Introduction & Problem Solving

Dr. Indrajit Pan

Associate Professor, Dept. of IT

RCC Institute of Information Technology, Kolkata

Acknowledgment:

Introduction to Computing and Problem Solving with Python, *Jeeva Jose and P. Sojan Lal*, Khanna Publishing, 2017

Preface

- **Inventor:** Guido van Rossum, Netherlands
- **Year:** 1985 – 1990
- Python has been derived many other programming languages like ***C, C++, Algol-68, SmallTalk, Unix shell*** etc.
- Name "Python" has no specific significance. Rossum wanted to keep it simple and small and hence named after a show of BBC Comedy Series – *Monty Python's Flying Circus*

Features of Python

- General purpose and simple constructs
- Interactive and interpreted
- High level Programming Language
- Object oriented
- Extendable
- GUI Programming and Databases
- Broad standard library

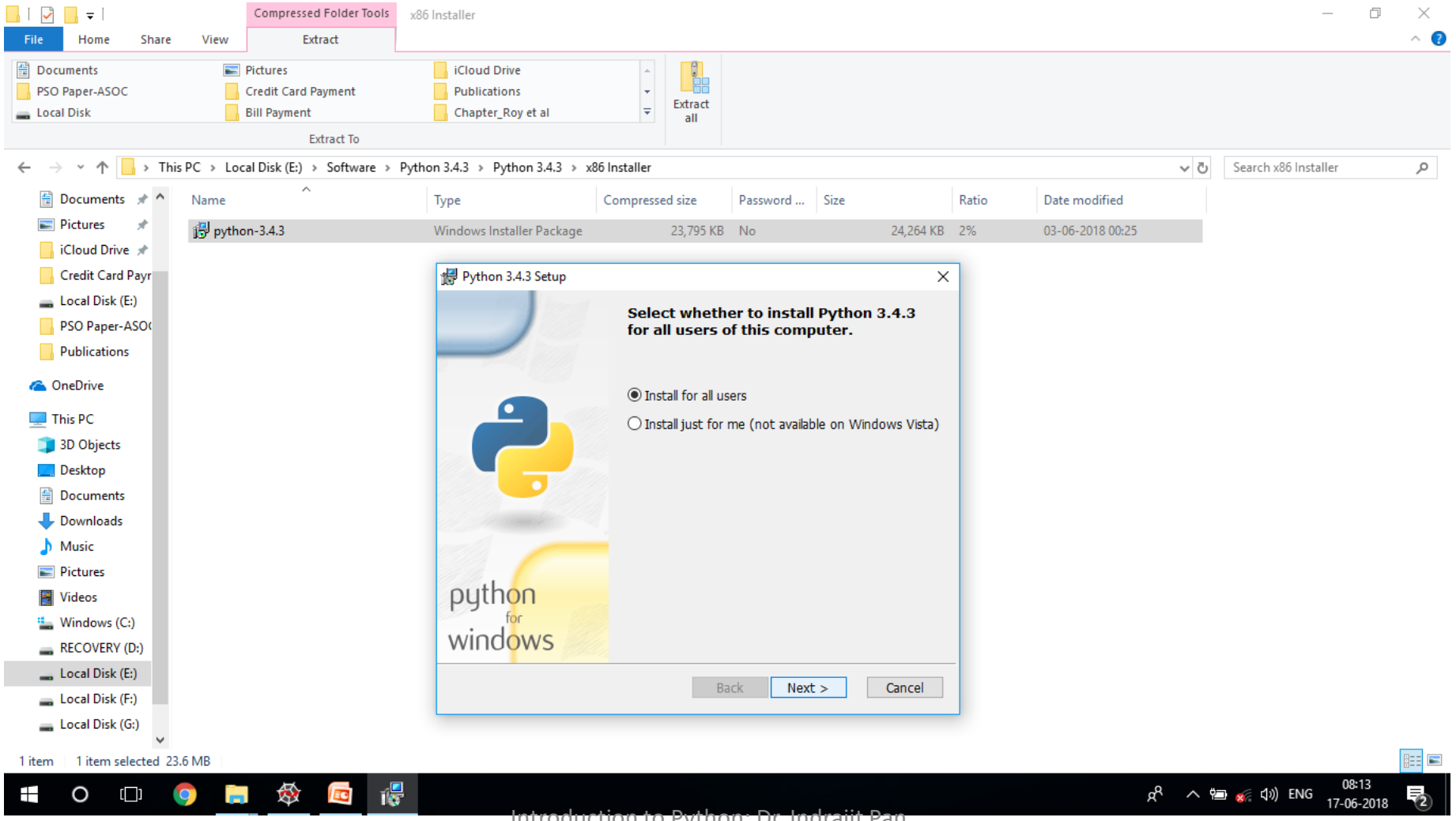
Python Development Tools

We can use

- Python-3.4.3 installer
  - [**WebSource:** ***https://www.python.org/downloads/release/python-343/***]
  - Latest available version: `Python 3.6.5`
- Anaconda IDE
  - [**WebSource:** ***https://anaconda.org/anaconda/python/files***]

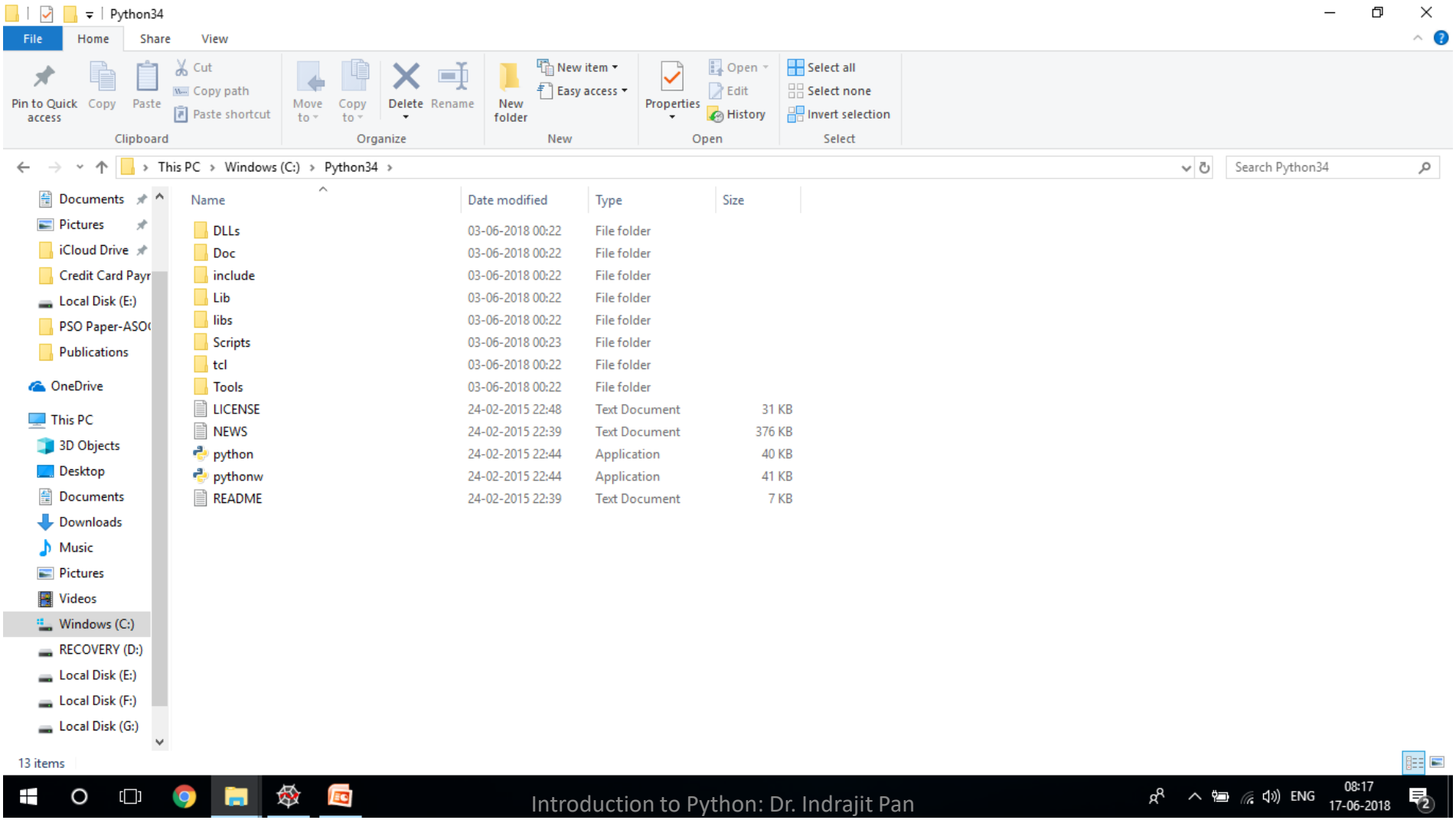
Installation of Python - 1

(python-3.4.3.amd installer)



Installation of Python - 2

(python-3.4.3.amd installer)



Installation of Python - 3
(python-3.4.3.amd installer)

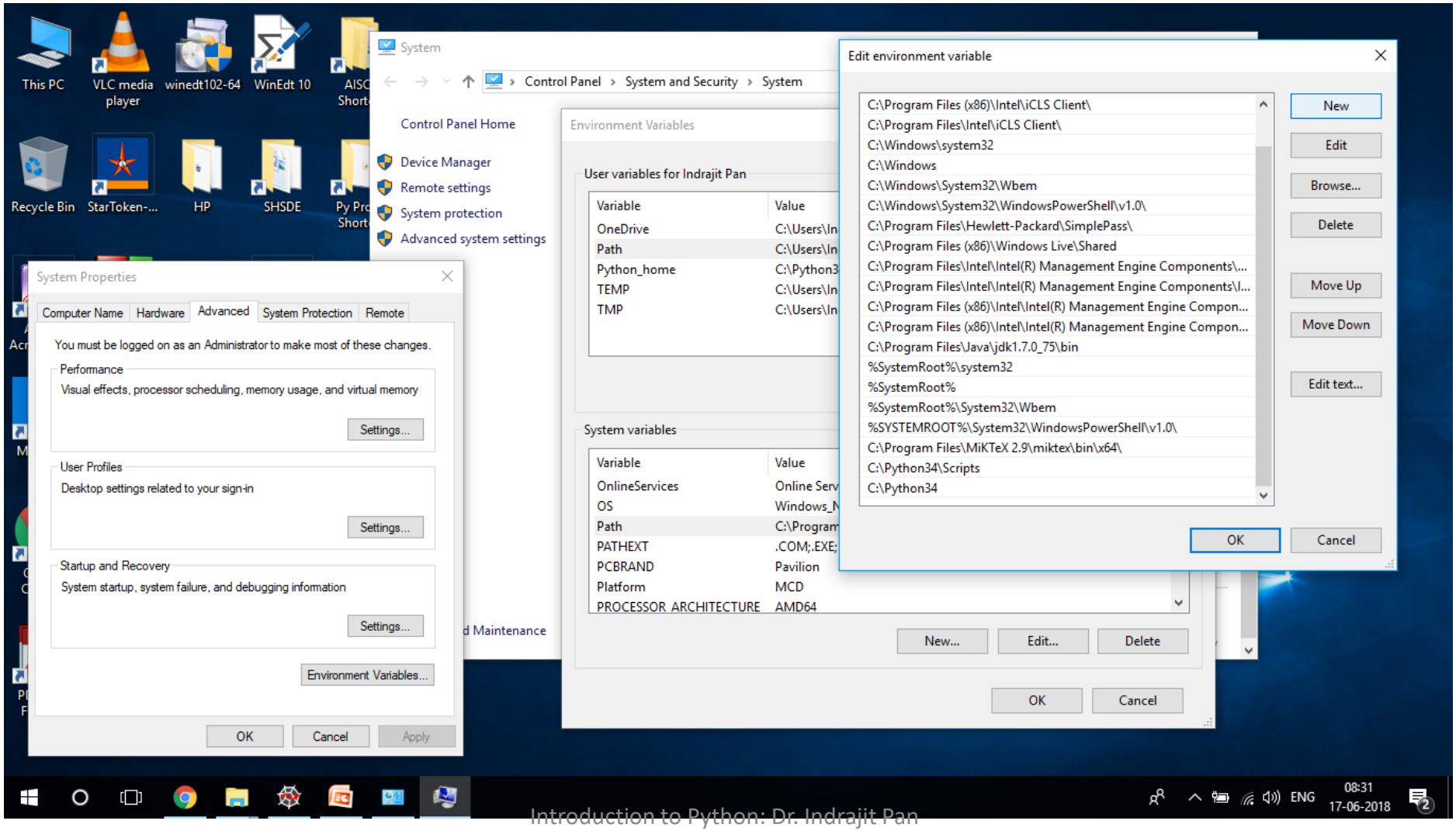
- After installation we need to set windows environment variable for seamless access of **python** (*python*) and **package management system** (*pip*).
- In order to set windows environment, parent **Python installation directory path** (e.g.: C:\Python34) need to be appended in path variable under environment settings.

Installation of Python - 4
(python-3.4.3.amd installer)

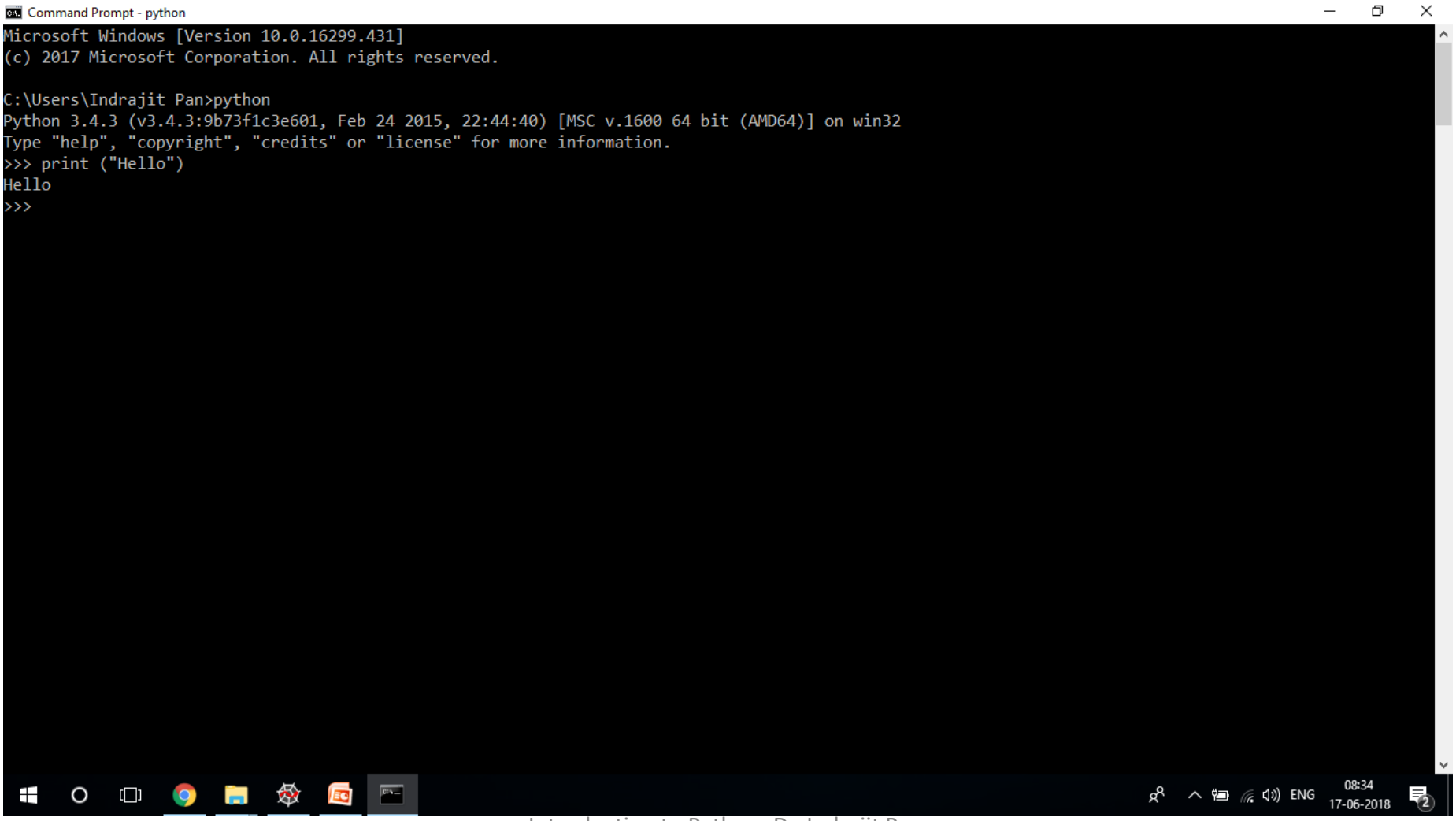
- **Python script directory path** (e.g.: C:\Python34\Scripts) also need to be appended under environment settings. Successful inclusion can be checked by issuing the command "pip"

Installation of Python - 5

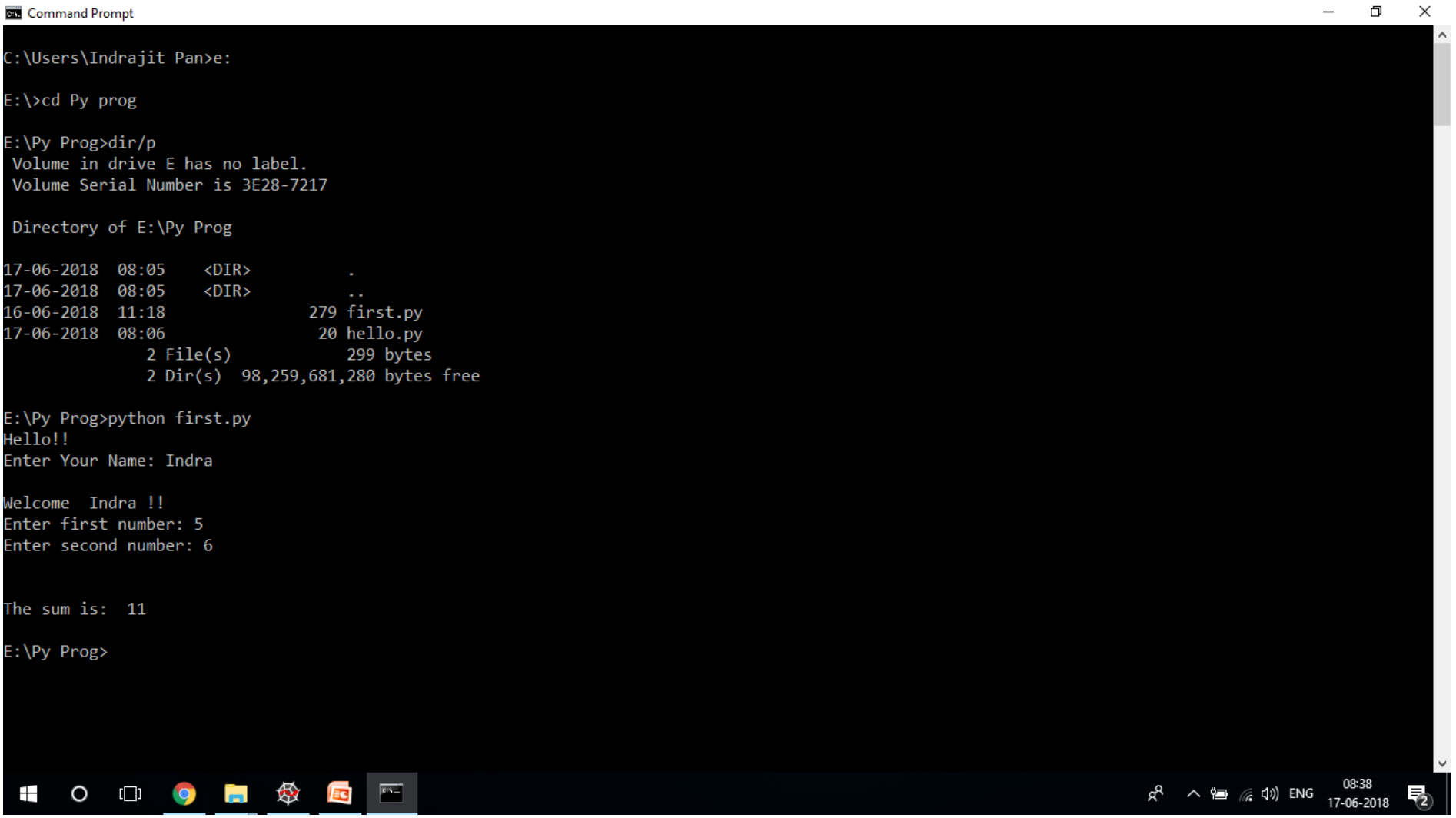
(python-3.4.3.amd installer)



Checking Installation



Working with Python



Basics of Python

First Program
(hello.py)

```
print("Hello")
```

Output: Hello

Execution of Program

- Whenever the Python script compiles, it automatically generates a compiled code called as byte code. The byte-code is not actually interpreted to machine code, unless there is some exotic implementation.
- The byte-code is loaded into the Python run-time and interpreted by a virtual machine, which is a piece of code that reads each instruction in the byte-code and executes whatever operation is indicated.
- Byte Code is created under ***__pycache__***. Sometimes it is also stored in ***sys.path***.
- Next time, when the program is run, python interpreter use this file to skip the compilation step, if there is no change in source file

.py and .pyc

- **.py** is extension of Python program
- .**pyc** contain the compiled byte code of **Python** source files. The **Python** interpreter loads .**pyc** files before .**py** files, so if they're present, it can save some time by not having to re-compile the **Python** source code
- Python automatically compiles your script to compiled code, so called byte code, before running it.

Is Python a compiled or interpreted language?

- **Python** will fall under byte code **interpreted**.
- **.py** source code is first **compiled** to byte code as **.pyc**.
- This byte code can be **interpreted** (official CPython), or JIT**compiled** (PyPy).
- **Python** source code (.py) can be **compiled** to different byte code also like IronPython (.Net) or Jython (JVM).

Reserved Words

```
import keyword
print(keyword.kwlist)

>>['False',   'None',   'True',   'and',   'as',
  'assert',   'break',   'class',   'continue',
  'def',   'del',   'elif',   'else',   'except',
  'finally',  'for',  'from',  'global',  'if',
  'import',  'in',  'is',  'lambda',  'nonlocal',
  'not',   'or',   'pass',   'raise',   'return',
  'try', 'while', 'with', 'yield']
```

- Number of keywords will vary in different versions of python

Variables

- Apart from the **reserved words** any valid identifier can be used as variable
- Rules for identifier construction are same as C, C++ or Java
- Python variables don't need explicit declaration to reserve memory space
- Automatic declaration happens at the time of assigning value to a variable
- Equal sign (=) is used to assign values to variables

Data Types

- Python supports four different numerical types
  - **int** (Signed integers)
  - **long** (long integers, they can also be represented in octal and hexadecimal)
  - **float** (Floating point real values)
  - **complex** (Complex numbers)

Example

(numerical.py)

```
a, b, c, d = 5,1.6,231456987,2+9j
print("a = ",a)
print("b = ",b)
print("c = ",c)
print("d = ",d)
```

Output:

a = 5

b = 1.6

c = 231456987

d = (2+9j)

Multi Line Statements

(mline.py)

```
a, b, c, d = 50,100,150,200
sum1 = a + b
sum2 = a + \
       b + \
       c + \
       d
print("sum1 = ",sum1)
print("sum2 = ",sum2)
```

Output:

sum1 = 150

sum2 = 500

Output Function [*print()*]

(ofun.py)

```
print(1,2,3,4)
print(1,2,3,4,sep='+')
print(1,2,3,4,sep='+',end='%')
```

Output:

1 2 3 4

1+2+3+4

1+2+3+4%

---------- **Syntax of Print function**

print *objects, sep=' ', end = '\n', file = sys.stdout, flush = False

Input Functions

- Python provides two built-in functions to read a line of text from default standard input keyboard
  - `raw_input` (not supported by python3.4.3)
  - `input`

    `input() returns entry as a string`

Input Function – [input()] (ifun.py)

```
n = input("Enter a new number: ")
print("The number is ",n)
```

Output:

Enter a new number: 5

The number is 5

Import Function
(impt.py)

- **import** keyword is used to include a module within a scope of another module
- A module is a file containing Python definitions and statements
- Python modules have file name & ends with **.py**

```
import math
print("The value of pi = ",math.pi)
The value of pi = 3.141592653589793
```

Operators

- Operators are the constructs which can manipulate the value of operands
  - Arithmetic Operators
  - Comparison (Relational) Operators
  - Assignment Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators
  - Identity Operators

Arithmetic Operators in Python

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponent
//	Floor division

Comparison Operators in Python

Operator	Operation
==	True if values of two operators are equal
!=	True if values of two operators are not equal
>	True if value of left operand is greater than right
<	True if value of left operand is less than right
>=	True if value of left operand is greater than or equal to right
<=	True if value of left operand is less than or equal to right

Assignment Operators in Python

Operator	Operation
=	Assign values from right side operand to left side operand
+=	Adds right operand to the left operand and assigns to left
-=	Subtracts right operand from the left operand and assigns to left
*=	Multiplies right operand with the left operand and assigns to left
/=	Divides left operand with the right operand and assigns to left
%=	Modulus using two operands assigns to left
**=	Performs exponential calculation and assigns to left
//=	Performs floor division calculation and assigns to left

Bitwise Operators in Python

Operator	Operation
&	Binary AND
|	Binary OR
^	Binary XOR
~	Binary Ones Complement
<<	Binary left shift
>>	Binary right shift

Logical Operators in Python

Operator	Operation
and	Logical AND
or	Logical OR
not	Logical NOT

```
a,b,c,d = 10,5,2,1

print((a>b) and (c>d))
print((a>b) or (d>c))
print(not(a>b))
```

Output:
True
True
False

Membership Operators in Python

Operator	Description
in	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise
not in	Evaluates to true if it does not find a variable in the specified sequence and false otherwise

```
s='abcde'
print('a' in s)
print('f' in s)
print('g' not in s)
```

Output:

True
False
True

Identity Operators in Python

Operator	Description
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise

```
a, b, c = 10, 10, 5
print(a is b)
print(a is c)
print(a is not b)
```

Output:
True
False
False

Data types in Python

Data Types

- Python supports six different data types
  - **Numbers**
  - **String**
  - **List**
  - **Tuple**
  - **Set**
  - **Dictionary**

Number Data Types

- Python supports four different numerical types
  - **int** (Signed integers)
  - **long** (long integers, they can also be represented in octal and hexadecimal)
  - **float** (Floating point real values)
  - **complex** (Complex numbers)

Example

(numerical.py)

```
a, b, c, d = 5,1.6,231456987,2+9j
print("a = ",a)
print("b = ",b)
print("c = ",c)
print("d = ",d)
```

Output:

a = 5

b = 1.6

c = 231456987

d = (2+9j)

Functions for Number Data Types

- Number data types works with different mathematical functions

Function	Description
abs(x)	Absolute of X
math.sqrt(x)	Square root of x
math.ceil(x)	Ceiling of x
math.floor(x)	Floor of X
pow(x,y)	x raised to y
math.exp(x)	e^x
math.log(x)	Natural logarithm of x for x>0
math.log10(x)	Logarithm base 10 for x>0
max(x1,x2,...,xn)	Largest of its arguments
min(x1,x2,...,xn)	Smallest of its arguments
round(x,[n])	For decimal numbers, x will be rounded to n digits
math.modf(x)	For decimal numbers returns integer and decimal as tuple

Functions for Number Data Types

- Number data types works with different trigonometric functions

Function	Functions
math.sin(x)	math.atan(x)
math.cos(x)	math.atan2(x, y)
math.tan(x)	math.hypot(x, y)
math.asin(x)	math.degrees(x)
math.acos(x)	math.radians(x)

String Data Types

- String is identified as contiguous set of characters represented in the quotation marks

e.g.

```
str = 'Welcome to python prog.'
print(str)
```

List Data Types

- List is an ordered set of items
- Mostly used in python
- All the elements need not to be of same type
- Items separated by commas are enclosed within brackets []
- Values stored in list can be accessed by slice operator ([] and [:]) with indices starting at 0 with indices starting at 0 and ending with -1
- + sign is the concatenation operator
- Asterisk (*) is the repetition operator
- Check program – *listpy.py*

Operation on List Data Types

Function	Operation
max(list)	Returns item from list with maximum value
min(list)	Returns item from list with minimum value
list.append(obj)	Appends an object obj passed to the existing list
list.count(obj)	Returns how many times object obj appears in a list
list.remove(obj)	Remove object obj from the list
list.index(obj)	Returns index of the object obj if found otherwise exception indicating the value doesnot exist
list.reverse()	Reverse objects in a list
list.insert(index,obj)	Returns a list with obj inserted at a given index
list.sort()	Sorts the items in ascending order and returns
list.sort([func])	Sort list according to given function

Input in List Data Type

```
str=input("Enter  a  list  (space
  separated): ")

lis = list(map(int,str.split()))

print(lis)
```

Tuple Data Types

- Tuple is another sequence data type similar to list
- A tuple consist of number of values separated by comma
- List is enclosed by square brackets ([]) and their elements and size can be changed
- Tuples are enclosed in () brackets and cannot be updated
- Tuple is a read only list

Set Data Types

- Set is an **unordered** collection of **unique** items
- Set is defined by value separated by comma and braces {}
- It can have any number of items and they may be different in types
- An element in set can't be accessed or changed using indexing or slicing
- Set operations like union, intersection, difference of two sets can be performed

Dictionary Data Types

- Dictionary is an unordered collection of **key-value** pair
- It is generally used when we have huge amount of data
- We must know the **key** to retrieve the values
- Dictionary is defined within {}
- Keys are usually numbers or strings
- Value can be any arbitrary Python object
- Check program ***dict.py***
- More than one entry per key is not allowed
- Keys are immutable, so list can't be used as key

Decision Making Blocks
&
Loops

Decision Making (if..elif..else)

```
num=int(input("Enter a number: "))
if num > 0:
  print("Positive number")
elif num == 0:
  print("Zero")
else:
  print("Negative number")
```

Decision Making (Nested if)

```
num=int(input("Enter a number: "))
if num >= 0:
  if num == 0:
     print("Zero")
  else:
     print("Positive number")
else:
  print("Negative number")
```

Loops
(for loop)

```
list = [2,3,4,5]
sum = 0
for item in list:
  sum = sum+item
print("The sum is: ",sum)
```

Loops
(for loop)

```
color=['red','blue','green']
for item in color:
  print("Color is: ",item)
```

Loops
(for loop)

```
for char in 'program':
  print("Char is: ",char)
```

***for* Loops**
(Assignment)

Write a program to populate a list of 6 numbers and then **linear search** for a specific number (user defined) within that list

for Loops
(Assignment Linear Search – Type 1)

```
str=input("Enter 6 elements of the list
  separated by a space: ")
user_list=list(map(int,str.split()))
print("The list is: ",user_list)
num=int(input("Enter the number to be searched:
  "))
flag=0
for item in user_list:
  if item == num:
      print("Element ",num," found in the list")
      flag=1
if flag == 0:
  print("Element not found")
```

for Loops
(Assignment Linear Search – Type 2)

```
str=input("Enter 6 elements of the list separated by
   a space: ")
user_list=list(map(int,str.split()))

print("The list is: ",user_list)

num=int(input("Enter the number to be searched: "))
for item in user_list:
   if item == num:
       print("Element ",num," found in the list")
       print("Location: ",user_list.index(item)+1)
       break
else:
   print("Element not found")
```

Not using loop
(Assignment Linear Search – Type 3)

```
str=input("Enter 6 elements of the list
  separated by a space: ")
user_list=list(map(int,str.split()))
print("The list is: ",user_list)
num=int(input("Enter the number to be
  searched: "))
if num in user_list:
  print("Element found at location
  ",user_list.index(num)+1)
else:
  print("Element not found")
```

for Loops
(Assignment Linear Search – Type 4)

```
str=input("Enter 6 elements of the list
  separated by a space: ")
user_list=list(map(int,str.split()))
print("The list is: ",user_list)
num=int(input("Enter the number to be
  searched: "))
for item in range(len(user_list)):
  if user_list[item] == num:
     print("Element  ",num,"  found  at
  position ",item+1)
     break
else:
  print("Element not found")
```

for Loops
(Entering elements in list)

```
lis=[]
n=int(input("Enter List Size: "))

for item in range(n):
  num=int(input("Enter element: "))
  lis.append(num)
print("\n The list is: ",lis)
```

for Loops (Entering elements in list)

```
n=int(input("Enter List Size: "))
lis=[0] * n
print("List after initialization:",lis)
print("Data entry in the list initiaing..")
for item in range(n):
  print("Enter element",end=':')
  lis[item]=int(input())
print("\n The list is: ",lis)
```

for Loops (*range(start, stop, step_size)*)

```
print("Running on range(0,10,1)****")
for item in range(0,10,1):
  print("Value of index: ",item)

print("Running on range(0,10,2)****")
for item in range(0,10,2):
  print("Value of index: ",item)

print("Running on range(10,0,-1)****")
for item in range(10,0,-1):
  print("Value of index: ",item)

print("Running on range(10,0,-2)****")
for item in range(10,0,-2):
  print("Value of index: ",item)
```

for Loops (*range(start, stop)*)

```
for letter in range(5,10):
  print(letter)
```

Output:

```
5
6
7
8
9
```

for Loops
(Nested for loop – Ex. 1)

```
for item in range(5):
  for index in range (item):
     print("*",end=' ')
  print(" ")
```

for Loops
(Nested for loop – Ex. 2)

```
for item in range(5):
  for index in range (item):
     print("*",end=' ')
  print(" ")
for item in range(5,0,-1):
  for index in range (item):
     print("*",end=' ')
  print(" ")
```

Two Dimensional List

- In Python any table can be represented as a list of lists (a list, where each element is in turn a list).

```
lis=[[1,2,3],[4,5,6],[7,8,9]]
for row in lis:
  for elem in row:
     print(elem, end=' ')
  print()
```

Two Dimensional List

```
lis=[[1,2,3],[4,5,6],[7,8,9]]
print("Printing list on indices")

for row in range(len(lis)):
  for col in range(len(lis[row])):
     print(lis[row][col], end=' ')
  print()
```

Two Dimensional List (Creation) – *listtwo2.py*

```
rw=int(input("Enter number of rows: "))
cl=int(input("Enter number of columns: "))
lis = [[0] * cl for i in range(rw)]
for row in range(len(lis)):
  for col in range(len(lis[row])):
     print("Element
  [",row,"][",col,"]",end=':')
     lis[row][col]=int(input())
print("\n\nThe list is::")
for row in range(len(lis)):
  for col in range(len(lis[row])):
     print(lis[row][col], end=' ')
  print()
```

while Loops

```
count = 5
while count >= 1:
  print(count)
  count = count - 1
```

while Loops (with *else* statement)

```
count = 5
while count >= 1:
  print("count ",count)
  count = count - 1
else:
  print("Exit with count ",count)
```

while Loops

(Assignment binary search – **bisearch.py**)

```
import math
lis=[]
n=int(input("Enter List Size: "))

for item in range(n):
    print("Enter element ",item+1,end=':')
    num=int(input())
    lis.append(num)
print("Entered list is: ",lis)
lis.sort()
print("Sorted list:",lis)
key=int(input("Enter the element to be searched: "))
first = 0
last = n - 1
while first <= last:
    mid = math.ceil((first+last)/2)
    if lis[mid] == key:
           print("Element found at ",mid+1)
           break
    else:
           if lis[mid] > key:
                      last = mid - 1
           else:
                      first = mid + 1
if first > last:
    print("Element not found")
```

continue Statement

```
for letter in 'abcd':
  if letter == 'c':
     continue;
  else:
     print(letter)
```

***while* Loops**
(Assignment)

Write a program to populate a list of 6 numbers and then perform **binary search** for a specific number (user defined) within that list

Assignment

1. Find **GCD** of two user specified numbers
2. Find the **factorial** of a user specified number
3. Generate **Fibonacci series** of ***n*** terms (starting with 0 and 1)
4. Find the **count of vowels** within a string entered by user
5. Write a program that generate following **pyramid:**

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Functions in Python

Functions in Python

Functions in Python are of two types,

- Built – in functions
  - Built-in functions are already defined within Python environment and used to perform various application specific tasks (e.g. input(), sqrt(), print() etc.)
- User defined functions
  - User defined functions are designed by the programmers to perform a particular task.

User-defined function

Basic structure of a user-defined function

def functionname (*parameters*):
    "*function_docstring*"
    function_body
    return *expression*

- Each definition begins with a keyword "def"
- Blue_italics parts are optional
- "function_docstring" is used for brief documentation of the function
- ***return*** with no argument is same as return none
- With no argument, ***return*** is optional

Example Program - 1

```
# Function body without parameter and return
def addition():
    a=int(input("Enter first number:"))
    b=int(input("Enter second number:"))
    sum = a + b
    print("The Sum = ",sum)
    return

# Program body
print("Going to call function")
addition()
print("Control returned to main program")
```

Example Program - 2

```
# Function body with parameters but without
   return
def div(a,b):
    div = a/b
    print("Division = ",int(div))
    rem = a%b
    print("Remainder = ",rem)
    return
# Program body
m=int(input("Enter first number:"))
n=int(input("Enter second number:"))
print("Going to call function")
sub(m,n)
print("Control returned to main program")
```

Example Program - 3

```
# Function body with parameters and return
# Function body
def sub(a,b):
    "Subtraction operation on two integer"
    if a >= b:
       sub = a - b
    else:
       sub = b - a
    return sub
# Program body
m=int(input("Enter first number:"))
n=int(input("Enter second number:"))
print("Going to call function")
x = sub(m,n)
print("Control returned to main program")
print("The sub = ",x)
```

Argument types in function

There are four types of arguments in function

- Required argument
- Keyword argument
- Default argument
- Variable length argument

Example Program – *Required Argument*

```
def name(str):
   print("Hello ",str)
   return

m=input("Hi, what's your name :")
name(m)
```

Example Program – *Keyword Argument*

```
def greet(Name,Add):
    print("Hello ",Name,"\n So you are
    from ",Add,"\n How can I help you")
    return

m=input("Hi, what's your name :")
n=input("Location :")
greet(Add=n,Name=m)
```

Example Program – *default Argument*

```
def greet(Name,Add="Kolkata"):
    print("Hello ",Name," nice to see
    that you are from ",Add)
    return

m=input("Hi, what's your name :")
n=input("Location :")
greet(m,n)
m=input("Hi, what's your name :")
greet(m)
```

Variable length Arguments

```
def greet(formal_args, *var_args):
    "function_docstring"
    function_statements
    return [expression]
```

- Variable arguments are in ***tuple*** representation. (ref.: **tuple** datatype in python)
- Use loops to pick up elements from tuple
- Only one reference to variable argument tuple can be passed
- Variable argument appears as rightmost parameter in the parameter list

Example Program – *variable length arguments*

```
def varg(a,*b):
   print("Formal argument: ",a)
   for item in b:
     print("Variable argument: ",item)
   return

varg(10,'Red',20, 25,'Blue')
```

Example Program – *function returning multiple values*

```
def fnmultireturn(a,b):
    sum = a + b
    if a >= b:
       sub = a - b
    else:
       sub = b - a
    mul = a * b
    return sum,sub,mul

m = int(input("Enter first number: "))
n = int(input("Enter second number: "))
x,y,z = fnmultireturn(m,n)
print("Sum = ",x)
print("Difference = ",y)
print("Multiplication = ",z)
```

Lambda Functions

- ***lambda*** is a keyword for defining anonymous function. (here ***def*** is not used) Lambda function has no unique name
- Lambda functions can take any number of arguments but return only one value in the form of expression
- It can't contain multiple expressions
- It can't have comments
- A lambda function can't be direct call to print because lambda requires expression
- Lambda functions have their own namespace and can't access variables other than those in their parameter list and those in the global namespace. New variable can't be declared within this and any global variable can't be used within lambda function.
- It is not equivalent to inline functions in C or C++

Example Program 1 – *lambda function*

```
square = lambda x: x*x;

n = int(input("Enter a number:"))
y = square(n)
print("Square = ",y)
```

Example Program 2 – *lambda function*

```
val = lambda x,y: x+y;

m = int(input("Enter first number:"))
n = int(input("Enter second number:"))
y = val(m,n)
print("Sum = ",y)
```

Example Program

filter() function with lambda

```
#lambda func. to filter odd numbers from a
   list using filter()


oldlist = [2,31,42,11,6,5,23,44]
newlist   =   list(filter(lambda   x:
   (x%2!=0),oldlist))
print("Oldlist: ",oldlist)
print("Newlist: ",newlist)
```

Example Program

map() function with lambda

```
#lambda func. to map numbers incremented by 2 from a list using map()


oldlist = [2,31,42,11,6,5,23,44]
newlist = list(map(lambda x: x+2,oldlist))
print("Oldlist: ",oldlist)
print("Newlist: ",newlist)
```

Recursive function

```
def recur_fact(x):
   if x == 1:
     return 1
   else:
     return(x*recur_fact(x-1))

num = int(input("Enter a number: "))
if num >= 1:
   print("The factorial of ",num," is:
   ",recur_fact(num))
```