Tuple Operations

* **Immutable**: Once created, the elements of a tuple cannot be changed. This makes them ideal for fixed collections like coordinates or calendar months.
* **Ordered**: Tuples preserve the order of elements, so indexing and slicing work just like with lists.
* **Indexed**: You can access elements using positive or negative indices, e.g., tup[0] or tup[-1].
* **Allow Duplicates**: Tuples can contain repeated values, just like lists.
* **Heterogeneous**: They can store elements of different data types — integers, strings, booleans, even other tuples.
* **Lightweight**: Tuples consume less memory than lists, making them efficient for read-only data.
* **Hashable**: If all elements are immutable, the tuple itself can be used as a dictionary key.
* **Nestable**: Tuples can contain other tuples, enabling complex data structures.

1.

info = ("Sasikala", 2025, True, (10, 20))

print(info[0]) # "Sasikala"

print(info[-1][1]) # 20

2.

info = ("Sasikala", 2025, True, (10, 20))

empty = ()

3.

print(len(info)) # 4

print(len(empty)) # 0

print(len(info[-1])) - output?

4.

combined = info + ("AI", "Python")

print(combined)

# Output: ('Sasikala', 2025, True, (10, 20), 'AI', 'Python')

4.

repeat = ("Hello",) \* 3

print(repeat)

# Output: ('Hello', 'Hello', 'Hello')

5.

print("Sasikala" in info) # True

print("Quantum" in info) # False

6.

name, year, status, coords = info print(name) # "Sasikala"

print(coords) # (10, 20)

name, \*rest=info :output?

7.

print(info[1:]) # (2025, True, (10, 20))

print(info[::-1]) # ((10, 20), True, 2025, 'Sasikala')

8.

info = ("Sasikala", 2025, True, (10, 20))

result = info[::-2]

print(result)

9

info = ("Sasikala", 2025, True, (10, 20))

print(info[2::-1])

10.

data = [(1, 'b'), (3, 'a'), (2, 'c')]

sorted(data) # Sorts by first element

sorted(data, key=lambda x: x[1]) # Sorts by second element

data.sort(): wrong statement? Why?

11.

data = [(2, 3), (4, 7), (8, 11), (3, 6)]

min\_x = min(data, key=lambda x: x[0]) # → (2, 3)

max\_y = max(data, key=lambda x: x[1]) # → (8, 11)

del data

12.

No data.append(), data.remove(), data.sort() methods because tuple is immutable

13.

data = [(3, 'c'), (1, 'a'), (2, 'b')]

data.sort() # Sorts by first element of each tuple

print(data)

14. not possible

data = ((3, 'c'), (1, 'a'), (2, 'b'))

data.sort() # Sorts by first element of each tuple

print(data)

15.

data\_tuple = tuple(data)

print(data\_tuple)

# Output: ('AI', 'Python', 2025)

16.

nested = [(1, 2), (3, 4)]

nested\_list = [list(t) for t in nested] # → [[1, 2], [3, 4]]

nested\_tuple = tuple(tuple(l) for l in nested\_list) # → ((1, 2), (3, 4))

1. Create a tuple of tuples containing student names and scores. Sort the tuple by score in descending order.

From a list of tuples representing (product, price), filter out products priced below ₹500.

products = [("Laptop", 45000), ("Mouse", 300), ("Keyboard", 700)] # Expected Output: [("Mouse", 300)]

🧠 3. Tuple of Tuples → Dictionary Conversion

**Question:**  
Convert a tuple of (key, value) pairs into a dictionary.

data = (("name", "Sasikala"), ("year", 2025), ("active", True)) # Expected Output: {'name': 'Sasikala', 'year': 2025, 'active': True}

🧠 4. Nested Tuple Access

**Question:**  
Access the value 7 from the following nested tuple:

nested = (1, (2, 3, (4, 5, (6, 7)))) # Expected Output: 7

🧠 5. Tuple Packing with Variable-Length Input

**Question:**  
Write a function that accepts any number of arguments and returns them as a tuple.

def pack\_into\_tuple(\*args): return args # pack\_into\_tuple

("AI", "Python", 2025) → ("AI", "Python", 2025)

🧠 6. Tuple XOR Operation

**Question:**  
Perform element-wise XOR between two tuples of equal length.

t1 = (1, 2, 3) t2 = (3, 2, 1) # Expected Output: (2, 0, 2)

🧠 7. Tuple Frequency Analysis

**Question:**  
Given a tuple of numbers, count how many times each number appears.

nums = (1, 2, 2, 3, 3, 3) # Expected Output: {1: 1, 2: 2, 3: 3}

Would you like me to turn these into a downloadable worksheet, or expand them into blog-ready explanations with code walkthroughs and visuals? I can also add real-world scenarios like sensor data or student grading systems!