1. Review on functions

A function, also referred as subroutine or procedure, is a portion of code that performs specific tasks when it is called from within a program. The main purpose of functions is to write ONCE the statements composing the functions so that they can be used MANY times.

In Python, function are defined in the following manner:

```python
def sayHello():
    print("Hello World!")
```

`def` is a Python keyword that tells Python that sayHello() is a function. The function written above contains a set of Python statements which are executed whenever that function is called. Note that the definition of the function does not cause the execution of the function. To execute the sayHello function, it must be called from within another program (or the Interactive window of IDLE) in the following manner:

```python
>>> sayHello()
Hello World!
```

In the example above, when you type sayHello() in the Interactive window Python understands that the name sayHello is a function (since you defined it earlier on). Then Python calls the function sayHello(). At this point the statements within the sayHello function are executed (in this case: print("Hello World!")). Then the function sayHello() terminates.

Once sayHello() completes, the program control returns back to the line after the call to it was made. In our example, the control will return to the Python Interactive window.

2. Parametrized functions:

Parametrized functions are the functions to which we can pass parameters/arguments as inputs. The function uses the input parameters/arguments to perform specific tasks. For example:

```python
def sayHello(aname):
    print("Hello", aname)
```

In this case, sayHello() prints out Hello followed by string contained in the variable aname which is provided as input parameter to it. Hence if we call this function:

```python
>>> param1 = "Fred"
>>> sayHello(param1)
```

the result of the call sayHello(param1) would be:

```python
>>> sayHello(param1)
Hello Fred
```

The variable with name “aname” in the function definition (i.e. def sayHello(aname)) is called the **formal parameter**. These (variable) names are known and accessible from within the body of the
function. The variable named “param1” that is passed by the caller (i.e. sayHello(param1)) is called the actual parameter. Note that the variable name param1, is NOT known to the function sayHello.

Once the function call is initiated, the calling program stops its execution at the point of function call. When the function sayHello is called (i.e. sayHello(param1)), the formal parameter of the function (in our case: aname) is assigned the value contained by the actual parameter in the function call. The body of the sayHello function is executed and then after its completion the control returns to the point just after the function call. The output here will be “Hello Fred”.

**Practice 1**

Write a function that takes temperature expressed in Fahrenheit degrees and prints the temperature in Celsius. Call that function thrice with different input parameters and note down the results.

**3. Functions that return a single value:**

We have so far seen functions which just do a specific task (prints hello) and do not return any value back to the caller. We might also come across situations where we need information back from a function. One way of getting any information back from a function is by using **return** statement.

```python
def calculateSum(input1, input2):
    result = input1 + input2
    return result
```

As inherent in the function name, calculateSum(input1, input2) function takes two parameters (input1 and input2 which are numbers) as its input parameters, calculates the sum of those two numbers and stores it in variable named “result” and returns back result using the “return” statement.

To call this function, we do the following in the interactive python console:

```python
>>> a = 10
>>> b = 20
>>> sum = calculateSum(a, b)
>>> print(sum)
30
```

Once the calculateSum() completes executing, the returned value is stored in “sum” and the program control will return to the next instruction after the function call which is the “print” statement. This program will output 30 in the Python interactive console.

**Practice 2**

Write a program that contains a function that takes the length of the side of a cube as input parameter and returns back its area. Print that area.

**4. Why use functions?**

You might wonder why we are using functions when we can write the code without using them. For example, if you consider calculateSum() function that is written above, an equivalent program can be:

```python
>>> a = 10
>>> b = 20
```
This program will have the same output as the calculateSum(a, b) function.

The answer to this question is quite intuitive. A function is often coded so that it can be started ("called") several times and/or from several places during a single execution of the program, including from other function, and then branch back (return) to the next instruction after the "call" once the function’s task is done. For instance, I can call calculateSum(a, b) several times and it will return back the sum as many times as it is called.

Thus, it saves us time, avoids redundancy of writing code within a program (instead of writing the same code again and again, just call the function), and makes the code more readable. Thus,

>>> sum1 = calculateSum(10, 20)
>>> sum2 = calculateSum(5, 10)
>>> print(sum1)
30
>>> print(sum2)
15

We can see from the example that we do not have to write the logic for calculating the sum again and again. We are just calling the function twice and that function is doing the job for us as many times as it is called with specific input parameters.