Prelab 8
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 subroutines or procedures is a portion of code within a large program that performs specific tasks when it is called from within that program.

In python, function can be defined in the following manner:

```python
defsayHello():
    print("Hello World!")
```

def is a keyword in python which tells python that sayHello() is a function. The function written above contains a set of tasks/instructions which are executed whenever that function is called. Once the function completes its execution, the control of the program returns back to the next instruction after the function call. We can invoke () function in the following manner:

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

Once sayHello() completed, the program control returns back to the line after function call is made (which is end of the program in this case). Hence, output when sayHello() is called, will print out “Hello World” in the Python interactive console and then exit.

References
You can read more introductory text on functions in the text book Chapter 6, section 6.2

2. Parametrized functions:
Parametrized functions are the functions to which we can pass parameters/arguments as inputs to, which it can use to perform specific tasks. For example,

```python
sayHello(name):
    print("Hello", name);
```

In this case, sayHello() prints out Hello with the name which is provided as input parameter to the sayHello(name) function. Hence if we call this function:

```python
>>>param1 = "Fred"
```
Here "param1" that is passed by the caller is called the actual parameter and the "name" in the function is called the formal parameter. Formal parameters are accessible within the body of the function whereas the actual parameters are not. Once the function call is initiated, the calling program stops its execution at the point of function call. The formal parameters of the function are assigned the values supplied by the actual parameters in the call (input is assigned the value supplied by param1), the body of function is executed and then after 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 6, section 6.4

**Practice 1**
Write a function that takes temperature as Fahrenheit and prints the temperature in Celsius. Call that function thrice with different input parameters and note down the results.

**3. Functions that return value:**
We have so far seen functions which just do a specific task (prints hello) and return back nothing 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 the function name indicates, 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.

References:
Please refer to textbook Chapter 6, section 6.5.1 and 6.5.2 for more details on returning values from a function.

Practice 2
Write a program that contains a function that takes the sidelength of a cube as a 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, comparing 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
```
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.

5. 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:

```
<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):
```
This will perform similar to 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 \( \text{math.log}(x) \)

Another equivalent program can be:

```python
from math import log

def calculate_log(x):
    return log(x)
```

**References:**
Please refer to textbook Chapter 4, section 4.3 to get detailed description of importing a library to a python program.

### 6. if Condition

A Decision is when a program has more than one choice of actions depending on a variable’s value. Think of a traffic light. When it is green, we continue our drive. When we see the light turn yellow, we reduce our speed, and when it is red, we stop. These are logical decisions that depend on the value of the traffic light. Python has a decision statement to help us when our application needs to make such decision for the user.

The most common decision statement type is the if statement. For example:

```python
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 23
>>> if(x > 0):
    print("The number is positive")
```

The following comparison operators can be used in expressions. They evaluate to True or False.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal</td>
</tr>
</tbody>
</table>
7. Range function

The `range()` function generates lists containing arithmetic progressions. For example,

```python
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The given end point is never part of the generated list; `range(10)` generates a list of 10 values.

It is also possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the 'step'):

```python
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(-10, -100, -30))
[-10, -40, -70]
```

8. For loops

Python’s `for` loop iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

For example, given a list

```python
a = [5, 6, 7, 8, 9]
```

Then use `for` loop to iterate over the items in `a`

```python
for num in a:
    print(num)
```

And the output is,

```
5
6
7
8
9
```

The `for` loop goes through each number in the sequence and sets `num` to each element in the sequence. Inside the body of the loop, each element is printed. The following example shows a function that uses a `for` loop to calculate the factorial of a number.

```python
def factorial(n):
    p = 1
    for n in range(1, n+1):
        p = p * n
    return p

>>> factorial(6)
```