TASK

Beginner Programming with Functions — Using Built-in Functions

Visit our website

Introduction

WELCOME TO THE BEGINNER PROGRAMMING WITH FUNCTIONS — USING BUILT-IN FUNCTIONS TASK!

This is an introduction to *functions* in Python. A function is a reusable and organised block of code that is used to perform a single action or specific task. Functions can either be user-defined or built-in. In this task, you will be introduced to functions that are built into the Python language itself and are readily available for us to use. We are going to be exploring some of the more useful and most common Python functions.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

A note from the
Hyperion Team

Why does 19200 Big Macs a year equal potential job security? Why can South African software developers buy more Big Macs than their American counterparts, and what do Big Macs and software development have in common anyway? All these questions are answered **HERE** when considering 5 key reasons why software development is a great career to find yourself in!

WHAT ARE FUNCTIONS?

A *function* is a reusable and organised block of code that is used to perform a single action or specific task. Functions are also sometimes referred to as '*methods*' (if you have used Java before, functions in Python are very similar to Java methods).

Functions in programming also relate to mathematical functions — perhaps you recall `f(x)` in mathematics. A mathematical function took some input, in this case `x`, did some computations with it, and returned a value, normally called `y`. For example, `y = f(x)` is a function that when called with the parameter `x` returns the value `y`.

Functions can either be user-defined or built-in. Built-in functions are built into the Python language itself and are readily available for us to use. You have actually already been using some built-in functions such as *print(), input(), len(), int(), str(),* and *float()*. See a list of available built-in functions **here**. The *write* command, i.e. *ofile.write(name+"\n")*, is in fact also a function, implemented by some programmer, that tells Python how to write the content you give it to the named output file.

There are thousands of functions already implemented in Python that you can use to get things done. Programmers have already written the logic for many common and even complex tasks. Sometimes you can find the exact built-in function that you need to complete a task.

However, you are not limited to these functions. You can also create your own functions to meet your needs. These are what are known as **user-defined**

functions.

IMPORTING OUTSIDE MODULES

We mentioned earlier that there are thousands of already written functions, but how do you get access to them? Modules are the answer. Modules, or libraries, are pieces of code already written by other programmers that you can *import* into your program. Though you don't see their code directly, you can access them. You import these modules using the *import* keyword followed by the module's name.

An example of a very useful module is the *math* module, which has many built-in functions for you to use. To find out more about the math module and its functions, look **here**. Python modules and their functions are usually very well documented online. See the example file that accompanies this task to see how some functions in the math module can be used.

COMMONLY USED BUILT-IN FUNCTIONS

Let's look at some more examples of the more commonly used built-in functions:

round()

```
long_number = 66.6564544
print(round(long_number,2))
```

Output:

```
66.66
```

The example above shows how we use the *round()* function. The function takes two arguments: a float, and the number of digits you want to round off to. In the example above, *long_number* is rounded off to 2 decimal places, so the output will be 66.66.

sorted()

```
random_list = [6,4,3,2,9,7,66,22,35,1]
print(sorted(random_list))
```

Output:

```
[1, 2, 3, 4, 6, 7, 9, 22, 35, 66]
```

The example above shows an example of how we use the *sorted()* function. The function takes one argument: something the function can iterate through (e.g. a list or dictionary). In the example above, *random_list* is sorted in ascending order.

min() and max()

```
random_list = [6,4,3,2,9,7,66,22,35,1]
print(min(random_list))
print(max(random_list))
```

The example above shows examples of the *min()* and *max()* functions with the same *random_list* above. Like the *sorted()* function, these two functions take one argument that is iterable. These functions will find the minimum and maximum values respectively, so the output would be:

Output:

```
1
66
```

pow() and abs()

```
num1 = -6
num2 = 3

power = pow(num1, num2) #num1 is base, num2 is exponent
absolute = abs(power)

print(power)
print(absolute)
```

The example above shows examples of the *pow()* and *abs()* functions. The *pow()* function takes two arguments, the base and the exponent respectively, so the value of *power* is -6^3. On the other hand, *abs()* gives the absolute (positive) value. Therefore the output would be:

Output:

```
-216
216
```

zip()

```
list_a = [1,2,3,4,5]
list_b = [10,9,8,7,6]
```

```
for a,b in zip(list_a, list_b):
    print(a,b)
```

The example above shows an example of the *zip()* function, which takes two arguments (these have to be iterable). These are then zipped together through a *for loop*. The output would be:

Output:

```
1 10
2 9
3 8
4 7
5 6
```

Instructions

First read **example.py**. Open it using VS Code or Anaconda.

- **example.py** should help you understand some simple Python. Every task will have example code to help you get started. Make sure you read all of **example.py** and try your best to understand.

- You may run **example.py** to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.

Compulsory Task 1

Follow these steps:

- Create a new Python file in your folder called **float_manipulation.py**

- You will need to import the *math* module and use its built-in functions to complete this task.

- Write a program that starts by asking the user to input 10 floats (these can be a combination of whole numbers and decimals). Store these numbers in a list.

- Find the total of all the numbers and print the result.
- Find the index of the maximum and print the result.
- Find the index of the minimum and print the result.
- Calculate the average of the numbers and round off to 2 decimal places. Print the result.
- Find the median number and print the result.

Compulsory Task 2

Follow these steps:

- Create a new Python file in this folder called **jokes.py**
- You are going to create a random joke generator using Python's *random* module. This will require a bit of research on your part. For more information on the *random* module look **here**.
- Create a list of jokes.
- Use the *random* module to display a random joke each time the code runs.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.