# Programação Web

**Professor: Euclides Paim**

*euclidespaim@gmail.com*

# *Fundamentos de JavaScript*

**Professor: Euclides Paim**

*euclidespaim@gmail.com*

# Programação Web

*Sumário*

- **Fundamentos de JavaScript**
  - Instruções *JavaScript*
  - Modelos Cognitivos
  - Sintaxe
  - Variáveis
  - Constantes
  - Operadores
    - aritméticos, de atribuição, de *strings, de comparação, lógicos.*
  - Tipos de Dados
  - Dados Dinâmicos
    - *Strings*, números, booleanos, *arrays* e objetos .
  - Operador *typeof*
  - Valores *undefined e null*
  - Dados Primitivos
  - Dados Complexos
- **Resumo**

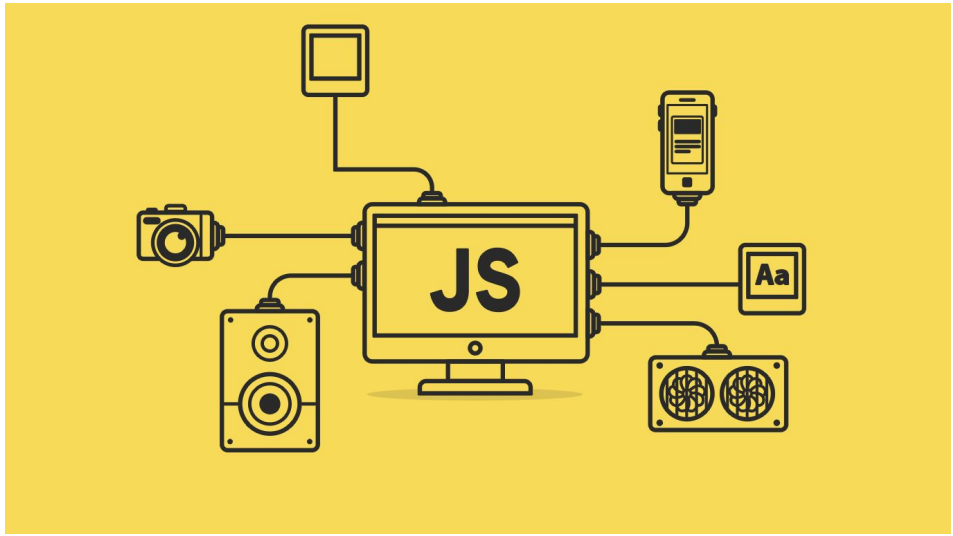
INSTITUTO FEDERAL
Catarinense
Campus Camboriú

# Desenvolvimento Web

*Introdução*

- **O que estudamos até agora?**
  - HTML
    - Elementos;
    - Atributos;
    - Imagens, tabelas, listas;
    - Classes;
    - Ids;
    - *Iframes;*
    - Elementos semânticos;
    - Formulários, elementos de formulários, tipos de *input.*
  - CSS
    - Sintaxe CSS;
    - Seletores;
    - Formas de inserir CSS, ordem de cascateamento;
    - *Box model*, *backgrounds*, *margins* e *padding,*
    - Posicionamento, transformações 2D e 3D e transições.

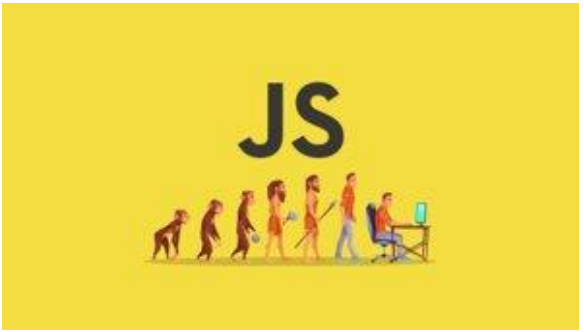
# Desenvolvimento Web I



# Desenvolvimento Web

*Introdução*

- ***Statements***
  - Um programa de computador é uma lista de "regras" a serem "executadas" por um computador. Na programação, essas regras de programação são chamadas de **instruções** (*statements*).
  - Um programa JavaScript é uma lista de **instruções** de programação. Em HTML, **os programas JavaScript** são executados pelo **navegador** da web.
  - As instruções JavaScript são compostas por valores, operadores, expressões, palavras-chave e comentários.



# Desenvolvimento Web

*Modelos Cognitivos*

- **Observe este código:**

```
var a = 10;
var b = a;
a = 0;
```

Quais são os valores de "a" e "b" após sua execução? Resolva isso em sua cabeça antes de irmos adiante...

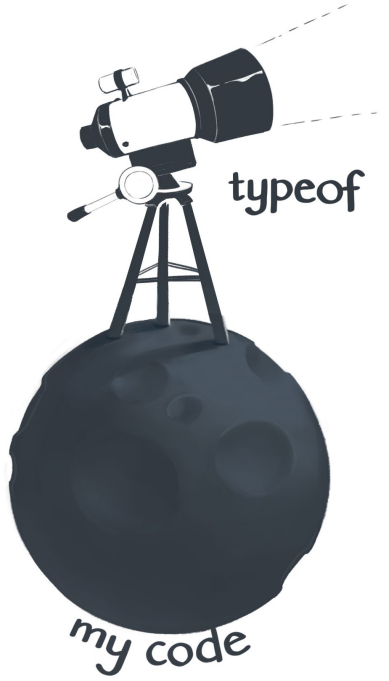
O objetivo deste exercício não é apresentar variáveis. Vamos falar mais delas em seguida. Em vez disso, esperamos fazer você notar e refletir sobre seu ***modelo cognitivo***.

- Leia o código anterior novamente com a intenção de realmente ter certeza qual é o resultado. (Veremos por que essa intenção é importante um pouco mais tarde.) Enquanto você lê pela segunda vez, preste muita atenção ao que está acontecendo em sua cabeça, passo a passo.
- Você pode notar um **monólogo** como este:

  - `var a = 10;`
    - Declare uma variável chamada "a". Atribua `10` para "a".

  - `var b = a;`
    - Declare uma variável chamada "b". Atribua "a" para "b".

  - `a = 0;`
    - Atribua `0` para "a".



# Desenvolvimento Web

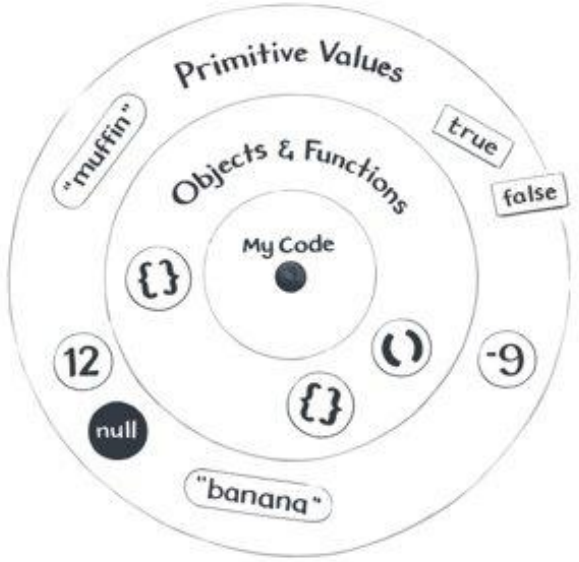
*Modelos Cognitivos*

- Talvez o seu monólogo seja um pouco diferente. Talvez você diga: *"a" recebe 10,* ou talvez você o leia em uma ordem ligeiramente diferente. Talvez você tenha chegado a um resultado diferente.
- Preste atenção em como exatamente era diferente. Observe como esse monólogo não captura o que realmente está acontecendo em sua cabeça.
- Você pode dizer *"defina b como a"*, mas o que significa **definir** uma variável?

# Desenvolvimento Web

*Modelos Cognitivos*

- Você vai notar que, para todo conceito fundamental de programação (como uma variável) e operações (como definir seu valor), há um conjunto de **analogias profundas** que você associou a eles.
- Algumas delas podem vir do mundo real. Outros podem ser redirecionados de outros campos que você aprendeu primeiro, como números de **matemática**.
- Essas analogias podem se sobrepor e até se contradizer, mas ainda ajudam a entender o que está acontecendo no código.



- Por exemplo, muitas pessoas aprenderam sobre variáveis como **“caixas”** nas quais você pode **colocar** coisas. Mesmo que você não use mais as caixas quando vê uma variável, elas ainda podem se comportar "quadradas" em sua imaginação.
- Essas aproximações de como algo funciona na sua cabeça são conhecidas como "**modelos cognitivos**".
- Pode ser difícil se você estiver programando há muito tempo, mas tente observar e introspectar seus modelos mentais. Eles provavelmente são uma combinação de atalhos mentais visuais, espaciais e mecânicos.

# Desenvolvimento Web

*Modelos Cognitivos*

- Essas analogias (como "caixas" de variáveis) **influenciam** como lemos o código por toda a vida. Mas, às vezes, nossos modelos mentais estão errados.
  - Talvez um tutorial que lemos no início tenha sacrificado a **correção** pela **facilidade** de explicação.
  - Talvez tenhamos transferido incorretamente uma analogia sobre um determinado recurso de **outra linguagem** que aprendemos anteriormente.
  - Talvez tenhamos inferido um modelo mental de algum trecho de código e nunca realmente verificado se ele era preciso.

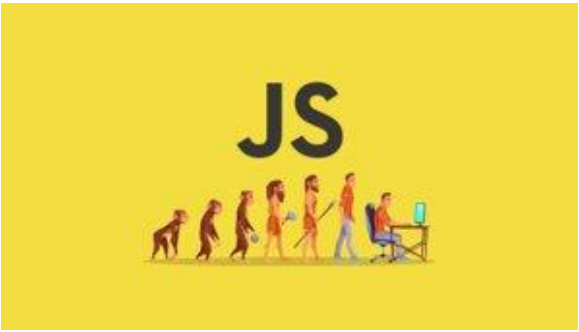
- Identificar e corrigir esses problemas é o objetivo deste exercício. Um **bom modelo mental** o ajudará a encontrar e corrigir erros mais rapidamente, a entender melhor o código de outras pessoas e a se sentir confiante no código que você escreve.

Fonte: https://justjavascript.com/

# Desenvolvimento Web

*Sintaxe*

- A **sintaxe** JavaScript é o **conjunto de regras** de como os programas JavaScript são construídos:
  - O **ponto-e-vírgula** separa as instruções. Adicionamos um ponto-e-vírgula no final de cada instrução executável;
  - Quando **separadas** por ponto e vírgula, são permitidas **várias instruções em uma linha**;
  - Finalizar uma instrução com ponto e vírgula não é obrigatório, mas é **altamente** recomendado.
  - JavaScript **ignora** múltiplos **espaços em branco**.
    - **Boa prática** é colocar espaços ao redor dos operadores (= + - * /):



- As instruções JavaScript podem ser agrupadas em **blocos de código**, dentro de chaves **{.....}**.
  - O objetivo dos blocos de código é definir instruções a serem executadas juntas.
  - Um lugar onde encontramos instruções agrupadas em blocos é nas **funções** JavaScript, exemplo:

```
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello World!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
```

# Desenvolvimento Web

*Sintaxe*

- JavaScript é **case-sensitive** e usa o conjunto de caracteres **Unicode.** Por exemplo, a palavra Früh (que significa "cedo" em Alemão) pode ser usada como nome de variável.

```
var Früh = "foobar";
```

  - Mas a variável **früh** não é a mesma que **Früh** porque JavaScript é *case-sensitive*.

- A sintaxe dos **comentários** em JavaScript é semelhante a do C++ e muitas outras linguagens:

```
1.  // comentário de uma linha
2.  …
3.  /* isto é um comentário longo
4.             de múltiplas linhas. */
5.  …
6.  /* Você não pode, porém, /* aninhar comentários */ SyntaxError */
7.  …
```

# Desenvolvimento Web

*Variáveis*

# Desenvolvimento Web

*Variáveis*

- **Variáveis** são nomes simbólicos para os valores em sua aplicação. O nome das variáveis, chamados de **identificadores**, obedecem determinadas regras:
  - Um identificador JavaScript **deve** começar com uma **letra**, ***underline*** (_), ou **cifrão** ($);
  - Os caracteres subsequentes podem também ser números (0-9).
  - JavaScript é ***case sensitive***, letras incluem caracteres de maiúsculos e caracteres minúsculos.
  - Podemos usar a **ISO 8859-1** ou caracteres **Unicode** tal como os identificadores å e ü.

Obs.: Hifens **não** são permitidos no JavaScript. Eles são reservados para subtrações.

# Desenvolvimento Web

*Variáveis*

- 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**.

***Fonte**:* **var**, **let** e **const**. (leitura complementar recomendada)

# Desenvolvimento Web

*Variáveis*

- Podemos **declarar** uma variável de três formas:
  - Com a palavra-chave **var**. Esta sintaxe pode ser usada para declarar tanto variáveis **locais** como variáveis **globais**.
    - **Ex.:** var x = 42
  - Por simples **adição de valor**. Isso declara uma **variável global**. Essa declaração gera um aviso de advertência no JavaScript. **Não recomendado**.
    - **Ex.:** x = 42
  - Com a palavra chave **let**. Essa sintaxe pode ser usada para declarar uma **variável local** de escopo de bloco.
    - **Ex.:** let y = 1

# Desenvolvimento Web

*Escopo de Variáveis*

Quando declaramos uma variável **fora** de qualquer **função**, ela é chamada de **variável *global***, e está disponível para qualquer outro código no documento atual.

Quando declaramos uma variável **dentro** de uma **função**, é chamada de **variável *local***, pois ela está disponível **somente dentro dessa função**.

**Leitura complementar**: ECMAScript 6 https://www.w3schools.com/js/js_es6.asp
**Ecma International**: http://www.ecma-international.org/

# Desenvolvimento Web

*Escopo de Variáveis*

- Por **exemplo** o código a seguir exibirá 5, **porque o escopo** de `x` **está na função** (ou contexto global) no qual `x` é declarado, não o bloco, que neste caso é a declaração **if**.

```
if (true) {
    var x = 5;
}
console.log(x); // 5
```

- Esse comportamento é alterado, quando usado a declaração **`let`** introduzida pelo ECMAScript 6.

```
if (true) {
    let y = 5;
}
console.log(y); // ReferenceError: y não está definido
```

# Desenvolvimento Web

*Constantes*

- Podemos criar uma **constante** apenas de leitura por meio da palavra-chave `const`. A sintaxe de um identificador de uma constante é semelhante ao identificador de uma variável:
  - Deve começar com uma **letra**, **sublinhado** ou **cifrão** e pode conter caractere alfabético, numérico ou sublinhado.

```
const PI = 3.14;
```

  - Uma constante **não pode alterar seu valor** por meio de uma atribuição ou ser declarada novamente enquanto o script está em execução. Deve ser inicializada com um valor.
  - As **regras de escopo** para as constantes são as **mesmas** para as variáveis ***let*** de escopo de bloco. Se a palavra-chave `const` for omitida, presume-se que o identificador represente uma variável.

# Desenvolvimento Web

*Operadores*

| Operador | Comparação |
|---|---|
| == | Igual |
| != | Diferente |
| < | Menor |
| > | Maior |
| <= | Menor Igual |
| >= | Maior Igual |

# Desenvolvimento Web

*Operadores*

- Em Javascript existem vários tipos de **operadores**, ex.:
  - Lógicos;
  - Aritméticos;
  - de *strings;*
  - de atribuição;
  - Unários;
  - de comparação;
  - bit a bit...
- Cada um desses operadores **pode ser utilizado** tanto com valores explícitos como com variáveis e até com funções e objetos, para somar, subtrair, comparar além de outras funções que vão dar lógica e fluxo à sua aplicação.

# Desenvolvimento Web

*Operadores*

- Em Javascript existem vários tipos de **operadores**, ex.:
  - Lógicos;
  - Aritméticos;
  - de *strings;*
  - de atribuição;
  - Unários;
  - de comparação;
  - bit a bit...
- Cada um desses operadores **pode ser utilizado** tanto com valores explícitos como com variáveis e até com funções e objetos, para somar, subtrair, comparar além de outras funções que vão dar lógica e fluxo à sua aplicação.

# Desenvolvimento Web

*Operadores Aritméticos*

- Operadores aritméticos são usados para executar operações em **números** ou **variáveis**. Uma operação aritmética típica opera em números que podem ser **literais**, exemplo:

```
var x = 100 + 50;
```

- em **variáveis**, exemplo:

```
var x = a + b;
```

- ou **expressões**, exemplo:

```
var x = (100 + 50) * a;
```

# Desenvolvimento Web

*Operadores e Operandos*

- Os números (em uma operação aritmética) são chamados **operandos**. A operação (a ser executada entre os dois operandos) é definida por um **operador** .

| Operando | Operador | Operando |
|---|---|---|
| 100 | + | 50. |

- Ex.: O operador de **adição** ( **+** ) adiciona números:

```
var x = 5;
var y = 2;
var z = x + y;
```

# Desenvolvimento Web

*Operadores Aritméticos*

- Operadores aritméticos JavaScript.

| Operador | Descrição |
|---|---|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| ** | Exponenciação (ES2016) |
| / | Divisão |
| % | Módulo (Resto da Divisão) |
| ++ | Incremento |
| -- | Decremento |

Fonte: https://www.w3schools.com/js/js_arithmetic.asp

# Desenvolvimento Web

*Operadores de Atribuição*

- Operadores de atribuição atribuem valores a variáveis JavaScript.

| Operador | Exemplo | Similar |
|---|---|---|
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| **= | x **= y | x = x ** y |

Fonte: https://www.w3schools.com/js/js_assignment.asp

# Desenvolvimento Web

*Operadores de Strings*

- O operador **+** também pode ser usado para adicionar (concatenar) seqüências de caracteres.

```
var texto = "Tenha um ";
texto += "bom dia!";
```

- O resultado de **texto** será:

```
Tenha um bom dia!
```

**Obs.:** Quando usado em cadeias, o operador + é chamado de operador de concatenação.

Fonte: https://www.w3schools.com/js/js_assignment.asp

# Desenvolvimento Web

*Operadores de Strings*

- A adição de dois números retornará a soma, mas a adição de um número e uma *string* retornará uma *string*, exemplo:

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
```

- O resultado de *x* , *y* , e *z* são os seguintes:

```
10
55
Hello5
```

**Obs.:** Se você adicionar um número e uma string, o resultado será uma **string**!

# Desenvolvimento Web

*Operadores de Comparação*

- Operadores de **comparação** e **lógicos** são usados para testar **`true`** ou **`false`**. Operadores de comparação são usados em instruções lógicas para determinar a igualdade ou diferença entre variáveis ou valores.

- O JavaScript é uma linguagem com variáveis que **não são fortemente tipadas**. Isso significa que, embora as variáveis possuam tipos, eles **só são atribuídos em tempo de execução**. Por isso, não há definição explícita do tipo de dado: tudo é feito utilizando o termo "var".

- Por exemplo, um dado do tipo **inteiro** é definido como **`var i = 0;`**, enquanto um dado do tipo **string** é definido como **`var s = 'texto';`**

- Devido a essa característica, o JavaScript possui dois tipos de operadores de igualdade e dois de desigualdade: **`==`** e **`===`** e **`!=`** e **`!==`**.

# Desenvolvimento Web

*Operadores de Comparação*

- Dado **`x = 5`**, as tabelas a seguir explicam os operadores de comparação:

| Operador | Descrição | Comparação | Retorna |
|---|---|---|---|
| == | Igual a | x == 8 | falso |
| | | x == 5 | verdadeiro |
| | | x == "5" | verdadeiro |
| === | valor igual e tipo igual | x === 5 | verdadeiro |
| | | x === "5" | falso |
| != | não igual | x != 8 | verdadeiro |

# Desenvolvimento Web

*Operadores de Comparação*

- Dado `x = 5;`

| Operador | Descrição | Comparação | Retorna |
|---|---|---|---|
| !== | Valor não igual ou tipo não igual | x !== 5 | falso |
| | | x !== "5" | verdadeiro |
| | | x !== 8 | verdadeiro |
| > | maior que | x > 8 | falso |
| < | menor que | x < 8 | verdadeiro |
| >= | maior que ou igual a | x >= 8 | falso |
| <= | menor que ou igual a | x <= 8 | verdadeiro |

Fonte: https://www.w3schools.com/js/js_comparisons.asp

# Desenvolvimento Web

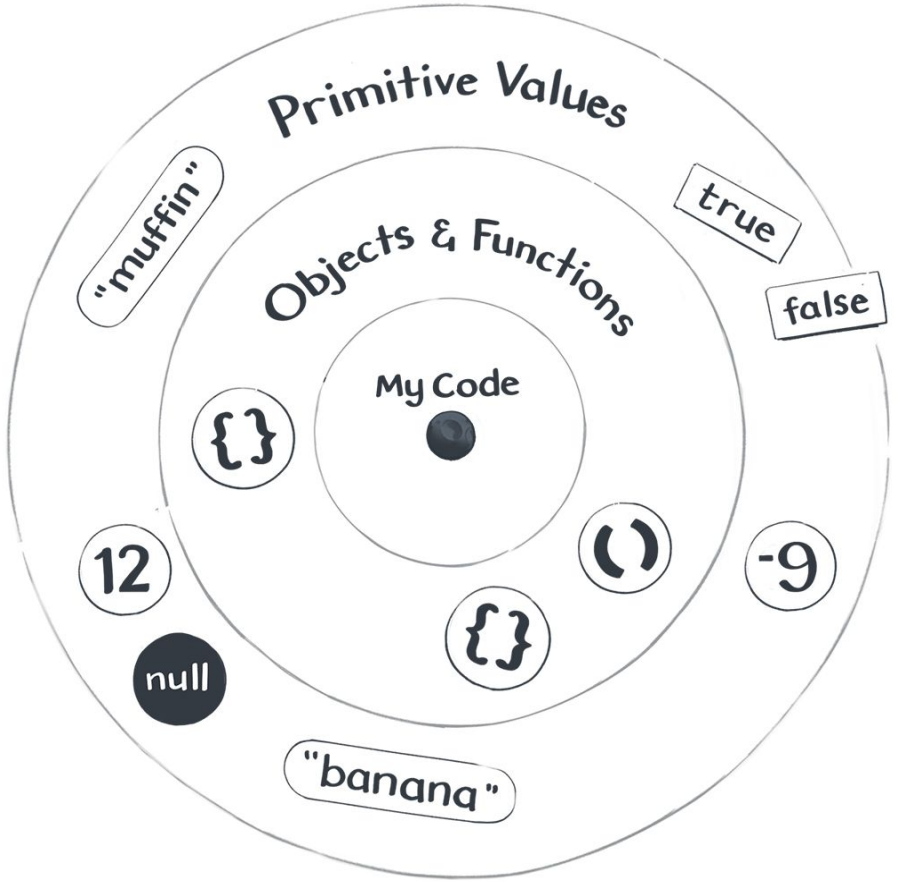
*Operadores Lógicos*

- Operadores lógicos são usados para determinar a lógica entre variáveis ou valores.
- Dado `x = 6` e `y = 3`, a tabela abaixo explica os operadores lógicos:

| Operador | Descrição | |
|---|---|---|
| `&&` | **e** lógico | `(x < 10 && y > 1)` é **true** |
| `\|\|` | **ou** lógico | `(x == 5 \|\| y == 5)` é **false** |
| `!` | **não** lógico | `!(x == y)` é **true** |

# Desenvolvimento Web

*Tipos de Dados*



# Desenvolvimento Web

*Tipos de Dados*

- Em Javascript as variáveis podem conter vários **tipos de dados** :
  - números;
  - *strings;*
  - objetos
  - entre outros:

```
var comprimento = 16;                                // Number
var lastName = "Johnson";                            // String
var x = {firstName:"John", lastName:"Doe"};          // Object
```

# Desenvolvimento Web

*Tipos de Dados*

- Na programação, os **tipos de dados** são um **conceito importante**. Para poder operar com variáveis, é importante saber de que tipo é determinada variável. Sem tipos de dados, um computador não pode resolver com segurança o exemplo a seguir:

```
var x = 16 + "Volvo";
```

- Faz algum sentido adicionar "Volvo" a 16? Irá produzir um erro ou produzirá um resultado? JavaScript tratará o exemplo acima como:

```
var x = "16" + "Volvo";
```

# Desenvolvimento Web

*Tipos de Dados*

- Ao somarmos um **número** e uma ***string***, o JavaScript tratará o número como uma *string*.

```
var x = 16 + "Volvo";  //16Volvo
var x =  "Volvo" + 16; //Volvo16
```

- *JavaScript* avalia expressões da **esquerda para a direita**. Sequências diferentes podem produzir resultados diferentes:

```
var x = 16 + 4 + "Volvo"; //20Volvo
var x = "Volvo" + 16 + 4; //Volvo164
```

- No primeiro exemplo, o JavaScript trata 16 e 4 como **números**, até atingir "Volvo". No segundo exemplo, como o primeiro **operando** é uma *string*, **todos** os operandos são tratados como *strings*.

- *JavaScript* tem tipos dinâmicos. Isso significa que a mesma variável pode ser usada para armazenar diferentes tipos de dados:

```
var x;               // Aqui x é undefined
x = 5;               // Aqui x é um Número
x = "John";          // Aqui x é uma String
```

# Desenvolvimento Web

*Strings*

- Uma *string* (ou uma *sequência* de texto) é uma série de caracteres como "John Doe". As *strings* são escritas **com aspas**. Você pode usar aspas simples ou duplas:

```
var carName1 = "Volvo XC60";   // Usando aspas duplas

var carName2 = 'Volvo XC60';   // Usando aspas simples
```

# Desenvolvimento Web

*Numbers*

- *JavaScript* possui apenas um tipo **numérico** (*number*). Os números podem ser escritos com ou sem decimais:

```
var x1 = 34.00;      // Escritos com decimais

var x2 = 34;         // Escritos sem decimais
```

- Números extra grandes ou extra pequenos podem ser escritos com notação científica (exponencial):

```
var y = 123e5;       // 12300000

var z = 123e-5;      // 0.00123
```

# Desenvolvimento Web

*Booleans*

- Booleanos podem ter apenas dois valores: `true` ou `false`:

```
var x = 5;
var y = 5;
var z = 6;
(x == y)        // Returns true
(x == z)        // Returns false
```

- Booleanos são comumente usados em testes condicionais.

# Desenvolvimento Web

*Arrays*

- ***Arrays*** em *JavaScript* são escritas com **colchetes**. Os itens do vetor são separados por vírgulas. O código a seguir declara (cria) um *array* chamada `carros`, contendo três itens (nomes de carros):

```
var cars = ["Saab", "Volvo", "BMW"];
```

- Os índices dos vetores em *JavaScript* são ***zero-based***, o que significa que o primeiro item será [**0**], o segundo será[**1**] e assim por diante.

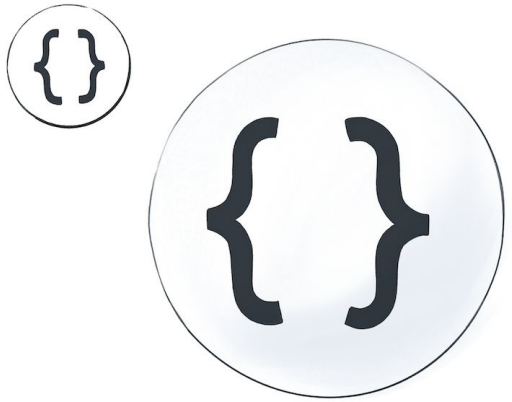
# Desenvolvimento Web

*Objects*

- **Objetos** *JavaScript* são escritos com chaves **{ }**. As propriedades do objeto são armazenadas como pares **nome: valor**, separados por vírgulas.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

- O objeto ***person*** no exemplo acima possui 4 propriedades:
  - *firstName;*
  - *lastName;*
  - *age;*
  - *eyeColor*.



# Desenvolvimento Web

*O operador typeof*

- Você pode usar o operador ***typeof*** para encontrar o tipo de uma variável *JavaScript*. O operador ***typeof*** retorna o tipo de uma variável ou expressão:

```
typeof "John Doe"      // Retorna "string"
typeof 3.14            // Retorna "number"
```

# Desenvolvimento Web

*Undefiened e Null*

- Em JavaScript, uma variável **sem** um valor, tem o valor ***undefined***. O tipo também será ***undefined***.

```
var car;    // Valor é undefined, tipo é undefined
```

- Em JavaScript ***null*** é "nada" e supostamente algo que não existe. Infelizmente, em JavaScript, o tipo de dados de null é um objeto. Você pode considerar um **bug** do JavaScript que *typeof null* seja um objeto enquanto deveria ser *null*.

```
typeof undefined             // undefined
typeof null                  // object
```

- Os valores ***undefined*** e ***null*** são iguais em valor, mas diferentes em tipo:

```
null === undefined           // false
null == undefined            // true
```

# Desenvolvimento Web

*Dados Complexos*

- O operador ***typeof*** pode retornar um dos dois tipos complexos:
  - *function*
  - *object*

- O operador ***typeof*** retorna "*object*" para objetos, *arrays* e *nulls*. O operador ***typeof*** não retorna "objeto" para funções. Exemplo:

```
typeof {name:'John', age:34} // Retorna "object"
typeof [1,2,3,4]             // Retorna "object" (veja nota abaixo)
typeof null                  // Retorna "object"
typeof function myFunc(){}   // Retorna "function"
```

**Obs.:** O operador ***typeof*** retorna "*object*" para matrizes porque, em *JavaScript*, matrizes são objetos.

# Desenvolvimento Web I

*Resumo*

- **Fundamentos de JavaScript**
  - Instruções *JavaScript*
  - Modelos Cognitivos
  - Sintaxe
  - Variáveis
  - Constantes
  - Operadores
    - aritméticos, de atribuição, de *strings, de comparação, lógicos.*
  - Tipos de Dados
  - Dados Dinâmicos
    - *Strings*, números, booleanos, *arrays* e objetos .
  - Operador *typeof*
  - Valores *undefined e null*
  - Dados Complexos
- **Resumo**

# Referências

## Referências Básicas

RESIG, John; BIBEAULT, Bear; MARZ, Josip. ***Secrets of the JavaScript Ninja***. 2. ed. Shelter Island: Manning Publications, 2016.
SILVA, Maurício Samy. HTML 5: a linguagem de marcação que revolucionou a web. São Paulo: Novatec, 2011.
GRONER, Loiane. ***Estrutura de dados e algoritmos com JavaScript:*** *escreva aplicações de JS modernas e performáticas utilizando estruturas de dados e algoritmos*. São Paulo: Novatec Editora, 2019.

## Referências Complementares

FLANAGAN, David. **o guia definitivo**. . O Really. 2012
SILVA, Maurício Samy. **Criando sites com HTML: sites de alta qualidade com HTML e CSS**. . Novatec. 2010
IEPSEN, Edécio Fernandes. **Lógica de Programação e Algoritmos com JavaScript.** Novatec. 2018.

## Referências na Internet

https://www.w3schools.com
https://developer.mozilla.org/pt-BR/docs/Web
https://illustrated.dev/advancedjs