INTRODUCTION TO FUNCTIONS

Introduction to Functions

In programming, as we start to write bigger and more complex programs, one thing we will start to notice is we will often have to repeat the same set of steps in many different places in our program.

Let's imagine we were building an application to help people plan trips! When using a trip planning application we can say a simple procedure could look like this:

```
1. Establish your origin and destination
2. Calculate the distance/route
3. Return the best route to the user
```

We will perform these three steps every time users have to travel between two points using our trip application. In our programs, we could rewrite the same procedures over and over (and over) for each time we want to travel, but there's a better way! Python gives us a useful concept called *functions*.

Functions are a convenient way to group our code into reusable blocks. A function contains a sequence of steps that can be performed repeatedly throughout a program without having to repeat the process of writing the same code again.

In this lesson, we are going to explore the idea of a function by slowly building out a Python program for our trip planning steps!

At the end of this lesson, you'll know how to:

- Write a function and return values from it.
- Allow functions to take custom input.
- Experiment with how functions access our other python code.

And much more!

Instructions

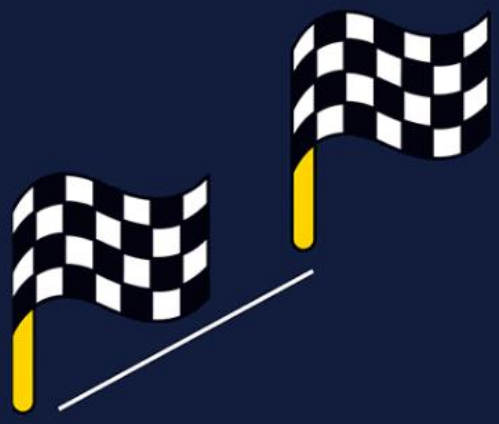
Review the visual for the function `navigation_steps()`.

Notice how the function `navigation_steps()` serves as a container for the three steps in the procedure and can be reused across multiple users as they plan their trips to different locations.

Click **Next** when you are ready to learn more about functions.

Function: navigation_steps()

1. Establish your starting + endpoints
2. Calculate the distance / route
3. Return the best route



Function: navigation_steps()

1. Establish your starting + endpoints
2. Calculate the distance / route
3. Return the best route

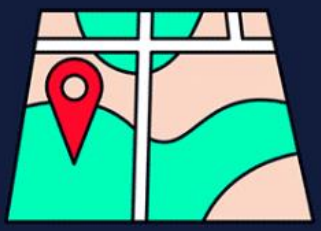
Function: navigation_steps()

1. Establish your starting + endpoints
2. Calculate the distance / route
3. Return the best route

Budapest

Function: navigation_steps()

1. Establish your starting + endpoints
2. Calculate the distance / route
3. Return the best route



New Delhi

Chiang Mai

Why Functions?

Let's come back to the trip planning application we just discussed in the previous exercise. The steps we talked about for our program were:

```
1. Establish an origin and destination
2. Calculate the distance/route
3. Return the best route
```

If we were to convert our steps into Python code, a very simple version that plans a trip between two popular New York tourist destinations might look like this:

```
print("Setting the Empire State Building as the starting point and
Times Square as our destination.")

print("Calculating the total distance between our points.")

print("The best route is by train and will take approximately 10
minutes.")
```

Anytime we want to go between these two points we would need to run these three print statements (for now we can assume the best route and time will stay the same).

If our program now had 100 new people trying to find the best directions between the Empire State Building and Times Square, we would need to run each of our three print statements 100 times!

Now, if you're thinking about using a loop here, your intuition would be totally right! Unfortunately, we won't be always traveling between the same two locations which means a loop won't be as effective when we want to customize a trip. We will address this in the upcoming sections!

For now, let's gain an appreciation for functions.

Instructions

1.

Run the pre-written `print()` statements to see what they output.

2.

Write the same set of print statements three more times. Run the code again and see the output.

Hint

Make sure that the three print statements are all duplicated three more times.

3.

Hopefully now you have some perspective about your life without functions!

In the next section, we will learn how we can refactor our code to utilize functions to reuse code.

Click ***Run*** your code again and then click ***Next*** to continue.

travel.py

```
# First user wants to travel between these two points!
print("Setting the Empire State Building as the starting point and Times Square a
s our destination.")
print("Calculating the total distance between our points.")
print("The best route is by train and will take approximately 10 minutes.")

# Second user wants to travel between these two points!
print("Setting the Empire State Building as the starting point and Times Square a
s our destination.")
print("Calculating the total distance between our points.")
print("The best route is by train and will take approximately 10 minutes.")


# Third user wants to travel between these two points!
print("Setting the Empire State Building as the starting point and Times Square a
s our destination.")
print("Calculating the total distance between our points.")
print("The best route is by train and will take approximately 10 minutes.")


# Fourth user wants to travel between these two points!
print("Setting the Empire State Building as the starting point and Times Square a
s our destination.")
print("Calculating the total distance between our points.")
print("The best route is by train and will take approximately 10 minutes.")
```

```
Setting the Empire State Building as the starting point
and Times Square as our destination.
Calculating the total distance between our points.
The best route is by train and will take approximately
10 minutes.
Setting the Empire State Building as the starting point
and Times Square as our destination.
Calculating the total distance between our points.
The best route is by train and will take approximately
10 minutes.
Setting the Empire State Building as the starting point
and Times Square as our destination.
Calculating the total distance between our points.
The best route is by train and will take approximately
10 minutes.
Setting the Empire State Building as the starting point
and Times Square as our destination.
Calculating the total distance between our points.
The best route is by train and will take approximately
10 minutes.
```

Defining a Function

A *function* consists of many parts, so let's first get familiar with its core - a function definition.

Here's an example of a function definition:

```
def function_name():
  # functions tasks go here
```

There are some key components we want to note here:

- The `def` keyword indicates the beginning of a function (also known as a function header). The function header is followed by a name in snake_case format that describes the task the function performs. It's best practice to give your functions a descriptive yet concise name.
- Following the function name is a pair of parenthesis `( )` that can hold input values known as parameters (more on parameters later in the lesson!). In this example function, we have no parameters.

- A colon `:` to mark the end of the function header.
- Lastly, we have one or more valid python statements that make up the function body (where we have our python comment).

Notice we've indented our `# function tasks go here` comment. Like loops and conditionals, code inside a function must be indented to show that they are part of the function.

Here is an example of a function that greets a user for our trip planning application:

```
def trip_welcome():
  print("Welcome to Tripcademy!")
  print("Let's get you to your destination.")
```

Note: Pasting this code into the editor and clicking ***Run*** will result in an empty output terminal. The `print()` statements within the function will not execute since our function hasn't been used. We will explore this further in the next exercise; for now, let's practice defining a function.

Instructions

1.

Two of the most common NYC attractions include the Empire State Building and Times Square.

In *travel.py*, we'll write a function that prints the directions via subway from the Empire State Building to Times Square.

First, define a function, `directions_to_timesSq()`. Leave the body of the function empty for now.

Note: When we run the code, we will see an error: `SyntaxError: unexpected EOF while parsing`. This will occur when we don't populate a function with any statements. We will populate it with code in the next step.

EOF stands for "End of File" — Python is telling you that it was expecting some code in the body of the function, but it hit the end of the file first.

Hint

Remember the core of a function - a definition. Check to make sure you have all the components for a function definition.

```
def my_function_name():
```

2.

Within the body of the function, use three print() statements to output the following directions:

```
Walk 4 mins to 34th St Herald Square train station
Take the Northbound N, Q, R, or W train 1 stop
Get off the Times Square 42nd Street stop
```

Remember, if you run your code, you shouldn't see any output in the terminal at this point. Check out the hint if you want to see how to see the output (we will be doing it in the next section as well!)

Hint

Check your statements for spaces, capitalization, and spelling. If you are still having problems getting the checkpoint to pass, try and copy and paste the text directly from the narrative into your print statement.

If you are interested in seeing the output, call your function like this:

```
directions_to_timesSq()
```

travel.py

```
# Your code below:
def directions_to_timesSq():
  print("Walk 4 mins to 34th St Herald Square train station")
  print("Take the Northbound N, Q, R, or W train 1 stop")
  print("Get off the Times Square 42nd Street stop")

directions_to_timesSq()
```

```
Walk 4 mins to 34th St Herald Square train station
Take the Northbound N, Q, R, or W train 1 stop
Get off the Times Square 42nd Street stop
```

Calling a Function

Now that we've practiced defining a function, let's learn about calling a function to execute the code within its body.

The process of executing the code inside the body of a function is known as calling it (This is also known as "executing a function"). To call a function in Python, type out its name followed by parentheses `( )`.

Let's revisit our `directions_to_timesSq()` function :

```
def directions_to_timesSq():
  print("Walk 4 mins to 34th St Herald Square train station.")
  print("Take the Northbound N, Q, R, or W train 1 stop.")
  print("Get off the Times Square 42nd Street stop.")
```

To call our function, we must type out the function's name followed by a pair of parentheses and no indentation:

```
directions_to_timesSq()
```

Calling the function will execute the `print` statements within the body (from the top statement to the bottom statement) and result in the following output:

```
Walk 4 mins to 34th St Herald Square train station.
Take the Northbound N, Q, R, or W train 1 stop.
Get off the Times Square 42nd Street stop.
```

Note that you can only call a function *after* it has been defined in your code.

Now it's your turn to call a function!

Instructions

1.
Call the `directions_to_timesSq()` function.

Click ***Run*** to see it execute and print out.

Hint
Make sure you call the function `directions_to_timesSq()` outside of the function definition. It should not be indented at all.

2.
Add an additional print statement to our `directions_to_timesSq()` function.

Have the statement print `"Take lots of pictures!"`

Run your code again and see how your output changes.

Hint
Remember to add the print statement inside of the function definition for `directions_to_timesSq()`. It should be indented and contain the text `"Take lots of pictures!"`. Make sure to place it after the other print statements in the function.

travel.py

```
def directions_to_timesSq():
  print("Walk 4 mins to 34th St Herald Square train station.")
  print("Take the Northbound N, Q, R, or W train 1 stop.")
  print("Get off the Times Square 42nd Street stop.")

  print("Take lots of pictures!")

# Call your function here:
directions_to_timesSq()
```

```
Walk 4 mins to 34th St Herald Square train station.
Take the Northbound N, Q, R, or W train 1 stop.
Get off the Times Square 42nd Street stop.
Take lots of pictures!
```

Whitespace & Execution Flow

Consider our welcome function for our trip planning application:

```
def trip_welcome():
  print("Welcome to Tripcademy!")
  print("Let's get you to your destination.")
```

The print statements all run together when `trip_welcome()` is called. This is because they have the same base level of indentation (2 spaces).

In Python, the amount of whitespace tells the computer what is part of a function and what is not part of that function.

If we wanted to write another statement outside of `trip_welcome()`, we would have to unindent the new line:

```
def trip_welcome():
  # Indented code is part of the function body
  print("Welcome to Tripcademy!")
  print("Let's get you to your destination.")

# Unindented code below is not part of the function body
print("Woah, look at the weather outside! Don't walk, take the
train!")

trip_welcome()
```

Our `trip_welcome()` function steps will not print `Woah, look at the weather outside! Don't walk, take the train!` on our function call. The `print()` statement was unindented to show it was not a part of the function body but rather a separate statement.

We would see the following output from this program:

```
Woah, look at the weather outside! Don't walk, take the train!
Welcome to Tripcademy!
Let's get you to your destination.
```

Lastly, note that the execution of a program always begins on the first line. The code is then executed one line at a time from top to bottom. This is known as *execution flow* and is the order a program in python executes code.

`Woah, look at the weather outside! Don't walk, take the train!` was printed before the `print()` statements from the function `trip_welcome()`.

Even though our function was defined before our lone `print()` statement, we didn't call our function until after.

Let's play around with indentation and the flow of execution!

Instructions

1.

We are going to help our trip planner users figure out if they should travel today based on the weather. Let's let our users know we can check the weather for them.

Write a `print()` statement that will output `Checking the weather for you!`.

Hint

To `print()` a string use the following syntax and replace `<Your string in here>` with the string you want to output:

```
print("<Your string in here>")
```

You can copy and paste the string from the checkpoint instructions to make sure there are no typos.

2.

We took a look outside and see a bright sunny day. Write a function called `weather_check()` that will `print` a message to our users that it's a great day to travel! The function should output:

```
Looks great outside! Enjoy your trip.
```

Note: Don't call your function just yet! We will do that in the next step.

Hint

Remember to use `def` before `weather_check():` to define the function and to indent the print statement to show that it is part of the function.

3.

Oh no! It looks like some clouds came in and it started raining. Our users shouldn't go on a trip in the rain. In our `weather_check()` function add a second `print()` statement under the first one which prints a warning message for our travelers! It should print:

```
False Alarm, the weather changed! There is a thunderstorm
approaching. Cancel your plans and stay inside.
```

Hint

Make sure to add another `print()` statement inside of your function under the first `print()` statement at the same level of indentation.

4.

Call the function `weather_check()`.

Hint

Be sure to add `weather_check()` at the bottom of your code without any indentation.

5.

Unindent your final print statement in your `weather_check()` function. Run the program again.

What is different?

Hint

Remember to unindent the front of the second `print` statement in your `weather_check()` function. You should notice a different order in your output terminal due to the change in the indentation.

travel.py

```
# Your code below:
print("Checking the weather for you!")

def weather_check():
  print("Looks great outside! Enjoy your trip.")
print("False Alarm, the weather changed! There is a thunderstorm approaching. Can
cel your plans and stay inside.")

weather_check()
```

```
Checking the weather for you!
False Alarm, the weather changed! There is a
thunderstorm approaching. Cancel your plans and stay
inside.
Looks great outside! Enjoy your trip.
```

Parameters & Arguments

Let's return to our `trip_welcome()` function one more time! Let's modify our function to give a welcome that is a bit more detailed.

```
def trip_welcome():
  print("Welcome to Tripcademy!")
  print("Looks like you're going to Times Square today.")

trip_welcome()
```

This will output:

```
Welcome to Tripcademy!
Looks like you're going to Times Square today.
```

Our function does a really good job of welcoming anyone who is traveling to Times Square but a really poor job if they are going anywhere else. In order for us to make our function a bit more dynamic, we are going to use the concept of function *parameters*.

Function parameters allow our function to accept data as an input value. We list the parameters a function takes as input between the parentheses of a function `( )`.

Here is a function that defines a single parameter:

```
def my_function(single_parameter)
  # some code
```

In the context of our `trip_welcome()` function, it would look like this:

```
def trip_welcome(destination):
  print("Welcome to Tripcademy!")
  print("Looks like you're going to " + destination + " today.")
```

In the above example, we define a single parameter called `destination` and apply it in our function body in the second `print` statement. We are telling our function it should expect some data passed in for destination that it can apply to any statements in the function body.

But how do we actually use this parameter? Our parameter of `destination` is used by passing in an *argument* to the function when we call it.

```
trip_welcome("Times Square")
```

This would output:

```
Welcome to Tripcademy!
Looks like you're going to Times Square today.
```

To summarize, here is a quick breakdown of the distinction between a parameter and an argument:

- The parameter is the name defined in the parenthesis of the function and can be used in the function body.

- The argument is the data that is passed in when we call the function and assigned to the parameter name.

Let's write a function with parameters and call the function with an argument to see it all in action!

Instructions

1.

We want to create a program that allows our users to generate the directions for their upcoming trip!

Create a function called `generate_trip_instructions()` that defines one parameter called `location`.

Note: Since we did not define any code in our function yet, we will receive an error in our output terminal. Don't worry, we will be filling in the code in the next step.

Hint

Function parameters must be defined in the parenthesis of our function definition:

```
def some_function(single_parameter):
  #some code
```

2.

`generate_trip_instructions()` should print out the following:

```
Looks like you are planning a trip to visit <location>
```

Where `<location>` will represent the `location` parameter.

Hint

Remember to concatenate the `location` to the end of the string in the `print` statement by using the `+` operator.

3.

`generate_trip_instructions()` should also let our users know they can reach their location using public transit.

Let's have `generate_trip_instructions()`also print out the following on a new line:

```
You can use the public subway system to get to <location>
```

Where `<location>` will represent the `location` parameter.

Hint

Remember to concatenate the `location` to the end of the string in the `print` statement by using the `+` operator.

4.

Time for some greenery! Let's see what happens when we call the function and input the argument `"Central Park"`, a backyard wonder in the heart of New York City.

Hint

Make sure that you call `generate_trip_instructions` outside of the function definition and that you pass `"Central Park"` as the argument to the function.

5.

The day trip is over and we need to get back to the train station to head home. Change the argument to `"Grand Central Station"` and call the function again.

What changed in the output?

Hint

Make sure that you call `generate_trip_instructions` outside of the function definition and that you pass `"Grand Central Station"` as the argument to the function.

travel.py

```
# Your code below:
def generate_trip_instructions(location):
  print("Looks like you are planning a trip to visit " + location)
  print("You can use the public subway system to get to " + location)

generate_trip_instructions("Central Park")
generate_trip_instructions("Grand Central Station")
```

```
Looks like you are planning a trip to visit Central
Park
You can use the public subway system to get to Central
Park
Looks like you are planning a trip to visit Grand
Central Station
You can use the public subway system to get to Grand
Central Station
```

Multiple Parameters

Using a single parameter is useful but functions let us use as many parameters as we want! That way, we can pass in more than one input to our functions.

We can write a function that takes in more than one parameter by using commas:

```
def my_function(parameter1, parameter2, parameter3):
  # Some code
```

When we call our function, we will need to provide arguments for each of the parameters we assigned in our function definition.

```
# Calling my_function
my_function(argument1, argument2)
```

For example take our trip application's `trip_welcome()` function that has two parameters:

```
def trip_welcome(origin, destination):
  print("Welcome to Tripcademy")
  print("Looks like you are traveling from " + origin)
  print("And you are heading to " + destination)
```

Our two parameters in this function are `origin` and `destination`. In order to properly call our function, we need to pass argument values for both of them.

The ordering of your parameters is important as their position will map to the position of the arguments and will determine their assigned value in the function body (more on this in the next exercise!).

Our function call could look like:

```
trip_welcome("Prospect Park", "Atlantic Terminal")
```

In this call, the argument value of `"Prospect Park"` is assigned to be the `origin` parameter, and the argument value of `"Atlantic Terminal"` is assigned to the `destination` parameter.

The output would be:

```
Welcome to Tripcademy
Looks like you are traveling from Prospect Park
And you are heading to Atlantic Terminal
```

Let's practice writing and calling a multiple parameter function!

Instructions

1.

Our travel application users want to calculate the total expenses they may have to incur on a trip.

Write a function called `calculate_expenses` that will have four parameters (in exact order):

1. plane_ticket_price
2. car_rental_rate
3. hotel_rate
4. trip_time

Each of these parameters will account for a different expense that our users will incur.

Note: Like before, we will see an error: `SyntaxError: unexpected EOF while parsing`, since there is no code in the body of the function. In the next step we will add statements to the function.

However, you can also add a `pass` statement inside your empty function and it will prevent that error. Remove the `pass` statement in the next step when you add code to your function.

Hint
Remember that the parameters go between the parentheses in the function definition and that they are separated by commas. The order of the parameters is also important!

2.
Within the body of the function, let's start to make some calculations for our expenses. First, let's calculate the total price for a car rental.

Create new variable called `car_rental_total` that is the product of `car_rental_rate` and `trip_time`.

Hint
Use `*` to perform multiplication between the two variables.

3.
Next, we want to apply the same logic but for our `hotel_rate`.

Create new variable called `hotel_total` that is the product of `hotel_rate` and `trip_time`.

We also have a coupon to give our users some cashback for their hotel visit so subtract `10` from that total in the same statement. Woohoo, coupons! 🎟️

Hint
Use `*` to perform multiplication between the two variables. Don't forget to subtract `10` after!

4.
Lastly, let's print a nice message for our users to see the total. Use print to output the sum of `car_rental_total`, `hotel_total` and `plane_ticket_price`.

Hint
Use `+` to perform the addition operation on the three variables.

5.
Call your function with the following argument values for the parameters listed:

- plane_ticket_price : 200
- car_rental_rate : 100

- hotel_rate : 100
- trip_time: 5

Hint
Your output should be:

```
1190
```

travel.py

```
# Write your code below:
#def calculate_expenses(plane_ticket_price, car_rental_rate, hotel_rate, trip_tim
e):
 #pass

def calculate_expenses(plane_ticket_price, car_rental_rate, hotel_rate, trip_time
):
  car_rental_total = car_rental_rate * trip_time
  hotel_total = hotel_rate * trip_time - 10
  print(car_rental_total + hotel_total + plane_ticket_price)

calculate_expenses(200, 100, 100, 5)
```

Types of Arguments

In Python, there are 3 different types of arguments we can give a function.

- Positional arguments: arguments that can be called by their position in the function definition.
- Keyword arguments: arguments that can be called by their name.
- Default arguments: arguments that are given default values.

Positional Arguments are arguments we have already been using! Their assignments depend on their positions in the function call. Let's look at a function called `calculate_taxi_price()` that allows our users to see how much a taxi would cost to their destination 🚕.

```
def calculate_taxi_price(miles_to_travel, rate, discount):
  print(miles_to_travel * rate - discount )
```

In this function, `miles_to_travel` is *positioned* as the first parameter, `rate` is positioned as the second parameter, and `discount` is the third. When we call our function, the position of the arguments will be mapped to the positions defined in the function declaration.

```
# 100 is miles_to_travel
# 10 is rate
# 5 is discount
calculate_taxi_price(100, 10, 5)
```

Alternatively, we can use *Keyword Arguments* where we explicitly refer to what each argument is assigned to in the function call. Notice in the code example below that the arguments do not follow the same order as defined in the function declaration.

```
calculate_taxi_price(rate=0.5, discount=10, miles_to_travel=100)
```

Lastly, sometimes we want to give our function arguments default values. We can provide a default value to an argument by using the assignment operator (=). This will happen in the function declaration rather than the function call.

Here is an example where the discount argument in our `calculate_taxi_price` function will always have a default value of 10:

```
def calculate_taxi_price(miles_to_travel, rate, discount = 10):
  print(miles_to_travel * rate - discount )
```

When using a default argument, we can either choose to call the function without providing a value for a discount (and thus our function will use the default value assigned) or overwrite the default argument by providing our own:

```
# Using the default value of 10 for discount.
calculate_taxi_price(10, 0.5)

# Overwriting the default value of 10 with 20
calculate_taxi_price(10, 0.5, 20)
```

Let's practice using these different types of arguments!

Instructions

1.

Tripcademy (our trusty travel app) needs to allow passengers to plan a trip (duh).

Write a function called `trip_planner()` that will have three parameters: `first_destination`, `second_destination` and `final_destination`.

Give the `final_destination` parameter a default value of `"Codecademy HQ"`.

Note: Since we did not define any code in our function yet, we will receive an error in our output terminal. Don't worry, we will be filling in the code in the next step.

Hint

Remember the structure of a multi-parameter function:

```
def my_function(param1, param2, param3):
  # Some code..
```

2.

First, we want to introduce the trip to users. Use `print()` in our function to output `Here is what your trip will look like!`.

Hint

Make sure to add the `print()` statement inside of your function by indenting it. Check spelling and capitalization for your string so it matches the one provided in the instructions.

3.

In our function definition let's provide an itinerary that will describe the destinations our user will visit in order. Print a statement that follows this form:

```
First, we will stop in <first_destination>, then
<second_destination>, and lastly <final_destination>
```

An example call to our function using positional arguments:

```
trip_planner("London", "India", "New Zealand")
```

Should output:

```
Here is what your trip will look like!
First, we will stop in London, then India, and lastly New Zealand
```

To test out your function, call `trip_planner()` with the following values for the parameters:

- first_destination: `"France"`
- second_destination: `"Germany"`
- final_destination: `"Denmark"`

Hint

Use `+` to concatenate strings together. Here is an example:

```
learning = "I am learning "
py_funcs = "Functions"
```

```
print(learning + py_funcs)
```
Would output:

```
I am learning Functions
```
Note: We can add an extra space at the end of learning to make sure the string is properly formatted when it is combined.

Your output should be:

```
Here is what your trip will look like!
First, we will stop in France, then Germany, and lastly Denmark
```
4.
Call the function `trip_planner()` again with the following values for the parameters:

- first_destination: `"Denmark"`
- second_destination: `"France"`
- final_destination: `"Germany"`

Note the difference in your output depending on the position of your arguments.

Hint
Your output should be:

```
Here is what your trip will look like!
First, we will stop in Denmark, then France, and lastly Germany
```
5.
Call the function `trip_planner()` again using keyword arguments in this exact order:

1. first_destination: `"Iceland"`
2. final_destination: `"Germany"`
3. second_destination: `"India"`

Hint
Make sure to enter the arguments in the order specified.

Remember, that when using keyword arguments, we explicitly refer to the assignment of each argument in the function call. Here is an example:

```
trip_planner(first_destination = "Disney World", final_destination
= "Legoland", second_destination = "Sea World")
```
6.
Lastly, go ahead and call the function `trip_planner()` using only two positional arguments to see the default argument in action:

1. first_destination: `"Brooklyn"`
2. second_destination: `"Queens"`

Hint

Your output should be:

```
Here is what your trip will look like!
First, we will stop in Brooklyn, then Queens, and lastly Codecademy
HQ
```

travel.py

```
# Write your code below:
def trip_planner(first_destination, second_destination, final_destination = "Code
cademy HQ"):
  print("Here is what your trip will look like!")
  print("First, we will stop in " + first_destination + ", then " + second_destin
ation + ", and lastly " + final_destination)

#Positional Arguments
trip_planner("France", "Germany", "Denmark")
trip_planner("Denmark", "France", "Germany")

#Keyword Arguments
trip_planner(first_destination="Iceland", final_destination="Germany", second_des
tination="India")

#Default Arguments
trip_planner("Brooklyn", "Queens")
```

```
Here is what your trip will look like!
First, we will stop in France, then Germany, and lastly
Denmark
Here is what your trip will look like!
First, we will stop in Denmark, then France, and lastly
Germany
Here is what your trip will look like!
First, we will stop in Iceland, then India, and lastly
Germany
Here is what your trip will look like!
First, we will stop in Brooklyn, then Queens, and
lastly Codecademy HQ
```

Built-in Functions vs User Defined Functions

There are two distinct categories for functions in the world of Python. What we have been writing so far in our exercises are called *User Defined Functions* - functions that are written by users (like us!).

There is another category called *Built-in functions* - functions that come built into Python for us to use. Remember when we were using `print` or `str`? Both of these functions are built into the language for us, which means we have been using built-in functions all along!

There are lots of different built-in functions that we can use in our programs. Take a look at this example of using `len()` to get the length of a string:

```
destination_name = "Venkatanarasimharajuvaripeta"

# Built-in function: len()
length_of_destination = len(destination_name)

# Built-in function: print()
print(length_of_destination)
```

Would output the value of `length_of_destination`:

```
28
```

Here we are using a total of two built-in functions in our example: `print()`, and `len()`.

And yes, if you're wondering, that is a real railway station in India!

There are even more obscure ones like `help()` where Python will print a link to documentation for us and provide some details:

```
help("string")
```

Would output (shortened for readability):

```
NAME
    string - A collection of string constants.

MODULE REFERENCE
    https://docs.python.org/3.8/library/string

    The following documentation is automatically generated from the
Python
    source files.  It may be incomplete, incorrect or include
features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at
the
    location listed above.
.....
```

Check out all the latest built-in functions in the official Python docs.

Let's practice using a few of them. You will need to rely on the provided Python documentation links to find your answers!

Instructions

1.

We were provided a list of prices for some gift shop items:

- T-shirt: 9.75
- Shorts: 15.50
- Mug: 5.99
- Poster: 2.00

Create a variable called `max_price` and call the built-in function `max()` with the variables of prices to get the maximum price.

Print `max_price`.

Hint

The `max()` built-in function takes in a series of consecutive arguments and returns the max value. Here is an example:

```
max_value = max(1, 4, 5, 9, 6)
print(max_value)
```
Would output:

```
9
```
For our example, pass the variable names for our prices in as the arguments to get the max value.

2.

Using the same set of prices, create a new variable called `min_price` and use the built-in function `min()` with the variables of prices to get the minimum price.

Print `min_price`.

Hint

The `min()` built-in function takes in a series of consecutive arguments and returns the min value. Here is an example:

```
min_price = min(4, 1, 5, 9, 6)
print(min_price)
```
Would output:

```
1
```
For our example, pass the variable names for our prices in as the arguments to get the min value.

3.

Use the built-in function `round()` to round the price of the variable `tshirt_price` by one decimal place.

Save the result to a variable called `rounded_price` and print it.

Hint

The `round()` built-in function takes in two arguments. The first argument is the number we want to round, followed by an argument on how many decimal places we want to round it.

Here is an example:

```
rounded_zero = round(10.54, 0)
rounded_one = round(10.54, 1)

print(rounded_zero)
print(rounded_one)
```

Would output:

```
11.00
10.5
```

travel.py

```
tshirt_price = 9.75
shorts_price = 15.50
mug_price = 5.99
poster_price = 2.00

# Write your code below:
max_price = max(tshirt_price, shorts_price, mug_price, poster_price)
print(max_price)

min_price = min(tshirt_price, shorts_price, mug_price, poster_price)
print(min_price)

rounded_price = round(tshirt_price, 1)
print(rounded_price)
```

Variable Access

As we expand our programs with more functions, we might start to ponder, where exactly do we have access to our variables? To examine this, let's revisit a modified version of the first function we built out together:

```
def trip_welcome(destination):
  print(" Looks like you're going to the " + destination + " today.
")
```

What if we wanted to access the variable `destination` outside of the function? Could we use it? Take a second to think about what the following program will output, then check the result below!

```
def trip_welcome(destination):
  print(" Looks like you're going to the " + destination + " today.
")

print(destination)
```

Output Results

```
NameError: name 'destination' is not defined
```

If we try to run this code, we will get a NameError, telling us that `destination` is not defined. The variable `destination` has only been defined inside the space of a function, so it does not exist outside the function.

We call the part of a program where `destination` can be accessed its *scope*. The *scope* of `destination` is only inside the `trip_welcome()`.

Take a look at another example:

```
budget = 1000

# Here we are using a default value for our parameter of
`destination`
def trip_welcome(destination="California"):
    print(" Looks like you're going to " + destination)
    print(" Your budget for this trip is " + str(budget))

print(budget)
trip_welcome()
```

Our output would be:

```
1000
Looks like you're going to California
Your budget for this trip is 1000
```

Here we are able to access the `budget` both inside the `trip_welcome` function as well as our `print()` statement. If a variable lives outside of any function it can be accessed anywhere in the file.

We will be exploring the concept of *scope* more after this entire lesson but for now, let's play around!

Note: Working with multiple functions can be a bit overwhelming at first. Don't hesitate to use hints or even look at the solution code if you get stuck.

Instructions

1.
Our users want to be able to save a list of their favorite places in our travel application.

We have received a rough draft for this implementation from another coder, but there are some problems with variable scope which prevent it from working properly.

Take a second to understand what the program is doing and then hit **Run** the code to see the error.

Hint
The error is a bit scary. Don't worry we will fix it, just take a second to think about what might be wrong.

```
Traceback (most recent call last):
  File "travel.py", line 16, in <module>
    show_favorite_locations()
  File "travel.py", line 12, in show_favorite_locations
    print("Your favorite locations are: " + str(favorite_locations))
NameError: name 'favorite_locations' is not defined
```

2.
Looking at the error, it seems like the main issue is that `favorite_locations` is not defined. Why would our program not be able to see our beautiful `favorite_locations` variable?

Aha! It must be a scope issue. Fix the scope of `favorite_locations` so that both our functions can access it.

```
Traceback (most recent call last):
  File "travel.py", line 11, in <module>
    show_favorite_locations()
  File "travel.py", line 8, in show_favorite_locations
    print("Your favorite locations are: " + favorite_locations)
NameError: name 'favorite_locations' is not defined
```

Hint
Right now the scope of `favorite_locations` only allows the function `print_count_locations()` to access it. Where would we need to move it to allow both functions to access it?

travel.py

```
# This function will print a hardcoded count of how many locations we have.
```

```
favorite_locations = "Paris, Norway, Iceland"

def print_count_locations():
  print("There are 3 locations")

# This function will print the favorite locations
def show_favorite_locations():
  print("Your favorite locations are: " + favorite_locations)

print_count_locations()
show_favorite_locations()
```

```
There are 3 locations
Your favorite locations are: Paris, Norway, Iceland
```

Returns

At this point, our functions have been using `print()` to help us visualize the output in our interpreter. Functions can also *return* a value to the program so that this value can be modified or used later. We use the Python keyword `return` to do this.

Here's an example of a program that will `return` a converted currency for a given location a user may want to visit in our trip planner application.

```
def calculate_exchange_usd(us_dollars, exchange_rate):
  return us_dollars * exchange_rate

new_zealand_exchange = calculate_exchange_usd(100, 1.4)

print("100 dollars in US currency would give you "
+ str(new_zealand_exchange) + " New Zealand dollars")
```

This would output:

```
100 dollars in US currency would give you 140 New Zealand dollars
```

Saving our values returned from a function like we did with `new_zealand_exchange` allows us to reuse the value (in the form of a variable) throughout the rest of the program.

When there is a result from a function that is stored in a variable, it is called a *returned function value*.

Let's try to `return` some data in the exercises!

Note: Working with multiple functions can be a bit overwhelming at first. Don't hesitate to use hints or even look at the solution code if you get stuck.

Instructions

1.
Our travel application is getting really popular. Some of our users have posted on social media that it would be useful if our application could help them track their budget during trips. We want to help them track their starting budget and let them know how much they have left after an expense.

We have provided some starting code to get started. Take a second to understand the code and then click **Run** and take a look at the output.

2.
Let's create a new function called `deduct_expense()` that will take two parameters.

The first parameter will be `budget` and the second parameter will be `expense`. Our function will be taking in a budget value as well as the expense we want to subtract.

We will want our function to *return* the budget minus the expense our travelers are incurring.

Hint
Remember to use the keyword `return` followed by the subtraction of `expense` from `budget`.

3.
Looks like the most common expense our travelers are incurring is a t-shirt purchase.

Let's create a variable called `shirt_expense` and for now, we will give it a set value of `9` (We are not accounting for currency changes at the moment). Make sure this is defined outside of the functions in your script.

Hint
Define `shirt_expense` outside of all of the functions in your script and set it equal to `9`.

4.

Now that we have an expense to subtract, create a new variable called `new_budget_after_shirt` and set it to be the function call of `deduct_expense()`.

Pass our `current_budget` variable as the first argument and the `shirt_expense` variable as the second argument.

Hint

Here is an example of calling a function with two arguments:

```
my_function(argument1, argument2)
```

5.

Lastly, we want our users to see the remaining budget.

Call the provided `print_remaining_budget()` function, passing in `new_budget_after_shirt` as the only argument.

Hint

Make sure to call `print_remaining_budget()` outside of the function definitions and with `new_budget_after_shirt` as the argument.

6.

Great Job! This is the biggest program with functions we have built so far! Take a second to review your code and click **Run** one last time when you are ready to move on.

travel.py

```
current_budget = 3500.75

def print_remaining_budget(budget):
  print("Your remaining budget is: $" + str(budget))

print_remaining_budget(current_budget)

# Write your code below:
def deduct_expense(budget, expense):
  return budget - expense

shirt_expense = 9

new_budget_after_shirt = deduct_expense(current_budget, shirt_expense)

print_remaining_budget(new_budget_after_shirt)
```

```
Your remaining budget is: $3500.75
Your remaining budget is: $3491.75
```

Multiple Returns

Sometimes we may want to return more than one value from a function. We can return several values by separating them with a comma. Take a look at this example of a function that allows users in our travel application to check the upcoming week's weather (starting on Monday):

```
weather_data = ['Sunny', 'Sunny', 'Cloudy', 'Raining', 'Snowing']

def threeday_weather_report(weather):
  first_day = " Tomorrow the weather will be " + weather[0]
  second_day = " The following day it will be " + weather[1]
  third_day = " Two days from now it will be " + weather[2]
  return first_day, second_day, third_day
```

This function takes in a set of data in the form of a list for the upcoming week's weather. We can get our returned function values by assigning them to variables when we call the function:

```
monday, tuesday, wednesday = threeday_weather_report(weather_data)

print(monday)
print(tuesday)
print(wednesday)
```

This will print:

```
Tomorrow the weather will be Sunny
The following day it will be Sunny
Two days from now it will be Cloudy
```

Let's practice using multiple returns by `return`ing to our previous code example.

Instructions

1.

Our users liked the previous functionality that we added to our travel application, but recently we have had an influx of users planning trips in Italy. We want to create a small function to output the top places to visit in Italy. Another member of our team already started on the implementation of this feature but it is still missing a few key details.

Take a second to review the code and click ***Run*** when you are ready to move on. For now, there will be no output.

2.

We want to be able to `return` the three most popular destinations from our function `top_tourist_locations_italy()`.

Add a line in the function `top_tourist_locations_italy()` that will return the values of `first`, `second`, `third` in that exact order.

Hint

A function that returns multiple values would like like this:

```
my_function():
  a = 1
  b = 2
  c = 3
  return a, b, c
```

3.

In order to use our three returned values from `top_tourist_locations_italy()` we need to assign them to new variables names after we call our function.

Set the return of the function `top_tourist_locations_italy()` to be equal to three new variables called `most_popular1`, `most_popular2`, and `most_popular3` in that exact order.

Hint

To set multiple return value variables from a function, declare them with commas in between and set them to the function call. For example:

```
return1, return2, return3 = multi_return()
```

4.

Use three `print()` statements to output the value of `most_popular1`, `most_popular2`, and `most_popular3`.

Hint

Your output should be:

```
Rome
Venice
Florence
```

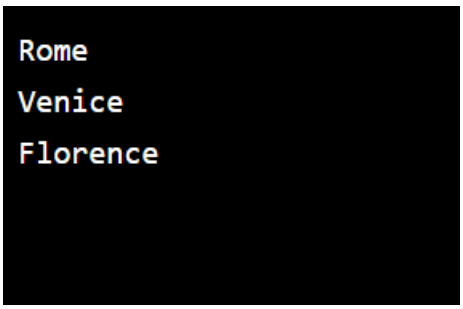
travel.py

```
def top_tourist_locations_italy():
  first = "Rome"
  second = "Venice"
  third = "Florence"
```

```
  return first, second, third

most_popular1, most_popular2, most_popular3 = top_tourist_locations_italy()

print(most_popular1)
print(most_popular2)
print(most_popular3)
```



Review

Excellent work! 👏 In this lesson, you've covered a major fundamental programming concept used in the majority of all programs to organize code into reusable blocks. To recap, we explored:

- What problems arise in our programs without functions
- What functions are and how to write them
- How whitespace plays an important role in how a program will execute our code and more importantly distinguish function code from non-function code
- How to pass input values into our functions using parameters and arguments
- The difference between built-in and user-defined functions and some common built-in functions python offers
- How we can reuse function output with the `return` keyword, as well as multiple returns.
- Where variables can be accessed in our programs that use functions

Let's practice putting all these concepts together!

Instructions

1.

Alright, this is it. We are going to use all of our knowledge of functions to build out TripPlanner V1.0.

First, like in our previous exercises, we want to make sure to welcome our users to the application.

Create a function called `trip_planner_welcome()` that takes one parameter called `name`. The function should use `print()` to output a message like this:

```
Welcome to tripplanner v1.0 <Name Here>
```

Where `<Name Here>` represents the parameter variable of `name` we defined.

Call `trip_planner_welcome()`, passing your name as an argument.

Hint

Let's remember what a function declaration looks like with one parameter:

```
def function_name(single_parameter):
  # Some Code
```

2.

Next, we are going to define a function called `estimated_time_rounded()` that will allow us to calculate a rounded time value based on a decimal for our user's trip.

An example call for this function will look like this:

```
estimated_time_rounded(2.5)
```

Where `2` represents 2 hours and `.5` represents 30 minutes.

Define the function `estimated_time_rounded()` with a single parameter `estimated_time`. The function should do two things:

1. Create a variable called `rounded_time` that is the result of calling the `round()` built-in function on the parameter `estimated_time`.
2. Return `rounded_time`.

After the function definition, call `estimated_time_rounded()` with a decimal argument and save the result to a variable called `estimate`. Since this amount represents a time, be sure to use a positive number.

Hint

Here is how you would use the `round()` function and return it from a user-defined function:

```
def return_round():
  rounded_value = round(5.53)
  return rounded_value
```

3.

Next, we are going to generate messages for a user's planned trip.

Create a function called `destination_setup()` that will have four parameters in this exact order:

1. `origin`
2. `destination`
3. `estimated_time`
4. `mode_of_transport`

Give the parameter `mode_of_transport` a default value of `"Car"`. The program will error out if we run it since we have not defined a function body yet. Don't worry we will do that in the next step.

Hint

Here is an example of a four-parameter function definition with a default value for the last parameter:

```
def four_params(param1, param2, param3, param4="Default Value"):
  # Some code
```

4.

Next, we are going to write four `print()` statements in our function. The output on this function call should look like this:

```
Your trip starts off in <origin>
And you are traveling to <destination>
You will be traveling by <mode_of_transport>
It will take approximately <estimated_time> hours
```

Each of these `print()` statements use a different parameter from our function `destination_setup()`.

Note: The `estimated_time` parameter will come in the form of an integer. Make sure to use `str()` to convert the parameter in your print statement.

After the function definition, call `destination_setup()` with the following arguments:

- `origin` and `destination` should be string values representing the places you will travel from and to
- Use the `estimate` you created earlier for `estimated_time`

- If you will be traveling by a mode other than Car, be sure to overwrite the default value of `mode_of_transport`

Hint
Use `+` to concatenate a string variable with another string. For example:

```
world = "World"

hello_world = "Hello" + world
print(hello_world)
```

Would output:

```
Hello World
```

Since we know `estimated_time` will be a number, we will need to use the `str()` function to convert it to a string. Here is an example:

```
number = 101
print("You are taking a " + str(101) + " course"
```

Would output:

```
You are taking a 101 course
```

Make sure to double-check your spelling, punctuation, and capitalization. If you are still having issues, use the string provided in the exercise.

5.

Great job! 👏

We have successfully finished our first version of the trip builder application. Go ahead and try passing different values into your functions!

travel.py

```
def trip_planner_welcome(name):
  print("Welcome to tripplanner v1.0 " + name)

trip_planner_welcome("Andres")

def estimated_time_rounded(estimated_time):
  rounded_time = round(estimated_time)
  return rounded_time

estimate = estimated_time_rounded(2.5)

def destination_setup(origin, destination, estimated_time, mode_of_transport="Car
"):
```

```
  print("Your trip starts off in " + origin)
  print("And you are traveling to " + destination)
  print("You will be traveling by " + mode_of_transport)
  print("It will take approximately " + str(estimate) + " hours")

destination_setup("Boston", "New Jersey", estimate, "Train")
```

```
Welcome to tripplanner v1.0 Andres
Your trip starts off in Boston
And you are traveling to New Jersey
You will be traveling by Train
It will take approximately 2 hours
```