# Python List

**Data Structure:**

A data structure is a collection of data elements (such as numbers or characters—or even other data structures) that is structured in some way, for example, by numbering the elements. The most basic data structure in Python is the "sequence".

-> List is one of the Sequence Data structure

-> Lists are collection of items (Strings, integers or even other lists)

-> Lists are enclosed in [ ]

-> Each item in the list has an assigned index value.

-> Each item in a list is separated by a comma

-> Lists are mutable, which means they can be changed.

## List Creation

In [1]:
```
emptyList = []

lst = ['one', 'two', 'three', 'four'] # list of strings

lst2 = [1, 2, 3, 4] #list of integers

lst3 = [[1, 2], [3, 4]] # list of lists

lst4 = [1, 'ramu', 24, 1.24] # list of different datatypes

print(lst4)
```

```
[1, 'ramu', 24, 1.24]
```

## List Length

In [2]:
```
lst = ['one', 'two', 'three', 'four']

#find length of a list
print(len(lst))
```

```
4
```

## List Append

In [3]:
```
lst = ['one', 'two', 'three', 'four']

lst.append('five') # append will add the item at the end

print(lst)
```

```
['one', 'two', 'three', 'four', 'five']
```

## List Insert

In [4]:
```
#syntax: lst.insert(x, y)

lst = ['one', 'two', 'four']

lst.insert(2, "three") # will add element y at location x

print(lst)
```

```
['one', 'two', 'three', 'four']
```

## List Remove

In [5]:
```
#syntax: lst.remove(x)

lst = ['one', 'two', 'three', 'four', 'two']

lst.remove('two') #it will remove first occurence of 'two' in a given list

print(lst)
```

```
['one', 'three', 'four', 'two']
```

## List Append & Extend

In [6]:
```
lst = ['one', 'two', 'three', 'four']

lst2 = ['five', 'six']

#append
lst.append(lst2)

print(lst)
```

```
['one', 'two', 'three', 'four', ['five', 'six']]
```

In [7]:
```
lst = ['one', 'two', 'three', 'four']

lst2 = ['five', 'six']

#extend will join the list with list1

lst.extend(lst2)

print(lst)
```

```
['one', 'two', 'three', 'four', 'five', 'six']
```

## List Delete

In [8]:
```
#del to remove item based on index position

lst = ['one', 'two', 'three', 'four', 'five']

del lst[1]
print(lst)

#or we can use pop() method
a = lst.pop(1)
print(a)

print(lst)
```

```
['one', 'three', 'four', 'five']
three
['one', 'four', 'five']
```

## List related keywords in Python

In [9]:
```
#keyword 'in' is used to test if an item is in a list
lst = ['one', 'two', 'three', 'four']

if 'two' in lst:
    print('AI')

#keyword 'not' can combined with 'in'
if 'six' not in lst:
    print('ML')
```

```
AI
ML
```

## - List Reverse

In [10]:
```
#reverse is reverses the entire list

lst = ['one', 'two', 'three', 'four']

lst.reverse()

print(lst)
```

```
['four', 'three', 'two', 'one']
```

## - List Sorting

- The easiest way to sort a List is with the sorted(list) function.
- That takes a list and returns a new list with those elements in sorted order.
- The original list is not changed.
- The sorted() optional argument reverse=True, ##### e.g. sorted(list, reverse=True), makes it sort backwards.

In [11]:
```
#create a list with numbers
numbers = [3, 1, 6, 2, 8]

sorted_lst = sorted(numbers)


print("Sorted list :", sorted_lst)

#original list remain unchanged
print("Original list: ", numbers)
```

```
Sorted list : [1, 2, 3, 6, 8]
Original list:  [3, 1, 6, 2, 8]
```

In [12]:
```
#print a list in reverse sorted order
print("Reverse sorted list :", sorted(numbers, reverse=True))

#orginal list remain unchanged
print("Original list :",  numbers)
```

```
Reverse sorted list : [8, 6, 3, 2, 1]
Original list : [3, 1, 6, 2, 8]
```

In [13]:
```
lst = [1, 20, 5, 5, 4.2]

#sort the list and stored in itself
lst.sort()

# add element 'a' to the list to show an error

print("Sorted list: ", lst)
```

```
Sorted list:  [1, 4.2, 5, 5, 20]
```

In [14]:
```
lst = [1, 20, 'b', 5, 'a']
print(lst.sort()) # sort list with element of different datatypes.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-351ad9879cf6> in <module>()
      1 lst = [1, 20, 'b', 5, 'a']
----> 2 print(lst.sort()) # sort list with element of different datatypes.

TypeError: unorderable types: str() < int()
```

## List Having Multiple References

In [15]:
```
lst = [1, 2, 3, 4, 5]
abc = lst
abc.append(6)
#print original list
print("Original list: ", lst)
```

```
Original list:  [1, 2, 3, 4, 5, 6]
```

## String Split to create a list

In [16]:
```
#let's take a string

s = "one,two,three,four,five"
slst = s.split(',')
print(slst)
```

```
['one', 'two', 'three', 'four', 'five']
```

In [17]:
```
s = "This is applied AI Course"
split_lst = s.split() # default split is white-character: space or tab
print(split_lst)
```

```
['This', 'is', 'applied', 'AI', 'Course']
```

# List Indexing

- Each item in the list has an assigned index value starting from 0.
- Accessing elements in a list is called indexing.

In [18]:
```
lst = [1, 2, 3, 4]
print(lst[1]) #print second element

#print last element using negative index
print(lst[-2])
```

```
2
3
```

# List Slicing¶

- Accessing parts of segments is called slicing.
- The key point to remember is that the <font color ="red">:</font> end value represents the first value that is not in the selected slice.

In [19]:
```
numbers = [10, 20, 30, 40, 50,60,70,80]

#print all numbers
print(numbers[:])

#print from index 0 to index 3
print(numbers[0:4])
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 20, 30, 40]
```

In [20]:
```
print (numbers)
#print alternate elements in a list
print(numbers[::2])


#print elemnts start from 0 through rest of the list
print(numbers[2::2])
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
[10, 30, 50, 70]
[30, 50, 70]
```

## List extend using "+"

In [21]:
```
lst1 = [1, 2, 3, 4]
lst2 = ['one', 'two', 'three', 'four']
new_lst = lst1 + lst2

print(new_lst)
```

```
[1, 2, 3, 4, 'one', 'two', 'three', 'four']
```

## List Count

In [22]:
```
numbers = [1, 2, 3, 1, 3, 4, 2, 5]

#frequency of 1 in a list
print(numbers.count(1))

#frequency of 3 in a list
print(numbers.count(3))
```

```
2
2
```

## List Looping

In [23]:
```
#loop through a list

lst = ['one', 'two', 'three', 'four']

for ele in lst:
    print(ele)
```

```
one
two
three
four
```

# List Comprehensions

- List comprehensions provide a concise way to create lists.
- Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

In [24]:
```
# without list comprehension
squares = []
for i in range(10):
    squares.append(i**2)   #list append
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [25]:
```
#using list comprehension
squares = [i**2 for i in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [26]:
```
#example

lst = [-10, -20, 10, 20, 50]

#create a new list with values doubled
new_lst = [i*2 for i in lst]
print(new_lst)

#filter the list to exclude negative numbers
new_lst = [i for i in lst if i >= 0]
print(new_lst)


#create a list of tuples like (number, square_of_number)
new_lst = [(i, i**2) for i in range(10)]
print(new_lst)
```

```
[-20, -40, 20, 40, 100]
[10, 20, 50]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64),
(9, 81)]
```

# Nested List Comprehensions

In [27]:
```
#let's suppose we have a matrix

matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]
]

#transpose of a matrix without list comprehension
transposed = []
for i in range(4):
    lst = []
    for row in matrix:
        lst.append(row[i])
    transposed.append(lst)

print(transposed)
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

In [28]:
```
#with list comprehension
transposed = [[row[i] for row in matrix] for i in range(4)]
print(transposed)
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```