1. What exactly is []?
   Solution 1: It is an empty list i.e. a list with no values.

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.)
   Solution 2: spam = [2,4,6,8,10]
   spam[2] = “hello”

   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)]?
   Solution 3: ‘d’ . Here'3' * 2 will be'33', which is passed to int()and will be 33 (string value converted to int) then divied by 11 . This eventually evaluates to 3.

4. What is the value of spam[-1]?
   Solution 4: ‘d’

5. What is the value of spam[:2]?
   Solution 5: [‘a’, ‘b’]

Let's pretend bacon has the list [3.14, 'cat,' 11, 'cat,' True] for the next three questions.

Since our list is: bacon=[3.14, 'cat’, 11, 'cat’, True]

6. What is the value of bacon.index('cat')?
   Solution 6: 1

7. How does bacon.append(99) change the look of the list value in bacon?
   Solution 7: [3.14, 'cat', 11, 'cat', True, 99]

8. How does bacon.remove('cat') change the look of the list in bacon?
   Solution 8: [3.14, 11, 'cat', True]

9. What are the list concatenation and list replication operators?
   Solution 9: for concatenation: + and for replication: *

10. What is difference between the list methods append() and insert()?
    Solution 10: append() method simply adds value at the end of a list meanwhile, with insert() we can add values at any position in the list. For e.g. if there is a list x then by using x.insert(4, ‘y’), y will get added the 4rth index.

11. What are the two methods for removing items from a list?
    Solution 11: We can use **remove()** method and **del** statement in order to remove items from a list. For example: del bacon[2], will remove item at 2nd index in list named bacon.

12. Describe how list values and string values are identical.
    Solution 12: Both lists and strings can be:
    - passed to len(), have indexes and slices
    - used in for loops

- concatenated or replicated
- used with the' in' and 'not' in operators.

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

List	Tuple
Mutable in nature	Immutable in nature
Operations are performed better	Elements can be accessed faster
More memory consumption	Less memory consumption
Consists of various built-in methods	Do not consist of any built-in methods
Errors occur frequently	No such problem occurs
Iterations are time-consuming in list	Iterations are faster in the tuple

14. How do you type a tuple value that only contains the integer 42?
    Solution 14: A tuple value that only contains the integer 42 can be written as- (42,)

15. How do you get a list value's tuple form? How do you get a tuple value's list form?
    Solution 15:
    - Getting a list value's tuple form: tuple()
    - Getting a tuple value's list form: list()

For example:

```
list1 = ['alex', 'brown', 'gitsey']
tuple1 = tuple(list1)
print(tuple1)
```

16. Variables that "contain" list values are not necessarily lists themselves. Instead, what do they contain?
    Solution 16: Such variables contain references to list values.

17. How do you distinguish between copy.copy() and copy.deepcopy()?
    Solution 17:
    The copy.copy() function will do a shallow copy of a list, while the copy.deepcopy() function will do a deep copy of a list. That is, only copy.deepcopy() will duplicate any lists inside the list. The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):

- A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.
- A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original. For example:

```
import copy
a = [1, 2, 3]
b = [4, 5, 6]
c = [a, b]
```

#Using normal assignment operatings to copy:

```
d = c

print id(c) == id(d)          # True - d is the same object as c
print id(c[0]) == id(d[0])    # True - d[0] is the same object as c[0]
```

#Using a shallow copy:

```
d = copy.copy(c)

print id(c) == id(d)          # False - d is now a new object
print id(c[0]) == id(d[0])    # True - d[0] is the same object as c[0]
```

#Using a deep copy:

```
d = copy.deepcopy(c)

print id(c) == id(d)          # False - d is now a new object
print id(c[0]) == id(d[0])    # False - d[0] is now a new object
```