# PYTHON ASSIGNMENT – 4

## QUESTIONS ;

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

**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.**

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

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

**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.**

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

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

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

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

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

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

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

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

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

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

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

**contain?**

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

## SOLUTIONS:

**1.[]** is an empty list in Python. It represents a collection of items with no elements.

2. To assign the value 'hello' as the third value in the list stored in the variable **spam**, you can use the following code:

```
spam = [2, 4, 6, 8, 10]
spam[2] = 'hello'
```

After this, the **spam** list will be **[2, 4, 'hello', 8, 10]**.

3. The value of **spam[int(int('3' * 2) / 11)]** can be calculated as follows:

- **'3' * 2** results in the string **'33'**.
- **int('33')** converts the string to an integer, which is **33**.
- **33 / 11** results in **3.0**.
- **int(3.0)** converts the float to an integer, which is **3**.
- Therefore, the expression is equivalent to **spam[3]**, which corresponds to the fourth element of the list (indexing starts from 0).

4. The value of **spam[-1]** is the last element of the list. In this case, it would be **10**.

5.The value of **spam[:2]** is a sublist containing the first two elements of the list. In this case, it would be **[2, 4]**.

6.The value of **bacon.index('cat')** is the index of the first occurrence of the string 'cat' in the list **bacon**. In this case, it would be **1**.

7.The **bacon.append(99)** method adds the value **99** to the end of the list **bacon**. After this operation, **bacon** will be **[3.14, 'cat', 11, 'cat', True, 99]**.

8.The **bacon.remove('cat')** method removes the first occurrence of the string 'cat' from the list **bacon**. After this operation, **bacon** will be **[3.14, 11, 'cat', True, 99]**.

9.The list concatenation operator is **+**, and the list replication operator is *****.

10The **append()** method adds an element to the end of the list, while the **insert()** method inserts an element at a specified index in the list.

11.The two methods for removing items from a list are **remove()** and **pop()**. The **remove()** method removes the first occurrence of a specified value, and the **pop()** method removes

an element at a specified index (or the last element if no index is provided).

12.Both list values and string values are ordered sequences of elements. They support indexing, slicing, and iteration.

13.Tuples and lists are both data structures in Python, but the key difference is that tuples are immutable, while lists are mutable. Once a tuple is created, its elements cannot be changed, added, or removed. Lists, on the other hand, can be modified after creation.

14.To type a tuple value that only contains the integer 42, you would use parentheses:

```
my_tuple = (42,)
```

15.To get a list value's tuple form, you can use the **tuple()** constructor, and to get a tuple value's list form, you can use the **list()** constructor. For example:

```
my_list = [1, 2, 3]

my_tuple = tuple(my_list)  # Converts list to tuple

new_list = list(my_tuple)  # Converts tuple to list
```

16.Variables that "contain" list values are not necessarily lists themselves. Instead, they contain references or pointers to the actual list objects. If you assign one list to another variable and modify the list through one variable, the other variable will reflect those changes because both variables point to the same list object.

17. The **copy.copy()** function creates a shallow copy of a list, which means it creates a new list object but does not create new objects for the elements within the list. Changes to mutable elements (e.g., lists or other objects) within the copied list will be reflected in both the original and copied lists. On the other hand, **copy.deepcopy()** creates a deep copy, meaning it creates a new list and new objects for all nested elements, ensuring that changes to nested elements do not

affect the original list or vice versa. It is used when dealing with nested structures or complex objects to avoid unintended side effects.