



# Создание функции, аргументы, параметры, возвращение результата

Функции

# Что такое функция?

Функция — это код, у которого есть имя, и выполняется этот код только тогда, когда его вызовут по имени.

Слово *print* — это функция.

Чтобы вызвать функцию *print*, нужно:

1. Написать название функции



```
print('Hello, world', end='!')
```

2. Аргумент, который выведется на экран



3. И необязательный дополнительный аргумент *end*



# Результат функции `print`

Результат функции `print` - отображение значения аргумента на экране.

```
Hello, world!
```

Функция – это обязательно какое-то действие. Поэтому имя функции должно отражать действие: *посчитать*, *вычислить*, *купить*.

Попробуем создать свою функцию, которая будет называться *buy* (купить). Покупать мы будем товары в магазине.

# Создание функции

Создание функции

Вызов функции

Результат функции

Вызов функции

Тело функции

# Создание функции

Новая функция будет выводить на экран количество товаров, которое можно купить за 50 монет, если один товар стоит 10 монет.

1. Каждая функция начинается со слова *def*

2. Затем пишется имя функции, например, *buy* (купить)

3. Скобки и двоеточие обязательны!

```
def buy():
```

```
    result = 50 // 10
```

```
    print(f'Товаров можно купить: {result}')
```

4. Вычисляем результат: количество товаров, которые можно купить за 50 монет

5. Выводим результат на экран функцией print



# Вызов функции

Функция с покупкой товаров в магазине готова.

Если запустить программу, то ничего не заработает, потому что действие *купить* только сохранилось в памяти компьютера, но ещё ни разу не использовалось.

Сразу же после создания функции вызовите функцию `buy` точно так же, как вызывали бы функцию *print*:

```
buy ()
```

# Результат функции

Результат функции – вывод на экран количества товаров.

Товаров можно купить: 5



# Как происходит вызов функции

1 Функция `buy` только сохраняется в памяти, а не срабатывает

```
def buy():
```

```
    result = 50 // 10
```

```
    print(f'Товаров можно купить: {result}')
```

```
buy()
```

2 Но сейчас отправляется команда "вызвать функцию `buy`"

3 И только теперь должно сработать тело функции

# Ошибка при вызове функции

Если вызвать функцию до её создания, то возникнет ошибка.

1 Попытаемся вызвать функцию *buy*

→ `buy()`

2 Но до этого момента её не существует

```
def buy():  
    result = 50 // 10  
    print(f'Товаров можно купить: {result}')
```

`NameError: name 'buy' is not defined`

`>>>`

3 И поэтому возникает ошибка



# Тело функции

В теле функции может быть что угодно: условия, циклы или даже вызываться другие функции, например:

```
def check_word():  
    string = input('Введите любое слово: ')  
    for letter in string:  
        if letter.isalpha():  
            print(letter)
```

## Функция `sell`-1 🏆

Создайте функцию `sell`.

Функция `sell` должна продавать 10 товаров по цене 20 монет за каждый.

Точно так же, как и функция `buy`, функция `sell` должна выводить результат на экран.

`sell()`



Прибыль составит: 200 монет

# Аргументы и параметры

Многократный вызов функции

Аргументы функции

Преобразование аргументов в параметры

Вызовы функций

# Множественный вызов функции

Каждый раз функцию не нужно создавать заново, её можно вызывать по имени в любой момент программы и сколько угодно раз.

```
buy ()
```

```
Товаров можно купить: 5
```

```
buy ()
```

```
Товаров можно купить: 5
```

```
buy ()
```

```
Товаров можно купить: 5
```

```
buy ()
```

```
Товаров можно купить: 5
```

И сколько бы раз не вызывалась функция, результат будет один и тот же – 50 монет.

# Функция shuffle

Импортируйте функцию shuffle:

```
from random import shuffle
```

Помните, что делает функция shuffle? Она перемешивает любой список, который ей укажут в скобках.

```
numbers = [3, 4, 5, 6]  
shuffle(numbers)  
print(numbers) → [6, 4, 5, 3]
```

# Аргументы функции

Результат функции `shuffle` зависит только от того, что ей отправят в скобках.

Значения, которые указывают в скобках, называются *аргументами*. Аргументов может быть несколько, и без них функция даже может не заработать, например:

```
shuffle()
```



```
shuffle()  
TypeError: shuffle() missing 1 required positional argument: 'x'
```

# Превращение аргументов в параметры

Дополните функцию buy:

1 Слова *money* (деньги) и *price* (цена) – это *параметры*. Имена параметров могут быть любыми.

```
def buy(money, price):  
    result = money // price  
    print(f'Товаров можно купить: {result}')
```

3 Аргументы 4 и 20 превращаются в параметры *money* и *price*. Их можно использовать в функции.

buy(**50, 10**)

2 При вызове функции указываем *аргументы* в том же порядке, в каком указаны *параметры*: 50 – это *money*, 10 – это *price*.

# Вызовы с разными аргументами

Аргументы могут быть любыми, они всегда будут превращаться в параметры *money* и *price*, а результат всегда будет разным.

```
def buy(money, price):  
    result = money // price  
    print(f'Товаров можно купить: {result}')
```

buy(50, 10)	→	Товаров можно купить: 5
buy(1000, 20)	→	Товаров можно купить: 50
buy(200, 5)	→	Товаров можно купить: 40
buy(30, 15)	→	Товаров можно купить: 2

# Вызов без аргументов

Но если теперь вызвать функцию `buy` без аргументов...

```
def buy(money, price):  
    result = money // price  
    print(f'Товаров можно купить: {result}')
```

```
buy()
```

...ТО ВОЗНИКНЕТ ОШИБКА “Функции не хватает аргументов”:

```
buy()  
TypeError: buy() missing 2 required positional arguments: 'money' and 'price'
```

## Функция `sell`-2

Дополните функцию `sell`.

У функции `sell` должны появиться параметры `goods` (товары) и `price` (цена за один товар).

Теперь при вызове функции `sell` пользователь должен указывать два обязательных аргумента.

Точно так же, как и функция `buy`, функция `sell` должна выводить результат на экран.

```
sell(10, 5)
```



Прибыль составит: 50 монет

# Возвращение результата

Где результат?

Возвращаемое значение

Использование результата

Оператор `return`

# Где результат?

Существуют функции, которые не выводят ничего на экран, например:

```
from random import randint  
randint(1, 5)
```

→ А где результат?





# Возвращаемое значение

Результатом такой функции является не выведенное на экране значение, а возвращаемое значение. Такой результат нужно либо записать в переменную, а потом использовать:

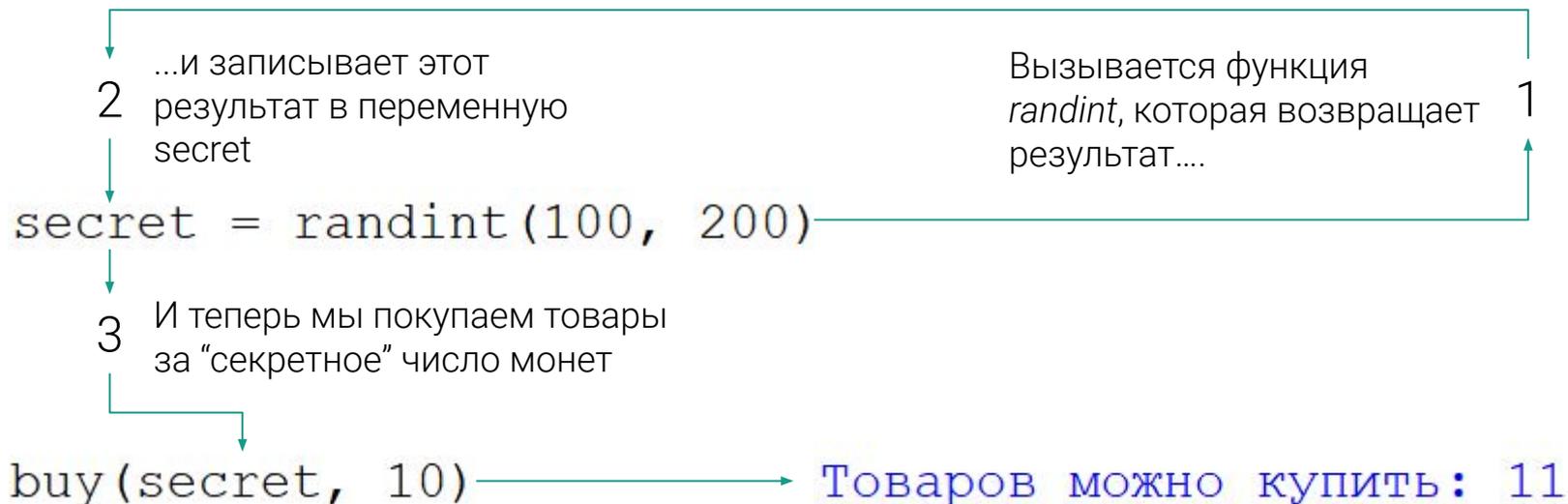
```
secret = randint(1, 5)
print(secret)
```

Либо не записывать в переменную, а сразу, если нужно, выводить функцией *print*:

```
print(randint(1, 5))
```

# А зачем?

Это очень удобно, если результат функции нужно не просто вывести, а использовать где-то дальше в программе.





# Использование результата

Как использовать полученный результат?

Товаров можно купить: 11

Как сделать так, чтобы количество монет можно было записать в переменную и использовать дальше?

# Возвращение результата

Исправьте функцию *buy*:

↓  
2 Результат функции – значение  
переменной *result* – возвращается в  
переменную *x*

↑  
Специальное слово *return* возвращает из  
функции любое значение – например,  
значение переменной *result*

```
def buy(money, price):  
    result = money // price  
    return result
```

→ `x = buy(200, 5)`



# Оператор *return*

Оператор *return* всегда означает конец функции. Если программа дошла до этого слова, всегда произойдёт выход из функции, поэтому это слово чаще всего пишут в самом конце тела функции.

Это очень похоже на слово *break* в циклах, только *return* ещё и возвращает значение.

# Перепродажа товара

Теперь мы можем использовать товары, полученные из функции *buy*, чтобы продать их функцией *sell* подороже и заработать много денег!

```
x = buy(200, 5)
sell(x, 10)
```



## Функция `sell`–3

Дополните функцию `sell`.

Точно так же, как и функция `buy`, функция `sell` должна НЕ выводить результат на экран, а возвращать его из функции.

Результат должен отобразиться на экране только в том случае, если результат был записан в переменную, а значение переменной выведено на экран.

```
y = sell(40, 10)  
print(y)
```



400