
**Федеральное государственное образовательное бюджетное учреждение высшего образования**
**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ**
**ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

(Финансовый университет)

Департамент анализа данных, принятия решений и финансовых технологий

**И.В. Миронова**

# Программирование на языке Python

Учебное пособие
по дисциплинам «Компьютерный практикум» и «Алгоритмы и структуры данных в языке Python»

для студентов, обучающихся по направлению подготовки
"Бизнес-информатика" и «Прикладная информатика» профиль
«Высокопроизводительные вычисления в цифровой экономике»
(очная и заочная формы обучения)

Москва, 2019


**Федеральное государственное образовательное бюджетное учреждение высшего образования**
**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ**
**ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

(Финансовый университет)

Департамент анализа данных, принятия решений и финансовых технологий

**И.В. Миронова**

# ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PYTHON. ЧАСТЬ 2

Учебное пособие
по дисциплинам «Компьютерный практикум» и «Алгоритмы и структуры данных в языке Python»

для студентов, обучающихся по направлению подготовки
"Бизнес-информатика" и «Прикладная информатика» профиль
«Высокопроизводительные вычисления в цифровой экономике»
(очная и заочная формы обучения)

*Рассмотрено и одобрено на заседании Департамента анализа данных, принятия решений и финансовых технологий*
*(протокол № 14 от «28» мая 2019 г.)*

Москва, 2019

УДК 004 (072)
ББК 32.973.2
М64

**Автор**: Миронова И.В., канд. физ.-мат. наук, доцент, доцент департамента анализа данных, принятия решений и финансовых технологий Финансового университета при Правительстве Российской Федерации

**Рецензент**: Утакаева И.Х., канд. физ.-мат. наук, доцент департамента анализа данных, принятия решений и финансовых технологий Финансового университета при Правительстве Российской Федерации

**Программирование на языке Python. Часть 2**: Учебное пособие по дисциплинам «Компьютерный практикум» и «Алгоритмы и структуры данных в языке Python» для студентов, обучающихся по направлениям подготовки "Бизнес-информатика" и «Прикладная информатика» профиль «Высокопроизводительные вычисления в цифровой экономике». - М.: Финансовый университет, департамент анализа данных, принятия решений и финансовых технологий, 2019. -51 с.

Дисциплины «Компьютерный практикум» и «Алгоритмы и структуры данных в языке Python» являются обязательными дисциплинами базовой части профессионального цикла ООП для соответствующих направлений. Изучение этих дисциплин нацелено на формирование у студентов практических навыков по использованию возможностей программирования для решения прикладных задач профессиональной деятельности. В учебном пособии рассматриваются темы: функции, модули, текстовые файлы и объектно-ориентированное программирование. По каждой теме приведены задания для самостоятельной работы.

УДК 004 (072)
ББК 32.973.2

**Учебное издание**

*Миронова Ирина Васильевна*

**Программирование на языке Python. Часть 2**

Учебное пособие
по дисциплинам «Компьютерный практикум» и
«Алгоритмы и структуры данных в языке Python»

Компьютерный набор, верстка И.В. Миронова
Формат 60x90/16. Гарнитура Times New Roman
Усл. п.л. 3,2. Изд. № - 2019.
Заказ № _____
Электронное издание

# Содержание

# 1. **Функции**

## *Создание и вызов функции*

***Функция*** – фрагмент программы, у которого есть имя. Для выполнения этого фрагмента в любом месте программы достаточно указать его имя.

Функции объявляются с помощью инструкции def:

**def** *Имя_Функции* ( [*Имена_Параметров*] ):

['''*Строка_Документирования*''']

*Тело_Функции*

[**return** *Значение*]

Как обычно, в квадратных скобках указаны элементы, которые могут отсутствовать.

Тело функции – набор инструкций, оформленный в виде блока, то есть с отступом. Если тело функции не содержит инструкций, то внутри необходимо поместить оператор **pass**.

Инструкция return может встречаться в произвольном месте функции, ее исполнение завершает работу функции и возвращает указанное значение в место вызова. Если функция не возвращает значения, то для завершения работы инструкция return используется без возвращаемого значения. Если инструкция return отсутствует, по умолчанию возвращается значение None. Если нужно вернуть несколько значений, то укажите их в инструкции return через запятую. В этом случае возвращается кортеж.

Строка документирования обычно содержит краткое описание функции. Получить этот текст в программе можно с помощью выражения *Имя_функции.__doc__*.

Функция вызывается с помощью конструкции:

*Имя_Функции* ( [*Значения_Параметров*] )

Определение функции должно предшествовать ее использованию.

Количество параметров в определении функции должно совпадать с количеством параметров при вызове. Если параметров у функции нет, то скобки все равно нужно указывать.

Передача параметров при вызове функции возможна:

- без указания имени параметров, при этом значения параметров передаются строго в той последовательности, как эти параметры были описаны в функции (позиционные параметры);
- с явным указанием имени параметров, при этом порядок перечисления параметров не имеет значения (по ключу или именованные аргументы).

Если при вызове функции часть параметров передается по ключу, а часть позиционным способом, то в команде вызова сначала указываются позиционные параметры, а затем те, что передаются по ключу.

*Примеры функций:*

```
def f1(x):
    '''вывод десятичного числа'''
    print("{:.2f}".format(x))

def f2(x):
    return "{:.2f}".format(x)

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

f1(1/3)  # вызов функции без возвращаемого значения

#вызов функций, возвращающих значения
print("результат f2 равен", f2(1/7))
X = summa(3, 5)
Y = summa(y=5, x=3)
Z = summa(3, y=5)
print(X, Y, Z)
```

При определении функции параметру можно присвоить значение (значение по умолчанию). Тогда при вызове функции такие параметры

можно не задавать. Необязательные параметры должны всегда указываться в конце списка параметров после обязательных.

```
import math
def fun(x1=0, y1=0, x2=0, y2=1):
    return math.sqrt((x2-x1)**2+(y2-y1)**2)

print(fun())  #все значения по умолчанию
print(fun(y2=1, x2=1)) #часть значений по умолчанию
print(fun(, , 1, 1))  #ошибка, пропускать параметры нельзя
```

Если значения параметров, которые нужно передать в функцию, содержатся в списке или кортеже, то в списке параметров при вызове функции перед этим объектом нужно указать символ звездочка. Объект (список или кортеж) будет распакован.

```
P = [1, 1, 4, 5]
print(fun(*P))   #эквивалентно print(fun(1, 1, 4, 5))
P1 = [0, 2]
print(fun(*P1))  #эквивалентно print(fun(0, 2))
```

Аналогично можно передавать параметры и в стандартные функции в том числе в функции с произвольным количеством параметров, например, print или max:

```
num = [4, 10, 25]
print(num)    #выведет [4, 10, 25]
print(*num)   #выведет 4 10 25 как при вызове print(4, 10, 25)
m = max(*num)
print(m)
```

Если значения параметров находятся в словаре, то распаковать параметры можно, указав в списке параметров перед именем словаря две звездочки:

```
D = {'x1':1, 'y1':1, 'x2':4, 'y2':5}
print(fun(**D)) #эквивалентно print(fun(x1=1, y1=1, x2=4, y2=5))
D1 = {'x1':0, 'y1':2}
print(fun(**D1))  #эквивалентно print(fun(x1=0, y1=2))
D2 = {'a':0, 'b':2}
print(fun(**D2))  #ошибка, у функции нет параметров a и b
```

***Глобальные и локальные переменные***

Глобальные переменные – переменные, созданные в программе вне функции.

Локальные переменные – переменные, которым внутри функции присваивается значение. Параметры функции являются локальными переменными. Локальные переменные недоступны за пределами функции, поэтому им можно давать любые имена, даже если такие имена уже использовались.

Если имя локальной переменной совпадает с именем глобальной переменной, то все операции внутри функции выполняются с локальной переменной, а значение глобальной переменной не изменяется.

Для доступа к значению глобальной переменной из функции достаточно, чтобы внутри функции не было переменной с таким же именем, как у глобальной переменной. Однако, чтобы изменять значение глобальной переменной из функции, нужно объявить ее внутри функции с помощью служебного слова global.

*Пример использования глобальной и локальной переменной*

```
def f3():
    print("f3: ", value)

def f4():
    value = 2
    print("f4: ", value)

def f5():
    global value
    value = 20

value = 1
f3()
f4()
print(value)
f5()
print(value)
```

Злоупотреблять использованием глобальных переменных не следует, так как это может привести к трудно обнаруживаемым ошибкам.

Если нужно, чтобы функция изменила значение объекта, то совсем необязательно использовать глобальную переменную. Обратите внимание на следующий факт: функция может изменять значение объекта, переданного в качестве параметра, если это объект изменяемого типа (список, словарь). Например, рассмотрим две функции:

```
def f6(L):
    for i in range(len(L)):
        L[i] = L[i] + 1

my_list = [1, 2, 3]
f6(my_list)
print(my_list)  # напечатает [2, 3, 4]

def f7(L):
    L_new = [0]*len(L)
    for i in range(len(L)):
        L_new[i] = L[i] + 1
    L = L_new

my_list = [1, 2, 3]
f7(my_list)
print(my_list)  # напечатает [1, 2, 3]
```

Как видим, получился разный результат, хотя внутри функций мы изменяли значение параметра L почти одинаково. В первом случае переменные L и my_list ссылаются на один и тот же объект. Функция, используя L, изменила содержимое объекта, но объект остался тем же. Поэтому значение my_list тоже изменилось. Во втором случае мы создали новый объект и присвоили его локальной переменной L, но значение глобальной переменной my_list не изменилось, там по-прежнему старое значение.

Если, наоборот, требуется, чтобы функция не изменяла преданное значение, передавайте в функцию копию изменяемого объекта или создавайте такую копию внутри функции.

***Задания для самостоятельной работы***

1. Написать функцию, имеющую 3 параметра: первые 2 - числа, третий - операция, которая должна быть произведена над ними. Если третий параметр «+», то нужно сложить числа, если «-» — вычесть, «*» — умножить, «/» — разделить (первое на второе). Функция возвращает результат выполнения операции над числами. Если операция не совпадает с указанными выше, то выводится сообщение "Неизвестная операция", и возвращается значение None.
2. Напишите функцию, которая для заданного радиуса r вычисляет площадь круга и длину окружности. Функция возвращает кортеж из 2 значений.
3. Для треугольника со сторонами $x, y, z$ угол $\alpha$ между сторонами $x, y$ можно вычислить следующим образом:

   $$d = \cos\alpha = \frac{x^2 + y^2 - z^2}{2xy};\ \alpha = \arccos d = \frac{\pi}{2} - \arcsin d$$

   Напишите функцию, которая находит угол $\alpha$ для треугольника со сторонами $x, y, z$ в градусах (воспользуйтесь функциями модуля math). Используя эту функцию напишите еще одну функцию, которая по заданным сторонам треугольника находит все его углы (в градусах). Функция возвращает кортеж из 3 чисел, причем первое число – угол, находящийся напротив стороны x, второе – угол напротив y, третье – угол напротив z.
4. Напишите функцию, которая находит наибольший общий делитель двух чисел, используя модифицированный алгоритм Евклида: нужно заменять большее число на остаток от деления большего на меньшее до тех пор, пока этот остаток не станет равен нулю; тогда второе и есть НОД. Функция должна возвращать найденное значение.

5. Дана дробь $\frac{n}{m}$, n и m - натуральные числа. Напишите 2 функции, которые сокращают эту дробь, то есть находят числа p и q такие, что $\frac{n}{m} = \frac{p}{q}$, и дробь $\frac{p}{q}$ — несократимая:
   - аргументами функции являются числа n, m, функция возвращает кортеж (p, q);
   - аргументом функции является список [n, m], функция не возвращает значения, а изменяет этот список на [p, q].

   Для поиска НОД воспользуйтесь функцией из предыдущего задания.

***Переменное число параметров в функции***

Если перед именем параметра в определении функции указать символ «*», то функции можно будет передать произвольное число параметров. Переданные параметры сохраняются в кортеже.

Если перед именем параметра в определении функции указать две звездочки, то функции можно будет передать произвольное число именованных аргументов. Переданные параметры сохраняются в словаре.

Перед параметрами со звездочками можно указать несколько обязательных параметров и параметров, имеющих значения по умолчанию. Параметры со звездочками всегда указываются в конце списка параметров, причем параметр с одной звездочкой указывается перед параметром с двумя звездочками.

```
def summa ( *numb ) :
    s = 0
    for x in numb :
        s += x
    return s

X = summa(1,-2,5)
Y = summa(1,3,5,7,9,11)
```

```
def summa1 ( **d ) :
    s = 0
    for x in d :
        s += d[x]
    return s

X = summa1(a=1,b=-2,c=5)
Y = summa1(z1=1, z2=3)
```

***Задания для самостоятельной работы***

1. Напишите функцию для решения уравнений степени не выше второй (квадратные и линейные):
   - если у функции три аргумента, их надо трактовать как a, b и c в уравнении $ax^2 + bx + c = 0$;
   - если два — как коэффициенты b и c в уравнении $bx + c = 0$;
   - если один — как коэффициент c в уравнении $c = 0$;
   - если список коэффициентов пуст или коэффициентов больше трёх, то функция должна вернуть None.

   Функция возвращает список, содержаний все корни уравнения (два, один или ни одного). Если корнем является любое значение x, функция возвращает список, содержащий символ «*» (["*"]).
2. Напишите функцию, параметрами которой являются слово и любое количество произвольных строк. Функция возвращает список строк, в которых встречается указанное в первом параметре слово. Строки могут состоять из букв и пробелов. Регистр букв не имеет значения. Например, при вызове f(‘Два’, ‘двадцать пять’, ‘сорок два’) функция вернет список [‘сорок два’]. Вызовите функцию для случая, когда строки хранятся в заранее созданном списке.
3. Напишите функцию, которая имеет произвольное количество именованных параметров. Функция возвращает список всех имен параметров и сумму всех значений. Например, при вызове f(a=2, b=3,

c=2, d=5) функция вернет (['a','b','c','d'], 12). Вызовите функцию для случая, когда параметры хранятся в заранее созданном словаре, например, res = {'Математика':92, 'Информатика':80, 'Физика':74}.

***Функции в качестве параметров***

В языке Python все является объектом, в том числе и функции. Инструкция def создает объект типа function и сохраняет ссылку на него в переменной с именем, указанным после def. Мы можем скопировать эту ссылку в переменную с другим именем и использовать новую переменную для вызова функции.

```
def f8(x):
    return x + 10

my_f = f8
print(my_f(5))
```

Можно передать ссылку на функцию в качестве параметра другой функции.

```
def print_table(fun,a=0,b=1,h=0.1):
    '''
    печать таблицы значений функции
    fun - ссылка на функцию с одним числовым параметром,
          возвращающую число
    a, b - диапазон изменения параметра
    h - шаг
    '''
    t = a
    while t <= b:
        #вызов реальной функции через переменную fun
        print("{0:5.2f}{1:7.2f}".format(t, fun(t)))
        t += h

print_table(f8, a=-1, h=0.2) #передается своя функция
print_table(math.sqrt)       #передается стандартная функция
```

Функции, которые принимают или возвращают другие функции, называются функциями высшего порядка.

### *Анонимные функции*

Очень часто в качестве аргумента для функций высшего порядка требуется совсем простая функция. Причем нередко такая функция нужна в программе только в одном месте, поэтому ей необязательно даже иметь имя.

Такие короткие безымянные (анонимные) функции можно создавать инструкцией

**lambda** *параметры*: *выражение*

Такая инструкция создаст ссылку на функцию, принимающую указанный список параметров и возвращающую результат вычисления выражения. Эту ссылку можно сохранить в переменной или передать в качестве параметра в другую функцию. Вызывается эта функция как обычная функция.

Тело лямбда-функции состоит из одного выражения. Инструкция return подразумевается, но не пишется. Скобки вокруг параметров отсутствуют, аргументы от выражения отделяет двоеточие. Параметры могут иметь значения по умолчанию.

```
f1 = lambda x: (x+2)/5
f2 = lambda x, y=1: y/(x*x+1)

print(f1(3))
print(f2(2))
print(f2(2, 2))
```

Пример анонимной функции с проверкой условия:

```
sign = lambda x: -1 if x < 0 else (0 if x == 0 else 1)
```

Эта функция возвращает -1 для отрицательных значений, 0 – для нулей, 1 – для положительных.

Анонимные функции используются:

- в качестве параметра функции:

```
print_table(lambda x: x+10, h=0.2)
```

- в качестве возвращаемого значения:

```
def pr(a):
    if a<0:
        return lambda x: abs(x)
    else:
        return lambda x: x

f = pr(-1)
v = -2.5
print(f(v))
```

***Задания для самостоятельной работы***

1. Напишите функцию, которая находит максимум функции f(x) в точках отрезка [a,b] с постоянным шагом h. Параметрами функции являются f, a, b, h. Параметры a, b, h – необязательные, по умолчанию a=0, b=1, h=0.1. Используя эту функцию, найдите максимум функции (2-x)sin(x/2) на отрезке [0, 4].
2. Напишите функцию, которая находит все точки, в которых достигается максимальное значение, на заданном отрезке. Функция возвращает список найденных точек. Считаем, что $f(x_1) = f(x_2)$, если $|f(x_1) - f(x_2)| \leq eps$. $eps$ является параметром функции, по умолчанию $eps = 10^{-3}$. Остальные параметры такие же, как в предыдущей функции. Используя эту функцию, найдите точки, в которых достигается максимум функций $sin^2(\frac{x}{2})$ и $2sin^2 3x$, на отрезке [0, 4].

***Встроенные функции высшего порядка***

Ниже рассматриваются некоторые встроенные функции высшего порядка, которые часто используются на практике. На самом деле их достаточно много. Описание следует искать в документации и специальной литературе.

map(f, *args) – преобразует элементы заданных последовательностей с помощью функции f в новый объект, поддерживающий итерацию. Если при вызове задана одна последовательность, то функция f по очереди выполняется для каждого элемента исходной последовательности (элемент является параметром функции f). Если последовательностей несколько, то при вычислении функции f берутся по одному элементу из каждой последовательности. Этот набор значений будет параметрами функции f. Вычисленные значения образуют новую последовательность, которая и является результатом функции map(). Полученный объект не является списком. Преобразовать его в список можно с помощью функции list(). Примеры:

- получаем список длин строк для заданного списка строк:

```
str_lengths = list(map(len, ['abc', 'aaattt', 'uf']))
print(str_lengths)  #результат [3, 6, 2]
```

- получаем список, составленный из попарных произведений двух исходных списков:

```
a = [1, 2, 3]
b = [10, 100, 1000]
c = list(map(lambda x,y:x*y, a, b))
print(c)   #результат  [10, 200, 3000]
```

filter(f, seq) – позволяет отобрать элементы последовательности, удовлетворяющие условию. Функция f должна возвращать True для элементов, включаемых в результат, и False – в противном случае. Второй параметр – исходная последовательность. Если в первом параметре указать значение None вместо имени функции, то каждый элемент исходной последовательности будет проверен на соответствие значению True. Как и в случае с map() функция

возвращает итерируемый объект. Для преобразования его в список используйте list(). Примеры:

- удалим «пустые» значения из списка (нули, пустые строки, пустые списки):

```
L = [1, 0, None, [], ['**'], '', '++']
L_new = list(filter(None, L))
print(L_new)  #результат [1, ['**'], '++']
```

- найдем все числа от 1 до 99, которые при делении на 15 дают 2 в остатке:

```
k = list(filter(lambda x: x % 15 == 2, range(1,100)))
print(k)  # результат [2, 17, 32, 47, 62, 77, 92]
```

reduce(f, seq[, initial]) – применяет указанную функцию f к парам элементов и накапливает результат. Функция f должна иметь два параметра. При первом обращении к f ее параметрами являются два первых элемента последовательности. При последующих обращениях в качестве первого параметра используется значение, которое f вернула на предыдущем шаге, а в качестве второго – очередной элемент последовательности. Значение initial, если оно задано, при вычислении добавляется перед элементами последовательности seq, а также является значением по умолчанию, если последовательность seq пуста. Функция находится в модуле functools. Примеры:

- вычислим сумму элементов списка:

```
from functools import reduce

sum_all = reduce(lambda x,y: x + y, [1,2,3,4,5])
print(sum_all) #результат 15 = ((((1+2)+3)+4)+5)
sum_all = reduce(lambda x,y: x + y, [1,2,3,4,5], 100)
print(sum_all) #результат 115 = (((((100+1)+2)+3)+4)+5)
sum_all = reduce(lambda x,y: x + y, [], 100)
print (sum_all) #результат 100
```

- вычислим, сколько раз символ «*» встречается в списке строк:

```
w = ['aaa*', '*bc*', 'abc']
s_count = reduce(lambda a, x: a + x.count('*'), w, 0)
print(s_count)   #результат 3
```

- вычислим наибольший элемент списка с помощью reduce^

```
items = [1, 24, 17, 14, 9, 32, 2]
all_max = reduce(lambda a,b: a if (a > b) else b, items)
print(all_max)
```

sorted(seq, key=None, reverse=False) – возвращает новый список, содержащий все элементы исходной последовательности в порядке возрастания значений элементов (по умолчанию). Если параметр reverse=True, то элементы располагаются в порядке убывания значений. В параметре key можно указать ссылку на функцию с одним параметром, возвращающую значение. Тогда сортировка выполняется следующим образом. Для каждого элемента последовательности вычисляется значение функции, указанной в key (элемент последовательности является параметром функции). Элементы исходной последовательности переставляются таким образом, чтобы вычисленные для них значения были отсортированы.

Для списков в языке определен метод sort(), который очень похож на функцию sorted(). Параметры метода key и reverse имеют тот же смысл. Отличие метода от функции состоит в том, что он применяется только для списков, и он не создает новый список, а сортирует имеющийся. Примеры:

- отсортируем список строк с использованием различных значений параметров:

```
L = ['aaaa','cbb','ba','c']
L1 = sorted(L)
print(L)   #результат ['aaaa', 'cbb', 'ba', 'c']
print(L1)  #результат ['aaaa', 'ba', 'c', 'cbb']
L.sort()
print(L)   #результат ['aaaa', 'ba', 'c', 'cbb']
L.sort(key = len)
print(L)   #результат ['c', 'ba', 'cbb', 'aaaa']
L.sort(reverse=True)
print(L)   #результат ['cbb', 'c', 'ba', 'aaaa']
```

- отсортируем список слов по алфавиту. Если слова содержат символы в разных регистрах, то сортировка получается неправильной, так как коды символа в разных регистрах разные. Чтобы получить правильный результат, нужно привести все символы к одному регистру. Для этого можно, например, воспользоваться методом lower() у строк. Обратите внимание, как можно сослаться на метод в параметре key:

```
L = ['мЯЧ', 'мел', 'Юла']
print(sorted(L))                #результат ['Юла','мЯЧ','мел']
print(sorted(L, key=str.lower)) #результат ['мел','мЯЧ','Юла']
```

- сформируем последовательность чисел от 1 до 9 такую, чтобы остатки от деления на 3 этих чисел шли в порядке возрастания:

```
numb = sorted(range(1,10), key=lambda x:x%3)
print(numb)  #результат [3, 6, 9, 1, 4, 7, 2, 5, 8]
```

***Задания для самостоятельной работы***

1. При помощи функций map/filter/reduce возвести в квадрат числа от 1 до 100, и рассчитать их сумму, не включая в сумму числа, кратные 9.
2. При помощи функций map/filter/reduce превратить список целых чисел в строку, содержащую строковое представление этих чисел, разделенных пробелами.

3. При помощи функций map/filter/reduce из несколько одинаковых подряд идущих элементов списка оставить только один. Например, [1, 2, 3, 4, 4, 4, 5, 6, 6, 7, 6, 1, 1] -> [1, 2, 3, 4, 5, 6, 7, 6, 1]
4. При помощи функций map/filter/reduce из списка списков извлечь элементы, содержащиеся во вложенных списках по индексу 1. Например, [[1, 2, 3], [2, 3, 4], [0, 1 , 1 , 1], [0, 0]] -> [2, 3, 1, 0]
5. Дан список A, состоящий из 2N элементов. Разбейте его на списки B и C по N элементов каждый так, чтобы каждый элемент B не превосходил каждого элемента C.
6. В списке 2n + 1 различных элементов. Найдите средний элемент списка. Под средним элементом понимают такой, для которого в списке n элементов больше его и n элементов меньше.
7. Дан список из *N* элементов. Вывести индексы списка в том порядке, в котором соответствующие им элементы образуют возрастающую последовательность.
8. Для списков вида [[‘Иванов’, 3,5,4], [‘Петров’, 4, 5, 5], [‘Сидоров’, 3, 3, 3], [‘Николаев’, 4, 4, 3]] напишите функцию, которая выводит на экран этот список в виде таблицы в отсортированном виде. Параметрами функции являются список, функция, которая определяет порядок сортировки (значение параметра key в sort), и параметр, определяющий, выполняется сортировка по возрастанию или убыванию. Функция не возвращает значение и не изменяет исходный список.

   Используя эту функцию, выведите исходный список, отсортированный:

   - по фамилиям в алфавитном порядке;
   - в порядке возрастания первой оценки;
   - в порядке убывания суммы баллов.

## 2. Модули

### *Использование модулей*

Модулем в языке Python называется файл с программой.

Чтобы воспользоваться модулем, нужно выполнить инструкцию import.

```
import math
x = math.sqrt(2)
```

Или так:

```
import math as mm  #указываем псевдоним
x = mm.sqrt(2)
```

Или так:

```
from math import sqrt, exp, log, sin   #только нужные имена
x=sqrt(2)
```

Получить информацию о содержимом модуля:

```
>>> dir(math)
```

### *Использование собственных модулей*

Содержимое модуля ***mod_test.py***:

```
def f1():
    print("функция f1")

x = 123
```

Содержимое модуля, в котором используется модуль mod_test:

```
import mod_test as mm

mm.f1()
print(mm.x)
```

Оба файла размещаем в одной папке.

После подключения ***mod_test.py*** внутри папки будет создан каталог **__pycache__** с файлом ***mod_test.cpython-35.pyc***. Этот файл содержит скомпилированный байт-код исходного модуля.

Если его переименовать в ***mod_test.pyc***, то полученный файл можно использовать вместо файла ***mod_test.py***, например, при распространении программ.

### *Повторная загрузка модулей*

Модуль загружается один раз при первой операции импорта.

Если в модуль вносились изменения, то его нужно перезагрузить. Иначе вы будете работать со старой версией модуля.

Для повторной загрузки модуля выполните команды:

```
>>> import imp
>>> reload(mm)
```

Или, если псевдоним не использовался,

```
>>> from imp import reload
>>> reload(mod_test)
```

### *Пути поиска модулей*

Поиск подключаемого модуля выполняется в папках, указанных в списке sys.path модуля sys. Чтобы увидеть этот список, выполните

```
>>> import sys
>>> sys.path
```

При поиске список просматривается слева направо. Поиск прекращается после первого найденного модуля.

Из программы список sys.path можно изменить, используя его методы:

```
import sys
sys.path.append(r"C:\Мои модули") #в конец списка
sys.path.insert(0, r"C:\Мои модули") #в начало списка
```

***Задания для самостоятельной работы***

1. Создайте собственный модуль, поместив в него ваши функции, созданные в предыдущих заданиях. Напишите любую программу с использованием функций этого модуля.

## 3. Текстовые файлы

***Текстовый файл*** – это файл, в котором хранится последовательность символов в определенной кодировке. Файл может быть разбит на строки с помощью специального символа «конец строки».

### *Открытие и закрытие файла*

При открытии файла создается объект, с помощью которого выполняется работа с файлом.

```
f1 = open('myfile.txt', 'r')
f1 = open(r'D:\Файлы\myfile.txt')
f1 = open('D:\\Файлы\\myfile.txt')
```

Режимы доступа к текстовым файлам:

'r' – чтение из текстового файла. Если файл не существует, то выдается сообщение об ошибке (режим по умоллчанию);

'w' – запись в текстовый файл. Если файл существует, то содержимое заменяется. Если файл не существует, то он создается;

'a' – дозапись в текстовый файл. Если файл существует, то данные дописываются в конец. Если файл не существует, то он создается.

Закрыть файл:

```
f1.close()
```

### *Чтение текстового файла*

***Метод read( )*** позволяет прочесть из файла указанное количество символов.

Метод возвращает строку, состоящую из прочитанных символов.

Каждый следующий вызов read() начинает работу оттуда, где завершил работу предыдущий.

Если все символы прочитаны, то очередной вызов read( )возвратит пустую строку. Например, так можно прочитать весь файл по одному символу:

```
s = ''
s1 = f1.read(1)
while s1 != '':
    s = s + s1
    s1 = f1.read(1)
print(s)
```

Если количество символов не указать, будет возвращен весь текстовый файл одной строкой:

```
s = f1.read()
print(s)
```

***Метод readline()*** позволяет прочитать указанное количество символов текущей строки. Если в строке символов меньше, чем указано, то будут прочитаны символы до конца строки.

Если количество символов не указать, будут прочитаны все символы с текущей позиции до конца строки.

Метод возвращает строку, состоящую из прочитанных символов, включая символ '\n'.

Каждый следующий вызов readline() начинает работу оттуда, где завершил работу предыдущий.

Если все символы прочитаны, то очередной вызов readline() возвратит пустую строку.

```
s = ''
s1 = f1.readline()
while s1 != '':
    s = s + s1
    s1 = f1.readline()
print(s)
```

***Метод readlines( )*** читает текстовый файл в список, элементами которого являются строки файла. Символ '\n' включается в строку.

```
L = f1.readlines()
```

***Перебор строк файла в цикле for***:

```
for line in f1:
    print(line.rstrip('\n'))
```

В этом примере в прочитанных строках убирается символ конца строки.

***Запись в текстовый файл***

Файл должен быть открыт с ключом 'w'  или 'a'.

***Метод write( )*** записывает строку в файл. Метод не вставляет символ '\n' в конце строки автоматически. Для перехода на новую строку символ нужно явно помещать в файл.

***Метод writelines( )*** записывает элементы списка строк в файл.

***Функция print()*** записывает информацию в файл, если ее необязательный параметр file равен ссылке на открытый файл.

```
f2 = open('результат.txt', 'a')
f2.write("строка1\nстрока2\n")
L = ['строка3\n', 'строка4\n']
f2.writelines(L)
print('строка5', end='', file = f2)
f2.close()
```

***Задания для самостоятельной работы***

1. Создать в текстовом редакторе файл, содержащий несколько строк. Определить максимальный и минимальный размер строки в файле и вывести их в другой файл. Вывести в этот же файл все строки максимальной длины.
2. Создайте текстовый файл, содержащий информацию о товарах и ценах на них. Каждая строка файла имеет вид: НАЗВАНИЕ ТОВАРА: ЦЕНА. Используя данный файл:
   - найдите цену указанного товара, или выдайте сообщение о том, что цена не известна;
   - добавьте в файл информацию о трех новых товарах;

- удалите из файла информацию о товаре;
- создайте новый файл, в котором товары будут упорядочены в порядке возрастания цен. Выведите на экран информацию о двух самых дешевых и двух самых дорогих товарах.

3. Создайте два текстовых файла, содержащие целые числа, в которых данные отсортированы по неубыванию. Каждая строка содержит одно число. Сформируйте новый файл из чисел входных файлов, чтобы его значения были также отсортированы по неубыванию (не использовать методы сортировки).

# 4. Объектно-ориентированное программирование

### *Создание и использование простого класса*

**Класс** описывает модель объекта, его свойства и поведение. С точки зрения программиста, класс — это тип данных, который создается для описания сложных объектов. Класс содержит набор переменных и функций. Переменные называются **атрибутами**. В них хранятся характеристики объекта (название, размер, цвет, количество и так далее). Функции называются **методами.** Метод определяет действие, которое объект может выполнять над самим собой или другими объектами

Класс создается с помощью инструкции вида:

**class** *ИмяКласса*:

['''*строка документирования*''']

*Переменная* = ЗНАЧЕНИЕ

…

**def** *ИмяМетода*(self, ...):

self.*Переменная* = ЗНАЧЕНИЕ

…

…

В Python принято, чтобы имена классов начинались с большой буквы. Если имя должно состоять из нескольких слов, то между словами не должно быть символов подчеркивания, а каждое слово внутри имени должно начинаться с большой буквы. Например, UniversityStudent.

**Объект** – экземпляр класса. Объект хранит конкретные значения атрибутов и информацию о принадлежности к классу, может выполнять методы. Например, int – это класс, а число 5 – это объект (экземпляр класса int).

Создать объект (экземпляр класса) можно так:

*ИмяОбъекта* = *ИмяКласса*([*Параметры_конструктора*])

Метод экземпляра класса – это функция, объявленная внутри класса. Методы экземпляра имеют обязательный первый параметр, который содержит ссылку на объект, для которого вызван метод. В Python принято называть этот параметр self. Не следует заменять это имя другим. Вызывая метод, вы не должны передавать значение для self явно – интерпретатор сделает это автоматически. Через эту ссылку выполняется доступ к атрибутам экземпляра класса.

Существуют методы со специальными именами, предназначенные для выполнения определенных действий. Одним из таких методов является метод __init__() (в начале и конце имени здесь стоят по два символа подчеркивания), который по аналогии с другими языками программирования называют конструктором. Конструктор автоматически вызывается при создании экземпляра класса. Обычно используется для задания начальных значений атрибутов. Этот метод, как любая функция, может иметь параметры (помимо self), которые указываются при создании объекта.

Метод __str()__представляет объект в виде строки. Метод вызывается при выводе с помощью функции print(), а также при использовании функции str().

Внутри класса можно создать метод, который будет доступен без создания экземпляра класса. Для этого используется конструкция:

*@staticmethod*

**def** *ИмяМетода*(…)

…

Такой метод называется статическим. В списке параметров статического метода отсутствует self. Обычно такие методы используют атрибуты класса. Для вызова метода используется конструкция *ИмяКласса.ИмяМетода*(…). Можно вызвать метод и через экземпляр класса *ИмяОбъекта.ИмяМетода*(…).

Атрибут – это переменная, созданная внутри класса. Чтобы создать атрибут нужно выполнить оператор присваивания. Важно понимать, что в классах есть 2 вида атрибутов: атрибуты объекта класса и атрибуты экземпляра класса.

**Атрибут объекта класса** (атрибут класса) создается при присваивании значения переменной внутри класса (но вне любого метода). Можно создавать атрибуты класса динамически (не в классе), используя инструкцию

*ИмяКласса.ИмяАтрибута* = ЗНАЧЕНИЕ

Атрибуты класса можно использовать даже тогда, когда ни одного экземпляра данного класса не существует. Атрибут класса доступен всем экземплярам класса, и для них всех это одна и та же переменная. Если атрибут класса изменить, то значение изменится для всех экземпляров класса. Для получения значения атрибута можно использовать конструкцию *ИмяОбъекта.ИмяАтрибута*, но для изменения значения только конструкцию *ИмяКласса.ИмяАтрибута*.

**Атрибут экземпляра класса** (атрибут) используется для хранения значений, которые для каждого экземпляра класса уникальны.

Создается атрибут внутри метода класса при выполнении инструкции присваивания:

*self.ИмяАтрибута* = ЗНАЧЕНИЕ

Для доступа к атрибуту внутри класса нужно использовать конструкцию *self.ИмяАтрибута*, а вне класса *ИмяОбъекта.ИмяАтрибута*. Атрибут можно создать динамически после создания объекта, выполнив инструкцию

*ИмяОбъекта.ИмяАтрибута* = ЗНАЧЕНИЕ

В качестве примера рассмотрим класс Счет, моделирующий банковский счет:

```
#класс Счет
class Счет():
    """Банковский счет"""

    count = 0 #создаем атрибут класса

    def __init__(self, number):
        self.number = number #создаем атрибуты
        self.balance = 0     #экземпляра
        Счет.count += 1      #изменяем атрибут класса

    def __str__(self):
        r = "Счет(number = " + str(self.number)
                             #используем атрибут экземпляра
        r = r + ", balance = " + str(self.balance) + ")"
        return r

    def change(self, summa):
        self.balance += summa #изменяем атрибут экземпляра

    @staticmethod             #создаем статический метод
    def info():               #используем атрибут класса
        print("Всего счетов: ", Счет.count)
```

```
#использование класса
print(Счет.count)    #используем атрибут класса

A = Счет(1234)       #создаем объект,
                     #вызываем метод __init__
A.change(2000)       #вызываем метод
print(A.balance)     #используем атрибут экземпляра

B = Счет(2453)       #создаем еще один объект
B.change(3000)

print(A)             #проверяем метод __str__()
print(B)
print(A.count)       #используем атрибут класса
Счет.info()          #вызываем статический метод
```

***Задания для самостоятельной работы***

1. Ниже приведены класс Point (точка), у которого имеются 2 атрибута x и y (координаты) и методы __init__() и __str__(), и класс Rect (прямоугольник), у которого есть:
   - два атрибута (верхний левый угол и правый нижний угол прямоугольника). Значениями атрибутов являются объекты класса Point;
   - методы __init__() и __str__();
   - метод sides(), возвращающий длины сторон прямоугольника;
   - метод perim(), вычисляющий периметр прямоугольника.

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

    def __str__(self):
        return "Point("+str(self.x)+","+str(self.y)+")"
```

```
class Rect:
    def __init__(self, top_left, bottom_right):
        self.A = top_left
        self.C = bottom_right

    def __str__(self):
        r = "Rect(" + str(self.A) + ","
        r = r + str(self.C) + ")"
        return r

    def sides(self):
        return abs(self.C.x-self.A.x),abs(self.A.y-self.C.y)

    def perim(self):
        a,b = self.sides()
        return 2*(a+b)
```

**Задание:**

- продемонстрируйте работу с классами, создав необходимые объекты и вызвав все их методы;
- создайте аналогичный класс для треугольника с такими же методами.

***Свойства***

Все элементы класса в языке Python являются открытыми, т.е. доступными вне класса. Однако между программистами существует соглашение о том, что переменная или метод, у которых имя начинается с одиночного подчеркивания, не предназначены для использования вне класса (защищенные элементы класса). Если же имя начинается с двух символов подчеркивания (закрытые члены класса), то оно автоматически заменяется в системе на имя *_ИмяКласса__ИмяЭлемента*. Этот механизм называется искажением имен. Он нужен для избегания конфликтов имен при наследовании.

При разработке собственных классов рекомендуется создавать методы для работы с атрибутами. Это гораздо безопаснее. Например, после

разработки класса вам может понадобиться добавить какую-то проверку или преобразование значения атрибута. Если вы использовали методы, то необходимые действия добавляются в метод, и не нужно вносить какие-либо изменения в остальной код. Если же вы использовали переменную, то изменения нужно делать во всех местах, где была использована эта переменная. Технология сокрытия информации о внутреннем устройстве объекта за внешним интерфейсом из методов называется **инкапсуляцией.**

Работа с методом не так удобна, как работа с переменной. Вызов метода обычно длиннее, чем просто обращение к переменной, поэтому в классах появилась возможность создавать специальные идентификаторы, через которые можно выполнять операции над атрибутами. Они называются свойствами.

Свойства при использовании выглядят как атрибут, но в отличие от атрибута для них можно задать необходимые действия в момент выполнения операций получения значения, присваивания значения и удаления значения. Создается свойство с помощью функции property():

*Свойство* = property(*МетодДляЧтения* [,
*МетодДляПрисваивания* [,
*МетодДляУдаления* [,
*Строка_документирования*]]])

Обязательным в данном случае является только первый параметр. Первые три параметра являются ссылками на методы класса.

Ниже приведен пример класса, в котором создано свойство v, и показано, как его можно использовать.

```
class Class1:
    def __init__(self, value):
        self.__var = value

    def __getVar(self):         # Чтение
        return self.__var

    def __setVar(self, value): # Запись
        self.__var = value

    def __delVar(self):         # Удаление
        del self.__var

    v = property(__getVar, __setVar, __delVar,
                 "Строка документирования")
c1 = Class1(10)
c1.v = 101        # Вызывается метод __setVar()
print (c1.v)      # Вызывается метод __getVar()
del c1.v          # Вызывается метод __delVar()
```

Методы __setVar(),__getVar(),__delVar реализованы в этом примере самым простым способом. На практике в них можно выполнять необходимые проверки и преобразования значений.

Добавим свойство balance в класс Счет и сделаем атрибуты закрытыми:

```
#класс Счет
class Счет():
    """Банковский счет"""

    __count = 0 #создаем атрибут класса

    def __init__(self, number):
        self.__number = number #создаем атрибуты
        self.__balance = 0     #экземпляра
        Счет.__count += 1      #изменяем атрибут класса

    def __str__(self):
        r = "Счет(number = " + str(self.__number)
                               #используем атрибут экземпляра
        r = r + ", balance = " + str(self.balance) + ")"
                               #используем свойство
        return r
```

```
    def change(self, summa):
        self.__balance += summa #изменяем атрибут экземпляра

    @staticmethod               #создаем статический метод
    def info():                 #используем атрибут класса
        print("Всего счетов: ", Счет.count)

    def __get_balance(self):
        x = input('Введите ваш код доступа: ')
        if  x == '123':
            return self.__balance
        else:
            print('Неверный код доступа')
            return None

    balance = property(__get_balance) #создаем свойство
                                      #только для чтения

#использование класса
A = Счет(1234);
print(A.balance)
A.change(2000)
print(A.balance)
A.balance = 2000    #возникнет ошибка
                    #AttributeError: can't set attribute
```

***Задания для самостоятельной работы***

1. Создайте класс Length (Длина), имеющий свойства:
   - value (значение),
   - unit (единица измерения).

   При изменении единицы измерения значение должно соответственно меняться. Например, при переходе от сантиметров к метрам значение должно уменьшаться в 100 раз. Допустимые значения свойства unit: ‘см’, ‘м’, ‘км’. Организуйте эту проверку. Продемонстрируйте работу с классом.
2. Имеется класс Arr, моделирующий работу с одномерным массивом, у которого индексы могут принимать значения из любого диапазона.

Обратите внимание на свойства a и b. Их значения определяются в момент создания объекта, потом эти значения изменить нельзя (метод для присваивания нового значения не реализован). Это сделано специально, так как от значений a и b зависит размер списка, в котором хранятся значения элементов.

```
class Arr:
    def __init__(self, a,b):
        self.__a = a
        self.__b = b
        self.__val = [0]*(self.b - self.a+1)

    a = property(lambda self:self.__a)
    b = property(lambda self:self.__b)

    def __str__(self):
        return " ".join(map(str,self.__val))

    def arr_range(self):
        return range(self.a, self.b+1)

    def get(self, i):
        if self.a <= i <= self.b:
            return self.__val[i - self.a]
        else:
            print("Недопустимый индекс", i)

    def put(self, i, x):
        if self.a <= i <= self.b:
            self.__val[i - self.a] = x
        else:
            print("Недопустимый индекс", i)

#использование класса
Z = Arr(-5,5)
print(Z.a, Z.b)
print(Z)
for i in Z.arr_range():
    Z.put(i,Z.get(i)+i)
print(Z)
```

**Задание:**

- добавьте в этот класс метод, заполняющий массив случайными целыми числами из заданного диапазона (границы диапазона – параметры метода);
- создайте класс, моделирующий работу с двумерным массивом с аналогичными методами. Индексы массива изменяются с1.

3. Создайте класс Ведомость, имеющий

   атрибут класса:

   - список_дисциплин (значением является список с названиями дисциплин);

   свойства:

   - дисциплина (при задании значения проверять наличие дисциплины в атрибуте список_дисциплин),
   - группа;

   методы:

   - put – добавляет в ведомость информацию об оценке студента (фамилия, оценка – параметры метода). Для хранения данных внутри класса используйте словарь, в котором ключом является фамилия студента. Возможные оценки – «отлично», «хорошо», «удовл.», «неудовл.», «н/я»;
   - get – возвращает оценку, полученную студентом (фамилия студента – параметр метода);
   - change – изменяет оценку, полученную студентом (фамилия студента и новая оценка – параметры метода);
   - del – удаляет информацию о студенте из ведомости (фамилия студента – параметр метода);
   - result – возвращает кортеж из 5 чисел (количество каждого вида оценок в ведомости);
   - __init__ – конструктор;

- __str__ – возвращает строку, содержащую заголовки (название экзамена, группа) и результаты экзамена в виде таблицы;
- count – возвращает количество студентов в ведомости;
- names – возвращает список фамилий, имеющихся в ведомости.

Продемонстрируйте работу с классом.

***Наследование и полиморфизм***

Возможность кода работать с разными типами данных называют **полиморфизмом**. Рассмотрим следующий пример:

```
from math import pi

class Circle:
    def __init__(self, radius):
        self.r = radius

    def perim(self):
        return 2 * pi * self.r


class Rect:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def perim(self):
        return 2 * (self.a + self.b)

def print_info(x):
    print(f"perimeter = {x.perim():.5f}")
```

В данном случае у нас имеются 2 разных класса с одинаковым интерфейсом (метод perim) и полиморфная функция print_info, которая печатает информацию о периметре фигуры. Мы можем использовать эту функцию, например, так:

```
c1 = Circle(1)
print_info(c1)  #  perimeter = 6.28319
r1 = Rect(2, 5)
print_info(r1)  # perimeter = 14.00000
```

Как видим, одна и та же функция работает и для окружности, и для прямоугольника. Причем для каждого объекта вызывается метод из его класса. Это и есть полиморфизм. Какой метод вызывать в каждом конкретном случае Pyton определяет с помощью специального атрибута __class__, который у каждого объекты содержит ссылку на его класс.

Чтобы полиморфизм работал, в Python достаточно, чтобы методы имели одинаковые имена, одинаковое количество параметров, одинаково возвращали (или не возвращали) значения. Кроме того, в разных классах методы с одинаковыми именами должны выполнять одинаковые по смыслу действия, хотя и по-разному.

Еще одна базовая концепция объектно-ориентированного программирования (наряду с инкапсуляцией и полиморфизмом) это – наследование. Наследование позволяет создавать новые классы из существующих. При этом реализуется отношение вида «класс B является частным случаем класса A». Класс A называется базовым классом (или суперклассом), а B – производным классом (или подклассом). В программе это записывается следующим образом:

```
class B(A):
    …
```

Созданный класс можно использовать в качестве базового для создания следующего класса. Используя один и тот же базовый класс можно создать несколько разных производных классов. Таким образом механизм наследования позволяет создавать целые иерархии классов.

Основное преимущество наследования состоит в том, что в новом (производном) классе будут доступны атрибуты и методы базового класса. Их не нужно создавать повторно.

При обращении к методу (или атрибуту) экземпляра производного класса метод сначала ищется в исходном (производном) классе. Если метода

там нет, он ищется в базовом классе. Эти шаги повторяются до тех пор, пока метод не будет найден, или пока не дойдем до класса, который ни от кого не наследуется.

В производном классе можно создать новые методы и атрибуты, в том числе может потребоваться создать метод с таким же именем, как в базовом классе, например, конструктор (имя конструктора во всех классах одинаковое). Это называется переопределением метода. При вызове алгоритм поиска метода всегда один и тот же независимо от того, был он переопределен или нет: если метод найден в текущем классе, то он и выполняется, если не найден, то ищется в базовом классе.

Таким образом, если вы в новом классе не создали новый конструктор, то при создании экземпляра этого нового класса будет вызываться конструктор базового класса. Если же вы создали в новом классе свой конструктор, то он и вызовется. Но учтите, что в этом случае конструктор базового класса автоматически вызываться не будет, хотя в большинстве случаев нам нужно, чтобы он тоже выполнился.

Для того, чтобы в производном классе B вызвать одноименный метод базового класса A, существует несколько способов. Рассмотрим два из них (на примере конструктора):

```
A.__init__(self)
super().__init__()
```

В первом случае явно указывается имя класса, в котором нужно найти метод, и в первом параметре передается ссылка на экземпляр класса (self). Во втором – используется функция super(). В этом случае self передавать не нужно, а поиск метода будет выполняться во всех базовых классах.

В качестве примера рассмотрим 3 класса: базовый класс People и производные Teacher и Student. Методы базового класса __init__() и __str__() переопределяются в производных классах таким образом, что в них

вызываются методы с таким же именем базового класса. В примере продемонстрированы оба способа вызова нужного метода базового класса.

Обратите внимание на полиморфный метод info(), реализованный в базовом классе. Он работает и в производных классах.

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

    def __str__(self):
        return f'Имя: {self.name}, Возраст: {self.age}'

    def info(self):
        print(self.__class__.__name__ + ': ' + str(self))


class Teacher(People):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary = salary

    def __str__(self):
        return f'{super().__str__()}, Зарплата: {self.salary}'


class Student(People):
    def __init__(self, name, age, marks):
        People.__init__(self, name, age)
        self.marks = marks

    def __str__(self):
        return f'{super().__str__()}, Оценки: {self.marks}'


t = Teacher('Иванова', 40, 30000)
s = Student('Петров', 20, [70, 75, 87])

members = [t, s]
for member in members:
    member.info()
```

***Задания для самостоятельной работы***

1. Используя класс People в качестве базового, создайте класс Сотрудник (Worker), имеющий

   свойства:

   - должность (post)
   - зарплата (salary)

   методы:

   - __init__ – конструктор;
   - __str__ – аналогично методу класса Teacher из примера.

   Используя класс Сотрудник в качестве базового создайте класс Преподаватель (Teacher), имеющий:

   - закрытый атрибут дисциплины (disciplines), в котором хранятся названия дисциплин, которые ведет преподаватель;
   - методы __init__ и __str__;
   - методы добавить_дисциплину (add_dis) и удалить_дисциплину (delete_dis), которые позволяют изменять список дисциплин.

   Создайте список, содержащий по 2 объекта каждого класса (People, Worker, Teacher). Для этого списка:

   - выведите информацию о каждом человеке с помощью метода info;
   - выведите фамилии тех, кто моложе 30 лет;
   - продемонстрируйте работу со свойствами должность и зарплата и методами добавить_дисциплину и удалить_дисциплину.

2. Создайте класс Заказ(Order), у которого есть свойства код_товара(code), цена(price), количество(count) и методы __init__ и __str__.

   Создайте 2 класса-потомка: Опт(Opt) и Розница(Retail). В этих классах создайте методы __init__, __str__.и сумма_заказа(summa), позволяющий узнать стоимость заказа. Для опта стоимость единицы товара составляет 95% от цены, а при покупке более 500 штук – 90% цены. В розницу стоимость единицы товара составляет 100% цены. Стоимость заказа равна произведению цены на количество.

   Продемонстрируйте работу с классами, создав необходимые объекты и обратившись к их свойствам и методам.

***Специальные методы***

В языке Python существует большое количество специальных методов, которые вы можете реализовать в своих классах. В литературе их также называют магическими методами. Имена таких методов начинаются и заканчиваются двумя символами подчеркивания. Мы уже сталкивались с ними: __init__(), __str__(). Имена специальных методов и их смысл определены создателями языка: создавать новые нельзя, можно только реализовывать существующие. Поэтому, если вы создадите свой метод, у которого имя имеет два символа подчеркивания в начале и в конце, он от этого не станет специальным, а вот если это имя совпадет случайно с именем специального метода, то вы можете получить непонятные побочные эффекты и ошибки. Поэтому используйте такие имена только для реализации специальных методов.

Описание специальных методов следует искать в литературе и документации по языку. Обращать внимание нужно не только на название метода, но и на параметры и возвращаемое значение. Ваша реализация должна точно соответствовать этому описанию. Это очень важно, так как

эти методы обычно вызываются неявно в отличие от обычных методов, поэтому их и называют магическими. Важно также четко понимать, в какой момент вызывается метод.

Типичное использование специальных методов – реализация различных операций для ваших объектов. Гораздо удобнее и естественнее написать a + b или a < b, чем использовать выражения вида a.add(b) или a.less(b). Для всех операторов языка существуют специальные методы, которые вы можете реализовать в своих классах. Однако придумать новые операторы и создать для них свои методы невозможно.

Ниже приводится пример класса, для которого реализована работа с некоторыми операторами. Какому оператору или функции соответствует метод указано в комментариях. Для двуместных операций (+, -, <, >, ==, !=) левым операндом является текущий объект (доступ через self), правый операнд передается через параметр метода. Обратите внимание, что методы __add__, __sub__ создают и возвращают новый объект класса Vector2D, а методы __iadd__, __isub__ изменяют текущий объект и его же возвращают.

```
import math

class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        #вызывается при выводе в интерактивной оболочке
        #вызывается функцией repr
        #вызывается при отсутствии __str__
        #обычно результат - строка, создающая объект
        return 'Vector2D({}, {})'.format(self.x, self.y)

    def __str__(self):
        #вызывается функциями str, print и format
        #обычно результат - строка, понятная человеку
        return '({}, {})'.format(self.x, self.y)
```

```
    def __add__(self, B):   #сложение A + B
        return Vector2D(self.x + B.x, self.y + B.y)

    def __sub__(self, B):   #вычитание (A - B)
        return Vector2D(self.x - B.x, self.y - B.y)

    def __iadd__(self, B):  # A += B
        self.x += B.x
        self.y += B.y
        return self

    def __isub__(self, B):   # A -= B
        self.x -= B.x
        self.y -= B.y
        return self

    def __neg__(self):   #унарный минус (-A)
        return Vector2D(-self.x, -self.y)

    def __len__(self):   #вызывается функцией len
        return math.sqrt(self.x**2 + self.y**2)

    def __lt__(self, B):   #A<B
        l1=math.sqrt(self.x**2 + self.y**2)
        l2=math.sqrt(B.x**2 + B.y**2)
        return l1 < l2

    def __gt__(self, B):   #A>B
        l1=math.sqrt(self.x**2 + self.y**2)
        l2=math.sqrt(B.x**2 + B.y**2)
        return l1 > l2

    def __eq__(self, B):   #A==B
        return self.x==B.x and self.y==B.y

    def __ne__(self, B):   #A!=B
        return self.x!=B.x or self.y!=B.y

a = Vector2D(3, 4)
print(a)
b = Vector2D(5, 6)
c=a+b-Vector2D(1, 1)
print(c)
print( -a)
print(a<b, a>b, a==b, a!=b)
```

***Задания для самостоятельной работы***

1. Имеется класс, в котором реализуется работа с обыкновенными дробями. Напишите реализацию тех методов, в которых она отсутствует. Проверьте работу этих методов.

```
class Cfract: #обыкновенная дробь

    def __reduce(a, b): #сокращаем дробь
        a1 = abs(a)
        b1 = b
        if a1 == 0:
            return 0, 1
        while a1 != b1:
            if a1 > b1:
                a1 = a1 - b1
            else:
                b1 = b1 - a1
        if a1>1:
            return a//a1, b//a1
        else:
            return a, b

    def __init__(self, w, n, d):
        #w - целая часть дроби, n - числитель, d - знаменатель
        self.__n = abs(w)*d + abs(n) #числитель неправильной дроби
        if (w < 0 and n >= 0) or (w == 0 and n < 0):
           self.__n = -self.__n
        self.__d = d #знаменатель
        self.__n, self.__d = Cfract.__reduce(self.__n, self.__d)

    def __str__(self):   #вызывается функциями str, print и format
        if self.__n == 0:
            return '0'
        elif abs(self.__n) < self.__d:
            return f'{self.__n}/{self.__d}'
        else:
            s = str(abs(self.__n) // self.__d)
            if self.__n < 0:
                s = '-' + s
            if abs(self.__n) % self.__d != 0:
                s += f' {abs(self.__n) % self.__d}/{self.__d}'
            return s

    def __float__(self):
        #преобразование в вещественное число с помощью float()
        pass
```

```
def __add__(self, y):   #сложение x + y
    if type(y) == Cfract:
        a = self.__n * y.__d + y.__n * self.__d
        b = self.__d * y.__d
        a,b = Cfract.__reduce(a, b)
        if a > 0:
            return Cfract(a//b, a % b, b)
        elif a < 0:
            return Cfract(-(abs(a) // b), abs(a) % b, b)
        else:
            return Cfract(0, 0, 1)
    elif type(y) == int:
        return self + Cfract(y, 0, 1)
    else:
        print('недопустимое значение операнда')

def __radd__(self, y):   #сложение y + x
    pass

def __iadd__(self, y):   #сложение и присваивание x +=y
    pass

def __sub__(self, y):   #вычитание x - y
    pass

def __rsub__(self, y):   #вычитание y - x
    pass

def __isub__(self, y):   #вычитание и присваивание x -= y
    pass

def __neg__(self):   #унарный минус -x
    pass

def __mul__(self, y):   #умножение x * y
    pass

def __rmul__(self, y):   #умножение y * x
    pass

def __imul__(self, y):   #умножение и присваивание x *= y
    pass
```

```
    def __truediv__(self, y):   #деление x / y
        pass

    def __rtruediv__(self, y):   #деление y / x
        pass

    def __itruediv__(self, y):   #деление и присваивание x /= y
        pass

    def __abs__(self):  #абсолютное значение
        pass

    def __lt__(self, y):   #x < y
        pass

    def __gt__(self, y):   #x > y
        pass

    def __eq__(self, y):   #x == y
        pass

    def __ne__(self, y):   #x != y
        pass

    def __le__(self, y):   #x <= y
        pass

    def __ge__(self, y):   #x >= y
        pass

a = Cfract(0,-2, 3)
print(a)
b = Cfract(-1,1,3)
print(b)
c=a+b+2
print(c)
```

2. В рассмотренный выше класс Arr (стр. 36) добавим специальные методы, обеспечивающие доступ по индексу и перебор элементов объекта в цикле:

```
class Arr:
    def __init__(self, a,b):
        self.__a = a
        self.__b = b
        self.__val = [0]*(self.b - self.a+1)
        self.__i = 0 #Текущий индекс

    a = property(lambda self:self.__a)
    b = property(lambda self:self.__b)

    ...

    # обеспечиваем доступ по индексу
    def __getitem__(self, i):
        if self.a <= i <= self.b:
            return self.__val[i - self.a]
        else:
            print("Недопустимый индекс", i)

    def __setitem__(self, i, x):
        if self.a <= i <= self.b:
            self.__val[i - self.a] = x
        else:
            print("Недопустимый индекс", i)

    def __len__(self):
        return len(self.__val)

    # обеспечиваем перебор элементов объекта
    def __iter__(self):
        return self
    def __next__(self):
        if self.__i >= len(self.__val):
            self.__i = 0
            raise StopIteration
        else:
            el=self.__val[self.__i]
            self.__i += 1
            return el

#использование класса
Z = Arr(-5,5)
print(Z.a, Z.b)
print(Z)
for i in Z.arr_range():
    Z.put(i,Z.get(i)+i)
print(Z)
```

```
#доступ по индексу
Z[-1]=100
Z[-2]=Z[-1]+10
print(Z)

#перебор элементов
for a in Z:
    print(a, end="_")
```

Реализуйте методы __getitem__, __setitem__, __len__ для класса Ведомость, рассмотренного выше.

Проверьте, что для элементов класса выполняется доступ по ключу.

# Программирование на языке Python

Учебное пособие
по дисциплине «Компьютерный практикум»

для студентов, обучающихся по направлению подготовки
"Бизнес-информатика" и «Прикладная информатика» профиль
«Высокопроизводительные вычисления в цифровой экономике»
(программа подготовки бакалавров
очная и заочная формы обучения)

**Автор:**

**Миронова Ирина Васильевна**, кандидат физ.-мат. наук, доцент, доцент департамента анализа данных, принятия решений и финансовых технологий

Компьютерный набор, верстка И.В. Миронова

Формат 60x90/16. Гарнитура Times New Roman

3,2 п.л. 2019 г. Электронное издание

Вычитка и корректура выполнены автором