Generators are special types of functions that can be paused and resumed. They are defined using the function\* syntax.

When a generator function is invoked, it returns an iterator object. You can use the yield keyword within a generator function to pause its execution and yield a value to the code that invoked it. The generator function can then be resumed later, and execution continues from where it was paused.

Here's a basic example:

| function\* generatorFunction() {  yield 1;  yield 2;  yield 3; }  const iterator = generatorFunction();  console.log(iterator.next().value); // Output: 1 console.log(iterator.next().value); // Output: 2 console.log(iterator.next().value); // Output: 3 console.log(iterator.next().value); // Output: undefined (generator function completed) |
| --- |

In this example:

* generatorFunction is a generator function that yields three values: 1, 2, and 3.
* We create an iterator object by invoking generatorFunction().
* We use the next() method of the iterator to get the next value yielded by the generator function. Each call to next() resumes execution of the generator function until the next yield statement is reached.
* The yield keyword allows the generator function to produce a sequence of values one at a time, making it useful for lazy evaluation, asynchronous operations, and more.

## Code Snippets & Sample

| function\* generateValues(list) {  for (let i = 0; i < list.length; i++) {  yield list[i];  } }  const myList = [10, 20, 30, 40, 50]; const iterator = generateValues(myList);  for (const value of iterator) {  console.log(value); } |
| --- |

* The generateValues function is a generator function that accepts a list as a parameter.
* Inside the function, a for loop iterates over each element of the list.
* The yield keyword is used to yield each element of the list one by one.
* iterator is created by invoking generateValues with myList.
* We then iterate over the iterator using a for...of loop, which internally calls next() on the iterator and prints each value yielded by the generator function.

This way, the generator function yields the next value from the list each time next() is called on the iterator, allowing you to loop over the list using a generator.