53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

RESIDÊNCIA DE

SOFTWARE

CAPACITAR
TREINAR
EMPREGAR

TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana

Revisão JavaScript

Prof. Diego Leite
diego@domob.me

Revisão JavaScript

- var, const e let
- Template Strings
- Operações em Arrays
- Funções Anônimas
- Arrow Functions
- Desestruturação
- Operadores Rest / Spread
- Async / Await

var, const e let

teste_const.js

```
void function(){
	const mensagem = 'valor01';
	console.log(mensagem);

	mensagem = 'valor02';
	console.log(mensagem);
} ();
```

var, const e let

teste_var.js

```
const exibeMensagem = function () {
	if (true) {
		var escopoFuncao = 'escopoFuncao';
		console.log('dentro escopo', escopoFuncao);
	}
	console.log('fora escopo', escopoFuncao);
}

exibeMensagem();
```

var, const e let

teste_let.js

```
const exibeMensagem = function () {
	if (true) {
		let escopoFuncao = 'escopoFuncao';
		console.log('dentro escopo', escopoFuncao);
	}
	console.log('fora escopo', escopoFuncao);
}

exibeMensagem();
```

var, const e let

Qual a diferença observada?

Template Strings

Introduz uma nova forma de se trabalhar com strings;

```
const str = `Hello World`;
console.log(str);
```

```
const str_multilinhas = `linha 1
linha 2`;
console.log(str_multilinhas);
```

Template Strings

Permite a utilização de código JavaScript sem a necessidade de realizar concatenação.

```
const str1 = `Olá ${nome_pessoa}`;
console.log(str1);
// Olá Diego
```

```
const str2= `Soma: ${1 + 1}`;
console.log(str2);
// Soma: 2
```

```
const total = 10;
const str3 = `Total: ${total + 1}`;
console.log(str3);
// Total: 11
```

```
const ativo = true;
const str4= `Conta ativa: ${ativo === true ? 'Sim': 'Não'}`;
console.log(str4);
// Conta ativa: Sim;
```

Operações com arrays

Indo além do

for (var i = 0; i < 10; i++) {...}

Operações com arrays

- forEach: Iterar todos os elementos;
- map: Iterar todos os elementos e fazer algo com seus valores
- filter: Filtrar os elementos dada uma condição;
- find: Encontrar um elemento;

Operações com arrays - forEach

```
var numeros = [1,2,3,4,5];

// forEach

numeros.forEach(function(numero){
    console.log(numero);
});
```

Operações com arrays - map

```
var numeros = [1,2,3,4,5];

// map

const dobro = numeros.map(function(numero) {
      return numero * 2;
});

console.log(dobro); // [2, 4, 6, 8, 10]
```

Operações com arrays - filter

```
var numeros = [1,2,3,4,5];

// filter

var maioresQueTres = numeros.filter(function(numero){
      return numero > 3;
});

console.log(maioresQueTres); // 4, 5
```

Operações com arrays - find

```
var numeros = [1,2,3,4,5];

// find

var tres = numeros.find(function(numero) {
      return numero === 3;
});

console.log(tres); // 3
```

Funções Anônimas

Como o próprio nome já diz, uma função anônima é uma função sem nome.

Pode ser chamada imediatamente após a declaração:

```
(function(){
    console.log(`Executada imediatamente após sua instâncialização`);
})();
```

Funções Anônimas

Pode ser passado parâmetro:

```
const usuario = {
        nome: "João",
        sobrenome: "Silva"
};

(function(){
        console.log(`${usuario.nome} ${usuario.sobrenome}`);
})(usuario);
```

Funções Anônimas

Pode ser usada como argumentos de outras funções

```
setTimeout(function(){
    console.log(`Executar a cada 1 segundo`);
}, 1000);
```

Podemos atribuir uma função anônima em uma variável

```
const anonima = function () {
    console.log(`Função anônima`);
}
```

Arrow Function

O ES6 trouxe uma nova forma de escrever funções, utilizando a sintaxe denominada arrow function, sendo seus principais benefícios:

- São menos verbosas do que as funções tradicionais;
- Seu valor de this é definido à partir das funções onde foram definidas. Ou seja, não é mais necessário fazer bind( ) ou armazenar o estado em that = this;

Arrow Function

função ES5 (tradicional):

```
function soma(a,b){
    return a + b;
}
```

função ES5 (anônima):

```
var soma = function (a,b) {
    return a + b;
}
```

função ES6 (arrow function):

```
const soma = (a,b) => {
    return a + b;
}
```

função ES6 (arrow function):

```
const soma = (a,b) => a + b;
```

Desestruturação

Em JavaScript, uma notação muito comum são objetos, aos quais são representados utilizando JSON, sendo este, uma notação estruturada, similar ao XML no tocante a finalidade, no entanto sua sintaxe é menos verbosa. Um exemplo da notação de um objeto hipotético em javascript:

```
const VideoGame = {
    modelo: 'PlayStation',
    fabricante: 'Sony',
    armazenamento: '1TB',
    controles: 2,
    preco: 'R$ 2000.00'
}
```

Desestruturação

Para acessar atributos deste objeto, por exemplo, o modelo do produto e seu respectivo preço utilizando JavaScript:

```
const modelo = VideoGame.modelo;
const preco = VideoGame.preco;
```

Com Desestruturação:

```
const {modelo, preco} = VideoGame;
```

Operadores Rest / Spread

Em JavaScript, Estes operadores auxiliam a lidar com múltiplos parâmetros dentro de funções. O **Rest** separa os parametros de interesse, e o resto dos parametros em uma única estrutura, enquanto o **Spread** serve para propagar uma estrutura para outra, e realizar possíveis alterações e/ou adições:

```
const Aluno = {
    nome: 'Pedro Sales',
    idade: 19,
    periodo: 7
}
```

```
// REST
const { nome, ...rest } = Aluno;
console.log(nome);
console.log(rest);

// SPREAD
const aluno_ = { ...Aluno, periodo: "setimo", turno: "noite" };
console.log(aluno_);
```

Promise

É utilizado para realizarmos requisições Assíncronas, como uma requisição em uma API externa. Para este propósito temos a Promise. Antes do ES6 a sintaxe de um exemplo seria:

```
new Promise((resolve, reject) => {
    resolve(...);
    reject(...);
})
```

Quem invocar esta promise irá aguardar o retorno da mesma, e esta define seus retornos a partir dos métodos resolve( ) e reject( )

Promise

Vamos simular uma API que demora 3 segundos para enviar uma resposta, com o seguinte trecho de código:

```
const fakeAPI = () =>
new Promise((resolve, reject) => {
        setTimeout(() => {
                resolve('resposta para requisição');
        }, 3000)
})
```

Ao ser chamada esta função fakeAPI, o retorno será apresentado a partir do método encadeado then(response => { }) em caso de sucesso, apresentando o que está dentro do resolve, e em caso de falha, apresentará o conteúdo do reject, a partir do método encadeado catch(err => { })

Promise

```
fakeAPI()
.then(response => {
      console.log(response);
}).catch(err => (
      console.log('error ', err)
));
```

Ao ser chamada esta função fakeAPI, o retorno será apresentado a partir do método encadeado then(response => { }) em caso de sucesso, apresentando o que está dentro do resolve, e em caso de falha, apresentará o conteúdo do reject, a partir do método encadeado catch(err => { })

Async / Await

O async/await é uma nova maneira de tratar Promises, evitando a criação de cascatas de then( ). Por trás continua utilizando Promises, mas elas ficam menos visíveis e verbosas.

```
async function executarPromise() {
    console.log(await fakeAPI());
    console.log(await fakeAPI());
    console.log(await fakeAPI());
}
```

Async / Await

E para lidar com o catch( ), devem ser tratados utilizando o try/catch.

```
async function executarPromise() {
    try {
        const response = await fakeApi();
        console.log(response);
    } catch (err) {
        console.log('Erro:', err);
    }
}
```

Dúvidas?

Atividade Prática

Com base no conteúdo explicado em sala, crie funções em JavaScript que o usuário poderá informar o array, e parâmetros necessários para realizar ações como:

1. Aplicar um desconto percentual em cada item do carrinho de compra;
2. Filtrar todos os itens com preço menor ou igual ao valor passado como parâmetro na função;
3. Verificar se determinado item está no carrinho;
4. Aplicar desconto percentual em todos os itens que tenham um determinado valor abaixo que o especificado

Dica: utilize arrow functions sempre que possível, e os novos métodos do array do ES6.

```
const itens = [
        { nome: "Arroz 1kg", valor: 5.90 },
        { nome: "Feijão Preto 1kg", valor: 8.90 },
        { nome: "Farinha 1kg", valor: 1.50 },
        { nome: "Leite 1l", valor: 4.50 },
        { nome: "Fubá", valor: 2.10 }
]
```

Atividade Prática - Resposta

1. Aplicar um desconto percentual em cada item do carrinho de compra;

```
const desconto = (percentual, itens) => {
        itens.map(item => {
                let valor_desconto = item.valor * (1 - (percentual / 100));
                valor_desconto = parseFloat(valor_desconto).toFixed(2);
                item.valor = valor_desconto;
        })

        return itens;
}

console.log('cesta_desconto', desconto(10, itens));
```

Atividade Prática - Resposta

2. Filtrar todos os itens com preço menor ou igual ao valor passado como parâmetro na função;

```
const filtrar_itens_valor = (valor, itens)  => {
        return itens.filter(function (item) {
                return item.valor <= valor;
        });
}
```

```
var itens_filtrados = filtrar_itens_valor(5.00, itens)

console.log('itens_filtrados', itens_filtrados);
```

Atividade Prática - Resposta

3. Verificar se determinado item está no carrinho;

```
const busca_item = (nome, itens) => {
        return itens.find(item => {
                return item.nome.indexOf(nome) > -1;
        })
}

var item_buscado = busca_item("Fubá", itens);

console.log('item_buscado', item_buscado);
```

Atividade Prática - Resposta

4. Aplicar desconto percentual em todos os itens que tenham um determinado valor abaixo que o especificado;

```
const desconto_condicional = (valor_elegivel, percentual_desconto, itens) => {
        const itens_filtrados = filtrar_itens_valor(valor_elegivel, itens);
        return desconto(percentual_desconto, itens_filtrados);
}

const produtos_com_desconto_condicional = desconto_condicional(5.00, 20, itens);

console.log('desconto_condicional', produtos_com_desconto_condicional);
```

Fontes

https://medium.com/code-prestige/as-funcionalidades-mais-legais-do-es6-atrav%C3%A9s-de-exemplos-983a330ca314

https://www.alura.com.br/artigos/entenda-diferenca-entre-var-let-e-const-no-javascript

https://blog.rocketseat.com.br/javascript-assincrono-async-await/