

# Classes

In this lesson you'll learn a clearer way of doing prototype inheritance in ES6.

## We'll cover the following



- Create a class
- Static methods
- set and get
- Extending our class
- Extending Arrays

Quoting MDN:

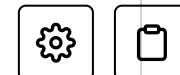
“Classes are primarily syntactic sugar over Javascript’s existing prototype-based inheritance. The class syntax **does not** introduce a new object-oriented inheritance model to JavaScript.”

That being said, let’s review prototypal inheritance before we jump into classes.

```
1 function Person(name,age) {  
2   this.name = name;  
}
```



```
3   this.age = age;
4 }
5
6 Person.prototype.greet = function(){
7   console.log("Hello, my name is " + this.name);
8 }
9
10 const alberto = new Person("Alberto", 26);
11 const caroline = new Person("Caroline",26);
12
13 alberto.greet();
14 // Hello, my name is Alberto
15 caroline.greet();
16 // Hello, my name is Caroline
```



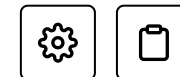
We added a new method to the prototype in order to make it accessible to all the new instances of Person that we created.

Ok, now that I refreshed your knowledge of prototypal inheritance, let's have a look at classes.

## Create a **class** #

There are two ways of creating a class:

- `class` declaration
- `class` expression



```
// class declaration
class Person {

}

// class expression
const person = class Person {
}
```



**Remember:** `class` declaration (and expression) are **not hoisted**, which means that unless you want to get a **ReferenceError** you need to declare your `class` before you access it.

Let's start creating our first `class` .

We only need a method called `constructor` (remember to add only one constructor, a `SyntaxError` will be thrown if the `class` contains more than one constructor method).



```
class Person {
  constructor(name,age){
    this.name = name;
    this.age = age;
  }
  greet(){
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old` );
  } // no commas in between methods
  farewell(){
    console.log("goodbye friend");
  }
}

const alberto = new Person("Alberto",26);

alberto.greet();
// Hello, my name is Alberto and I am 26 years old
alberto.farewell();
// goodbye friend
```



As you can see everything works just like before. As we mentioned at the beginning, Classes are just a syntactic sugar, a nicer way of doing inheritance.

## Static methods #

Right now the two new methods that we created- `greet()` and `farewell()` - can be accessed by every new instance of `Person`, but what if we want a method that can only be accessed by the class itself, similarly to `Array.of()` for arrays?

The following example will throw an error:

```
class Person {
  constructor(name,age){
    this.name = name;
    this.age = age;
  }
  static info(){
    console.log("I am a Person class, nice to meet you");
  }
}

const alberto = new Person("Alberto",26);

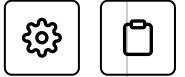
alberto.info();
// TypeError: alberto.info is not a function
```



The following will now work:



```
class Person {  
  constructor(name,age){  
    this.name = name;  
    this.age = age;  
  }  
  static info(){  
    console.log("I am a Person class, nice to meet you");  
  }  
}  
const alberto = new Person("Alberto",26);  
  
Person.info();  
// I am a Person class, nice to meet you
```



## set and get #

We can use setter and getter methods to set and get values inside our class .



```
class Person {
  constructor(name,surname) {
    this.name = name;
    this.surname = surname;
    this.nickname = "";
  }
  set nicknames(value){
    this.nickname = value;
    console.log(this.nickname);
  }
  get nicknames(){
    console.log(`Your nickname is ${this.nickname}`);
  }
}

const alberto = new Person("Alberto","Montalesi");

// first we call the setter
alberto.nicknames = "Albi";
// "Albi"

// then we call the getter
alberto.nicknames;
// "Your nickname is Albi"
```



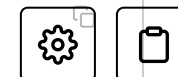
## Extending our class #

What if we want to have a new class that inherits from our previous one? We use `extends` keyword for this purpose. Take a look at the following example:

```
// our initial class
class Person {
  constructor(name,age){
    this.name = name;
    this.age = age;
  }
  greet(){
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old` );
  }
}

// our new class
class Adult extends Person {
  constructor(name,age,work){
    this.name = name;
    this.age = age;
    this.work = work;
  }
}

const alberto = new Adult("Alberto",26,"software developer");
```



We created a new class `Adult` that inherits from `Person`, but if you try to run this code, you'll get an error:

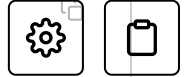
```
ReferenceError: must call super constructor before using |this| in Adult class constructor
```



The error message tells us to call `super()` before using `this` in our new class. What it means is that we basically have to create a new `Person` before we create a new `Adult` and the `super()` constructor will do exactly that.



```
class Adult extends Person {  
  constructor(name,age,work){  
    super(name,age);  
    this.work = work;  
  }  
}
```



Why did we set `super(name,age)` ? Because our `Adult` class inherits name and age from the `Person`, therefore we don't need to redeclare them. Super will simply create a new `Person` for us.

If we now run the code again we will get this:

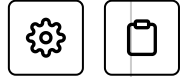


```
// our initial class
class Person {
  constructor(name,age){
    this.name = name;
    this.age = age;
  }
  greet(){
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old` );
  }
}

// our new class
class Adult extends Person {
  constructor(name,age,work){
    super(name,age);
    this.work = work;
  }
}

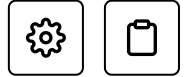
const alberto = new Adult("Alberto",26,"software developer");

console.log(alberto.age);
// 26
console.log(alberto.work);
// "software developer"
alberto.greet();
// Hello, my name is Alberto and I am 26 years old
```



As you can see, our `Adult` inherited all the properties and methods from the `Person` class.

# Extending Arrays #



We want to create something like this- something similar to an array where the first value is a property to define our classroom and the rest are our students and their marks.

```
// we create a new Classroom
const myClass = new Classroom('1A', [
  {name: "Tim", mark: 6},
  {name: "Tom", mark: 3},
  {name: "Jim", mark: 8},
  {name: "Jon", mark: 10},]
);
```



What we can do is create a new `class` that extends the array.



```
class Classroom extends Array {
  // we use rest operator to grab all the students
  constructor(name, ...students){
    // we use spread to place all the students in the array individually otherwise we would push an array into an
    super(...students);
    this.name = name;
    // we create a new method to add students
  }
  add(student){
    this.push(student);
  }
}



const myClass = new Classroom('1A',
  {name: "Tim", mark: 6},
  {name: "Tom", mark: 3},
  {name: "Jim", mark: 8},
  {name: "Jon", mark: 10},
);

// now we can call
myClass.add({name: "Timmy", mark:7});
myClass[4];
// Object { name: "Timmy", mark: 7 }

// we can also loop over with for of
for(const student of myClass) {
  console.log(student);
}
// Object { name: "Tim", mark: 6 }
// Object { name: "Tom", mark: 3 }
// Object { name: "Jim", mark: 8 }
// Object { name: "Jon", mark: 10 }
// Object { name: "Timmy", mark: 7 }
```



Can you remember all of that for the quiz to follow?

Interviewing soon? We've partnered with Hired so that companies apply to you, instead of the other way around.  

[utm\\_source=educative&utm\\_medium=lesson&utm\\_location=CA&utm\\_campaign=December\\_2020&utm\\_content=Hired](https://www.educative.io/courses/complete-guide-to-modern-javascript/x1ZYZjxLPx9?utm_source=educative&utm_medium=lesson&utm_location=CA&utm_campaign=December_2020&utm_content=Hired)




← Back

Quiz

Next →

Quiz

 Completed



Report an Issue



Ask a Question

([https://discuss.educative.io/tag/classes\\_\\_es6\\_\\_the-complete-guide-to-modern-javascript](https://discuss.educative.io/tag/classes__es6__the-complete-guide-to-modern-javascript))