Python: функции | Я.Шпора

Функция — именованный блок кода, который выполняет конкретную задачу. Функции помогают организовать код в более мелкие, но при этом самодостаточные фрагменты.

Работа с функцией обычно состоит из двух шагов:

1. Объявить функцию и описать в ней те действия, которые она должна выполнять, то есть написать тело функции.
2. Вызвать функцию, чтобы код функции выполнился.

```
# Объявление функции.
def welcome():
    # Тело функции.
    print('Добро пожаловать!')


# Вызов функции.
welcome()

# Вывод в терминал: Добро пожаловать!
```

Как называть функции

- Имя функции должно давать общее представление о том, чего следует ожидать от работы этой функции.
- Если имя состоит более чем из одного слова, его нужно писать в стиле *snake_case*.
- Имена должны быть достаточно краткими, чтобы их было легко читать и писать, но не настолько короткими, чтобы их было сложно понять. Например, `calculate_average()` лучше, чем `calc_avg()`, а `calc_avg()` лучше, чем `ca()`.
- Не следует использовать имена, которые уже заняты встроенными функциями.
- Имена лучше начинать с глаголов, описывающих действие, например таких: *get*, *calculate*, *check*, *set*, *show*, *remove*.

Параметры и аргументы функции

При объявлении функции указывают **параметры функции** — переменные, которые будут обрабатываться в теле этой функции. Имена для параметров придумывает сам разработчик.

Значения для этих параметров передаются при вызове — их называют **аргументами функции**.

```
# name — это параметр функции.
def welcome(name):
    print(f'Добро пожаловать, {name}!')


# 'Тоня' — это аргумент функции.
welcome('Тоня')

# Вывод в терминал: Добро пожаловать, Тоня!
```

Параметры указывают в круглых скобках **при объявлении** функции.

Аргументы указывают в круглых скобках **при вызове** функции.

Параметров может быть несколько. Они указываются через запятую.

При вызове функции аргументы передаются в соответствии с порядком записи (тоже через запятую): первый аргумент передаётся в первый параметр, второй аргумент — во второй параметр. Такие аргументы называют позиционными.

```
# Функция с двумя параметрами.
def welcome(profession, name):
    print('Добро пожаловать,', profession, name)


# Аргументы передаются в соответствии с порядком записи параметров.
welcome('фермер', 'Тоня')
```

```
# Вывод в терминал: Добро пожаловать, фермер Тоня!
```

Именованные аргументы

При вызове функции также можно передавать именованные аргументы — явно указывать, какому параметру какой аргумент соответствует. Если аргументы поименованы, их можно передавать в любом порядке:

```
def action(name, food):
    print(name, 'заготавливает на зиму', food)


action(food='инжир', name='Женя')

# Вывод в терминал: Женя заготавливает на зиму инжир
```

Значения параметров по умолчанию

При необходимости можно указать значения по умолчанию для параметров функции. Это позволит не указывать их при вызове этой функции. Такие параметры называют необязательными или опциональными.

При объявлении функции можно комбинировать обязательные параметры с опциональными. Опциональные параметры должны быть определены после обязательных.

```
def welcome(name, profession='программист'):
    print('Добро пожаловать,', profession, name)


welcome('Тоня')

# Вывод в терминал: Добро пожаловать, программист Тоня!
```

Возврат значений из функций

Функция может возвращать результат своей работы. Для этого есть ключевое слово `return`: после него указывают значение, которое должна вернуть

функция.

```
def welcome(profession, name):
    hello = 'Добро пожаловать в программу, ' + profession + ' ' + r
    # Функция возвращает значение, которое сохранено в переменной h
    return hello


# Значение, которое вернёт функция, будет присвоено переменной mess
message = welcome('фермер', 'Тоня')
print(message)

# Вывод в терминал: Добро пожаловать в программу, фермер Тоня
```

Пустая функция

Создаётся с помощью ключевого слова `pass`...

```
def empty():
    pass


empty()
```

...или многоточия `Ellipsis`:

```
def empty():
    ...


empty()
```

`pass` — это инструкция, команда, а `...` — значение, которое можно присваивать переменным и возвращать в качестве ответа функции.