

Основы проектной деятельности

Занятие 5

Определение

Функция — это мини-программа внутри вашей основной программы, которая делает какую-то одну понятную вещь. Вы однажды описываете, что это за вещь, а потом ссылаетесь на это описание.

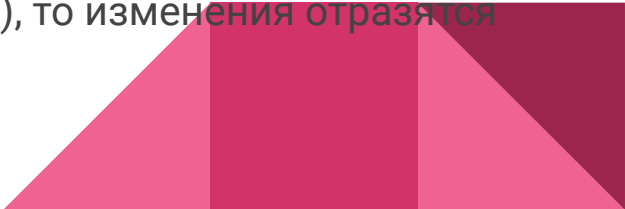


Определение

Например, вы пишете игру. Каждый раз, когда игрок попадает в цель, убивает врага, делает комбо, заканчивает уровень или падает в лаву, вам нужно добавить или убавить ему очков. Это делается двумя действиями: к старым очкам добавляются новые, на экран выводится новая сумма очков. Допустим, эти действия занимают 8 строк кода.

Допустим, в игре есть 100 ситуаций, когда нужно добавить или убавить очки — для каждого типа врага, преграды, уровня и т. д. Чтобы в каждой из ста ситуаций не писать одни и те же восемь строк кода, вы упаковываете эти восемь строк в функцию. И теперь в ста местах вы пишете одну строку: например, `changeScore(10)` — число очков повысится на 10.

Если теперь изменить, что происходит в функции `changeScore()`, то изменения отразятся как бы во всех ста местах, где эта функция вызывается.



Пример

Например, вы хотите написать программу, которая проигрывает песню. И в ней 7 раз звучит игра на барабанах. Вы бы прописали следующий алгоритм для барабанщика:

1. Взять палочки.
2. Поднять руку.
3. Совершить удар по барабану.
4. Нажать на педаль ногой.
5. Помотать головой в такт.

Без функции вам пришлось бы прописывать 7 раз одно и то же в тех частях композиции, где нужны ударные. Но гораздо удобнее оформить этот алгоритм в функцию `playDrums` и вызывать её каждый раз, когда необходимо. Это экономит время.

Зачем нужны функции?

Функции нужны, чтобы заметно упрощать и сокращать код, адаптировать его для разных платформ, делать более отказоустойчивым, легко отлаживать. И вообще порядок в функциях — порядок в голове.



Определение

Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции def.

```
def add() :  
    x = 5  
    y = 6  
    return x + y
```

```
c = add()
```



Определение

Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции `def`.

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


```
a = 4
```

```
b = 4
```

```
c = add(a, b)
```

```
d = add(c, 6)
```

Инструкция **return** говорит, что нужно вернуть значение. В нашем случае функция возвращает сумму `x` и `y`.



Область видимости

Переменные, созданные в теле функции, нельзя использовать после того, как эта функция завершит работу, поскольку они существуют только во время ее выполнения. В таких случаях программисты говорят, что область видимости переменных ограничена функцией.



Пример

```
def func1():  
    x = 5  
    y = 6  
    return x * 100 + y * 10
```

```
res = func1()  
y = res - 5  
print(res)  
print(x)  
print(y)
```



Пример

```
def func1():  
    x = 5  
    y = 6  
    return x * 100 + y * 10
```

```
res = func1()  
y = res - 5  
print(res)           560  
print(x)  
print(y)
```



Пример

```
def func1():  
    x = 5  
    y = 6  
    return x * 100 + y * 10
```

```
res = func1()  
y = res - 5  
print(res)  
print(x)  
print(y)
```

```
Traceback (most recent call last):  
  File "<pyshell#50>", line 1, in <module>  
    print(x)  
NameError: name 'x' is not defined
```

Пример

```
def func1():  
    x = 5  
    y = 6  
    return x * 100 + y * 10
```

```
res = func1()  
y = res - 5  
print(res)  
print(x)  
print(y)
```

555



Пример 2

```
def my_func(a, b):  
    x = 5  
    print(x)
```

```
x = 10  
my_func(1, 2)  
print(x)
```

1. Что, по-вашему, должно произойти?
2. Выдаст ли код цифру 10 дважды?



Пример 2

```
def my_func(a, b):  
    x = 5  
    print(x)
```

```
x = 10  
my_func(1, 2)  
print(x)
```

Нет, не выдаст. Причина в том, что мы имеем две переменные `x`. Переменная `x` внутри `my_func` имеет локальную область видимости функции и переопределяет переменную `x` вне функции. Так что когда мы вызываем функцию `my_func`, в выдаче мы видим 5, а не 10. Затем, когда функция возвращается, переменная `x` внутри функции `my_func` является кучей мусора и область для выдачи `x` срабатывает еще один раз. По этой причине последний оператор выдачи выдает именно 10.



Пример 2

```
def my_func(a, b):  
    print(x)  
    x = 5  
    print(x)
```

```
x = 10  
my_func(1, 2)  
print(x)
```

Когда вы запустите этот код, вы получите ошибку:

```
UnboundLocalError: local variable 'x' referenced  
before assignment
```

Это происходит потому, что Python замечает, что вы назначаете `x` в функцию `my_func` позже, что и приводит к ошибке, так как `x` еще не определен.



Задание 1

Пользователь вводит времени в новосибирске. Необходимо написать функции конвертации его в:

Москва

Владивосток



Задание 1

```
def to_msk(h, m):  
    h -= 4  
    print('in moskov: ', h, ':', m)
```

```
def to_vladivostok(h, m):  
    h += 3  
    print('in vladivostok:', h, ':', m)
```

```
h = int(input('H:'))  
m = int(input('M:'))  
to_msk(h, m)  
to_vladivostok(h, m)
```

