DL ACADEMY

Программирование
и вэб-разработка

directlinedev.ru

УРОК 3
ФУНКЦИИ И РАБОТА С ФАЙЛАМИ

ЗАГОЛОВОК

Пользователь ожидает, что элементы интерфейса откликнутся на действия

- Для взаимодействия приложения
- Для тестирования

РЕГЛАМЕНТ УРОКА

- Время урока - 1.5 часа
- Домашние задания
- Вопросы – в специальное время.

ПЛАН УРОКА

- Встроенные функции (часть 1)
- Функции
  - Документирование функций
  - Аргументы функции
  - Глобальные и локальные переменные
  - Функция как объект
  - Lambda-функции
  - Область видимости
  - Произвольное количество аргументов
  - Именованные аргументы
  - Значения по умолчанию
- Встроенные функции (часть 2)
  - zip()
  - map()
  - Filter()
- Работа с файлами

ВСТРОЕННЫЕ ФУНКЦИИ (ЧАСТЬ 1)

DL ACADEMY
Программирование
и вэб-разработка

ВСТРОЕННЫЕ ФУНКЦИИ (ЧАСТЬ 1)

Встроенные функции решают наиболее часто возникающие задачи. Мы уже пользовались некоторыми встроенными функциями python, такими как:

- print()
- input()
- len()
- Функциями преобразования типов int(), float(), bool() и т.д.

Кратко рассмотрим ещё группу наиболее используемых функций:

- range ([start=0], stop, [step=1]) - арифметическая прогрессия от start до stop с шагом step.
- abs(x) - Возвращает абсолютную величину (модуль числа).
- max (iter, [args...]*[,key]) - Максимальный элемент последовательности. min()
- round (X[,N]) - Округление до N знаков после запятой.
- sum (iter,start=0) - Сумма членов последовательности. type(object) - Возвращает тип объекта.
- enumerate (string) - Возвращает пары,(элемент, его индекс)

функции

```
# определение функции
def summ(a, b):
    c = a + b
    return c
# вызов функции
res = summ(5, 10)
print(res)
```

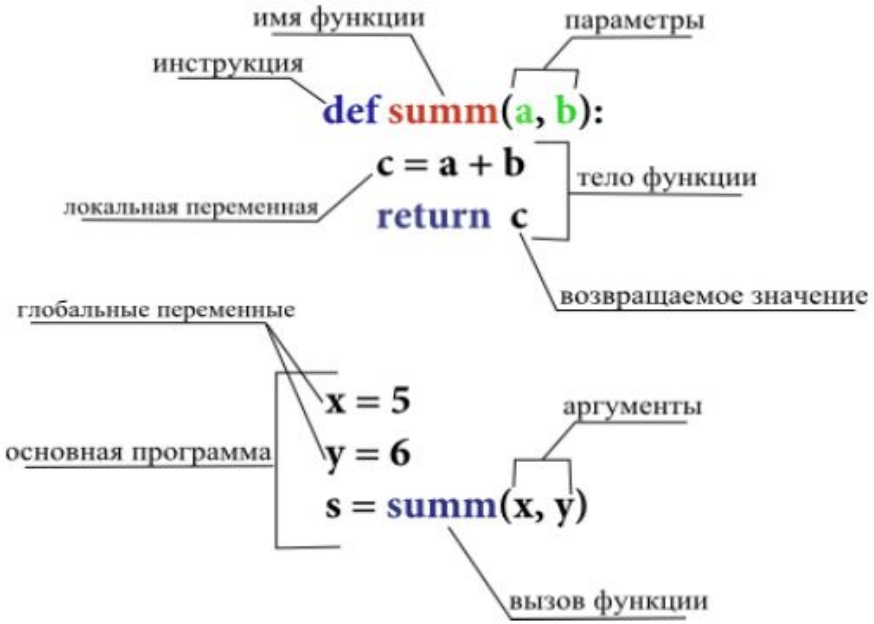


Рис. Схема и термины функции

функции

```
# определение функции
def summ(a, b):
    c = a + b
    return c
# вызов функции
res = summ(5, 10)
print(res)
```

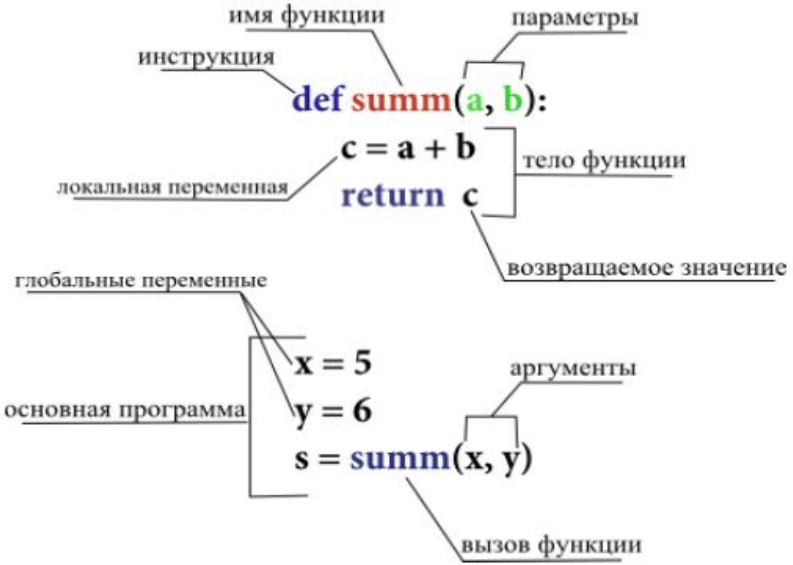
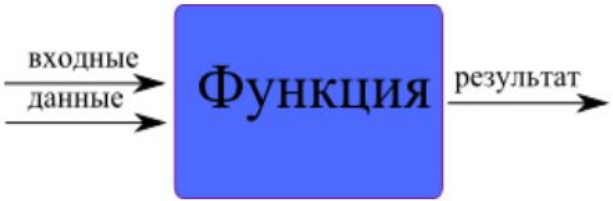


Рис. Схема и термины функции



DL ACADEMY
Программирование
и вэб-разработка

directlinedev.ru

ДОКУМЕНТИРОВАНИЕ ФУНКЦИЙ

```
def summ(a, b):
    """
    Возвращает сумму аргументов
    """
    c = a + b
    return c
```

АРГУМЕНТЫ ФУНКЦИИ

```
def summ(a, b):
    """
    Возвращает сумму аргументов
    """
    c = a + b
    return c
```

ГЛОБАЛЬНЫЕ И ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

Все переменные, созданные внутри функции, а также переменные - параметры (указанные в скобках после имени функции) являются **локальными** и существуют только во время выполнения функции.

Глобальные переменные – переменные, объявленные в основной части программы, т.е. Вне функции. Глобальные переменные существуют до конца времени выполнения программы.

```
z = 15
def local(x):
    # x, y - локальные переменные, доступны только во время выполнения функции
    # z - глобальная переменная, доступна во всем модуле (.py файле)
    y = 10
    print('x = {}, y = {}, z = {}'.format(x, y, z))
local(5)
```

ФУНКЦИЯ КАК ОБЪЕКТ

```
access = True
if access:
    def render(name):
        return 'Welcome, %s' % name
else:
    def render(name):
        return '%s, sorry. Try to enter again.' % name

print(render('Иван'))
```

Когда интерпретатор встречает инструкцию deftest - создаётся переменная test с указателем на объект - функцию, поэтому функции можно объявлять внутри других инструкций и даже в других функциях.

```
def mult(x):
    return x**2
# Переменная a - ссылка на функцию
a = mult
# Используя эту переменную, можно вызвать функцию
print(a(4))  # тоже, что и  print(mult(4))
```

А также возможно передавать ссылку на функцию внутрь другой функции

LAMBDA-ФУНКЦИИ

Анонимные (lambda) функции могут содержать лишь одно выражение, но и выполняются они быстрее. Анонимные функции создаются с помощью инструкции lambda. Кроме этого, их не обязательно присваивать переменной.

```
# В переменную f будет сохранена ссылка на объект-функцию
f = lambda x: x ** 2
# Суть в том, что инструкция lambda возвращает ссылку на функцию
# Тоже, что и
# def mult(x):
#    return x**2
# Только объявляется компактнее и работает быстрее
print(f(4))
# lambda-функции можно использовать и в выражении
print((lambda x: x ** 2)(4))
```

ОБЛАСТЬ ВИДИМОСТИ

Под термином область видимости подразумевается пространство имён, то есть место в программном коде, где имени было присвоено значение, и где это имя доступно в программе.

В python есть 4 области видимости:

- Локальная.
- Объемлющей функции.
- Глобальная (модуля).
- Встроенная (builtins)-предопределенные имена, например имена встроенных функций.

ОБЛАСТЬ ВИДИМОСТИ

```
x = 5          # глобальная переменная - доступна в любом месте данного модуля(файла)
def outside():
    y = 10     # доступна в теле данной функции + во всех вложенных
    def inside():
        z = 15 # доступна только в теле данной функции
        print('inside x: {}, y: {}, z: {}'.format(x, y, z))
    inside()
    print('outside x: {}, y: {}, z: {}'.format(x, y, 'z недоступна'))
outside()
print('inside x: {}, y: {}, z: {}'.format(x, 'y недоступна', 'z недоступна'))
x = 5
def wrapper():
    def test1():
        x = 10 # локальная переменная x перекрывает видимость глобальной x
        print('test1 x = ', x)
    def test2():
        print('test2 x = ', x)
# x = 22 # ^-- ошибка, выше используем переменную, объявленную позднее
    def test3():
        global x # инструкция global - поиск переменной в глобальной области
# Есть инструкция nonlocal -  поиск переменной в объемлющей функции
        print('test3 x = ', x)
        x = 25
    test1()
    test2()
    test3()
wrapper()
print('after wrapper x = ', x)
```

Поиск переменной происходит поочередно с 1 по 4-ую

ПРОИЗВОЛЬНОЕ КОЛИЧЕСТВО АРГУМЕНТОВ

Для получения неопределенного (любого) количества аргументов используют конструкцию: *args в качестве параметра функции, где args–произвольное имя.

```
def average(*args):
    summ = 0
    for arg in args:
        summ += arg
return summ/len(args)
```

ИМЕНОВАННЫЕ АРГУМЕНТЫ

```
def print_info(**kwargs):
        print("You name is %s %s. You age is %s. And your address is: %s"%
        (kwargs["name"],kwargs["surname"],kwargs["age"],kwargs["adress"]))
print_info(name="Василий",surname="Иванов",age="12",adress="ул.Белана 22")
```

```
{'adress': 'ул.Белана 22', 'age': '12', 'surname': 'Иванов', 'name': 'Василий'}
```

ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

```
def welcome(name="Инкогнито"):
    print("Приветствую вас, %s"%(name))
```

```
def print(*args, sep=' ', end='\n', file=None)
```

```
>>> welcome("User")
: Приветствую вас, User
>>> welcome()
: Приветствую вас, Инкогнито
```

```
>>> print("Иван","Иванович","Иванов",sep="//",end="!!!")
```

ВСТРОЕННЫЕ ФУНКЦИИ (ЧАСТЬ 2)

DL ACADEMY
Программирование
и вэб-разработка

ВСТРОЕННЫЕ ФУНКЦИИ (ЧАСТЬ 2)

zip()

Принцип работы этой функции проще показать на примерах.

```
a = [1,2]
b = [3,4]
print(zip(a,b))
```

Выведет: [(1, 3), (2, 4)]

```
a = [1,2,4]
b = [3,4]
c = [5,6,0]
print zip(a,b,c)
```

Выведет:[(1, 3, 5), (2, 4, 6)]

Берёт по минимальному количеству элементов, остальные будут отброшены.

ВСТРОЕННЫЕ ФУНКЦИИ (ЧАСТЬ 2)

map()

Позволяет применить функцию к каждому элементу последовательности, результаты функции возвращает в виде итератора:

Например, нам нужно возвести каждый элемент последовательности в квадрат:

```
print(list(map(lambda x: x*x, [2, 5, 12, -2])))
```

map(func_link, <итератор>) --> итератор, каждым элементом которого является применение функции funk_link к элементам исходного итератора.

Результат оборачиваем в list(), чтобы увидеть полный результат.

ВСТРОЕННЫЕ ФУНКЦИИ (ЧАСТЬ 2)

filter()

```
# filter(filter_func, <итератор>) --> итератор с отфильтровыванием элементов функцией filter_func
print(list(filter(lambda x: x > 5, [2, 10, -10, 8, 2, 0, 14])))
# Отбрасываем все элементы долиной НОЛЬ
print(list(filter(len, ['', 'not null', 'bla', '', '10'])))
```

T.e. filter() отбрасывает те элементы, для которых функция возвращает False.

РАБОТА С ФАЙЛАМИ

РАБОТА С ФАЙЛАМИ

```
import os
# не самый хороший способ задания пути:
path = 'files/text.txt'
# хороший кроссплатформенный метод указания пути:
path = os.path.join('files', 'text.txt')
f = open(path, 'r', encoding='UTF-8')
# Считываем всю информацию из файла в виде списка строк
print(f.readlines())
f.close()
```

РАБОТА С ФАЙЛАМИ

Режимы работы с файлом:

Режим	Обозначение
‘r’	открытие на чтение (является значением по умолчанию).
‘w’	открытие на запись, содержимое файла удаляется, если файла не существует, создается новый.
‘x’	открытие на запись, если файла не существует, иначе исключение.
‘a’	открытие на дозапись, информация добавляется в конец файла.
‘b’	открытие в двоичном режиме.
‘+’	открытие на чтение и запись

РАБОТА С ФАЙЛАМИ

```
path = os.path.join('files', 'text.txt')
f = open(path, 'r', encoding='UTF-8')
wanted_symbol = "+"
for line in f:                      # считываем файл построчно
    if wanted_symbol in line:  # пока не найдем нужную информацию
        print(line)
        break                       # как нашли, заканчиваем чтение файла
```

```
# Наиболее правильный способ работы с файлами
# По окончанию инструкции with, файл гарантировано будет закрыт, даже если произойдёт ошибка
with open(path, 'r', encoding='UTF-8') as f:
    print(f.readlines())
```

```
my_file = open("some.txt", "w")
my_file.write("Мне нравится Python!\Это классный язык!")
my_file.close()
```

ДОМАШНЕЕ ЗАДАНИЕ

Смотреть https://github.com/DanilXO/python_lesson_l1

Большинство заданий делятся на три категории easy, normal и hard:

- easy — простенькие задачи на понимание основ;
- normal — если вы делаете эти задачи, то вы хорошо усвоили урок;
- hard — наиболее хитрые задачи, часто с подвохами, для продвинутых слушателей.

ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ

Всё то, о чём сказано здесь, но подробнее:

- Функции и аргументы
- Функции map, zip, lambda
- Работа с файлами