Exemplar_ Functions

October 31, 2023

Exemplar: Functions

0.1 Introduction

In this lab, you will practice defining functions and returning values, writing clean code, and using comments to scaffold code.

As a data professional, you'll often need to reuse the same block of code more than once when you're writing Python code to automate a certain task. This is why functions are important. You can call functions whenever you need the computer to execute those steps. Python has built-in functions that have already been defined and also provides tools for users to define their own functions.

In this lab, you'll define and call a function that calculates the revenue a theater makes depending on the number of tickets sold, then test the function with a new value.

0.2 Tips for completing this lab

As you navigate this lab, keep the following tips in mind:

- `### YOUR CODE HERE ###` indicates where you should write code. Be sure to replace this with your own code before running the code cell.
- Feel free to open the hints for additional guidance as you work on each task.
- To enter your answer to a question, double-click the markdown cell to edit. Be sure to replace the "[Double-click to enter your responses here.]" with your own answer.
- You can save your work manually by clicking File and then Save in the menu bar at the top of the notebook.
- You can download your work locally by clicking File and then Download and then specifying your preferred file format in the menu bar at the top of the notebook.

0.3 Task 1: Analyze a function

In your work as a data analyst for a movie theater, you're asked to quickly calculate how much the theater has made in sales so far.

In the following cell, analyze the function named `total_sales` and make note of your observations.

You do not need to run this cell, but if you do it will not produce an output.

[1]:
```
# Define a function named `total_sales`
def total_sales():

    # Return a value
    return
```

Hint 1

You can refer back to what you've learned about functions.

Hint 2

When analyzing the function definition, make sure to observe the function body, which is the indented block of code after the function header.

Hint 3

Notice that the code comments on each line clearly explain how the function operates.

Code comments can also be used to scaffold code as you build it, since Python will not interact with the text captured as a comment.

0.4 Task 2: Add a parameter and calculations to the function

As you continue refining your code, imagine that the price of tickets is $7.99. Now you will need to add a parameter to the function definition and the calculation to the function body.

0.4.1 2a: Add a calculation to the function

In this task, update the `total_sales` function so it returns the value of the number of tickets `n` multiplied by the price `7.99`.

[2]:
```
# Add a parameter to the function
def total_sales(n):

    # Multiply the parameter by the ticket price to get total sales
    return n * 7.99
```

0.4.2 2b: Call the function

With the calculations added to the `total_sales` function, test it with a value of `59`.

Example:

```
[IN]  total_sales(59)
[OUT] 471.41
```

[3]:
```
# Call the function
total_sales(59)
```

[3]: 471.41

Hint 1

You can refer back to what you learned about functions and arithmetic operators.

Hint 2

To create a function, use the `def` keyword to define the function, then write the name of the function, followed by parentheses and a colon: `():`. This set of parentheses is where a parameter would be added.

The body of the function begins on the next line and is indented, conventionally by four spaces. This is where to write the function's operations (such as arithmetic calculations) and a `return` statement.

To call a function, type the function name followed by parentheses, with a parameter in the parentheses.

Hint 3

After defining the `total_sales` function, add a parameter `n` to the function.

In the body of the function, multiply the parameter by the ticket price to get total sales.

To call the function, replace `n` with the number of tickets in the function's parenthesis, then run the cell.

1 Task 3: Create a function that accepts two values

You learn that the price of tickets is more expensive for opening night, and now you need to create a function that will accept two values and return their total value.

1.0.1 3a: Refine the `total_sales` function

- Update your `total_sales` function so it accepts two parameters:
  - `price` for the price per ticket
  - `num_tickets` for the total number of tickets.
- Return the product of the two arguments. Use the Python `round()` function to round the returned value to two decimal places. Refer to the round() Python documentation for more information regarding this function.

[4]:
```
# Refine the `total_sales` function
def total_sales(price, num_tickets):
    return round(price * num_tickets, 2)
```

1.0.2 3b: Call the function

Then, call the function with the value of `15.99` for `price` and `1001` for the number of tickets sold (`num_tickets`).

Example:

```
[IN]  total_sales(15.99, 1001)
[OUT] 16005.99
```

[5]:
```
# Call the function with two values
total_sales(15.99, 1001)
```

[5]:
```
16005.99
```

Hint 1

Refer to what you learned about functions, arithmetic operators, and the `round()` function.

Hint 2

Defining up a function that accepts two parameters is a very similar process to defining a function with just one.

Begin by defining the function and naming it, but instead of a single parameter within the parentheses, place two parameters separated by a comma (`,`).

The body of the function is still on a new line and indented four spaces. The `round()` function is included here, and the calculation of the two parameters takes place within the `round()` function's parentheses.

When calling the new function, type the name of the function followed by a set of parentheses. Then, place the parameters that the function is operating on within the parentheses, separated by a comma.

Hint 3

Define a function called `total_sales` that calculates the total dollar value of the tickets sold based on the per-ticket price (`price`), and the number of tickets sold (`num_tickets`).

In the body of the function, multiply the parameters to get the total and round the calculation to two decimal places.

To call the function, write the name of the function on a new line and replace the parameters with the price per ticket and number of tickets sold.

1.1 Task 4: Test the function with different values

As prices and sales fluctate, your function will need to accept different values and still return the total sales.

In this task, practice calling your function with different values.

Example:

```
[IN]  total_sales(16.99, 1232)
[OUT] 20931.68

[IN]  total_sales(10.50, 1472)
[OUT] 15456.0

[IN]  total_sales(5.00, 349)
[OUT] 1745.0
```

[6]:
```
# Call `total_sales` with 16.99 price and 1232 num_tickets
total_sales(16.99, 1232)
```

[6]: 20931.68

[7]:
```
# Call `total_sales` with 10.50 price and 1472 num_tickets
total_sales(10.50, 1472)
```

[7]: 15456.0

[8]:
```
# Call `total_sales` with 5.00 price and 349 num_tickets
total_sales(5.00, 349)
```

[8]: 1745.0

Hint 1

You can refer to what you've learned about calling functions with multiple parameters.

Hint 2

Call the function by writing the name, followed by parenthesis. Write the two values you want to calculate inside the parenthesis, separated by a comma.

Hint 3

To run the function and calculate a new total, call `total_sales()` and write `16.99, 1232` inside the parenthesis.

1.2 Conclusion

What are your key takeaways from this lab?

- Python allows you to define and call functions that you create.
- The main components of a function are the definition statement, the body, and the output statement (usually `return` or `print`).
  - The function header includes the `def` keyword, followed by the name of the function, followed by parantheses, followed by a colon.
  - The function body includes an indented block of code that instructs the computer on what to do when the function is called.
  - The output statement is what the function should produce as a result when executed.

- Functions allow you to reuse common code.
  - Frequently used calculations can be saved to a function and reused without writing out the equation multiple times.
- Functions accept single or multiple parameters.
  - Functions can be customized to accept more than one parameter and perform calculations on them, which allows greater flexibility when working with fluctuating values.
- Including functions such as `round()` to the return statement helps keep calculated values tidy and easy to read.
- Scaffolding functions with code comments helps improve readibility.
  - Clearly commented code explains how the code works as well as the intended results.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.