Week 4 / Chapter 10 exercises

Matin Ahmadi Rouzbahani /40113841054052

Answer 1 :

Yes, a Python list can hold a mixture of integers and strings.

Answer 2:

If you attempt to access an element of a list using a negative index in Python, it will return the element from the end of the list. For example, if you have a list with five elements and you try to access the element at index -1, it will return the last element in the list. Similarly, if you try to access an element at index -2, it will return the second-to-last element in the list, and so on. However, if you try to access an index that is beyond the length of the list using negative indexing, it will result in an "IndexError" exception.

Answer 3 :

The Python statement that produces a list containing the values 45, −3, 16 and 8, in that order is:

```
python
my_list = [45, -3, 16, 8]
```

Answer 4 :

A) The expression lst[0] represents the very first element of lst, which is 10.

B) The expression lst[-1] represents the very last element of lst, which is 29.

(C) lst[0] is 10.
(D) lst[3] is 29.
(E) lst[1] is -4.
(F) lst[-1] is 29.
(G) lst[-4] is 10.
(H) The expression lst[3.0] is illegal because the index must be an integer, not a float.

Answer 5 :

(A) lst[0] = 3
(B) lst[3] = 5
(C) lst[x] = lst[2] = 1

(D) lst[-x] = lst[-2] = 5
(E) lst[x + 1] = lst[3] = 5
(F) lst[x] + 1 = 1 + 1 = 2
(G) lst[lst[x]] = lst[lst[2]] = lst[1] = 0
(H) lst[lst[lst[x]]] = lst[lst[lst[2]]] = lst[lst[1]] = lst[0] = 3

Answer 6 :

The function that returns the number of elements in a list in Python is

`len()`

.

Answer 7 :
The expression that represents the empty list in Python is [] or list().

Answer 8 :

A) [20, 1, -34, 40, -8, 60, 1, 3]

B) [20, 1, -34]

C) [-8, 60, 1, 3]

D) [-8, 60, 1, 3]

E) [-8, 60]

F) [20, 1,-34]

G) [-8,60 ,1 ,3]

H) [20 ,1 ,-34 ,40 ,-8 ,60 ,1 ,3]

I) [20 ,1 ,-34 ,40]

J) [1 ,-34 ,40 ,-8]

K) True

L) False

M) 8

Answer 9 :

Original List [2,4,6,8,10]

Target List [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

m = len(a), n = len(a) + 5

a[m:n] = [12,14,16,18,20]

Modified List [2,4,6,8,10,12,14.16.18.20]

Target List [-10,-8,-6,-4,-2.0.2.4.6.8.10]

m = -11,n = -1

a[m:n] = [-10,-8,-6,-4,-2,O.2.4.6.8.lO]

Modified List [-10.-8.-6.-4.-2.O.2A.A.A.lO]

Target List [2.3A.A.A.A.A.lO]

m = l,n=7

a[m:n]=[2,A,A,A,A,lO]

Modified List [2,A,A,A.A.lO]

Target List [2A.A.cA.lOJ

m=3,n=6

a[m:n]=['a','b','c']

Modified List [2A.A.'a','b','c',8,lO]

Target List []

m=n=0

a[m:n]=[]

Modified List []

Target List [l0.B.B.B.BJ

m=-1,n=-6

a[m:n]=[l0,B,B,B]

Modified List[2A.A.'a','b','c',B,B,l0]

Target List [246J

m=0,n=3

a[m:n]=[2A.,4.,6.]

Modified list[246810]

Target list[6810J

m=3,n=len(a)

a[m:n]=[8,10]

Modified list[2A.A.8.lO]

Target List [2.10]

m=0,n=2

a[m:n]=[2,lO]

Modified List [2A.A.8.lO]

Target List [468J

m=1,n=4

a[m:n]=[4,6,8]

Modified List[2A.,4.,6.,8,lO]

_

Original List [2,4,6,8,10] Target List m:n

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20] 5:11
[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10] -12:11
[2 ,3 ,4 ,5 ,6 ,7 ,8 ,10] :
[2 ,4 ,6 ,'a' ,'b' ,'c' ,8 ,10] 3:6
[] :
[10 ,8 ,6 ,4 ,2] ::-1
[2 ,4 ,6][:3]
[6 ,8 ,10][:-1]
[2 ,10][::len([2])]
[4 ,6 ,8][:-1]

Answer 10 :

(A) [8, 8, 8, 8]
(B) [2, 7, 2, 7, 2, 7]
(C) [1, 2, 3, 'a', 'b', 'c', 'd']
(D) [1, 2, 1, 2, 1, 2, 4, 2]
(E) [1, 2, 4, 2, 1, 2, 4 ,2 ,1 ,2 ,4 ,2]

Answer 11 :

(A) [3, 5, 7, 9]
(B) [50, 60, 70, 80, 90]
(C) [12, 15, 18]
(D) [(0,0), (0,1), (0,2), (0,3), (1,0), (1,1), (1,2), (1,3), (2,0), (2,1), (2,2), (2,3)]
(E) [(0,0), (0,2), (1,1), (1,3), (2,0), (2,2)]

Answer 12 :

(A) [x**2 for x in range(1, 6)]

(B) [x/4 for x in range(1, 7)]

(C) [(char, num) for char in ['a', 'b'] for num in range(3)]

Answer 13 :

The expression to check whether x is a member of lst is:

```
python
x in lst
```

This expression returns a boolean value True if x is present in the list lst, and False otherwise.

Answer 14 :

The

`reversed()`
function in Python returns a reverse iterator. It reverses the order of the elements in a sequence such as a list, tuple, or string. The original sequence is not modified, but a new reversed sequence is created.
The

`reversed()`
function can be used with a for loop or converted to a list using the

`list()`
function.

For example:

```
my_list = [1, 2, 3, 4]
for i in reversed(my_list):
   print(i)

# Output:
# 4
# 3
# 2
# 1

reversed_list = list(reversed(my_list))
print(reversed_list)

# Output: [4, 3, 2, 1]
```

Answer 15 :

Here's the code to add up all the positive values in a list of integers:

```
def sum_positive(a):
    total = 0
    for num in a:
        if num > 0:
            total += num
    return total if a else 0
```

Answer 16 :

Here's the code to count the even numbers in a list of integers:

```
def count_evens(lst):
    count = 0
    for num in lst:
        if num % 2 == 0:
            count += 1
    return count
```

Answer 17 :

Here's the code for the function:

```
  python
def print_big_enough(numbers, minimum):
   for num in numbers:
      if num >= minimum:
         print(num)
```

___

To use this function with the data provided earlier, you can call it like this:

```
  python
data = [3, 5, 7, 2, 8, 4]
print_big_enough(data, 5)
```

___

This will output:

___

```
5
```

7
8
___

Answer 18 :

Here's the code for the next_number function:

```
python
def next_number(lst):
    lst = sorted(lst)
    next_num = 1
    for num in lst:
        if num == next_num:
            next_num += 1
    return next_num
```

Explanation:

- The function first sorts the input list in ascending order using the

`sorted()`
function.
- It then initializes a variable

`next_num`
to 1, which is the smallest positive integer.
- The function then loops through each element in the sorted list. If an element is equal to

`next_num`
, it means that

`next_num`
is already present in the list, so we increment

`next_num`
by 1. If an element is greater than

`next_num`
, it means that

`next_num`
is not present in the list, so we can return it as the smallest positive integer not in the list.
- If we have looped through all elements in the list and haven't found a missing positive integer, it means that all integers from 1 to len(lst)+1 are present in the list (since there are no duplicates), so we can simply return len(lst)+1 as the smallest missing positive integer.

Here are some test cases:

```
python
print(next_number([5, 3, 1])) # Output: 2
print(next_number([5, 4, 1, 2])) # Output: 3
print(next_number([2, 3])) # Output: 1
print(next_number([])) # Output: 1
```

Answer 19 :

Here's the code for the reverse function:

```
python
def reverse(a):
    """
    Reverses the contents of a list in place.
    """
    i = 0
    j = len(a) - 1
    while i < j:
        a[i], a[j] = a[j], a[i]
        i += 1
        j -= 1
```

To use this function, simply call it with your list as an argument:

```
python
my_list = [1, 2, 3, 4, 5]
reverse(my_list)
print(my_list) # Output: [5, 4, 3, 2, 1]
```

Answer 20 :

```
m = [[0.1]*9 for i in range(7)]
m[2][4] = 0
for row in m:
  print(row)
```

Answer 21 :

1. Using a list comprehension:

```
lst = [i for i in range(1, 11)]
```

2. Using the range() function and converting it to a list:

```
lst = list(range(1, 11))
```

3. Using the append() method in a for loop:

```
lst = []
for i in range(1, 11):
   lst.append(i)
```

4. Using the extend() method with an empty list and the range() function:

```
lst = []
lst.extend(range(1, 11))
```

5. Using the * operator to repeat a tuple and then converting it to a list:

```
tup = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
lst = list(tup * 1)
```

Answer 22 :

Here's one way to write the function:

```
python
def check_square_list(square_list):
    n = len(square_list)
    for i in range(n):
        row_sum = sum(square_list[i])
        col_sum = sum(square_list[j][i] for j in range(n))
        if row_sum == col_sum:
            return True
    return False
```

This function takes a square 2D list as input and returns True if any row has the same sum as any column. It does this by iterating over each row and calculating its sum, then iterating over each column and calculating its sum. If it finds a match, it immediately returns True. If it finishes iterating over all rows and columns without finding a match, it returns False.

Note that this function assumes that the input is a square 2D list (i.e., the number of rows equals the number of columns). If you want to add error checking to ensure that this is the case, you can add something like this at the beginning of the function:

```
python
def check_square_list(square_list):
    n = len(square_list)
    if any(len(row) != n for row in square_list):
        raise ValueError("Input must be a square 2D list")
    # rest of function...
```

This will raise a ValueError if any row has a different length than the number of rows/columns.

Answer 23 :

Here's the code for the check_winner function:

```
def check_winner(board):
    # Check rows
    for row in board:
        if row.count("X") == 3:
            return "X"
        elif row.count("O") == 3:
            return "O"

    # Check columns
    for i in range(3):
        column = [board[j][i] for j in range(3)]
        if column.count("X") == 3:
            return "X"
        elif column.count("O") == 3:
            return "O"

    # Check diagonals
    diagonal1 = [board[i][i] for i in range(3)]
    diagonal2 = [board[i][2-i] for i in range(3)]
    if diagonal1.count("X") == 3 or diagonal2.count("X") == 3:
        return "X"
    elif diagonal1.count("O") == 3 or diagonal2.count("O") == 3:
        return "O"

    # No winner found
    return " "
```

finish