоборот, возвращает элементы сета присутствующие в другом сете.

python set, сеты в Python

**Преобразование типов данных:**

Иногда может возникнуть необходимость преобразовать один тип данных в другой. Для этого существуют специальные встроенные функции Python. Вот некоторые из них:

|  |  |
| --- | --- |
| int(x [,base]) | Преобразовывает х в целое число. Например, int(12.4) -> 12 |
| long(x [,base] ) | Преобразовывает х в long. Например, long(20) -> 20L |
| float(x) | Преобразовывает х в число с плавающей точкой. Например float(10) -> 10.0 |
| complex(real [,imag]) | Создает комплексное число. Например complex(20) -> (20+0j) |
| str(x) | Преобразовывает х в строку. Например str(10) -> '10' |
| tuple(s) | Преобразовывает s в кортеж. Например tuple("hello") -> ( "h","e","l","l","o" ) |
| list(s) | Преобразовывает s в список. Например list("Python") -> [ "P","y","t","h","o","n" ] |
| dict(d) | Создает словарь из d. Например dict( [ (1,2), (3,4) ] ) -> { 1:2, 3:4 } |