1. What exactly is []?

[] is an empty **List**. Square braces will be used to represent **List** in python.

```
In [1]:  1  list1 = []

In [2]:  1  type(list1)

Out[2]: list
```

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.)

```
In [1]:  1  spam = [2, 4, 6, 8, 10]
         2
         3  print(spam[2])  # identifying the third value
         4
         5  spam[2] = "hello" # assigning the string to third value
         6
         7  spam

         6

Out[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)]?

```
In [1]:  1  spam = ['a', 'b', 'c', 'd']
         2  spam[int(int('3' * 2) / 11)]

Out[1]: 'd'
```

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

```
In [2]:  1  spam = ['a', 'b', 'c', 'd']
         2  spam[-1]

Out[2]: 'd'
```

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

```
In [3]:
1 spam = ['a', 'b', 'c', 'd']
2 spam[:2]

Out[3]: ['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')?

```
In [1]:
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.index('cat')

Out[1]: 1
```

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

```
In [2]:
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.append(99)
3 bacon

Out[2]: [3.14, 'cat', 11, 'cat', True, 99]
```

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

```
In [3]:
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.remove('cat')
3 bacon

Out[3]: [3.14, 11, 'cat', True]
```

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

Plus(+) is the **list concatenation** operator.

```
In [1]:
1 spam = ['a', 'b', 'c', 'd']
2 bacon = [3.14, 'cat' ,11, 'cat', True]
3 # plus(+) is the concatination operator.
4 spam + bacon

Out[1]: ['a', 'b', 'c', 'd', 3.14, 'cat', 11, 'cat', True]
```

Asterik(*) is the **list replication** operator.

```
In [2]:
1 spam = ['a', 'b', 'c', 'd']
2 #Asterik(*) is the list replication operator.
3 spam * 3

Out[2]: ['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']
```

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

append() adds an element **at the end** of the existing list.

```
In [1]:
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.append(99)
3 bacon

Out[1]: [3.14, 'cat', 11, 'cat', True, 99]
```

insert() adds an element **at the specific index** of the existing list.

```
In [2]:
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.insert(2,99)
3 bacon

Out[2]: [3.14, 'cat', 99, 11, 'cat', True]
```

```
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.insert(2,99)
3 baco

Signature: bacon.insert(index, object, /)
Docstring: Insert object before index.
Type:      builtin_function_or_method
```

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

i. using **remove()**

```
In [1]:
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.remove(11)
3 bacon

Out[1]: [3.14, 'cat', 'cat', True]
```

```
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.remove(11)
```

```
Signature: bacon.remove(value, /)
Docstring:
Remove first occurrence of value.

Raises ValueError if the value is not present.
Type:      builtin_function_or_method
```

ii. using **pop()**

```
In [2]: 1 bacon = [3.14, 'cat' ,11, 'cat', True]
        2 bacon.pop(2)
        3 bacon

Out[2]: [3.14, 'cat', 'cat', True]
```

```
1 bacon = [3.14, 'cat' ,11, 'cat', True]
2 bacon.pop(2)
```

```
Signature: bacon.pop(index=-1, /)
Docstring:
Remove and return item at index (default last).

Raises IndexError if list is empty or index is out of range.
Type:      builtin_function_or_method
```

iii. using **del**

```
In [3]: 1 bacon = [3.14, 'cat' ,11, 'cat', True]
        2 del bacon[2]
        3 bacon

Out[3]: [3.14, 'cat', 'cat', True]
```

Syntax : del list_Name[index]

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

List values and String values have similar **indexing.**

```
In [1]:
1  _list = ["a","b","c","d","e"]
2  for i in _list:
3      print(i, _list.index(i))
```

```
a 0
b 1
c 2
d 3
e 4
```

```
In [2]:
1  _string = "abcde"
2  for i in _list:
3      print(i, _string.index(i))
```

```
a 0
b 1
c 2
d 3
e 4
```

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

LIST	TUPLE
List is mutable	Tuple is Immutable
Implication of Iteration is Time-Consuming.	Implication of iterations is comparatively Faster.
The List is better for performing operations, such as insertion and deletion.	Tuples consume less memory as compared to the list
Lists have several built-in methods	Tuple does not have many built-in methods
The unexpected changes and errors are more likely to occur	In tuple, unexpected changes and errors are hard to take place

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

```
In [2]:
1  _tuple = (42)
2  _tuple
```

```
Out[2]: 42
```

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

```
In [1]:
1 # Tuple converted to List
2 _tuple = ("a","b","c","d","e")
3 list(_tuple)

Out[1]: ['a', 'b', 'c', 'd', 'e']

In [2]:
1 # List converted to Tuple
2 _list = ["a","b","c","d","e"]
3 tuple(_list)

Out[2]: ('a', 'b', 'c', 'd', 'e')
```

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

```
In [1]:
1 variable =  'cat' ,11, 'cat', True, [1,2,3], [5,6,7] , 3.14
2 print(type(variable))
3 print(type([1,2,3]))
4 print(type([5,6,7]))

<class 'tuple'>
<class 'list'>
<class 'list'>
```

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

copy.copy() is a **Shallow** Copy.

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

In **Shallow copy**, the **changes** made to the **“Copied Object(s)”** will **get reflected** to the **“Original Object(s)”**.

In **Deep copy**, the **changes** made to the **“Copied Object(s)”** will **not get reflected** to the **“Original Object(s)”.**