Довольно часто встроенных функций не хватает даже новичкам. В таком случае нет другого выбора, кроме как создать свою собственную функцию, используя ключевое слово def (справа, производное от **define**). Давайте посмотрим на синтаксис:

def function\_name(parameter1, parameter2, ...):

# function's body

...

return "return value"

После def мы пишем имя нашей функции (чтобы вызвать ее позже) и имена **параметров**, которые может принимать наша функция, заключенные в круглые скобки. Не пропустите двоеточие в конце строки. Имена функции и ее параметров соответствуют тому же соглашению, что и имена **переменных**, то есть они должны быть написаны **строчными буквами с подчеркиванием между словами**.

Отступ в 4 пробела показывает интерпретатору, где начинается тело функции и где оно заканчивается. Все инструкции в теле функции должны иметь отступ. Вы можете выполнять вычисления внутри своей функции и использовать return ключевое слово для отправки результата обратно. Только когда отступ отсутствует, определение функции заканчивается.

Позже параметры принимают значения, переданные при вызове функции. Те значения, которые мы передаем функции, известны как **аргументы**. Единственное различие между параметрами и аргументами заключается в том, что мы вводим параметры в определение функции и даем аргументы (некоторые конкретные значения) при вызове функции. Вот менее абстрактный пример функции:

# Function definition

def multiply(x, y):

return x \* y

# Function calls

a = multiply(3, 5) # 15

b = multiply(a, 10) # 150

В случае, если вы не хотите передавать какие-либо аргументы, круглые скобки остаются пустыми:

def welcome():

print("Hello, people!")

Вы также можете объявить своего рода пустую функцию с помощью pass инструкции:

# This function does nothing (yet)

def lazy\_func(param):

pass

Когда вы решите вызвать lazy\_func() с произвольным значением в качестве аргумента, ничего не произойдет. So pass - это просто заполнитель, но, по крайней мере, ваш код будет действителен с ним.

**Параметры против аргументов**

Сейчас не совсем ясно, каковы параметры, не так ли? На самом деле, параметры - это просто псевдонимы для значений, которые могут быть переданы функции. Рассмотрим следующий пример:

def send\_postcard(address, message):

print("Sending a postcard to", address)

print("With the message:", message)

send\_postcard("Hilton, 97", "Hello, bro!")

# Sending a postcard to Hilton, 97

# With the message: Hello, bro!

send\_postcard("Piccadilly, London", "Hi, London!")

# Sending a postcard to Piccadilly, London

# With the message: Hi, London!

Как вы можете видеть, эта функция представляет собой повторно используемый фрагмент кода, который может выполняться с разными аргументами, то есть с разными значениями, передаваемыми этой функции. Здесь address и message просто псевдонимы, под которыми функция получает значения, а затем обрабатывает их в теле.

Эта функция принимает ровно 2 аргумента, поэтому вы не сможете выполнить ее с более или менее чем 2 аргументами:

send\_postcard("Big Ben, London")

TypeError: send\_postcard() missing 1 required positional argument: 'message'

**Выполнение и возврат**

Наша предыдущая функция выполняла только некоторые действия, но у нее не было никакого **возвращаемого значения**. Однако вы можете захотеть что-то вычислить в функции и в какой-то момент вернуть результат. Проверьте следующий пример:

def celsius\_to\_fahrenheit(temps\_c):

temps\_f = temps\_c \* 9 / 5 + 32

return round(temps\_f, 2)

# Convert the boiling point of water

water\_bp = celsius\_to\_fahrenheit(100)

print(water\_bp) # 212.0

Ключевое слово return используется для указания того, какие значения выводит функция. По сути, это результат вызова функции. Итак, в приведенном выше примере мы сохранили значение, возвращаемое нашей функцией, в переменной water\_bp. На всякий случай мы напечатали результат.

Еще одна вещь, которую нужно сказать, это то, что функции не обязательно имеют возвращаемые значения. Хорошо известная print() функция фактически ничего не возвращает. Изучите приведенный ниже код:

chant = print("We Will Rock You")

print(chant)

И его вывод:

We Will Rock You

None

Мы объявили переменную chant и вызвали print(). Очевидно, что функция была выполнена. Но сама переменная оказалась объектом **None**, что означает, что вызванной функции нечего возвращать. Значение chant равно **None**.

Интерпретатор Python перестает выполнять функцию после return. Но что, если тело функции содержит **более** одного оператора возврата? Тогда выполнение завершится после первого. Пожалуйста, имейте это в виду!

**Краткие сведения**

Таким образом, мы изучили синтаксис для объявления функций. Теперь вы также знаете, что:

* Параметры функции - это просто псевдонимы или заполнители для значений, которые вы будете им передавать. Параметры повторно инициализируются при каждом вызове функции. Внутри функции у вас есть доступ к этим значениям, что означает, что вы можете выполнять вычисления по ним.
* Функция может просто выполнять действие, ничего не возвращая, или возвращать определенный результат. Если ваша функция ничего не возвращает, присвоите ее результат переменной или распечатайте егоNone.

Объявление ваших собственных функций делает ваш код более структурированным и пригодным для повторного использования. Всякий раз, когда вы используете один и тот же фрагмент кода более одного раза, попробуйте создать из него функцию!

**Практика**







