4. Tuple

❖Introduction to tuples, immutability.

- **introduction to tuples & immutability :**

a tuple is an **ordered**, **immutable** sequence type in python. unlike lists, once a tuple is created, **its content cannot change**—no adding, removing, or assigning new values. they're ideal for representing fixed collections—like coordinates or database records—and can even serve as **dictionary keys** when composed of only immutable items

syntax :

```
t = (1, 2, 3)
t = (42,)
t = tuple([1, 2, 3])
```

❖Creating and accessing elements in a tuple.

- tuples are **ordered** and **immutable**—once created, you **cannot** change, add, or remove elements
- attempts to modify (like t[0] = x) raise a typeerror.

- creating & accessing elements in a tuple
  t = (1, 2, 3)
  t = (42,)
  t = tuple([1, 2, 3])

- accessing elements
  indexing by position, zero-based, supports negative indexes :
  t[0]
  t[-1]
  t[1:4], t[:-1], t[::-1]

❖ Basic operations with tuples: concatenation, repetition, membership.

- **concatenation (+)**
  - the + operator merges two tuples into a **new tuple**; originals stay unchanged
  - e.g. (1, 2) + (3, 4) yields (1, 2, 3, 4).
  - both operands must be tuples (trying to concatenate a list or other type raises typeerror)
  - you can also use +=, which under the hood creates a new tuple and reassigns it .

- **repetition (*)**
  - the * operator repeats the **entire tuple's contents** n times, returning a new tuple
  - e.g. (1, 2) * 3 → (1, 2, 1, 2, 1, 2).
  - repeating a single-element tuple works too, e.g. (10,) * 5 → (10, 10, 10, 10, 10) .
  - underlying tuples are immutable so repetition just builds a new sequence.

- **membership (in, not in)**
  - x in tup returns true if x exists in the tuple; otherwise false. similarly, not in checks the inverse .
  - internal check uses a **linear search**, so it's O(n).
  - e.g. 2 in (1, 2, 3) → true, 'a' not in (1, 2, 3) → true.