Functions in Python

In this lab we will work with functions in Python. Functions are reusable blocks of code that can be reused . They can accept parameters and return data.

For this lab we will create script files rather than enter statements at the Python interpreter (shell).

Part 1 - Create a Basic Function

In this part we will create a basic function that prints a greeting.

__1. Open a terminal and navigate to your working directory. (Following instructions in earlier labs this would be LabWorkPython. Use that directory if it exists. If not you can either create it, create another directory or use an existing directory. It doesn't really make a difference where you save the files as long as you remember where they are.)

__2. Open your text editor (gedit or nano were suggested earlier) and create a file named:

```
function1.py
```

__3. Add the following text to the file:

```
#!/usr/bin/env python3

def hello():
  print('Hello Python!')

hello()
```

The function signature starts with the keyword 'def' and the function's name followed by parenthesis. The signature ends with a colon ':'.

The code block that implements the function is indented from the function signature. In this case the print statement is the only line of code in that block. Later we will add more statements to the block and they will all need to be indented like print() is.

The last line in the file executes the function.

__4. Save the file and add execute permissions:

```
chmod +x ./function1.py
```

__5. Execute the file.

```
./function1.py
```

The function hello() will output a greeting:

```
Hello Python!
```

The function we wrote is very basic. It does not take any parameters and it does not return any values. Its only action is the side-effect of printing a greeting.

It is always a good idea to document your code. When writing functions we can add something called a 'docstring' as the first line in the function. The docstring includes a description of what the function does. A docstring is delimited by three quotes '''. Lets add a docstring to our function.

__6. Edit function1.py and add a docstring to the hello function (shown in bold in the code below). Notice that the docstring is indented to the same level as the print statement:

```
def hello():
  '''prints a greeting'''
  print('Hello Python!')
hello()
```

__7. Save and run the file. You should notice that nothing has changed. The docstring is ignored when the function is executed.

Part 2 - Add a Parameter

In this part we'll add a parameter and use its value inside the function.

__1. Open function1.py in your editor.

__2. Update the hello() function to take a parameter named 'person':

```
def hello( person ):
  print('Hello ', person)
hello()
```

__3. Now that the function takes a parameter we will have to supply one when we call it. Modify the line in the file that calls the function to pass in the name of a person:

```
hello( 'Scott' )
```

__4. Save and run the file. It should output:

```
Hello Scott
```

Part 3 - Calling the function from code

In our file we are calling the function once just to demonstrate how that works. Now we will add a list of people and call the hello function for each of them.

__1. Edit function1.py and replace the line that calls hello() with the following function:

```
def greet_team():
  people = ['Bob', 'Cindy', 'Scott', 'Tom', 'Mindy']
  for person in people:
    hello(person)
```

Note how the code in the for loop code block (the call to hello()) is indented from the main block of code for the greetings function which is indented from the function's signature.

__2. At the end of the file add a statement that calls the **greet_team**() method:

```
greet_team()
```

__3. Save and run the file. You should get the following output:

```
Hello Bob
Hello Cindy
Hello Scott
Hello Tom
Hello Mindy
```

__4. To make the greet_team more flexible lets pass the array of people in when we call greet_team(). This way we can greet different teams with the same function:

```
def greet_team(people):
  for person in people:
    hello(person)

people = ['Bob', 'Cindy', 'Scott', 'Tom', 'Mindy']
greet_team(people)
```

In refactoring the function we:

- Moved the people array outside the function
- Added a parameter named 'people' to the function's signature
- Passed the people array into greet_team when we called it.

__5. Save and run the file. Fix any errors that you find. Verify that the output is the same as it was earlier:

```
Hello Bob
Hello Cindy
Hello Scott
Hello Tom
Hello Mindy
```

Part 4 - Create a Function that Returns Data

In the real world it is more likely that we will have one large data set that includes all employees along with some information about what team they are on. In this part we will create a data structure like that and then a function that will output a list of employees for a given team.

__1. Open **function1.py** in an editor.

__2. Replace the current 'people' list with the following data structure:

```
employees = [ {'name':'Bob', 'team': 'IT' },
 {'name':'Cindy', 'team': 'HR' },
 {'name':'Scott', 'team': 'HR' },
 {'name':'Tom', 'team': 'HR' },
 {'name':'Mindy', 'team': 'IT' }]
```

__3. Add a function to the file, after the employees variable and before the call to greet_team called get_team_members. The function should take a parameter named 'team' and return a list with the names of the employees on the team matching the one passed into the function. There are several ways to implement such a function. The code below shows one possible solution:

```
def get_team_members(team):
  tmlist = []
  for employee in employees:
    if employee['team'] == team:
      tmlist.append(employee['name'])
  return tmlist
```

The last line of the function holds the return statment. We are returning a list that was constructed as we iterated the employees data structure. Before you move on make sure everything is indented properly.

__4. Add a line after this new function that calls it, passing in the value 'IT' and assigning the result to a variable named team_people

```
team_people = get_team_members('IT')
```

__5. Modify the call to greet_team() and replace the 'people' parameter with 'team_people':

```
greet_team(team_people)
```

__6. Save and run the file. Fix any errors that come up. The output you should get is:

```
Hello Bob
Hello Mindy
```

__7. Change the call for get_team_members and pass in 'HR' instead of 'IT'.

```
team_people = get_team_members('HR')
```

__8. Save and re-run the file. The output you get should be:

```
Hello Cindy
Hello Scott
Hello Tom
```

Part 5 - Using Variable Parameter Lists

In this part we are going to first refactor the greet_team method so that we can pass the name of the team to greet into it directly. After that we will modify greet_team again so it can take a variable list of teams to greet.

__1. Open function1.py in your editor.

__2. Modify the greet_team method so that it takes a team name instead of an array of people. Inside the function it should call the get_team_members function and then iterate over the resulting list of names, calling hello() for each one. Also lets add a print statement before we start calling hello to show which team we are greeting:

```
def greet_team(team):
  team_members = get_team_members(team)
  print( '-- Welcome', team, '--' )
  for person in team_members:
    hello(person)
    print('')
```

__3. Delete the following line from the file:

```
team_people = get_team_members('HR')
```

__4. Edit the call to greet_team to pass in 'IT' instead of team_people:

```
greet_team('IT')
```

At this point the complete code should look like this:

```
#!/usr/bin/env python3

def hello( person ):
  print('Hello', person)

def greet_team(team):
  team_members = get_team_members(team)
  print( '-- Welcome', team, '--' )
  for person in team_members:
    hello(person)
  print('')

employees = [ {'name':'Bob', 'team': 'IT' },
 {'name':'Cindy', 'team': 'HR' },
 {'name':'Scott', 'team': 'HR' },
```

```
 {'name':'Tom', 'team': 'HR' },
 {'name':'Mindy', 'team': 'IT' }]


def get_team_members(team):
  tmlist = []
  for employee in employees:
    if employee['team'] == team:
      tmlist.append(employee['name'])
  return tmlist

greet_team('IT')
```

__5. Save and run the file. Fix any errors that you find. It should print out the following greetings:

```
Hello Bob
Hello Mindy
```

Now that greet_team can be called with a team name lets have it accept a variable number of parameters so that if two teams names are passed in it will print greetings for members of both.

__6. Edit the greet_team method. Change the parameter so that it will accept a variable length parameter list. Inside the function that parameter will now be a list rather than a single team name. Loop over the parameter list and call the remaining code. The existing code will need to be indented further so that it appears in the loop block. The resulting greet_team function should look like this:

```
def greet_team(*teams):
  for team in teams:
    team_members = get_team_members(team)
    print( '-- Welcome', team, '--' )
    for person in team_members:
      hello(person)
    print('')
```

__7. Save and run the file. Fix any errors that you find. It should still print out the following greetings:

```
-- Welcome IT -
Hello Bob
Hello Mindy
```

__8. Modify the call to greet_team to pass in 'HR' as well as 'IT':

```
greet_team('IT', 'HR')
```

__9. Save and run the file. It should now be greeting both teams:

```
-- Welcome IT --
Hello Bob
Hello Mindy

-- Welcome HR --
Hello Cindy
Hello Scott
Hello Tom
```

__10. Close the Terminal.

Part 6 - Review

In this lab we created several functions that work together to print greetings for employees on various teams. Various features of functions were used. We passed parameters to functions, including a variable parameter list. Some functions returned data as well.