# Математические и логические операции.

|  |  |
| --- | --- |
| **+** | сложение; возвращает тот же тип данных, который использовался. |
| **-** | вычитание; возвращает тот же тип данных, который использовался. |
| **/** | деление; возвращает тот же тип данных, который использовался. |
| **\*** | умножение; всегда возвращает тип данных float, даже при целых числах. |
| **\*\*** | возведение в степень print (2\*\*10) # 1024 |
| **//** | целая часть от деления print (50 // 30) # 1 |
| **%** | остаток от деления print (50 % 30) # 20 |
| **==** | равно; всегда возвращает тип данных bool. print (20 == 20) # True |
| **!=** | не равно; всегда возвращает тип данных bool. print (20 != 20) # False |
| **>** | больше; всегда возвращает тип данных bool. print (20 > 20) # False |
| **<** | меньше; всегда возвращает тип данных bool. print (20 < 20) # False |
| **>=** | > или =; всегда возвращает тип данных bool. print (20 >= 20) # True |
| **<=** | < или =; всегда возвращает тип данных bool. print (20 <= 20) # True |
| **+=** | a += 1 аналогично a = a+1 |
| **and** | и; всегда возвращает тип данных bool, проверяет до первого False  возвращает первый ложный элемент или последний истинный  print (20 == 20 and 20 == 30) # False  print (1 and 'a' and [23]) # [23]  import math  numbers = [1,2,3,4,5,0,-1,-2,-3]  result = []  for number in numbers:  if number > 0 and math.sqrt(number)<2:  result.append(number)  print(result) # [1, 2, 3]  import math  numbers = [1,2,3,4,5,0,-1,-2,-3]  result = [number for number in numbers if number > 0 and math.sqrt(number)<2]  print(result) # [1, 2, 3]  [x for x in range(1,11) if x % 2 == 0] |
| **or** | или; всегда возвращает тип данных bool, проверяет до первого True  возвращает последний ложный элемент или первый истинный  print (20 == 20 or 20 == 30) # True |
| **not** | не; всегда возвращает тип данных bool. print(not 20 == 20) # False |

# Типы данных.

|  |  |  |
| --- | --- | --- |
| **Целое число**  **неизмен**  **упоряд** | int  #int  Методы: | a = 1  // - целочисленное деление  % - остаток от деления  divmod(x, y) - пара (x // y, x % y)  \*\* - возведение в степень  abs - модуль числа  a = 5 // 2 # 2  a = 5 % 2 # 1  a = 6 \*\* 2 # 36  a = abs(b)  x, y = 5, 2  a, b = divmod(x, y)  print (a,b) # a = 2, b = 1  a += 1 # a = a + 1  num //= 10 # num = num // 10  num = 4567887  while num > 0:  print(num % 10)  num //= 10 # вывод цифр по 1 с конца  x | y побитовое или  x ^ y побитовое исключающее или  x & y побитовое и  x << n битовый сдвиг влево  x >> y битовый сдвиг вправо  ~x инверсия битов  x = 0b101 # x = 5 двоичная система  x = 0o11 # x = 9 восьмеричная система  y = 0x0a # y = 10 шестнадцатеричная система  print(ob101010) # 42  print (bin(19)) # 0b10011 - преобразование в 2-ую  print (oct(19)) # 0o23 - преобразование в 8-ую  print (hex(19)) # 0x13 - преобразование в 16-ую  print(int('10011', 2)) # 19 - преобразование в 10-ую  из 2-36-ричной |
|  | .bit\_length() | количество бит, необходимых для представления числа в двоичном виде, без учёта знака и лидирующих нулей.  print((-37).bit\_length()) # 6 или print(n.bit\_length()) |
| .to\_bytes | to\_bytes (length, byteorder, \*, signed=False) - возвращает строку 'байтов', представляющих это число.  (-1024).to\_bytes(10, byteorder='big', signed=True)  # b'\xff\xff\xff\xff\xff\xff\xff\xff\xfc\x00'  x.to\_bytes((x.bit\_length() // 8) + 1, byteorder='little')  # b'\xe8\x03' |
| .from\_bytes | int.from\_bytes(bytes, byteorder, \*, signed=False) - возвращает число из данной строки байтов.  int.from\_bytes(b'\xfc\x00', byteorder='big', signed=False)  # 64512  int.from\_bytes([255, 0, 0], byteorder='big')  # 16711680 |
|  |  |
|  |  |
| **Число с плавающей точкой**  **(веществен-ное число)**  **неизмен**  **упоряд** | float  #float  Методы: | a = 1.3456  операции с веществен числами в питоне неточные, и не поддерживают длинную арифметику - для таких целей используют объекты Decimal и Fraction.  print(f'{a:.2f}') # округление до 2 знаков  a = round(random.uniform(0, 50), 2) # окр до 2 знаков  min\_cost = float('inf') - бесконечность |
| .as\_integer  \_ratio() | float.as\_integer\_ratio() - пара целых чисел, чьё отношение равно этому числу.  print (12.5.as\_integer\_ratio()) # (25, 2) |
| .is\_integer() | float.is\_integer() - является ли значен целым числом.  print (12.5.is\_integer()) # False |
| .hex() | float.hex() - переводит float в hex (шестнадцатеричную систему счисления).  print ((1.5).hex()) # 0x1.8000000000000p+0 |
| float.fromhex | float.fromhex(s) - float из шестнадцатерич строки.  print(float.fromhex('0x1.8000000000000p+0')) # 1.5 |
|  |  |
|  |  |
| **Комплекс-ное число**  **неизмен**  **упоряд** | complex  #complex | x = complex(1, 2)  print(complex(1, 2)) # (1+2j)  print(x.conjugate()) # (1-2j) Сопряжённое число  print(x.imag) # 2.0 Мнимая часть  print(x.real) # 1.0 Действительная часть  ! print(x > y) # Комплексн числа нельзя сравнить  print(x == y) # False Проверка на равенство  abs(3 + 4j) # 5.0 Модуль комплексного числа  pow(3 + 4j, 2) # (-7+24j) Возведение в степень |
| **Логический тип**  **неизмен** | bool  #bool | a = True # True и False  True считается: не пустая строка, список и т.п., число не равное 0, число с плавающей точкой и т.п.  False считается: "", (), [], 0, None и т.п.  if len(text)>0: то же самое if text: |
| **Пустой тип** | None  NoneType |  |
| **Строка**  **неизмен**  **упоряд** | str  #str  Методы: | variable = 'привет'  можно использовать ', ", ''' кавычки  Срезы:  variable [1:3] – с 1 до 3 (не вкл) символ;  variable [:3] – с начала до 3 (не вкл);  variable [2:] – с 2 символа до конца.  variable [start:end:step] - с индекса start по индекс  end через шаг step  каждый символ имеет свой индекс, нумерация слева направо начинается с 0, а справа налево с (-1)  varieble = '''привет'''  print (varieble [1:3]) # ри  string[1] = "R" # ошибка - строка неизмен  Форматирование строк:   1. Конкатенация (сложение – не рекоменд)   name = 'Лео'  age = 30  print ('Привет ' + name + ', тебе', age, 'лет.')  Привет Лео, тебе 30 лет.   1. %   name = 'Лео'  age = 30  print ('Привет %s, тебе %d лет.' %(name, age))  Привет Лео, тебе 30 лет.   1. format   name = 'Лео'  age = 30  print ('Привет {}, тебе {} лет.'.format(name, age))  Привет Лео, тебе 30 лет.   1. f - строка   number = random.randint (1,100)  print (f'вы загадали {number}')  Кодировка (декодировка):  ascii американские символы  latin-1 европейские символы  utf-8 универсальная кодировка  Типы строк:  str – обычные строки  bytes – строки байт (возможно взятие индекса  sb[0] – это будет код, а не  символ, и среза sb[1:3])  bytearray – изменяемая строка байт  bs = b'Hello'  print(bs) # b'Hello'  print(bs[1]) # 101  print(bs [1:3]) # b'el'  print(type(bs)) # <class 'bytes'>  bs = b'Hel'  for item in bs:  print(item) # 72 101 108  s = 'Hello мир'  sb = s.encode('utf-8')  print(sb) # b'Hello \xd0\xbc\xd0\xb8\xd1\x80'  sb = sb.decode('utf-8')  print (sb) # Hello мир |
| .find() | поиск символа в строке, возвращает индекс найденного символа или первого символа из группы символов, если символа нет – возвр '-1'  find(str, start, end)  varieble = '''привет'''  print (varieble.find ('т')) # 5  # поиск с 10 по 15 индекс  index = welcome.find("wor",10,15)  print(index) # -1 |
| .replace() | заменяет в строке одну подстроку на другую,  параметр num указывает, сколько вхождений подстроки old надо заменить на new  replace(old, new, num)  # замена только первого дефиса  edited\_phone = phone.replace("-", "", 1) |
| .split() | разбивает строку на список подстрок в зависимости от разделителя (по умолч пробел):  split(): в качестве разделителя используется пробел  split(delimeter): в качестве разделителя delimeter  split(delimeter, num): параметр num указывает, сколько вхождений delimeter используется для разделения. Оставшаяся часть строки добавляется в список без разделения на подстроки  varieble = 'Привет, Вася'  print (varieble.split())  ['Привет,', 'Вася']  varieble = 'Привет, Вася'  print (varieble.split(','))  ['Привет', ' Вася']  names = '02.11.2002'  d, m, y = names.split('.')  print (d, m, y)  result = f '{d} {m} {y}'  print (result)02 11 2002  # разбиение по первым пяти пробелам  splitted\_text = text.split(" ", 5) |
| .join | сбор строки из списка с разделителем S  объединяет строки в одну строку, вставляя между ними определенный разделитель  A = ['red', 'green', 'blue'] print(' '.join(A)) # red green blue print(''.join(A)) # redgreenblue print('\*\*\*'.join(A)) # red\*\*\*green\*\*\*blue  word = "hello" # если на вход, строка а не список  joined\_word = "|".join(word)  print(joined\_word) # h|e|l|l|o |
| .upper | приведение строки к верхнему регистру, не изменяет саму переменную  varieble = 'Привет, Вася'  print (varieble.upper())  print (varieble)  ПРИВЕТ, ВАСЯ  Привет, Вася |
| .lower | приведение строки к нижнему регистру, не изменяет саму переменную  varieble = 'Привет, Вася'  print (varieble.lower()) # привет, вася  print (varieble) # Привет, Вася |
| . capitalize() | переводит в верхний регистр первую букву только самого первого слова строки |
| . title() | начальные символы всех слов в строке переводятся в верхний регистр |
| . lstrip() | удаляет начальные пробелы из строки |
| . rstrip() | удаляет конечные пробелы из строки |
| . strip() | удаляет начальные и конечные пробелы из строки  string = string.strip() |
| . ljust(width) | если длина строки меньше параметра width, то справа от строки добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по левому краю  print("iPhone 7:", "52000".ljust(10)) |
| .rjust(width) | если длина строки меньше параметра width, то слева от строки добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по правому краю |
| center(width) | если длина строки меньше параметра width, то слева и справа от строки равномерно добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по центру |
| .isdigit | возвращает True, если все символы строки - цифры  varieble = 'Привет, Вася'  print (varieble.isdigit()) # False |
| . isnumeric() | возвращает True, если строка - число |
| . isalpha() | возвращает True, если строка состоит только из алфавитных символов |
| . isupper() | возвращает True, если все символы строки в верхнем регистре |
| . islower() | возвращает True, если все символы строки - цифры |
| . startswith() | возвращает True, если строка начинается с подстроки str  starts\_with\_hello = file\_name.startswith("hello") #True |
| . endswith() | возвращает True, если строка заканчивается на подстроку str |
| ord | выводит код символа в кодировке Unicode  b = ord('z') |
| chr | выводит символ по коду  chr(122) |
| in | поиск подстроки в строке  exist = "sword" in string  print(exist) # False |
| **Список**  **измен**  **упоряд** | list  #list  Методы: | variable = ['hello', 282, True]  Можно использовать индексы и срезы как в 'строке', но срезы возвращают список.  variable = ['hello', 282, True]  print (variable [1]) # 282  print (variable [1:]) # [282, True]  numbers = [5] \* 6 # [5, 5, 5, 5, 5, 5]  numbers[0] = 125 # изм первый элемент списка  users[0][1] = 225 # обращ к элем вложен списка  variable = [] # пустой список  users3 = users1 + users2  a = input().split() # создание списка, по умолч  a = input().split('-') # введение данных через пробел  b=a[0:9:2] # c 0 до 9 не вкл с шагом 2  b = a[::-1] # список в обратном порядке  numbers = list(range(10)) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  numbers = list(range(2, 10)) # [2, 3, 4, 5, 6, 7, 8, 9]  numbers = list(range(10, 2, -2)) # [10, 8, 6, 4]  a = [10, 20, 30, 40]  a = [i + 5 for i in a]  print(a) # [15, 25, 35, 45]  Список нельзя просто присвоить другой переменной (будут как 1 переменная):  a = [1,2,3]  b=a  b[1]=200  print(a) # [1, 200, 3]  Копирование списков:   1. Создание среза от начала до конца (не работает при вложенных списках)   b=a[:]  b=a[0:9:2] # c 0 до 9 не вкл с шагом 2   1. Метод copy (не работает при вложен списках)   b=a.copy()   1. Модуль copy.copy() ( без вложенных списков)   b=copy.copy(a)   1. Модуль copy.deepcopy() (подходит для глубокого копиров даже вложенных списков)   b=copy.deepcopy(a)  import copy  a = [1,[5,6,7],3]  b=copy.deepcopy(a)  b[1][1]=200  print(a) # [1, [5, 6, 7], 3]  print(b) # [1, [5, 200, 7], 3]  a = [1,2,[1,2,3],4,5,6,7,8,9]  b=a[0:9:2] # c 0 до 9 не вкл с шагом 2  b[1][1]=200  b[0] = 99  print(a) # [1, 2, [1, 200, 3], 4, 5, 6, 7, 8, 9]  print(b) # [99, [1, 200, 3], 5, 7, 9]  Встроенные функции для работы со списками:  len(list): возвращает длину списка  sorted(list, [key]): возвращает отсортирован список  min(list): возвращает наименьший элемент списка  max(list): возвращает наибольший элемент списка |
| .append() | добавление нового элемента в конец списка  variable = ['Leo', 'Max', 'Kate']  variable.append ('Ron')  print (variable) # ['Leo', 'Max', 'Kate', 'Ron'] |
| .insert() | добавляет элемент item в список по индексу index  insert(index, item)  users.insert(1, "Bill") # ["Tom", "Bill", "Bob", "Alice"] |
| .remove() | удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError  variable = ['Leo', 'Max', 'Kate']  variable.remove('Max')  print (variable) # ['Leo', 'Kate']  если в цикле for вместо friends написать friends[:], то будет создана копия переменной, и изменения самой переменной в цикле не повлияют на цикл for, это особенно важно при использовании .remove |
| .pop() | удаляет и возвращает элемент по индексу, если индекс не передан, то удаляет последний элемент.  removed\_item = users.pop(1) # ["Bill", "Bob", "Alice"]  variable = ['Leo', 'Max', 'Kate']  print (variable.pop()) # Kate  print (variable) # ['Leo', 'Max'] |
| .clear() | удаление всех элементов из списка  variable = ['Leo', 'Max', 'Kate']  variable.clear()  print (variable) # [] |
| .index() | возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError  i = users.index("Tom") |
| .reverse() | расставляет все элементы в списке в обратном порядке, сразу изменяет переменную  variable = ['hello', 282, True]  variable.reverse()  print (variable) # [True, 282, 'hello']  или так  a = [10, 20, 30, 40]  a = a[::-1]  print(a) # [40, 30, 20, 10] |
| .count() | возвращает кол-во вхождений элемента в список  my\_list\_1 = [1, 3, 5, 7, 9, 11, 12, 3, 5, 9, 3]  n = my\_list\_1.count(3)  print (n) # 3 |
| .sort() | сортирует элементы и изменяет список. По умолч сортирует по возраст. Но с помощью параметра key мы можно передать функцию сортировки. Заглавные буквы имеют приоритет над строчными.  users.sort()  print(users) # ["Alice", "Bill", "Bob", "Sam", "Tom"]  users.sort()  users.reverse()  print(users) # ["Tom", "Sam", "Bob", "Bill", "Alice"]  users.sort(key=str.lower)  print(users) # ["alice", "Bill", "bob", "Sam", "Tom"]  функция sorted(list, key): сортирует список list, применяя к элементам функцию key. Функция не изменяет сортируемый список, а отсортированные элементы помещает в новый список, который возвращается в качестве результата.  sorted\_users = sorted(users, key=str.lower)  print(sorted\_users) # ["Bill", "bob","Sam","Tom"] |
| .join | сбор строки из списка с разделителем S  A = ['red', 'green', 'blue'] print(' '.join(A)) # red green blue print(''.join(A)) # redgreenblue print('\*\*\*'.join(A)) # red\*\*\*green\*\*\*blue |
| .copy | копирование списка  не работает при вложенных списках (для глубокого копирования нужен модуль copy, а не метод)  метод copy()  b=a.copy() # обычное копиров  или  модуль copy()  import copy  users2 = copy.copy(users1) # обычное копиров  users2 = copy.deepcopy(users1) # глубокое копиров |
|  |  |
|  |  |
| **Кортеж**  **неизмен**  **упоряд** | tuple  #tuple | variable = ('hello', 282, True)  variable = 'hello', 282, True  variable = ('hello',) # если 1 элемент необход запятая  список, который нельзя менять  users\_tuple = tuple(users\_list)  разложение кортежа на элементы:  user = ("Tom", 22, False)  name, age, isMarried = user  print(name) # Tom  print(age) # 22  print(isMarried) # False  Когда функция возвращает несколько значений, фактически она возвращает в кортеж:  def get\_user():  name = "Tom"  age = 22  is\_married = False  return name, age, is\_married  user = get\_user()  print(user) # ('Tom', 22, False) |
| **Словарь**  **измен**  **неупоряд** | dict  #dict  Методы: | friends = {'Max': 21, 'Leo': 22}  objects = {1: "Tom", "2": True, 3: 100.6}  неупорядоченные коллекции произвольных объектов с доступами по ключу, ключом могут быть только неизменяемые типы данных  friends = {'Max': 21, 'Leo': 22}  print (friends['Leo']) # 22  friends = {'Max': 21, 'Leo': 22}  friends ['Kate'] = 18  print (friends) # {'Max': 21, 'Leo': 22, 'Kate': 18}  friends = {'Max': 21, 'Leo': 22, 'Kate': 18}  del friends ['Leo']  print (friends) # {'Max': 21, 'Kate': 18}  users\_list = [  ["+111123455", "Tom"],  ["+384767557", "Bob"],  ["+958758767", "Alice"]]  users\_dict = dict(users\_list)  если запросить значение с ключом, которого нет в словаре, то Python сгенерирует ошибку KeyError:  user = users["+4444444"] # KeyError  old\_email = users["Tom"]["email"] # вложен словарь |
| .keys() | перебор словаря по ключам. По умолчанию словарь перебирается по ключам.  friends = {'Max': 21, 'Leo': 22, 'Kate': 18}  for key in friends.keys(): # key in friends  print (key)  Max  Leo  Kate  или  for key in friends: |
| . values() | перебор словаря по значениям  friends = {'Max': 21, 'Leo': 22, 'Kate': 18}  for val in friends.values():  print (val)  21  22  18 |
| .items() | перебор словаря по ключам и значениям, возвращает кортежи.  friends = {'Max': 21, 'Leo': 22, 'Kate': 18}  for item in friends.items():  print (item)  ('Max', 21)  ('Leo', 22)  ('Kate', 18)  friends = {'Max': 21, 'Leo': 22, 'Kate': 18}  for key, val in friends.items():  print (key)  print(val)  Max  21  Leo  22  Kate  18 |
| .get | возвращает из словаря элемент с ключом key  get(key, default)  users = {  "+11111111": "Tom",  "+33333333": "Bob",  "+55555555": "Alice"}  user\_1 = users.get("+55555555")  user\_2 = users.get("+66666666"'', "Unknown user")  print(user\_1) # Alice  print(user\_2) # Unknown user |
| .del | удаления элемента по ключу, если ключа нет в словаре, то будет исключение KeyError.  del users["+55555555"] |
| .pop | удаляет элемент по ключу key и возвр удаленный элемент. Если элемент с данным ключом отсутствует, то возвращается значение default  pop(key, default)  users.pop("+55555555") # ошибка если нет ключа  user = users.pop("+4444444", "Unknown user")  # нет ошибки, если нет ключа |
| .clear | удаление всех элементов из словаря  users.clear() |
| .copy | копирует содержимое словаря, возвращая новый словарь  users = {"+1111111": "Tom","+3333333": "Bob","+5555555": "Alice"}  users2 = users.copy() # users = {} |
| .update | объединяет два словаря  users.update(users2) # словарь users измениться  us3 = users.copy() # создастся новый словарь us3  us3.update(users2) |
| **Множество**  **измен**  **неупоряд** | set  #set  Методы: | cities = {'Las Vegas', 'Paris', 'Moscow', 'Paris'}  неупорядоченные коллекции, содержащие неповторяющиеся элементы  a = set() - создание пустого множества  cities = {'Las Vegas', 'Paris', 'Moscow', 'Paris', 'Moscow'}  print (cities) # {'Las Vegas', 'Moscow', 'Paris'}  Операции:   1. Объединение |   m\_things = {'Зонт', 'Майка', 'Шорты', 'Телефон'}  k\_things = {'Зонт', 'Блузка', 'Юбка', 'Телефон'}  print (max\_things | kate\_things)  # {'Майка', 'Шорты', 'Зонт', 'Юбка', 'Телефон', 'Блузка'}   1. Пересечение & (логическое умножение)   m\_things = {'Зонт', 'Майка', 'Шорты', 'Телефон'}  k\_things = {'Зонт', 'Блузка', 'Юбка', 'Телефон'}  print (max\_things & kate\_things) # {'Телефон', 'Зонт'}   1. Разность –   m\_things = {'Зонт', 'Майка', 'Шорты', 'Телефон'}  k\_things = {'Зонт', 'Блузка', 'Юбка', 'Телефон'}  print (max\_things - kate\_things) # {'Майка', 'Шорты'} |
| .add() | добавление элемента в множество  cities = {'Las Vegas', 'Paris', 'Moscow'}  cities.add ('Burma')  print (cities) # {'Moscow', 'Burma', 'Las Vegas', 'Paris'} |
| .remove() | удаление элемента из множества, генерирует исключения при отсутствии элемента  cities = {'Las Vegas', 'Paris', 'Moscow'}  cities.remove ('Moscow')  print (cities) # {'Paris', 'Las Vegas'} |
| .discard() | удаление элемента из множества, не генерирует исключения при отсутствии элемента  users.discard(user) |
| .clear() | удаление всех элементов  users.clear() |
| . copy() | копирование документа  users3 = users.copy() |
| .union()  | | объединяет 2 множества и возвр новое множество  users3 = users.union(users2) |
| .intersection  & | производит операцию пересечения множеств и возвращает новое множество  users3 = users.intersection(users2) |
| .difference  - | разность множеств возвращает элементы, которые есть в первом множестве, но отсутствуют во втором  users3 = users.difference(users2) |
| . issubset | возвращает True, если текущее множество является подмножеством (то есть частью) другого множества  print(users.issubset(superusers)) # True  print(superusers.issubset(users)) # False |
| . issuperset | возвращает True, если текущее множество является надмножеством (т. е. содержит) другое множество  print(users.issuperset(superusers)) # False  print(superusers.issuperset(users)) # True |
| **Неизменяем множество**  **неизмен**  **неупоряд** | frozenset | users = frozenset({"Tom", "Bob", "Alice"})  неизменяемое множество, в функцию frozenset передается набор элементов - список, кортеж, другое множество.  поддерживает ограниченный набор операций:  len(s): возвращает длину множества  x in s: возвращ True, если элемент x в множестве s  x not in s: возвращает True, если x отсутствует в s  s.issubset(t): возвращает True, если t содержит s  s.issuperset(t): возвр True, если t содержится в s  s.union(t): возвращает объединение множеств s и t  s.intersection(t): возвр пересечение множеств s и t  s.difference(t): возвращает разность множеств s и t  s.copy(): возвращает копию множества s |
|  |  |  |

# Встроенные функции и операторы.

Функция – это фрагмент программного кода ('подпрограмма'), к которому можно обратиться из другого места программы. Функции обычно имеют имя. В python можно создавать функции без имени.

Функция в python - объект, принимающий аргументы и возвращающий значение.

|  |  |
| --- | --- |
| **abs** | модуль числа  print (abs (-7)) # 7 |
| **max** | максимальное знач в последовательности или среди переменных,  можно сравнивать и строковые переменные  numbers = [1, 5, -9, 2, -1, 0]  print (max (numbers)) # 5  print(max(a,b,c)) # можно передавать разные последовательности  max(a,key=abs) # max по абсолютному значению  max(a, key=lambda x: int(x)) # в key можно записать функцию  numbers = [1, 5, -9, 2, -1, 0]  print (max ((numbers), key=lambda x: x\*\*2)) |
| **min** | минимальное знач в последовательности или среди переменных  numbers = [1, 5, -9, 2, -1, 0]  print (min (numbers)) # -9 |
| **sum** | сумма элементов последовательности  sum (numbers, <с какого числа начать суммирование>)  numbers = [1, 5, -9, 2, -1, 0]  print (sum (numbers)) # -2  print (sum ((numbers), 2)) # 0 т.е. сумму прибавлять к 2 |
| **round** | округление числа (банковское округление, если 5, то до ближайшего четного числа). Округление не всегода производится точно, т.к. числа переводятся в 2-ную систему и они м.б. чуть больше ожидаемого числа  round (<число>, <кол-во знаков после запятой>)  print (round (10.0398, 2)) # 10.04  print (round (6.5)) # 6  print (round (7.5)) # 8 |
| **divmod()** | Возвращает частное и остаток от деления a на b  a = 5  b = 2  print(divmod(a,b)) |
| **slice()** | slice([start=0], stop, [step=1]) - объект среза от start до stop с шагом step. |
| **enumerate**  #enumerate | генерирует кортежи, состоящие из двух элементов - индекса элемента и самого элемента.  enumerate (<имя последоват>, <с какого числа нумеровать>)  a = [2, 5, 3, 4, 1, 5]  b = list(enumerate(a)) # [(0, 2), (1, 5), (2, 3), (3, 4), (4, 1), (5, 5)]  b = dict(enumerate(a, 10)) # {10: 2, 11: 5, 12: 3, 13: 4, 14: 1, 15: 5}  champions = ['Ron', 'Leo', 'Jim']  for number, winner in enumerate (champions, 5):  print (number, winner) # 5 Ron 6 Leo 7 Jim  champions = ['Ron', 'Leo', 'Jim']  for winner in enumerate (champions, 5):  print (winner) # (5, 'Ron') (6, 'Leo') (7, 'Jim')  a = [10, 20, 30, 40] # короче a = [i + 5 for i in a]  for id, item in enumerate(a):  a[id] = item + 5 |
| **type** | возвращает тип переменной  type (name) |
| **all** | all(последовательность) - Возвращает True, если все элементы истинные (или, если последовательность пуста). |
| **any** | any(последовательность) - Возвращает True, если хотя бы один элемент - истина. Для пустой последовательности возвращает False. |
| **print** | выводит заданные объекты на стандартное устройство вывода *(экран)*или отправляет их текстовым потоком в файл.  по умолчанию разделяет данные пробелом  по умолчанию каждый print выводится с новой строки  print ('Добрый день!')  print (name\_1)  print (name\_1, '- ', age\_1, 'лет.')  number = random.randint (1,100)  print (f'вы загадали {number}')  sep= использование различных разделителей слов  разделитель '/n' – новая строка, ввод  print (name, age, sep= ' --- ')  Петя --- 20  end= использование различных разделителей строк  print (name\_1, end= ' --- ')  print (age\_1, end= ' ')  print ('лет')  Петя --- 20 лет  info = open('python.txt', 'w')  print("Круто же, правда?", file = info)  info.close()  печать столбиками, отступы  a = [1,2,3,4,55,66,77,88,100, 101]  for i in a:  print(f'{i:>4}', end='')  if i == 55:  print()  graf = [[j for j in range(4)] for i in range(4)]  print(\*graf, sep='\n')  [0, 1, 2, 3]  [0, 1, 2, 3]  [0, 1, 2, 3]  [0, 1, 2, 3] |
| **input** | ввод данных в программу, всегда создает строковую переменную.  result = input ()  name\_1 = input ('Введите Ваше имя: ')  age\_1 = int(input ('Сколько Вам лет: '))  a = input().split() # создание списка, по умолч через пробел  a = input().split('-')  a = sum(list(map(int, input().split()))) # сумма введенных данных |
| **is** | является  print(a is b) # False |
| **if** | если; условный оператор.  age = 10  if age < 18:  print ('Доступ запрещен!') |
| **if-else** | если, иначе; условные операторы.  age = 20  if age < 18:  print ('Доступ запрещен!')  else:  print ('Доступ разрешен!') |
| **if-elif-else** | если, иначе-если, иначе; условные операторы.  if age < 18:  print ('Доступ запрещен!')  if age % 5 == 0:  print ('Ой, да у Вас юбилей!')  elif age == 18:  print ('Что же с Вами делать?')  else:  print ('Доступ разрешен!') |
| **while** | цикл 'пока'.  используется, если цикл связан с условием, но не с последовательностью.  name = input ('Кто создатель Python? ')  while name != 'Гвидо':  print ('Неверно!')  name = input ('Кто создатель Python? ')  print ('Правильно!') |
| **while-else** | цикл 'пока'-'иначе'. 'else' выполняется только в том случае, если цикл выполнен до конца, если цикл прервался по 'break', по 'else' не выполняется.  number = 0  n = int (input ('Введите n: '))  while number <= 10:  print (number)  if number > n:  break  number += 1  else:  print ('end')  print ('finish') |
| **for**  **for in**  #for | цикл, позволяющий перебирать последовательности по очереди без указания индекса.  используется для перебора последовательностей, если не нужен индекс.  friends = ['Leo', 'Max', 'Kate']  for friend in friends:  print (friend)  Leo  Max  Kate  если вместо friends написать friends[:], то будет создана копия переменной, и изменения самой переменной в цикле не повлияют на цикл for, это особенно важно при использовании .remove  enumerate - for n, m in enumerate(list, 0) # индекс (c 0) и значение |
| **range**  **for i in range** | создает последовательность целых чисел. Возвращает тип range.  range (начало, конец, шаг)  используется для перебора последовательностей, если нужен индекс или необходимо пропустить некоторые элементы или необходимо идти с конца в начало.  print (list(range(5, 10, 2)))  [5, 7, 9]  number = range (10)  print (number) # range(0, 10)  print (type(number)) # <class 'range'>  print (list(number)) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  friends = ['Leo', 'Max', 'Kate']  for i in range(len(friends)):  print(i+1, friends[i])  1 Leo  2 Max  3 Kate  friends = ['Leo', 'Max', 'Kate']  for i in range(1, len(friends) + 1):  print(i, friends[i-1])  1 Leo  2 Max  3 Kate  # перебор элементов в обратном порядке  friends = ['a','b','c','d','i','f']  for i in range(len (friends), 3, -1):  print(friends[i-1]) # f i d |
| **break** | оператор - досрочное прерывание цикла.  name = None  while True:  name = input ('Кто создатель Python? ')  if name == 'Гвидо':  break  print ('Неверно!')  print ('Правильно!') |
| **continue** | оператор – переход на следующую итерацию цикла, минуя оставшееся тело цикла.  number = 0  n = int (input ('Введите n:'))  while number <= n:  if number % 2 == 0:  number += 1  continue  print (number)  number += 1 |
| **pass** | оператор-заглушка для недописанного кода  создание пустого блока |
| **len** | подсчет количества элементов  name = 'Вася'  print (len (name))  4 |
| **del** | удаление  variable = ['Leo', 'Max', 'Kate']  del variable [1]  print (variable)  ['Leo', 'Kate'] |
| **in** | оператор – в  возвращает bool  variable = ['Leo', 'Max', 'Kate']  print ('Leo' in variable)  True |
| **global** | позволяет изменять глобальную переменную внутри функции, но лучше этого не делать  global\_a = 1 #изменение глобальной переменной в функции  def variable ():  global global\_a  global\_a = 5  print (global\_a)  variable ()  print (global\_a)  1  5 |
| **sorted()** | возвращает сортированный список, не изменяет переменную  variable = ['Leo', 'Max', 'Kate']  print (sorted (variable)) # ['Kate', 'Leo', 'Max']  print (variable) # ['Leo', 'Max', 'Kate']  sorted (iterable, \*,key = None, reverse = False)  аргументы: последовательность, ключ для сортировки, порядок  numbers = [1, 9, 3, 23, 0, 6, 3, 8, 19]  cities = [('Москва', 1000), ('Лас-Вегас', 500), ('Антверпен', 2000)]  print (sorted (numbers)) # [0, 1, 3, 3, 6, 8, 9, 19, 23]  print (sorted (numbers, reverse = True)) # [23, 19, 9, 8, 6, 3, 3, 1, 0]  print (sorted (cities))  # [('Антверпен', 2000), ('Лас-Вегас', 500), ('Москва', 1000)]  def by\_count (city): #параметр функции 'city'  return city [1] #возвр 2 элемент кортежа  print (sorted (cities, key = by\_count))  # [('Лас-Вегас', 500), ('Москва', 1000), ('Антверпен', 2000)]  lambda # упрощение  cities = [('Москва', 1000), ('Лас-Вегас', 500), ('Антверпен', 2000)]  print (sorted (cities, key = lambda city: city [1])) |
| **filter** | фильтрация последовательности  Должна принимать элемент фильтруемого объекта. Если функция вернёт False, данный элемент не попадёт в результат.  filter (function, iterable)  аргументы: функция фильтрации, последовательность  numbers = (1, 2, 3, 4, 5, 6, 7, 8)  def is\_even (number):  return number % 2 == 0  result = filter (is\_even, numbers)  print (result) # возвращает адрес фильтр-объекта  # <filter object at 0x000001D69A154390>  print (list(result)) # [2, 4, 6, 8]  lambda <входные параметры>:<результат>  def is\_even(number):  return number %2 == 0  names = ['Max', 'Leo', 'Kate']  print (list (filter (lambda x: len(x) > 3, names))) # ['Kate'] |
| **map**  #map | применение функции к каждому элементу последовательности  map (function, iterable …)  аргументы: функция к-рую надо применить, последовательность  a = [1,2,3]  b = map(str, a)  c = list(map(str, a))  print(b) # <map object at 0x000001A3711A6908>  print(c) # ['1', '2', '3']  numbers = [5, 3, 4, 7, 8]  print (list (map (lambda x: x\*\*2, numbers))) # [25, 9, 16, 49, 64]  print (list (map (lambda x: str(x), numbers))) # ['5', '3', '4', '7', '8']  a = sum(list(map(int, input().split()))) |
| **zip**  #zip | проходится одновременно по нескольким итерируемым объектам.  Функция zip возвращает итератор, который останавливается, когда исчерпывается самая короткая последовательность. Если требуется учесть все значения из самой длинной, то следует использовать функцию zip\_longest из модуля itertools (itertools.zip\_longest(a,b,c)) - недостающие данные замен None.  a = [10, 20, 30, 40]  b = ['a', 'b', 'c', 'd']  c = list(zip(a, b)) # [(10, 'a'), (20, 'b'), (30, 'c'), (40, 'd')]  c = dict(zip(a, b)) # {10: 'a', 20: 'b', 30: 'c', 40: 'd'}  a = []  b = []  c = []  for i, j in zip(range(10,20), range(1,10)):  c.append(i+j)  a.append(i)  b.append(j) |
| **open** | открывает файл в указанном режиме  open(<имя файла>, <режим открытия>, <кодировка>)  Режимы открытия mode:  r - чтение  w - запись, если файла нет – создается новый  x - запись, если файла нет – ошибка  a - дозапись  b -двоичный режим  + - открытие на чтение и запись  open ('first.txt', 'w')  f = open ('first.txt', 'w')  Запись текста в файл:  write – записать строку в файл  writelines – записать список строк в файл  \n – символ конца строки  f=open('second.txt','w')  f.write('Hello, ')  f.write('Leo!\nHi')  f.close()  Hello, Leo!  Hi  f=open('second.txt','w')  f.writelines(['Hello, ', 'Python!\n', 'Hi'])  f.close()  Hello, Python!  Hi  Чтение из файла:  read – чтение всего файла  for line in f – чтение файла построчно  readlines – чтение файла в список строк  f = open('second.txt', 'r')  print(f.read())  f = open('second.txt', 'r')  for line in f:  print(line) или print(line.replace('\n',''))  если надо убрать перенос строки, чтобы не печат через строчку.  f = open('second.txt', 'r')  a = f.readlines() или print(f.readlines())  print(a)  Закрытие файла:  close() – если до закрытия файла произойдет исключительная  ситуация, файл не будет закрыт  with – файл закроется автоматически после окончания действия  try:  somefile = open("hello.txt", "w")  try:  somefile.write("hello world")  except Exception as e:  print(e)  finally:  somefile.close()  except Exception as ex:  print(ex)  with open("hello.txt", "w") as somefile:  somefile.write("hello world")  with open('second.txt', 'r') as f:  for line in f:  print(line.replace('\n',''))  Работа с файлами в режиме байтов:  open(<имя файла>, <режим открытия>, encoding = 'utf-8')  'wb' – запись в режиме байтов, 'rb' – чтение в режиме байтов и т.д.  with open ('str.txt', 'w') as s:  text\_1 = 'Привет мир!'  s.write (text\_1)  with open ('bytes.txt', 'wb') as b:  b.write('Привет заврики!'.encode('utf-8'))  with open ('str.txt', 'r') as s:  print(s.read())  with open ('bytes.txt', 'rb') as b:  res = b.read()  result = res.decode('utf-8')  print (res)  print(result)  Привет мир!  b'\xd0\x9f\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82 \xd0\xb7\xd0\xb0\xd0\xb2\xd1\x80\xd0\xb8\xd0\xba\xd0\xb8!'  Привет заврики! |
| **try:**  **except:**  **as e**  **try:**  **except:**  **else**  **finally** | обработка исключений.  try:  блок с возможной исключительной ситуацией  except <тип исключения>:  код, выполняемый при исключительной ситуации  else:  код, выполняемый, если исключения не произошло  finally:  код, выполняемый всегда  number = int(input('Введите число: '))  try:  result = 100 / number  print(result)  except:  print('Что-то пошло не так!')  print('Конец')  Что-то пошло не так!  Конец  number = int(input('Введите число: '))  try:  result = 100 / number  print(result)  except ZeroDivisionError:  print('Деление на 0 невозможно')  except Exception: # любое исключение  print('Неизвестная ошибка')  print('Конец')  number = int(input('Введите число: '))  try:  result = 100 / number  except ZeroDivisionError as e:  print('Деление на 0 невозможно', e)  except Exception as e: # в e записать сведения об искл  print('Неизвестная ошибка', e)  else:  print(result)  finally: # часто используют для закрытия файла  print('Конец')  Деление на 0 невозможно division by zero  Конец |
| **raise** | Создание исключительной ситуации.  try:  number1 = int(input("Введите первое число: "))  number2 = int(input("Введите второе число: "))  if number2 == 0:  raise Exception("Второе число не должно быть равно 0")  print("Результат деления двух чисел:", number1/number2)  except ValueError:  print("Введены некорректные данные")  except Exception as e:  print(e)  print("Завершение программы") |
| **bin()** | перевод в двоичную систему  print(bin(42)) # ob101010  print(ob101010) # 42 |
| **oct()** | перевод в восьмеричную систему  print(oct(42)) # 0o52 |
| **hex()** | перевод в шестнадцатеричную систему  print(hex(42)) # 0x2a |
| **assert** | проверка истинности утверждения  assert item == func(a)  используется в отладочных целях. Если проверка не прошла, возбуждается исключение AssertionError.  assert 1 < 0, 'ошибка такая-то' |
| **iter**  **next**  **генератор выражений**  **yield**  **генератор функций** | создает итератор  s = [1, 2, 3] # список, итерируемый объект  d = iter(s) # итератор, поддерживает ф-цию next, помнит какой  объект будет следующим, обходит список только 1 раз  print(next(d)) # 1  print(next(d)) # 2  print(next(d)) # 3  a = [i\*\*2 for i in range(1,11)] # генератор списка  a = (i\*\*2 for i in range(1,11)) # итератор-генератор списка  print(next(a)) # 1  print(next(a)) # 4  print(next(a)) # 9  # список не хранит в памяти, каждое след число считается на лету.  for i in a: # такой список можно обойти только 1 раз  print(i)  # 1, 4, 9, 16 …  lst = list(a)  # [1, 4, 9, 16 …]  останавливает выполнение функции на yield при повторном запуске продолжает с yield  def fact\_gen(n):  pr = 1  for I in range(1, n + 1)  pr = pr \* i  yield pr  s = fact\_gen(10)  print(next(s)) # 1  print(next(s)) #2  print(next(s)) #6 |
| **time** | time.sleep(5) |
| **datetime** | from datetime import datetime, date, time  print( date.today()) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Встроенные модули.

Модуль (библиотека) – это файл с программой.

Варианты подключения модуля:

1. import math - подключение модуля целиком [print (math.sin(38))].
2. import math as mt - подключение модуля с созданием псевдонима.
3. from math import\* - импорт всего содержимого, не импортируя весь модуль (не рекомендуется) [print (sin(38))].
4. from math import sin, cos - импорт конкретных классов, функций и т.д. из модуля, не импортируя весь модуль [print (sin(38))].

|  |  |  |
| --- | --- | --- |
| **random** | Функции: | генератор случайных чисел, букв, последовательностей |
| .randint () | целое случайное число (<от>,<до вкл>)  number = random.randint (1,100)  print (number)  48 |
| .uniform() | не целое случайное число (<от>,<до вкл>)  a = round(random.uniform(0, 50), 2) - 2 знака после |
| .choice | случайный элемент последовательности  playes = ['Leo', 'Max', 'Kate']  print (random.choice (playes)) |
| .sampler | список длиной n из последовательности  .sampler (<переменная>,<кол-во элементов>)  playes = ['Leo', 'Max', 'Kate', 'Ron']  print (random.sample (playes, 2)) |
| .shuffle | перемешивает последовательность  cards = ['6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']  random.shuffle (cards)  print (cards) |  |
|  |  |  |
|  |  |  |
| **math** | Функции | математический модуль  расстояние между 2-ух точек с координатами  print (math.sqrt((x1-x2)\*\*2+(y1-y2)\*\*2)) |
| .pi | число пи  print (math.pi) |
| .sin | синус числа, угол в радианах  print (math.sin(38)) |
| .cos |  |
| .asin |  |
| .acos |  |
| .factorial | факториал  print (math.factorial(5)) |
| .exp | экспонента |
| .sqrt | квадратный корень  math.sqrt(number) |
| .pow (,) | возведение в степень, pow (<число>,<степень>)  print (math.pow (2,3)) |
| .log10 | логарифмы |
|  |  |
|  |  |
| **os** | Функции | Содержит функции для работы с операционной системой  Не зависит от конкретной операционной системы |
| .name | имя операционной системы  print (os.name)  nt |
| .chdir | смена текущей директории |
| .getswd() | текущая рабочая директория  print (os.getcwd())  D:\Tefi\Python |
| .mkdir() | создание директории (папки)  new\_path = os.path.join(os.getcwd(), 'new\_folder')  os.mkdir (new\_path)  new\_path = os.path.join(r'D:\Tefi\Конспекты', 'New\_fold')  os.mkdir (new\_path) |
| .rmdir | удаление директории (папки) |
| .remove | удаление файла |
| .path | модуль для работы с путями |
| .path.join | соединяет пути с учётом особенностей операционной системы  new\_path = os.path.join(os.getcwd(), 'new\_folder')  new\_path = os.path.join(r'D:\Tefi\Конспекты', 'New\_fold')  os.mkdir (new\_path) |
| .listdir | список файлов и директорий в папке.  по умолчанию в папке программы  print (os.listdir())  ['Kotya.py', 'Lesson', 'Trening.py'] |
|  |  |
| **sys** |  | обеспечивает взаимодействие с интерпретатором python |
| .executable | путь к интерпретатору python  print (sys.executable) |
| .exit() | выход из python |
| .platform | информация об операционной системе  print (sys.platform)  win32 |
| .path | список путей поиска подключаемых модулей  имеет тип данных list  добавить путь к своему модулю  sys.path.append('D:') |
| .argv | список аргументов командной строки при запуске скрипта python  путь до нашего скрипта (нулевой элемент)  print (sys.argv[0])  D:\Tefi\Python\Trening.py  остальные параметры передаются при вызове скрипта через пробел  python my\_script.py par1 par2 par3  import os, sys #запуск только через командную строку  def ping():  print('pong')  def get\_info():  print(os.listdir())  def hello (name):  print ('Hello', name)  command = sys.argv[1]  if command == 'ping':  ping()  elif command == 'list':  get\_info()  elif command == 'name':  name = sys.argv [2]  hello(name) |
| **.setrecursionlimit** | увеличение глубины рекурсии (повторов)  sys.setrecursionlimit(3000) |
| .getsizeof | размер памяти объекта в байтах  print(sys.getsizeof('Hello world!'))  61 |
|  |  |
|  |  |
| **pickle** | Функции: | Сереализация – это процесс преобразования объекта в поток байтов для сохранения или передачи в память, базу данных или файл.  Десереализация – обратный процесс. |
| dump | сохранение объекта в файл  import pickle  person = {'name':'Leo', 'phone': [123, 456]}  with open('person.dat', 'wb') as f:  pickle.dump(person,f) |
| dumps | преобразование объекта в байты |
| load | загрузка объекта из файла  import pickle  with open('person.dat', 'rb') as f:  person\_new = pickle.load(f)  print (person\_new) |
| loads | загрузка объекта из набора байт |
| **json** | Функции: | JavaScript Object Notation  текстовый формат обмена данными (строка), основанный на JavaScript (чаще всего используется в web разработке для передачи данных про протоколу http) |
| dump | сохранение объекта в формате json файл  import json  person = [{'name':'Leo', 'age':23,'phones':[123, 456]}, {'name':'Ron', 'age':19,'phones':[789, 543]}]  with open('person.json', 'w', encoding='utf-8') as f:  person\_json = json.dump(person,f)  with open('person.jon', 'r', encoding='utf-8') as f:  person\_new = json.load(f)  print(person\_new)  print(type(person\_new))  [{'name': 'Leo', 'age': 23, 'phones': [123, 456]}, {'name': 'Ron', 'age': 19, 'phones': [789, 543]}]  <class 'list'> |
| dumps | преобразование объекта в json  import json  person = [{'name':'Leo', 'age':23,'phones':[123, 456]}, {'name':'Ron', 'age':19,'phones':[789, 543]}]  person\_json = json.dumps(person)  print (person\_json)  print(type(person\_json))  [{"name": "Leo", "age": 23, "phones": [123, 456]}, {"name": "Ron", "age": 19, "phones": [789, 543]}]  <class 'str'> |
| load | загрузка объекта из файла  import json  person = [{'name':'Leo', 'age':23,'phones':[123, 456]}, {'name':'Ron', 'age':19,'phones':[789, 543]}]  with open('person.jon', 'w', encoding='utf-8') as f:  person\_json = json.dump(person,f)  with open('person.jon', 'r', encoding='utf-8') as f:  person\_new = json.load(f)  print(person\_new)  print(type(person\_new)))  [{'name': 'Leo', 'age': 23, 'phones': [123, 456]}, {'name': 'Ron', 'age': 19, 'phones': [789, 543]}]  <class 'list'> |
| loads | загрузка объекта из формата json  import json  person = [{'name':'Leo', 'age':23,'phones':[123, 456]}, {'name':'Ron', 'age':19,'phones':[789, 543]}]  person\_json = json.dumps(person)  person\_new = json.loads(person\_json)  print(person\_new)  print(type(person\_new))  [{'name': 'Leo', 'age': 23, 'phones': [123, 456]}, {'name': 'Ron', 'age': 19, 'phones': [789, 543]}]  <class 'list'> |
| **copy** |  | подходит для глубокого копирования при вложенных списках и т.п.  import copy  a = [1,2,[1,2,3]]  b=copy.deepcopy(a)  b[2][1]=200  print(a,b)  [1, 2, [1, 2, 3]] [1, 2, [1, 200, 3]] |
| **re** |  | модуль регулярные выражения |
| .findall() | ищет слова в тексте string подходящие под шаблон  re.findall(<'pattern(шаблон)'>,<string(текст)>,<flags=0>)  mail = re.findall('\w+@\w+\.\w+', text)  print(mail)  Шаблоны:  . – заменяет любой символ кроме переноса строки  ('Ст.л' – Стол, Стул)  ('<!--.\*,-->', re.DOTALL) все знаки, в т.ч. перенос  строки  \ - экранирование спецсимволов  \. – если подразумевается именно '.' ('mail\.ru')  \d – любая цифра  \D – что угодно кроме цифры  \s – любой пробельный символ (пробел, табуляция)  \S – что угодно кроме пробела  \w – любой символ а-я,a-z,0-9,\_  \W - что угодно кроме а-я,a-z,0-9,\_  + - квантификатор, повтор 1 и > раз, склеивает  однородные символы, идущие подряд  ('2008', а не '2' '0' '0' '8')  ? – квантификатор, символ может быть, а может и не  быть ('\d+,?\d+%') - ',' м.б. или не быть.  \* - квантификатор, повтор 0, 1 или > раз, т.е. может  быть, а может и не быть. ('\d+,?\d\*%')  \*? – квантификатор, сколько угодно раз, но  остановиться как только встретишь то, что после  ('<!--.\*,-->')  {} – квантификатор, точное кол-во повторений ('%{3}')  ('%{2,3}') ('%{3}[^\.,]')-что угодно кроме четвертой  точки и запятой {n,m} – от n до m {n,} - >=n  [] – любой символ, который указан в [].  ([0123456789,]+%) или ([0-9,]+%)  [^] – любой символ, кроме указанного в [] - [^1].  (['^a-z']) – не является буквой  \w+ - любой символ повторенный >= 1 раз  '\w+@\w+\.\w+' – электронная почта  () – вывести только ту часть шаблона, что в скобках  ('(\d+,?\d\*)%'), т.е. без знака %  | - создание двух взаимозаменяемых подшаблонов  подшаблоны внутри () - ('(\d+,?\d\*|\d+/.?\d\*)%')  ([^/]+) |
| .sub | изменяет текст  re.sub(<'шаблон, к-рый надо изменить'>, <'на что надо изменить'>,<текст>)  text = re.sub('[Фф]ранцузы', 'Россияне', text) |
| .compile | создание объекта 'pattern' и использование к нему методов .findall(<текст>) или re.sub(<pattern>, <на что заменить>, <текст>)  pattern = re.compile ('\d{1,2}\-[йя]')  pattern.findall (text\_1)  print (re.sub(pattern, 'n', text\_3)) |
| .search | ищет первое вхождение объекта подходящего под шаблон и возвращает объект хранящий инфо о месте нахождения объекта (индекс первой и последней буквы) и инфо о самом объекте нахождения. Или не возвращает None.  re.search(pattern\_1, text\_1) |
| .match | то же самое, что search, но ищет только в начале строки, а не по всему документу. |
| .split | разделение строки на список по разделителю 'шаблон'  re.split(pattern\_1, text\_1)  len(re.split(pattern\_1, text\_1)) – длина списка  re.split(pattern\_1, text\_1)[1] – 2-й элемент списка |
|  |  |
|  |  |
| **collections** |  | специализированный тип данных на основе словарей, кортежей, множеств, списков. |
| .Counter  elements  (ключи)  most\_common  (наибольшие значения)  substract  (вычитание)  clear  (очистка)  +  -  (выводит только не отриц значения)  & (лог и)  значение есть и там и там  | (лог или)  значение есть или там или там  -z  +z  (унарное сложение выводит только полож знач, унарное вычитание выводит только отриц знач меняя у них знак) | вид словаря, неупорядоченная коллекция пар "ключ"-"значение", где значение частота вхождения ключа (позволяет считать кол-во неизменяемых объектов)   from collection import Counter  a = collections.Counter('abracadabra')  b = collections.Counter(cat = 4, dog = 5)  print(a, b, sep='\n')  print(b['z'])  Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1})  Counter({'dog': 5, 'cat': 4})  0  print(set(a))  {'b', 'd', 'c', 'a', 'z', 'r'}  a['z'] = 0  print(a)  print(list(a.elements()))  Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1, 'z': 0})  ['a', 'a', 'a', 'a', 'a', 'b', 'b', 'r', 'r', 'c', 'd']  print(list(a.most\_common(2)))  [('a', 5), ('b', 2)]  c = collections.Counter(a=5, b=0)  d = collections.Counter(a=3, b=-2)  c.subtract(d)  print(c)  Counter({'a': 2, 'b': 2})  a.clear()  print(a)  Counter()  c = collections.Counter(a=5, b=0)  d = collections.Counter(a=3, b=-2)  print(c + d)  Counter({'a': 8})  c = collections.Counter(a=4, b=0, c=1)  d = collections.Counter(a=3, b=-2, c=1)  print(c & d)  Counter({'a': 3, 'c': 1})  c = collections.Counter(a=4, b=0, c=1)  d = collections.Counter(a=3, b=-2, c=1)  print(c | d)  Counter({'a': 4, 'c': 1})  d = collections.Counter(a=3, b=-2, c=1)  print(+d)  print(-d)  Counter({'a': 3, 'c': 1})  Counter({'b': 2}) |
| Deque  deque(d, 2) - из f 2 последних элемента  maxlen() - длина очереди  clear() - очистка очереди  append - добавить 1 элемент в конец очереди  appendleft - добавить 1 элемент в начало очереди  extend - добавить несколько элементов в конец очереди  extendleft - добавить несколько элементов в начало очереди  pop() - забрать 1 элемент из конца очереди  popleft() - забрать 1 элемент из начала очереди  count() - подсчитать сколько раз встречается элемент  index() - выводит индекс элемента  insert(index, element) - вставить элемент в середину очереди  reverse() - развернет очередь  rotate(n) - перемещает n элементов из конца очереди в начало  rotate(-n) - перемещает n элементов из начала очереди в конец | очередь  a = collections.deque('abrakadabra')  b = collections.deque([1, 2, 3, 4, 5])  print(a, b, sep='\n')  deque(['a', 'b', 'r', 'a', 'k', 'a', 'd', 'a', 'b', 'r', 'a'])  deque([1, 2, 3, 4, 5])  d = collections.deque([i for i in range(0, 5)])  print(d)  f = collections.deque(d, 2)  print(f)  deque([0, 1, 2, 3, 4])  deque([3, 4], maxlen=2)  a = collections.deque('kadabra', maxlen=4)  b = collections.deque([1, 2, 3, 4, 5], maxlen=3)  print(a, b, sep='\n')  deque(['a', 'b', 'r', 'a'], maxlen=4)  deque([3, 4, 5], maxlen=3)  a.clear()  deque([])  d = collections.deque([i for i in range(0, 5)])  print(d)  d.append(5)  d.appendleft(6)  print(d)  deque([0, 1, 2, 3, 4])  deque([6, 0, 1, 2, 3, 4, 5])  d = collections.deque([i for i in range(0, 5)])  print(d)  d.extend([5, 6])  d.extendleft([7, 8])  print(d)  deque([0, 1, 2, 3, 4])  deque([8, 7, 0, 1, 2, 3, 4, 5, 6])  d = collections.deque([i for i in range(0, 5)])  print(d)  f\_1 = d.pop()  f\_2 = d.popleft()  print(f\_1, f\_2)  deque([0, 1, 2, 3, 4])  4 0  d = collections.deque([i for i in range(0, 5)])  print(d)  print(d.count(2))  1  d = collections.deque([i for i in range(0, 5)])  print(d)  print(d.index(2))  2  d = collections.deque([i for i in range(0, 5)])  d.insert(2, '\*')  print(d)  deque([0, 1, '\*', 2, 3, 4])  d = collections.deque([i for i in range(0, 5)])  d.reverse()  print(d)  deque([4, 3, 2, 1, 0])  d = collections.deque([i for i in range(0, 5)])  print(d)  d.rotate(-2)  print(d)  deque([0, 1, 2, 3, 4])  deque([2, 3, 4, 0, 1]) |
| Defaultdict | словарь, при определении передается функция default\_factory  счетчик символов:  a = 'abrakadabra'  b = collections.defaultdict(int)  for i in a:  b[i] += 1  print(b)  defaultdict(<class 'int'>, {'a': 5, 'b': 2, 'r': 2, 'k': 1, 'd': 1})  счетчик сигналов датчиков:  a = [('a', 1), ('b', 0), ('a', 4), ('b', 2), ('c', 7)]  b = collections.defaultdict(list)  for k,v in a:  b[k].append(v)  print(b)  defaultdict(<class 'list'>, {'a': [1, 4], 'b': [0, 2], 'c': [7]})  счетчик уникальных сигналов датчиков:  a = [('a', 1), ('b', 0), ('a', 4), ('b', 2), ('c', 7), ('a', 1), ('b', 2)]  b = collections.defaultdict(set)  for k,v in a:  b[k].add(v)  print(b)  defaultdict(<class 'set'>, {'a': {1, 4}, 'b': {0, 2}, 'c': {7}})  a = collections.defaultdict(lambda: 'unknown')  a.update(rex='dog', tomas='cat')  print(a)  print(a['rex'])  print(a['fox'])  defaultdict(<function <lambda> at 0x0000021D1036B598>, {'rex': 'dog', 'tomas': 'cat'})  dog  unknown |
| OrderedDict  move\_to\_end ( )  переместить элемент в конец  словаря  move\_to\_end (last=False)  переместить элемент в начало  словаря | упорядоченный словарь, при этом запоминает порядок добавления элементов  a = {'cat': 5, 'mouse':7, 'dog': 1}  b\_1 = collections.OrderedDict(sorted(a.items(), key=lambda x:x[0]))  b\_2 = collections.OrderedDict(sorted(a.items(), key=lambda x:x[1]))  print(b\_1, b\_2, sep='\n')  print(b\_1 == b\_2)  OrderedDict([('cat', 5), ('dog', 1), ('mouse', 7)])  OrderedDict([('dog', 1), ('cat', 5), ('mouse', 7)])  False  a = {'cat': 5, 'mouse':7, 'dog': 1, 'fox':6, 'bird':10}  b\_1 = collections.OrderedDict(sorted(a.items(), key=lambda x:x[0]))  print(b\_1)  b\_1.move\_to\_end('cat')  print(b\_1)  b\_1.move\_to\_end('cat', last=False)  print(b\_1)  сортировка по длине ключа  a = {'cat': 5, 'mouse':7, 'dog': 1, 'fox':6, 'bird':10}  b\_1 = collections.OrderedDict(sorted(a.items(), key=lambda x: len(x[0])))  print(b\_1)  OrderedDict([('cat', 5), ('dog', 1), ('fox', 6), ('bird', 10), ('mouse', 7)]) |
| namedtuple  rename=True  переименовать в допустимое имя  \_make  создать кортеж из списка  \_asdict  преобразует кортеж в словарь OrderedDict  \_replace  изменение кортежа  \_fields  показывает поля кортежа  defaults=[0, 0]  задает значения по умолчанию  .\_fields\_defaults  показывает значения по умолчанию | именованный кортеж.  создание персонажей Класса Person:  Person = collections.namedtuple('Person', 'name, race, health, mana, strenght')  hero\_1 = Person('Aaz', 'Izverg', 100, 0.0, 250)  print(hero\_1.race)  Izverg  Person = collections.namedtuple('Person', 'name, race, \_health, mana, strenght', rename=True )  hero\_1 = Person('Aaz', 'Izverg', 100, 0.0, 250)  print(hero\_1)  Person(name='Aaz', race='Izverg', \_2=100, mana=0.0, strenght=250)  Point = collections.namedtuple('Point', 'x, y, z')  pl\_1 = Point(3, z=4, y=5)  print(pl\_1)  Point(x=3, y=5, z=4)  Point = collections.namedtuple('Point', 'x, y, z')  pl\_1 = Point(3, 4, 5)  t=[5, 10, 15]  pl\_2 = Point.\_make(t)  print(pl\_1)  print(pl\_2)  Point(x=3, y=4, z=5)  Point(x=5, y=10, z=15)  d = pl\_2.\_asdict()  print(d)  OrderedDict([('x', 5), ('y', 10), ('z', 15)])  Point = collections.namedtuple('Point', 'x, y, z')  pl\_1 = Point(3, 4, 5)  pl\_2 = pl\_1.\_replace(x=6)  print(pl\_2.\_fields)  print(pl\_2)  ('x', 'y', 'z')  Point = collections.namedtuple('Point', 'x, y, z', defaults=[0, 0])  pl\_1 = Point(3,)  print(pl\_1)  print(pl\_1.\_fields\_defaults)  Point(x=3, y=0, z=0)  {'y': 0, 'z': 0} |
| ChainMap  new\_child()  добавляет словарь в начало коллекции, или пустой словарь()  maps[]  возвращает словарь по индексу  parents  возвращает родителей коллекции  list()  список всех ключей  list(d.values)  список всех значений | цепочка отображений, организовывает работу с несколькими словарями  при поиске по ключу выводит только первое совпадение  d\_1 = {'a':1, 'b':2, 'c':3, 'd':4}  d\_2 = {'a':14, 'b':24, 'g':34, 'h':44}  d\_3 = collections.ChainMap(d\_1, d\_2)  print(d\_3)  print(d\_3['a'])  ChainMap({'a': 1, 'b': 2, 'c': 3, 'd': 4}, {'a': 14, 'b': 24, 'g': 34, 'h': 44})  1  d\_1 = {'a':1, 'b':2, 'c':3, 'd':4}  d\_2 = {'a':14, 'b':24, 'g':34, 'h':44}  d\_3 = collections.ChainMap(d\_1, d\_2)  d\_4 = d\_3.new\_child(d\_1)  print(d\_4)  print(d\_4.maps[2])  print(d\_4.parents)  ChainMap({}, {'a': 1, 'b': 2, 'c': 3, 'd': 4}, {'a': 14, 'b': 24, 'g': 34, 'h': 44})  {'a': 14, 'b': 24, 'g': 34, 'h': 44}  ChainMap({'a': 1, 'b': 2, 'c': 3, 'd': 4}, {'a': 14, 'b': 24, 'g': 34, 'h': 44})  добавление в 1 словарь  d\_1 = {'a':1, 'b':2, 'c':3, 'd':4}  d\_2 = {'a':14, 'b':24, 'g':34, 'h':44}  d\_3 = collections.ChainMap(d\_1, d\_2)  d\_4 = d\_3.new\_child()  print(d\_4)  d\_4['a'] = 333  print(d\_4)  print(list(d\_4))  print(list(d\_4.values()))  ChainMap({}, {'a': 1, 'b': 2, 'c': 3, 'd': 4}, {'a': 14, 'b': 24, 'g': 34, 'h': 44})  ChainMap({'a': 333}, {'a': 1, 'b': 2, 'c': 3, 'd': 4}, {'a': 14, 'b': 24, 'g': 34, 'h': 44})  ['a', 'b', 'g', 'h', 'c', 'd']  [333, 2, 34, 44, 3, 4] |
|  |  |
|  |  |
| **hashlib** |  | import hashlib  print(hashlib.sha1(b'Hello World!').hexdigest())  print(hashlib.sha1(b'Secret Word!' + b'Hello world!').hexdigest())  a = 'qwe'  b = hashlib.sha1(a.encode('utf-8')).hexdigest()  print(b)  s = hashlib.sha1(b'Hello World!').hexdigest()  print(hashlib.sha1(b'Secret word!' + bytes(s.encode('utf-8')) ).hexdigest()) |
| **itertools()**  #itertools |  | сборник полезных итераторов  import itertools  <https://pythonworld.ru/moduli/modul-itertools.html> |
| **permutations**  #permutations | генериует все перестановки и размещения для итерируемого объекта, размещение отличается от перестановки ограничением на количество доступных ячеек  for i in permutations('abc'): # перестановка  print(i, end=' ') # abc acb bac bca cab cba  for i in permutations('abc', 2): # размещение  print(i, end=' ') # ab ac ba bc ca cb |
| **product**  #product | размещения с повторениями  for i in product('abc', repeat=2):  print(i, end=' ') # aa ab ac ba bb bc ca cb cc |
| **combinations**  #combinations | сочетания без повторений  for i in combinations('abcd', 2):  print(i, end=' ') # ab ac ad bc bd cd |
| **combinations**  **\_with**  **\_replacement** | сочетания без повторений, но с повторами  for i in combinations\_with\_replacement('abcd', 2):  print(i, end=' ') # aa ab ac ad bb bc bd cc cd dd |
| **count()** | бесконечная арифметическая прогрессия с первым членом start и шагом step  (start=0, step=1)  for i in itertools.count(): |
| **.cycle(iter)** | возвращает по одному значению из последовательности, повторенной бесконечное число раз  cycle(iterable) |
| **.repeat()** | повторяет elem n раз  repeat(elem, n=Inf) |
| **.accumulate()** | аккумулирует суммы  accumulate(iterable) |
| **.chain()** | возвращает по одному элементу из первого итератора, потом из второго, до тех пор, пока итераторы не закончатся.  .chain(\*iterables) |
| **.compress()** | .compress(data, selectors)  compress('ABCDEF', [1,0,1,0,1,1]) --> A C E F |
|  |  |
|  |  |
| **Decimal** |  | создание чисел с фиксированной точностью  from decimal import Decimal  import decimal  i = Decimal('4.24')  decimal.getcontext().prec=12  # точность знаков после точки |
|  |  |  |
|  |  |  |

# Дополнительные модули и библиотеки.

Модуль (библиотека) – это файл с программой.

Варианты подключения модуля:

1. pip3 install beautifulsoup4

|  |  |  |
| --- | --- | --- |
| **beautifulsoup** | классы | парсер для синтаксического разбора файлов HTML/XML  pip3 install beautifulsoup4  from bs4 import BeautifulSoup as bs # для html  from bs4 import BeautifulStoneSoup # для XML  import bs4 # для всего |
| BeautifulSoup  prettify()  get\_text()  title  title.string  div  div['class']  find\_all('div')  find\_all('a') | создание экземпляра класса (исходный текст в кодировке html)  soup = BS(text\_orign, 'html.parser')  print (soup.prettify())  преобразует исходный код к красивому и читаемому.  print (soup.get\_text())  чистит исходный код от тегов и комментариев  print (soup.title)  возвращает заголовок страницы (вкладки) с тегом  <title>Новости дня в России и мире — РБК</title>  print (soup.title.string)  возвращает заголовок страницы без тега  Новости дня в России и мире — РБК  print (soup.div)  возвращает первый блочный элемент  print (soup.div['class'])  возвращает класс блочного элемента  soup\_1 = soup.find\_all('div')  print (soup\_1)  возвращает все указанные блочные элементы  print (len(soup\_1))  возвращает кол-во блочных элементов  soup\_1 = soup.find\_all('a')  for n in soup\_1:  print(n.get('href','')) # print(n['href'])  возвращает все ссылки  li = soup.find\_all('span', attrs={'class': "total-users"})  total\_user = geek\_users.find(class\_='total-users')  total\_u = geek\_users.find\_all(class\_='total-users')  for n in total\_u:  print(n.contents)  result=re.findall('[0-9 ]{3,}', spans[1].get\_text())[0].strip() |
| **requests** | get | выполняет запросы к серверу и обрабатывает ответы  req = requests.get('https://yandex.ru/')  print (req)  # <Response [200]> # успешный ответ  req = requests.get('https://yandex.ru/')  text\_original = req.text # свойство сайта, исходник  print (req)  print(text\_original)  **API**  import requests  import json  link = 'https://samples.openweathermap.org/data/2.5/  weather?q=London,uk&appid=b6907d289e10d714  a6e88b30761fae22'  resp = requests.get(link)  text\_1 = resp.text  date = json.loads(text\_1)  print(date['main']['temp'])  280.32  **AVIASALES**  import requests  import json  city = 'LED'  link = 'http://min-prices.aviasales.ru/calendar\_preload?origin='+city+'&destination=NCE'  link\_1 = 'http://min-prices.aviasales.ru/calendar\_preload?one\_way=true&origin='+city+'&destination=NCE'  resp = requests.get(link).text  date = json.loads(requests.get(link).text)  date\_1 = json.loads(requests.get(link\_1).text)  print(date['best\_prices'][0])  print(date\_1['best\_prices'][0]) |
| **flask** |  | from flask import Flask  создание локального сервера  from flask import Flask  app = Flask(\_\_name\_\_) #создание объекта app, где  будет храниться наш сайт,  экземпляр Flask.Передаем  переменную - имя нашего  файла.  @app.route('/') #добавляем путь  (начальная страница)  def index(): # функция, которая генерирует  нашу страницу  return 'Привет! Ты добрался!' #Возвращает текст,  отображающийся  на первой cтр  if \_\_name\_\_ == '\_\_main\_\_': # запуск сайта, только  когда запускается файл,  а не импотритуется  app.run() # запустить сайт, если эта  страничка главная  import requests  print (requests.get('http://127.0.0.1:5000/').text)  обращение к созданному локальному серверу |
| **lxml** |  | библиотека для обработки XML и HTML файлов, используется для парсинга веб-страниц. |
| **timeit** | .timeit | подсчет времени работы программы  import timeit  print(timeit.timeit(x = 2 + 2))  из терминала:  python -m timeit -n 1000 -s "import Less\_4\_Task\_2a" "Less\_4\_Task\_2a.simple\_1(100)" |
| **cPrifile** | .run | подсчет времени работы программы  import cPrifile  cProfile.run('main()') |
| **functools** |  | сборник функций высокого уровня: взаимодействующих с другими функциями или возвращающие другие функции.  import functools  @functools.lru\_cash() |
|  | @functools.lru\_cash() | декоратор, который сохраняет результаты maxsize последних вызовов. Это может сэкономить время при дорогих вычислениях, если функция периодически вызывается с теми же аргументами. |
| **binarytree** |  | создание бинарных деревьев  import binarytree |
|  | .tree | a = binarytree.tree(height=4, is\_perfect=False) |
|  | .bst | a = binarytree.bst(height=4, is\_perfect=True) |
|  | .Node | a = binarytree.Node(7)  a.left = binarytree.Node(3)  a.right = binarytree.Node(11)  a.left.left = binarytree.Node(1)  a.left.right = binarytree.Node(5)  a.right.left = binarytree.Node(9)  a.right.right = binarytree.Node(13)  print(a) |
|  | .build | a = binarytree.build([7,3,11,1,5,9,3,None,2,None,6])  print(a) |
|  |  |  |
|  |  |  |
|  |  |  |
| **numpy**  #numpy  #np |  | import numpy as np  библиотека, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.  Основной объект NumPy - однородный многомерный массив элементов (обычно чисел), одного типа (numpy.ndarray).  Срезы:  *Вторая строка, третий столбец:*  b[2,3] или b[(2,3)] или b[2][3]  b[:,2] *# Третий столбец*  b[:2] *# Первые две строки*  b[1:3, :, : ] *# Вторая и третья строки*  b[-1] *# Последняя строка. Эквивалентно b[-1,:]*  x[1, 2, ...] эквивалентно x[1, 2, :, :, :],  x[... , 3] то же самое, что x[:, :, :, :, 3] и  x[4, ... , 5, :] это x[4, :, :, 5, :].  FOR:  for str in a: # перебирает построчно  print(str)  for item in a.flat: # перебирает поэлементно  print(item)  Математические операции с массивами:  <https://docs.scipy.org/doc/numpy/reference/routines.math.html> |
| array() | cоздает объект типа ndarray из обычных списков или кортежей. Трансформирует вложенные последовательности в многомерные массивы. Тип элементов массива зависит от типа элементов исходной последовательности (но можно и переопределить его в момент создания).  dtype – определяет тип переменных в массиве  a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  print(a)  print(type(a))  [[1 2 3]  [4 5 6]  [7 8 9]]  <class 'numpy.ndarray'>  a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype = np.bool)  print(a)  print(type(a))  [[ True True True]  [ True True True]  [ True True True]]  <class 'numpy.ndarray'> |
| zeros() | создает массив из нулей, (по умолчанию тип создаваемого массива — float64)  dtype – определяет тип переменных в массиве  b = np.zeros((3, 5))  print(b)  [[0. 0. 0. 0. 0.]  [0. 0. 0. 0. 0.]  [0. 0. 0. 0. 0.]]  b = np.zeros((3, 5), dtype=np.int)  print(b) |
| ones() | создает массив из единиц аналогично zeros() |
| eye() | создаёт единичную матрицу (двумерный массив)  b = np.eye((5), dtype=np.int)  print(b)  [[1 0 0 0 0]  [0 1 0 0 0]  [0 0 1 0 0]  [0 0 0 1 0]  [0 0 0 0 1]] |
| empty() | создает массив без его заполнения. Исходное содержимое случайно и зависит от состояния памяти на момент создания массива (то есть от того мусора, что в ней хранится):  b = np.empty((2,3), dtype=np.int)  print(b) |
| arange() | создает последовательность чисел, возвращает массивы, и принимает не только целые значения.  np.arange(<от>, <до не вкл>, <шаг>)  e = np.arange(10, 30, 5)  print(e)  [10 15 20 25]  ar = np.arange(12)  print(ar)  [ 0 1 2 3 4 5 6 7 8 9 10 11] |
| linspace() | создает последовательность чисел, аналог arange()  np.linspace(<от>, <до вкл>, <кол-во чисел>) |
| random()  .randn  .sample  .randint  .shuffle | создание массивов со случайными элементами  randn - из нормального распределения(со средним 0 и среднеквадратическим отклонением 1 )  f = np.random.randn(3, 4)  sample - от 0 до 1  a = np.random.sample((3, 4))  print(a)  randint - массив из целых чисел в указ диапазоне  c = np.random.randint(0, 100, (3, 4))  print(c)  перемешивает случайным образом элементы данного массива  np.random.shuffle(c) |
| choice() | возвращает случайно выбранные элементы из заранее заданного массива  a = np.arange(-10, 0)  d = np.random.choice(a, (3, 4))  print(d) |
| fromfunction() | применяет функцию ко всем комбинациям индексов  numpy.fromfunction(function, shape, \*\*kwargs)  **shape**- целое число, список, кортеж целых чисел  Задает размеры необходимого массива.  **dtype**- тип данных NumPy (необязательный)  Определяет тип данных выходного массива.  Аргументы функции номер строки и столбца.  def func\_1(str, stolb):  return 5 \* (str + stolb)  print (np.fromfunction(func\_1, (4, 4), dtype = int))  [[ 0 5 10 15]  [ 5 10 15 20]  [10 15 20 25]  [15 20 25 30]] |
| set\_printoptions | настраивает печать массивов "под себя".  precision : количество отображаемых цифр после запятой (по умолчанию 8).  threshold : количество элементов в массиве, вызывающее обрезание элементов (по умолчанию 1000).  edgeitems : количество элементов в начале и в конце каждой размерности массива (по умолчанию 3).  linewidth : количество символов в строке, после которых осуществляется перенос (по умолчанию 75).  suppress : если True, не печатает маленькие значения в scientific notation (по умолчанию False).  nanstr : строковое представление NaN (по умолчанию 'nan').  infstr : строковое представление inf (по умолчанию 'inf').  formatter : позволяет более тонко управлять печатью массивов. |
| view() | c = a.view() # создает новый объект массива, являющийся представлением тех же данных. |
| copy() | создание нового объекта - массива с новыми данными  d = a.copy()  c = a[1:3, 2:4].copy() |
| ndim | выводит размерность массива (двухмерный …)  print(a.ndim) # 2 |
| shape | выводит форму массива, всегда кортеж, размер которого равен размерности массива. Каждый элемент этого кортежа - это размер в каждом измерении.  print(a.shape) # (2, 3) - 2 строки 3 столбца |
| size | выводит общее количество элементов массива  print(a.size) # 9 |
| add | сложение векторов, матриц  c = a + b # массивы np можно складывать +  c = np.add(a, b) |
| subtract | вычитание векторов, матриц  d = a - b # массивы np можно вычитать -  d = np.subtract(a, b) |
| multiply | умножение вектора и матрицы на скаляр  d = a \* 3  d = np.multiply(a, 3) |
| dot | умножение вектора и матрицы на скаляр, скалярное произведение векторов, произведение матриц  d = a \* 3 # умножение вектора на скаляр  d = np.dot(a, 3) # функция np.dot  d = a.dot(3) # метод dot  sp = a @ b # скалярное произведение векторов  sp = a.dot(b) # скалярное произведение векторов  d = a.dot(b) # произведение матриц стр\_1= стоб\_2 |
| linalg  .matrix\_power  .det  .matrix\_rank  .inv | возведение в степень квадратичной матрицы  d = np.linalg.matrix\_power(b, 3)  считает определитель матрицы  d = np.linalg.det(b)  ранг матрицы - это число линейно независимых строк данной матрицы  r = np.linalg.matrix\_rank(b)  расчет обратной матрицы (матрица, которая при умножении на исходную матрицу даёт единичную матрицу), если определитель матрицы равен 0 - обратной матрицы не существует:  f\_inv = np.linalg.inv(f) |
| transpose | транспонирование матриц  a\_t = a.T # атрибут T  a\_t = np.transpose(a) # функция np.transpose  a\_t = a.transpose() # метод transpose |
| sum() | сумма элементов массива  d = a.sum() # всех элементов  d = a.sum(axis=0) # по столбиками  d = a.sum(axis=1) # по строкам |
| min() | минимальный элемент массива, аналогично sum  b = a.min() # минимальное значение |
| max() | максимальный элемент массива, аналогично sum |
| mean() | среднее значение, аналогично sum  a.mean(axis=1) # средние значения строк |
| std() | среднее квадратическое отклонение, аналогично sum |
| var() | дисперсия |
| corrcoef() | коэффициент корреляции Пирсона  np.corrcoef(a, b) |
| булева индексация | & - и  ∣ - или  ~ - не  a = [19 9 11 3 14 14 15 16 8 5]  b = a[a > 10] # [19 11 14 14 15 16]  c = a[(a > 0) & (a % 2 == 0)] # [14 14 16 8] |
| where | индексы истинных значений из булевой индексации  ind1 = np.where(a > 10) # (array([0, 2, 4, 5, 6, 7]),)  d = a[ind1] # [19 11 14 14 15 16]  e = a[[0, 4, 7]] # [19 14 16] - возврат значений по инд  D\_inv = np.where(D\_inv < 0, 0, 1) # заменяет на 0  все отрицательные числа,  а на 1 все оставшиеся числа)  cond = [[True, False], [False, True]] # маска  a = np.where(cond, x, y) # использование cond как  маски для создания нового массива |
| argsort | сортировка (возвращает массив из индексов массива a в порядке их возрастания по заданной оси)  ind = a.argsort(axis=0) # в столбцах стоят индексы строк массива a, расположенные по возрастанию значений столбца  [[0 1 2 0]  [1 2 1 2]  [2 0 0 1]]  ind1 = a[:, 1].argsort() # отсорт второй столбец  [1 2 0]  c = a[a[:, 1].argsort(), :] # массив отсорт по 2 столбцу  [[2 2 2 4]  [5 2 0 3]  [2 4 3 3]]  b = a[[1, 2, 0], :] # то же самое вручную |
| reshape() | изменение формы массива  b = np.reshape(a, (3, 4)) # функция  b = a.reshape(3, 4) # метод  b = a.reshape(3, -1) #если не знаем кол-во столбцов |
| resize() | изменение формы исходного массива  a.resize(3, 4) |
| flatten() | преобразует матрицу в одномерный вектор.  c = a.flatten() |
| ravel() | преобразует матрицу в одномерный вектор.  order = 'C' - массив собирается из строк  order = 'F' - массив собирается из столбцов  b = np.ravel(a)  b = np.ravel(a, order='F') |
| hstack() | соединяет массивы горизонтально (дописывает один правее другого)  c = np.hstack((a, b)) |
| vstack() | соединяет массивы вертикально (дописывает один под другим)  c = np.vstack((a, b)) |
| dstack() | соединяет массивы вглубину (вдоль новой третьей оси)  e = np.dstack((a, b)) |
| concatenate() | соединяет массивы по указанной оси  c = np.concatenate((a, b), axis=0) # вертикально  c = np.concatenate((a, b), axis=1) # горизонтально |
| сolumn\_stack() | объединяет одномерные массивы в качестве столбцов двумерного массива |
| row\_stack() | объединяет одномерные массивы в качестве строк двумерного массива |
| hsplit() | разбивает массив вдоль горизонтальной оси, указывается, либо число возвращаемых массивов одинаковой формы, либо номера столбцов, после которых массив разрезается "ножницами".  a = np.arange(12).reshape((2, 6)) |
| vsplit() | разбивает массив вдоль вертикальной оси |
| array\_split() | указываются оси, вдоль которых произойдет разбиение массива. |
| save | запись массива в файл  np.save('a.npy', a) |
| load | чтение массива из файла  a = np.load('a.npy') |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **pandas**  #pandas  #pd |  | import pandas as pd  библиотека, позволяющая удобно работать с таблицами.  https://pandas.pydata.org/pandas-docs/stable/reference/frame.html  pd.options.display.max\_columns = 100  # позволяет просматривать необходимое кол-во столбцов  planets = ['Mercury', 'Venus', 'Earth', 'Mars',  'Jupiter', 'Saturn', 'Uranus', 'Neptune']  masses = [0.055274, 0.815, 1.0, 0.107,  317.8, 95.0, 14.6, 17.147]  planets\_s = pd.Series(masses, index=planets) # Series  planets\_df = pd.DataFrame(planets\_s) # DataFrame |
| **Series** | упорядоченный одномерный словарь с индексами  a = ['a', 'b', 'c'] # Series из списка  b = pd.Series(a)  print(b)  0 a  1 b  2 c  dic\_1 = {'1st': 'a', '2nd': 'b', '3rd': 'c'} # из словаря  c = pd.Series(dic\_1)  1st a  2nd b  3rd c  a = [['a', 'b', 'c'], ['d', 'e','f']] #Series из вложен списка  d = pd.Series(a)  0 [a, b, c]  1 [d, e, f]  b = pd.Series(np.arange(1, 12, 2)) # из numpy  a = ['a', 'b', 'c'] # назначение индекса при создании  ind = ['2019-04-01', '2019-04-03', '2019-04-05']  b = pd.Series(a, index=ind)  # b[0] или b['2019-04-01']  2019-04-01 a  2019-04-03 b  2019-04-05 c  print('2019-04-03' in b) # True  e = pd.Series(a, dtype=np.float32) # опр типа данных  print(c['1st']) # a значения по индексу  print(b[[0, 1]]) # 0 a 1 b [[]] - несколько знач  b[[0, 1]] = 7  print(b) # 0 7 1 7 -изменение знач  b = pd.Series([1, 2, 3, 4, 5])  print(b[b > 3] \* 2) # 3 8 4 10 -фильтрация и т.п.  b = b[b > 3] \* 2  e[e > 2] = -1  & и  ∣ или  ∼ не  каждое условие ставится в круглые скобки:  e[(e > 2) | (e == 1)]  Можно складывать Series с одинаковыми индексами (сложаться значения):  ser3 = ser1 + ser2  ser3 = ser1.reset\_index()[0] + ser2  # если индексы разные, их можно сбросить, но должно быть одинаковое кол-во элементов, если кол-во разное, то одинаковые сложаться, а в остальных будет значение NaN  длина Series:  len(s)  s.size  s.shape  c.index # выводит значения индекса массива  print(b.index) # RangeIndex(start=0, stop=3, step=1)  print(c.index) #Index(['1st', '2nd', '3rd'], dtype='object') |
| .index | c.index = [1, 2, 3] # изменение индекса |
| .reset\_index() | сброс индекса  ser1.reset\_index()[0] |
| .values | f.values # вывод значения - массив numpy.ndarray  print(c.values) # ['a', 'b', 'c'] |
| .astype() | e = e.astype(np.float32) # изменение типа данных  import numpy as np # изменение типа данных  e = pd.Series(a, dtype=np.float32) |
| .name | задает имя объекта или индекса  b.name = 'letter' # имя массива  b.index.name = 'date' # имя индекса массива |
| .head() | выводит первые несколько значений (по умолч 5)  c.head(2) |
| .tail() | выводит последние несколько знач (по умолч 5)  c.tail(3) |
| .append() | добавление значений, массивов(не изм исх массив)  g = e.append(f) |
| .drop() | удаление элементов по индексам(не изм массив)  h = g.drop([0, 4, 2]) |
| .copy() | копирование массива  e = d.copy() |
| .to\_pickle() | запись массива Series в файл  h.to\_pickle("h.pkl") |
| .read\_pickle() | чтение массива Series из файла  h = pd.read\_pickle("h.pkl") |
| **DataFrame** | упорядоченная двухмерная таблица данных  столбцы в DataFrame являются объектами Series  строки исследование  столбцы однотипные данный  a = {"col1": [1, 2, 4, 5],  "col2": ["a", "c", "e", "g"]}  b = pd.DataFrame(a) # из словаря  col1 col2  0 1 a  1 2 c  2 4 e  3 5 g  print(b['col1']) # выводит столбец Series  print(b[['col1']]) # выводит столбец DataFrame  print(b[['col1', 'col2']]) # выводит нужные столбцы  print(b.col1[2]) # выводит элемент из столбца  type(b["col1"]) # выводит тип данных  c = pd.Series([3, 1, 2])  d = pd.DataFrame(c) # конвертация данных в DF  задать названия столбцов  x = pd.DataFrame(ds\_boston['data'], columns=ds\_boston['feature\_names'])  d['density'] = d['population'] / d['square'] \* 1000000  # добавление нового столбца  m\_df.loc[m\_df["book\_title"].notnull(), "quantity"] = 1  # значение непустых строк столбца заменить на 1  m\_df["quantity"].fillna(0, inplace=True)  # значение пустых строк столбца заменить на 0 |
| .T | Транспонирование датафрейма, столбцы становятся строками, применяется, когда много столбцов и они все не умещаются на экран  df.head(3).T |
| .append() | добавление  ignore\_index=True - индексы соединяемых таблиц не учитывются, в рез таблице назначаются стандартные индексы с 0  df.append(df2, ignore\_index=True)  df = pd.DataFrame(columns=['A']) |
| .drop() | удаление (axis=0 строка, axis=1 столбец)  m\_df.drop("price", axis=1, inplace=True) # столбец  m\_df.drop(1, axis=0, inplace=True) # строка  d.drop(['density'], axis='columns') # удаление столбца  del df['density'] # удаление столбца  df.drop([0, 1]) # удаление строк по индексу  df.drop(index='cow', columns='small') # строк и столб |
| .dropna() | удаление строк или столбцов содержащие пустые значения  df.dropna(axis = 'rows') |
| .rename() | переименование  df.rename(columns={"A": "a", "B": "c"})  df.rename(index={0: "x", 1: "y", 2: "z"})  df.rename(index=str).index  df.rename(str.lower, axis='columns') # заменить  названия столбцов на строчные буквы  df.rename(columns={"A": "a", "B": "b", "C": "c"}, errors="raise") # сообщить об ошибке |
| .astype() | изменение типа данных в столбце  m\_df["quantity"] = m\_df["quantity"].astype(int) |
| .shape | выводит форму массива  print(b.shape) # (4, 2) |
| .columns | выводит названия столбцов  print(b.columns) # Index(['col1', 'col2'], dtype='object')  задать названия столбцов  x = pd.DataFrame(ds\_boston['data'], columns=ds\_boston['feature\_names']) |
| .index | выводит информация об индексах  print(b.index) # RangeIndex(start=0, stop=4, step=1) |
| .set\_index() | назначение значений одного из столбцов индексами  rged\_df.set\_index("author\_id", inplace=True) |
| .reset\_index() | сброс индексов (по умолчанию текущие индексы становятся столбцом)  b.reset\_index(inplace=True)  merged\_df.reset\_index(drop=True, inplace=True)  # текущие индексы удалить |
| .info | общая информация о массиве  наименование столбцов, сколько не пустых значений, тип данных в столбцах, размер df в мегабайтах |
| .describe() | статистические характеристики по столбцам с числовыми значениями: среднее значение, среднее квадратическое отклонение, максимум, минимум, квантили и пр.  col1  count 4.000000  mean 3.000000  std 1.825742  min 1.000000  25% 1.750000  50% 3.000000  75% 4.250000  max 5.000000 |
| .head() | выводит первые несколько строк (по умолч 5)  print(b.head(3)) |
| .tail() | выводит последние несколько строк (по умолч 5)  print(b.tail(3)) |
| .loc | выводит данные из строк таблицы по индексу строки и названию столбца, + булева индексация  print(b.loc[[0, 2]]) # 0 и 2 строчка, все столбцы  print(b.loc[[0, 2], 'col1']) # 0 и 2 строчка, столбец col1  print(b.loc[2, 'col1']) # элемент из 3 стр и ст col1  print(b.loc[(b["col1"] > 3) | (b["col1"] == 1), "col2"])  m\_df.loc[m\_df["book\_title"].notnull(), "quantity"] = 1  # значение непустых строк столбца заменить на 1  m\_df["quantity"].fillna(0, inplace=True)  # значение пустых строк столбца заменить на 0  print(sber.loc[(sber['name'] == 'Количество заявок на потребительские кредиты')]) |
| .iloc | выводит данные из строк таблицы по индексу строки и индексы столбца  print(b.iloc[2, 1]) # элемент из 3 стр и 2 столбца |
| .between(1, 3) | все строки между указанными (включая оба конца), возвращает True и False  print(b["col1"].between(1, 3))  print(b[(b['col1'].between(3, 6)) & (~b['col2'].isin(['a', 'z']))]) |
| .isin() | все строки содержащие значения из списка, возвращает True и False  print(b["col2"].isin(["a", "z"]))  print(b[(b['col1'].between(3, 6)) & (~b['col2'].isin(['a', 'z']))]) |
| .query() | упрощенный синтаксис для булевой индексации: подается строка, содержащая булевы условия на значения столбцов (имена столбцов без кавычек).  print(b.query('(col1 < 6) & (col2 > "c")')) |
| .copy() | копирование массива  e = d.copy() |
| .sample() | случайный выбор строк из массива  frac - доля возвращаемых строк от общего числа строк (число от 0 до 1)  n - кол-во возвращаемых строк  replace - индикатор возможного повторения строк в выборке (True или False)  print(b.sample(frac=0.5, replace=True))  print(b.sample(frac=1, replace=False))  # перемешивает строки массива |
| .to\_csv() | сохранение массива DataFrame  sep - символ для разделения значения столбцов между собой. По умолч это ",", но можно использовать ";", "\t" и др.  index - булево значение, индикатор нужно ли сохранять столбец индексов.  b.to\_csv('test.csv', sep=';', index=False) |
| .read\_csv() | чтение массива из файла  b = pd.read\_csv("test.csv", sep=";") |
| .to\_excel() | сохранение массива excel |
| .read\_excel() | чтение массива excel |
| .to\_pickle() | сохранение массива pickle |
| .read\_pickle() | чтение массива pickle |
| .merge() | слияние данных, сопоставление столбцов  on - столбец по к-рому будет слияние  how - принцип слияния:  "inner" - внутреннее слияние, строки, к-рые  присутствуют в обоих таблицах  "left" - все строки из левой таблицы  "right" - все строки из правой таблицы  "outer" - внешнее слияние, все строки из всех  таблиц.  pd.merge(authors, books, on='author\_id', how='inner') |
| .isnull()  .isna() | выводит все пустые значения (NaN)  merged\_df[merged\_df["author\_name"].isnull()] |
| .notnull() | выводит все непустые значения  merged\_df[merged\_df["author\_name"].notnull()] |
| .fillna() | заполняет пропущенные значения 'своим'  merged\_df["author\_name"] = merged\_df["author\_name"].fillna("unknown")  metod = 'ffill'  df.fillna(metod = 'ffill')  # последнее не пропущенное значение дублируется во все пропущенные  df.fillna(df.mean()) |
| .sort\_values() | сортировка массива  merged\_df.sort\_values(by="author\_id", inplace=True) |
| .concat() | соединяет таблицы (axis=0 вертикально, axis=1 горизонтально; ignore\_index=True без сохранения индексов, по умолч с сохранением)  df = pd.concat([m\_df, df1], axis=0, ignore\_index=True) |
| .max() | df4["price"].max() - максимум |
| .min() | df4["price"].min() - минимум |
| .mean() | df4["price"].mean() - среднее  df.fillna(df.mean()) |
| .median() | df4["price"].median() - медиана |
| .std() | df4["price"].std() - средн квадратическое значение |
| .sum() | df.sum() |
| .var() | df4["price"].var() - дисперсия |
| .nlargest() | выводит наибольшие значения  .nlargest(<сколько>, <по какому столбцу>)  df4.nlargest(3, "price") |
| .nsmallest() | выводит наименьшие значения  df4.nsmallest(3, "price") |
| .unique() | уникальные значения заданного столбца  df4["author\_name"].unique() |
| .nunique() | выводит количество уникальных значений  df4["author\_name"].nunique() |
| .value\_counts() | выводит сколько раз каждое уникальное значение появляется в данном столбце  df4["author\_name"].value\_counts() |
| .count() | выводит кол-во не пустых значений по столбикам  df.count() |
| .apply() | выполняет функции, к-рых нет в pandas  df4["author\_name"].apply(lambda x: x.upper()) |
| .groupby() | группировка по повторяющимся значениям выбранного столбца, что прозволяет вычислять агрегированные значения  groupby = df4.groupby("author\_name")  df4.groupby("author\_name")["price"].mean()  df4.groupby("author\_name")["price"].mean().agg({"price": "max", "total": "count"})  print(titanic\_df.groupby(['PClass', 'Survived'])['PassengerID'].count()) |
| .agg() | вычисление агрегированных функций сгруппированных значений, позволяет вычислять одновременно несколько различных агрегирующих функций от разных столбцов  df4.groupby("author\_name")["price"].mean().agg({"price": "max", "total": "count"})  price 700.0  total 3.0 |
| **matplotlib**  #matplotlib |  | библиотека для визуализации данных, для работы с 2-х и 3-х мерной графикой  import matplotlib.pyplot as plt  import matplotlib as mpl  from matplotlib import pyplot as plt  from pylab import rcParams  %matplotlib inline # отображ графиков в нотебуке  юпитера, а не в отдел окне  %config InlineBackend.figure\_format = 'svg'  # магическая команда позволяет рисовать графики в формате svg (масштабируемая векторная графика), это придаёт изображениям большую чёткость.  rcParams["figure.figsize"] = 9, 12  # размер изображения в дюймах  Цвета  'b' - синий цвет  'g' - зеленый цвет  'r' - красный цвет  'c' - голубой цвет  'm' - пурпурный цвет  'y' - желтый цвет  'k' - черный цвет  'w' - белый цвет  Полные названия цветом - в конце файла |
| .plot() | построение линейных диаграмм (ломаная линия)  Можно использовать данные из массивов numpy, pandas  color - цвет графика (любым способом, например,  символ из набора {‘b’, ‘g’, ‘r’, ‘c’, ‘m’, ‘y’, ‘k’, ‘w’})  linestyle - стиль линии  ‘-‘ или ‘solid’ - непрерывная линия  ‘--‘ или ‘dashed’ - штриховая линия  ‘-.’ или ‘dashdot’ - штрихпунктирная линия  ‘:’ или ‘dotted’ - пунктирная линия  ‘None’ или ' ' - не отображать линию  linewidth - толщина линии  fmt - тип маркера (‘.’ Точка (point marker)  ‘,’ пиксель (pixel marker)  ‘o’ окружность (circle marker)  ‘v’ ('1') треугольник вниз (triangle\_down marker)  ‘^’ ('2') треугольник вверх(triangle\_up marker)  ‘<‘('3') треугольник влево (triangle\_left marker)  ‘>’ ('4')треугольник вправо (triangle\_right marker)  ‘s’ квадрат (square marker)  ‘p’ пятиугольник (pentagon marker)  ‘\*’ звезда (star marker)  ‘h’ шестиугольник (hexagon1 marker)  ‘H’ шестиугольник (hexagon2 marker)  ‘+’ плюс (plus marker)  ‘x’ х-образный маркер (x marker)  ‘D’ ромб (diamond marker)  ‘d’ ромб (thin\_diamond marker)  ‘|’ вертикальная линия (vline marker)  ‘\_’ горизонтальная линия (hline marker))  plt.plot() # выводит пустое поле  plt.show()  plt.plot([1, 7, 3, 5, 11, 1]) # по y значения из списка  plt.show() # по x соответст индексы  x = np.arange(0, 11) # x - массив numpy  y = x \*\* 2 # y - массив numpy  plt.plot(x, y, label='steel price') # x, y, название граф.  plt.show()  planets\_info.plot() # график из pd Series, DataFrame  plt.plot(planets\_s)  plt.plot(planets\_df)  b.plot(kind='bar') # столбчатая диаграмма - см bar.  planets\_s.plot(kind="bar")  planets\_df.plot(kind="bar")  plt.bar(planets\_df.index, planets\_df[0])  plt.plot(x, y, color='red') # параметры отображ граф  plt.plot([1, 7, 3, 5, 11, 1], ':') # сразу после координат  можно не подписывать  plt.plot([1, 7, 3, 5, 11, 1], ':', linewidth=10)  plt.plot([1, 7, 3, 5, 11, 1], 'r:') # красная пунктирная  plt.plot([1, 7, 3, 5, 11, 1], 'rx') # красные крестики  без линии  # отображение нескольких графиков - 1 способ:  x = [1, 5, 10, 15, 20]  y1 = [1, 7, 3, 5, 11]  y2 = [i\*1.2 + 1 for i in y1]  y3 = [i\*1.2 + 1 for i in y2]  y4 = [i\*1.2 + 1 for i in y3]  plt.plot(x, y1, '-', x, y2, '--', x, y3, '-.', x, y4, ':')  # отображение нескольких графиков - 2 способ:  plt.plot(x, y1, '-')  plt.plot(x, y2, '--')  plt.plot(x, y3, '-.')  plt.plot(x, y4, ':') |
| .scatter() | построение диаграмм разброса (точечные диаграммы)  plt.scatter(x, y)  plt.show() |
| .bar | построение столбчатых диаграмм  x = np.arange(0, 11)  y = x \*\* 2  plt.bar(x, y, label='steel price')  plt.show()  plt.bar(planets\_df.index, planets\_df[0])  b = pd.DataFrame(a)  b.plot(kind='bar')  plt.show()  planets\_info.plot(kind="bar", logy=True)  # логарифмический масштаб по оси y |
| .barh | горизонтальная столбчатая диаграмма |
| .hist | гистограмма  hist\_info = plt.hist(y, bins=3)  # значения по x разбивается на заданное число промежутков bins  plt.hist(y, bins=3, orientation="horizontal", ec="black")  # ec - цвет границы бинов (обрамление столбика)  plt.hist(bm\_price, color="lightgrey", ec="white")  # color - цвет бина (столбика) |
| .pie | пирожковая диаграмма (доли круга) |
| .box | блок-график |
| area | закрашенная площадь участка b.plot(kind='area') |
| hexbin | гексагональная диаграмма |
| .step | ступенчатый график |
| .stackplot | стековый график |
| .stem | stem-график |
|  |  |
| .setp() | модифицирование параметров настройки отображения графика  plt.setp( color='red', linewidth=1) |
| .scale() | масштаб по оси x или y  'linear' - линейный  'log' - логарифмический  'symlog' - симметричный логарифмический  'logit' - логит масштаб  plt.yscale(value="log")  plt.yscale('symlog', linthreshy=0.01)  plt.yscale('logit') |
| .subplot() | размещение графиков на разных полях |
| .xlabel() | наименование оси x  labelpad - расстояние между осью и подписью,  число или None (умолч)  fontsize или size - размер шрифта (число или  значение из списка: {‘xx-small’, ‘x-small’,  ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx-large’}  fontstyle - стиль шрофта, значение из списка:  {‘normal’, ‘italic’, ‘oblique’}  fontweight - толщина шрифта, число в диапазоне  от 0 до 1000 либо значение из списка:  {‘ultralight’, ‘light’, ‘normal’, ‘regular’, ‘book’,  ‘medium’, ‘roman’, ‘semibold’, ‘demibold’,  ‘demi’, ‘bold’, ‘heavy’, ‘extra bold’, ‘black’}.  color - цвет шрифта  plt.xlabel('Day', fontsize=15, color='blue') |
| .ylabel() | наименование оси y (см. xlabel) |
| .title() | заголовок графика (см. xlabel)  plt.title('Chart price', fontsize=17, loc='left')  loc - положение заголовка ('center', 'left', 'right')  fontsize - размер текста  fontweight - насыщенность текста  color - цвет текста (название цвета, или код цвета в  формате HEX (например, #808080).  family - семейство шрифтов  title\_font = {  "fontsize": 16,  "fontweight": "bold",  "color": "#808080",  "family": "serif"}  plt.title("График функции", fontdict=title\_font)  # параметры шрифтов можно задавать отдельным словарем) |
| .text() | размещение текста на поле графика  (координата по x, координата по y, сам текст)  plt.text(1, 1, 'hello') |
| .legend() | легенда графика (цвет линии и название графика)  fontsize - размера шрифта надписи легенды int или  float или {'xx-small', 'x-small', 'small', 'medium',  'large', 'x-large', 'xx-large'}  frameon- отображение рамки легенды, bool  framealpha - прозрачность легенды, None или float  facecolor - цвет заливки  edgecolor - цвет рамки  title - текст заголовка  title\_fontsize - Размер шрифта, str  plt.plot(x, y, label='steel price') # название графика  plt.legend() # вывести легенду  plt.legend(labels=["x^2", "x^3"])  # легенда по порядку рисования графиков  plt.legend(loc="lower right", frameon=False)  # местоположение легенды на изображении, наличие рамки  legend\_font = {  "size": 7,  "family": "serif"}  plt.legend(loc="lower right", prop=legend\_font)  # параметры форматирования  plt.setp(legend.get\_texts(), color="DarkBlue")  # цвет текста легенды |
| .axis() | желаемые границы по оси x и границы по оси y  plt.axis([-5, 5, -8, 8]) |
| .grid() | отображение сетки на графике  plt.grid(True)  plt.grid()  plt.grid(axis="x", color="lightgrey") |
| fill\_between() | заливка области между графиком и осью |
| .show() | вывести изображение  plt.show() |
| .savefig | сохранение графика в файл  plt.savefig("bm\_price", fmt="png") |
|  |  |
| Объектно-ориентированный подход | |
| .figure() | область, на которой отображаются все остальные объекты  num : номер фигуры, целое число или строка, если  не определено, будет создана новая фигура,  и номер фигуры будет увеличен. Если num -  строка - заголовок окна будет установлен на  этот рисунок num.  figsize : ширина, высота в дюймах.  dpi : целое число, разрешение фигуры (умолч 100).  facecolor: цвет фона.  edgecolor: цвет границы.  frameon : рамка рисунка, bool (умолч True)  FigureClass : подкласс Figure (пользовательский)  clear : очищение, bool (умолч: False)  fig = plt.figure()  fig.set(facecolor = 'green')  plt.figure(figsize=(20,10))  fig.set\_size\_inches(18.5, 10.5, forward=True)  fig.savefig('test2png.png', dpi=100)  figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k') |
| .axis() | графики, располагающиеся на фигуре  чаще всего задаются с помощью .subplots()  fig.add\_axes  ax = fig.add\_subplot(111)  fig, ax = plt.subplots()  ax.set(facecolor = 'red')  ax.set(facecolor = 'red',  xlim = [-10, 10],  ylim = [-2, 2],  title = 'Основы анатомии matplotlib',  xlabel = 'ось абцис (XAxis)',  ylabel = 'ось ординат (YAxis)')  ax3.plot(x, y3)  ax3.set\_title("График sin(x)")  ax\_1 = fig.add\_subplot(2, 2, 1)  ax\_1.set(title = 'ax\_1', xticks=[], yticks=[]) |
| .subplots() | определяет количество областей axis (количество и расположение графиков на фигуре)  subplot(nrows, ncols, index)  nrows: Количество строк.  ncols: Количество столбцов.  index: Местоположение элемента.  ax = fig.add\_subplot(111)  # размещение axis на figure в данном местоположении: 1 строка, 1 столбец, 1 ячейка  fig, ax = plt.subplots()  fig, ax = plt.subplots(nrows=1, ncols=2)  ax[0].plot(x, y)  ax[1].plot(x, y2)  # в таком случае ax - одномерный массив numpy  fig, ax = plt.subplots(nrows=2, ncols=2)  ax1, ax2, ax3, ax4 = ax.flatten()  ax1.plot(x, y)  ax2.plot(x, y2)  ax3.plot(x, y3)  ax4.plot(x, y4)  # в таком случае ax - двухмерный массив numpy, выравниваем многомерный массив до одномерного  fig.subplots\_adjust(wspace=0.3, hspace=0.3)  # расстояние между axis (графиками) по ширине и высоте |
|  |  |
|  |  |
| **scikit-learn**  **#scikit-learn**  **#sk** |  | библиотека для решения задач классического машинного обучения  pip3 install scikit-learn  <https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics> |
| datasets | from sklearn import datasets  from sklearn.datasets import load\_boston  встроенные наборы данных, например, iris и digits для классификации, boston house prices dataset для регрессионного анализа и т.д., и  генераторы выборок для различных моделей  boston = load\_boston()  iris = datasets.load\_iris()  iris.keys() # содержание, поля  iris.data # данные X  iris.data.shape # размерность X  iris.target # целевая переменная y  iris.feature\_names # название признаков X  iris.target\_names # название целевых классов y  iris.DESCR # описание датасета  iris.filename # адрес файла  X, y = iris.data, iris.target  Полный список наборов данных:   1. load\_boston()   Набор данных о ценах на жилье в Бостоне  506 регрессия   1. load\_breast\_cancer()   Рак молочной железы Висконсин  569 классификация (двоичная)   1. load\_diabetes()   Набор данных диабета  442 регрессия   1. load\_digits(n\_class)   Цифры данных  1797 классификация   1. load\_iris()   Набор данных Iris  150 классификация (многоклассовый)   1. load\_linnerud()   Набор данных Linnerud  20 многомерная регрессия |
| preprocessing | from sklearn import preprocessing  from sklearn.preprocessing import StandardScaler  пакет, содержащий функции для изменения необработанных векторов в более подходящую форму для дальнейшего применения в моделях  **Scale**  стандартизация данных по любой оси  sklearn.preprocessing.scale(X, \*, axis=0, with\_mean=True, with\_std=True, copy=True)  X - матрица, данные для центр и масштабир.  axis int (по умолчанию 0)  ось, используемая для вычисления средних значений и стандартных отклонений. 0 - стандартизируйте каждую функцию независимо, 1 - стандартизируйте каждый образец.  with\_mean логический, по умолчанию True  True - центрируйте данные перед масштабированием.  with\_std логический, по умолчанию True  True - масштабируйте данные до единиц отклонения (стандартного отклонения единиц).  copy логическое, необяз, по умолчанию True  установите значение False, чтобы выполнить нормализацию строки на месте и избежать копирования (если вход уже является массивом numpy или матрицей scipy.sparse CSC и если ось равна 1).  # в массив numpy (по умолч)  train\_data\_scaled\_st = preprocessing.scale(train\_data)  # в DataFrame  train\_data\_scaled\_st = pd.DataFrame(preprocessing.scale(train\_data))  **StandardScaler**  стандартизация данных  Центрирование и масштабирование происходят независимо для каждой функции путем вычисления соответствующей статистики по выборкам в обучающем наборе. Затем среднее и стандартное отклонение сохраняются для использования в последующих данных.  StandardScaler не может гарантировать сбалансированные масштабы признаков при наличии выбросов.  приведение признаков к нормальному гауссовскому распределению в масштаб с диапазоном [-1 0 1].  После преобразования среднее значение превратится в 0, мин в -1, мах в 1.  NaN обрабатываются как пропущенные значения: не учитываются при подгонке и сохраняются при преобразовании.  sklearn.preprocessing.StandardScaler(\*, copy=True, with\_mean=True, with\_std=True)[source]  copy логическое, необяз, по умолч True  Если False, постарайтесь избежать копирования и вместо этого выполните масштабирование на месте. Это не гарантирует, что всегда будет работать на месте; например, если данные не являются массивом NumPy или CSR-матрицей scipy.sparse, копия все равно может быть возвращена.  with\_mean логический, по умолчанию True  True - центрируйте данные перед масштабированием. Это не работает (и вызовет исключение) при попытке использовать разреженные матрицы, поскольку их центрирование влечет за собой построение плотной матрицы, которая в обычных случаях использования, вероятно, будет слишком большой, чтобы поместиться в памяти.  with\_std логический, по умолчанию True  True, масштабируйте данные до единиц отклонения (стандартного отклонения единиц).  Атрибуты(только в numpy, df.mean\_):  scale\_ ndarray или None, shape (n\_features,)  Относительное масштабирование данных по каждой функции. Это рассчитывается с использованием np.sqrt(var\_). Равно Noneкогда with\_std=False.  mean\_ ndarray или None, shape (n\_features,)  Среднее значение для каждой функции в обучающем наборе. Равно None когда with\_mean=False.  var\_ ndarray или None, shape (n\_features,)  Дисперсия для каждой функции в обучающем наборе. Используется для вычислений scale\_. Равно Noneкогда with\_std=False.  n\_samples\_seen\_ int или массив, форма (n\_features,)  Количество выборок, обработанных оценщиком для каждой функции. Если нет пропущенных образцов, n\_samples\_seenбудет целое число, в противном случае это будет массив. Будет сброшен при новых вызовах, чтобы соответствовать, но увеличивается между partial\_fitвызовами.  # в массив numpy  scaler = StandardScaler()  train\_data\_scaled\_st = scaler.fit(train\_data) train\_data\_scaled\_st.scale\_  # в DataFrame  scaler = StandardScaler()  train\_data\_scaled\_st = pd.DataFrame(scaler.fit\_transform(train\_data), columns=train\_data.columns)  **MinMaxScaler** или **minmax\_scale**  масштабирует и преобразует каждую функцию индивидуально так, чтобы она находилась в заданном диапазоне обучающего набора, например, между нулем и единицей.  специально разработан для масштабирования разреженных данных и являются рекомендуемым способом сделать это  NaN обрабатываются как пропущенные значения: не учитываются при подгонке и сохраняются при преобразовании.  Очень чувствителен к наличию выбросов.  sklearn.preprocessing.MinMaxScaler( feature\_range = (0 , 1) , \* , copy = True )  **MaxAbsScaler** или **maxabs\_scale**  Этот оценщик масштабирует и переводит каждую функцию индивидуально, так что максимальное абсолютное значение каждой функции в обучающем наборе будет 1,0. Он не смещает / центрирует данные и, следовательно, не уничтожает разреженность.  специально разработан для масштабирования разреженных данных и являются рекомендуемым способом сделать это  NaN обрабатываются как пропущенные значения: не учитываются при подгонке и сохраняются при преобразовании.  **RobustScaler и robust\_scale**  Масштабирует функции, используя статистику, устойчивую к выбросам.  Этот скейлер удаляет медианное значение и масштабирует данные в соответствии с квантильным диапазоном (по умолчанию IQR: межквартильный диапазон). IQR - это диапазон между 1-м квартилем (25-й квантиль) и 3-м квартилем (75-й квантиль).  Центрирование и масштабирование происходят независимо для каждой функции путем вычисления соответствующей статистики по выборкам в обучающем наборе. Затем медиана и межквартильный размах сохраняются для использования в последующих данных с использованием этого transform метода.  **QuantileTransformer и quantile\_transform**  Этот метод преобразует функции в соответствии с равномерным или нормальным распределением. Следовательно, для данного объекта это преобразование имеет тенденцию распространять наиболее частые значения. Это также снижает влияние (маргинальных) выбросов: следовательно, это надежная схема предварительной обработки.  Преобразование применяется к каждому объекту независимо. Сначала оценка совокупной функции распределения признака используется для сопоставления исходных значений с равномерным распределением. Полученные значения затем отображаются в желаемое выходное распределение с использованием связанной функции квантиля. Значения функций новых / невидимых данных, которые попадают ниже или выше установленного диапазона, будут сопоставлены с границами выходного распределения. Обратите внимание, что это преобразование нелинейное. Это может исказить линейные корреляции между переменными, измеренными в одном масштабе, но делает переменные, измеренные в разных масштабах, более сопоставимыми.  имеет дополнительный output\_distributionпараметр, позволяющий согласовывать гауссово распределение вместо равномерного распределения  в отличие от RobustScaler, QuantileTransformerтакже автоматически сбрасывает любые выбросы, устанавливая их на заранее определенные границы диапазона (0 и 1)  NaN обрабатываются как пропущенные значения: не учитываются при подгонке и сохраняются при преобразовании.  **PowerTransformer**  Преобразования мощности - это семейство параметрических монотонных преобразований, которые применяются для придания данным более гауссовского типа. Это полезно для моделирования проблем, связанных с гетероскедастичностью (непостоянной дисперсией) или других ситуаций, когда желательна нормальность.  В настоящее время PowerTransformer поддерживает преобразование Бокса-Кокса и преобразование Йео-Джонсона. Оптимальный параметр для стабилизации дисперсии и минимизации асимметрии оценивается через максимальное правдоподобие.  Box-Cox требует, чтобы входные данные были строго положительными, в то время как Йео-Джонсон поддерживает как положительные, так и отрицательные данные.  По умолчанию к преобразованным данным применяется нормализация с нулевым средним и единичной дисперсией.  NaN обрабатываются как пропущенные значения: игнорируются fitи сохраняются transform.  **Normalizer и normalize**  Нормализовать образцы индивидуально до единичной нормы.  Каждая выборка (то есть каждая строка матрицы данных) с по крайней мере одним ненулевым компонентом масштабируется независимо от других выборок, так что ее норма (l1, l2 или inf) равна единице.  Масштабирование входных данных к нормам единиц - обычная операция, например, для классификации текста или кластеризации.  **OrdinalEncoder**  Кодируйте категориальные признаки как целочисленный массив.  Вход в этот преобразователь должен быть подобным массиву целых чисел или строк, обозначающих значения, принимаемые категориальными (дискретными) функциями. Функции преобразуются в порядковые целые числа. В результате получается один столбец целых чисел (от 0 до n\_categories - 1) для каждой функции.  **KBinsDiscretizer**  дискретизация, разбивает непрерывные данные на интервалы.  **Binarizer**  бинаризация  определение пороговых значений числовых функций для получения логических значений. Это может быть полезно для последующих вероятностных оценок, которые предполагают, что входные данные распределяются согласно многомерному распределению Бернулли.  **PolynomialFeatures**  создание полиномиальных признаков  усложняет модель, учитывая нелинейные особенности входных данных.  **FunctionTransformer**  Создает преобразователь из произвольного вызываемого объекта. |
| **.model\_selection**  **.train\_test\_split** | from sklearn.model\_selection import train\_test\_split  разбиение данных на тренировочную и тестовую выборки  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.30, random\_state=42)  test\_size указывает долю данных выделяемую под  тест (число из отрезка [0,1)), или число  объектов в тестовой выборке (число >= 1)  train\_size указывает долю данных выделяемую под  обучение (число из отрезка [0,1)), или  число объектов в обучающей выборке  (число >= 1)  random\_state начальное число, используемое  генератором случайных чисел, по умолч.  используется np.random  shuffle перемешивание данных перед разделением  True или False  stratify перемешивание данных стратифицированно |
| **model\_selection.cross\_val\_score** | from sklearn.model\_selection import cross\_val\_score  кросс-валидация  LinReg\_Ridge = linear\_model.Ridge()  print(cross\_val\_score(LinReg\_Ridge, x\_train, y\_train, cv=5)) |
| **.metrics** | **mean\_squared\_error**  from sklearn.metrics import mean\_squared\_error as mse  среднеквадратичное отклонение (mse) и корень из неё (root mean\_squared\_error)  используется на моделях регресии  используется, если есть выбросы, и они для вас важны (важны при использовании модели)  def calc\_mse(y, y\_pred):  err = np.mean((y - y\_pred)\*\*2)  return err  только для mse, точный рассчет формулы  W = np.linalg.inv(X @ X.T) @ X @ y # array([47.23214286, 3.91071429])  где 47.23 - это k, 3.91 - это b  в остальных случаях для расчета формулы используют метод градиентного спуска  mse = mean\_squared\_error(y\_true, y\_pred)  rmse = mse(y\_test, y\_pred, squared = False)  **mean\_absolute\_error**  from sklearn.metrics import mean\_absolute\_error as mae  средняя абсолютная ошибка (не так сильно реагирует на выбросы в данных)  используется на моделях регресии  используеся, если выбросы не важны, модель должна работать хорошо на большинстве объектов  def calc\_mae(y, y\_pred):  err = np.mean(np.abs(y - y\_pred))  return erry  mae = mean\_absolute\_error(y\_true, y\_pred)  **median\_absolute\_error**  from sklearn.metrics import median\_absolute\_error as med\_ae  средняя абсолютная ошибка по медиане  используется на моделях регресии  устойчивa к выбросам  Убыток рассчитывается путем взятия медианы всех абсолютных различий между целью и прогнозом.  med\_ae (y\_true, y\_pred)  **r2\_score**  from sklearn.metrics import r2\_score as r2  # коэффициент детерминации (доля дисперсии в зависимой переменной, наилучшее значение 1, м.б. отрицательной)  используется на моделях регресии  Единица минус отношение средней квадратичной ошибки модели к средней квадратичной ошибке среднего значения тестовой выборки.  Показывает, насколько условная дисперсия модели отличается от дисперсии реальных значений Y. Если этот коэффициент близок к 1, то условная дисперсия модели достаточно мала и весьма вероятно, что модель неплохо описывает данные. Если же коэффициент R-квадрат сильно меньше, например, меньше 0.5, то, с большой долей уверенности модель не отражает реальное положение вещей.  Обычно принимает значения от 0 до 1, чем ближе значение коэффициента к 1, тем сильнее зависимость, значение <0 свидетельствует о том, что модель работает хуже, чем простое случайное предсказание.  Однако, у статистики R-квадрат есть один серьезный недостаток: при увеличении числа предикторов эта статистика может только возрастать. Поэтому, может показаться, что модель с большим количеством предикторов лучше, чем модель с меньшим, даже если все новые предикторы никак не влияют на зависимую переменную.  r2 = r2\_score(y\_true, y\_pred)  **explained\_variance\_score**  from sklearn.metrics import explained\_variance\_score  оценка объясненной дисперсии (похожа на r2)  Наилучшая возможная оценка - 1.0, более низкие значения - хуже.  **max\_error**  from sklearn.metrics import max\_error  максимальная остаточная ошибка  используется на моделях регресии  показатель, который фиксирует худшую ошибку между предсказанным значением и истинным значением. В идеально подобранной модели регрессии с одним выходом он max\_error будет находиться 0  **mean\_squared\_log\_error**  from sklearn.metrics import mean\_squared\_log\_error  ожидаемое значение квадрата логарифмической (квадратичной) ошибки или потери  используется на моделях регресии  Эту метрику лучше всего использовать, когда цели имеют экспоненциальный рост, например, численность населения. Эта метрика штрафует заниженную оценку больше, чем завышенную оценку.  mean\_squared\_log\_error(y\_true, y\_pred)  **mean\_tweedie\_deviance**  from sklearn.metrics import mean\_tweedie\_deviance  используется на моделях регресии  power=0 это эквивалентно mean\_squared\_error.  отклонение масштабируется квадратично  power=1 это эквивалентно mean\_poisson\_deviance.  отклонение масштабируется линейно  power=2 это эквивалентно mean\_gamma\_deviance.  одновременно масштабируется y\_true и y\_pred  mean\_tweedie\_deviance (y\_true, y\_pred), power=1)  **roc\_auc\_score**  from sklearn.metrics import roc\_auc\_score as  площадь под кривой ошибок  используется на моделях бинарной классификации  roc\_auc\_score(y\_true, y\_scores) |
| **linear\_model** | модели линейной регрессии  from sklearn import linear\_model  from sklearn.linear\_model import LinearRegression  **LinearRegression**  from sklearn.linear\_model import LinearRegression  обычная линейная регрессия методом наименьших квадратов  модель линейной регрессии  y=a+bx (простая парная линейная регрессия)  𝑓(𝑥)=𝑤0+𝑤1⋅𝑥1+⋯+𝑤𝑛⋅𝑥𝑛.  from sklearn import linear\_model  from sklearn.linear\_model import LinearRegression  model\_reg = linear\_model.LinearRegression()  # создание модели линейной регресии  lr = LinearRegression()  # обучение модели на тренировочных данных  .fit  lr.fit(х\_train, y\_train)  # получение предсказанных значений  .predict  y\_pred = lr.predict(x\_test)  **Ridge**  from sklearn.linear\_model import Ridge  модель линейной регрессии наименьших квадратов с L2 регуляцией (гребневая регрессия)  Гребневая регрессия или ридж-регрессия (англ. ridge regression) — один из методов понижения размерности. Применяется для борьбы с избыточностью данных, когда независимые переменные коррелируют друг с другом, вследствие чего проявляется неустойчивость оценок коэффициентов многомерной линейной регрессии.  **RidgeCV**  from sklearn.linear\_model import RidgeCV  Ридж регрессия со встроенной перекрестной проверкой |
|  | **SGDRegressor**  from sklearn.linear\_model import SGDRegressor  модель линейной регрессии подбирается путем минимизации регуляризованных эмпирических потерь с SGD (Стохастический градиентный спуск)  Суть градиентного спуска – минимизировать функцию, делая небольшие шаги в сторону наискорейшего убывания функции. |
|  |  |
|  | from sklearn.ensemble import RandomForestClassifier  # модель случайный лесной классификатор  from sklearn.model\_selection import GridSearchCV  # исчерпывающий поиск по указанным значениям параметров для оценщика.  model\_gs.best\_params\_ - параметры модели, которая дала наилучшие результаты |
|  |  |
|  |  |
| DecisionTree  Classifier | from sklearn.tree import DecisionTreeClassifier  модель дерево решений |
| RandomForest  Regressor | from sklearn.ensemble import RandomForestRegressor  модель случайный лес  предпочтительно использовать массивы numpy, а не DataFrame  model = RandomForestRegressor(n\_estimators=1000, max\_depth=12, random\_state=42)  n\_estimators - количество деревьев в лесу  (по умолчанию 100)  max\_depth - максимальная глубина дерева  (по умолчанию None)  random\_state - управляет случайностью начальной  загрузки выборок (если bootstrap=True),  и выборкой функций, учитывающихся  при поиске наилучшего разделения на  каждом узле (если max\_features <  n\_features) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **MongoDB**  **#** **MongoDB** |  | pip3 install pymongo  from pymongo import MongoClient  client = MongoClient('localhost', 27017)  # создание клиента для подключения к серверу  db = client['db\_pars']  # подключение к базе данных (или её создание)  collection = db.prof # создание коллекции в БД  $lt — меньше,  $lte — меньше или равно,  $gt — больше,  $gte — больше или равно,  $ne — не равно. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Создание собственной функции.

|  |  |
| --- | --- |
| **Создание собственной функции** | def <имя функции> (<параметры>):  <тело функции>  <retunn …> #функция с возвратом  <имя функции> (<параметры>) #вызов функции  \* - args – передача любого количества параметров по порядку  kwargs - передача любого количества параметров по имени  def print\_sep (): #функция без параметров  print (':' \* 20)  print\_sep ()  ::::::::::::::::::::  def print\_sep (sep, sep\_len): #функция ничего не возвращает  print (sep \* sep\_len) #только печатает  a = print\_sep ('#', 30)  print\_sep ('#', 30)  print\_sep ('&', 20)  print (a)  ##############################  &&&&&&&&&&&&&&&&&&&&  None  def greeting (who, say):  print (say, who)  greeting (say='Hi',who ='Leo') #присвоение параметров по имени  Hi Leo  def greeting (who, say='Hello'): #параметр say по умолчанию  print (say, who)  greeting ('Leo')  Hello Leo  def greeting (say, \*who): #\* передача любого кол-ва параметров  print (say, who) #но возвращает картеж, \*who = \*args  greeting ('Hi', 'Leo', 'Max', 'Kate')  Hi ('Leo', 'Max', 'Kate')  def get\_person (\*\*kwargs): #\*\*любое кол-во параметров по имени  for k, v in kwargs.items():  print (k, v)  get\_person (name='Leo', age=20, has\_car=True)  name Leo  age 20  has\_car True  def print\_sep (sep, sep\_len): #функция с возвратом  return sep \* sep\_len  print (print\_sep ('#', 34))  sep\_1 = print\_sep ('-', 10)  print ('Name {} Surname {}'.format (sep\_1,sep\_1))  ##################################  Name ---------- Surname ----------  global\_a = 1 #изменение глобальной переменной в функции  def variable ():  global global\_a  global\_a = 5  print (global\_a)  variable ()  print (global\_a)  1  5  def some\_f():  return 10  a = some\_f() #переменная a становится результатом функции  b = some\_f #переменная b становится самой функцией  print (a)  print (b)  print (type (b))  c = b()  print (c)  10  <function some\_f at 0x0000022D3EAA8598> #область памяти  <class 'function'> #тип переменной b  10  def my\_filter (numbers): #функция выводит нечетные числа  result = []  for number in numbers:  if number % 2 == 0:  result.append(number)  return result  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  print (my\_filter(numbers))  [2, 4, 6, 8]  def my\_filter (numbers, function):  result = []  for number in numbers:  if function(number): #если возвр True (number % 2 == 0)  result.append(number)  return result  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  def is\_noteven (number):  return number % 2 != 0  def big\_4 (number):  return number > 4  print (my\_filter(numbers, is\_even))  print (my\_filter(numbers, is\_noteven))  print (my\_filter(numbers, big\_4))  [2, 4, 6, 8]  [1, 3, 5, 7, 9]  [5, 6, 7, 8, 9]  lambda <входные параметры>: <результат>  def my\_filter (numbers, function):  result = []  for number in numbers:  if function(number):  result.append(number)  return result  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]  print (my\_filter(numbers, lambda number: number % 2 == 0))  print (my\_filter(numbers, lambda number: number % 2 != 0))  print (my\_filter(numbers, lambda number: number > 4))  [2, 4, 6, 8]  [1, 3, 5, 7, 9]  [5, 6, 7, 8, 9]  Создание файла:  def create\_file (name, text=None):  with open (name, 'w', encoding = 'utf-8') as f:  if text:  f.write(text)  Создание папки:  def create\_folder (name):  try:  os.mkdir(name)  except FileExistsError:  print ('такая папка уже есть')  Список папок и файлов:  def get\_list(folders\_only=False):  result = os.listdir()  if folders\_only:  result=[f for f in result if os.path.isdir(f)]  print(result)  Удаление папок и файлов:  def delete\_filedir(name):  if os.path.isdir(name):  os.rmdir(name)  else:  os.remove(name)  Копирование папок и файлов:  def copy\_filedir(name, new\_name):  if os.path.isdir(name):  try:  shutil.copytree(name, new\_name)  except FileExistsError:  print('такая папка уже есть')  else:  shutil.copy(name, new\_name)  Запись информации о работе менеджера в файл:  def save\_info(message):  current\_time = datetime.datetime.now()  result = f'{current\_time} - {message}'  print(result)  with open('log.txt', 'a', encoding = 'utf-8') as f:  f.write(result + '\n') |

# Создание собственного модуля и пакета.

|  |  |
| --- | --- |
| **Создание собственного модуля** | Создаём файл с расширением .py и пишем в нем свой модуль (программу)  Импорт собственного модуля:   1. import Card   имя модуля без расширения и без пути, если он находится в той же папке, что и программа   1. import Lesson.Card   имя модуля без расширения и путь, если он находится в подпапке той же папки, что и программа   1. с помощью добавления пути поиска модулей в переменную sys.path   При любом варианте импорта скрипты будут выполняться сразу, но внутри модуля можно выполняемые действия вписать в условие  if \_\_name\_\_ == '\_\_main\_\_':  и тогда данные скрипты будут выполняться не при импорте модуля, а при обращении к нему |
| **Создание собственного пакета** | Пакет – это каталог, включающий в себя другие пакеты и модули.  Пакет содержит внутри себя файл \_\_init\_\_.py  Создается папка (<Пакет1>), которая содержит внутри себя пустой файл \_\_init\_\_.py  Пакет формирует пространство имен, организует работу с модулями с указанием уровня вложенности, уровни отделяются точкой <пакет1.пакет2.модуль>  Вложенность пакетов может быть любой (пакет в пакете ...)  Импорт другого модуля из того же пакета  import .модуль  Импорт другого модуля из другого пакета  import пакет.модуль |

# Тернальный операнд.

|  |  |
| --- | --- |
|  | - это операция, возвращающая свой первый или третий операнд в зависимости от значения логического выражения, заданного вторым операндом.  результат\_1 если выражение истинно иначе результат\_2  russian = False  print ('Привет!' if russian else 'Hello!')  password = input('Введите пароль: ')  print ('Войти' if password == 'secret' else 'Вход воспрещен')  word ='мышка'  result = []  for i in range(len(word)):  if i%2 != 0:  letter = word[i].lower()  else:  letter = word[i].upper()  result.append(letter)  result = ''.join(result)  print(result)  МыШкА  word ='мышка'  result = []  for i in range(len(word)):  letter = word[i].lower() if i%2 != 0 else word[i].upper()  result.append(letter)  result = ''.join(result)  print(result)  МыШкА |

# Ламбда, генератор, тернарный оператор.

|  |  |
| --- | --- |
| **классический метод** | numbers = [1, 3, 5, -3, -9, 0, -2]  result = []  for number in numbers:  if number > 0:  result.append(number)  print (result) # [1, 3, 5] |
| **ламбда метод**  #lambda | создание анонимной функции без имени  lambda x: x + 1 # 1 аргумент  (lambda x: x + 1)(2) # 1 аргумент  lambda x, y: x + y # несколько агрументов  a = (lambda x, y: x + y)(3,4) # несколько агрументов  a = lambda x: x if x % 2 == 0 else 0 # с условием  a = lambda x: x + 1 # создание функции, к-рую надо вызывать  print(a(3)) # 4 вызов функции с агрументом 3  print(type(a)) # <class 'function'>  a = (lambda x: x + 1)(2) # результат функции с аргументом 2  print(a) # 3 a-это результат функции  print(type(a)) # <class 'int'>  a = lambda x, y: x + y  print(a(3, 4)) # 7  a = (lambda x, y: x + y)(3,4)  print(a) # 7  a = lambda x: x if x % 2 == 0 else 0  print (a(5)) # 0  a = lambda x, y: x + y  b = [1,2,3,4,5,6,7]  for i in range(len(b) - 1):  print(a(b[i], b[i+1]))  a = list(map(lambda x: x \*\* 2 + 10, [2,3,4,5])) # к каждому элем  print(a) # [14, 19, 26, 35]  a = lambda x, y: x + y # для 2 списков  print(list(map(a,[2,3,4,5], [6,7,8,9]))) # [8, 10, 12, 14]  a = list(filter(lambda x: x % 2 == 0, [2,3,4,5])) # элем соотв условию  print(a) # [2, 4]  a = lambda x: x % 2 == 0  print(list(filter(a,[2,3,4,5])))  from functools import reduce # аккумулированное значение  sum = reduce(lambda x,y: x + y, [1,2,3,4,5]) # сумма элементов  print (sum) |
| **генератор**  #генератор | [number for number in numbers if number > 0]  a = [<выражение><последовательность>]  a = [i\*\*2 for i in range(1,11)]  a = [i \* 3 for i in [2,3,4]] # [6, 9, 12]  a = [i \* 3 for i in [2,3,4] if i != 3] # [6, 12]  a = [i \* m for i in [2,3,4] if i != 3 for m in [2,3,4] if m != 9]  print(a) # [4, 6, 8, 8, 12, 16]  a = [i for i in range(0,4)] # [0, 1, 2, 3]  a = [i for i in range(0,4) if i % 2 ==0] # [0, 2]  a = [i\*\*2 for i in range(1,3)] # [1, 4]  a = {i:i\*\*2 for i in range(10,13)} # {10: 100, 11: 121, 12: 144}  pairs = [(1, 'a'), (2, 'b'), (3, 'c')]  result = {pair[0]:pair[1] for pair in pairs}  print (result) # {1: 'a', 2: 'b', 3: 'c'}  import random  result = [random.randint(1,100) for i in range(10)]  print (result) # [49, 80, 70, 86, 83, 37, 70, 94, 92, 23]  names = ['Ann', 'Ron', 'Alice', 'Leo', 'Max']  names\_a = [name for name in names if name.startswith('A')]  print (names\_a) # ['Ann', 'Alice'] |
| **тернарный оператор**  # тернарный оператор | - это операция, возвращающая свой первый или третий операнд в зависимости от значения логического выражения, заданного вторым операндом.  результат\_1 если выражение истинно иначе результат\_2  password = input('Введите пароль: ')  print ('Войти' if password == 'secret' else 'Вход воспрещен')  i = 6  a = i if i % 2 == 0 else 0 # 6  b, c = 5, 6  a = b - c if b > c else c - b # 1  russian = False  print ('Привет!' if russian else 'Hello!')  word ='мышка'  result = []  for i in range(len(word)):  letter = word[i].lower() if i%2 != 0 else word[i].upper()  result.append(letter)  result = ''.join(result)  print(result)  МыШкА |
| **генератор + тернарный оператор** |  |

# Объектно-ориентированное программирование.

|  |  |  |
| --- | --- | --- |
| **class** |  | создание класса  self – вызывает сам объект с его свойствами  \_step – 1 уровень инкапсуляции (говорит о том что  не следует менять это свойство у экземпляра  класса)  \_\_step – 2 уровень инкапсуляции (свойство экземпляра  класса, названное таким образом изменить  нельзя (оно меняется только в самом  описании класса или обходными путями))  <https://habr.com/ru/post/186608/> - статья по магическим методам в питоне  class Bird: #создали класс  name = 'Unknown' #имя имя экземпляров по умолч  owl = Bird() #объект класса  print (owl.name) #имя объекта класса  owl.name = 'Owl' #переименование  print (owl.name)  Unknown  Owl  class Bird:  name = 'Unknown'  def call(self): # метод  this\_name=self.name  print('My name is', this\_name)  owl = Bird()  owl.name = 'Owl'  owl.call()  My name is Owl  class Bird:  name = 'Unknown'  distance = 0  speed = 100  def call(self):  this\_name=self.name  print('My name is', this\_name)  def run(self, dist=False):  if dist == False:  dist = self.speed  self.distance = self.distance + dist  owl = Bird()  owl.name = 'Owl'  owl.speed = 150  owl.call()  owl.run(300)  eagle = Bird()  eagle.name = 'Eagle'  eagle.speed = 400  eagle.run()  print(owl.distance)  print(eagle.distance)  My name is Owl  300  400 |
| **\_\_init\_\_** | конструктор класса  class Bird:  name = 'Unknown'  distance = 0  speed = 100  def call(self):  this\_name=self.name  print('My name is', this\_name)  def run(self, dist=False):  if dist == False:  dist = self.speed  self.distance = self.distance + dist  def \_\_init\_\_(self, name, speed, steps):  self.name = name  self.speed = speed  for n in range(steps):  self.run()  owl = Bird('Owl', 150, 2)  eagle = Bird('Eagle', 300, 4)  owl.call()  print(owl.distance)  eagle.call()  print(eagle.distance)  My name is Owl  300  My name is Eagle  1200  class Word:  '''text, part '''  def \_\_init\_\_(self, text, part):  self.text = text  self.part = part  class Sentence:  '''text, part '''  def \_\_init\_\_(self,words):  self.content = words  def show(self):  s=''  for n in self.content:  s += words[n].text + ' '  return s  def show\_parts(self):  li = []  for n in self.content:  if not words[n].part in li:  li.append(words[n].part)  return li  words = [['красная', 'прилагательное'],  ['шапочка', 'существительное'],  ['пошла', 'глагол'],  ['в', 'предлог'],  ['лес', 'существительное'],  ['за', 'предлог'],  ['грибами', 'существительное']]  for n in range(len(words)):  words[n] = Word(words[n][0], words[n][1])  #print(words)  #print(words[0].text)  sentence = Sentence([0,1,2,3,4,5,6])  #print(sentence.content)  print(sentence.show())  print(sentence.show\_parts())  красная шапочка пошла в лес за грибами  ['прилагательное', 'существительное', 'глагол', 'предлог'] |
| **\_\_str\_\_** | меняет название экземпляра  class Bird():  def \_\_init\_\_(self, v1, v2):  self.field\_1 = v1  self.field\_2 = v2  def \_\_str\_\_(self):  return str(self.field\_1)  owl = Bird(10, 20)  print(owl)  10  вместо  <\_\_main\_\_.Bird object at 0x000001B1F71A2A58> |
| **\_\_gt\_\_** | метод, отвечающий за то, как будет себя вести объект при операции '>'  class Bird():  def \_\_init\_\_(self, v1, v2):  self.field\_1 = v1  self.field\_2 = v2  def \_\_str\_\_(self):  return str(self.field\_1)  def foo(self, other):  return self.field\_1 > other.field\_1  owl = Bird(10, 20)  eagle = Bird(20,15)  print(owl.foo(eagle))  class Bird():  def \_\_init\_\_(self, v1, v2):  self.field\_1 = v1  self.field\_2 = v2  def \_\_str\_\_(self):  return str(self.field\_1)  def \_\_gt\_\_(self, other):  return self.field\_1 > other.field\_1  def \_\_lt\_\_(self, other):  return self.field\_1 < other.field\_1  owl = Bird(10, 20)  eagle = Bird(20,15)  print(owl > eagle)  False |
| **\_\_lt\_\_** | метод '<' |
| **\_\_ne\_\_** | метод '!=' |  |
| **\_\_le\_\_** | метод '<=' |  |
| **\_\_ge\_\_** | метод '>=' |  |
| **\_\_eq\_\_** | метод '==' |  |
| **\_\_cmp\_\_** | метод, определяющий поведение всех операоров сравнения (==, >=, <= b и т.д.) |  |
|  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |  |
|  |  |  |

python.org – официальный сайт, установка последней версии интерпретатора python.

версия интерпретатора python – в консоли cmd набрать команду 'python --version' или 'python3 --version'.

редакторы питона: Pycharm IDE, Sublime Text, Atom, Notepad++, Vim.

пути поиска модулей указаны в переменной sys.path

numpy – библиотека для работы с матрицами

html:

Head (заголовок вкладки, не всё видимо пользователем): meta, title (название страницы)

Body: article, p, h1, a, img, br(перенос строки)

<https://tilda.education/courses/landing-page/design-landing-page/> принципы дизайна

<https://color.adobe.com/ru/create/color-wheel> подбор цветовой гаммы для веб-дизайна

Цвета matplotlib:



import seaborn as sns

plt.figure(figsize=(16,10), dpi= 80)

sns.kdeplot(test\_scaled\_st['age'], shade=True, color="g", label="Cyl=4", alpha=.7)

plt.show()

import matplotlib.pyplot as plt

import numpy as np

fig1 = plt.figure() # создание фигуры (поллотно, холст)

ax = fig1.add\_subplot() # создание области графика (axis и subplot это синонимы)

fig1.set(facecolor='greenyellow') # присвоить фигуре цвет

ax.set(facecolor='yellow') # присвоить области графика цвет

x = np.linspace(0, 30, 10) # создвние массива от 0 до 30, 10 элементов равноудаленных друг от друга

y = x

ax = plt.plot (x, y, color = 'red', linestyle = '--', label='legend1') # создание линейного графика

plt.legend() # отразить подпись

plt.grid() # добавить сетку

plt.title('График 1') # название графика

plt.xlabel('Ось x') # подпись оси x

plt.ylabel('Ось y') # подпись оси y

fig2, axis = plt.subplots(2,3) # создание несколькох областей для графиков (строки, столбцы)