



→ [The JavaScript language](#) → [Classes](#)

20th January 2020

# Class basic syntax

*In object-oriented programming, a class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods).*

“ Wikipedia

In practice, we often need to create many objects of the same kind, like users, or goods or whatever.

As we already know from the chapter [Constructor, operator "new"](#), `new` function can help with that.

But in the modern JavaScript, there's a more advanced "class" construct, that introduces great new features which are useful for object-oriented programming.

## The "class" syntax

The basic syntax is:

```
1 class MyClass {
2   // class methods
3   constructor() { ... }
4   method1() { ... }
5   method2() { ... }
6   method3() { ... }
7   ...
8 }
```

Then use `new MyClass()` to create a new object with all the listed methods.

The `constructor()` method is called automatically by `new`, so we can initialize the object there.

For example:

```
1 class User {
2
3   constructor(name) {
4     this.name = name;
5   }
6
7   sayHi() {
```



```

8     alert(this.name);
9   }
10
11 }
12
13 // Usage:
14 let user = new User("John");
15 user.sayHi();

```

When `new User("John")` is called:

1. A new object is created.
2. The `constructor` runs with the given argument and assigns `this.name` to it.

...Then we can call object methods, such as `user.sayHi()`.

### No comma between class methods

A common pitfall for novice developers is to put a comma between class methods, which would result in a syntax error.

The notation here is not to be confused with object literals. Within the class, no commas are required.

## What is a class?

So, what exactly is a `class`? That's not an entirely new language-level entity, as one might think.

Let's unveil any magic and see what a class really is. That'll help in understanding many complex aspects.

In JavaScript, a class is a kind of function.

Here, take a look:

```

1 class User {
2   constructor(name) { this.name = name; }
3   sayHi() { alert(this.name); }
4 }
5
6 // proof: User is a function
7 alert(typeof User); // function

```



What `class User {...}` construct really does is:

1. Creates a function named `User`, that becomes the result of the class declaration. The function code is taken from the `constructor` method (assumed empty if we don't write such method).
2. Stores class methods, such as `sayHi`, in `User.prototype`.

After `new User` object is created, when we call its method, it's taken from the prototype, just as described in the chapter [F.prototype](#). So the object has access to class methods.

We can illustrate the result of `class User` declaration as:





Here's the code to introspect it:

```

1  class User {
2    constructor(name) { this.name = name; }
3    sayHi() { alert(this.name); }
4  }
5
6  // class is a function
7  alert(typeof User); // function
8
9  // ...or, more precisely, the constructor method
10 alert(User === User.prototype.constructor); // true
11
12 // The methods are in User.prototype, e.g:
13 alert(User.prototype.sayHi); // alert(this.name);
14
15 // there are exactly two methods in the prototype
16 alert(Object.getOwnPropertyNames(User.prototype)); // constructor, sayHi
  
```

## Not just a syntactic sugar

Sometimes people say that `class` is a “syntactic sugar” (syntax that is designed to make things easier to read, but doesn't introduce anything new), because we could actually declare the same without `class` keyword at all:

```

1  // rewriting class User in pure functions
2
3  // 1. Create constructor function
4  function User(name) {
5    this.name = name;
6  }
7  // any function prototype has constructor property by default,
8  // so we don't need to create it
9
10 // 2. Add the method to prototype
11 User.prototype.sayHi = function() {
12   alert(this.name);
13 };
14
15 // Usage:
16 let user = new User("John");
17 user.sayHi();
  
```

The result of this definition is about the same. So, there are indeed reasons why `class` can be considered a syntactic sugar to define a constructor together with its prototype methods.

Still, there are important differences.

1. First, a function created by `class` is labelled by a special internal property `[[FunctionKind]]: "classConstructor"`. So it's not entirely the same as creating it manually.

And unlike a regular function, a class constructor must be called with `new` :

```
1 class User {  
2   constructor() {}  
3 }  
4  
5 alert(typeof User); // function  
6 User(); // Error: Class constructor User cannot be invoked without 'new'
```

Also, a string representation of a class constructor in most JavaScript engines starts with the “class...”

```
1 class User {  
2   constructor() {}  
3 }  
4  
5 alert(User); // class User { ... }
```

2. Class methods are non-enumerable. A class definition sets `enumerable` flag to `false` for all methods in the “prototype” .

That's good, because if we `for...in` over an object, we usually don't want its class methods.

3. Classes always use `strict` . All code inside the class construct is automatically in strict mode.

Besides, `class` syntax brings many other features that we'll explore later.

## Class Expression

Just like functions, classes can be defined inside another expression, passed around, returned, assigned, etc.

Here's an example of a class expression:

```
1 let User = class {  
2   sayHi() {  
3     alert("Hello");  
4   }  
5 };
```

Similar to Named Function Expressions, class expressions may have a name.

If a class expression has a name, it's visible inside the class only:

```
1 // "Named Class Expression"  
2 // (no such term in the spec, but that's similar to Named Function Expression)  
3 let User = class MyClass {  
4   sayHi() {  
5     alert(MyClass); // MyClass name is visible only inside the class  
6   }  
7 };  
8  
9 new User().sayHi(); // works, shows MyClass definition
```

```
10
11 alert(MyClass); // error, MyClass name isn't visible outside of the class
```

We can even make classes dynamically “on-demand”, like this:

```
1 function makeClass(phrase) {
2   // declare a class and return it
3   return class {
4     sayHi() {
5       alert(phrase);
6     };
7   };
8 }
9
10 // Create a new class
11 let User = makeClass("Hello");
12
13 new User().sayHi(); // Hello
```



## Getters/setters, other shorthands

Just like literal objects, classes may include getters/setters, computed properties etc.

Here's an example for `user.name` implemented using `get/set` :

```
1 class User {
2
3   constructor(name) {
4     // invokes the setter
5     this.name = name;
6   }
7
8   get name() {
9     return this._name;
10  }
11
12  set name(value) {
13    if (value.length < 4) {
14      alert("Name is too short.");
15      return;
16    }
17    this._name = value;
18  }
19
20 }
21
22 let user = new User("John");
23 alert(user.name); // John
24
25 user = new User(""); // Name is too short.
```



The class declaration creates getters and setters in `User.prototype` , like this:

```
1 Object.defineProperty(User.prototype, {
2   name: {
3     get() {
4       return this._name
5     },
6     set(name) {
7       // ...
8     }
9   }
10 });
```

Here's an example with a computed property name in brackets [ ... ] :

```
1 class User {
2
3   ['say' + 'Hi']() {
4     alert("Hello");
5   }
6
7 }
8
9 new User().sayHi();
```

## Class properties

### ⚠ Old browsers may need a polyfill

Class-level properties are a recent addition to the language.

In the example above, `User` only had methods. Let's add a property:

```
1 class User {
2   name = "Anonymous";
3
4   sayHi() {
5     alert(`Hello, ${this.name}!`);
6   }
7 }
8
9 new User().sayHi();
10
11 alert(User.prototype.sayHi); // placed in User.prototype
12 alert(User.prototype.name); // undefined, not placed in User.prototype
```

The property `name` is not placed into `User.prototype`. Instead, it is created by `new` before calling the constructor, it's a property of the object itself.

## Summary

The basic class syntax looks like this:

```
1 class MyClass {
2   prop = value; // property
3
4   constructor(...) { // constructor
5     // ...
6   }
7
8   method(...) {} // method
9
10  get something(...) {} // getter method
11  set something(...) {} // setter method
12
13  [Symbol.iterator]() {} // method with computed name (symbol here)
14  // ...
15 }
```

`MyClass` is technically a function (the one that we provide as `constructor`), while methods, getters and setters are written to `MyClass.prototype`.

In the next chapters we'll learn more about classes, including inheritance and other features.

## ✓ Tasks

### Rewrite to class

importance: 5

The `Clock` class is written in functional style. Rewrite it the “class” syntax.

P.S. The clock ticks in the console, open it to see.

[Open a sandbox for the task.](#)

solution



Previous lesson

Next lesson



Share  

 [Tutorial map](#)

## 💬 Comments

- If you have suggestions what to improve - please [submit a GitHub issue](#) or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.

- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>` , for more than 10 lines – use a sandbox ([plnkr](#), [JSBin](#), [codepen](#)...)