Solution/Model Answer
Python Application Programming – 15CS664 – IAT-2
6th Sem CSE (A, B & C)
Dr. P. N. Singh, Professor(CSE)

1.

a. Differentiate array and list in Python. [05]

Ans:

List

- List stores different data types & mostly used in python
- Takes more memory space for variable length of data.

Example:

```
>>> marks_list = [ [222, 'Rubiya', 67], [333,'Khushboo', 58],[444,'Chinmay', 82]]
>>> for marks in marks_list:
        print(marks)

[222, 'Rubiya', 67]
[333, 'Khushboo', 58]
[444, 'Chinmay', 82]
```

Array

- Array stores values of same type
- We will have to import arrays
- Useful for memory efficiency

```
from array import *
>>> x=array('i',[3,6,9,12]) # i is typecode
>>> for a in x:
        print(a/3)

1.0
2.0
3.0
4.0
>>>x.append(15)
>>>print( x)
array('i', [3, 6, 9, 12, 15])
```

Some comments over list vs array:

- Python lists are implemented internally as variable-length arrays, rather than linked lists.
- Main difference is the function performed on them
- Since list may have different data types so any operation for all elements like array cannot be performed

b. Differentiate pop() method and del operator in list with examples [05]

Ans:

pop() – deleting elements, if element number is not given then it pops & deletes last element. Popped value can be stored to other variable.

```
>>> namelist.pop(5)
'paras'
>>> namelist
['Akrur', 'Vinay', 'Vinayak', 'ajay', 'kunti', 'samartha']
>>> namelist.pop()
'samartha'
>>> namelist
['Akrur', 'Vinay', 'Vinayak', 'ajay', 'kunti']
>>> x=namelist.pop(0)
>>> x
'Akrur'
>>> namelist
['Vinay', 'Vinayak', 'ajay', 'kunti']
>>>
```

del operator deletes the value
(if not required to save to other variable)

```
>>> del namelist[1]
>>> namelist
['Vinay', 'ajay', 'kunti']
>>>
```

2.

a. Imply split and join in list with examples. [05]

Ans:

split() method

We can call split with an optional argument called a delimiter that specifies which characters to use as word boundaries. The following example uses a hyphen as a delimiter:

```
>>> gana="dil-mera-lay-gai-Sona-Sona"
>>> gana.split(delimeter)
['dil', 'mera', 'lay', 'gai', 'Sona', 'Sona']
```

join() method

join is the inverse of split. It takes a list of strings and concatenates the elements. join is a string method, so you have to invoke it on the delimiter and pass the list as a parameter:

```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter=' '
>>> delimiter.join(t)
'pining for the fjords'
```

b. Define aliasing & implement with list example. [05]

Ans:

Aliasing: if a refers to an object we assign b = a, then both variables refer to same object. The association of a variable with an object is called a reference. In this example, there are two references to the same object. We say the object is *aliased*.

```
>>> nlist3=nlist1
>>> nlist1 is nlist3
True
>>>
```

If the aliased object is mutable, changes made with one alias affect the other: In general, it is safer to avoid aliasing when you are working with mutable objects.

```
>>> nlist3[0]=99
>>> nlist1
[99, 3, 4]
```

So, in case of strings we know that strings are immutable. Aliasing the string object aliasing is not as much of a problem but we can't change/modify characters of string.

```
>>> name1[0]='k'
TypeError: 'str' object does not support item assignment
```

3. Write a program Python to pass the list as argument to a function and to return list. [10]

Ans:

Passing list as arguments: When we pass a list to a function, the function gets a reference to the list. If the function modifies a list parameter, the caller sees the change.

```
>>> names=['Paras','Kumar','Yash']
>>> def delfirst(strlist):
        del strlist[0]

>>> delfirst(names)
>>> names
['Kumar', 'Yash']
```

 # Program to return a list

```
def delfirstlast(list1):
    del list1[0]
    del list1[len(list1)-1]
    return list1

list2=[1,3,5,7,9]
list3=delfirstlast(list2)
```

```
print(list3)
```

output:
[3, 5, 7]

4.

a. Define tuple of python with example of initializing a tuple. 'Tuples are immutable': explain with example. [06]

Ans:

A tuple is a sequence of values much like a list. The values stored in a tuple can be any type, and they are indexed by integers. The important difference is that **tuples are immutable**. Tuples are also comparable and hashable so we can sort lists of them and use tuples as key values in Python dictionaries.

Syntactically, a tuple is a comma-separated list of values:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Although it is not necessary, it is common to enclose tuples in parentheses to help us quickly identify tuples when we look at Python code:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

To create a tuple with a single element, you have to include the final comma:

```
>>> t1 = ('a',)
>>> type(t1)
 <type 'tuple'>
```

Without the comma Python treats ('a') as an expression with a string in parentheses that evaluates to a string:

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
>>> names=('paras','chandan','vikram')
>>> type(names)
<class 'tuple'>
```

Another way to construct a tuple is the built-in function tuple. With no argument, it creates an empty tuple:

```
>>> t = tuple()
>>> print(t)
 ()
```

b. Write output with justification of following comparison of tuples: [04]

I. (4,2,5) > (4,3,1)
II. (2,2,2) > (2>2>2)
III. (2,2,2,1) > (2>2>2)
IV. ('abc',2,'xyz') < ('xyz',2,'abc')

Ans:

i. >>> (4,2,5) > (4,3,1)
False # 4 equals 4 so second elements 2 > 3 compared & results False.

ii. >>> (2,2,2) > (2>2>2)
True # second expression is false & True > False is True
If second expression is (2,2,2) then False because it has not reached to conclusion.

iii. (2,2,2,1) > (2>2>2)
True # second expression is false & True > False is True
If second expression is (2,2,2) then also it will return True because 4th element of first tuple has nothing to compare with second

iv. ('abc',2,'xyz') < ('xyz',2,'abc')
True # because first expression 'abc' is < 'xyz'

5. How to print keys of dictionary in alphabetical order? Explain with example code in Python. [10]
Ans:

If we want to print the keys in alphabetical order, we first make a list of the keys in the dictionary using the keys method available in dictionary objects, and then sort that list and loop through the sorted list, looking up each key and printing out key-value pairs in sorted order as follows:

```
>>> marks={'Ricky':45,'Micky':67,'Donald':76,'Janhavi':55,'Yash':88}
>>> for key in marks:
        print(key, marks[key])
```

```
Ricky 45
Donald 76
Micky 67
Yash 88
Janhavi 55
```

Now for keys in alphabetical order:

```
>>> list1=list(marks.keys())
>>> list1.sort()
>>> for key in list1:
        print(key,marks[key])
```

```
Donald 76
Janhavi 55
Micky 67
Ricky 45
Yash 88
```

6.

a. Differentiate tuple and list with examples. [05]

Ans:

Tuple & List

- A tuple is an (immutable) ordered list of values. It is in many ways similar to a list.
- **Most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot.**
- **Tuples have structure, lists have order.**
- Tuple occupies smaller size so it becomes a bit faster for big-data range.
- Tuple is immutable so can be used as key in dictionary

Comments:

- If a list is unlikely to be changed, we should use tuples instead of lists.
- Lists are more common than tuples, mostly because they are mutable. But there are a few cases where we may prefer tuples:
    - In some contexts, like a return statement, it is syntactically simpler to create a tuple than a list. In other contexts, you might prefer a list.
    - If you want to use a sequence as a dictionary key, you have to use an immutable type like a tuple or string.
    - If you are passing a sequence as an argument to a function, using tuples reduces the potential for unexpected behavior due to aliasing.

Example of list:

```
>>> grade=['a','First',3,2.5]
>>> grade
['a', 'First', 3, 2.5]
>>> type(grade)
<class 'list'>
>>>lst1=[]  #empty list
>>>lst1
[]
```

Example of Tuple:

```
>>> grade=('a','First',3,2.5)
>>> grade
('a', 'First', 3, 2.5)
>>> type(grade)
<type 'tuple'>
>>>t1=()
>>>t1
()
```

b. Write a program to open the file romeo.txt and read it line by line. For each line, split the line into a list of words using the split function. For each word, check to see if the word is already in a list. If the word is not in the list, add it to the list. When the program completes, sort and print the resulting words in alphabetical order. [05]

Ans:

```
#sorting unique words of a file
fhand = open('romeo.txt')
allwords=[]
for line in fhand:
   words = line.split()  #splitting line in words
   for word in words:
      if word not in allwords:
         allwords.append(word)  # adding only unique words in list

allwords.sort() # case sensitive on ASCII value
print('\nAll unique words in sorted manner: - ASCII value')
print(allwords)

allwords.sort(key=lambda x:x.lower()) #  alphabetical order ignoring the case
print('\nAll unique words in alphabetical order: - ignoring the case')
print(allwords)
```

7.

a. Open file employee.txt and write Python command with regular expression (importing re) for the followings: [06]

   i. To print the lines having words “singh” and “sinha”
   ii. To print lines having words “agarwal” and “agrawal”
   iii. To print the lines ending with “manager”
   IV. To print the lines starting with ‘P’

Ans:

```
>>> open('employee.txt')
>>> import re
```

i. To print the lines having words “singh” and “sinha”

```
>>> for line in file1:
        if re.search('sin??',line):
            print(line)
```

ii. To print lines having words “agarwal” and “agrawal”

```
>>>for line in file1:
        if re.search('Ag..wal',line):  #here . matches any character except a new line
            print(line)
```

iii. To print the lines ending with “manager”

```
>>> for line in file1:
        if re.search('manager$',line):
            print(line)
```

iv. To print the lines starting with ‘P’

```
>>>for line in file1:
        if re.search('^P.*',line):  # here . any character and * for 0 or more repeat
            print(line)
```

b. Explain use of \S and [ ] in regular expression. [04]

Ans:

\S represents a non-whitespace character.

Example to find a sub-string look like email address(does not contain white space)

```
>>> s = 'A message from csev@umich.edu to cwen@iupui.edu about meeting @2PM'
>>> lst=re.findall('\S+@\S+',s)
>>> lst
['csev@umich.edu', 'cwen@iupui.edu']
```

[ ] are used to indicate a set of multiple acceptable characters for matching

Example: **for substrings that start with a single lowercase letter, uppercase letter, or number "[a-zA-Z0-9]", followed by zero or more non-blank characters ("\S*")**

Searching for marks 60 to 99 i.e. first digit is 6 to 9 and 2nd digit is 0 to 9

```
>>> for line in file1:
        if re.search('[6-9][0-9]',line):
            print(line,end='')
```

8.

a. Define a class employee in python using constructor to initialize empid, name and salary data of instances. Write methods to display employee details and to count number of employees. [05]

Ans:

```
class Employee:
  'Common base class for all employees'
  empCount = 0
  def __init__(self, name, salary):     #class constructor for initialization
    self.name = name
    self.salary = salary
    Employee.empCount += 1

  def displayCount(self):
   print ("Total Employee %d" % Employee.empCount )

  def displayEmployee(self):
    print( "Name : ", self.name,  ", Salary: ", self.salary )

emp1 = Employee("Dr. P. N. Singh", 150000) #created first objects with its data
emp2 = Employee("Dr. A. K. Nayak", 160000) #created first objects with its data
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount )

#executing the program output will be
Name :  Dr. P. N. Singh , Salary:  150000
Name :  Dr. A. K. Nayak , Salary:  160000
Total Employee 2
```

b. Define attribute, constructor, destructor, namespace & scope for object oriented programming in python. [05]

Ans:

- **Attribute:** A variable that is part of a class.

- **Constructor:** An optional specially named method __init__ () that is called at the moment when a class is being used to construct an object. Usually this is used to set up initial values for the object.

- **Destructor:** An optional specially named method __del__() that is called at the moment just before an object is destroyed. Destructors are rarely used.

- **namespace:** A namespace is a mapping from names to objects. Namespaces are created at different moments and have different lifetimes. The namespace containing the built-in names is created when the Python interpreter starts up, and is never deleted. The global namespace for a module is created when the module definition is read in; normally, module namespaces also last until the interpreter quits.

- **Scope:** A scope is a textual region of a Python program where a namespace is directly accessible. **"Directly accessible"** here means that an unqualified reference to a name attempts to find the name in the namespace.