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

Internal Assessment Test II(set2) – Dec 2022

Sub:	Application Development Using Python-Scheme and Solution				Sub Code:	18CS55	Branch:	ISE
Date:	03/12/2022	Duration:	90 mins	Max Marks: 50	Sem/Sec:	V A,B&C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1a)	Explain the use of following string related methods with examples code snippets i) center(), rjust() and ljust() methods ii) join() and split()	6	CO2	L2
Ans		3+3		

1. center(), rjust() and ljust() methods (1*3M)

The center() string method works like ljust() and rjust() but centers the text rather than justifying it to the left or right.

Ex:

```
>>> 'Hello'.center(20)
' Hello '
>>> 'Hello'.center(20, '=')
'=======Hello========'
```

The rjust() and ljust()

The rjust() and ljust() string methods return a padded version of the string they are called on, with spaces inserted to justify the text. The first argument to both methods is an integer length for the justified string.

Ex:

```
>>> 'Hello'.rjust(20)
' Hello'
>>> 'Hello World'.rjust(20)

' Hello World'

>>> 'Hello'.ljust(10) 'Hello '
```

2. join() and split()(1.5*2M)

The join() method is useful when you have a list of strings that need to be joined together into a single string value. The join() method is called on a string, gets passed a list of strings, and returns a string. The returned string is the concatenation of

	each string in the passed-in list Ex: >>> **', '.join(['cats', 'rats', 'bats'])** 'cats, rats, bats' >>> **' '.join(['My', 'name', 'is', 'Simon'])** 'My name is Simon' The split() method does the opposite of Join It's called on a string value and returns a list of strings. EX: >>> **'My name is Simon'.split()** ['My', 'name', 'is', 'Simon'] >>> **'MyABCnameABCisABCSimon'.split('ABC')** ['My', 'name', 'is', 'Simon'] >>> **'My name is Simon'.split('m')** ['My na', 'e is Si', 'on']			
1b)	Write short notes on the following modules. Give examples for their usage. i) re ii) pyperclip (2*2Marks)	4	CO2	L2
Ans	**re Module:** re stands for regular expression This module provides regular expression for matching operations Passing a string value representing your regular expression to re.compile() returns a Regex pattern object (or simply, a Regex object). EX: To create a Regex object that matches the phone number pattern >>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') We should always make sure to import re module whenever we are writing any regular expression *Otherwise,will get a NameError: name 're' is not defined error message.* ***Pyperclip Module:*** The pyperclip module has copy() and paste() functions that can send text to and receive text from your computer's clipboard. Sending the output of your program to the clipboard will make it easy to paste it to an email, word processor, or some other software. Pyperclip does not come with Python we need to install it from third party module. EX: >>> import pyperclip			

	>>> pyperclip.copy('Hello world!') >>> **pyperclip.paste()** 'Hello world!			
2a)	Describe the following with suitable Python code snippets. (i) Greedy and Non Greedy Pattern Matching (ii) findall() method of Regex object.(1*4+1*2 Marks)	6	CO3	L3
	(i) Greedy and Non Greedy Pattern Matching Python's regular expressions are *greedy* by default, which means that in ambiguous situations they will match the longest string possible. The *non- greedy* version of the curly brackets, which matches the shortest string pos- sible, has the closing curly bracket followed by a question mark. >>> greedyHaRegex = re.compile(r'(Ha){3,5}') >>> mo1 = greedyHaRegex.search('HaHaHaHaHa') >>> mo1.group() 'HaHaHaHaHa' >>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?') >>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa') >>> mo2.group() 'HaHaHa' the question mark can have two meanings in regular expres- sions: declaring a nongreedy match or flagging an optional group. **II) findall() method** findall() method will return the strings of *every* match in the searched string >>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') >>> mo = phoneNumRegex.search('Cell: 415-555-9999 Work: 212-555-0000') >>> mo.group() '415-555-9999' findall() will not return a Match object but a list of strings—*as long as there are no groups in the regular expression*. Each string in the list is a piece of the searched text that matched the regular expression. >>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # has no groups			

>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000') ['415-555-9999', '212-555-0000']

If there *are* groups in the regular expression, then findall() will return a list of tuples. Each tuple represents a found match, and its items are the matched strings for each group in the regex
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # has groups >>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]

Below points should be in answer:

1. When called on a regex with no groups, such as \d\d\d-\d\d\d-\d\d\d\d, the method findall() returns a list of string matches, such as ['415-555- 9999', '212-555-0000'].
2. When called on a regex that has groups, such as (\d\d\d)-(\d\d\d)-(\d\ d\d\d), the method findall() returns a list of tuples of strings (one string for each group), such as [('415', '555', '1122'), ('212', '555', '0000')].

2b) Differentiate List, Set, and Tuple with examples. (1*4 points in each carries 4 marks) — 4 — CO2 — L2

List	**set**	**Tuple**
Lists is Mutable	Set is Mutable	Tuple is Immutable
It is Ordered collection of items	It is Unordered collection of items	It is Ordered collection of items
Items in list can be replaced or changed	Items in set cannot be changed or replaced	Items in tuple cannot be changed or replaced
1. Ex: L=[10,20.”a”,40]	Ex: s={1,2,3,4}	Ex: t=(1,2,3,4,4)

3a) Write a python program to store 5 players’ data in Nested Dictionary. Have players names as the keys and dictionary which contains score 1, score 2 and score 3 in 3 matches as values. Write a function displayaverage() that would take match name either ‘Match 1’, or ‘Match 2’ or ‘Match 3’ as argument and display the average scores of for the Match name passed**.(4 Marks program+ 1marks o/p)** — 5 — CO2 — L3

```
Players = {'A': {'match1': 5, 'match2': 12,'match3':20},
          'B': {'match1': 5, 'match2': 13,'match3':30},
          'C': {'match1': 5, 'match2': 15,'match3':40},
          'D':{'match1': 5, 'match2': 12,'match3':50},
           'E':{'match1': 5, 'match2': 25,'match3':60}}

def displayaverage(players, item):
    numBrought = 0
    for k, v in players.items():
        numBrought = numBrought + v.get(item, 0)
    average=numBrought/5
    return average
print('average:')
print('Match1 average  : '+str(displayaverage(Players,'match1')))
print('Match2 average  : '+str(displayaverage(Players,'match2')))
print('Match3 average  : '+str(displayaverage(Players,'match3')))
```

Output:

```
average:
Match1 average  : 5.0
Match2 average  : 15.4
Match3 average  : 40.0
```

3b)	Explain the use of get() and setdefault() methods related to dictionary with suitable code snippet. (2.5*2 Marks)	5	CO2	L2

The get() Method

dictionaries have a get() method that takes two arguments: the key of the value to retrieve and a fallback value to return if that key does not exist.

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.' 'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.' 'I am bringing 0 eggs.'
```

Because there is no 'eggs' key in the picnicItems dictionary, the default value 0 is returned by the get() method. Without using get(), the code would have caused an error message,

The setdefault() Method

The first argument passed to the method is the key to check for, and

	the second argument is the value to set at that key if the key does not exist. If the key does exist, the setdefault() method returns the key's value. Ex: >>> spam = {'name': 'Pooka', 'age': 5} >>> spam.setdefault('color', 'black') 'black' >>> spam {'color': 'black', 'age': 5, 'name': 'Pooka'} >>> spam.setdefault('color', 'white') 'black' >>> spam {'color': 'black', 'age': 5, 'name': 'Pooka'}			
4a)	Write a python program to read n number of email IDs and extract the domain names from the email IDs using regular expression. (4 marks program+2 marks o/p)	6	CO3	L3
	`` import re n=int(input('number of email')) en=[] for i in range(0,n): email=input('enter email id') en.append(email) #print(en) for i in en: emails = re.findall(""@(\w+\.+\w+)"", i) print(emails) ``` output ``` number of email 1 enter email id ash.d@cmr.com ['cmr.com'] ``` Or ``` import re n=int(input('number of email')) en=[] for i in range(0,n): email=input('enter email id') en.append(email) pat = re.compile(r"(\w+)@((\w+)+.(\w{2,3}?).(\w{2,3})?)") for i in en: r2 = pat.search(i) print(r2.group(1)+" "+r2.group(3)) ``` Execution: ``` number of email 2 enter email id abc@yahoo.com enter email id xyz@gmail.com Input: ```			

	abc yahoo xyz gmail			
4b)	Describe the usage of following special symbols using examples.(1*4 marks) i) { } ii) * iii) ? iv) +	4	CO3	L2
	{} -Matching Specific Repetitions with Curly Brackets If you have a group that you want to repeat a specific number of times, fol- low the group in your regex with a number in curly brackets. For example, the regex (Ha){3} will match the string 'HaHaHa', but it will not match 'HaHa', You can also leave out the first or second number in the curly brackets to leave the minimum or maximum unbounded. For example, (Ha){3,} will match three or more instances of the (Ha) group, while (Ha){,5} will match zero to five instances. Curly brackets can help make your regular expres- sions shorter. (Ha){3,5} ((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha)(Ha)) ***Matching Zero or More with the Star** The * (called the star or asterisk) means "match zero or more"—the group that precedes the star can occur any number of times in the text. It can be completely absent or repeated over and over again >>> batRegex = re.compile(r'Bat(wo)*man') >>> mo1 = batRegex.search('The Adventures of Batman') >>> mo1.group() 'Batman' **+ Matching One or More with the Plus** While * means "match zero or more," the + (or plus) means "match one or more." Unlike the star, which does not require its group to appear in the matched string, the group preceding a plus must appear at least once. It is not optional. >>> batRegex = re.compile(r'Bat(wo)+man') >>> mo1 = batRegex.search('The Adventures of Batwoman') >>> mo1.group() 'Batwoman' **? Optional Matching with the Question Mark** Sometimes there is a pattern that you want to match only optionally. That is, the regex should find a match whether or not that bit of text is there. The ? character flags the group that precedes it as an optional			

	part of the pattern >>> batRegex = re.compile(r'Bat(wo)?man') >>> mo1 = batRegex.search('The Adventures of Batman') >>> mo1.group() 'Batman'			
5a)	Differentiate the usage of group() and groups() methods with example code. (2*2 marks)	4	CO3	L2
	group() Say you want to separate the area code from the rest of the phone number. Adding parentheses will create *groups* in the regex: (\d\d\d)-(\d\d\d-\d\d\d\d). Then you can use the group() match object method to grab the matching text from just one group. >>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)') >>> mo = phoneNumRegex.search('My number is 415-555-4242.') >>> mo.group(1) '415' >>> mo.group(2) '555-4242' >>> mo.group(0) '415-555-4242' >>> mo.group() '415-555-4242' groups() If you would like to retrieve all the groups at once, use the groups() method—note the plural form for the name. >>> mo.groups() ('415', '555-4242') >>> areaCode, mainNumber = mo.groups() >>> print(areaCode) 415 >>> print(mainNumber) 555-4242 Since mo.groups() returns a tuple of multiple values, you can use the multiple-assignment trick to assign each value to a separate variable, as in the previous areaCode, mainNumber = mo.groups() line.			
5b)	Write a Python code to read the string as input and count the occurrence of each vowels.	6	CO2	L3

```
#program

def Check_Vow(string, vowels):

    # casefold has been used to ignore cases
    string = string.lower()

    # Forms a dictionary with key as a vowel
    # and the value as 0
    count = {}.fromkeys(vowels, 0)

    # To count the vowels
    for character in string:
        if character in count:
            count[character] += 1
    return count

# Driver Code
vowels = 'aeiou'

string = input("enter string")
print (Check_Vow(string, vowels))
```

output

```
enter string sonal
{'a': 1, 'e': 0, 'i': 0, 'o': 1, 'u': 0}
```

6a)	Illustrate with example how the copy.copy() is different from copy.deepcopy() which is relevant to lists or dictionaries in Python. (2.5*2.5Marks)	5	CO2	L2

copy.copy(), can be used to make a duplicate copy of a mutable value like a list or dictionary, not just a copy of a reference.

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> cheese = copy.copy(spam)
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

Now the spam and cheese variables refer to separate lists, which is why only the list in cheese is modified when you assign 42 at index 1. As you can see in Figure 4-7, the reference ID numbers are no longer the same for both vari- ables because the variables refer to independent

lists.

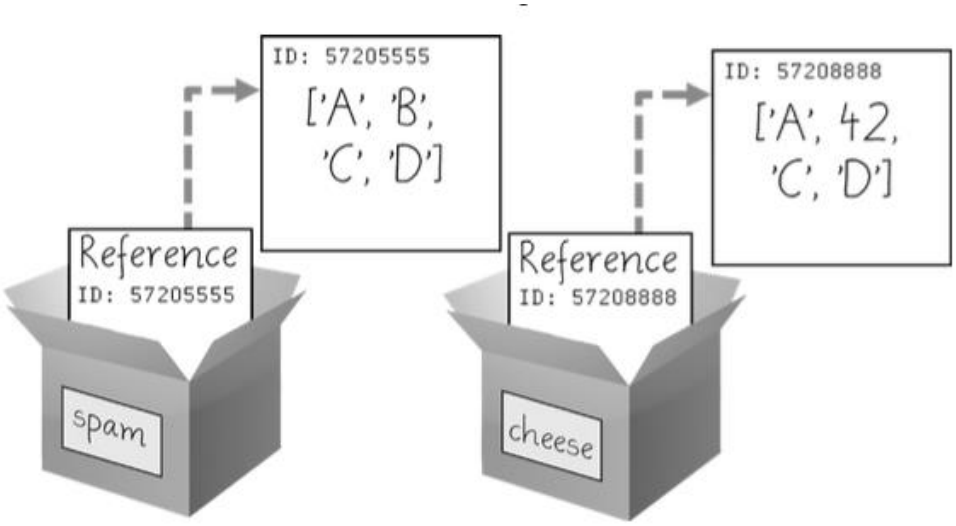

Figure 4-7: cheese = copy.copy(spam) creates a second list that can be modified independently of the first.

If the list you need to copy contains lists, then use the copy.deepcopy() The deepcopy() function will copy these inner lists as well.

#Shalow copying copy.copy()

```
import copy

list1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
list2 = copy.copy(list1)

list1.append([13, 14,15])

print("Old list:", list1)
print("New list:", list2)
print('\nID of Old List:', id(list1))
print('ID of New List:', id(list2))
```

Execution:

```
Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13,
14, 15]]
New list: [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

#Deep copying example:

```
import copy
x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
z = copy.deepcopy(x)

x[2][2] = 'Hello'
print(x)
print(z)
print('\nID of Old List:', id(x))
print('ID of New List:', id(z))
```

Execution:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 'Hello']]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

	ID of Old List: 2440781390016 ID of New List: 2440781129984			
(6b)	What is dictionary? Explain about the following methods related to dictionary with code snippets values() ii) keys() iii) items() (1+1.5*3 marks with example code for each method)	5	CO3	L2
	The keys(), values(), and items() Methods There are three dictionary methods that will return list-like values of the dictionary's keys, values, or both keys and values: keys(), values(), and items(). The values returned by these methods are not true lists: They cannot be modified and do not have an append() method. But these data types (dict_keys, dict_values, and dict_items, respectively) can be used in for loops **values()- returns values in dictionary** >>> spam = {'color': 'red', 'age': 42} >>> for v in spam.values(): print(v) red 42 **keys()- returns keys in dictionary** >>> for k in spam.keys(): print(k) color age **items() returns key values both** >>> for i in spam.items(): print(i) ('color', 'red') ('age', 42)			