Review Module 03

In Module 03 we have covered the following topics...

- The concept of `flow controls` in Python.
- `Types` of flow control statements, that is.
  - conditionals
  - loops
  - control statements
- Use cases of conditionals, i.e., `if, if/else, and if/elif/else`.
- Use cases of `for loop` and `while loop`.
- Use cases of `break`, `continue`, and `pass`.
- Solved pratical examples of `flow controls` using Python

Functions in Python

A function is a block of code that performs a specific task. A function only runs when it is called. There are two important points for functions to remember.

- Create a function
- Call a function

Recommended Chapter 3
https://automatetheboringstuff.com/2e/chapter3/

Types of functions

There are two types of functions in Python programming:

- Standard library functions - These are built-in functions in Python that are available to use.
- User-defined functions - We can create our own functions based on our requirements.

Standard library functions

print(), input(), len(), int(), str(), float(), append(), ...

User defined functions

As a programmer, we need functions specific to our needs. In that case, we can create our own functions.

In [ ]:
```
# Syntax of a function
def function_name(parameters):
    """docstring"""
    statement(s)
    return (optional)
```

1: Keyword `def` that marks the start of the function header.
2: A function `name` to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.
3: Parameters `arguments` through which we pass values to a function. They are optional.
4: A colon `:` to mark the end of the function header.
5: Optional documentation string `docstring` to describe what the function does.
6: One or more valid python `statements` that make up the function body. Statements must have the same indentation level (usually 4 spaces).
7: An optional `return` statement to return a value from the function.

Types of user-defined functions

1: Functions without parameters/arguments
2: Functions with fixed number of parameters/arguments (positional & Keyword arguments)
3: Functions with default arguments
4: Functions with variable number of parameters/arguments (positional = `*args`, keyword = `**kwards`))

1: Functions without parameters/arguments

This type of function does not require any parameters.
All the commands to be executed are located inside the body of the function.

In [1]:
```
# Example 1
print('Hello World!')
print('Hello World!')
print('Hello World!')
```

```
Hello World!
Hello World!
Hello World!
```

In [2]:
```
# Example 1
# Creating a function
def greet():
    print('Hello World!')
    print('Hello World!')
    print('Hello World!')
```

In [3]:
```
# Calling the function
greet()
```

```
Hello World!
Hello World!
Hello World!
```

In [4]:
```
# Example 1
# Creating a function
def greet():
    print('Hello World!')
```

In [5]:
```
# Calling the function
greet()
greet()
greet()
```

```
Hello World!
Hello World!
Hello World!
```

In [6]:
```
# Example 1
# Creating a function
def greet():
    print('Hello World!')

for i in range(3):
    greet()
```

```
Hello World!
Hello World!
Hello World!
```

In [7]:
```
# Example 2
# Creating a function
def greet():
    print('Hello World!')

print('Outside of function')
```

```
Outside of function
```

In [8]:
```
# call the function
greet()
```

```
Hello World!
```

In [9]:
```
# Example 3
# Creating a function
def greet():
    print('Hello World!')

# call the function
greet()

print('Outside of function')
```

```
Hello World!
Outside of function
```

When the function is called, the control of the program goes to the function definition.

All codes inside the function are executed.

The control of the program jumps to the next statement after the function call.

`Note:` In python, the function definition should always be present before the function call, therwise, we will get an error

In [10]:
```
greet1()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23024/3954236873.py in <module>
----> 1 greet1()

NameError: name 'greet1' is not defined
```

In [11]:
```
# Example 4
# Creating a function
def greet1():
    """
    This function greets to
    the person.
    """
    fname = 'Henry'
    print("Hello, " + fname + ". Good morning!")

greet1()
```

```
Hello, Henry. Good morning!
```

In [12]:
```
# function call
greet()
```

```
Hello World!
```

In [13]:
```
# Example 5
# Creating a function
def greet():
    print('Hello Class!')

# call the function
greet()
```

```
Hello Class!
```

In [14]:
```
# call the function
greet()
```

```
Hello Class!
```

Time to think: Function call is coming first and function definition is coming second in the last example. Still it is working. Why?

In [15]:
```
# Example 6
def add_numbers():
    num1 = 5
    num2 = 10
    num3 = 15
    sum = num1 + num2 + num3
    print('Sum of given numbers is: ', sum)

add_numbers()
```

```
Sum of given numbers is:  30
```

2a: Functions with fixed number of positional arguments

In [16]:
```
# Example 1
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
```

In [17]:
```
greet('Tim')
```

```
Hello, Tim. Good morning!
```

In [19]:
```
# Example 2
def add_numbers(num1, num2, num3):
    sum = num1 + num2 + num3
    print(sum)

add_numbers(10, 15, 5)
```

```
30
```

In [21]:
```
# Example 2
def add_numbers(num1, num2, num3):
    sum = num1 + num2 + num3
    #print(sum)

print('Sum of given numbers is:', add_numbers(10, 15, 5))
```

```
Sum of given numbers is: None
```

In [22]:
```
# Example 2
def add_numbers(num1, num2, num3):
    sum = num1 + num2 + num3
    return sum

add_numbers(10, 15, 5)
```

Out[22]:
```
30
```

In [26]:
```
# Example 2
def add_numbers(num1, num2, num3):
    sum = num1 + num2 + num3
    return sum

print('Sum of given numbers is:', add_numbers(10, 15, 5))
```

```
Sum of given numbers is: 30
```

In [31]:
```
# Example 2
def add_numbers(num1, num2, num3):
    sum = num1 + num2 + num3
    #print(sum)
    return None

print('Sum of given numbers is:', add_numbers(10, 15, 5))
```

```
Sum of given numbers is: None
```

When creating a function you can specify what the function should evaluate to (or return) using the `return` keyword.

In [33]:
```
# Example 3
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""

    if num >= 0:
        return num
    else:
        return -num

print(absolute_value(-2))
```

```
2
```

In [34]:
```
print(absolute_value(-4))
```

```
4
```

2b: Functions with fixed number of keyword/name arguments

In [35]:
```
# Example 1
def add_numbers1(num1, num2, num3):
    sum = num1 + num2 + num3
    return sum

add_numbers1(5, 10, 15)
```

Out[35]:
```
30
```

In [36]:
```
# Example 2
def add_numbers1(num1, num2, num3):
    sum = num1 + num2 + num3
    return sum

add_numbers1(num3=5, num1=10, num2=15)
```

Out[36]:
```
30
```

In [37]:
```
# Example 3
def function1(child3, child2, child1):
```

```
    print("The youngest child is " + child3)

function1(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

```
The youngest child is Linus
```

In [38]:
```
# Example 4
def student(firstName,lastName,idNumber):
    return idNumber + " - " + firstName + " " + lastName

print(student("Justin", "Pilon", "000554433"))
```

```
000554433 - Justin Pilon
```

In [39]:
```
# Example 4
def student(idNumber, firstName, lastName):
    return idNumber + " - " + firstName + " " + lastName

print(student("Justin", "Pilon", "000554433"))
```

```
Justin - Pilon 000554433
```

In [40]:
```
# Example 4
def student(idNumber, firstName, lastName):
    return idNumber + " - " + firstName + " " + lastName

print(student(firstName="Justin", lastName="Pilon", idNumber="000554433"))
```

```
000554433 - Justin Pilon
```

3: Functions with default arguments

In [41]:
```
# Example 1
def my_function(country = "Canada"):
    print("I am from " + country)

my_function()
```

```
I am from Canada
```

In [42]:
```
my_function("Sweden")
my_function("India")
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Brazil
```

In [43]:
```
# Example 2
def add_numbers(a = 7,  b = 8):
    sum = a + b
    return sum

# function call with two arguments
print(add_numbers(2, 3))
```

```
#  function call with one argument
print(add_numbers(a = 2))

# function call with no arguments
print(add_numbers())
```

```
5
10
15
```

In [44]:

```
# Example 3
def classroom(classId,computers = True,classSize = 5):
    if (classSize < 5):
        return "Cannot justify offering this course."
    else:
        if (computers):
            return "Course will be offered with computers!"
        else:
            return "Course will be offered without computers."

# print(classroom(classSize = 25, classId = 123456789))
print(classroom(classSize = 2, computers = False, classId = 123456789))
# print(classroom(123456789, classSize = 1))
# print(classroom(123456789))
# print(classroom(123456789, computers = False, classSize = 6))
# print(classroom(123456789, computers = True, classSize = 6))
```

```
Course will be offered with computers!
```

4a: Functions with arbitrary number of positional arguments

In [45]:

```
# Example 1
def myIntro(*args):
    hi = 'My first name is '+ args[0]+ ' , last name is '+ args[1]
    print(hi)
```

In [46]:

```
myIntro('Keith', 'Walters')
```

```
My first name is Keith , last name is Walters
```

In [47]:

```
myIntro('Keith', 'Walters', 51)
```

```
My first name is Keith , last name is Walters
```

In [48]:

```
# Example 2

def find_sum(*numbers):
    result = 0
    for i in numbers:
        result = result + i
    return result

# function call with 4 arguments
print(find_sum(1, 2, 3, 10))
```

```
# function call with 2 arguments
print(find_sum(4, 9))
```

```
16
13
```

In [50]:

```
# Example 3
def average(*args):
    total = 0
    count = 0
    for i in args:
        total = total + i
        count = count + 1
    return total/count

print("Class Average: ", average(54,67,89,45,90,75, 100, 90, 80))
```

```
Class Average:  76.66666666666667
```

4b: Functions with arbitrary number of keyword arguments

In [51]:

```
# Example 1
def myIntro(**args):
    hello = 'My first name is '+ args['firstName']+ ' , last name is '+ args['lastName'
    print(hello)

myIntro(lastName='Trudeau', firstName='Justin', age=51)
```

```
My first name is Justin , last name is Trudeau, and my age is 51
```

In [52]:

```
# Example 2
def myIntro(**args):
    hello = "My first name is " + args["firstName"] + " and my last name is " + args["l
    print(hello)

myIntro(lastName='Trudeau', firstName='Justin', age=51)
```

```
My first name is Justin and my last name is Trudeau
```

Python Variable Scope

A variable scope specifies the region where we can access a variable.

A variable has eith `local` or `global` scope.

A variable created inside a function belongs to the `local` scope of that function, and can only be used inside of the function.

A variable created outside of a function belongs to the `global` scope of the function, and can be used inside and outside of the function.

A variable which is inside the function can get a global scope by using the `global` keyword.

In [53]:

```
# Example 1
def myFunc():
    z1 = 300
    print(z1)
```

```
myFunc()
```

```
300
```

In [54]:

```
# try to access variable z1 outside myfunc() function
print(z1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23024/2764727428.py in <module>
      1 # try to access variable z1 outside myfunc() function
----> 2 print(z1)

NameError: name 'z1' is not defined
```

In [55]:

```
x = 10
print(x)

x = 20
print(x)
```

```
10
20
```

In [56]:

```
# Example 2
def my_func():
    x = 10
    print("Value of x inside function:", x)

x = 20
my_func()

print("Value of x outside function:", x)
```

```
Value of x inside function: 10
Value of x outside function: 20
```

- **Time to think**: Is there any other way to check that these variables are different?

In [57]:

```
# Example 3
def my_func():
    x = 10
    print("Value of x inside function:", x)
    print(id(x))

x = 20
my_func()

print("Value of x outside function:", x)
print(id(x))
```

```
Value of x inside function: 10
2712800815696
Value of x outside function: 20
2712800816016
```

In [58]:
```
# Example 4
def my_func():
#     x = 10
    print("Value inside function:",x)
    print(id(x))

x = 20
my_func()
print("Value outside function:",x)
print(id(x))
```

```
Value inside function: 20
2712800816016
Value outside function: 20
2712800816016
```

In [59]:
```
# Example 5
def my_func():
    x1 = 10
    print("Value inside function:",x1)
    print(id(x1))

# x = 20
my_func()
print("Value outside function:",x1)
print(id(x))
```

```
Value inside function: 10
2712800815696
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23024/1520404328.py in <module>
      7 # x = 20
      8 my_func()
----> 9 print("Value outside function:",x1)
     10 print(id(x))

NameError: name 'x1' is not defined
```

The variable inside the function is local and is not available/accesible outsite

The variable outside the function is global and is also available/accessible inside the function

In [60]:
```
# Example 6
# global keyword
def my_func():
    global x1
    x1 = 10
    print("Value inside function:",x1)
    print(id(x1))

my_func()
print("Value outside function:",x1)
print(id(x1))
```

```
Value inside function: 10
2712800815696
```

```
Value outside function: 10
2712800815696
```

In [61]:

```python
# global
x = 1

def myFunction():
    print("(inside function) globalVar = ", x)
    global innerVar
    innerVar = 3
    print("(inside function) innerVar = ", innerVar)
    localVar = 2
    print("(inside function) localVar = ", localVar)

myFunction()

print("")
print("(outside function) globalVar = ", x)
print("(outside function) innerVar = ", innerVar)
print("(outside function) localVar = ", localVar)
```

```
(inside function) globalVar =  1
(inside function) innerVar =  3
(inside function) localVar =  2

(outside function) globalVar =  1
(outside function) innerVar =  3
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23024/2496871854.py in <module>
     15 print("(outside function) globalVar = ", x)
     16 print("(outside function) innerVar = ", innerVar)
---> 17 print("(outside function) localVar = ", localVar)

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

In [ ]: