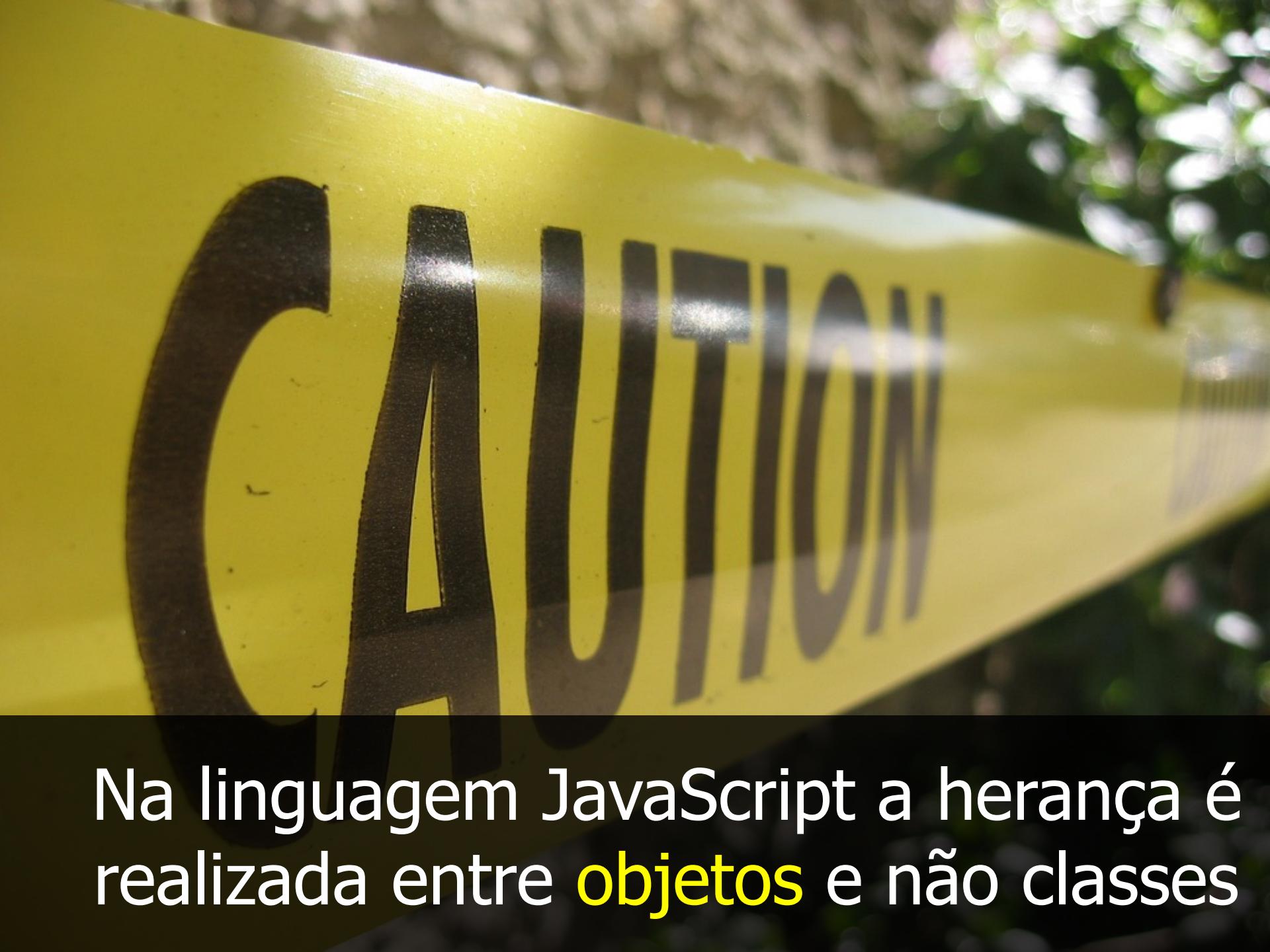


Herança



O principal objetivo da herança é permitir o reuso de código por meio do compartilhamento de propriedades entre objetos, evitando a duplicação

A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is slightly curved, and the background shows some green foliage.

CAUTION

Na linguagem JavaScript a herança é realizada entre **objetos** e não classes

A screenshot of a Mac OS X desktop environment showing a terminal window. The window title is "inheritance_1.js — javascriptmasterclass". The main pane contains the following JavaScript code:

```
JS inheritance_1.js x
1 const scheme = {
2     name: "Scheme",
3     year: 1975,
4     paradigm: "Functional"
5 };
6 const javascript = {
7     name: "JavaScript",
8     year: 1995,
9     paradigm: "Functional"
10};
11 console.log(scheme);
12 console.log(javascript);
13
```

The right side of the window shows the terminal output:

```
rodrigobranas:javascriptmasterclass $ node inheritance/inheritance_1.js
{ name: 'Scheme', year: 1975, paradigm: 'Functional' }
{ name: 'JavaScript', year: 1995, paradigm: 'Functional' }
rodrigobranas:javascriptmasterclass $ █
```

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the command `node inheritance/inheritance_2.js` being run, followed by the output:

```
rodrigobranas:javascriptmasterclass $ node inheritance/inheritance_2.js
{ paradigm: 'Functional' }
{ name: 'Scheme', year: 1975, paradigm: 'Functional' }
{ name: 'JavaScript', year: 1995, paradigm: 'Functional' }
rodrigobranas:javascriptmasterclass $
```

The code editor window on the left contains the file `inheritance_2.js`:

```
JS inheritance_2.js x
inheritance_2.js — javascriptmasterclass

1 const functionalLanguage = {
2   paradigm: "Functional"
3 };
4 const scheme = {
5   name: "Scheme",
6   year: 1975,
7   paradigm: "Functional"
8 };
9 const javascript = {
10   name: "JavaScript",
11   year: 1995,
12   paradigm: "Functional"
13 };
14 console.log(functionalLanguage);
15 console.log(scheme);
16 console.log(javascript);
17
```

A propriedade proto é uma referência para o protótipo do objeto

A screenshot of a Mac OS X desktop environment. On the left, a code editor window titled "inheritance_3.js — javascriptmasterclass" displays the following JavaScript code:

```
JS inheritance_3.js x
1 const functionalLanguage = {
2     paradigm: "Functional"
3 };
4 const scheme = {
5     name: "Scheme",
6     year: 1975,
7     __proto__: functionalLanguage
8 };
9 const javascript = {
10    name: "JavaScript",
11    year: 1995,
12    __proto__: functionalLanguage
13 };
14 console.log(functionalLanguage);
15 console.log(scheme);
16 console.log(javascript);
17
```

The code defines three objects: "functionalLanguage", "scheme", and "javascript". The "functionalLanguage" object has a "paradigm" property set to "Functional". The "scheme" and "javascript" objects have "name" and "year" properties, and their "__proto__" property is set to "functionalLanguage", demonstrating prototypal inheritance.

On the right, a terminal window titled "TERMINAL" shows the output of running the script with node:

```
rodrigobranas:javascriptmasterclass $ node inheritance/inheritance_3.js
{ paradigm: 'Functional' }
{ name: 'Scheme', year: 1975 }
{ name: 'JavaScript', year: 1995 }
rodrigobranas:javascriptmasterclass $
```



Porque a propriedade **paradigm** não foi
exibida dentro do objeto?

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the command `node inheritance/inheritance_4.js` being run, followed by the output:

```
rodrigobranas:javascriptmasterclass $ node inheritance/inheritance_4.js
{ paradigm: 'Functional' }
Functional
Functional
rodrigobranas:javascriptmasterclass $
```

The code editor on the left contains the file `inheritance_4.js` with the following content:

```
JS inheritance_4.js x
inheritance_4.js — javascriptmasterclass
1 const functionalLanguage = {
2   paradigm: "Functional"
3 };
4 const scheme = {
5   name: "Scheme",
6   year: 1975,
7   __proto__: functionalLanguage
8 };
9 const javascript = {
10   name: "JavaScript",
11   year: 1995,
12   __proto__: functionalLanguage
13 };
14 console.log(functionalLanguage);
15 console.log(scheme.paradigm);
16 console.log(javascript.paradigm);
17
```

O método **hasOwnProperty** pode ser
utilizado para determinar se uma
propriedade pertence ao objeto

A screenshot of a Mac OS X desktop environment showing a terminal window. The window title is "inheritance_5.js — javascriptmasterclass". The main pane contains the following JavaScript code:

```
JS inheritance_5.js x
1 const functionalLanguage = {
2     paradigm: "Functional"
3 };
4 const scheme = {
5     name: "Scheme",
6     year: 1975,
7     __proto__: functionalLanguage
8 };
9 const javascript = {
10    name: "JavaScript",
11    year: 1995,
12    __proto__: functionalLanguage
13 };
14 for (let key in scheme) {
15     console.log(key, scheme.hasOwnProperty(key));
16 }
17
```

The right side of the window shows the terminal output:

```
TERMINAL ... 1: bash + = 1
rodrigobranas:javascriptmasterclass $ node i
de inheritance/inheritance_5.js
name true
year true
paradigm false
rodrigobranas:javascriptmasterclass $ █
```

Os métodos `Object.setPrototypeOf` e
`Object.getPrototypeOf` permitem a
interação com o protótipo do objeto

inheritance_6.js — javascriptmasterclass

JS inheritance_6.js x

TERMINAL ... 1: bash + =

```
1 const functionalLanguage = {
2     paradigm: "Functional"
3 };
4 const scheme = {
5     name: "Scheme",
6     year: 1975
7 };
8 Object.setPrototypeOf(scheme, functionalLanguage);
9 const javascript = {
10     name: "JavaScript",
11     year: 1995
12 };
13 Object.setPrototypeOf(javascript, functionalLanguage);
14 for (let key in scheme) {
15     console.log(key, scheme.hasOwnProperty(key));
16 }
17
```

rodrigobranas:javascriptmasterclass \$ node inheritance/inheritance_6.js
name true
year true
paradigm false
rodrigobranas:javascriptmasterclass \$

Com o método `Object.create` é possível
criar um objeto passando o seu
protótipo por parâmetro

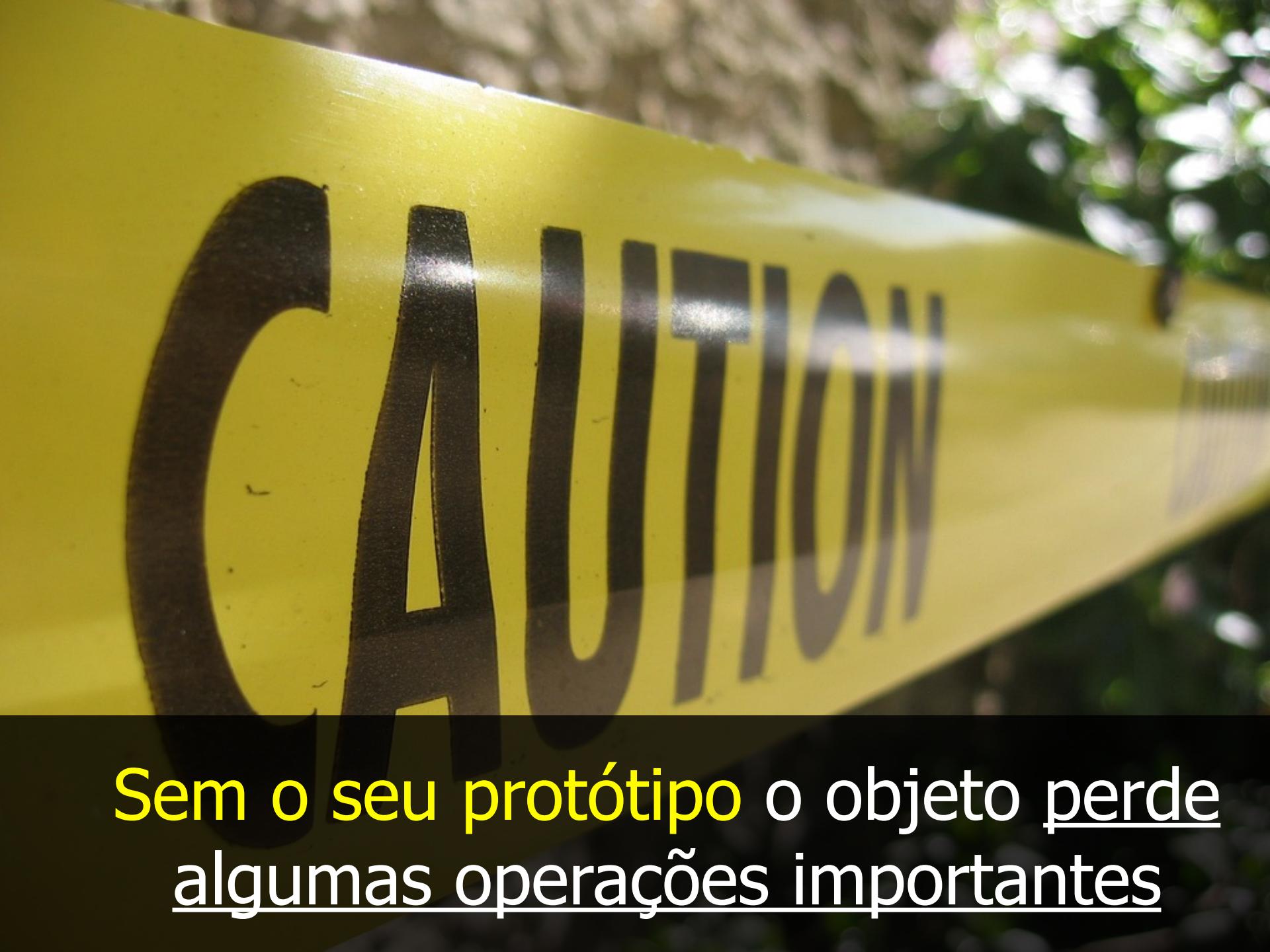
A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the command `node inheritance/inheritance_7.js` being run, followed by the output:

```
rodrigobranas:javascriptmasterclass $ node inheritance/inheritance_7.js
name true
year true
paradigm false
rodrigobranas:javascriptmasterclass $
```

The code editor on the left contains the following JavaScript code:

```
1 const functionalLanguage = {
2   paradigm: "Functional"
3 };
4 const scheme = Object.create(functionalLanguage);
5 scheme.name = "Scheme";
6 scheme.year = 1975;
7 const javascript = Object.create(functionalLanguage);
8 javascript.name = "JavaScript";
9 javascript.year = 1995;
10 for (let key in scheme) {
11   console.log(key, scheme.hasOwnProperty(key));
12 }
13
```

A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is slightly curved, and the background shows some green foliage.

CAUTION

Sem o seu protótipo o objeto perde
algumas operações importantes

The screenshot shows a macOS desktop environment with a terminal window open. The title bar of the terminal window reads "inheritance_8.js — javascriptmasterclass". The main pane of the terminal displays the output of a Node.js command:

```
rodrigobranas:javascriptmasterclass $ node inheritance/inheritance_8.js
/Users/rodrigobranas/development/workspace/javascriptmasterclass/inheritance/inheritance_8.js:10
  console.log(key, scheme.hasOwnProperty(key));
^

TypeError: scheme.hasOwnProperty is not a function
    at Object.<anonymous> (/Users/rodrigobranas/development/workspace/javascriptmasterclass/inheritance/inheritance_8.js:10:29)
    at Module._compile (module.js:643:30)
    at Object.Module._extensions..js (module.js:654:10)
    at Module.load (module.js:556:32)
    at tryModuleLoad (module.js:499:12)
    at Function.Module._load (module.js:491:3)
    at Function.Module.runMain (module.js:684:10)
    at startup (bootstrap_node.js:187:16)
    at bootstrap_node.js:608:3
rodrigobranas:javascriptmasterclass $
```

The code being run is from a file named "inheritance_8.js". It defines three objects: "functionalLanguage", "scheme", and "javascript". The "functionalLanguage" object has a "paradigm" property set to "Functional". The "scheme" object is created from "functionalLanguage" and has a "name" property set to "Scheme" and a "year" property set to 1975. The "javascript" object is also created from "functionalLanguage" and has a "name" property set to "JavaScript" and a "year" property set to 1995. A loop iterates over the properties of "scheme", logging each key and the result of calling "hasOwnProperty" on that key. The error message indicates that "hasOwnProperty" is not a function, which is likely due to a typo in the code where "hasOwnProperty" was misspelled as "hasOwnProperty".

Caso a mesma propriedade exista no objeto e no seu protótipo, a propriedade do próprio objeto é retornada, fazendo sombra à propriedade do protótipo

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the command `node inheritance/inheritance_9.js` being run, followed by the output:

```
rodrigobranas:javascriptmasterclass $ node inheritance/inheritance_9.js
name JavaScript
year 1995
paradigm OO
rodrigobranas:javascriptmasterclass $
```

The code editor window on the left contains the file `inheritance_9.js`:

```
JS inheritance_9.js × inheritance_9.js — javascriptmasterclass
1 const functionalLanguage = Object.create({});
2 functionalLanguage.paradigm = "Functional";
3 const scheme = Object.create(functionalLanguage);
4 scheme.name = "Scheme";
5 scheme.year = 1975;
6 const javascript = Object.create(functionalLanguage);
7 javascript.name = "JavaScript";
8 javascript.year = 1995;
9 javascript.paradigm = "OO";
10 for (let key in javascript) {
11     console.log(key, javascript[key]);
12 }
13
```

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor. The terminal window is titled 'inheritance_10.js — javascriptmasterclass' and contains the command 'node inheritance/inheritance_10.js'. The output shows three lines of text: 'Functional', 'Functional', and '00'. The code editor window is titled 'JS inheritance_10.js' and contains the following JavaScript code:

```
1 const functionalLanguage = Object.create({});  
2 functionalLanguage.paradigm = "Functional";  
3 const scheme = Object.create(functionalLanguage);  
4 scheme.name = "Scheme";  
5 scheme.year = 1975;  
6 const javascript = Object.create(functionalLanguage);  
7 javascript.name = "JavaScript";  
8 javascript.year = 1995;  
9 javascript.paradigm = "OO";  
10 console.log(javascript.paradigm);  
11 console.log(javascript.__proto__.paradigm);  
12 console.log(Object.getPrototypeOf(javascript).paradigm);  
13
```