Objetos em JavaScript

Conhecimentos	
Data da aula	@12 de agosto de 2024
Tipo	Atividade em Classe Aula Expositiva

Objetivos de aprendizado

1. Quais são os principais objetivos de aprendizagem desta lição?
2. Que papel esta lição desempenha no curso principal?
3. Como você imagina esta aula fomentando a profundidade de conhecimento, julgamento, habilidades analíticas ou outras capacidades de cada aluno?

Objetos

Existem apenas sete tipos de dados fundamentais em JavaScript, e seis deles são os tipos de dados primitivos: string, número, booleano, nulo, indefinido e símbolo. Com o sétimo tipo, objetos, abrimos nosso código para possibilidades mais complexas.

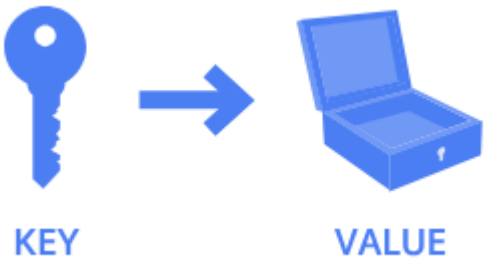
Em sua essência, os objetos JavaScript são contêineres que armazenam dados e funcionalidades relacionadas, mas essa tarefa aparentemente simples é extremamente poderosa na prática.

Criando Objetos Literais

Os objetos podem ser atribuídos a variáveis como qualquer tipo de JavaScript. Usamos chaves, `{}`, para designar um *objeto literal*:

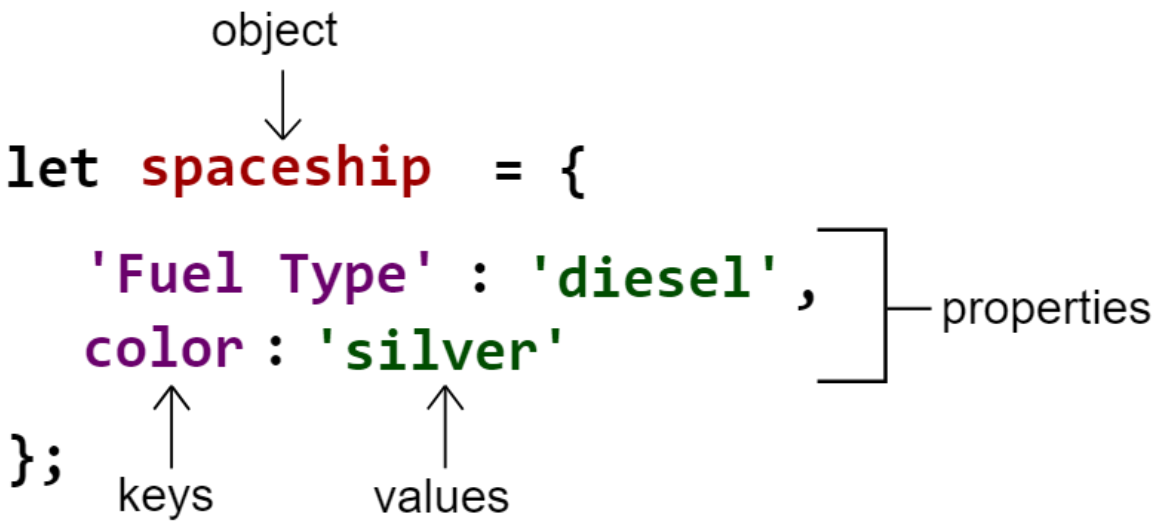
```
let spaceship = {}; // spaceship é um objeto vazio
```

Preenchemos um objeto com dados não ordenados. Esses dados são organizados em *pares de valores-chave*. Uma chave é como um nome de variável que aponta para um local na memória que contém um valor.



O valor de uma chave pode ser de qualquer tipo de dados na linguagem, incluindo funções ou outros objetos.

Fazemos um par chave-valor escrevendo o nome da chave, ou *identificador*, seguido por dois pontos e depois o valor. Separamos cada par de valores-chave em um literal de objeto com uma vírgula (`,`). Chaves são strings, mas quando temos uma chave que não contém nenhum caractere especial, o JavaScript nos permite omitir as aspas:




```
// Um objeto literal com dois pares de valores-chave
let spaceship = {
  'Fuel Type': 'diesel',
  color: 'silver'
};
```

O objeto `spaceship` tem duas propriedades `Fuel Type` e `color`. `'Fuel Type'` tem aspas porque contém um caractere de espaço.

Acessando Propriedades

Existem duas maneiras de acessar a propriedade de um objeto. Vamos explorar a primeira forma – notação de ponto, `.`.

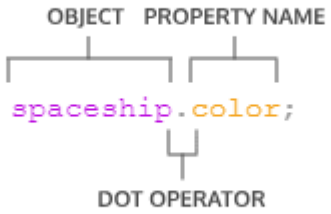
Você usou a notação de ponto para acessar as propriedades e métodos de objetos integrados e instâncias de dados:

```
'hello'.length; // Returns 5
```

Com a notação de ponto de propriedade, escrevemos o nome do objeto, seguido pelo operador ponto e depois pelo nome da propriedade (chave):

```
let spaceship = {
  homePlanet: 'Earth',
  color: 'silver'
```

```
};
spaceship.homePlanet; // Retorna 'Earth',
spaceship.color; // Retorna 'silver',
```



Se tentarmos acessar uma propriedade que não existe naquele objeto, `undefined` será retornado.

```
spaceship.favoriteIcecream; // Returns undefined
```

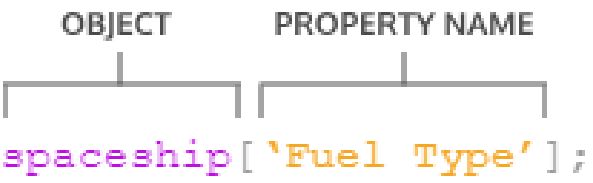
Notação de colchetes

A segunda maneira de acessar o valor de uma chave é usando a notação de colchetes, `[ ]`.

Você usou a notação de colchetes ao indexar uma matriz:

```
['A', 'B', 'C'][0]; // Retorna 'A'
```

Para usar a notação de colchetes para acessar a propriedade de um objeto, passamos o nome da propriedade (chave) como uma string.



Devemos **usar** a notação de colchetes ao acessar chaves que contenham números, espaços ou caracteres especiais. Sem a notação de colchetes nessas situações, nosso código geraria um erro.

```
let spaceship = {
  'Fuel Type': 'Turbo Fuel',
  'Active Duty': true,
  homePlanet: 'Earth',
  numCrew: 5
};
spaceship['Active Duty'];   // Retorna true
spaceship['Fuel Type'];   // Retorna 'Turbo Fuel'
spaceship['numCrew'];   // Retorna 5
spaceship['!!!!!!!!!!!!!!!'];   // Retorna undefined
```

Com a notação de colchetes você também pode usar uma variável dentro dos colchetes para selecionar as chaves de um objeto. Isto pode ser especialmente útil ao trabalhar com funções:

```
let returnAnyProp = (objectName, propName) => objectName[prop

returnAnyProp(spaceship, 'homePlanet'); // Retorna 'Earth'
```

Se tentássemos escrever a função `returnAnyProp()` com notação de ponto (`objectName.propName`), o computador procuraria uma chave `'propName'` em nosso

objeto e não o valor do parâmetro `propName`.

Atribuição de Propriedade

Depois de definirmos um objeto, não ficaremos presos a todas as propriedades que escrevemos. Os objetos são *mutáveis,* o que significa que podemos atualizá-los depois de criá-los!

Podemos usar a notação de ponto, `.` ou a notação de colchetes, `[]` e o operador de atribuição `=` para adicionar novos pares de valores-chave a um objeto ou alterar uma propriedade existente.

Uma de duas coisas pode acontecer com a atribuição de propriedade:

- Se a propriedade já existir no objeto, qualquer valor que ela mantivesse antes será substituído pelo valor recém-atribuído.
- Caso não exista nenhuma propriedade com esse nome, uma nova propriedade será adicionada ao objeto.

É importante saber que embora não possamos reatribuir um objeto declarado com `const`, ainda podemos modificá-lo, o que significa que podemos adicionar novas propriedades e alterar as propriedades que estão lá.

```
const spaceship = {type: 'shuttle'};
spaceship = {type: 'alien'}; // TypeError: Assignment to cons
spaceship.type = 'alien'; // Changes the value of the type pr
spaceship.speed = 'Mach 5'; // Creates a new key of 'speed' w
```

Você pode excluir uma propriedade de um objeto com o operador `delete`.

```
const spaceship = {
  'Fuel Type': 'Turbo Fuel',
  homePlanet: 'Earth',
  mission: 'Explore the universe'
};
```

```
delete spaceship.mission;  // Removes the mission property
```

Métodos

Quando os dados armazenados em um objeto são uma função, chamamos isso de *método*. Uma propriedade é o que um objeto possui, enquanto um método é o que um objeto faz.

Os métodos de objeto parecem familiares? Isso porque usamos o tempo todo! Por exemplo, `console` é um objeto global do JavaScript global e `.log()` é um método desse objeto. `Math` também é um objeto global JavaScript e `.floor()` é um método dele.

Podemos incluir métodos em nossos objetos literais criando pares de valores-chave comuns separados por dois pontos. A chave serve como nome do nosso método, enquanto o valor é uma expressão de função anônima.

```
const alienShip = {
  invade: function () {
    console.log('Hello! We have come to dominate your planet.
  }
};
```

Com a nova sintaxe de método introduzida no ES6 podemos omitir os dois pontos e a palavra `function`.

```
const alienShip = {
  invade () {
    console.log('Hello! We have come to dominate your planet.
  }
};
```

Os métodos são invocados anexando o nome do objeto com o operador ponto seguido pelo nome do método e parênteses:

```
alienShip.invade(); // Prints 'Hello! We have come to dominat
```

Objetos Aninhados

No código do aplicativo, os objetos geralmente são aninhados — um objeto pode ter outro objeto como propriedade que, por sua vez, pode ter uma propriedade que é um array de ainda mais objetos!

Em nosso objeto `spaceship`, queremos um objeto `crew`. Este conterá todos os membros da tripulação que realizam trabalhos importantes na nave. Cada um desses membros `crew` são objetos. Eles têm propriedades como `name` e `degree` e cada um deles possui métodos exclusivos com base em suas funções. Também podemos aninhar outros objetos `spaceship` como a `telescope` ou aninhar detalhes sobre os computadores da nave espacial dentro de um `nanoelectronics` objeto pai.

```
const spaceship = {
     telescope: {
        yearBuilt: 2018,
        model: '91031-XLT',
        focalLength: 2032
     },
    crew: {
        captain: {
            name: 'Sandra',
            degree: 'Computer Engineering',
            encourageTeam() { console.log('We got this!') }
         }
    },
    engine: {
        model: 'Nimbus2000'
     },
     nanoelectronics: {
         computer: {
            terabytes: 100,
            monitors: 'HD'
         },
        'back-up': {
           battery: 'Lithium',
           terabytes: 50
         }
```

```
    }
};
```

Podemos encadear operadores para acessar propriedades aninhadas. Teremos que prestar atenção em qual operador faz sentido usar em cada camada. Pode ser útil fingir que você é o computador e avaliar cada expressão da esquerda para a direita, para que cada operação comece a parecer um pouco mais gerenciável.

```
spaceship.nanoelectronics['back-up'].battery; // Returns 'Lit
```

Objetos podem ser *passados como referência*. Isso significa que quando passamos uma variável atribuída a um objeto para uma função como argumento, o computador interpreta o nome do parâmetro como apontando para o espaço na memória que contém esse objeto. Como resultado, funções que alteram as propriedades do objeto, na verdade, modificam o objeto permanentemente (mesmo quando o objeto é atribuído a uma variável `const`).

```
const spaceship = {
  homePlanet : 'Earth',
  color : 'silver'
};

let paintIt = obj => {
  obj.color = 'glorious gold'
};

paintIt(spaceship);

spaceship.color // Retorna'glorious gold'
```

Nossa função `paintIt()` mudou permanentemente a cor do nosso objeto `spaceship`. No entanto, a reatribuição da variável `spaceship` não funcionaria da mesma maneira:

```
let spaceship = {
  homePlanet : 'Earth',
```

```
  color : 'red'
};
let tryReassignment = obj => {
  obj = {
    identified : false,
    'transport type' : 'flying'
  }
  console.log(obj) // Imprime {'identified': false, 'transpor

};
tryReassignment(spaceship) // A tentativa de reatribuição não
spaceship // Continua retornando {homePlanet : 'Earth', color

spaceship = {
  identified : false,
  'transport type': 'flying'
}; // A reatribuição regular ainda funciona.
```

Loops através de objetos

Aprendemos como iterar arrays usando sua indexação numérica, mas os pares chave-valor nos objetos não são ordenados! JavaScript nos deu uma solução alternativa para iterar objetos com a sintaxe `for...in`.

`for...in` executará um determinado bloco de código para cada propriedade em um objeto.

```
let spaceship = {
  crew: {
    captain: {
      name: 'Lily',
      degree: 'Computer Engineering',
      cheerTeam() { console.log('You got this!') }
    },
    'chief officer': {
      name: 'Dan',
      degree: 'Aerospace Engineering',
      agree() { console.log('I agree, captain!') }
    },
```

```
    medic: {
      name: 'Clementine',
      degree: 'Physics',
      announce() { console.log(`Jets on!`) } },
    translator: {
      name: 'Shauna',
      degree: 'Conservation Science',
      powerFuel() { console.log('The tank is full!') }
    }
  }
};

// for...in
for (let crewMember in spaceship.crew) {
  console.log(`${crewMember}: ${spaceship.crew[crewMember].na
}
```

Nosso `for...in` irá iterar através de cada elemento do objeto `spaceship.crew`. Em cada iteração, a variável `crewMember` é definida como uma das chaves `spaceship.crew`', permitindo-nos registrar uma lista de funções e `name`.

A palavra-chave this

Objetos são coleções de dados e funcionalidades relacionadas. Armazenamos essa funcionalidade em métodos em nossos objetos:

```
const goat = {
  dietType: 'herbivore',
  makeSound() {
    console.log('baaa');
  }
};
```

Em nosso objeto `goat` temos um método `.makeSound()`. Podemos invocar o método `.makeSound()` em `goat`.

```
goat.makeSound(); // Imprime baaa
```

E se quiséssemos adicionar um novo método ao nosso objeto chamado `.diet()` que imprime o `dietType` de `goat`?

```
onst goat = {
  dietType: 'herbivore',
  makeSound() {
    console.log('baaa');
  },
  diet() {
    console.log(dietType);
  }
};
goat.diet();
// Output will be "ReferenceError: dietType is not defined"
```

Isso é estranho, por que `dietType` não está definido mesmo sendo uma propriedade de `goat`? Isso porque dentro do escopo do método `.diet()` não temos acesso automático às outras propriedades do `goat` objeto.

É aqui que a palavra-chave `this` vem em socorro. Se mudarmos o método `.diet()` para usar o `this`, `.diet()` funciona! :

```
const goat = {
  dietType: 'herbivore',
  makeSound() {
    console.log('baaa');
  },
  diet() {
    console.log(this.dietType);
  }
};

goat.diet();
// Output: herbivore
```

A palavra-chave `this` faz referência ao *objeto de chamada* que fornece acesso às propriedades do objeto de chamada. No exemplo acima, o objeto de chamada é `goat` e usando `this` estamos acessando o próprio objeto `goat` e,

em seguida, a propriedade de `goat` , `dietType,` usando a notação de ponto de propriedade.

Atividades de Aprendizado

Exercício 1

Crie uma variável chamada casa que execute as seguintes operações:

1. Atribua um objeto a variável com as propriedades:
    a. quartos = 2
    b. tipo = “casa”
    c. endereco = “rua teste 123 – ZS”
2. Exiba no console a seguinte frase (usando TODOS as propriedades): Casa com 2 quartos localizada na rua teste 123 – ZS.

Exercício 2

Corrija o objeto abaixo de forma a não ocorrer erro quando você executar o comando console.log(produtos.jogos.acao[2]).

```
let produtos = {
    videos : {
        comedia: ["comedia1","comedia2"],
        romance: ["romance1", "romance2"]
    },

    revistas : [
        moda  = ["lalala", "lililili"],
        saude  = ["lalala", "lililili"],
    ]

    jogos : {
        rpg: ["rpg1", "rpg2", rpg3],
        acao: ["acao1", "God of War"]
    }
}
```

Exercício 3

Escreva um programa JavaScript para listar as propriedades de um objeto:

Objeto de exemplo:

```
var student = {
nome: "João Carlos",
sclass: "Informatica Basica",
rollno: 12 };

// Exemplo de saída : name,sclass,rollno
```

Exercício 4

Escreva um programa JavaScript para excluir a propriedade rollno do objeto a seguir. Imprima também o objeto antes e depois de excluir a propriedade.

```
Objeto de exemplo :
var student = {
nome: "David Rayy",
sclass: "VI",
rollno: 12 };
```

Exercício 5

Escreva um programa JavaScript para exibir o status de leitura (ou seja, exibir o nome do livro, o nome do autor e o status de leitura) dos seguintes livros.

```
var biblioteca = [
   {
       autor: 'Bill Gates',
       título: 'The Road Ahead',
       readingStatus: true
   },
   {
       autor: 'Steve Jobs',
       título: 'Walter Isaacson',
       readingStatus: true
   },
```

```
    {
        autor: 'Suzanne Collins',
        título: 'A Esperança: O Livro Final de Jogos Vorazes',
        readingStatus: false
    }];
```

Exercício 6

Escreva um programa JavaScript para obter o volume de um cilindro com quatro casas decimais usando classes de objetos.

Volume de um cilindro: $V = \pi r^2 h$

onde r é o raio e h é a altura do cilindro