# Языки программирования

Лекция 6

# Пользовательские функции

- *Функция в Python* – объект, принимающий аргументы и возвращающий значение. Пользовательские функции позволяют уменьшить избыточность программного кода и повысить его ст

```
def <Имя функции>([<Параметры>]):
    ["""  Строка документирования """]
    <Тело функции>
    [return <Результат>]
```

# Пользовательские функции

```
def func():
    pass
```

Пример функции, которая ничего не делает

```
def func():
    print("Текст до инструкции return")
    return "Возвращаемое значение"
    print("Эта инструкция никогда не будет выполнена")

print(func()) # Вызываем функцию
```

**Результат выполнения:**

```
Текст до инструкции return
Возвращаемое значение
```

# Определение функций

```
def print_ok():
    """ Пример функции без параметров """
    print("Сообщение при удачно выполненной операции")

def echo(m):
    """ Пример функции с параметром """
    print(m)

def summa(x, y):
    """ Пример функции с параметрами,
        возвращающей сумму двух переменных """
    return x + y
```

Вызов функций:

```
print_ok()              # Вызываем функцию без параметров
echo("Сообщение")       # Функция выведет сообщение
x = summa(5, 2)         # Переменной x будет присвоено значение 7
a, b = 10, 50
y = summa(a, b)         # Переменной y будет присвоено значение 60
```

# Определение функций

```
def summa(x, y):
    return x + y

print(summa("str", "ing"))    # Выведет: string
print(summa([1, 2], [3, 4])) # Выведет: [1, 2, 3, 4]
```

Сохранение ссылки на функцию в

```
def summa(x, y):
    return x + y

f = summa                          # Сохраняем ссылку в переменной f
v = f(10, 20)                      # Вызываем функцию через переменную f
```

Функции, передаваемые по ссылке, обычно называются **функциями обратного вызова:**

```
def func (f, a, b):
    return f (a, b)


# передаем ссылку на функцию в качестве параметра
v = func (summa, 10, 20); v
30
```

# Функции

- _name_ - название функции в виде строки
- _doc_ - строка документирования

```
>>> def summa(x, y):
        """ Суммирование двух чисел """
        return x + y
>>> summa.__doc__
' Суммирование двух чисел '
```

Расположение определения функций:

**Правильно:**

```
def summa(x, y):
    return x + y
v = summa(10, 20) # Вызываем после определения. Все нормально
```

**Неправильно:**

```
v = summa(10, 20) # Идентификатор еще не определен. Это ошибка!!!
def summa(x, y):
    return x + y
```

# Функции

```
n = input("Введите 1 для вызова первой функции: ")
if n == "1":
    def echo():
        print("Вы ввели число 1")
else:
    def echo():
        print("Альтернативная функция")

echo() # Вызываем функцию
input()
```

---

```
def echo():
    print ("hello1")


def echo():
    print ("hello2")


echo()
```

# Необязательные параметры и сопоставление по ключам

Необязательные параметры:

```
def summa(x, y=2):          # y – необязательный параметр
    return x + y
a = summa(5)                # Переменной a будет присвоено значение 7
b = summa(10, 50)           # Переменной b будет присвоено значение 60
```

Сопоставление по параметрам:

```
def summa(x, y):
    return x + y
print(summa(y=20, x=10))    # Сопоставление по ключам
```

---

```
def summa(a=2, b=3, c=4): # Все параметры являются необязательными
    return a + b + c
print(summa(2, 3, 20))    # Позиционное присваивание
print(summa(c=20))        # Сопоставление по ключам
```

# Функции

Пример передачи значений из кортежа и списка:

```
def summa(a, b, c):
    return a + b + c
t1, arr = (1, 2, 3), [1, 2, 3]
print(summa(*t1))                    # Распаковываем кортеж
print(summa(*arr))                   # Распаковываем список
t2 = (2, 3)
print(summa(1, *t2))                 # Можно комбинировать значения
```

Пример передачи значений из словаря:

```
def summa(a, b, c):
    return a + b + c
d1 = {"a": 1, "b": 2, "c": 3}
print(summa(**d1))                   # Распаковываем словарь
t, d2 = (1, 2), {"c": 3}
print(summa(*t, **d2))               # Можно комбинировать значения
```

# Функции

```
def func(a, b):
    a, b = 20, "str"
x, s = 80, "test"

func(x, s)                  # Значения переменных x и s не изменяются
print(x, s)                 # Выведет: 80 test
```

---

```
def func(a, b):
    a[0], b["a"] = "str", 800
x = [1, 2, 3]               # Список
y = {"a": 1, "b": 2}        # Словарь
func(x, y)                  # Значения будут изменены!!!
print(x, y)                 # Выведет: ['str', 2, 3] {'a': 800, 'b': 2}
```

# Переменное число параметров в функции

```
def summa(*t):
    """ Функция принимает произвольное количество параметров """
    res = 0
    for i in t:       # Перебираем кортеж с переданными параметрами
        res += i
    return res
print(summa(10, 20))                    # Выведет: 30
print(summa(10, 20, 30, 40, 50, 60))    # Выведет: 210
```

Можно указать обязательные параметры и значения по умолчанию:

```
def summa(x, y=5, *t): # Комбинация параметров
    res = x + y
    for i in t:        # Перебираем кортеж с переданными параметрами
        res += i
    return res
print(summa(10))                        # Выведет: 15
print(summa(10, 20, 30, 40, 50, 60))    # Выведет: 210
```

# Переменное число параметров в функции

Сохранение переданных данных в

```
def func(**d):
    for i in d:      # Перебираем словарь с переданными параметрами
        print("{0} => {1}".format(i, d[i]), end=" ")
func(a=1, b=2, c=3) # Выведет: a => 1 c => 3 b => 2
```

Комбинирование параметров:

```
def func(*t, **d):
    """ Функция примет любые параметры """
    for i in t:
        print(i, end=" ")
    for i in d:     # Перебираем словарь с переданными параметрами
        print("{0} => {1}".format(i, d[i]), end=" ")
func(35, 10, a=1, b=2, c=3) # Выведет: 35 10 a => 1 c => 3 b => 2
func(10)                    # Выведет: 10
func(a=1, b=2)              # Выведет: a => 1 b => 2
```

# Переменное число параметров в функции

```
def func(*t, a, b=10, **d):
    print(t, a, b, d)
func(35, 10, a=1, c=3) # Выведет: (35, 10) 1 10 {'c': 3}
func(10, a=5)          # Выведет: (10,) 5 10 {}
func(a=1, b=2)         # Выведет: () 1 2 {}
func(1, 2, 3)          # Ошибка. Параметр a обязателен!
```

```
def func(x=1, y=2, *, a, b=10):
    print(x, y, a, b)
func(35, 10, a=1)      # Выведет: 35 10 1 10
func(10, a=5)          # Выведет: 10 2 5 10
func(a=1, b=2)         # Выведет: 1 2 1 2
func(a=1, y=8, x=7)    # Выведет: 7 8 1 10
func(1, 2, 3)          # Ошибка. Параметр a обязателен!
```

# Анонимные функции

- Помимо обычных, язык Python позволяет использовать анонимные функции, которые также называются *лямбда-функциями*. Анонимная функция описывается с помощью ключевого слова lambda по следующей схеме:
- lambda [<Параметр1[, …, <ПараметрN>]>]: <Возвращаемое значение>

```
f1 = lambda: 10 + 20                    # Функция без параметров
f2 = lambda x, y: x + y                 # Функция с двумя параметрами
f3 = lambda x, y, z: x + y + z          # Функция с тремя параметрами
print(f1())                             # Выведет: 30
print(f2(5, 10))                        # Выведет: 15
print(f3(5, 10, 30))                    # Выведет: 45
```

# Анонимные функции

```
>>> def f (x): return x**2
...
>>> print f(8)
64
>>>
>>> g = lambda x: x**2
>>>
>>> print g(8)
64
```

```
>>> def make_incrementor (n): return lambda x: x + n
>>>
>>> f = make_incrementor(2)
>>> g = make_incrementor(6)
>>>
>>> print f(42), g(42)
44 48
>>>
>>> print make_incrementor(22)(33)
55
```

```
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>>
>>> print filter(lambda x: x % 3 == 0, foo)
[18, 9, 24, 12, 27]
>>>
>>> print map(lambda x: x * 2 + 10, foo)
[14, 46, 28, 54, 44, 58, 26, 34, 64]
>>>
>>> print reduce(lambda x, y: x + y, foo)
139
```

# Анонимные функции

Необязательные параметры в анонимных функциях:

```
f = lambda x, y=2: x + y
print(f(5))                    # Выведет: 7
print(f(5, 6))                 # Выведет: 11
```

Сортировка без учета регистра символов:

```
arr = ["единица1", "Единый", "Единица2"]
arr.sort(key=lambda s: s.lower())
for i in arr:
    print(i, end=" ")
# Результат выполнения: единица1 Единица2 Единый
```

# Функции-генераторы

- *Функцией-генератором* называется функция, которая может возвращать одно значение из нескольких значений на каждой итерации. Приостановить выполнение функции и превратить функцию в генератор позволяет ключевое слово yield.

Пример использования функций-генераторов:

```
def func(x, y):
    for i in range(1, x+1):
        yield i ** y

for n in func(10, 2):
    print(n, end=" ")     # Выведет: 1 4 9 16 25 36 49 64 81 100
print()                   # Вставляем пустую строку
for n in func(10, 3):
    print(n, end=" ")     # Выведет: 1 8 27 64 125 216 343 512 729 1000
```

# Функции-генераторы

Использование метода __next()__:

```
def func(x, y):
    for i in range(1, x+1):
        yield i ** y

i = func(3, 3)
print(i.__next__())      # Выведет: 1 (1 ** 3)
print(i.__next__())      # Выведет: 8 (2 ** 3)
print(i.__next__())      # Выведет: 27 (3 ** 3)
print(i.__next__())      # Исключение StopIteration
```

# Вызов одной функции-генератора из другой

- yield from <Вызываемая функция-генератор>

```
def gen(l):
    for e in l:
        yield from range(1, e + 1)

l = [5, 10]
for i in gen([5, 10]): print(i, end = " ")
```

Результат:
1 2 3 4 5 1 2 3 4 5 6 7 8 9 10

```
def gen2(n):
    for e in range(1, n + 1):
        yield e * 2

def gen(l):
    for e in l:
        yield from gen2(e)

l = [5, 10]
for i in gen([5, 10]): print(i, end = " ")
```

Результат:
2 4 6 8 10 2 4 6 8 10 12 14 16 18 20

# Декораторы функций

- *Декораторы* позволяют изменить поведение обычных функций - например, выполнить какие-либо действия перед выполнением функции.

```
def deco(f):                                    # Функция-декоратор
    print("Вызвана функция func()").
    return f                                    # Возвращаем ссылку на функцию
@deco
def func(x):
    return "x = {0}".format(x)

print(func(10))
```

**Выведет:**

```
Вызвана функция func()
x = 10
```

**Эквивалентно:**

```
# Вызываем функцию func() через функцию deco()
print(deco(func)(10))
```

# Указание нескольких декораторов

```
def deco1(f):
    print("Вызвана функция deco1()")
    return f
def deco2(f):
    print("Вызвана функция deco2()")
    return f
@deco1
@deco2
def func(x):
    return "x = {0}".format(x)
print(func(10))
```

**Выведет:**

```
Вызвана функция deco2()
Вызвана функция deco1()
x = 10
```

**Использование двух декораторов эквивалентно следующему коду:**

```
func = deco1(deco2(func))
```

# Рекурсия

- *Рекурсия* - это возможность функции вызывать саму себя

```
def factorial(n):
    if n == 0 or n == 1: return 1
    else:
        return n * factorial(n - 1)

while True:
    x = input("Введите число: ")
    else:
        print("Вы ввели не число!")
print("Факториал числа {0} = {1}".format(x, factorial(x)))
```

**Эквивалентно:**

```
>>> import math
>>> math.factorial(5), math.factorial(6)
(120, 720)
```

# Глобальные и локальные переменные

- *Глобальные переменные* – это переменные, объявленные в программе вне функции. В Python глобальные переменные видны в любой части модуля, включая функции

```
def func(glob2):
    print("Значение глобальной переменной glob1 =", glob1)
    glob2 += 10
    print("Значение локальной переменной glob2 =", glob2)

glob1, glob2 = 10, 5
func(77) # Вызываем функцию
print("Значение глобальной переменной glob2 =", glob2)
```

**Результат выполнения:**

```
Значение глобальной переменной glob1 = 10
Значение локальной переменной glob2 = 87
Значение глобальной переменной glob2 = 5
```

# Глобальные и локальные переменные

- *Локальные переменные* – это переменные, объявляемые внутри функций. Если имя локальной переменной совпадает с именем глобальной переменной, то все операции внутри функции осуществляются с локальной переменной, а значение глобальной переменной не изменяется. Локальные переменные видны только внутри тела функции

```
def func():
        local1 = 77          # локальная переменная
        glob1 = 25           # локальная переменная
        print ("значение glob1 внутри функции = ", glob1)

glob1 = 10
func ()
print ("значение glob1 вне функции = ", glob1)
try:
        print (local1)      # вызовет исключение NameError
except NameError:
        print ("Переменная local1 не видна вне функции")
```

```
Значение glob1 внутри функции = 25
Значение glob1 вне функции = 10
Переменная local1 не видна вне функции
```

# Ключевое слово global

```
def func():
    # Объявляем переменную glob1 глобальной
    global glob1
    glob1 = 25          # Изменяем значение глобальной переменной
    print("Значение glob1 внутри функции =", glob1)
glob1 = 10              # Глобальная переменная
print("Значение glob1 вне функции =", glob1)
func()                  # Вызываем функцию
print("Значение glob1 после функции =", glob1)
```

**Результат выполнения:**

```
Значение glob1 вне функции = 10
Значение glob1 внутри функции = 25
Значение glob1 после функции = 25
```

# Глобальные и локальные переменные

- globals() – возвращает словарь с глобальными идентификаторами
- locals() – возвращает словарь с локальными идентификаторами
- vars([Объект]) – если вызывается без параметра внутри функции, то возвращает словарь с локальными идентификаторами. Если вызывается без параметра вне функции, то возвращает словарь с глобальными идентификаторами. При указании объекта возвращает идентификаторы этого объекта.

# Вложенные функции

```
def func1(x):
    def func2():
        print(x)
    return func2

f1 = func1(10)
f2 = func1(99)
f1()                    # Выведет: 10
f2()                    # Выведет: 99
```

# Самостоятельно

- Ключевое слово nonlocal
- Аннотации функций

# Вопросы

- 1. Что такое функция в языке программирования Python?
- 2. Приведите примеры определения и вызова функций.
- 3. Как в Python сохранить ссылку на функцию в другой переменной?
- 4. Что такое функциям обратного вызова?
- 5. Приведите пример передачи значения в функцию, используя сопоставление по ключам.
- 6. Как происходит распаковка списка, кортежа и словаря при передаче в качестве пара-метра функции?
- 7. Что такое функция-генератор?

# Вопросы

- 1. Какие существуют способы создания словарей?
- 2. Что позволяет делать метод get()?
- 3. Как работает метод setdefault()?
- 4. Какая функция позволяет получить количество ключей в словаре?
- 5. Какими способами можно осуществить перебор элементов словаря?