Chapter 2 Notes

Introduction Java Script

1.1 Introduction to Java Script

- In Web application development java script is one of the most important language, using java script developer can do lots of logical stuff both on client and server side.
- Traditionally java script mainly used on the client side mostly for client side validation purpose for example password field should not be blank, password and re password should be match etc.
- With the time java script language used not only on the client side but for many more uses like server side, mobile app development, game development, desktop app development etc.
- As of writing this the java script is one of the dominant and most preferred language in web application development and other use cases.
- Hence Learning java script from scratch is most important skills to acquire for future web development, mobile development or game development carriers.

1.2 What is Java Script?

- JavaScript is a lightweight, cross-platform, and interpreted compiled programming language which is also known as the scripting language for web pages. It is well-known for the development of web pages, many non-browser environments also use it. JavaScript can be used for Client-side developments as well as Server-side developments

- JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at —

displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved.

- JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else. (Okay, not everything, but it is amazing what you can achieve with a few lines of JavaScript code.

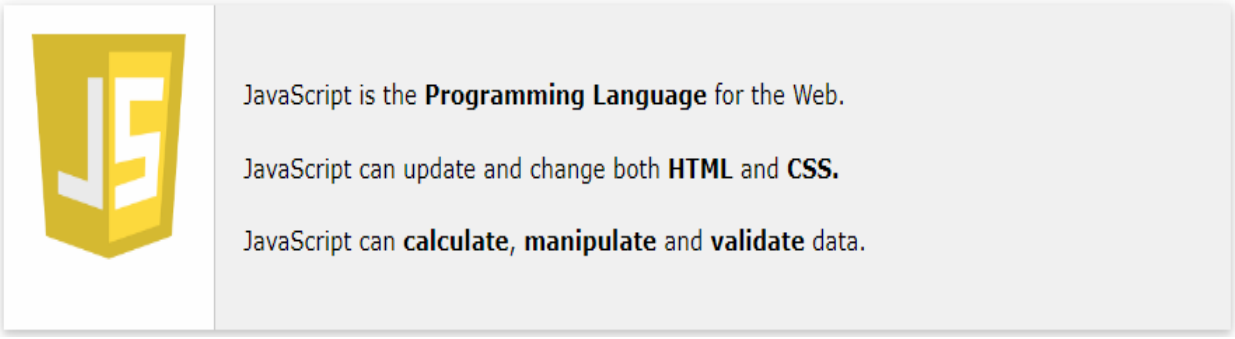


Figure 2.1 What Is Java Script

1.3 Advantages of JavaScript

1. Less server interaction − you can validate user input before sending the page off to the server. This saves server traffic, which means less loads on your server.
2. Immediate feedback to the visitors − they don't have to wait for a page reload to see if they have forgotten to enter something.
3. Increased interactivity − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
4. Richer interfaces − you can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

1.4 Java Script Engine

A JavaScript engine is a software component that executes JavaScript code. The first JavaScript engines were mere interpreters, but all relevant modern engines use just-in-time compilation for improved performance.

JavaScript engines are typically developed by web browser vendors, and every major browser has one. In a browser, the JavaScript engine runs in concert with the rendering engine via the Document Object Model.

Why JavaScript engine is needed?

For the most part, the JS Engine is used to interpret JavaScript. It's used to **parse** JavaScript and run it on a web page.

JavaScript is not understandable by computer but the only browser understands JavaScript. So, we need a program to convert our JavaScript program into computer-understandable language.

A JavaScript engine is a computer program that executes JavaScript code and converts it into computer understandable language.

List of JavaScript Engines:

Browser	Name of Javascript Engine
Google Chrome	V8
Edge (Internet Explorer)	Chakra
Mozilla Firefox	Spider Monkey
Safari	Javascript Core Webkit

Note – All the Students should Explore the detailed role of How Java Script Engine work internally in the different web Bowsers.

Java Script DOM Model

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

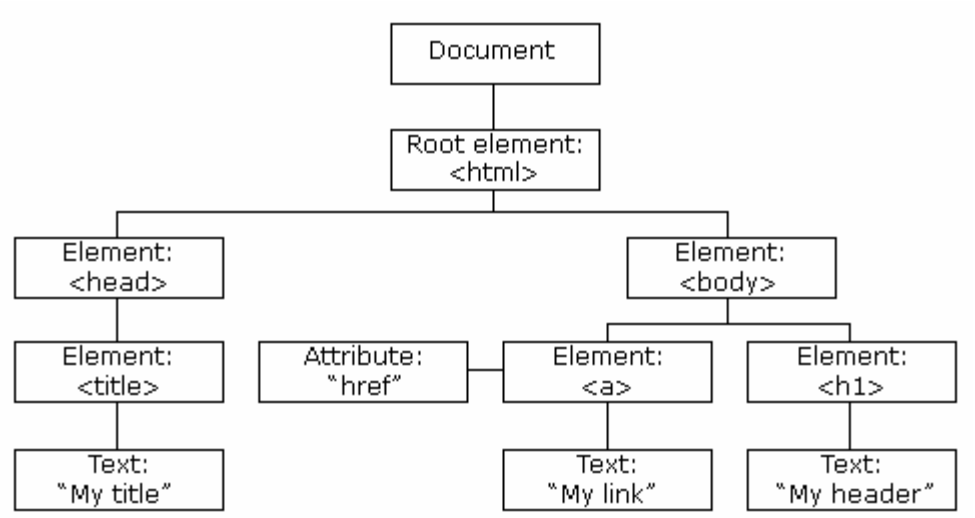


Figure DOM Model

As shown in figure DOM Model represent all the elements in the tree structure, some elements are root and some elements are parent. This DOM structure is important if you are accessing elements from the html from java script, for example

Document.getElementById() method is used to access html element by id as DOM.

1.5 Java Script Language Construct

JavaScript is an interpreted programming language with object-oriented (OO) capabilities. Syntactically, the core JavaScript language resembles C, C++, and Java, with programming constructs such as if statement, the while loop, and the && operator etc.

The similarity ends with this syntactic resemblance, however. JavaScript is a loosely typed language, which means that variables do not need to have a type specified.

Java Script Data Types

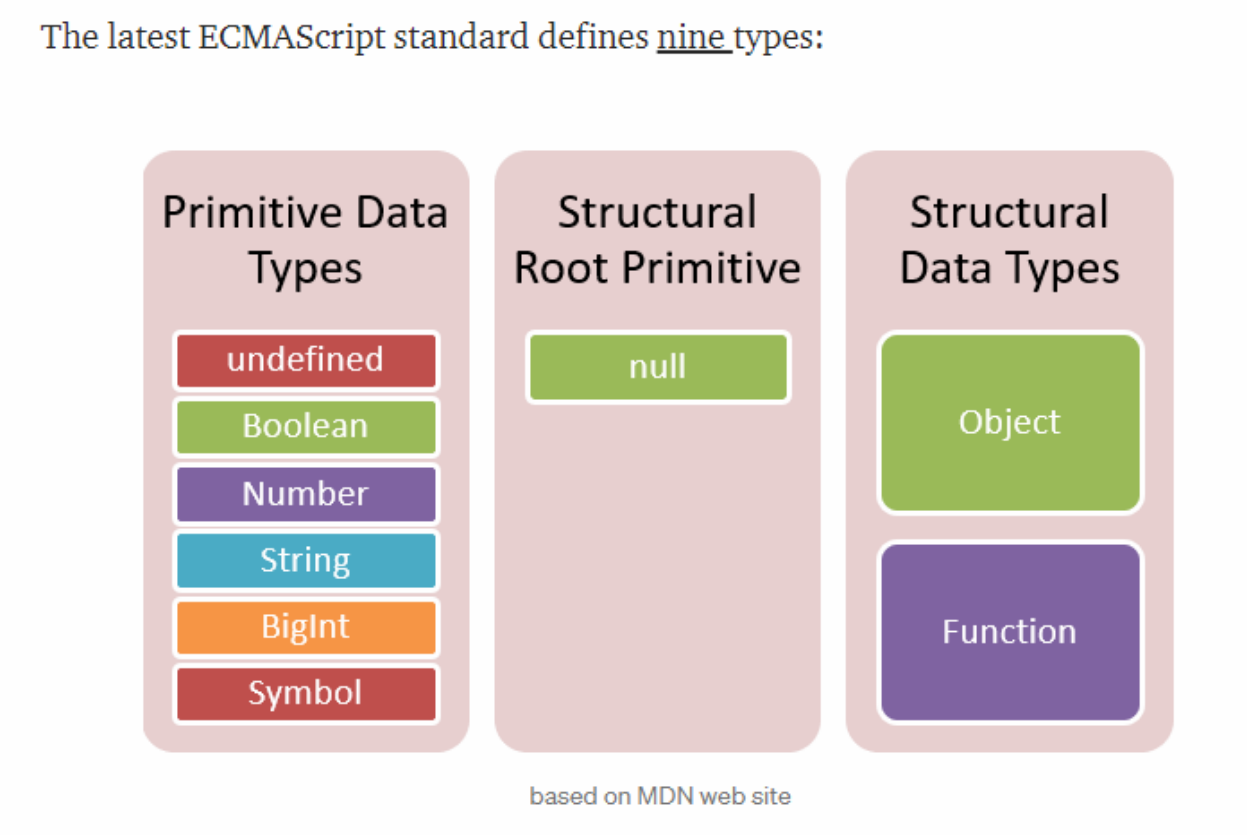
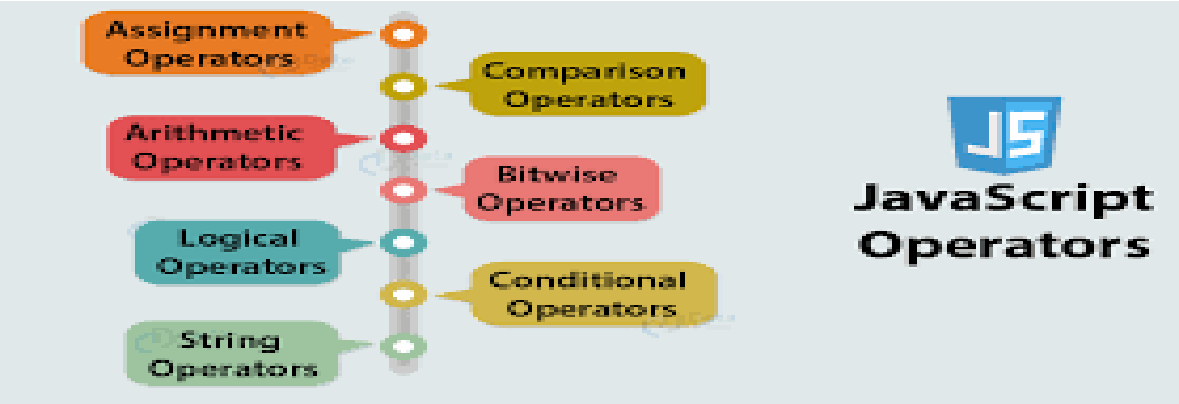


Figure 2.2 Java Script Data Types

Note - All types except objects define immutable values (that is, values which can't be changed). For example (and unlike in C), Strings are immutable. We refer to values of these types as "primitive values".

Java Script Operators



Java Script Variables

Where does JavaScript fit?

JavaScript is a loosely typed and dynamic language. Since JavaScript is a loosely typed language; variables in JavaScript are not directly associated with any particular value type. As mentioned above, you don't have to specify what type of information will be stored in a variable in advance.

4 Ways to Declare a JavaScript Variable:

Using var

Using let

Using const

Using nothing

When to Use JavaScript var?

Always declare JavaScript variables with var,let, orconst.

The var keyword is used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

If you want your code to run in older browser, you must use var.

When to Use JavaScript const?

If you want a general rule: always declare variables with const.

If you think the value of the variable can change, use let.

In this example, price1, price2, and total, are variables:When to Use JavaScript const?

If you want a general rule: always declare variables with const.

Example 1.1 Java Script Variable

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, price1, price2, and total are variables.</p>
<p id="demo"></p>
<script>
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
document.getElementById("demo").innerHTML ="The total is: " + total;
</script>
</body>
</html>
```

1,6 Java Script Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.
You can use conditional statements in your code to do this.
In JavaScript we have the following conditional statements:
Use if to specify a block of code to be executed, if a specified condition is true
Use else to specify a block of code to be executed, if the same condition is false
Use else if to specify a new condition to test, if the first condition is false
Use switch to specify many alternative blocks of code to be executed

Example of if else if in Java Script

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript if .. else</h2>
<p>A time-based greeting:</p>
<p id="demo"></p>
<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML = "Time is"+time+ " so " +greeting;
</script>
</body>
</html>
```

Note – based on the system time it will greet the user. All students note how to get date and use them using if else if in java script.

Switch Case in Java Script

The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
switch (expression) {
case condition 1: statement(s)
break;

case condition 2: statement(s)
break;
...

case condition n: statement(s)
break;

default: statement(s)
}
```

Example switch case in Java Script

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
          case 'A': document.write("Good job<br />");
          break;
```

```
            case 'B': document.write("Pretty good<br />");
            break;

            case 'C': document.write("Passed<br />");
            break;

            case 'D': document.write("Not so good<br />");
            break;

            case 'F': document.write("Failed<br />");
            break;

            default:  document.write("Unknown grade<br />")
          }
          document.write("Exiting switch block");
        //-->
    </script>
    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Note – Students try above example with user input grade at run time.

Java Script Loops

JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- for/of - loops through the values of an iterable object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true

<!DOCTYPE html>

Example 1 For Loop in Java Script

```
<html>
<body>
<h2>JavaScript For Loop</h2>
<p id="demo"></p>
<script>
let text = "";
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Example 2 Java Script For for/in

The JavaScript for in statement loops through the properties of an Object:

Syntax

```
for (key in object) {
  // code block to be executed
}
```

Example Code

<!DOCTYPE html>

```
<html>
<body>
<h2>JavaScript For In Loop</h2>
<p>The for in statement loops through the properties of an object:</p>
<p id="demo"></p>
<script>
const person = {fname:"John", lname:"Doe", age:25};
let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>
</body>
</html>
```

Example 3 Java Script for of

The For Of Loop

The JavaScript for of statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

Syntax

```
for (variable of iterable) {
```

```
  // code block to be executed
}
```

Example Code Loop over Array in Java Script

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For Of Loop</h2>
<p>The for of statement loops through the values of any iterable object:</p>
<p id="demo"></p>
<script>
const cars = ["BMW", "Volvo", "Mini"];
let text = "";
for (let x of cars) {
  text += x + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

1.6 Java Script ES5 and ES 6 Difference

Based on	ES5	ES6
Definition	ES5 is the fifth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International)	ES6 is the sixth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International).
Release	It was introduced in 2009.	It was introduced in 2015.
Data-types	ES5 supports primitive data types that are **string, number, boolean, null,** and **undefined**.	In ES6, there are some additions to JavaScript data types. It introduced a new primitive data type **'symbol'** for supporting unique values.
Defining Variables	In ES5, we could only define the variables by using the **var** keyword.	In ES6, there are two new ways to define variables that are **let** and **const**.
Performance	As ES5 is prior to ES6, there is a non-presence of some features, so it has a lower performance than ES6.	Because of new features and the shorthand storage implementation ES6 has a higher performance than ES5.
Support	A wide range of communities supports it.	It also has a lot of community support, but it is lesser than

		ES5.
Object Manipulation	ES5 is time-consuming than ES6.	Due to destructuring and speed operators, object manipulation can be processed more smoothly in ES6.
Arrow Functions	In ES5, both **function** and **return** keywords are used to define a function.	An arrow function is a new feature introduced in ES6 by which we don't require the **function** keyword to define the function.
Loops	In ES5, there is a use of **for** loop to iterate over elements.	ES6 introduced the concept of **for...of** loop to perform an iteration over the values of the iterable objects

Table Difference ES5 and ES6

1. **8 Object Oriented in Java Script**

There are certain features or mechanisms which makes a Language Object-Oriented like:

- Object
- Classes
- Encapsulation
- Inheritance

Let's dive into the details of each one of them and see how they are implemented in JavaScript.

1. Object– An Object is a unique entity that contains properties and methods. For example "car" is a real life Object, which has some characteristics like color, type, model, horsepower and performs certain actions like drive. The characteristics of an Object are called Properties in Object-Oriented Programming and the actions are called methods. An Object is an instance of a class. Objects are everywhere in JavaScript, almost every element is an Object whether it is a function, array, or string.

Note: A Method in javascript is a property of an object whose value is a function.

Object can be created in two ways in JavaScript:

Using an Object Literal

```
//Defining object

let person = {

   first_name:'Mukul',

   last_name: 'Latiyan',
```

```
    //method
    getFunction : function(){
        return (`The name of the person is
         ${person.first_name} ${person.last_name}`)
    },
    //object within object
    phone_number : {
        mobile:'12345',
        landline:'6789'
    }
}
console.log(person.getFunction());
console.log(person.phone_number.landline);
```

Using an Object Constructor:

```
//using a constructor
function person(first_name,last_name){
  this.first_name = first_name;
  this.last_name = last_name;
}
//creating new instances of person object
let person1 = new person('Mukul','Latiyan');
let person2 = new person('Rahul','Avasthi');
```

```
console.log(person1.first_name);

console.log(`${person2.first_name} ${person2.last_name}`);
```

2. Classes

Classes are blueprint of an Object. A class can have many Objects because class is a template while Object are instances of the class or the concrete implementation.

Before we move further into implementation, we should know unlike other Object Oriented Language there are no classes in JavaScript we have only Object. To be more precise, JavaScript is a prototype based Object Oriented Language, which means it doesn't have classes, rather it defines behaviors using a constructor function and then reuse it using the prototype.

Note: Even the classes provided by ECMA2015 are objects.

JavaScript classes, introduced in ECMAScript 2015, are primarily syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax is not introducing a new object-oriented inheritance model to JavaScript. JavaScript classes provide a much simpler and clearer syntax to create objects and deal with inheritance.

–Mozilla Developer Network

Example:

Lets use ES6 classes then we will look at the traditional way of defining an Object and simulate them as classes.

```
// Defining class using es6

class Vehicle {

  constructor(name, maker, engine) {

    this.name = name;
```

```
    this.maker =  maker;
    this.engine = engine;
  }
  getDetails(){
      return (`The name of the bike is ${this.name}.`)
  }
}
// Making object with the help of the constructor
let bike1 = new Vehicle('Hayabusa', 'Suzuki', '1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');

console.log(bike1.name);    // Hayabusa
console.log(bike2.maker);   // Kawasaki
console.log(bike1.getDetails());
```

Traditional Way.

```
// Defining class in a Traditional Way.
function Vehicle(name,maker,engine){
    this.name = name,
    this.maker = maker,
    this.engine = engine
```

```
};

Vehicle.prototype.getDetails = function(){
    console.log('The name of the bike is '+ this.name);
}

let bike1 = new Vehicle('Hayabusa','Suzuki','1340cc');
let bike2 = new Vehicle('Ninja','Kawasaki','998cc');

console.log(bike1.name);
console.log(bike2.maker);
console.log(bike1.getDetails());
```

Inheritance –

It is a concept in which some properties and methods of an Object are being used by another Object. Unlike most of the OOP languages where classes inherit classes, JavaScript Objects inherit Objects i.e. certain features (property and methods) of one object can be reused by other Objects.

Let's understand inheritance with an example:

```
//Inheritance example
class person{
   constructor(name){
      this.name = name;
   }
   //method to return the string
   toString(){
      return (`Name of person: ${this.name}`);
   }
}
class student extends person{
   constructor(name,id){
      //super keyword for calling the above class constructor
      super(name);
```

```
        this.id = id;
    }
    toString(){
        return (`${super.toString()},Student ID: ${this.id}`);
    }
}
let student1 = new student('Mukul',22);
console.log(student1.toString());
```

1.9 Java Script Arrow Function

Arrow functions are introduced in ES6, which provides you a more accurate way to write the functions in JavaScript. They allow us to write smaller function syntax. Arrow functions make your code more readable and structured.

Arrow functions are **anonymous functions** (the functions without a name and not bound with an identifier). They don't return any value and can declare without the function keyword. Arrow functions cannot be used as the constructors. The context within the arrow functions is lexically or statically defined. They are also called as **Lambda Functions** in different languages.

Arrow functions do not include any prototype property, and they cannot be used with the new keyword.

```
1. const functionName = (arg1, arg2, ?..) => {
2.     //body of the function
3. }
```

There are three parts to an Arrow Function or Lambda Function:

- **Parameters:** Any function may optionally have the parameters.
- **Fat arrow notation/lambda notation:** It is the notation for the **arrow (=>).**
- **Statements:** It represents the instruction set of the function.

Let us try to understand it with an example.

In the following example, we are defining three functions that show **Function Expression, Anonymous Function, and Arrow Function.**

```
// function expression
1.
2. var myfun1 = function show() {
3.  console.log("It is a Function Expression");
4. }
5.
6. // Anonymous function
7.
```

```
var myfun2 = function () {
    console.log("It is an Anonymous Function");
}

//Arrow function

var myfun3 = () => {
    console.log("It is an Arrow Function");
};

myfun1();
myfun2();
myfun3();
```

Output

```
It is a Function Expression
It is an Anonymous Function
It is an Arrow Function
```

Syntactic Variations

There are some syntactic variations for the arrow functions that are as follows:

- **Optional parentheses**

For example

```
var show = (a,b,c) => {
    console.log(a + " " + b + " " + c );
}
show(100,200,300);
```

Output

```
100 200 300
```

1.10 Java Script Generator Function

Like Python Generators, JavaScript also supports Generator functions and Generator Objects.

Generator-Function : 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 function's execution and sends a value back to caller, but retains enough state to enable function to resume where it is left off. When resumed, the function continues execution immediately after the last yield run.

Syntax :

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

Generator-Object : Generator functions return a generator object. Generator objects are used either by calling the next method on the generator object or using the generator object in a "for of" loop (as shown in the above program)

The Generator object is returned by a generating function and it conforms to both the iterable protocol and the iterator protocol.

Example 1 Generator Function

```
<script>
// Generate Function generates three
// different numbers in three calls
function * fun()
{
	yield 10;
	yield 20;
	yield 30;
}

// Calling the Generate Function
var gen = fun();
document.write(gen.next().value);
document.write("<br>");
document.write(gen.next().value);
document.write("<br>");
document.write(gen.next().value);
</script>
```

Example 2 Generator Function

```
<script>// Generate Function generates an
// infinite series of Natural Numbers
function * nextNatural()
{
	var naturalNumber = 1;

	// Infinite Generation
	while (true) {
		yield naturalNumber++;
```

```
        }
}

// Calling the Generate Function
var gen = nextNatural();

// Loop to print the first
// 10 Generated number
for (var i = 0; i < 10; i++) {

        // Generating Next Number
        document.write(gen.next().value);

        // New Line
        document.write("<br>");
}
</script>
```

1.11 Java Script Promise

A JavaScript Promise object can be:

Pending
Fulfilled
Rejected
The Promise object supports two properties: state and result.

- While a Promise object is "pending" (working), the result is undefined.
- When a Promise object is "fulfilled", the result is a value.
- When a Promise object is "rejected", the result is an error object.

myPromise.state myPromise.result

"pending" undefined
"fulfilled" a result value
"rejected" an error object
You cannot access the Promise properties state and result.

You must use a Promise method to handle promises.

Promise How To
Here is how to use a Promise:

```
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Promise.then() takes two arguments, a callback for success and another for failure.

Both are optional, so you can add a callback for success or failure only.

Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;

// The producing code (this may take some time)

  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
```

```
  }
});

myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

Java Script promise example for wating for a File

```
<p id="demo"></p>

<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function getFile(myCallback) {
  let req = new XMLHttpRequest();
  req.open('GET', "mycar.html");
  req.onload = function() {
    if (req.status == 200) {
      myCallback(this.responseText);
    } else {
      myCallback("Error: " + req.status);
    }
  }
  req.send();
}

getFile(myDisplayer);
</script>

</body>
```

```
</html>
```

1.12 Java Script Fetch

- The fetch() method starts the process of fetching a resource from a server.
- The fetch() method returns a Promise that resolves to a Response object.
- No need for XMLHttpRequest anymore.

Example 1

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Fetch API</h1>
<h2>The fetch() Method</h2>
<p id="demo">Fetch a file to change this text.</p>

<script>
getText("fetch_info.txt");

async function getText(file) {
  let myObject = await fetch(file);
  let myText = await myObject.text();
  document.getElementById("demo").innerHTML = myText;
}
</script>

</body>
</html>
```

Note – Please refer class Notes and Example Code share to create Fetch APi Code to get the data from Mysql Server using php code

1.13 References

1. MHSS Class and Practical's Notes.
2. Tutorials List - Javatpoint
3. https://www.w3schools.com/
4. https://www.geeksforgeeks.org/
5. https://www.tutorialspoint.com/
6. https://www.cloudflare.com/
7. Wikipedia