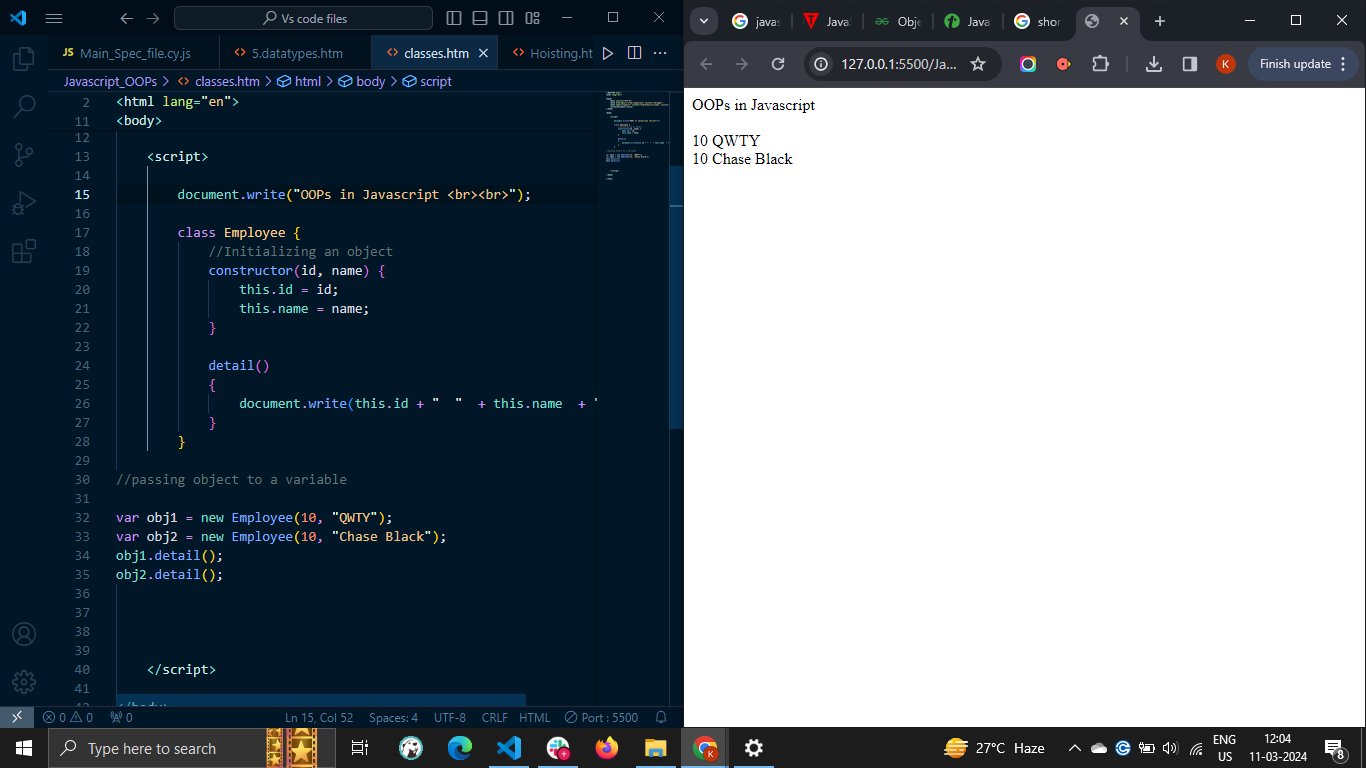
# JavaScript Classes

* In JavaScript, classes are a special type of functions. We can define the class just like function declarations and function expressions.
* The JavaScript class contains various class members within a body including methods or constructor. The class is executed in strict mode. So, the code containing the silent error or mistake throws an error.
* The JavaScript classes are a blueprint or template for object creation. They encapsulate the data and functions to manipulate that data.
* We can create the object using classes. Creating an object from a class is referred to as instantiating the class.
* In JavaScript, the classes are built on prototypes. The classes are introduced to JavaScript in ECMAScript 6 (ES6) in 2009.
* For example, you can think about writing code to represent the car entity. A code can contain the class having car properties. For different cars, you can create an instance of the class and initialize the car properties according to the car model.
* Before ES6, the constructor function was used to define a blueprint of the object. You can define the constructor function as shown below.
* The class syntax contains two components:
  + Class declarations
  + Class expressions

## Class Declarations

* A class can be defined by using a class declaration. A class keyword is used to declare a class with any particular name.
* According to JavaScript naming conventions, the name of the class always starts with an uppercase letter.



## The constructor() method

# When you use the function as an object blueprint, you can initialize the object properties inside the function body. Similarly, you need to use the constructor() method with the class to initialize the object properties.

# Whenever you create an instance of the class, it automatically invokes the constructor() method of the class.

# In below example, we use the constructor() method to create a Car class −

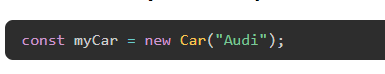
# 

The constructor() method has no specific name but can be created using the 'constructor' keyword. You can initialize the class properties using the 'this' keyword inside the constructor function.

## Creating JavaScript Objects

To create an object of a JavaScript class, we use **new** operator followed by the class name and a pair of parentheses. We can pass thee arguments to it also.

Let's create an object called **myCar** as follows −

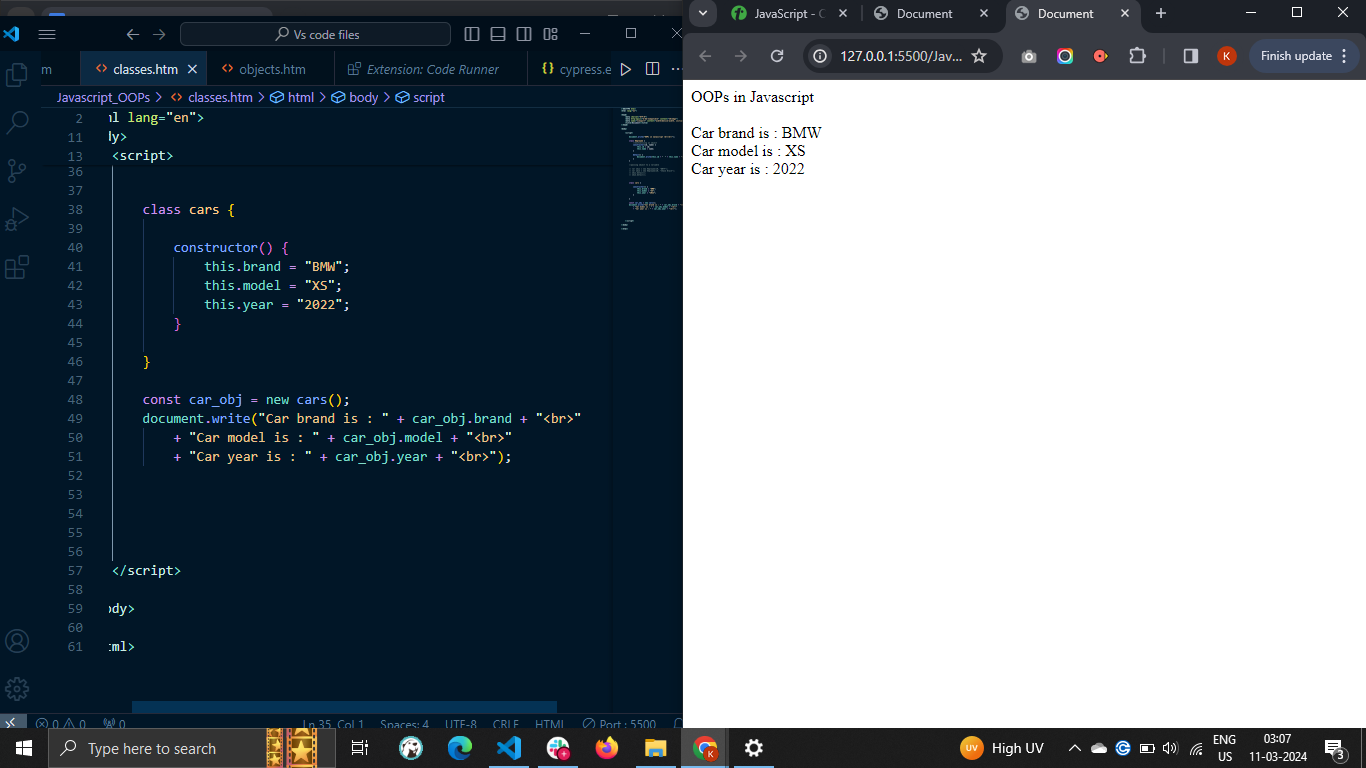


# The this keyword inside the constructor function refers to an object that is executing the current function.

### Example: Creating class objects without arguments

In the example below, we have defined the 'Car' class. The class contains the constructor and initializes the properties with default values.

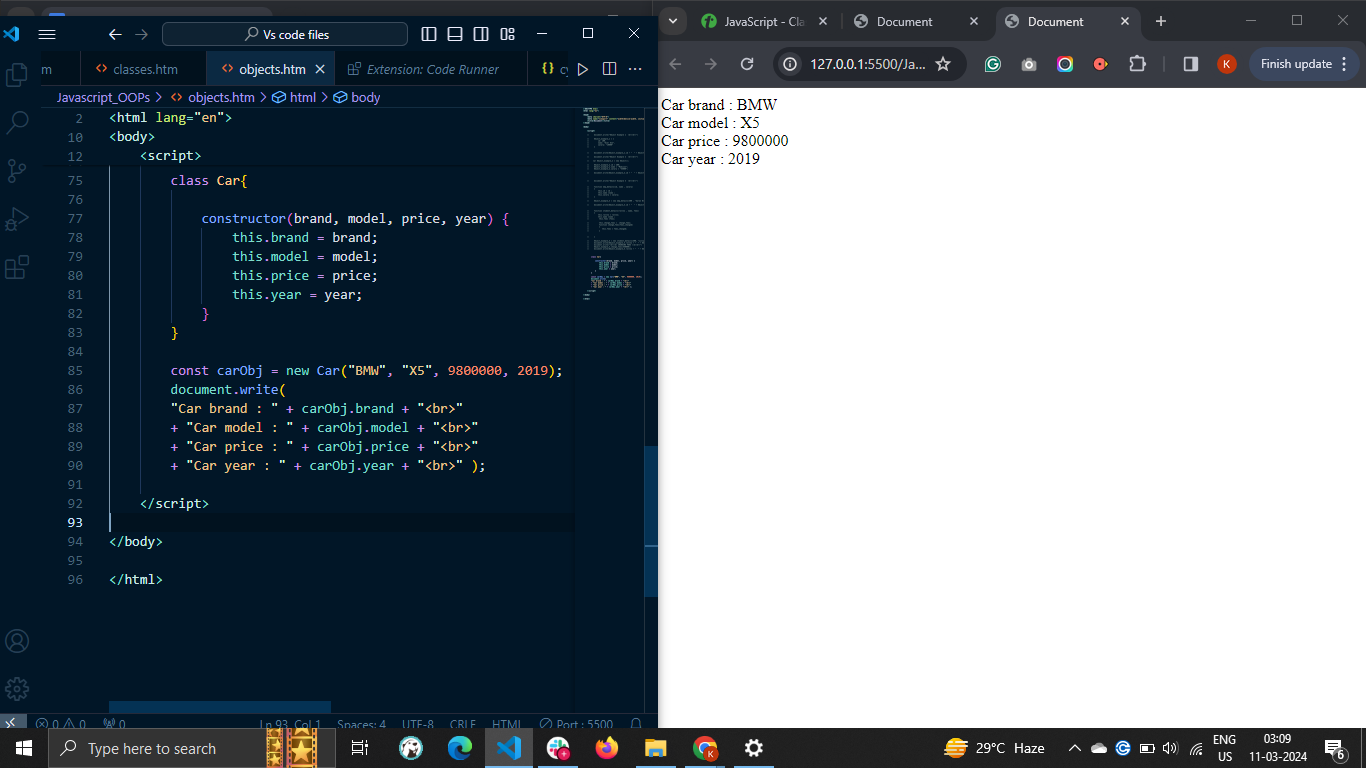
After that, we have created the instance of the class, and you can observe it in the output.



### Example: Creating class objects with arguments

In the example below, we have defined the 'Car' class. The constructor() method of the class takes 4 parameters and initializes the class properties with parametric values.

While creating the 'Car' class instance, we passed 4 arguments. In this way, you can initialize the class properties dynamically.



## JavaScript Class Methods

You can also define the methods inside the class, which can be accessed using the class instance.

### Syntax

Follow the syntax below to define methods inside the class.



In the above syntax, 'methodName' is a dynamic name of the method. To define a class method, you don't need to write any keyword like 'function' before the method name.

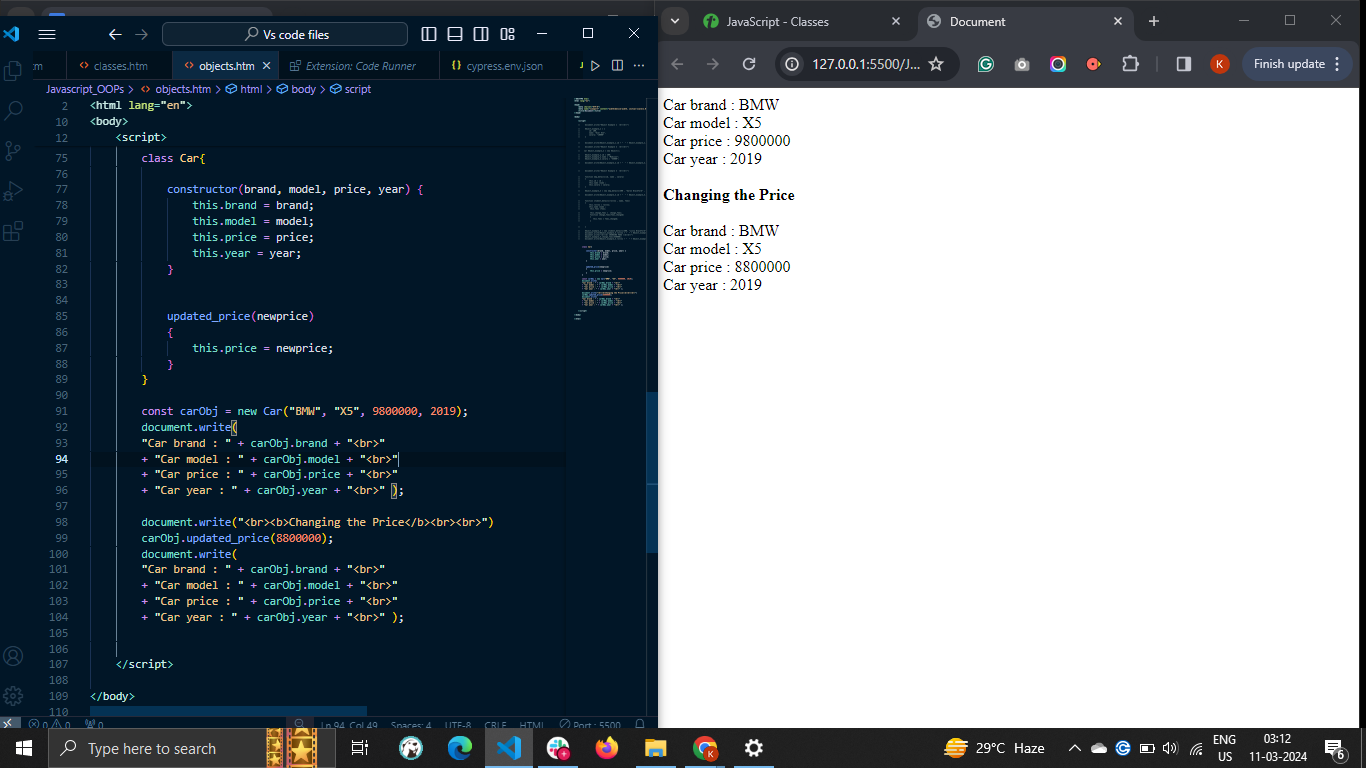
To invoke the class method, you need to use the instance of the class. Here, 'obj' is an instance of the class. You can also pass the parameters to the method.

### Example

The example below demonstrates how to pass parameters to the class methods.

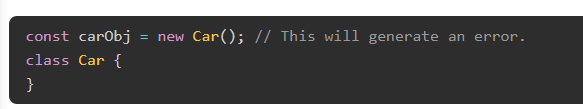
Here, we have defined the updateprice() method to update the price of the car. So, while invoking the updateprice() method, we pass the new price as an argument and use it inside the method body to update the price.

You can see the original and updated price of the car in the output.



## JavaScript Class Hoisting

In JavaScript, the declaration of the class is not hoisted at the top of the code. So, you always need to define the class before you use it.

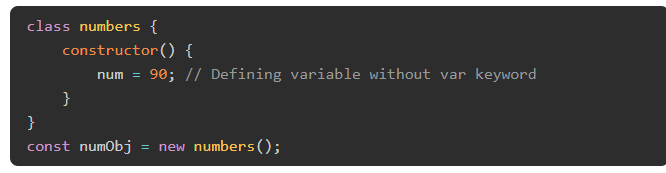


You can try to run the above code. It will generate a reference error as the car class is used before its initialization.

## Strict Mode with Classes

The strict mode is used to avoid unusual errors. The class code is always in the strict mode by default.

Let's understand it via the example below.

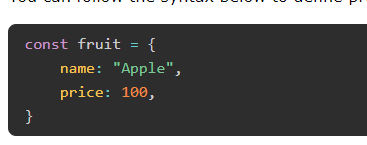


In the above code, we define the 'num' global variable in the constructor() method. In the strict mode of JavaScript, it is not allowed to define the variables without using the var, let, or const keywords. So, the above code will throw an error.

# JavaScript 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.
* An **object property** in JavaScript is a key: value pair, where key is a string and value can be anything. The key in key: value pair is also called property name. So the **properties** are association between key (or name) and value.
* An object is in other terms a collection of properties (key: value pairs). However, key: value pairs are not stored in the specific order in the object. To write an object syntax, the curly braces are used. Each key: value pair is written within curly braces separated by a comma.
* You can manipulate the object properties in JavaScript. For example, you can add, delete, or update the object's properties.

### Syntax

* You can follow the syntax below to define properties in the object.
* 
* In the above syntax, fruit is an object. The fruit object contains the name and price properties. The value of the name property is 'Apple’, and the price is 100.
* In an object, the key can either be a string or a symbol only. If you use another data type as a key, the object implicitly converts it into the string.
* The property value can be anything like an object, set, array, string, set, function, etc.

## Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

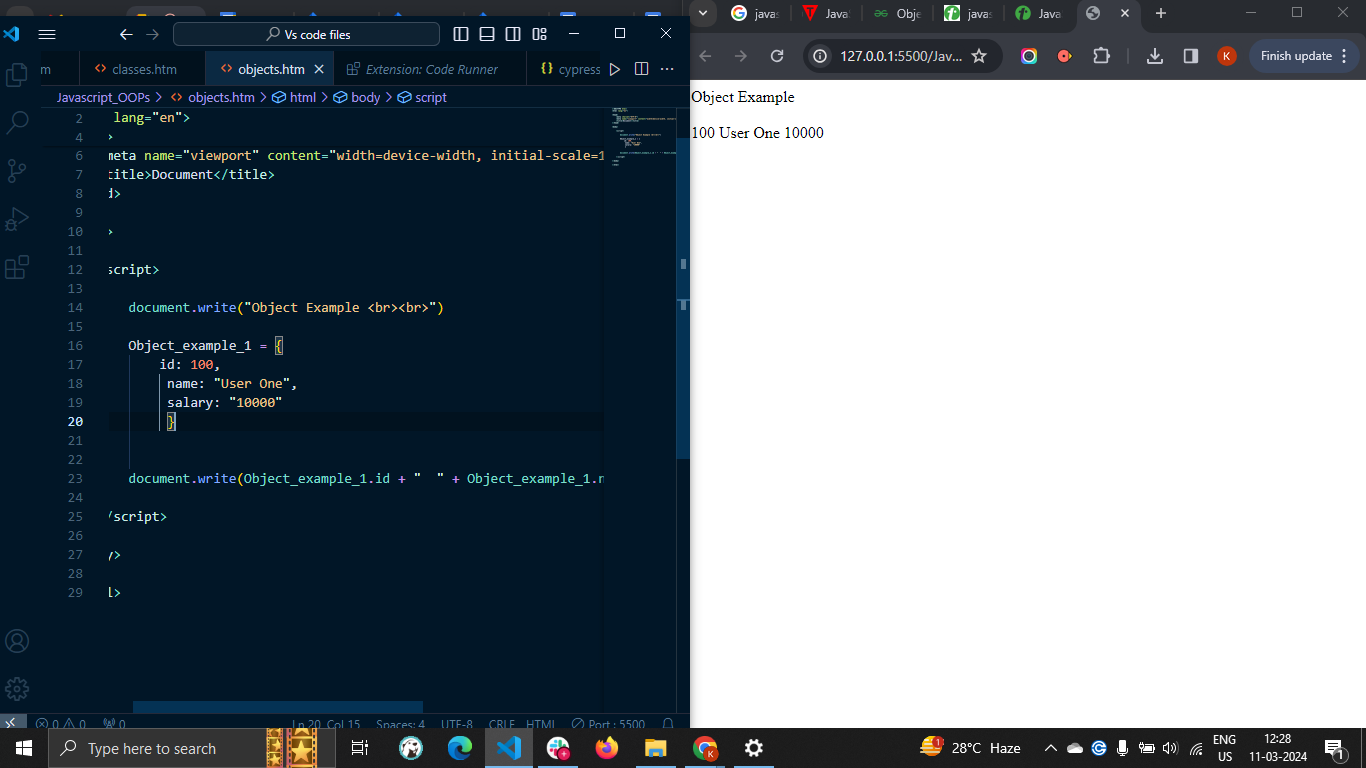
## 1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

1. object={property1:value1,property2:value2.....property:valueN}

As you can see, property and value is separated by : (colon).

Let’s see a simple example of creating object in JavaScript.

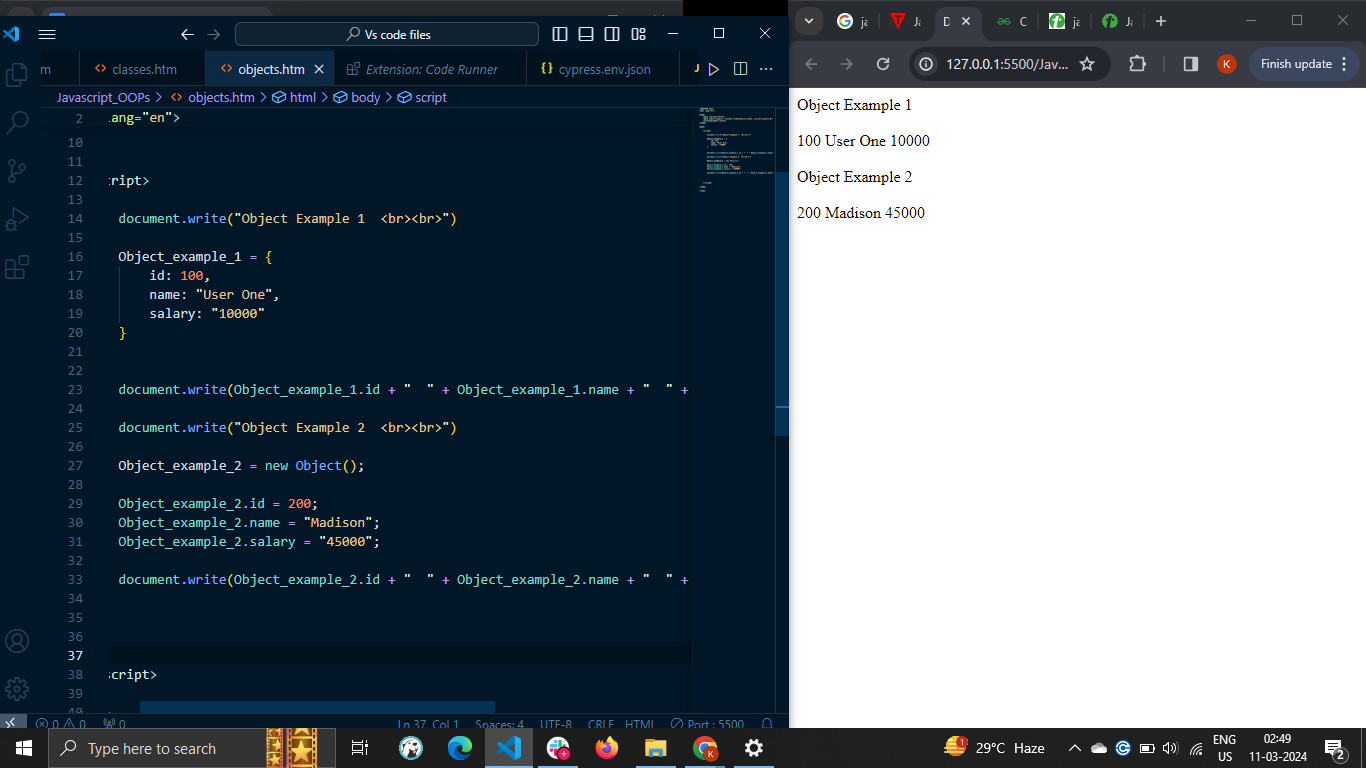
****

## 2) By creating instance of Object

**The syntax of creating object directly is given below:**

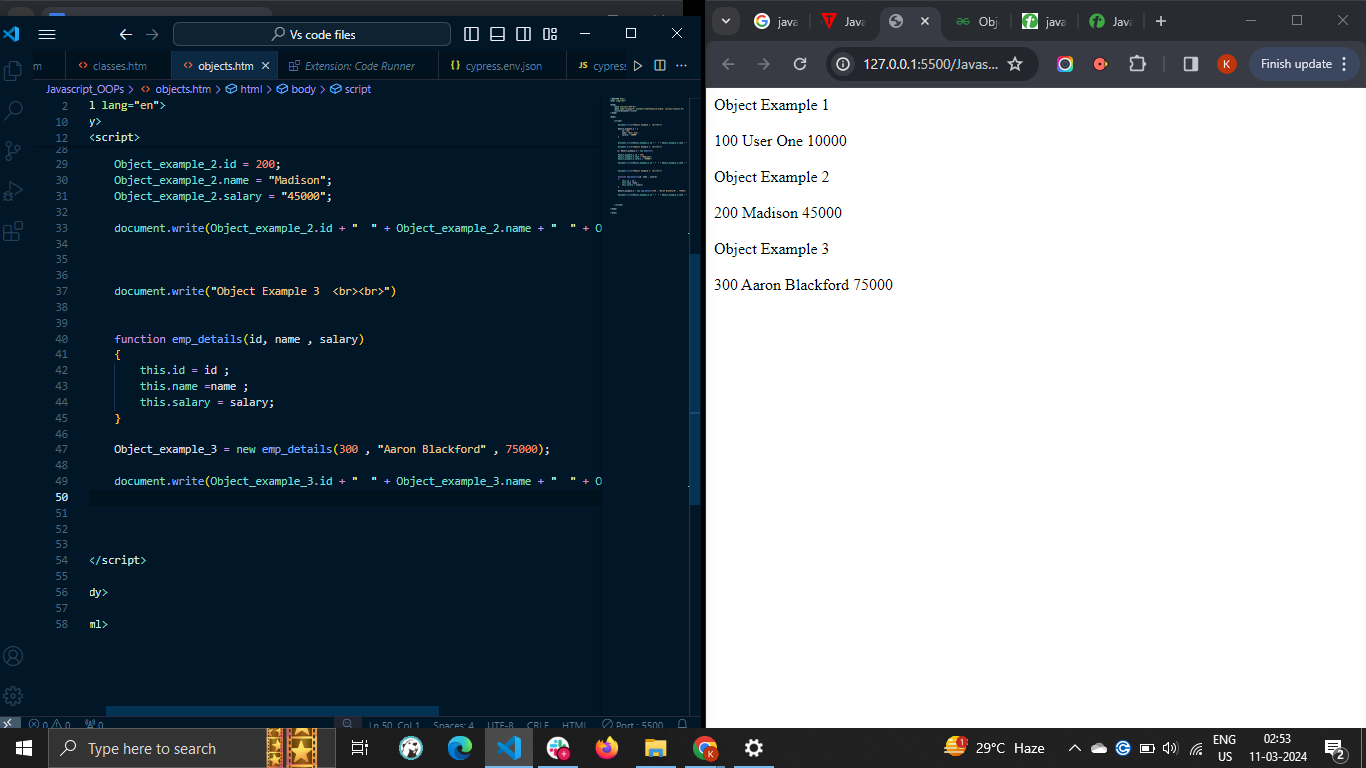
1. **var objectname=new Object();**

**Here, new keyword is used to create object.**

****

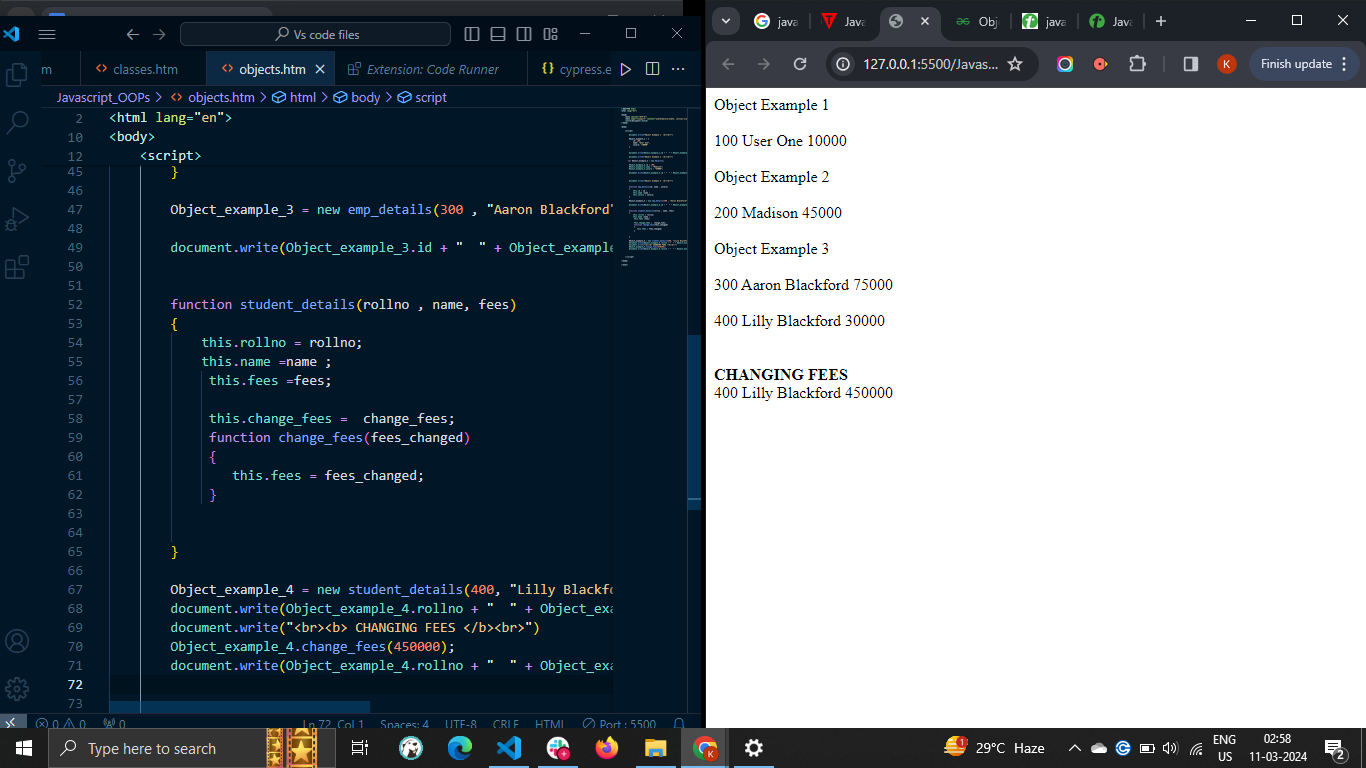
## 3) By using an Object constructor

* 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.



## Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.



## Accessing Object Properties

There are 3 ways to access object properties in JavaScript.

* Using the dot notation
* Using the square bracket notation
* Using the expression

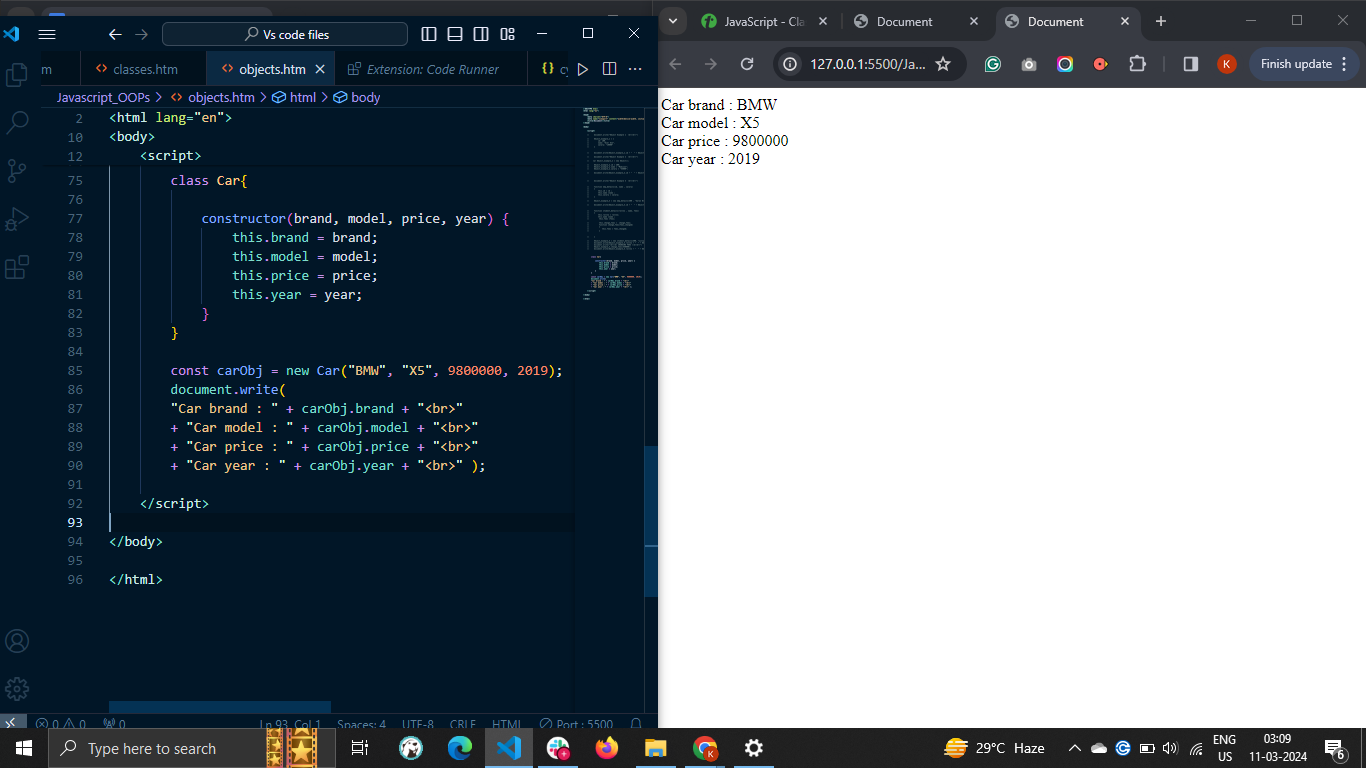
### The Dot Notation

You can access the object property using the dot notation/ syntax.

In the above syntax, 'obj' is an object, and 'prop' is its property whose value you need to access.

#### Example

The 'fruit' object in the example below contains the name and price property. We access the object properties using the dot notation, and you can see property values in the output.

****

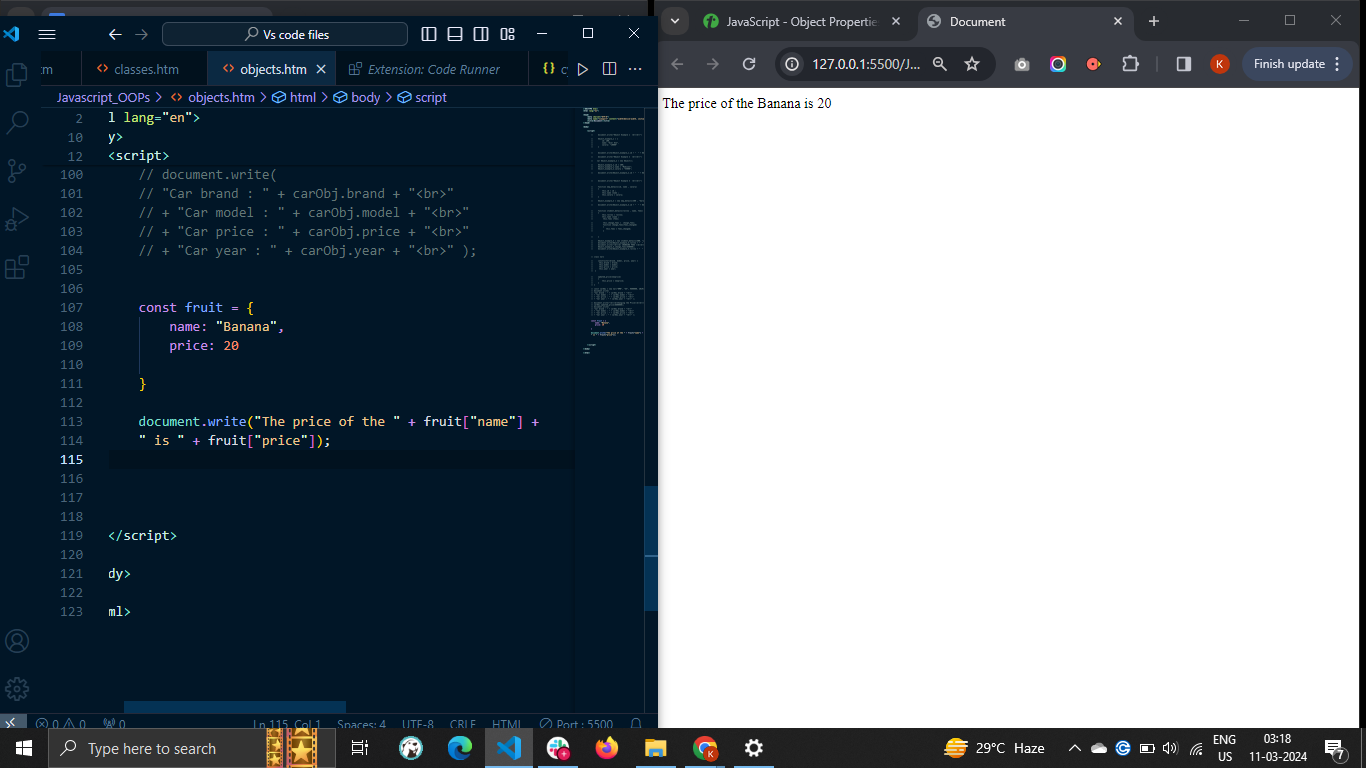
### The Square Bracket Notation

You can use the square bracket pair containing the property as a string followed by the object name to access a particular property.

****

In the above syntax, we access the 'prop' property from the object.

**NOTE :** You can't access the property using the dot notation when you use invalid identifiers as an object key. So, you need to use the square bracket notation. The identifier is invalid if it starts from a number, contains a space, or a hyphen.

****

### Using the expression inside the bracket

Sometimes, you require to access the object properties dynamically using the variable or expression. So, you can write the expression inside the square bracket notation. The expression can be a variable, a mathematical expression, etc.

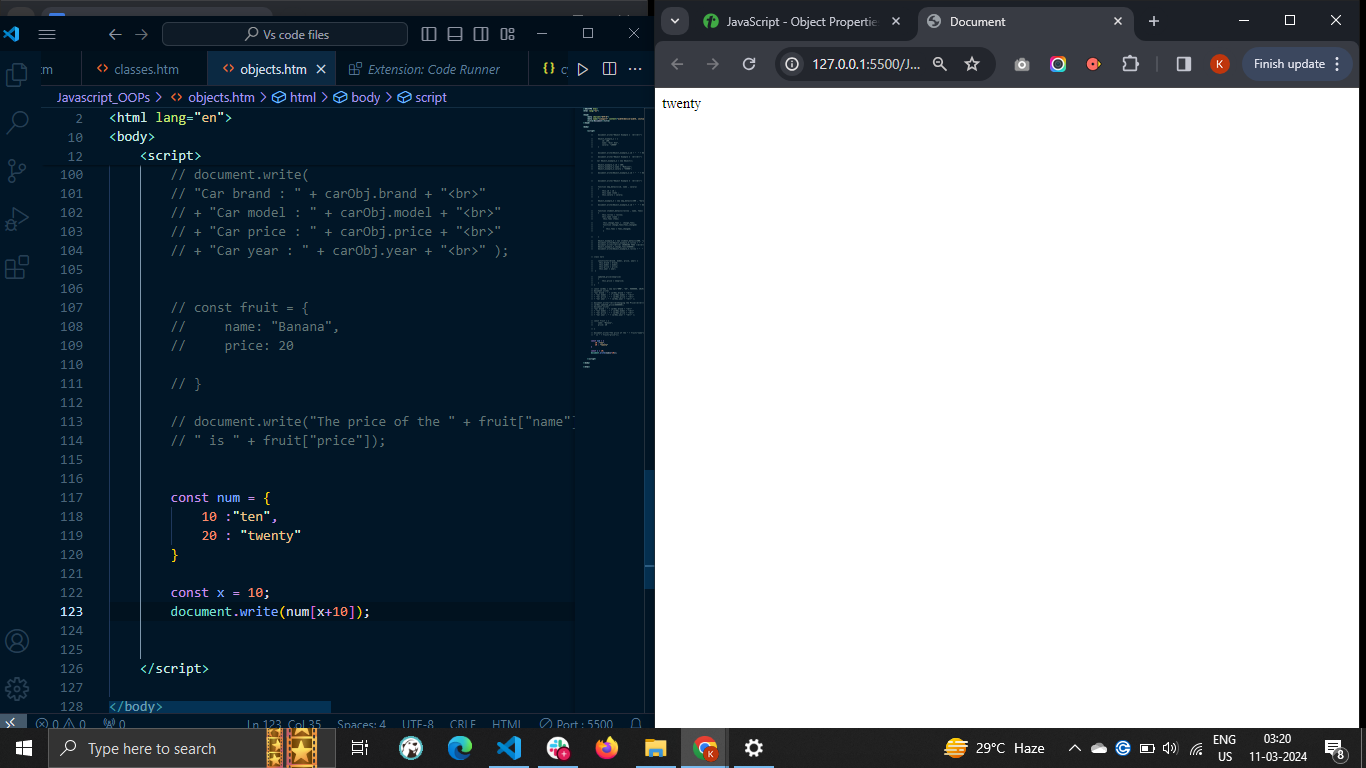
****

The above syntax evaluates the expression first and accesses the property same as a resultant value from the object. You don't need to write the expression in quotes.

#### Example

In the example below, the num object contains the number as a key in the string format and its word representation as a value.

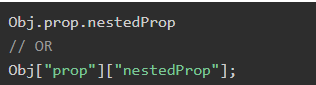
We use the variable x to access the property value from the object. Also, we used the "x + 10" mathematical expression to access the object property dynamically.

****

## Accessing the Nested Object Properties

Accessing the nested object properties is very similar to accessing the object properties. You can either use the dot or square bracket notation.

### Syntax

****

In the above syntax, the prop is a property of the obj object, and nestedProp is a property of the 'prop' object.

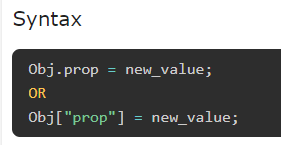
### Example

In the below code, the 'cars' object contains the nested objects named OD and BMW. We access the nested object properties using the dot and square bracket notation**.**

****

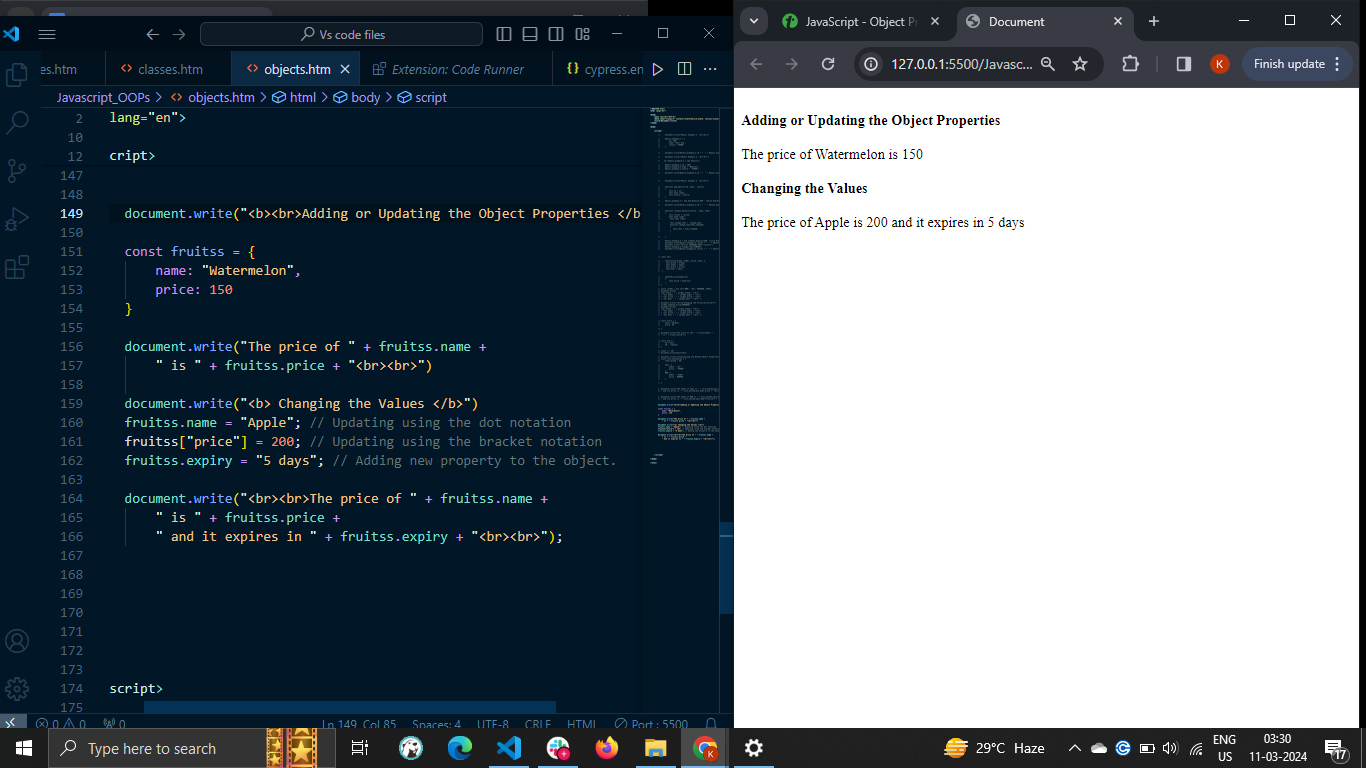
## Adding or Updating the Object Properties

You can update or add new properties to the object using the dot or square bracket notation. You can access the object property and assign a new value to it. If the property already exists, it updates the property value. Otherwise, it adds the property to the object.



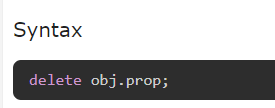
### Example

In the example below, we update the name and price property of the fruit object. Also, we add the expiry property to the fruit object.



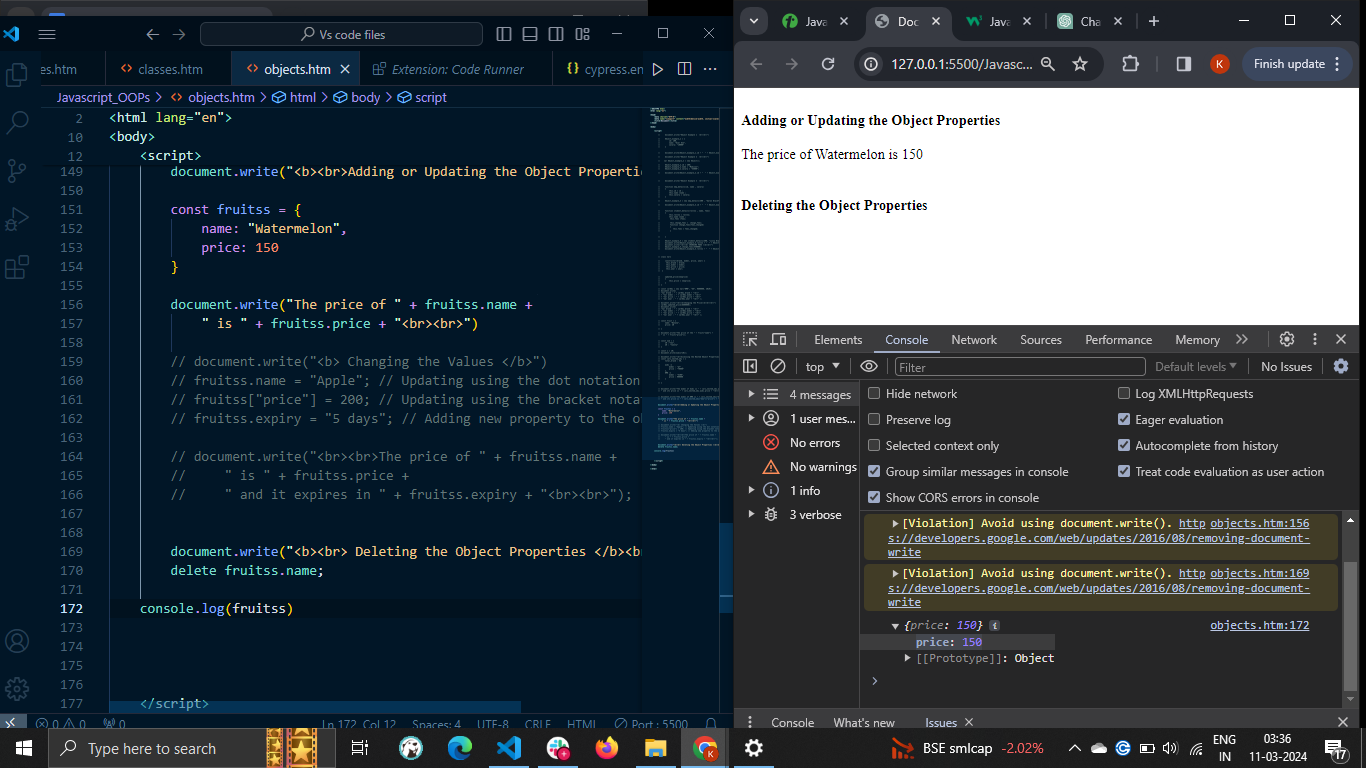
## Deleting the Object Properties

You can use the 'delete' operator to delete the specific object properties.



Example

In the example below, we delete the name property from the fruit object using the delete operator. The output shows that the fruit object contains only the price property after deleting the name property.

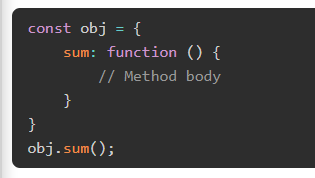


# **JavaScript - Object Methods**

* JavaScript object methods are object properties that contains function definitions. An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function; in that case the property is known as a method.
* You can either directly add a method to the object or add it as a property value. The method can also take the parameters and return the value. Object methods are a powerful way to add functionality to objects. They allow you to encapsulate code and make it reusable.

### Syntax

Follow the syntax below to add a method to the object.



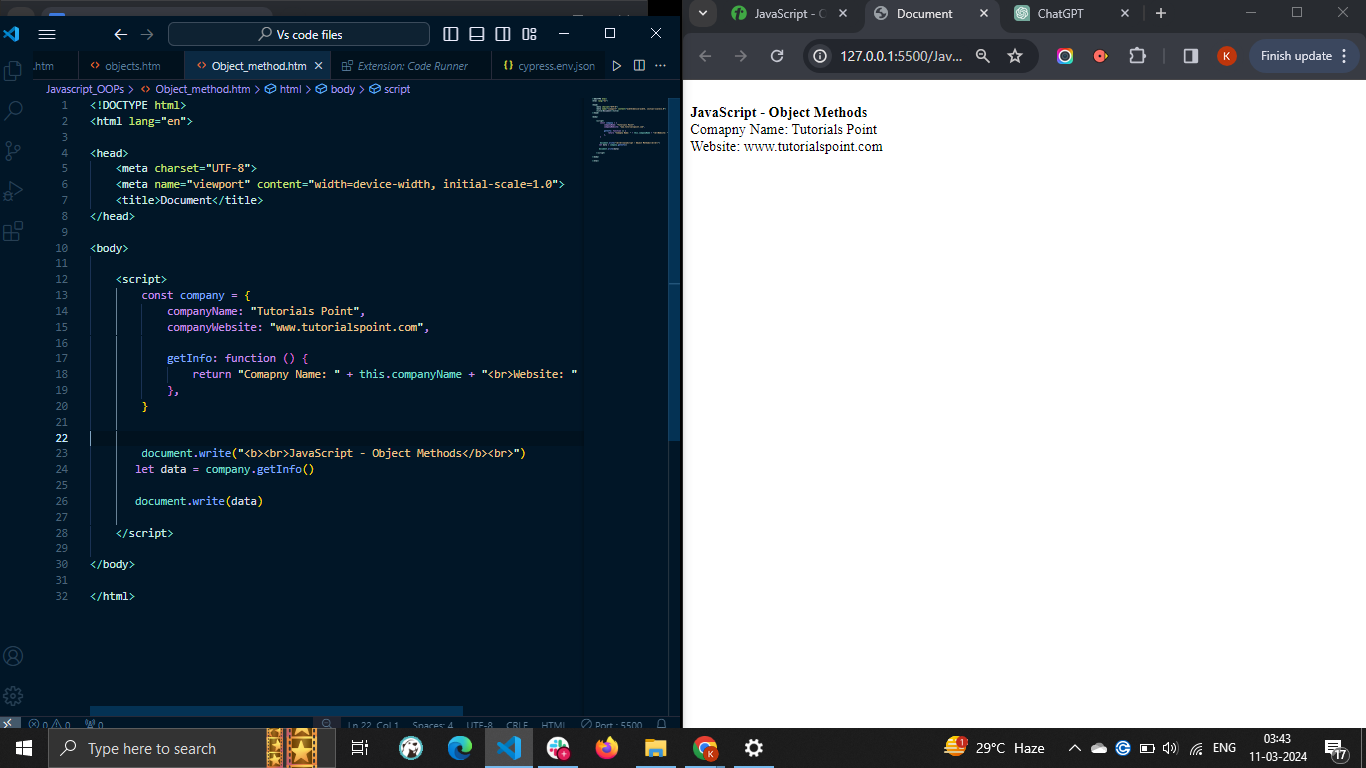
In the above syntax, 'sum' is a method defined inside the 'obj' object. You can access the method as you access the object property and add the pair of parenthesis to invoke the method.

### Example

We added the getInfo() method in the 'company' object in the example below. The getInfo() method returns the string containing the object properties.

Here, we used the 'this' keyword to access the object properties inside the object. The 'this' keyword represents the object itself.

After that, we used the object as a reference to invoke the method.

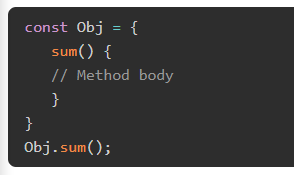


## Object Method Shorthand

The ES6 provides the shortest way to define a method into the object.

### Syntax

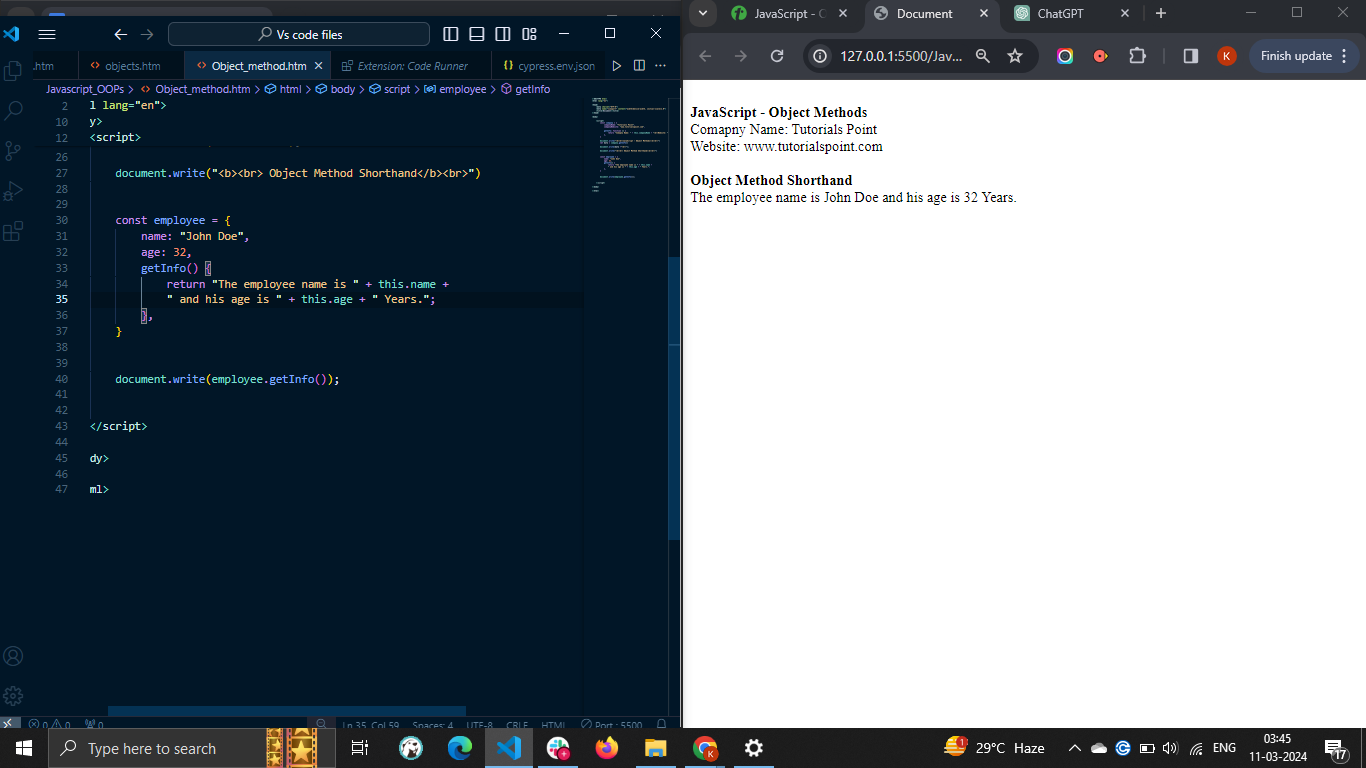
Follow the syntax below to add a method to the object.



Like the previous one, you can access and invoke the method in the above syntax.

### Example

In the example below, we defined the getInfo() method as the previous example.



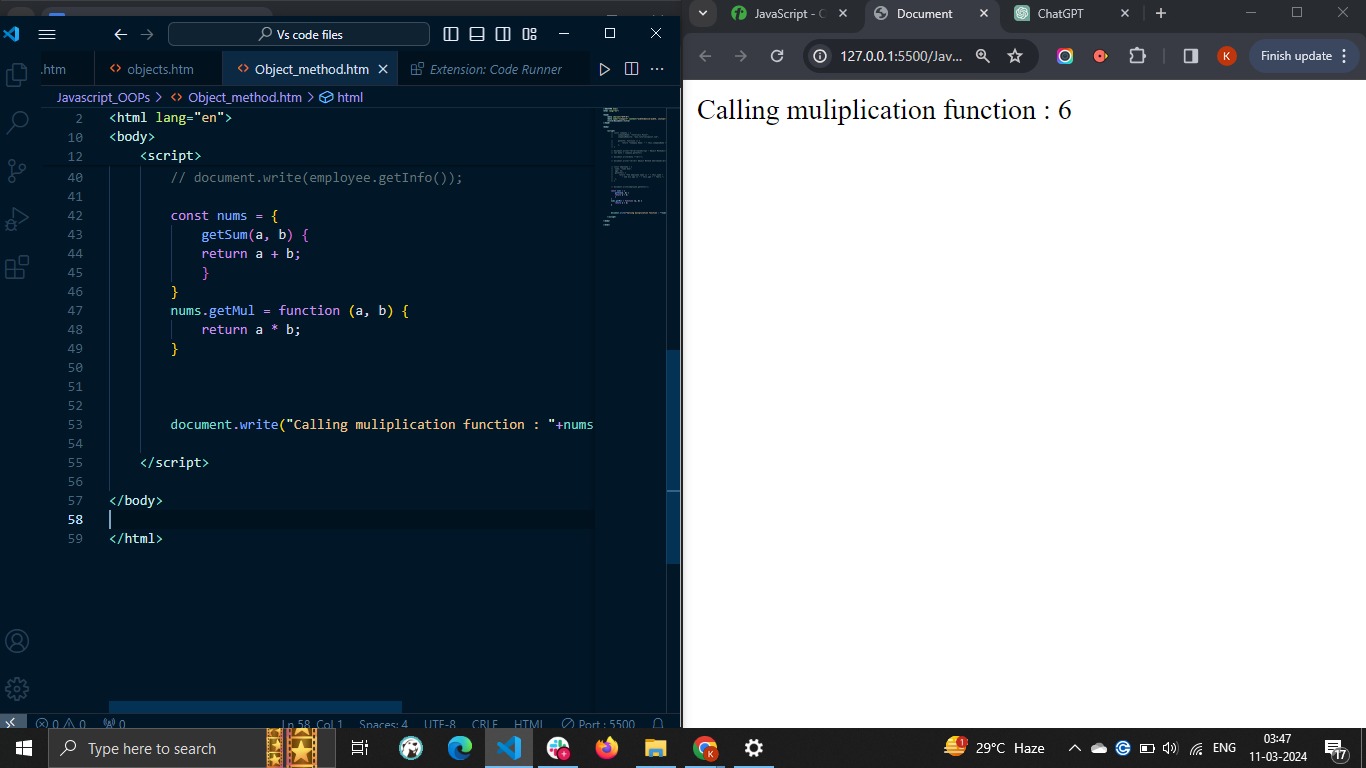
## Updating or Adding a Method to the Object

In JavaScript, updating or adding a new method to the object is same as updating or adding new proeprties to the object. You can either use the dot or square bracket notation to update or add method to the object.

### Example

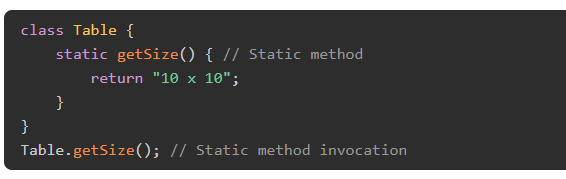
The example below defines the getSum() method inside the 'nums' object.

After that, we add the getMul() method inside the nums object. We invoke the getMul() method by passing two arguments to get the multiplication of them.



# **JavaScript - Static Methods**

* A static method in JavaScript is defined using the static keyword followed by the method name. You can execute the static method by taking the class name as a reference rather than an instance of the class.
* The main benefit of the static method is that it can be used to create a utility function that doesn't require the instance of the class for the execution. For example, a Math object contains various static methods, which we can invoke through Math class directly.
* Also, you can use static methods to add all related methods under a single namespace. Furthermore, static methods give better performance than the regular class methods due to memory optimization.

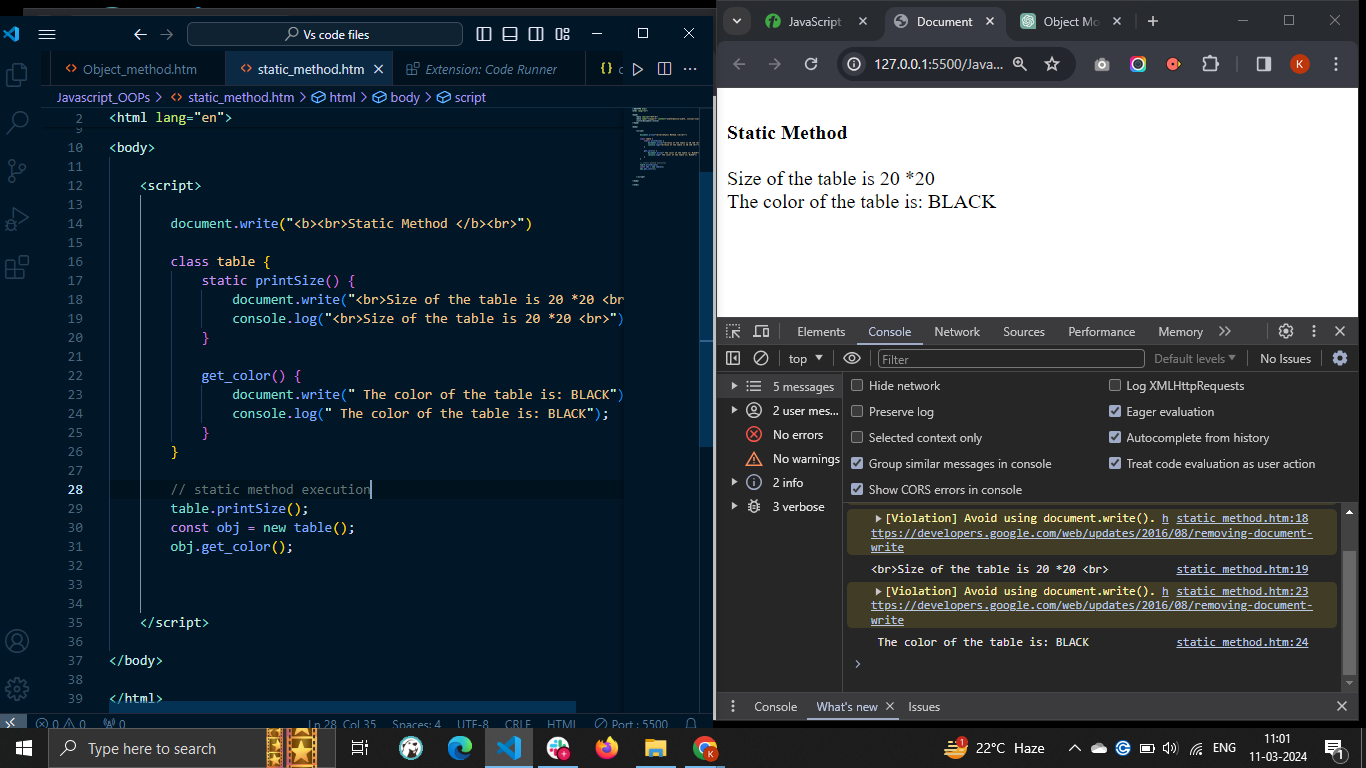


In the above syntax, getSize() is a static method. We used the class name to execute the getSize() method.

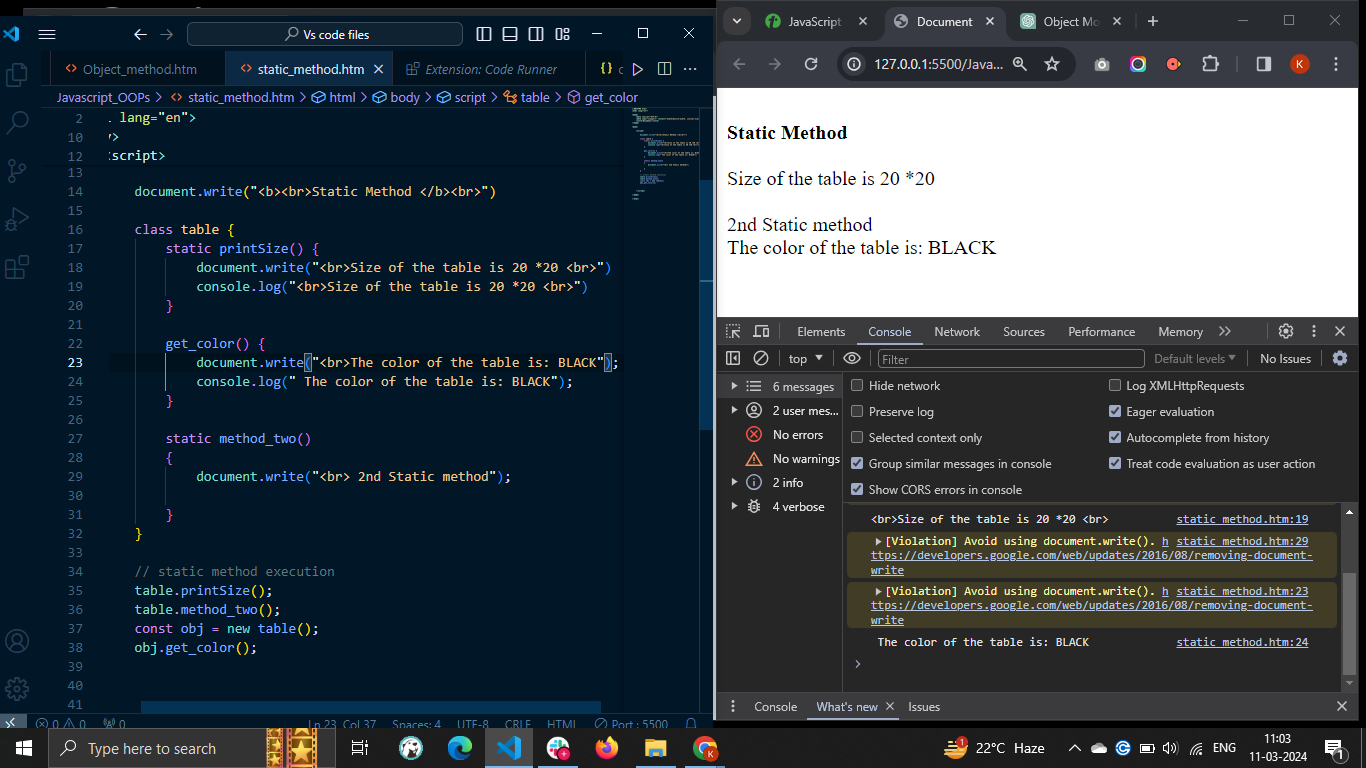
### Example: Static Method

In the example below, printSize() is a static method, and getSize() is a regular method in the table class. You can see that printSize() method is invoked using the table class name, and getSize() method is executed using the class instance.

So, the class can contain static and non-static methods.

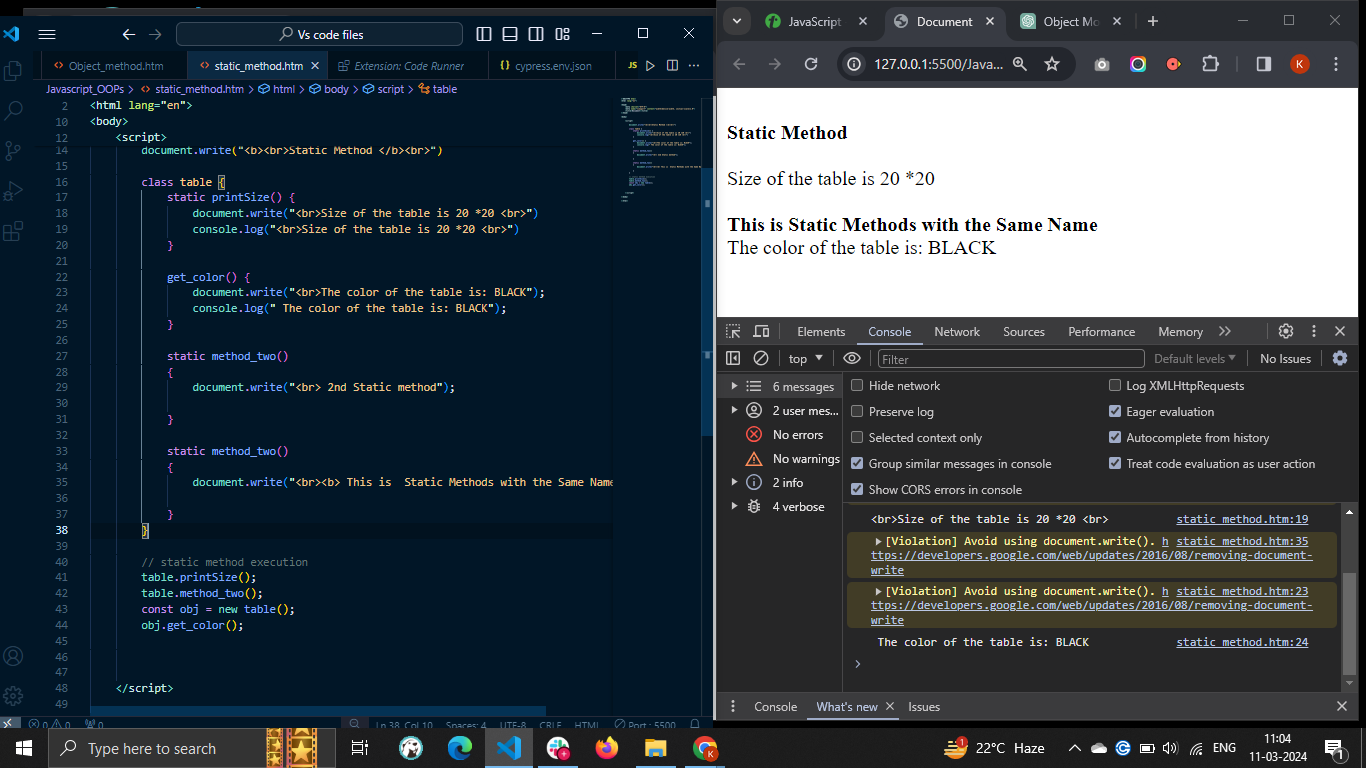


The single class can also contain multiple static methods.



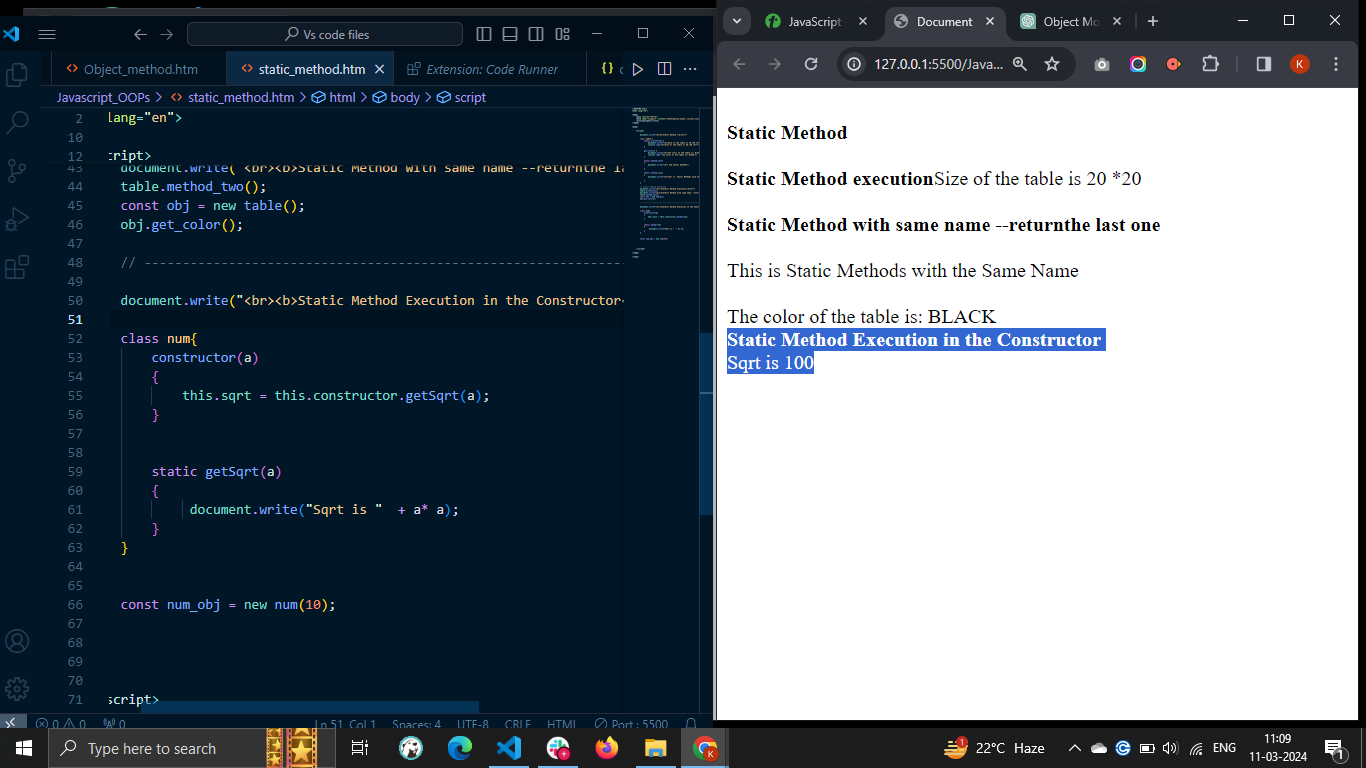
A single class can contain multiple static methods with the same name. When you execute the static method with the same name, it executes the last method.

### Example: Static Methods with the Same Name



You can also execute the static method of the class in the constructor. You can use this keyword followed by the constructor keyword followed by the static method name to execute the static method in the constructor.

### Static Method Execution in the Constructor

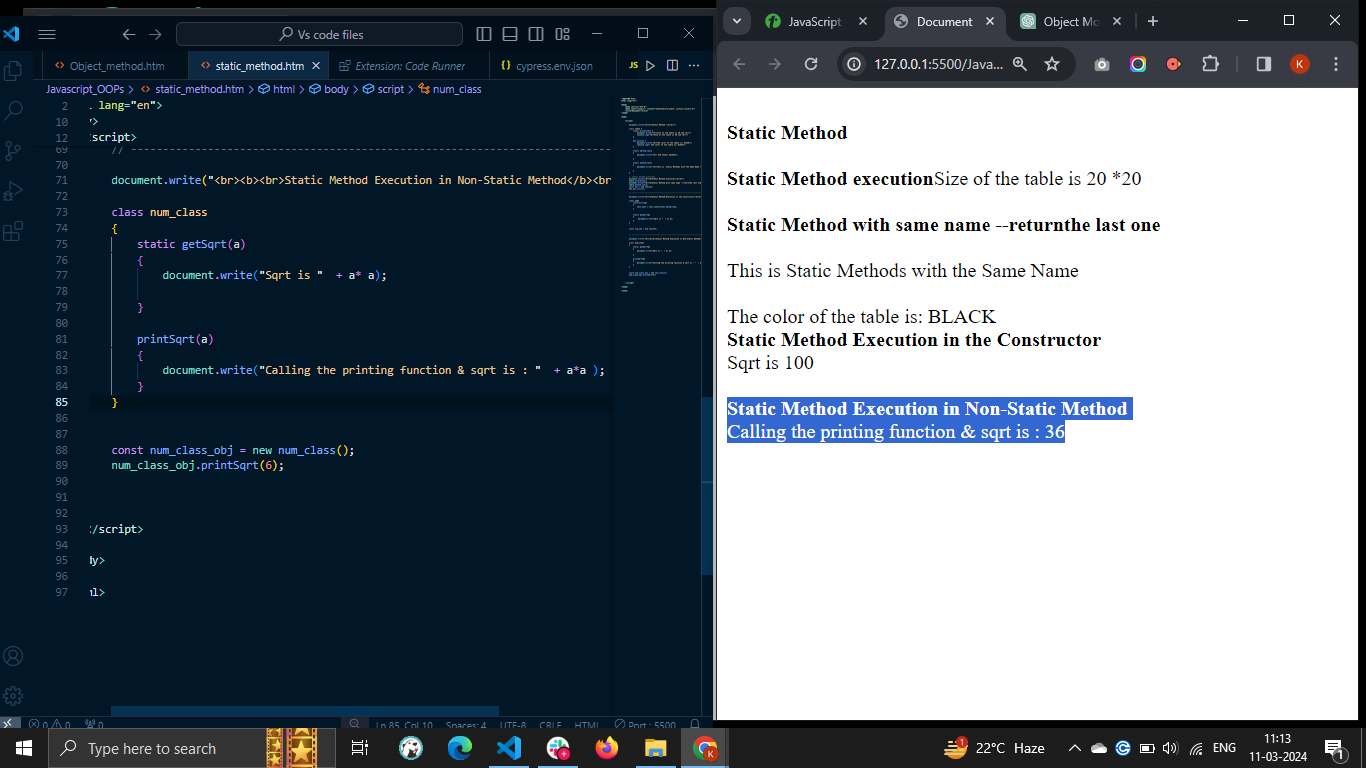


You can also execute the static method in the non-static method. You need to use the class name as a reference to execute the static method in the non-static method.

### Static Method Execution in Non-Static Method

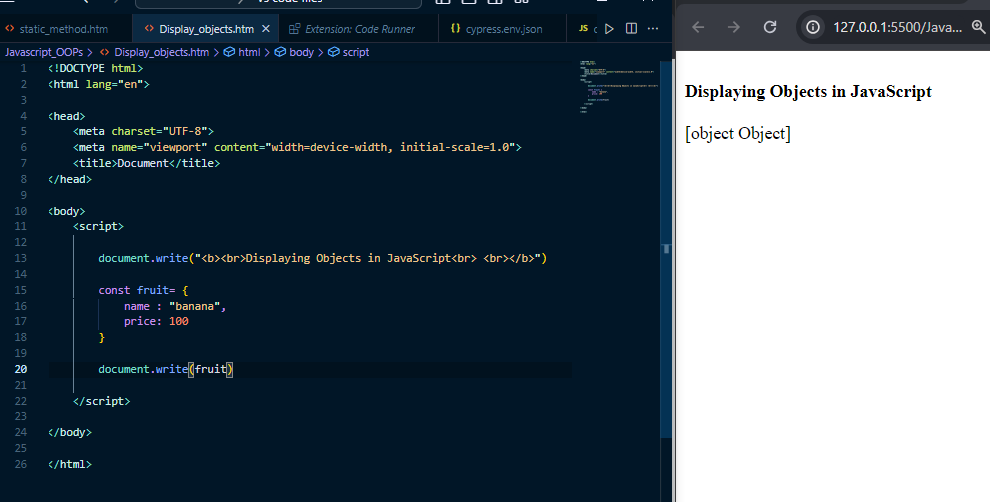
In the example below, getSqrt() is a static method, and printSqrt() is a regular class method. In the printSqrt() method, we execute the getSqrt() method.

We used the instance of the Num class to execute the printSqrt() method.



# **JavaScript - Display Objects**

* There are different ways to display objects in JavaScript. Using the console.log() method, we can display the object in the web console. Sometimes developers require to display the object properties and their value in the HTML or for debugging the code.
* For displaying an object, we can access the different properties and display them. We can also convert the object to a JSON string and display it as a string.
* When you print the object like other variables in the output, it prints the '[object Object]'



To overcome the above problem, you need to use specific approaches to display the object.

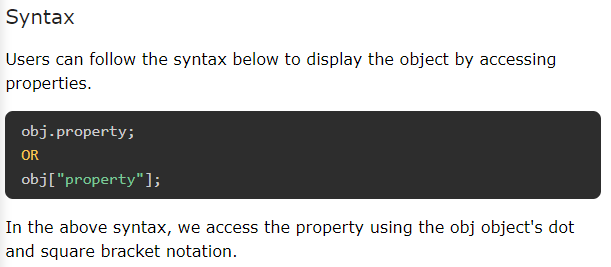
Some approaches to display the JavaScript objects are as follows −

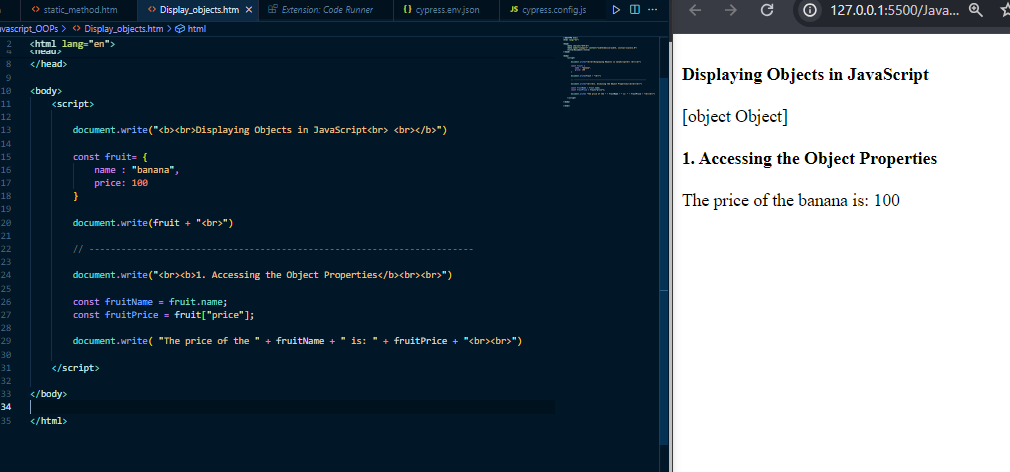
* Accessing the object properties
* Using the JSON.stringify() method
* Using the Object.entries() method
* Using the for...in loop

## 1. Accessing the Object Properties

You can use the **dot notation** or **square bracket** notation to display the property values.

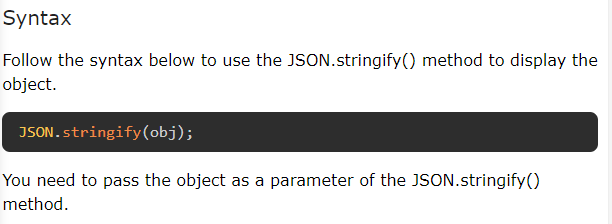
This way, you may get all property values and display them in the output.

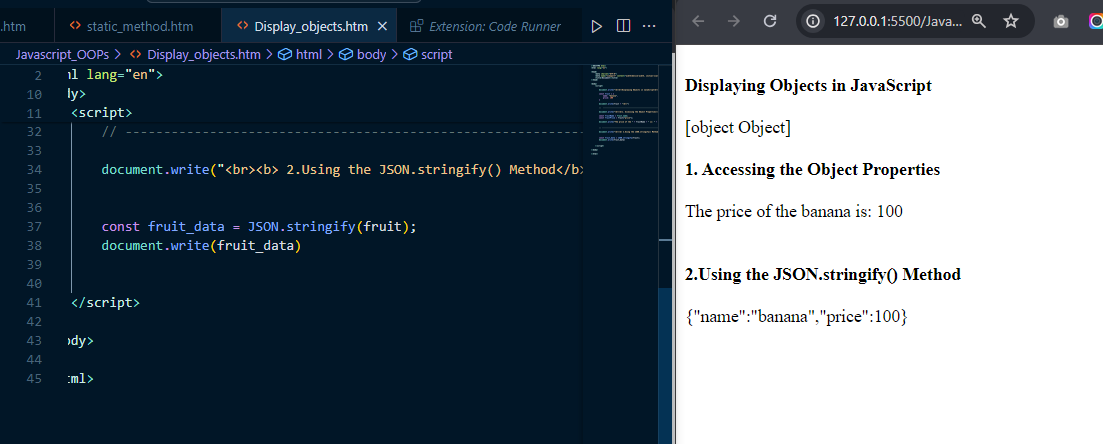




2. Using the JSON.stringify() Method

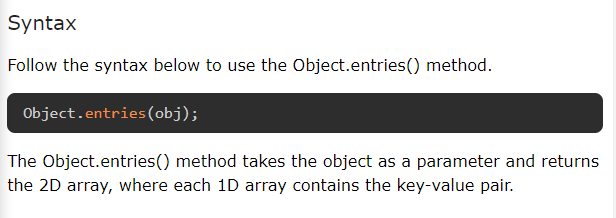
When object contains the dynamic properties or you don't know object properties, you can't print properties and values using the first approach. So, you need to use the JSON.stringify() method. It converts the object into a string.

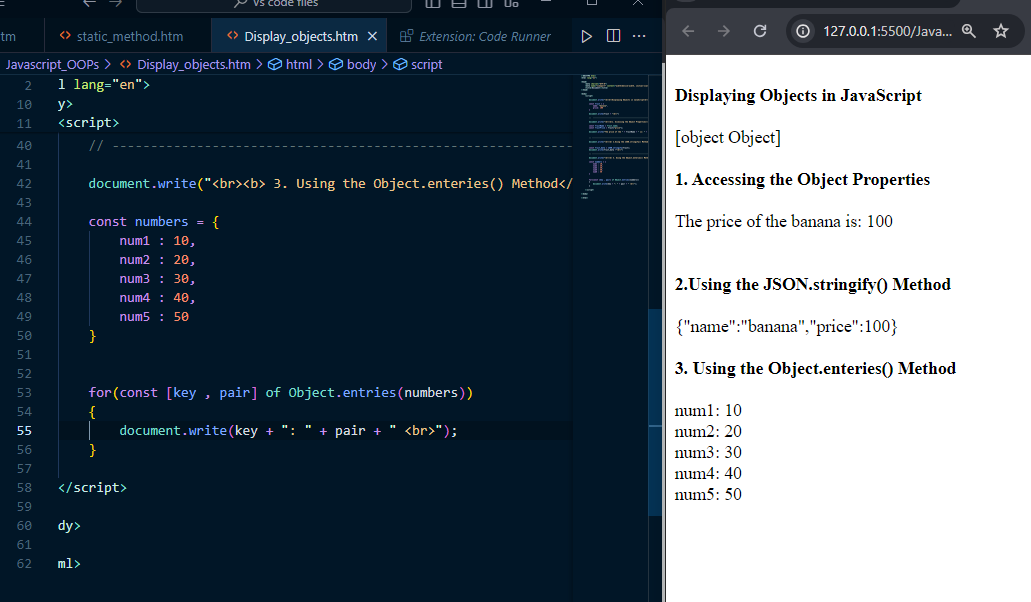




## 3. Using the Object.entries() Method

The Object.entries() is a static method of the Object class, allowing you to extract the properties and values in the 2D array. After that, you can use the loop to traverse the array and display each property and value pair individually.



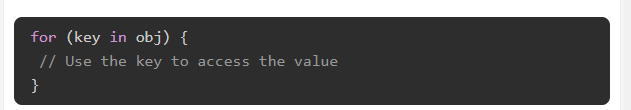


## 4. Using the for...in Loop

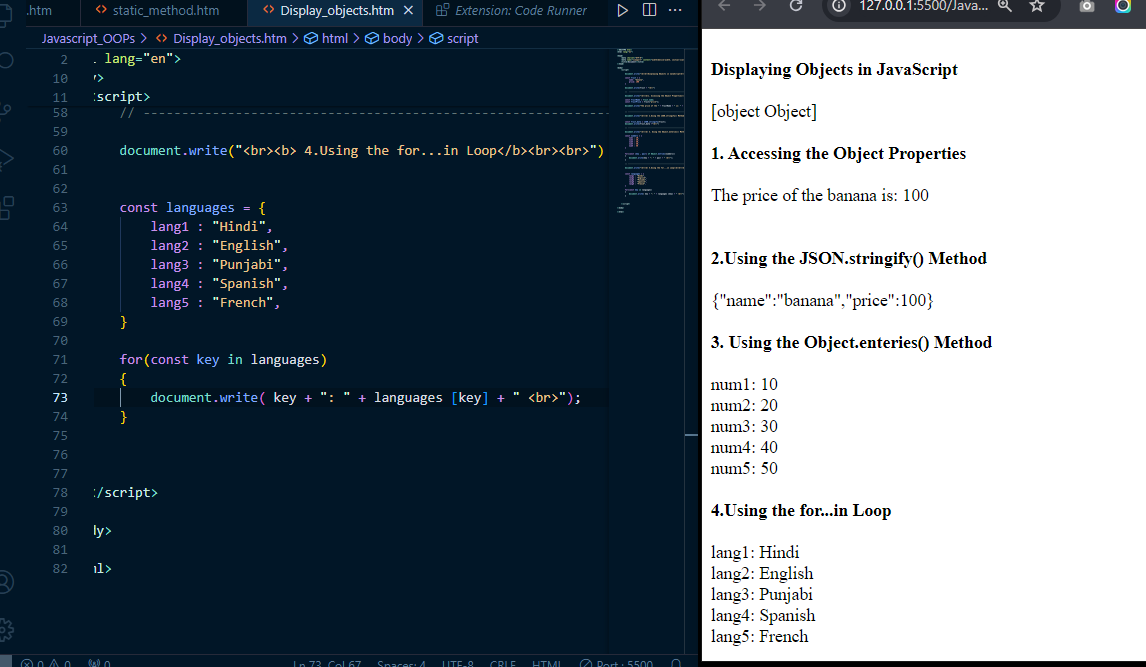
The for...in loop is used to traverse the iterable, and the object is one of them.

### Syntax

Users can follow the syntax below to use the for...in loop to traverse the object and display it in the output.



In the above syntax, obj is an object to display. In the loop body, you can access the value related to the key and print the key-value pair.



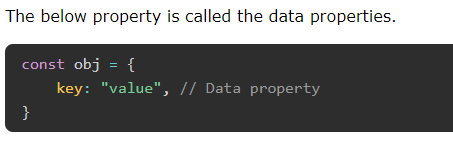
**NOTE : The best way to display the object is using the JSON.stringify() method. It converts the object into a flat string. Other approaches can't be used to display the nested objects, but JSON.stringify() method can be used.**

# **JavaScript - Object Accessors**

The **object accessor** properties in JavaScript are methods that get or set the value of an object. They are defined using the get and set keywords. Accessor properties are a powerful way to control how your objects are accessed and modified.

The JavaScript object can have two kinds of properties.

* Data properties
* Accessor properties



## Object Accessor Properties

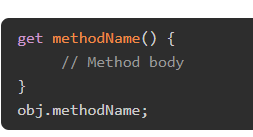
In JavaScript, you can use the getters to get the object properties and setters to set the object properties.

There are two keywords to define accessor properties.

* get − The get keyword is used to define a method to get the object property value.
* set − The set keyword is used to define a method to update the object property value.

## JavaScript Getters

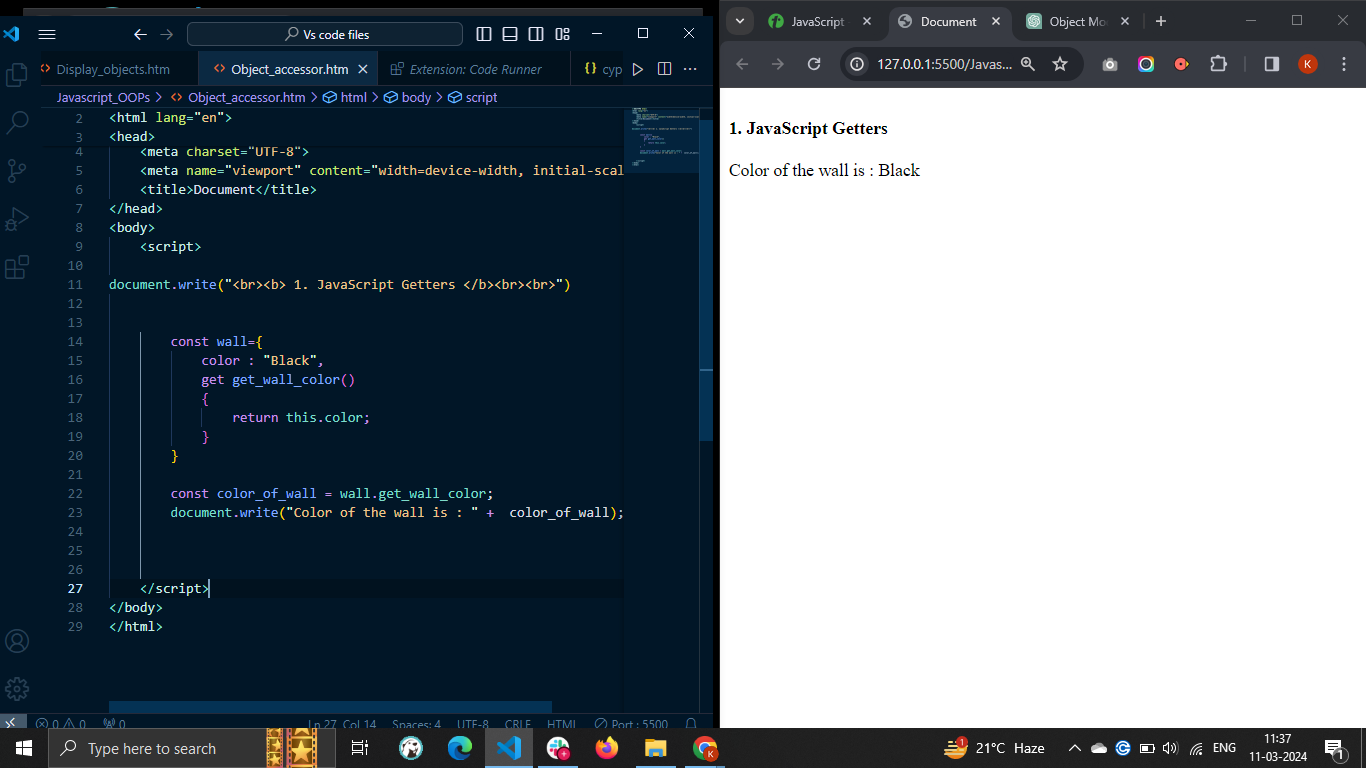
The getters are used to access the object properties. To define a method as getter, we use get keyword followed by method name. Follow the syntax below to define the getter.



In the above syntax, we have defined the getters using the 'get' keyword followed by the method name.

You can use the method name as an object property to get its returned value.

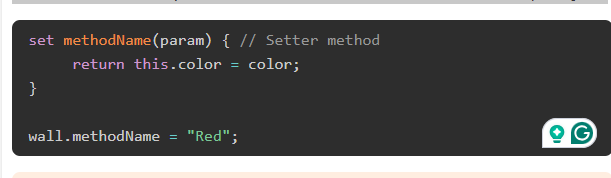
*You don't need to write the pair of parenthesis followed by the method name to execute the getters. You can access it like the object property.*



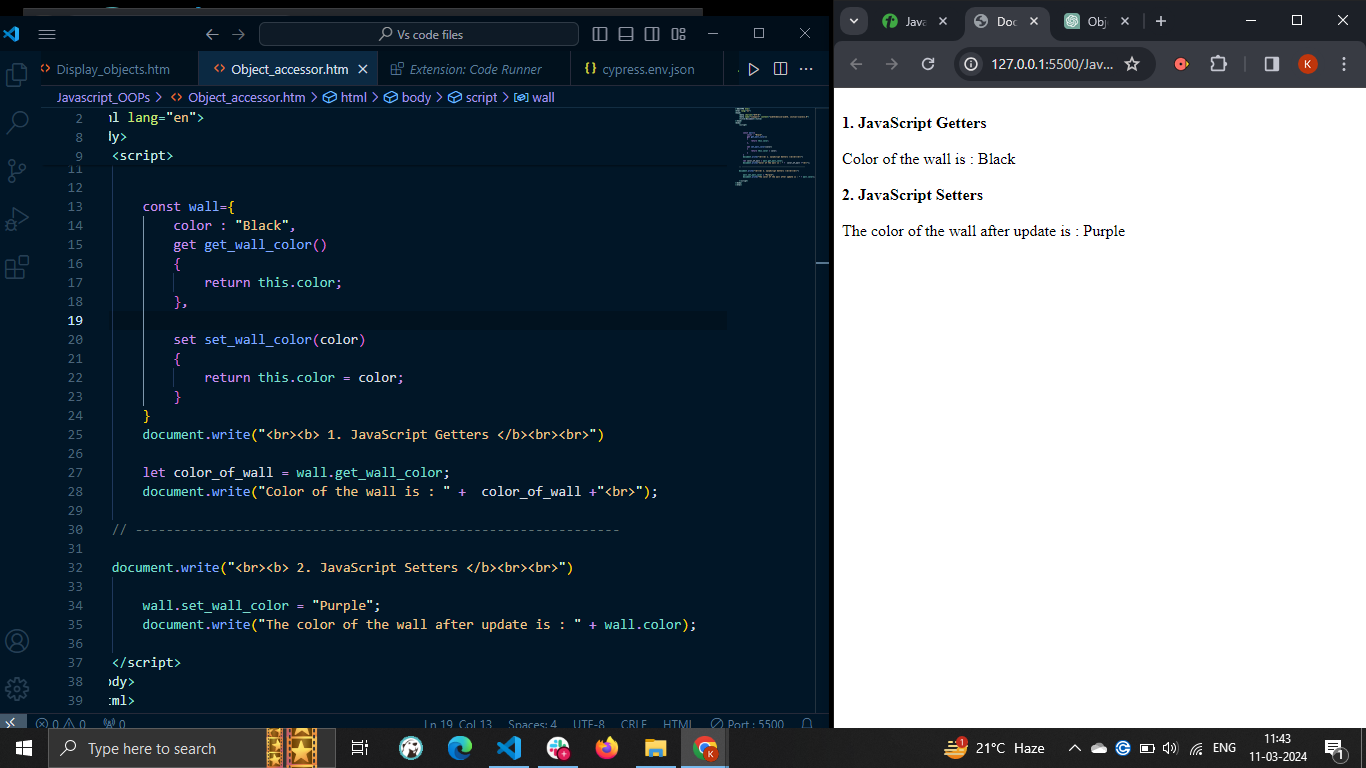
## JavaScript Setters

The **setters** are used to update the JavaScript object properties. To define a method as setter, we use **set** keyword followed by method name You can follow the syntax below to define setters in the JavaScript object.

Syntax :



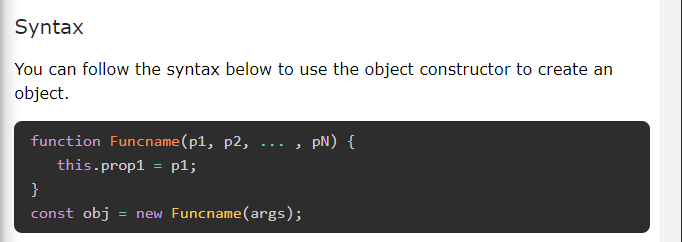
* In the above syntax, the 'set' keyword is used to define the setter method.
* The method\_name can be any valid identifier.
* The setter method always takes a single parameter. If you don't pass a parameter or multiple parameters, it will give you an error.
* You can assign value to the setter method as you assign value to the property.



# 

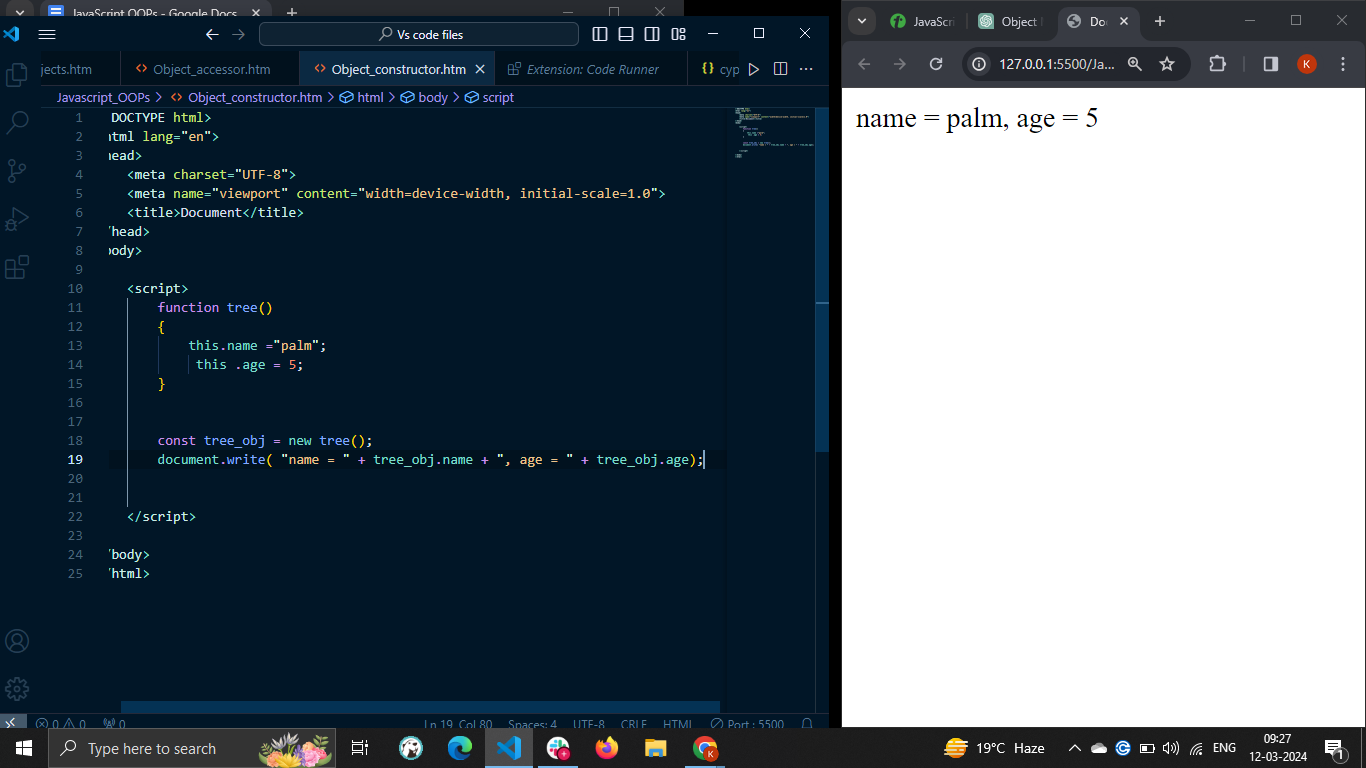
# **JavaScript - Object Constructors**

* An object constructor in JavaScript is a function that creates an instance of a class, which is typically called an object. A constructor is called when you declare an object using the new keyword. The purpose of a constructor is to create an object and set values if there are any object properties present.
* There are two ways to create a template for the object in JavaScript - using a class and using an object constructor.
* Whenever you need to create multiple objects with the same syntax, you require a template for the object. For example, you are managing the car inventory. So, it is not a good idea to create a new object every time using the object literal. In such cases, you need to use the object constructors.
* The main benefit of the object constructors is that you can reuse the code.



* In the above syntax, Funcname() is a constructor function, and you can replace any valid identifier with the Funcname.
* A p1, p2, …, and pN are parameters you can use inside the function body. You need to pass arguments to the constructor while creating the object.
* The 'this' keyword represents the function context you are using. Here, the 'this' keyword refers to the current instance of the object.
* To create an object, you can use the function constructor with a 'new' keyword.

### **Creating an object using a constructor function**



### **Constructor function with parameters**

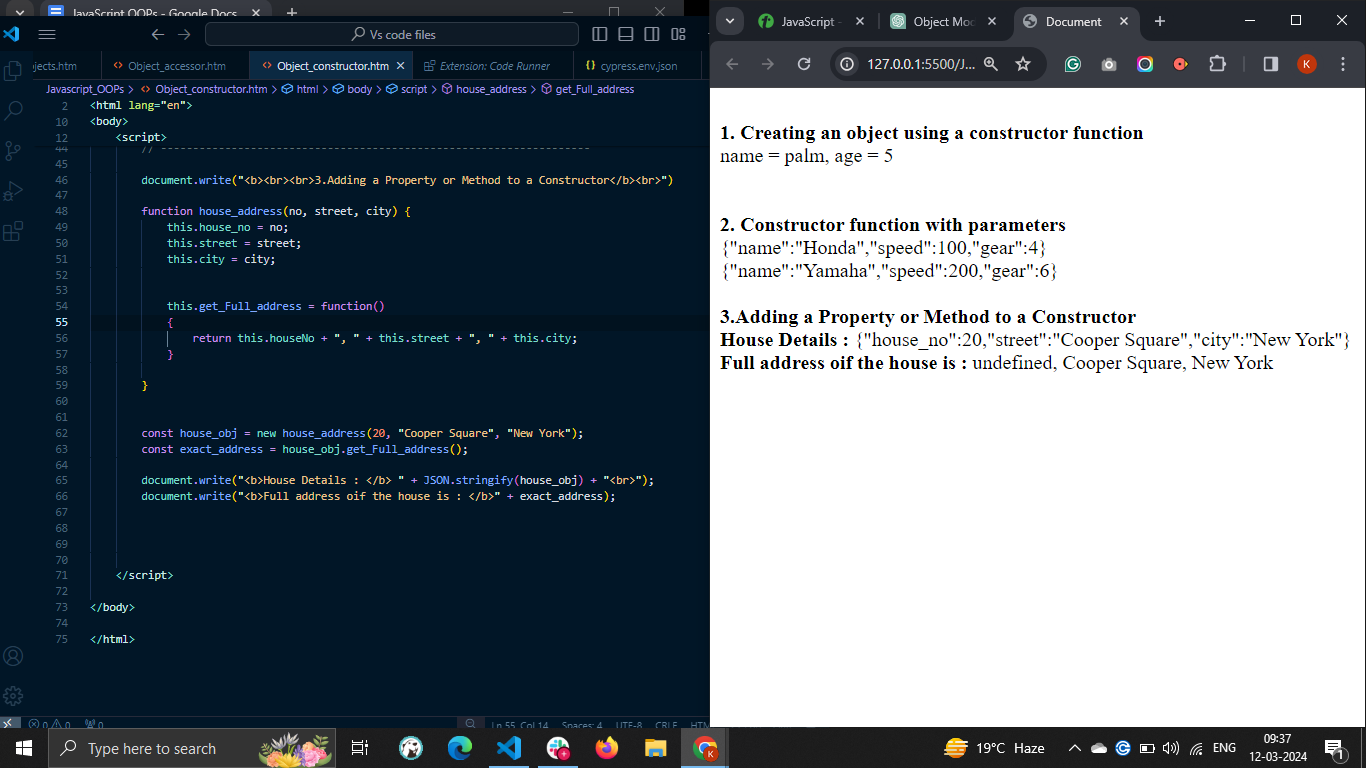


In this way, you can use the object constructor to reuse the object syntax code and create multiple objects of the same type.

## Adding a Property or Method to a Constructor

You learned to add a property of method using the dot or square bracket notation into the object in the Objects chapter. But what if you want to add a property or method to the object constructor?

The object constructor doesn't allow you to add the property or method after defining it. So, you always need to add the required properties and methods while defining it. Let's understand it via the example below.



If you add the method or property as shown below. It will be added to the particular object but not to the constructor function.

Obj.prop = 20;

Other objects created using the object constructor don't contain the prop property as it is added to the obj object only.

## JavaScript Object Prototype

In JavaScript, each object contains the prototype property by default, and the object constructor is also one kind of object. So, you can add properties or methods to the object prototype.

### Syntax

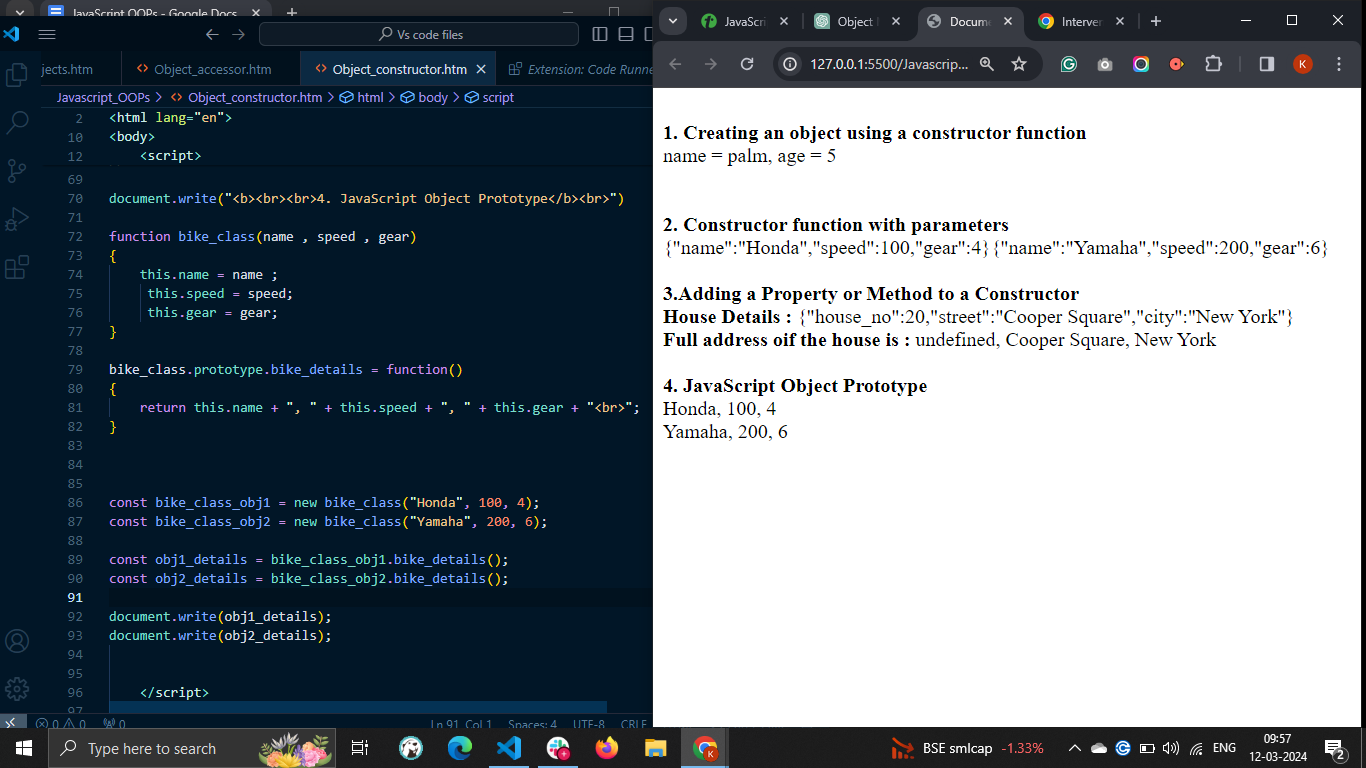
You can follow the syntax below to add properties or methods to the object constructor prototype.

obj.prototype.name = value;

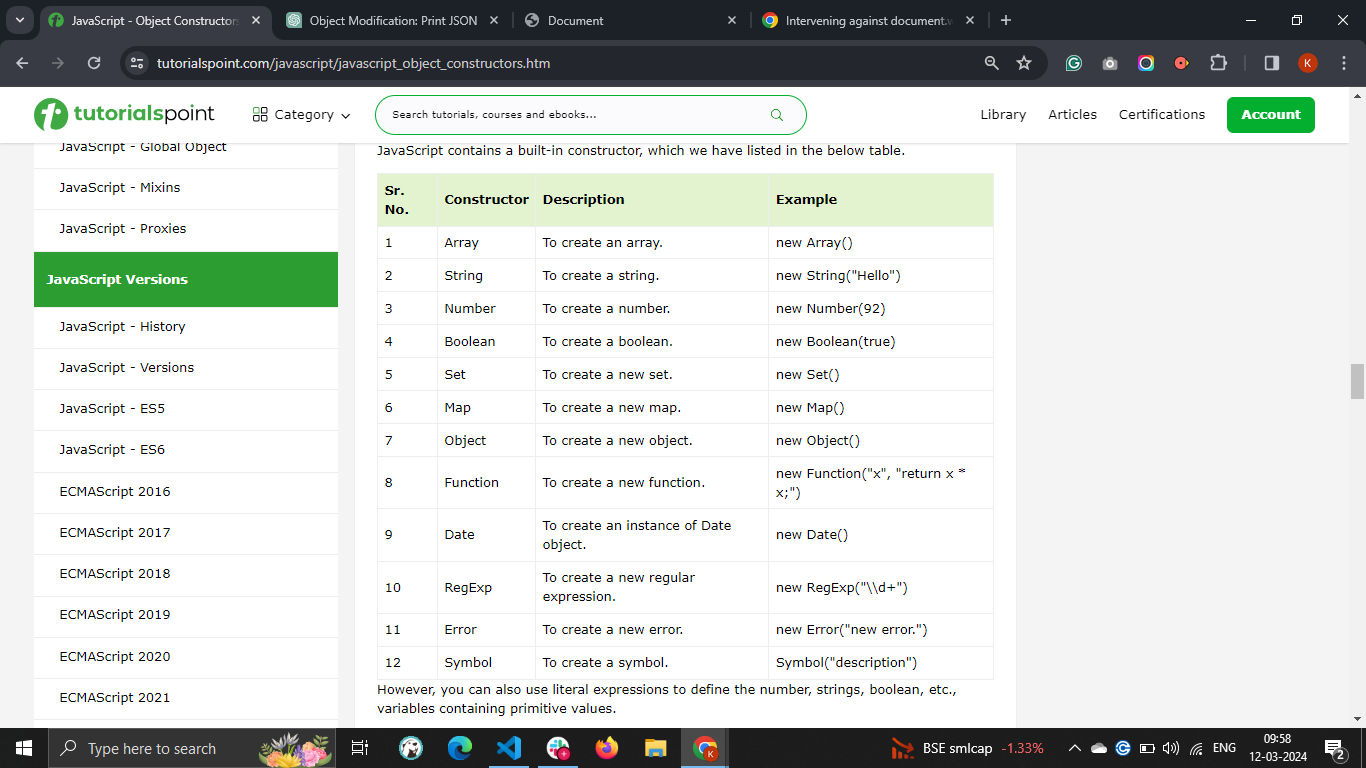
In the above syntax, 'obj' is an object constructor in which you need to add a property or method.

The 'name' is a property or method name.

For the property, you can replace a 'value' with an actual value; for the method, you can replace a 'value' with a function expression.



## Built-in Object Constructors in JavaScript



# **JavaScript - Native Prototypes**

## Native Prototypes

The **native prototypes** in JavaScript are property of Object.prototype object. The prototypes are the mechanism by which the objects inherit features from one another.

In JavaScript, each object contains the prototype property. The prototype of each object contains the methods and properties related to the object. So, it is also called the native prototype.

However, you can update or add new methods and properties to the native prototype object, but you can't delete any already existing.

The JavaScript object and object constructor are the main prerequisites to understand JavaScript native prototypes.

### Syntax

You can follow the syntax below to access the native prototype of the object.

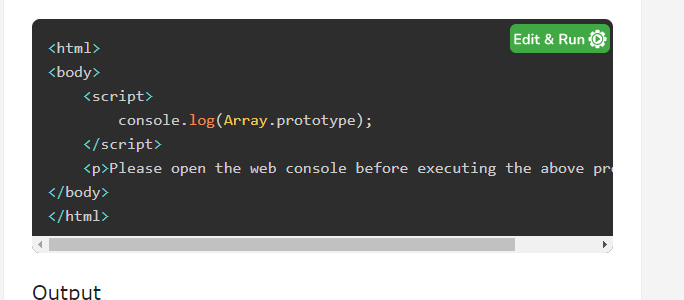
Object.prototype;

In the above syntax, the object can be any JavaScript object.

### Example: Accessing the array's prototype

Whenever you execute the below code in the browser, it will print the prototype of the array in the browser console. In the same way, you can check the prototype of the other objects.

In the console, you can see that the prototype object contains the methods which you can use with array methods.





## Updating the Native Prototype

You can update the existing method or property of the native prototype object or add a new method or property to the native prototype object.

### Syntax

You can follow the syntax below to update or add new properties or methods to the prototype object.

objName.prototype.name = value

In the above syntax, objName is an object whose prototype you need to update.

The 'name' is a method or property name. You can assign new values or function expressions to the 'name’.

### Example: Updating the toLowerCase() method of the string object's prototype

The prototype of the String object contains the toLowerCase() method. Here, we update the toLowerCase() method.

The updated toLowerCase() method returns the string in the uppercase. In the output, you can observe the string, which is in uppercase.

In this way, you can update the functionality of the built-in methods of the object.

<html>

<body>

<p id = "output">After updating the string.toLowerCase() method: </p>

<script>

String.prototype.toLowerCase = function () {

return this.toUpperCase();

}

let str = "Hello World";

document.getElementById("output").innerHTML += str.toLowerCase();

</script>

</body>

</html>

#### Output

After updating the string.toLowerCase() method: HELLO WORLD

*You shouldn't update the methods and properties of the native prototype object. However, you can add new as shown in the example below.*

### Example: Adding a new method to the prototype object

You can also add a new method to the Object prototype. Here, we added the firstCase() method in the object prototype.

The firstCase() method returns a string after converting the string's first character to the uppercase.

<html>

<body>

<p id = "output">After executing the string.firstCase() method: </p>

<script>

String.prototype.firstCase = function () {

// First character in uppercase. Other characters in lowercase.

return this.charAt(0).toUpperCase() + this.slice(1).toLowerCase();

}

let str = "hello world";

document.getElementById("output").innerHTML += str.firstCase();

</script>

</body>

</html>

#### Output

After executing the string.firstCase() method: Hello world

## Adding a method to the constructor function

Whenever you define an object using the constructor function, you can't add a method or property to the constructor function using its instance. So, you need to add the method to the constructor function prototype. So it can be accessible through all instances of the object.

### Example

In the example below, the Person() is a constructor function that initializes object properties.

After that, we added the display() method to the prototype of the person() function.

Next, we created two instances of the Person() function and used the display() method with them. So, methods and properties added in the prototype of the object constructor can be accessed through all instances of the constructor function.

<html>

<body>

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

<script>

const output = document.getElementById("demo");

function Person(id, name) {

this.id = id;

this.name = name;

}

Person.prototype.display = function () {

output.innerHTML += this.id + ", " + this.name + "<br>";

}

const p1 = new Person(1, "James");

const p2 = new Person(2, "Nayan");

p1.display();

p2.display();

</script>

</body>

</html>

#### Output

1, James

2, Nayan

*All instances of the object inherit the properties and methods from their parent's prototype.*

## JavaScript Prototype Chaining

Simply, you can say that the prototype stores the default values of the properties. The code overrides the prototype property value if the object constructor and its prototype contain the same properties.

### Example

In the below code, the Person() function contains the name property. We have added the name and age property in the function's prototype.

We have created the p1 object using the Person() constructor function. The value of the name property of the p1 object is 'Nayan' as the name already exists in the constructor. The value of the age property is 20, which is the same as the age property value in the prototype.

<html>

<body>

<p id = "output"> </p>

<script>

function Person(id, name) {

this.id = id;

this.name = name;

}

Person.prototype.name = "John";

Person.prototype.age = 20;

const p1 = new Person(1, "Adam");

document.getElementById("output").innerHTML =

"Id: " + p1.id + ", Name: " + p1.name + ", Age: " + p1.age;

</script>

</body>

</html>

#### Output

Id: 1, Name: Adam, Age: 20

# **JavaScript - Encapsulation**

## What is Encapsulation?

Encapsulation in JavaScript is a way to keep the related properties and methods under a single namespace by bundling them. It can be a function, a class or an object. In JavaScript, the encapsulation can be implemented using closures, classes and getters and setters.

Encapsulation is a fundamental concept in Object-oriented programming languages, along with inheritance and polymorphism. JavaScript is an object oriented programming language.

It is used to hide the data from the outside world and give access to required data only to improve the integrity and security of the data.

## What is the need for encapsulation?

Let's discuss the need for encapsulation in JavaScript via the following example.

For example, you have defined the below object in your code.

const car = {

Brand: "Honda city",

model: "sx",

year: 2016,

}

Anyone can access the properties of the car object, as shown below.

car.Brand

Also, anyone can change the value of any property of the car object, as shown below.

car.Brand = true;

Here, the value of the Brand property is changed to the boolean from the string. So, it is required to secure the original data of the object and give limited access to the data to the outside world.

In this situation, the concept of encapsulation comes into the picture.

## Different Ways to Achieve Encapsulation in JavaScript

There are three different ways to achieve encapsulation.

* Using the function closures
* Using the ES6 classes
* Using the Getters and Setters

## 1.Achieving Encapsulation Using the Function Closures

A JavaScript function closure is a concept allowing the inner function to access the variable defined in the outer function even after the outer function is executed. The variables defined in the outer function can't be accessed outside its functional scope but can be accessed using the inner scope.

### Example

In the below code, shoppingCart() function is an outer function that contains the variables and function. The outer function has its private scope.

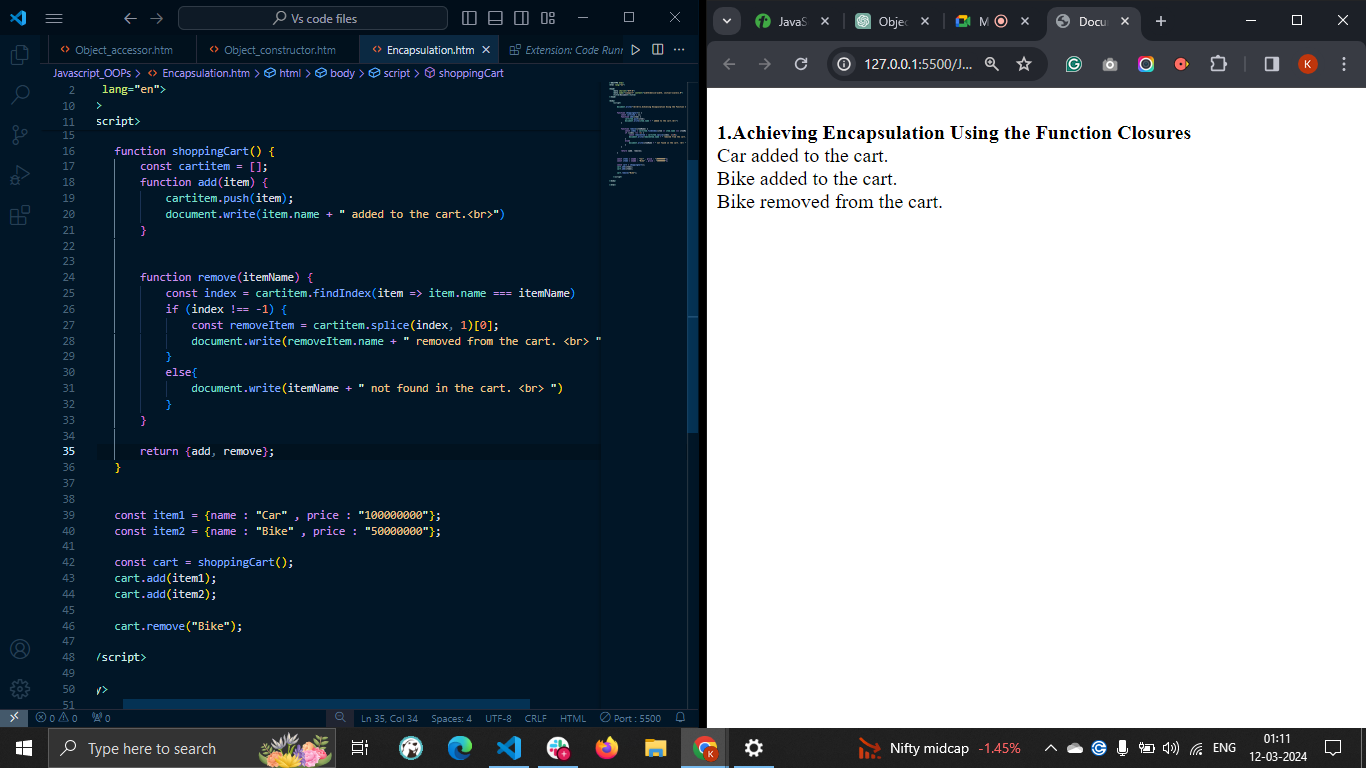
The carItems[] array is used to store the shopping cart's items.

The add() function can access the carItems[] array and add items.

The remove() function checks whether the cartItems[] contains the items you need to remove. If yes, it removes the item. Otherwise, it prints the message that you can't remove the item.

The shoppingCart() function returns the object containing the add() and remove() functions.

After creating a new instance of the shoppingCart() function, you can use the add() and remove() functions to manipulate the shopping cart data.



## 2.Achieving Encapsulation Using ES6 Classes and Private Variables

In JavaScript, you can use classes and private variables to achieve the encapsulation.

### Private Variables (Fields) in JavaScript

To define the private class variables, you can write a variable name followed by the ‘#’ sign. For example, 'name' is a private variable in the below code.

class car {

#name= "TATA";

}

If you try to access the name by the instance of the class, it will give you an error that private fields can't be accessed outside the class.

To achieve encapsulation, you can define the private variables in the class and give them access to the outside world using different methods.

### Example

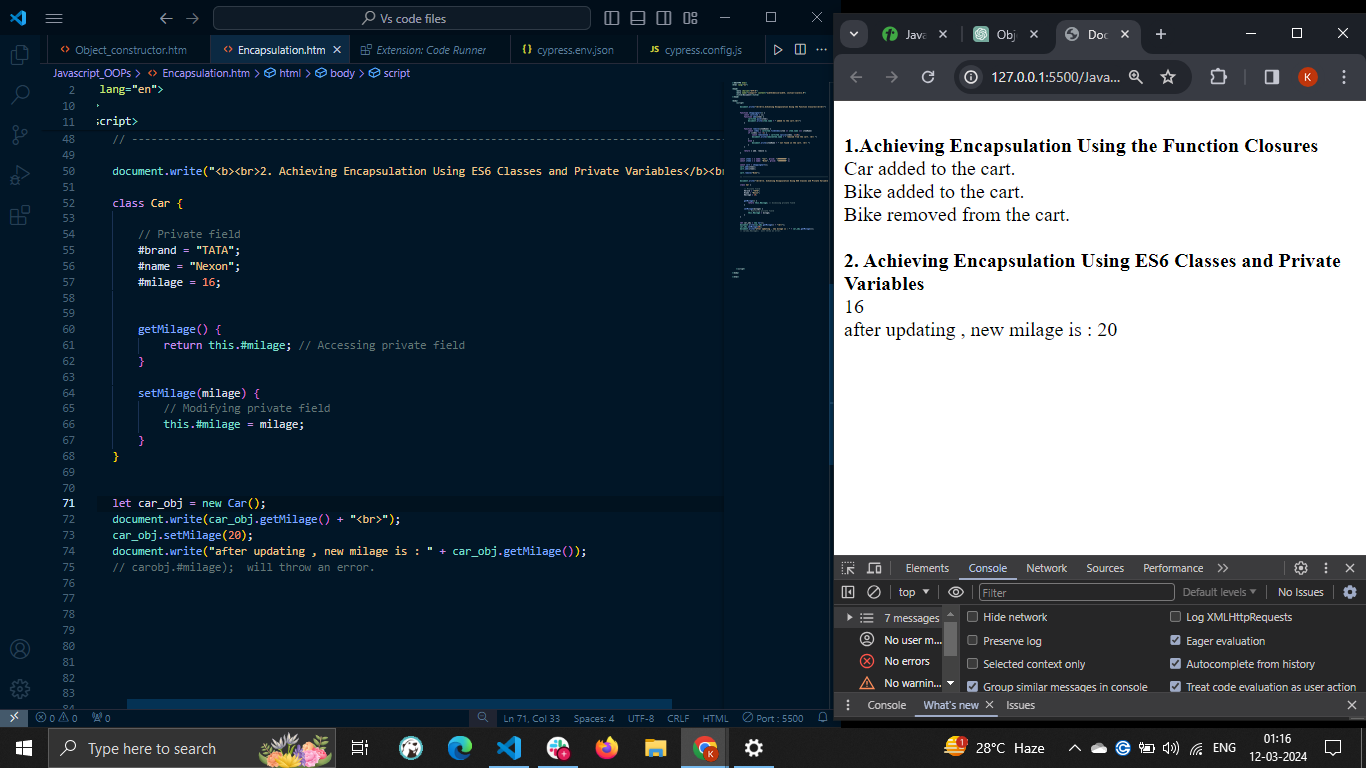
In the example below, we have defined the car class.

The car class contains the 'brand', 'name', and 'milage' private variables.

The getMilage() method is defined to return the milage of the car, and the setMilage() method is used to set the milage of the method.

We created the car class's object and used the method to access and modify the private fields. If you try to access the private field of the class, the code will throw an error.

You can also define more methods in the class to access and modify other private fields.



## 3. Achieving Encapsulation Using the Getters and Setters

The JavaScript getters and setters can be defined using the get and set keywords, respectively. The getters are used to get the class properties, and setters are used to update the class properties.

They are very similar to the class methods but defined using the get/set keyword followed by the method name.

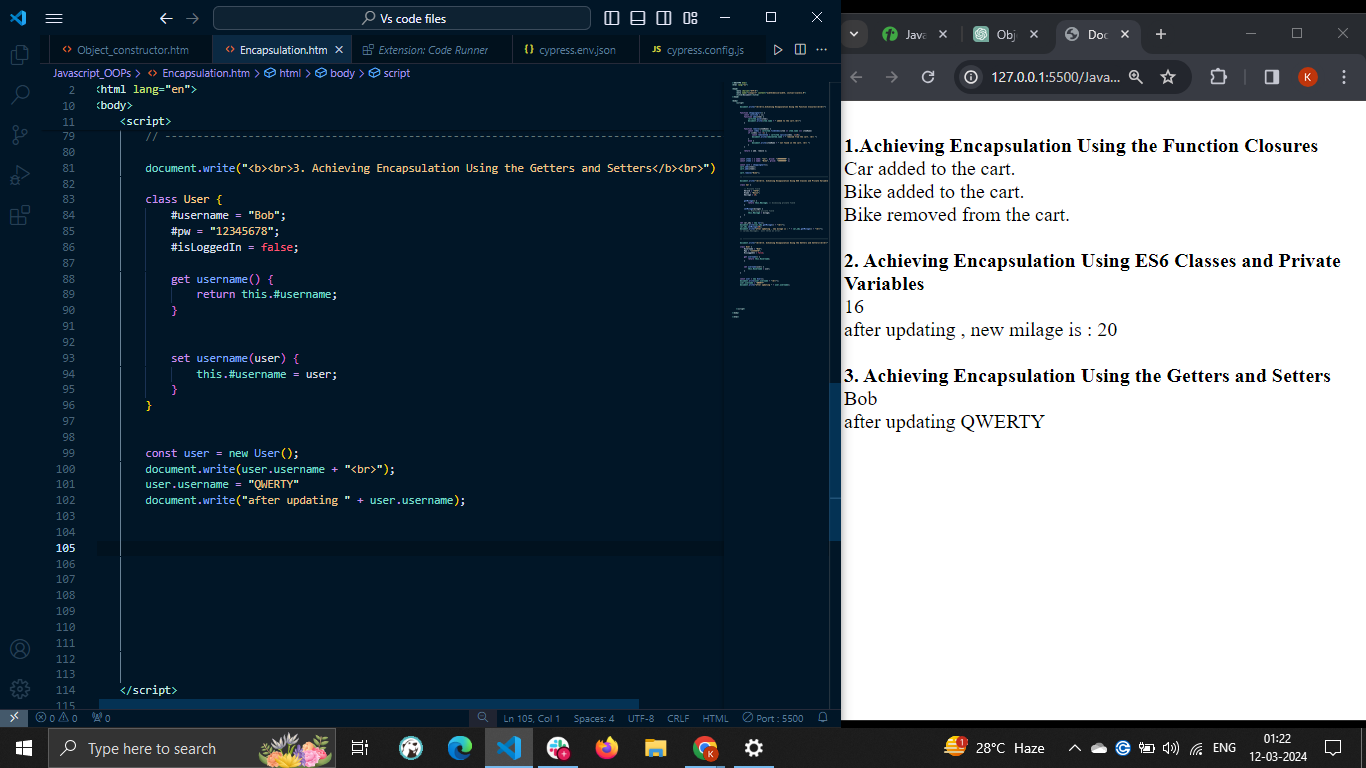
### Example

In the example below, we have defined the User class containing the three private fields named username, password, and isLoggedIn.

The getters and setters named username are defined to get and set user names. Here, you can observe that name of the getters and setters method is the same.

After that, we create an object of the class and use the getters and setters as the property to access and update the username field of the class.

You may also create getters and setters for the other class fields.



Benefits of Encapsulation in JavaScript

Here, we have listed some benefits of encapsulation in JavaScript −

* **Data protection** − The encapsulation allows you to control the access of the class data by making them private. You can expose the required data and methods only. So, no one can modify the data by mistake. Also, you can validate the data while updating them. If new data is not valid, you can throw an error.
* **Code reusability** − The class is a template for the object, and you can reuse it to create objects with different data.
* **Code Maintenance** − The encapsulation makes it easy to maintain the code as each object is independent, and if you make changes to one object, it doesn't affect the other code.

# **JavaScript - Inheritance**

The concept of **inheritance** in JavaScript allows the child class to inherit the properties and methods of the parent class. Inheritance is also a fundamental concept of object-oriented programming like encapsulation and polymorphism.

Sometimes, you must add the properties and methods of the one class into another. For example, you have created a general class for the bike containing the same properties and methods for each bike. After that, you create a separate class for the bike "Honda", and you need to add all properties and methods to the "Honda" class. You can achieve it using inheritance.

Before ECMAScript 6 (ES6), the object's prototype was used for inheritance, but in ES6, the 'extends' keyword was introduced to inherit classes.

The following terminologies are used in this chapter

* **Parent class** − It is a class whose properties are inherited by other classes.
* **Child class** − It is a class that inherits the properties of the other class.

## 1.JavaScript Single Class Inheritance

You can use the '**extends**' keyword to inherit the parent class properties into the child class. In single class inheritance only a single class inherits the properties of another class.

### Syntax

You can follow the syntax below for the single-class inheritance.



In the above syntax, you can replace the 'childClass' with the name of the child class and 'parentClass' with the name of the parent class.

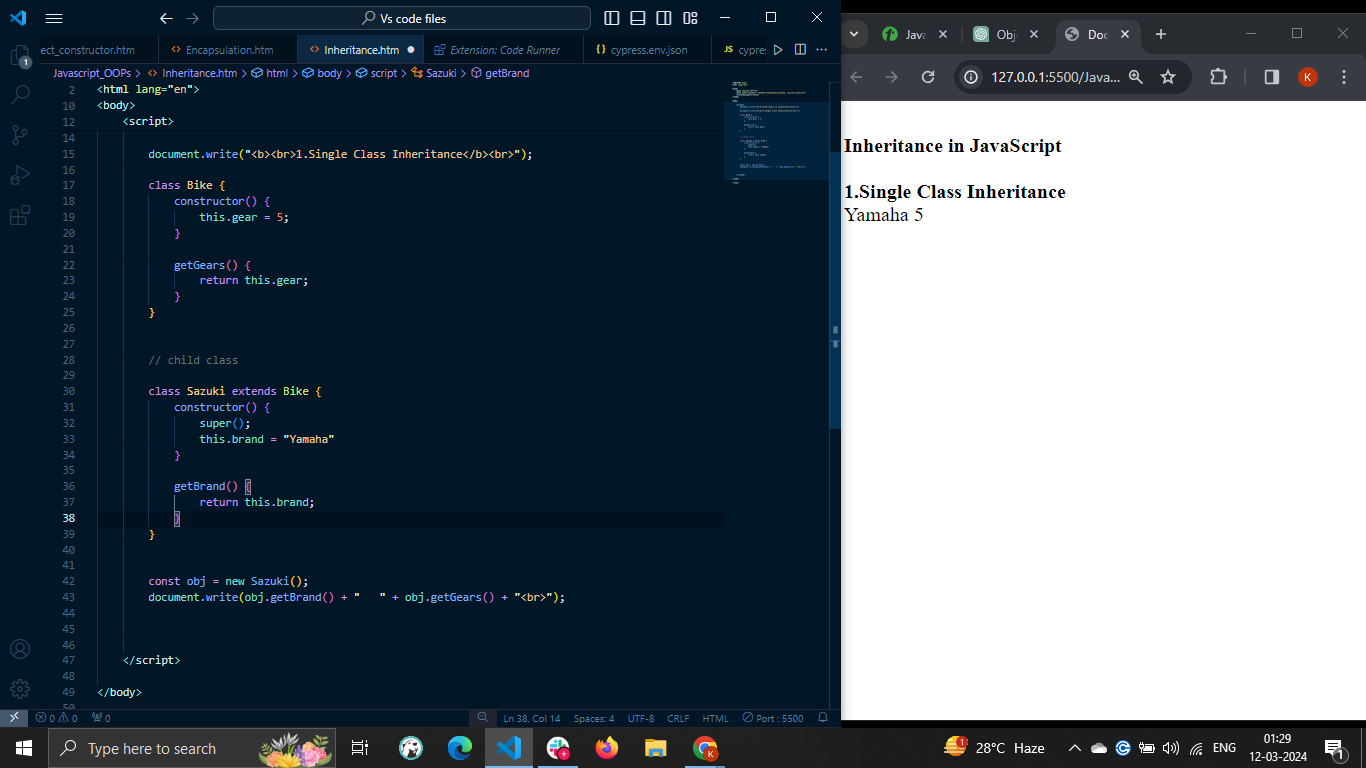
### Example: Single Class Inheritance

In the example below, the 'Bike' class is a parent class, and the 'Suzuki' class is a child class. The suzuki class inherits the properties of the Bike class.

The Bike class contains the constructor() method initializing the gears property and the getGears() method returning the value of the gears property.

The suzuki class contains the constructor() method to initialize the brand property and getBrand() method, returning the value of the brand property.

We have created an object of the 'suzuki' class. Using the 'suzuki' class instance, we invoke the getBrand() and getGears() methods.



## JavaScript super() Keyword

In the above example, we have initialized the 'gear' property of the Bike class with a static value. In real life, you need to initialize it with the dynamic value according to the model of the bike.

Now, the question is how to initialize the properties of the parent class from the child class. The solution is a super() keyword.

The super() keyword is used to invoke the method or access the properties of the parent class in the child class. By default, the super() keyword invokes the constructor function of the parent class. You can also pass the parameters to the super() keyword to pass it to the constructor of the parent class.

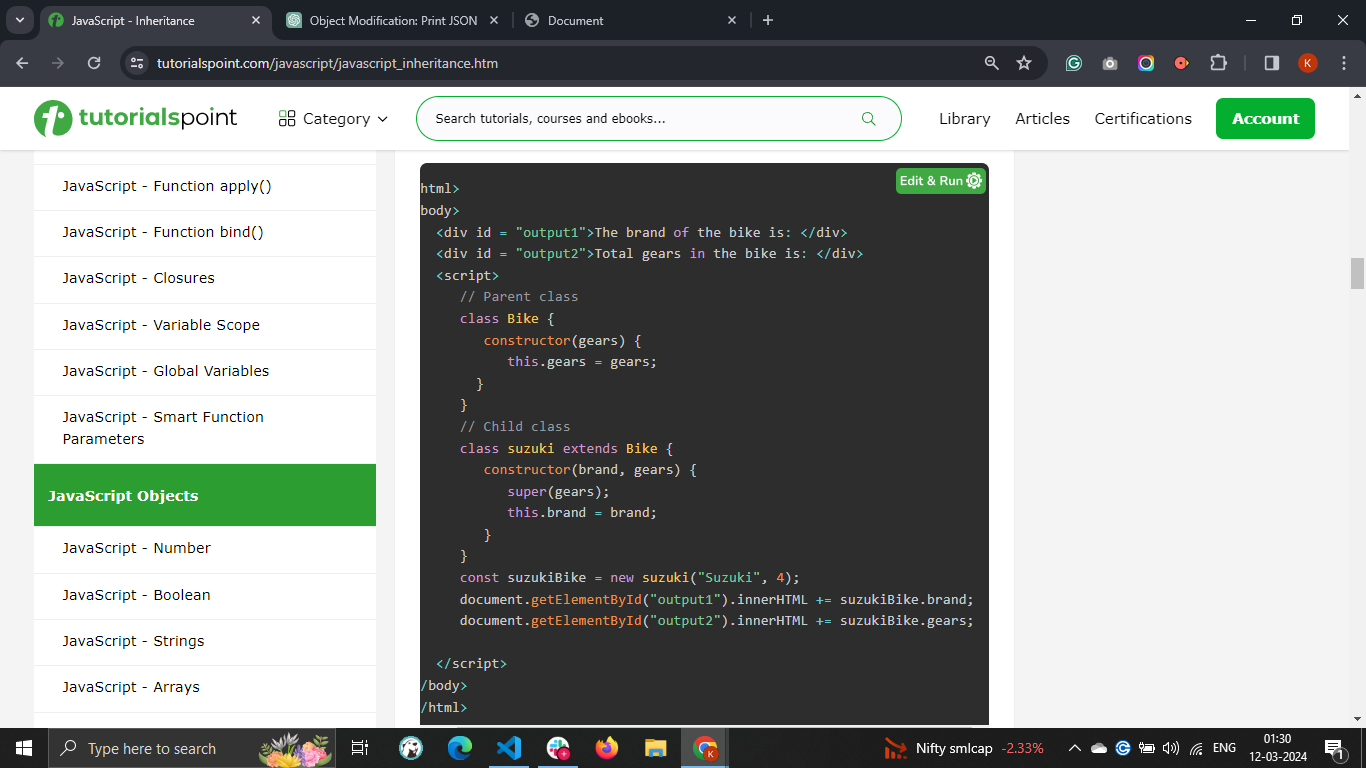
### Example: Using super() keyword to initialize the parent class properties

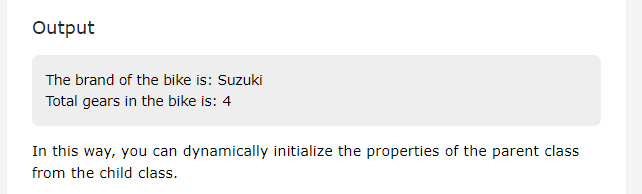
In the example below, suzuki class extends the Bike class.

The Bike class contains the constructor, taking gears as parameters and, using it, initializes the gears property.

The 'suzuki' class also contains the constructor, taking a brand and gears as a parameter. Using the brand parameter, it initializes the brand property and passes the gears parameter as an argument of the super() keyword.

After that, we create an object of the 'suzuki' class and pass the brand and gears as an argument of the constructor. You can see the dynamic value of the brand and gear property in the output.



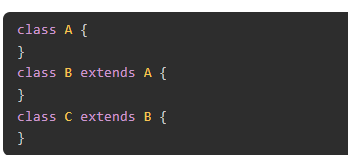


## 2. JavaScript Multilevel Inheritance

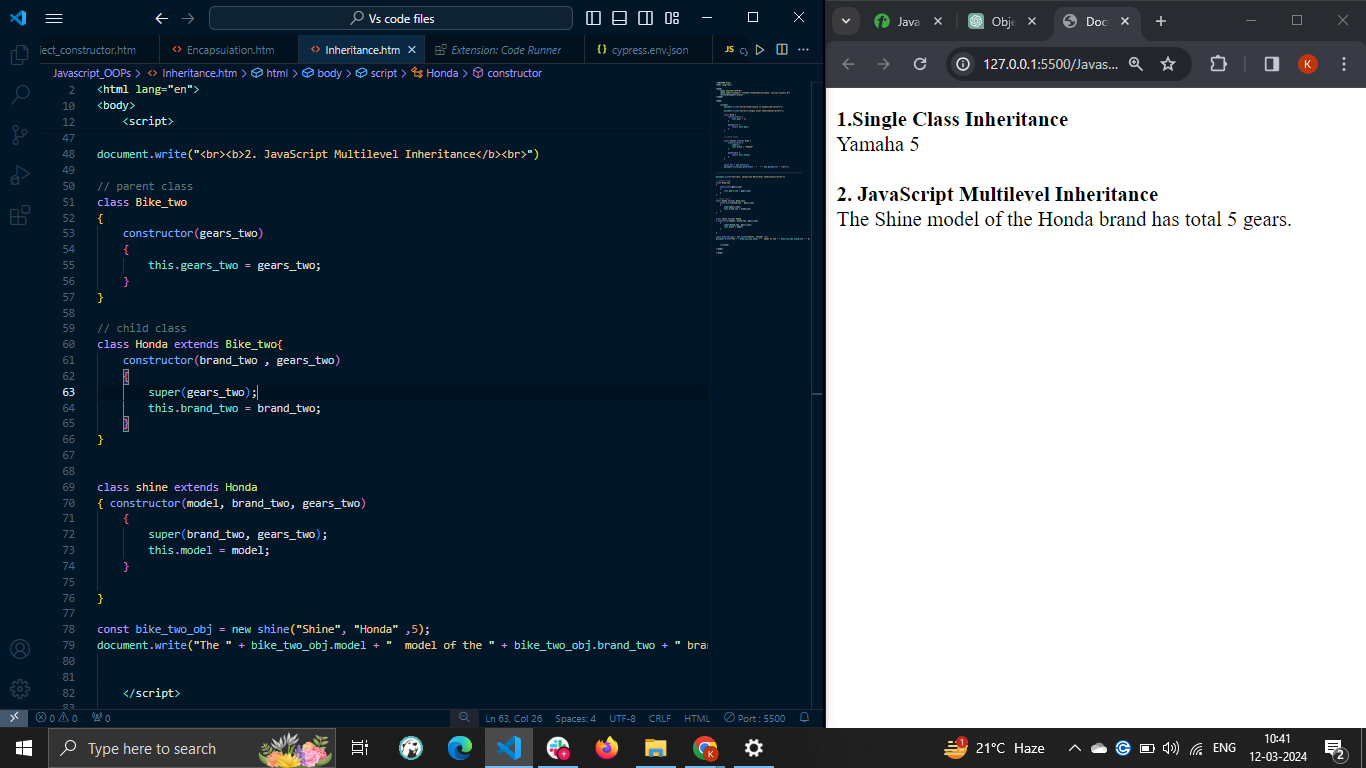
Multilevel inheritance is a type of inheritance in JavaScript. In multilevel inheritance, one class inherits the properties of another class, and other classes inherit current class properties.

### Syntax

Users can follow the syntax below for the multilevel inheritance.



Example :



## 3. JavaScript Hierarchical Inheritance

In JavaScript hierarchical inheritance, one class is inherited by multiple classes.

### Syntax

The syntax of JYou can follow the syntax below for the hierarchical inheritance.

class A {

}

class B extends A {

}

Class C extends A {

}

In the above syntax, B and C both classes inherit the properties of the A class.

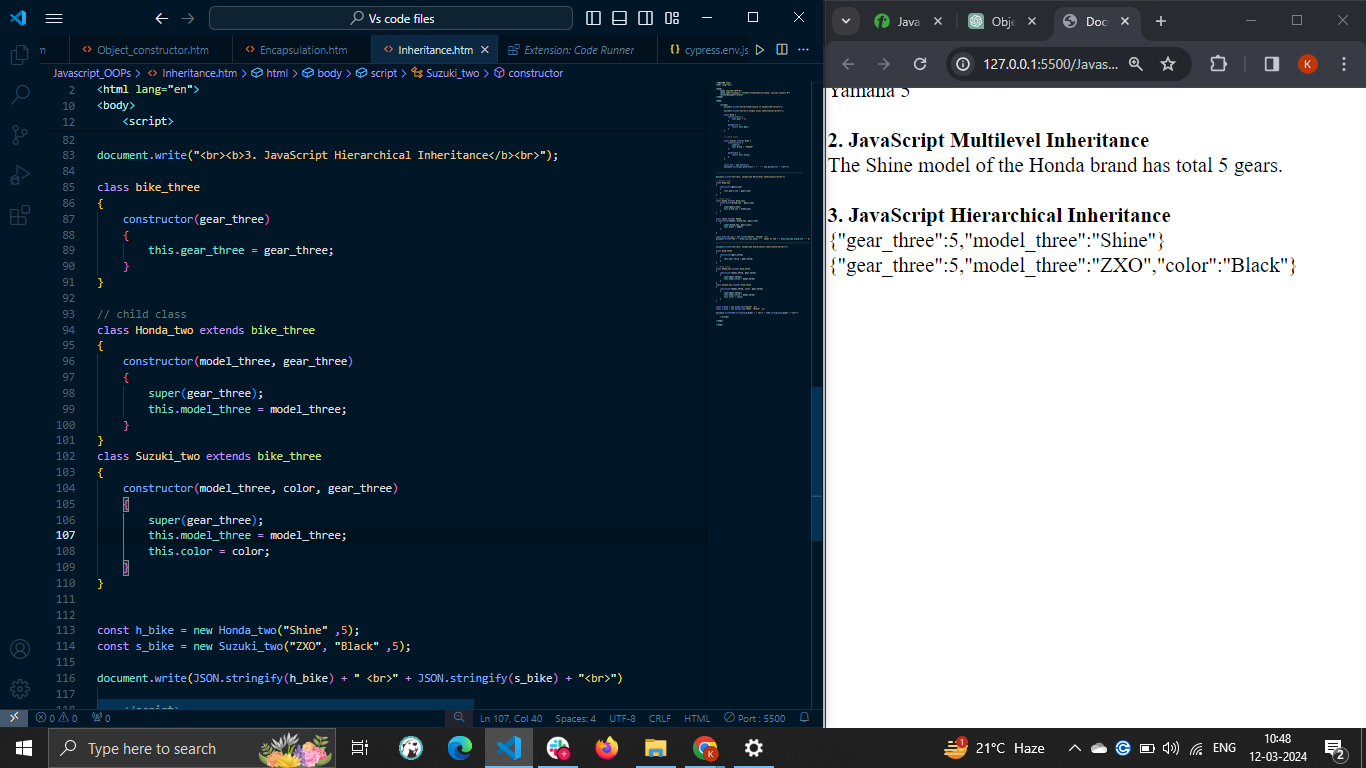
### Example

In the example below, the Bike class contains the gears property and is initialized using the constructor() method.

The Honda class extends the Bike class. The constructor() method of the Honda class initializes the properties of the Bike class using the super() keyword and model property of itself.

The Suzuki class inherits the Bike class properties. The constructor() method of the Suzuki class also initializes the Bike class properties and the other two properties of itself.

After that, we create objects of both Honda and Suzuki classes and access their properties.



## Inheriting Static Members of the Class

In JavaScript, you can invoke the static methods of the parent class using the super keyword in the child class. Outside the child class, you can use the child class name to invoke the static methods of the parent and child class.

### Example

In the example below, the Bike class contains the getDefaultBrand() static method. The Honda class also contains the Bikename() static method.

In the Bikename() method, we invoke the getDefaultBrand() method of the parent class using the 'super' keyword.

Also, we execute the Bikename() method using the 'Honda' class name.

<html>

<body>

<p id = "output">The bike name is: </p>

<script>

// Parent class

class Bike {

constructor(gears) {

this.gears = gears;

}

static getDefaultBrand() {

return "Yamaha";

}

}

// Child class

class Honda extends Bike {

constructor(model, gears) {

super(gears);

this.model = model;

}

static BikeName() {

return super.getDefaultBrand() + ", X6";

}

}

document.getElementById("output").innerHTML += Honda.BikeName();

</script>

</body>

</html>

#### Output

The bike name is: Yamaha, X6

*When you execute any method using the 'super' keyword in the multilevel inheritance, the class finds the methods in the parent class. If the method is not found in the parent class, it finds in the parent's parent class, and so on.*

## JavaScript Prototype Based Inheritance

You can also update or extend the prototype of the class to inherit the properties of the multiple classes to the single class. So, it is also called multiple inheritance.

### Syntax

You can follow the syntax below to use prototype-based inheritance.

Child.prototype = Instance of parent class

In the above syntax, we assign the parent class instance to the child object's prototype.

### Example: JavaScript Prototype Based Inheritance

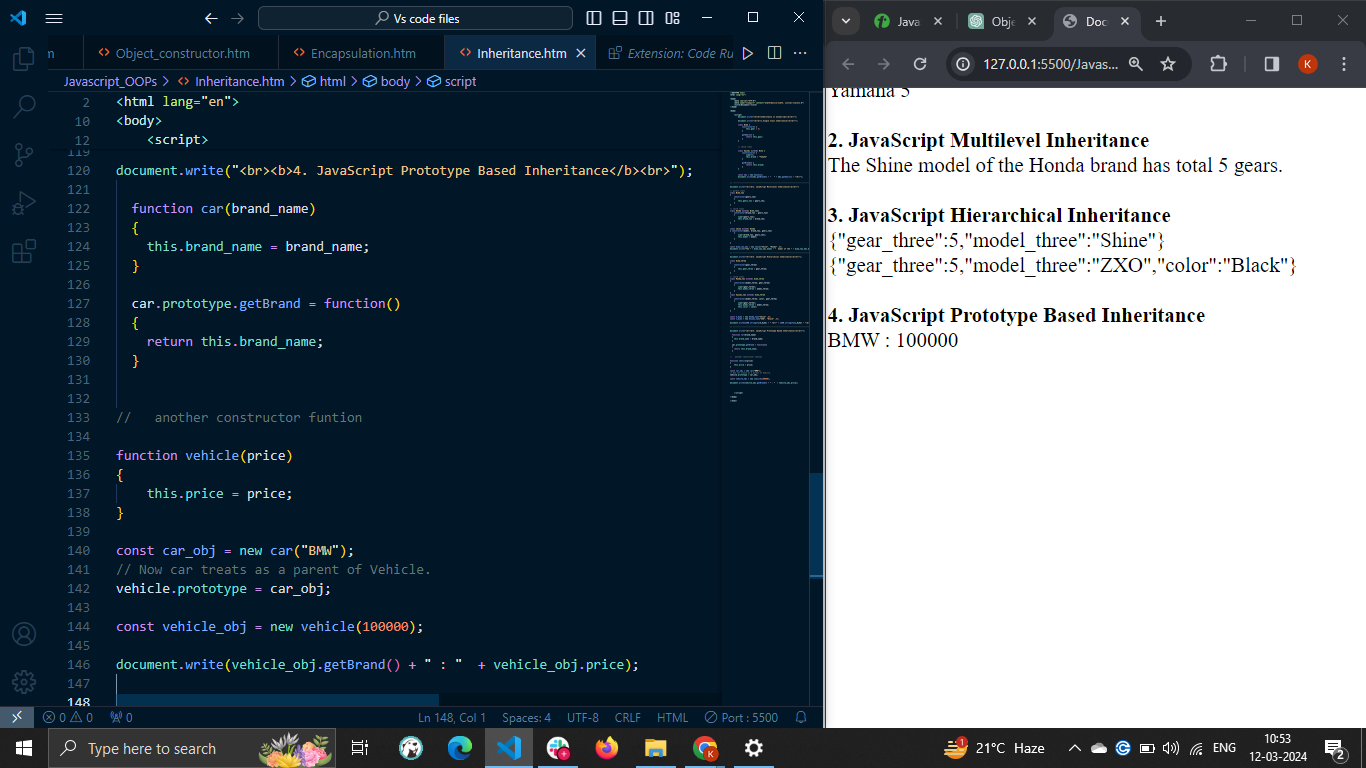
In the example below, Bike() is an object constructor that initializes the brand property.

After that, we add the getBrand() method to the prototype of the Bike() function.

Next, we have created the Vehicle() object constructor and instance of the Bike() constructor.

After that, we update the prototype of the Vehicle class with an instance of the Bike. Here, Vehicle works as a child class and Bike as a parent class.

We access the getBrand() method of the Bike() function's prototype using the instance of the Vehicle () function.



*You can't access the private members to the parent class in the child class.*

## Benefits of Inheritance

Here, we will learn the benefits of the inheritance concept in JavaScript.

* **Code reusability** − The child class can inherit the properties of the parent class. So, it is the best way to reuse the parent class code.
* **Functionality extension** − You can add new properties and methods to extend the parent class functionality in each child class.
* **Code maintenance** − It is easier to maintain a code as you can divide the code into sub-classes.
* Multilevel and hierarchical inheritance allows you to combine data together.

# **JavaScript - Abstraction**

## Abstraction in JavaScript

The Abstraction in JavaScript can be achieved using the abstract class. In object-oriented programming, the abstraction concept allows you to hide the implementation details and expose features only to the users.

For example, you can execute the Math object methods in JavaScript by accessing the method using its name but can’t see how it is implemented. Same way array methods like push(), pop(), etc., can be executed, but you don’t know how it is implemented internally.

So, the abstraction makes code cleaner by exposing the required features and hiding the internal implementation details.

## How to Achieve Abstraction in JavaScript?

In most programming languages, you can achieve abstraction using the **abstract class**. The **abstract class** contains only method declaration but not implementation. Furthermore, you need to implement the methods declared in the abstract class into the child class. Also, you can’t create an instance of the abstract class.

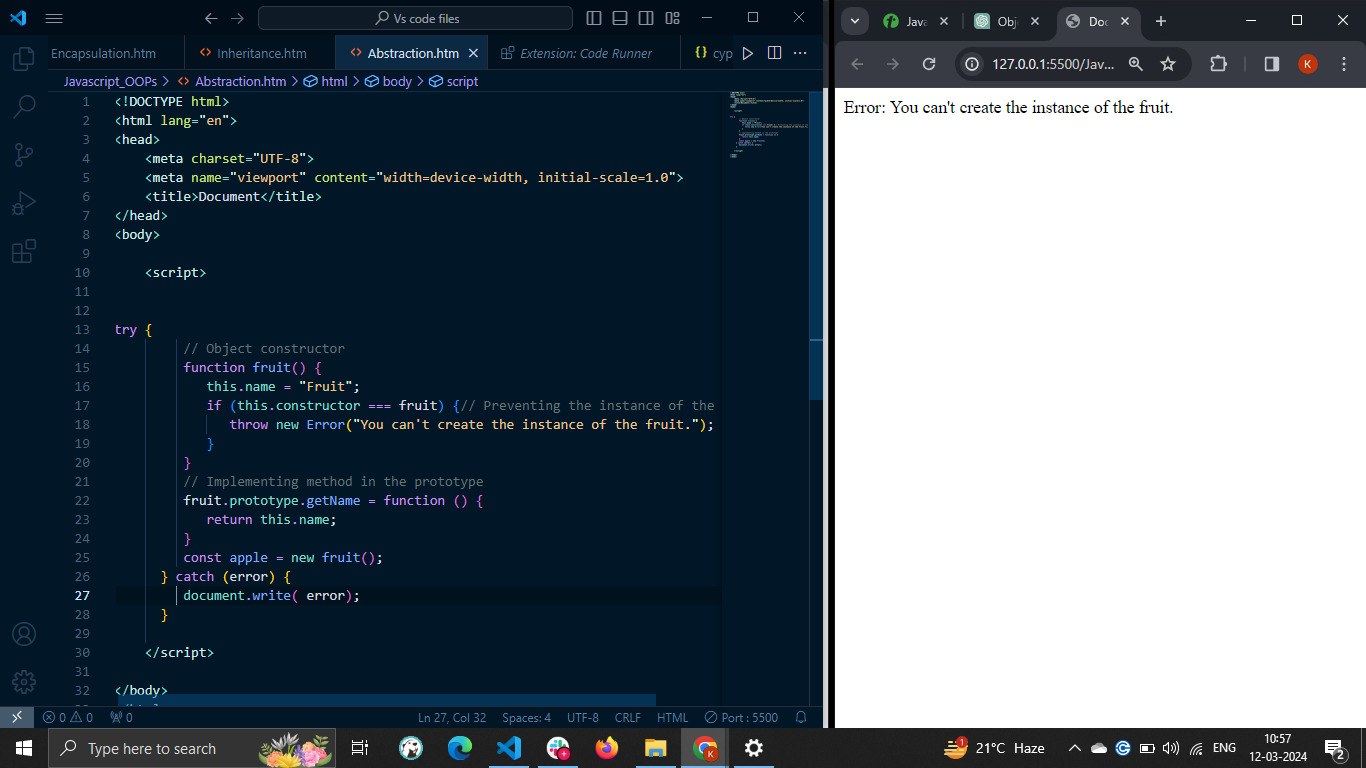
JavaScript doesn’t allow to create an abstract class like Java or CPP natively, but you can achieve the same functionality using the object constructor function.

First, let’s create an abstract class using the example below.

### Creating the Abstract Class

In the below example, the fruit() function initializes the name property. When anyone creates an instance of the fruit(), the value of the constructor property becomes equal to the ‘fruit’. So, we throw an error to prevent creating an instance of the fruit.

Also, we have added the getName() method to the prototype. After that, we create an instance of the fruit() constructor, and you can observe the error in the output



In the above example, you learned to achieve the abstract class functionality.

Now, you will learn to extend the abstract class and implement the methods defined in the abstract class via the example below.

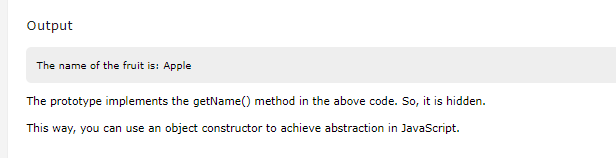
### Extending the Abstract Class

In the example below, fruit() constructor is similar to the above example. We have implemented the Apple() constructor, initializing the name property.

After that, we assign the prototype of the fruit() function to the Apple() function using the Object.create() method. It means Apple() function inherits the properties and methods of the fruit() class.

After that, we have created the instance of the Apple() class and invoked the getName() method.





.