

A close-up photograph of a person's hands holding a small, rectangular gift wrapped in light brown kraft paper. A vibrant red ribbon is tied in a bow around the middle of the box. The hands are visible at the bottom, one on each side, gripping the ribbon. The background is plain white.

new

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor. The terminal window is titled '1: bash' and contains the command 'node new\_1.js' followed by its output: an object with properties name, city, year, and getAge(). The code editor window is titled 'new\_1.js' and shows the corresponding JavaScript code.

new\_1.js — javascriptmasterclass

JS new\_1.js x ⌂ ⌓ ⌚ TERMINAL ... 1: bash + ⌒

```
1 const person = {  
2   name: "Linus Torvald",  
3   city: "Helsinki",  
4   year: 1969,  
5   getAge() {  
6     return ((new Date()).getFullYear() - this.year);  
7   }  
8 };  
9 console.log(person);  
10 console.log(person.getAge());  
11
```

```
rodrigobranas:javascriptmasterclass $ node new_1.js  
{ name: 'Linus Torvald',  
  city: 'Helsinki',  
  year: 1969,  
  getAge: [Function: getAge] }  
49  
rodrigobranas:javascriptmasterclass $
```

The screenshot shows a terminal window titled "new\_2.js — javascriptmasterclass" with the following content:

```
JS new_2.js x TERMINAL ... 1: bash
rodrigobranas:javascriptmasterclass $ node new_2.js
{ name: 'Linus Torvald',
  city: 'Helsinki',
  year: 1969,
  getAge: [Function: getAge] }
49
{ name: 'Bill Gates',
  city: 'Seattle',
  year: 1955,
  getAge: [Function: getAge] }
63
rodrigobranas:javascriptmasterclass $
```

The terminal output shows two objects being logged to the console. Each object has properties: name, city, year, and getAge. The getAge function returns the current age based on the current date.

```
1  const person1 = {
2      name: "Linus Torvald",
3      city: "Helsinki",
4      year: 1969,
5      getAge() {
6          return ((new Date()).getFullYear() - this.year);
7      }
8  };
9  const person2 = {
10     name: "Bill Gates",
11     city: "Seattle",
12     year: 1955,
13     getAge() {
14         return ((new Date()).getFullYear() - this.year);
15     }
16 };
17 console.log(person1);
18 console.log(person1.getAge());
19 console.log(person2);
20 console.log(person2.getAge());
21
```



Como fazer para criar um objeto **a partir**  
**da mesma estrutura?**

A função fábrica, que é um tipo de padrão, retorna um novo objeto após ser invocada diretamente

new\_3.js — javascriptmasterclass

JS new\_3.js x ⌂ ⌓ ⌚ ⌚ TERMINAL ... 1: bash + ⌚

```
1 const createPerson = function(name, city, year) {
2   return {
3     name,
4     city,
5     year,
6     getAge() {
7       return ((new Date()).getFullYear() - this.year);
8     }
9   }
10 };
11 const person1 = createPerson("Linus Torvald", "Helsinki", 1969);
12 const person2 = createPerson("Bill Gates", "Seattle", 1955);
13 console.log(person1);
14 console.log(person1.getAge());
15 console.log(person2);
16 console.log(person2.getAge());
```

rodrigobranas:javascriptmasterclass \$ node new/new\_3.js

{ name: 'Linus Torvald',  
 city: 'Helsinki',  
 year: 1969,  
 getAge: [Function: getAge] }

49 { name: 'Bill Gates',  
 city: 'Seattle',  
 year: 1955,  
 getAge: [Function: getAge] }

63 rodrigobranas:javascriptmasterclass \$



O que fazer para **eliminar** a duplicação e  
**reusar** propriedades entre os objetos?

The screenshot shows a Mac OS X desktop environment with a terminal window open. The terminal window has a title bar "new\_4.js — javascriptmasterclass". The main pane of the terminal displays the output of running the JavaScript file "new\_4.js" with the command "node new\_4.js". The output shows two objects created by the script, each with a "name", "city", and "year" property, and a "getAge" method. The first object represents Linus Torvald from Helsinki in 1969, and the second represents Bill Gates from Seattle in 1955. Both objects have their "getAge" methods printed as "[Function: getAge]". The final line of output is "true", indicating a comparison result.

```
new_4.js — javascriptmasterclass
JS new_4.js x TERMINAL ... 1: bash +
1 const personPrototype = {
2     getAge() {
3         return ((new Date()).getFullYear() - this.year);
4     }
5 };
6 const createPerson = function(name, city, year) {
7     const person = {
8         name,
9         city,
10        year
11    };
12     Object.setPrototypeOf(person, personPrototype);
13     return person;
14 };
15 const person1 = createPerson("Linus Torvald", "Helsinki", 19
16 const person2 = createPerson("Bill Gates", "Seattle", 1955);
17 console.log(person1);
18 console.log(person1.__proto__);
19 console.log(person1.getAge());
20 console.log(person2);
21 console.log(person2.__proto__);
22 console.log(person2.getAge());
23 console.log(person1.__proto__ === person2.__proto__);
24
```

```
rodrigobranas:javascriptmasterclass $ node new_4.js
{ name: 'Linus Torvald', city: 'Helsinki', year: 1969 }
{ getAge: [Function: getAge] }
49
{ name: 'Bill Gates', city: 'Seattle', year: 1955 }
{ getAge: [Function: getAge] }
63
true
rodrigobranas:javascriptmasterclass $
```

A função construtora retorna um novo  
objeto ao ser invocada por meio do  
operador new

The screenshot shows a Mac OS X desktop environment. In the top-left corner, there's a small window icon. The main window title bar reads "new\_5.js — javascriptmasterclass". The window contains two tabs: "JS new\_5.js" and "x". On the right side of the window are standard OS X controls for zooming and closing. Below the window is a toolbar with icons for search, refresh, and more. To the right of the window is a terminal window titled "1: bash". The terminal output shows the execution of a JavaScript file named "new\_5.js". The file defines a "Person" constructor function that logs its properties and a "getAge" method. It then creates two instances of "Person" for Linus Torvald and Bill Gates, logs their details, and checks if their prototypes are equal.

```
new_5.js — javascriptmasterclass
JS new_5.js x ⌂ ⌓ ⌚ ⌚ ⌚ TERMINAL ⌚ 1: bash + ⌚

1 const Person = function(name, city, year) {
2   this.name = name,
3   this.city = city,
4   this.year = year,
5   this.getAge = function() {
6     return ((new Date()).getFullYear() - this.year);
7   }
8 };
9 const person1 = new Person("Linus Torvald", "Helsinki", 1969)
10 const person2 = new Person("Bill Gates", "Seattle", 1955);
11 console.log(person1);
12 console.log(person1.__proto__);
13 console.log(person1.getAge());
14 console.log(person2);
15 console.log(person2.__proto__);
16 console.log(person2.getAge());
17 console.log(person1.__proto__ === person2.__proto__);
18

rodrigobranas:javascriptmasterclass $ node new/new_5.js
Person {
  name: 'Linus Torvald',
  city: 'Helsinki',
  year: 1969,
  getAge: [Function] }
Person {}
49
Person {
  name: 'Bill Gates',
  city: 'Seattle',
  year: 1955,
  getAge: [Function] }
Person {}
63
true
rodrigobranas:javascriptmasterclass $
```

Toda função tem uma propriedade  
chamada `prototype`, que é vinculada ao  
`__proto__` do objeto criado pelo  
operador `new`

The screenshot shows a Mac OS X desktop environment with a terminal window open. The terminal window has a title bar "new\_6.js — javascriptmasterclass". The main pane of the terminal displays the output of a Node.js script named "new\_6.js". The script defines a "Person" constructor function and logs several objects to the console. The output shows two instances of the "Person" object with their properties and a reference to their prototype.

```
new_6.js — javascriptmasterclass
JS new_6.js x ⌂ ⌓ ⌚ TERMINAL ... 1: bash + ⌒
1 const Person = function(name, city, year) {
2   this.name = name,
3   this.city = city,
4   this.year = year
5 };
6 Person.prototype.getAge = function() {
7   return ((new Date()).getFullYear() - this.year);
8 };
9 const person1 = new Person("Linus Torvald", "Helsinki", 1969)
10 const person2 = new Person("Bill Gates", "Seattle", 1955);
11 console.log(person1);
12 console.log(person1.__proto__);
13 console.log(person1.getAge());
14 console.log(person2);
15 console.log(person2.__proto__);
16 console.log(person2.getAge());
17 console.log(person1.__proto__ === person2.__proto__);
18
```

```
rodrigobranas:javascriptmasterclass $ node new_6.js
Person { name: 'Linus Torvald', city: 'Helsinki', year: 1969 }
Person { getAge: [Function] }
49
Person { name: 'Bill Gates', city: 'Seattle', year: 1955 }
Person { getAge: [Function] }
63
true
rodrigobranas:javascriptmasterclass $
```

The screenshot shows a terminal window titled "new\_7.js — javascriptmasterclass" with the following content:

```
JS new_7.js x TERMINAL ... 1: bash + = 1

rodrigobranas:javascriptmasterclass $ node new_7.js
Person { name: 'Linus Torvald', city: 'Helsinki', year: 1969 }
Person { getAge: [Function] }
49
Person { name: 'Bill Gates', city: 'Seattle', year: 1955 }
Person { getAge: [Function] }
63
true
rodrigobranas:javascriptmasterclass $
```

The terminal output shows the execution of the `new_7.js` file, which defines a constructor function `_new` and a Person class. It creates two Person objects, logs their properties, and checks if their prototypes are equal.

```
1  const _new = function(fn, ...params) {
2      const obj = {};
3      Object.setPrototypeOf(obj, fn.prototype);
4      fn.apply(obj, params);
5      return obj;
6  };
7  const Person = function(name, city, year) {
8      this.name = name,
9      this.city = city,
10     this.year = year
11 };
12 Person.prototype.getAge = function() {
13     return ((new Date()).getFullYear() - this.year);
14 };
15 const person1 = _new(Person, "Linus Torvald", "Helsinki", 19
16 const person2 = _new(Person, "Bill Gates", "Seattle", 1955);
17 console.log(person1);
18 console.log(person1.__proto__);
19 console.log(person1.getAge());
20 console.log(person2);
21 console.log(person2.__proto__);
22 console.log(person2.getAge());
23 console.log(person1.__proto__ === person2.__proto__);
24
```



CAUTION

Não esqueça de utilizar o operador **new**  
 quando utilizar funções construtoras