JavaScript Objects and Classes

.NET

In JavaScript, classes are "special functions". Just as you can define function expressions and function declarations, the class syntax has two components: class expression and class declaration.

HTTPS://DEVELOPER.MOZILLA.ORG/EN-US/DOCS/WEB/JAVASCRIPT/REFERENCE/CLASSES

JavaScript objects

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash
https://javascript.info/object

In JavaScript, everything is an object. This means any variable in JS can potentially be used to store ***properties*** (***key:value*** pairs) and other objects.

JS objects are stored by ***reference***. An empty ***object*** can be created in two ways. The class variable holds a reference to the memory location of the object on the ***heap***.

```
1  let user = new Object(); // "object constructor" syntax
2  let user = {};  // "object literal" syntax
```

An ***Object Literal*** is created with properties. Property values are accessible using dot (.) notation.

```
1  let user = {     // an object
2    name: "John",  // by key "name" store value "John"
3    age: 30        // by key "age" store value 30
4  };
```

JS Objects – Property Values and Shorthand

https://javascript.info/object#property-value-shorthand
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this#The_bind_method

---

JavaScript has a shorthand for declaring and setting object variables. The below examples are all equivalent objects. How objects are declared determines if you end up with a reusable template for objects.

```
1  function makeUser(name, age) {
2    return {
3      name: name,
4      age: age,
5      // ...other properties
6    };
7  }
8
9  let user = makeUser("John", 30);
10 alert(user.name); // John
```

makeUser is reusable

```
1  function makeUser(name, age) {
2    return {
3      name, // same as name: name
4      age,  // same as age: age
5      // ...
6    };
7  }
```

makeUser is reusable

```
1  let user = {
2    name,  // same as name:name
3    age: 30
4  };
```

user is NOT reusable

JS Objects – Objects in Objects

https://javascript.info/object#cloning-and-merging-object-assign

An object can contain another object. In this example, you would access ***height*** with

let height = user.sizes.height

```
let user = {
  name: "John",
  sizes: {
    height: 182,
    width: 50
  }
};

alert( user.sizes.height ); // 182
```

Another ***object*** or ***function*** can be assigned to an object after creation. Here, user is dynamically assigned the function sayHi() as a new property (also called sayHi).

```
let user = {
  // ...
};

// first, declare
function sayHi() {
  alert("Hello!");
};

// then add as a method
user.sayHi = sayHi;

user.sayHi(); // Hello!
```

JS Objects - Accessing Properties

https://javascript.info/object#property-existence-test-in-operator
https://javascript.info/object#the-for-in-loop

It's possible to access any property of an ***object***. The below will return undefined because the property doesn't exist.

let value = user.key;

The in operator returns true of the property exists, false if not.

let exists = 'name' in user;

Use the ***for...in*** loop to access each property of an object in sequence.

The keyword ***this*** can be used to specify the containing object to disambiguate variable names.

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // keys
  alert( key );  // name, age, isAdmin
  // values for the keys
  alert( user[key] ); // John, 30, true
}
```

```
let user = {
  name: "John",
  age: 30,

  sayHi() {
    // "this" is the "current object"
    alert(this.name);
  }

};

user.sayHi(); // John
```

JavaScript Classes

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

The ***class*** syntax does not introduce a new object-oriented inheritance model to JavaScript. Classes are "special functions".

Just as you can define ***function expressions*** and ***function declarations***, the class syntax has two components: ***class expressions*** and ***class declarations***.

Class Declaration	Class Expression

class Rectangle {
  constructor(height, width) {
  this.height = height;
  this.width = width;
  }
}
	Class expressions can be named or unnamed. There is an implicit 'name' property in the class object. The name given to a class expression is local to the classes body. It can be retrieved through the classes (not the instance's) name property.

A class must be declared before they can be accessed. (no ***Hoisting***)	

```
1  // unnamed
2  let Rectangle = class {
3    constructor(height, width) {
4      this.height = height;
5      this.width = width;
6    }
7  };
8  console.log(Rectangle.name);
9  // output: "Rectangle"
10
11 // named
12 let Rectangle = class Rectangle2 {
13   constructor(height, width) {
14     this.height = height;
15     this.width = width;
16   }
17 };
18 console.log(Rectangle.name);
19 // output: "Rectangle2"
```

JS Class Parts

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

- The *constructor method* creates and initializes an object created from a class template. There can be only one constructor in each class.
- *Instance Properties* must be defined inside of class methods.
- *Prototype Methods* are declared in the class and are available through an instance of the class.
- *Static Methods* are called without instantiating their class and cannot be called through a class instance. (below) Static members cannot call non-static members.

```
15  const p1 = new Point(5, 5);
16  const p2 = new Point(10, 10);
17  p1.distance; //undefined
18  p2.distance; //undefined
19
20  console.log(Point.distance(p1, p2));
```

```
1   class Point {
2     constructor(x, y) {
3       this.x = x;
4       this.y = y;
5     }
6
7     static distance(a, b) {
8       const dx = a.x - b.x;
9       const dy = a.y - b.y;
10
11      return Math.hypot(dx, dy);
12    }
13  }
```

```
1   class Rectangle {
2     constructor(height, width) {
3       this.height = height;
4       this.width = width;
5     }
6     // Getter
7     get area() {
8       return this.calcArea();
9     }
10    // Method
11    calcArea() {
12      return this.height * this.width;
13    }
14  }
15
16  const square = new Rectangle(10, 10);
17
18  console.log(square.area); // 100
```

JS Functions – Constructors and 'new'

https://javascript.info/constructor-new

A ***constructor function*** in JavaScript serves the same purpose as a ***Class constructor*** in C#. ***Constructor functions*** technically are regular functions.

By convention:

1. Their names are in Pascal case.
2. They should be executed exclusively with the new operator.

```
1  function User(name) {
2    this.name = name;
3    this.isAdmin = false;
4  }
5
6  let user = new User("Jack");
7
8  alert(user.name); // Jack
9  alert(user.isAdmin); // false
```

When a function is executed with new, it implicitly does the following steps:

1. A new empty object is created and assigned to this.
2. The function body executes. Usually it modifies this, by adding new properties to it.
3. The value of this is returned.

The main purpose of constructors is to implement reusable object creation code.

```
1  function User(name) {
2    // this = {};  (implicitly)
3
4    // add properties to this
5    this.name = name;
6    this.isAdmin = false;
7
8    // return this;  (implicitly)
9  }
```

Getters and Setters

https://javascript.info/property-accessors#getters-and-setters

Accessor properties are functions that begin with the keywords get and set. They look like regular ***properties*** to external code.

Getters and ***Setters*** are accessed like properties. (instanceName.getterName).

Getters and ***Setters*** allow validation to be written into the class functionality.

Setters must have one parameter.

```
6   set name(value) {
7     if (value.length < 4) {
8       alert("Name is too shor
9       return;
10    }
11    this._name = value;
12  }
```

```
1  let user = {
2    name: "John",
3    surname: "Smith",
4
5    get fullName() {
6      return `${this.name} ${this.surname}`;
7    },
8
9    set fullName(value) {
10     [this.name, this.surname] = value.split(" ");
11   }
12 };
13
14 // set fullName is executed with the given value.
15 user.fullName = "Alice Cooper";
16
17 alert(user.name); // Alice
18 alert(user.surname); // Cooper
```

JavaScript [[Prototype]]

https://javascript.info/prototype-inheritance

*__proto__ is outdated and deprecated.

Objects have a hidden property, [[*Prototype*]], that is either null or references another object. This object is called a "***prototype***". When we want to read a property from an object and it isn't found, it's taken from the ***prototype***. This is called "***prototypal inheritance***". [[*Prototype*]] is internal and hidden, but you can manually set it.

- Multiple prototype inheritance is not allowed.
- __proto__ does not support writing or deleting.
- Inheritance can be chained.
- Inheritance cannot be circular.
- Getters/Setters are also inherited.

Inherit Methods

```
let animal = {
  eats: true,
  walk() {
    alert("Animal walk");
  }
};

let rabbit = {
  jumps: true,
  __proto__: animal
};

// walk is taken from the prototype
rabbit.walk(); // Animal walk
```

Hierarchical Inheritance

```
let animal = {
  eats: true,
  walk() {
    alert("Animal walk");
  }
};

let rabbit = {
  jumps: true,
  __proto__: animal
};

let longEar = {
  earLength: 10,
  __proto__: rabbit
};

// walk is taken from the prototype chain
longEar.walk(); // Animal walk
alert(longEar.jumps); // true (from rabbit)
```

JavaScript Prototypes

https://javascript.info/function-prototype
https://javascript.info/prototype-methods

*__proto__ is outdated and deprecated.

Prototypal Inheritance was one of the core features of JS originally, but there was no direct access to it. The only method that worked reliably was a "prototype" property of the constructor function.

There are many scripts that still use __proto__. Remember that ***prototype*** is a default ***property*** provided in the ***constructor***.

In this example, setting

Rabbit.prototype = animal

sets its ***prototype*** to ***animal***.

```
1  let animal = {
2    eats: true
3  };
4
5  function Rabbit(name) {
6    this.name = name;
7  }
8
9  Rabbit.prototype = animal;
10
11 let rabbit = new Rabbit("White Rabbit");
12 //  rabbit.__proto__ == animal
13 alert( rabbit.eats ); // true
```

JavaScript Objects without __proto__

https://javascript.info/function-prototype
https://javascript.info/prototype-methods

***__proto__ is outdated and deprecated.**

Instead of __proto__, use:

- *Object.create(proto class)*. This returns an empty object with given proto class as [[Prototype]]
- *Object.create(proto class, {additional descriptors})* adds optional property descriptors.
- *Object.getPrototypeOf(obj)* – returns the [[Prototype]] of obj.
- *Object.setPrototypeOf(obj, proto)* – sets the [[Prototype]] of obj to proto.

```
let animal = {
  eats: true
};

// create a new object with animal as a prototype
let rabbit = Object.create(animal);

alert(rabbit.eats); // true

alert(Object.getPrototypeOf(rabbit) === animal); // true

Object.setPrototypeOf(rabbit, {}); // change the prototyp
                                   // change the prototype of rabbit to {}
```

JavaScript Class Inheritance

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes#Sub_classing_with_extends

---

The ***extends*** keyword is used in class declarations or class expressions to create a ***class*** as a ***child*** of another ***class***.

If there is a ***constructor*** present in the ***subclass***, it needs to first call ***super()*** before using "***this***".

```
1  class Animal {
2    constructor(name) {
3      this.name = name;
4    }
5
6    speak() {
7      console.log(`${this.name} makes a noise.`);
8    }
9  }
10
11 class Dog extends Animal {
12   constructor(name) {
13     super(name); // call the super class constructor
14   }                and pass in the name parameter
15
16   speak() {
17     console.log(`${this.name} barks.`);
18   }
19 }
20
21 let d = new Dog('Mitzie');
22 d.speak(); // Mitzie barks.
```