USN | | | | | | | | | | |

Internal Assessment Test 2 –April 2019 (ANSWER KEY)

| Sub: | Python Application Programming | | | | | Sub Code: | 15CS664 | Branch : | CSE |
|---|---|---|---|---|---|---|---|---|---|
| Date: | 20/04/2019 | Duration: | 90 min's | Max Marks: | 50 | Sem/Sec : | 6th A/B / C | | OBE |

Answer any FIVE FULL Questions

| | | MARKS | CO | RBT |
|---|---|---|---|---|
| 1 (a) | What are lists? Is List mutable? Justify your answer with an example.<br>a *list* is a sequence of values<br>in a list, values can be any type<br>values in list are called *elements* or sometimes *items*.<br>[10, 20, 30, 40]<br>['spam', 2.0, 5, [10, 20]]<br>lists are mutable because you can change the order of items in a list or reassign an item in a list.<br>>>> numbers = [17, 123]<br>>>> numbers[1] = 5<br>>>> print(numbers)<br>[17, 5] | [05] | CO3 | L2 |
| (b) | How tuples are created in Python? Explain different of accessing and creating them.<br>A tuple is a sequence of values much like a list.<br>**Create tuple** as a comma-separated list of values:<br>>>> t = 'a', 'b', 'c', 'd', 'e'<br>enclose tuples in parentheses to help us quickly identify tuples<br>>>> t = ('a', 'b', 'c', 'd', 'e')<br>To create a tuple with a single element<br>>>> t1 = ('a',)<br>To create an empty tuple<br>>>> t = tuple()<br>>>> print(t)<br>()<br>If the argument is a sequence (string, list, or tuple), the result of the call to tuple is a tuple with the elements of the sequence<br>>>> t = tuple('lupins')<br>>>> print(t)<br>('l', 'u', 'p', 'i', 'n', 's')<br><br>**Accessing tuples**<br>bracket operator indexes an element:<br>>>> t = ('a', 'b', 'c', 'd', 'e')<br>>>> print(t[0])<br>*'a'*<br>slice operator selects a range of elements<br>>>> print(t[1:3])<br>('b', 'c') | [05] | CO3 | L2 |
| 2 (a) | Implement a Python program using Lists to store and display the N integers accepted from the user. Write a function called swap to swap the first even number | [05] | CO3 | L3 |

of the list with the second odd number of the same list.

**Program**

```
N=list()
print("Enter integers. Type \"done\" to exit ")
while True:
    n = input(">")
    if n == 'done':
        break
    N.append(int(n))

print(N)

#index of first even number
index_f_even = None
for i in range(len(N)):
    if N[i]%2==0:
        index_f_even=i
        break

#index of second odd number
index_s_odd=None
count = 0
for i in range(len(N)):
    if N[i]%2!=0:
        index_s_odd=i
        count+=1
        if count==2:
            break;
print("Swap first even number ", N[index_f_even], "at index ",index_f_even,"with
")
print("second odd number",N[index_s_odd],"at index ", index_s_odd)

def swap(i1, i2, lst):
    tmp = lst[i1]
    lst[i1]=lst[i2]
    lst[i2]=tmp

swap(index_f_even, index_s_odd, N)
print("After Swapping")
print(N)
```

**Output:**

```
C: \py_prog>iat2.py
Enter integers. Type "done" to exit
>1
>2
>3
>4
>done
[1, 2, 3, 4]
Swap first even number  2 at index  1 with
second odd number 3 at index  2
After Swapping
[1, 3, 2, 4]
```

| | | | | |
|---|---|---|---|---|
| (b) | Explain Dictionaries. Demonstrate it with a Python program that finds the frequency of letters in a given string.<br>A *dictionary* is like a list. In a dictionary, the indices can be (almost) any type.<br>a dictionary as a mapping between a set of indices (which are called *keys*) and a set of values. Each key maps to a value. The association of a key and a value is called a *key-value pair* or sometimes an *item*.<br>**Program**<br>word = 'brontosaurus'<br>d = dict()<br>**for** c in word:<br>**if** c not in d:<br>d[c] = 1<br>**else**:<br>d[c] = d[c] + 1<br>print(d) | [05] | CO3 | L2 |
| 3 (a) | Write a Python program to search and print for lines in a file that starts with the word 'From' and a character followed by a two digit number between 00 and 99 and then followed by ':'. Assume any input file.<br>import re<br>hand = open(sample.txt')<br>**for** line in hand:<br>line = line.rstrip()<br>**if** re.search('^From.[0-9][0-9]::', line):<br>print(line) | [06] | CO2,CO3 | L3 |
| (b) | Discuss *search()* and *findall()* functions of *re* module.<br>search() uses the regular expression. It matches the regular expression.<br>If we want to extract data from a string in Python we can use the findall() method to extract all of the substrings which match a regular expression.<br>import re<br>s = 'A message from csev@umich.edu to cwen@iupui.edu about meeting @2PM'<br>lst = re.findall('\S+@\S+', s)<br>print(lst)<br>Output:<br>['csev@umich.edu', 'cwen@iupui.edu']<br>when you are using findall(), parentheses indicate that while you want the whole expression to match, you only are interested in extracting a portion of the substring that matches the regular expression.<br>import re<br>hand = open('mbox-short.txt')<br>**for** line in hand:<br>line = line.rstrip()<br>x = re.findall('^X\S*: ([0-9.]+)', line)<br>**if** len(x) > 0:<br>print(x)<br>Output:<br>['0.8475']<br>['0.0000']<br>['0.6178']<br>['0.0000']<br>['0.6961']<br>['0.0000']<br>.. | [04] | CO3 | L2 |

| | | | | | |
|---|---|---|---|---|---|
| 4 | (a) | Write a Python program to read all the lines from a file accepted from the user and print all the email addresses contained in it. Assume the email addresses contain only non-white space characters.<br>**Program:**<br>try:<br>file = input("Enter a file name: ")<br>fhand=open(file)<br><br>except:<br>print("Invalid File")<br>exit()<br>import re<br>for line in fhand:<br>x = re.findall('\S+@\S+', line)<br>if len(x) > 0:<br>print(x)<br>Output:<br>['apache@localhost)']<br>['source@collab.sakaiproject.org;']<br>['zqian@umich.edu']<br>['source@collab.sakaiproject.org']<br>['zqian@umich.edu']<br>['zqian@umich.edu']<br>['aaronz@vt.edu']<br>… | [06] | CO2,CO3 | L3 |
| | (b) | Write the expected output:<br>1. >>> a = [1, 2, 3]<br>>>> b = [4, 5, 6]<br>>>> print( a + b )<br>[1,2,3,4,5,6]<br>2. >>>[1,2,3]*3<br>[1,2,3,1,2,3,1,2,3]<br><br>3. >>>t=[‘a’,’b’,’c’,’d’,’e’,’f’]<br>>>> t[:4]<br>[‘a’,’b’,’c’,’d’]<br><br>4. >>> t1=(1,2,3)<br>>>> print(t1.append(4))<br>AttributeError: 'tuple' object has no attribute 'append' OR<br>Results in error. Append can be used on lists. Tuples are immutable | [04] | CO3 | L2 |
| 5 | (a) | Create a student class and initialize it with name and roll number.<br>Design methods to:<br>1. Display () – to display all information of the student.<br>2.setAge () – to assign age to student.<br>3.setMarks () – to assign marks to student.<br>**Program**<br>class Student:<br>def __init__(self, rollno, name):<br>self.rollno = rollno<br>self.name = name | [06] | CO4 | L3 |

```
        self.age = None
        self.marks = None

    def display(self):
        print("Roll Number:", self.rollno," Name: ", self.name)
        print("Age:", self.age," Marks: ", self.marks)
    def setAge(self, a):
        self.age = a
    def setMarks(self, m):
        self.marks = m


s=Student('G-34', 'Jake')

s.setAge(18)
s.setMarks(52)

s.display()
```

Sample Output:
C: \py_prog>student_class.py
Roll Number: G-34  Name:  Jake
Age: 18  Marks:  52

| | | | | |
|---|---|---|---|---|
| (b) | What is the difference between the 2 ways of converting a string to a list? Explain using the following string.<br>>>>s = "Good Morning"<br>To convert from a string to a **list of characters**, you can use list:<br>>>> t = list(s)<br>>>> print(t)<br>['G', 'o', 'o', 'd', ' ', 'M', 'o', 'r', 'n', 'i', 'n', 'g']<br>The list function breaks a string into individual letters. If you want **to break a string into words**, you can use the split method:<br>>>> t = s.split()<br>>>> print(t)<br>['Good', 'Morning'] | [04] | CO3 | L2 |
| 6 (a) | Suppose "romeo.txt" consists of the text from *Romeo and Juliet* Act 2, Scene 2. Read the file line by line, print the ten most common words in the text. Make sure to remove punctuations and convert case of word. Use dictionaries, tuples and lists.<br>**Program** | [10] | CO2,CO3 | L3 |

```
import string
fhand=open("romeo.txt")

counts=dict()
for line in fhand:
    line = line.translate(line.maketrans(' ',' ',string.punctuation))
    line = line.lower()
    words = line.split()
    for word in words:
        counts[word]=counts.get(word,0)+1

l=list()
```

```
for key, value in counts.items():
    l.append((value,key))
    #print(key,":",value)

l.sort(reverse=True)

for key, val in l[:10]:
    print(key,":",val)
```

Output:

```
C:\ py_prog>countCommon.py
3 : the
3 : is
3 : and
2 : sun
1 : yonder
1 : with
1 : window
1 : who
1 : what
1 : through
```

7 (a) Write a short note on: 1. Parsing lines& 2. Object Aliasing [06] CO3 L2

Usually when we are reading a file we want to do something to the lines other than just printing the whole line. Often we want to find the "interesting lines" and then *parse* the line to find some interesting *part* of the line.

*Give any example program that reads a file, parses lines and prints only the certain strings of interest in that line.*

```
#Print day of week
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From '): continue
    words = line.split()
print(words[2])
```

**Output:**

```
Sat
Fri
Fri
Fri
...
```

2. Object Aliasing

If a refers to an object and you assign b = a, then both variables refer to the same object:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

The association of a variable with an object is called a *reference*. In this example, there are two references to the same object.

An object with more than one reference has more than one name, so we say that the object is *aliased*.
If the aliased object is mutable, changes made with one alias affect the other:

```
>>> b[0] = 17
>>> print(a)
[17, 2, 3]
```

(b) Explain DSU pattern with respect to tuples. Give example. [04] CO3 L2

This feature lends itself to a pattern called *DSU* for
**Decorate** a sequence by building a list of tuples with one or more sort keys preceding the elements from the sequence,
**Sort** the list of tuples using the Python built-in sort, and
**Undecorate** by extracting the sorted elements of the sequence.

**Code Snippet**

```
txt = 'but soft what light in yonder window breaks'
words = txt.split()
t = list()
for word in words:
      t.append((len(word), word))  #Decorate
t.sort(reverse=True) #Sort
res = list()
for length, word in t:
    res.append(word) #Undecorate
print(res)
```

8 (a) Discuss any 5 meta characters used in regular expressions with suitable example. [05] CO3 L1

1. the caret character is used in regular expressions to match “the beginning” of a line.

```
# Search for lines that start with 'From'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line):
        print(line)
```

2. The period or full stop, which matches any character.

```
# Search for lines that start with 'F', followed by
# 2 characters, followed by 'm:'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^F..m:', line):
        print(line)
```

3. Indicate that a character can be repeated any number of times using the * or + characters in your regular expression. These special characters mean that instead of matching a single character in the search string, they match zero-or-more characters (in the case of the asterisk) or one-or-more of the characters (in the case of the plus sign).

```
# Search for lines that start with From and have an at sign
import re
```

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From.+@', line):
        print(line)
```

4. matches a non-whitespace character (\S).

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

Output:

```
['csev@umich.edu', 'cwen@iupui.edu']
```

5. Square brackets are used to indicate a set of multiple acceptable characters we are willing to consider matching.

```
# Search for lines that have an at sign between characters
# The characters must be a letter or number
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('[a-zA-Z0-9]\S+@\S+[a-zA-Z]', line)
    if len(x) > 0:
        print(x)
```

**Any of the following characters with code snippet or regular expression.**

ˆ Matches the beginning of the line.
$ Matches the end of the line.
. Matches any character (a wildcard).
\s Matches a whitespace character.
\S Matches a non-whitespace character (opposite of \s).
* Applies to the immediately preceding character(s) and indicates to match zero or more times.
*? Applies to the immediately preceding character(s) and indicates to match zero or more times in “non-greedy mode”.
+ Applies to the immediately preceding character(s) and indicates to match one or more times.
+? Applies to the immediately preceding character(s) and indicates to match one or more times in “non-greedy mode”.
? Applies to the immediately preceding character(s) and indicates to match zero or one time.
?? Applies to the immediately preceding character(s) and indicates to match zero or one time in “non-greedy mode”.
[aeiou] Matches a single character as long as that character is in the specified set. In this example, it would match “a”, “e”, “i”, “o”, or “u”, but no other characters.
[a-z0-9] You can specify ranges of characters using the minus sign. This example is a single character that must be a lowercase letter or a digit.
[ˆA-Za-z] When the first character in the set notation is a caret, it inverts the logic. This example matches a single character that is anything *other than* an uppercase or lowercase letter.
( ) When parentheses are added to a regular expression, they are ignored for the

purpose of matching, but allow you to extract a particular subset of the matched string rather than the whole string when using findall().
\b Matches the empty string, but only at the start or end of a word.
\B Matches the empty string, but not at the start or end of a word.
\d Matches any decimal digit; equivalent to the set [0-9].
\D Matches any non-digit character; equivalent to the set [ˆ0-9].

(b) What is the need of escape characters in regular expressions? Give suitable code snippet. [05] CO3 L2

We can indicate that we want to simply match a character by prefixing that characterwith a backslash. For example, we can find money amounts with the following regular expression.

**Code Snippet**

```
import re
x = 'We just received $10.00 for cookies.'
y = re.findall('\$[0-9.]+',x)
```