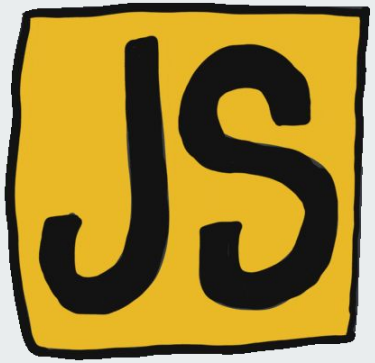
INSTITUTO FEDERAL
Ceará
Campus Tauá

INTRODUÇÃO AO JAVASCRIPT



Programação Web II - Análise e Desenvolvimento de Sistemas

Profº: Me. Lucas Ferreira Mendes

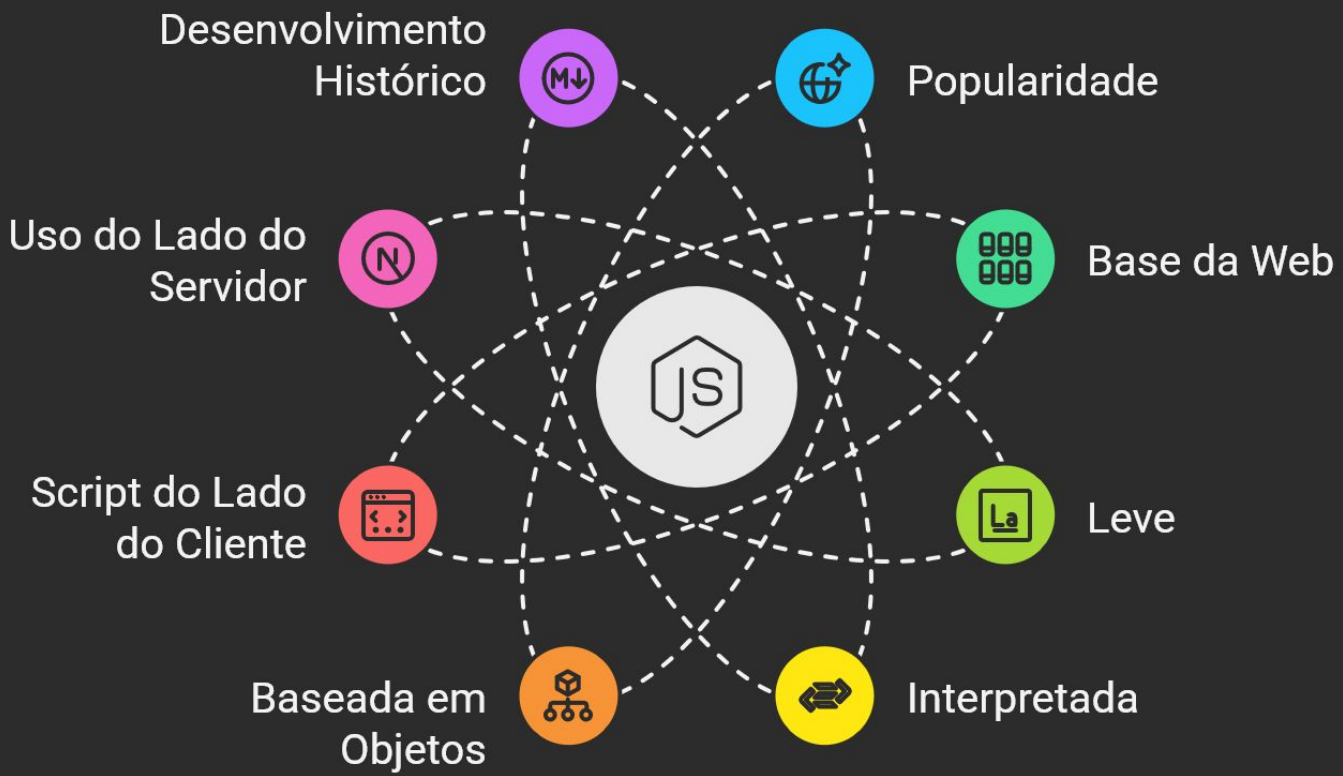
ROTEIRO

- Conceitos iniciais
- Adicionando JavaScript no documento HTML
- Comentários e saída de dados
- Statements e sintaxe
- Variáveis, operadores e tipos de dados
- Condicionais
- Estruturas de Repetição
- Funções

O que é JavaScript?

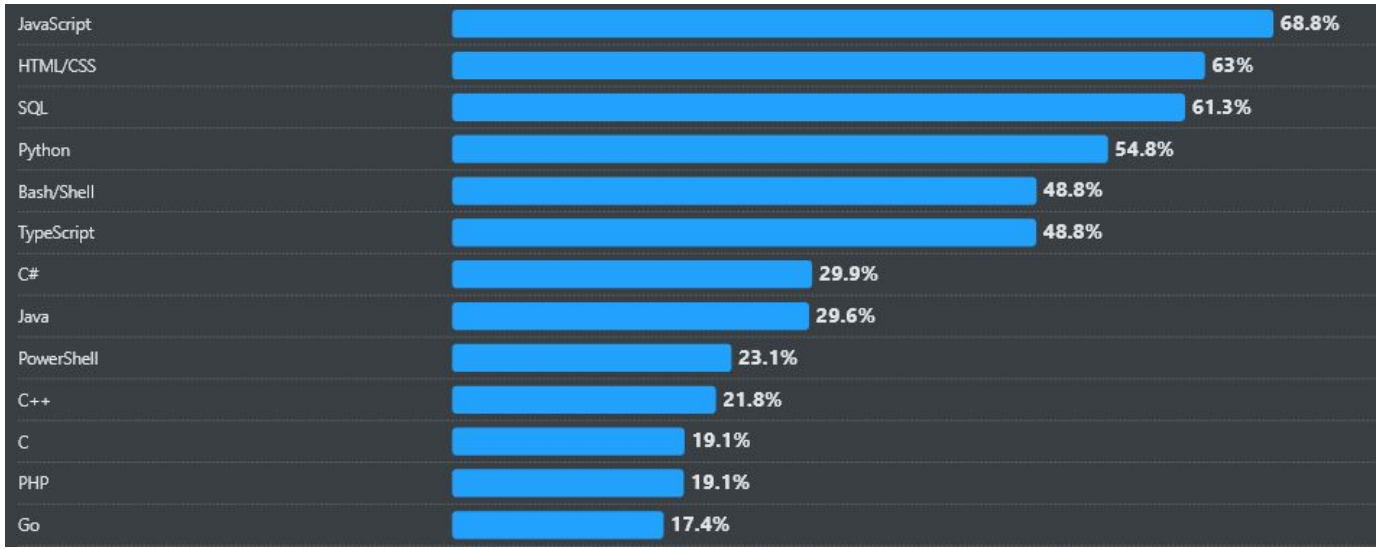
- Linguagem de programação **mais popular** do mundo atualmente
- Linguagem de programação base da Web
- Linguagem leve, interpretada e baseada em objetos
- Além disso, é uma linguagem de script para a Web que é executada no lado cliente por um navegador
- Recentemente tem sido bastante utilizada também como linguagem de programação no lado servidor, através de plataformas como o Node.js
- O JavaScript surge em 1995 e torna-se um padrão ECMA em 1997, tendo como nome oficial **ECMAScript**

Visão Geral do JavaScript



O que é JavaScript?

- Ranking de Linguagens de Programação, Script e de Marcação do Stack Overflow (2025 Developer Survey)

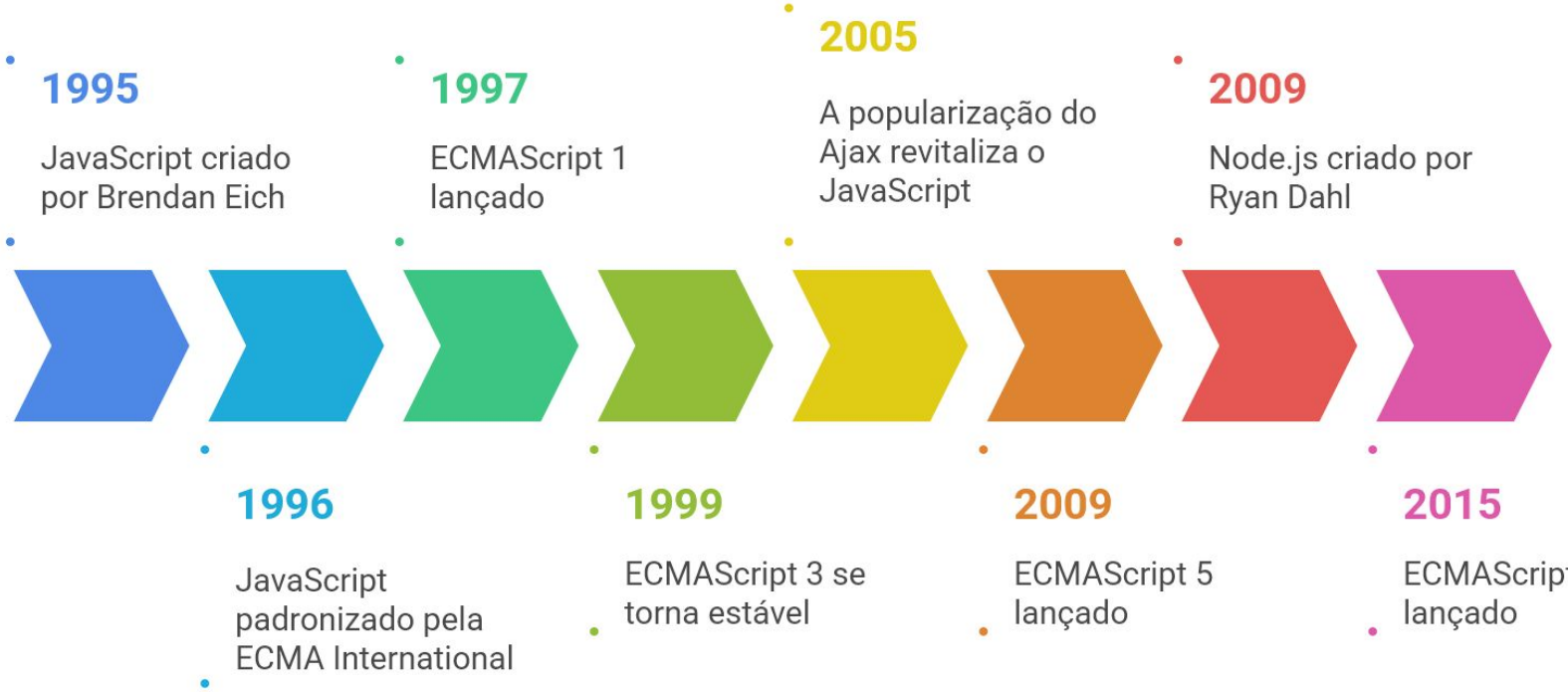


Breve Histórico do JavaScript

Brendan Eich

- 1995 - O JavaScript surge, criado por **Brendan Eich** na Netscape;
- 1996 - JS é transferido para o **ECMA International** para ser padronizada;
- 1997 - Lançamento da primeira versão do **ECMAScript (ES)** batizada de ECMA-262;
- 1999 - **ES3** vira uma base estável por vários anos;
- 2005 - Popularização do termo **Ajax** para apps web assíncronos (Gmail/Maps, páginas sem reload), fazendo o JS voltar aos holofotes;
- 2008 - **ES4** abortado por discordâncias sobre a complexidade e os objetivos;
- 2009 - Lançado o **ES5**, que é a versão mais compatível com os browsers atuais;
- 2009 - **Ryan Dahl** cria o **Node.js** para tratar o **problema C10K**;
- 2015 - Lançado o **ECMAScript 6 (ES2015/ES6)**, sendo a maior revisão desde a origem. A partir desse ponto, o termo **JavaScript moderno (ES6+)** ganha força e o padrão passa por **revisões anuais**.

A Jornada do JavaScript ao Longo do Tempo



O que é possível fazer com JavaScript?

Adicionando JavaScript no documento HTML

- Dentro de uma tag **<script>** no HTML

index.html

```
<script>
      document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

- Arquivos JS externos

myScript.js

```
function myFunction() {
      document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

index.html

```
<script src="myScript.js"></script>
```

Adicionando JavaScript no documento HTML

- Adicionar código diretamente nos atributos

index.html

```
<button onclick="console.log('JS na tag!')">Clique aqui!</button>
```

não recomendado!

Adicionando JavaScript no documento HTML

- Problemas e estratégias para o carregamento de Scripts
  - A ordem de carregamento pode impactar no resultado e causar problemas
  - Scripts que dependem de outros scripts devem ser carregados na sequência
  - Scripts que manipulam elementos da página devem ser carregados após o HTML ser completamente carregado
  - Soluções modernas: *async* e *defer*

index.html

```
<script async src="js/vendor/jquery.js"></script>
<script async src="js/myScript.js"></script>
```

Adicionando JavaScript no documento HTML

- Problemas e estratégias para o carregamento de Scripts
  - A ordem de carregamento pode impactar no resultado e causar problemas
  - Scripts que dependem de outros scripts devem ser carregados na sequência
  - Scripts que manipulam elementos da página devem ser carregados após o HTML ser completamente carregado
  - Soluções modernas: *async* e *defer*

index.html

```
<script defer src="js/vendor/jquery.js"></script>
<script defer src="js/myScript.js"></script>
```

Comentários em JavaScript

- Assim como em outras linguagens de programação, é possível escrever comentários dentro do código JS, que serão ignorados pelo navegador
- Há dois tipos de comentários em JS: *comentário de linha* e *comentário de múltiplas linhas*

myScript.js

```
// Eu sou um comentário de uma única linha
```

myScript.js

```
/*
      Eu sou um comentário
      de múltiplas linhas
*/
```

Saídas de dados

- O JavaScript pode “exibir” dados de diferentes maneiras:
  - escrevendo dentro de um elemento HTML com a propriedade `innerHTML`
  - escrevendo na saída do documento HTML usando `document.write()`
  - escrevendo dentro de uma caixa de alerta usando `window.alert()`
  - escrevendo no console do *browser* usando `console.log()`

JavaScript Statements

- Statements (declarações de instruções) no JavaScript são compostas de: valores, operadores, expressões, palavras reservadas e comentários
- O ponto e vírgula (;) separa as instruções em JavaScript, embora possamos codar sem utilizá-lo

myScript.js

```
let a, b, c;
a = 5;
b = 6;
c = a + b;

let x
x = c + 2
```

JavaScript Statements

- Instruções JavaScript podem ser agrupadas em blocos de códigos demarcados por chaves **{ }**

myScript.js

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

JavaScript Statements

- Instruções geralmente iniciam com uma palavra-chave (palavra reservada) para indicar a ação que deve ser executada
- Algumas das palavras reservadas do JS, são:

Palavra reservada	Descrição
var	Declara uma variável
let	Declara uma variável de bloco
const	Declara uma constante de bloco
if	Indica um bloco de instruções que serão executadas a depender de uma condição
switch	Indica um bloco de instruções que serão executadas em diferentes casos
for	Indica um bloco de instruções que serão executadas em um loop
function	Declara uma função
return	Finaliza uma função
try	Implementa um gerenciador de erro para um bloco de instruções

Sintaxe JavaScript: detalhes importantes

- Regras gerais para nomear uma variável ou função:
  - um nome deve começar com uma letra (**A-Z** ou **a-z**), um símbolo de cifrão (**$**) ou um underline (**_**)
  - números não são permitidos para iniciar o nome de uma variável ou função
  - os caracteres subsequentes podem ser letras, digitos, underline ou símbolos de cifrão
- JavaScript é **Case Sensitive**:
  - as variáveis ***lastName*** e ***lastname***, são variáveis distintas

Sintaxe JavaScript: detalhes importantes

- JavaScript não define um padrão para representar nomes compostos para variáveis ou funções, porém é muito comum a adoção do estilo ***Camel Case*** para esse propósito
- Algumas formas de juntar palavras para nomear variáveis e funções:
    - ***Hífens (Kebab Case):*** last-name, first-name
    - ***Underscore (Snake Case):*** first_name, last_name
    - ***Upper Camel Case***: FirstName, LastName
    - ***Lower Camel Case:*** firstName, lastName

Variáveis

- Variáveis em JavaScript podem ser declaradas de 4 formas diferentes:
  - automaticamente
  - usando `var`
  - usando `let`
  - usando `const`

Variáveis

- Variáveis em JavaScript podem ser declaradas de 4 formas diferentes:
  - automaticamente
  - usando `var`
  - usando `let`
  - usando `const`

myScript.js

```
x = 5;
y = 6;
z = x + y;
```

- **Variáveis não declaradas, são declaradas automaticamente em seu primeiro uso!**

Variáveis

- Variáveis em JavaScript podem ser declaradas de 4 formas diferentes:
  - automaticamente
  - usando **`var`**
  - usando `let`
  - usando `const`

myScript.js

```
var x = 5;
var y = 6;
var z = x + y;
```

- O var só deve ser usado em códigos escritos para navegadores antigos
- O gerenciamento de escopo é “confuso”
- Usado antes do ES6

Variáveis

- Variáveis em JavaScript podem ser declaradas de 4 formas diferentes:
  - automaticamente
  - usando `var`
  - usando `let`
  - usando `const`

myScript.js

```
let x = 5;
let y = 6;
let z = x + y;
```

- Pensado para trazer o conceito de escopo de bloco
- Agora uma variável declarada em um bloco específico tem seu escopo limitado a ele
- Variáveis declaradas com let não podem ser re-declaradas
- Devem ser declaradas antes do uso

Variáveis

- Variáveis em JavaScript podem ser declaradas de 4 formas diferentes:
  - automaticamente
  - usando `var`
  - usando `let`
  - usando `const`

myScript.js

```
const x = 5;
const y = 6;
const z = x + y;
```

- Usa-se o const quando o valor* não deve ser alterado
- Somente use let quando não for possível usar o const

Variáveis

- Variáveis em JavaScript podem ser declaradas de 4 formas diferentes:
  - automaticamente
  - usando `var`
  - usando `let`
  - usando `const`

myScript.js

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Toyota";
cars.push("Audi");
```

- Usa-se o const quando o valor* não deve ser alterado
- Somente use let quando não for possível usar o const
- *const não define um valor constante, define uma referência constante

Escopo de Código

- **O que é escopo de código e como o JS lida com isso?**
  - **Escopo:** região do código (***contexto léxico***) onde um identificador (variável/funcão) é visível/acessível.
  - **Escopo de bloco** (`{ }`): introduzido com `let`/`const`.
  - **Escopo de função**: limites definidos por `function`.
    - *A palavra reservada `var` possui escopo global ou de função.*
  - **Escopo global:** escopo mais amplo existente em um programa, onde identificadores declarados aqui podem ser acessados em qualquer parte do programa.

Escopo de Código

Escopo Global

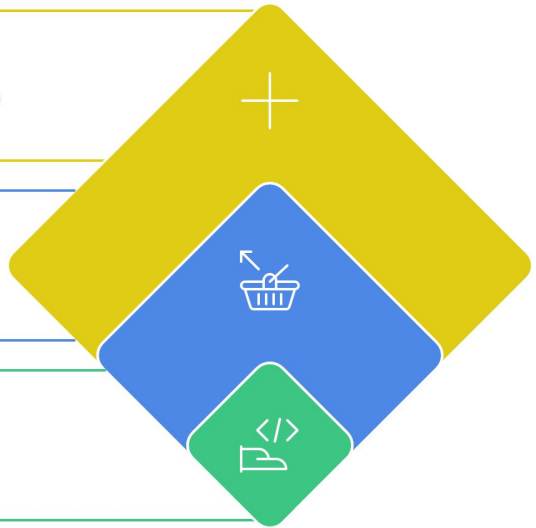
Escopo mais amplo acessível em todo o programa

Escopo de Função

Limites definidos por function

Escopo de Bloco

Escopo mais restritivo introduzido por let/const



Escopo de Código

```
1  // bloco vs função
2  {
3    let a = 1;
4    var b = 2;
5  }
6
7  console.log(a); // ReferenceError (fora do bloco)
8  console.log(b); // 2 (var "vaza" do bloco)
9
10 function f() {
11   const x = 10;
12   return x;
13 }
14
15 console.log(x); // ReferenceError (fora da função)
```

Escopo de Código

- **Escopo Léxico**
  - No JS, o *ambiente de resolução* de nomes é determinado no momento em que o código é escrito (não em runtime).
  - Ou seja, os identificadores são acessíveis a partir de onde (“local físico”) foram declarados no código.

```
1  const exterior = "campus";
2
3  function funcaoExterna() {
4    const unidade = "Tauá";
5    function funcaoInterna() {
6      console.log(exterior, unidade); // acessa o que está LEXICAMENTE fora
7    }
8
9    return funcaoInterna;
10 }
11
12 const g = funcaoExterna();
13 g(); // "campus Tauá"
14
15 console.log(exterior, unidade); // ReferenceError: unidade is not defined
```

Como declarar variáveis em JavaScript?

Declaração Automática

Usada quando o escopo da variável não é crítico.

Usando const

Ideal para valores/referências que não devem ser alterados.

Usando var

Adequada para compatibilidade com versões mais antigas do JavaScript.

Usando let

Recomendada para escopo de bloco e flexibilidade.

Operadores

- Operadores aritméticos

Operador	Descrição
=	Operador de atribuição
+, *, -, /	Operadores aritméticos básicos
**	Exponenciação
%	Módulo
--, ++	Decremento e Incremento

Operadores

- Operadores de comparação

Operador	Descrição
==	Igual a
===	Valor e tipo iguais
!=	Diferente
!==	Valor e tipo diferentes
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
?	Operador ternário

Operadores

- Operadores lógicos

Operador	Descrição
&&	AND
||	OR
!	NOT

- Operadores de tipo

Operador	Descrição
typeof	Retorna o tipo de uma variável
instanceof	Retorna verdadeiro se um objeto é uma instância de um tipo de objeto

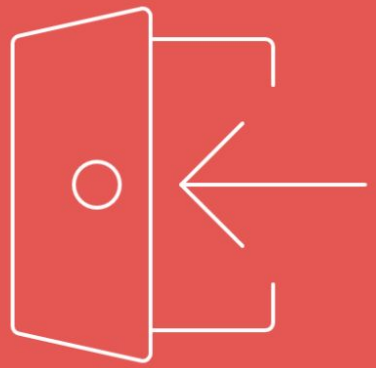
Avaliação de Curto-Circuito (Short-Circuiting)

- **Valores *Truthy* e *Falsy*:**
  - Em JavaScript, **cada valor tem uma interpretação booleana** inerente quando avaliado em um **contexto booleano**.
  - Os valores que são avaliados como `true` são considerados verdadeiros, enquanto os que são avaliados como `false` são considerados falsos.
  - Valores **truthy** (avaliados como verdadeiros) são todos aqueles que não são avaliados como **falsy**.
  - **Valores Falsy em JavaScript:**
    - `0, ''(string vazia), null, NaN, undefined, false`

Avaliação de Curto-Circuito (Short-Circuiting)

- **Avaliação de Curto-Circuito** 🤔
  - O curto-circuito é um comportamento exibido pelos operadores lógicos `&& (and)` e `|| (or)` em que **a avaliação do segundo operando é ignorada se o resultado final já puder ser definido pela avaliação do primeiro operando**.
  - O operador `&&` retorna o primeiro valor **falsy**.
  - O operador `||` retorna o primeiro valor **truthy**.

Qual operador lógico usar para avaliação de curto-circuito?



Operador AND (&&)

Avalia e retorna o primeiro valor falsy

VS

Operador OR (||)

Avalia e retorna o primeiro valor truthy

Avaliação de Curto-Circuito (Short-Circuiting)

- Exemplos com operador &&

```
1  const value = 0;
2  const result = value && 'Truthy Value';
3  console.log(result); // Output: 0
```

```
1  const value = 'Hello';
2  const result = value && 'Truthy Value';
3  console.log(result); // Output: Truthy Value
```

Avaliação de Curto-Circuito (Short-Circuiting)

- Exemplos com operador **||**

```
1  const name = '';
2  const displayName = name || 'Convidado';
3  console.log(displayName); // Output: Convidado
```

```
1  const name = 'Alice';
2  const displayName = name || 'Convidado';
3  console.log(displayName); // Output: Alice
```

Avaliação de Curto-Circuito (Short-Circuiting)

- Vamos para um exemplo mais “real”

```
1  const produtos = [
2    { nome: "Camisa", preco: 29.99 },
3    { nome: "Calça", preco: 49.99 },
4    { nome: "Tênis", preco: 89.99 }
5  ];
6
7  function exibirProdutos(produtos) {
8    console.log(produtos);
9  }
10
11 // Como tem produtos na lista, a função será chamada
12 produtos && exibirProdutos(produtos);
```

```
1  const produtos = null;
2
3  function exibirProdutos(produtos) {
4    console.log(produtos);
5  }
6
7  // Como não tem produtos na lista, a função não será chamada
8  produtos && exibirProdutos(produtos);
```

Avaliação de Curto-Circuito (Short-Circuiting)

- Vamos para um exemplo mais “real”

```
1  const userName = "Carlos";
2
3  let message = "Bem-vindo, ";
4
5  message += userName || "Visitante";
6
7  // Como há um valor em userName,
8  // será exibido "Bem-vindo, Carlos"
9  console.log(message);
```

```
1  const userName = "";
2
3  let message = "Bem-vindo, ";
4
5  message += userName || "Visitante";
6
7  // Como não há um valor em userName,
8  // será exibido "Bem-vindo, Visitante"
9  console.log(message);
```

Tipos de Dados

- Antes de estudar sobre os tipos de dados em JavaScript, vamos entender alguns conceitos importantes sobre **tipagem de dados** em linguagens de programação.

- **O que é um tipo?**

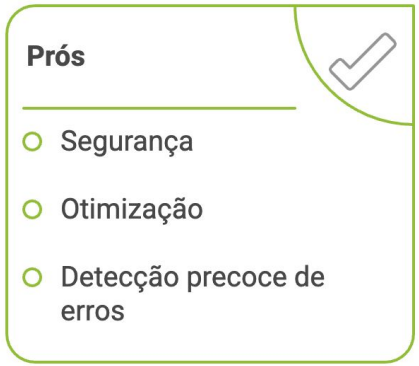
> Uma classificação que determina quais operações são válidas sobre um valor e o significado dessas operações (Pierce, 2002).

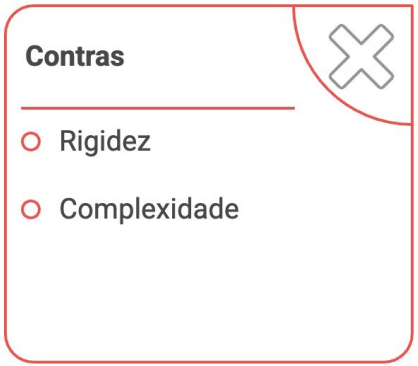
Tipos de Dados

Tipagem Estática

- O tipo de cada variável/expressão é **conhecido em tempo de compilação**.
- Erros de tipo são detectados antes da execução.
- Favorece **segurança e otimização**.
- Ex.: Java, C#, TypeScript (no modo estático).
- ***Exemplo:***

```
let x: number = 10;    // tipo number
x = "dez";     // Erro de compilação em TS
```





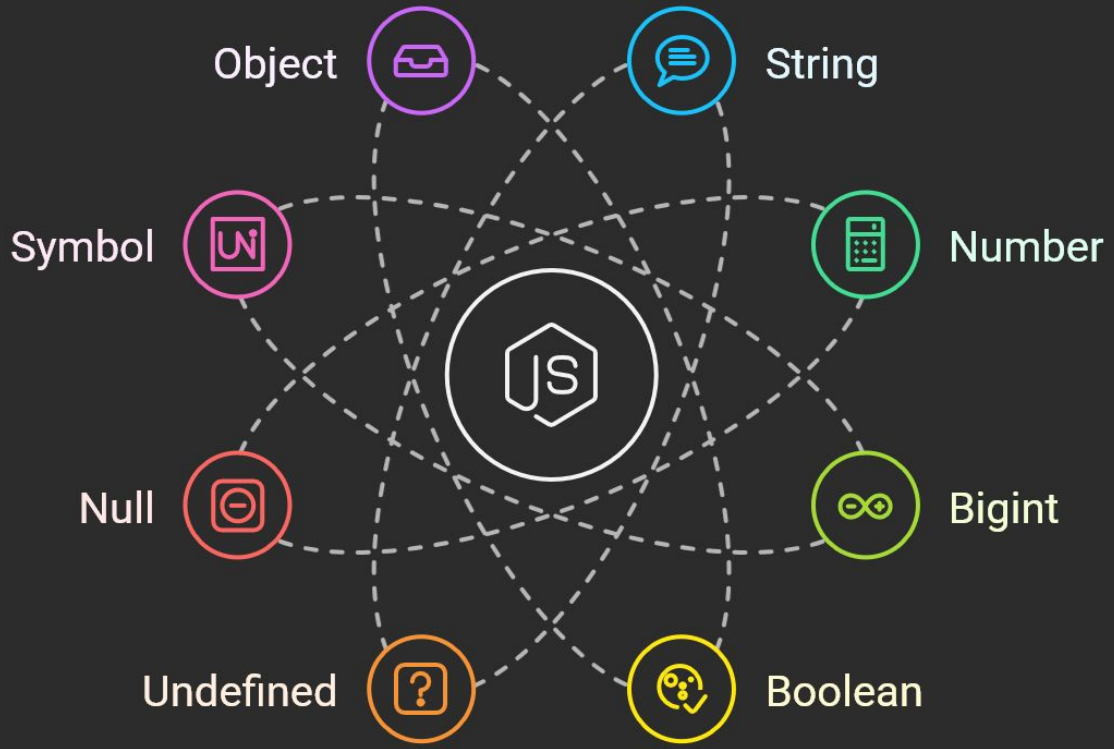
Tipos de Dados

Tipagem Dinâmica

- O tipo é **determinado em tempo de execução**.
- Valores carregam metadados de tipo (*type tags*).
- Mais flexível, mas menos eficiente.
- Ex.: JavaScript, Python, Ruby.
- ***Exemplo:***

```
let x = 10;     // tipo number
x = "dez";      // permitido em JS
```

Visão Geral dos Tipos de Dados JavaScript



Tipos de Dados

- O JS tem 8 tipos de dados:
  - String
  - Number
  - Bigint
  - Boolean
  - Undefined
  - Null
  - Symbol
  - Object

myScript.js

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;
```

Tipos de Dados

- O JS tem 8 tipos de dados:
  - String
  - Number
  - Bigint
  - Boolean
  - Undefined
  - Null
  - Symbol
  - Object

- **O tipo de dado *object* pode conter:**
  - **um objeto**
  - **um array**
  - **uma data**

myScript.js

```
// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

Como lidar com tipos de dados e operações em JavaScript?

Tipos de Dados

- **Coerção de Tipos: quando acontece?**
  - Operadores aritméticos e comparação soltam coerções — conversões automáticas/implícitas (ex.: `+` concatena se houver `string`).
  - **Comparação solta** (`==`) segue regras do “Abstract Equality” (complexas e cheias de casos).

```
1  "2" + 1;      // "21"
2  "2" - 1;      // 1
3  [] + {};      // "[object Object]"
4  [] == 0;      // true   (evite)
5  0 == "";      // true   (evite)
```

    - ***Regra básica:*** *prefira o operador de igualdade estrita* `===` *e conversões explícitas —* `Number()`, `String()`, `Boolean()`.

Tipos de Dados

- **Conversão de Tipos**
  - Variáveis JavaScript podem ser convertidas para uma nova variável e para outro tipo de dados:
    - através de **conversão explícita** com funções; ou
    - pelo próprio JavaScript automaticamente (**conversão implícita / coerção**).
  - **Exercício Rápido**
    - Leia o tutorial de conversão de tipos em JavaScript no site W3Schools: https://www.w3schools.com/js/js_type_conversion.asp
    - Responda o exercício ao final da página no link acima.

Condicionais

- Frequentemente, ao escrever um código, nos deparamos com a necessidade de executar ações que dependem de alguma condição
- No JavaScript temos algumas estruturas de código para realizar controle de fluxo condicinal:
  - `if`
  - `else`
  - `else if`
  - `switch`

Condicionais

- Usamos a estrutura **`if`** quando queremos executar um bloco de código no caso da condição testada retornar verdadeiro **`(true)`**
- Sintaxe:

`myScript.js`

```
if (condition) {
  //  bloco de código que será executado caso a condição seja verdadeira
}
```

Condicionais

- Usamos a estrutura **`else`** quando queremos executar um bloco de código no caso da condição testada retornar falso **`(false)`**
- Sintaxe:

myScript.js

```
if (condition) {
  //  bloco de código que será executado caso a condição seja verdadeira
} else {
  //  bloco de código que será executado caso a condição seja falsa
}
```

Condicionais

- Usamos a estrutura **`else if`** quando queremos testar uma nova condição, caso a primeira condição tenha retornada falsa
- Sintaxe:

myScript.js

```
if (condition1) {
  //  bloco de código que será executado caso a condição 1 seja verdadeira
} else if (condition2) {
  //  bloco de código que será executado caso a condição 1 seja falsa e a condição 2 seja verdadeira
} else {
  //  bloco de código que será executado caso as condições 1 e 2 sejam falsas
}
```

Condicionais

- Usamos a estrutura **`switch`** para selecionar um bloco de código, entre vários, para ser executado a depender do caso de teste
- Sintaxe:

myScript.js

```
switch(expression) {
  case x:
      // bloco de código
      break;
  case y:
      // bloco de código
      break;
  default:
    // bloco de código
}
```

myScript.js

```
switch(expression) {
  case x:
      // bloco de código
      break;
  case y:
      // bloco de código
      break;
  default:
    // bloco de código
}
```

Condicionais

- Usamos a estrutura **`switch`** para selecionar um bloco de código, entre vários, para ser executado a depender do caso de teste
- Exemplo:

myScript.js

```
switch(expression) {
  case x:
      // bloco de código
      break;
  case y:
      // bloco de código
      break;
  default:
    // bloco de código
}
```

myScript.js

```
switch(mes) {
  case 1:
      return "Janeiro";
      break;
  case 2:
      return "Fevereiro";
      break;
  default:
    return "Mês inválido";
}
```

Laços de Repetição

- Outro tipo de estruturas de controle de fluxo bastante úteis são os *loops*, ou laços de repetição
- Com essas estruturas, é possível executar um mesmo trecho de código repetidas vezes, geralmente atuando sobre dados diferentes.
- No JavaScript temos as seguintes estruturas básicas para implementar laços de repetição:
  - `for`
  - `for in`
  - `for of`
  - `while / do while`

Laços de Repetição

- Com a estrutura **`for`** percorremos um bloco de código várias vezes (geralmente uma quantidade específica de vezes)
- Sintaxe:

`myScript.js`

```
for (expression 1; expression 2; expression 3) {
  // bloco de código a ser executado
}
```

- **Expressão 1:** executada ***somente uma vez***, antes da execução do bloco de código, geralmente para inicializar uma variável de controle
- **Expressão 2:** define a ***condição*** para a execução do bloco de código
- **Expressão 3:** executada após a execução do bloco de código, geralmente para alterar o valor da variável de controle

Laços de Repetição: `for`

- Com a estrutura **`for`** percorremos um bloco de código várias vezes (geralmente uma quantidade específica de vezes)
- Exemplo:

myScript.js

```
for (let i = 0; i < 5; i++) {
  console.log("O número é " + i);
  soma += i;
}
```

Laços de Repetição: `for in`

- Com a estrutura **`for in`** conseguimos percorrer as propriedades de um objeto
- Sintaxe:

myScript.js

```
for (key in object) {
  // bloco de código a ser executado
}
```

- **key:** variável utilizada para armazenar o valor de cada propriedade percorrida dentro do objeto, em cada iteração do for
- **object:** referência ao objeto a ser percorrido

Laços de Repetição: `for in`

- Com a estrutura **`for in`** conseguimos percorrer as propriedades de um objeto
- Exemplo:

myScript.js

```
const car = {brand:"Fiat", model:"Toro", year:2024};

for (let p in car) {
  console.log(p);
}
```

Laços de Repetição: `for of`

- Com a estrutura **`for of`** conseguimos percorrer valores de um ***objeto iterável***
- Um ***Objeto Iterável*** é um objeto que pode ser percorrido (**iterado**) de forma sequencial em um laço de repetição
- Sintaxe:

`myScript.js`

```
for (variable of iterable) {
  // bloco de código a ser executado
}
```

- **variable:** variável utilizada para armazenar o valor de cada elemento individual do objeto iterável
- **iterable:** referência ao objeto a ser iterado

Laços de Repetição: `for of`

- Com a estrutura **`for of`** conseguimos percorrer valores de um ***objeto iterável***
- Exemplo:

myScript.js

```
// exemplo de iteração em array
const cars = ["Fiat", "Ford", "Toyota"];

for (let x of cars) {
  console.log(x);
}
```

myScript.js

```
// exemplo de iteração em string
let language = "JavaScript";

for (let x of language) {
  console.log(x);
}
```

Laços de Repetição: `while`

- Com a estrutura **`while`** podemos executar um bloco de código repetidas vezes, **enquanto a condição testada for verdadeira**
- Sintaxe:

`myScript.js`

```
while (condition) {
  // bloco de código a ser executado
}
```

- **condition:** condição que é sempre testada no início de cada iteração. O bloco de código vai ser executado enquanto essa condição retornar **true**.

Laços de Repetição: `while`

- Com a estrutura **`while`** podemos executar um bloco de código repetidas vezes, **enquanto a condição testada for verdadeira**
- Exemplo:

myScript.js

```
let i = 0;

while (i < 10) {
  console.log("O valor de i é: " + i);
  i++;
}
```

Laços de Repetição: `do while`

- A estrutura **`do while`** é uma variante da estrutura **`while`** e também executa um bloco de código repetidas vezes, **enquanto a condição testada for verdadeira**
- Contudo, diferentemente da estrutura while, na estrutura do while, o bloco de código é executado antes da verificação da condição
- Como consequência, o bloco de código é **executado pelo menos uma vez**
- Sintaxe:

myScript.js

```
do {
  // bloco de código a ser executado
} while (condition);
```

Laços de Repetição: `do while`

- A estrutura **`do while`** é uma variante da estrutura **`while`** e também executa um bloco de código repetidas vezes, **enquanto a condição testada for verdadeira**
- Contudo, diferentemente da estrutura while, na estrutura do while, o bloco de código é executado antes da verificação da condição
- Exemplo:

myScript.js

```
let i = 0;

do {
  console.log("O valor de i é: " + i);
  i++;
} while (i < 10);
```

Laços de Repetição: `break` e `continue`

- Em algumas situações é interessante parar a execução de um laço de repetição e sair dele, principalmente quando o objetivo inicial do laço já foi alcançado
- Em outras situações é interessante pular uma iteração do laço e passar diretamente para a iteração seguinte sem terminar de executar o código dentro do laço
- Para essas situações podem ser utilizadas as palavras reservadas break e continue, respectivamente

Laços de Repetição: `break` e `continue`

- Em algumas situações é interessante parar a execução de um laço de repetição e sair dele, principalmente quando o objetivo inicial do laço já foi alcançado
- Em outras situações é interessante pular uma iteração do laço e passar diretamente para a iteração seguinte sem terminar de executar o código dentro do laço
- Para essas situações podem ser utilizadas as palavras reservadas break e continue, respectivamente

myScript.js

```
for (let i = 1; i <= 10; i++) {
  if (i % 5 === 0 ) { break; }
  console.log(i);
}
```

myScript.js

```
for (let i = 1; i <= 10; i++) {
  if (i % 5 === 0 ) { continue; }
  console.log(i);
}
```

Exercício Rápido

- Escreva um programa que calcule a soma dos cinco primeiros números ímpares encontrados no intervalo entre dois valores informados pelo usuário. Se não houver pelo menos cinco números ímpares no intervalo informado, o programa deve alertar isso ao usuário e solicitar um novo intervalo de valores. O programa encerra quando o usuário informar um intervalo que contenha pelo menos cinco números ímpares e a soma deles seja exibida.

Funções

- Uma função nada mais é do que um bloco de código que executa uma tarefa específica quando chamado

myScript.js

```
// Função para calcular o produto de p1 e p2
function myFunction(p1, p2) {
  return p1 * p2;
}
```

- Sintaxe:
  - definida pela palavra reservada **function**, seguida pelo seu **nome** e por **parênteses ()**
  - os parênteses podem incluir nomes de **parâmetros** separados por vírgula
  - o bloco de código da função é colocado dentro de **chaves {}**

Funções

- Uma função também pode ser definida utilizando uma **expressão**:

myScript.js

```
// Função para calcular o produto de p1 e p2
const x = function (p1, p2) {return p1 * p2};
let z = x(4, 3);
```

- A abordagem acima é conhecida como **função anônima**

Funções: parâmetros com valor default

- O JavaScript permite definir valores padrão (default) para parâmetros de funções

myScript.js

```
// Se não o argumento y não é passado, então y = 10 por padrão
function myFunction(x, y = 10) {
  return x + y;
}
myFunction(5);
```

Funções: rest parameters

- O recurso de **rest parameters (...)** permite uma função tratar um número indefinido de argumentos como um array

myScript.js

```
// O parâmetro args encapsula um array com os argumentos passados
function sum(...args) {
  let sum = 0;
  for (let arg of args) sum += arg;
  return sum;
}

let x = sum(4, 9, 16, 25, 29, 100, 66, 77);
```

Funções: chamada e retorno

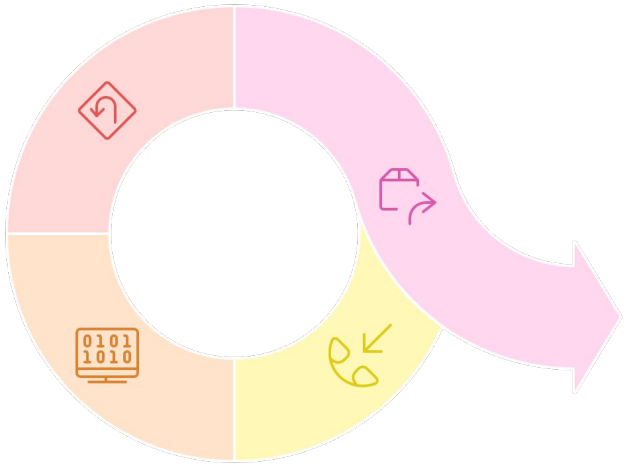
- O código de uma função é executado quando ela é chamada (invocada), o que pode ocorrer quando:
    - um evento ocorre (quando o usuário clica em um botão, por exemplo);
    - é chamada a partir de uma instrução de código JavaScript;
    - automaticamente.
- Uma função para de executar quando encontra a instrução **return**. Essa instrução é utilizada quando queremos retornar um valor computado na função para quem a chamou, como no exemplo a seguir:

myScript.js

```
// A função é chamada e o valor retornado é guardado na variável x
let x = myFunction(4, 3);

function myFunction(a, b) {
// A função retorna o produto de a e b
  return a * b;
}
```

Ciclo de Execução de Função



1

Função Chamada

A função é invocada para execução.

2

Execução de Código

O código dentro da função é executado.

3

Instrução de Retorno

A função encontra uma instrução de retorno.

4

Retornar Valor

A função retorna um valor ao chamador.

Referências

Documentação e artigos on-line

- **MDN Web Docs — JavaScript:** https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
- **W3Schools — JavaScript:** https://www.w3schools.com/js/
- **How Does Short-Circuiting Work in JavaScript? —** https://www.freecodecamp.org/news/short-circuiting-in-javascript/

Livros

- **Silva, Maurício Samy.** ***JavaScript: Guia do Programador.*** São Paulo: Novatec.
- **Flanagan, David.** ***JavaScript: O Guia Definitivo.*** Porto Alegre: Bookman. *(tradução da 6ª ed.)*