**ASSIGNMENT 4:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 1. What exactly is []?**

**Answer 1 :**

[] is List a sequence data type used to store a collection of data.

[] is an empty List which does not have any values or objects.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 2. In a list of values stored in a variable called spam, how would you assign the value 'hello' as the third value? (Assume [2, 4, 6, 8, 10] are in spam.)**

**Let's pretend the spam includes the list ['a', 'b', 'c', 'd'] for the next three queries.**

**Answer 2 :**

Answer is spam[2]

**Solution as below:**

spam = [2,4,6,8,10]

spam[2]='hello'

spam

**Output :**

[2, 4, 'hello', 8, 10]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 3. What is the value of spam[int(int('3' \* 2) / 11)]?**

**Answer 3:**

Answer is 8

**Solution as below:**

‘ 3’ \* 2 = ‘33’

int(‘33’) = 33

int(int('3' \* 2) / 11)= 3

spam[int(int('3' \* 2) / 11)] = spam[3] = 8

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 4. What is the value of spam[-1]?**

**Answer 4:**

Answer is 10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 5. What is the value of spam[:2]?**

**Let's pretend bacon has the list [3.14, 'cat,' 11, 'cat,' True] for the next three questions.**

**Answer 5:**

Answer is [2, 4] or [3.14, ‘cat]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 6. What is the value of bacon.index('cat')?**

**Answer 6:**

Answer is index 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 7. How does bacon.append(99) change the look of the list value in bacon?**

**Answer 7:**

append() add the value in the list after True in the rightmost position.

List will be updates as **[3.14, 'cat', 11, 'cat', True, 99]**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 8. How does bacon.remove('cat') change the look of the list in bacon?**

**Answer 8:**

remove () is used to remove data from a list. This bacon.remove('cat') removes the first ‘cat’ value from the list however there is an appearance of ‘cat’ two times in the list.

List will be updated as [3.14, 11, 'cat', True, 99]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 9. What are the list concatenation and list replication operators?**

**Answer 9:**

1. List Concatenation means obtaining a new string by combining two or more original lists by using append() , or by using ‘ + ‘ Operator, or by using extend().

List Concatenation operator is ‘ + ’ operator.

**Example 1 :**

L1 = [1,2,3,4,5,6]

L2 = [7,8,9,10,11,12]

L3 = [11,12,13,14,15,16]

L4 = L1 + L2 + L3

L4

**Output 1 :**

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 11, 12, 13, 14, 15, 16]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Example 2 :**

L1 = [1,2,3,4,5,6]

L2 = [7,8,9,10,11,12]

for i in L2:

L1.append(i)

L1

**Output 2 :**

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Example 3 :**

L1 = [1,2,3,4,5,6]

L2 = [7,8,9,10,11,12]

L1.extend(L2)

L1

**Output 3 :**

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Example 4 :**

L1 = [1,2,3,4,5,6]

L2 = [7,8,9,10,11,12]

L= [\*L1, \*L2]

L

**Output 4 :**

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

1. List Replication means duplication or the exact same copy of List.

List Replication operator is ‘ \* ’ operator.

**Example 4 :**

L1 = [1,2,3,4,5,6]

L = L1 \* 2

L

**Output 4 :**

[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 10. What is the difference between the list methods append() and insert()?**

**Answer 10:**

**append() :**

Append is a function to insert a single element in an existing list at the end of list data

**Example 1:**

List = ['a','b','c','d','e','f']

List.append('10')

List

**Output 1:**

['a', 'b', 'c', 'd', 'e', 'f', '10']

**insert():**

Insert is function to insert given element at given index in list

**Example 1:**

List = ['a', 'b', 'c', 'd', 'e', 'f', '10']

List.insert(7,'G')

List

**Output 1:**

['a', 'b', 'c', 'd', 'e', 'f', '10', 'G']

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 11. What are the two methods for removing items from a list?**

**Answer 11 :**

There are three methods to remove items from the list.

**1). remove() :**

Remove function is used to remove or delete values or objects from existing list using corresponding value or object

**#Example 1 :**

**List = [1,2,3,4,5,6]**

**## remove object or value 3 from List**

**List.remove(3)**

**List**

**#Output 1 :**

**[1, 2, 4, 5, 6]**

**2). del :**

Del is used to remove or delete values or objects from existing list using index of corresponding value or object

**#Example 1 :**

**List = [1,2,3,4,5,6]**

**## delete object or value having index 3 from List**

**del List[3]**

**List**

**#Output 1 :**

**[1, 2, 3, 5, 6]**

**3). pop() :**

Pop function is used to remove or delete values or objects from existing list using index of corresponding value or object

**#Example 1 :**

**List = [1,2,3,4,5,6]**

**## delete object or value having index 5 from List**

**List.pop(5)**

**List**

**#Output 1 :**

**[1, 2, 3, 4, 5]**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 12. Describe how list values and string values are identical.**

**Answer 12:**

Each value in List as well as String has their own position i. e Index.

Each value in List and string can be accessed using indexing.

Both List values and string values are iterables.

Concatenation and Replication for both List and string is possible.

List values are mutable . It can be replaced

String values are not mutable. It can not be replaced.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 13. What's the difference between tuples and lists?**

**Answer 13:**

| **List :** | **Tuple :** |
| --- | --- |
| It is a comma separated sequence of values enclosed in square brackets [ ]. | It is a comma separated sequence of values enclosed in parenthesis ( ). |
| It is mutable. Its values can be changed or replaced. | It is not mutable. Its values can not be changed or replaced. |
|  |  |
|  |  |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 14. How do you type a tuple value that only contains the integer 42?**

**Answer 14:**

(42)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 15. How do you get a list value's tuple form? How do you get a tuple value's list form?**

**Answer 15**

**Convert list values into tuple form : Use tuple()**

**Example :**

List = [ 1,2,3,4 ]

Tuple = tuple( List )

Tuple

**Output:**

( 1,2,3,4 )

**Convert tuple into List form : Use list()**

**Example :**

tuple = ( 1,2,3,4 )

List = list( tuple )

List

**Output:**

[ 1,2,3,4 ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 16. Variables that "contain" list values are not necessarily lists themselves. Instead, what do they contain?**

**Answer 16 :**

“ contain “ is string data type value which is refer as list value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Question 17. How do you distinguish between copy.copy() and copy.deepcopy()?**

**Answer 17 :**

copy.copy() and copy.deepcopy() both are functions of module ‘ copy ‘.

**Shallow Copy ( copy.copy() ) -**

A shallow copy is the construction of a new collection object (like our list / dict / set) followed by adding references to the child objects from the original object.

Shallow copies are just one level deep. The shallow copying process is not recursive, meaning the copies of child objects will not be created.

A shallow copy creates a new object which stores the reference of the original elements.

Now, this sounds complex, but the crux is that in the case of shallow copy if we make a change to the copy of the original object, it will reflect in the original object as well.

**Deep Copy ( copy.deepcopy() )**  -

Deep copy is the opposite of a shallow copy. In this case, the deep copy process is recursive.

Like in shallow copy, we create a new collection object and recursively add copies of the child objects from the original.

A deep copy creates a new object and recursively adds the copies of nested objects present in the original elements.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*