Homework-second week-ordibehesht-#Hooman_ileayi#

1. Can a Python list hold a mixture of integers and strings?

Yes, a Python list can hold a mixture of integers and strings. In fact, a list can hold any combination of data types, including other lists and even objects.

2. What happens if you attempt to access an element of a list using a negative index?

If you attempt to access an element of a list using a negative index, Python will count from the end of the list instead of the beginning. For example, if you have a list `my_list = [1, 2, 3, 4, 5]`, then `my_list[-1]` will return `5`, `my_list[-2]` will return `4`, and so on. However, if the absolute value of the negative index is greater than or equal to the length of the list, then Python will raise an `IndexError` exception.

3. What Python statement produces a list containing the values 45, -3, 16 and 8, in that order?

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

python```

[۸ ,۱۶ ,۳- ,۴۵] = my_list

```

This creates a list called `my_list` with four elements, each of which is one of the specified values.

4. Given the statement
lst = [10, -4, 11, 29]
(a) What expression represents the very first element of lst?

(b) What expression represents the very last element of lst?

(c) What is lst[0]?

(d) What is lst[3]?

(e) What is lst[1]?

(f) What is lst[-1]?

(g) What is lst[-4]?

(h) Is the expression lst[3.0] legal or illegal?

.`(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 equal to `10
```

.`(d) `lst[3]` is equal to `29

.`(e) `lst[1]` is equal to `-4

.`(f) `lst[-1]` is equal to `29

.`(g) `lst[-4]` is equal to `10

(h) The expression `lst[3.0]` is illegal because list indices must be integers, not floating-point numbers.

5. Given the statements
lst = [3, 0, 1, 5, 2]
x = 2
evaluate the following expressions: (a) lst[0]?
(b) lst[3]?

(c) lst[x]?

(d) lst[-x]?

(e) lst[x + 1]?

(f) lst[x] + 1?

(g) lst[lst[x]]?

(h) lst[lst[lst[x]]]?

:Given the statements `lst = [3, 0, 1, 5, 2]` and `x = 2`, the following expressions evaluate as follows

.`(a) `lst[0]` evaluates to `3

.`(b) `lst[3]` evaluates to `5

.`(c) `lst[x]` evaluates to `1

.`(d) `lst[-x]` evaluates to `1

.`(e) `lst[x + 1]` evaluates to `5

.`(f) `lst[x] + 1` evaluates to `2

.`(g) `lst[lst[x]]` evaluates to `0

`(h) `lst[lst[lst[x]]]` evaluates to `3

6. What function returns the number of elements in a list?

The `len()` function can be used to return the number of elements in a list. For example, if you have a list `my_list = [1, 2, 3, 4, 5]`, then `len(my_list)` will return `5

7. What expression represents the empty list?

The expression `[]` represents the empty list. This is a list with no elements, and it is commonly used .as a starting point for building up lists using other operations like appending or concatenation

8. Given the list
lst = [20, 1, -34, 40, -8, 60, 1, 3]
evaluate the following expressions:
(a) lst
(b) lst[0:3]
(c) lst[4:8]
(d) lst[4:33]
(e) lst[-5:-3]
(f) lst[-22:3]
(g) lst[4:]
(h) lst[:
] (i) lst[:4]
(j) lst[1:5]
(k) -34 in lst
(l) -34 not in lst

(m) len(lst)

:Given the list `lst = [20, 1, -34, 40, -8, 60, 1, 3]`, the following expressions evaluate as follows

.`[٣ ,١ ,٦٠ ,٨- ,٤٠ ,٣٤- ,١ ,٢٠]` (a) `lst` evaluates to

.`[٣٤- ,١ ,٢٠]` (b) `lst[0:3]` evaluates to

.`[٣ ,١ ,٦٠ ,٨-]` (c) `lst[4:8]` evaluates to

.`[٣ ,١ ,٦٠ ,٨-]` (d) `lst[4:33]` evaluates to

.`[٦٠ ,٨-]` (e) `lst[-5:-3]` evaluates to

.`[٣٤- ,١ ,٢٠]` (f) `lst[-22:3]` evaluates to

.`[٣ ,١ ,٦٠ ,٨-]` (g) `lst[4:]` evaluates to

.`[٣ ,١ ,٦٠ ,٨- ,٤٠ ,٣٤- ,١ ,٢٠]` (h) `lst[:]` evaluates to

.`[٤٠ ,٣٤- ,١ ,٢٠]` (i) `lst[:4]` evaluates to

.`[٨- ,٤٠ ,٣٤- ,١]` (j) `lst[1:5]` evaluates to

.`(k) `-34 in lst` evaluates to `True

.`(l) `-34 not in lst` evaluates to `False

.`(m) `len(lst)` evaluates to `8

10. Write the list represented by each of the following expressions.
(a) [8] * 4
(b) 6 * [2, 7]
(c) [1, 2, 3] + ['a', 'b', 'c', 'd']
(d) 3 * [1, 2] + [4, 2]
(e) 3 * ([1, 2] + [4, 2])

:The lists represented by each of the following expressions are

`[٨ ,٨ ,٨ ,٨]` (a)

This expression creates a list with a single element `8`, and then multiplies it by `4` to create a new list .with four `8`s

`[٧ ,٢ ,٧ ,٢ ,٧ ,٢]` (b)

This expression creates a list with two elements `2` and `7`, and then multiplies it by `6` to create a new .list with alternating `2`s and `7`s

`(c) `[1, 2, 3, 'a', 'b', 'c', 'd']

.`This expression concatenates two lists: `[1, 2, 3]` and `['a', 'b', 'c', 'd']

`[٢ ,٤ ,٢ ,١ ,٢ ,١ ,٢ ,١]` (d)

This expression first creates a list with two elements `1` and `2`, and then multiplies it by `3` to create a .`[٢ ,٤]` new list with alternating `1`s and `2`s. It then concatenates this list with

`[٢ ,٤ ,٢ ,١ ,٢ ,٤ ,٢ ,١ ,٢ ,٤ ,٢ ,١]` (e)

This expression first creates a list with two elements `1` and `2`, and another list with two elements `4` and `2`. It then concatenates these two lists together to form a list `[1, 2, 4, 2]`. This list is then multiplied .by `3` to create a new list with repeating elements

11. Write the list represented by each of the following list comprehension expressions.
(a) [x + 1 for x in [2, 4, 6, 8]]
(b) [10*x for x in range(5, 10)]
(c) [x for x in range(10, 21) if x % 3 == 0] (d) [(x, y) for x in range(3) for y in range(4)]
(e) [(x, y) for x in range(3) for y in range(4) if (x + y) % 2 == 0]

:The lists represented by each of the following list comprehension expressions are

`[٩ ,٧ ,٥ ,٣]` (a)

.`[٨ ,٦ ,٤ ,٢]` This expression adds `1` to each element of the list

`[٩٠ ,٨٠ ,٧٠ ,٦٠ ,٥٠]` (b)

.`This expression multiplies each value in the range `range(5, 10)` by `10

`[١٨ ,١٥ ,١٢]` (c)

.`This expression creates a list of numbers in the range `range(10, 21)` that are divisible by `3

`[(٣ ,٢) ,(٢ ,٢) ,(١ ,٢) ,(٠ ,٢) ,(٣ ,١) ,(٢ ,١) ,(١ ,١) ,(٠ ,١) ,(٣ ,٠) ,(٢ ,٠) ,(١ ,٠) ,(٠ ,٠)]` (d)

This expression creates a list of all possible tuples `(x, y)` where `x` is in the range `range(3)` and `y` is in .`(٤)the range `range

`[(٢ ,٢) ,(٠ ,٢) ,(٣ ,١) ,(١ ,١) ,(٢ ,٠) ,(٠ ,٠)]` (e)

This expression creates a list of all possible tuples `(x, y)` where `x` is in the range `range(3)` and `y` is in the range `range(4)`, but only includes tuples where the sum of `x` and `y` is even.

12. Provide a list comprehension expression for each of the following lists.

(a) [1, 4, 9, 16, 25]
(b) [0.25, 0.5, 0.75, 1.0, 1.25. 1.5]
(c) [('a', 0), ('a', 1), ('a', 2), ('b', 0), ('b', 1), ('b', 2)]

:The list comprehension expressions for the following lists are

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

.`(۶ ,۱)This expression creates a list of squares of numbers in the range `range

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

.`This expression creates a list of numbers in the range `range(1, 7)` divided by `4

`(c) `[(x, y) for x in ['a', 'b'] for y in range(3)]

This expression creates a list of tuples where the first element of each tuple is a letter in the list `['a', 'b']`
.`(۳)and the second element is a number in the range `range

13. If lst is a list, what expression indicates whether or not x is a member of lst?

The expression `x in lst` indicates whether or not `x` is a member of the list `lst`. If `x` is in the list, this
.`expression evaluates to `True`. Otherwise, it evaluates to `False

14. What does reversed do?

reversed()` is a built-in Python function that returns a reverse iterator over the elements of a sequence. `
.When applied to a list, it returns a reverse iterator that allows you to traverse the list in reverse order

For example, if you have a list `my_list = [1, 2, 3, 4, 5]`, calling `reversed(my_list)` will return an iterator
.`[۱ ,۲ ,۳ ,۴ ,۵]` :that allows you to traverse the list in reverse order

Note that `reversed()` does not modify the original list, but instead returns a new object that can be used
.to access the elements of the original list in reverse order

15. Complete the following function that adds up all the *positive* values in a list of integers. For example, if list a contains the elements 3;-3;5;2;-1; and 2, the call sum_positive(a) would evaluate to 12, since 3+5+2+2 = 12. The function returns zero if the list is empty.

```
def sum_positive(a):
# Add your code...
```

:Here's the completed function that adds up all the positive values in a list of integers

python```

:def sum_positive(a)

total = 0 # initialize the total to zero

for num in a: # iterate over each element of the list

if num > 0: # check if the element is positive

total += num # add the positive element to the total

return total # return the final total

```

This function first initializes a variable `total` to zero, and then iterates over each element of the input list `a`. For each element that is positive, it adds that element to the current total. Finally, the function returns .the total of all positive elements. If the list is empty, the function returns zero

16. Complete the following function that counts the even numbers in a list of integers. For example, if list a contains the elements 3;5;4;-1; and 0, the call count_evens(a) would evaluate to 2, since a contains two even numbers: 4 and 0. The function returns zero if the list is empty. The function does not affect the contents of the list.
def count_evens(lst):
# Add your code...

:Here's the completed function that counts the even numbers in a list of integers

python```

:def count_evens(lst)

count = 0  # initialize the count to zero

for num in lst:  # iterate over each element of the list

if num % 2 == 0:  # check if the element is even

count += 1  # increment the count for each even element

return count  # return the final count

```

This function first initializes a variable `count` to zero, and then iterates over each element of the input list `lst`. For each element that is even (i.e., divisible by 2), it increments the current count. Finally, the .function returns the total count of all even elements. If the list is empty, the function returns zero

17. Write a function named print_big_enough that accepts two parameters, a list of numbers and a number. The function should print, in order, all the elements in the list that are at least as large as the second parameter.

:Here's the implementation of the `print_big_enough` function

python```

:def print_big_enough(lst, num)

:for element in lst

:if element >= num

print(element)

```

The `print_big_enough` function accepts two parameters: `lst`, which is a list of numbers, and `num`, which is a number. The function then iterates over each element in the list using a `for` loop. For each element, it checks if the element is greater than or equal to the second parameter `num`. If the element is at least as large as `num`, the function prints the element using the `print()` function.

18. Write a function named next_number that accepts a list of integer values. All the elements in the list are unique, and all elements in the list are greater than or equal to one. (The caller must ensure that these conditions are met before passing the list to next_number.) The next_number function should return the smallest positive integer **not** in the list. (Note that 1 is the smallest positive integer.)
As examples,
• next_number([5, 3, 1]) would return 2
• next_number([5, 4, 1, 2]) would return 3
• next_number([2, 3]) would return 1
• next_number([]) would return 1

:Here's the implementation of the `next_number` function

python```

:def next_number(lst)

lst.sort()  # sort the list in ascending order

smallest = 1  # initialize the smallest positive integer to 1

:for num in lst

if num == smallest:  # if the current number is equal to the smallest integer, increment the smallest integer

smallest += 1

return smallest  # return the smallest positive integer not in the list

```

The `next_number` function accepts a list of integer values as its parameter. The function first sorts the list in ascending order using the `sort()` method. It then initializes a variable `smallest` to `1`, which is the smallest positive integer. The function then iterates over each element in the sorted list using a `for` loop. For each element, it checks if the element is equal to the current value of `smallest`. If it is, it increments `smallest` by `1`. Finally, the function returns the smallest positive integer not in the list, which is the .`current value of `smallest`. If the input list is empty, the function returns `1

19. Write a function named reverse that reorders the contents of a list so they are reversed from their original order. a is a list. Note that your function must physically rearrange the elements within the list, not just print the elements in reverse order.

:Here's the implementation of the `reverse` function that reverses the contents of a list

python```

:def reverse(a)

left = 0 # initialize the left index to 0

right = len(a) - 1 # initialize the right index to the last element of the list

while left < right: # swap elements until the left and right indices meet in the middle

a[left], a[right] = a[right], a[left] # swap the elements at the left and right indices

left += 1 # increment the left index

right -= 1 # decrement the right index

```

The `reverse` function accepts a list `a` as its parameter. The function first initializes two variables: `left` to `0`, which is the index of the first element in the list, and `right` to `len(a) - 1`, which is the index of the last element in the list. The function then enters a `while` loop that swaps elements until the left and right indices meet in the middle of the list. For each iteration of the loop, it swaps the elements at the left and right indices using tuple unpacking. After each swap, it increments the left index and decrements the right index. Finally, when the loop terminates, the contents of the input list `a` are reversed. Note that this .function modifies the input list in place, rather than returning a new list

20. Write a Python program that creates the matrix
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
and assigns it to the variable m. Pretty print m to ensure the contents are correct. Next, reassign m[2][4] to 0, and print m again to ensure your code modified the correct element.

Here's the Python program that creates the matrix and modifies an element:

```python
# create the matrix
m = [[1 for j in range(9)] for i in range(6)]

# print the matrix
for row in m:
    print(row)

# modify an element of the matrix
m[2][4] = 0

# print the modified matrix
print()
for row in m:
    print(row)
```

This program first creates the matrix `m` using a nested list comprehension. The outer list comprehension creates a list of `6` sublists, and the inner list comprehension creates a sublist of `9` elements, each initialized to `1`. The program then uses a `for` loop to pretty print the matrix, which consists of `6` rows and `9` columns, with each element being `1`. The program then modifies an element of the matrix by assigning `0` to the element at row `2`, column `4` (which is 0-indexed). Finally, the program prints the modified matrix to ensure that the correct element was modified.
```

21. Provide five different ways to create the list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and assign it to the variable lst.
Here are five different ways to create the list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]` and assign it to the variable `lst`:

1. Using a list comprehension:

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

2. Using the `range()` function and the `list()` constructor:

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

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

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

4. Using the `extend()` method and a list of numbers:

```python
lst = []
lst.extend([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

5. Using the unpacking operator `*` and a tuple of numbers:

```python
lst = [*range(1, 11)]
```

All of these methods will create the same list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]` and assign it to the variable `lst`.

22. In a square 2D list the number of rows equals the nnumber of columns. Write a function that accepts a square 2D list and returns True if the left to right contents of any row equals the top to bottom contents of any column. If no row matches any column, the function returns False.
Here's the implementation of the function that checks if any row matches any column in a square 2D list:

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

The `check_match` function accepts a square 2D list `lst` as its parameter. The function first gets the length of the list `n` using the `len()` function. It then iterates over each row of the list using a `for` loop. For each row, it calculates the sum of its elements using the `sum()` function and assigns it to the variable `row_sum`. It then calculates the sum of the corresponding column elements by iterating over all rows

and selecting the element at the current row and column index. The sum is assigned to the variable `col_sum`. If `row_sum` is equal to `col_sum`, it means that the left to right contents of the current row matches the top to bottom contents of the current column, so the function returns `True`. If no row matches any column, the function returns `False`.

23. We can represent a Tic-Tac-Toe board as a 3 × 3 grid in which each position can hold one of the following three strings: "X", "O", or " ". Write a function named check_winner that accepts a 3 × 3 list as a parameter. If "X" appears in a winning Tic-Tac-Toe pattern, the function should return the string "X". If "O" appears in a winning Tic-Tac-Toe pattern, the function should return the string "O". If no winning pattern exists, the function should return the string " ".

Here's the implementation of the `check_winner` function that checks if there is a winner in a Tic-Tac-Toe board:

```python
def check_winner(board):
    for i in range(3):
        # check rows
        if board[i][0] == board[i][1] == board[i][2] and board[i][0] != " ":
            return board[i][0]
        # check columns
        if board[0][i] == board[1][i] == board[2][i] and board[0][i] != " ":
            return board[0][i]
    # check diagonals
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != " ":
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != " ":
        return board[0][2]
    # no winner
    return " "
```

The `check_winner` function accepts a 3 × 3 list `board` as its parameter. The function first iterates over each row and column of the list using a `for` loop. For each row and column, it checks if all three elements are equal and not equal to `" "`. If this condition is true, it means that there is a winning Tic-Tac-Toe pattern in that row or column, so the function returns the winning string `"X"` or `"O"`. The function then checks the two diagonals of the list using `if` statements. If either diagonal has all three equal elements that are not equal to `" "`, it returns the winning string `"X"` or `"O"`. If no winning pattern exists, the function returns the string `" "`.