Generators in Javascript | JavaScript Tutorial In Hindi #53

In today's tutorial, we will study the **generator in JavaScript**, along with examples. A generator is a special kind of function that was introduced in ES6. In JavaScript, once we execute a function, it has to be executed entirely. But, generator functions enable us to create functions that another code can enter multiple times. Nothing from outside of the generator function can make it pause. Generator function pauses itself when it runs into a yield expression. Once the execution reaches the yield expression, the generator cannot continue execution on its own. Something from *outside* has to *continue* its execution.

Another important difference between generators and normal functions is that generator functions can produce multiple values during its execution. Hence, they can generate a sequence of values, not all at once, but on a per request basis. At every request, the generator function gives us a value until it reaches the end of its execution. Once that happens, *the done* flag will be set to *true*. Here is the syntax of defining the generator function:

Syntax:-

We already know how to declare a normal function. So, the syntax of declaring the generator function is quite similar to traditional functions. We declare a generator function by using the * (asterisk) operator after the *function* keyword:

```
function* myGenerator(){
  //code
}
```

Yield:-

The **yield** keyword pauses the **generator** function execution, and the value of the expression following the **yield** keyword is returned to the **generator's** caller. It acts as a **generator**-based version of the return keyword. In the following example, to pause the generator's execution, and we use the statement *yield*.:

```
function* awesomeGenerator(){
  yield 'Hello World' // We pause the execution here
  console.log('We are back again') // When we resume, we are here
}
```

next() method:-

A generator gives us the *next()* method, which is used to resume the execution. This method returns an object with two properties. These are value and done:

```
{
  value: [ next value ],
  done: [ true if we reach the end, else false]
}
```

Here is an example:

```
function* myGenerator() {
    yield 1;
    yield 2;
}
let iterator = myGenerator();
let result;
do {
result = iterator.next();
console.log(result);
} while (!result.done);
//Output:-
//{value: 1, done: false}
//{value: 2, done: false}
//{value: undefined, done: true}
```

Code Explanation:-

As we can see from the above code, when we first call the generator function, it returns us *iterator* object. When we call the next() for the first time, it starts executing the generator function, and it *yields* the first value: 1. Furthermore, calling it for the second time gives us the second and last value 2. Finally, the third call returns no value, and the *done* is set to true, which means that we have finished iterating through the generator.

Code as described/written in the video

```
console.log("this is tutorial 53");
// Generators in JavaScript
// 1 - 1B

function* numbersGen(){
    let i = 0;

    // yield 1;
    // yield 2;
    // yield 3;
    // yield 4;
    while(true){
        yield i++;
        // yield (i++).toString();
    }
}

const gen = numbersGen();
console.log(gen.next().value);
console.log(gen.next().value);
```

Previous Next

CodeWithHarry