**Assignment 4**

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. What exactly is []?**

[] represents an **empty list** in Python. Lists are used to store multiple items in a single variable, and an empty list contains no elements.

**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.)**

To assign 'hello' as the third value in the list spam, you can access the third position by its index (which is 2, since indexing starts at 0) and assign the new value:

python

Copy code

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

spam[2] = 'hello'

After this, spam would be [2, 4, 'hello', 8, 10].

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

Let's break down this expression:

* '3' \* 2 produces '33' (the string '3' repeated twice).
* int('33') converts the string '33' to the integer 33.
* 33 / 11 equals 3.0, and int(3.0) is 3.

So, this expression is equivalent to spam[3].

Assuming spam = ['a', 'b', 'c', 'd'], spam[3] would be 'd'.

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

In Python, -1 refers to the last element in a list.

For spam = ['a', 'b', 'c', 'd'], spam[-1] is 'd'.

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

spam[:2] returns a slice of the list spam from the beginning up to, but not including, index 2.

For spam = ['a', 'b', 'c', 'd'], spam[:2] would be ['a', 'b'].

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

The index() method returns the first index at which the specified element appears.

For bacon = [3.14, 'cat', 11, 'cat', True], bacon.index('cat') would return 1, as 'cat' first appears at index 1.

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

The append() method adds an element to the end of the list.

If bacon = [3.14, 'cat', 11, 'cat', True], then after bacon.append(99), bacon would be [3.14, 'cat', 11, 'cat', True, 99].

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

The remove() method removes the first occurrence of the specified value.

If bacon = [3.14, 'cat', 11, 'cat', True], then after bacon.remove('cat'), bacon would be [3.14, 11, 'cat', True].

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

* **Concatenation**: The + operator combines two lists into one.

python

Copy code

[1, 2] + [3, 4] # Results in [1, 2, 3, 4]

* **Replication**: The \* operator repeats the list a specified number of times.

python

Copy code

[1, 2] \* 3 # Results in [1, 2, 1, 2, 1, 2]

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

* append() adds an item to the **end** of the list.
* insert(index, value) adds an item at a specified **index** in the list.

Example:

python

Copy code

lst = [1, 2, 3]

lst.append(4) # Results in [1, 2, 3, 4]

lst.insert(1, 'a') # Results in [1, 'a', 2, 3, 4]

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

* **remove(value)**: Removes the first occurrence of a value.
* **pop(index)**: Removes the item at the specified index. If no index is specified, it removes the last item.

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

Lists and strings are both **sequences** in Python, meaning:

* Both are indexed, so you can access elements by position.
* Both support slicing, allowing for subsetting.
* Both can be looped over using for loops.

The main difference is that lists are mutable, whereas strings are immutable.

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

* **Lists** are mutable, meaning you can change their contents.
* **Tuples** are immutable, meaning once created, their contents cannot be changed.
* Lists are defined with square brackets ([]), while tuples are defined with parentheses (()).

Example:

python

Copy code

lst = [1, 2, 3] # List

tup = (1, 2, 3) # Tuple

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

To define a single-value tuple, add a comma after the value:

python

Copy code

single\_value\_tuple = (42,)

Without the comma, Python would interpret it as an integer, not a tuple.

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

* To convert a list to a tuple: tuple(list\_variable)
* To convert a tuple to a list: list(tuple\_variable)

Example:

python

Copy code

lst = [1, 2, 3]

tup = (4, 5, 6)

tuple\_form = tuple(lst) # (1, 2, 3)

list\_form = list(tup) # [4, 5, 6]

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

Variables contain **references** to list values, not the actual list itself. This means the variable points to a location in memory where the list is stored.

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

* copy.copy() creates a **shallow copy** of an object, meaning it only copies the object itself, not the nested objects inside it. If the original object has nested objects (like lists within lists), changes to nested objects in the copy will reflect in the original.
* copy.deepcopy() creates a **deep copy** of an object, meaning it copies the object and all nested objects, so changes to the copy do not affect the original at any level.

Example:

python

Copy code

import copy

lst = [[1, 2], [3, 4]]

shallow\_copy = copy.copy(lst)

deep\_copy = copy.deepcopy(lst)

lst[0][0] = 'changed'

print(shallow\_copy) # [[ 'changed', 2], [3, 4]]

print(deep\_copy) # [[1, 2], [3, 4]]

In this example, shallow\_copy reflects the change, while deep\_copy does not.