

# 4

## Chapter 4

### ✓ Chapter 4 – Lists and Tuples in Python



lists.py

```
name = ["Prathamesh", "hello", 3890273, 873487.23982398, True, None]
print(name[4]) # Accessing 5th element: True
```

```
name[4] = False # Modifying an element
print(name[4]) # Now it prints False
```

```
print(name[1:5:2]) # Slicing with a step of 2 → ['hello', 873487.23982398]
```

#### 🧠 Concepts:

- Lists are ordered collections of items, enclosed in `[]`.
- **Lists are mutable** (i.e., you can modify, add, remove items).
- Elements can be of any data type.
- Indexing starts at **0**, and negative indexing is allowed.
- Slicing: `list[start:stop:step]` skips values using step.



## listmethods.py

```
name = ["Prathamesh", "hello", 3890273, 873487.23982398, True, None]
name.append(False) # Adds item at end
print(name)
```

```
l1 = [12, 212, 3, 44, 55252, 5, 355, 363, 837, 29867, 5234967, 93485, 42]
l1.sort()      # Sorts the list in ascending order
print(l1)
```

```
l1.pop(12)     # Removes element at index 12
print(l1)
```

```
l1.count(3)    # Returns count of value 3
print(l1)
```

```
l1.reverse()   # Reverses the list in-place
print(l1)
```

```
l1.insert(3, 3333) # Inserts 3333 at index 3
print(l1)
```

```
l1.remove(3)   # Removes first occurrence of 3
print(l1)
```



### Important List Methods:

Method	Description
<code>.append(x)</code>	Add <code>x</code> to end of list
<code>.sort()</code>	Sort list (ascending, by default)
<code>.pop(i)</code>	Remove element at index <code>i</code>
<code>.count(x)</code>	Count number of times <code>x</code> appears
<code>.reverse()</code>	Reverse the order of elements
<code>.insert(i,x)</code>	Insert <code>x</code> at index <code>i</code>
<code>.remove(x)</code>	Remove first occurrence of <code>x</code>



**Note:** List methods change the original list (**in-place operations**).

## tuple.py

```
# Wrong: a = (1)    → This is an integer, not a tuple
# Correct: a = (1,) → This is a single-element tuple

name = (12, 27823, True, "Prathamesh", 263.782367)
print(name)
print(type(name)) # <class 'tuple'>
```

### Concepts:

- Tuples are **immutable** (once defined, they can't be changed).
- Tuples use `()` instead of `[]`.
- Use a comma `,` for single-element tuples: `(value,)`

## tuplemethods.py

```
name = (12, 27823, 12, True, "Prathamesh", 263.782367)

a = name.count(12) # Counts how many times 12 appears → 2
print(a)

b = name.index(12) # First index where 12 appears → 0
print(b)
```

### Tuple Methods:

Method	Description
<code>.count(x)</code>	Number of times <code>x</code> appears in the tuple
<code>.index(x)</code>	First index of <code>x</code>

 Tuples are **highly restrictive** and allow only a few methods.

## Chapter 4 Practice Problems: Lists & Tuples

### Problem 1: Store Fruits in a List

```
# Write a program to store 7 fruits entered by user in a list
```

```
a = input("Enter Fruit Number 1: ")  
b = input("Enter Fruit Number 2: ")  
c = input("Enter Fruit Number 3: ")  
d = input("Enter Fruit Number 4: ")  
e = input("Enter Fruit Number 5: ")  
f = input("Enter Fruit Number 6: ")  
g = input("Enter Fruit Number 7: ")
```

```
fruits = [a, b, c, d, e, f, g]  
print(fruits)
```

 **Concepts used:** List creation, `input()`, storing data in a sequence.

## Problem 2: Sort Student Marks

```
# Write a program to enter marks of 6 students and sort them
```

```
a = int(input("Enter marks of student 1: "))  
b = int(input("Enter marks of student 2: "))  
c = int(input("Enter marks of student 3: "))  
d = int(input("Enter marks of student 4: "))  
e = int(input("Enter marks of student 5: "))  
f = int(input("Enter marks of student 6: "))
```

```
marks = [a, b, c, d, e, f]  
marks.sort()  
print(marks)
```

 **Alternate:** Use `.append()` inside a loop to make it compact.

## Problem 3: Tuple Immutability Check

```
# Check that a tuple type cannot be changed by python
```

```
a = (23, 24, True, "Prathamesh")  
a[2] = False # ❌ This will raise an error  
print(a)
```

### Error:

TypeError: 'tuple' object does not support item assignment

### Problem 4: Sum of a List of Four Numbers

# Write a program to sum a list with four numbers

```
list = []
```

```
a = int(input("Enter Number 1: "))
```

```
list.append(a)
```

```
b = int(input("Enter Number 2: "))
```

```
list.append(b)
```

```
c = int(input("Enter Number 3: "))
```

```
list.append(c)
```

```
d = int(input("Enter Number 4: "))
```


```
list.append(d)
```

# Manual sum

```
print(list[0] + list[1] + list[2] + list[3])
```

# Simpler way

```
print(sum(list))
```

 **Concepts:** List building, `.append()`, `sum()` function.

### Problem 5: Count Zeros in a Tuple

# Create a program to find number of zeros in a following tuple

```
a = int(input("Enter Number 1: "))
```

```
b = int(input("Enter Number 2: "))
```

```
c = int(input("Enter Number 3: "))
```

```
d = int(input("Enter Number 4: "))
```

```
e = int(input("Enter Number 5: "))
```

```
num = (a, b, c, d, e)
```

```
zeros = num.count(0)
print(zeros)
```

💡 Use `.count()` to efficiently count specific values in a tuple.



## Chapter 4 Summary – Lists and Tuples

Topic	Notes
<b>Lists</b>	Mutable, ordered collection defined using <code>[]</code>
<b>Tuples</b>	Immutable, ordered collection defined using <code>()</code>
<b>List Methods</b>	<code>.append()</code> , <code>.sort()</code> , <code>.pop()</code> , <code>.count()</code> , <code>.reverse()</code> , <code>.insert()</code>
<b>Tuple Methods</b>	Only <code>.count()</code> and <code>.index()</code> are available
<b>Indexing</b>	Starts from 0, supports slicing and negative indexing
<b>Slicing Format</b>	<code>list[start:stop:step]</code>
<b>Mutability</b>	Lists <input checked="" type="checkbox"/> Yes, Tuples <input type="checkbox"/> No
<b>Use Cases</b>	Use <b>lists</b> when data may change; use <b>tuples</b> for fixed data