Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Part-I
Basics of Programming in Python

P1: Output in Python

1. There is a `print()` statement in python to print something on the screen.
   For example, `print("Welcome to Python Programming")` will print
   Welcome to Python Programming
   on the computer screen
   Python is case-sensitive programming language. Thus `print()` is a valid statement, but Print() is not.

   Each statement should be written in a single line. Thus, `print("X")  print("Y")` is not valid.

2. Write a program to read the name of the user and greet him with a hello message.

```
name = input("Type your name: ")
print("Hello ", name, "!")
```

   Hint: Use `input()` statement to give a prompt to read a name as well as read whatever you type on the keyboard.

3. See the following program. The objective of the program is to read two integer values from the console and print the sum of the numbers. If the following program does not work, do the needful correction.

```
num1 = input("Enter num1: ")  #Read the first number
num2 = input("Enter num2: ")   #Read the second number

num3 = num1 + num2             #Add two numbers
print("Product is: ", num3)
```

   Hint: The return type of input() function is string, so we need to convert the input to integer as `int(input())`. See the following.

```
num1 = int(input("Enter num1: "))     #Read the first number
num2 = int(input("Enter num2: "))     #Read the second number

num3 = num1 * num2                    #Multiply two numbers
print("Product is: ", num3)           #Now, it works!
```

   Note: How a comment against a statement can be given

4. You have to print the following message in nicely formatted way as given below.

```
The Quick Brown Fox jumps over the Lazy Dog
                    and
  Pack my box with Five Dozen Liquor Jugs
```

5. In the following program, see the print statements. Run the program and comment on if they work for you or not. If works, then interpret the output.

```
# This program should read first name and last name
# of the user and display the name that user has entered.

fName = input("Enter your first name: ")
print("\nThanks")
lName = input("Enter your last name: ")

print(fName, lName)        #Two strings can be printed with a single
print()

print("\n" + fName + " " + lName)     #This is another and a better way
```

Note: See how a comment a large enough to fit in a line in a Python program can be given.

6. The following program gives an idea about more on 'print' statement in Python.
The use of `sep=''` allows to separate the text with no space (e.g., `sep=''`), blank space (e.g., `sep='      '`) and or with any character (e.g., `sep=' * '`.
Also, the `end=' '` can be used to end a message with any character including '\n' explicitly.

```
print('Debasis')                #print statement is delimited with a
default '\n'
print("Samanta \n")       #print statement is delimited with an explicit
'\n'

print('F','U', sep='   ', end='  ')     #Here default '\n' is suppressed
print('N')

#\n provides new line after printing the year
print('25',    '12',    '1997', sep='-',     end='\n')
                                  #Extra space does not have any effect
#The different parts are separated with comma and a special '@' after the
#end instead of default '\n'
print('Red','Green','Blue', sep=',', end='@')
```

7. The following program with string modulo operator (%), with which you can print with a number of format to print numbers.

```
# Python program showing how to use string modulo operator(%)

print("Two digit value : %2d, Float value : %5.2f" % (1, 354.1732))

print("Total students : %3d, Boys : %7d" % (240, 120))
# print integer values with different

print("Octal of %2d is %7.3o" % (25, 25))    # print octal value

print("Gravitational constant, G = %10.3E" % (6.6743e-11))    # print
exponential value
```

8. A few more formatting you can learn from the following Python program. Run and understand how they work.

```
print('I love {} for "{}!"'.format('Python', 'Programming', 'for', 'Data
Analytics'))

# Using format() method and referring a position of the object
print('{0} and {1}'.format('Debasis', 'Samanta'))

print('{1} and {0}'.format('Debasis', 'Samanta'))

print(f"I love {'Programming'} in \"{'Python'}!\"")    #Note the use of 'f'
here

# Using format() method and referring a position of the object
print(f"{'Python'} is easy for {'programming'}")
```

Note: Indentation is not allowed while you write a program statement. For example,

print("Hello")

print("Debasis") is wrong!

9. Write a program to read a string and then print the length of characters in it.
   Hint: Use `len()` function. For example, `length = len("Welcome")`

10. Write a program which will merge the words "Welcome", "to", "Python", "Programming" into a single string "Welcome to Python Programming"

11. Following is a program. Interpret the statements under Part-A and Part-B, how they make sense.

```
#Part-A
 x = 10          #Here, x is an integer variable
 print('x =',   x)
 x = 20          #Here, x takes another value
 print('x =',   x)
 x = 30
 print('x =',   x)

#Part-B
 x = 10
 print('x = ' + str(x)) #An integer value is converted into a string value
 x = 20
 print('x = ' + str(x))
 x = 30
 print('x = ' + str(x))
```

12. Write a Python program to delete an i-th letter from a string. Read the string and i from the user. Print the string after the removal.
    Hint: Use string slicing. Delete the previous string. Hint: Use del command.

P2: Data Types in Python

Integer, float, and string are the common data types in Python

13. A variable can be declared as integer, string or a float. See the program below.

```
x = 10              #x is declared as an integer variable
print(x)
y = 20              #y is another integer variable
print(x+y)          #+ is sued to add two integer values and it results their
sum = x + 8
print(sum)
g =6.023e23         # Here, you declare a float point variable
print("x=%d, y=%d, g = %f", x,y g)

fname = "Debasis" #Here, you declare a string variable
lname="Samanta"   #Another string variable
name = fname + lname    # name is a string variable
print(name) #+ is used to merge two string values and it results merged string

#Following is an example of declaring variables of different types:
x, y, z = 10, -15.6, "Philadelphia USA"

#How to print them and their type?

print('x = ', x, type(x), 'y = ', y, type(y), 'z =', z, type(z));
```

Note: Different types of variables can be "explicitly" declared by assigning different types of values

14. Assignment may not only changes a variable's value during its use within an executing program; the type of a variable can change as well. See the following program.

```
a = 10
print('First, variable a has value', a, 'and type', type(a))
a = 15.5
print('Next, variable a has value', a, 'and type', type(a))
a = 'ABC'
print('Now, variable a has value', a, 'and type', type(a))
```

Note: Same variables can be of different types in the same program

15. See the following program and then explain why the program is not correct?

```
x = 10
x = y + x    #Why this is not correct

y=5
x=y+2
y+2=x        #This is not correct either
```

Note: A variable if it is not assigned a value, then it is not declared and hence invalid variable in the program. Further, value cannot be assigned to an expression; expression is should be at right

P3: Input-Output in Python

16. The following program will show how to read values of different types and do some operations on them

```
#By default Python read a value of type string
val  = input("Type anything to read from the keyboard:")
print(val)       #Type: 15, 23.4, Bobby in different runs

#Reading an integer type value
val = int(input("Enter a number: "))
print(val, " ", type(val)). #Type: 15, 23.4, Bobby in different
runs

#Reading a floating type value
val = float(input("Enter a decimal number: "))
print(val, " ", type(val))

#Reading string type value: By default
val = input("Enter your name \n")
print(val, " ", type(val)) #Type: 15, 23.4, Bobby in different
runs
```

Note: When you want to read a float value, say then you should type value accordingly

17. The following program will show some operations with variables.

```
#  This program shows a dvision operation, for an example
#  Get two integers from the user
print('Please enter two numbers to divide.')
x = int(input('Please enter the dividend: '))
y = int(input('Please enter the divisor: '))
#  Divide them and report the result
print(x, '/', y, "=", x/y)
```

Note: Run this program with a) x = 10, y = 2 b) x = 10, y = 0

18. The following program is written without any error. However, it will produce error, if data is/are not entered appropriately.

```
#This following block will work fine for any input value entered
# input
input1 = input()
input2 = input()

# output
print(input1)
print(input2)
```

Hint: Run this program with a) 567 and 3.1732 b) 1.45e12 and ABC

```
#However, the following block may produce errors!
# input
num1 = int(input())
num2 = float(input())

# printing the sum of the values
print(num1 + num2)
```

Hint: Run this program with a) 567 and 3.1732 b) 1.45e12 and ABC

19. **Typecasting:** In Python, you can convert one type of value to another. This is called typecasting. The following program will show different conversions

```
# input
num1 = int(input())      #Read an input value from the keyboard
num2 = float(input())    #User should type a float value here

# Printing the sum in float
print(num1 + num2)

#Typecating int to float
num3 = float(num1)      # Converting int to float
print(num3, type(num3)

#Typecasting float to int  #Converting float to int

num4 = int(num2)
print(num4, type(num4)

print(num3+ num4)
       #Printing the results: Default is float, which is in higher rank

# input
string1 = str(input())   #Anyway, this will read everything as string

# output
print(string1)     #Okay

# Or by default
string2 = input()

# output
print(string2)

#Type casting: From string to int: Here, enter value: 567
x = int(string1)

#Type casting: From string to float: Here, you should enter value: 3.17
y = float(string2)
print(x+y)     #You will see the result in floating point values
```

Note: float to int will result a loss of value as it truncates the value after decimal point

20. **Multi-scanning:** The following program shows how a multiple input can be read form the keyboard with a single **`input()`** statement (with **`split()`**)

```
# Taking two inputs at a time
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
print(x, y, sep=',', end='@')

# Taking three inputs at a time
x, y, z = input("Enter three values: ").split()
print("Number of boys is : ", x)
print("Number of girls is : ", y)
print("They are from : ", z)
print(x, y, z, sep=',', end='@')

# Taking two inputs at a time
a, b = input("Enter two values: ").split()
print("First number is {} and second number is {}".format(a, b))

# Taking multiple inputs at a time
# and type casting using list() function
x = list(map(int, input("Enter multiple values: ").split()))
print("List of students: ", x)
```

Note: the Concept of **`list`**; you will learn ore more about it later

21. Comma separated input to Python program. Alternative to P 19, you can also read input separated by comma, instead of blank space.

```
# Taking multiple inputs at a time separated by comma

x = [int(x) for x in input("Enter multiple values: ").split(",")]
print("Number of list is: ", x)
```

Hint: Run with input: 5,10,15,20,25,30

22. Following are the few programming exercises and you are advised to write programs for them.

    Write a program which will read number of seconds. Then it will print in the format: hh-mm-ss. For example, if you read 1000, then it will print 1-46-40.
    Hint: Use // (gives floor of *x* divided by *y*, e.g., 25//4 = 6) and % (gives remainder of *x* divided by *y*, e.g. 25%4 gives 1). Note, both x and y should be integer values. Similarly, x**y return the value of x to the power y. A hint of the program is given below. However, first you should try of your own!

```
#  Get the number of seconds
 seconds = int(input("Please enter the number of seconds:"))
 #  First, compute the number of hours in the given number of seconds
 #  Note: integer division with possible truncation
 hours = seconds // 3600 # 3600 seconds = 1 hours
 #  Compute the remaining seconds after the hours are accounted for
 seconds = seconds % 3600
 #  Next, compute the number of minutes in the remaining number of seconds
 minutes = seconds // 60    # 60 seconds = 1 minute
 #  Compute the remaining seconds after the minutes are accounted for
 seconds = seconds % 60
 #  Report the results
 print(hours, "hr,", minutes, "min,", seconds, "sec")
```

a. When the program runs, it produces -17.778, which is not correct. Why the following program does not produce correct results? Debug the program and then correct it.

```
# Establish some variables
 degreesF, degreesC = 0, 0
 #  Define the relationship between F and C
degreesC = 5/9*(degreesF - 32)
 #  Prompt user for degrees F
 degreesF = float(input('Enter the temperature in degrees F: '))
 #  Report the result
 print(degreesF, "degrees F =', degreesC, 'degrees C')
```

b. Consider the following program that attempts to compute the circumference of a circle given the radius entered by the user. Given a circle's radius, *r*, the circle's circumference, *C* is given by the formula C = $2\pi r$, where = 3.14159

```
PI = 3.14159
# Formula for the area of a circle given its radius
C = 2*PI*r
# Get the radius from the user
r = float(input("Please enter the circle's radius: "))

# Print the circumference
print("Circumference is", C)
```

The program does not produce the intended result. Why? How can it be repaired so that it works correctly?

P4: A few more on data types in Python

Integer, float and string we have learned. String is an important data type, which needs a few more practices to deal with. Further, in Python, there are other data types, such as **list**, **set**, **tuple**, and **dictionary**, which we are going to learn and practice.

23. The following program gives an idea about the different data types in Python. You have to Google search for the data type which are new to you. The program below is not complete. You have to read the value for each followed by print the value with it's type.

```
# Data type: str
x = "Hello World"

# Data type: int
x = 50

# Data type: float
x = 60.5

# Data type: complex
x = -2+3j

# Data type: list
x = ["apple", "banana", "mango"]

# Data type: tuple
x = ("Ram", "10CS23", 85.5)

# Data type: range
x = range(10)

# Data type: dict
x = {"name": "Suraj", "age": 24}

# Data type: set
x = {"Rose", "Lilly", "Lotus"}

# Data type: frozenset
x = frozenset({"Ada", "Java", "Python"})

# Data type: bool
x = True

# Data type: bytes
x = b"Apple"

# Data type: bytearray
x = bytearray(4)

# Data type: memoryview
x = memoryview(bytes(6))
#Data type: noneType
x = None
```

24. A string is a sequence of characters that can be a combination of letters, numbers, and special characters. It can be declared in python by using single quote (‘), double quote (“), or even triple quote (‘’’). These quotes are not a part of a string, they define only starting and ending of the string. Following program illustrates how a string can be created in a Python program.

```
# Assigning string to a variable
a = 'This is a string'
print (a)
b = "This is a string"
print (b)
c = '''This is a string'''
print (c)
# Python Program for declaring a string variable

# Creating a string with single quotes
String1 = 'Welcome to the Python programming'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a string with double quotes
String2 = "I love Python"
print("\nString with the use of Double Quotes: ")
print(String2)
print(type(String2))
# Creating a string with triple quotes
String3 = '''I'm a programmer and I live programming in "Python"'''
print("\nString with the use of Triple Quotes: ")
print(String3)
print(type(String3))

# Creating String with triple quotes allows multiple lines
String4 = '''  Python
                is an
           Object-oriented
        programming language'''
print("\nCreating a multiline String: ")
print(String4)
```

25. An element in a string can be accessed by using indexing or slicing. Characters in a string of length n are indexed from 0 to n-1. Check the following program to know how a element in string can be accessed.

```
string1 = "Python is Python!"
print("Initial string: ")
print(string1)

# Printing first character
print("\nFirst character of string is: ")
print(string1[0])

# Printing 10-th character
print("\nFirst character of String is: ")
print(string1[10])
```

Continued to…

```
# Printing the last character
print("\nLast character of String is: ")
print(string1[-1])

#Alternatively
l = len(string1)
print("\nLast character of String is: ")
print(string1[l-1])
```

Note: Elements in a string is indexed from last to first as 0, 1, 2, 3, …Also, they are indexed from last to first as -1, -2, -3, …

26. **String slicing**: Slicing is to find substring of a string. Check the following program, which is self-explanatory.

```
# Python Program to demonstrate String slicing

# Creating a String
string1 = "Welcome to Python"
print("Initial String: ")
print(string1)

# Printing 3rd to 12th character
print("\nSlicing characters from 3-12: ")
print(string1[3:12])

# Printing characters between 3rd and 2nd last character
print("\nSlicing characters between " +
      "3rd and 2nd last character: ")
print(string1[3:-2])
```

27. **String reversing**: Reverse a string using slicing. The following program reverses its elements in opposite sequence.

```
#Program to reverse a string
str = "Debasis Samanta"
print(len(str))
str2 = str[::-1]
print(str)

#You cannot write str = str[::-1]
```

Note: String is a immutable data, that is, a string once created cannot be changed. For example, str[0] = 'a' is not an executable statement. Hence, str = str[::-1] is invalid!

28. **String merging**: Two or mor string data can be merged to a single string. The following is a simple program for you.

```
# Python Program to merge two strings

string1 = "Python"
print("Initial string: ")
print(string1)

string2 = "Programming"
print("Another string: ")
print(string2)

# Concatenating the two strings
string3 = string1 + string2
print("\nMerging the two strings: ")
print(string3)
```

29. There are some methods for easy some operations with string data. Following program lists some of the important of them.

```
#Python program for string handling function
String1 = "Hello World!"
#To find the length of string
l = len(string1)

#To change into upper case string
upperString = string.upper()
print(upperString)

#To change into lower case string
lowerString = string.lower()
print(lowerString)

#Converts the first character of each word to upper case
String2 = 'my name is khan'
print(string2.title())  # Also, use capitalize()method

#To find a position of an element in a string
txt = "Hello, welcome to my world."
x = txt.index("welcome")
print(x)
x = txt.find("e")          #Alternative, use index() method
print(x)
x = txt.find("e", 5, 10)   #Search between position 5 and 10
print(x)
```

Note: There many more methods, which you can Google search them

There are four collection data types in the Python programming language:

- **List** is a heterogeneous collection which is ordered, changeable (i.e., mutable) and indexed. Allows duplicate members. Example, list = ['Apple', 35, 45.6, 35]
- **Tuple** is a heterogeneous collection which is ordered, unchangeable (i.e., immutable) and indexed. Allows duplicate members.
  Example: tuple = ("Debasis", 56, 9434008349, 56)

- **Set** is a heterogeneous collection which is unordered, changeable (i.e., mutable), and unindexed. Does not allow duplicate members.
  Example: set = {'A', 'E'. 'I', 'O', 'U', 5}

- **Dictionary** is a heterogeneous collection which is ordered, and changeable (i.e., mutable), and indexed. Does not allow duplicate members.
  Example: dict = {1: 'Apple', 2: 'Banana', 3: 'Mango', 8: 23.45}

In the following few practice program, we shall learn about each of them.

30. **List:** List is an ordered, heterogeneous collection of data. It allows duplicate elements in it. List is mutable, that is, you can add, remove, or change any element in it. Each element in a list is indexed starting from 0. The following program show you how to create a list and do some operation with it.

```
x = 5
# Declaring a list
List1 = ["Apple", 'Banana', "Cherry", x, 4+6]
print(list)

#Displaying second element in the list
print (list1[1])

#Display the number of elements in the list
n = len(list1)
print(n)

#Adding an element in the list
list1.append(6)                #Append 6 at the end
list1.append('Apple')          #By default, append at the end of the list
print(list1)

#Deleting last element from a list
List1.pop()
print (list)

#Creating a list of lists
fruits = ['Apple', 'Banana', 'Mango', 'Orange']
flowers = ['Lilly', 'Lotus', 'Rose']
vowels = ["A", "E", "I", "O", "U"]
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]

megaList [fruits,flowers, digits, vowels]
print("Multidimensional list:")
print(megaList)
print(megalist[3])

#How you can access an element in a list in a list of lists

#Whether a list can allow to include other collection data such, as
# tuple, set, or dictionary
```

Note: The square bracket to declare a list, where each elements are separated by comma(,).
There are many methods to operate a list, which you can learn through a Google search.

31. **Tuple:** A tuple of a student is (<name>, <marks>, <deptt>). For example, ('Debasis', 98.5, "CS") is a tuple.
Consider the following listing, which also shows how to create tuples, list of tuples, tuple of lists and tuple of tuples, etc.

```
# Creating three tuples
tuple1 = ('Debasis', 98.5, "CS")
tuple2 = ("Appu", 65.9, 65.9, "EE")
tuple3 = ("Jyoti", 99.8, "EC")

# List of tuples
list1 = [tuple1, tuple2, tuple3]      #Putting tuples into a list:
print(list1)                          #Printing the list

# Tuple of lists: Example
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list3 = [7, 8, 9]

tuple1 = (list1, list2, list3)  #A tuple is a collection of lists
# Printing iteratively
for x in tuple1
    print(x, sep='/n')

# Tuple of tuples: Example
tuple1 = (1, 2, 3)
typle2 = ('A', 'B', 'C')
tuple3 = ('@', '#', '$")
tuple4 = (tuple1, tuple2, tuple3)
print(tuple4)
```

Note: **Tuple** is a heterogeneous collection which is ordered, unchangeable (i.e., immutable) and indexed. Allows duplicate members. For example: tuple = ("Debasis", 56, 9434008349, 56)

32. Following listing illustrates the difference between list and tuple.

```
 list1 = [1, 2, 3, 4, 5, 6, 7]  #Creating a list
 tuple1 = (1, 2, 3, 4, 5, 6, 7) #Creating a tuple
 print('The list:', list1)
 print('The tuple:', tuple1)

# Access an element
 print('The first element in the list:', list1[0])
 print('The first element in the tuple:', tuple1[0])

# Iterate over the elements of a list
 print('All the elements in the list:', end=' ')
 for x in list1:
     print(x, end=' ')
 print()

# Iterate over the elements of a list
 print('All the elements in the tuple:', end=' ')
 for x in tuple1:
    print(x, end=' ')
print()
```

Continued on …

```
# Slicing the list and tuple
print('List slice:', list1[2:5])  Sublist from 3rd to 6th elements
print('Tuple slice:', tuple1[2:5])

# Modifying the list
print('Try to modify the first element in the list . . .')
list1[0] = 9          # Modify the list
print('The list:', list)

print('Try to modify the first element in the tuple . . .')
tuple1[0] = 9     # ERROR: Tuple is immutable and hence no modification
                  # addition, removal, etc.
```

Note: Tuples are similar to lists, except tuples are immutable.

1) List is […], whereas tuple is (…)
2) Indexing: a = list[i], a = tuple[i]
3) Element modification: list[i] = a (i.e., mutable); tuple[i] = a is NOT possible (i.e. immutable)
4) Element addition: list +=a is possible; tuple += a is NOT possible
5) Element deletion: del list[i] is possible; del tuple[i] is NOT possible
6) Slicing: list[i:j] is possible; tuple[i:j] is possible
7) Iteration: for x in list print(x); for x in tuple print(x);

33. The following program illustrates a few more features with tuple in Python.

```
# Creating a tuple with repetition
tuple1 = ('OM',) * 3
print("\nTuple with repetition: ")
print(tuple)

tuple2 = tuple('GOD IS GOOD',)  # There exist a method tuple()
print("\nTuple with characters from a string: ")
print(tuple2)

# Tuple unpacking
Tuple3 = ("God", "is", "Great!")

(a, b, c) = Tuple2
print("\nValues after unpacking: ")
print(a)
print(b)
print(c)

# Concatenation of tuples
tuple1 = (0, 1, 2, 3)
tuple2 = ('A', 'B', 'C')

tuple3 = tuple1 + tuple2
print("\nTuples after concatenation: ")
print(tuple3)

# Slicing a tuple with a range
print("\nPrinting elements between the range 4-9: ")
tuple4 = tuple3[4:9]
```

Continued on …

```
print(tuple4)
# Slicing a tuple

tuple5 = tuple[1:]      #Slicing from the 2nd to the rest ...
print(tuple5)

# Reversing a tuple
print("\nTuple after sequence of Element is reversed: ")
print(tuple5[::-1])

# Code to test that tuples are immutable
tuple1 = (0, 1, 2, 3)
tuple1[0] = 4        #ERROR: This will not run
print(tuple1)

# Code to test that lists are mutable
list1 = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print("Original list ", list1)

list1[3] = 77
print("Example to show mutability ", list1)
```

34. **Set**: A set collection is similar to tuple and list. There are, however, some difference among them.
    - All are heterogeneous collection of data.
    - Set is an unordered collection, whereas list and tuple are ordered collection. This means, {1, 2, 3} are {3, 1, 2} are the same sets. On the other hand, [1, 2, 3] and [3, 1, 2] are two different lists.
    - Set does not allow duplicate elements, that is, {1, 1, 2, 2} is not a valid set, whereas list and tuples allow duplicate elements; thus (1, 1, 2, 2) is a valid tuple and [1,1,1,1] is a valid list.
    - Set and list both are mutable, i.e., changeable unlike tuple, which is immutable.

Following is a program shows the different ways that a set can be created.

```
#A simple way to create a set
fruits = {"Apple", "Banana", "Orange"}
print(set1, type(fruits))

fruit_list = ['Apple', 'Banana', 'Mangoes', 'Guava', 'Banana']
print(fruit_list, type(fruit_list))

#A list can be converted to a set, that is, typecasting a list to a
set
set2 = set(fruit_list)
print(set2, type(set2))

#A set from a list: Another example
l = [10, 13, 10, 5, 6, 13, 2, 10, 5]
print("List:", l)
s = set(l) # A set is type casted to a list data
print("List:",s)

s2 = {x**2 for x in range(10)} #Create a set of elements within 100
print("Generating set:",s2)
```

Note: The element ordering is not preserved, and duplicate elements appear only once in the set. With set comprehensions and generator expressions, we can build sets. It is also applicable to list.

35. There are several methods to work with set. To know the different methods for set operation, you are advised to Google search. In the following, a few examples are shown.

```
#Set is mutable: You can add, remove, append element, etc.
#Create a set
set1 = set(["a", "b", "c"])
print("Initial set:", set1)

# Adding element to the set: Use add() method
set1.add("d")
print("After adding a:", set2)

#What will happen if you add "a" to set1?
set1.add("a")
print(("Again after adding a:", set1)

#Removing an element into a set: Use remove() method
set1.remove("d")
print(("After removing d:", set1)

#What will happen if you want to remove "e"?
set1.remove("e")
print("After removing e:", set1)

#To remove all elements in a set: Use clear() method
set1.clear()
print("After clear:", set1)
```

36. Following is a listing to illustrate a few set related operations (theses are known in relational algebra).

Operation	Notation	Return	Operator	Method
Union	$A \cup B$	Elements in A or B or both	A|B	union()
Intersection	$A \cap B$	Elements common to both A and B	A&B	intersection()
Set difference	$A - B$	Elements in A but not in B	A-B	difference()
Symmetric difference	$A \oplus B$	Elements in A or B, but not both	A^B	Symmetric_difference
Set membership	$x \in A$	x is a member of A	x in A	x in A
Set membership	$x \notin A$	x is not a member of A	x not in A	x not in A
Set equality	A = B	Sets A and B contain exactly the same elements	A ==B	A ==B

```
# Python Program to demonstrate union oftwo sets
people = {"Jay", "Idrish", "Archil"}
vampires = {"Karan", "Arjun"}
dracula = {"Deepanshu", "Raju"}

# Union using union()function
population = people.union(vampires)

print("Union using union() function")
print(population)

# Union using "|" operator
population = people|dracula

print("\nUnion using '|' operator")
print(population)

# Python program to demonstrate intersection of two sets

set1 = set()       #Initially set1 is empty
set2 = set()       #Initially set2 is empty

for i in range(5):
    set1.add(i)
print("Set 1:", set1)

for i in range(3,9):
    set2.add(i)
print("Set 2:", set2)

# Intersection using intersection() function
set3 = set1.intersection(set2)

print("Intersection using intersection() function")
print("Set 3:", set3)

# Intersection using "&" operator
set3 = set1 & set2

print("\nIntersection using \'&' operator")
print("Set 3:", set3)

# To demonstrate difference of two sets using difference() function
set3 = set1.difference(set2)
set4 = set2.difference(set1)

print(" Difference of two sets using difference() function")
print("Set 3:", set3)

Difference of two sets using '-' operator
set3 = set1 - set2
set4 = set2 - set1
print("\nDifference of two sets using '-' operator")
print("Set 3", set3)
print("Set 4:", set4)
```

Hint: Write programs to check other set operations

37. There are a few more things that you should aware about the set operations. Following is a listing to check comprehensively.

```
#Values of a set cannot be changed. The following statement gives error
fruits = {"Apple", "Banana", "Orange"}
fruits[1] = "Cherry"    #ERROR: Banana cannot be changed to Cherry!

#A set can store heterogeneous elements
set2 = {"Python", 10, 52.7, True, "Fine"}
print("Heterogenous elements in set:", set2)

#List or tuples cannot be put in a set. The following is not valid
set3 = {["A", "a"], [1, 2], [4.5, 6.7]}

# A set can be made immutable using the frozenset() Python program to
demonstrate differences
# between normal and frozen set

normal_set = set(["a", "b","c"])   # Same as {"a", "b","c"}
print("Normal Set")
print("A set:", normal_set)
normal_set.add("d")
print("Afer adding d:", normal_set)

# A frozen set
frozen_set = frozenset(["e", "f", "g"])

print("\nFrozen Set")
print(frozen_set)

# Let's tryto add element to a frozen set
frozen_set.add("h").   #ERROR: Frzoen set is immutable
```

Note: A set otherwise mutable can be made immutable making it as a frozen set

38. **Dictionary:** Dictionary is a collection of **`key:value`** pairs. Dictionary holds **key:value** pair as a sequence of elements within curly **{}** braces, separated by 'comma'. Values in a dictionary can be of any data type (i.e., heterogeneous) and can be duplicated, whereas keys can't be repeated and must be *immutable*. Ordering of elements is not important.

Following is a short listing of a few dictionary management operations.

```
#Creating an empty dictionary
dict = {}
print("Empty dictionary: ")
print(dict)

# Creating a dictionary with integer keys
dict1 = {1: 'Apple', 2: 'Banana', 3: 'Mango'}
print("\nDictionary with the use of integer keys: ")
print(dict1)

# Creating a dictionary with mixed values of keys
dict2 = {'fruit1': 'Cherry', 1: [1, 2, 3, 4], 2: 567, 9: 3.732}
print("\nDictionary with the use of mixed keys: ")
print(dict2)
```

Continued on …

```
# Adding elements one at a time
dict[0] = 'Apple'     # Add  0:'Apple'
dict[2] = 'Banana'    # Add  2:'Banana'
dict[3] = 123         # Add  3: 123
print("\nDictionary after adding 3 elements: ")
print(dict)

# Updating existing Key's Value
Dict[2] = 'Mango'    # 2: 'Banana' becomes 2:'Mango'
print("\nUpdated key value: ")
print(dict)

# To demonstrate accessing an element from a Dictionary

# Creating a Dictionary
dict = {1: 'Apple', 'name': 'Banana', 3: 'Orange'}

# Accessing an element using key
print("Accessing an element using key:")
print(dict['name'])

# Accessing an element using key
print("Accessing a element using key:")
print(dict[1])

# Accessing an element using get()
print("Accessing an element using get() method:")
print(dict.get(3))

# Deleting an element using del() method

print("Dictionary =")
print(dict)
#Deleting some of the element with key value 1
del(dict[1])
print("Data after deletion Dictionary=")
print(dict)
```

39. Few problems for practice is given below. Write program for each.

    a. Write a program which will take the following lists and set to create a larger list, which should include all the elements in the lists list1 and list2 and set set3.
    Also, find the union, intersection and difference of the sets obtained from the list list1 and list2. Print all the sets you have obtained.

```
list1 = [1, 2, 3, 4]
list2 = [1, 4, 2, 3, 5]
set3 = {'a', 'b', 'c'}
```

b. The following is a program which shows how to create a set given a list and can be updated.

```
# Python program to demonstrate the use of update() method
list1 = [1, 2, 3]
list2 = [5, 6, 7]
list3 = [10, 11, 12]

# Lists converted to sets
set1 = set(list2)
set2 = set(list1)

# Update method
set1.update(set2)

# Print the updated set
print(set1)

# List is passed as an parameter which gets automatically
converted to a set
set1.update(list3)
print(set1)
```

Consider another list, say list4 = [1, 3, 5, 7, 9, 11, 13] and update the list list3 with the elements from the list list4.

c. Given a set and a dictionary as below. Write a program which will include all the elements from the dictionary to set.

```
Set1 = {1, 2, 3, 4, 5}
myDict = {6: 'Six', 7: 'Seven', 8: 'Eight', 9: 'Nine', 10: 'Ten'}
```

Hint: `Set1.update(myDict)`

d. The following program create a dictionary and then updating elemnets/ key values in it. Run the program and do the following tasks.

```
# Creating an empty Dictionary
Dict = {}

# Adding elements one at a time
Dict[0] = 'Apple'
Dict[1] = 'Banana'
Dict[2] = ‘Mango’
print("\nDictionary after adding 3 elements: ", Dict)

# Adding set of values to a single Key
Dict[3] = ‘Orange’, ‘Cherry’, ‘Guava’
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[2] = 'Water Mellon'
print("\nUpdated key value: ")
print(Dict)
```

1. Add another element ‘Pine apple’ in the dictionary
2. Delete the element ‘Banana’
3. Delete the element with key value 3

Methods working with a list

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Methods working with set

Functions Name	Description
add()	Adds a given element to a set
clear()	Removes all elements from the set
copy()	Returns a shallow copy of the set
difference()	Returns a set that is the difference between two sets

Functions Name	Description
difference_update()	Updates the existing caller set with the difference between two sets
discard()	Removes the element from the set
frozenset()	Return an immutable frozenset object
intersection()	Returns a set that has the intersection of all sets
intersection_update()	Updates the existing caller set with the intersection of sets
isdisjoint()	Checks whether the sets are disjoint or not
issubset()	Returns True if all elements of a set A are present in another set B
issuperset()	Returns True if all elements of a set A occupies set B
pop()	Returns and removes a random element from the set
remove()	Removes the element from the set
symmetric_difference()	Returns a set which is the symmetric difference between the two sets
symmetric_difference_update()	Updates the existing caller set with the symmetric difference of sets
union()	Returns a set that has the union of all sets
update()	Adds elements to the set

Methods working with tuple

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Methods working with Dictionary

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

P5: Flow Control in Python

40. **Simple if-statement**: The syntax for the statement is

If (Condition):
    Block 1 #This should be indentation
Block2

Syntax:

```
if (condition):
     Executes this block if condition is true
Execute this block if the condition is false
```

Consider the following program, which will read two integer values and then print the largest value

```
x = input("Enter an integer value for x: ", end=' ')
y = input("Enter another integer value for y: ", end=' ')
x = int(x)
y = int(y)

print("\n")
if (x>y):
   print(x, "is the largest")
print(y, "is the largest")
```

41. **If-else statement**:

If (Condition):
    Block 1 #This should be indentation
else:
    Block2

Syntax:

```
if (condition):
     Executes this block if condition is true
else:
      Execute this block if the condition is false
```

Rewriting the program 43, using if-else statelet

```
x = input("Enter an integer value for x: ", end=' ')
x = input("Enter another integer value for y: ", end=' ')
x = int(x)
y = int(y)

print("\n")
if (x>y):
   print(x, "is the largest")
else:
   print(y, "is the largest")
```

42. Consider the following program, which would read any integer number from the user and then print the digits in word only if it is in between 0 to 5, both inclusive

```
value = int(input("Please enter an integer in the range 0...5: "))
if value < 0:
     print("Too small")
 else:
     if value == 0:
         print("zero")
     else:
        if value == 1:
             print("one")
         else:
             if value == 2:
                 print("two")
             else:
                 if value == 3:
                     print("three")
                 else:
                     if value == 4:
                         print("four")
                     else:
                         if value == 5:
                             print("five")
                         else:
                             print("Too large")

print("Done")
```

Note: This is an example called nested if-else statement.

43. Check the following listing which is a better program to divide a number by another number.

```
#  Get two integers from the user
 dividend = int(input('Please enter the number to divide: '))
 divisor = int(input('Please enter dividend: '))
 #  If possible, divide them and report the result
 if divisor != 0:
     print(dividend, '/', divisor, "=", dividend/divisor)
 else:
     print('Division by zero is not allowed')
```

44. Write a program which will read any integer value (positive or negative) and then print the value only if it is positive; it will do nothing for a negative value.

```
x = int(input("Enter any integer value:"))
if x < 0:
    pass  # Do nothing
else:
    print(x)
```

45. **if-elif-else statement**:

If (Condition1):
Block 1 #This should be indentation
elif (Conditin2):
Block2
else:
Block3

Syntax:

```
if (condition-1):
     Executes this block if condition-1 is true
elif (condition-2):
      Execute this block if the condition-2 is true
else:
      Execute this block
```

46. Check the program in 45 which have been rewritten with the if-elif-else statement

```
value = int(input("Please enter an integer in the range 0...5: "))
if value < 0:
     print("Too small")
 elif value == 0:
     print("zero")
 elif value == 1:
     print("one")
 elif value == 2:
     print("two")
 elif value == 3:
     print("three")
 elif value == 4:
     print("four")
 elif value == 5:
     print("five")
 else:
     print("Too large")
 print("Done")
```

47. **Loop-statement: (while):** You can iterate a block of statement. The syntax is

while (Condition):
Block 1 #This should be with indentation

Syntax:

```
while (condition):
     Executes this block
```

Consider the following program, which will print the sum of first n integer values, that is

Sum = 1 + 2 + 3 + …+n

```
i = 0                  #Loop initialization value: Iterator
sum = 0
n = int(input())
while (i <= n)         # Condition checking
   sum = sum + i
    i += 1             # Change (increase) the iterator value

print("Sum of first  %d integer values is %d\n", n, sum)
```

48. **Loop-statement: (for):** You can iterate a block of statement. The syntax is

for (Condition):
Block 1 #This should be with indentation

Syntax:

```
for (condition):
     Executes this block
```

Here, condition should be stated in the form
Condition: i in range(begin, end, step)

- *begin* is the first value in the range; if omitted, the default value is 0
- *end* is one past the last value in the range; the *end* value is always required and may not be omitted
- *step* is the amount to increment or decrement; if the *step* parameter is omitted, it defaults to 1 (counts up by ones)

Consider the following program which will show you the looping with for-statement for the same program as in 51.

```
sum = 0
n = int(input())
for i in range(n+1):       # Condition checking
   sum = sum + i

print("Sum of first  %d integer values is %d\n", n, sum)
```

49. The following listing elaborates how you can use for-loop to calculate the following sum

Sum = 1 + 3 + 5 + 7 +. . . upto n-term

```
sum = 0
n = int(input("How many terms?"))                    #
for i in range(1, 2*n, 2)        # Condition checking
   sum = sum + i

print("Sum of first  %d integer values is %d\n", n, sum)

#The following program will print the first n positive integer
# values starting from 0
for i in range(n):          # Condition checking
   print(i, sep=",", end="@")
```

50. The following listing illustrates a few more task executions with for-statement.

```
#For-staement is used to iterate over a list (also, string, set,
# tuple, and dictionary).

#Following is an example of for-statement  for list iteration
vowels = ['A', 'E', 'I', 'O', 'U']
for x in vowels:           # Iterate for all elements in the list
   print(x)

#Iteration over a string
for x in "Welcome":
   print(x)
print("\n")

#For-loop over a tuple
myTuple = ("Debasis", "CSE", 98.6)
for x in myTuple:
   print(x)
else: print("\n")    #Note: else can be used with for

#For-loop over a set
people = {"Joe", "Bob", "Mary"}
for person  in people:
   print(x)
else: print("\n")    #Note: else can be used with for

#For-loop over a dictionary
people = {'Joe':27, 'Bob':35, 'Mary':25}
for person  in people:
   print(person, sep=" ", end=' ')      #Print the keys
else: print("\n")

for person  in people:
   print(people[person], sep=" ", end=' ')      #Print the values
else: print("\n")
```

51. For-(or while-) statement with **`break`** statement: The break statement can be used to terminate the execution of a loop. Following are the few examples to illustrate the use of break statement.

```
#Run a loop to read numbers and quits when a negative value as input
sum = 0
while(True):    #This is to loop for arbitrary times
   n = int(input("\nEnter any number (positive or negative): "))
   if (n<0): break
   sum = sum + n
print("Sum = ",sum)

#Write the same code with for-statement
 #Hint: for i in range()

fruits = ['apple', 'banana', 'mango', 'orange']
for item in fruits:        # iterate over the collection in the list
   print(item)
  if item == 'mango':         #Note the equality check in Python
      break;
```

52. For-(or while-) statement with **`continue`** statement: The continue statement can be used to bypass (i.e., skip) some statement inside a loop.

```
#Run a loop to read numbers and find the sum of all positive numbers
#Quit the loop when it finds 0

sum = 0
count = 0
while():    #This is to loop for arbitrary times
   n = int(input("\n Enter any number (positive or negative): "))
   if (n>0):
         sum = sum + n
         count +=1
   if n == 0: break
print("\nSum of %d positive numbers is %d", count, sum)


fruits = ['banana', 'apple', 'banana', 'mango', 'orange', 'apple']
for item in fruits:        # iterate over the collection in the list
  if item == 'apple':         #Note the equality check in Python
      continue;
   if item == 'banana':          #Note the equality check in Python
      continue;
   if item = 'orange': break
   print(item, sep=' ', end='')
```

P6: Functions in Python

Function is a programming paradigm, with which a programmer can define a procedure to solve a problem and then call the function from the main program or from another function, even the same function (then it is called recursive function). Function writing supports code debugging, code reusability, code maintainability, etc.The syntax for defining a function:

```
Syntax:
def function_name([parameter: data_type]) [-> return_type:]
    ["""Docstring"""]
    # body of the function
    [return expression]
```

The clauses with […] is optional.

53. Following listing illustrates how to define a function and call functions from programs

```
def my_function():
    print("Hello from a function")

#To call a function, use the function name followed by parenthesis
print("\n 1. Calling the function..")
my_function()

print("\n 2. Calling the function again…")
my_function()

#Defining a function with argument(s)
def function1(fname):
    print("Hi" + fname +"!")

#Calling the function...
function1("Ananya")
function1("Debasis)
function1("Anwesha")

#Functin with two arguments
def function2(fname, lname):
   print(\n How are you " + fname + lname +"?")

function2("Ananya", "Samanta")
function2("Angelina", "Fox")
function2("Maradona", "Diego")

#Functin with three arguments
def function3(fname, lname, age):
   print("\n Are you %d years old?", age)

function3("Abc", "Xyz", 25)
function3("Mnp", "Wvx", 52)
function3("Ijk", "Pqr", 49)
#Functions with specific type of arguments and return type
def add(num1: int, num2: int) -> int:
    """Add two numbers and return result"""
    num3 = num1 + num2
    return num3
```

Continued on ...

```
# Driver code
num1, num2 = 5, 15
ans = add(num1, num2)
print(f"\n The addition of {num1} and {num2} results {ans}.")
```

54. You can define many functions at the same time. Following listing illustrates the same.

```
def sum(a: int, b: int) -> int:
     """ This function adds two numbers"""
 return (a + b)

def sub(a: int, b: int) -> int:
    """ This function subtracts between two numbers"""
    return (a - b)

def div(a: int, b: int) -> float:
    """ This function divides one number by other"""
    return (a/b)
def mod(a: int, b: int) -> int:
    """ This function calculates modulo remiander between two
numbers"""
    return (a%b)

def mul(a: int, b: int) -> int:
    """ This function calculates product of two numbers"""
    return (a*b)

# Driver code
a = int(input('Enter 1st number: '))
b = int(input('Enter 2nd number: '))

print(f'Sum of {a} and {b} is {sum(a, b)}')
print(f'Subtraction of {a} and {b} is {sub(a, b)}')
print(f'Division of {b} by {a} is {div(a, b)}')
print(f'Modulo remainder when {a} is divided by {b} is {mod(a,
b)}')
print(f'Multiplication of {a} and {b} is {mul(a, b)}')
```

Note: If you call a function with wrong type of values, it will produce a run-time error.

55. **Function with unknown number of arguments passed**

If we do not know how many arguments that will be passed into a function, then add a * before the parameter name in the function definition. This way the function will receive a *tuple* of arguments, and can access the items accordingly. Following is a listing to illustrate it.

```
def var_arg_f(*kids):
    print(kids)

# Call with one argument
var_arg_f("John”)
```

Continued on ...

```
# Call with two arguments
var_arg_f("Kareena", "Karishma")

# Call with three arguments
var_arg_f("Amit", "Bishnu", "Rahim")

# Call with four arguments
   var_arg_f("Ram", "Laxman", "Bhara", "Shatrughna")
```

56. **Function with default argument(s)**.

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. Zero or more arguments can be defined with their default values. Following is a program to illustrate it.

```
# Python program to demonstrate default arguments
def myFun(x, y=50):
    z = x + y
    print(" Sum of %d and %d is : %d", x, y, z)

# Driver code
myFun(10)         #For this call the default value of y is 50
myFun(15, 25)     #For this call the value passed to y is 25
```

Note: You should always pass the value to non-default argument always

57. **Function call with arbitrary ordering**.

Python allows the caller to specify the argument name with values so that the caller does not need to remember the order of parameters. Following is an illustration.

```
# Python program to demonstrate varible ordering in arguments
def student(fname, lname):
    print(fname, lname, sep=' ')

def child(child3, child2, child1):
  print("The youngest child is " + child3)

# Driver code with keyword arguments
student(fname='Albert', lname='Einstein')
student(lname='Einstein', fname='Albert')

child(child1 = "Ram", child2 = "Laxman", child3 = "Bharat")

# Function argument if not call with position, it may give wrong result
def nameAge(name, age: int):
    print("Hi, I am", name)
    print("My age is ", age)

# You will get correct output if arguments are given in order
print("Case-1:")
nameAge("Suraj", 27)

# You will get incorrect output because arguments are not in order
print("\nCase-2:")
nameAge(27, "Suraj")
```

58. **Variable number of arguments passed.**
In extension to #60, Python allows to pass a variable number of arguments. For this, define a function with *argv. Check the following listing.

```
# Python program to illustrate *args for variable number of arguments
def myFun(*argv):
    for arg in argv:
        print(arg, end=' ')

myFun('Hello', 'Welcome', 'to', 'Python')
print('\n')
myFun('A', 'E', 'I', , 'O', 'U', 555)
print('\n')
myFun()
myFun(123, 345, 567, 789)
```

Note: Here, all arguments' values are passed as a string.

59. This is the one support when you don't know how many arguments to be passed while you call a function and call the function with (key, value) pair(s). Define such a function with **kwargs as argument. Following is an example illustrating this.

```
def myFun(**kwargs):
    for key, value in kwargs.items():
        print("%s=%s" % (key, value))

# Driver code
myFun(roll='CS61061', name='Zebra', marks=98)
myFun(rollNo='CS41043', fname='Lion', lname=' King', marks=86)
myFun(a='A', b='B', c='C', d=86, e='It is a value to key e')
```

60. Passing a list as an argument.

```
def my_function(food):
  for x in food:
    print(x, sep=',', end=' ')

fruits = ["apple", "banana", "cherry"]
people = ["Joe", "Kim", "Linda", "Bob", "Kim")
print("\n Fruits:)
my_function(fruits)
print("\n People:)
my_function(people)
```

Hint: Similarly, other collections also can be passed as an argument to a function.

61. **Nested function.** A function can call other function. Check the following listing to undersand the concept.

```
def f1():
    print("\n1. I am f1: \n")

def f2():
   print("2. I am from f2:\n")

    def f23():
        print("3. I am inside f2: ")
        f1()          #fi() is called from here...


# Driver's code
f1()
f2()
```

62. **Recursive function writing:** When a function call itself, is called a recursive function. Following code includes a recursive functions to calculate a) Sum of first n integers and b) factorial of a positive integer value.

```
      def sum(n: int) -> int:
            """To find the sum of first n numbers recursively"""
            if (n<=0):
                  result = 0 # Boundary condition to terminate recursion
            else:
                  result = n + sum(n-1) #Recursive call of the function
      return result

      # Driver code
      n = int(input( "Enter the value of n :"))
      print(f"Sum of first{n} integers is {result}')


      def factorial(n: int) -> int:
            """To find the factorial of first n numbers recursively"""
            if (n=0):
                  result = 1 # Boundary condition to terminate recursion
            else:
                  result = n * factorial(n-1) #Recursive call of the
      function
      return result

      # Driver code
      n = int(input( "Enter the value of n :"))
      if (n < 0): print("Invalid entry! Enter positive ineger value:\n")
      else:
            print(f"factorial of first{n} integers is {result}')
```

Hint: Write the equivalent codes for iterative functions.
For all iterative functions, equivalent recursive functions are possible.

P7: File handling in Python

Concept:

The key function for working with files in Python is the open() function.

fName = open(filename, fMode)

The open() function takes two parameters; filename, and mode.

There are four different modes for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist
"a" - Append - Opens a file for appending, creates the file if it does not exist
"w" - Write - Opens a file for writing, creates the file if it does not exist
"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode
"b" - Binary - Binary mode (e.g. images)

63. Following are the small pieces of codes illustrating how the different ways that a file can be handles and used. You are advised to test each program.

Creating a file to write some text into it

```
# Python code to create a file
file = open('test.txt','w')  # Opening the file in write mode
file.write("This is the write command")
file.write("It allows us to write in a particular file")
file.close()   # Closing the file

# Alternative method: with-write()
# Python code to illustrate with() along with write()
with open("test2.txt", "w") as f:
    f.write("Hello World!!!")
```

Opening an existing file

```
# a file named "test.txt" is already available in your machine
# This program will open the file in reading mode.

file1 = open('test.txt')       # Default is read and text mode
file2 = open('test.txt', 'r')
file3 = open('test.txt', 'rt')

# This will print every line one by one in the file
for each in file1:
    print (each)
```

Other ways of file opening

```
# Alternative Way 1: Using read() method
# Python code to illustrate read() mode
file = open("text.txt", "r")
print (file.read())


# Alternative Way 2: append mode
# Python code to illustrate append() mode
file = open('test.txt', 'a')
file.write("This will add at the end of the file.")
file.close()

file1 = open('test.txt')
for each in file1:
    print (each)


# Alternative Way 3: with-read()
# Python code to illustrate with()
with open("text2.txt") as file:
    data = file.read()
print(data)


# Another Way 4: Reading a set of character of a given size
# Python code to illustrate read() mode character wise
file = open("text.txt", "r")
print (file.read(5))    # Read five characters, by default whole


#Alternative Way 5: Read all words in a file, using split() method
# Python code to illustrate split() function
with open("test.txt", "r") as file:
    data = file.readlines()
    for line in data:
        word = line.split()
        print (word)
```

Writing some important file handling functions

```
import os

def create_file(filename):
    try:
        with open(filename, 'w') as f:
            f.write('Hello, world!\n')
        print("File " + filename + " created successfully.")
    except IOError:
        print("Error: could not create file " + filename)

def read_file(filename):
    try:
        with open(filename, 'r') as f:
            contents = f.read()
            print(contents)
    except IOError:
        print("Error: could not read file " + filename)

def append_file(filename, text):
    try:
        with open(filename, 'a') as f:
            f.write(text)
        print("Text appended to file " + filename + " successfully.")
    except IOError:
        print("Error: could not append to file " + filename)

def rename_file(filename, new_filename):
    try:
        os.rename(filename, new_filename)
        print("File " + filename + " renamed to " + new_filename + "
successfully.")
    except IOError:
        print("Error: could not rename file " + filename)

def delete_file(filename):
    try:
        os.remove(filename)
        print("File " + filename + " deleted successfully.")
    except IOError:
        print("Error: could not delete file " + filename)


if __name__ == '__main__':
    filename = "example.txt"
    new_filename = "new_example.txt"

    create_file(filename)
    read_file(filename)
    append_file(filename, "This is some additional text.\n")
    read_file(filename)
    rename_file(filename, new_filename)
    read_file(new_filename)
    delete_file(new_filename)
```

Python has a set of methods available for the file object.

Method	Description
close()	Closes the file
detach()	Returns the separated raw stream from the buffer
fileno()	Returns a number that represents the stream, from the operating system's perspective
flush()	Flushes the internal buffer
isatty()	Returns whether the file stream is interactive or not
read()	Returns the file content
readable()	Returns whether the file stream can be read or not
readline()	Returns one line from the file
readlines()	Returns a list of lines from the file
seek()	Change the file position
seekable()	Returns whether the file allows us to change the file position
tell()	Returns the current file position
truncate()	Resizes the file to a specified size
writable()	Returns whether the file can be written to or not
write()	Writes the specified string to the file
writelines()	Writes a list of strings to the file

---*---