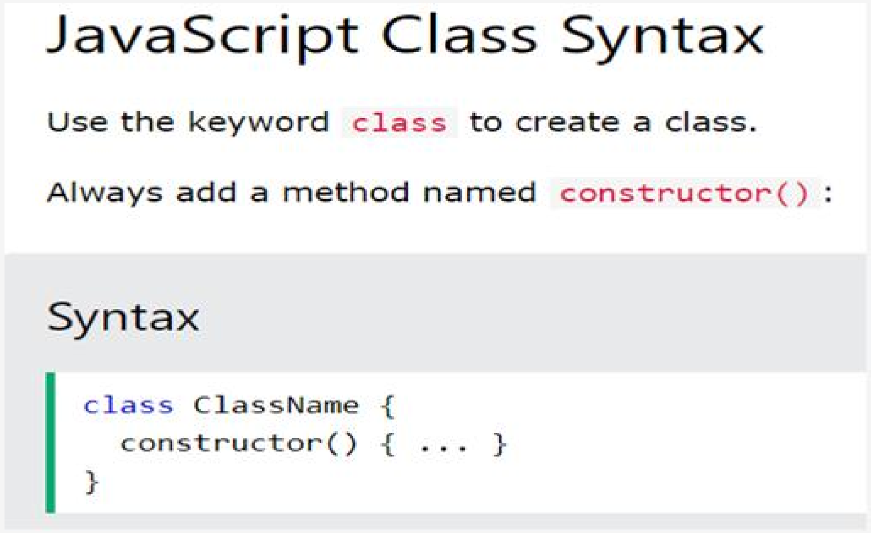
# JavaScript –Classes and Inheritance

Vijesh M. Nair
Assistant Professor
Dept. of CSE (AI-ML)

# JavaScript Classes

ECMAScript 2015, also known as ES6, introduced JavaScript Classes.

JavaScript Classes are templates for JavaScript Objects.



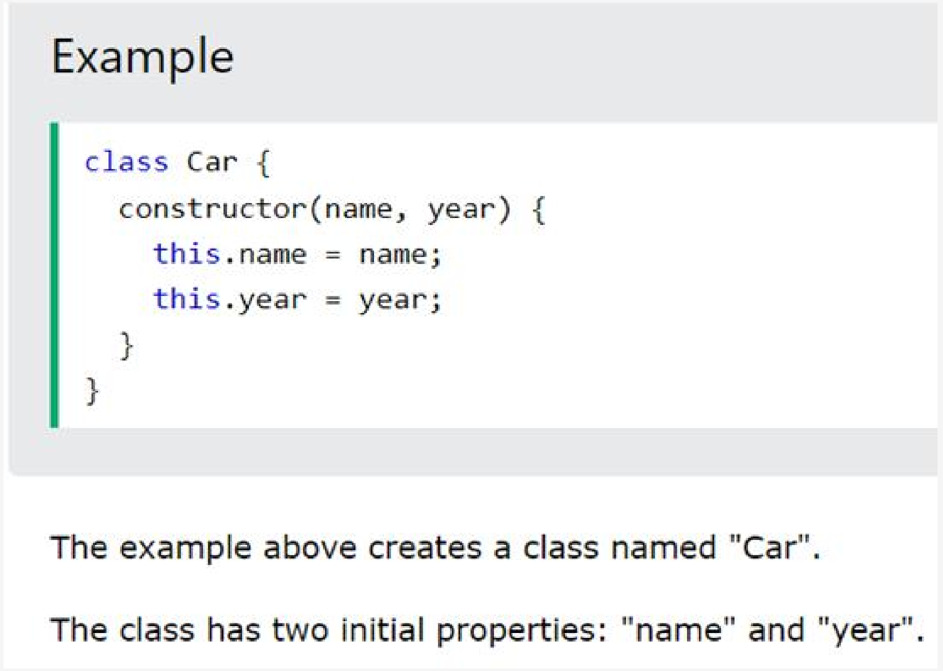
## JavaScript Class Syntax

Use the keyword `class` to create a class.

Always add a method named `constructor()`:

### Syntax

```
class ClassName {
  constructor() { ... }
}
```



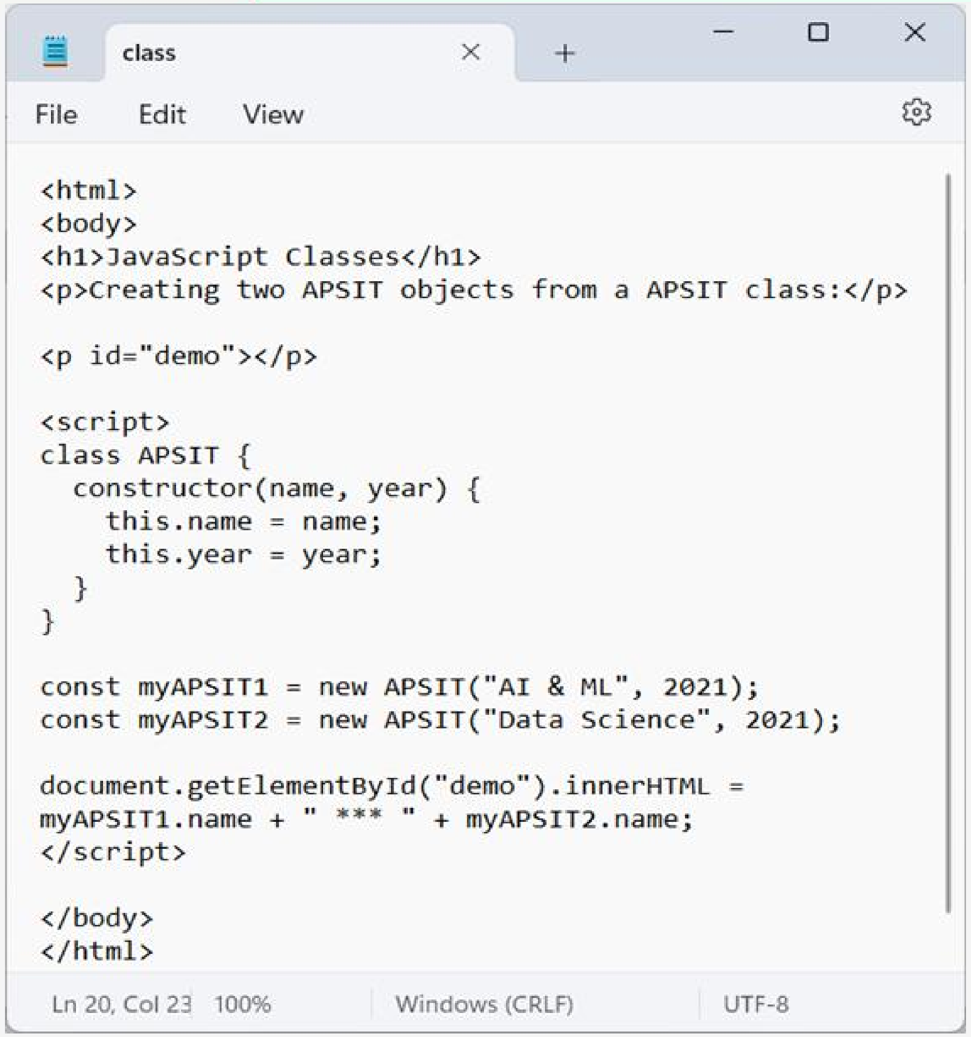
### Example

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

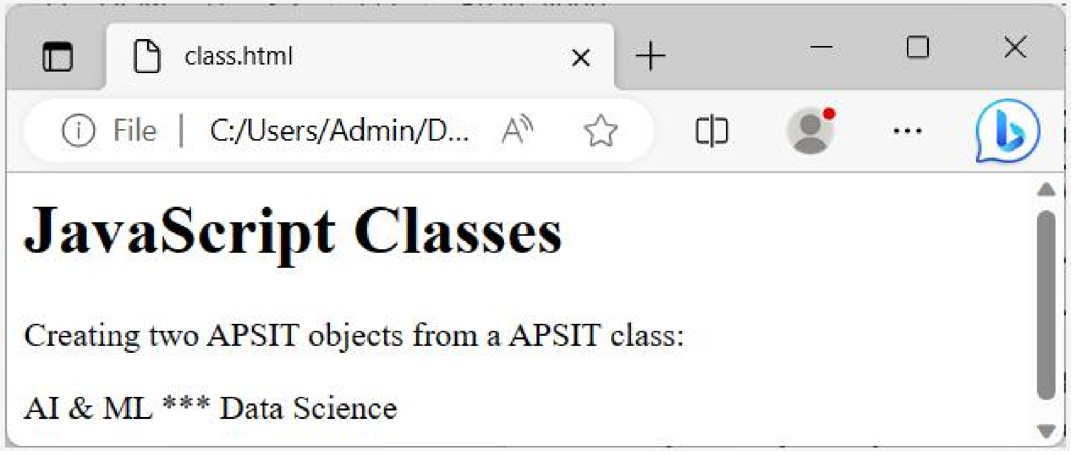
The example above creates a class named "Car".

The class has two initial properties: "name" and "year".

## JavaScript Classes

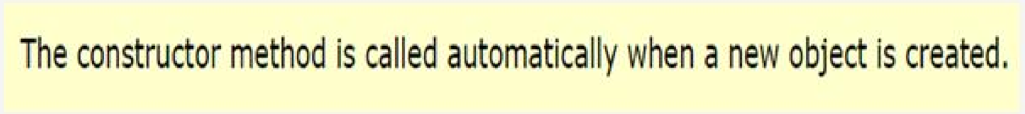


```
<html>
<body>
<h1>JavaScript Classes</h1>
<p>Creating two APSIT objects from a APSIT class:</p>

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

<script>
class APSIT {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}

const myAPSIT1 = new APSIT("AI & ML", 2021);
const myAPSIT2 = new APSIT("Data Science", 2021);

document.getElementById("demo").innerHTML =
myAPSIT1.name + " *** " + myAPSIT2.name;
</script>

</body>
</html>
```

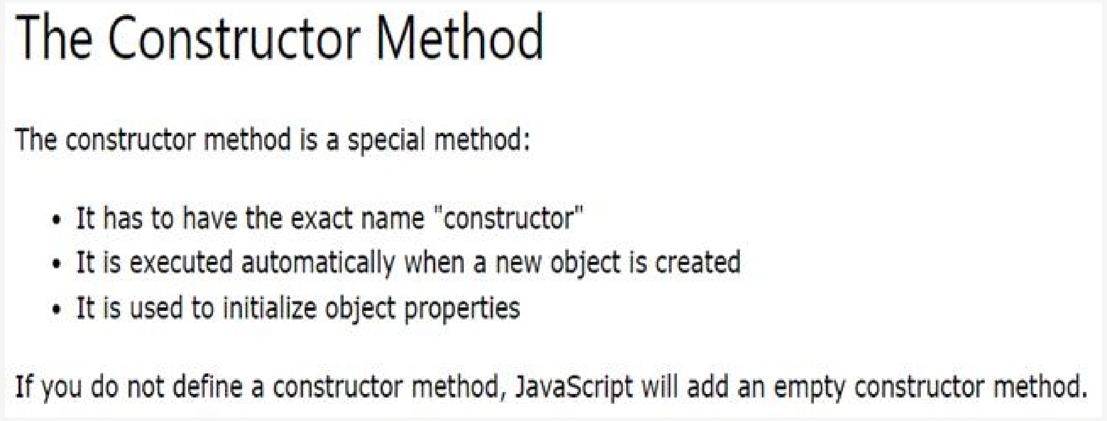


```
JavaScript Classes

Creating two APSIT objects from a APSIT class:

AI & ML *** Data Science
```

# JavaScript Constructor and Class Methods



The constructor method is called automatically when a new object is created.

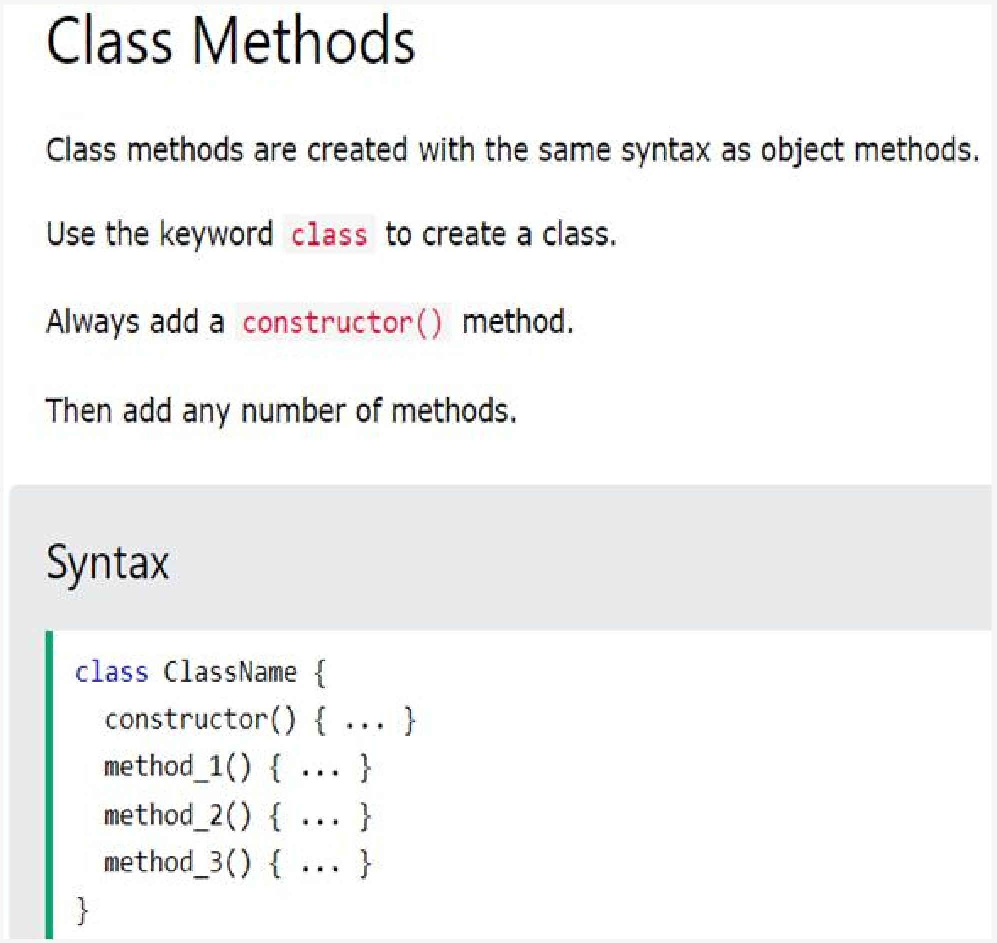


## The Constructor Method

The constructor method is a special method:

- It has to have the exact name "constructor"
- It is executed automatically when a new object is created
- It is used to initialize object properties

If you do not define a constructor method, JavaScript will add an empty constructor method.



## Class Methods

Class methods are created with the same syntax as object methods.

Use the keyword `class` to create a class.

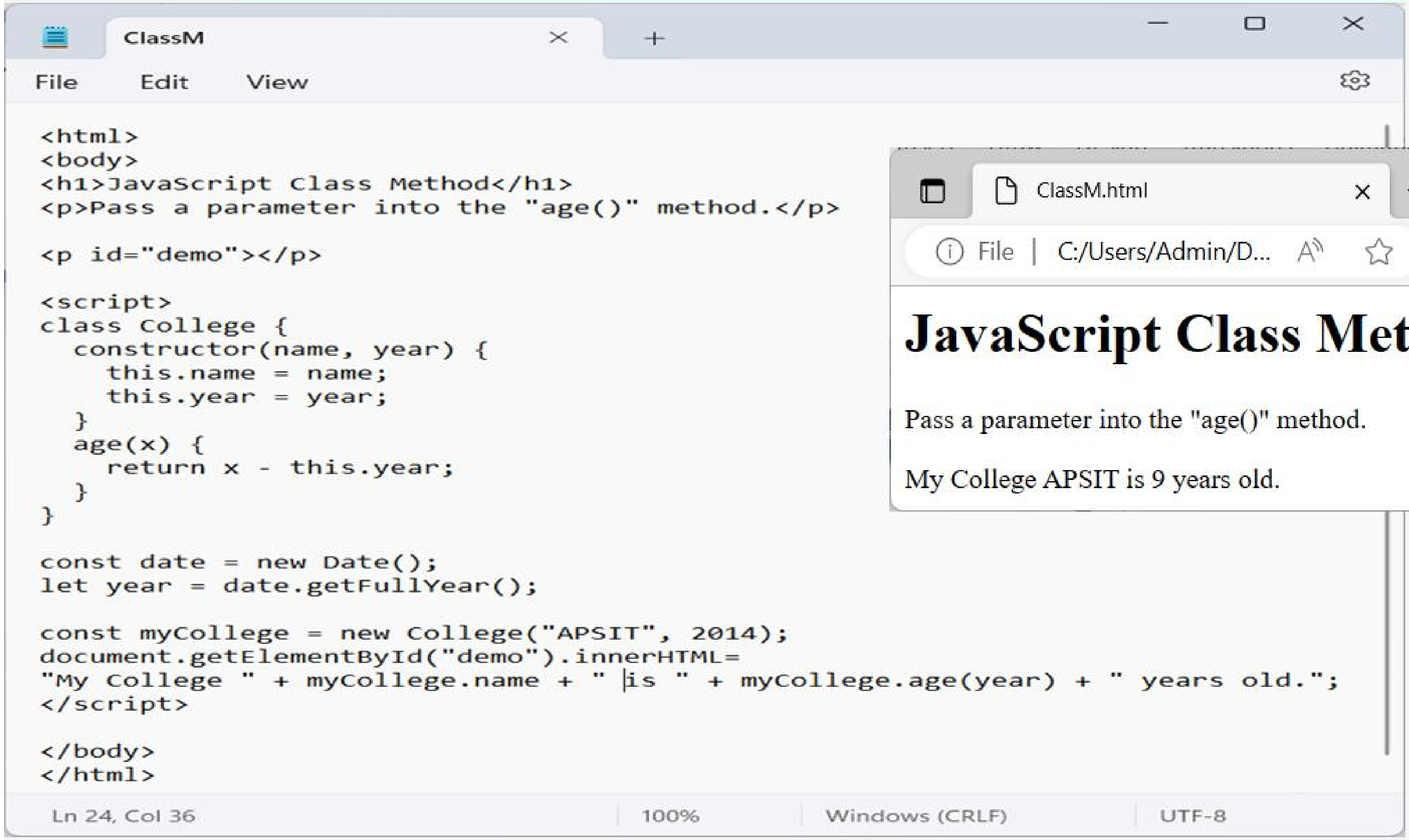
Always add a `constructor()` method.

Then add any number of methods.

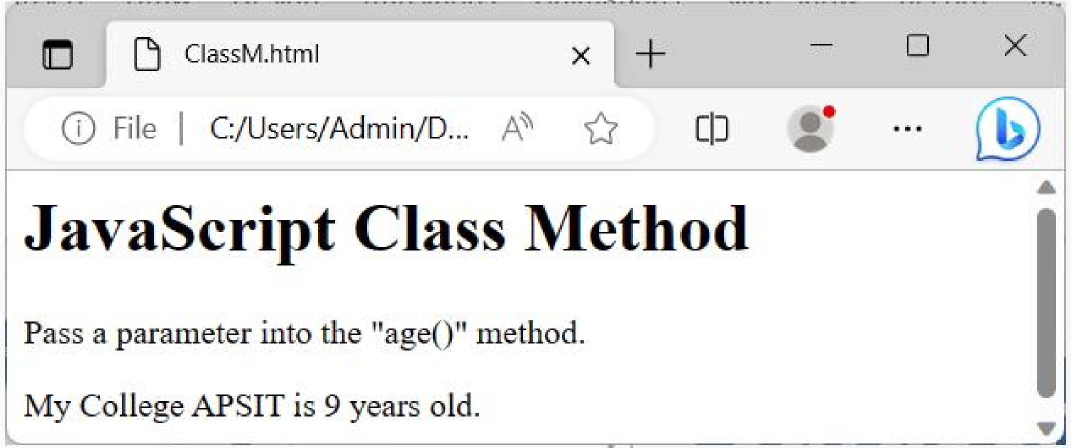
### Syntax

```
class ClassName {
  constructor() { ... }
  method_1() { ... }
  method_2() { ... }
  method_3() { ... }
}
```

## JavaScript Constructor and Class Methods



```
<html>
<body>
<h1>JavaScript Class Method</h1>
<p>Pass a parameter into the "age()" method.</p>

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

<script>
class College {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age(x) {
    return x - this.year;
  }
}

const date = new Date();
let year = date.getFullYear();

const myCollege = new College("APSIT", 2014);
document.getElementById("demo").innerHTML=
"My College " + myCollege.name + " is " + myCollege.age(year) + " years old.";
</script>

</body>
</html>
```



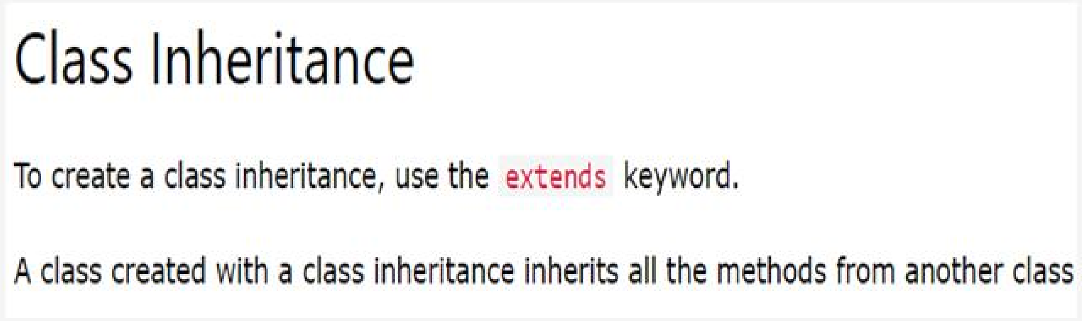
# JavaScript Class Method

Pass a parameter into the "age()" method.

My College APSIT is 9 years old.

## JavaScript Inheritance

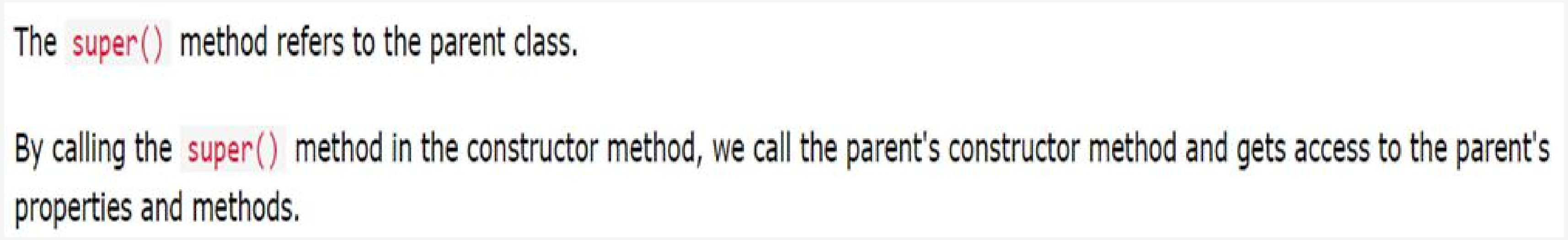
Inheritance is useful for code reusability: reuse properties and methods of an existing class when you create a new class.



### Class Inheritance

To create a class inheritance, use the `extends` keyword.

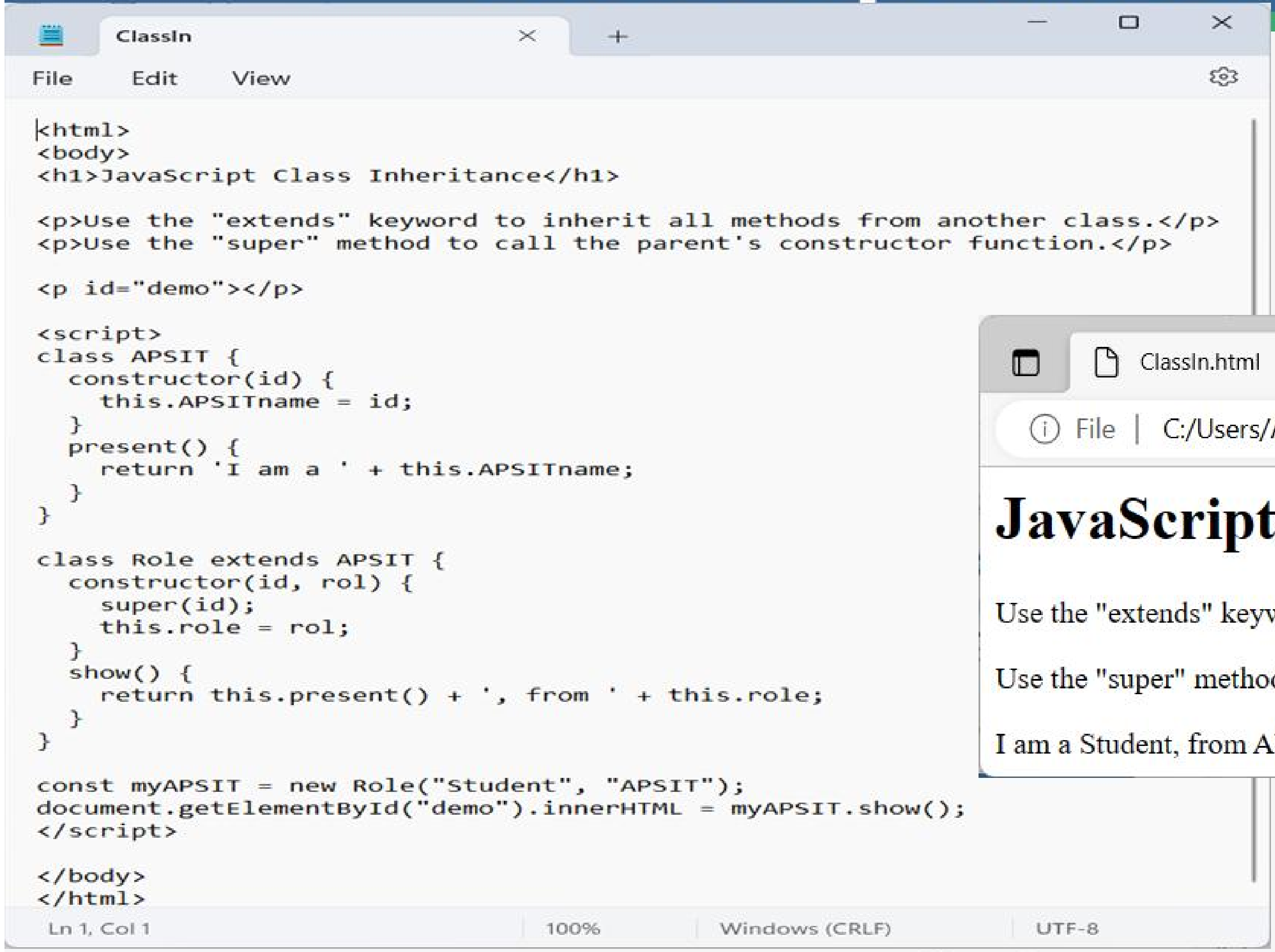
A class created with a class inheritance inherits all the methods from another class



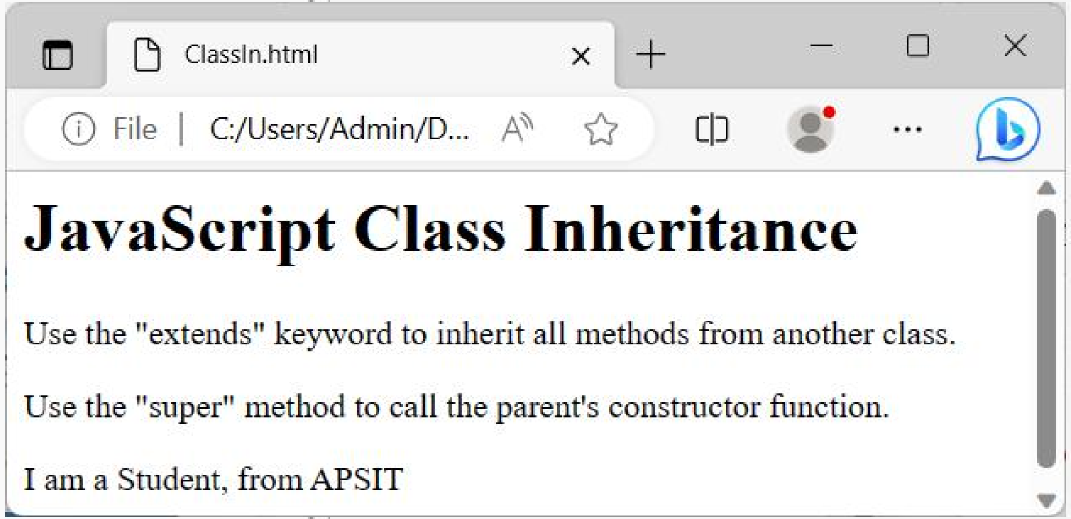
The `super()` method refers to the parent class.

By calling the `super()` method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

# JavaScript Inheritance



```
<html>
<body>
<h1>JavaScript Class Inheritance</h1>

<p>Use the "extends" keyword to inherit all methods from another class.</p>
<p>Use the "super" method to call the parent's constructor function.</p>

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

<script>
class APSIT {
  constructor(id) {
    this.APSITname = id;
  }
  present() {
    return 'I am a ' + this.APSITname;
  }
}

class Role extends APSIT {
  constructor(id, rol) {
    super(id);
    this.role = rol;
  }
  show() {
    return this.present() + ', from ' + this.role;
  }
}

const myAPSIT = new Role("Student", "APSIT");
document.getElementById("demo").innerHTML = myAPSIT.show();
</script>

</body>
</html>
```



## JavaScript Class Inheritance

Use the "extends" keyword to inherit all methods from another class.

Use the "super" method to call the parent's constructor function.

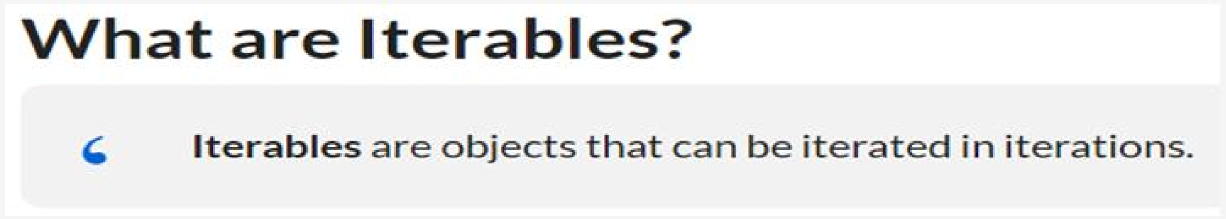
I am a Student, from APSIT

## JavaScript – Iterators

# JavaScript Iterators

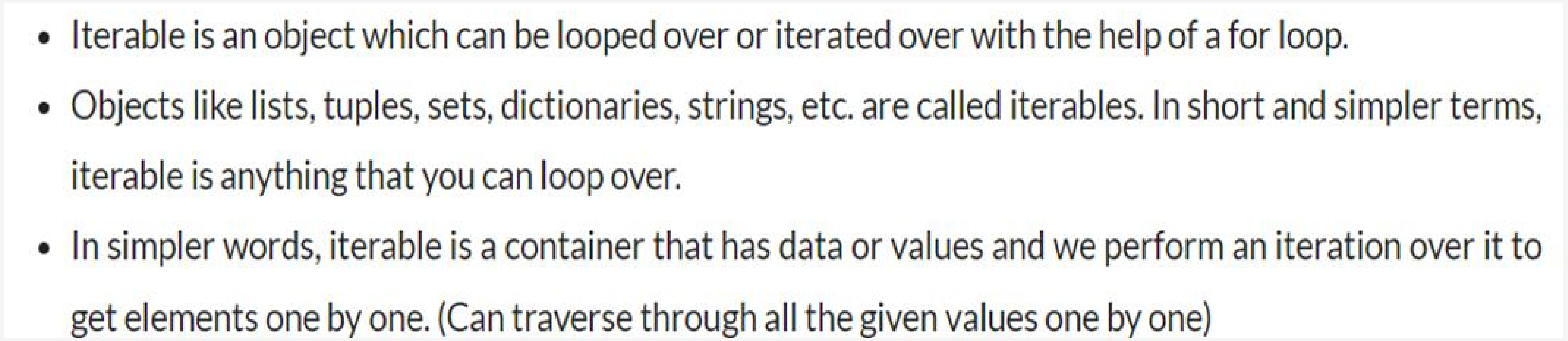
**What is Iteration?**

- ➢ Iteration means **'repeating steps'** in layman's language.
- ➢ In Programming, Iteration is defined as a repetition of a block of code a specified number of times.
- ➢ To achieve the iterations, we can use loops such as **for loop**, etc.



## What are Iterables?

> **Iterables** are objects that can be iterated in iterations.



- Iterable is an object which can be looped over or iterated over with the help of a for loop.
- Objects like lists, tuples, sets, dictionaries, strings, etc. are called iterables. In short and simpler terms, iterable is anything that you can loop over.
- In simpler words, iterable is a container that has data or values and we perform an iteration over it to get elements one by one. (Can traverse through all the given values one by one)

# JavaScript Iterators - Iterables

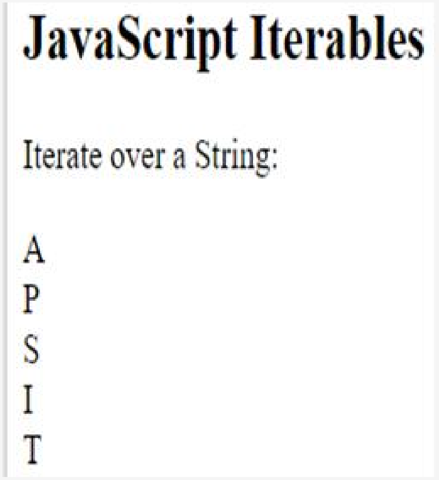
Iterable objects are objects that can be iterated over with `for..of`.

Technically, iterables must implement the `Symbol.iterator` method.

*Example: Iterating Over a String*

```html
<html>
<body>
<h2>JavaScript Iterables</h2>
<p>Iterate over a String:</p>

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

<script>
// Create a String
const name = "APSIT";

// List all Elements
let text = ""
for (const x of name) {
  text += x + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

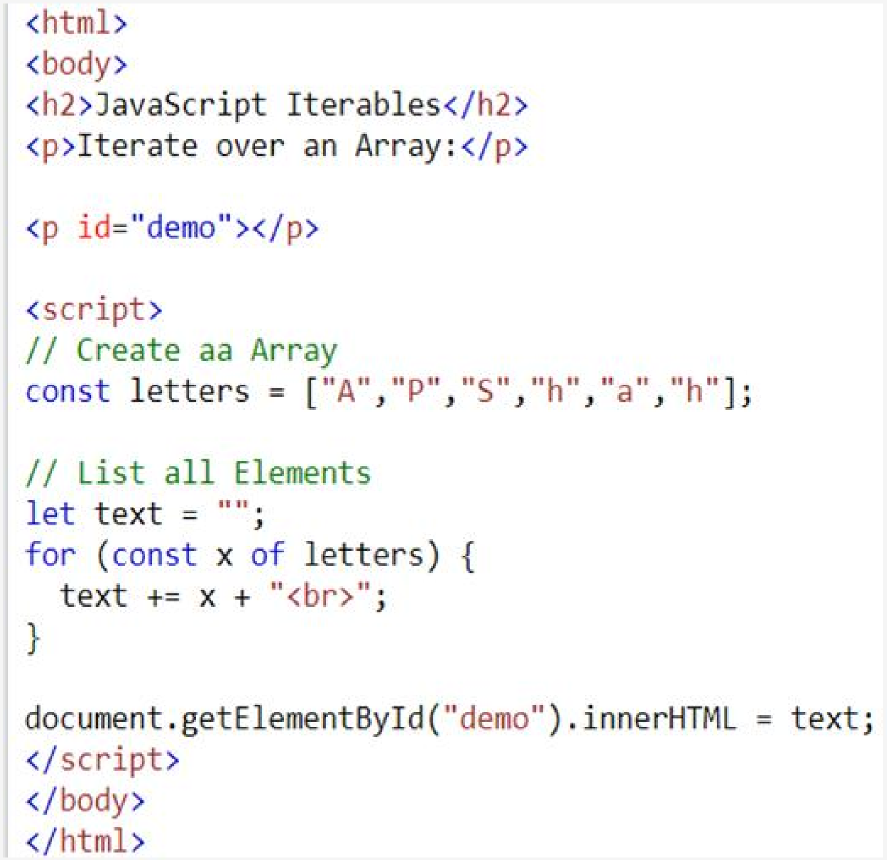
*Output:*



```
JavaScript Iterables

Iterate over a String:

A
P
S
I
T
```

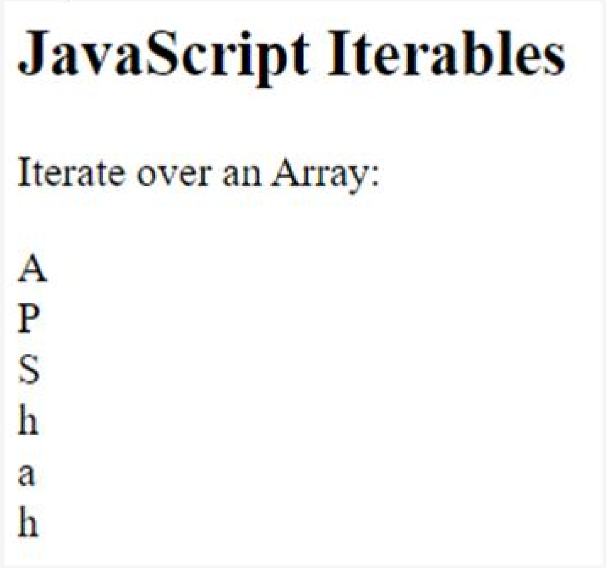
## JavaScript Iterators - Iterables

*Example: Iterating Over an array*



```
<html>
<body>
<h2>JavaScript Iterables</h2>
<p>Iterate over an Array:</p>

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

<script>
// Create aa Array
const letters = ["A","P","S","h","a","h"];

// List all Elements
let text = "";
for (const x of letters) {
  text += x + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

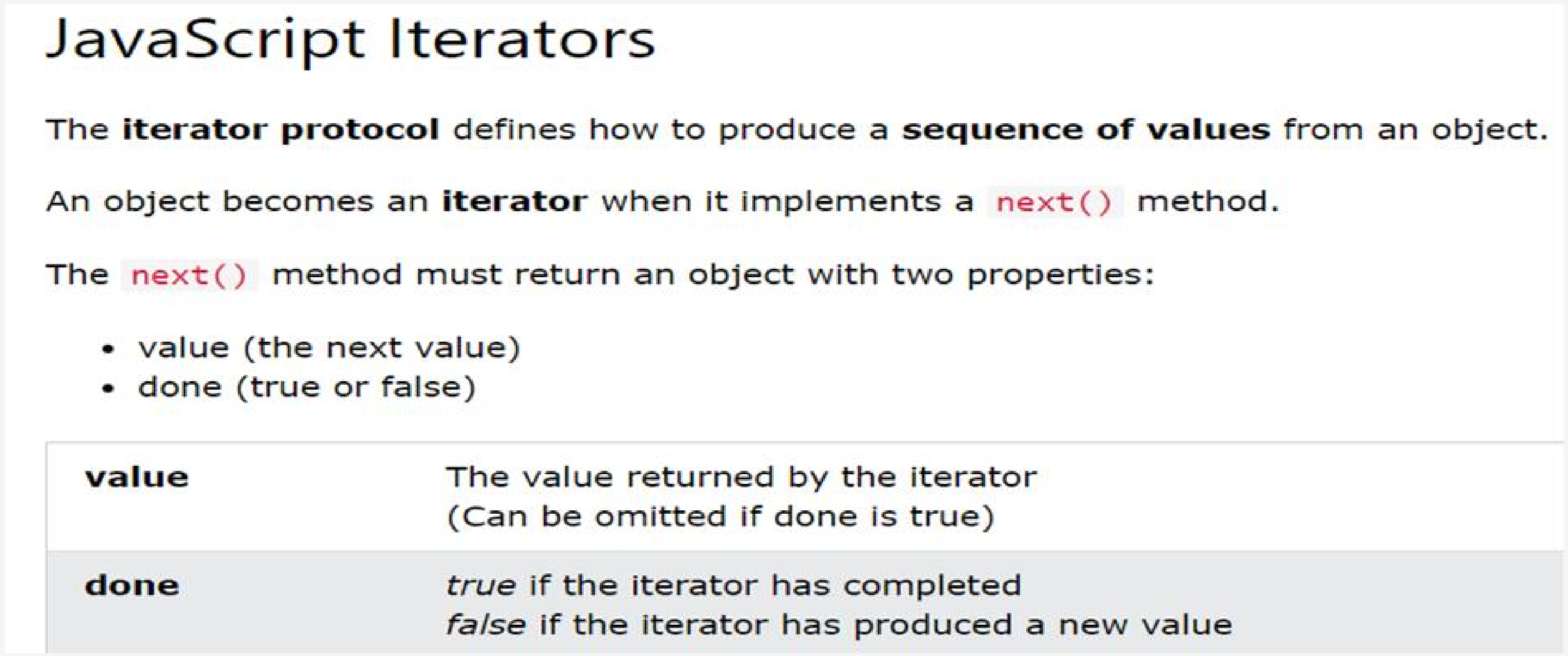
*Output:*



```
JavaScript Iterables

Iterate over an Array:

A
P
S
h
a
h
```

Vijesh Nair

# JavaScript Iterators



## JavaScript Iterators

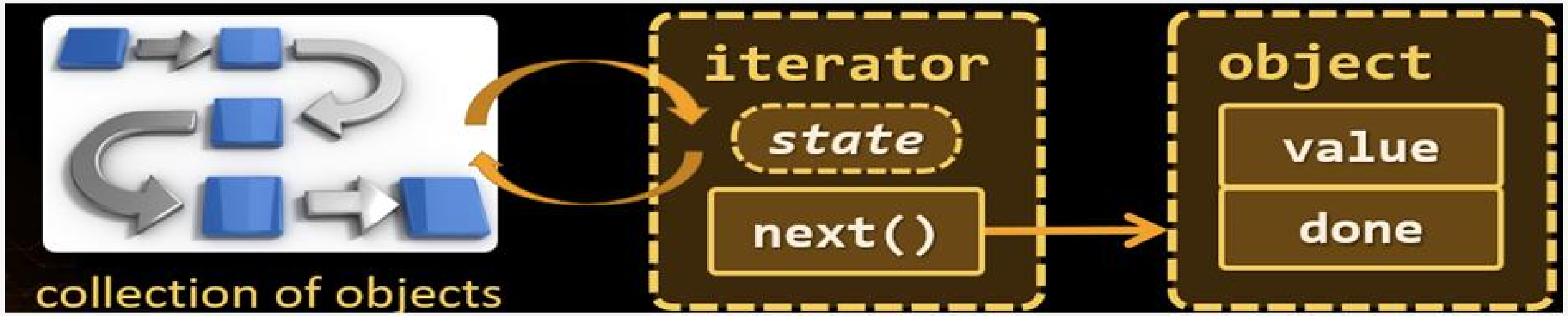
The **iterator protocol** defines how to produce a **sequence of values** from an object.

An object becomes an **iterator** when it implements a `next()` method.

The `next()` method must return an object with two properties:

- value (the next value)
- done (true or false)

| | |
|---|---|
| **value** | The value returned by the iterator<br>(Can be omitted if done is true) |
| **done** | *true* if the iterator has completed<br>*false* if the iterator has produced a new value |

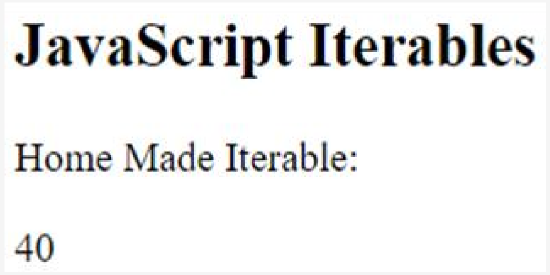


## JavaScript Iterators

### Example: Default Variable

```
<html>
<body>
<h2>JavaScript Iterables</h2>
<p>Home Made Iterable:</p>

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

<script>
// Home Made Iterable
function myNumbers() {
  let n = 0;
  return {
    next: function() {
      n += 10;
      return {value:n, done:false};
    }
  };
}

// Create Iterable
const n = myNumbers();
n.next(); // 10
n.next(); // 20
n.next(); // 30

document.getElementById("demo").innerHTML = n.next().value;
</script>
</body>
</html>
```

### Output:



```
JavaScript Iterables

Home Made Iterable:

40
```

The problem with a home made iterable:

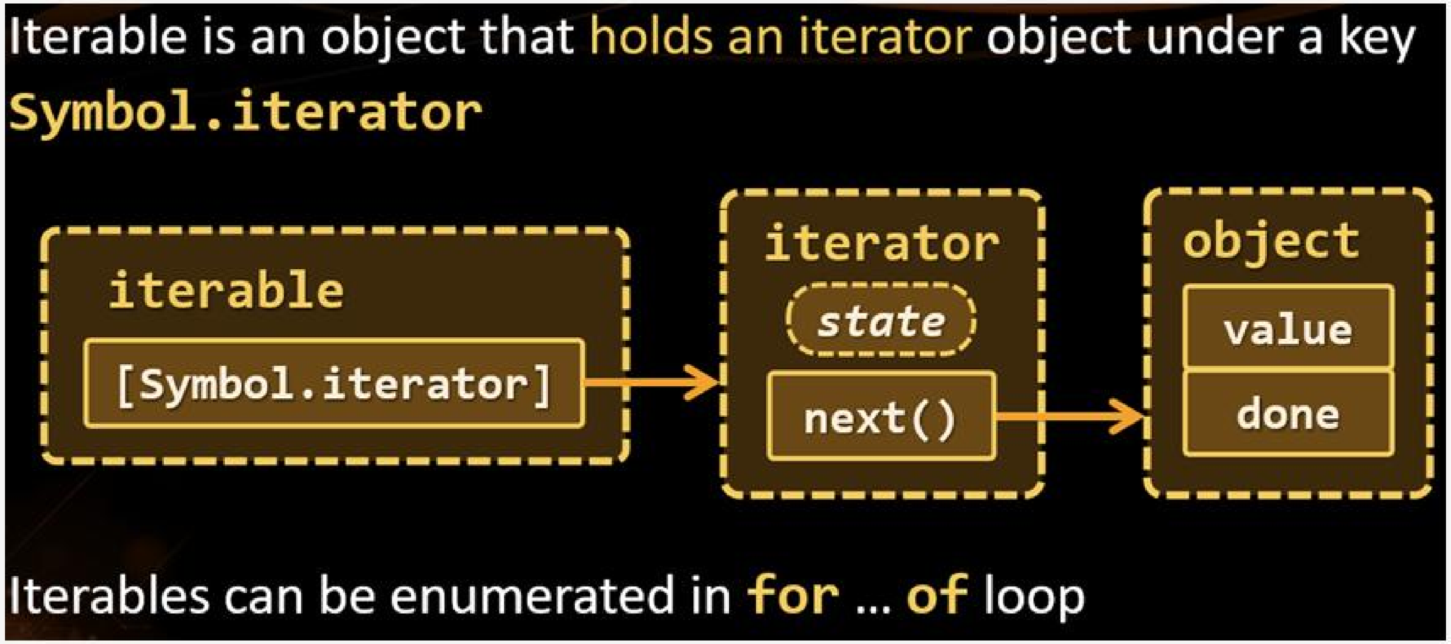
It does not support the JavaScript `for..of` statement.

## JavaScript `Symbol.iterator`

A JavaScript iterable is an object that has a **Symbol.iterator**.

The `Symbol.iterator` is a function that returns a `next()` function.

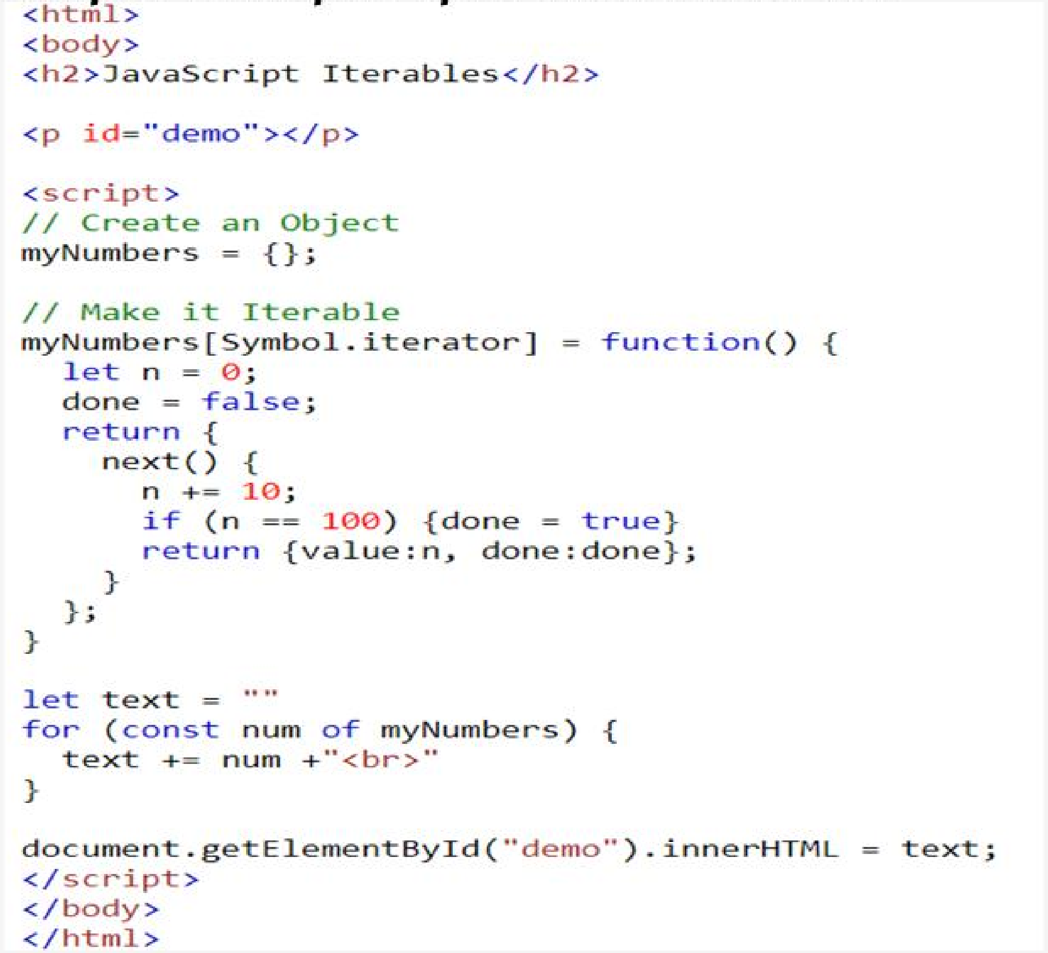
An iterable can be iterated over with the code: `for (const x of iterable) { }`


Iterable is an object that holds an iterator object under a key **Symbol.iterator**

Iterables can be enumerated in **for** ... **of** loop

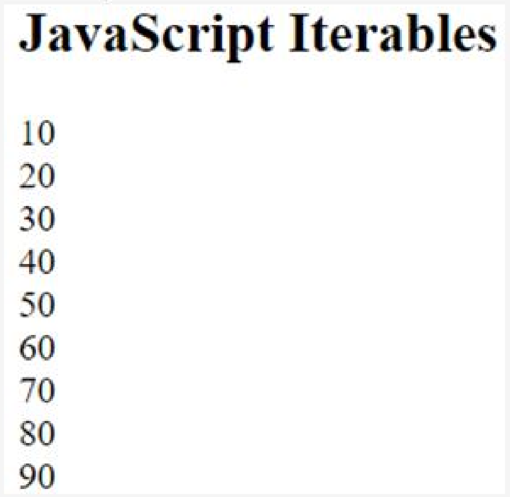
## JavaScript `Symbol.iterator`

*Example: Multiples of 10 till less than 100*



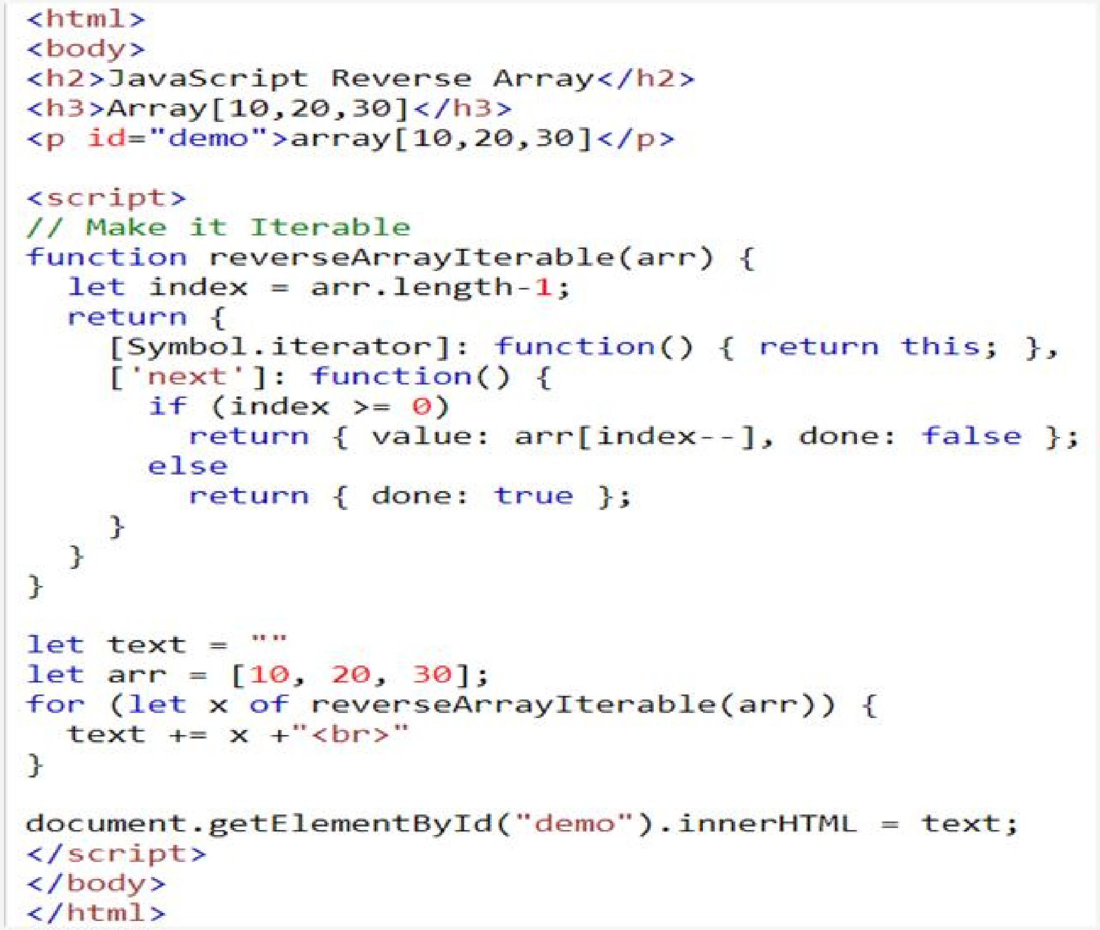
```
<html>
<body>
<h2>JavaScript Iterables</h2>

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

<script>
// Create an Object
myNumbers = {};

// Make it Iterable
myNumbers[Symbol.iterator] = function() {
  let n = 0;
  done = false;
  return {
    next() {
      n += 10;
      if (n == 100) {done = true}
      return {value:n, done:done};
    }
  };
}

let text = ""
for (const num of myNumbers) {
  text += num +"<br>"
}

document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

*Output:*



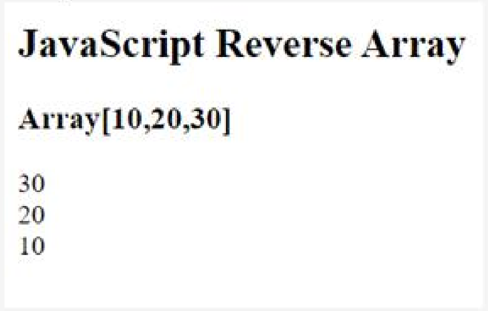
```
JavaScript Iterables

10
20
30
40
50
60
70
80
90
```

# JavaScript Iterators



```
<html>
<body>
<h2>JavaScript Reverse Array</h2>
<h3>Array[10,20,30]</h3>
<p id="demo">array[10,20,30]</p>

<script>
// Make it Iterable
function reverseArrayIterable(arr) {
  let index = arr.length-1;
  return {
    [Symbol.iterator]: function() { return this; },
    ['next']: function() {
      if (index >= 0)
        return { value: arr[index--], done: false };
      else
        return { done: true };
    }
  }
}

let text = ""
let arr = [10, 20, 30];
for (let x of reverseArrayIterable(arr)) {
  text += x +"<br>"
}

document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

*Output:*



```
JavaScript Reverse Array

Array[10,20,30]

30
20
10
```

# Thank You!