



Imersão Desenvolvimento Web

Modulo I - Dia 3

Taylane Brandão

Renan Verissimo

Laion Luiz

Indice:




- **Intro JavaScript I**
- **Intro JavaScript II**
- **DOM**

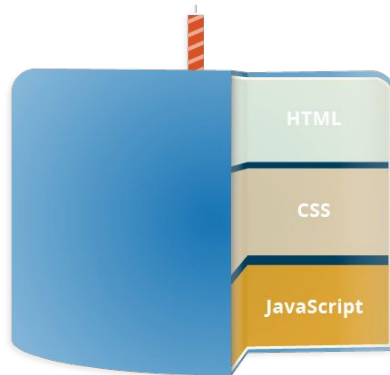
Intro JavaScript I

Parte I

—

O que é JavaScript ?

- Uma das linguagens de programação **mais utilizadas** da internet.
- Permite a criação de conteúdo **dinâmico** na Web.
- Pode ser usada para a criação de:
 - **Games** 
 - **Mobile** 
 - **Desktop** 



Comentários

```
//Isto é um comentário
```

```
/*Isto é um comentário  
com multiplas linhas*/
```

- Quando queremos anotar algo que **não seja código**.
- O computador **ignora** comentários.



Variável

- São caixinhas onde podemos **guardar** dados.

Declarar
`var caixa;`



Atribuir
`var caixa = "valor";`

Variável



- **Nomes** podem conter letras, números, _ e \$.
- **Nomes** são *case sensitive*. `var caixa` \neq `var Caixa`
- Palavras **reservadas do JS** não podem ser usadas.

Atribuição

- Uma variável **recebe** o valor **atribuído** a ela.
- Para atribuir um valor para uma variável **já declarada**, basta omitir o **var**.

```
var caixa = "valor";  
caixa = "outrovalor";
```

Output



- São formas de **exibir** os dados.
- As mais simples são o **console.log(valor)** e **window.alert(valor)**.
- O **primeiro** exibe os valores no **terminal** onde está sendo **executado**.
- O segundo cria uma **janela de alerta** numa **página HTML**.

Console

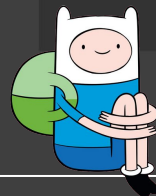


- Conjunto de ferramentas para *debugar* seu código.
- Debugar -> **Debug** -> **Depurar** -> Analisar e ajustar bugs no código.
- O browser abre as **ferramentas de desenvolvedor** com **F12**.
- O **terminal** do navegador fica na aba **console**.

Executando o Código

- Depois de adicionar variáveis, atribuições, comentários, é hora de **rodar** esse código bonito!
- Para **JavaScript**, é necessário usar um arquivo com a **extensão js**.
- No **terminal** referente a pasta onde está o arquivo JS, use o **comando** **node** **nomedoarquivo.js**.
- **E pronto!** Seu código foi executado.

É hora de codar!



- Crie um arquivo com extensão **JS**.
- Crie uma **variável** e **atribua** o valor “Hello World”.
- Use o **console.log** para exibir o valor da **variável** criada.
- Execute o arquivo e veja o **resultado**.

Números



- Imagina um **trabalho** no qual é preciso fazer um **mesmo** cálculo muitas e muitas **vezes**.
- Além de ser o famoso **corneo job**, é muito **suscetível a erros**.
- Então por que não deixar **o computador** fazer isso para a gente?
- **Bora ver!**

Fazendo uns cálculos

- **Variável** = Número Operador Número;
- **Adição**: `var a = 1 + 2; [a = 3]`
- **Subtração**: `var s = 10 - 2; [s = 8]`
- **Multiplicação**: `var m = 5 * 3; [m = 15]`
- **Divisão**: `var d = 30 / 3; [d = 10]`



Precedência

- As regras da **aritmética** se aplicam na programação da mesma forma.
- Tanto **operadores** quanto **parênteses** influenciam na **execução**.
- `var a = 10 + 5 * 2; [a = 20]`
- `var b = (10 + 5) * 2; [b = 30]`

Acredite, você vai usar: ++ e --

- Em algum momento da sua vida como dev, você precisará **aumentar** ou **diminuir** o valor de uma variável em **1**.
- Ao invés de usar `a = a + 1`; use `a++`;
- Ao invés de usar `b = b - 1`; use `b--`;

Acredite, você vai usar: resto

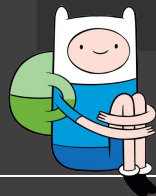
- Em algum outro momento da sua vida como dev, você precisará do **resto** de uma **divisão**.
- Para isso usamos o operador **%**.
- `var d = 5 / 3; [d = 2]`

Atribuição Rápida



- Uma forma de atribuir valores a variáveis já **existentes** é usando a **atribuição rápida**.
- Consiste em **remover** a variável repetida, **unir** e **inverter** a ordem dos operadores: `a = a + 1;` -> `a += 1;`
- Funciona com as operações básicas: `b = b / 3;` -> `b /= 3;`

É hora de codar!



- Imagine o seguinte cálculo: **Se 20 mais 10, dividido por 5, menos 1, vezes 7 for dividido por 3. Qual é o resto da divisão mais 1?**
- Escreva um código que **resolva** o cálculo acima e **mostre** o resultado.

Intro JavaScript II

Parte II

—

Tipos de Dados

- Agora que você sabe o que são variáveis, é hora de falar sobre os **tipos** que podem assumir: **numérico**, **booleano**, **string** e vários outros.
- Os tipos servem para diferenciar **comportamentos** no código.
- `var d = 1 + 1; [d = 2]` - O resultado é do tipo **numérico**.
- `var e = "Olá" + " " + "cara!"; [e = Olá cara!]`

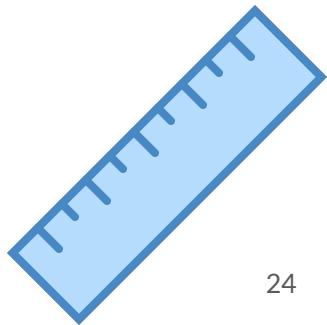
O resultado é do tipo **string**. Mas o que é isso?

String

- São variáveis que guardam informações do tipo **texto**.
- São declaradas usando **""** (aspas duplas) ou **"** (aspas simples).
- **var** Jukera = **"0 que você tá fazendo menor?!!!"**;
- **var** Gratis = **'Eu tô tiltado cara?'**;

String.length

- Para saber o **tamanho** de uma string, usamos o **length**.
- `var a = "É0Q?"; [a.length = 4]`
- Espaços também entram para a contagem.
- `var b = "É0Q MALUKO?"; [b.length = 11]`



Aspas dentro de Aspas

- Caso queira usar **aspas duplas** dentro de uma string declarada com **aspas duplas**, é preciso usar a \ (contrabarra) antes de cada símbolo.
- `var a = "Por que está usando \"aspas\"?";`
- Também é possível usar **aspas simples** na declaração e assim usar as **aspas duplas** normalmente dentro da string.
- `var b = 'Por que está usando "aspas"?';`

Variáveis dentro da String

- Para adicionar variáveis dentro de strings, usamos o **operador +**.
- `var a = 2;`
- `var b = 4;`
- `var c = a + " mais " + a + " é " + b + ".";`

`// c = 2 mais 2 é 4.`

Template Strings



- Uma forma **otimizada** da string comum. Deixa o código mais **legível**, além de aceitar **várias linhas** e **interpolação**.
- São declaradas usando o `` (acento grave).
- `var a = `Fala tu!`;`

Template Strings

- Múltiplas linhas:
- `var a = "Não consigo ler nada.";`
- `var b = `Não consigo
ler nada.`;`

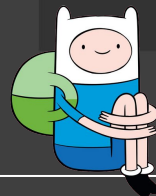
Template Strings

- Interpolação:
 - `var a = "dog";`
 - `var b = "What the" + a + "doin'?"`;
 - `var c = `What the ${a} doin'?``;
- `// b,c = What the dog doin'?`

Template Strings

- Legibilidade:
 - `var a = a + " mais " + a + " é " + b + ".";`
 - `var b = `${a} mais ${a} é ${b}.`;`
- `// a,c = 2 mais 2 é 4.`

É hora de codar!



- Calcule o **resto da divisão 5/4** e mostre o resultado dentro da frase:
O resto é resultado.

Mais de 8000!?



- Dado um número **x**, você precisa **exibir** o **resto** da divisão dele por **y**.
- Usando o **último exercício** como base:

```
var resto = 5 % 4;  
console.log(`0 resto é ${resto}.`);
```

- Agora ao invés de apenas **1 par de entradas**, são **8000**. Será que é preciso escrever o mesmo código outras **7999 vezes** e ir trocando os valores em cada um?

Função $f(x)$

- Para isso, nós usamos uma **função**: execução de um **pedaço** de código que seja **reutilizável**.

```
function calculaResto(x,y) {  
    var resto = x % y;  
    console.log(`0 resto é ${resto}.`);  
}
```

Usando uma Função



- Para usar uma função, ela deve ser chamada por seu nome seguido de parênteses e seus parâmetros.

```
function calculaResto(x,y) {  
    var resto = x % y;  
    console.log(`0 resto é ${resto}.`);  
}
```

```
calculaResto(5,4);  
calculaResto(53,7);
```

Resultado da Função



- No terminal, você receberá os seguintes outputs:

0 resto é 1.

0 resto é 4.

- Perceba que **uma função** conseguiu resolver problemas **pares de valores diferentes**.
- **Não foi preciso** de repetir todo o código da função e alterar as variáveis diversas vezes.

Estrutura de uma Função

- Declaração de uma função: **function** nomeDaFuncao() {}.
- A palavra **function** identifica que é uma **função**.
- O **nomeDaFuncao** tem as mesmas propriedades das **variáveis**.
- **()** indicam os **parâmetros** que a função utiliza.
- **{ }** é onde fica o código que será **executado** pela função.

Parâmetros de uma Função

- **Parâmetros** funcionam como **variáveis** dentro da função.

```
calculaResto(5,4) -> x = 5, y = 4;
```

```
calculaResto(x,y) {  
    var resto = 5 % 4;  
    console.log(`O resto é ${resto}.`);  
}
```

Parâmetros de uma Função

- Uma função pode não ter parâmetros.

```
function imprimeMsg() {  
    console.log("Não consigo ler nada.");  
}
```

```
imprimeMsg();
```

Output: Não consigo ler nada.

Retorno de uma Função

- No geral, as funções são projetadas para **retornar dados**. Isso é feito através do **return**.
- Ter um retorno faz com que funções possam ser **atribuídas** e não apenas executadas de forma independente.
- Sempre que o **return** é lido, **encerra** a execução da função e continua de **onde parou** no código.

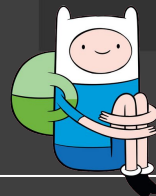
Retorno de uma Função

- Dessa forma, a função **realiza os cálculos e retorna o resultado**. E **outra parte do código** fica responsável em **exibir**.

```
function calculaResto(x,y) {  
    return x % y;  
}
```

```
var res = calculaResto(5,4);  
console.log(res);
```

É hora de codar!



- Imagine o seguinte cálculo: **Se um valor mais 10, dividido por 5, menos 1, vezes 7 for dividido por 3. Qual é o resto da divisão mais 1?**
- Crie uma função que **resolva** o cálculo acima e **retorna** o resultado.

Escopo



- **Escopo** = Visibilidade.
- Se uma variável está **visível**, é possível **acessar** e **atribuir** valores.
- JavaScript possui **3** escopos: **Global**, **Função** e **Bloco**.
- Cada um desses escopos possuem **visibilidades diferentes**.

Escopo Global



- Variáveis declaradas **fora** de **funções** e **blocos**.
- Vivem no **nível mais alto** da aplicação.
- Visíveis em **todo o código** e podem ser atribuídas em **qualquer lugar**.

Escopo Global

```
//Escopo Global
var global = 1;

function a(x,y) {
    global = 2;
}

console.log(global);
a();
console.log(global);
```

Output: 1
2

Escopo de Função



- Variáveis declaradas dentro de uma **função**.
- Visíveis apenas **dentro da função**.
- Isso inclui **funções** ou **blocos** dentro da função em questão.

Escopo de Função

```
//Escopo Global
var global = 1;

function a(x,y) {
    //Escopo de Função
    var local = 2;
}

console.log(local);
a();
```

Output: **ReferenceError: local is not defined**

```
//Escopo Global
var global = 1;

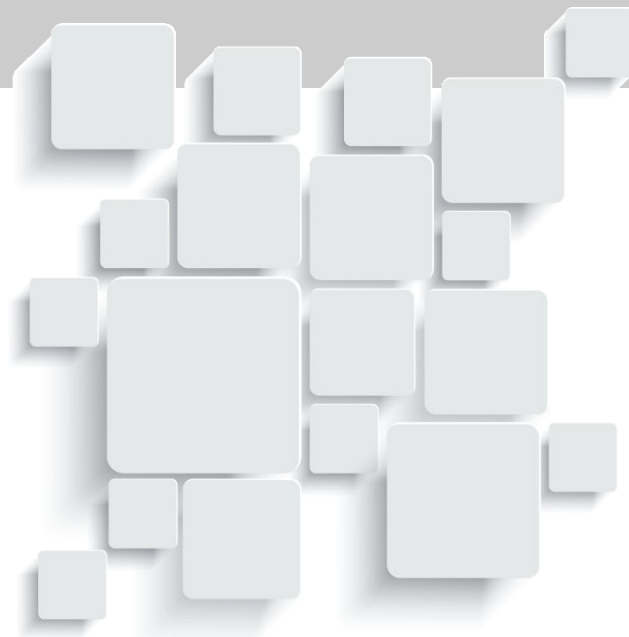
function a(x,y) {
    //Escopo de Função
    var local = 2;
    console.log(local);
}

a();
```

Output: 2

Escopo de Bloco

- Variáveis declaradas **dentro** de {}.
- Visíveis apenas **dentro do bloco**.



Escopo de Bloco



```
//Escopo Global  
var global = 1;  
  
{  
    //Escopo de Bloco  
    var bloco = 7;  
}  
  
console.log(bloco);
```

Output: 7

Escopo de Bloco



- Perceberam que tem algo de **errado**?
- Mesmo fora do **escopo**, a variável estava **visível**.
- Esse é o **problema** de declarar variáveis com **var**. O **escopo de bloco não é respeitado**.
- E assim surgiram o **const** e **let**.

Const e Let

- Ambos declaram variáveis da mesma forma que o **var**.

```
const a = 1;  
let bez = 1;
```

- Além de resolver o problema do **var**, trouxeram melhorias que **ajudam** na **manutenção** do código e evitam **bugs**.

Const e Let

- Não é possível **redeclarar** variáveis no mesmo escopo:

```
const a = "Declarei!";  
const a = "De novo!"; // SyntaxError: 'a' has already been declared.
```

```
let b = "Declarei!";  
let b = "De novo!"; // SyntaxError: 'b' has already been declared.
```

Const e Let

- Respeitam o **escopo de bloco**:

```
{  
  const bloco = 1;  
  console.log(bloco);  
}  
{  
  const bloco = 2;  
  console.log(bloco);  
}
```

Output: 1 2

```
{  
  let bloco = 1;  
  console.log(bloco);  
}  
{  
  let bloco = 2;  
  console.log(bloco);  
}
```

Output: 1 2

Const

- Feita para **guardar constantes**. Dessa forma deve ser sempre inicializada:

```
const b;  
console.log(b); // SyntaxError: Missing initializer in const  
declaration.
```

- E não pode ser **reatribuída**:

```
const a = "Atribuído!";  
a = "Reatribuído"; // TypeError: Assignment to constant variable.
```

Let

- Similar ao **var**, faz o que **const** não pode. Não precisa ser **inicializada**:

```
let a;  
console.log(a); // undefined
```

- E pode ser **reatribuída**:

```
let a = "Atribuído!";  
a = "Reatribuído"; // Tudo certo!
```



Aprendemos a:
Largar o **var**; Usar **const** e **let**.

DOM

Parte III

—



DOOM?

- **DOM**, *Document Object Model*, é uma forma de **interagir** com o **documento** através do **JavaScript**.
- Assim conseguimos começar a fazer **mágica**! Isso significa **alterar o HTML** conforme nossas necessidades.

Conectando o JS ao HTML

- Para *linkar* o JS com o HTML, usamos a tag `<script>`.
- `<script src="pasta/arquivo.js"></script>`
- Colocar antes do fechamento do `<body>`:

```
<body>
  <p>Salve!</p>
  ...
  <script src="pasta/arquivo.js"></script>
</body>
```

Encontrando elementos

- É possível encontrar um elemento no HTML buscando por seu **id**, **tag** ou **classe**.
- `document.getElementById(id);`
- `document.getElementsByTagName(tag);`
- `document.getElementsByClassName(name);`

Editando elementos



- Ao encontrar um elemento, é possível alterar **suas informações**.
- `element.innerHTML;`
- `element.atributo;`
- `element.style.prop;`

innerHTML

- Altera o conteúdo HTML **dentro** do elemento selecionado.

```
let element = document.getElementById("1");  
element.innerHTML = "Metheu essa?";
```

- `<p id="1">Metheu essa?</p>`.

Atributos

- Altera os **atributos** do elemento selecionado.

```
var element = document.getElementById("12");  
element.src = "/image/casimiro.png";
```



Style

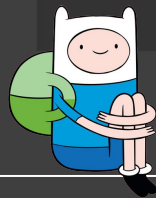
- Altera o **estilo** do elemento selecionado.

```
var element = document.getElementById("123");  
element.style.color = "green";
```

- `<h1 id="123">Ih, transformou em verde!</h1>`

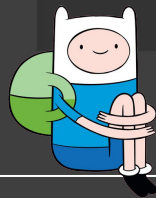


É hora de codar!



- Crie um arquivo **HTML** e um **JS** para o exercício e *link* ambos.
- Faça o download das **duas imagens indicadas** e salve na **mesma pasta** dos arquivos acima.
- Crie um **elemento de imagem** que exiba o **.png** que foi baixado.

É hora de codar!



- Crie **duas funções** que encontram o elemento da imagem.
- Na **primeira**, ela altera o atributo **src para .gif** que foi baixado.
- Na **segunda**, ela altera o atributo **src para .png** que foi baixado.
- Voltaremos aqui em **breve!** :)

