Prelab3

We encourage you to work together on the Pre Lab. The Pre Lab is not graded but will help you prepare for your lab session. If you have any questions on the material in the Pre lab, first check the book and recitation slides, if you do not find your answer please email your recitation TA or the course instructors.

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:

```
>>> 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.

References

You can read more introductory text on functions in the text book Chapter 4, section 1 & 2.

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:
>>>param1 = "Fred"
>>>sayHello(param1)

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

>>>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”.

References:

You can read detailed description of functions with parameters in the text book Chapter 4, section 3

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.

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:

>>>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.

References:

Please refer to textbook Chapter 4, section 4 for more on function returns.

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. Functions can return multiple values:

Python functions also have an ability to return back multiple values via return statements. For example,

```python
def prodDiv(input1, input2):
    prod = input1 * input2
    div = input1 / input2
    return prod, div
```

This function is called as:

```python
>>> a = 10
>>> b = 2
>>> product, division = prodDiv(a, b)
>>> print(product)
20
>>> print(division)
5
```

Thus prodDiv function can return multiple values at the same time. In this case it returns both the product and division of two numbers. The returned values are stored in variables product and division respectively. The output of the program will thus be 20, 5.

Practice 3

Write a function that takes the length of a line (in meters) as parameter and returns back the converted length both in inches and yards. Print both the returned values.

5. 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
>>> c = a + b
>>> print(c)
30
```
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,

```python
>>> 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.

6. Importing libraries to a Python program:

You may want to use some of the library functionality that python provides in your program to make your life easier. Libraries contain some useful definitions that you can use in your programs without writing your own function. One of the examples of a library that can be used in Python programs is the “math” library which contains all the mathematical functions such as calculating the square root (sqrt), sin, cos, tan of an angle passed to it, calculating the logarithm of a given value and so on.

To use a library in a python program, you need to import it first. Importing a library to a Python program can be done in two ways. One way is to use the “import” keyword as follows:

```python
import math
def calculateLog(x):
    return math.log(x)
```

Calling this method in the python interactive command line, we get:

```python
>>> calculateLog(math.e)
1.0
```

The python program given above imports the “math” library using “import” keyword. When a library is imported, all its functions and constants become accessible to that python program. Thus inside the calculateLog(x) function, we have a statement that calls log function of math library which returns back the logarithm of parameter passed to it. math.e is a way to access the mathematical constant “e”. The syntax for calling the functions of a library is:

```python
<library_name>.<function_name>
```

Another method to import library functions and constants is to use a combination of “from” and
“import” keywords. The “from” statement allows you to load any set of functions from a library module. You can either list the name of functions you want to import or just use asterisk (*) to import everything from a module. For example, the code given above can also be written as:

```python
from math import *
def calculateLog(x):
    return log(x)
```

This will perform in the same way as the initial method of importing. Please note that if you import in this manner, you can directly call the library functions instead of using the dot notation (syntax given above). You can see that here we are using log(x) instead of math.log(x)

Another equivalent program can be:

```python
from math import log
def calculateLog(x):
    return log(x)
```

7. Math Library:

Math library is a set of useful mathematical definitions that can be used in your Python program. It contains some mathematical constants like e and pi, and also contains a lot of useful mathematical functions that can be used in mathematical calculations in your Python program.

References:

Refer to [http://docs.python.org/release/3.2/library/math.html](http://docs.python.org/release/3.2/library/math.html) (Python 3.2 documentation) to get the list of math library definitions.

Practice 4:

Write a python program to calculate the hypotenuse of a right angled triangle given its two perpendicular sides. Use the math library to calculate the squares and square roots of the sides. Try using the two methods of import given above. Hint: pow(x,y) returns back x raised to the power y

8. Strings "arithmetic"

In Python you can define complex arithmetic expressions by “combining” variables that hold numeric values with the familiar numeric operators (see the Numeric Operators Table ). You already saw examples of arithmetic expressions such as:

```python
>>> a = 10
>>> b = 50
>>> c = a+b
```

In addition to variables that hold numeric values, Python allows you to use string of characters, such as “Hello”, “Today is a beautiful day”, and so forth. A string of characters is delimited between a pair of “ symbols. You will see much more about strings in Week 4. At the most basic you can enter a string of character in the Python shell, and Python will display it:

```python
>>> "hello"
```
'hello'

You can pass a string of character as a parameter to a function, as in the case of the built-in function `print`:

```python
>>> print("Hello")
'Hello'
```

You can also assign a string of characters to a variable:

```python
>>> myName = "John"
```

Now the variable `myName` holds as value the string of characters “John”. If you now type the variable `myName` in the Python shell, Python will print ‘John’:

```python
>>> myName = "John"
>>> myName
'John'
```

You can change the value of the variable `myName` by assigning it a new string value:

```python
>>> myname = "Frank"
>>> myName
'Frank'
```

Another interesting thing you can do in Python with strings is to “sum”, that is, concatenate two (or more) strings:

```python
>>> "John"
'John'
>>> "Doe"
'Doe'
>>> "John"+"Doe"
'JohnDoe'
```

You can do that also on variables holding as a value a character string:

```python
>>> firstname ="John"
>>> lastname = "Doe"
>>> firstname+lastname
'JohnDoe'
```

And you can assign the result of the “sum” of two character string values to a third variable:

```python
>>> myFullName = firstname+lastname
>>> myFullName
'JohnDoe'
```
Hence, if you want to concatenate, say, 3 times the same character string, for example “Hello “, you can do that “summing” 3 times the same string:

```python
>>> "Hello " + "Hello " + "Hello 
'Hello Hello Hello '
``` 

This makes sense to Python, since it interprets it as: 1. get the first string, 2. concatenate the second string to it 3. concatenate the third string to the one obtained in 2. above

Or you can “multiply” a certain number of times the same string:

```python
>>> 3* "Hello 
'Hello Hello Hello '
``` 

Python interprets the example above as concatenate 3 times the string “Hello”.

Of course you can not “multiply” two strings:

```python
>>> "Hello" *"Hello"
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    "Hello" *"Hello"
TypeError: can't multiply sequence by non-int of type 'str'
``` 

**High level Intuition:**

In this prelab we have discovered functions. We came across different types of functions. Some functions do not take any argument, while some functions take parameters/arguments. Some functions do not return any value, while other functions may return one or multiple values. We also found out the reason the functions are defined and implemented in a program. Without functions our code will become redundant, unreadable(messy) and lengthy. Functions give us a great way to call a specific set of code again and again by just calling it from a different point in a program. Another useful thing learned in this prelab is that there exists a lot of useful libraries which contain some predefined and useful definitions which can be used in your code to make life easier instead of you implementing them by yourself. To use these library functions we need to import the library first and only then we can use its definitions. Math library is a great tool in python which can be used in different situations where mathematical calculations are needed.