Aula 5 - Arrays e Objetos

aTip Learn - Lógica de Programação com JavaScript

Objetivos da Aula

- Entender o conceito de arrays e objetos em JavaScript.
- Criar e manipular arrays, adicionando, removendo e acessando elementos.
- Criar e manipular objetos, acessando e modificando propriedades.
- Utilizar laços de repetição para percorrer arrays e objetos.
- Aplicar métodos e propriedades de arrays e objetos para processar dados.

Arrays

Um array é uma estrutura de dados que armazena uma coleção de elementos, que podem ser acessados por meio de um índice numérico. Em JavaScript os arrays são um tipo de objeto especial, que nos permite armazenar diversos valores em uma única variável. Para criar um array, utilizamos colchetes `[]` e separamos os elementos por vírgulas. Por exemplo:

```
let frutas = ['maçã', 'banana', 'laranja', 'uva'];
```

Neste exemplo, criamos um array chamado `frutas`, que contém quatro elementos do tipo string: 'maçã', 'banana', 'laranja' e 'uva'. Os array em JavaScript podem conter elementos de diferentes tipos, como números, strings, objetos, ou até mesmo outros arrays. Por exemplo:

```
let lista = [1, 'dois', { nome: 'João', idade: 30 }, [4, 5, 6]];
```

Acessando Elementos por Índice

Para acessar um elemento específico de um array, utilizamos o índice correspondente, a partir do índice 0. Por exemplo, para acessar a primeira fruta do array `frutas`, utilizamos `frutas[0]`, para acessar a segunda fruta, utilizamos `frutas[1]`, e assim por diante.

```
console.log(frutas[0]); // maçã
console.log(frutas[1]); // banana
```

Utilizando o índice podemos ainda modificar os elementos do array, atribuindo um novo valor a uma posição específica. Por exemplo, para alterar a segunda fruta do array `frutas`, podemos fazer:

```
frutas[1] = 'morango';
console.log(frutas); // ['maçã', 'morango', 'laranja', 'uva']
```

Adicionando e Removendo Elementos

Existem diversos métodos para adicionar e remover elementos de um array em JavaScript. Alguns dos métodos mais comuns são:

- `array.push(elemento)`: Adiciona um elemento ao final do array.
- `array.pop()`: Remove o último elemento do array.
- `array.shift()`: Remove o primeiro elemento do array.
- `array.unshift(elemento)`: Adiciona um elemento ao início do array.

Por exemplo, se desejarmos adicionar a fruta abacaxi ao final do array e a fruta pêra ao início do array, podemos fazer:

```
frutas.push('abacaxi');
frutas.unshift('pêra');
console.log(frutas); // ['pêra', 'maçã', 'morango', 'laranja', 'uva', 'abacaxi']
```

Para remover o último elemento do array e o primeiro elemento do array, podemos fazer:

```
frutas.pop(); // remove 'abacaxi'
frutas.shift(); // remove 'pêra'
console.log(frutas); // ['maçã', 'morango', 'laranja', 'uva']
```

É importante notar que todos esses métodos modificam o array original, ou seja, eles não retornam um novo array modificado, mas sim alteram o array existente!

Comprimento do Array

Para saber o comprimento de um array, ou seja, quantos elementos ele possui, podemos utilizar a propriedade `length`. Por exemplo, para saber quantas frutas temos no array `frutas`, podemos fazer:

```
console.log(frutas.length); // 4
```

Laços de Repetição

Os laços de repetição são estruturas de controle que permitem executar um mesmo bloco de código várias vezes, de forma automática. Em JavaScript, existem diversos tipos de laços de repetição, como `for`, `for...of`, `for...in`, `while` e `do...while`. Todos esses laços possuem a mesma finalidade, executar um bloco de código repetidas vezes, entretanto cada um deles possui suas particularidades e é mais adequado para determinadas situações. Nessa aula exploraremos os laços `for`, `for...of` e `while`, que são os mais comuns e versáteis.

Laço for

O laço `for` é um dos laços de repetição mais comuns nas linguagens de programação. Costumamos utilizar o laço for quando sabemos o número de vezes que queremos repetir um bloco de código, seja esse um número fixo, ou uma variável como o comprimento de um array.

A estrutura do laço `for` é composta por três partes: um *inicializador*, uma *condição* de continuação e um *incremento* (ou decremento).

```
for (inicializador; condição; incremento) {
  // bloco de código a ser repetido
}
```

Por exemplo, para exibirmos os números de 1 até 3 no console, podemos fazer:

```
console.log('Início');
for (let i = 1; i <= 3; i = i + 1) {
  console.log(i);
}
console.log('Fim');
```

Vejamos passo a passo o que é feito nesse laço:

- O *inicializador* é executado e variável `i` é inicializada com o valor 1.
- A *condição* é verificada, como `i` é menor ou igual a 3, o bloco de código é executado e o valor `1` é exibido no console.
- O *incremento* é executado, a variável `i` é incrementada em 1, passando a valer 2.
- A *condição* é verificada novamente, como `i` é menor ou igual a 3, o bloco de código é executado e o valor `2` é exibido no console.
- O *incremento* é executado novamente, a variável `i` é incrementada em 1, passando a valer 3.
- A *condição* é verificada novamente, como `i` é menor ou igual a 3, o bloco de código é executado e o valor `3` é exibido no console.
- O *incremento* é executado novamente, a variável `i` é incrementada em 1, passando a valer 4.
- A *condição* é verificada novamente, como `i` é maior que 3, o laço é encerrado.

Dessa forma, a variável `i` é incrementada de 1 em 1, até que a condição `i <= 3` seja falsa, encerrando o laço. A variável `i` é chamada de *variável de controle*, pois controla a execução do laço, determinando quando ele deve ser encerrado.

É preciso ter cuidado para que a condição de continuação seja falsa em algum momento, caso contrário o laço se tornará infinito, executando indefinidamente e travando o Navegador ou o NodeJS. Esse tipo de erro poderia ocorrer em nosso exemplo anterior caso estivessemos decrementando a variável `i`, pois nesse caso o valor de `i` nunca seria maior que 3.

Podemos utilizar o laço `for` para percorrer os elementos de um array, utilizando o comprimento do array como condição de continuação. Por exemplo, para exibirmos os elementos do array `frutas` no console, podemos fazer:

```
for (let i = 0; i < frutas.length; i = i + 1) {
  console.log(frutas[i]);
}
```

Nesse exemplo percorrermos o array `frutas` utilizando a variável `i` como índice, exibindo cada elemento do array no console. Perceba que a condição de continuação é `i < frutas.length`, ou seja, o laço será executado enquanto `i` for menor que o comprimento do array `frutas`. Assim, se o comprimento do nosso array for 4, o laço será executado para `i = 0`, `i = 1`, `i = 2` e `i = 3`, exibindo todos os elementos do array.

Laço for...of

O laço `for...of` é uma forma simplificada de percorrer os elementos de um array, sem a necessidade de utilizar um índice numérico. A estrutura do laço `for...of` é a seguinte:

```
for (let elemento of array) {
  // bloco de código a ser repetido
}
```

Por exemplo, para exibirmos os elementos do array `frutas` no console, podemos fazer:

```
for (let fruta of frutas) {
  console.log(fruta);
}
```

Nesse exemplo, a variável `fruta` assume o valor de cada elemento do array `frutas`, sem a necessidade de utilizar um índice numérico. O laço `for...of` é mais simples e legível do que o laço `for`, e pode ser utilizado sempre que não precisarmos do índice numérico do array.

Laço while

O laço `while` é um tipo de laço mais flexível do que o laço for, isso por que ele não possui um *inicializador* ou um *incremento* explícito, apenas uma *condição* que deve ser satisfeita para que o bloco de código seja executado. A estrutura do laço `while` é a seguinte:

```
while (condição) {
  // bloco de código a ser repetido
}
```

Por exemplo, suponha que desejamos exibir e então remover cada um dos itens do nosso array de frutas, até que o array esteja vazio. Podemos fazer isso utilizando um laço `while` da seguinte forma:

```
while (frutas.length > 0) {
  let fruta = frutas.pop();
  console.log(fruta);
}
```

Nesse exemplo, o laço `while` é executado enquanto o comprimento do array `frutas` for maior que 0. A cada iteração, removemos o último elemento do array utilizando o método `pop()`, armazenamos esse elemento na

variável `fruta` e exibimos o seu valor no console. O laço é encerrado quando o array `frutas` estiver vazio, ou seja, quando o comprimento do array for igual a 0.

Assim como no laço `for`, é importante ter cuidado para que a condição seja falsa em algum momento, caso contrário o laço se tornará infinito, executando idefinidamente e podendo travar o Navegador ou o NodeJS.

Objetos

Os objetos são estruturas de dados mais complexas do que os arrays, que nos permitem armazenar dados em pares chave-valor. Cada propriedade de um objeto é identificada por uma chave (ou nome), que é única dentro do objeto, e possui um valor associado. Para criar um objeto, utilizamos chaves `{}` e separamos as propriedades por vírgulas. Por exemplo:

```
let pessoa = {
  nome: 'João',
  idade: 30,
  cidade: 'São Paulo'
};
```

Neste exemplo, criamos um objeto chamado `pessoa`, que possui três propriedades: `nome`, `idade` e `cidade`. Cada propriedade é identificada por uma chave (ou nome), como `nome`, `idade` e `cidade`, e possui um valor associado, como `'João'`, `30` e `'São Paulo'`.

Os valores das propriedades de um objeto podem ser de qualquer tipo, como números, strings, objetos, arrays, funções, etc. Por exemplo:

```
let carro = {
  marca: 'Fiat',
  modelo: 'Uno',
  ano: 2010,
  proprietario: {
    nome: 'Maria',
    idade: 25
  },
  revisoes: [2011, 2012, 2013]
};
```

As chaves de um objeto podem ser strings, números ou símbolos. Entretanto, geralmente utilizamos strings como chaves, pois são mais fáceis de lembrar e de acessar. Quando inicializamos um objeto geralmente não precisamos utilizar aspas para delimitar as chaves, exceto quando a chave contiver espaços, caracteres especiais ou for uma palavra reservada do JavaScript.

Acessando Propriedades

Para acessar uma propriedade de um objeto, utilizamos a notação de ponto `.` ou a notação de colchetes `[]`. Por exemplo, para acessar a propriedade `nome` do objeto `pessoa`, podemos fazer:

```
console.log(pessoa.nome); // João
```

Também podemos acessar a propriedade `nome` do objeto `pessoa` utilizando a notação de colchetes `[]`, passando a chave como uma string. Por exemplo:

```
console.log(pessoa['nome']); // João
```

É mais comum o uso da notação de ponto `.`. A notação de colchetes `[]` é útil quando a chave contém espaços, caracteres especiais ou é uma palavra reservada do JavaScript. Por exemplo:

```
let pessoa = {
  'nome completo': 'João Silva',
  'idade': 30,
```

```
  'cidade': 'São Paulo'
};

console.log(pessoa['nome completo']); // João Silva
```

Modificando Propriedades

Para modificar o valor de uma propriedade de um objeto, basta atribuir um novo valor à propriedade. Por exemplo, para alterar a idade da pessoa de 30 para 31 anos, podemos fazer:

```
pessoa.idade = 31;
console.log(pessoa.idade); // 31
```

Podemos ainda criar uma nova propriedade em um objeto, atribuindo um valor a uma chave que não existe. Por exemplo, para adicionar a propriedade `email` ao objeto `pessoa`, podemos fazer:

```
pessoa.email = 'foo@bar.com`;
console.log(pessoa.email); //
```

Para excluir uma propriedade de um objeto, utilizamos o operador `delete`. Por exemplo, para excluir a propriedade `cidade` do objeto `pessoa`, podemos fazer:

```
delete pessoa.cidade;
console.log(pessoa.cidade); // undefined
```

Iterando sobre Propriedades

Para iterar sobre as propriedades de um objeto, podemos utilizar o laço `for...in`, que percorre todas as chaves do objeto. Não confunda o laço `for...IN` com o laço `for...OF`, que percorre os elementos de um array. A estrutura do laço `for...in` é a seguinte:

```
for (let chave in objeto) {
  // bloco de código a ser repetido
}
```

Por exemplo, para exibirmos todas as propriedades do objeto `pessoa` no console, podemos fazer:

```
for (let chave in pessoa) {
  let valor = pessoa[chave];
  console.log(chave + ': ' + valor);
}
```

Nesse exemplo, a variável `chave` assume o valor de cada chave do objeto `pessoa`, e a variável `valor` assume o valor associado a essa chave. Assim, exibimos no console cada chave e seu valor correspondente.

Métodos de Objetos

Os objetos em JavaScript podem conter métodos, que são funções associadas a um objeto. Os métodos são propriedades de um objeto que contêm uma função, e podem ser chamados para executar uma ação ou retornar um valor. Por exemplo, podemos adicionar um método `apresentar` ao objeto `pessoa`, que exibe uma mensagem com o nome e a idade da pessoa:

```
let pessoa = {
  nome: 'João',
  idade: 30,
  apresentar: function() {
    console.log('Olá, meu nome é ' + this.nome + ' e tenho ' + this.idade + ' anos.');
  }
};
```

Nesse exemplo, o método `apresentar` exibe uma mensagem com o nome e a idade da pessoa, utilizando as propriedades `nome` e `idade` do objeto `pessoa`. Para chamar um método de um objeto, utilizamos a notação de ponto `.` seguida do nome do método e dos parênteses `()`. Por exemplo:

```
pessoa.apresentar(); // Olá, meu nome é João e tenho 30 anos.
```

Como qualquer função, os métodos podem receber parâmetros e retornar valores. Por exemplo, podemos adicionar um método `calcularIdade` ao objeto `pessoa`, que recebe um ano como parâmetro e retorna a idade da pessoa em relação a esse ano:

```
let pessoa = {
  nome: 'João',
  anoNascimento: 1990,
  calcularIdade: function(anoAtual) {
    return anoAtual - this.anoNascimento;
  }
};

let idade = pessoa.calcularIdade(2024);
```

Arrays e Objetos em Constantes

Como vimos nas aulas anteriores podemos declarar variáveis em JavaScript utilizando as palavras-chave `var`, `let` e `const`. As variáveis declaradas com `var` e `let` são mutáveis, ou seja, podemos alterar o valor da variável a qualquer momento. Por outro lado, as variáveis declaradas com `const` são imutáveis, ou seja, não podemos alterar o valor da variável após a sua inicialização.

Entretanto, quando lidamos com objetos e arrays em JavaScript, a imutabilidade de uma variável declarada com `const` não impede que possamos modificar as propriedades de um objeto ou os elementos de um array. Nesse caso, o que é imutável é a referência da variável para o array ou objeto, e não o conteúdo do array ou objeto. O exemplo abaixo não gera nenhum tipo de erro:

```
const frutas = ['maçã', 'banana', 'laranja'];
frutas.push('uva');
console.log(frutas); // ['maçã', 'banana', 'laranja', 'uva']
```

Já o seguinte exemplo, que tenta atribuir um novo array à variável `frutas`, gera um erro:

```
const frutas = ['maçã', 'banana', 'laranja'];
frutas = ['uva', 'pera', 'melancia']; // Erro!
```

Desafios

1. Crie uma função chamada calcularMedia que recebe um array de números e retorna a média aritmética desses números.
2. Crie uma função chamada buscarMaior que recebe um array de números e retorna o maior número desse array.