PRELAB 4:

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. In the Pre Lab, you may find questions to answer. We do not require you to provide us the answers, but we do recommend you to try to answer these questions. 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. In this prelab you will learn some of the decision control features of Python.

Review

Booleans

Python, like many other computer programming languages, uses Boolean logic for its decision control. That is, Python compares one or more values in order to decide whether to execute a piece of code or not. Booleans are either True or False.

The following values are considered false:

- None
- False
- Zero of any numeric type, for example, 0, 0.0.
- Any empty sequence, for example, an empty string

All other values are considered true. Boolean Operations — and, or, not

These are the Boolean operations, ordered by ascending priority:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x or y</td>
<td>if x is false, then y, else x</td>
</tr>
<tr>
<td>x and y</td>
<td>if x is false, then x, else y</td>
</tr>
<tr>
<td>not x</td>
<td>if x is false, then True, else False</td>
</tr>
</tbody>
</table>

Conditionals (if, elif)

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 color. 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 = 10
if (x > 0):
    print("The number is positive")
```
The number *is* positive

\[(x > 0)\] in the example above is the condition that is evaluated. If the condition is true, then we will execute the print statement.

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>equal</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
</tr>
</tbody>
</table>

We can also specify an alternate course of action for the program if the condition in the if-statement is not satisfied. This can be done with the elif and else statements. elif stands for “else if,” which means that if the original ‘if’ condition is false and the elif condition is true, execute the block of code following the elif statement will be executed.

elif is used in so called multiway decisions (checking multiple conditions) where the different decisions are mutually exclusive. It is else and if combined which could be used to test another condition if the first condition is false, instead of just going to a default with else.

For example:

```python
#Asks for a number
#Prints if it is even or odd
number = 17
if (number % 2 == 0):
    print(number, "is even.")
elif number % 2 == 1:
    print(number, "is odd.")
else:
    print(number, "is very strange.")
```

**Nested Conditionals**

One conditional can also be nested within another.

For example,

```python
if (x == y):
    print(x, "and", y, "are equal")
else:
    if x < y:
        print(x, "is less than", y)
    else:
        print(x, "is greater than", y)
```

The outer conditional contains two branches. The first branch contains a simple output statement. The
second branch contains another if statement, which has two branches of its own. Those two branches are both output statements, although they could have been conditional statements as well.

Although the indentation of the statements makes the structure apparent, nested conditionals can become difficult to read very quickly.

Logical operators often provide a way to simplify nested conditional statements. For example, we can rewrite the following code using a single conditional:

```python
if(0< x):
    if x <10:
        print( "x is a positive single digit." )
```

The print statement above is executed only if we make it past both the conditionals, so we can use the and operator:

```python
if(0< x and x <10):
    print("x is a positive single digit number.")
```

### EXAMPLES

Here are other example functions demonstrating the usage of if, else, elif statements:

1) Write a function min(a, b, c) that takes 3 integers as input and returns the min of the 3 values.

```python
def min1(a,b,c):
    m = a;
    if (m > b):
        m = b;
    if (m > c):
        m = c;
    return m;
```

Method 2:

```python
def min2(a,b,c):
    if (a < b):
        if (a < c):
            return a;
        else:
            return c;
    if (b < c):
        return b;
    else:
        return c;
```

Output:

```python
>>> min1(4,5,6)
```
2) Write a function CheckTwo(x,y) that takes two integers values x, y and returns TRUE if x = 2y or y = 2x else it returns FALSE

```python
def CheckTwo(x, y):
    if x == 2*y or y == 2*x:
        return True
    else:
        return False
```

OUTPUT:

```
>>> CheckTwo(3, 4)
False
>>> CheckTwo(4, 8)
True
>>> CheckTwo(10, 5)
True
```

3) Write a function changeCase( c ) that takes an alphabet as input and does the following:

- If c is an uppercase alphabet, then converts it to lowercase and prints it
- If c is a lowercase alphabet, then converts it to uppercase and prints it
- Else prints a message “not an alphabet”

```
# ASCII values of uppercase alphabets lie between 65 to 90 ( ord('A') = 65, ord('Z')=90)
# ASCII values of lowercase alphabets lie between 97 to 122 (ord('a') = 97 and ord('z') = 122)

def changeCase(c):
    if (ord(c) >= 65 and ord(c) <= 90):
        newcVal = ord(c) + 32; # computes ASCII of corresponding lowercase alphabet
        newc = chr(newcVal);
        print(newc);
    elif (ord(c) >= 97 and ord(c) <= 122):
        newcVal = ord(c) - 32; # computes ASCII of corresponding uppercase alphabet
        newc = chr(newcVal);
        print(newc);
    else:
        print(c, "is not an alphabet");
```

OUTPUT:
Practice Problems:

1) Write a function CheckThree(x,y,z) that takes three values x, y, and z as command-line parameters and returns TRUE if the values are strictly ascending or descending (x< y< z or x> y> z), and FALSE otherwise.

Sample Output:

```python
>>> CheckThree(3.3, 4.1, 5)
TRUE
>>> CheckThree(2, 3, 1.2)
FALSE
>>> CheckThree(9, 6, 2)
TRUE
```

2) Write a function max(a,b,c) which takes 3 numbers as input and returns the maximum of the three sides of a triangle. Do not use the max function of the math package.

Sample Output:

```python
>>> max(2, 5, 7)
7
>>> max(4.5, 8.3, 2.1)
8.3
```

3) Write a function triangleType(s1,s2,s3) which takes in three sides of a triangle as input and prints a string defined as follows:

- “Regular” if all three sides are equal,
- “Symmetric” if any two sides are equal,
- “Irregular” if no two sides are equal.

Sample Output:

```python
>>> triangleType(2,3,5)
Irregular
>>> triangleType(8,7,8)
Symmetric
>>> triangleType(7,7,7)
Regular
```