# Практическая работа № 14

## *Организация функций в Python*

***Время –*** 1 час

***Цель работы:*** получение практических навыков программирования в создании и использовании функций.

### Общие положения

***Функция*** – особым образом сгруппированный набор команд, которые выполняются последовательно, но воспринимаются как единое целое. При этом функция может возвращать (или не возвращать) свой результат.

**Создание функции**

Для того чтобы использовать какую-нибудь собственную функцию, вначале необходимо ее объявить (создать).

Блок функции начинается с ключевого слова **def**, после которого следуют название функции и круглые скобки ( ). Любые аргументы, которые принимает функция, должны находиться внутри этих скобок. После скобок идет двоеточие и с новой строки с отступом начинается тело функции.

**def <имя функции>(аргументы):**
    **<тело функции>**

Пример функции в Python:

```
def my_function(argument):
   print argument
```

После функции до кода, который находится вне функции, необходимо делать ***отступ в две пустые строки*** для повышения читаемости кода. Если у вас есть несколько функций в одном файле, между кодом одной и сигнатурой другой функции тоже надо оставлять две пустые строки.

**Вызов функции**

После создания функции, ее можно исполнять, вызывая из другой функции или напрямую из оболочки Python. Для вызова функции следует ввести ее имя и добавить в скобках аргументы.

**<имя функции>(аргументы)**

Обращение к ранее объявленной функции с целью выполнения ее команд называется ***вызовом***.

Имена функций должны состоять из маленьких букв, а слова разделяться символами подчеркивания. Аргументы (параметры) могут изменять поведение функции.

**Аргументы функции в Python**

Вызывая функцию, мы можем передавать ей следующие типы аргументов:

1. Обязательные аргументы (Required arguments)
2. Аргументы-ключевые слова (Keyword argument)
3. Аргументы по умолчанию (Default argument)
4. Аргументы произвольной длины (Variable-length argumens)

**Обязательные аргументы функции.** Если при создании функции мы указали количество передаваемых ей аргументов и их порядок, то и вызывать ее мы должны с тем же количеством аргументов, заданных в нужном порядке.

Например:

```
def bigger(a,b):                # В описании функции указано, что она принимает 2 аргумента
    if a > b:
        print a
    else:
        print b

bigger(5,6)     # Корректное использование функции

# Некорректное использование функции
bigger()
bigger(3)
bigger(12,7,3)
```

**Аргументы - ключевые слова** используются при вызове функции. Благодаря ключевым аргументам, вы можете задавать произвольный (то есть не такой, каким он описан при создании функции) порядок аргументов.

Например:

```
def person(name, age):
    print name, "is", age, "years old"
```

Хотя в описании функции первым аргументом идет имя, мы можем вызвать функцию вот так

```
person(age=23, name="John")
```

**Аргументы, заданные по умолчанию.** Аргумент по умолчанию, это аргумент, значение для которого задано изначально, при создании функции.

Например:

```
def space(planet_name, center="Star"):
    print planet_name, "is orbiting a", center

space("Mars")                   # В результате получим: Mars is orbiting a Star
space("Mars", "Black Hole")     # В результате получим: Mars is orbiting a Black Hole
```

**Аргументы произвольной длины.** Иногда возникает ситуация, когда вы заранее не знаете, какое количество аргументов будет необходимо принять функции. В этом случае следует использовать аргументы произвольной длины. Они задаются произвольным именем переменной, перед которой ставится звездочка (*).

Например:

```
def  unknown(*args):
    for argument in args:
        print argument

unknown("hello","world")                # напечатает оба слова, каждое с новой строки
unknown(1,2,3,4,5)                      # напечатает все числа, каждое с новой строки
```

*unknown()* *# ничего не выведет*

**Ключевое слово return**

Выражение **return** прекращает выполнение функции и возвращает указанное после выражения значение. Выражение **return** без аргументов это то же самое, что и выражение **return None**. Соответственно, теперь становится возможным, например, присваивать результат выполнения функции какой-либо переменной.

Например:

```
def bigger(a,b):
    if a > b:
        return a          # Если a больше чем b, то возвращаем a и прекращаем выполнение функции
    return b      # Незачем использовать else. Если мы дошли до этой строки, то b, точно не меньше чем a

num = bigger(23,42)          # присваиваем результат функции bigger переменной num
```

**Пустая функция.** Чтобы создать пустую функцию, нужно в её теле использовать оператор заглушки **pass**. Тогда она будет существовать и не выполнять никаких действий.

Такие функции могут использоваться для различных специфичных задач, например, при работе с классами, асинхронной отправкой форм.

```
def example():
    pass
```

**Область видимости переменных**

В Python две базовых области видимости переменных:

- ***локальные переменные*** - переменные, создаваемые внутри функций, недоступны извне и существуют только внутри функции.
- ***глобальные переменные*** – переменные, создаваемые вне функции, могут быть доступны из функций.

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:

```
age = 44              # глобальная переменная age

def info():
    print age         # печатаем глобальную переменную age

def local_info():
    age = 22          # создаем локальную переменную age
    print age

info()                # напечатает 44
local_info()          # напечатает 22
```

Важно помнить, что для того чтобы получить доступ к глобальной переменной, достаточно лишь указать ее имя. Однако, если перед нами стоит задача изменить глобальную переменную внутри функции - необходимо использовать ключевое слово **global**.

Например:

```
age = 13            # глобальная переменная age

def get_older():    # функция изменяющая глобальную переменную
    global age
    age += 1

print age           # напечатает 13
get_older()         # увеличиваем age на 1
print age           # напечатает 14
```

## Порядок выполнения работы

**Задание 1.** Напишите две функции S(r) и l(r), принимающие в качестве аргумента радиус окружности и возвращающие площадь круга и длину этой окружности соответственно. Затем напишите функцию krug(), которая спрашивает у пользователя радиус окружности, а затем при помощи функций S(r) и l(r) выводит на экран площадь круга и длину окружности, разделённые пробелом.

**Задание 2.** Составить программу, согласно полученному варианту задания. Ввод данных сопровождать соответствующими запросами, а вывод - наименованиями выводимых переменных.

| Вариант | Задание |
|---|---|
| 1 | Напишите функцию, которая будет принимать в качестве аргумента список чисел, и возвращать список, в котором оставлены только четные числа больше 10. В программе обеспечить ввод списка с клавиатуры. |
| 2 | Напишите функцию, которая принимает строку в качестве аргумента и подсчитывает количество пробелов в строке, если число пробелов является четным, то печатается: «Четное число», в противном случае выводится сообщение «Нечетное число». В программе обеспечить ввод строки с клавиатуры. |
| 3 | Напишите функцию, которая принимает список целых чисел в качестве аргумента и печатает среднее значение элементов этого списка. В случае, когда список пустой, функция должна напечатать ноль. В программе обеспечить ввод списка с клавиатуры. |
| 4 | Напишите функцию, которая принимает список вещественных чисел в качестве аргумента и печатает сумму и произведение элементов этого списка. В случае, когда список пустой, функция должна напечатать ноль. В программе обеспечить ввод списка с клавиатуры. |
| 5 | Напишите функцию, которая принимает в качестве аргумента список из трех чисел (длин отрезков) и определяет, можно ли из этих отрезков составить треугольник. Функция должна печатать «Это треугольник», если составить треугольник можно, и «Это не треугольник», если нельзя. |

| | |
|---|---|
| | (Чтобы из трех отрезков можно было составить треугольник, необходимо и достаточно, чтобы сумма длин любых двух отрезков была строго больше третьего.) В программе обеспечить ввод списка с клавиатуры. |
| 6 | Напишите функцию, которая принимает фразу в качестве аргумента и печатает эту фразу, если она есть в списке, иначе добавляет с список. В программе обеспечить ввод списка с клавиатуры. |
| 7 | Напишите функцию, которая в качестве аргумента принимает строку и рассчитывает количество букв верхнего и нижнего регистра. В программе обеспечить ввод строки с клавиатуры. |
| 8 | Напишите функцию, которая принимает в качестве аргумента список, состоящий из номера дня недели и языка (русский или английский), а возвращает его название. В программе обеспечить ввод списка с клавиатуры. |
| 9 | Напишите функцию, которая принимает два аргумента: строку, состоящую из целых чисел, написанных через точку с запятой, и список. Функция должна извлечь из строки числа и добавить их в конец списка. В программе обеспечить ввод строки и списка с клавиатуры. |
| 10 | Напишите функцию, которая принимает в качестве аргумента целое число (день месяца), лежащее в диапазоне 1 - 30, а возвращает его название. Дни недели пронумерованы следующим образом: 1 — понедельник, 2 — вторник, ..., 6 — суббота, 7 — воскресенье. Известно, что в этом месяце 1 число было пятницей. В программе обеспечить ввод числа клавиатуры. |

**Задание 3.** Даны два списка: первый содержит произвольный список целых чисел, второй – три числа **a, b, c**. Напишите функцию, которая принимает их в качестве аргументов, и выводит элементы, которые одновременно больше **a**, меньше **b** и делятся на **c** без остатка, а все остальные элементы списка необходимо просуммировать и вывести конечный результат.

## Содержание отчета

1. Постановка задачи.
2. Текст программы.
3. Результаты выполнения программы.