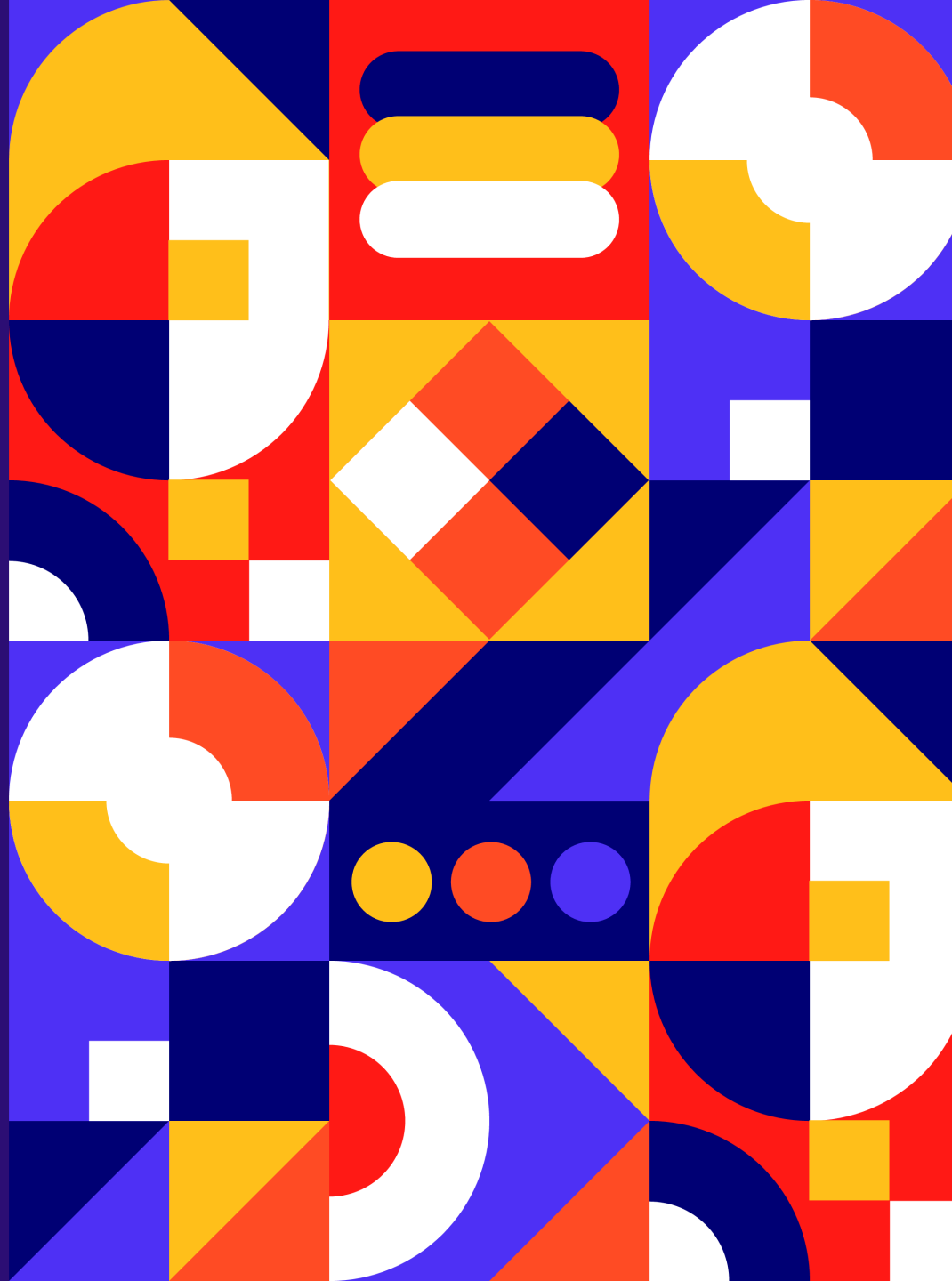
escola britânica de artes criativas & tecnologia

Módulo 25

Recursos ES6+

Gian Souza



MÓDULO 25 | Recursos ES6+

escola britânica de artes criativas & tecnologia

Durante este módulo vamos conhecer as principais novidades que foram introduzidas pelo ECMAScript 6.

Você pode consultar o código escrito durante este módulo por **esse link** do Github.

Gian Souza

ECMAScript

O ECMAScript é uma especificação de linguagem de programação, o JavaScript faz a implementação do ECMAScript.

O JavaScript surgiu em 1995 e foi implementado no navegador Netscape, possibilitando a criação de páginas web mais dinâmicas.

A Microsoft, um pouco depois do lançamento do Internet Explorer criou uma linguagem semelhante ao JavaScript, o JScript.

ECMAScript

Depois desse evento a Netscape submeteu o JavaScript para que fosse criado uma especificação, assim nasce o ECMA-262, que é a especificação do ECMAScript.

O ECMAScript 6 (ES6), foi lançado em 2015 e trouxe muitas novidades, as quais abordaremos nesse módulo.

Babel

Atualmente a maioria dos recursos do ES6 estão implementados nativamente nos navegadores, em suas versões mais recentes, porém nem sempre foi assim, quando o ES6 foi lançado era necessário utilizar o Babel, esta ferramenta faz a tradução de um código JavaScript moderno para um código que os navegadores mais antigos entendem.

Arrow function

No ES6 tivemos a introdução de uma nova forma de escrever funções, o arrow function:

```
const minhaFuncao = (argumento) => {
  console.log(“hello world”);
}
```

Além da escrita ser menos verbosa o contexto da execução de uma arrow function é diferente, considere:

```
const objeto = {
    nome: "gian",
    funcaoArrow: () => { console.log(this) },
    funcao: function() { console.log(this) }
}
```

Arrow function

```
> objeto.funcaoArrow()
  ▶ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
< undefined
> objeto.funcao()
  ▶ {nome: 'gian', funcaoArrow: ƒ, funcao: ƒ}
< undefined
```

Chamando as funções do objeto criado anteriormente teremos esses resultados, sendo que apenas a função escrita do modo convencional terá acesso ao contexto do objeto.

Arrays

As maiores novidades do ES6 estão na manipulação dos arrays, neste contexto tivemos a adição dos métodos filter, map, reduce, foreach, every, some e find.

filter: filtrar os itens do array para atender a um requisito, também chamado de predicado, por exemplo, para filtrar os itens maiores que cinco de um array:

```
[20, 4, 2, 3, 5, 6, 10, 12].filter(function(item) {
  return item > 5;
});
```

Arrays

map: itera pelo array e retorna o array que pode ser modificado, por exemplo, para criar um array com o dobro dos valores de um outro array:

```
const nums = [2, 4, 6];
const dobro = nums.map(function(item) {
  return item * 2;
});
// dobro -> 4, 8, 12
```

Arrays

foreach: itera pelo array, mas não tem nenhum retorno

every: verifica se todos os itens do array satisfazem a uma condição, por exemplo:

```
[2, 3, 4, 5, 6].every(function(item) {
  return item > 4;
})
// retornará false, pois nem todos números são
maiores que 4
```

some: parecido com o every, porém se apenas um item satisfazer a condição o retorno será true.

Arrays

foreach: itera pelo array, mas não tem nenhum retorno

every: verifica se todos os itens do array satisfazem a uma condição, por exemplo:

```
[2, 3, 4, 5, 6].every(function(item) {
  return item > 4;
})
// retornará false, pois nem todos números são
maiores que 4
```

some: parecido com o every, porém se apenas um item satisfazer a condição o retorno será true.

Arrays

find: faz a busca de um item em um array:

```
const alunos = [“gian”, “paulo”, “ana”, “sofia”];
alunos.find(function(item) {
  return item == “ana”;
})
// retornará “ana”
```

Arrays

reduce: itera pelos itens do array e possibilidade retornar a agregação dos itens:

```
const nums = [10, 20, 30];
const soma = nums.reduce(function(total, itemAtual) {
  total += itemAtual;
  return total;
}, 0) // 0 = valor inicial

// soma -> 60
```

Conjuntos Map e Set

Map: é um conjunto de dados chave-valor:

```
const meuMap = new Map();
meuMap.set(“nome”, “gian”)
meuMap.set(“tecnologias”, “html, css,
Javascript”);
```

Conjuntos Map e Set

Map: é um conjunto de dados onde temos apenas o valor e este valor não pode se repetir:

```
const meuSet = new Set();
meuSet.add("gian");
meuSet.add("ana");
MeuSet.add("gian"); // nada irá acontecer
```

Operadores Rest e Spread

Com o operador Rest podemos receber um número indefinido de argumentos em uma função, escrevemos o operador Rest utilizando as reticências (...) antes do argumento:

```
function somarNumeros(...numeros) {}
```

Assim teremos acesso a um array chamado numeros.

Importante: podemos possuir apenas um argumento com o operador Rest na função, e este argumento sempre deverá ser o argumento último da função.

Operadores Rest e Spread

Com o operador Spread podemos distribuir, espalhar os dados de um array ou objeto, sua sintaxe é igual a do operador Rest, utilizando as reticências.

```
const array1 = [1, 2, 3, 4];
const array2 = [...array1, 5, 6, 7, 8];
```

Assim podemos unir o conteúdo de dois arrays.

Operadores Rest e Spread

O seu uso com objetos nos permite copiar propriedades e também criar novos objetos.

```
const carroDoJoao = {
  marca: “vw”,
  cor: “prata”,
}
const carroDaAna = {
  ...carroDoJoao,
  cor: “azul”
}
```

Operadores Rest e Spread

No exemplo anterior criamos um objeto literal “carroDoJoao” e outro objeto “carroDaAna” ambos são da marca VW e sua única diferença é a cor, assim fizemos a cópia dos atributos através do operador Spread e alteramos apenas o atributo que é diferente entre os objetos.

Programação assíncrona

O JavaScript é uma linguagem de programação single-thread, ou seja, não sabe trabalhar em mais de um processador, executando tarefas de forma paralela. Apesar disso no ES6 tivemos a inclusão das Promises, o que nos permite ter um paralelismo no JavaScript.

Orientação a Objetos com ES6

O ECMAScript trouxe algumas novidades na programação orientada a objetos no JavaScript, até então não tínhamos classes no JavaScript, apenas funções construtoras.

Para criar uma classe utilizamos a palavra reservada class:

```
class Pessoa {
}
```

Orientação a Objetos com ES6

Assim não utilizamos mais as funções construtoras, apenas as classes e dentro de uma classe temos acesso ao que chamamos de construtor, uma função que irá construir o objeto, configurando seus atributos:

```
class Funcionario {
  construtor(nome, cargo, salario) {
    this.nome = nome;
    this.cargo = cargo;
    this.salario = salario;
  }
}
```

Orientação a Objetos com ES6

Junto com a adição das classes tivemos a introdução da palavra reservada extends, utilizada para fazer a herança entre classes:

```
class Pessoa {}
class Funcionario extends Pessoa {}

// classe Funcionario é herdeira herda da classe Pessoa
```

Orientação a Objetos com ES6

Também tivemos melhorias na esfera de encapsulamento, podendo criar membros privados de uma forma mais fácil, utilizando o símbolo # antes do nome da propriedade.

```
class Funcionario extends Pessoa {
  #salario = 1000; // só é acessível dentro da própria classe

  atribuiSalario(novoSalario) {
    this.#salario = novoSalario;
  }
}
```

Links úteis

[ECMA-262 Especificação do ECMAScript]

[Github do TC39 – Comitê responsável pelo ECMAScript]

[Babel]

[Can I Use]