HyperionDev

Programming With Built-In Functions

Task

Visit our website

Introduction

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 are also sometimes referred to as “methods” (if you have used Java before, functions in Python are very similar to Java methods).

Functions can be classified into three types:

1. **Built-in functions:** These are functions that come pre-packaged with Python and are readily available for use.
2. **External functions:** These are functions that you import from external modules or libraries. Python offers a vast ecosystem of modules that provide additional functionality, such as mathematical operations, data manipulation, and web development.
3. **User-defined functions:** These are functions that you create yourself to perform specific tasks. They allow you to define your own logic and behaviour.

In this task, we'll look into built-in functions and explore how to import external modules to expand your Python programming capabilities. We will explore user-defined functions in a later task.

Functions

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

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` method, i.e., `outputfile.write(name+"\n")`, is in fact also a function, implemented by some programmers, 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 function that you need to complete a task. Some of these functions come with the Python installation while others you have to install separately.

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.

Commonly used built-in functions

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

1. `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 two decimal places, so the output will be `66.66`.

2. `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.

3. `min()` and `max()`

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

   Output:

   ```
   1
   66
   ```

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

4. `pow()` and `abs()`

```
num1 = -6
num2 = 3

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

print(power)
print(absolute)
```

Output:

```
-216
216
```

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 the power is `-6`^{3}. On the other hand, `abs()` gives the absolute (positive) value.

5. `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)
```

Output:

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

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.

Importing external 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. You should always place `import` statements at the top of your code file, as per the **PEP 8 guidelines**.

```python
import math

# Calculate the circumference of a circle
radius = 5
circ = 2*math.pi*radius
print(f"The circle's circumference is about {round(circ,2)}.")
```

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, read through **its documentation**. 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.

Instructions

Read and run the accompanying **example files** provided before doing the task to become more comfortable with the concepts covered in this task.

Take note

The tasks below are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select “Request Review”, the task is automatically completed, and you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you’ve done that, feel free to progress to the next task.

Auto-graded task 1

1. Create a new Python file in your folder called float_manipulation.py.
2. You will need to import the statistics module and use its built-in functions to complete this task.
3. 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.
4. Find the total of all the numbers and print the result.
5. Find the index of the maximum and print the result.
6. Find the index of the minimum and print the result.
7. Calculate the average (mean) of the numbers and round off to two decimal places. Print the result.
8. Find the median number and print the result.

Auto-graded task 2

1. Create a new Python file in this folder called jokes.py.
2. You are going to create a random joke generator using Python's random module. This will require a bit of research on your part.
3. Create a list of jokes that include their punchlines.
4. Use the random module to display a random joke each time the code runs.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.