**Python - шпаргалка**

**типы данных**

• int - целое

• float - с плавающей точкой

• bool - true\false

• string (str) - “текст”

# - комментарии

* **списки -** Списки можно конкатонировать и умножить как обычные переменные.

test = ["a", "b", "c", ["d", "e", "f"]]

test[0] = "v"

print (test [3][5])

>>> e

test2= test[начальное\_число\_списка:Последнее\_число\_списка:шаг(не обязательно)]

test2= test[начальное\_число\_списка:] - выводит от изначально-заданного числа списка до конца

test2= test[:Последнее\_число\_списка] - выводит от начала списка до заданого места

test2= test[:шаг] - выводит пункты списка согласно шагу

• **.append** - метод, который добавляет в список значение(в конец списка)

test.append("inform")

print (test)

>>> ['a', 'b', 'c', ['d', 'e', 'f'], 'information']

• **.remove** - удаляет элемент из списка

• **.insert** - добавляет элемент в список с указанием места, куда вставить

test.insert[в какое место, что]

>>> test.insert[1,"Ivanov"]

• **.count** -подсчитывает сколько заданных элементов в списке

test =[1,11,1,2,2,3,4,5,4,3]

print (test.count(1))

>> 2

• **.reverse** - переворачивает список. Не использовать с print

test = [1,2,3,4]

test.reverse()

print (test)

>>> 4,3,2,1

• **range** - выводит заданный диапазон цифр. Используется с list

number = list(range(от, до, шаг))

number = range(1, 101)[::2] - выводит только четные значение

• **The strip()** method removes any whitespace from the beginning or the end:

a = " Hello, World! "

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

• **in**

words = ["spam", "egg", "spam", "sausage"]

print("spam" in words)

print("egg" in words)

print("tomato" in words)

• **not in** - To check if an item is not in a list, you can use the not operator in one of the following ways:

nums = [1, 2, 3]

print(not 4 in nums)

print(4 not in nums)

print(not 3 in nums)

print(3 not in nums)

•**len** - показывает длину списка(к-во элементов)

nums = [1, 3, 5, 2, 4]

print(len(nums))

5

• **index** - Метод index находит первое вхождение элемента списка и возвращает его индекс.Если элемент отсутствует в списке, он вызывает ошибку ValueError.

letters = ['p', 'q', 'r', 's', 'p', 'u']

print(letters.index('r'))

print(letters.index('p'))

print(letters.index('z'))

result

>>>

2

0

ValueError: 'z' is not in list

>>>

• list.count(obj): Returns a count of how many times an item occurs in a list

• list.remove(obj): Removes an object from a list

• list.reverse(): Reverses objects in a list

**Модули**

import module

**math** - математические функции

#округление в меньшую сторону

print(math.float(a))

#округление в большую сторону

print(math.ceil(a))

#число Пи

print(math.pi)

**os** --- импортирует системные команды

* os.system('cls||clear') ---- очистка консоли

**random** Модуль random управляет генерацией случайных чисел. Его основные функции:

• **random()**: генерирует случайное число от 0.0 до 1.0

• **randint(min, max)**: возвращает случайное число из определенного диапазона

• **randrange()**: возвращает случайное число из определенного набора чисел

• **shuffle()**: перемешивает список

• **choice()**: возвращает случайный элемент списка

Функция random() возвращает случайное число с плавающей точкой в промежутке от 0.0 до 1.0. Если же нам необходимо число из большего диапазона, скажем от 0 до 100, то мы можем соответственно умножить результат функции random на 100.

import random

number = random.random() # значение от 0.0 до 1.0

print(number)

number = random.random() \* 100  # значение от 0.0 до 100.0

print(number)

import random

number = random.randint(20, 35)  # значение от 20 до 35

print(number)

* **randrange()** возвращает случайное целое число из определенного набора чисел.
* **randrange(stop)**: в качестве набора чисел, из которых происходит извлечение случайного значения, будет использоваться диапазон от 0 до числа stop
* **randrange(start, stop)**: набор чисел представляет диапазон от числа start до числа stop
* **randrange(start, stop, step)**: набор чисел представляет диапазон от числа start до числа stop, при этом каждое число в диапазоне отличается от предыдущего на шаг step

import random

number = random.randrange(10)  # значение от 0 до 10

print(number)

number = random.randrange(2, 10)  # значение в диапазоне 2, 3, 4, 5, 6, 7, 8, 9, 10

print(number)

number = random.randrange(2, 10, 2)  # значение в диапазоне 2, 4, 6, 8, 10

print(number)

* **Работа со списком** Для работы со списками в модуле random определены две функции: функция **shuffle()** перемешивает список случайным образом, а функция **choice()** возвращает один случайный элемент из списка:

numbers = [1, 2, 3, 4, 5, 6, 7, 8]

random.shuffle(numbers)

print(numbers) # 1

random\_number = random.choice(numbers)

print(random\_number)

**Циклы -** зацикливают выполнения. Выполняется пока test == True

**while:**

test = True

while test:

print ("test print")

* **break** - принудительно останавливает цикл

i = 1

while i:

  print (i)

  i = i++1

  if i == 100:

      break

* **continue** -  продолжает выполнение цикла, даже если условие не выполнилось

number =0

while number <= 100:

    number += 1

    if (number % 2) != 0:

        continue;

    print ("все четные числа:" + str(number))

**for** цикл используется для перебора последовательности

fruits = ["apple", "banana", "cherry"]

for x in fruits:

  print(x)

**if -** проверка на пустое

i = input()

if not i:

  do somth

**Операторы**

* and
* in
* not in
* or

test = ["damon", "alex", "john", "david"]

if "damon" in test:

  print("нашел им в списке")

Чтобы проверить, есть ли элемент в списке, можно использовать оператор in. Возвращает True, если элемент встречается в списке один или несколько раз, и False, если нет.

• **None** - возвращает ничего

**Dic - словарь**

test ={

  "ключ 1" : "значение1",

}

print(test["ключ 1"])

* **.get - возвращает елемент словаря**

car =        {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

print(car.get("model"))

* **Изменить элемент словаря**

car =        {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

car["year"] = 2018

* **Добавляет новый элемент словаря**

car =        {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

car["color"] = "red"

* **.pop - remove element from dic**

thisdict = {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

thisdict.pop("model")

print(thisdict)

* **.clear - очищает словарь**

car =        {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

car.clear()

**Функции**

Параметры **print**

• **end** - перевод строки. после каждой функции print по умолчанию осуществляется перевод строки. Для изменения того, что будет печататься после вывода всего, что есть в функции print можно использовать именованный параметр end.

print ('1', '2', '3', sep='+', end='')

print ('=', 1+2+3)

1+2+3 = 6

• **sep** - разделитель. Та строка, которая передается в качестве параметра sep будет подставляться вместо пробела в качестве разделителя

print ('1', '2', '3', sep '+')

1+2+3

**round()** - округление

**len** - показывает сколько элементов в списке

print (test + len(test)

**max и min** - выводит наибольший и наименьший элемент в списке

spisok = [1,2,3,99]

print (max(spisok))

>>>99

.**title** - Предложение с первой большой буквой

.**upper** - большие буквы

**def**- создание своей функции. После вызова своей функции начинает выполнятся весь блок кода, который записан под def. Функция должна быть в начале определена, а потом ее вызывать. Параметры внутри функции являються для нее локальным и не распространяются на остальной код.

  def my\_function():

    print...

    for...

    etc

my\_function()

• **return** - возвращает значение функции. Останавливает выполнения функции.

def max(x, y):

    if x > y:

       return x

    else:

       return y

x = int(input('Choose a number 1'))

y = int(input('Choose a number 2'))

print( max(x,y) )

**lambda** A lambda function is a small anonymous function.

x = lambda a : a + 10

print(x(5))

**Работа с файлами**

• **open** открытие файла

text = open('test.txt', '???')

#\* r - чтения файла

#\* w - перезапись

#\* а - добавление в файл

print(text.read())

text.write()

\*\*\*\*\*\*\*

text.close()

with open('text.txt', 'r') as f:

  print(f.read())

* **копирование файла в файл**

from shutil import copyfile

copyfile(src, dst)

**Действие со строками**

**.format - Форматирование строк**

'Hello, {}!'.format('Vasya')

**Синтаксис**

* поле замены ::= "{" [имя поля] ["!" преобразование] [":" спецификация] "}"
* имя поля ::= arg\_name ("." имя атрибута | "[" индекс "]")\*
* преобразование ::= "r" (внутреннее представление) | "s" (человеческое представление)
* спецификация::= см. ниже

**Теперь спецификация формата:**

* спецификация ::= [[fill]align][sign][#][0][width][,][.precision][type]
* заполнитель ::= символ кроме '{' или '}'
* выравнивание ::= "<" | ">" | "=" | "^"
* знак ::= "+" | "-" | " "
* ширина ::= integer
* точность ::= integer
* тип := "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"

**Выравнивание производится при помощи символа-заполнителя. Доступны следующие варианты выравнивания:**

Флаг - Значение

**'<'** Символы-заполнители будут справа (выравнивание объекта по левому краю) (по умолчанию).

**'>'** выравнивание объекта по правому краю.

**'='** Заполнитель будет после знака, но перед цифрами. Работает только с числовыми типами.

**'^'** Выравнивание по центру.

**Опция "знак" используется только для чисел и может принимать следующие значения:**

Флаг - Значение

**'+'** Знак должен быть использован для всех чисел.

**'-''-'** для отрицательных, ничего для положительных.

**'Пробел' '-'** для отрицательных, пробел для положительных.

name = "Damon"

print("Hollo{0:\*^13}").format( name )

\*\*\*\*\*\*\*Damon\*\*\*\*\*\*\*

• **Экранирование** - \”text\”

>>>print("пример \"экранирования\" в пайтоне")

пример "экранирования" в пайтоне

• **Перевод строки** \n

>>>print (“перевод\n строки”)

• **Многострочное форматирование** - форматирование текста так, как написано в редакторе

>>> print (''' text

here''')

text

here

• **Конкатенация** - склеивание переменной и функции. Только с одними типами данных

**Математические и логические действия**

**+**

**-**

**\*** - умножение

**\*\*** - возведение в степень

**/** - делить

**%** -остаток от деления

**унарный минус** - меняет знак числа

• **and**

if a == b and c == d:

• **or**

if a == b or c == d:

• **assert — предположение**

**• Условия - ложь(False)**

if not time =="12-30" …

• **Приоритетность:** https://prnt.sc/aq8vk5

a. Скобки - ()

b. возведение в степень - \*\*

c.   \* \

d. + -

**Кортеж** - не изменяемые значения

tuple = 1,2,3,4,5,6

print(tuple[0])

**SETS —** Множества

fruits = {"apple", "banana", "cherry"}

* **.add** - add element

fruits.add("orange")

* **.update** - add set1 in set2

fruits = {"apple", "banana", "cherry"}

more\_fruits = ["orange", "mango", "grapes"]

fruits.update(more\_fruits)

* **.remove** - remove element from set

fruits = {"apple", "banana", "cherry"}

fruits.remove("banana")

* **.discard** - отбрасывать element

fruits = {"apple", "banana", "cherry"}

fruits.discard("banana")

**ООП**

Создание объекта на основе класса называется **созданием экземпляра.**

**Метод \_\_init\_\_()**

Функция, являющаяся частью класса, называется **методом**. Все, что вы узнали

ранее о функциях, также относится и к методам; единственное практическое раз-

личие — способ вызова методов. Метод \_\_init\_\_() — специальный метод,

который автоматически выполняется при создании каждого нового экземпляра

на базе класса Dog . Имя метода начинается и заканчивается двумя символами

подчеркивания; эта схема предотвращает конфликты имен стандартных методов

Python и методов ваших классов.

class Dog():

"""Простая модель собаки."""

def \_\_init\_\_(self, name, age):

"""Инициализирует атрибуты name и age."""

self.name = name

self.age = age

def sit(self):

"""Собака садится по команде."""

print(self.name.title() + " is now sitting.")

def roll\_over(self):

"""Собака перекатывается по команде."""

print(self.name.title() + " rolled over!")

Метод \_\_init\_\_() определяется с тремя параметрами: self , name и age . Пара-

метр self обязателен в определении метода; он должен предшествовать всем

остальным параметрам. Он должен быть включен в определение для того, чтобы

при ­будущем вызове метода \_\_init\_\_() (для создания экземпляра Dog ) автомати-

чески передавался аргумент self . При каждом вызове метода, связанного с клас-

сом, автоматически передается self — ссылка на экземпляр; она предоставляет

конкретному экземпляру доступ к атрибутам и методам класса. Когда вы создаете

экземпляр Dog , Python вызывает метод \_\_init\_\_() из класса Dog .