1. What exactly is []?

Ans. The square brackets are used to create a list in python. Eg. L = [1,2,3,4,5,6]

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.)

```
l = [2, 4, 6, 8, 10]
```

```
l[2]='hello'
```

```
l
```

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

Ans.

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)]?

Ans. 'd'

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

Ans. 'd'

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

Ans. ['a','b','c']

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')?

Ans. 1

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

```
bacon.append(99)
```

```
bacon
```

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

Ans.

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

```
bacon.remove('cat')
```

```
bacon
```

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

Ans.

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

```
l = [1,2,3,4,5]
l1 =[1,2,3,4,5]
```

```
l2= l + l1 #list concatenation operator '+'
```

```
l2
```

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

```
l*3 #list replication operator '*'
```

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

Ans.

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

Ans.

```
l
```

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

```
l.append("Prince")  #append will insert the object at the end of the list
```

```
l
```

```
[1, 2, 3, 4, 5, 'Prince']
```

```
l.insert(1,"Francis") #insert will insert object at any index preferred
```

```
l
```

```
[1, 'Francis', 2, 3, 4, 5, 'Prince']
```

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

```
l
```

```
[1, 'Francis', 2, 3, 4, 5]
```

```
l.pop()   #pop method
l
```

```
[1, 'Francis', 2, 3, 4]
```

```
l.remove("Francis")  #remove method
```

```
l
```

```
[1, 2, 3, 4]
```

Ans.

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

Ans. list and string values can be iterated through. They are iterables.

```
l=[1, 2, 3, 4, 5, 6, 7]
s = "Prince"
```

```
for i in l:
    print(i)
```

```
1
2
3
4
5
6
7
```

```
for i in s:
    print(i)
```

```
P
r
i
n
c
e
```

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

Ans. Tuples are immutable whereas lists are mutable.

```
l = [1, 2, 3, 4, 5, 6, 7]
t = (1,2,3,4,5,6,7)
```

```
l[3]=100
```

```
l
```

```
[1, 2, 3, 100, 5, 6, 7]
```

```
t[3]=100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-3502da552cce> in <module>
----> 1 t[3]=100

TypeError: 'tuple' object does not support item assignment
```

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

Ans.

```
t =(42,)
type(t)
```

```
tuple
```

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

Ans. We have to use Type casting to do that.

```
l = ["Prince","Francis","Annies"]
l = tuple(l)
type(l)
```

```
tuple
```

```
t = ("Prince","Francis","Annies")
t = list(t)
type(t)
```

```
list
```

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

Ans.

```
v = (1,2,3,4,[1,"sdf",34],"Prince")
s = {1:[1,2,3,4,5], "z":(1,"ouoi",232)}
```

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

Ans. shall copy creates a new compound object however it will not copy the the child objects to the new compound object instead will reference the child objects to the new object. A change in the new compound objects will make a change in the original objects.

```
import copy
l1 = [1,2,3,[50,60],5,6]
```

```
l2 = copy.copy(l1)
```

```
l2[3][1]= 50
```

```
l2  #after shallow copying
```

```
[1, 2, 3, [50, 50], 5, 6]
```

```
l1   # original list after shallow copying
```

```
[1, 2, 3, [50, 50], 5, 6]
```

Deep copy creates a new compound object and one by one copies each child object to the new compound object. A change in new child objects will not impact or change the values in the original object.

```
l1  = [1,2,3,[50,60],5,6]
```

```
l2 = copy.deepcopy(l1)
```

```
l2[3][0]= 100
```

```
l2  #after deep copying
```

```
[1, 2, 3, [100, 60], 5, 6]
```

```
l1 #after deep copying
```

```
[1, 2, 3, [50, 60], 5, 6]
```