Лабораторная работа №1
Пользовательские функции и основы функционального программирования в Python

Цель: настоящей работы состоит в том, чтобы изучить Пользовательские функции и основы функционального программирования в Python

.

Задачи:

Выполнение практической работы предполагает решение ***следующих задач***:

1. Изучение процесса создания пользовательских функций.
2. Применение глобальных и локальных переменных.
3. Использование переменное число параметров в функции.
4. Использование функции в качестве параметров.
5. Использование анонимных функций.
6. Рассмотрение встроенных функций высшего порядка
7. Подготовка отчета, содержащего минимальный объем информации по каждому этапу выполнения работы.

Последовательность выполнения работы :

Использован материал И.В. Мироновой, Программирование на языке Python. Часть 2: Учебное пособие - М.: Финансовый университет, департамент анализа данных, принятия решений и финансовых технологий, 2019. -51 с.

Создание и вызов функции

Функция – фрагмент программы, у которого есть имя. Для выполнения этого фрагмента в любом месте программы достаточно указатьего 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 естьс 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 значениев 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 returnчерез 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 указыватьсяв 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 вызовефункции 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 втом числе в функции с произвольным количеством параметров, например, print или max:

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

```
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)
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 глобальнойпеременной из функции, нужно объявить ее внутри функции с помощью служебного слова global.

```
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)
```

Пример использования глобальной и локальной переменной

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 илисоздавайте такую копию внутри функции.

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 двумязвездочками.

```
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)
```

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.

Можно передать ссылку на функцию в качестве параметра другой функции.

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

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

```
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 порядкатребуется совсем простая функция. Причем нередко такая функция нужна впрограмме 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, принимающуюуказанный 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.

Пример анонимной функции с проверкой условия:

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

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))
```

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))
```

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 специальнойлитературе.

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 результатомфункции 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 двухисходных 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 насоответствие значению True. Как и в случае с map() функция

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

- удалим «пустые» значения из списка (нули, пустые строки, пустыесписки):
- найдем все числа от 1 до 99, которые при делении на 15 дают 2 востатке:

```
L = [1, 0, None, [], ['**'], '', '++']
L_new = list(filter(None, L))
print(L_new)  #результат [1, ['**'], '++']
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 элементови накапливает результат. Функция f должна иметь два параметра. Припервом обращении к f ее параметрами являются два первых элемента последовательности. При последующих обращениях в качествепервого 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 жесмысл. Отличие метода от функции состоит в том, что он применяетсятолько для списков, и он не создает новый список, а сортирует имеющийся. Примеры:

- отсортируем список строк с использованием различных значенийпараметров:

```
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:
- 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]
```

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

Требования к результатам выполнения работы

Условия успешной сдачи лабораторной работы:

1. Подготовлено пошаговое описание решений представленных задач.
2. Разработана программная реализация представленных задач.
3. Подготовлено краткое описание разработанного программного кода.
4. Программная реализация представленных примеров выложены в личный репозиторий на GitHub.

Литература

Копырин, А. С. Программирование на Python : учебное пособие / А. С. Копырин, Т. Л. Салова. — Сочи : СГУ, 2018. — 48 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/147665 (дата обращения: 15.07.2021). — Режим доступа: для авториз. пользователей.