Hyperiondev

TASK

Beginner Programming with Functions — Defining Your Own Functions

Visit our website

Introduction

WELCOME TO THE BEGINNER PROGRAMMING WITH FUNCTIONS — DEFINING YOUR OWN FUNCTIONS TASK!

This task reintroduces functions and will focus on teaching you how to create your own functions. It will also show you how functions can be used to compute certain values using list elements and text file contents.

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!

HOW TO DEFINE A FUNCTION

A function can be defined as follows:

```
def add_one(x): # function called add_one has one parameter, x
    y = x + 1
    return y
```

The function that has been created in the example above is named *add_one*. It takes the parameter **x** as input. A *parameter* is a variable that is declared in a function definition. *Parameters* store the data needed to perform the logic that the function specifies. *Parameters* are filled when data is passed to the function as an *argument* when the function is called (which you will learn about soon). The code indented under *def add_one* is the *logic* of the function. It defines what happens when the function is called. You can pretty much do anything you want in a function. You can create new data structures, use conditionals etc.

The function in the example above computes a new variable, **y**, which is the value stored in variable **x** with 1 added to it. It then 'returns' the value **y**.

The general syntax of a function in Python is as follows:

```
def functionName(parameters):
      statements
      return (expression)
```

THE *DEF* AND *RETURN* KEYWORDS

Note the *def* keyword. Python knows you're defining a function when you start a line with this keyword. After the keyword, *def*, put a function name, its input parameters, and then a colon, with the logic of the function indented underneath.

Note the *return* keyword. A function doesn't have to include a *return* statement. The value after the keyword *return* will be returned/passed back to whatever code called the function, at which point the execution of that block of code stops.

CALLING A FUNCTION

It is good practice to define all your functions at the top of your code and the 'calling' of them below. Call a function by using the function's name, followed by the values you would like to pass to the parameters within parentheses. The values that you pass to the function are referred to as *arguments*.

Here is an example of calling our **add_one** function, and assigning the value returned from it to a variable:

```
def add_one(x): # function called add_one has one parameter, x
    y = x + 1
    return y
result_num = add_one(5) #create a variable and set it to the value returned by
the function. When the function is called, 5 is passed in as an argument.
print(result_num) #prints 6
```

Now let's look at another example (below) showing something very similar. Here, the **add_one** function is called again. In this example, we pass the value '10' as an argument to the function. As we created a parameter called **x** when we defined the function **add_one**, passing the argument 10 to the function will result in the parameter **x** being assigned the value 10.

```
num_plus_one = add_one(10)
# The result that the 'add_one' function returns is stored in the num_plus_one
variable.

print ("10 plus 1 is equal to: " + str(num_plus_one)+".")

# Or even
num = 10
print ("10 plus 1 is equal to: " + str(add_one(num))+".")
```

Think of a call to the function (e.g. **add_one(num)**), as a 'placeholder' for some computation. The function will run its code and return its result in that place.

You can define a function, but it will not run unless called somewhere in the code. For example, although we have defined the function **add_one** above, the code indented underneath it would never be executed unless another line that calls **add_one** with the command **add_one(some_variable)** is added somewhere in the main body of your code.

FUNCTION PARAMETERS

In the function definition, the parameters are between the parentheses after the function name. You can have more than one of these variables or parameters — simply separate them by commas. When you call a function, you place the value you would like to pass to the function as an *argument* in parentheses after the

function name, e.g.

```
result_num = add_one(5)
```

(If we didn't *do something* with the returned value, such as set up a variable to hold it, or a print statement to output it, what do you think would happen?)

FROM SEQUENTIAL TO PROCEDURAL PROGRAMMING

A major switch has happened when we introduced functions. Before, all your programs have been sequential. This means that code is always executed in the same order in which we read it; from the top of the file to the bottom. With functions, we lose this. You can define a function anywhere in your file, but it will not run unless it is called somewhere. This means that the statements in your code are no longer necessarily executed in the same order that they are written.

WHY USE FUNCTIONS?

There are many benefits to using functions:

- Creating functions allows you to have **reusable code**. There are many tasks that, as a programmer, you may need to code over and over again. For example, say you wrote several lines of code that, given a filename, can open the file, read its contents and print out its contents to the screen. It may be useful to 'save' that code somewhere so you could easily reuse it. A programmer can define a function, named something like `read_file`, that would encode this logic. That way, the next time they need to read the contents of a file they 'call'(use) the function `read_file`. This will *return* the result of that function, which in this case will result in the output being printed to the screen.

- Functions also make **error checking/validating your code easier**. Each module can be tested separately, possibly by different developers.

- Functions **divide your code up into manageable chunks** — to make the code easier to understand and troubleshoot.

- Modular programming can also lead to **more rapid application development**. Each module (set of functions) can be coded by a different developer or team of developers. This means that many modules can be developed at the same time, increasing the speed at which applications can be developed. Also, existing modules can be reused in new applications, which also leads to more rapid software development.

- Using functions can also make it **easier to maintain applications**. If a part

of a system needs to be updated, the whole program doesn't need to be modified. Instead, just the necessary function or functions can be changed.

SCOPE

Scope is what we call a program's ability to find and use variables in a program. The rule of thumb is that a function is covered in one-way glass: it can see out, but no one can see in. This means that a function can call variables that are outside the function, but the rest of the code cannot call variables that are defined within the function. Let's look at an example:

```
def adding(a, b):
    total = a + b
    return (description + str(total))

x = 2
y = 3
description = "Total: "

sum = adding(x, y)

print(sum)
```

Output:

```
Total: 5
```

In the code above, the function makes use of the *description* variable inside the function. This shows that the function can look outside and use variables from outside the function. Now let's see what happens if we put *description* inside the function:

```
def adding(a, b):
    total = a + b
    description = "Total: "
    return (str(total))

x = 2
y = 3

sum = adding(x, y)
```

```
print(description + sum)
```

Output:

```
Exception has occurred: NameError
name 'description' is not defined
```

See how the program complains that it can't find the *description* variable? That's because of the 'one-way glass': the rest of the code can't see into the function and so does not know that a *description* variable exists.

DEFAULT VALUES

When creating your own functions, it is possible to create default arguments. Let's look at the example below:

```
def multiply(num1,num2=5):
    total = num1 * num2
    print(f"{num1} * {num2} = {total}")

times_5 = multiply(6)
print(times_5)
```

Here we have a function that multiplies two numbers and prints a string with the total. The default value of **num2** is 5, so when we call the function and give the argument 6, that means that **num1** = 6 - we don't need to give an argument for **num2**, it will default to 5. Therefore, when we print out **times_5**, the output will be:

```
6 * 5 = 30
```

However, it is possible to change the value of **num2**. Have a look at the example below:

```
def multiply(num1,num2 = 5):
    total = num1 * num2
    print(f"{num1} * {num2} = {total}")

times_7 = multiply(6, 7)
print(times_7)
```

Here, even though `num2` still has a default value of 5, we have overwritten that to give it a value of 7. Now, the output will be:

```
6 * 7 = 42
```

We could also call the function using keyword arguments so the order in which we write the arguments doesn't matter. For example, using the above function:

```
times_9 = multiply(num2=6, num1=9)
print(times_9)
```

Output:

```
9 * 6 = 54
```

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 Python file called **my_function.py** in your folder.
- Create your own function that prints all the days of the week.
- Create your own function that takes in a sentence and replaces every second word with the word “Hello”.

Compulsory Task 2

Follow these steps:

- Create a Python file called **holiday.py** in your folder.
- You will need to create four functions:
    - *hotel_cost* — This function will take the number of nights a user will be staying at a hotel as an argument, and return a total cost for the hotel stay (You can choose the price per night charged at the hotel).
    - *plane_cost* — This function will take the city you are flying to as an argument and return a cost for the flight (*Hint: use if/else if statements in the function to retrieve a price based on the chosen city*).
    - *car_rental* — This function will take the number of days the car will be hired for as an argument and return the total cost of the car rental**.**
    - *holiday_cost* — This function will take three arguments: the number of nights a user will be staying in a hotel, the city the user will be flying to, and the number of days that the user will be hiring a car for. Using these three arguments, you can call all three of the above functions with respective arguments and finally return a total cost for your holiday.
- Print out all the details about the holiday in a meaningful way!
- Try using your program with different combinations of input to show its compatibility with different options.

Compulsory Task 3

Follow these steps:

- Create a Python file called **amazon.py** in your folder.
- Write code to read the content of the text file **input.txt**.

- For each line in **input.txt**, write a new line in the new text file **output.txt** that computes the answer to some operation on a list of numbers.
- If the input.txt has the following:
    - Min: 1,2,3,5,6
    - Max: 1,2,3,5,6
    - Avg: 1,2,3,5,6
- Your program should generate **output.txt** as follows:
    - The min of [1, 2, 3, 5, 6] is 1.
    - The max of [1, 2, 3, 5, 6] is 6.
    - The avg of [1, 2, 3, 5, 6] is 3.4.
- Assume that the only operations given in the input file are min, max, and avg, and that the operation is always followed by a list of comma-separated integers.
- You should define the functions *min*, *max*, and *avg* that take in a list of integers and return the *min*, *max*, or *avg* of the list.
- Your program should handle any combination of operations and any length of input numbers.
- You can assume that the list of input numbers are always valid integers and that the list is never empty, so as not to have to write code to check for these cases.
- *Hint: there is something strange about the first line of* ***input.txt***.

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

Optional Bonus Task 1

Follow these steps:

- Change your program to also handle the operation **px** where **x** is a number from 10 to 90 and defines the **x** percentile of the list of numbers. For example:

  input.txt:
  min: 1,2,3,5,6
  max: 1,2,3,5,6
  avg: 1,2,3,5,6
  p90: 1,2,3,4,5,6,7,8,9,10
  sum: 1,2,3,5,6
  min: 1,5,6,14,24
  max: 2,3,9
  p70: 1,2,3

  Your **output.txt** should read:
  The min of [1, 2, 3, 5, 6] is 1
  The max of [1, 2, 3, 5, 6] is 6
  The avg of [1, 2, 3, 5, 6] is 3.4
  The 90th percentile of [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] is 9
  The sum of [1, 2, 3, 5, 6] is 17
  The min of [1, 5, 6, 14, 24] is 1
  The max of [2, 3, 9] is 9
  The 70th percentile of [1, 2, 3] is 2

To calculate the **x** percentile:

1. In an ordered list of values, multiply x percent by the number of values in the list. E.g. if you are trying to find the 90th percentile in a list of 10 values: 90/100 * 10 = 0.9 * 10 = 9.
2. If the value calculated above contains a decimal value, round the number up.
3. The number now gives you the index of the value that you want.
4. Do some more research to find out what percentiles are and why we use them.

Optional Bonus Task 2

Follow these steps:

- Create a new Python file in your folder called **optional_task.py**
- Write a program that will act as a calculator.
- Create functions named *add_num*, *subtract_num*, *multiply_num*, and *divide_num* that asks the user to enter two numbers and adds, subtracts, multiplies, or divides them, respectively.
- Print out the following menu and ask the user to input a number that corresponds to the option they would like to choose:
- Option 1 - add
- Option 2 - subtract
- Option 3 - multiply
- Option 4 - divide
- If the user enters 1 call the *add_num* function
- If the user enters 2 call the *subtract_num* function
- If the user enters 3 call the *multiply_num* function
- If the user enters 4 call the *divide_num* function
- Make sure that the result of the calculation is printed out to the screen.

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.