Лабораторная работа №5

Функции

Теоретическая часть

1. Функции

Функции в программировании можно представить как изолированный блок кода, обращение к которому в процессе выполнения программы может быть многократным. Это позволяет сократить объем исходного кода - рационально вынести часто повторяющиеся выражения в отдельный блок и затем по мере надобности обращаться к нему.

Рассмотрим в качестве примера вычисление корня нелинейного уравнению с помощью метода деления отрезка пополам.

Пусть на интервале `[a; b]` имеется ровно один корень уравнения `f(x) = 0`. Значит, значение функции в точках `a` и `b` - `f(a)` и `f(b)` - имеют разные знаки. Найдём `f(c),` где `c = (a+b)/2`. Если `f(c)` того же знака, что и `f(a),` значит корень расположен между `c` и `b`, иначе — между `a` и `c.`

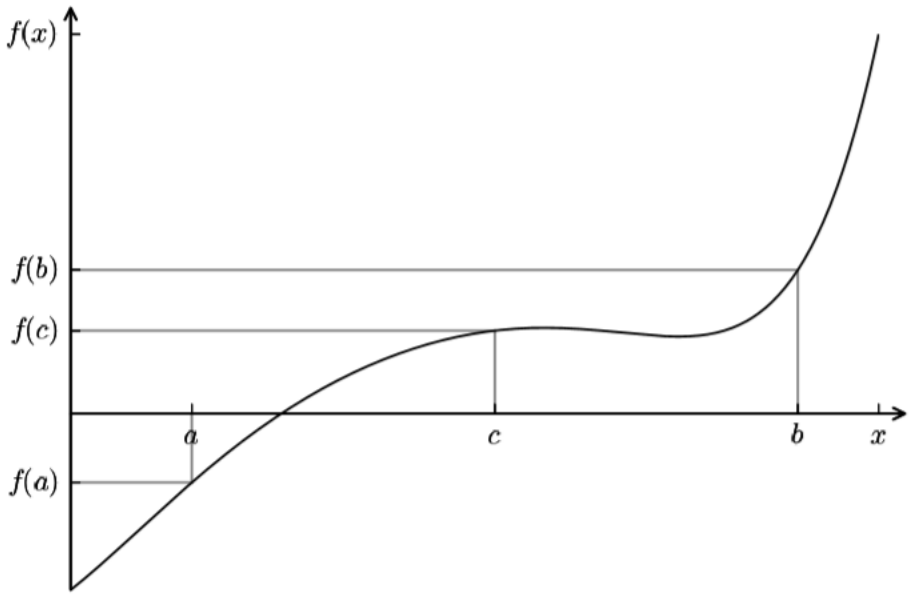


Рисунок 1 – Иллюстрация к методу деления отрезка пополам

Пусть теперь начало нового интервала (будь то a или c в зависимости от того, где находится корень) обозначается `a1` (что означает после первой итерации), а конец, соответственно, `b1`. Исходные начало и конец также будем обозначать `a0` и `b0` для общности. В результате у нас получится снизить неопределённость того, где находится корень, в два раза.

Такой процесс можно повторять сколько угодно раз (при расчёте на компьютере столько, сколько позволяет точность представления данных ЭВМ), последовательно сужая интервал, в котором находится корень. Обычно процесс заканчивают по достижении на `n-ой` итерации интервалом `[an;bn]` величины менее некоторого заранее заданного `ε.`

Итак, напишем программу для вычисления корня нелинейного уравнения `f(x) = x2 - 4:`

```
from math import *
a = -10
b = 10
while (b-a) > 10**(-10):
    c = (a+b)/2
    f_a = a**2-4
    f_b = b**2-4
    f_c = c**2-4
    if f_a*f_c > 0:
        a = c
    else:
          b = c
print ((a+b)/20029
```

Если мы захотим вычислить корень другой нелинейной функции, например, `f(x) = x2 + 4x + 4`, то придётся переделывать выражения сразу в трёх строчках. Кажется логичным выделить наше нелинейное уравнение в отдельный блок программы. Перепишем программу с помощью функции:

```
from math import *
def funkcija(x):
    f = x**2+4*x+4
    return f

a = -10
b = 10
while (b-a) > 10**(-10):
       c = (a+b)/2
       f_a = funkcija(a)
       f_b = funkcija(b)
       f_c = funkcija(c)
       if f_a*f_c > 0:
           a = c
       else:
           b = c
print ((a+b)/2)
```

Программный код подпрограммы описывается единожды перед телом основной программы, затем из основной программы можно им пользоваться многократно. Обращение к этому программному коду из тела основной программы осуществляется по его имени (имени подпрограммы).

Инструкция `def` — это команда языка программирования `Python,` позволяющая создавать функцию; `funkcija` — это имя функции, которое (так же как и имена переменных) может быть почти любым, но желательно осмысленным. После в скобках перечисляются параметры функции. Если их нет, то скобки остаются пустыми. Далее идет двоеточие, обозначающее окончание заголовка функции (аналогично с условиями и циклами). После заголовка с новой строки и с отступом следуют выражения тела функции. В конце тела функции обычно присутствует инструкция `return`, после которой идёт значение или выражение, являющееся результатом работы функции. Именно оно будет подставлено в главной (вызывающей) программе на место функции. Принято говорить, что функция «возвращает значение», в данном случае — результат вычисления выражения `x2 + 4x + 4` в конкретной точке `x.`

Те же самые инструкции можно переписать в виде функции `Bisection,` которой передаются данные из основной программы, и которая возвращается корень уравнения с заданной точностью:

```
from math import *
def function(x):
    f = x**2-4
    return f
def Bisection(a, b, e):
    while (b-a) > e:
        c = (a+b)/2
        f_a = function(a)
        f_b = function(b)
        f_c = function(c)
        if f_a*f_c > 0:
            a = c
        else:
            b = c
    return ((a+b)/2)
A = -10
B = 10
E = 10**(-15)
print (Bisection(A, B, E))
```

Вывод программы:
`-2.0000000000000004`
В `Python` результатом функции может быть только одно значение. Если необходимо в качестве результата выдать значения сразу нескольких переменных, используют **кортеж**. Продемонстрируем это, дополнив программу вычислением количества шагов, за которые достигается значения корня с заданной точностью:

```
from math import *
def function(x):
    f = x**2-4
    return f
def Bisection(a, b, e):
    n = 0
    while (b-a) > e:
```

```
        n = n + 1
        c = (a+b)/2
        f_a = function(a)
        f_b = function(b)
        f_c = function(c)
        if f_a*f_c > 0:
            a = c
        else :
             b = c
    return ((a+b)/2, n)
A = -10
B = 10
E = 10**(-15)
print (Bisection(A, B, E)
```

В качестве результата выдаётся кортеж из двух чисел: значения корня и количества шагов, за которое был найден этот корень:
`(-2.0000000000000004, 55)`
Выражения тела функции выполняются лишь тогда, когда она вызывается в основной ветке программы. Так, например, если функция присутствует в исходном коде, но нигде не вызывается, то содержащиеся в ней инструкции не будут выполнены ни разу.

2. Параметры и аргументы функций.

Часто функция используется для обработки данных, полученных из внешней для нее среды (из основной ветки программы). Данные передаются функции при её вызове в скобках и называются аргументами. Однако, чтобы функция могла «взять» передаваемые ей данные, необходимо при её создании описать параметры (в скобках после имени функции), представляющие собой переменные. Например:

```
def summa(a,b):
    c = a + b
    return c
num1 = int (input ('Введите первое число: '))
num2 = int (input ('Введите второе число: '))
summa(num1 , num2)
```

Здесь `num1` и `num2` суть аргументы функции, `a` и `b` суть параметры функции. В качестве аргументов могут выступать как числовые или строковые константы вроде `12.3, -9` или `'Hello'`, так и переменные или выражения, например, `a+2*i`.

Обязательные и необязательные аргументы
Аргументы функции могут быть обязательными или необязательными. Для всех необязательных аргументов необходимо указать значение по умолчанию. Рассмотрим функцию, возводящую один свой параметр в степень, заданную другим:

```
def Degree(x, a):
    f = x**a
    return  f
```

Если мы попробуем вызвать эту функцию с одним аргументом вместо двух, получим ошибку:

```
>>> Degree(3)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    Degree(3)
TypeError: Degree() missing 1 required positional argument: 'a'
```

Интерпретатор недоволен тем, что параметру с именем a не сопоставили ни одного значения. Если мы хотим, чтобы по умолчанию функция возводила x в квадрат, можно переопределить её следующим образом:

```
Def Degree(x, a=2):
    f = x**a
    return f
```

Тогда наш вызов с одним аргументом станет корректным:

```
>>>Degree(3):
9
```

При этом мы не теряем возможность использовать функцию `Degree` для возведения в произвольную степень:

```
>>>Degree(3, 4):
81
```

В принципе, все параметры функции могут иметь значение по умолчанию, хотя часто это лишено смысла. Параметры, значение по умолчанию для которых не задано, называются обязательными. Важно помнить, что *обязательный параметр не может стоять после параметра, имеющего значение по умолчанию*. Попытка написать функцию, не удовлетворяющую этому требованию, приведёт к синтаксической ошибке:

```
>>>def Degree(x=2, a):
   f = x**a
   return f
SyntaxError: non-default argument follows default argument
```

Именованные аргументы

Бывает, что у функции много или очень много параметров. Или вы забыли порядок, в котором они расположены, но помните их смысл. Тогда можно обратиться к функции, используя имена параметров как ключи. Пусть у нас есть следующая функция:

```
def Clothing (Dress , ColorDress , Shoes , ColorShoes):
    S = 'Сегодня я надену ' + ColorDress + ' ' \
      + Dress + ' и '+ ColorShoes + ' ' + Shoes
    return S
```

Теперь вызовем нашу функцию, с аргументами не по порядку:

```
print (Clothing(ColorDress='красное', Dress='платье',
                ColorShoes='чёрные', Shoes='туфли'))
```

Будет выведено:

```
Сегодня я надену красное платье и чёрные туфли
```

Как видим, результат получился верный, хотя аргументы перечислены не в том порядке, что при определении функции. Это происходит потому, что мы явно указали, какие параметры соответствуют каким аргументам.
Следует отметить, что часто программисты на `Python` путают параметры со значением по умолчанию и вызов функции с именованными аргументами. Это происходит оттого, что синтаксически они плохо различимы. Однако важно знать, что наличие значения по умолчанию не обязывает вас использовать имя параметра при обращении к нему. Также и отсутствие значения по умолчанию не означает, что к параметру нельзя обращаться по имени.

Например, для описанной выше функции `Degree` все следующие вызовы будут корректными и приведут к одинаковому результату:

```
>>> Degree(3)
9
>>> Degree(3, 2)
9
>>> Degree(3, a=2)
9
>>> Degree(x=3, a=2)
9
>>> Degree(a=2, x=3)
9
```

Нельзя ставить обязательные аргументы после необязательных, если имена параметров не указаны:

```
>>> Degree(a=2, 3)
SyntaxError: non-keyword arg after keyword arg
```

Произвольное количество аргументов

Иногда возникает ситуация, когда вы заранее не знаете, какое количество аргументов будет необходимо принять функции. Для такого случая есть специальный синтаксис: все параметры обозначаются одним именем (обычно используется имя `args`) и перед ним ставится звёздочка `*`. Например:

```
def unknown(*args):
    for argument in args:
    print (argument) unknown('Что ', 'происходит', '?') unknown('Не
знаю!')
```

Вывод программы:

```
Что происходит
?
Не знаю!
```

При этом тип значения `args` — кортеж, содержащий все переданные аргументы по порядку.

Локальные и глобальные переменные

Если записать приведённую ниже функцию, и затем попробовать вывести значения переменных, то обнаружится, что некоторые из них почему-то не существуют:

```
def mathem(a, b):
    a = a/2
    b = b+10
    print (a+b)
>>> num1 = 100
>>> num2 = 12
>>> mathem(num1 , num2)
72.0
>>> num1
>>>100
>>> num2
12
>>> a
Traceback (most recent call last):
File "<pyshell#10>", line 1, i n <module >
a
NameError: name 'a' is not defined
>>> b
Traceback (most recent call last):
File "<pyshell#11>", line 1, in <module >
b
NameError: name 'b' i s n o t defined
>>>
```

Переменные `num1` и `num2` не изменили своих первоначальных значений. Дело в том, что в функцию передаются копии значений. Прежние значения из основной ветки программы остались связанны с их переменными.

А вот переменных a и b, оказывается, нет и в помине (ошибка name 'b' is not defined переводится как "переменная b не определена"). Эти переменные существуют лишь в момент выполнения функции и называются локальными. В противовес им, переменные num1 и num2 видны не только во внешней ветке, но и внутри функции:

```
>>>def mathem2():
   print (num1+num2)
>>> mathem2()
112
>>>
```

Переменные, определённые в основной ветке программы, являются **глобальными**. Итак, в Python две базовых области видимости переменных:

1) глобальные переменные,
2) локальные переменные.

Переменные, объявленные внутри тела функции, имеют локальную область видимости, те, что объявлены вне какой-либо функции, имеют глобальную область видимости. Это означает, что доступ к локальным переменным имеют только те функции, в которых они были объявлены, в то время как доступ к глобальным переменным можно получить по всей программе в любой функции.

Например:

```
Place = 'Солнечная система' # Глобальная переменная
def global_Position():
    print (Place)

def local_Position():
    Place = 'Земля' # Локальная переменная
    Print (Place)

S = input()
if S == 'система':
    global_Position()
else:
    local_Position()
```

Вывод программы при двух последовательных запусках:

```
система
Солнечная система
>>>
========== RESTART: /home/paelius/Python/1.py ==========
планета
Земля
```

Важно помнить, что для того чтобы получить доступ к глобальной переменной на чтение, достаточно лишь указать её имя. Однако если перед нами стоит задача

изменить глобальную переменную внутри функции, необходимо использовать ключевое слово `global`. Например:

```
Number = 10
def change():
    global Number
    Number = 20
print (Number)
change()
print (Number)
```

Вывод программы:

```
10
20
```

Если забыть написать строчку `global Number,` то интерпретатор выдаст следующее:

```
10
10
```

3. Лямбда-функции

Лямбда-функция в `Python` — это функция без имени, о которой известно только количество аргументов и формула для вычисления итогового значения, причём формула должна записываться единым выражением.

`Python` разрешает создание анонимных функций (например, функции, которые не связаны с именем) во время выполнения, используя конструкцию “lambda”. Пример лямбда-функции, складывающей три числа:

```
lambda x,y,z: x+y+z
```

Ниже приведен пример использования обычной функции и лямбда-функции для возведения в квадрат переменной `x`.

Без применения лямбда-функции:

```
def f(x):
    return x**2
    print f(2)
>>>4
```

С применения лямбда-функции:

```
g = lambda x: x**2
```

```
print g(2)
>>>4
```

4. Функция map() в Python:

В Python функция map принимает два аргумента: функцию и аргумент составного типа данных, например, список. map применяет к каждому элементу списка переданную функцию. Например, вы прочитали из файла список чисел, изначально все эти числа имеют строковый тип данных, чтобы работать с ними - нужно превратить их в целое число:

```
old_list = ['1', '2', '3', '4', '5', '6', '7']
new_list = []
for item in old_list:
    new_list.append(int(item))
print (new_list)

>>> [1, 2, 3, 4, 5, 6, 7]
```

Тот же эффект мы можем получить, применив функцию map:

```
old_list = ['1', '2', '3', '4', '5', '6', '7']
new_list = list(map(int, old_list))
print (new_list)

>>>[1, 2, 3, 4, 5, 6, 7]
```

Как видите такой способ занимает меньше строк, более читабелен и выполняется быстрее. map также работает и с функциями созданными пользователем:

```
def miles_to_kilometers(num_miles):
    # Converts miles to the kilometers
    return num_miles * 1.6

mile_distances = [1.0, 6.5, 17.4, 2.4, 9]
kilometer_distances = list(map(miles_to_kilometers, mile_distances))
print (kilometer_distances)

>>>[1.6, 10.4, 27.84, 3.84, 14.4]
```

А теперь то же самое, только используя lambda выражение:

```
mile_distances = [1.0, 6.5, 17.4, 2.4, 9]
kilometer_distances = list(map(lambda x: x * 1.6, mile_distances))

print (kilometer_distances)

>>>[1.6, 10.4, 27.84, 3.84, 14.4]
```

Функция map может быть так же применена для нескольких списков, в таком случае функция-аргумент должна принимать количество аргументов, соответствующее количеству списков:

```
l1 = [1,2,3]
l2 = [4,5,6]

new_list = list(map(lambda x,y: x + y, l1, l2))
print (new_list)

>>>[5, 7, 9]
```

Если же количество элементов в списках совпадать не будет, то выполнение закончится на минимальном списке:

```
l1 = [1,2,3]
l2 = [4,5]
new_list = list(map(lambda x,y:  + y, l1, l2))
print (new_list)
>>>[5,7]
```

5. Функция filter() в Python:

Функция filter предлагает элегантный вариант фильтрации элементов последовательности. Принимает в качестве аргументов функцию и последовательность, которую необходимо отфильтровать:

```
mixed = ['мак', 'просо', 'мак', 'мак', 'просо', 'мак', 'просо', 'просо',
'просо', 'мак']
zolushka = list(filter(lambda x: x == 'мак', mixed))

print (zolushka)
>>>['мак', 'мак', 'мак', 'мак', 'мак']
```

Обратите внимание, что функция, передаваемая в filter должна возвращать значение True / False, чтобы элементы корректно отфильтровались.

6. Функция reduce() в Python:

Функция reduce принимает 2 аргумента: функцию и последовательность. reduce() последовательно применяет функцию-аргумент к элементам списка, возвращает единичное значение. Обратите внимание в Python 2.x функция reduce доступна как встроенная, в то время, как в Python 3 она была перемещена в модуль functools.

Вычисление суммы всех элементов списка при помощи reduce:

```
from functools import reduce
items = [1,2,3,4,5]
sum_all = reduce(lambda x,y: x + y, items)
```

```
print (sum_all)

>>>15
```

Вычисление наибольшего элемента в списке при помощи reduce:

```
from functools import 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)
>>>32
```

7. Функция zip() в Python:

Функция zip объединяет в кортежи элементы из последовательностей переданных в качестве аргументов.

```
a = [1,2,3]
b = "xyz"
c = (None, True)

res = list(zip(a, b, c))
print (res)

>>>[(1, 'x', None), (2, 'y', True)]
```

Обратите внимание, что zip прекращает выполнение, как только достигнут конец самого короткого списка.

Задания

1. Реализовать примеры определения функций, листинг которых приведен в теоретической части.

2. Вычислить значение выражения f(x) в точке x, значение которой ввести с клавиатуры. Выражение задать функцией. Варианты приведены в таблице 1 (номер варианта согласовать с преподавателем).

Таблица 1.

Вариант	Выражение
1	$y = x^6 - 2x - 7$
2	$y = x^4 - 5x + 1$
3	$y = x^5 + 1{,}025x - 3{,}116$
4	$y = (x-1)^2 - \sin 2x$
5	$y = x^3 - 12x - 8$

Вариант	Выражение
6	$y = 2^x + x^2 - 1{,}15$
7	$y = x^3 \sin x - 0{,}985x - 0{,}991$
8	$y = 3^{-x} - x^2 + 1$
9	$y = x^3 - 3x + 3$
10	$y = x^3 - \cos \pi x$
11	$y = \sqrt{x} - \cos\ 0{,}387x$
12	$y = (x-1)^2 - \sin 2x$
13	$y = (x-1)^2$
14	$y = 2^x + x^2$
15	$y = x^3 - 12x$

3. Вычислить значение выражения f(x, y) в точке (x, y) значение которой ввести с клавиатуры. Выражение задать функцией. Варианты приведены в таблице 2 (номер варианта согласовать с преподавателем).

Таблица 2.

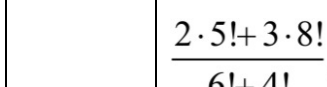
Вариант	Выражение
1	$z_1 = \sin(x+1) - y - 1{,}2$ $z_2 = 2x + \cos y - 2$
2	$z_1 = \sin y + 2x - 2$ $z_2 = y + \cos(x-1) - 0{,}7$
3	$z_1 = \cos(x-1) + y - 0{,}5$ $z_2 = x - \cos y - 3$
4	$z_1 = \cos y + x - 1{,}5$ $z_2 = 2y - \sin(x - 0{,}5) - 1$
5	$z_1 = \sin x + 2y - 2$ $z_2 = \cos(y-1) + x - 0{,}7$
6	$z_1 = \sin(y + 0{,}5) - x - 1$ $z_2 = y + \cos(x-2)$
7	$z_1 = \cos x + y - 1{,}5$ $z_2 = 2x - \sin(y - 0{,}5) - 1$
8	$z_1 = \cos(y + 0{,}5) + x - 0{,}8$ $z_2 = -2y + \sin x - 1{,}6$

Вариант	Выражение
9	$z_1 = \sin(x + 0{,}5) - y - 1$ $z_2 = x + \cos(y - 2)$
10	$z_1 = \sin(y - 1) + x - 1{,}3$ $z_2 = y - \sin(x + 1) - 0{,}8$
11	$z_1 = \cos(x + 0{,}5) + y - 0{,}8$ $z_2 = -2x + \sin y - 1{,}6$
12	$z_1 = 2x - \cos(y + 1)$ $z_2 = y + \sin x + 0{,}4$
13	$z_1 = \sin(x - 1) + y - 1{,}3$ $z_2 = x - \sin(y + 1) - 0{,}8$
14	$z_1 = \cos(y + 0{,}5) - x - 2$ $z_2 = \sin x - 2y - 1$
15	$z_1 = -\cos(x + 1) + 2y$ $z_2 = x + \sin y + 0{,}4$

4. Решить задачу, определив дополнительно функцию. Варианты приведены в таблице 3 (номер варианта согласовать с преподавателем)

Таблица 3.

Вариант	Выражение
1	Даны две последовательности целых чисел: a1, a2, ..., a8 и b1, b2, ..., b8. Найти количество четных чисел в первой из них и количество нечетных во второй. (Определить функцию, позволяющую распознавать четные числа.)
2	Даны натуральное число n и целые числа a1, a2, ..., an. Найти количество чисел ai (i = 1, 2, ..., n), являющихся полными квадратами. (Определить функцию, позволяющую распознавать полные квадраты.)
3	Даны натуральное число n и целые числа a1, a2, ..., an. Найти количество чисел ai (i 1, 2, ..., n), являющихся степенями пятерки. (Определить функцию, позволяющую распознавать степени пятерки.)
4	Найти все трехзначные простые числа. (Определить функцию, позволяющую распознавать простые числа.)
5	Два простых числа называются "близнецами", если они отличаются друг от друга на 2 (таковы, например, числа 41 и 43). Напечатать все пары чисел-"близнецов", не превышающих число 200. (Определить функцию, позволяющую распознавать простые числа.)

Вариант	Выражение
6	Найти значение выражения  где n! означает факториал числа n (n! = 1 2 ... n). (Определить функцию для расчета факториала натурального числа.)
7	Даны два натуральных числа. Выяснить, в каком из них сумма цифр больше. (Определить функцию для расчета суммы цифр натурального числа.)
8	Даны два натуральных числа. Выяснить, в каком из них больше цифр. (Определить функцию для расчета количества цифр натурального числа.)
9	Получить все шестизначные счастливые номера. Счастливым называют такое шестизначное число, в котором сумма его первых трех цифр равна сумме его последних трех цифр. (Определить функцию для расчета суммы цифр трехзначного числа.)
10	Даны два натуральных числа. Выяснить, является ли хоть одно из них палиндромом ("перевертышем"), т. е. таким числом, десятичная запись которого читается одинаково слева направо и справа налево. (Определить функцию, позволяющую распознавать числа-палиндромы.)
11	Даны шесть различных чисел. Определить максимальное из них. (Определить функцию, находящую максимум из двух различных чисел.)
12	Даны натуральные числа a и b. Найти их наименьшее общее кратное. (Определить функцию для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида.)
13	Даны натуральные числа a и b, обозначающие соответственно числитель и знаменатель дроби. Сократить дробь, т. е. найти такие натуральные числа p и q, не имеющие общих делителей, что p/q = a/b. (Определить функцию для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида.)
14	Найти наибольший общий делитель трех натуральных чисел, имея в виду, что НОД(a, b, c) = НОД(НОД(a, b), c). (Определить функцию для расчета наибольшего общего делителя двух натуральных чисел, используя алгоритм Евклида.)
15	Составить программу для нахождения общего количества заданной буквы в трех заданных предложениях. (Определить функцию для расчета количества некоторой буквы в предложении.)

5. Дан список целых чисел. Написать функцию, возвращающую... (продолжение условия по варианту в таблице 4). Реализовать двумя способами: 1- функция принимает на вход список, 2- функция принимает на вход переменное число параметров.

Таблица 4.

Вариант	Выражение
1	минимальное и максимальное значения
2	сумму и среднее значение
3	среднее значение и стандартное отклонение
4	минимальное значение и максимальное по модулю значение
5	минимальное из положительных значений и стандартное отклонение
6	максимальное значение и количество элементов
7	максимальное, минимальное и среднее значения
8	сумму абсолютных значений и максимальное отрицательное значение
9	минимальное четное значение и сумму четных значений
10	сумму квадратов элементов и максимальное значение
11	сумму кубов элементов и минимальное значение
12	число элементов и среднее значение
13	стандартное отклонение и минимальный элемент
14	сумма квадратов элементов и среднее значение
15	стандартное отклонение и максимальное по модулю значение

6. Написать рекурсивную процедуру перевода натурального числа из десятичной системы счисления в N-ричную. Значение N в основной программе вводится с клавиатуры (2<=N<=16).

7. Написать рекурсивную функцию, определяющую, является ли заданное натуральное число простым (простым называется натуральное число, большее 1, не имеющее других делителей, кроме единицы и самого себя)

8. Дан список вещественных чисел. Используя lambda-функцию и функцию map выведите список остатков от деления этих чисел на 7.

9. Дан список имен детей: [‘катя’, ’маша’, ’таня’, ’саша’]. Сделайте так, чтобы имена начинались с заглавной буквы.

10. Попробуйте переписать следующий код через map и lambda-функцию. Он принимает список реальных имён и заменяет их прозвищами, используя надёжный метод.

```
names = ['Маша', 'Петя', 'Вася']

for i in range(len(names)):
    names[i] = hash(names[i])

print(names)
# => [6306819796133686941, 8135353348168144921, -1228887169324443034]
```

11. Следующий пример считает, как часто слово «сети» встречается в списке строк:

```
sentences = ['научиться плести рыболовные сети',
             'обучать нейронные сети',
             'паук ловит в сети мух']

cap_count = 0
for sentence in sentences:
    cap_count += sentence.count('сети')

print(cap_count)
# => 3
```

Перепишите этот же код с использованием reduce и lambda-функции.

12. Дан список натуральных чисел. Вывести список чисел, кратных 7, используя lambda-функцию и filter.

13. Дано три списка: 1- имена абитуриентов, 2 – баллы за ЕГЭ по математике, 3- баллы за ЕГЭ по русскому, 4 – баллы за ЕГЭ по информатике. Составить список в формате: [('Василий Акимович Кузнецов',85,42,65), ('Петр Николаевич Чириков',79,49,78), ...].