6) Generators And Iterators

- **Understanding how generators work in Python..**

What is Generator?

Generators in Python are a way to create **iterators** without storing the entire sequence in memory. They generate values **on-the-fly** using the yield keyword, making them **memory-efficient** for large datasets or infinite sequences.

Advantages of Generators

- **Memory efficiency**: They don't store the whole sequence.
- **Convenience**: Easy to write and read for custom iterators.
- **Lazy evaluation**: Values are generated on the fly.

- **Difference between yield and return.**

yield and **return** are used to send values back from a function, but they behave **very differently** in terms of **execution flow** and **memory usage.**

- **Yield**

  Pauses the function and returns a value.

  Resumes execution when next() is called again.

  Used in **generator functions** to produce sequences.

  Generates values lazily (memory-efficient)

- **Return**

  Exits the function immediately and returns a value.

  Cannot resume execution after return.

  Used in normal function (not generators).

  No memory of previous calls.

- **Understanding iterators and creating custom iterators.**

  What is an **Iterator** in Python?

  An **iterator** is an object that allows you to **iterate over** (i.e., loop through) a sequence of values, one at a time.

- Implements the **iterator protocol** (__iter__() and __next__())
- Maintains **internal state** to track position
- Raises StopIteration when exhausted