Тема урока: Функции.

Функция в программировании представляет собой обособленный участок кода, который можно вызывать, обратившись к нему по имени, которым он был назван. При вызове происходит выполнение команд тела функции.

Функции можно сравнить с небольшими программами, которые сами по себе, т. е. автономно, не исполняются, а встраиваются в обычную программу. Нередко их так и называют – подпрограммы. Функции также при необходимости могут получать и возвращать данные. Только обычно они их получают не с ввода с клавиатуры, файла и др., а из вызывающей программы. Сюда же они возвращают результат своей работы.

Например, print(), input(), int() – это тоже функции. Код их тела нам не виден, он где- то "спрятан внутри языка". Нам же предоставляется только интерфейс – имя функции.

С другой стороны, программист всегда может определять свои функции. Их называют пользовательскими.

В языке программирования Python функции определяются с помощью оператора def.

Раздаточный материал № 36

```
def countFish():
    a = int(input())
    b = int(input())
    print("Всего", a+b, "шт.")
```

Функция состоит из заголовка и тела. Заголовок оканчивается двоеточием и переходом на новую строку. Тело имеет отступ.

Ключевое слово def сообщает интерпретатору, что перед ним определение функции. За def следует имя функции. Оно может быть любым, но со смыслом. После имени функции ставятся скобки. В приведенном примере они пустые. Это значит, что функция не принимает никакие данные из вызывающей ее программы. Однако она могла бы их принимать, и тогда в скобках были бы указаны параметры.

После двоеточия следует тело, содержащее инструкции, которые выполняются при вызове функции. Описание и вызов функции могут находиться в разных местах основной программы, но описание функции обязательно предшествует ее вызову. Можно определить функцию, но ни разу ее не вызвать.

Вызывается функция по имени со скобками. Если в функцию не предаются параметры, то скобки остаются пустыми. При вызове функции основной поток выполнения программы приостанавливается до момента завершения работы функции. Затем программа продолжает свою работу.

Раздаточный материал № 37

```
# программа выводит SOS

def CharS():
    print('S', end='')


def CharO():
    print('O', end='')


CharS()
CharO()
CharS()
```

Локальные и глобальные переменные

В программировании особое внимание уделяется концепции о локальных и глобальных переменных, а также связанное с ними представление об областях видимости. Соответственно, локальные переменные видны только в локальной области видимости, которой может выступать отдельно взятая функция. Глобальные переменные видны во всей программе. "Видны" – значит, известны, доступны. К ним можно обратиться по имени и получить связанное с ними значение.

К глобальной переменной можно обратиться из локальной области видимости. К локальной переменной нельзя обратиться из глобальной области видимости, потому что локальная переменная существует только в момент выполнения тела функции. При выходе из нее, локальные переменные исчезают. Компьютерная память, которая под них отводилась, освобождается. Когда функция будет снова вызвана, локальные переменные будут созданы заново.

Раздаточный материал № 38

```
def rectangle():
    a = float(input("Ширина %s: " % figure)) # обращение к глобальной
    b = float(input("Высота %s: " % figure)) # переменной figure
    print("Площадь: %.2f" % (a*b))
def triangle():
    a = float(input("Основание %s: " % figure))
    h = float(input("Высота %s: " % figure))
    print("Площадь: %.2f" % (0.5 * a * h))
figure = input("1-прямоугольник, 2-треугольник: ")
if figure == '1':
    rectangle()
elif figure == '2':
    triangle()
```

Здесь пять переменных. Глобальной является только figure. Переменные a и b из функции rectangle(), а также a и h из triangle() – локальные. При этом локальные переменные с одним и тем же идентификатором a, но объявленные в разных функциях, – разные переменные.

Идентификаторы rectangle и triangle, хотя и не являются именами переменных, а представляют собой имена функций, также имеют область видимости. В данном случае она глобальная, так как функции объявлены непосредственно в основной ветке программы.

В приведенной программе к глобальной области видимости относятся заголовки объявлений функций, объявление и присваивание переменной figure, конструкция условного оператора.

К глобальным переменным можно обращаться из функций.

При построении функции необходимо иметь в виду, что изменять значения глобальных переменных внутри функции можно, но это плохая практика программирования. Вместо это необходимо предусмотреть возврат полученного в функции значения в глобальную область видимости. Это делает программу более понятной и позволяет избежать проблем, связанных с поиском причины изменения глобальной переменной.

В Пайтон внутри одной функции можно определить другую. Так же функция может возвращать значение в то место, откуда она была вызвана. Для этого используется оператор return. Как только интерпретатор встречает return, то он "забирает" значение, указанное после этой команды, и "уходит" из функции.

Раздаточный материал № 39

```
# В основной ветке программы вызывается функция cylinder(), которая
вычисляет площадь
# цилиндра. В теле cylinder() определена функция circle(), вычисляющая
площадь круга по
# формуле πr2. В теле cylinder() у пользователя спрашивается, хочет ли он
получить только
# площадь боковой поверхности цилиндра, которая вычисляется по формуле
2πrh, или полную
# площадь цилиндра. В последнем случае к площади боковой поверхности
цилиндра должен
# добавляться удвоенный результат вычислений функции circle().

SC = 0
SQ = 0


def cylinder():
    r = float(input('Введи радиус:  '))

    def circle():
        SC = 3.14 * r * 2
        return SC

    c = input('1 - площадь боковой поверхности цилиндра, 2 - полная
площадь цилиндра:  ')
    if c == '1':
        print(circle())
    elif c == '2':
        h = float(input('Введи высоту:  '))
        SQ = 2 * 3.14 * r * h + 2 * circle()
        print(SQ)


cylinder()
```

Если необходимо сохранить результаты, передаваемые функцией, то их необходимо присвоить переменной. Например,

Раздаточный материал № 40

Port = `cylinder()`

В функции может быть несколько операторов return. Но выполнится только один, до которого первым дойдет поток выполнения (например, ветка except).

В Питоне можно возвращать из функции несколько объектов, перечислив их через запятую после команды return:

Раздаточный материал № 41

```
def duble():
    width = float(input('Введи ширину:   '))
    height = float(input('Введи высоту:   '))
    ploch = width * height
    perim  = 2 * (width + height)
    return ploch, perim

g_ploch, g_perim = duble()
```

```
print('Полощадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Перечисление значений через запятую создает объект типа «кортеж» (tuple). Когда же кортеж присваивается сразу нескольким переменным, то происходит сопоставление его элементов соответствующим в очереди переменным. Это называется распаковкой.

Таким образом, когда из функции возвращается несколько значений, на самом деле из нее возвращается один объект класса «кортеж». Перед возвратом эти несколько значений упаковываются в кортеж. Если же после оператора return стоит только одна переменная или объект, то ее/его тип хранится как есть.

Распаковка не является обязательной.

Раздаточный материал № 42

```
print(duble())

Введи ширину:   10
Введи высоту:   20
(200.0, 60.0) – скобки говорят, что выводится кортеж
```

Параметры и аргументы функции

В программировании функции могут не только возвращать данные, но также принимать их, что реализуется с помощью параметров, которые указываются в скобках в заголовке функции (формальные параметры) через запятую. Количество параметров может быть любым.

Параметры представляют собой локальные переменные, которым присваиваются значения в момент вызова функции (фактические параметры). Конкретные значения, которые передаются в функцию при ее вызове, будем называть аргументами.

Передача данных по значению

Раздаточный материал № 43

```
def duble(a, b):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:   '))
height = float(input('Введи высоту:   '))
g_ploch, g_perim = duble(width, height)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Параметры `width, height` являются глобальными и при вызове функции являются фактическими параметрами. Параметры a и b являются формальными и при вызове функции им будут присвоены значения `width, height.` Имена формальных и фактических параметров могут не совпадать (это даже не желательно).

Изменение значений a и b в теле функции никак не скажется на значении переменных `width, height`. Они останутся прежними. Поэтому говорят, что в функцию данные передаются по значению.

При работе с параметрами необходимо учитывать:

1. Количество формальных и фактических параметров должно совпадать (a,b и `width, height`).
2. Последовательность передачи значений (a присвоится значение `width`, b присвоится значение `height`).

В Python у функций бывают параметры, которым уже присвоено значение по-умолчанию. В таком случае, при вызове можно не передавать соответствующие этим параметрам аргументы. Хотя можно и передать. Тогда значение по умолчанию заменится на переданное.

Раздаточный материал № 44

```
# 1
def duble(a, b=20):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:   '))
g_ploch, g_perim = duble(width)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)

# 2
def duble(a, b=20):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:   '))
height = float(input('Введи высоту:   '))
g_ploch, g_perim = duble(width, height)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

При вызове функции, можно явно указывать, какое значение соответствует какому параметру. В этом случае их порядок не играет роли.

Раздаточный материал № 45

```
def duble(a, b):
    ploch = a * b
    perim = 2 * (a + b)
    return ploch, perim


g_ploch, g_perim = duble(b=20, a=10)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Функция может быть определена так, что в нее можно передать множество параметров:

Раздаточный материал № 46

```
def oneOrMany(*a):
    print(a)

oneOrMany(1)
oneOrMany('1', 1, 2, 'abc')
oneOrMany()
```

Результат:

```
(1,)
('1', 1, 2, 'abc')
()
```

Встроенные функции

Язык Python включает много уже определенных, т. е. встроенных в него, функций. Программист не видит их определений, они скрыты внутри языка. Достаточно знать, что эти функции принимают и что возвращают, то есть их интерфейс. Примеры print(), input(), int(), float(), str(), type().

Раздаточный материал № 47 (справочно)

abs() - возвращает абсолютное значение числа. Если это комплексное число, то абсолютным значением будет величина целой и мнимой частей.

chr() - возвращает строку, представляющую символ Unicode для переданного числа. Она является противоположностью ord(), которая принимает символ и возвращает его числовой код.

callable() - сообщает, является ли объект вызываемым. Если да, то возвращает True, а в противном случае — False. Вызываемый объект — это объект, который можно вызвать.

```
>>> callable(5)
False
```

round() - округляет вещественное число до определенного знака после запятой. Если второй аргумент не задан, то округление идет до целого числа. Второй аргумент может быть отрицательным числом. В этом случае округляться начинают единицы, десятки, сотни и т. д., то есть целая часть.

```
>>> a = 10/3
>>> a
3.3333333333333335
>>> round(a,2)
3.33
>>> round(a)
3

>>> round(5321, -1)
5320
>>> round(5321, -3)
5000
>>> round(5321, -4)
10000
```

divmod() - выполняет одновременно деление нацело и нахождение остатка от деления. Возвращает кортеж.

```
>>> divmod(10, 3)
(3, 1)
>>> divmod(20, 7)
(2, 6)
```

pow() - возводит в степень. Первое число – основание, второе – показатель. Может принимать третий необязательный аргумент - это число, на которое делится по модулю результат возведения в степень.

```
>>> pow(3, 2)
9
>>> pow(2, 4)
16

>>> pow(2, 4, 4)
0
>>> 2**4 % 4
0
```

dict() - используется для создания словарей. Это же можно делать и вручную, но функция предоставляет большую гибкость и дополнительные возможности. Например, ей в качестве параметра можно передать несколько словарей, объединив их в один большой.

```
>>> dict({"a":1, "b":2}, c = 3)
{'a': 1, 'b': 2, 'c': 3}

>>> list = [["a",1],["b",2]]
>>> dict(list)
{'a': 1, 'b': 2}
```

dir() - получает список всех атрибутов и методов объекта. Если объект не передать, то функция вернет все имена модулей в локальном пространстве имен.

```
>>> x = ["Яблоко", "Апельсин", "Гранат"]
>>> print(dir(x))
['__add__', '__class__', '__contains__',....]
```

enumerate() - в качестве параметра эта функция принимает последовательность. После этого она перебирает каждый элемент и возвращает его вместе со счетчиком в виде перечисляемого объекта. Основная особенность таких объектов — возможность размещать их в цикле для перебора.

```
>>> x = "Строка"
>>> list(enumerate(x))
[(0, 'С'), (1, 'т'), (2, 'р'), (3, 'о'), (4, 'к'), (5, 'а')]
```

eval() - обрабатывает переданное в нее выражение и исполняет его как выражение Python. После этого возвращается значение. Чаще всего эта функция используется для выполнения математических функций.

```
>>> eval('2+2')
4
>>> eval('2*7')
14
>>> eval('5/2')
2.5
```

filter() - функция используется для перебора итерируемых объектов и последовательностей, таких как списки, кортежи и словари. Но перед ее использованием нужно также иметь подходящую функцию, которая бы проверяла каждый элемент на валидность. Если элемент подходит, он будет возвращаться в вывод.

```
list1 = [3, 5, 4, 8, 6, 33, 22, 18, 76, 1]
result = list(filter(lambda x: (x%2 != 0) , list1))
print(result)
```

float() - конвертирует число или строку в число с плавающей точкой и возвращает результат. Если из-за некорректного ввода конвертация не проходит, возвращаются ValueError или TypeError.

hash() - у большинства объектов в Python есть хэш-номер. Функция hash() возвращает значение хэша переданного объекта. Объекты с __hash__() — это те, у которых есть соответствующее значение.

```
>>> hash('Hello World')
-2864993036154377761
>>> hash(True)
1
```

help() - предоставляет простой способ получения доступа к документации Python без интернета для любой функции, ключевого слова или модуля.

```
>>> help(print)
Help on built-in function print in module builtins:
```

int() - функция возвращает целое число из объекта, переданного в параметра. Она может конвертировать числа с разным основанием (шестнадцатеричные, двоичные и так далее) в целые.

```
>>> int(5.6)
5

>>> int('0101', 2)
5
```

iter() - принимает объект и возвращает итерируемый объект. Сам по себе он бесполезен, но оказывается крайне эффективным при использовании в циклах for и while. Благодаря этому объект можно перебирать по одному свойству за раз.

```
>>> lis = ['a', 'b', 'c', 'd', 'e']
>>> x = iter(lis)
>>> next(x)
'a'
>>> next(x)
'b'
>>> next(x)
'c'
>>> next(x)
'd'
```

max() - функция используется для нахождения «максимального» значения в последовательности, итерируемом объекте и так далее. В параметрах можно менять способ вычисления максимального значения.

```
>>> max('a', 'A')
'a'

>>> x = [5, 7, 8, 2, 5]
>>> max(x)
8

>>> x = ["Яблоко", "Апельсин", "Автомобиль"]
>>> max(x, key = len)
'Яблоко'
```

min() - функция используется для нахождения «минимального» значения в последовательности, итерируемом объекте и так далее. В параметрах можно менять способ вычисления минимального значения.

```
>>> min('a','A')
'A'

>>> x = [5, 7, 8, 2, 5]
>>> min(x)
2

>>> x = ["Виноград", "Манго", "Фрукты", "Клубника"]
>>> min(x)
'Виноград'
```

len() - функция используется для вычисления длины последовательности или итерируемого объекта.

```
>>> x = (2, 3, 1, 6, 7)
>>> len(x)
5
>>> len("Строка")
6
```

list() - в качестве параметра функция list() принимает итерируемый объект и возвращает список. Она обеспечивает большие гибкость и скорость при создании списков по сравнению с обычным способом.

```
>>> list("Привет")
['П', 'р', 'и', 'в', 'е', 'т']

>>> list({1:"a", 2:"b", 3:"c"})
[1, 2, 3]
```

map() - используется для применения определенной функции к итерируемому объекту. Она возвращает результат в виде итерируемого объекта (списки, кортежи,

множества). Можно передать и несколько объектов, но в таком случае нужно будет и соответствующее количество функций.

```
>>> def inc(x):
        x = x + 1
        return x

>>> lis = [1,2,3,4,5]
>>> result = map(inc,lis)

>>> for x in result:
        print(x)

2
3
4
5
6
```

next() - используется для итерируемых объектов. Умеет получать следующий (next) элемент в последовательности. Добравшись до конца, выводит значение по умолчанию.

```
>>> lis = ['a', 'b', 'c', 'd', 'e']
>>> x = iter(lis)
>>> next(x)
'a'
>>> next(x)
'b'
>>> next(x)
'c'
>>> next(x)
'd'
```

ord() - принимает один символ или строку длиной в один символ и возвращает соответствующее значение Unicode. Например, ord("a") вернет 97, а 97 — a.

```
>>> ord('a')
97

>>> ord('A')
65
```

reversed() - предоставляет простой и быстрый способ развернуть порядок элементов в последовательности. В качестве параметра она принимает валидную последовательность, например, список, а возвращает итерируемый объект.

```
>>> x = [3,4,5]
>>> b = reversed(x)
>>> list(b)
[5, 4, 3]
```

range() - используется для создания последовательности чисел с заданными значениями от и до, а также интервалом. Такая последовательность часто используется в циклах, особенно в цикле for.

```
>>> list(range(10,20,2))
[10, 12, 14, 16, 18]
```

reduce() - выполняет переданную в качестве аргумента функцию для каждого элемента последовательности. Она является частью functools, поэтому перед ее использованием соответствующий модуль нужно импортировать.

```
>>> list1 = [2, 5, 3, 1, 8]
>>> functools.reduce(operator.add,list1)
19

>>> list1 = [2, 5, 3, 1, 8]
>>> functools.reduce(operator.mul,list1)
240

>>> list1 = [2, 5, 3, 1, 8]
>>> functools.reduce(operator.truediv,list1)
0.016666666666666666
```

sorted() - используется для сортировки последовательностей значений разных типов. Например, может отсортировать список строк в алфавитном порядке или список числовых значений по возрастанию или убыванию.

```
>>> X = [4, 5, 7, 3, 1]
>>> sorted(X)
[1, 3, 4, 5, 7]
```

str() - используется для создания строковых представлений объектов, но не меняет сам объект, а возвращает новый. У нее есть встроенные механизмы кодировки и обработки ошибок, которые помогают при конвертации.

```
>>> str(5)
'5'

>>> X = [5,6,7]
>>> str(X)
'[5, 6, 7]'
```

set() - используется для создания наборов данных, которые передаются в качестве параметра. Обычно это последовательность, например, строка или список, которая затем преобразуется в множество уникальных значений.

```
>>> set()
set()

>>> set("Hello")
{'e', 'l', 'o', 'H'}
```

```
>>> set((1,2,3,4,5))
{1, 2, 3, 4, 5}
```
sum() - автоматически суммирует все элементы и возвращает сумму.
```
>>> x = [1, 2, 5, 3, 6, 7]
>>> sum(x)
24
```

tuple() - принимает один аргумент (итерируемый объект), которым может быть, например, список или словарь, последовательность или итератор и возвращает его в форме кортежа. Если не передать объект, то вернется пустой кортеж.

```
>>> tuple("Привет")
('П', 'р', 'и', 'в', 'е', 'т')

>>> tuple([1, 2, 3, 4, 5])
(1, 2, 3, 4, 5)
```

type() - применяется в двух сценариях. Если передать один параметр, то она вернет тип этого объекта. Если же передать три параметра, то можно создать объект type.

```
>>> type(5)
<class 'int'>

>>> type([5])
<class 'list'>
```

Вопросы для самоподготовки:

- Понятие функции.
- Оператор описание функции. Пример.
- Сколько раз функция может быть вызвана на выполнение.
- Как правильно разместить описание функции в программе.
- Как функция вызывается на выполнение.
- Понятие локальной переменной.
- Понятие глобальной переменной.
- Почему изменять значения глобальных переменных внутри функции не желательно.
- Назначение оператора return.
- Понятие формального параметра функции.
- Понятие фактического параметра функции.
- Сколько параметров может иметь функция.
- Что необходимо учитывать при работе с параметрами.
- В каком случае последовательность передачи значений не играет роли.
- Как определить функцию для передачи в нее заранее неизвестного числа параметров.