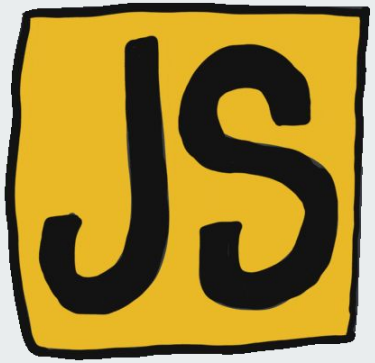
JAVASCRIPT MODERNO: ES6+

Principais recursos adicionados



Versões do JavaScript

- Como já vimos, o JavaScript se tornou um padrão ECMA em 1997
- Seu nome oficial é **ECMAScript**
- As versões do ECMAScript são abreviadas para ES1, ES2, ES3, ES5 e ES6+ (2015)
- Desde 2016, as versões passaram a ser nomeadas com o ano de lançamento (ECMAScript 2016, 2017, 2018, 2019...)
- Em 2015 foi realizada a segunda principal revisão na linguagem (**ES6**), adicionando diversos recursos novos e modernos
- Esse conjunto de recursos possibilitou um rápido avanço na utilização de *frameworks* de desenvolvimento Web, bem como ambientes de execução JavaScript como o Node.js, o que levou o ES6 a ser popularmente conhecido como **JavaScript Moderno**

Novos Recursos no ES6

- Entre os novos recursos adicionados no ES6, destacam-se:
  - as palavras-chave **let** e **const**
  - definição de **Arrow Functions**
  - métodos de iteração em arrays, como **filter()**, **reduce()** e **map()**
  - uso de **template literals**
  - recurso de **destructuring**
  - uso do operador **spread**
  - uso de **classes** e **herança**
  -

Let e Const

- As palavras-reservadas let e const foram incluídas para trabalhar com escopo de bloco
- Declarando uma variável com let nós não podemos fazer re-declaração e só podemos usá-la depois de ser declarada
- Uma variável declarada com const não pode ter uma nova atribuição de valor, porém o que se mantém constante é a referência ao valor

myScript.js

```
let x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

myScript.js

```
const x = 10;
// Here x is 10
{
  const x = 2;
  // Here x is 2
}
// Here x is 10
```

Arrow Functions

- As arrow functions permitem simplificar a escrita de definições de funções
- É uma função anônima
- Não é necessário utilizar a palavra reservada **function**
- Se a função tem um único statement não é necessário utilizar **return** nem mesmo utilizar chaves **{}**

myScript.js

```
// ES5
var x = function(x, y) {
   return x * y;
}

// ES6
const x = (x, y) => x * y;
```

Métodos filter() e map()

- O método **filter()** é útil quando queremos criar um novo array a partir de elementos que satisfazem uma determinada condição em um array existente
- O método **map()** cria um novo array executando uma função em cada elemento de um array existente

myScript.js

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

myScript.js

```
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

Método reduce() e reduceRight()

- O método **reduce()** é útil quando queremos realizar algum processamento nos elementos do array, produzindo um único valor ao final
- O método **reduceRight()** é análogo ao **reduce()**, porém ele itera o array da direita para a esquerda

myScript.js

```
const numbers = [45, 4, 9, 16, 25];
const sum = numbers.reduce(myFunction);

function myFunction(total, value, index,
array) {
  return total + value;
}
```

myScript.js

```
const numbers = [45, 4, 9, 16, 25];
const sum = numbers.reduceRight(myFunction);

function myFunction(total, value, index,
array) {
  return total + value;
}
```

Método reduce() e reduceRight()

- Podemos definir quais argumentos necessitamos. Exemplo: se somente é necessário o valor de cada elemento do array, não precisamos passar os argumentos para índice e nem para o próprio array;
- Além disso, o reduce permite passar um valor inicial para o parâmetro acumulador

myScript.js

```
const numbers = [45, 4, 9, 16, 25];
const sum = numbers.reduce(myFunction,100);

function myFunction(total, value) {
  return total + value;
}
```

Template Literals

- O recurso de **template literals** no JavaScript permite a interpolação de variáveis e expressões dentro de strings
- Com isso, podemos produzir uma string final de forma dinâmica usando interpolação de valores sem precisar recorrer ao operador de concatenação
- Usa-se back-ticks (``) para envolver toda a string, ao invés de aspas simples ou duplas
- Para interpolar uma variável ou expressão a sintaxe é a seguinte: **${...}**

myScript.js

```
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!;
```

Destructuring

- O **destructuring** é um recurso que pode ser usado em arrays e objetos para desestruturar dados agregados em variáveis independentes
- Simplifica a declaração de n variáveis para apenas 1 linha

myScript.js

```
const numbers = [45, 4, 9];
const [n1, n2, n3] = numbers;

console.log(n1); // vai ser impresso 45
```

myScript.js

```
// Declarando e definindo um objeto chamado car
const car = {type:"Fiat", model:"500", color:"white"};

const {type, model, color} = car;
console.log(`O modelo do carro é ${model}`);
```

Spread Operator

- O **spread operator (...)** é um recurso que expande um objeto iterável (como um array) em mais elementos
- Podemos inclusive criar novos objetos a partir da união de objetos existentes

myScript.js

```
const q1 = ["Jan", "Feb", "Mar"];
const q2 = ["Apr", "May", "Jun"];
const q3 = ["Jul", "Aug", "Sep"];
const q4 = ["Oct", "Nov", "May"];

const year = [...q1, ...q2, ...q3, ...q4];
```

myScript.js

```
const carType = {type:"Fiat"};
const carModel = {model:"500"};
const carColor = {type:"white"};

const car = {...carType , ...carModel, ...carColor};
console.log(car);
```

Classes e Herança

- Classes em JavaScript são templates para objetos, onde temos acesso a: construtor, propriedades e métodos

myScript.js

```
class Vehicle {
  constructor(model, year) {
      this.model = model;
      this.year = year;
  }

  age(currentYear){
    return currentYear - this.year;
  }
}
```

Classes e Herança

- Também podemos usar herança no ES6, utilizando a palavra **extends** para referir a classe herdada e enviando as propriedades herdadas via função **super()**

myScript.js

```
class Car extends Vehicle {

  constructor(model, year, is4wd) {
      super(model, year);
      this.is4wd = is4wd;
  }

}
```