***Lab #6***

***FOR LOOP and LIST***

**Introduction**

Like other programming languages, for loops in Python are a little different in the sense that they work more like an iterator and less like a for keyword. In Python, there is not C like syntax

**for(i=0; i<n; i++) but you use for in n.**

They can be used to iterate over a sequence of a list, string, tuple, set.

Given a list of elements, for loop can be used to iterate over each item in that list and execute it.

The for loops in Python are zero-indexed.

**EXAMPLE # 1:**

for i in range(10):

print (i+1)

**OUTPUT:**

1

2

3

4

5

6

7

8

9

10

**EXAMPLE # 2:**

for i in "SCIENCE":

if i == 'E':

print (i)

**OUTPUT:**

E

E

**Object:**

Write a program to compute distance between two points taking input from the user

(Pythagorean Theorem).

## Description:

The Pythagorean Theorem is the basis for computing distance between two points. Let (x1,y1) and (x2,y2) be the co-ordinates of points on xy-plane. From Pythagorean theorem, the distance between two points is calculated using the formula:



To find the distance, we need to use the method **sqrt()**. This method is not accessible directly, so we need to import **math** module and then we need to call this method using math static object.

To find the power of a number, we need to use \*\* operator.

## Algorithm:

**Input:** x1, y1, x2 and y2

**Output:** Distance between two points.

Step1: Start

Step2: Import math module

Step3: Read the values of x1, y1, x2 and y2

Step4: Calculate the distance using the formula **math.sqrt( (x2 - x1)\*\*2 + (y2 - y1)\*\*2 )** and store the result in distance

Step5: Print distance Step6: Stop

**Object:**

Using a for loop, write a program that prints out the decimal equivalents of 1/2, 1/3, 1/4, . 1/10

## Description:

A loop statement allows us to execute a statement or group of statements multiple times. We can use for loop to calculate the decimal equivalents of given set of numbers.

## For loop statement:

It has the ability to iterate over the items of any sequence, such as a list or a string. Iterating over a sequence is called Traversal.

To find the decimal equivalents, we need to use the method **pow()**. This method is not accessible directly, so we need to import **math** module and then we need to call this method using math static object.

## Algorithm:

**Output**: Decimal equivalents of 1/2, 1/3..,1/10

Step1: Start

Step2: Import math module

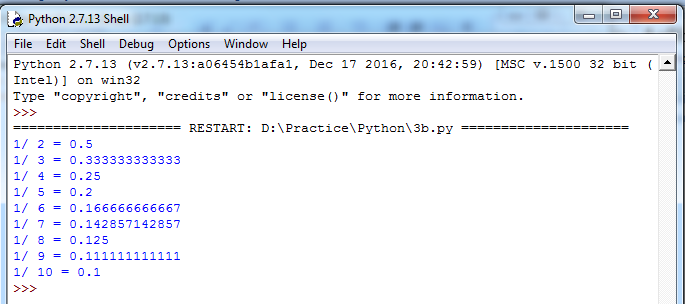
Step3: Initialize i = 2

Step4: Repeat Steps 5,6 until i = 11

Step5: Print math.pow(i,-1)

Step6: Increment i by 1

Step7: Stop

**Output:**

**Object:**

Write a program using a for loop that loops over a sequence. What is sequence?

## Description:

A Sequence is the generic form for an ordered set. There are several types of sequences in Python. The following 3 are most important.

**Lists:** There are the most versatile sequence type. The elements of a list can be any object and lists are mutable.

**Tuples:** These are like lists, But Tuples are immutable.

**Strings:** These are a special type of sequence that can store only characters and having special notations.

## Algorithm:

**Output**: Elements of sequence(List).

Step1: Start

Step2: Initialize the list named a as a = [“AHMED”,19,85.75]

Step3: Repeat Step4 until the last element in the list is reached

Step4: Print ith element

Step5: Stop

## Output:

## AHMED

## 19

## 85.75

## OBJECT:

## Write a program to find the sum of all primes below two million.

## Description:

A Prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. A natural number greater than 1 and that is not a prime number is called a composite number.

Here, we are using a **for** loop and a **break** statement.

## break:

break statement terminates the loop and resumes execution at the next statement, just like the traditional break statement in C. The break statement can be used in both while and for loops.

If we are using break statement inside a nested loop, it stops the execution of the innermost loop and start executing the next line of code after the block.

## Algorithm:

**Output**: Sum of primes below 2 million

Step1: Start

Step2: Initialize sum = 0 and i = 2

Step3: Repeat Steps 4 to 13 while i < 2000000

Step4: Initialize c = 0

Step5: Check whether i is greater than 2 and i is divisible by 2 or not. If yes, goto Step6 else, goto Step7

Step6: Set c = 1 and goto Step7: Initialize j = 3

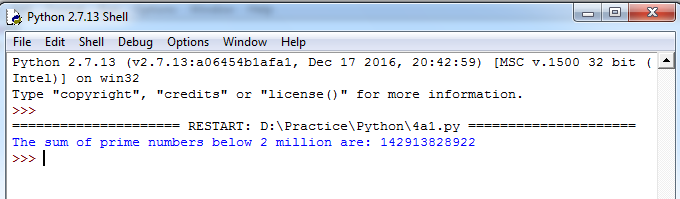
Step8: Repeat Steps 9 to 11 while j <= int(i\*\*0.5)

Step9: Check whether i is divisible by j or not. If yes, goto Step10. Otherwise, goto Step11 Step10: Set c = 1 and goto Step12

Step11: Increment j by 2

Step12: Check whether c is 0 or not. If yes, goto Step13 Step13: Add i to the sum and store the result in sum Step14: Display sum

Step15: Stop



## EXERCISE QUESTION:

## Exercise 1: Write a program which repeatedly reads numbers until the

## user enters “done”. Once “done” is entered, print out the total, count,

## and average of the numbers. If the user enters anything other than a

## number, detect their mistake and print an error message and skip to the next number.

## OUTPUT:

## Enter a number: 4

## Enter a number: 5

## Enter a number: bad data

## Invalid input

## Enter a number: 7

## Enter a number: done

## 16 3 5.333333333333333

## Exercise 2: Write another program that prompts for a list of numbers

## as above and at the end prints out both the maximum and minimum of

## the numbers instead of the average.