**CLASSES**

JavaScript Classes

[❮ Previous](https://www.w3schools.com/js/js_function_closures.asp)[Next ❯](https://www.w3schools.com/js/js_class_inheritance.asp)

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".

A JavaScript class is **not** an object.

It is a **template** for JavaScript objects.

Using a Class

When you have a class, you can use the class to create objects:

Example

let myCar1 = new Car("Ford", 2014);  
let myCar2 = new Car("Audi", 2019);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_class_init)

The example above uses the **Car class** to create two **Car objects**.

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.

ADVERTISEMENT

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() { ... }  
}

Create a Class method named "age", that returns the Car age:

Example

class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
  age() {  
    let date = new Date();  
    return date.getFullYear() - this.year;  
  }  
}  
  
let myCar = new Car("Ford", 2014);  
document.getElementById("demo").innerHTML =  
"My car is " + myCar.age() + " years old.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_class_method)

You can send parameters to Class methods:

Example

class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
  age(x) {  
    return x - this.year;  
  }  
}  
  
let date = new Date();  
let year = date.getFullYear();  
  
let myCar = new Car("Ford", 2014);  
document.getElementById("demo").innerHTML=  
"My car is " + myCar.age(year) + " years old.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_class_method2)

# JavaScript Class Inheritance

[❮ Previous](https://www.w3schools.com/js/js_class_intro.asp)[Next ❯](https://www.w3schools.com/js/js_class_static.asp)

## 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:

### Example

Create a class named "Model" which will inherit the methods from the "Car" class:

class Car {  
  constructor(brand) {  
    this.carname = brand;  
  }  
  present() {  
    return 'I have a ' + this.carname;  
  }  
}  
  
class Model extends Car {  
  constructor(brand, mod) {  
    super(brand);  
    this.model = mod;  
  }  
  show() {  
    return this.present() + ', it is a ' + this.model;  
  }  
}  
  
let myCar = new Model("Ford", "Mustang");  
document.getElementById("demo").innerHTML = myCar.show();

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_classes_inherit)

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.

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

ADVERTISEMENT

## Getters and Setters

Classes also allows you to use getters and setters.

It can be smart to use getters and setters for your properties, especially if you want to do something special with the value before returning them, or before you set them.

To add getters and setters in the class, use the get and set keywords.

### Example

Create a getter and a setter for the "carname" property:

class Car {  
  constructor(brand) {  
    this.carname = brand;  
  }  
  get cnam() {  
    return this.carname;  
  }  
  set cnam(x) {  
    this.carname = x;  
  }  
}  
  
let myCar = new Car("Ford");  
  
document.getElementById("demo").innerHTML = myCar.cnam;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_classes_getters)

**Note:** even if the getter is a method, you do not use parentheses when you want to get the property value.

The name of the getter/setter method cannot be the same as the name of the property, in this case carname.

Many programmers use an underscore character \_ before the property name to separate the getter/setter from the actual property:

### Example

You can use the underscore character to separate the getter/setter from the actual property:

class Car {  
  constructor(brand) {  
    this.\_carname = brand;  
  }  
  get carname() {  
    return this.\_carname;  
  }  
  set carname(x) {  
    this.\_carname = x;  
  }  
}  
  
let myCar = new Car("Ford");  
  
document.getElementById("demo").innerHTML = myCar.carname;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_classes_getters2)

To use a setter, use the same syntax as when you set a property value, without parentheses:

### Example

Use a setter to change the carname to "Volvo":

class Car {  
  constructor(brand) {  
    this.\_carname = brand;  
  }  
  get carname() {  
    return this.\_carname;  
  }  
  set carname(x) {  
    this.\_carname = x;  
  }  
}  
  
let myCar = new Car("Ford");  
myCar.carname = "Volvo";  
document.getElementById("demo").innerHTML = myCar.carname;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_classes_setter)

## Hoisting

Unlike functions, and other JavaScript declarations, class declarations are not hoisted.

That means that you must declare a class before you can use it:

### Example

//You cannot use the class yet.  
//myCar = new Car("Ford")  
//This would raise an error.  
  
class Car {  
  constructor(brand) {  
    this.carname = brand;  
  }  
}  
  
//Now you can use the class:  
let myCar = new Car("Ford")

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_classes_hoisting)

# JavaScript Static Methods

[❮ Previous](https://www.w3schools.com/js/js_class_inheritance.asp)[Next ❯](https://www.w3schools.com/js/js_callback.asp)

Static class methods are defined on the class itself.

You cannot call a static method on an object, only on an object class.

### Example

class Car {  
  constructor(name) {  
    this.name = name;  
  }  
  static hello() {  
    return "Hello!!";  
  }  
}  
  
let myCar = new Car("Ford");  
  
// You can call 'hello()' on the Car Class:  
document.getElementById("demo").innerHTML = Car.hello();  
  
// But NOT on a Car Object:  
// document.getElementById("demo").innerHTML = myCar.hello();  
// this will raise an error.

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_class_static)

If you want to use the myCar object inside the static method, you can send it as a parameter:

### Example

class Car {  
  constructor(name) {  
    this.name = name;  
  }  
  static hello(x) {  
    return "Hello " + x.name;  
  }  
}  
let myCar = new Car("Ford");  
document.getElementById("demo").innerHTML = Car.hello(myCar);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_class_static2)