LIST

Ordered, Indexed and Changeable. Allows duplicate. Elements can be of any data type.

In [1]:

```
# list.append(elmnt) (element can be string, number, object etc.)
fruits = ['apple', 'banana', 'cherry']
fruits.append("orange")
print(fruits)
b = ["Ford", "BMW", "Volvo"]
fruits.append(b)
print(fruits)

fruits = ['apple', 'banana', 'cherry']
cars = ['Ford', 'BMW', 'Volvo']
fruits.extend(cars) # iterable will be in the (). Required. Any iterable (list, set, tuple, etc.)
print(fruits)
fruits = ['apple', 'banana', 'cherry']
points = (1, 4, 5, 9)
fruits.extend(points)
print(fruits)

fruits = ['apple', 'banana', 'cherry']
car = {"brand": "Ford", "model": "Mustang",  "year": 1964}
fruits.extend(car)
print(fruits)

fruits = ['apple', 'banana', 'cherry']
veg_tup = ("potato", "onion", "capsicum")
fruits.extend(veg_tup)
print(fruits)

fruits = ['apple', 'banana', 'cherry']
veg_list = ["potato", "onion", "capsicum"]
fruits = fruits + veg_list  # concats only list with list
print(fruits)

fruits = ['apple', 'banana', 'cherry']
fruits.insert(1, "orange")  # An element of any type (string, number, object etc.)
print(fruits)

thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["watermelon", "guava", "apricot", "tomato"]    # RHS could be any collection. LHS will be list
print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

```
['apple', 'banana', 'cherry', 'orange', ['Ford', 'BMW', 'Volvo']]
['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']
['apple', 'banana', 'cherry', 1, 4, 5, 9]
['apple', 'banana', 'cherry', 'brand', 'model', 'year']
['apple', 'banana', 'cherry', 'potato', 'onion', 'capsicum']
['apple', 'banana', 'cherry', 'potato', 'onion', 'capsicum']
['apple', 'orange', 'banana', 'cherry']
```

In [2]:

```
fruits = ['apple', 'banana', 'cherry', 'orange']
fruits.clear()  # to remove ALL ELEMENTS from the list
print(fruits)

fruits = ['apple', 'banana', 'cherry']
x = fruits.pop(1)   # list.pop([pos]) --> pos default value is -1, which returns the last item. returns removed valu
print(fruits, x)
x=fruits.pop()
print(x)
fruits = []
#x = fruits.pop()   # it will throw IndexError because the list is empty.

fruits = ['apple', 'banana', 'cherry']
fruits.remove("banana") # removes the first occurrence of the element with the REQUIRED specified value. Required. A
print(fruits)
```

```
[]
['apple', 'cherry'] banana
cherry
['apple', 'cherry']
```

In [3]:

```
fruits = ['apple', 'banana', 'cherry', 'orange']
x = fruits.copy()
print(x)

# list1 = list2     this list1 is just a reference to list2. To copy the list2 into list1, use .copy()
newfruits = list(fruits)    # another way of list copying
print(newfruits)
```

```
['apple', 'banana', 'cherry', 'orange']
['apple', 'banana', 'cherry', 'orange']
```

In [4]:
```
fruits = ['apple', 'banana', 'cherry']
x = fruits.count("cherry")  # Required. Any type (string, number, list, tuple, etc.). The value to search for.
print(x)
```

```
1
```

In [5]:
```
fruits = ['apple', 'cherry', 'banana', 'cherry']
x = fruits.index("cherry")  # list.index(elmnt[,start][,end]) . elmnt is Required. Any type (string, number, list,
print(x)
```

```
1
```

In [6]:
```
dir(fruits)
```

Out[6]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
```

```
'__reduce_ex__',
'__repr__',
'__reversed__',
'__rmul__',
'__setattr__',
'__setitem__',
'__sizeof__',
'__str__',
'__subclasshook__',
'append',
'clear',
'copy',
'count',
'extend',
'index',
'insert',
'pop',
'remove',
'reverse',
```

In [7]:

```
fruits = ['zebra', 'elephant', 'apple', 'banana', 'cherry']
fruits.reverse()    # reverses list
print(fruits)
```

```
['cherry', 'banana', 'apple', 'elephant', 'zebra']
```

In [8]:

```
alph = ["b", "a", "c", "d"]
ralph = reversed(alph)  # (<any iterable object>)  .returns a reversed iterator object, not a list. Original list wo
print(alph)
print(ralph)

x = iter(["apple", "banana", "cherry"]) # iter(object, [sentinel])  . object is iterable object. sentinel is the obj
print(next(x))
print(next(x))
print(next(x))

class DoubleIt:
    def __init__(self):
        self.start = 1

    def __iter__(self):
        return self

    def __next__(self):
        self.start *= 2
        return self.start

    __call__ = __next__

my_iter = iter(DoubleIt(), 16)
for x in my_iter:
    print(x)
```

```
['b', 'a', 'c', 'd']
<list_reverseiterator object at 0x7f2e6d73ea90>
apple
banana
cherry
2
4
8
```

In [9]:

```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort() # list.sort([reverse=True|False], [key=myFunc]) . key can be the function to specify the sorting criteri
print(cars)

cars = ['Ford', 'BMW', 'Volvo']
cars.sort(reverse=True)
print(cars)

def myFunc(e):
  return len(e)
cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']
cars.sort(key=myFunc)
print(cars)


def myFunc(e):
  return e['year']
cars = [
  {'car': 'Ford', 'year': 2005},
  {'car': 'Mitsubishi', 'year': 2000},
  {'car': 'BMW', 'year': 2019},
  {'car': 'VW', 'year': 2011}
]
cars.sort(key=myFunc)
print(cars)

def myFunc(e):
  return len(e)
cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']
cars.sort(reverse=True, key=myFunc)
print(cars)
```

```
['BMW', 'Ford', 'Volvo']
['Volvo', 'Ford', 'BMW']
['VW', 'BMW', 'Ford', 'Mitsubishi']
[{'car': 'Mitsubishi', 'year': 2000}, {'car': 'Ford', 'year': 2005}, {'car': 'VW', 'year': 2011}, {'car': 'BMW', 'ye
ar': 2019}]
['Mitsubishi', 'Ford', 'BMW', 'VW']
```

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list. The expression is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list

```
newlist = \[expression for item in iterable if condition == True\]
```

In [10]:
```
# w/o LC
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
  if "a" in x:
    newlist.append(x)
print(newlist)
```

```
['apple', 'banana', 'mango']
```

In [11]:
```
# w/ LC
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
newlist = [x.upper() for x in fruits]
newlist = ['hello' for x in fruits]
newlist = [x if x != "banana" else "orange" for x in fruits]
newlist = [print(x) for x in fruits]
```

```
['apple', 'banana', 'mango']
apple
banana
cherry
kiwi
mango
```

In [12]:
```
# Removing duplicates from the list
mylist = ["a", "b", "a", "c", "c"]
mylist = list(dict.fromkeys(mylist))
print(mylist)
```

```
['a', 'b', 'c']
```

In [13]:
```
listOfList = [ [1, 2, 3, 4, 5],
               [11, 22, 33, 44, 55],
               [17, 18, 19, 20, 21] ]
flatList = [ item for elem in listOfList for item in elem]
print('Flat List : ', flatList)

flatList = []
for elem in listOfList:
    flatList.extend(elem)
print('Flat List : ', flatList)
```

```
Flat List :  [1, 2, 3, 4, 5, 11, 22, 33, 44, 55, 17, 18, 19, 20, 21]
Flat List :  [1, 2, 3, 4, 5, 11, 22, 33, 44, 55, 17, 18, 19, 20, 21]
```

In [14]:
```
lst = [1, 2, 3, 4, 5, 11, 22, 33, 44, 55, 17, 18, 19, 20, 21]
print(lst*2)

print(lst+2)
```

```
[1, 2, 3, 4, 5, 11, 22, 33, 44, 55, 17, 18, 19, 20, 21, 1, 2, 3, 4, 5, 11, 22, 33, 44, 55, 17, 18, 19, 20, 21]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-0856afac4d05> in <module>
      2 print(lst*2)
      3
----> 4 print(lst+2)

TypeError: can only concatenate list (not "int") to list
```