Week-5, Graded

- Question-1 [2 Marks]
  - Statement
  - Options
    - (a)
    - (b)
    - (c)
    - (d)
  - Answer
  - Solution
- Question-2 [2 Marks]
  - Statement
  - Options
    - (a)
    - (b)
    - (c)
    - (d)
  - Answer
  - Solution
- Question 3 [3 Marks]
  - Statement
  - Answer
  - Solution
- Question (4 - 5) [7 Marks]
  - Statement
  - Explanation of procedure
- Question 4 [4 Marks]
  - Statement
  - Options
    - (a)
    - (b)
    - (c)
    - (d)
  - Answer
  - Solution
- Question 5 [3 Marks]
  - Statement
  - Options
    - (a)
    - (b)
    - (c)
    - (d)
  - Answer
  - Solution
- Question 6 [5 Marks]
  - Statement
  - Options
    - (a)
    - (b)
    - (c)
    - (d)
  - Answer
  - Solution
- Question 7 [5 Marks]

- Statement
  - Answer
- Solution

Question 8 [6 Marks]
- Statement
- Options
  - (a)
  - (b)
  - (c)
  - (d)
  - (e)
- Answer
- Solution

Question (9-10)
- Statement

Question 9 [4 Marks]
- Statement
- Options
  - (a)
  - (b)
  - (c)
  - (d)
- Answer
- Solution

Question 10 [5 Marks]
- Options
  - (a)
  - (b)
  - (c)
  - (d)
- Answer
- Solution

Question-1 [2 Marks]

Statement

What will be the value of **mList** at the end of the execution of the given pseudocode?

```
L = [[0, 210], [1, 198], [2, 188], [3, 173], [4, 240]]
mList = []
foreach element in L{
    mList = mList ++ [last(element)]
}
```

Options

(a)

[0, 210, 1, 198, 2, 188, 3, 173, 4, 240]

(b)

[0, 1, 2, 3, 4]

(c)

[[0, 210], [1, 198], [2, 188], [3, 173], [4, 240]]

(d)

[210, 198, 188, 173, 240]

Answer

(d)

Solution

Line 3, one element of **L** is picked up for the iteration. List **L** has 5 elements therefore, there will be 5 iterations. The below table shows the change in **mList** through out the iterations.

L = [[0, 210], [1, 198], [2, 188], [3, 173], [4, 240]]

Iteration count	Value of element	last(element)	mList
1	[0, 210]	210	[]++[210] = [210]
2	[1, 198]	198	[210]++[198] = [210, 198]
3	[2, 188]	188	[210, 198] ++ [188] = [210, 198, 188]
4	[3, 173]	173	[210, 198, 188, 173]
5	[4, 240]	240	[210, 198, 188, 173, 240]

Question-2 [2 Marks]

Statement

What will be the value of **mList** at the end of the execution of the given pseudocode?

```
L = [[0, 210, 78], [1, 198, 91], [2, 188, 77], [3, 173, 78], [4, 240, 89]]
mList = []
foreach element in L{
    mList = mList ++ [last(init(element))]
}
```

Options

(a)

[0, 210, 1, 198, 2, 188, 3, 173, 4, 240]

(b)

[0, 1, 2, 3, 4]

(c)

[[0, 210], [1, 198], [2, 188], [3, 173], [4, 240]]

(d)

[210, 198, 188, 173, 240]

Answer

(d)

Solution

The below table shows the change in **mList** through out the iterations.

L = [[0, 210, 78], [1, 198, 91], [2, 188, 77], [3, 173, 78], [4, 240, 89]]

Iteration count	Value of element	last(init(element))	mList
1	[0, 210, 78]	210 (middle element of triple)	[]++[210] = [210]
2	[1, 198, 91]	198	[210]++[198] = [210, 198]
3	[2, 188, 77]	188	[210, 198] ++ [188] = [210, 198, 188]
4	[3, 173, 78]	173	[210, 198, 188, 173]
5	[4, 240, 89]	240	[210, 198, 188, 173, 240]

Question 3 [3 Marks]

Statement

Let **N** be a list of first 50 positive integers (i.e., **N** = [1, 2, 3, ..., 49, 50]). What will be the value of **count** at the end of the execution of the given pseudocode? (NAT)

```
1  count = 0
2  A = someList(N)
3  B = someList(rest(N))
4  foreach Y in A{
5      foreach Z in B{
6          if(Z == Y){
7              count = count + 1
8          }
9      }
10 }
11 Procedure someList(X)
12     outlist = [], newList = X
13     while(length(newList) > 0){
14         outlist = outlist ++ [first(newList)]
15         newList = rest(rest(newList))
16     }
17     return(outlist)
18 End someList
```

Answer

0

Solution

This question has a nested iteration and a procedure. Let us first see what does the procedure do? In procedure **someList**(X), **outlist** was initialized to an empty list. Line 17 shows that the procedure returns **outlist** which is a list.

From line 13, **length**(**newList**) tells that **newList** is also a list. **newList** is initialized with **X**. This is usually done to avoid the side effects of a procedure on the list. Basically **newList** is nothing but **X**.

In the line 14, first element of new list is being appended to **outlist**. In line 15, two times **rest** function is used on **newList** which will result a list having all the element of **newList** except the first two elements of **newList**. In the next iteration **outlist** gets appended with the first element of updated **newList** which was the 3rd element of the original **newList** or third element of X.

Again **newList** gets updated. Again the first element of updated **newList** gets appended into **outlist** which was the fifth element of the original **newList** i.e., fifth element of the **X**. This means **someList** accepts a list and returns a list which contains the elements at odd positions that is the first, third, fifth and so on elements. Or we can say it returns a list with alternate elements starting from the first element.

Now **A** = **someList**(N), which means **A** is a list containing the alternate elements of list N, starting from the first element. Therefore, **A** = [1, 3, 5, ..., 49] i.e, the list of odd numbers from 1 to 49.

And **B** = **someList**(**rest**(N)), which means **B** is a list containing the alternate elements of list **rest**(N), starting from the first element. Since, **rest**(N) = [2, 3, 4, 5, ..., 50] therefore, **A** = [2, 4, 6, ..., 50] i.e, the list of even numbers from 0 to 50.

Now in the nested iteration (Line 4 - 10), every element of **A** is being picked up and it is being compared with every element of **B**. If the element in outer loop is same as the element in inner loop then **count** is being incremented. Therefore, **count** is the number of elements of **B** which are also the element of **A**. It is clear that there is no common elements in **A** and **B**. Therefore, **count** will always be 0.

Question (4 - 5) [7 Marks]

Statement

Consider the procedure given below. If **L1** and **L2** are two lists, and **L** = **eliminate(L1, L2)**, then answer the following questions.

```
1  Procedure eliminate(L1, L2)
2      L3 = [], Found = False
3      foreach i in L1{
4          foreach j in L2{
5              if(i == j){
6                  Found == True
7              }
8          }
9          if(not Found){
10             L3 = L3 ++ [i]
11         }
12         Found = False
13     }
14     return(L3)
15 End eliminate
```

Explanation of procedure

Let us first understand about what does the procedure do? It is clear that it accepts two lists and returns a list. In line 2, **L3** is initialized as an empty list which gives us a sense that something will get appended in **L3** during the execution. **Found** is a flag which is initialized to False.

Nested iterations are used from line 3 to line 13. Based on line 3 and 4 it is clear that for every element of **L1**, every element of **L2** will be checked. **i** represents the element of **L1** and **j** represents the element of **L2**. **Found** is set to be True when **i** == **j**. This means one element of list **L1** is picked up and being checked if there is at least one same element is present in **L2** or not.

Line 9, checks for not **Found** which means an element **i** from **L1** will be appended to **L3** if **i** is not present in **L1**. Flag **Found** is being reset to False for the next element of **L1**. And the same procedure will be repeated. Therefore, **L3** is a list of elements of **L1** which are not common with **L2**.

Question 4 [4 Marks]

Statement

Choose the correct options(s) regarding **L**. It is a Multiple Select Question (MSQ).

Options

(a)

It will contain all the elements of **L2** that are not present in **L1**

(b)

It will contain all the elements of **L1** that are not present in **L2**

(c)

It will contain all the elements common to **L1** and **L2**

(d)

It will contain the elements present in **L1** or **L2** but not both

Answer

(b)

Solution

L is just the value of **L3** so it represents a list of elements of **L1** which are not common with **L2**.

Question 5 [3 Marks]

Statement

Which of the following option(s) is/are always correct? It is a Multiple Select Question (MSQ).

Options

(a)

length(**L1**) - **length**(**L2**) = **Length**(**L**)

(b)

length(**L1**) > **length**(**L2**)

(c)

length(**L1**) >=**length**(**L**)

(d)

length(**L2**) <= **length**(**L**)

Answer

(c)

Solution

It is clear that elements of **L** are from list **L1** therefore, **L1** can have maximum number of elements equal to the number of elements in **L1** when there is no common element between **L1** and **L2**. Otherwise the number of elements in **L** will be less than the number of elements in **L1**. Therefore, **length**(**L1**) >=**length**(**L**).

There is no relationship mentioned anywhere between the number of elements in **L1** and **L2**.

As the elements in **L** does not belong to **L2** we can not comment about their relationship.

Question 6 [5 Marks]

Statement

A word is said to be perfect if no letter is repeated. Let **isPerfect** be a procedure that takes a row X in the "Words" table as input and returns True if the word in row X is a perfect word otherwise returns False. Choose the correct implementation of the procedure **isPerfect**.

Options

(a)

```
Procedure isPerfect(X)
    C = []
    i = 1
    while(i <= X.LetterCount){
        A = ith letter in X.Word
        if(member(C,A)){
            return(False)
        }
        else{
            return(True)
        }
        i = i + 1
    }
End isPerfect
```

(b)

```
Procedure isPerfect(X)
    C = []
    i = 1
    while(i <= X.LetterCount){
        A = ith letter in X.Word
        if(member(C,A)){
            return(False)
        }
        else{
            C = C ++ [A]
        }
        i = i + 1
    }
    return(True)
End isPerfect
```

(c)

```
Procedure isPerfect(X)
    C = []
    i = 1
    while(i <= X.LetterCount){
        A = ith letter in X.Word
        if(member(C,A)){
            C = C ++ [A]
```

```
        }
        else{
            return(False)
        }
        i = i + 1
    }
    return(True)
End isPerfect
```

(d)

```
Procedure isPerfect(X)
    C = []
    i = 1
    while(i <= X.LetterCount){
        A = ith letter in X.Word
        if(member(C,A)){
            return(True)
        }
        else{
            C = C ++ [A]
        }
        i = i + 1
    }
    return(False)
End isPerfect
```

Answer

(b)

Solution

To check if a word is a perfect or not we need to check that there are no repeated alphabets in the given word. There are mainly two methods we know to do this.

- First, pick the first letter from the word let us say **i** in outer loop and iterate through the remaining letters of word (i.e, second letter to last letter in inner loop) let us say **j**. If **i** == **j** for any letter in remaining letters, then directly return False. If not, then pick the second letter (this will become **i** now) and check if there is any **j** for which **i** == **j** in the next remaining letters (i.e., third letter to last letter). Repeat the procedure until **i** represents the second last letter of the given word.

  If we get a repeated letter at any point of time, the procedure should return False. If The procedure does not return False, then it should return True. In this method we will need a nested iteration. We can clearly see that no nested iteration is used so this method is not used in the options.

- Second method is using the **member** concept of List. In this method an empty list (let us say **C**) will be taken. First letter of the word will be picked up and check if it is already a member of **C**. As this the first letter and **C** is empty then the comparison will be False. Then append the first letter into the list. Then the second letter will be picked up and checked if it is already a member of **C**. If yes then the word has repeated letters and the procedure should immediately return False. If no, then the second letter should be appended to **C**. Same thing will happen with third, fourth, and other letters. If the procedure has not returned False after checking the last letter, it should return True.

It is clear that the second method is used in the question. We will go option wise to check which is the correct option.

Options

(a)

Line 10 : return(True)

The procedure will return True even after checking the first element. Because the first element will definitely not be the member of an empty list.

(b)

Line 7: return(False)

The procedure will return False if the letter is member of **C**.

Line 14: return(True)

The procedure will return True after going through all the letters. This seems to be a correct option.

(c)

Line 7 : The letters which are repeated will get appended to the list **C**.

Lien 10: return(False)

The procedure will return False if the letter is not the member of list **C**. This is not what we want.

(d)

The procedure will return True if the word has repeated element. Otherwise will return False. This is totally opposite of what we want.

Question 7 [5 Marks]

Statement

The given pseudocode is executed using a dataset having the same fields as the “Words” dataset, and contains the following words -

“I ordered this product from Gitark. I am very happy to share my review regarding this awesome product. It is not only nice to use, but also has a very cool look. I think this is the best product which can be bought in this price range. ”

Consider the following information:

1. **unique**(**L**) returns a list of unique elements of list **L**. For example **unique**(["think", "like", "toppers", "think"]) will return ["think", "like", "toppers"].
2. **comNo**(**L1, L2**) returns the number of common elements in lists **L1** and **L2**.
3. Ignore the upper and lower case, and punctuation symbols while comparing with other words

```
positiveList = [“happy”, “awesome”, “nice”, “fine”, “best”, “cool”]
posSen = 0, L = []
while(Table 1 has more rows){
    Read the first row X in Table 1
    L = L ++ [X.Word]
    if(X.Word ends with full stop){
        L = unique(L)
        posCount = comNo(positiveList, L)
        if(posCount >= 2){
            posSen = posSen + 1
        }
        L = []
    }
    Move X to Table 2
}
```

What will be the value of **posSen** at the end of the execution of the above pseudocode? (NAT)

Answer

2

Solution

```
positiveList = [“happy”, “awesome”, “nice”, “fine”, “best”, “cool”]
posSen = 0, L = []
while(Table 1 has more rows){
    Read the first row X in Table 1
    L = L ++ [X.Word] ---- Every word is being appended to the List. As L is being re-initialized after the word ends with a full stop, L is the list of words of the current sentence.
    if(X.Word ends with full stop){
        L = unique(L) ---- L is being updated. This is the list of unique words of the current sentence.
        posCount = comNo(positiveList, L) --- posCount is the number of common words between the current sentence and positiveList.
```

```
        if(posCount >= 2){
            posSen = posSen + 1 --- posSen is number of sentences in which
at least two words from positiveList are present.
        }
        L = []
    }
    Move X to Table 2
}
```

First sentence:

“I ordered this product from Gitark."

- No common words
- **posCount** = 0
- **posSen** = 0

Second Sentence:

"I am very happy to share my review regarding this awesome product."

- Two common words, "happy" and "awesome"
- **posCount** = 2
- **posSen** = 1

Third Sentence:

"It is not only nice to use, but also has a very cool look."

- Two common words, "nice" and "cool"
- **posCount** = 2
- **posSen** = 2

Fourth Sentence:

"I think this is the best product which can be bought in this price range. ”

- One common word, "best"
- **posCount** = 1
- **posSen** = 2

Question 8 [6 Marks]

Statement

Mona tells Sona that at least 50 percent of sentences have nouns just after an adjective. Sona writes the following pseudocode to find if Mona is right or not. At the end of the execution of the pseudocode given below, **A** stores True if Mona is right otherwise False. But Sona might have made mistakes in one or more lines. Identify such lines (if any). It is a Multiple Select Question (MSQ). Assume that there is at least one adjective in every sentence.

```
1  A = False, trueCount = 0, totalCount = 0, posList = []
2  while(Table 1 has more rows){
3      Read the first row X in Table 1
4      posList = posList ++ [X.PartOfSpeech]
5      if(X.Word ends with a full stop){
6          if(isTrue(posList) == 0){
7              trueCount = trueCount ++ [1]
8          }
9          else{
10             totalCount = totalCount + 1
11         }
12         posList = []
13     }
14     Move X to Table 2
15 }
16 if(trueCount / totalCount >= 0.5){
17     A = True
18 }
19
20 Procedure isTrue(L)
21     count = 0
22     while(length(L) >= 2){
23         if(first(L) == "Adjective"){
24             count = count + 1
25             if(first(rest(L)) == "Noun"{
26                 count = count - 1
27             }
28         }
29         L = rest(L)
30     }
31     return(count)
32 End isTrue
```

Options

(a)

Line 4: Invalid addition operation for appending in **posList**

(b)

Line 7: Invalid increment of trueCount

(c)

Line 9 - 11: These three lines should be replaced by

```
totalCount = totalCount + 1
```

(d)

Line 26: The current statement should be replaced by

```
count = count + 1
```

(e)

No mistakes

Answer

(b), (c)

Solution

Line 16 - 18 are not mentioned into the options. This means these are correct statements. **A** is turned to True if **trueCount** is half of **totalCount**. This means **trueCount** is the number of sentences which satisfies the hypothesis and **trueCount** is the number of sentences having at least one adjective (in this question number of total sentences). Therefore, these two variables should be initialized to 0.

Line 4 and line 12: **posList** is the list of part of speech of the words in the same sequence as the words are in the current sentence. Therefore, **posList** should be initialized to an empty list and should be re-initialized once the sentence ends. Appending is done properly in line 4.

Line 7: **trueCount** is an integer. Therefore, normal addition should be done. **ERROR**

Line 10: **totalCount** is the number of sentences in the dataset. Therefore, it should be incremented for every sentence irrespective any other conditions. Therefore, No else block is required. **Option c**

Procedure **isTrue** returns **count**. **count** is incremented for every adjective. In the nested filtering, it is being checked if the sentence has noun just after an adjective. From line 6, the sentence is supposed to follow the hypothesis if the procedure returns 0 which means the value of count is 0. Therefore, in line 26, count should be decremented by 1. Which is properly done inside the procedure.

Question (9-10)

Statement

Let **a** and **b** be positive integers. Procedure **remainder(a, b)** returns remainder if **a** is divided by **b**.

```
Procedure doSomething(X)
    j = 2, Flag = True
    if(X == 1){
        return(False)
    }
    while(j < X){
        if(remainder(X, j) == 0){
            Flag = False
            exitloop
        }
        j = j + 1
    }
    return(Flag)
End doSomething
```

Question 9 [4 Marks]

Statement

When will procedure **doSomething**(**X**) return True?

Options

(a)

X is a prime number

(b)

X is an even number

(c)

X is an odd number

(d)

X is more than 1

Answer

(a)

Solution

In this question it is asked that when procedure **doSomething** returns True. Let us understand the procedure what it does? It is clear that it accepts an integer and returns either True or False. In line 2, **i** and **Flag** are initialized to 1 and True. In line 3, when **X** is 1, procedure returns False. If **X** is 2 then while loop will not be executed since **j** is initialized to 2. Therefore it will return True.

When X is greater than 2 then while loop will be executed. Now let us check while loop closely. In while loop , if -block checks the returned value of the procedure **remainder**. If **X** is divisible by any value of **J** (less than **X**), **Flag** is updated to False and loop is terminated by exitloop. It means if **X** is not divided by any number other than itself and 1 then procedure **doSomething** returns True.

Hence procedure **doSomething(X)** will return True when **X** is a prime number. For example X = 2, 3, 5, 7, ...

Question 10 [5 Marks]

Consider the procedure discussed above. What will be the value of **M** at the end of the execution of the given pseudocode below?

```
L = [6, 10, 11, 23, 7, 50]
M = []
position = 1
foreach element in L{
    if(doSomething(position) and doSomething(element)){
        M = M ++ [element]
    }
    position = position + 1
}
```

Options

(a)

M = [11, 7]

(b)

M = [11, 23, 7]

(c)

M = [11]

(d)

M = [23, 7]

Answer

(a)

Solution

In this poblem, **M** and **position** is initialized with an empty list and 1 respectively. **foreach** loop iterates all the elements of list **L**. Inside the foreach loop, **element** will be appended to list **M** only when return values of **doSomething(position)** and **doSomething(element)** are True. And from the previous problem we know that **doSomething(X)** returns True only when **X** is a prime number.

Hence, an element of **L** will be appended to **M** only when it is a prime number and also its position in the list **L** is a prime number.

Therefore value of **M** = [11, 7]