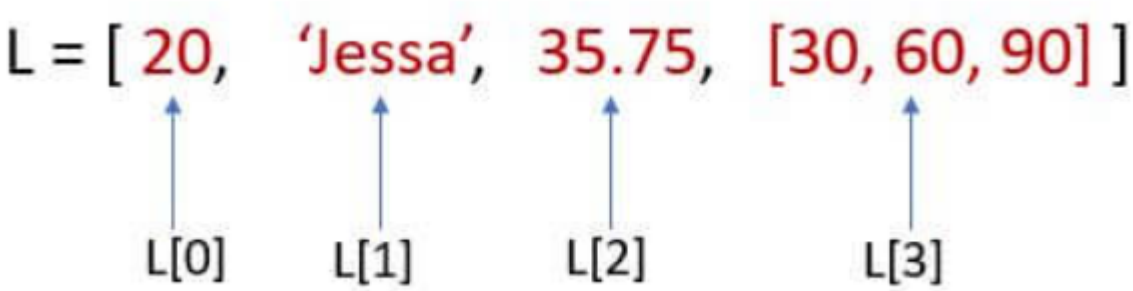
Python List

What are Lists ?

List is a data type where you can store multiple items under 1 name. More technically, lists act like dynamic arrays which means you can add more items on the fly.



- Why Lists are required in programming?
  - To store multiple homogenous or heterogenous elements in one box.

Array vs Lists

- Fixed Vs Dynamic Size
- Homogenous vs. heterogeneous
- Speed of Execution good Vs bad
- Memoryless Vs More
  - In a homogenous list, it stores data as an array in an allocated memory block

- In a heterogenous list, the address of each list element is stored, and that array works as a referential array. referential array can store pointers, addresses, or references.

In [26]:
```
Listt=[1,2,3,4,"list",True]
print(id(Listt))
print(Listt[0])
print(id(Listt[0]))
print(id(1))
```

```
3222962422656
1
140735539263928
140735539263928
```

How lists are stored in memory

Characterstics of a List

- Ordered
- Changeble/Mutable
- Hetrogeneous
- Can have duplicates
- are dynamic
- can be nested
- items can be accessed
- can contain any kind of objects in python

Creating a list

In [39]:
```
# Empty
print([])
# 1D -> Homo
print([1,2,3,4,5])
# 2D
print([1,2,3,[4,5]])
```

```
# 3D
print([[[1,2],[3,4]],[[5,6],[7,8]]])
# Hetrogenous
print([1,True,5.6,5+6j,'Hello'])
# Using Type conversion
print(list('hello'))
```

```
[]
[1, 2, 3, 4, 5]
[1, 2, 3, [4, 5]]
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
[1, True, 5.6, (5+6j), 'Hello']
['h', 'e', 'l', 'l', 'o']
```

Accessing items from a list

In [45]:
```
# Indexing
L = [[[1,2],[3,4]],[[5,6],[7,8]]]

#positive Indexing
print(L[0][0][1])

#negative indexing
print(L[-1][-2][-1])
```

```
2
6
```

In [53]:
```
# Slicing
L=[1,2,3,4,5,6,7,8]
print(L[0:3])
print(L[-3:])
print(L[0:2])
print(L[-5:-2:2])
```

```
[1, 2, 3]
[6, 7, 8]
[1, 2]
[4, 6]
```

Adding items to a list

In [81]:
```
L1=[1,2,3,4,5]
L2=[6,7,8,9]
# append - add item at last of list
L1.append(6)
L2.append([10,12,134])
print(L1,L2)

# extend - add elemnts of a list
L2.extend([12,14,56,78])
L1.extend("kumar")
print(L1,L2)

# insert - insert element at any position
L3=[12,134,14,156]
L3.insert(0,190)
print(L3)
```

```
[1, 2, 3, 4, 5, 6] [6, 7, 8, 9, [10, 12, 134]]
[1, 2, 3, 4, 5, 6, 'k', 'u', 'm', 'a', 'r'] [6, 7, 8, 9, [10, 12, 134], 12, 14, 56, 78]
[190, 12, 134, 14, 156]
```

Editing items in a list

In [86]:
```
L = [1,2,3,4,5]

# editing with indexing
L[-1] = 500

# editing with slicing
L[1:4] = [200,300,400]

print(L)
```

```
[1, 200, 300, 400, 500]
```

Deleting items from a list

In [92]:
```
# del
L = [1,2,3,4,5]

# indexing
del L[-1]
print(L)
# slicing
del L[1:3]
print(L)
```

```
[1, 2, 3, 4]
[1, 4]
```

In [106…
```
# remove - remove specific items if exist else error
L = [1,2,3,4,5]
L.remove(5)
print(L)

# pop - delete last item from a list if it is not empty else error
L = [1,2,3,4,5]
L.pop()
print(L)

# clear - clear the items in a list
L = [1,2,3,4,5]
L.clear()
print(L)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
[]
```

Operations on list

In [112…
```
# Arithmetic (+ ,*)

L1 = [1,2,3,4]
L2 = [5,6,7,8]

# Concatenation/Merge
```

```
print(L1 + L2)
print(L1*3)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

In [114…

```
# membership
L1 = [1,2,3,4,5]
L2 = [1,2,3,4,[5,6]]

print(5 not in L1)
print([5,6] in L2)
```

```
False
True
```

In [122…

```
# loops
L1 = [1,2,3,4,5]
L2 = [1,2,3,4,[5,6]]
L3 = [[[1,2],[3,4]],[[5,6],[7,8]]]

for i in L3:
  print(i,end='=')
```

```
[[1, 2], [3, 4]]=[[5, 6], [7, 8]]=
```

List functions

In [128…

```
# len/min/max/sorted
L = [2,1,5,7,0]

print(len(L),min(L),max(L),sorted(L,reverse=True))
```

```
5 0 7 [7, 5, 2, 1, 0]
```

In [130…

```
# count
L = [1,2,1,3,4,1,5]
L.count(5)
# reverse
L = [2,1,5,7,0]
# permanently reverses the list
```

```
L.reverse()
print(L)
# sort (vs sorted) - sort do sorting on actual list permanently but sorted return a new listt
L = [2,1,5,7,0]
print(L)
print(sorted(L))
print(L)
L.sort()
print(L)
```

```
[0, 7, 5, 1, 2]
[2, 1, 5, 7, 0]
[0, 1, 2, 5, 7]
[2, 1, 5, 7, 0]
[0, 1, 2, 5, 7]
```

In [140…

```
# copy -> shallow - it create a new list with new address
L = [2,1,5,7,0]
L2=L
L[-1]=100
print(L,L2,id(L),id(L2))
L1 = L.copy()
L[-1]=2000
print(L,L1,id(L),id(L1))
```

```
[2, 1, 5, 7, 100] [2, 1, 5, 7, 100] 3222962619136 3222962619136
[2, 1, 5, 7, 2000] [2, 1, 5, 7, 100] 3222962619136 3222954392384
```

List Comprehension

List Comprehension provides a concise way of creating lists.

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

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

Advantages of List Comprehension

- More time-efficient and space-efficient than loops.
- Require fewer lines of code.
- Transforms iterative statement into a formula.

Examples

In [152…
```
# Add 1 to 10 numbers to a list
listt=[i for i in range(1,11)]
print(listt)
```

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

In [156…
```
# Scalar multiplication on a vector
v=[2,3,4]
s=-3
print([i*s for i in v])
```

```
[-6, -9, -12]
```

In [158…
```
# Add squares
L = [1,2,3,4,5]
print([i**2 for i in L])
```

```
[1, 4, 9, 16, 25]
```

In [160…
```
# Print all numbers divisible by 5 in the range of 1 to 50
print([i for i in range(1,51) if i%5==0])
```

```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

In [162…
```
# find languages which start with letter p
languages = ['java','python','php','c','javascript']
print([i for i in languages if i.startswith('p')])
```

```
['python', 'php']
```

In [166…
```
# Nested if with List Comprehension
basket = ['apple','guava','cherry','banana','akama']
my_fruits = ['apple','kiwi','grapes','banana','akama']

# add new list from my_fruits and items if the fruit exists in basket
```

```
# and also starts with 'a'
print([i for i in my_fruits if i in basket and i.startswith('a')])
```

```
['apple', 'akama']
```

In [178…
```
# Print a (3,3) matrix using list comprehension -> Nested List comprehension
print([[i*j for i in range(1,4)]for j in range(1,4)])
```

```
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

In [180…
```
# cartesian products -> List comprehension on 2 lists together
L1 = [1,2,3,4]
L2 = [5,6,7,8]
print([i*j for i in L1 for j in L2])
```

```
[5, 6, 7, 8, 10, 12, 14, 16, 15, 18, 21, 24, 20, 24, 28, 32]
```

2 Ways to traverse a list

- itemwise
- indexwise

In [187…
```
# itemwise
L = [1,2,3,4]

for i in L:
  print(i)
```

```
1
2
3
4
```

In [189…
```
# indexwise
L = [1,2,3,4]

for i in range(0,len(L)):
  print(L[i])
```

```
1
2
3
4
```

Zip function

The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together.

If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

In [201…
```
# Write a program to add items of 2 lists indexwise

L1 = [1,2,3,4]
L2 = [-1,-2,-3,-4]
print(zip(L1,L2))
print(list(zip(L1,L2)))
[i+j for i,j in zip(L1,L2)]
```

```
<zip object at 0x000002EE670C7300>
[(1, -1), (2, -2), (3, -3), (4, -4)]
```

Out[201… [0, 0, 0, 0]

In [223…
```
# Make sure you haven't overridden the len function elsewhere in your code
listt = [12, 34, 456, 57, 79, 23]
index = [i for i in range(6)]
paired_list = list(zip(index, arr))

print("Paired List:", paired_list)
```

```
Paired List: [(0, 12), (1, 34), (2, 456), (3, 57), (4, 79), (5, 23)]
```

In [225…
```
L = [1,2,print,type,input]

print(L)
```

```
[1, 2, <built-in function print>, <class 'type'>, <bound method Kernel.raw_input of <ipykernel.ipkernel.IPythonKernel object at
0x000002EE4CD3DF40>>]
```

Disadvantages of Python Lists

- Slow
- Risky usage
- eats up more memory

In [228…

```
a = [1,2,3]
b = a.copy()

print(a)
print(b)

a.append(4)
print(a)
print(b)

# lists are mutable
```

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3]
```

END