# Tema 10 – Classe e Funções

## Introdução

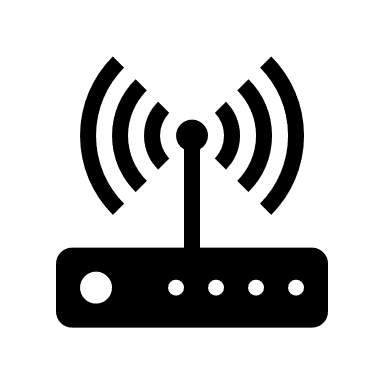
Olá, futuras mentes desenvolvedoras, sejam bem-vindos do curso de Frontend do Instituto da Oportunidade Social. Nessa aula, você aprenderá como utilizar classes e funções.

## Criação do Projeto inicial

Siga os passos para criar o projeto:

1. Abra o VS Code e escolha um diretório de trabalho para o seu projeto.
2. Crie um diretório para seu projeto com o nome representativo, por exemplo, 10\_JS\_Classe\_Func.
3. Crie um arquivo dentro do diretório do projeto com o nome index.html.
4. Insira o seguinte código no seu arquivo index.html.

|  |
| --- |
| <!DOCTYPE html>  <html lang="pt-br">      <head>          <meta charset="UTF-8" />  <link rel="shortcut icon" href="#" />          <title>Classes e Funções</title>      </head>      <body>          <script src="./js/main.js"></script>      </body>  </html> |

1. Esse código mostra a marcação <script> sem nenhum código JavaScript entre a abertura e o fechamento da tag, apenas o atributo **src** com o valor main.js. Isso significa que o código JavaScript está em um arquivo externo. Portanto, temos que criar esse novo arquivo **main.js** dentro do diretório do projeto.
2. Vamos deixar o arquivo main.js vazio e à medida que vamos aprendendo coisas novas iremos completar a implementação do código JavaScript.
3. Abra o arquivo index.html, clique no botão **Go Live** da extensão **Live Server** e abra as Ferramentas de desenvolvimento do navegador web (Atalho **F12**).

## Classes

Classes foram introduzidas no JS no ECMAScript 2015, mais conhecido como ES6, e elas são simplificações da linguagem para utilizarmos herança baseadas nos protótipos. Uma classe JavaScript não é um objeto e sim um template para objetos JavaScript. A sintaxe para classes **não** introduz um novo modelo de herança de orientação a objetos em JavaScript. Classes em JavaScript provêm uma maneira mais simples e clara de criar objetos e lidar com herança. O bloco de instruções (corpo) da declaração de classes é executados em modo estrito.

**Para saber mais**: Você pode saber mais sobre o modo estrito no link:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Strict_mode>

A sintaxe de uma classe em JavaScript é:

|  |
| --- |
| class NomeClasse {   constructor() { ... } } |

Por exemplo, a classe a seguir tem o nome Carro e duas propriedades iniciais: **nome** e **ano**. Algumas literaturas chamam essas propriedades de atributos e/ou estados da classe.

|  |
| --- |
| class Carro {   constructor(nome, ano) {   this.nome = nome;   this.ano = ano;   } } |

### Vamos praticar

Vamos implementar o código no arquivo **main.js** para ver tudo acontecendo.

1. No arquivo, **main.js** digite o seguinte código.

|  |  |
| --- | --- |
| 1 | // Classes  class Carro {      constructor(nome, ano) {          this.nome = nome;          this.ano = ano;      }  }  // Intanciando objetos à classe  let meuCarro1 = new Carro('Ford', 2014);  let meuCarro2 = new Carro('Audi', 2019);  //Imprimindo valores  console.log(meuCarro1.nome);  console.log(meuCarro1.ano);  console.log(meuCarro2.nome);  console.log(meuCarro2.ano); |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 17 |
| 18 |

1. Vamos ver o resultado mostrado no console.

Graphical user interface, text, application, email

Description automatically generated

A classe é criada nas linhas 2 até 7 e usada nas linhas 10 e 11 para instanciar objetos à classe, ou seja, para criar objetos utilizando o template da classe. Nesse exemplo, o template da classe possui duas propriedades **nome** e **ano**. Desse modo, o primeiro valor colocado entre parênteses na instrução new Carro('Ford', 2014) é associado a propriedade nome e o segundo valor é associado à propriedade ano. Se você inverter a ordem de passagem dos dados, verá que os valores trocarão de propriedade.

Ao imprimirmos cada uma das propriedades dos objetos (linhas 14 até 18), podemos visualizar os conteúdos armazenados nas propriedades dos objetos criados nas linhas 10 e 11.

O método **constructor** (linhas 3 até 6) é um tipo especial de método usada para criar e iniciar um objeto criado pela classe. Esse método é chamado automaticamente, quando um novo objeto é criado utilizando o template da classe.

A palavra-chave **this** é usada de dentro da classe para referencia a instância atual. Porém, em JS, essa palavra-chave pode ter outras funções. Como veremos na próxima seção, quando utilizarmos em funções.

Importante: Diferente de funções, que podem ser declaradas antes ou depois de serem invocadas, as classes devem ser declaradas antes de serem utilizadas. Caso contrário, o programa gerará erro de referência da classe.

### **Métodos de uma classe**

Os métodos de uma classe são criados com a mesma sintaxe de um método de objetos. Além do **constructor**, você pode criar diversos protótipos de métodos na classe:

|  |
| --- |
| class NomeClasse {   constructor() { ... }   metodo\_1() { ... }   metodo\_2() { ... }   metodo\_3() { ... } } |

### Vamos praticar

1. Continuando a implementação do projeto, no arquivo, **main.js** digite o seguinte código:

|  |  |
| --- | --- |
| 20 | // Declarar nova classe com métodos  console.clear();  class NovoCarro {      constructor(nome, ano) {          this.nome = nome;          this.ano = ano;      }      idadeCarro(ano) {          return ano - this.ano;      }  }  //Buscando o ano atual automaticamente  let dataHoje = new Date();  let ano = dataHoje.getFullYear();  // console.log(dataHoje);  // Intanciando o objeto à classe  let meuNovoCarro = new NovoCarro('Ford', 2014);  console.log(meuNovoCarro.idadeCarro(ano)); |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 27 |
| 28 |
| 29 |
| 30 |
| 31 |
| 32 |
| 33 |
| 34 |
| 35 |
| 36 |
| 37 |
| 38 |
| 39 |
| 40 |

1. O resultado é mostrado no console do navegador.

Graphical user interface, text, application, email

Description automatically generated

O método **idadeCarro**, que foi declarado nas linhas 27 a 29, calcula a quantidade de anos do carro de acordo com o ano atual. Para executar esse método, basta colocar o nome do objeto, ponto final e o nome do método com o parâmetro (quando houver) passado entre os parênteses.

Outra coisa interessante no código é como buscamos o ano atual. Na linha 33 criamos um objeto com a classe Date do JavaScript. Essa classe já está embutida na linguagem e retorna o data atual como dia, mês, ano, hora, minutos, segundos e outras informações. Se imprimir o valor da variável dataHoje, descomentando o comando console.log(dataHoje), poderá ver a data dia atual impressa no console. Por exemplo, no momento da produção desse conteúdo a data de hoje é:



O ano atual é obtido pelo método **getFullYear** que existe na declaração dessa classe.

**Para saber mais**: Você pode ver mais sobre essa classe na documentação disponibilizada pela Mozilla no link:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Date>

### Mais um exemplo de classe

Para fixar mais o conceito de classe, vamos para mais um exemplo.

1. Continuando a implementação do projeto, no arquivo, **main.js** digite o seguinte código:

|  |  |
| --- | --- |
| 42 | // Mais um exmeplo de classe  console.clear();  class ClassePessoa {      constructor(firstName, lastName, dob) {          this.firstName = firstName;          this.lastName = lastName;          this.dob = new Date(dob);      }      getBirthYear() {          return this.dob.getFullYear();      }      getFullName() {          return `${this.firstName} ${this.lastName}`;      }  }  // Instantiate object  const pessoa1 = new ClassePessoa('John', 'Doe', '1980-04-03');  const pessoa2 = new ClassePessoa('Marry', 'Smith', '1970-06-13');  console.log(pessoa1);  console.log(pessoa2.dob.getFullYear());  console.log(pessoa2.getBirthYear());  console.log(pessoa1.getFullName()); |
| 43 |
| 44 |
| 45 |
| 46 |
| 47 |
| 48 |
| 49 |
| 50 |
| 51 |
| 52 |
| 53 |
| 54 |
| 55 |
| 56 |
| 57 |
| 58 |
| 59 |
| 60 |
| 61 |
| 62 |
| 63 |
| 64 |
| 65 |
| 66 |
| 67 |
| 68 |
| 69 |

1. O resultado é mostrado no console do navegador.

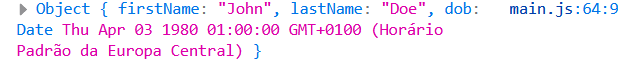
Graphical user interface, text, application, email

Description automatically generated

Nesse exemplo, criamos uma classe (linhas 43 a 58) com os atributos **firstName** (primeiro nome), **lastName** (último nome) e **dob** (Data of Birth, data de nascimento) e com os métodos **getBirthYear** (busca ano do nascimento) e **getFullName** (buscar nome completo [ primeiro e último nome] ).

Nas linhas 61 e 62 criamos dois objetos instanciados à classe e passamos os valores: primeiro nome, último nome e data de nascimento (no padrão AAAA-MM-DD, AAAA – ano com quatro dígitos, MM-mês com dois dígitos e DD-dia com dois dígitos).

Na linha 64, imprimimos o objeto **pessoa1** completo e o resultado é:



Na linha 66, buscamos o ano de nascimento do objeto **pessoa2** através do método **getFullYear**, que existe na classe **Date** (veja que o atributo **dob** é um objeto criado com essa classe [linha 48]) e o resultado é:



Também, podemos buscar o ano de nascimento pelo método criado na classe **getBirthYear** e o resultado será o mesmo:



Por fim, podemos usar o método **getFullName** da classe para busca do nome completo do objeto **pessoa1** na linha 69 e o resultado é:



## Lexical this no contexto de Funções

Podemos utilizar a palavra-chave **this** no contexto de funções. Em muitos casos, o valor **this** é determinado pela forma como a função é chamada. Por exemplo:

* Em um método, o **this** faz referência ao próprio objeto.
* Sozinho, o **this** faz referência ao objeto global.
* Em uma função, o **this** faz referência ao objeto global.
* Em uma função, no modo estrito, o **this** é **undefined**.
* Em um evento, o **this** faz referência ao elemento que disparou o evento.
* Em método como **call()**e **apply()** podem fazer referência do **this** para qualquer objeto.

### Vamos praticar

1. No arquivo, **main.js** digite o seguinte código.

|  |  |
| --- | --- |
| 71 | // Lexical this em uma função  console.clear();  // Constructor de função - lexical this  function PessoaFunc(firstName, lastName, dob) {      this.firstName = firstName;      this.lastName = lastName;      this.dob = new Date(dob);      this.getBirthYear = function () {          return this.dob.getFullYear();      };      this.getFullName = function () {          return `${this.firstName} ${this.lastName}`;      };  }  // Instanciando os objetos  const pessoa3 = new PessoaFunc('John', 'Doe', '1980-04-03');  const pessoa4 = new PessoaFunc('Marry', 'Smith', '1970-06-13');  console.log(pessoa3);  console.log(pessoa4.dob.getFullYear()); |
| 72 |
| 73 |
| 74 |
| 75 |
| 76 |
| 77 |
| 78 |
| 79 |
| 80 |
| 81 |
| 82 |
| 83 |
| 84 |
| 85 |
| 86 |
| 87 |
| 88 |
| 89 |
| 90 |
| 91 |

1. O resultado é mostrado no console do navegador.

Graphical user interface, text, application, email

Description automatically generated

Nesse exemplo, criamos uma função com a mesma funcionalidade da ClassePessoa, criada anteriormente. Nele temos, os atributos:

|  |
| --- |
| this.firstName = firstName;  this.lastName = lastName;  this.dob = new Date(dob); |

E os métodos:

|  |
| --- |
| this.getBirthYear = function () {      return this.dob.getFullYear();  };  const getFirstName = () => {      return `${this.firstName} ${this.lastName}`;  }; |

Observe, que nesse exemplo, o método **getBirthYear** foi criado com o lexical **this** e o método **getFirstName** foi criado com **Arrow Function**. Isso foi para mostrar que você tem várias maneiras de criar métodos em JS.

Nas linhas 87 e 88, criamos as instâncias de objetos com os valores iniciais e nas linhas 90 e 91 acessamos esses valores para imprimir no console. Observe que essa função se comporta como a classe criada anteriormente. Portanto é importante você saber as duas abordagens, pois nos sistemas em produção de uma empresa, você encontrará trechos de código com um outro tipo de implementação.

### Prototype de objetos

O JS permite criar protótipos de objetos, que basicamente é permitir criar atributos ou métodos em objetos já criados anteriormente. E esses novos protótipos são herdados automaticamente pelos

### Vamos praticar

1. No arquivo, **main.js** digite o seguinte código.

|  |  |
| --- | --- |
| 93 | // Adicionando prototipos  PessoaFunc.prototype.getBirthDayMonth = function () {      let dados = [this.dob.getDate(), this.dob.getMonth() + 1];      return dados;  };  PessoaFunc.prototype.getFirstName = function () {      return `${this.firstName}`;  };  console.clear();  console.log(pessoa3.getBirthDayMonth());  console.log(pessoa4.getFirstName()); |
| 94 |
| 95 |
| 96 |
| 97 |
| 98 |
| 99 |
| 100 |
| 101 |
| 102 |
| 103 |
| 104 |
| 105 |

1. O resultado é mostrado no console do navegador.

Graphical user interface, text, application, email

Description automatically generated

Nesse exemplo, dois protótipos novos na função **PessoaFunc()**:

|  |
| --- |
| PessoaFunc.prototype.getBirthDayMonth = function () {      let dados = [this.dob.getDate(), this.dob.getMonth() + 1];      return dados;  };  PessoaFunc.prototype.getFirstName = function () {      return `${this.firstName}`;  }; |

Esses métodos são automaticamente herdados pelos objetos **pessoa3** e **pessoa4**, que foram instanciados anteriormente nas linhas 87 e 88. Por isso, podemos usar esses métodos normalmente como mostra nas linhas 104 e 105.

## Para aprender mais

Procure sempre aprender e estudar mais. Seguem alguns links para você estudar e aprender mais:

Classes:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes>

<https://www.w3schools.com/js/js_classes.asp>

this

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/this>

<https://www.w3schools.com/js/js_this.asp>

Modo estrito:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Strict_mode>

<https://www.w3schools.com/js/js_strict.asp>

Prototype:

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/Object_prototypes>

<https://www.w3schools.com/js/js_object_prototypes.asp>