Data Scientist Codecademy

Python Fundamentals

Python Functions

Introduction to Functions

Introduction

When we start writing bigger and more complex programs, we notice that we repeat the same steps in many different places in our program.

For example: Let's say we are building an application to let users plan trips. When using a trip planning application we can say a simple procedure could look like this:

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

These steps will be repeated every time the users have to travel between two points using our trip application. We can rewrite the same procedures over and over again or use a better approach. We can use the concept of 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.

Why Functions?

Functions help us to refactor our code and increase reusability.

Defining a Function

A function consist of many parts, so let's first get familiar with its core – a function definition.

```
Def function_name():
```

```
#functions tasks go here
```

Some key components to take note are:

- 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 functions descriptive yet concise name.
- Following the function name is a pair of parenthesis `()` that can hold input values known as 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.

Like loops and conditionals, code inside a function must be indented to show that they are part of the function.

A function is not executed until called.

Calling a Function

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

A function can be called only after it has been defined in code.

Whitespace & Execution Flow

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

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.

Parameters and Arguments

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().

Our parameters get defined from the arguments passed while calling the function.

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.

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.

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

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.

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

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

trip_planner("France","Germany","Denmark")
trip_planner("Denmark","France","Germany")
trip_planner(first_destination="Iceland",final_destination="Germany",second_destination="India")
trip_planner("Brooklyn","Queens")
```

Output:

```
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 programmers.

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!

Variable Access

We call the part of a program where a variable can be accessed its scope. The scope of a function parameter is only inside the function.

For a variable defined outside a function, it can be accessed anywhere in the file.

Returns

Functions can also return a value to the program so that this value can be modified or used later. We use the keyword `return` to do this.

Saving the values returned from a function allows us to reuse the value 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.

Multiple Returns

We can return several values by separating them with a comma. We can get our returned function values by assigning them to variables by assigning them to variables when we call the function.

```
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

monday, tuesday, wednesday = threeday_weather_report(weather_data)

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