

- Material disponível em: <https://github.com/DevilAnseSenior/Introdu-JavaScript>

# JavaScript

---

- Linguagem de programação interpretada e estruturada;
  - Script de alto nível e tipagem dinâmica fraca;
  - Multi-paradigma(Protótipos, orientada a objetos, imperativo e funcional);
  - Uma das três tecnologias WWW;
  - Criação de páginas Web Interativas;
- 

A large, bold, black 'JS' logo is centered on a solid yellow rectangular background. The letters are thick and stylized, with the 'J' having a curved bottom and the 'S' being a simple, rounded shape.

# Primeiros passos com JavaScript

---

- É uma linguagem usada para adicionar interatividade ao seu site(Ex.: Jogos, respostas quando botões são pressionados ou dados inseridos em formulários, estilo dinâmico, animações).
- 

## Um exemplo "Olá mundo"

---

- Para começar, veja como adicionar algum JS básico a sua página, criando um "olá mundo!".
- 

1. **Primeiro** crie um arquivo chamado index.html e escreva o seguinte trecho de código:

```
<html>
  <head>
    <!--Configurações iniciais-->
    <title>Exemplo 1</title>
  </head>
  <body>
    <!--Estrutura do site-->

    <script src="scripts/main.js"></script>
  </body>
</html>
```

2. **Nesta pasta crie uma pasta chamada scripts;**
3. **Dentro da pasta crie um arquivo chamado main.js e escreva a instrução a seguir:**

```
console.log('Olá mundo!!!');
```

4. **Para executar o código, abra o navegador use o comando CTRL + o e abra o arquivo index.js, na página principal abra as opções de desenvolvedor e verique a saída no console;**
- **O que aconteceu?**
-

- Basicamente fizemos com que uma mensagem fosse impressa no console.

## Curso intensivo de fundamentos da linguagem: Sintaxe e tipos

---

- Vamos explicar alguns dos principais recursos do JS, para dar um melhor entendimento de como tudo funciona.

### Declarações

---

- Existem três tipos de declarações em JavaScript.
  - **var**
    - Declara uma variável, opcionalmente, inicializando-a com um valor;
  - **let**
    - Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor;
  - **const**
    - Declara uma constante de escopo de bloco, apenas de leitura.

### Variáveis

---

- São espaços na memória do computador onde são armazenados dados.
- Tem nomes simbólicos para valores na aplicação. Esse nomes obedecem determinadas regras:
  1. Deve começar com uma letra, underline(\_) ou cifrão(\$);
  2. Os caracteres subsequentes podem ser numeros (0 - 9);
  3. Letras maiúsculas são diferidas das minúsculas.
- Variáveis são iniciadas com as palavras-chaves **var**, **let** ou **const** seguida por qualquer nome que você queira chamá-la:

```
var nome;  
const PI;  
var idade;  
let minhaVariavel;
```

Nota: Ponto-e-virgula no final da linha indica onde uma instrução termina, colocá-las é uma boa prática;

Nota: JS é case sensitive, ou seja, 'minhaVariavel' é diferente de 'minhavariavel'.

- **Depois de declarar uma váriavel, você pode dar um valor a ela:**

```
var nome = 'Bob';
```

- **Se quiser é possível fazer as duas operações na mesma linha:**

```
var genero = 'Masculino';
```

- **Você pode retornar o valor chamando a váriavel pelo nome:**

```
nome;
```

- **Depois de dar um valor a variável, também é possível mudá-lo:**

```
var nome = 'Bob';  
nome = 'Steve';
```

**Note que as variáveis tem diferentes tipos de dados:**

Variáveis	Explicação	Exemplo
String	Sequência de texto é conhecida como string. Para mostrar que o valor é uma String, deve ser envolvida entre aspas.	var nome = 'Bob';

Variáveis	Explicação	Exemplo
Number	Um número. Números não tem aspas.	var num = 10;
Boolean	Um valor verdadeiro ou falso. As palavras 'true' e 'false' são reservadas em JS e não precisam de aspas.	var bool = true;
Array	Uma estrutura que permite armazenar vários valores em uma única variável.	var pessoa = [1, 'Bob', 10] Cada item pode ser acessado: pessoa[0], pessoa[1], etc.
Object	Basicamente, qualquer coisa. Em JS tudo é objeto e pode ser armazenado em uma variável. Tenha isso em mente enquanto aprende.	var titulo = document.querySelector('h1');

- **Variáveis são necessárias para qualquer coisa interessante na programação. Se os valores não podessem mudar, nada dinâmico seria criado.**
- **JavaScript é uma linguagem fortemente tipada. Isso significa que você não precisa especificar tipo de dado de uma variável quando declará-la;**
- **São convertidos automaticamente conforme a necessidade, por exemplo:**

```
var answer = 42;
```

- **E depois, posso atribuir uma string para a mesma variável, por exemplo:**

```
answer = "Obrigado pelos peixes...";
```

## Escopo de variável

- **Quando uma variável é declarada fora de qualquer função, ela é chamada de variável global;**
- **Quando declarada dentro de uma função, é considerada variável local.**

## Comentários

- **É possível colocar comentários em códigos JS:**

```
/*  
    Tudo no meio é um comentário.  
*/
```

- **Se o seu comentário não tiver apenas uma linha, coloque dessa maneira:**

```
// Isto é um comentário
```

## Operadores

---

- **Simbolo matemática que produz um resultado baseado em dois valores;**
- **Vejamos alguns deles:**

Operador	Explicação	Simbolo(s)	Exemplo
adição/concatenação	Usado para somar dois numeros ou juntar duas strings	+	6 + 9; "Olá " + "Mundo!"
subtrair, multiplicar, dividir	Fazem exatamente o que você espera que eles façam na matemática básica.	-, *, /	9 - 3; 8 * 2; 9 / 3;
operador de atribuição	Já vimos, ela associa um valor a uma variável.	=	var nome = 'Bob';
operador de igualdade	Faz um teste para ver se dois valores são iguais um ao outro, retornando um resultado true/false (booleano)	===	var numero = 3; numero === 4;
negação, não igual(diferente)	Retorna o valor lógico oposto do sinal; transforma um true em false, etc. Quando usado junto com o operador de igualdade, o operador de negação testa se os valores são diferentes.	!, !==	"Não igual" dá basicamente o mesmo resultado da sintaxe diferente. Aqui estamos testando "É numero NÃO é igual a 3". Isso retorna false porque numero É igual a 3. var numero = 3; numero !== 3;

- Há vários outros operadores para explorar, mas por enquanto esse são suficientes.
- Em expressões envolvendo valores numérico e string com o operador +, JavaScript converte valores numéricos para string. Ex.:

```
x = "A resposta é " + 42;  
y = 42 + " é a resposta.";
```

- Nas declarações envolvendo outros operdores, JS não converte em valores numéricos para string. Ex.:

```
"37" - 7;  
"37" + 7;
```

## Convertendo strings para números

---

- No caso de um valor que representa um número está armazenado na memória como uma string, existem métodos para conversão.

- parseInt()

- Irá retornar apenas números inteiros, uso restrito para casas decimais.

- parseFloat()

- Irá retornar um número real, restrito a números sem parte decimal.

- Um método alternativo de conversão de um número em forma de string é com o operador + (operador soma):

```
"1.1" + "1.1" = "1.11.1"
(+ "1.1") + (+ "1.1") = 2.2
// Nota: Parênteses usados para deixar legível, ele não é requerido.
```

## Controle de Fluxo e Manipulação de Erro

---

### Declaração em bloco

---

- Utilizada para agrupar declarações, este bloco é delimitado por chaves:

```
{
  declaracao_1;
  declaracao_2;
  .
  .
  .
  declaracao_n;
}
```

### Exemplo

```
var x = 1;
{
  var x = 2;
}
console.log(x);
```

- Este código exibe 2 pois a declaração de var x antes do bloco.

## Condicionais

---



- Permite testar se uma expressão retorna verdadeiro ou não;
- Executa um código alternativo, dependendo da situação;
- Uma forma comum de condicional é a instrução if... else. Declarada da seguinte maneira:

```
if (condicao) {  
    declaracao_1;  
} else {  
    declaracao_2;  
}
```

- Onde condicao pode ser qualquer expressão que seja avaliada como verdadeira ou falso. Ex.:

```
var sorvete = 'chocolate';  
if (sorvete === 'chocolate') {  
    alert('Gosto desse sabor');  
} else {  
    alert('Bom, mas o meu sabor favorito é chocolate.');
```

- A expressão dentro do if (...) é o teste - ela usa o operador de igualdade (como descrito anteriormente) para comparar a variável sorvete com a string chocolate para ver se elas são iguais;
- Se a comparação retorna true, o primeiro bloco é executado;
- Se for falsa, o primeiro bloco é ignorado e o segundo executado.
- Também é possível combinar declarações utilizando else if para obter várias condições testadas em sequência. Ex.:

```
if (condicao) {  
    declaracao_1;  
} else if (condicao_2) {  
    declaracao_2;  
} else if (condicao_n) {  
    declaracao_n;  
} else {  
    declaracao_final;  
}
```

- **Aqui conseguimos visualizar claramente a declaração de blocos. Em geral, é uma boa prática prática, especialmente ao aninhar if's:**

```
if(condicao) {  
    declaracao_1_executada_se_condicao_for_verdadeira;  
    declaracao_2_executada_se_condicao_for_verdadeira;  
} else {  
    declaracao_3_executada_se_condicao_for_falsa;  
    declaracao_4_executada_se_condicao_for_falsa;  
}
```

- **Recomenda-se não utilizar atribuições simples em uma expressão condicional porque o símbolo de atribuição poderia ser confundido com o de igualdade. Por exemplo, não utilize o seguinte código:**

```
if (x = y) {  
    /* faça a coisa certa desgraçado */  
}
```

- **Caso tenha que utilizar uma atribuição em uma expressão condicional, uma prática comum é colocar parênteses adicionais em volta da atribuição. Por exemplo:**

```
if((x = y)) {  
    /*Faça a coisa certa*/  
}
```

## **Valores avaliados como falsos**

- **Os seguintes valores são avaliados como falsos:**
  - false;
  - undefined;
  - null;
  - 0;
  - NaN;
  - string vazia("");
- **Todos os outros valores, incluindo todos os objetos, são avaliados como verdadeiros quando passados para uma declaração condicional.**

# Declaração switch

- Permite que um programa avalie uma expressão e tente associar a um caso(case).
- Se o correspondente é encontrado, a operação associada é executada.
- Vejamos uma declaração switch:

```
switch (expressao) {  
    case rotulo_1:  
        declaracoes_1  
        [break;]  
    case rotulo_2:  
        declaracoes_2  
        [break;]  
    ...  
    default:  
        declaracoes_padrao  
        [break;]  
}
```

- O programa primeiro procura um case com o rótulo correspondente;
- Se nenhum correspondente é encontrado, o programa procura pela clausula opcional default;
- A instrução break associada a cada clausula, garante que o programa sairá do switch assim que a correspondente for executada;

## Exemplo

- No exemplo a seguir, se tipofruta for avaliada como "Banana", o programa faz a correspondência do valor com case "Banana" e executa a declaração associada. Quando o break é encontrado o programa termina.

```
switch (tipofruta) {  
    case "Laranja":  
        console.log("O quilo da laranja está R$0,59.<br>");  
        break;  
    case "Maçã":  
        console.log("O quilo da maçã está R$0,32.<br>");  
        break;  
    case "Banana":  
        console.log("O quilo da banana está R$0,48.<br>");  
        break;  
    case "Cereja":  
        console.log("O quilo da cereja está R$3,00.<br>");  
}
```

```

        break;
    case "Manga":
        console.log("O quilo da manga está R$0,56.<br>");
        break;
    case "Mamão":
        console.log("O quilo do mamão está R$2,23.<br>");
        break;
    default:
        console.log("Desculpe, não temos " + tipofruta + "<br>");
    }
    console.log("Gostaria de mais alguma coisa?<br>");

```

## Declaração e manipulação de Error(Erros)

- Você pode chamar uma exceção usando a declaração throw e manipulá-la usando try...catch.

### Tipos de exceções

- Praticamente pode-se utilizar throw em qualquer objeto JS. Todavia, nem todos os objetos ativados throw são igualmente criados.

### Declaração throw

- Use a declaração throw para lançar uma exceção. Quando lançada, devemos especificar a expressão que conterá o valor a ser lançado:

```
throw expressão;
```

- É possível lançar qualquer tipo de expressão. O código a seguir lança varias exceções de tipos diferentes:

```

throw "Error2"; //tipo string
throw 42; //tipo numérico
throw true; //tipo booleano
throw {toString: function() { return "Eu dou um objeto"; } };

```

**Nota:** É possível especificar um objeto quando se lança uma expressão. É possível especificar essas propriedades de um objeto no bloco catch. O exemplo a seguir cria um objeto myUserException do tipo userException e o usa em uma declaração throw.

```

// Criando um objeto do tipo UserException
function UserException(mensagem) {
    this.mensagem = mensagem;
    this.nome = "UserException";
}

// Realiza a conversão da exceção para uma string adequada quando
// usada como uma string.
// (ex. pelo console erro)

```

```

    UserException prototype.toString = function() {
        return this.name + ': ' + this.message + '';
    }

    //Cria uma instância de um tipo de objeto e lança ela
    throw new UserException("Valor muito alto");

```

## Declaração try...catch

- A declaração try...catch com um bloco de declarações em try, e especifica uma ou mais respostas para uma exceção lançada. Se uma exceção é lançada, a declaração try...catch pegá-a.
- O exemplo a seguir usa a declaração try...catch. O exemplo chama uma função que recupera o nome de um mês no array com base no valor passado para a função. Se o valor não corresponde ao número de um mês (1-12), uma exceção é lançada com o valor "InvalidMonthNo" e as declarações no bloco catch define a variável monthName para unknown.

```

function getMonthName(mo) {
    mo = mo - 1; // Ajusta o número do mês para o índice do array (1
    = Jan, 12 = Dez)
    var months = ["Jan", "Fev", "Mar", "Abr", "May", "Jun", "Jul",
    "Aug", "Sep", "Oct", "Nov", "Dez"];
    if (months[mo]) {
        return months[mo];
    } else {
        throw "InvalidMonthNo"; //lança uma palavra-chave aqui usada.
    }
}

try { // statements to try
    monthName = getMonthName(myMonth); // função poderia lançar
    exceção
}
catch (e) {
    monthName = "unknow";
    logMyErrors(e); // passa a exceção para o manipulador de erro ->
    sua função local.
}

```

## O bloco catch

- Você pode usar o bloco catch para lidar com todas as exceções que podem ser geradas no bloco try.

```

catch (catchID) {
    declaracoes
}

```

- O bloco catch é especificado por um identificador, que contem a especificação dada pelo throw; Esse identificador contem informações da exceção lançada. O identificador funciona enquanto o bloco catch está em execução.
- Por exemplo, o seguinte código lança uma exceção. Quando a exceção ocorre, o controle é transferido para o bloco catch.

```
try {  
    throw "MyException"; //Lança uma exceção  
}  
catch(e) {  
    // declarações de lidar com as exceções  
    logMyErrors(e); // passar a exceção para o manipulador de erro  
}
```

## O bloco finally

- O finally contém instruções para executar após os blocos try e catch, mas antes das declarações seguinte a declaração try...catch. O bloco finally é executado com ou sem o lançamento de exceção. Se uma exceção é lançada, a declaração do bloco finally executa, mesmo que nenhum catch seja processado.

```
openMyFile();  
try {  
    writeMyFile(theData); // Isso pode lançar um erro  
} catch(e) {  
    handleError(e); // Se temos um erro temos que lidar com ele  
} finally {  
    closeMyFile(); //Sempre feche o recurso  
}
```

- Se o bloco finally retorna um valor, este valor se torna toda a entrada try-catch-finally, independente de quaisquer declarações:

```
function f() {  
    try {  
        console.log(0);  
        throw "bogus";  
    } catch(e) {  
        console.log(1);  
        return true; // Essa declaração de retorno é suspensa  
                    // até que o bloco finally seja concluído  
        console.log(2); // não executa  
    } finally {  
        console.log(3);  
        return false; // substitui o return anterior  
    }  
}
```

```

        console.log(4); // não executa
    }
    // "return false" é executado agora
    console.log(5); // não executa
}
f(); // exibe 0, 1, 3; retorna false

```

- Substituições de valores de retorno pelo bloco finally também se aplica a exceções lançadas ou re-lançadas dentro do bloco catch:

```

function f() {
    try {
        throw "bogus";
    } catch(e) {
        console.log('captura interior "falso"');
        throw e; // essa instrução throw é suspensa até
                // que o bloco finally seja concluído
    } finally {
        return false; // substitui "throw" anterior
    }
    // "return false" é executado agora
}

try {
    f();
} catch(e) {
    // isto nunca é executado porque o throw dentro
    // do catch é substituído
    // pelo return no finally
    console.log('captura exterior "falso"');
}

// SAIDA
// captura interior "falso"

```

- Também é possível aninhar declarações try...catch.

## Loops

- Permite que uma parte do código continue executando repetidamente, até que determinada condição seja satisfeita. Vamos testar o seguinte:

```

for (var i = 1 ; i < 21; i++) {
    console.log(i);
}

```

- O que aconteceu? Os números de 1 a 20 foram exibidos no seu console. Isso acontece por causa do loop. Um loop for utiliza a inserção de três valores (argumentos):

1. **Um valor inicial:** Nesse caso estamos iniciando a contagem com 1, mas pode ser qualquer numero da minha escolha;
2. **Uma condição de saída:** Aqui nos especificamos  $i < 21$  - O loop irá continuar até que  $i$  não seja mais menor que 21.
3. **Incremento:** Especificado como  $i++$ . Significa "Adicione 1 à  $i$ ".

## Funções

---

- Maneira de encapsular funcionalidades reutilizaveis;
- Apresentada a necessidade, posso chamar a função pelo nome, ao invés de, reescrever o código inteiro;
- Já vimos um exemplo:

```
alert('hello');
```

- Essa função é pré-definida nos navegadores para ser usada quando quiser;
- Se ver alguma coisa com um nome de variável, mas com parênteses - () - depois, provavelmente é uma função;
- Geralmente tem argumentos;
- Esses colocados dentro dos parênteses e separados por vírgula.
- Também é possível definirmos nossas próprias funções, façamos uma que multiplica dois numeros:

```
function multiplicacao(num1, num2) {  
    var resultado = num1 * num2;  
    return resultado;  
}
```

- Tente executar essa função no console e teste com vários argumentos. Ex.:

```
Próxima  
multiplicacao(4,7);  
multiplicacao(20,20);  
multiplicacao(0.5,3);
```

**Nota:** A instrução `return` diz ao navegador que retorne a variável `resultado` da função. Isso é necessário pois, variáveis definidas dentro de funções só estão disponíveis dentro de funções.