Тема 1.5.1. Функции в Python. Создание функций.

```
# функция должна быть описана прежде ее вызова

# регистр в названии функции имеет значение

# имя функции не может начинаться с цифры


#def <Имя функции>([<Параметры>]):

#    ["""Строка документирования"""]

#    <Тело функции>

#    [return <Значение>] - необязательная инструкция, после которой ничего не выполняется

# концом функции считается выражение, перед которым меньшее количество пробелов


# если функция не содержит выражений, внутри надо разместить оператор pass

# пример функции, которая ничего не делает:

def f_pass():

  # pass


# количество параметров при вызове функции должно совпадать с количеством при описании


# определения функций

# пример функции без параметров:

def f_print_ok():

   print('все успешно! ')


# пример функции с параметром

def f_print(m):

   print(m)


def f_sum(x,y):
```

```
    return x+y

# вызов функции

f_print_ok()
f_print(2345)
f_print('Питон')



z = f_sum(10,20)
print('z=', z)   #  z= 30

z = f_sum('str', str(20))
print('z=', z)  # z= str20

print(f_sum('str', str(1120)))  # str1120

# сохранение ссылки на функции в переменной
def f_sum(x,y):
    return x+y

z = f_sum                       #  в переменную z записываем ссылку на функцию
print(type(z))                  # <class 'function'>

rez = z(15,40)
print('rez=', rez)   # rez= 55

# функции обратного вызова - функции, передаваемые по ссылке
def f_sum(x,y):
    return x+y
```

```
def f_sum2(f, a, b):
    return f(a, b)                  # через f будет доступна ссылка на ф-ю f_sum()


v = f_sum2(f_sum, 40, 10)
print('v=', v)                      # v= 50


z = f_sum                           #  в переменную z записываем ссылку на функцию f_sum()
v = f_sum2(z, 40, 70)
print('v=', v)                     # v= 110



# определение функции в зависимости от условия
m = input('Введите 1 для вызова 1-й функции:')
if m == '1':
    def f_print():
        return 'Вы ввели число 1'
else:
    def f_print():
        return 'Альтернативная функция'
print(f_print())


# необязательные параметры
# чтобы сделать параметр необязательным, нужно присвоить ему начальное значение при объявлении  ф-и
# необязательные параметры должны следовать после обязательных
def f_sum(x, y = 2):                # y - необязательный параметр
    return x+y


v1 = f_sum(5)
print('v1=', v1)  # v1= 7
```

```
v2 = f_sum(10, 20)
print('v2=', v2)  # v2= 30

# сопоставление по ключам - при вызове функции присваиваются значения
def f_sum(x, y):
    return x+y

print('v3=', f_sum(x = 16, y = 74))     # v3 = 90

# сопоставление по ключам удобно использовать, если ф-я имеет несколько необязат.параметров
def f_sum(a = 2, b = 3, c = 4):                    # # все параметры необязательные
    return a+b+c

print('r1=', f_sum(2, 3, 20))                    # позиционное присваивание  r1= 25
print('r2=', f_sum(c = 20))                       # сопоставление по ключам   r2= 25

# передача параметров значениями из кортежей и списков

def f_sum(a, b, c):
    return a+b+c

t1, arr = (1, 2, 3), [1, 2, 3]
print('t1:', f_sum(*t1))                    # передача параметров кортежем t1: 6
print('arr:', f_sum(*arr))                  # передача параметров списком arr: 6

t2 = (2, 3)
print('t2:', f_sum(1, *t2))                 # значения можно комбинировать t2: 6

# передача значений параметров из словаря
def f_sum(a, b, c):
```

```
    return a+b+c

d1 = {'a': 1, 'b': 2, 'c': 3}
print('d1 ', f_sum(**d1))  # d1  6

t, d2 = (1, 2), {'c': 3}
print(f_sum(*t, **d2))   #   6  можно комбинировать значения


# при передаче в функцию объектов НЕИЗМЕНЯЕМОГО типа (числа, строки)
# изменение значений  внутри функции не затронет значение переменной вне ф-и
def f_test(a, b):
    a, b = 20, 'str'

x, s = 80, 'test'
f_test(x, s)
print('x=', x, 's=', s)                    # x= 80 s= test значения переменных x,s не изменились

# при передаче в функцию объектов ИЗМЕНЯЕМОГО типа (список, словарь)
# изменение значений  внутри функции изменит значение переменной вне ф-и
def f_test(a, b):
    a[0], b['a'] = 'str', 800

x = [1, 2, 3]        # список
y = {'a': 1, 'b': 2}  # словарь
f_test(x, y)
# значения переменных вне ф-и изменились
print('x=', x, 'y=', y)          # x = ['str', 2, 3]     y= {'a': 800, 'b': 2}
```

```
# чтобы избежать изменения значений вне ф-и, надо создать копию объекта
def f_test(a, b):
    a = a[:]     # создаем поверхностную копию списка
    b = b.copy() # создаем поверхностную копию словаря
    a[0], b['a'] = 'str', 800

x = [1, 2, 3]        # список
y = {'a': 1, 'b': 2}  # словарь
f_test(x, y)
print('x=', x, 'y=', y)                # x= [1, 2, 3]     y= {'a': 1, 'b': 2} значения вне ф-и не
                                                                                изменились

# также чтобы значения вне ф-и не изменились, можно сразу при вызове функции передавать копии объектов :
f_test(x[:], y.copy())
print('x=', x, 'y=', y)                    #   x= [1, 2, 3]           y= {'b': 2, 'a': 1}


# если указать объект, имеющий изменяемый тип, в качестве значения по умолчанию,
# то этот объект будет сохраняться между вызовами функции:
def f_test(a = []):
    a.append(2)    # добавляем элемент в список
    return a

print(f_test())   # [2]
print(f_test())   # [2, 2]
print(f_test())   # [2, 2, 2]  значения накапливаются внутри списка

# чтобы обойти эту проблему:
def f_test(a = None):
    if a is None: a = []  # если значение равно None, создаем новый список
```

```
    a.append(2)
    return a


print(f_test())  # [2]
print(f_test())  # [2]
print(f_test())  # [2]
print(f_test([1]))  # [1, 2]
print(f_test([1]))  # [1, 2]
```