Amey Thakur
SE-B-50

Terna Engineering College
Department of Computer Engineering

Assignment No. 1

Class: SE – A, B, C **Year: FH - 20**
Subject: OSTL

Sr. No.	Question	CO Mapping	PO Mapping	BL
1	Explain the following: 1. Control Statements 2. Data types 3. Inheritance 4. Exception Handling	1		
2	Differentiate tuple and list with example.	1		
3	Discuss file handling concepts along with Packages.	2		
4	Elaborate Text Processing and Regular Expression in Python.	2		
5	Write a Program for creating Stack and Queue.	3		
6	Write a Program for creating graph and tree.	3		

Amey Thakur
SE-B-50

54 / 60 Pencil

Terna Engineering College, Navi Mumbai

Q.1 Explain the following

1] Control statements

- Control statements in python are used to control the order of execution of the program based on the values and logic.

Python provides us with 3 types of control statements:

→ Continue
→ Break
→ Pass

① Continue statement

- When the program encounters continue statement, it will skip the statements which are present after the continue statement inside the loop and proceed with the next iteration.

Syntax : continue

Example :

```
for char in 'Python':
if (char == 'y'):
    continue
print ("Current character: ", char)
```

Output :

```
Current Character : P
Current Character : t
Current Character : h
Current Character : o
Current Character : n
```

② Break statement

- The break statement is used to terminate the loop containing it, the control of the program will come out of that loop

Syntax : break

Example:

```
for char in 'Python' :
    if (char == 'h') :
        break
    print (" Current Character : ", char)
```

Output:

```
Current Character : P
Current Character : y
Current Character : t
```

③ Pass statement

- Pass statement in python is a null operation, which is used when the statement is required syntactically

Syntax : pass

Example:

```
for char in 'python' :
    if ( char == 'h') :
        pass
    print (" Current Character : ", char)
```

Output:

```
Current Character : p
Current Character : y
Current Character : t
Current Character : h
Current Character : o
Current Character : n
```

Terna Engineering College, Navi Mumbai

2] Data Types

- A data type describes the characteristic of a variable
- Python has six standard Data Types
  - → Numbers
  - → String
  - → List
  - → Tuple
  - → set
  - → Dictionary

① Numbers

- In numbers, there are mainly 3 types which include Integer, Float and complex
- These 3 are defined as a class in python.
- In order to find to which class the variable belongs to you can use type() function

② String

- A string is an ordered sequence of characters.
- We can use single quotes or double quotes to represent strings.
- Multi-line strings can be represented using triple quotes
- Strings are immutable which means once we declare a string we can't update the already declared string

③ List
- A list can contain a series of values
- List variables are declared by using brackets [ ].
- A list is mutable, which means we can modify the list

④ Tuple
- A tuple is a sequence of python objects separated by commas.
- Tuples are immutable, which means tuples once created cannot be modified.
- Tuples are defined using parenthesis ( ).

⑤ Set
- A set is an unordered collection of items.
- Set is defined by values separated by a comma inside braces { }.

⑥ Dictionary
- Dictionaries are the most flexible built in data type in python.
- Dictionary items are sorted and fetched by using the key.
- Dictionaries are used to store a huge amount of data
- To retrieve the value we must know the key.
- In python, dictionaries are defined within braces { }
- We use the key to retriere the respective value. But not the other way around

Terna Engineering College, Navi Mumbai

3] Inheritance

- It is an important aspect of the object oriented paradigm.
- Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch.
- In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provides its specific implementation to the functions of the parent class
- In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name.

10

4] Exception Handling

- If the python program contains suspicious code that may throw the exception, we must place that code in the try block
- The try block must be followed with the except statement which contains a block of code that will be executed if there is some exception in the try block
- Syntax:

```
try
    {
        ..... Run this code
    }
except
    {
        ..... Run this code if an exception
        occurs.
    }
```

Terna Engineering College, Navi Mumbai

Q2. Differentiate tuple and list with example

List

- List is a container to contain different types of objects and is used to iterate objects
- Example : list = ['a', 'b', 'c', 'd', 'e']

Tuples

- Tuple is similar to list but it contains immutable objects.
- Tuple processing is faster than list
- Example : tuples = ('a', 'b', 'c', 'd', 'e')

	Key	List	Tuple
1	Type	List is mutable	Tuple is immutable
2	Iteration	List iteration is slower and is time consuming	Tuple iteration is faster
3	Appropriate for	List is useful for insertion and deletion operations	Tuple is useful for read only operations like accessing elements
4	Memory Consumption	List consumes more memory	Tuple consumes less memory
5	Methods	List provides many in built methods	Tuples have less in built methods
6	Error prone	List operations are more error prone	Tuples operations are safe.

8

Terna Engineering College, Navi Mumbai

Q3. Explain file handling and packages

File Handling

- The key function for working with files in Python is the open() function
- The open() function takes two parameters
  - → filename
  - → mode
- There exists 4 different methods (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 specific file, returns an error if the file exists.
- In addition to this, file can be handled as
  - → Binary Mode
  - → Text Mode
  - "t" - Text - Default value. Text mode
  - "b" - Bingry mode (eg images)

Syntax: To open a file for reading it is enough to specify the name of the file

```
f = open ("demofile.txt")
```

Terna Engineering College, Navi Mumbai

Packages

- Package is a collection of python modules. i.e. a package is a directory of Python modules containing an additional __init__.py file.
- The __init__.py distinguishes a package from a directory that just happens to contain a bunch of Python scripts.

8

Terna Engineering College, Navi Mumbai

Q4. Elaborate Text Processing and Regular Expression in python.

Text Processing

- Python programming can be used to process text data for the requirements in various textual data anlysis.
- A very important area of application of such text processing ability of python is for NLP (Natural Language Processing).
- NLP is used in search engines, newspapers feed analysis and more recently for voice based applications like Siri and Alexa
- Python's Natural Language Toolkit (NLTK) is a group of libraries that can be used for creating such Text Processing systems

Terna Engineering College, Navi Mumbai

Regular Expression or RegEx

- A RegEx or Regular Expression is a sequence of characters that forms a search pattern.
- RegEx can be used to check if a string contains the specific search pattern.
- Regular Expressions are used to identify whether a pattern exists in a given sequence or characters (strings) or not.
- They help in manipulating textual data, which is often a pre-requisite for data science projects that involve text mining
- In python, regular expressions are supported by the re module.
- Example:

```
import re

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

Reg Ex Functions

- The re module offers a set of functions that allows us to search a string for a match

Terna Engineering College, Navi Mumbai

Function	Description
① Findall	- Returns a list containing all matches
② Search	- Returns a match object if there is a match anywhere in the string
③ split	- Returns a list where the string has been split at each match
④ sub	- Replaces one or many matches with a string

Q5. Write a program for creating stack and queue.

- Program for creating stack

```
class stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def push(self, data):
        self.items.append(data)

    def pop(self):
        return self.items.pop()

s = Stack()
while True:
    print('push <value>')
    print('pop')
    print('quit')
    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()
    if operation == 'push':
        s.push(int(do[1]))
    elif operation == 'pop':
        if s.is_empty():
            print('stack is empty.')
        else:
            print('popped value:', s.pop())
    elif operation == 'quit':
        break
```

Terna Engineering College, Navi Mumbai

- Program for creating queue.

```
class Queue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def enqueue(self, data):
        self.items.append(data)

    def dequeue(self):
        return self.items.pop(0)

q = Queue()
while True:
    print('enqueue <value>')
    print('dequeue')
    print('quit')
    do = input('what would you like to do?').split()

operation = do[0].strip().lower()
    if operation == 'enqueue':
        q.enqueue(int(do[1]))
    elif operation == 'dequeue':
        if q.is_empty():
            print('Queue is empty!')
        else:
            print('Dequeue value:', q.dequeue())
    elif operation == 'quit':
        break
```

Q6. Write a program for creating graph and tree

- Program for creating graph

```
class Graph:
    def __init__(self):
        # dictionary containing keys that map to the corresponding
          vertex object.
        self.vertices = { }

    def add_vertex(self, key):
        "" Add a vertex with the given key to the graph""
        vertex = Vertex(key)
        self.vertices[key] = vertex

    def get_vertex(self, key):
        "" Return vertex object with the corresponding key.""
        return self.vertices[key]

    def __contains__(self, key):
        return key in self.vertices

    def add_edge(self, src_key, dest_key, weight=1):
        """ Add edge from src_key to dest_key with given weight."""

        self.vertices[src_key].add_neighbour(self.vertices[dest_key],
        weight)

    def does_edge_exist(self, src_key, dest_key):
        "" Return True if there is an edge from src_key to dest_key.""
        return
        self.vertices[src_key].does_it_point_to(self.vertices[dest_key])

    def __iter__(self):
        return iter(self.vertices.values())
```

Terna Engineering College, Navi Mumbai

```
class Vertex :
    def __init__(self, key):
        self.key = key
        self.points_to = {}

    def get_key(self):
        """ Return key corresponding to this vertex object """
        return self.key

    def add_neighbour(self, dest, weight):
        """ Make this vertex point to dest with given edge weight. """
        self.points_to[dest] = weight

    def get_neighbours(self):
        """ Return all vertices pointed to by this vertex. """
        return self.points_to.keys()

    def get_weight(self, dest):
        """ Get weight of edge from this vertex to dest """
        return self.points_to[dest]

    def does_it_point_to(self, dest):
        """ Return True if this vertex points to dest. """
        return dest in self.points_to

g = Graph()
print('Menu')
print('add vertex <key>')
print('add edge <src> <dest> [weight]')
print('display')
print('quit')
```

```
while True:
  do = input ('What would you like to do?'). split()

operation = do[0]
if operation == 'add':
  suboperation = do[1]
  if suboperation == 'vertex':
    key = int (do[2])
    if key not in g:
      g. add_vertex (key)
    else:
      print ('Vertex already exists.')
    elif suboperation == 'edge':
      src = int (do[2])
      dest = int (do[3])
      if src not in g:
        print ('vertex {} does not exist', format (src))
      elif dest not in g:
        print ('vertex {} does not exist'. format (dest))
      else:
        if not g. does_edge_exist (src, dest):
          if len (do) == 5:
            weight = int (do[4])
            g. add_edge (src, dest, weight)
          else:
            g. add_edge (src, dest)
          else:
            print ('Edge already exists')
```

```
elif operation == 'display':
    print('Vertices:', end = '')
for v in g:
    print (v.get_key(), end = ' ')
print()

print ('Edges: ')
for v in g:
    for dest in v.get_neighbours():
        w = v.get_weight(dest)
        print ('(src={}, dest={}, weight{})'.format(v.get_key(),
                                                    dest.get_key(), w))
    print()

elif operation == 'quit':
    break
```

- Program for creating Trees

```
class BinaryTree :
  def __init__(self, key = None):
    self.key = key
    self.left = None
    self.right = None

  def set_root(self, key):
    self.key = key

  def inorder(self):
    if self.left is not None:
      self.left.inorder()
    print(self.key, end = ' ')
    if self.right is not None:
      self.right.inorder()

  def insert_left(self, new_node):
    self.left = new_node

  def insert_right(self, new_node):
    self.right = new_node

  def search(self, key):
    if self.key == key:
      return self
    if self.left is not None:
      temp = self.left.search(key)
```

```
    if temp is not None:
        return Temp
    if self.right is not None:
        temp = self.right.search(key)
        return temp
    return None

btree = None

print('Menu (this assumes no duplicate keys)')
print('insert <data> at root')
print('insert <data> left of <data>')
print('insert <data> right of <data>')
print('quit')

while True:
    print('inorder traversal of binary tree:', end=" ")
    if btree is not None:
        btree.inorder()
    print()

    do = input('What would you like to do? ').split()

    operation = do[0].strip().lower()
    if operation == 'insert':
        data = int(do[1])
        new_node = BinaryTree(data)
        suboperation = do[2].strip().lower()
        if operation == 'at':
            btree = new_node
```

```
else :
    position = do [4] . strip () . lower ()
    key = int (position)
    ref_node = None
    if btree is not None :
        ref_node = btree . search (key)
    if ref_node is None :
        print ('No such key.')
        continue
    if suboperation == 'left' :
        ref_node . insert_left (new_node)
    elif suboperation == 'right' :
        ref_node . insert_right (new_node)

elif operation == 'quit' :
    break
```