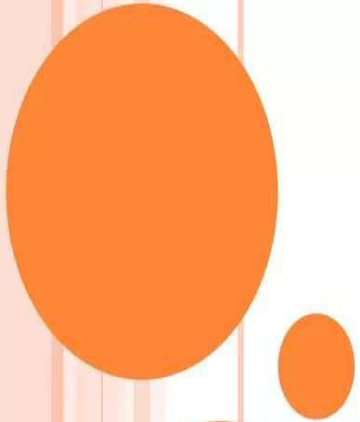
List Methods



List Creation

- A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).
- Example:

```
list = [1, 2, 3]    # list with same data-types
list = [1, "Hello", 3.4] # list with mixed data-types
```

Access Items from a List

- It can be access in several ways
- Use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
- ✓ Example:

```
list = ['p','r','o','b','e']

print(list[2]) #Positive Indexing

print(list[-2]) #Negative Indexing
```

Output

O

b

SLICE LISTS

- Accessing a range of items in a list by using the slicing operator [ ] using (colon :).
- Slicing can be best visualized by considering the index to be between the elements.
- ✓ Example:

```
list = ['p','r','o','b','e']

print(list[0:4]) #Positive

print(list[-2:-1]) # Negative
```

Output

```
['p', 'r', 'o', 'b']
['b']
```

List Methods

- append() - Add an element to the end of the list.
- count() - Returns the count of number of items passed as an argument.
- extend() - Add all elements of a list to the another list.
- index() - Returns the index of the first matched item.
- insert() - Insert an item at the defined index.
- pop() - Removes and returns an element at the given index.
- copy() - Returns a shallow copy of the list
- remove() - Removes an item from the list.
- reverse() - Reverse the order of items in the list.
- sort() - Sort items in a list in ascending order.

APPEND() METHODS

- The append() method adds a single item to the existing list.
- The append() method only modifies the original list. It doesn't return any value.
- The append() method takes a single *item* and adds it to the end of the list.
- The *item* can be numbers, strings, another list, dictionary etc.
- Syntax

```
list.append(item)
```

APPEND() METHODS - PROGRAMS

Example 1: Adding Element to a List

```
list = ['hi', 'hello']

print('old list: ', list)

list.append('welcome!')

print('Updated list: ', list)
```

Output

```
old list: ['hi', 'hello']
Updated list: ['hi', 'hello‘,'welcome!']
```

Example 2: Adding List to a List

```
list = ['hi', 'hello']

print('old list: ', list)

list1=['welcome‘]

list.append(list1)

print('Updated list: ', list)
```

Output

```
old list: ['hi', 'hello']
Updated list: ['hi', 'hello', ['welcome']]
```

COUNT() METHODS

- Count() method counts how many times an element has occurred in a list and returns it.
- The count() method takes a single argument:
  - **element** - element whose count is to be found.
- The count() method returns the number of occurrences of an element in a list.
- Syntax

```
list.count(element)
```

COUNT() METHODS - PROGRAMS

Example: Count the occurrence of an element in the list

```
list = ['h','i','h','o','w','y','o','u','!','!']

print('The count of i is:',list.count('i'))

print('The count of o is:',list.count('o'))
```

Output

```
The count of i is: 1
The count of o is: 2
```

Example 2: Count the occurrence of tuple and list inside the list

```
list = ['a', ('h', 'i'), ('a', 'b'), [8, 4]]

count = list.count(('a', 'b'))

print("The count of ('a', 'b') is:", count)

count = list.count([3, 4])

print("The count of [3, 4] is:", count)
```

Output

```
The count of ('a', 'b') is: 1
The count of [3, 4] is: 0
```

EXTEND() METHODS

- The extend() extends the list by adding all items of a list to the end.
- Extend() method takes a single argument (a list) and adds it to the end.
- This method also add elements of other native data_types ,like tuple and set to the list,
- Syntax

```
list1.extend(list2)
```

Note : The elements of *list2* are added to the end of *list1*.

```
list.extend(list(tuple_type))
```

Note : The elements of *tuple* are added to the end of *list1*.

EXTEND() METHODS - PROGRAMS

Example 1: Using extend() Method

```
language = ['C', 'C++', 'Python']

language1 = ['JAVA', 'COBOL']

language.extend(language1)

print('Language List: ', language)
```

Output

```
Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL']
```

EXTEND() METHODS - PROGRAMS

Example 2: Add Elements of Tuple and Set to List

```
language = ['C', 'C++', 'Python']
language_tuple = ['JAVA', 'COBOL']
language_set = {'.Net', 'C##'}
language.extend(language_tuple)
print('New Language List: ', language)
language.extend(language_set)
print('Newest Language List: ', language)
```

Output

New Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL'] Newest Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL', '.Net', 'C##']

INDEX( ) METHODS

- Index() method search and find given element in the list and returns its position.
- However, if the same element is present more than once, index() method returns its smallest/first position.
- Index in Python starts from 0 not 1.
- The index() method returns the index of the element in the list.
- The index method takes a single argument:
  - **element** - element that is to be searched.
- Syntax

```
list.index(element)
```

INDEX( ) METHODS - PROGRAMS

Example 1: Find position of element present in the list and not present in the list

```
list = ['h','i','h','o','w','y','o','u','!','!']

print('The count of i is:',list.index('i'))

print('The count of o is:',list.index('o'))

print('The count of o is:',list.index('z'))
```

Output

```
The count of i is: 1

The count of o is: 3

ValueError: 'z' is not in list
```

INDEX( ) METHODS - PROGRAMS

Example 2: Find index of tuple and list inside a list

```
random = ['a', ('a', 'd'),('a','b'), [3, 4]]

index = random.index(('a', 'b'))

print("The index of ('a', 'b'):", index)

index = random.index([3, 4])

print("The index of [3, 4]:", index)
```

Output

```
The index of ('a', 'b'): 2
The index of [3, 4]: 3
```

INSERT( ) METHODS

- The insert() method inserts the element to the list at the given index.
- The insert() function takes two parameters:
  - **index** - position where element needs to be inserted
  - **element** - this is the element to be inserted in the list
- The insert() method only inserts the element to the list. It doesn't return any value.
- Syntax

```
list.insert(index, element)
```

INSERT( ) METHODS - PROGRAMS

Example 1: Inserting Element to List

```
vowels=['a','i','o','u']

print("old list =",vowels)

vowels.insert(1,'e')

print("new list =",vowels)
```

Output

```
old list = ['a', 'i', 'o', 'u']
new list = ['a', 'e', 'i', 'o', 'u']
```

INSERT( ) METHODS - PROGRAMS

Example 2: **Inserting a Tuple (as an Element) to the List**

```
list = [{1, 2}, [5, 6, 7]]

print('old List: ', list)

number_tuple = (3, 4)

list.insert(1, number_tuple)

print('Updated List: ', list)
```

Output

```
old List: [{1, 2}, [5, 6, 7]]
Updated List: [{1, 2}, (3, 4), [5, 6, 7]]
```

POP( ) METHODS

- The pop() method takes a single argument (index) and removes the element present at that index from the list.
- If the index passed to the pop() method is not in the range, it throws **IndexError: pop index out of range** exception.
- The parameter passed to the pop() method is optional. If no parameter is passed, the default index **-1** is passed as an argument.
- Also, the pop() method removes and returns the element at the given index and updates the list.
- Syntax

```
list.pop(index)
```

POP( ) METHODS - PROGRAMS

Example 1: **Pop Element at the Given Index from the List**

```
list = ['Python', 'Java', 'C++', 'French', 'C‘]

return_value = language.pop(3)

print('Return Value: ', return_value)

print('Updated List: ', language)
```

Output

```
old list = ['Python', 'Java', 'C++', 'French', 'C']
Return Value: French
Updated List: ['Python', 'Java', 'C++', 'C']
```

POP( ) METHODS - PROGRAMS

Example 2: pop() when index is not passed

```
language = ['Python', 'Java', 'C++‘, 'C']
print('Return Value: ', language.pop())
print('Updated List: ', language)
print('Return Value: ', language.pop(-1))
print('Updated List: ', language)
```

Output

```
Return Value: C
Updated List: ['Python', 'Java', 'C++']
Return Value: C++ Updated List:
['Python', 'Java']
```

COPY( ) / ALIASING METHODS

- The copy() method returns a shallow copy of the list.
- A list can be copied with **=** operator.

```
old_list = [1, 2, 3]

new_list = old_list
```

- The problem with copying the list in this way is that if you modify the *new_list*, the *old_list* is also modified.
- Syntax

```
new_list = old_list
```

COPY( ) / ALIASING METHODS - PROGRAMS

Example 1: Copying a List

```
old_list = [1, 2, 3]

new_list = old_list

new_list.append('a')

print('New List:', new_list )

print('Old List:', old_list )
```

Output

New List: [1, 2, 3, 'a']

Old List: [1, 2, 3, 'a']

Shallow Copy( ) / Cloning Methods

- We need the original list unchanged when the new list is modified, you can use copy() method.
- This is called **shallow copy**.
- The copy() method doesn't take any parameters.
- The copy() function returns a list. It doesn't modify the original list.
- Syntax

```
new_list = list.copy()
```

SHALLOW COPY( ) / CLONING METHODS - PROGRAMS

Example 2: Shallow Copy of a List Using Slicing

```
list = [1,2,4]
new_list = list[:]
new_list.append(5)
print('Old List: ', list)
print('New List: ', new_list)
```

Output

Old List: [1, 2, 4]

New List: [1, 2, 4, 5]

REMOVE( ) METHODS

- The remove() method searches for the given element in the list and removes the first matching element.
- The remove() method takes a single element as an argument and removes it from the list.
- If the element(argument) passed to the remove() method doesn't exist, valueError exception is thrown.
- Syntax

```
list.remove(element)
```

REMOVE( ) METHODS - PROGRAMS

Example 1 :Remove Element From The List

```
list = [5,78,12,26,50]

print('Original list: ', list)

list.remove(12)

print('Updated list elements: ', list)
```

Output

```
Original list:  [5, 78, 12, 26,
50]Updated list:  [5, 78, 26, 50]
```

REVERSE( ) METHODS

- The reverse() method reverses the elements of a given list.
- The reverse() function doesn't return any value. It only reverses the elements and updates the list.
- Syntax

```
list.reverse()
```

REVERSE( ) METHODS - PROGRAM

Example 1 :Reverse a List Using Slicing Operator

```
list = [1,5,8,6,11,55]

print('Original List:', list)

reversed_list = list[::-1]

print('Reversed List:', reversed_list)
```

Output

Original List: [1, 5, 8, 6, 11, 55]

Reversed List: [55, 11, 6, 8, 5, 1]

Example 2: Reverse a List

```
list = [1,5,8,6,11,55]
print('Original List:', list)
list.reverse()
print('Reversed List:', list)
```

Output

Original List: [1, 5, 8, 6, 11, 55]

Reversed List: [55, 11, 6, 8, 5, 1]

SORT( ) METHODS

- The sort() method sorts the elements of a given list.
- The sort() method sorts the elements of a given list in a specific order - Ascending or Descending.
- Syntax

```
list.sort()
```

REVERSE( ) METHODS - PROGRAM

Example 1 :Sort a given List in ascending order

```
list = [1,15,88,6,51,55]

print('Original List:', list)

list.sort()

print('sorted List:', list)
```

Output

Original List: [1, 15, 88, 6, 51, 55]

sorted List: [1, 6, 15, 51, 55, 88]

Example 2 : Sort the list in Descending order

```
list = [1,15,88,6,51,55]

print('Original List:', list)

list.sort(reverse=True)

print('sorted List:', list)
```

Output

Original List: [1, 15, 88, 6, 51, 55]

sorted List: [88, 55, 51, 15, 6, 1]

Python Dictionary

- Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Creating a dictionary

- Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.
- An item has a key and the corresponding value expressed as a pair, key: value.
- While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

ACCESS ELEMENTS FROM A DICTIONARY

- While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.

```
d = {'name':'Ram', 'age': 26}
print("name is :",d['name'])
d['age']=33
d['Initial']='S‘
print("Updated dict :",d)
del d['Initial']
print("Updated dict :",d)
```

Output

```
name is : Ram
Updated dict : {'name': 'Ram', 'age': 33, 'Initial': 'S'}
Updated dict : {'name': 'Ram', 'age': 33}
```