HW-WEEK5-#BaharSohrabi

Chapter10

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

2. If you attempt to access an element of a list using a negative index in python, it will return the element starting from the end of the list. For example, if a list has 5 elements and you access the element at index -1, it will return the last element of the list. Similarly, if you access an element at index -2, it will return the second last element of the list and so on. However, if you try to access an element with an index lower than the negative length of the list, it will raise an "IndexError: list index out of range" error.

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

list1 = [45, -3, 16, 8]

4.

(a) lst[0]

(b) lst[-1]

(c) 10

(d) 29

(e) -4

(f) 29

(g) 10

(h) Illegal.

5.

(a) lst0 = 3

(b) lst3 = 5

(c) lstx = lst2 = 1

(d) lst-x = lst-2 = 5

(e) lstx + 1 = lst3 = 5

(f) lstx + 1 = 1 + 1 = 2

(g) lstlst[x] = lstlst[2] = lst1 = 0

(h) lstlst[lst[x]] = lstlst[lst[2]] = lstlst[1] = lst0 = 3

6. The len() function returns the number of elements in a list.

7. The expression representing an empty list is simply a pair of square brackets with no elements inside: [].

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, 40, -8

(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

9.

Original List	Target List	m	n
[2, 4, 6, 8, 10]	[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]	0	5
[2, 4, 6, 8, 10]	[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10]	-6	-1
[2, 4, 6, 8, 10]	[2, 3, 4, 5, 6, 7, 8, 10]		
[2, 4, 6, 8, 10]	[2, 4, 6, 'a', 'b', 'c', 8, 10]		
[2, 4, 6, 8, 10]	[2, 4, 6, 8, 10]	0	5
[2, 4, 6, 8, 10]	[]		
[2, 4, 6, 8, 10]	[10, 8, 6, 4, 2]	-1	-6
[2, 4, 6, 8, 10]	[2, 4, 6]	0	3
[2, 4, 6, 8, 10]	[6, 8, 10]	2	5
[2, 4, 6, 8, 10]	[2, 10]		
[2, 4, 6, 8, 10]	[4, 6, 8]	1	4

10.

(a) [8] * 4= [8, 8, 8, 8]

(b) 6 * [2, 7]= [2, 7, 2, 7, 2, 7, 2, 7, 2, 7, 2, 7]

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

(d) 3 * [1, 2] + [4, 2]= [1, 2, 1, 2, 1, 2, 4, 2]

(e) 3 * ([1, 2] + [4, 2])= [1, 2, 4, 2, 1, 2, 4, 2, 1, 2, 4, 2]

11.

a>>> [x + 1 for x in [2, 4, 6, 8]]

[3, 5, 7, 9]

b>>> [10*x for x in range(5, 10)]

[50, 60, 70, 80, 90]

c>>> [x for x in range(10, 21) if x % 3 == 0]

[12, 15, 18]

d>>> [(x, y) for x in range(3) for y in range(4)]

[(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>>> [(x, y) for x in range(3) for y in range(4) if (x + y) % 2 == 0]

[(0, 0), (0, 2), (1, 1), (1, 3), (2, 0), (2, 2)]

12.

(a) [1, 4, 9, 16, 25]=

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

(b) [0.25, 0.5, 0.75, 1.0, 1.25. 1.5]=

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

(c) [('a', 0), ('a', 1), ('a', 2), ('b', 0), ('b', 1), ('b', 2)]=

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

13. The expression to check if x is a member of lst in Python is

x in lst

This will return True if x is in lst, and False otherwise.

14. The reversed function in programming allows you to iterate through a sequence of values in reverse order. For example, if you have a list of elements, you can use the reversed() function to loop through the list in reverse order. The reversed() function returns a reverse iterator that allows you to iterate through the values in the sequence from the last element to the first. This is useful if you need to perform some operation on elements in reverse order, such as printing them in reverse, or reversing a string or list.

15.

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

```
a = [3,-3,5,2,-1,2]
print(sum_positive(a))

b = []
print(sum_positive(b))
```

16.

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

a = [3, 5, 4, -1, 0]
print(count_evens(a))

b = []
print(count_evens(b))
```

17. Here's the code for the function:

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

Here's an example of how to use the function:

```
my_list = [1, 5, 3, 8, 6]
print_big_enough(my_list, 5)
```

Output:

```
5
8
6
```

18.

```
def nextnumber(lst):
    if not lst or 1 not in lst:
        return 1
```

```
    maxval = max(lst)
    for i in range(1, maxval + 2):
        if i not in lst:
            return i

# Testing the function with the given examples
print(nextnumber([5, 3, 1]))  # should return 2
print(nextnumber(5, 4, 1, 2))  # should return 3
print(nextnumber([2, 3]))  # should return 1
print(nextnumber())  # should return 1
```

19.

```
def reverse(a):
    """
    Reverses the order of elements in a list.
    Modifies the list in place (i.e., no return value).
    """
    n = len(a)
    for i in range(n // 2):
        j = n - i - 1
        a[i], a[j] = a[j], a[i]
```

To test the function, you can create a list of integers, call the function with that list as argument, and then print the list to see if it was reversed correctly:

```
>>> a = [1, 2, 3, 4, 5]
>>> print(a)
[1, 2, 3, 4, 5]

>>> reverse(a)
>>> print(a)
[5, 4, 3, 2, 1]
```

20.

```
m = [[1]*9 for i in range(6)]
print("Initial matrix: ")
for row in m:
    print(row)

m[2][4] = 0
print("\nModified matrix: ")
for row in m:
```

```
    print(row)
```

Output:

Initial 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]

Modified matrix:

[1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 0, 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]

21.

1. Using a list comprehension:

```
lst = x for x 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 loop:

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

4. Using the extend method with a range object:

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

5. Manually creating the list:

```
lst = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

22.

```
def has_matching_row_and_column(square_list):
    for row in square_list:
        for j in range(len(row)):
```

```
        column = [square_list[i][j] for i in range(len(row))]
        if row == column:
            return True

    return False
```

The function loops over all rows in the list using a for loop, and for each row it creates a new list column containing the elements of the corresponding column using a list comprehension. It then compares the row and the column using the == operator, and if they are equal it returns True. If no match is found after checking all rows and columns, the function returns False.

23.

```
def checkwinner(board):
    for row in board:
        if row == "X" "X" "X":
            return "X"
        if row == "O" "O" "O":
            return "O"
```

```
    for i in range(3):
        if board0i == board1i == board2i == "X":
            return "X"
        if board0i == board1i == board2i == "O":
            return "O"

    if board00 == board11 == board22 == "X":
        return "X"
    if board00 == board11 == board22 == "O":
        return "O"
    if board02 == board11 == board20 == "X":
        return "X"
    if board02 == board11 == board20 == "O":
        return "O"

    return " "
```

The function first checks each row for a winning pattern. If one is found, it returns the corresponding player ("X" or "O"). If no winner is found in the rows, the function proceeds to check the columns and then the diagonals, returning the player if a winning pattern is detected. If no winner is found, the function

returns a blank space. Note that the function only checks for a winning pattern starting from the top-left corner and going to the bottom-right corner; there could be other winning patterns that the function doesn't detect.

The End>.<