1. [] represents the empty list value, which has no items in it.
2. To assign the value ‘hello’ in the list stored in the variable, spam – by using the following code:

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

spam[2] = ‘hello’

print(spam)

**OUTPUT:**

spam = [2, 4, ‘hello’, 8, 10]

1. The value of spam[int(int(‘3’\*2)/11)] is d
2. The value of spam[-1] is d
3. The value of spam[:2] is [‘a’, ‘b’]
4. The value of bacon.index(‘cat’) is 1
5. bacon = [3.14, ‘cat’, 11, ‘cat’, True, 99]
6. bacon = [3.14, 11, ‘cat’, True]
7. The list concatenation (+) operators create a new list that contains all the elements from List 1 followed by List 2. The list replication (\*) operators create a new list that replicates the specific number of times the elements in List 1.
8. In the list methods,
   * append() helps to add element at the end of the list without mentioning the exact position; simply appended to the end of the list
   * insert() helps to insert the element in the specified index position. Need to mention the index and element.
9. The two methods for removing items from a list in python:
   * remove() helps to remove the first occurrence of a specified value from the list.
   * pop() helps to remove and return the element from the specific index.
10. List values and String values are identical but with different characteristics.
    * Elements in both the data types are accessed by using indices and maintain the order in which they are added.
    * It can be iterated over each element using the loops
    * Both support slicing operations, leads to extract a portion of the data based on the specific indices
    * Both list and string values can concatenate using the ‘+’ operator.

|  |  |
| --- | --- |
| List | Tuple |
| They are mutable | They are immutable |
| Represent using square brackets ‘[ ]’ | Represent using parentheses ‘( )’ |
| It can be utilized, while working with dynamic data set, throughout program | It can be utilized, while using the constant data, throughout program |
| Requires more memory, also slower performance | Memory efficient and Faster access time, as compared to lists |

1. The tuple value that only contains the integer 42 can be type as

my\_tuple = (42,)

1. **List value’s tuple form:**

My\_list = [1, 2, 3, 4, 5]

My\_tuple = tuple(My\_list)

**OUTPUT:**

My\_tuple = (1, 2, 3, 4, 5)

**Tuple value’s list form:**

My\_tuple = (‘a’, ‘b’, ‘c’, ‘d’)

My\_list = list(My\_tuple)

**OUTPUT:**

My\_list = [‘a’, ’b’, ‘c’, ‘d’]

1. The variables that “contains” list values are represents the reference/ points to the lists rather than list themselves. In more precisely, the variables store the memory address or reference to the location where the list is stored in the computer’s memory. This concept is known as reference or pointer semantics.

|  |  |
| --- | --- |
| copy.copy() | copy.deepcopy() |
| Creates a new object, but does not create new objects for the elements inside it. | Creates a completely independent copy of the original object and all its nested elements |
| Changes in nested elements will be reflected in both the original and copied object | Changes in nested elements will not affect the copied object. |