3. Working with Lists

❖Iterating over a list using loops.

- **for-loop** :
  - python uses a **foreach-style for-loop**—it pulls each element directly from the list
  - for item in lst:
  - example :

```
for i, item in enumerate(lst):
        print(I,item)
```

- **while-loop**:
  - i = 0

```
while i < len(lst):
   item = lst[i]
   i += 1
```

❖Sorting and reversing a list using sort(), sorted(), and reverse().

- sorting and reversing :

.sort()

- **in-place** sort method that changes the original list
- accepts:
  - key=… for custom comparison,
  - reverse=True for descending order

sorted()

- **built-in function** that returns a **new sorted list**, leaving original unchanged .
- also supports key= and reverse=.

reverse()

- list method that **reverses the list in-place**, without sorting

reversed()

- built-in function that returns an **iterator** yielding elements in reverse; wrap with list() to get a reversed list without modifying original .

❖ Basic list manipulations: addition, deletion, updating, and slicing.

- **addition**
  - using .append(x) to add x to end.
  - using .insert(i, x) to insert at index i.
  - concatenation: a + b or a.extend(b) adds one list to another.
- **deletion**
  - .remove(x): removes first matching x.
  - .pop() or .pop(i): removes (and returns) last or ith element.
  - del lst[i]: deletes element at index i.
- **updating**
  - assign directly: lst[i] = new_value.
- **slicing**
  - create sublists: lst[1:4], lst[:3], lst[-3:], lst[::2], lst[::-1].