

Internal Assessment Test –II, November 2020

Sub:	Programming Using Python						Code:	18MCA32		
Date:	3-11-2020	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MCA	
Answer Any 5 QUESTIONS								Marks	OBE	
									CO	RBT
1a	Consider the following list:li = [1,7,9,12,16]. Give outputs of the following commands: i) li[0:3] ii) li[0:-1] (iii) li[::-1] iv) li[-1:-4] v) li[:] vi) li[:4]						06	CO3	L3	
Ans	i) [1,7,9] ii) [1,7,9,12] iii) [16,12,9,7,1] iv) [] v) [1,7,9,12,16] vi)) [1,7,9,12]									
b	Write a Python program to check if a given string is a palindrome.						04	CO3	L2	
Ans	<pre>s=input('Please enter the string to be checked') rev=s[::-1] if s==rev: print 'palindrome' else: print 'not palindrome'</pre>									
2	Consider the list qrty = {5,4,7,3,6,2,1} and white the Python code to perform the following operation without using built in methods: i) Insert an element 9 at the beginning of the list ii) Insert an element 8 at the end of the list iii) Insert an element 8 at the index position 3 of the list iv) Delete an element at the beginning of the list v) Delete an element at the end of the list vi) Delete an element at index position 3 vii) Print all the elements in reverse order viii) Delete all elements of the list						10	CO3	L4	
Ans	<pre>qrty=[5,4,7,3,6,2,1] qrty=[9]+qrty print 'After inserting 9 '+str(qrty) qrty=qrty+[8] print 'After inserting 8 at end '+str(qrty) qrty= qrty[0:2]+[3]+qrty[2:] print 'After inserting 8 at index 3 '+str(qrty) qrty=qrty[1:] print 'After deleting from beginning '+str(qrty) qrty= qrty[:-1] print 'After deleting from end '+str(qrty) qrty=qrty[:2]+qrty[3:] print 'After deleting at index 3 '+str(qrty) qrty=qrty[::-1] print 'Elements in reverse order '+str(qrty) qrty=qrty[:] print 'After deleting all elements '+str(qrty)</pre>									

3	Discuss about the four techniques for reading files in detail.	10	CO4	L2, L3
Ans	<p><u>Methods of Reading from file</u></p> <p>a) read - format - read(<<number of characters>>) Example - fp = open('file1.txt','r') fp.read(10)</p> <p>This command specifies that 10 characters from the file - file1.txt needs to be read</p> <p>b) readline - for reading one line from the file i.e. till the next newline</p> <p>Example - fp = open('file1.txt','r') line=fp.readline() print line</p> <p>It returns the line as a string</p> <p>c) readlines() - Reads all the lines of the file. Returns a list of strings consisting of individual lines in the file Example - fp = open('file1.txt','r') lines = fp.readlines() for l in lines: print l fp.close()</p> <p>d) For line in file - A straightforward method of processing lines in a file is using loop Example fp = open('file1.txt','r') for line in fp: print l fp.close()</p>			
4	<p>What is dictionary? What is the difference between a set and dictionary? What is the result of the following python code fragment: A=([1,4],2,3) A[0][1]=5 A[2]=[4,5]</p>	10	CO3	L3
Ans	<p>A dictionary is an associative array. Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any Python data type. So dictionaries are unordered key-value-pairs. An empty dictionary can be created as d=dic() or d={ } d = { 'cat':2,'dog':4,'chicken':2}</p> <p>Here 'cat','dog' and 'chicken' are keys and the values 2,4,2 are values of the dictionaries. To know the count of chicken we can simply index it .e.g. print d['chicken']</p> <p>Several methods are defined for dictionaries For instance d.keys() returns a list of all keys d.values() returns a list of all values</p>			

	<p><u>Difference between set and dictionary</u></p> <p>Sets are just an unordered collection of elements and do not have key value associated with each other. They have methods such as union, intersection etc defined whereas dictionary have methods like keys(), values() etc. Sets cannot be indexed whereas dictionaries can using the keys. Both of them are mutable and iterable</p> <p>After the first statement A==([1,4],2,3) After the second statement A[0][1]=5 -- A=([1,5],2,3) The third statement A[2]=[4,5] will give an error since elements of a tuple cannot be modified since its immutable</p>			
--	--	--	--	--

5	<p>Explain each method of set given below. Write the Python code to take two sets as inputs from user and perform following operations on them. i) Union ii) Difference iii) Symmetric difference iv) Intersection v) issubset</p>	10	CO3	L3
---	--	----	-----	----

Ans	<pre> difference Creates a set with elements set([0, 2, 4]) lows.difference(odds) from one set, but not the other intersection Creates a set with elements lows.intersection(odds) set([1, 3]) that are in both sets issubset Asks are all of one set's lows.issubset(ren) elements True contained in another? symmetric_difference Creates a set with elements lows.symmetric_difference(odds) set([0, 2, 4, 5, 7, 9]) that are in exactly one set union Creates a set with elements lows.union(odds) set([0, 1, 2, 3, 4, 5, 7, 9]) that are in either set </pre> <p>myset1=input('Enter the first set as a comma separated list of elements') myset1=set(myset1.split(',')) myset2=input('Enter the second set as a comma separated list of elements') myset2=set(myset2.split(','))</p> <p>myset3=myset1 myset3=myset3.union(myset2) print 'After union :: '+str(myset3) myset3=myset1 myset3=myset3.intersection(myset2) print 'After intersection :: '+str(myset3) myset3=myset1 myset3=myset3.symmetric_difference(myset2) print 'After symm difference:: '+str(myset3) myset3=myset1 myset3=myset3.difference(myset2) print 'After difference:: '+str(myset3) print(myset2.issubset(myset1))</p> <p>Output Enter the first set as a comma separated list of elements'1,2,3,5,6' Enter the second set as a comma separated list of elements'3,4,7,8' After union :: set(['1', '3', '2', '5', '4', '7', '6', '8']) After intersection :: set(['3']) After symm difference:: set(['1', '2', '5', '4', '7', '6', '8']) After difference:: set(['1', '2', '5', '6'])</p>			
-----	---	--	--	--

	False			
6 a	Describe the different ways of opening a file explain with clear example.	06	CO4	L2
Ans	<p><u>Opening a file</u></p> <p>Method 1: using the open function - the function takes two arguments - the file name and the mode to open the file. Example fp=open('abc.txt','r')</p> <p>The above statement opens a file called abc.txt in read mode and returns . Files can be opened in three modes 'r' - read mode - for reading from the file. The file cursor is positioned at the beginning of the file. The file is expected to exist. An error is given if the file does not exist 'w' -write mode - for writing into file. If the file does not exist then a new file with the name is created. If the file exists then the contents are erased and cursor is placed at the beginning of the file. 'a' -append mode - for appending into file. The cursor is positioned at the end of file if the file exists and a write operation writes to the end of file. If the file does not exist then a new file with the name is created.</p> <p>After the file operations the file needs to be closed by using fp.close()</p> <p>Method 2: Using with .. as ...</p> <pre>using open(<<filename>>,<<mode>>) as <<file_handle>>: <<body>></pre> <p>Example: using open('file1.txt','r') as fp: for line in fp: print line</p> <p>The advantage of this method is that on exit of the with block the file automatically is closed and the programmer need not close it explicitly.</p>			
b	What are the different operations that can be done on files?	04	CO4	L1
Ans	<p>i) Opening a file using function open(Format explained in Q6(a)) ii) closing a file - Format <<file handle>>.close() - Must be done when the use of file is over iii) read ,readline,readlines - Detailed in Q6(a) iv) seek - used to move the file cursor to different locations Format - seek(<<number of steps>>,<<relative to>>) seek(...) seek(offset[, whence]) -> None. Move to new file position.</p> <p>Argument offset is a byte count. Optional argument whence defaults to 0 (offset from start of file, offset should be >= 0); other values are 1 (move relative to current position, positive or negative), and 2 (move relative to end of file, usually negative, although many platforms allow seeking beyond the end of a file). If the file is opened in text mode, only offsets returned by tell() are legal. Use of other offsets causes undefined behavior. Note that not all file objects are seekable.</p>			

	v) tell: tell(...) tell() -> current file position, an integer (may be a long integer).							
7	Explain In operator and range function with example. Write a program to read string in lower case and count the occurrence of each alphabet with the help of dictionary.	10	CO3	L1,L3				
Ans	<p>in and not in are the membership operators; used to test whether a value or variable is in a sequence.</p> <p>in True if value is found in the sequence</p> <p>not in True if value is not found in the sequence</p> <pre>x = 'Geeks for Geeks' y = {3:'a',4:'b'}</pre> <pre>print('G' in x) print('geeks' not in x) print('Geeks' not in x) print(3 in y) print('b' in y) Output: True True False True False</pre> <p>The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.</p> <p>Syntax</p> <pre>range(start, stop, step)</pre> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>start</i></td> <td>Optional. An integer number specifying at which position to start. Default is 0</td> </tr> </tbody> </table>	Parameter	Description	<i>start</i>	Optional. An integer number specifying at which position to start. Default is 0			
Parameter	Description							
<i>start</i>	Optional. An integer number specifying at which position to start. Default is 0							

stop

Required. An integer number specifying at which position to end.

step

Optional. An integer number specifying the incrementation. Default is 1

Example:

```
x = range(3, 6)
```

```
for n in x:
```

```
    print(n)
```

O/P: 3

4

5

```
all_freq = {}
```

```
test_str=input().lower()
```

```
for i in test_str:
```

```
    if i in all_freq:
```

```
        all_freq[i] += 1
```

```
    else:
```

```
        all_freq[i] = 1
```

```
print ("Count of all characters in string is :\n "
```

```
        + str(all_freq))
```

8

Compare the storage collection types string, list, tuple, set, dictionary.

10

CO3

L2

Ans

There are quite a few data structures available. The builtins data structures are: lists, tuples, dictionaries, strings, sets and frozensets.

Lists, strings and tuples are ordered sequences of objects. Unlike strings that contain only characters, list and tuples can contain any type of objects. Lists and tuples are like arrays. Tuples like strings are immutable. Lists are mutable so they can be extended or reduced at will. Sets are mutable unordered sequence of unique elements whereas frozensets are immutable sets.

Lists are enclosed in brackets:

```
l = [1, 2, "a"]
```

Tuples are enclosed in parentheses:

```
t = (1, 2, "a")
```

Tuples are faster and consume less memory. See [Tuples](#) for more information.

Dictionaries are built with curly brackets:

```
d = {"a":1, "b":2}
```

Sets are made using the `set()` builtin function.

collection	Mutable?	Ordered?	Use when
Str	yes	Yes	You want to keep track of text
List	Yes	Yes	You want to keep track of an unordered sequence you want to update

	Tuple	No	Yes	You want to build an ordered sequence that you want to use as a key in a dictionary or as a value in a set.		
	Set	Yes	No	You want to keep track of values, but order doesn't matter, and you don't want to keep duplicates. The values must be immutable.		
	dictionary	Yes	No	You want to keep a mapping of keys to values. The keys must be immutable.		