ASSIGNMENT 4

ANSWER 1:

`[]` is an empty list, which is a list that contains no items in it.

ANSWER 2:

There are two ways to insert `'hello'` at the third place in the given list.

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

`spam[2] = 'hello'` this command will replace value `6` with `'hello'`, and the order of other values present in the list and the length of the list will not change. After the execution of the command, the spam list will have the following items in it.

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

or,

`spam.insert(2, 'hello')`, this command will insert 'hello' at 2nd index and will change the order of other existing items in the list and will increase the length of list by one. After the execution of the command, the original spam list will have the following items in it.

```
spam = [2, 4, 'hello', 6, 8, 10]
```

`spam = ['a', 'b', 'c', 'd']`

ANSWER 3: `'d'`

```
spam[int(int('3' * 2) / 11)]
```

```
'3' * 2 = '33'
int('33') = 33
33 / 11 = 3
int(3) = 3
```

```
spam[3]='d'
```

ANSWER 4:

`spam[-1] = 'd'`, negative indexing starts from the end of the list.

ANSWER 5:

`spam[:2] = ['a', 'b']`

By default the lower limit will be taken as 0 and it will go till index one less than the upper limit.

```
bacon = [3.14, 'cat', 11, 'cat', True]
```

ANSWER 6:
`bacon.index('cat') = 1`

ANSWER 7:
`bacon.append(99)` will add 99 at the end of the list named bacon.
`bacon = [3.14, 'cat', 11, 'cat', True, 99]`

ANSWER 8: `[3.14, 11, 'cat', True]`

ANSWER 9:
List concatenation operation is **+** and list replication operator is ***.**

ANSWER 10:

- `append()` will add value to the end of the list while `insert()` can add the value at any given index.
- `append(value)` takes only the value to be added at the end of list as an argument but `insert(index, value)` will index and value that needs to be added as arguments.

ANSWER 11:
There are 3 methods to delete items from the list.

1. `remove(value)` function is used to remove a particular element, it takes the value to be removed as an argument.
2. `pop(index)` function is used to remove the item from the list, and takes the index of the item as an argument.
3. `del` keyword is used to delete one item or multiple items from the list.
   Example: `del ListName[index]` will delete only one item at the particular index.
   `del ListName[start_index : end_index]` will remove multiple items from the list.

NOTE: there is one more function `clear()` which will clear the whole list.

ANSWER 12:
Reasons for both *string values* and *list values* being identical are enumerated below:

1. Both have indexes starting from start as 0 or starting from end as -1.
2. Both can be passed through `len()`.
3. Both can be sliced.
4. Both can be iterated through using loops.
5. Both can be concatenated or replicated.
6. Both can be used with the `in` and `not in` operators.

ANSWER 13:

1. lists are mutable and tuples are immutable.
2. Lists take more time and more memory when iterated through a loop.

For eg:
`total = sum([i for in range(1,100000)])` will consume more time and more memory than `total = sum((i for in range(1,100000))`

ANSWER 14:
`T = (42,)`
But if we did `T = (42)` then the compiler will just assign `42` as integer value to variable `T`, which can be checked using `type()` function.

ANSWER 15:
- One can get a list value's tuple form by using `tuple()` function.
- Similarly, we can get a tuple value's list form by using the list`()` function.

ANSWER 16:
The variable itself does not necessarily hold the entire list but it holds the reference to the list.

ANSWER 17:
In `copy.copy()`, a reference of the object is copied to another object. Therefore any changes made in the copied object will cause the original to be changed as well. This is also called shallow copy.

In case of `copy.deepcopy()`, a copy of the object is copied to another object. Thus, any changes made to the copy of the original object do not reflect in the original object.