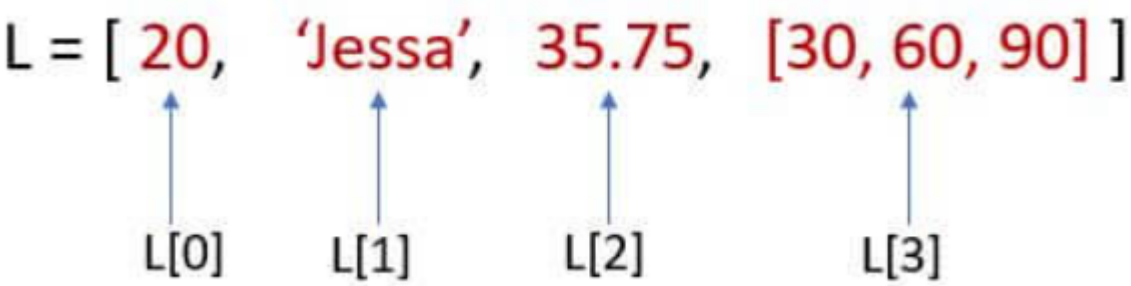
# 1. Lists

- What are Lists?
- Lists Vs Arrays
- Characterstics of a List
- How to create a list
- Access items from a List
- Editing items in a List
- Deleting items from a List
- Operations on Lists
- Functions on Lists

## 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?

# Array Vs Lists

- Fixed Vs Dynamic Size
- Convenience -> Hetrogeneous
- Speed of Execution
- Memory

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

print(id(L))
print(id(L[0]))
print(id(L[1]))
print(id(L[2]))
print(id(1))
print(id(2))
print(id(3))
```

```
140163201133376
11126688
11126720
11126752
11126688
11126720
11126752
```

# 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

```
L = [1,2,3,1]
L1 = [3,2,1]

L == L1
```

```
False
```

## Creating a List

```
# 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

```
# Indexing
L = [[[1,2],[3,4]],[[5,6],[7,8]]]
#positive
#print(L[0][0][1])

# Slicing
L = [1,2,3,4,5,6]

print(L[::-1])
```

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

## Adding Items to a List

```
# append
L = [1,2,3,4,5]
L.append(True)
print(L)
```

```
[1, 2, 3, 4, 5, True]
```

```
# extend
L = [1,2,3,4,5]
L.extend([6,7,8])
print(L)
```

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

```
L = [1,2,3,4,5]
L.append([6,7,8])
print(L)
```

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

```
L = [1,2,3,4,5]
L.extend('delhi')
print(L)
```

```
[1, 2, 3, 4, 5, 'd', 'e', 'l', 'h', 'i']
```

```
# insert
L = [1,2,3,4,5]

L.insert(1,100)
print(L)
```

```
[1, 100, 2, 3, 4, 5]
```

## Editing items in a List

```
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

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

# indexing
del L[-1]

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

```
[1, 4]
```

```
# remove

L = [1,2,3,4,5]

L.remove(5)

print(L)
```

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

```
# pop
L = [1,2,3,4,5]

L.pop()

print(L)
```

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

```
# clear
L = [1,2,3,4,5]

L.clear()

print(L)
```

```
[]
```

## Operations on Lists

- Arithmetic
- Membership
- Loop

```
# Arithmetic (+ ,*)

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

# Concatenation/Merge
print(L1 + L2)
```

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

```
print(L1*3)
```

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

```
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
```

```
# 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)
```

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

## List Functions

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

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

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

```
# count
L = [1,2,1,3,4,1,5]
L.count(5)
```

```
1
```

```
# index
L = [1,2,1,3,4,1,5]
L.index(1)
```

```
0
```

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

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

```
# sort (vs sorted)
L = [2,1,5,7,0]
print(L)
print(sorted(L))
print(L)
L.sort()
print(L)
```

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

```
# copy -> shallow
L = [2,1,5,7,0]
print(L)
print(id(L))
L1 = L.copy()
print(L1)
print(id(L1))
```

```
[2, 1, 5, 7, 0]
140163201056112
[2, 1, 5, 7, 0]
140163201128800
```

## 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.

```
# Add 1 to 10 numbers to a list
L = []

for i in range(1,11):
  L.append(i)

print(L)
```

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

```
L = [i for i in range(1,11)]
print(L)
```

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

```
# scalar multiplication on a vector
v = [2,3,4]
s = -3
# [-6,-9,-12]

[s*i for i in v]
```

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

```
# Add squares
L = [1,2,3,4,5]

[i**2 for i in L]
```

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

```
# Print all numbers divisible by 5 in the range of 1 to 50

[i for i in range(1,51) if i%5 == 0]
```

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

```
# find languages which start with letter p
languages = ['java','python','php','c','javascript']

[language for language in languages if language.startswith('p')]
```

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

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

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

[fruit for fruit in my_fruits if fruit in basket if fruit.startswith('a')]
```

```
['apple']
```

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

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

```
# cartesian products -> List comprehension on 2 lists together
L1 = [1,2,3,4]
L2 = [5,6,7,8]

[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

```
# itemwise
L = [1,2,3,4]

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

```
1
2
3
4
```

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

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

```
1
2
3
4
```

## Zip

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.

```
# Write a program to add items of 2 lists indexwise

L1 = [1,2,3,4]
L2 = [-1,-2,-3,-4]

list(zip(L1,L2))

[i+j for i,j in zip(L1,L2)]
```

```
[0, 0, 0, 0]
```

```
L = [1,2,print,type,input]

print(L)
```

```
[1, 2, <built-in function print>, <class 'type'>, <bound method Kernel.raw_in
```

## Disadvantages of Python Lists

- Slow
- Risky usage
- eats up more memory

```
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]
```

# List Programs

```
# Create 2 lists from a given list where
# 1st list will contain all the odd numbers from the original list and
# the 2nd one will contain all the even numbers

L = [1,2,3,4,5,6]


# How to take list as input from user


# Write a program to merge 2 list without using the + operator
L1 = [1,2,3,4]
L2 = [5,6,7,8]



# Write a program to replace an item with a different item if found in the list
L = [1,2,3,4,5,3]
# replace 3 with 300


# Write a program that can convert a 2D list to 1D list


# Write a program to remove duplicate items from a list
```