SNS COLLEGE OF ENGINEERING

An Autonomous Institution

Coimbatore-107

CS8651-INTERNET PROGRAMMING

UNIT-1

JAVASCRIPT AND BASICS OF MERN STACK

Objects - Generators, advanced iteration

Objects

- A javaScript object is an entity having state and behavior (properties and method).
- For example: car, pen, bike, chair, glass, keyboard, monitor etc.
- JavaScript is an object-based language. Everything is an object in JavaScript.
- JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

Creating Objects in JavaScript

- There are 3 ways to create objects.
- By object literal
- By creating instance of Object directly (using new keyword)
- By using an object constructor (using new keyword)

1) JavaScript Object by object literal

- The syntax of creating object using object literal is given below:
- var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

  or

- var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
  };

Object Properties

- The **name:values** pairs in JavaScript objects are called **properties**

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

2) By creating instance of Object

- The syntax of creating object directly is given below:
- var objectname=new Object();
- Here, **new keyword** is used to create object.

```
<html>
<body>
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```

101 Ravi Malik 50000

- var person = new Object();
  person.firstName = "John";
  person.lastName = "Doe";
  person.age = 50;
  person.eyeColor = "blue";

3) By using an Object constructor

- Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.
- The **this keyword** refers to the current object.
-

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

```
103 Vimal Jaiswal 30000
```

What is Generator?

- A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return.
- The yield statement suspends the function's execution and sends a value back to the caller, but retains enough state to enable the function to resume where it is left off.
- When resumed, the function continues execution immediately after the last yield run.

What is Generator?

```
// An example of generator function
function* gen(){
    yield 1;
    yield 2;
    ...
    ...
}
```

Example

```
function* generate()
{
  console.log('invoked 1st time');
   yield 1;
   console.log('invoked 2nd time');
   yield 2;
}
// Code to invoke generator()
let gen = generate();
```

- Let’s examine the generate() function in detail.
- First, you see the asterisk (*) after the function keyword. The asterisk denotes that the generate() is a generator, not a normal function.
- Second, the yield statement returns a value and pauses the execution of the function.

```
function* forever()
{
   let index = 0;
   while (true)
        {
                yield index++;
         }
}
let f = forever();
console.log(f.next()); // 0
console.log(f.next()); // 1
console.log(f.next()); // 2
```

OUTPUT

- { value: 0, done: false }
- { value: 1, done: false }
- { value: 2, done: false }

value: It is the yielded value.

done: It is a Boolean value which gives true if the function code has finished. Otherwise, it gives false.

JavaScript Iterators and Iterables

- JavaScript Iterator is an object or pattern that allows us to traverse over a list or collection.
- Iterators define the sequences and implement the iterator protocol that returns an object by using a **next()** method that contains the value and is done.
- The value contains the next value of the iterator sequence and the **done** is the boolean value true or false if the last value of the sequence has been consumed then it's true else false.

- JavaScript provides a protocol to iterate over data structures. This protocol defines how these data structures are iterated over using the for...of loop.
- The concept of the protocol can be split into:
  - iterable
  - iterator
- The iterable protocol mentions that an iterable should have the Symbol.iterator key.

- JavaScript provides a protocol to iterate over data structures. This protocol defines how these data structures are iterated over using the for...of loop.
- The concept of the protocol can be split into:
  - iterable
  - iterator
- The iterable protocol mentions that an iterable should have the Symbol.iterator key.

JavaScript Iterables

- The data structures that have the Symbol.iterator() method are called iterables. **For example, Arrays, Strings, Sets, etc.**

```
const dept = "CSE DEPARTMENT";
for (let n of  dept[Symbol.iterator]())
{
   console.log(n);
}
```

O/P: CSE DEPARTMENT

JavaScript Iterators

- An iterator is an object that is returned by the Symbol.iterator() method.
- The iterator protocol provides the next() method to access each element of the iterable (data structure) one at a time.
- The iterator protocol defines how to produce a sequence of values from an object.
- An object becomes an iterator when it implements a next() method.

Example

```
const arr = ['h', 'e', 'l', 'l', 'o'];
let arrIterator = arr[Symbol.iterator]();
console.log(arrIterator.next()); // {value: "h", done: false}
console.log(arrIterator.next()); // {value: "e", done: false}
console.log(arrIterator.next()); // {value: "l", done: false}
console.log(arrIterator.next()); // {value: "l", done: false}
console.log(arrIterator.next()); // {value: "o", done: false}
console.log(arrIterator.next()); // {value: undefined, done:
true}
```

Thank You