The ‘**new**’ operator

```
var first = new Student('John', 26);
```

The Four Rules.

The simplest way to understand the `new` operator is to understand what it does. When you use `new`, four things happen:

1. It creates a new, empty object.
2. It binds `this` to our newly created object.
3. It adds a property onto our newly created object called “__proto__” which points to the constructor function’s prototype object.
4. It adds a `return this` to the end of the function, so that the object that is created is returned from the function.

Create a constructor function called `Student`. This function will take two parameters, `name`, and `age`. It will then set these properties on the value of `this`.

```
function Student(name, age) {
  this.name = name;
  this.age = age;
}
```

Now let's invoke our constructor function with the `new` operator. We're going to pass in two arguments: `'John'`, and `26`.

```
var first = new Student('John', 26);
```

So, what happens when we run the above code?

1. A new object is created — the `first` object.
2. `this` is bound to our `first` object.

   So any references to `this` will point to `first` .
3. Our __proto__ is added.

   So `first.__proto__` will now point to `Student.prototype` .
4. After everything is done, our brand new `first` object is returned to our new `first` variable.

Let's run a few simple `console.log` statements to test if it worked:

```
console.log(first.name);
// John


console.log(first.age);
// 26
```