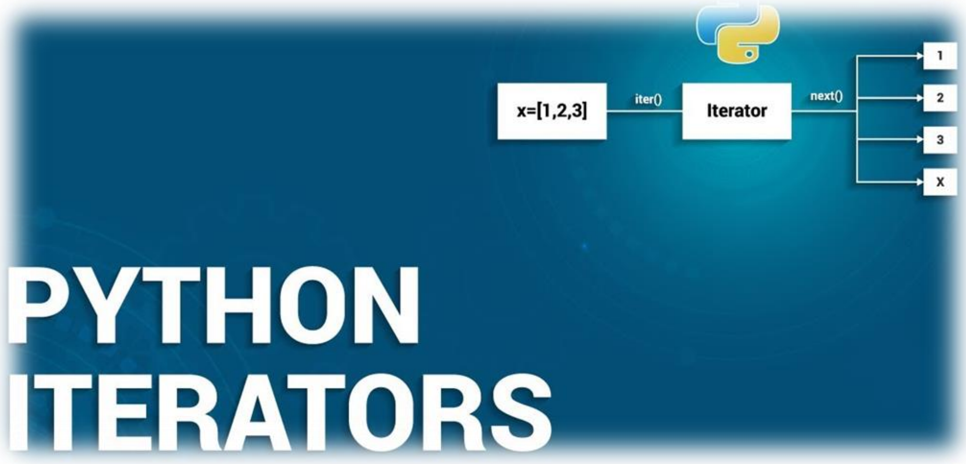
Python

Full stack Skills Bootcamp

Introducing Python Iterators

- What are Iterators?
  - Iterators are objects that allow you to traverse through all elements of a collection (like lists, strings, etc.) one at a time.
  - They implement the _iter_() and _next_() methods.
- Difference between iterables and iterators:
  - Iterable: An object capable of returning its elements one at a time.
  - Iterator: The object that produces the next element from the iterable when next() is called.

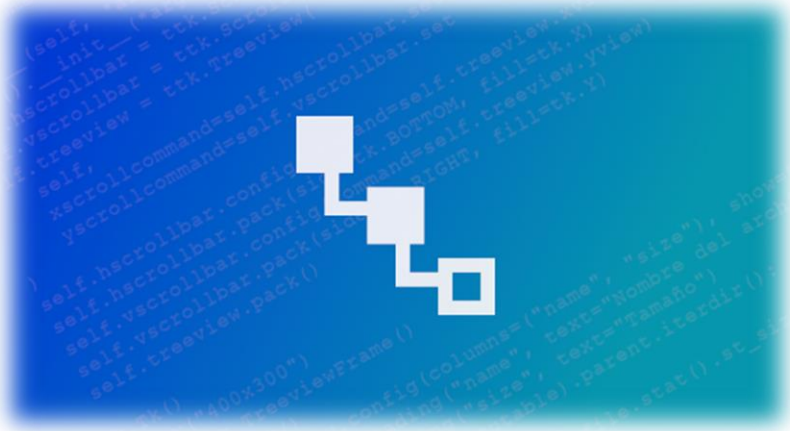


Basic Iterator Example

Basic Iterator Example with Lists:

```python
my_list = [10, 20, 30, 40, 50]
iterator = iter(my_list)
print(next(iterator))  # Output: 10
print(next(iterator))  # Output: 20
print(next(iterator))  # Output: 30
```

- Lists are iterables, meaning we can convert them into an iterator using the iter() function.
- Using next(), we retrieve the next element in the sequence.
- The iterator internally maintains its state, meaning it “remembers” the current position in the sequence.
- Important: When the iterator has no more elements, it raises a StopIteration exception.



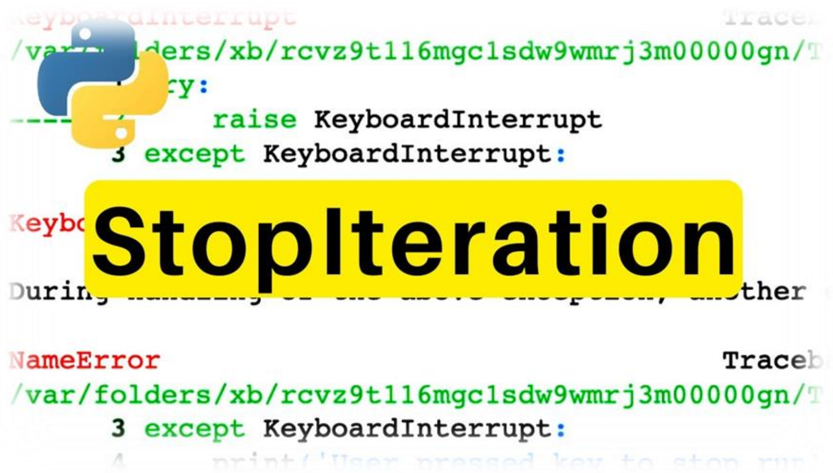
StopIteration Exception

Handling the End of an Iterator:

python

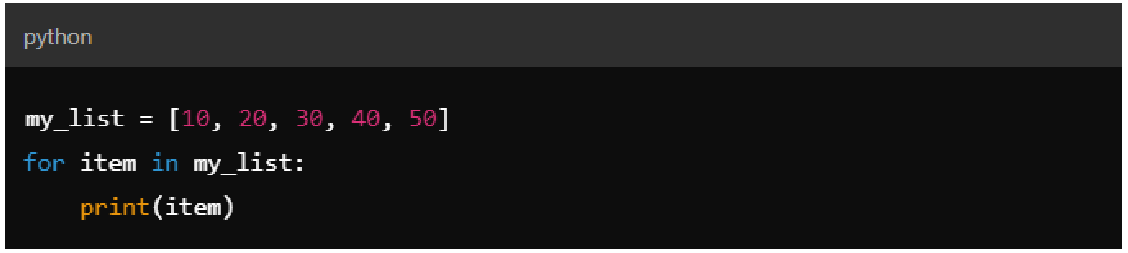
```python
# print(next(iterator))  # Raises StopIteration
```

- When the iterator is exhausted (i.e., all elements are iterated over), calling next() again raises a StopIteration exception.
- This is how Python signals that there are no more elements left to iterate.
- In large datasets, this behavior can help efficiently manage resources and stop when necessary.



Using for loop with an Iterator

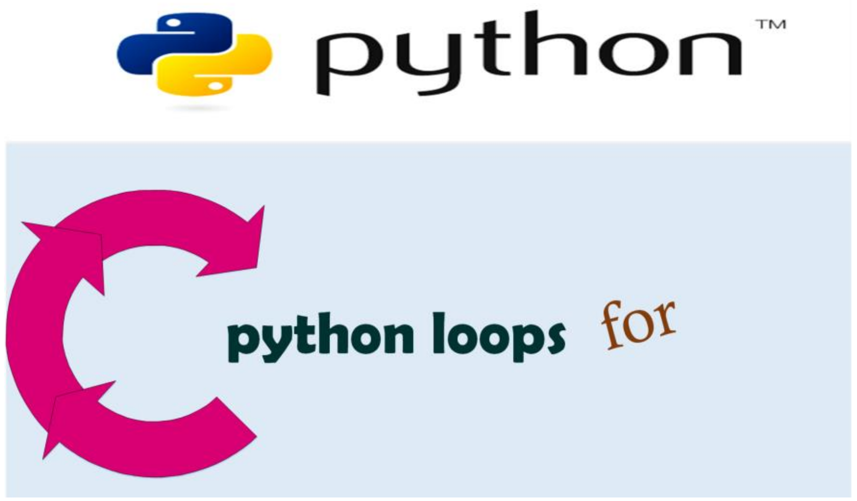
For Loop Simplifies Iteration:



python

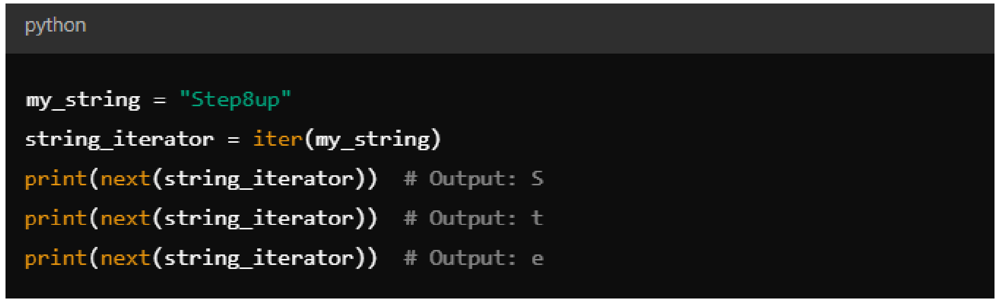
```python
my_list = [10, 20, 30, 40, 50]
for item in my_list:
    print(item)
```

- Python’s for loop abstracts the use of iter() and next() functions.
- The for loop automatically handles the iteration and stops when StopIteration is raised.
- Why use for loops: They provide a cleaner and more readable way to iterate over collections.
- This pattern is highly optimized and preferred in Python.



Iterating over Strings

Strings as Iterables

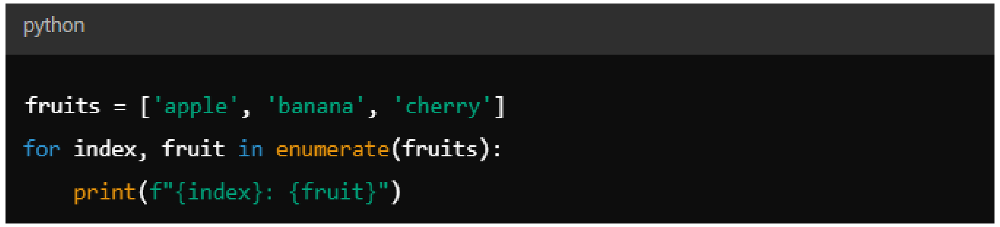

python

```python
my_string = "Step8up"
string_iterator = iter(my_string)
print(next(string_iterator))  # Output: S
print(next(string_iterator))  # Output: t
print(next(string_iterator))  # Output: e
```

- Just like lists, strings are also iterables.
- We can create an iterator from a string using iter().
- Using next(), characters in the string are retrieved one at a time.
- This concept applies to all iterables, such as tuples, sets, and dictionaries.

Built-in Functions with Iterators

Working with Built-in Functions



```python
fruits = ['apple', 'banana', 'cherry']
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
```

- Python offers various built-in functions to enhance working with iterators.
- enumerate() is a common function that returns both the index and the element during iteration.
- This is particularly useful when both the element and its position are needed.

Step8Up Academy

Conclusion

Real-World Use Cases of Iterators

Iterators are used extensively in Python, especially when working with:

- Large datasets: Iterators avoid loading all elements into memory at once.
- Generators: They are a type of iterator and provide a memory-efficient way of handling data.
- File reading: Iterators are used when processing files line by line.
- Data streaming: Iterators are ideal for streaming and processing data chunks in real-time.