

The for-each loop in Java:

```
for (String fruit : fruits) { System.out.println(fruit); }
```

This loop is **syntactic sugar** for using an **iterator** behind the scenes.

Internally, it works like this:

When you use:





```
for (String fruit : fruits)
```

Java **automatically** does something like:


```
Iterator<String> iterator = fruits.iterator(); while (iterator.hasNext()) {  
String fruit = iterator.next(); System.out.println(fruit); }
```

So what happens step-by-step?

Let's say `fruits = ["Apple", "Banana", "Mango"]`.

1. `iterator.hasNext()` →  true (points to "Apple")
 2. `iterator.next()` → returns "Apple" → print it
 3. `iterator.hasNext()` →  true (now points to "Banana")
 4. `iterator.next()` → returns "Banana" → print it
 5. `iterator.hasNext()` →  true (now points to "Mango")
 6. `iterator.next()` → returns "Mango" → print it
 7. `iterator.hasNext()` →  false → loop ends
-

Your understanding:

"It gets the next method similar to Python to get next index" → 

Yes! It's like `next()` in Python iterators, and the loop continues while `hasNext()` is true.

Let me know if you want a visual explanation with drawings or a Python comparison!

so it has next and hasNext method

Great question — and you're almost there!

In Java:

When you use an **iterator**, it provides **two main methods**:

✅ 1. hasNext()

- Checks if **there is another element** in the collection.
- Returns `true` or `false`.

```
iterator.hasNext(); // returns true if there's another element
```

✅ 2. next()

- Returns the **next element** in the collection.
- Also **moves the pointer** to the next element.

```
String item = iterator.next(); // gets the next element
```

❌ **There is no method called** `has next` (with a space).

You must use:

```
iterator.hasNext(); // ✅ correct
```

Summary:

Method	Purpose
hasNext()	Checks if there are more elements
next()	Returns the next element in the iteration