Hyperiondev

TASK

Programming with User-defined Functions

Visit our website

Introduction

WELCOME TO THE PROGRAMMING WITH USER-DEFINED FUNCTIONS TASK!

This task 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.

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. 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 that the **return** keyword doesn't have to be included in a function. The value after the **return** statement 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 file and 'call' them as needed later in the code file. Call a function by using the function's name, followed by the values you would like to pass to the parameters in 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 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.

Note 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 only be executed if 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 happens when you introduce functions to your code. 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 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 repeatedly. 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 can easily reuse it. A programmer can define a function named `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 and validating your code easier**. Each module can be tested separately, possibly by different developers.

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

- Modular programming can also lead to **more rapid application development**. A different developer or team of developers can code each module (set of functions). This means that many modules can be developed simultaneously, increasing the speed at which teams can develop applications. Also, developers can reuse existing modules in new applications, leading 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 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)
```

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)
```

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)
```

Output:

```
9 * 6 = 54
```

Instructions

- First, read and run the **example files** using VS Code or the IDE/editor of your choice. Feel free to write and run your own example code before doing the compulsory task to become more comfortable with the concepts covered in this task.

Compulsory Task 1

Follow these steps:

- Create a Python file called **holiday.py** in your folder. Your task will be to calculate a user's holiday cost, **including** the plane cost, hotel cost, and car rental cost.

- First, get the following user inputs:
    - `city_flight`: The city they will be flying to. (You can create some options for them. Remember each city will have different flight costs.)
    - `num_nights`: The number of nights they will be staying at a hotel
    - `rental_days`: The number of days that they will be hiring a car for.

- Next, create the following four functions:
    - `hotel_cost`: This function will take `num_nights` 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 `city_flight` as an argument and return a cost for the flight (**hint**: use if-elif-else statements in the function to retrieve a price based on the chosen city).
    - `car_rental`: This function will take `rental_days` as an argument and return the total cost of the car rental (you can choose the daily rental cost.)
    - `holiday_cost`: This function will take the three arguments `hotel_cost, plane_cost, car_rental`. 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 readable way!

- Try using your program with different combinations of input to show its compatibility with different options.

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.