* **Container:**
  + Container is an entity that contains multiple data items.
  + Also known as a collection or a compound data type
* **Container Data Types:**
  + **Lists**
  + **Tuples**
  + **Sets**
  + **Dictionaries**
* **List**
  + Known as dynamic array as it can grow or shrink during execution of the program.
  + Commonly used for handling variable length data.
  + A list is defined by writing Comma-separated elements within **[ ]**.
    - num = [ 18, 2 , 72, 26, 4, 74 ]
    - names = [‘Darshit’, ‘Ragi’]
  + List can contain dissimilar data types, though usually they are a collection of similar types.
    - stud\_data = [‘Darshit’, 18, 2 , 72 ]
  + Items in a list can be repeated. i.e. a list may contain duplicate items.
    - ages = [50, 48, 50]
    - num = [10] \* 5 🡨 stores [10, 10, 10, 10, 10]
    - lst = [ ] 🡨 empty list, valid
* **Accessing List Elements**
  + print (num) 🡨 entire list can be printed by just using the name of the list.
    - print (names[0]) 🡨 ‘Darshit’ 🡨 Individual elements can also be accessed using

indices like we used earlier in strings.

* + - print num[1:4] 🡨 2, 72, 26 🡨 like strings, lists can also be sliced.
* **Looping in Lists**
  + Using **for** and **while**
  + #using for loop #using enumerate() to keep track of index

for i in num: for index, a in enumerate(num):

print (i) print (index, a)

* + #using while loop

i = 0

while i < len(num):

print(num[i])

i = i + 1

* **Basic List Operations**
  + **Lists are mutable (changeable).** 
    - name[0] = ‘Aashna’
    - num[0:3] = [15 , 8, 6 ]
  + **One list can be concatenated (appended) at the end of another list.**
    - lst1 = [ 15, 8 , 6]
    - lst2 = lst1 + [70, 80, 90]
  + **Two lists can be merged to create a new list.**
    - lst3 = lst1 + lst2
  + **Conversion:** 
    - **A string/tuple/set can be converted into a list using the list() conversion function.**
      * l = list(‘PDPU’) 🡪 l[‘P’,’D’,’P’,’U’]
  + **Aliasing:** 
    - **On assigning one list to another, both refer to the same list. Changing one changes the other.**
    - **Also known as shallow copy or aliasing.**
      * lst1 = [ 15, 8 , 6]
      * lst2 = lst1 🡨 doesn’t copy list. lst2 refers to the same list as lst1.
      * print (lst1 is lst2) 🡨 True
  + **Cloning:** 
    - **This involves copying contents of one list into another.**
    - **After copying both refer to different lists, although both contain same values.**
    - **Changing one list doesn’t change another.**
    - **Also known as deep copy.**
      * lst1 = [ 15, 8 , 6]
      * lst2 = [ ]
      * lst2 = lst2 + lst1
      * print (lst1 is lst2) 🡨 False
  + **Searching:**
    - **An element can be searched in a list using the [in] membership operator.**
      * print (15 **in** lst1) 🡨 True
  + **Identity:**
    - **Use [is] operator to check whether the two variables are referring to the same list.**
      * lst1 = [10 , 20 , 30 , 40 , 50 ]
      * lst2 = [10 , 20 , 30 , 40 , 50 ]
      * lst3 = lst1
      * print (lst1 is lst2, lst1 is not lst2, lst1 is lst3) 🡨 False, True, True
  + **Comparison:**
    - **We can compare contents of two lists.**
    - **Comparison is done item by item till there is a mismatch.**
      * A = [1, 2, 3, 4]
      * B = [1, 2, 5]
      * print ( A < B) 🡨 True
  + **Emptiness:** 
    - **We can check if a list is empty using not operator.**
      * lst = [ ]
      * if not lst:
        + print (“Empty list.”)
      * print (bool(lst))
  + **Built-in functions on Lists**
    - len(lst) - max(lst) - min(lst)
    - sum(lst) - any(lst) - all(lst)
    - del(lst) - sort(lst) - reverse(lst)
    - Examples of del( )
      * lst1 = [10, 20, 30, 40, 50]
      * del ( lst1[3] ) 🡨 [10, 20 , 30, 50]
      * del ( lst1[2:5] ) 🡨 [10, 20]
      * del(lst1[:]) or lst = [ ] 🡨 deletes entire list.
      * If multiple variables are referring to the same list, then deleting one doesn’t delete the others.
        + lst1 = [10, 20, 30, 40, 50]
        + lst3 = lst2 = lst1
        + lst1 = [ ]
        + print( lst1, lst2, lst3) 🡨 [ ] [10,20,30,40,50] [10,20,30,40,50]
        + print ( lst2 is lst3) 🡨 True
      * If multiple variables are referring to the same list and if we wish to delete all lists:
        + lst2[:] = [ ]
        + print (lst2, lst3 ) 🡨 [ ] [ ]
    - Examples of sort( ) and reverse( )
      * sort ( ) and reverse ( ) don’t return a list but both manipulate the list in place.
        + lst = [72, 60, 58 , 52, 54, 56 ]
        + lst.reverse()
        + print(lst) 🡨 [56, 54, 52, 58, 60, 72]
        + lst.sort()
        + print(lst) 🡨 [52, 54, 56, 58, 60, 72]
        + lst.sort(reverse = True)
        + print(lst) 🡨 [72, 60, 58, 56, 54, 52]
  + **List Methods**
    - **Any list is an object of type list.**
    - **Its methods can be accessed using the syntax lst.method( )**
      * lst = [72, 60, 58 , 52, 54, 56 ]
      * lst.append(74) # to append 74 at the end.
      * lst.remove(58) # deletes 58 from the list.
      * lst.remove(50) # reports valueError as 50 is not present in the list.
      * lst.pop() # removes last item from the list.
      * lst.pop(4) # removes 4th item from the list.
      * lst.insert(3,6) # inserts 6 at 3rd position.
      * lst.count(72) # counts frequency of 72 in the list.
      * i = lst.index(72) # returns index of item 72 in the list.
      * i = lst.index(50) # returns valueError as 50 is not there in the list.
  + **List Varieties**
    - **Nested List is allowed. We can create a list of lists.**
      * a = [ 1, 3, 5, 7, 9]
      * b = [2, 4, 6, 8, 10]
      * c = [a, b] 🡨 [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
      * print( c[0] [0], c [1] [2] ) #0th element of 0th list and 2nd element of 1st list.
    - A list may be embedded in another list
      * a = [ 1, 3, 5, 7, 9]
      * b = [2, 4, a, 6, 8, 10]
      * print (b) 🡨 [2, 4, [1, 3, 5, 7, 9], 6, 8, 10]
      * # unpack a string or list within a list using \* operator
      * b = [2, 4, \*a, 6, 8, 10]
      * print (b) 🡨 [2, 4, 1, 3, 5, 7, 9, 6, 8, 10]
  + **List Comprehensions**
    - Create a list with squares of integers from 1 to 10.
      * squares = []
      * for x in range(1, 11):
        + squares.append(x\*\*2)
    - The above code can be rewritten as
      * squares = [**x\*\*2** for x in range(1, 11)]
    - Create a list with squares of integers from 1 to 10 and take only the even numbers from the result
      * Even\_squares = [x \*\* 2 for x in range(1, 11) if x %2 == 0]
  + **Stack and Queue**

|  |  |
| --- | --- |
| **Stack** | **Queue** |
| Last In First Out (LIFO) | First In First Out (FIFO) |
| Adding an element is known as Push operation | Adding an element is known as insert operation |
| Removing an element is known as Pop operation | Removing an element is known as delete operation |
| Lists are efficient for implementation of Stack | Lists are not efficient for implementation of Queue.  Removal of items from beginning is not efficient as it involves shifting of all other elements by one position after deletion. So, **dequeue** class of **collections** module is preferred. |
| Methods  To push a value: append()  To pop a value: pop() | import collections  q = collections.deque()  Methods  To insert the value on the queue: List.append(item)  To delete the value from the queue: list.popleft() |

* **Write following programs considering list in mind:**

1. Create a list of 5 odd integers using random nos. Similarly create a list of 4 even integers using random nos. Replace the third element of odd integers with a list of 4 even integers. Flattern, sort and print the list. Provide appropriate message at each stage.
2. Generate 20 random integers and store them in a list. Accept a number from the user and print position of all occurrences of that number in the list.
3. Generate 50 random numbers in the range 1 and 30. Remove all duplicate values from the list.
4. Generate 30 random numbers and put them in a list. Create two more lists – one containing only +ve numbers and another with –ve nos.
5. A list contains 5 strings. Convert all these strings to uppercase.
6. Convert list of temperatures in Fahrenheit degrees to equivalent Celsius degrees.
7. Write a menu-driven program to implement the stack data structure.
8. Write a menu-driven program to implement the Queue data structure.
9. Take two lists of numbers. Create third list of numbers for only those numbers from first list which are not there in 2nd list (use list comprehension).