E-BOOK
DO ESTUDANTE

Orientação a objetos

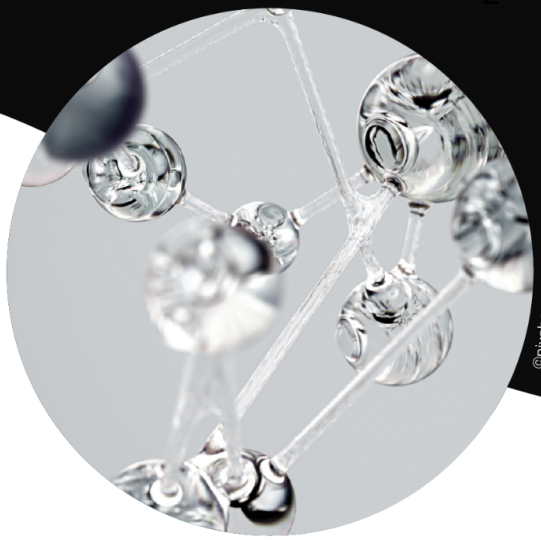
Agrupamentos

RESILIA

SUMÁRIO

Orientação a objetos

Agrupamentos



Neste e-book, exploraremos o conceito de **objetos**, suas utilizações e como eles mudaram a forma como nós pensamos a programação. Objetos são conjuntos de dados que simbolizam objetos reais, como pessoas ou qualquer outra coisa que contenha mais do que somente uma única definição ou um atributo.

Em nosso cotidiano, quase que de forma automática, nós percebemos que um gato é diferente de um cachorro, por exemplo. Afinal, esses dois animais contêm muitas diferenças de **atributos** entre si, e nosso cérebro consegue identificar em segundos as propriedades de cada um. Isso com o tempo se torna tão natural que, quando olhamos para um desses animais, conseguimos identificar que tipo de animal ele é.

Mas e no caso de um computador? Como ele vai reconhecer qual tipo de animal está presente em uma foto que ele está recebendo?

Já pensou se tivéssemos que definir variáveis para cada propriedade de um objeto para que um computador conseguisse entender que animal é aquele na foto?

Os objetos servem para nos ajudar a agrupar valores e funções em um único lugar com fácil acesso. Assim, no caso dos animais, teríamos um objeto que simbolizaria um gato e um objeto que simbolizaria um cachorro, e cada um desses objetos teria seus próprios métodos, por exemplo: raça, estrutura óssea, entre outros, que ajudariam o computador a diferenciar um do outro.

Ao final deste e-book, você será capaz de **estruturar dados** e aprenderá como **utilizar um objeto para facilitar o seu desenvolvimento dentro do Javascript**. Além disso, você compreenderá para que servem **class** e **objetos**.

CONTEXTUALIZANDO

Objetos foram desenvolvidos para ser possível agrupar dados que sejam do mesmo escopo e/ou que tenham a mesma finalidade, sem que seja necessário definir diversas variáveis e diversas funções.

A utilização de objetos é constante e comum no mercado de trabalho. Diariamente vemos programadores mexendo com estruturas de objetos em diferentes linguagens. Uma inteligência artificial, por exemplo, é construída, organizada e estruturada à base de objetos. Além disso, linguagens como **java**, **typescript**, c e outras diversas utilizam a estrutura de objeto como **estrutura-padrão** para o seu desenvolvimento. Os bilhões de dados que são transportados de um lado para o outro na web também têm estruturas de objetos.

Todos esses exemplos nos mostram que um objeto é muito importante para o computador e que, portanto, ele é muito importante para o nosso desenvolvimento como programadores.

Agora que já entendemos que, sem dúvida, iremos utilizar objetos diariamente trabalhando na área de TI, vamos entender então o que é um objeto e como nós podemos manipulá-lo para criar, assim, um código que seja totalmente orientado a objetos. Vamos lá?

OBJETOS

Na programação, o objeto foi desenvolvido para ser possível armazenar vários conjuntos de dados em um único grupo. Imagine o seguinte exemplo: uma pessoa tem uma gata da raça Siamês chamada Jujuba. Quando nós vamos passar esse dado para o computador, precisamos usar um objeto para armazenar todos os dados dessa gata, desta forma:

```
const gato = {
  nome: "Jujuba",
  raca: "Siamês",
  sexo: "F",
};
```

Note que há uma **constante**. Vale lembrar que uma *constante* é uma variável que não pode ser reatribuída, ou seja, o valor dela não pode variar. Assim, após atribuir o valor inicial a essa variável, esse dado não pode mais ser substituído por outro.

Além da constante, notamos que há **{}** (**chaves**). Um objeto é simbolizado por chaves, permitindo, assim, que os valores dentro dele sejam estruturados na estrutura JSON, ou seja, que os valores dentro dele sempre tenham uma chave (nome) e um valor. No exemplo estudado, vemos a propriedade: chave "raça" e valor "Siamês". *Propriedade* é o nome dado a uma "variável" que armazena um valor dentro de um objeto.

Note ainda que, dentro do objeto, estão reunidas diversas informações, como: o nome, a raça e o sexo da gata, sendo todas essas informações referentes a um único objeto que foi atribuído para a constante de nome *gato*.

Também encontramos **funções** dentro de um objeto, ou seja, blocos de código que executam uma ou mais ações, permitindo ter, por exemplo, nossas propriedades (variáveis) e nossos métodos (funções). Dentro de um objeto, uma função se comporta um pouco diferente. Além de seu nome ser *método*, ela tem uma sintaxe diferente, como podemos ver no exemplo abaixo:

```
const gato = {
  nome: "Jujuba",
  raca: "Siamês",
  sexo: "F",
  miar: () => {
    console.log("Miau Miau!!!");
  },
};
```

Note que temos o mesmo objeto do exemplo anterior, porém agora identificamos um método chamado *miar*. Assim, sempre que ele for executado, esse método exibirá no console o texto "Miau Miau!!!".

Ao analisar o exemplo, você percebeu que a sintaxe mudou um pouco?

Dentro de objetos, um método é composto por uma chave (nome) e uma função anônima. No exemplo, vemos a chave "miar" e estamos usando uma *arrow function*, que é mais conhecida no Brasil como *função de seta*. Esse tipo de função nos permite ter menos código para conseguir declará-la, além de se comportar como uma função anônima, ou seja, ela precisa receber um nome de uma variável (constante) ou de uma chave. Esta *arrow function* está inserindo um *log* no console com o texto "Miau Miau!!!".

Vimos, então, que um objeto é composto por propriedades e métodos, e isso permite que ele agrupe todos esses dados em um único bloco de código,

resultando em um código mais limpo e dispensando a criação de variáveis para cada um dos valores.

A orientação a objetos é diariamente utilizada para o funcionamento de diversas linguagens, como java, python, c, c++, entre outras. Ela é também um artifício muito utilizado para separar arquivos.

Portanto, entender como um objeto funciona e saber usá-lo na programação é muito importante não só para o andamento do nosso curso, mas também, e principalmente, para o enriquecimento da sua carreira e para o seu desenvolvimento em múltiplas linguagens.

Vimos como um objeto se comporta e como nós criamos e atribuímos valores e funções a ele, mas como utilizamos um objeto?

Para utilizar um objeto, utilizamos sempre a sintaxe de **`nomeDoObjeto.nomeDaChave`**, como no exemplo abaixo:

```
const gato = {
  nome: "Jujuba",
  raca: "Siamês",
  sexo: "F",
  miar: () => {
    console.log("Miau Miau!!!");
  },
};

console.log(“Olá, pessoal, esta é minha gata ” + gato.nome)
console.log(“Ela é uma gata ” + gato.raca)
```

Note que os consoles *log* são utilizados para mostrar os dados. No primeiro console *log*, vemos o seguinte texto: “Olá, pessoal, esta é minha gata”, que está sendo concatenado (juntado) ao valor da propriedade “nome” dentro do objeto “gato”, gerando o resultado: “Olá, pessoal, esta é minha gata Jujuba”.

No segundo console *log*, vemos a mesma sintaxe, porém, desta vez, o texto “Ela é uma gata” é concatenado com o valor da propriedade “raça” dentro do objeto *gato*, tendo como retorno o seguinte valor: “Ela é uma gata siamês”. Note como a utilização e o acesso a propriedades dentro do objeto são simples e eficientes, de modo a nos permitir selecionar um valor através do nome de sua chave.

Como vimos, o objeto é bastante útil na programação, pois nos permite, em diversas situações, simplificar o modo como armazenamos, organizamos e estruturamos os dados e as informações.

A orientação a objetos também faz parte de diversos sistemas na *web* e contribui para que os muitos dados sejam transportados de um lado para o outro na *web* sem se perderem.

Já pensou se tudo isso fosse feito utilizando variáveis e se, em algum momento, perdêssemos algum dado? Isso seria trágico para a segurança, pois, além de facilitar a vida de um *hacker*, poderíamos acabar enviando dados para pessoas erradas ou até mesmo perdendo esses dados.

Aprender sobre os objetos e saber a importância deles nos permite, inclusive entender como funciona a programação nas inteligências artificiais e em muitos outros projetos que vemos hoje em dia. É por isso que sempre precisamos estar cientes de cada recurso que as linguagens nos oferecem e sempre nos atualizar e conhecer coisas novas.

Atividade: Meu *pet*

Crie um objeto que defina algum dos seus pets ou outro animal que você conhece.

Utilize propriedades e métodos que este pet ou animal contém na vida real, aproximando-se o máximo possível do real.

Esta atividade ajudará você a entender melhor como os objetos podem simbolizar elementos do nosso cotidiano e, além disso, ajudará a fixar os conceitos vistos até o momento.

Para refletir

- Qual é a vantagem de utilizar objetos?
- O que um objeto representa dentro da programação?
- Em qual situação nós utilizamos um objeto?

CLASS

Até o momento, vimos como podemos criar um objeto e como inserimos valores dentro dele manualmente. Mas como podemos passar valores e preencher aquele objeto com dados aleatórios, que nem sempre serão os mesmos?
Para isso, utilizamos a *class*, ou seja, a classe. Ela nos permite criar um objeto de modo a estruturar os dados e modelá-los.

A *class* é muito utilizada dentro da programação, principalmente nas linguagens tipadas (aquelas que têm suas variáveis com tipos, por exemplo: variáveis numéricas somente recebem números e assim seletivamente).

Para entender como uma *class* funciona, podemos pensar em um molde, tal como aquele molde de chocolate usado para fazer ovos de Páscoa. Esse molde serve para receber o chocolate derretido e, então, moldar o chocolate na forma oval, resultando, assim, em um ovo de chocolate. É também assim que funciona uma *class* dentro da programação: ela serve para modelar dados e para criar objetos com esses dados modelados.

Quando temos muitos dados ou muitas informações e precisamos de alguma forma tornar todas essas informações iguais, é necessário estruturá-las e modelá-las para que elas tenham as mesmas propriedades e os mesmos métodos. Para isso, utilizamos a *class*.

Dentro do Javascript, a *class* ainda não tem a mesma complexidade do que a *class* de linguagens tipadas, mas ela está em evolução para atingir sua maior potência, então isso deve mudar já na próxima versão do Javascript.

Para entender melhor como conseguimos criar classes e modelar dados dentro do Javascript, vamos dar uma olhada na sintaxe de uma *class*:

```
class Gato {
  contructor(nome, raca, sexo) {
    this.nome = nome;
    this.raca = raca;
    this.sexo = sexo;
  }
}
```

Note que temos agora uma sintaxe totalmente diferente da sintaxe de criação de um objeto estático. Isso ocorre porque a *class* é um molde, então ela precisa construir um objeto ao invés de já ser o objeto final.

Nessa sintaxe, vemos então uma classe “Gato”. Repare que a primeira letra do nome da classe é maiúscula, o que é uma boa prática porque ajuda a diferenciar a *class* das variáveis e dos métodos. Então, é uma boa prática sempre atribuir a primeira letra em maiúscula.

Depois do nome da classe, temos uma abertura de chaves e, dentro das chaves, temos o construtor, o qual é um método.

Um método dentro de uma *class* se comporta um pouco diferente de um método dentro de um objeto estático. Note que o próprio método já contém seus parênteses e já abre suas chaves, sem precisar de uma propriedade para a nomeação. Isso ocorre porque, dentro de uma *class*, os métodos são nomeados e funcionam identicamente às funções, porém não carregam o nome *function* antes do nome.

Como ele é um método interno, então a palavra “*constructor*” dentro de uma *class* deve ser sempre utilizada para a criação dos atributos do objeto.

Ao repararmos dentro do *contructor*, vemos que há três parâmetros, sendo eles: nome, raça e sexo. Os parâmetros de um objeto se comportam exatamente igual aos de uma função, ou seja, são variáveis que esperam dados de fora e que serão passados como argumentos ao executar a função.

Na sintaxe que estamos analisando, vemos ainda uma palavra que ainda não exploramos. É a palavra *this*. Dentro do *contructor,* a palavra *this* sempre simboliza que estamos construindo uma propriedade (por exemplo, “nome”) dentro da classe em que essa propriedade está (classe Gato). Ou seja, o *this* simboliza que estamos executando, criando ou manipulando algo dentro da *class* em que ele está (neste caso, a *class* Gato).

Vimos, então, como é a sintaxe de declaração de uma *class* (sintaxe de criação). Porém, para podermos usá-la, precisamos instanciá-la. Mas o que seria “instanciar”?

Instanciar é como se pegássemos aquele molde preenchido com o chocolate derretido e o colocássemos na geladeira para ele ficar duro o suficiente para conseguirmos tirá-lo do molde. Esse processo de resfriamento e de tratamento dos ovos de Páscoa é um exemplo do processo de instanciar.

Antes de um carro ser colocado à venda, por exemplo, ele passa pelo processo de instância, já que ele chega em partes na fábrica, onde essas partes são juntadas e onde é feita a montagem, a testagem e a revisão, para só depois o carro poder ser liberado para o comércio. Todo esse processo de montagem, testes, ajustes e tudo mais que fica entre a criação e o produto final fazem parte de um processo de instância. Assim, chamamos de "instância" o processo que existe entre a criação de um produto e o produto final.

O mesmo ocorre na programação com a *class*. O processo de instância de uma *class* são todos aqueles processos que o Javascript precisa fazer para que ele consiga entregar o produto final — que é o objeto construído com os dados passados como argumentos.

Para instanciar o Javascript, nós usamos o operador "*new*", que permite pegar uma classe e criar um objeto a partir do modelo dessa classe. Para entender melhor, vamos ver um exemplo:

```
class Gato {
  contructor(nome, raca, sexo) {
    this.nome = nome;
    this.raca = raca;
    this.sexo = sexo;
  }

  miar(){
    console.log("miau")
  }
}

const Jujuba = new Gato("Jujuba","siamês","F")
```

Note que nós temos nossa classe Gato com todas aquelas propriedades que já vimos anteriormente e temos de novo o nosso método *miar* que traz um console *log* com a palavra "miau". Como o método dentro da *class* se comporta de um modo diferente, então não precisamos criar uma propriedade para armazenar uma função anônima.

Dentro de uma *class*, um método tem uma sintaxe parecida com a de uma função, contendo: seu próprio nome, seus parâmetros e seu bloco código. Porém, em contraste com uma função, um método não carrega o nome *function* antes do nome, e isso ajuda a distinguir quando temos um método (uma função dentro de uma *class*) de quando temos uma função nomeada.

Também temos novamente a nossa constante de nome “Jujuba”, a qual está recebendo um *novo Gato* que terá: o valor de sua propriedade “nome” como “Jujuba”; o valor de sua propriedade “raça” como “siamês”; e o valor de sua propriedade “sexo” como “F”.

Estamos usando o operador *new*, e isso quer dizer que criaremos um novo objeto modelado pela *class* Gato contendo os valores que foram passados como argumentos.

Note que poderíamos facilmente adicionar uma quantidade ilimitada de gatos, necessitando apenas adicionar uma constante e os valores referentes aos gatos que quiséssemos criar. Eles sairiam, então, com as mesmas propriedades e os mesmos métodos da gata Jujuba. A única coisa que mudaria seriam os valores.

Quando o processo de instância se encerra, a *class* nos retorna um objeto, ou seja, o produto final do processo. No caso da Jujuba, o objeto seria o mesmo que nós já vimos:

```
{
  nome: "Jujuba",
  raca: "siamês",
  sexo: "F",
  miar: () => {
    console.log("miau");
  },
};
```

Ou seja, o que antes criamos manualmente, agora criamos a partir de um modelo que pode se repetir para uma quantidade ilimitada de gatos, sem precisarmos inserir tudo manualmente, nem criar objeto por objeto de forma manual com seus dados estáticos.

Eis a vantagem de uma *class*! Ela nos possibilita ter um modelo para sempre acessar e criar dados partindo daquele modelo, sem termos que ficar criando linhas e linhas para cada objeto diferente, tornando um código reutilizável e tornando também nosso código mais limpo.

Este *e-book* demonstrou como nós podemos utilizar estruturas de dados (objetos) a nosso favor e como geramos essas estruturas de forma reutilizável para que, em nossos trabalhos do dia a dia, tenhamos uma maior agilidade.

Objeto e *class* são conceitos um pouco difíceis e complexos de se entender, porém, quanto mais os praticamos, mais conseguimos evoluir e compreender melhor a utilidade de cada um.

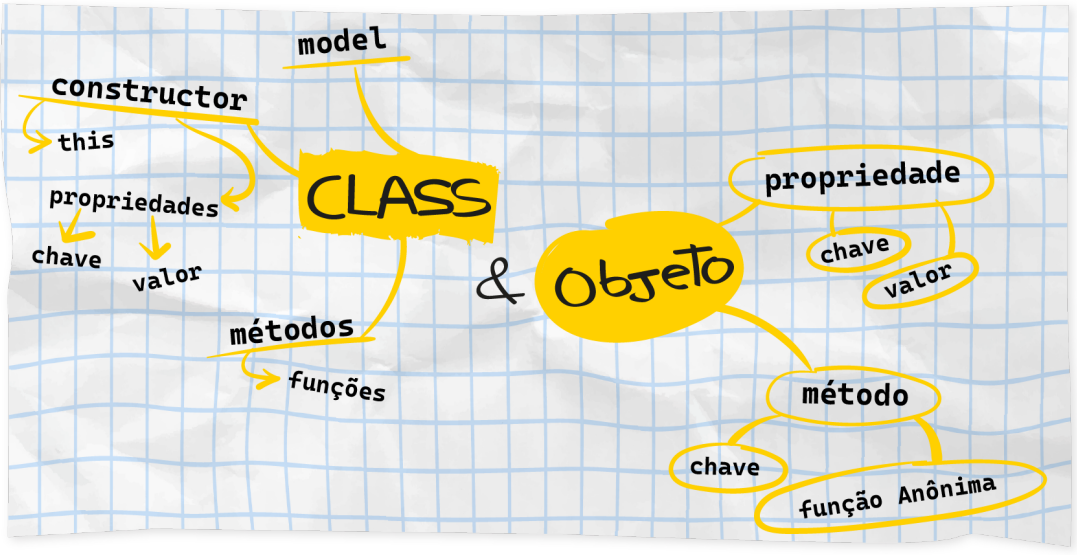
Para refletir

- Qual é a vantagem da utilização de uma class?
- Qual é a utilidade do operador new?
- O que é um processo de instância? O que ele faz em uma class?

Para ir além

- Como vimos, os objetos são fundamentais para o desenvolvimento em Javascript. Para se aprofundar e conhecer mais funcionalidades de um objeto, acesse:
  <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object >
  <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects>
  <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/Basics>

NUVEM DE PALAVRAS



Até a próxima e
#confianoprocesso

RESILIA