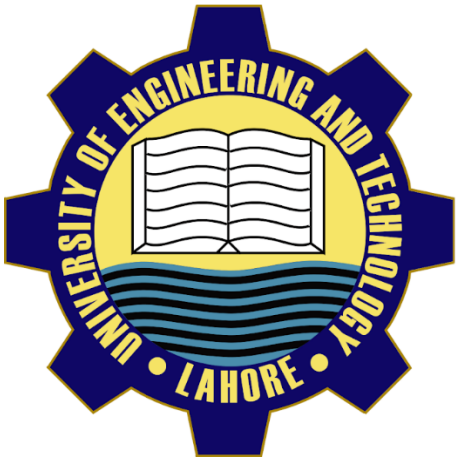
MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9



Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

LAB 13: LOOP/PROGRAM CONTROL & OTHER MISCILINEOUS STATEMENTS

Break:

The **break** statement terminates the loop execution and transfers the execution to the statement immediately outside the loop block. See the following code to understand its working:

```
for i in range(10):
    print(i)
    if(i==5):
        break
print("Outside For Loop.")
```

Here is the output:

```
0
1
2
3
4
5
Outside For Loop.
```

You can see that on iteration where **i=5**, the break statement executed and terminated the loop.

Tasks:

[1] We did the task of finding whether the input number is prime or not. To have efficient solution, we said that instead of iterating from **2** to **n-1** (where **n** is the input number), we should iterate from **2** to square root of **n**. But this can be made more efficient. Consider when **n=48** and loop starts from **2**. As **2** is factor of **48** hence **48** is not a prime number and we should not go till square-root of **48**. Therefore, use **break** statement in your program to find input number is prime or not making it more efficient. Don't create a user defined function.

The **break** statement breaks the execution of its parent loop only. See the following code:

```
for h in range(4):
    for i in range(10):
        print(i)
        if(i==2):
            break
    print('End of Inner loop')
```

The output clearly shows that **break** statement is part of inner loop, so it terminated the inner loop every time, and the outer loop completed its iterations:

```
0
1
2
End of Inner loop
0
1
2
End of Inner loop
0
1
2
End of Inner loop
0
1
2
End of Inner loop
```

Guess the output of following code and verify that by executing it:

```
for i in range(4):
    for j in range(10):
        print(j)
    if(i==1):
        break
```

Now once again guess the output and verify that for following code:

```
for i in range(4):
    for j in range(10):
        print(j)
        if(j==3):
            break
    if(i==1):
        break
```

Using break with infinite while loop:

One very specific use of **break** statement is with infinite **while** loop. Firstly, as we know that infinite **while** loop is a loop whose condition never gets false and the loop keeps iterating. One possibility of such **while** loop is as under:

```
while(True):
    #Block of loop
```

The condition of above loop is **True** always and hence will keep iterating. But such loops can be terminated if there is **break** statement inside its block. Let's see a specific example of such scenario.

Example:

Write a program that will keep asking user to enter a cartesian point (x,y) until the entered point has unity magnitude. This can be done in a couple of ways. One possibility using an infinite loop and **break** statement is shown here:

```
while(True):
    x,y=eval(input("Enter point x,y : "))
    if(x**2+y**2==1):
        print("Yes it has unit magnitude!")
        break
    print("It does not has unit magnitude!")
```

Continue statement:

A **continue** statement in the block of a loop causes to skip the remaining part of the loop block for that iteration and jumps to next iteration. You can say that it skips that single iteration only. See the following code to understand its working:

```
for i in range(10):
    if(i==5):
        continue
    print(i)
print("Outside For Loop.")
```

Here is the output where you can see 5 is not printed because of continue statement:

```
0
1
2
3
4
6
7
8
9
Outside For Loop.
```

One possible use of continue statement is shown in following code:

```
n=1
sum=0
while(n!=0):
    n=eval(input("Enter positive number (0 to exit): "))
    if(n<0):
        print("Only positive numbers allowed.")
        continue
    sum+=n
```

```
print("Sum of numbers entered: "+str(sum))
```

Here user is not allowed to enter negative numbers and positive numbers entered buy user are summed together. If user enters a negative number, the statement **sum+=n** will not be executed because of **continue** statement.

Pass statement:

The **pass** statement is a null operation; nothing happens when it executes. The **pass** statement is useful in places where your code will eventually go but has not been written yet. This can be used in block of **if-else**, loop or user-defined function. For example, following **if-else** statement indicates that the **else** block of the code will be written afterwards.

```
x=100
if (x<10):
    print("Less than 10")
else:
    pass
```

Exit function:

The built-in function **exit()** when executed, ends the whole program. This function can be used in block of **if-else**, loop, user-defined function or the main program. See the following code:

```
for i in range(4):
    for j in range(10):
        print(j)
        if(j==2):
            exit()
    print('End of Inner loop')
```

It has the output as shown:

```
0
1
2
```

The else in loop:

Interestingly, there can be **else** linked with loop as shown here:

```
for i in range(5):
    print(i)
else:
    print("Outside For Loop.")
```

If you run the above code you will see the output of last **print** statement, meaning that **else** block is executed. But it would have been executed if you had written that without the **else** statement. Let's see this program:

```
for i in range(5):
    if(i==2):
        break
    print(i)
else:
    print("Outside For Loop.")
```

Now you will not see the output of the **else** block. So is that something related to **break** statement?

Let's see this code now, where we have the **break** statement but that will not get executed because the condition is **False**.

```
for i in range(5):
    if(i==10):
        break
    print(i)
else:
    print("Outside For Loop.")
```

This time you will see the **else** block getting executed. So how is the **else** of a loop related to **break** statement?

If a loop doesn't get terminated by the **break** statement, the **else** block will get executed.

This can be a typical use of **else** with loop:

```
from math import sqrt
num=input(eval('Enter an integer: '))
for i in range(2,int(sqrt(num))+1):
    if(num%i==0):
        print(f'{num} is a NOT a Prime Number!')
else:
    print(f'{num} is a Prime Number!')
```

Tasks:

[2] Write a program that reads the numbers from the user and when the user enters 0, gives the average of the numbers entered (except the last entered 0). This is the same task as in one of the previous lab sessions. Here you must use:

```
while(True):
```

Inside the loop write an `if` statement that adds the value into `sum` variable if it is not `0` and if it is `0` then use appropriate instruction from today's lab session to terminate the loop.

Clear Screen and Sleep functions:

If you want to clear the previous contents of the screen, there is a function named as `system()` available in `os` module (os stands for operating system). Following code describes one example:

```
import os
print("Mechatronics")
os.system('cls')
print("Session 2019")
```

When you will run the above code, you will see `"Session 2019"` on the screen only. `"Mechatronics"` was displayed but immediately after that the `system('cls')` cleared the content of the screen and next print statement displayed `Session 2019`.

In above code we were not able to see `"Mechatronics"` on the screen because of the next statement which cleared the content immediately. However, if we introduce a delay of say 5 seconds after first print statement, we will be able to see that. For this case and many other situations, we need to add some delay in between execution of statements. This can be done using `sleep()` function available in `time` module. An example is shown here:

```
import os,time
print("Mechatronics")
time.sleep(5)
os.system('cls')
print("Session 2019")
```

Tasks:

[3] Write a program that will display digital clock (minutes and seconds) on the output screen as **hh:mm** starting from **00:00**.
Both minutes and seconds should be in 2-digit format. You must clear previous screen each second to display updated clock.

Keyboard hit kbhit() and Get Character:

Keyboard hit and get-character are two methods for user input available in `msvcrt` module, having different functionality than default `input()` method. Firstly, the keyboard hit is explained here:

The default `input()` function when executed waits for the user input and during that time no other instruction is executed. Now consider the above task of clock display, what if we want to pause the clock when some key say `'P'` is pressed by the user? If we will use default `input()` function, it will wait for the input and the clock will never run. For these cases we use keyboard hit `kbhit()` method which

doesn't wait for the input but if any key is pressed this function returns `True`. So, the normal code keeps running and some key press is detected by this function. One example program is explained here:

A simple program to keep displaying a message `"We Welcome You!"` until user presses some key, can be written as:

```
import msvcrt
while True:
    print("We Welcome You!")
    if(msvcrt.kbhit()):
        break
print("Thanks!")
```

The other function i.e. Get Character has a few variants. We will use `getwch()` which stands for get wide character. This function takes a single input character from user. Default `input()` function can also be used to take single character but using `getwch()`, the user need not to press `enter` after the input. The key pressed is taken as input without need to press `enter`.

In above example of `kbhit()` if we want to track what key was pressed by the user we can use `getwch()` with it. For example, if we want to exit program if `'e'` was pressed and display `'Thanks'` for any other character, it can be done as:

```
import msvcrt
while True:
    print("We Welcome You!")
    if(msvcrt.kbhit()):
        a=msvcrt.getwch()
        break
if(a=='e' or a=='E'):
    exit()
print("Thanks!")
```

Tasks:

[4] Repeat task 3 such that the clock must keep running as earlier and if `'P'` is pressed the clock must pause there. If any other key is pressed, the clock must restart from 00:00.

Use of keyword “as”:

The last thing we will discus is about the keyword `as`. Suppose we want to use the `sqrt` function of the `math` module. We can do it like:

```
from math import sqrt
print(sqrt(10))
```

While importing something, we can customize the name of that thing. For example, for some reason if you don't feel good with the name `sqrt`, you can change it to anything, for example `sq`, while importing it using the keyword `as`. It is given here:

```
from math import sqrt as sq
print(sq(10))
```

Usually we do it when the name of the importing entity (it can be function inside module, directly a module, some other class etc.) is large.