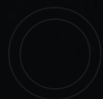

SHIFT

 FIAP



000000



FRONT-END SPECIALIST

DESENVOLVIMENTO WEB COM ANGULAR & REACT

LÓGICA DE PROGRAMAÇÃO

JAVASCRIPT



ALEX SANDER RESENDE DE DEUS

Professor

- Há 25 anos ensinando programação, é um apaixonado por tecnologia. Atualmente é coordenador de cursos na ETEC Albert Einstein do Centro Paula Souza.
- Na FIAP é professor da FIAP School e FIAP Corporate, lecionando C#, Banco de Dados e Desenvolvimento Mobile.

✉ profalex.deus@fiap.com.br

AGENDA

1

AULA 1

Definição e histórico

Fundamentos Básicos da Programação – Tipos de dados, variáveis e estruturas básicas; Usando o GIT

2

AULA 2

Estruturas condicionais, operadores lógicos e switch case

3

AULA 3

Laços de Repetição: For, while,

4

AULA 4

Objetos nativos JavaScript – Date, String, Math

AGENDA

5

AULA 5

Arrays, Filter, Map

Reduce, Splite, forEach, for in, for of

6

AULA 6

Funções: tipos, declarações, escopo, retorno, clousers, call-back, IIFE, factory

7

AULA 7

DOM – Document Object Model

8

AULA 8

DOM – Document Object Model

AGENDA

9

AULA 9

Eventos

Objetos: Criação, prototype, getters, setters

10

AULA 10

Classes JavaScript Assíncrono

AULA 6

FUNÇÕES JAVASCRIPT

```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

RETORNO DE FUNÇÃO

Quando o JavaScript atinge uma instrução `return`, a função para de ser executada.

Se a função foi chamada a partir de uma instrução, o JavaScript "retornará" para executar o código após a instrução de chamada.

O valor de retorno é "`retornado`" de volta para o "`chamador`":

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3      <head>
4          <meta charset="UTF-8">
5          <title>Funções</title>
6      </head>
7      <body>
8          <script type="text/javascript" src="js/script.js"></script>
9      </body>
10 </html>
```

```
1  function paraCelsius(f) {
2      return (5/9) * (f-32);
3  }
4
5  alert(paraCelsius(80));
6  //Chama a função e exibe o resultado
```

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3      <head>
4          <meta charset="UTF-8">
5          <title>Funções</title>
6      </head>
7      <body>
8          <script type="text/javascript" src="js/script.js"></script>
9      </body>
10 </html>
```

```
1  var x = multiplicar(4, 3);
2  alert (x);
3
4  function multiplicar(a, b) {
5      return a * b;
6  }
7  //Neste exemplo a variável recebe o resultado
8  //da função para exibir o valor depois.
```

FUNÇÕES AUTO-INVOCÁVEIS

Esse tipo de construção recebe o nome de função **auto-invocável**. Ou seja, ela executa assim que é **definida**. Em inglês, é também chamada de **Immediately Invoked Function Expression (IIFE)**.

```
1  (function() {  
2      alert('Oi! Ninguém me chamou, mas executei mesmo assim!');  
3      //posso também:  
4      //return 'Oi! Ninguém me chamou, mas executei mesmo assim!'  
5  })();
```


OBJETO ARGUMENTS

Os Argumentos que uma função recebe são guardados em um **objeto do tipo array** chamado **arguments**. Quando você cria uma função, ela já vem com esse objeto embutido.

Então você pode usar diretamente esse objeto dentro de uma função para **manipular os argumentos que ela recebe**.

```
1 function frutas() {
2     let texto="";
3     for (var i = 0; i < arguments.length; i++) {
4         texto += (arguments[i] + ",");
5     }
6     alert(texto);
7 }
8
9 frutas('Laranja', 'Mamão', 'Abacaxi', 'Melão', 'Banana', 'Maçã', 'Quiuí');
```

OPERADOR REST

A partir do **ES6**, surgiu o operador **rest**. Ele é representado por **reticências** Esse operador, quando colocado antes do último parâmetro nomeado de uma função, **coleta todos os argumentos passados para esse parâmetro e coloca-os em um array**.

No exemplo abaixo, o rest coletará, do **segundo argumento** em diante. Pois o primeiro, foi mapeado para o parâmetro **a**.

```
1  function coleta(a, ...b) {  
2      alert(a);  
3      alert(b);  
4  }  
5  
6  coleta(1, 'teste', true, 'sim', 20);
```

PARÂMETRO PADRÃO

Outra novidade, surgida no mesmo pacote de especificações **ES6**, foi a dos parâmetros padrão. Vamos entender a utilidade deles.

Se eu declarar um parâmetro em uma função e não atribuir valor a ele, seu valor padrão será **undefined**.

Usando parâmetro padrão, ao não passar um argumento, **o padrão será aplicado**.

E passando um argumento, ele será atribuído **normalmente** ao parâmetro.

```
1  function retorna(a=2) {  
2      |    return a;  
3  }  
4  //Passando o parâmetro a função  
5  //retornará o valor passado  
6  alert(retorna(3));  
7  //Sem passar o parâmetro, a função  
8  //retornará 2, que é o valor padrão  
9  alert(retorna());
```

OUTROS PARÂMETROS

Acabamos de ver como invocar funções passando como argumentos, valores **primitivos**. Mas, como funções são especiais, podem também **receber objetos e outras funções como argumentos**.

Posso criar objetos separadamente, e passá-los depois para dentro da função.

```
1  let pessoa = {  
2    nome: 'Gabriel',  
3    idade: 20  
4  };  
5  
6  function mostra(argumento) {  
7    alert (`${argumento.nome}, ${argumento.idade}`);  
8  };  
9  
10 mostra(pessoa);
```


ESCOPO DE FUNÇÕES

Escopos são **limitadores** de acesso em programação, e podem ser utilizados para estabelecer, por exemplo, que determinados dados **não devem estar visíveis para além das fronteiras** delimitadas por ele.

Funções criam escopo e, quando invocadas, retornam para o mundo externo, apenas aquilo que estiver indicado através da instrução **return**.

```
1 function minhaFuncao() {  
2     let minhaVariavel = 'Variável no escopo de função';  
3     return minhaVariavel;  
4 }  
5 alert(minhaFuncao()); //Acessando a função a variável é retornada e exibida  
6 alert(minhaVariavel); //Mas aqui não, pois a variável fica encapsulada na função
```

Por outro lado, ao declarar uma variável dentro da função, **sem utilizar a palavra-chave var ou let**, ela será criada no escopo global. Vamos para um exemplo, que é bem mais fácil de entender.

```
1  function minhaFuncao() {  
2      minhaVariavel = 'Variável no escopo de função';  
3      return minhaVariavel;  
4  }  
5  alert(minhaFuncao()); //Acessando a função a variável é criada  
6  alert(minhaVariavel); //E agora posso acessar diretamente
```



MOMENTO **HANDS ON**


LEMBRE-SE DE VERSIONAR
SEUS **PROJETOS NO GIT**

BIBLIOGRAFIA

- <https://developer.mozilla.org/pt-BR/docs/Aprender/JavaScript>
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Math
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array
- https://developer.mozilla.org/pt-BR/docs/DOM/Referencia_do_DOM
- <https://www.w3schools.com/>
- <http://docplayer.com.br/17393758-Javascript-eventos-e-objetos-nativos.html>
- <https://www.adalgisa-souza.appspot.com/javascript/>

OBRIGADO

 profalex.deus@fiap.com.br

 /alexsanderresende

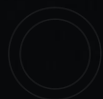
FIAP

Copyright © 2020 | Alex Sander Resende de Deus

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento,
é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

SHIFT

 FIAP



000000