Лекция №13.
Тема: **Функции.**
Назначение подпрограмм (ПП). Особенности использования ПП в Python. Назначение, описание и вызов функций. Область действия переменных. Виды параметров ПП. Примеры использования функций.
Ключевые слова: подпрограмма, функция, объявление функции, формальный параметр, фактический параметр, вызов функции.
Keywords: subroutine, function, function definitions, parameter, argument, function call.

1 Назначение ПП

ПП – поименованная логически завершенная последовательность операторов языка, которая может быть вызвана с различными исходными данными необходимое количество раз (см. https://en.wikipedia.org/wiki/Subroutine). Обработка данных, которая выполняется в ПП, заменяется в тексте программы вызовом ПП. Так как ПП разрабатываются для различных случаев применения, то при вызове ПП требуется настройка конкретного вызова через механизм передачи параметров – при вызове указываются фактические параметры (аргументы) ПП и перед обработкой формальные параметры ПП получают значения фактических параметров.

Выделение логической структуры программы с помощью создания ПП, как самостоятельных этапов обработки данных, позволяет скрыть подробности обработки данных. Следствиями использования ПП являются:

- многократное использование программного кода;
- уменьшение избыточности программного кода;
- повышение читабельности программного кода;
- необходимый уровень тестируемости программного кода с ПП;
- увеличение, в общем случае, времени выполнения программного кода за счет времени на вызов ПП;
- возможность создания библиотек ПП.

2 Особенности использования подпрограмм в Python

В языке Python ПП реализуются с помощью функций. Назначение функции – определение некоторого алгоритма обработки данных, результатом которого является получение определенного значения. Функции часто называют алгоритмически определяемыми значениями.

Формальное определение функции:

```
funcdef         ::=  [decorators]"def" funcname"("[parameter_list] ")"
                     ["->" expression] ":" suite
decorators      ::=  decorator+
decorator       ::=  "@" dotted_name ["(" [parameter_list [","]] ")"]
                     NEWLINE
dotted_name     ::=  identifier ("." identifier)*
parameter_list  ::=  (defparameter ",")*
                     | "*" [parameter] ("," defparameter)*
                     ["," "**" parameter] | "**" parameter
                     | defparameter [","] )
```

```
parameter       ::=  identifier [":" expression]
defparameter    ::=  parameter ["=" expression]
```

```
funcname        ::=  identifier
```

Рассмотрим особенности использования функций в языке Python:

1. Описание функции содержит ключевое слово def, имя функции (funcname), список формальных параметров ("("[parameter_list] ")"), за которым следует ":" и тело функции (suite). Так как функция является составным оператором, то операторы тела функции оформляются с одинаковыми отступами от левого края.

Формальный параметр – параметр, указываемый при объявлении функции (см. https://en.wikipedia.org/wiki/Parameter_(computer_programming)).

Фактический параметр (аргумент) – параметр, указываемый при вызове функции.

```
# Функция расчета площади треугольника через основание и высоту.
# a, h - формальные параметры функции.
def TriangleArea(a, h):
    return 0.5 * a *h

a = float(input('Основание треугольника = '))
b = float(input('Высота треугольника = '))
# Вызов функции.
# a, b - аргументы функции
S = TriangleArea(a, b)
```

2. Вызов функции может встретиться везде, где возможно написание выражения (например, элемент выражения в правой части оператора присваивания, аргумент при вызове других функций).

3. Python поддерживает документирование программного кода. Документирование функций осуществляется с помощью строки документации (docstring). Docstring является необязательным оператором функции; располагается вначале тела функции в виде многострочной строки. Общепринятое правило оформления docstring: первая строка начинается с заглавной буквы и заканчивается точкой. Вторая строка оставляется пустой, а подробное описание начинается с третьей. Доступ к docstring функции можно получить с помощью атрибута функции __doc__ или с помощью функции help().

```
def TriangleArea(a, h):
    '''Функция расчета площади треугольника.

       Площадь определяется через основание и высоту.'''
    return 0.5 * a *h
print(TriangleArea.__doc__)
help(TriangleArea)
>>> print(TriangleArea.__doc__)
Функция расчета площади треугольника.
```

```
        Площадь определяется через основание и высоту.
>>> help(TriangleArea)
Help on function TriangleArea in module __main__:

TriangleArea(a, h)
    Функция расчета площади треугольника.

    Площадь определяется через основание и высоту.
```

4. Оператор return завершает выполнение функции и возвращает некоторое значение.

```
return_stmt ::=  "return" [expression_list]
```

Если оператор return вызван без аргумента (expression_list) или вообще отсутствует в теле функции, то функция возвращает None.

```
def TriangleArea(a, h):
    '''Функция расчета площади треугольника.

       Площадь определяется через основание и высоту.'''

    if a > 0 and h > 0:
        return 0.5 * a *h

>>> print(TriangleArea(3, 8))
12.0
>>> print(TriangleArea(-3, 8))
None
```

Функции-генераторы для передачи какого-либо значения используют оператор yield (в данной лекции функции-генераторы не рассматриваются).

5. Допускается вложенность описания функций.

```
def TriangleArea():
    def TriangleAreaVar1(a, h):
        """Функция расчета площади треугольника.

        Площадь определяется через основание и высоту. """
        return 0.5 * a *h

    def TriangleAreaVar2(a, b, c):
        """Функция расчета площади треугольника.

        Площадь определяется по формуле Герона."""
        p = (a + b + c) / 2
        if p != 0 and p != a and p != b and p != c:
            return (p * (p - a) * (p - b) * (p - c)) ** 0.5
    var   =   int(input('Выберите   Формулу   расчета   площади
треугольника \n(1 - по основанию и высоте; 2 - по формуле Герона)\n'))
    if var == 1:
```

```
            a = float(input('Основание треугольника = '))
            b = float(input('Высота треугольника = '))
            if a > 0 and b > 0:
                return TriangleAreaVar1(a, b)
        if var == 2:
            a = float(input('A = '))
            b = float(input('B = '))
            c = float(input('C = '))
            if a > 0 and b > 0 and c > 0:
                return TriangleAreaVar2(a, b, c)

    print('S =', TriangleArea())
```

6. Допускается вложенность вызова функций - рекурсия (функция содержит вызовы самой себя).

7. Функции в программе взаимодействуют путем передачи аргументов в функции и возвращения сформированных в функциях значений в место вызова функций. Возможно взаимодействие функций через глобальные переменные.

8. Аргументы в функцию передаются посредством операции присваивания, т.е. копирования ссылки на объект. Изменение имени формального параметра не изменяет имя объекта в вызывающей программе, но модификация объекта внутри функции может приводить к изменению объектов в вызывающей программе.

9. Если имя функции совпадает с именем встроенной функции, то используется имя функции пользователя.

```
>>> a = -3
>>> abs(a)
3
>>> def abs(x):
     print('Hello World!')
>>> abs(a)
Hello World!
```

3 Область действия переменных

В языке Python выделяют локальный и нелокальный контексты функции, и глобальный контекст программы. Локальный контекст функции – это формальные параметры функции и переменные, которым присваиваются значения внутри оператора def. Нелокальный контекст функции – это переменные, которым присваиваются значения в пределах объемлющего оператора def (если такой имеется). Глобальный контекст программы - это переменные, которым присваиваются значения за пределами всех операторов def (область действия глобальной переменной – модуль).

Рассмотрим особенности взаимодействия локальных и глобальных переменных:

1. Порядок сопоставления имен переменных осуществляется по правилу LEGB (Local-Enclosing-Global-Builtin): поиск переменной интерпретатор начинает в локальном контексте функции, затем в нелокальном контексте функции, после в глобальном контексте программы и, наконец, во встроенной области видимости.

Поиск завершается, как только будет найдено первое присваивание значения переменной.

```
>>> x = 'Global Variable'
>>> def enclosing_function():
    x = 'Enclosing Variable'
    def local_function():
        x = 'Local Variable'
        print(x)
    local_function()
    print(x)
>>> enclosing_function()
Local Variable
Enclosing Variable
>>> print(x)
Global Variable
```

2. Из правила LEGB следует, что глобальной переменной невозможно присвоить непосредственно значение внутри функции, так как переменная будет восприниматься интерпретатором как локальная.

```
>>> x = 3
>>> def function():
     x += 3
>>> function()
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    function()
  File "<pyshell#57>", line 2, in function
    x += 3
UnboundLocalError: local variable 'x' referenced before
assignment
```

3. Для изменения значения глобальной переменной в языке Python используется оператор global

```
global_stmt ::=  "global" identifier ("," identifier)*
```

Пример

```
>>> x = 3
>>> y = 7
>>> def function():
     global x, y
     x, y = y, x
>>> function()
>>> x
7
>>> y
3
```

4. Для изменения значения переменной, которая находится в объемлющем операторе def, используется оператор nonlocal

```
nonlocal_stmt ::=  "nonlocal" identifier ("," identifier)*
```

```
>>> x, y = 3, 7
>>> def enclosing_function():
      x, y = 3, 7
      def local_function():
          nonlocal x, y
          x, y = y, x
      local_function()
      print(x, y)


>>> enclosing_function()
7 3
>>> print(x, y)
3 7
```

Функции взаимодействуют между собой и программой через аргументы и через глобальные переменные, которые доступны во всех функциях. Однако использование глобальных переменных для взаимодействия функций считается некорректным, так как может вызвать побочный эффект, который затрудняет понимание программы или даже нарушает ее работу (изменив глобальную переменную в одной функции, можно не учесть это изменение в других частях программы).

4 Виды параметров функции

Функции в языке Python работают со следующими видами параметров (см. https://docs.python.org/3.5/glossary.html#term-parameter):

1. Обязательные параметры (positional): аргументы передаются в функцию последовательно слева направо в правильном позиционном порядке; количество аргументов должно совпадать с количеством параметров функции.

```
>>> # surname - первый параметр, name - второй параметр
>>> def print_me(surname, name):
      print('Surname is', surname)
      print('Name is', name)
      return
>>> print_me(' Python', 'Monty')
Surname is  Python
Name is Monty

>>> # В функцию print_me обязательно передаются два аргумента
>>> print_me('Python')
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print_me('Python')
TypeError: print_me() missing 1 required positional argument:
'name'
```

2. Поименованные параметры (keyword): аргументы передаются в функцию в виде «имя_параметра = значение». В этом случае для передачи аргументов функции используется имя формального параметра (ключ) вместо позиции.

```
>>> def print_me(surname, name):
      print('Name:', name, surname)

>>> # Соблюдение позиции аргументов не обязательно
>>> print_me(surname='Python', name='Monty')
Name: Monty Python
>>> print_me(name='Monty', surname='Python')
Name: Monty Python
```

3. Параметры по умолчанию (default) – параметры, значение которых по умолчанию устанавливается в объявлении функции.

```
>>> def print_me(surname='Python', name='Monty'):
      print('Name:', name, surname)

>>> # Для параметров функции установлены значения по умолчанию,
>>> # которые будут переданы в функцию как аргументы,
>>> # если пользователь не укажет собственные.
>>> print_me()
Name: Monty Python
>>> print_me('van Rossum', 'Guido')
Name: Guido van Rossum
```

4. Параметр переменной длины (var-positional, var-keyword): переменное количество аргументов передается в функцию в виде последовательности объектов. В объявлении функции параметру необходимо добавить префикс * (аргументы передаются в функцию в виде кортежа) или ** (аргументы передаются в функцию в виде словаря «имя_аргумента»:«значение_аргумента»).

```
>>> def print_me(*name):
      print('Name:', end=' ')
      for item in name:
            print(item, end=' ')

>>> print_me('Guido', 'van', 'Rossum')
Name: Guido van Rossum
```

Примечание:

1. Обязательные параметры должны предшествовать параметрам со значением по умолчанию.

```
>>> # Корректное объявление функции.
>>> def print_me(message, times = 1):
      print(message * times)
```

```
>>> # Некорректное объявление функции.
>>> def print_me(times = 1, message):
      print(message * times)

SyntaxError: non-default argument follows default argument
```

2. Параметры, объявленные после параметра переменной длины, доступны только по ключу.

```
>>> def print_me(*message, times):
      for i in range(times):
            for item in message:
                  print(item, end=' ')
            print()

>>> print_me('Guido', 'van', 'Rossum', 2)
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    print_me('Guido', 'van', 'Rossum', 2)
TypeError: print_me() missing 1 required keyword-only argument:
'times'
>>> print_me('Guido', 'van', 'Rossum', times=2)
Guido van Rossum
Guido van Rossum
```

3. Если аргументы должны передаваться в функцию только по ключу и аргументы переменной длины отсутствуют, то необходимо при объявлении функции указать в списке параметров '*' без имени параметра.

```
>>> def print_me(*, surname, name):
      print('Name:', name, surname)

>>> print_me('van Rossum', 'Guido')
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    print_me('van Rossum', 'Guido')
TypeError: print_me() takes 0 positional arguments but 2 were
given
>>> print_me(name='Monty', surname='Python')
Name: Monty Python
```

Пример. Написать программу, которая вычисляет Z-сумму значений функций при a=2, b=3.

$$Z = f(a,b) + f(a,b-1),$$

$$f(u,t) = \begin{cases} u + \sin(t), \text{если } t > 0 \\ \cos(u) - t, \text{если } t \le 0 \end{cases}.$$

```
import math

def func(u, t):
    if t > 0:
        return u + math.sin(t)
    return math.cos(u) - t

a = float(input('A = '))
b = float(input('B = '))
Z = func(a, b) + func(a, b - 1)
print(Z)
```

Результат работы программы:

```
A = 3
B = 2
Z= 7.750768411633579
```

В данной программе использованы следующие параметры:

- u, t – формальные параметры функции func;
- a, b – аргументы функции func.

При вызове func переменные u и t получают значения (копии) переменных a и b соответственно. Программа содержит 2 вызова функции func с разными аргументами; первым выполняется вызов func(a, b), а затем вызов func(a, b - 1).