Коротко о Python

Содержание

Сделано по материалам с сайта Python Cheatsheet.

1 Основы Python

1.1 Математические операторы

В порядке их выполнения

Оператор	Описание	Пример
**	Возведение в степень	`2 ** 3 = 8`
%	Остаток от деления	`22 % 8 = 6`
//	Целочисленное деление	`22 // 8 = 2`
/	Деление	`22 / 8 = 2.75`
*	Умножение	`3 * 3 = 9`
-	Вычитание	`5 - 2 = 3`
+	Сложение	`2 + 2 = 4`

Примеры использования математических операторов:

```
>>> 2 + 3 * 6
20
```

```
>>> (2 + 3) * 6
30
```

```
>>> 2 ** 8
256
```

```
>>> 23 // 7
3
```

```
>>> 23 % 7
2
```

```
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

1.2 Типы данных

Тип данных	Примеры
Целые числа	`-2, -1, 0, 1, 2, 3, 4, 5`
Вещественные числа	`-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25`
Строки	`'a', 'aa', 'aaa', 'Привет!', '11 котов'`

1.3 Переменные

Имя переменной должно подчиняться следующим правилам:

1. Может состоять из одного слова (без пробелов).
2. Может состоять только из букв, цифр и символа подчёркивания (_) character.
3. Не может начинаться с цифры.

Пример:

```
>>> spam = 'Привет'
>>> spam
'Привет'
```

1.4 Комментарии

Однострочный комментарий:

```
# Это комментарий
```

Многострочный комментарий:

```
# Это
# Многострочный комментарий
```

Код с комментарием:

```
a = 1  # инициализация
```

1.5 Функция print(). Форматирование вывода

Выводит данные на экран.

```
>>> print('Привет, мир!')
Привет, мир!
```

```
>>> a = 1
>>> print('Привет, мир!', a)
Привет, мир! 1
```

Для форматирования вывода можно использовать **f-строки** (f-string):

```
age = 18
name = "Павел"
print(f"Привет, {name}. Тебе {age} лет.")
>>> Привет, Павел. Тебе 18 лет.
```

В более старых версиях Python для форматирования можно использовать функцию **format**:

```
age = 18
name = "Павел"
print(f"Привет, {0}. Тебе {1} лет.".format(name, age))
>>> Привет, Павел. Тебе 18 лет.
```

1.6 Функция input()

Ввод данных с клавиатуры.

```
>>> print('Как тебя зовут?')   # спрашиваем имя
>>> myName = input()
>>> print('Приятно познакомиться, {}'.format(myName))
```

```
Как тебя зовут?
Иван
Приятно познакомиться, Иван
```

1.7 Функции str(), int(), и float()

Преобразование целого или вещественного числа в строку:

```
>>> str(29)
'29'
```

```
>>> print('Мне {} лет.'.format(str(29)))
Мне 29 лет.
```

```
>>> str(-3.14)
'-3.14'
```

Вещественное число в целое:

```
>>> int(7.7)
7
```

```
>>> int(7.7) + 1
8
```

2 Алгоритмические конструкции

2.1 Операторы сравнения

Operator	Meaning
==	Равно
!=	Не равно
<	Меньше чем

Operator	Meaning
`>`	Больше чем
`<=`	Меньше или равно
`>=`	Больше или равно

В результате использования операторов сравнения могут получиться два значения - True (Истина) или False (Ложь).

Примеры:

```
>>> 42 == 42
True
```

```
>>> 40 == 42
False
```

```
>>> 'hello' == 'hello'
True
```

```
>>> 'hello' == 'Hello'
False
```

```
>>> 'собака' != 'кот'
True
```

```
>>> 42 == 42.0
True
```

```
>>> 42 == '42'
False
```

2.2 Логические операторы

Существует три логических оператора: and, or, и not.

Таблица истинности оператора **and**:

Выражение	Результат
`True and True`	`True`
`True and False`	`False`
`False and True`	`False`
`False and False`	`False`

Таблица истинности оператора **or**:

Выражение	Результат
`True or True`	`True`
`True or False`	`True`
`False or True`	`True`
`False or False`	`False`

Таблица истинности оператора **not**:

Выражение	Результат
`not True`	`False`
`not False`	`True`

2.3 Логические операторы и операторы сравнения

```
>>> (4 < 5) and (5 < 6)
True
```

```
>>> (4 < 5) and (9 < 6)
False
```

```
>>> (1 == 2) or (2 == 2)
True
```

2.4 Оператор if

Необходим для добавления в программу ветвления.

```
if name == 'Алиса':
    print('Привет, Алиса.')
```

2.5 Оператор else

```
name = 'Боб'
if name == 'Алиса':
    print('Привет, Алиса.')
else:
    print('Привет, Незнакомец.')
```

2.6 Оператор elif

```
name = 'Боб'
age = 5
if name == 'Алиса':
    print('Привет, Алиса.')
elif age < 12:
    print('Ты не Алиса и возраст не тот.')
```

```
name = 'Боб'
age = 30
if name == 'Алиса':
    print('Привет, Алиса.')
elif age < 12:
    print('Ты не Алиса и возраст не тот.')
else:
    print('Ничего не сходится.')
```

2.7 Цикл while

Цикл с предусловием.

```
spam = 0
while spam < 5:
    print('Привет, мир.')
    spam = spam + 1
```

2.8 Оператор break

Если интерпретатор встречает оператор break, выполнение цикла сразу же останавливается:

```
while True:
    print('Введите имя')
    name = input()
    if name == 'Павел':
        break
print('Спасибо!')
```

2.9 Оператор continue

Когда интерпретатор встречает оператор continue остальные команды внутри цикла пропускаются, цикл переходит к следующей итерации:

```
while True:
    print('Кто вы?')
    name = input()
    if name != 'Иван':
        continue
    print('Привет, Иван. Введите пароль:')
    password = input()
    if password == 'swordfish':
        break
```

2.10 Цикл for и функция range

```
>>> print('Меня зовут:')
>>> for i in range(5):
>>>     print(f'Петя ({i})')
Меня зовут:
Петя (0)
Петя (1)
Петя (2)
Петя (3)
Петя (4)
```

Функция *range()* может быть вызвана с двумя тремя аргументами. Первые два аргумента задают начальное и конечное значения, а третье - шаг изменения. Шаг - это велечина, которая прибавляется к переменной цикла на каждой итерации.

```
>>> for i in range(0, 10, 2):
>>>    print(i)
0
2
4
6
8
```

Шаг может быть отрицательным числом.

```
>>> for i in range(5, -1, -1):
>>>     print(i)
5
4
3
2
1
0
```

3 Строки

3.1 Умножение в применении к строкам

Умножив строку на число, можно повторить её нужно количество раз:

```
>>> "Привет" * 5
ПриветПриветПриветПриветПривет
```

3.2 Многострочные данные с тройными кавычками

```
>>> print('''Дорогая Алиса,
>>>
>>> Кошка Ева была арестована за похищение мяты, кражу со взломом
и вымогательство.
>>>
>>> Искренне,
>>> Боб''')
Дорогая Алиса,

Кошка Ева была арестована за похищение мяты, кражу со взломом
и вымогательство.

Искренне,
Боб
```

3.3 Получение символа. Срез

```
H   e   l   l   o       w   o   r   l   d    !
0   1   2   3   4   5   6   7   8   9   10   11
```

```
>>> spam = 'Hello world!'

>>> spam[0]
'H'
```

```
>>> spam[4]
'o'
```

```
>>> spam[-1]
'!'
```

Срез (Slicing):

```
>>> spam[0:5]
'Hello'
```

```
>>> spam[:5]
'Hello'
```

```
>>> spam[6:]
'world!'
```

```
>>> spam[6:-1]
'world'
```

```
>>> spam[:-1]
'Hello world'
```

```
>>> spam[::-1]
'!dlrow olleH'
```

```
>>> spam = 'Hello world!'
>>> fizz = spam[0:5]
>>> fizz
'Hello'
```

3.4 Операторы in и not in применительно к строкам

```
>>> 'Hello' in 'Hello World'
True
```

```
>>> 'Hello' in 'Hello'
True
```

```
>>> 'HELLO' in 'Hello World'
False
```

```
>>> '' in 'spam'
True
```

```
>>> 'cats' not in 'cats and dogs'
False
```

3.5 Строковый метод count()

Метод `count()` позволяет посчитать, сколько раз одна строка встречается в другой строке:

```
>>> message = "программа работает? да, программа заработала"
>>> c = message.count("программа")
>>> c
2
```

3.6 Строковый метод find()

Метод `find()` позволяет найти позицию, в которой подстрока появляется в строке:

```
>>> message = "Привет, как дела?"
>>> ind = message.find("как")
```

```
>>> ind
8
```

Если подстрока не найдена, метод `find()` вернёт –1:

```
>>> message = "Привет, как дела?"
>>> ind = message.find("нормально")
>>> ind
-1
```

3.7 Строковый метод replace()

Метод `replace()` позволяет заменить одну подстроку на другую:

```
message = "Привет, как дела? Привет, нормально"
>>> new_message = message.replace("Привет", "Здравствуй")
>>> new_message
Здравствуй, как дела? Здравствуй, нормально
```

3.8 Строковые методы upper(), lower(), isupper(), and islower()

Методы `upper()` и `lower()` преобразуют строку в верхний и нижний регистр соответственно:

```
>>> spam = 'Hello world!'
>>> spam = spam.upper()
>>> spam
'HELLO WORLD!'
```

```
>>> spam = spam.lower()
>>> spam
'hello world!'
```

Методы `isupper()` и `islower()` позволяют определить, являются ли символы в строке прописными или строчными

```
>>> spam = 'Hello world!'
>>> spam.islower()
False
```

```
>>> spam.isupper()
False
```

```
>>> 'H'.isupper()
True
```

```
>>> 'abc12345'.islower()
True
```

```
>>> '12345'.islower()
False
```

```
>>> '12345'.isupper()
False
```

4 Списки

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']

>>> spam
['cat', 'bat', 'rat', 'elephant']
```

4.1 Получение элемента списка

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
```

```
>>> spam[1]
'bat'
```

```
>>> spam[2]
'rat'
```

```
>>> spam[3]
'elephant'
```

4.2 Отрицательные индексы

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
```

```
>>> spam[-3]
'bat'
```

```
>>> 'The {} is afraid of the {}.'.format(spam[-1], spam[-3])
'The elephant is afraid of the bat.'
```

4.3 Получение среза списка

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam[1:3]
['bat', 'rat']
```

```
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[:2]
['cat', 'bat']
```

```
>>> spam[1:]
['bat', 'rat', 'elephant']
```

Срез всего списка позволяет получить копию списка:

```
>>> spam2 = spam[:]
['cat', 'bat', 'rat', 'elephant']
>>> spam.append('dog')
>>> spam
['cat', 'bat', 'rat', 'elephant', 'dog']
>>> spam2
['cat', 'bat', 'rat', 'elephant']
```

4.4 Получение количества элементов с помощью фукнции len()

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

4.5 Изменение значений в списке с помощью индексирования

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark'

>>> spam
['cat', 'aardvark', 'rat', 'elephant']

>>> spam[2] = spam[1]
```

```
>>> spam
['cat', 'aardvark', 'aardvark', 'elephant']

>>> spam[-1] = 12345

>>> spam
['cat', 'aardvark', 'aardvark', 12345]
```

4.6 Сложение (конкатенация) и умножение списков

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']

>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']

>>> spam = [1, 2, 3]

>>> spam = spam + ['A', 'B', 'C']

>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

4.7 Удаление элементов списка с помощью оператора del

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
```

```
>>> del spam[2]
>>> spam
['cat', 'bat']
```

4.8 Использование цикла for и списков

```
>>> supplies = ['pens', 'staplers', 'flame-throwers', 'binders']
>>> for i, supply in enumerate(supplies):
>>>       print('Index {} in supplies is: {}'.format(str(i),
supply))
Index 0 in supplies is: pens
Index 1 in supplies is: staplers
Index 2 in supplies is: flame-throwers
Index 3 in supplies is: binders
```

4.9 Операторы in и not in

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
```

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
```

```
>>> 'howdy' not in spam
False
```

```
>>> 'cat' not in spam
True
```

4.10 Поиск индекса элемента по его значению с помощью метода index()

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']

>>> spam.index('Pooka')
1
```

4.11 Добавление элементов с помощью методов append() и insert()

append():

```
>>> spam = ['cat', 'dog', 'bat']

>>> spam.append('moose')

>>> spam
['cat', 'dog', 'bat', 'moose']
```

insert():

```
>>> spam = ['cat', 'dog', 'bat']

>>> spam.insert(1, 'chicken')

>>> spam
['cat', 'chicken', 'dog', 'bat']
```

4.12 Удаление элемента из списка по его значение. Метод remove()

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']

>>> spam.remove('bat')

>>> spam
['cat', 'rat', 'elephant']
```

Если значение встречается несколько раз, будет удалено только первое встреченное значение.

4.13 Удаление значений из списка с помощью метода pop()

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']

>>> spam.pop()
'elephant'

>>> spam
['cat', 'bat', 'rat']

>>> spam.pop(0)
'cat'

>>> spam
['bat', 'rat']
```

4.14 Сортировка списка с помощью метода sort()

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
```

```
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

5 Словари

Пример словаря:

```
myCat = {'размер': 'толстый', 'цвет': 'серый', 'характер':
'громкий'}
```

5.1 Методы keys(), values(), и items()

values():

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
>>>     print(v)
red
42
```

keys():

```
>>> for k in spam.keys():
>>>     print(k)
color
age
```

items():

```
>>> for i in spam.items():
>>>     print(i)
('color', 'red')
('age', 42)
```

Используя методы keys(), values(), and items() и цикл for можно перебирать ключи, значения и пары ключ-значение соответственно.

```
>>> spam = {'color': 'red', 'age': 42}
>>>
>>> for k, v in spam.items():
>>>     print('Key: {} Value: {}'.format(k, str(v)))
Key: age Value: 42
Key: color Value: red
```

5.2 Проверяем наличие ключа или значения в словаре

```
>>> spam = {'name': 'Zophie', 'age': 7}
```

```
>>> 'name' in spam.keys()
True
```

```
>>> 'Zophie' in spam.values()
True
```

5.3 Метод get()

Метод get обладает двумя параметрами: ключ и значение по умолчанию, если ключа нет в словаре.

```
>>> picnic_items = {'яблоко': 5, 'кружка': 2}
>>> f'Я принесу {str(picnic_items.get("кружка", 0))} чашки.'
'Я принесу 2 чашки.'
```

```
>>> f'Я принесу {str(picnic_items.get("яйцо", 0))} яиц.'
'Я принесу 0 яиц.'
```

6 Функции

6.1 Описание функции

```
def happy():
    print("С днём рожденья тебя")

def square(x):
```

```
    y = x * x
    return y
```

6.2 Сопоставление параметров и аргументов

Сопоставление по позиции:

```
def formatDate(day, month, year):
    print(f"Сегодня {day}.{month} {year} года")

formatDate(22, 11, 2021)
# Сегодня 22.11 2021 года

formatDate(11, 2021, 22)
# Сегодня 11.2021 22 года
```

Сопоставление по именам:

```
def formatDate(day, month, year):
    print(f"Сегодня {day}.{month} {year} года")

formatDate(month=11, year=2021, day=22)
# Сегодня 22.11 2021 года
```

Использование значений по умолчанию:

```
def formatDate(day, month, year=2021):
    print(f"Сегодня {day}.{month} {year} года")

formatDate(1, 9)
# Сегодня 1.9 2021 года

formatDate(1, 9, 1991)
# Сегодня 1.9 1991 года
```

Переменное число аргументов:

```
def sum(*nums):
    s = 0
    for n in nums:
        s = s + n
    return s

sum(1, 2)
sum(5)
sum(5, 3, 6, 4, 3, 5, 1)
```

7 Создание классов. Модули

7.1 Описание класса

Для описания класса используется служебное слово `class`:

```
class User:
    def __init__(self, name):
        self.name = name
```

Данные, которые хранятся в объекте называют **полями**, а функции, которые обрабатывают эти данные - **методами**.

Метод `__init__` является **конструктором класса**. Каждый метод класса всегда должен содержать минимум один параметр, который принято называть `self`. Он всегда хранит ссылку на объект, для которого выполняется метод.

Класс может содержать ряд методов:

```
class Point:
    def __init__(self, initX, initY):
        self.x = initX
        self.y = initY

    def getX(self):
```

```
        return self.x

    def getY(self):
        return self.y

p = Point(7, 6)
print(p.getX())
print(p.getY())
```

7.2 Наследование

Один класс может наследовать поля и методы другого класса:

```
class Point:
    def __init__(self, initX, initY):
        """Конструктор"""
        self.x = initX
        self.y = initY

    def distanceFromOrigin(self):
        """Расстояние до начала координат"""
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5
    def __str__(self):
        """Преобразование в строку"""
        return "x=" + str(self.x) + ", y=" + str(self.y)


class LabeledPoint(Point):

    def __init__(self, initX, initY, label):
        super().__init__(initX, initY)
        self.label = label

    def __str__(self):
        return super().__str__() + f" ({self.label})"


labeledPt = LabeledPoint(7,6,"Точка")
print(labeledPt)
```

Пошаговое выполнение данного кода доступно по ссылке.

8 Модули

Каждый файл с расширением *.py* является **модулем**, который можно использовать самостоятельно, или **импортировать** в другой модуль (файл с кодом).

```
# файл calc.py

def sum(x,y):
    return x + y

def average(x, y):
    return (x + y)/2

def power(x, y):
    return x**y
```

Импортируем всё содержимое модуля *calc.py*:

```
# файл main.py
import calc

s = calc.sum(2, 2)
avg = calc.average(5, 10)

print(s)
print(avg)
```

Импортируем только необходимые функции из модуля *calc.py*:

```python
# файл main.py
from calc import sum, power

sqr = power(2, 4)
print(sqr)
```