Class 08: List & Tuple in Python

Presented by: Md Rasel Sarker

List in the Python

A **built-in** data type that **stores a set** of values.

It can **store elements** of **different types** (int, float, string, etc.).

```
marks = [87, 64, 33, 95, 76]        # marks[0], marks[1] ...

Students = [“Rasel”, 100, “Dhaka”]    # Students[0], Students[1] ...

student[0] = “Rasel“     # allowed in python

len(student)      # return length
```

Why Use Lists in Python?

- Stores multiple values in a single variable
- Supports multiple data types (int, float, string, etc.)
- Ordered and indexed - allows access via position
- Mutable - elements can be changed or updated
- Supports nesting - lists within lists
- Powerful built-in methods like append(), sort(), pop()

List Manipulation Techniques in Python

1. List slicing

Similar to String Slicing

```
# list[ starting_idx : ending_idx ]   # ending idx is not included

marks = [87, 64, 33, 95, 76]
marks[1:4]          # output: [64, 33, 95]

marks[:4]           # output: [87, 64, 33, 95]

marks[1:]           # output: [64, 33, 95, 76]

marks[-3:-1]        # output: [33, 95]
```

2. Accessing Elements

```
List = [2, 3, 4, 8,]

my_list[2]      # access by index
my_list[-1]     # access last item
```

3. Modifying Elements

```
my_list[0] = 99    # change value at index 0
```

4. Adding Elements

```
list.append(50)          # add at end
list.insert(idx, el)     # insert at index 2
list.extend([60, 70])    # add multiple elements
```

5. Removing Elements

```
list.remove(30)      # remove by value
list.pop()          # remove last el
list.pop(1)         # remove by index

del my_list[0]            # delete specific index
    list.clear()          # remove all items
```

6. Searching

```
list.index(40)    # get index of value
40 in my_list     # check existence
```

7. Sorting and Reversing

```
list.sort()              # sort ascending
list.sort(reverse=True)  # sort descending
list.reverse()           # reverse order
```

8. Copying Lists

```
list = my_list.copy()
new_list = my_list[:]        # Slice copy
```

9. List Comprehension

- squared = [x**2 for x in my_list]

10. Nested Lists

- nested = [[1, 2], [3, 4]]
- nested[0][1]   # Access 2

Tuples in Python

A build-ng data type that let's as create immutable sequence of values

Characteristic	Description
Immutable	Once created, values cannot be changed, added, or removed.
Ordered	Maintains the order of elements. Indexing and slicing are allowed.
Allows Duplicates	Tuples can store duplicate values.
Heterogeneous Data	Supports multiple data types (int, float, str, etc.).
Faster than Lists	More memory-efficient and faster than lists.
Can be Nested	Can contain other tuples or even lists as elements.
Hashable	Can be used as keys in dictionaries if all elements are hashable.
Tuple Packing and Unpacking	Supports packing multiple values and unpacking them into variables.

Tuple Methods

```
tup = (87, 64, 33, 95, 76)      # tup[0], tup[1], ...


tup[0] = 43                     # NOT allowed in Python (tuples are immutable)


tup1 = ()                       # Empty tuple
tup2 = (1,)                     # Single-element tuple (note the comma)
tup3 = (1, 2, 3)                # Multi-element tuple
```

Tuple Methods

```
tup = (1, 2, 3, 4, 5)

tup.index(el)  # returns the index of the first occurrence     tup.index(1) is 1

tup.count(el)  # count total occurrence     tup.count(1) is 1
```

Let’s practice

1. WAP to add 5 user-given numbers to a list and print their average.
2. WAP to find the second-largest number in a list.
3. WAP to remove all duplicate elements from a list.
4. WAP to sort a list in descending order without using built-in sort.
5. WAP to merge two lists and remove duplicates.
6. WAP to create a tuple from user input of 4 values and print the sum of numeric values only.
7. WAP to reverse a tuple.
8. WAP to check if all items in a tuple are of the same type.