

# Theory: Invoking a function

🕒 10 minutes

11 / 11 problems solved

Start practicing

15390 users solved this topic. Latest completion was 30 minutes ago.

Even though invoking functions in Python is not about casting a spell or the like, it does sometimes work wonders. Let's start with the concept. Basically, a **function** is a structured fragment of code we may want to use in more than one place and more than one time. For another thing, functions allow us to read both our code and that of someone else way better. Haven't they become your favorite yet?

Here is a simple function call:

```
multiply(1, 7)
```

Here `multiply` is the name of the function, and numbers in parentheses `(1, 7)` are its **arguments**. What is an argument? Well, it's just a value, that will be used inside the body of the function. Let's go deeper into it!

## §1. Invoking print()

To **call**, or **invoke**, a **function** in your program, simply write its name and add parentheses after it. That's all it takes! Fun fact: if you've ever typed an expression like this `print("Hello, world!")`, you already know a little about functions. In this tiny example, however, we see the message "Hello, world!" in the parentheses after the name of the `print` function. What does it mean? This string is just an argument. And more often than not, functions do have arguments. As for the `print` function, we may as well use it with no argument at all or even with multiple arguments:

```
1 print("Hello, world!")
2 print()
3 print("Bye,", "then!")
```

And here is the output:

```
1 Hello, world!
2
3 Bye, then!
```

So, the first call prints one string, the second call of `print` without arguments prints, in fact, an empty line, and the last call outputs our two messages as one expression. Are you surprised by these results? Then you may learn how the `print` function works in more detail from its [documentation](#). The Python documentation contains all sorts of information about the function of your interest, for example, which arguments it expects.

## §2. Built-in functions

Functions can make life easier, provided one is aware of their existence. Many algorithms are already written, so there is no need for reinvention, except perhaps for educational purposes. The Python interpreter has a number of functions and types **built into** it, so they are always available. Currently, the number of [built-in functions amounts to 69](#) (in the latest version **Python 3.8**). Some of them are used to convert the **object type**, for example, `str()` returns a string, `int()` returns an integer, `float()` returns a floating-point number. Others deal with **numbers**: you can `round()` them and `sum()` them, find the minimum `min()` or the maximum `max()`. Still others give us information about the object: its `type()` or length `len()`. Let's consider them in action!

In the following example, `len()` counts the number of characters in the string (the same goes for any **sequence**).

Current topic:

✓ [Invoking a function](#)

Stage 116★ ...

Topic depends on:

✓ [Variables](#)

Stage 117★ ...

Topic is required for:

✓ [Declaring a function](#)

Stage 110★ ...

✓ [Basic string methods](#)

Stage 113★ ...

✓ [Load module](#)

Stage 15★ ...

✓ [Files in Python](#)

Stage 2 ...

[Overview of NLTK](#) ...

Table of contents:

[1 Invoking a function](#)

[§1. Invoking print\(\)](#)

[§2. Built-in functions](#)

[§3. Recap](#)

[Feedback & Comments](#)

```
1 number = "111"
2
3 # finding the length of an object
4 print(len(number)) # 3
```

Then we declare the variables `integer` and `float_number` and write their sum into `my_sum`. By the way, the `sum()` function also deals with sequences.

```
1 # converting types
2 integer = int(number)
3 float_number = float(number)
4 print(str(float_number)) # "111.0"
5
6 # adding and rounding numbers
7 my_sum = sum((integer, float_number))
```

The result is a floating-point number, which becomes clear after printing `my_sum`.

```
1 print(my_sum) # 222.0
2 print(round(my_sum)) # 222
```

Furthermore, you can see how to find the minimum and maximum values: the smallest number equals `111` and the largest number `222.0` belongs to floats.

```
1 # finding the minimum and the maximum
2 print(min(integer, float_number)) # 111
3 print(type(max(integer, float_number, my_sum))) # <class 'float'>
```

There is far more to explore, but let's sum it up.


## §3. Recap

The beauty of functions is that we can use them without clear insight into their inner structure and how they manage to perform what we need. But if you want to put a function to good use, make sure to check its documentation or try invoking the special function `help()` with the name of the function in question wrapped in parentheses. It may become necessary if, for example, the function does not return any value, writes processed data to a file or prints the output on the screen.

Let's make a brief summary:

- functions are meant to be reusable, which means we are free to apply it multiple times with different arguments,
- to call a function write its name followed by parentheses and place the arguments within,
- normally, a function has documentation, which sometimes might be of huge help.

 Report a typo

 Thanks for your feedback!

Start practicing

[Comments \(12\)](#)

[Hints \(0\)](#)

[Useful links \(2\)](#)

[Show discussion](#)