Python_Basic_assignment_4

August 7, 2023

1. What exactly is []?

```
[ ]: [] is represent empty list or empty array in python
```

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.)

```
[1]: # solution by changing the value in index 3
     spam = [2,4,6,8,10]
     spam[2]='Hello'
     spam
```

```
[1]: [2, 4, 'Hello', 8, 10]
```

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)]?

```
[3]: spam = ['a','b','c','d']
     spam[int(int('3'*2)/11)] #spam[33/11] = spam[3]
```

```
[3]: 'd'
```

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

```
[ ]: spam = ['a','b','c','d']
     spam[-1] #negative indexing
```

```
[ ]: 'd'
```

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

```
[9]: spam = ['a','b','c','d']
     spam[:2]
```

```
[9]: ['a', 'b']
```

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')?

```
[13]: bacon = [3.14, 'cat', 11, 'cat', True]
      bacon.index('cat') # it returns the index of first occourance of 'cat'
```

```
[13]: 1
```

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

```
[16]: bacon = [3.14, 'cat', 11, 'cat', True]
      bacon.append(99) # append adds the item at the end of the list
      bacon
```

```
[16]: [3.14, 'cat', 11, 'cat', True, 99]
```

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

```
[18]: bacon = [3.14, 'cat', 11, 'cat', True]
      bacon.remove('cat') # remove first occurance of item
      bacon
```

```
[18]: [3.14, 11, 'cat', True]
```

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

```
[ ]: ( * ) is list replication operator ( + ) is list concatination operator
```

```
[19]: l1 = [3,4]
      l2 = [5,7]
      # list concatination
      l1 + l2
```

```
[19]: [3, 4, 5, 7]
```

```
[21]: l3 = [8,0]
      # list replication
      l3*4
```

```
[21]: [8, 0, 8, 0, 8, 0, 8, 0]
```

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

```
[ ]: append() appends the object at the end of the list
     insert() inset object before index
```

```
[22]: bacon = [3.14, 'cat', 11, 'cat', True]
      bacon.append(99) # append adds the item at the end of the list
      bacon
```

```
[22]: [3.14, 'cat', 11, 'cat', True, 99]
```

[25]:
```
bacon = [3.14, 'cat', 11, 'cat', True]
bacon.insert(4,'Pankaj') # inserting value at 4th index
bacon
```

[25]:
```
[3.14, 'cat', 11, 'cat', 'Pankaj', True]
```

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

[26]:
```
#remove(item) - remove first occurance of an item.
bacon = [3.14, 'cat', 11, 'cat', True]
bacon.remove('cat')
bacon
```

[26]:
```
[3.14, 11, 'cat', True]
```

[28]:
```
#pop() - Remove and returns item at index default last.
bacon = [3.14, 'cat', 11, 'cat', True]
bacon.pop()
bacon
```

[28]:
```
[3.14, 'cat', 11, 'cat']
```

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

[ ]:
```
1. Both lists and strings can be passed to len()
2. Have indexes and slices
3. Can be used in for loops
4. Can be concatenated or replicated
5. Can be used with the in and not in operators
```

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

[ ]:
```
List: List are mutable.
      They can have values added, removed,or changed.
      List written in  the square brackets []

Tuple: Tuples are immutable in nature.
       They cannot be changed at all.
       tuples are written in prentheses ()
```

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

[31]:
```
my_tuple = (42,)
my_tuple
```

[31]:
```
(42,)
```

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

[ ]:
```
l1 = [5,7]
l2 = tuple(l1)
l2
```

[38]:
```
t1 = (4,5)
t2 = list(t1)
t2
```

[38]:
```
[4, 5]
```

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

[ ]:
```
 They contain references to list values
```

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

[ ]:
```
copy.copy() creates a shallow copy of an object, while copy.deepcopy() creates␣
 ↪a deep copy of an object.
```