Week-4, Graded

This document has 10 questions.

Question-1

Statement

Let `X` be a vector. We wish to compute the dot product of `X` with itself and store it in a variable called `dp`. Select the correct code snippet(s) from the options given below. (MSQ)

Options

(a)

```python
dp = 0
for elem in X:
    dp += elem * elem
```

(b)

```python
dp = 0
for i in range(len(X)):
    dp += X[i] ** 2
```

(c)

```python
dp = 0
for i in range(X):
    dp += X[i] * X[i]
```

(d)

```python
i, dp = 0, 0
while i <= len(X):
    dp += X[i] ** 2
    i += 1
```

(e)

```python
i, dp = 0, 0
while i < len(X):
    dp += X[i] ** 2
    i += 1
```

Answer

(a), (b), (e)

Feedback

Correct options

- Option-(a): Here, we pick up each element from `X`, multiply it with itself and add it to `dp`.
- Option-(b): The only difference from the previous option is the use of indices.
- Option-(e): This uses a while loop while the other two correct solutions use `for`.

This is to demonstrate that there are multiple approaches to compute the dot product.

Wrong options

- As far as option-(c) is concerned, do you think `range(X)` makes sense when `X` is a list?
- What is the difference between options (d) and (e)?

Common Data for Questions 2, 3

Statement

Execute the following snippet of code and then answer questions 2 and 3.

```python
import random

rolls = [ ]
for i in range(100000):
    # randint includes the endpoints
    roll = random.randint(1, 6)
    rolls.append(roll)

count = 0
primes = [2, 3, 5]
for roll in rolls:
    if roll in primes:
        count += 1
```

Question-2

Statement

What is the size of the list `rolls`? (NAT)

Answer

100000

Feedback

The idea of the program is to simulate $100,000$ rolls of a dice. This is accomplished using Python's built-in `random` library:

```python
import random

rolls = [ ]
for i in range(100000):
    # randint includes the endpoints
    roll = random.randint(1, 6)
    rolls.append(roll)
```

It is a good idea to run this locally and see what you get. `len(rolls)` will be $100,000$ in this case.

Question-3

Statement

Execute the following snippet of code.

```
some_var = count / len(rolls)
```

What does the variable `some_var` represent?

Options

(a)

It represents the probability that a number chosen at random between 1 and 100000 is a prime.

(b)

It represents the number of primes between 1 and 100000

(c)

It represents the number of prime numbers in the list `rolls`.

(d)

It represents the probability that a number chosen at random from the list `rolls` is a prime.

(e)

It represents the probability that a number chosen at random from the list `primes` is a prime.

Answer

(d)

Feedback

Questions to ask yourself:

- What is the purpose of the for-loop from lines 11-13?
- What does the variable `count` represent?
- When does the expression `roll in primes` in line-12 evaluate to `True`?

Common data for questions 4 and 5

Statement

Assume that `L` is a non-empty list of positive integers. Also assume that the list is a distinct collection of numbers, i.e., no two numbers are alike. Consider the following code. Answer questions 4 and 5 based on this code.

```python
S = 0
for x in L:
    S += x

flag = False
y = -1
for x in L:
    if x * len(L) == S:
        flag = True
        y = x
        break
```

Question-4

Statement

If `flag` is `True` at the end of execution of the code given above, which of the following statements are true? Note that the options should be true for any list `L` that satisfies the conditions given in the common data. Multiple options could be correct.

Options

(a)

`y` is an element in the list `L`

(b)

`y` is the smallest number in the list

(c)

`y` is the greatest number in the list

(d)

`y` is the average (arithmetic mean) of the numbers in the list

(e)

`y` is the element at index `len(L) // 2` in the list `L`

Answer

(a), (d)

Feedback

The basic idea is this: if we replace all the elements in the list `L` with one of its elements, say `y`, the sum of the resulting list is the same as the old list. What does this say about the element `y`? How is this element related to the average (arithmetic mean) of the numbers in the old list?

As a concrete example, consider the following list:

```python
L1 = [5, 1, 4, 3, 2]
```

Here, if we replace all the elements with the arithmetic mean of the list, we get:

```python
L2 = [3, 3, 3, 3, 3]
```

We see that `sum(L1) == sum(L2)`. For this specific example, we see why (a) and (d) should be correct. But these options hold good for any non-empty list `L` of distinct positive integers. A more abstract argument is as follows. Consider a list of n numbers, not necessarily in sorted order:

$$L = [x_1, \cdots, x_n]$$

For the if-condition to be `True`, there should be some element y in the list such that:

$$n \cdot y = x_1 + \cdots + x_n$$

Dividing both sides by n, we get:

$$y = \frac{x_1 + \cdots + x_n}{n}$$

Thus, we see that y is nothing but the arithmetic mean of the elements in the list.

Question-5

Statement

Assume that `L` is a list of the first n positive integers, where $n > 0$. Under what conditions will the variable `flag` be `True` at the end of execution of the code given above?

Options

(a)

`n` is an odd integer

(b)

`n` is an even integer

Answer

(a)

Feedback

Continuing with the argument, we have:

$$y = \frac{1 + \cdots + n}{n} = \frac{n+1}{2}$$

For y to be an integer, n has to be odd.

Question-6

Statement

Consider the following code snippet.

```python
L = input().split(' ')  # there is a single space between the quotes
for i in range(len(L) - 1):
    if len(L[i]) > len(L[i + 1]):
        L[i], L[i + 1] = L[i + 1], L[i]

count = 0
for i in range(len(L)):
    if len(L[i]) <= len(L[-1]):
        count += 1

print(count)
```

What is the output of the code given above for the following input?

Input

```
seriously this world needs heroes like you now more than ever
```

There is exactly one space between consecutive words. There are no spaces before the first word after the last word.

Answer

11

Feedback

Recall that `x, y = y, x` is the way to swap the values in the two variables.

Here, the code bubbles the longest word to the rightmost end of the list. This is an important step in a sorting algorithm called bubble sort. In bubble sort, you swap two adjacent elements in the list, whenever the left-element is bigger (whatever this means) than the right element. Let us see how this works with numbers first.

Example

Consider a list of length 5. The idea is to loop through the list and compare a pair of adjacent elements, say $(\text{left}, \text{right})$, in each iteration. If $\text{left} > \text{right}$, then the two elements are swapped. If $\text{left} \leq \text{right}$, then we don't disturb their positions. The rationale behind this is to "bubble" the largest element and push it to the right end of the list:

```python
L = [5, 10, 3, 2, 4]
for i in range(len(L) - 1):
    if L[i] > L[i + 1]:
        L[i], L[i + 1] = L[i + 1], L[i]
    print(f'After iteration {i + 1}: {L}')
```

This is the output:

```
After iteration 1: [5, 10, 3, 2, 4]
After iteration 2: [5, 3, 10, 2, 4]
After iteration 3: [5, 3, 2, 10, 4]
After iteration 4: [5, 3, 2, 4, 10]
```

Note the position of 10 across all iterations. In the first iteration, we compare $(5, 10)$. Since 10 is to the right of 5, we leave it as it is. But in the next three iterations, we make the following comparisons:

- $(10, 3)$: swap
- $(10, 2)$: swap
- $(10, 4)$: swap

Now, you can use this idea of bubble sort to understand the original question. Here, the longest word is being bubbled to the right. Once this is done, the following code is executed:

```python
count = 0
for i in range(len(L)):
    if len(L[i]) <= len(L[-1]):
        count += 1
```

Here, `count` will be the length of the list `L`, which is the same as the number of words in the sentence. Why is this the case?

Question-7

Statement

Consider the following sequence:

$$7, 77, 777, 7777, \cdots$$

Let x be the smallest element in this sequence that is divisible by 2003. How many digits does x have?

Hint: Think about while loops.

Answer

1001

Feedback

One doesn't necessarily have to use lists to solve this problem. Here is one using lists:

```python
L = [7]
last = L[-1]
while last % 2003 != 0:
    num = L[-1] * 10 + 7
    L.append(num)
    last = L[-1]
print(len(L))
```

Here is one that doesn't use lists:

```python
num = 7
count = 1
while num % 2003 != 0:
    num = num * 10 + 7
    count += 1
print(count)
```

Question-8

Statement

Consider the following snippet of code.

Hint: `'a b c'.split(' ')` results in the list `['a', 'b', 'c']`.

```python
 1  n = int(input())
 2  M = [ ]
 3  for i in range(n):
 4      row = [ ]
 5      for num in input().split(','):
 6          row.append(int(num))
 7      M.append(row)
 8
 9  some_var = 0
10  for i in range(n):
11      for j in range(n):
12          some_var += M[i][j]
13
14  print(some_var)
```

Find the output of the code if the input is:

```
5
1,2,3,4,5
6,7,8,9,10
11,12,13,14,15
16,17,18,19,20
21,22,23,24,25
```

Answer

325

Feedback

We shall try to break the code into two parts. Line-8, an empty line, provides a natural point of separation:

Part-1

We are accepting a matrix as input here. Based on the code given below, does M have to be a square matrix?

```python
n = int(input())
M = [ ]
for i in range(n):
    row = [ ]
    for num in input().split(','):
        row.append(int(num))
    M.append(row)
```

Part-2

```python
some_var = 0
for i in range(n):
    for j in range(n):
        some_var += M[i][j]
```

For this particular problem, it turns out that we are working with a square matrix of size $n \times n$. `some_var` is storing the sum of the elements of the matrix.

Question-9

Statement

What is the output of the following snippet of code?

Hint:

(1) If `L = [1, 2, 3, 4, 5]`, then `L[1: 3]` is the list `[2, 3]`. Slicing a list is very similar to slicing a string. All the rules that you have learned about string-slicing apply to list-slicing.

(2) If `P = [1, 2, 3]` and `Q = [4, 5, 6]` then `P + Q` is the list `[1, 2, 3, 4, 5, 6]`. Again, list concatenation is very similar to string concatenation.

```
L = [90, 47, 8, 18, 10, 7]
S = [L[0]]  # list containing just one element
for i in range(1, len(L)):
    flag = True
    for j in range(len(S)):
        if L[i] < S[j]:
            before_j = S[: j]   # elements in S before index j
            new_j = [L[i]]      # list containing just one element
            after_j = S[j: ]    # elements in S starting from index j
            # what is the size of S now?
            S = before_j + new_j + after_j
            # what is the size of S now?
            flag = False
            break
    if flag:
        S.append(L[i])
print(S)
```

Options

(a)

```
[90, 47, 8, 18, 10, 7]
```

(b)

```
[7, 10, 18, 8, 47, 90]
```

(c)

```
[7, 8, 10, 18, 47, 90]
```

(d)

```
[90, 47, 18, 10, 8, 7]
```

(e)

```
[90, 7, 8, 10, 18 47]
```

Answer

(c)

Feedback

The code in question is a sorting algorithm called insertion sort. The core idea behind insertion sort is to insert an item into an already sorted list and iteratively use this idea to sort the entire list. Initially, the sorted list, S, has just one element, namely the first element of the original list, L. In each subsequent step, we pick up the next element from the original list L and insert it in the right place in the sorted list S. Eventually, we end up with a list S which has the same elements as L but in sorted order. Let us first see the algorithm's execution on the list in question:

Step-1

In this step, L is the list to be sorted. S is a singleton list that has the first element of `L` and is already sorted.

$$L = [90, \quad 47, \quad 8, \quad 18, \quad 10, \quad 7]$$

$$S = [90]$$

Step-2

We now pick up the second element in L. Since $47 < 90$, that is inserted before 90 in S.

$$L = [90, \quad 47, \quad 8, \quad 18, \quad 10, \quad 7]$$

$$S = [47, \quad 90]$$

Step-3

We now pick up the third element in L. Since $8 < 47$, that is inserted before 47 in S.

$$L = [90, \quad 47, \quad 8, \quad 18, \quad 10, \quad 7]$$

$$S = [8, \quad 47, \quad 90]$$

Step-4

We now pick up the fourth element in L. Since 18 is not less than 8, we compare it with the next element in S. Here, $18 < 47$, so this is the right place to insert 18 in S.

$$L = [90, \quad 47, \quad 8, \quad 18, \quad 10, \quad 7]$$

$$S = [8, \quad 18, \quad 47, \quad 90]$$

Step-5

We now pick up the fifth element in L. You must have a fair idea of what is happening.

$$L = [90, \quad 47, \quad 8, \quad 18, \quad 10, \quad 7]$$

$$S = [8, \quad 10, \quad 18, \quad 47, \quad 90]$$

Step-6

Finally, we pick up 7.

$$L = [90, \quad 47, \quad 8, \quad 18, \quad 10, \quad 7]$$

$$S = [7, \quad 8, \quad 10, \quad 18, \quad 47, \quad 90]$$

Code

How do we code this up? Notice that there is a nested loop. The outer loop iterates through the elements of L. We start iterating from 1 as we have already taken care of the zeroth element in L. We will use zero-indexing for the rest of the explanation.

```python
for i in range(1, len(L)):
    flag = True
```

`flag` is a variable that tells us if the element $L[i]$ has been inserted into the list S or not. The moment $L[i]$ is inserted into S within the inner loop, `flag` becomes `False`. Now, the inner loop does the insertion part. It iterates through the list S:

```python
for j in range(len(S)):
    if L[i] < S[j]:
        before_j = S[: j]    # elements in S before index j
        new_j = [L[i]]       # list containing just one element
        after_j = S[j: ]     # elements in S starting from index j
        # what is the size of S now?
        S = before_j + new_j + after_j
        # what is the size of S now?
        flag = False
        break
```

Note the comparison of $L[i]$ with each element $S[j]$ in the list S. The moment we see that $L[i] < S[j]$ for some j, we know that it is time to insert $L[i]$ into S. To this end, we create two sub-lists `before_j` and `after_j`.

- `before_j` represents the first j elements in S. That is, $S[0], \cdots, S[j-1]$
- `after_j` represents all elements after the j^{th} element in S. That is, $S[j], \cdots, S[len(S)-1]$
.

Since $L[i] < S[j]$, we see that $L[i]$ is going to become the new $S[j]$. This is how the insertion happens:

```python
S = before_j + new_j + after_j
```

Note that all three objects in the RHS of the above assignment are lists. The moment insertion is done `flag` is updated to `False`. Now, some of you might be wondering what the following lines do in the original code:

```python
if flag:
    S.append(L[i])
```

This is left as an exercise to you. One additional note, for some iterations in this code, we notice that $L[i] < S[0]$. In such a case `before_j` is $S[:0]$. This is nothing but an empty list. Also, $S[0:]$ is nothing but S itself. This scenario occurs when $L[i]$ is smaller than the smallest (zeroth) element in S. For example, consider the value of S just before step-6.

$$L = [90, \quad 47, \quad 8, \quad 18, \quad 10, \quad 7]$$

$$S = [8, \quad 10, \quad 18, \quad 47, \quad 90]$$

Here, $L[5] = 7$ and $L[5] < S[0]$, so:

```
before_j = [ ]
new_j = [7]
after_j = [8, 10, 18, 47, 90]
```

S is the concatenation of these three lists.

Question-10

Statement

A square matrix M is said to be symmetric if $M[i][j] = M[j][i]$ for $0 \leq i, j \leq n-1$. Note the use of zero-based indexing here.

Assume that a square matrix `M` is already defined. Select all correct implementations of a program that prints `YES` if it is a symmetric matrix and `NO` if it is not a symmetric matrix. (MSQ)

Options

(a)

```python
# M is a square matrix which is already defined.
sym = True
for i in range(len(M)):
    for j in range(len(M)):
        if(M[i][j] == M[j][i]):
            sym = True
        else:
            sym = False
if sym:
    print('YES')
else:
    print('NO')
```

(b)

```python
# M is a square matrix which is already defined.
sym = True
for i in range(len(M)):
    for j in range(len(M)):
        if(M[i][j] == M[j][i]):
            sym = True
        else:
            sym = False
            break
if sym:
    print('YES')
else:
    print('NO')
```

(c)

```python
# M is a square matrix which is already defined.
sym = True
for i in range(len(M)):
    for j in range(len(M)):
        if(M[i][j] == M[j][i]):
            sym = True
            break
        else:
            sym = False
if sym:
    print('YES')
else:
    print('NO')
```

(d)

```python
# M is a square matrix which is already defined.
sym = True
for i in range(len(M)):
    for j in range(i):
        if(M[i][j] == M[j][i]):
            sym = True
        else:
            sym = False
            break
if sym:
    print('YES')
else:
    print('NO')
```

(e)

```python
# M is a square matrix which is already defined.
sym = True
for i in range(len(M)):
    for j in range(i):
        if(M[i][j] != M[j][i]):
            sym = False
            break
    if not sym:
        break
if sym:
    print('YES')
else:
    print('NO')
```

Answer

(e)

Feedback

Consider the matrix M:

$$M = \begin{bmatrix} 3 & 6 & 1 \\ 5 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

In Pythonic terms, this matrix is:

```python
M = [[3, 6, 1], [5, 3, 2], [1, 2, 3]]
```

Note that parts of the matrix may be symmetric. For example, in this matrix M all the blue elements conform to the specification for a symmetric matrix. The only two elements which break this are $M[0][1]$ and $M[1][0]$.

$$M = \begin{bmatrix} {\color{blue}3} & 6 & {\color{blue}1} \\ 5 & {\color{blue}3} & {\color{blue}2} \\ {\color{blue}1} & {\color{blue}2} & {\color{blue}3} \end{bmatrix}$$

We iterate through the rows of the matrix in the outer-loop. Ideally, the moment we see a pair of elements that break the property in any given row, we must break the iterative process. The important point is, we have to break out of both the loops to entirely terminate the process. With this understanding let us look at the correct option.

Correct option

```python
 1  # M is a square matrix which is already defined.
 2  sym = True
 3  for i in range(len(M)):
 4      for j in range(i):
 5          if(M[i][j] != M[j][i]):
 6              sym = False
 7              break
 8      if not sym:
 9          break
10  if sym:
11      print('YES')
12  else:
13      print('NO')
```

- `sym` is a state-variable that is set to `True` in line-2. That is, we are optimistically assuming that the matrix is symmetric to begin with. In line-5, we are checking if a pair of elements are breaking the property. If they do so, then we update `sym` to `False` and break out of the inner loop. But it is not enough to just break out of the inner loop, we also need to break out of the outer loop. This is why we have an additional condition in line-8.
- Another interesting point to note is the range that is used for the inner loop. In the i^{th} row, we are only going from $M[i][0]$ till $M[i][i-1]$. We are not interested in the diagonal elements $M[i][i]$. But more importantly, the elements in this row with column-index $j > i$, will be eventually covered in subsequent iterations of the outer loop. This kind of iteration can be pictured as follows:

$$M = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Wrong options

- Option-(a) is wrong as we are not breaking out of the loop anywhere when `sym` becomes `False`. This is a problem because the bottom right element in M will always make `sym` `True` because $i = j$ for this case.
- In option-(b), we are breaking out of the inner loop but not out of the outer loop.
- Option-(c) is totally wrong as we are breaking at the wrong place.
- Option-(d) is not breaking twice.