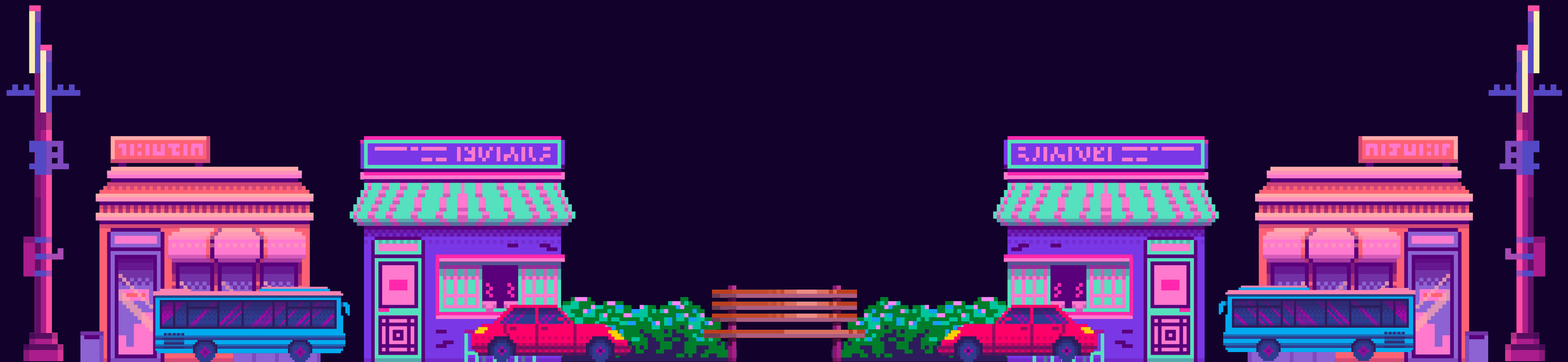

HELLO
WORLD

PROTOTYPE

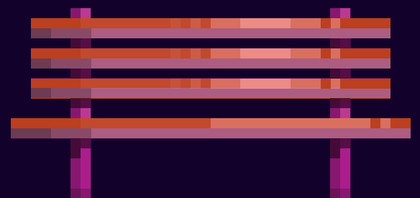
COMO QUASE TUDO NO JAVASCRIPT É UM OBJETO

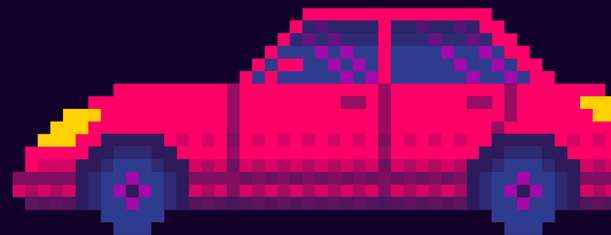


CLASSES NO JAVASCRIPT

Você conhece a expressão que algo atua como açúcar sintático? Em programação, é quando há o intuito de facilitar o entendimento, ou seja, sendo mais “doce” ou melhor diluído.

É o caso das Classes em JS, que são introduzidas apenas no ES5 (ECMAScript 5 (ES5) é a quinta versão do padrão ECMAScript, um padrão de linguagem de script para a Web. Foi publicado em dezembro de 2009 e adicionou novas funcionalidades ao JavaScript, como o uso de classes, a adição de métodos para trabalhar com arrays e objetos e o suporte ao uso de strict mode.) na intenção de ser uma versão simplificada de construção de objetos, em comparação a sintaxe verbosa que possui as funções construtoras e a herança nessa linguagem, que é baseada em protótipos. Mas, antes de conversarmos sobre protótipos, vamos entender como funcionam as classes no JS?





ENTENDENDO AS CLASSES

01

No console do navegador, criei uma classe que foi nomeada como Pessoa.

```
> class Pessoa {}
```

02

Para descobrir qual é o protótipo dessa Classe/construtor, acesso com a propriedade .prototype.

```
> Pessoa.prototype
< ▼{constructor: ƒ}
    ▶constructor: class Pessoa
    ▼[[Prototype]]: Object
      ▶constructor: ƒ Object()
      ▶hasOwnProperty: ƒ hasOwnProperty()
      ▶isPrototypeOf: ƒ isPrototypeOf()
      ▶propertyIsEnumerable: ƒ propertyIsEnumerable()
      ▶toLocaleString: ƒ toLocaleString()
      ▶toString: ƒ toString()
      ▶valueOf: ƒ valueOf()
      ▶__defineGetter__: ƒ __defineGetter__()
      ▶__defineSetter__: ƒ __defineSetter__()
      ▶__lookupGetter__: ƒ __lookupGetter__()
      ▶__lookupSetter__: ƒ __lookupSetter__()
        __proto__: (...)
      ▶get __proto__: ƒ __proto__()
      ▶set __proto__: ƒ __proto__()
>
```

Na indicação da seta, há a informação que o [[prototype]] de uma Classe é no final das contas um Object.

TUDO (OU QUASE TUDO) EM JAVASCRIPT É UM OBJETO!

A prova disso é que se você colocar no console do navegador Number.prototype, String.prototype ou Array.prototype por exemplo,
você terá a informação, que assim como a Class, o modelo de construção de números, frases e listas, também é um Objeto.
Isso ocorre para quase todos os tipos e é por isso que temos a afirmação que no JavaScript, tudo ou quase tudo em sua raiz de construção é um objeto, com exceção apenas dos tipos primitivos null e undefined.

SAIBA MAIS

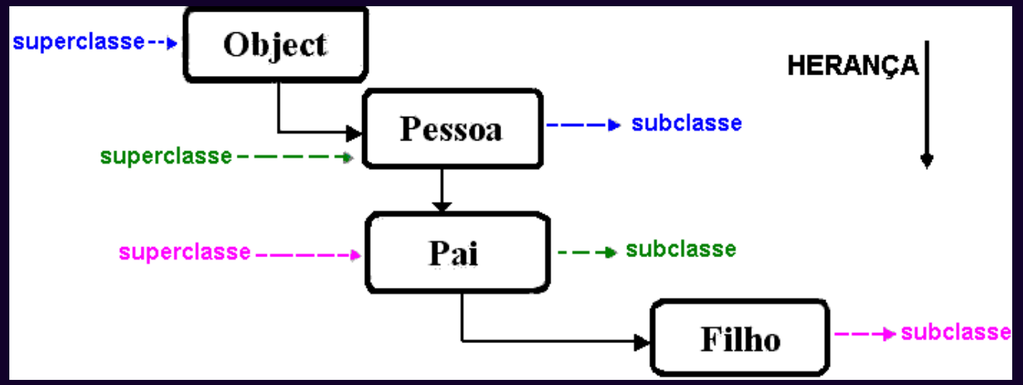
HERANÇA NO JAVASCRIPT

A herança em JavaScript é baseada em protótipos e é conhecida como Herança Prototípica. Isso significa que todo objeto em JavaScript possui uma propriedade chamada prototype que possibilita a herança.

Prototype ou protótipo, atua como um modelo do qual herda métodos e propriedades. Todos os objetos em JavaScript herdam de pelo menos um outro objeto. Em Pessoa, o seu protótipo é o próprio Object que é encontrado em sua raiz, e a classe que possui como prototype um Object, irá herdar métodos dessa classe nativa(Object), como o **valueOf** e **hasOwnProperty**, por exemplo.

CADEIA DE PROTÓTIPOS

Entretanto, podemos aumentar os níveis dessa dinâmica! É possível que um objeto tenha um objeto de protótipo e este por sua vez, também tenha um objeto de protótipo, e assim por diante. Esse fenômeno é chamado de cadeia de protótipos.



CADEIA DE PROTÓTIPOS

01

Crie uma instância da Classe Pessoa, que armazeno na variável const pessoa100.

```
> const pessoa100 = new Pessoa()
```

02

Acesso a propriedade de protótipo da instância com __proto__, mas você também pode fazer essa verificação com Object.getPrototypeOf(pessoa100).

```
> pessoa100.__proto__
◂ ▾ {constructor: ƒ} ⓘ
    ▸ constructor: class Pessoa
    ▸ [[Prototype]]: Object
>
```

03

Entrando no bloco de constructor: class Pessoa e abrindo o bloco prototype, temos:

```
> pessoa100.__proto__
◂ ▾ {constructor: ƒ} ⓘ
    ▾ constructor: class Pessoa
        length: 0
        name: "Pessoa"
      ▾ prototype:
        ▸ constructor: class Pessoa
        ▸ [[Prototype]]: Object
          arguments: (…)
          caller: (…)
          [[FunctionLocation]]: VM122:1
        ▸ [[Prototype]]: ƒ ()
    ▸ [[Prototype]]: Object
```

CONSTRUINDO UM OBJETO USANDO CLASS

As Classes são, de fato, "funções especiais", e, assim como você pode definir "function expressions" e "function declarations", a sintaxe de uma classe possui dois componentes: "class expressions" e "class declarations".

SAIBA MAIS

DECLARANDO CLASSES

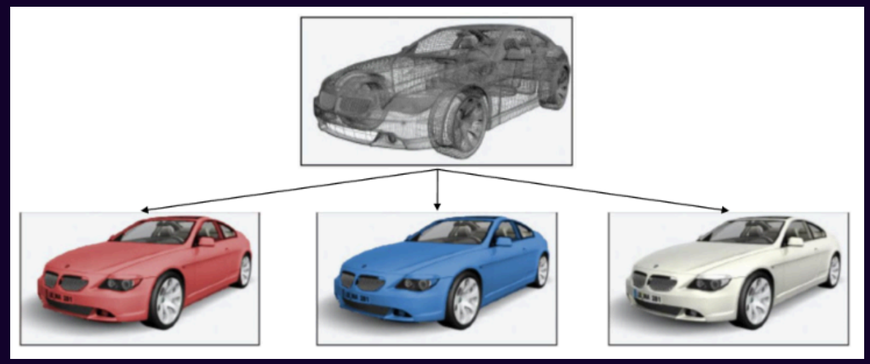
Uma class em JavaScript define a estrutura de um objeto, como propriedades e métodos, e permite a criação de instâncias desse objeto. Além disso, podemos incluir um construtor (constructor), que é uma função especial chamada automaticamente quando um novo objeto é instanciado a partir da classe.

```
class Carro {
  constructor(marca, modelo, ano) {
    this.marca = marca;
    this.modelo = modelo;
    this.ano = ano;
  }

  exibirInfo() {
    console.log(`Carro: ${this.marca} ${this.modelo}, Ano: ${this.ano}`);
  }

  acelerar() {
    console.log(`${this.marca} ${this.modelo} está acelerando!`);
  }

  frear() {
    console.log(`${this.marca} ${this.modelo} está freando!`);
  }
}
```

Uma diferença importante entre declarações de funções das declarações de classes, é que declararações de funções são hoisted e declarações de classes não são. Primeiramente deve declarar sua classe para só então acessá-la, pois do contrário o código a seguir irá lançar uma exceção: ReferenceError

O QUE É UMA INSTÂNCIA?

Uma instância é um objeto criado a partir de uma classe. Quando você define uma classe, está criando um "molde" que descreve como os objetos desse tipo devem ser construídos e o que podem fazer (suas propriedades e métodos). A instância é uma ocorrência concreta desse molde, com valores específicos.



INSTANCIANDO A CLASSE CARRO

01

```
// Criando uma instância da classe Carro
const meuCarro = new Carro('Toyota', 'Corolla', 2021);

// Outra instância da classe Carro
const outroCarro = new Carro('Honda', 'Civic', 2020);
```

02

```
meuCarro.exibirInfo();
// Output: Carro: Toyota Corolla, Ano: 2021

outroCarro.exibirInfo();
// Output: Carro: Honda Civic, Ano: 2020
```

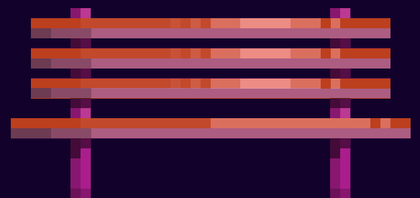
CLASSES E CONSTRUTORES NO ES6

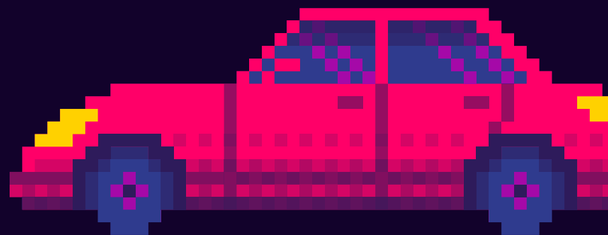
No ES6, a introdução da sintaxe de classes trouxe uma forma mais clara e simples de definir e trabalhar com objetos e herança em JavaScript. A sintaxe de classes é uma forma mais elegante e moderna de criar construtores e métodos em objetos.
Uma classe é definida usando a palavra-chave class, seguida pelo nome da classe. O código dentro da classe pode incluir o construtor e métodos.

```
class Pessoa {
  // O construtor é um método especial para criar e inicializar objetos criados a partir da classe
  constructor(nome, idade) {
    this.nome = nome; // Atributo nome
    this.idade = idade; // Atributo idade
  }

  // Método da classe
  apresentar() {
    console.log(`Olá, meu nome é ${this.nome} e eu tenho ${this.idade} anos.`);
  }
}
```

O método constructor é um método especial para criar e inicializar objetos criados a partir da classe. Ele é chamado automaticamente quando você cria uma nova instância da classe.

```
const pessoa1 = new Pessoa('João', 30);
pessoa1.apresentar(); // Saída: Olá, meu nome é João e eu tenho 30 anos.
```






BORA
CODAR