Tecnologias Web

Tecnologias e Sistemas de Informação Para a Web

Politécnico
do Porto
Escola
Superior
de Media Artes e
Design

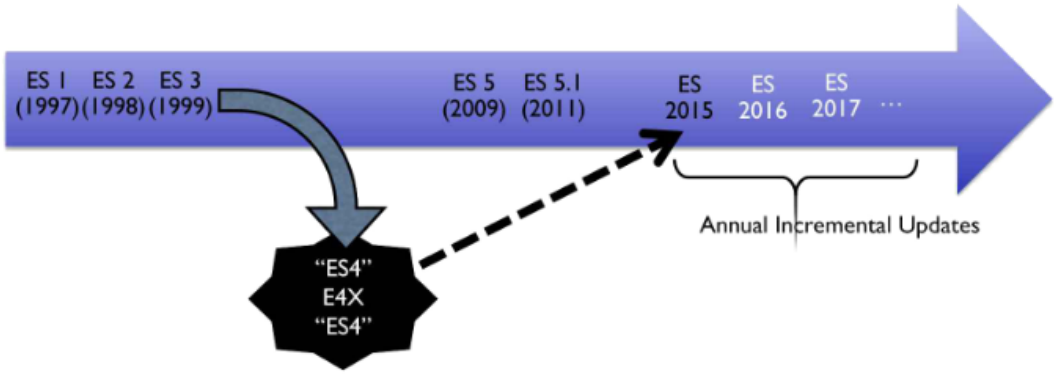
2022/2023

M04 -JavaScript

Mo4 - JavaScript

Introdução ao JavaScript

- **ECMAScript**: especificação de linguagem de script padronizada pela ECMAScript International



- A especificação **ES2015** (6ª edição do standard ECMA-262) é uma das mais populares
- Desta especificação nasceram várias implementações (**JavaScript**, Jscript e ActionScript)
- Nesta sebenta descreve-se a sua implementação em **JavaScript**

Introdução ao JavaScript

- Linguagem de programação **interpretada**
- Criada por **Brendan Eich** (programador da Netscape) em 1995

- Começou por se chamar Mocha, depois LiveScript, e finalmente, **JavaScript**.

- Originalmente implementada para ser usada em browsers
- Hoje em dia já é usada em:
  - servidores web (Node.JS)
  - aplicações desktop (Electron, ...)
  - aplicações mobile (React Native, ...)

Integração numa página HTML Para usar JavaScript numa página HTML use o elemento **script**

- **De forma direta no elemento head ou body**

```
<script>
  // código JavaScript
</script>
```

- **Como referência a ficheiro externo (ficheiro.js)**

```
<script src="myscript.js">
```

Integração numa página HTML: referência a ficheiro externo

```
<script src="myscript.js">
```

- Script vai comportar-se como se estivesse onde a tag script está localizada
- O ficheiro externo não pode conter o elemento <script>
- Vantagens:
  - Separa o HTML do código
  - Tornam o HTML e o JavaScript mais fáceis de ler/manter
  - Permite ao browser fazer cache dos ficheiros JS, acelerando o carregamento das páginas.

Depuração de dados

- Escrita para a consola do browser
  - Escreva o seguinte trecho de código no elemento <head> ou <body>

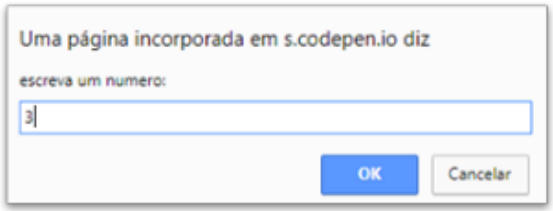
```
<script>
  console.log(5 + 6)
</script>
```

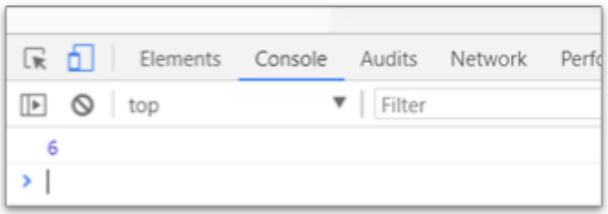
- Abra um browser (Chrome)
- Ative as ferramentas do programador (no Chrome: **ctrl+shift+i**)
- Seleccione o separador **Console**
- Visualize os resultados que surgem na consola

Depuração de dados

- Exibição de modal através de função prompt
  - Escreva o seguinte trecho de código no elemento <head> ou <body>

```
<script>
  var num = prompt("escreva um numero:")
  console.log(parseInt(num) * 2)
</script>
```





Sintaxe

Sintaxe

- Conjunto de regras para escrever programas
- Cada especificação de linguagem define a sua própria sintaxe
- Código JavaScript organizado por instruções executadas na ordem pela qual foram de definidas

```
var price = 5
console.log(price)
```

- Se várias na mesma linha, estas devem ser separadas por ponto e vírgula (;)

```
var price = 5; console.log(price)
```

Sintaxe

Um programa de JavaScript pode ser composto por:

- **Variáveis**: bloco de memória com nome que pode armazenar valores para o programa
- **Literais**: valores constantes/fixos
- **Operadores**: símbolos que definem como os operandos serão processados
- **Keywords**: palavras com significado especial no contexto duma linguagem (var, for, etc.)
- **Módulos**: blocos de código que podem ser reutilizados em diferentes programas/scripts
- **Comentários**: usado para melhorar a legibilidade do código
- **Identificadores**: nomes dados aos elementos dum programa, como variáveis, funções, etc.

Sintaxe - Comentários

- Usados para tornar o código mais legível
- Usados também para prevenir a execução de código quando são testadas outras alternativas
- Linha simples, texto a partir de **//** até ao final da linha é ignorado.

```
// um comentário
var y = 5
var z = y + 1 // outro comentário
console.log(price)
```

- Múltiplas linhas, agregado por **/*** e ***/**.

```
var y = 5
/*
var z = y + 1
console.log(z)
*/
```

Sintaxe - Variáveis

- São contentores para armazenamento de valores
- Os nomes das variáveis são chamados de identificadores:
  - Podem ser nomes curtos (como x e y), ou nomes mais descritivos (idade, soma, totalVolume)
- Variáveis com mais que uma palavra podem ser dificeis de ler
- **convenções de nomeação**:
  - **camelCase**
  - **snake_case**
  - **spinal-case**
  - **PascalCase**

Sintaxe - Variáveis

- Podem conter letras, números, underscores e sinais de dólar
- Não podem começar com um número
- São sensíveis a maiúsculas e minúsculas (y e Y são variáveis diferentes)
- Não podem ser usadas palavras reservadas (**Keywords** em JavaScript)

Sintaxe - Variáveis

- **Declaração:**
  - Uso da palavra-chave **var** para declaração de uma variável
  - Após a declaração, a variável não tem valor (tecnicamente ela tem o valor **undefined**).

```
var price
```

- **Atribuição:**
  - Para atribuir um valor a uma variável , use o operador de atribuição **=**

```
var price = 5
console.log(price)
------------------------------------------------------
5
```

Sintaxe - Variáveis

- Variáveis em JavaScript podem conter qualquer tipo de dados
- Ao contrário de outras linguagens, não é necessário especificar durante a declaração de variável qual o tipo de valor que esta irá manter.
- O tipo de valor de uma variável pode mudar durante a execução dum programa e o JavaScript gere isso de forma automática
- Este recurso é denominado como **tipagem dinâmica**

```
var price = 5
console.log(price)
price = "car"
console.log(price.toUpperCase())
----------------------------------------------------
5
CAR
```

Sintaxe - Variáveis

- O escopo de uma variável (scope) é a região do programa onde é definida
- Tradicionalmente, o JavaScript define apenas dois escopos:
  - **Global**: variável pode ser acedida a partir de qualquer parte do código JavaScript
  - **Local**: variável pode ser acedida a partir de uma função onde é declarada

```
var num = 10
function test() {
  var num = 100
  console.log(num)
}
console.log(num)
test()
console.log(num)
-----------------------------------------------------
10
100
10
```

- **Hoisting**
  - A declaração duma variável em runtime é movida para o topo do âmbito onde foi definida

```
var idade = 20
if (idade == 30) {
  var teste = "ola"
  console.log(teste)
}
console.log(teste)
------------------------------
undefined
```

```
var idade = 20
var teste // hoisting
if (idade == 30) {
  teste = "ola"
  console.log(teste)
}
console.log(teste)
------------------------------
undefined
```

Sintaxe - Variáveis

- **var** atribui um **escopo de função ou global** à variável
- **let** permite que o script restrinja o acesso à variável ao **bloco envolvente mais próximo**
- O **escopo de bloco** restringe o acesso de uma variável ao bloco no qual ele é declarado (**adicionado no ES6**)

```
var idade = 20
if (idade == 30) {
  let teste = "ola"
  console.log(teste)
}
console.log(teste)
-------------------------------------------------------------------------
Uncaught ReferenceError: teste is not defined
```

```
var idade = 30
if (idade == 30) {
  let teste = "ola"
  console.log(teste)
}
console.log(teste)
-------------------------------------------------------------------------
ola
Uncaught ReferenceError: teste is not defined
```

Sintaxe - Variáveis

- O ES6 adicionou um novo tipo de declaração de variáveis: const
- **const** cria uma referência somente de leitura para um valor
- Valor duma constante não pode mudar por reatribuição e não pode ser redeclarado
- Em termos de escopo comporta-se como a keyword let

```
let idade = 72
const MAX = 60
if (idade > MAX) {
  console.log("maior de idade")
}
MAX = 65
----------------------------------------------------------------
maior de idade
Uncaught TypeError: Assignment to constant variable.
```

Sintaxe - Operadores

- **Operadores aritméticos**
  - São usados para realizar operações aritméticas em números (literais ou variáveis):
    - Operações básicas: **+**, **-**, *****, **/**
    - Resto da divisão: **%**
    - Incremento e decremento: **++** e **- -**
- **Operadores de atribuição**
  - Atribuem valores a variáveis
    - Atribuição: **=**
    - Operação e atribuição: **+=**, **-=**, ***=**, **/=**, **%=**

- **Operadores string**
  - Concatenação: **+**
  - Adição de um número com uma string resulta numa string

```
let name = "Rui"
let surname = "Silva"
let fullName = name + " " + surname
console.log(fullName)
-------------------------------------------------------
Rui Silva
```

```
let x = 5 + 5
let y = "5" + 5
let z = "ola" + 5
console.log(x)
console.log(y)
console.log(z)
-------------------------------------------------------
10
55
ola5
```

Sintaxe - Operadores

- **Operadores relacionais**
  - **==** igual a
  - **===** igual a, e do mesmo tipo.
  - **!=** diferente
  - **!==** diferente no valor ou no tipo
  - **>** maior
  - **<** menor
  - **>=** maior ou igual a
  - **<=** menor ou igual a
- **Operadores lógicos**
  - **&&** conjunção (and)
  - **||** disjunção (or)
  - **!** negação (not)

- **Operador condicional (ternary)**
  - Define testes simples
  - Sintaxe: **Test ? Expr1 : Expr2**

```
let num = 12
let result = num > 0 ? "positivo" : "negativo"
console.log(result)
----------------------------------------
positivo
```

- **Operador de tipo**
  - Operador unário typeof
  - Retorna o tipo de dados do operando
  - Lista os tipos de dados e os valores retornados pelo operador typeof em JavaScript
    - **number**
    - **string**
    - **boolean**
    - **object**

```
let num = 12
let result = num > 0 ? "positivo" : "negativo"
console.log(typeof num)
-----------------------------------------
number
```

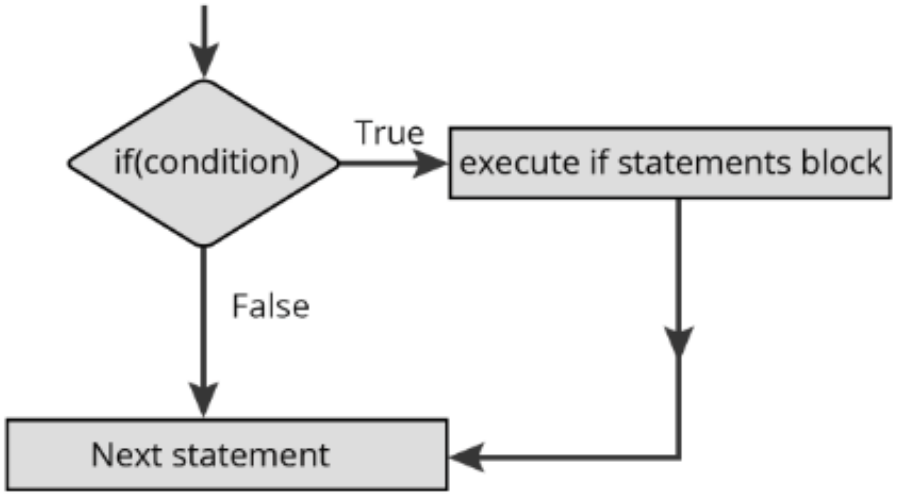
Sintaxe - Estruturas de Decisão

- Executam diferentes ações baseadas em diferentes condições
- Em JavaScript temos as seguintes estruturas:
  - **if**: para especificar bloco de código a ser executado, se determinada condição é verdadeira.
  - **else**: para especificar bloco de código a ser executado, se a mesma condição é falsa.
  - **else...if**: para especificar uma nova condição para testar, se a primeira condição é falsa.
  - **switch**: para especificar blocos de código alternativos a serem executados

Sintaxe - Estruturas de Decisão

- **if**: para especificar bloco de código a ser executado se determinada condição é verdadeira

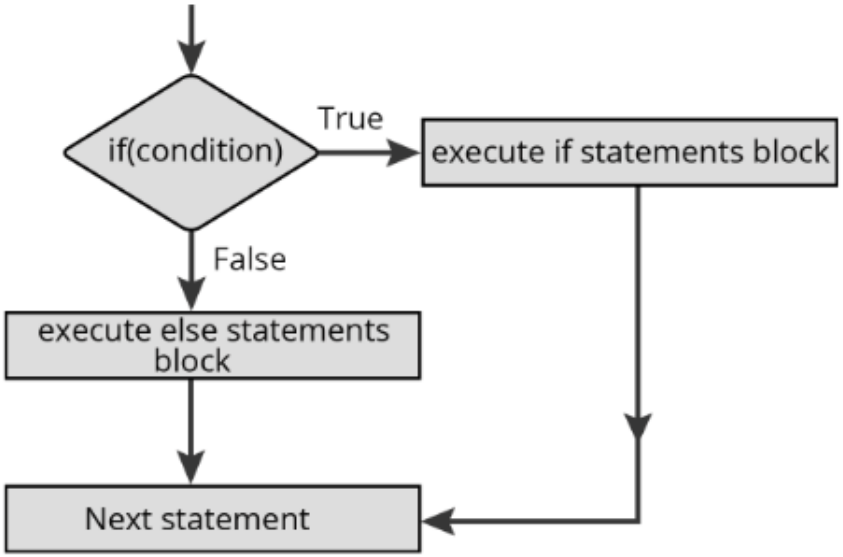
```
let num = 5
if (num > 0) {
  console.log("número é positivo")
}
-----------------------------------
número é positivo
```



Sintaxe - Estruturas de Decisão

- **else**: para especificar bloco de código a ser executado, se a mesma condição é falsa.

```
let num = 12
if (num % 2 == 0) {
  console.log("par")
} else {
  console.log("ímpar")
}
-----------------------------------
par
```

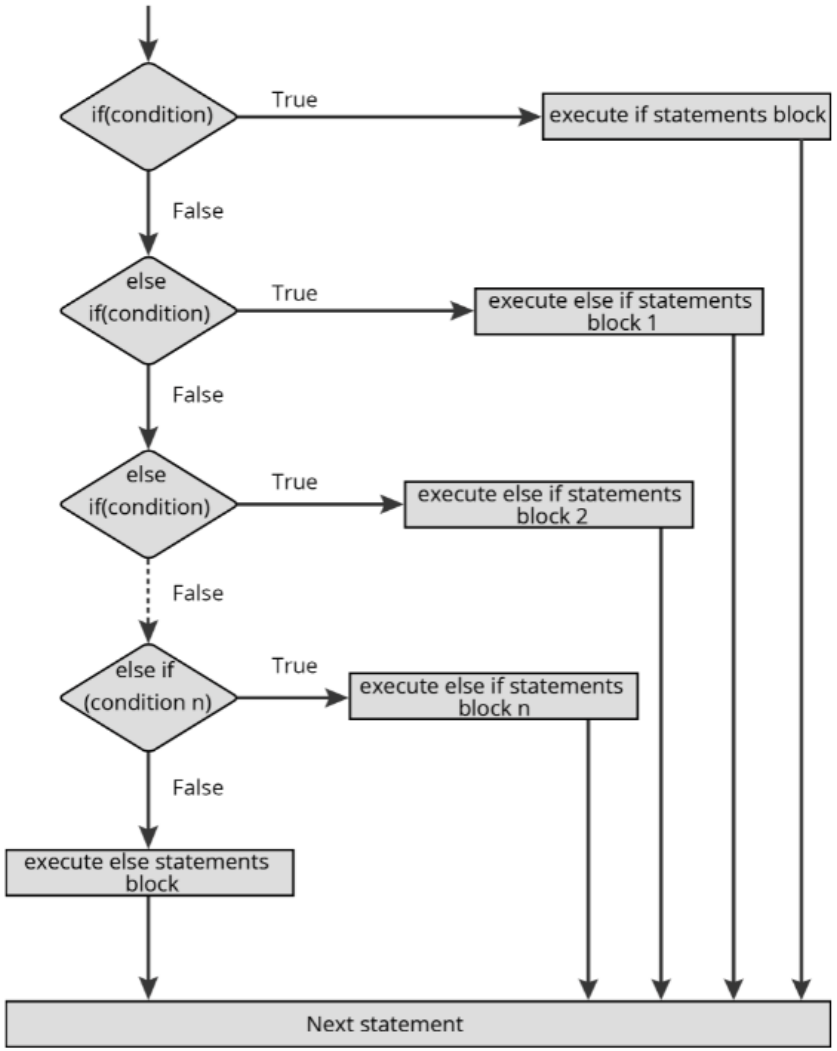


- **else...if**: para especificar testes múltiplos

```
let num = 8
if (num >= 10) {
  console.log("positiva")
} else if (num >= 7) {
  console.log("oral")
} else {
  console.log("negativa")
}
-----------------------------------
oral
```

Sintaxe - Estruturas de Decisão

- **else...if**

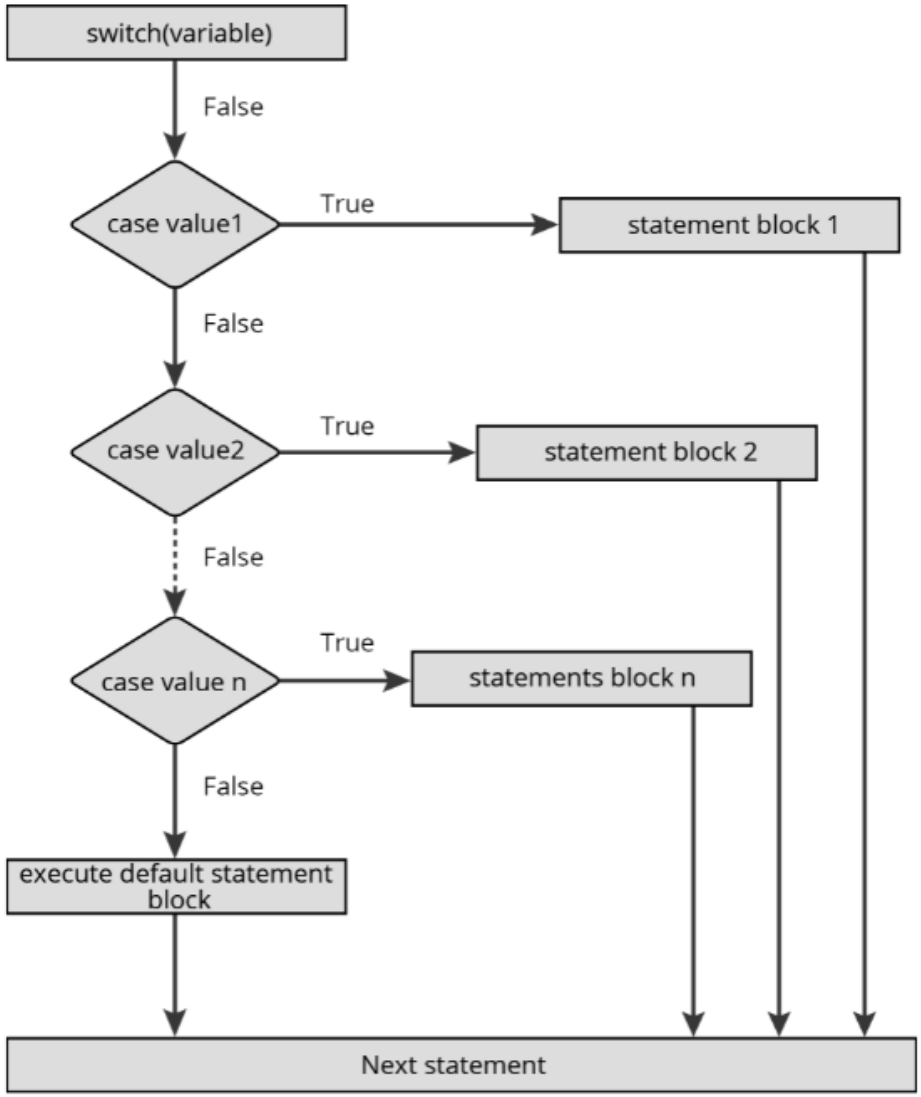


Sintaxe - Estruturas de Decisão

- **switch**: avalia uma expressão, faz o matching do valor da expressão para uma cláusula case e executa as instruções associadas a esse caso.

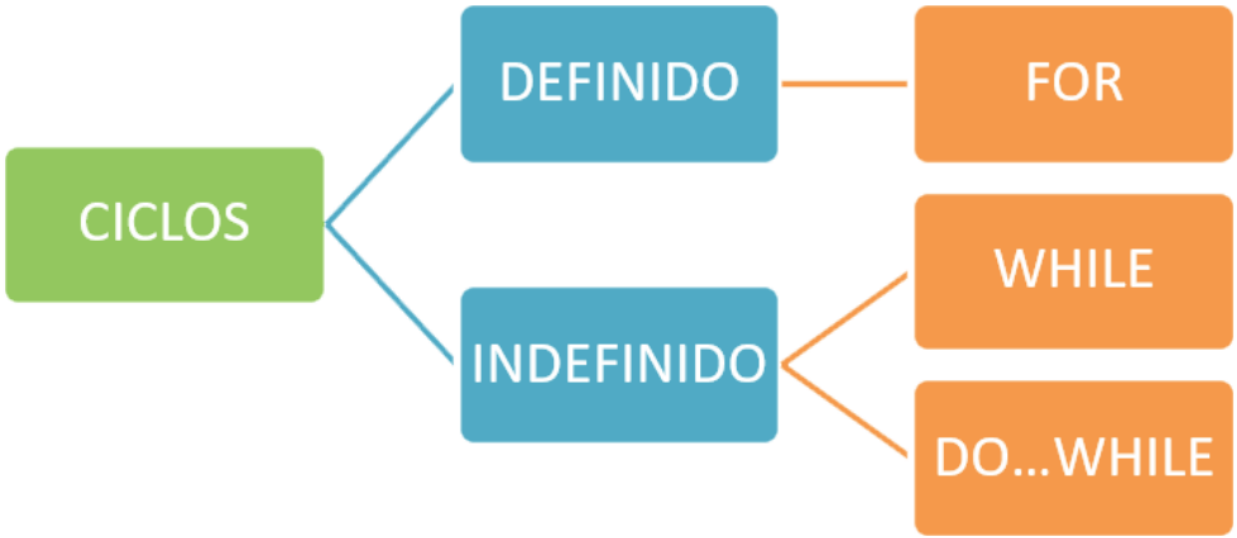
```
let grade = "C"
switch (grade) {
  case "A":
    console.log("excelente")
    break
  case "B":
    console.log("bom")
    break
  case "C":
    console.log("médio")
    break
  case "D":
    console.log("fraco")
    break
  default:
    console.log("escolha inválida")
}
----------------------------------------
médio
```

- **switch**


switch(variable)
False
case value1
True
statement block 1
False
case value2
True
statement block 2
False
case value n
True
statements block n
False
execute default statement block
Next statement

Sintaxe - Estruturas de Repetição

- Por vezes, certas instruções requerem **execução repetida**
- Os **ciclos** são uma maneira ideal de reproduzir esse efeito
- Um ciclo representa um conjunto de instruções que devem ser repetidas
- No contexto de um ciclo, uma repetição é denominada como uma **iteração**.
- **Classificação de ciclos:**



Ciclo definido

- Um ciclo cujo número de iterações são definidas/fixadas
- O **ciclo** for é uma implementação de um ciclo definido
- Três variantes:
  - **for**: executa o bloco de código por um número específico de vezes
  - **for...in**: usado para percorrer as propriedades de um objeto
  - **for...of**: usado para iterar sobre iteráveis em vez de literais de objeto

```
for (let i = 0; i < 5; i++) {
  console.log(i)
}
----------------------------------
0
1
2
3
4
```

Ciclo definido

```
var arr = ["John", "Paul", "Mary"]

for (var i in arr) {
  console.log(i)
}
-----------------------------------
0
1
2
```

```
var arr = ["John", "Paul", "Mary"]

for (var i of arr) {
  console.log(i)
}
-----------------------------------
John
Paul
Mary
```

Ciclo indefinido

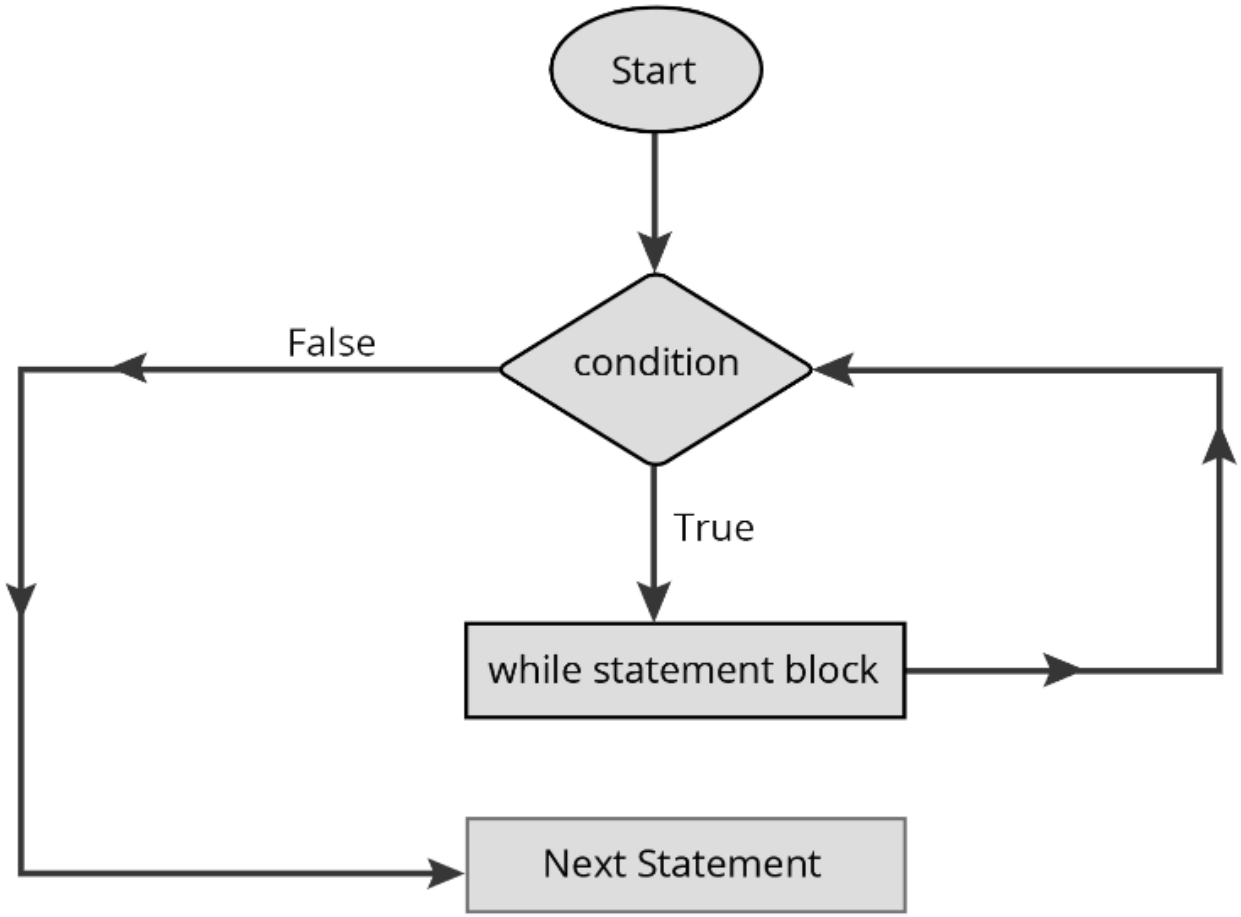
- Usado quando o número de iterações num ciclo é indeterminado ou desconhecido
- Duas variantes:
  - **while**: executa as instruções cada vez que a condição especificada é avaliada como verdadeira
  - **do...while**: é similar ao anterior, exceto que não avalia a condição pela 1ª vez que o ciclo é executado.

Ciclo while

- Executa as instruções sempre que a condição especificada é verdadeira

```
let num = 5
let factorial = 1
while (num >= 1) {
  factorial = factorial * num
  num--
}
console.log("O factorial é " + factorial);
---------------------------------------
O factorial é 120
```

Ciclo while

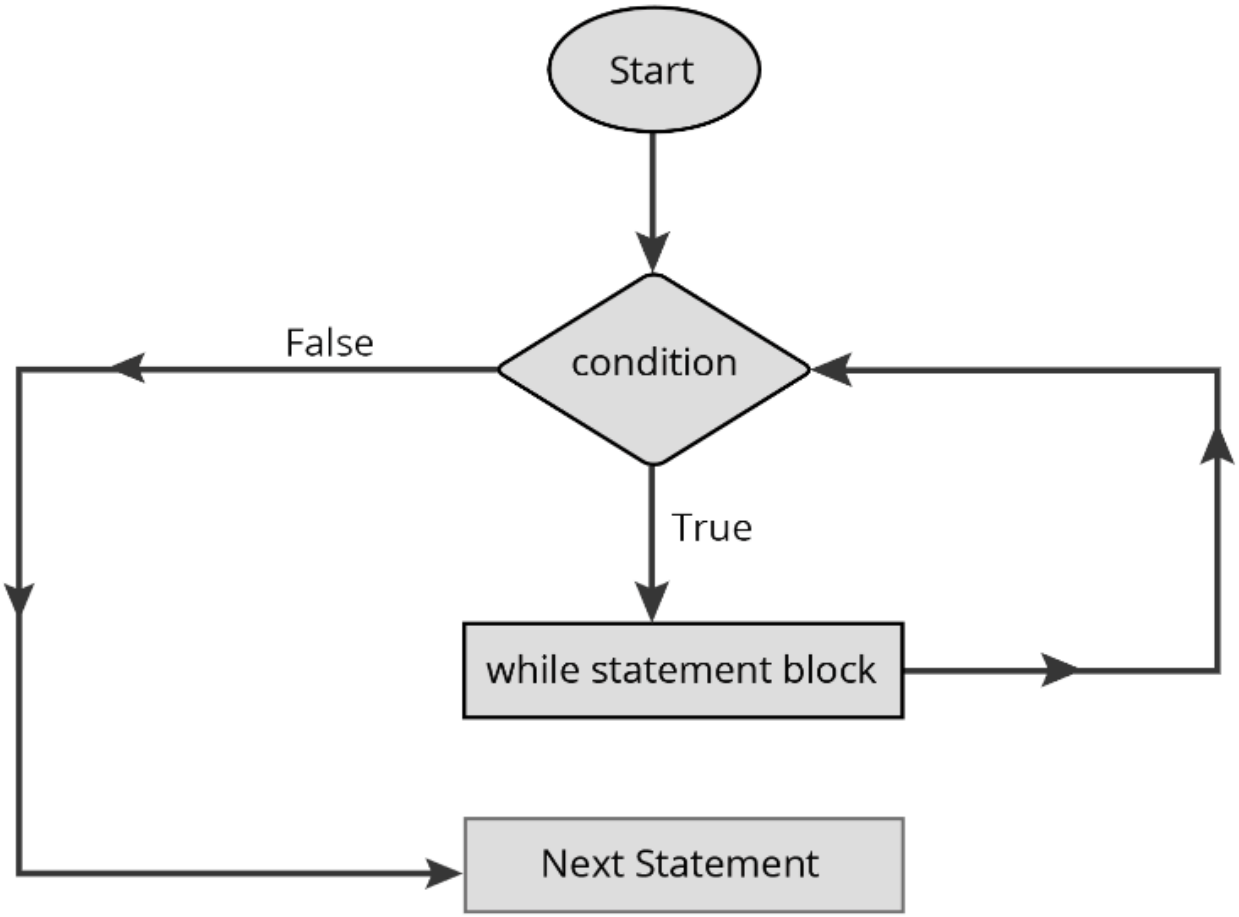

Start
condition
False
True
while statement block
Next Statement

Ciclo do...while

- Semelhante ao ciclo while, exceto que o ciclo do... while não avalia a condição pela 1ª vez que o ciclo é executado.
- No entanto, a condição é avaliada para as iterações subsequentes.
- Em suma, o bloco de código será executado pelo menos uma vez neste tipo de ciclos.

```
let n = 5
do {
  console.log(n)
  n--
} while (n >= 0)
---------------------------------
5
4
3
2
1
0
```

Ciclo do...while



Sintaxe - Estruturas de Repetição

- Os ciclos possuem keywords de **controlo de fluxo**:
  - **break**: usado para sair de um ciclo
  - **continue**: ignora as declarações subsequentes na iteração atual e leva o controlo de volta ao início do ciclo

```
/*
  Exemplo break
*/
let i = 1
while (i <= 10) {
  if (i % 5 == 0) {
    console.log("O 1º múltiplo de 5 entre 1 e 10 é " + i)
    break     // Sai do ciclo se o 1º múltiplo é encontrado
  }
  i++
}
--------------------------------------------------------------------------------
O 1º múltiplo de 5 entre 1 e 10 é 5
```

Sintaxe - Estruturas de Repetição

```
/*
  Exemplo continue
*/
let count = 0
for (let num = 0; num <= 20; num++) {
  if (num % 2 == 0) {
    continue
  }
  count++
}
console.log("A contagem de nºs ímpares entre 0 e 20 é " + count)
-------------------------------------------------------------------------
A contagem de nºs ímpares entre 0 e 20 é 10
```

Sintaxe - Funções

- Funções são blocos de construção do código legível, sustentável e reutilizável.
- São definidas usando a palavra-chave **function**

```
// definição da função
function test() {
  console.log("isto é um teste")
}
test() // invocação da função
-------------------------------------------------
isto é um teste
```

- **Funções** podem ser classificadas como:
  - Funções **com retorno**
  - Funções **parametrizadas**

Funções com retorno

- Retornam um valor de volta a quem a invocou
- Uso da keyword **return**

```
function saudacao() {
  return("olá mundo!")
}
let msg = saudacao()
console.log(msg)
------------------------------------------------------------------------
olá mundo!
```

- Características principais::
  - Deve terminar com uma declaração de retorno
  - Pode retornar no máximo um valor. Noutras palavras, pode haver apenas uma declaração de retorno por função.
  - A declaração de retorno deve ser a última declaração na função

Funções parametrizadas

- Os parâmetros:
  - São um mecanismo para passar valores para funções
  - Formam parte da assinatura da função
  - Os seus valores são passados para a função durante a sua invocação
- A menos que explicitamente especificado, o número de valores passados para uma função deve corresponder ao número de parâmetros definidos.

```
function add(n1, n2) {
  let sum = n1 + n2
  return sum
}
console.log(add(3, 4))
----------------------------------------------------------------------
7
```

Sintaxe - Funções

Com o ES6 aparecem novos tipos de funções

- **Funções com parâmetros por omissão**
- **Funções com parâmetros rest**
- **Funções anónimas**
- **Funções lambda (arrow functions)**
- **Immediately Invoked Function Expression (IIFE)**
- **Funções geradoras**

```
let foo = (x) => 5 + x
console.log(foo(10))
---------------------------------
15
```

Funções com parâmetros por omissão

- Permite que os parâmetros sejam inicializados com valores padrão, se nenhum valor for passado ou não for definido.

```
function add(n1, n2 = 3) {
  let sum = n1 + n2
  return sum
}
console.log(add(3))
console.log(add(3, 4))
-----------------------------------------------------------
6
7
```

Funções com parâmetros rest

- Não restringem o número de valores que se pode passar para uma função. No entanto, os valores passados devem ser todos do mesmo tipo.
- Noutras palavras, os **parâmetros rest** actuam como espaços reservados para vários argumentos do mesmo tipo.
- **Nota**: os parâmetros rest devem ser os últimos de uma lista de parâmetros

```
function functionWithRestParams(...params) {
  console.log(params.length)
}
functionWithRestParams()
functionWithRestParams(5)
functionWithRestParams(5, 6, 12)
-----------------------------------------------
0
1
3
```

Funções anónimas

- Não vinculadas a um identificador (nome da função)
- Declaradas dinamicamente no tempo de execução
- Podem aceitar inputs e retornar outputs
- Podem ser atribuídas a variáveis
- A declaração é chamada de **expressão de função**

```
let f = function () { return "olá" }
console.log(f())
-----------------------------------
olá
```

```
let f = function (x, y) { return x * y }
function product() {
  let result
  result = f(10, 20)
  return result
}
console.log(product())
----------------------------------------
200
```

Funções lambda (ou funções arrow)

- São um mecanismo conciso para representar **funções anónimas**
- Também chamadas como funções arrow (**=>**), existem em duas variantes:
  - Expressões lambda: expressão de **FA** que aponta para uma única linha de código
  - Declarações lambda: expressão de **FA** que aponta para um bloco de código

```
let foo = (x) => 5 + x
console.log(foo(10))
---------------------------
15
```

```
let msg = () => {
  console.log("função invocada")
}
msg()
---------------------------------
função invocada
```

Funções lambda (ou funções arrow)

```
let foo = (x) => 5 + x
console.log(foo(10))
----------------------------
15
```

```
let msg = () => {
  console.log("função invocada")
}
msg()
---------------------------------
função invocada
```

- **Notas:**
  - Parêntesis opcionais para parâmetro único
  - Parêntesis vazios para zero parâmetros
  - Chavetas opcionais para instrução única

Sintaxe - Eventos

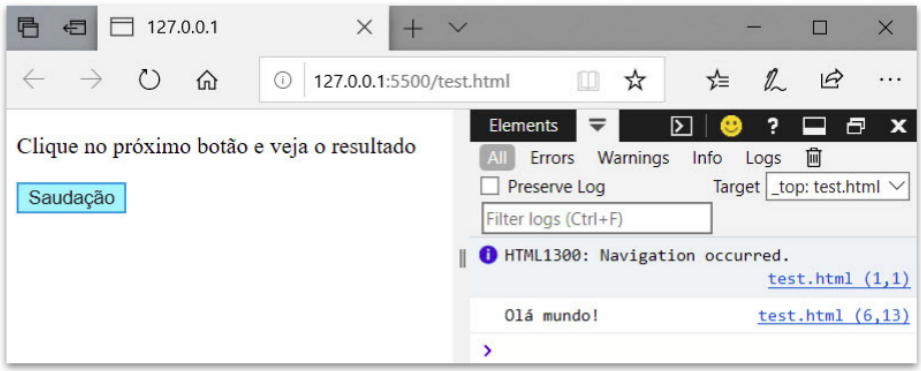
- O JavaScript adiciona interatividade às páginas html através de **eventos**:
  - São parte do nível 3 do **DOM (Document Object Model)** onde cada elemento HTML contém um conjunto de eventos que podem desencadear código JavaScript
  - Acção ou ocorrência reconhecida pelo software
  - Pode ser acionado por um utilizador ou pelo sistema
- **Exemplos**:
  - Utilizador a clicar num botão ou num link
  - Carregamento da página Web

Sintaxe - Eventos

- Podemos definir como os **eventos** serão processados em JavaScript usando manipuladores de eventos (**event handlers**)
- Principais tipos de eventos:
  - **onclick**
  - **onsubmit**
  - **onmouseover/onmouseout**
  - **onload**
  - **onkeypress**
  - **...**

Evento onclick

```
<!DOCTYPE html>

<head>
  <script type="text/javascript">
    function sayHello() {
      console.log("Olá mundo!")
    }
  </script>
</head>

<body>
  <p>Clique no próximo botão e veja o resultado</p>
  <input type="button" onclick="sayHello()" value="Saudação" />
</body>

</html>
```

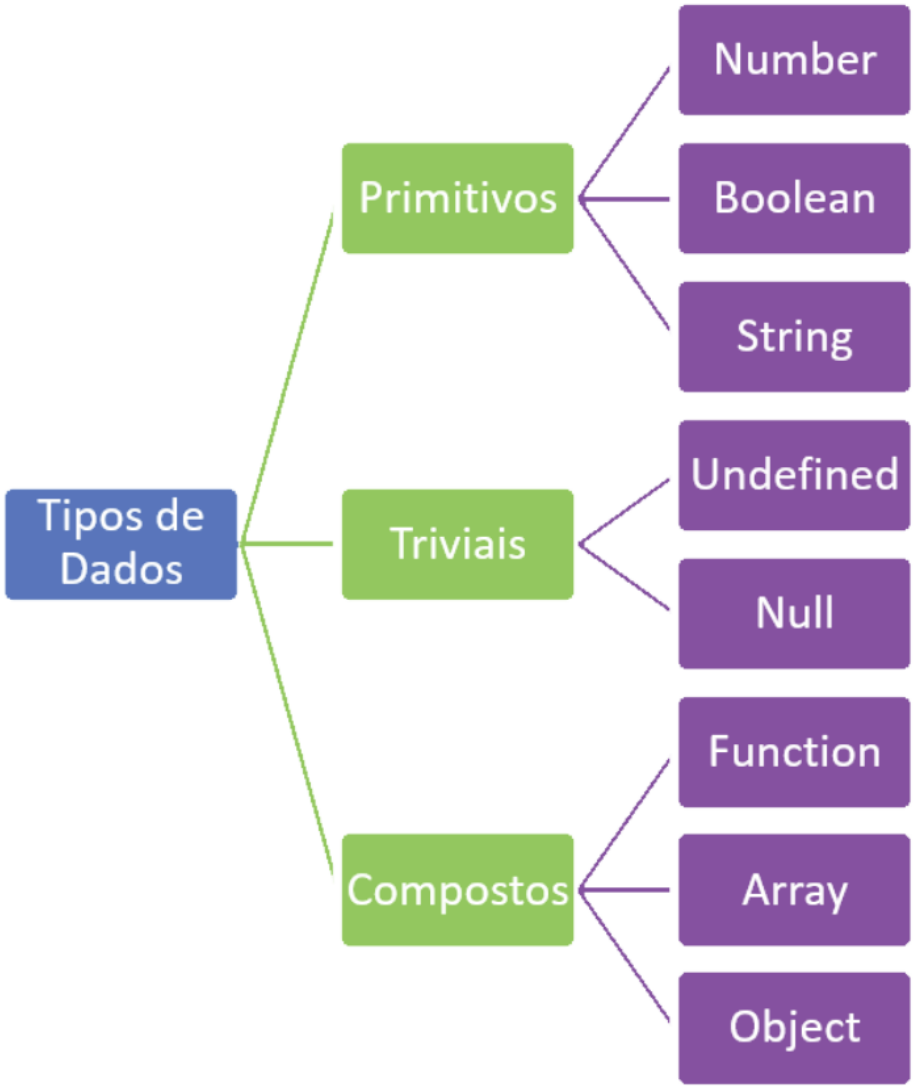


Evento onsubmit

```
<!DOCTYPE html>

<head>
  <script type="text/javascript">
    function validate() {
      // validação aqui
      return true // ou false
    }
  </script>
</head>

<body>
  <form method="POST" action="run.php" onsubmit="return validate()">
    <!-- ... -->
    <input type="submit" value="submeter" />
  </form>
</body>

</html>
```

Tipos de Dados

Tipos de Dados



Tipos de Dados

- **Primitivos**: valor de dados simples sem propriedades e métodos adicionais

```
console.log(typeof 3.14)
console.log(typeof "esmad")
console.log(typeof true)
console.log(typeof undefined)
-------------------------------------------------
number
string
boolean
undefined
```

- **Compostos**: valor de dados complexos com propriedades e métodos adicionais

```
console.log(typeof { name: "John Doe", age: 30 })
console.log(typeof [1, 2, 3, 4])
console.log(typeof function myFunc() { })
-------------------------------------------------
object
object
function
```

Tipos de Dados - Number

- O JavaScript suporta apenas um tipo de valor númerico: **number**
- Um número pode ter ou não casas decimais

```
let x = 34.02
let y = 12
```

- Tipo de dados **number** tem precisão até 15 dígitos

Tipos de Dados - Number

- **Infinity**: valor que o JavaScript retorna se calcular um número acima do maior número possível

```
let x = 2 / 0
let y = -2 / 0
console.log(x)
console.log(y)
-------------------------------------------------------------
Infinity
-Infinity
```

- Infinity é um número: typeof de Infinity devolve number

```
console.log(typeof Infinity)
-------------------------------------------------------------
number
```

Tipos de Dados - Number

- **NaN** é uma palavra reservada para um valor não numérico
- Tentando fazer aritmética com uma string não numérica resultará em **NaN (Not a Number)**

```
let x = 100 / "esmad"; console.log(x);
-------------------------------------------------------------
NaN
```

- Contudo, se a string contém um valor numérico, o resultado será um número.

```
let x = 100 / "10"; console.log(x);
-------------------------------------------------------------
10
```

- Pode-se usar a função global **isNaN** para aferir se o valor é um número

```
let x = 100 / "esmad"; console.log(isNaN(x));
-------------------------------------------------------------
true
```

Tipos de Dados - Number

- Referências online
  - **MDN**[1]
  - **W3Schools**[2]

[1] https://developer.mozilla.org/(...)/Reference/Global_Objects/Number
[2] https://www.w3schools.com/jsref/jsref_obj_number.asp

Tipos de Dados - String

- Strings são usadas para armazenar e manipular texto (sequência de caracteres)
- Um valor string deve estar entre aspas ou plicas

```
let school = "esmad"
let name = 'miguel'
let restaurant = "McDonald's"
```

- Aferir tamanho de uma string: propriedade length

```
let school = "esmad"
console.log(school.length)
-------------------------------------------------
5
```

- Caracteres especiais: uso de caracter de escape ****

```
let y = "We are the \\Vikings\' \n from the north."
console.log(y)
-------------------------------------------------------
"We are the \Vikings'
 from the north."
```

Tipos de Dados - String

Procura de texto

- **indexOf(str)**: devolve a posição da primeira ocorrência de um texto numa string
- **lastIndexOf(str)**: devolve a posição da última ocorrência

```
let msg = "Eu gosto de Vila do Conde e gosto da Póvoa de Varzim"
let firstPos = msg.indexOf("gosto")
let lastPos = msg.lastIndexOf("gosto")
console.log(firstPos)
console.log(lastPos)
-----------------------------------------------------------------
3
28
```

- As posições de caracteres numa string iniciam-se em zero
- Ambos os métodos devolvem -1 caso o texto não seja encontrado
- Um parâmetro extra pode ser adicionado indicando a posição onde deve iniciar a procura (por omissão inicia-se em zero)

Extração de texto

- **slice(start, end)**: extrai uma parte da string (iniciada em start e terminada em end) e devolve a parte numa nova string

```
let str = "Apple, Banana, Kiwi"
let res = str.slice(7, 13)
console.log(res)
-----------------------------------------------------------------
"Banana"
```

- **substr(start, length)**: igual ao anterior, mas o segundo parâmetro indica o número de caracteres a extrair

```
let str = "Apple, Banana, Kiwi"
let res = str.substr(7, 6)
console.log(res)
-----------------------------------------------------------------
"Banana"
```

- **substring(start)**: extrai dados de uma posição até ao fim da string

Substituição de texto

- **replace(locStr, newStr)**: procura uma parte da string (locStr) e substitui por outra (newStr)

```
let str = "Eu gosto da ESEIG!"
let newStr = str.replace("ESEIG", "ESMAD")
console.log(newStr)
-----------------------------------------------------------------
"Eu gosto da ESMAD!"
```

- Por omissão, a substituição é feita apenas na primeira ocorrência.
- Para propagar a substituição em todas as ocorrências use a expressão regular **/g**

```
let str = "Eu gosto da ESEIG e da ESEIG!"
let newStr = str.replace(/ESEIG/g, "ESMAD")
console.log(newStr)
-----------------------------------------------------------------
"Eu gosto da ESMAD e da ESMAD!"
```

Tipos de Dados - String

Iteração de texto

- **charAt(position)**: retorna o caracter no índice especificado numa string

```
let str = "ESMAD"
for (let i = 0; i < str.length; i++) {
  console.log(str.charAt(i))
}
-------------------------------------------------------------------
E
S
M
A
D
```

Tipos de Dados - String

- **Template strings** são literais de strings que permitem expressões incorporadas
- Usam acentos graves, **back-ticks** em vez de aspas.
- Exemplo de uma string template:

```
let str = `ESMAD`
```

- Podem usar marcadores de posição para substituição de strings usando a sintaxe **${}**

```
let name = "Miguel"
console.log(`Olá, ${name}!`)
-----------------------------------------------------
Olá, Miguel!
```

- Suporte para expressões

```
let a = 10; let b = 10
console.log(`A soma de ${a} com ${b} é ${a + b}`)
-----------------------------------------------------
A soma de 10 com 10 é 20
```

Tipos de Dados - String

- Os **literais template (ou template strings)** permitem o uso de funções

```
function fn() { return "Olá Mundo"; }
console.log(`Mensagem: ${fn()} !!`);
-------------------------------------------------
Mensagem: Olá Mundo !!
```

Tipos de Dados - String

- Referências online
  - **MDN**[3]
  - **W3Schools**[4]

[3] https://developer.mozilla.org/(...)/Reference/Global_Objects/String
[4] https://www.w3schools.com/jsref/jsref_obj_string.asp

Tipos de Dados - Array

Objectos similares a listas, usados para armazenar **múltiplos** valores numa variável.

- **Criar** um array

```
let cars = ["Saab", "Volvo", "BMW"]
let cars = new Array("Saab", "Volvo", "BMW")
```

- **Aceder** a um elemento de um array

```
console.log(cars[0]);
------------------------------------------------------
Saab
```

- **Modificar** um elemento de um array

```
cars[0] = "Opel"
console.log(cars[0]);
------------------------------------------------------
Opel
```

Tipos de Dados - Array

- Adicionar um elemento num array, método **push**.

```
let fruits = ["Banana", "Orange", "Apple"]
fruits.push("Lemon")
console.log(fruits);
-----------------------------------------------------------------
["Banana", "Orange", "Apple", "Lemon"]
```

- Alternativa, usando a propriedade **length**.

```
let fruits = ["Banana", "Orange", "Apple"]
fruits[fruits.length] = "Lemon"
console.log(fruits);
-----------------------------------------------------------------
["Banana", "Orange", "Apple", "Lemon"]
```

- Adicionando "buracos" no array (criação de vários elementos undefined)

```
let fruits = ["Banana", "Orange", "Apple"]
fruits[5] = "Lemon"
console.log(fruits);
-------------------------------------------------------------------
["Banana", "Orange", "Apple", undefined, undefined, "Lemon"]
```

Iteração

- Uso do **for**

```
let fruits = ["Banana", "Orange", "Apple"]
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
------------------------------------------------------------
Banana
Orange
Apple
```

- Uso do **for...in**

```
let fruits = ["Banana", "Orange", "Apple"]
for (let i in fruits) {
  console.log(fruits[i]);
}
------------------------------------------------------------
Banana
Orange
Apple
```

Iteração

- Uso do **forEach**

```
let fruits = ["Banana", "Orange", "Apple"]
fruits.forEach(function (val, index) {
  console.log(val)
})
------------------------------------------------------------
Banana
Orange
Apple
```

Principais métodos dos arrays

- Conversão para strings: **valueOf**, **toString** e **join**.
- Adição de elementos: **push**, **unshift**, **splice**.
- Remoção de elementos: **pop**, **shift**, **splice**.
- Ordenar elementos: **sort**
- Inverter elementos: **reverse**
- Junção de arrays: **join**
- Extração de arrays: **slice**

Tipos de Dados - Array

- Exemplo de como juntar 2 arrays

```
let myGirls = ["Cecilie", "Lone"]
let myBoys = ["Emil", "Tobias", "Linus"]
let myChildren = myGirls.concat(myBoys);
console.log(myChildren);
------------------------------------------------------------
["Cecilie", "Lone", "Emil", "Tobias", "Linus"]
```

- Exemplo de como extrair parte de um array

```
let fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]
let citrus = fruits.slice(1, 3)
console.log(citrus);
------------------------------------------------------------
["Orange", "Lemon"]
```

- **reduce**: aplica uma função simultaneamente contra dois valores do array (da esquerda para a direita) para reduzi-lo a um único valor

```
let total = [1, 2, 3].reduce(function (a, b) { return a + b })
console.log(total)
-----------------------------------------------------------------
6
```

- **every**: testa se todos os elementos dum array passam o teste implementado pela função fornecida

```
function isBigEnough(element, index, array) {
  return (element >= 10);
}
let passed = [12, 5, 8, 130, 44].every(isBigEnough)
console.log("TESTE: " + passed)
-----------------------------------------------------------------
TESTE: false
```

- O método **some** é similar, bastando no entanto um elemento passar o teste.

Tipos de Dados - Array

- **find**: Permite iterar no array e obter o 1º elemento que corresponde a uma condição
- O método **findIndex** é similar ao **find**, mas neste caso devolve a posição do elemento.

```
let numbers = [1, 2, 3]
let oddNumber = numbers.find((x) => x % 2 == 1)
console.log(oddNumber)
-----------------------------------------------------------------
1
```

- **map**: cria uma novo array com os resultados da aplicação de uma função em cada elemento desse array

```
let numbers = [1, 4, 9]
let roots = numbers.map(Math.sqrt)
console.log(roots)
-----------------------------------------------------------------
[1, 2, 3]
```

Tipos de Dados - Array

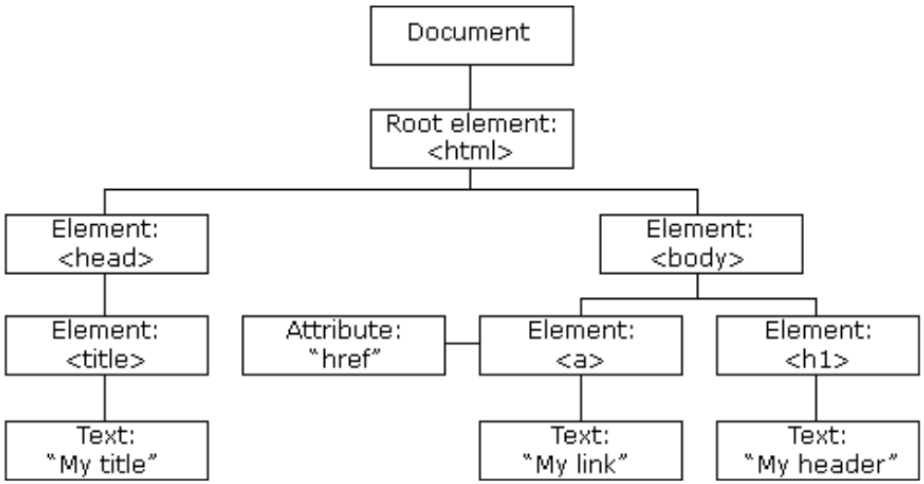
- Referências online
  - **MDN**[5]
  - **W3Schools**[6]

[5] https://developer.mozilla.org/(...)/Reference/Global_Objects/Array
[6] https://www.w3schools.com/jsref/jsref_obj_array.asp

DOM

DOM

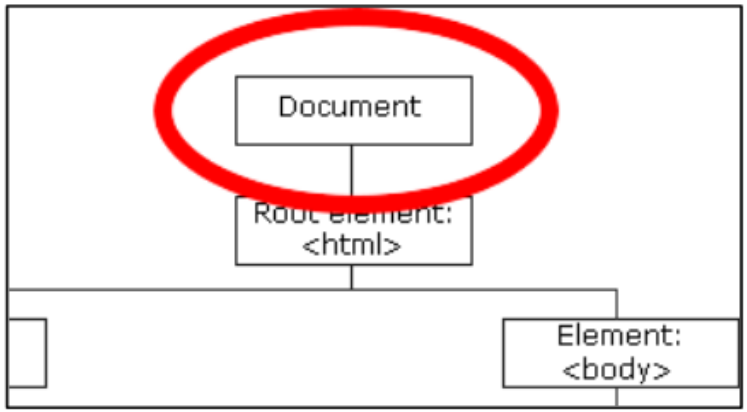
- Quando uma página é carregada, o browser cria um **Document Object Model (DOM)** da página.
- O modelo HTML DOM é um standard da W3C e é construído como uma árvore de objetos



- Com o modelo de objetos, o JS pode criar **HTML dinâmico**
  - Pode **alterar** todos os elementos/atributos HTML na página
  - Pode **adicionar/remover** elementos HTML e atributos existentes
  - Pode **criar/reagir** a eventos HTML existentes na página

DOM - Introdução

Objecto **document** representa a página Web e é usado para várias ações



- **Encontrar elementos**

```
document.getElementById(id)
document.getElementsByTagName(name)
document.getElementsByClassName(name)
```

DOM - Introdução

Objecto **document** representa a página Web e é usado para várias ações

- **Adicionar/remover elementos**

```
document.createElement(element)
document.removeChild(element)
document.appendChild(element)
document.replaceChild(element)
document.write(text)
```

- **Adicionar manipuladores de eventos**

```
document.getElementById(id).onclick = function () { /* code */ }
```

- **Encontrar objetos**

```
document.images
document.URL
document.forms
```

DOM - Procurar Elementos

- Muitas vezes, com JavaScript, pretendemos manipular elementos HTML.
- Para fazer isso, é necessário **encontrar os elementos** por:
  - **id**
  - **nome de tag**
  - **nome de classe**
  - **seletores CSS**
  - **coleções de objetos HTML**

DOM - Procurar Elementos

- Encontrar elementos por:
  - **id**
  - nome de tag
  - nome de classe
  - seletores CSS
  - coleções de objetos HTML

```
<p id="intro">Olá</p>
<p id="intro2">João</p>
<script>
  let myRef = document.getElementById("intro")
  console.log(myRef)
</script>
-----------------------------------------------------------
<p id="intro">Olá</p>
```

DOM - Procurar Elementos

- Encontrar elementos por:
  - id
  - **nome de tag**
  - nome de classe
  - seletores CSS
  - coleções de objetos HTML

```
<p id="intro">Olá</p>
<p id="intro2">João</p>
<script>
  let myRef = document.getElementsByTagName("p")
  console.log(myRef.length)
</script
-------------------------------------------------------------
2
```

DOM - Procurar Elementos

- Encontrar elementos por:
  - id
  - nome de tag
  - **nome de classe**
  - seletores CSS
  - coleções de objetos HTML

```
<p id="intro">Olá</p>
<p id="intro2">João</p>
<h1 class="intro">bom dia</h1>
<script>
  let myRef = document.getElementsByClassName("intro")
  console.log(myRef.length)
</script>
-----------------------------------------------------------
1
```

DOM - Procurar Elementos

- Encontrar elementos por:
  - id
  - nome de tag
  - nome de classe
  - **seletores CSS**
  - coleções de objetos HTML

```
<p class="intro">Olá</p>
<p class="intro2">João</p>
<h1 class="intro">bom dia</h1>
<script>
  let myRef = document.querySelectorAll("p.intro")
  console.log(myRef.length)
</script>
-----------------------------------------------------------
1
```

DOM - Procurar Elementos

- Encontrar elementos por:
  - id
  - nome de tag
  - nome de classe
  - seletores CSS
  - **coleções de objetos HTML**

```
<img src="flower.jpg">
<img src="sun.jpg">
<script>
  let myRef = document.images
  let text = ""

  for (let i = 0; i < myRef.length; i++) {
    text += myRef[i].src + "\n"
  }
  console.log(text)
</script>
-----------------------------------------------------------------
flower.jpg
sun.jpg
```

DOM - Alterar Conteúdo de Elementos

- Modificar conteúdo de elemento HTML: propriedade **innerHTML**.

```
document.getElementById("id").innerHTML = "novo HTML"
```

- Modificar valor de um atributo HTML

```
document.getElementById("id").<attributeName> = "novo valor"
```

DOM - Alterar Conteúdo de Elementos

- Valor de um atributo HTML

```
<!DOCTYPE html>
<html>
<body>
  <img id="myImage" src="flowers.jpg">
  <script>
    document.getElementById("myImage").src = "sun.jpg"
  </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
  <input id="myTxt" type="text" value="Hello World">
  <script>
    document.getElementById("myTxt").value = "Hi"
  </script>
</body>
</html>
```

DOM - Estilos CSS

- Para alterar o estilo de um elemento HTML, use esta sintaxe:

```
document.getElementById("id").style.<property> = /* novo estilo */
```

- Exemplo prático número 1:

```
<!DOCTYPE html>
<html>
<body>
  <p id="p2">Olá Mundo!</p>
  <script>
    document.getElementById("p2").style.color = "blue";
  </script>
</body>
</html>
```

DOM - Estilos CSS

- Para alterar o estilo de um elemento HTML, use esta sintaxe:

```
document.getElementById("id").style.<property> = /* novo estilo */
```

- Exemplo prático número 2:

```
<!DOCTYPE html>
<html>
<body>
  <p id="p2">Olá Mundo!</p>
  <input
    type="button"
    value="esconder texto"
    onclick="document.getElementById('p2').style.visibility='hidden'"
  >
</body>
</html>
```

DOM - Eventos

- Exemplos de eventos HTML:
  - Utilizador clica no rato
  - Página web foi carregada
  - Uma imagem foi carregada
  - Rato move-se sobre um elemento
  - Campo de input é alterado
  - Formulário HTML é submetido
  - Utilizador pressiona uma tecla

```
<!DOCTYPE html>
<html>
<body>
  <p onclick="changeText(this)">Clica-me</p>
  <script>
    function changeText(id) {
      id.innerHTML = " clicado"
    }
  </script>
</body>
</html>
```

Adicionar o código JavaScript a um atributo de evento HTML (ex.: onclick)

Adicionar o código JavaScript a um **atributo de evento** HTML

- Eventos mais usuais:
  - **onload**
  - **onclick**
  - **onchange**
  - **onmousedown**
  - **onmouseup**
  - **onfocus**

```
<!DOCTYPE html>
<html>
<body>
  <h1 onmouseover="style.color='red'" onmouseout="style.color='black'">
    Mouse over this text
  </h1>
</body>
</html>
```

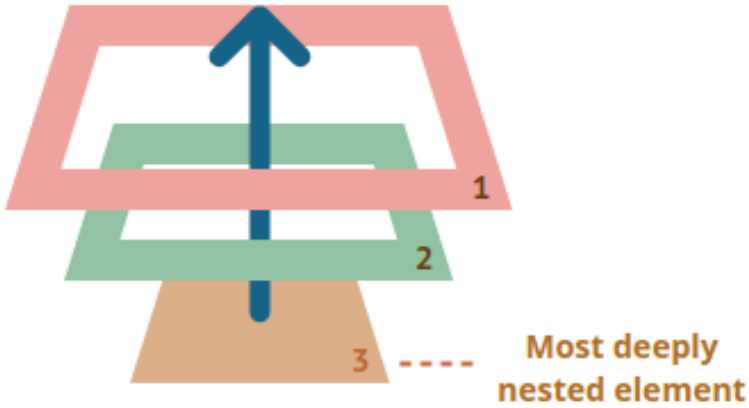
DOM - Eventos

- Método **addEventListener** anexa um manipulador de eventos a um elemento
- Podem ser adicionados mais que um manipulador de eventos do mesmo tipo a um elemento (ex.: dois eventos de clique)
- Event listeners podem ser aplicados a qualquer objeto DOM, não apenas elementos HTML (ex.: objeto window)

```
elemento.addEventListener(event, function, useCapture)
```

Event listener

- O JavaScript é separado da marcação HTML: melhor legibilidade.
- Removidos usando o método **removeEventListener**
- Permite manipular a propagação de eventos[7]:
  - **Bubbling**: usecapture a falso por omissão, o evento do elemento mais interno é tratado primeiro e depois o externo.
  - **Capturing**: usecapture a verdadeiro, o evento do elemento mais externo é tratado primeiro e depois o interno.



[7] https://javascript.info/bubbling-and-capturing

DOM - Eventos

- Adicionar um manipulador de evento a um elemento com uma função

```
document.getElementById("myBtn").addEventListener("click", myFunction)
function myFunction() {
  alert("Olá Mundo!")
}
```

- Adicionar muitos eventos ao mesmo elemento, sem sobrescrever eventos existentes.

```
document.getElementById("myBtn").addEventListener("click", myFunction)
document.getElementById("myBtn").addEventListener("click", myFunction2)
```

- Adicionar eventos de diferentes tipos ao mesmo elemento

```
document.getElementById("myBtn").addEventListener("mouseover", myFunction)
document.getElementById("myBtn").addEventListener("click", myFunction2)
document.getElementById("myBtn").addEventListener("mouseout", myFunction3)
```

DOM - Eventos

- Adicionar um manipulador de eventos ao objeto **window**

```
window.addEventListener("resize", function () {
  document.getElementById("demo").innerHTML = sometext;
});
```

- Passagem de parâmetros (definição de função anónima)

```
let p1 = 5, p2 = 7

document.getElementById("myBtn").addEventListener("click", function () {
  myFunction(p1, p2)
});

function myFunction(a, b) {
  let result = a * b
  document.getElementById("demo").innerHTML = result
}
```

- O método **removeEventListener** remove manipuladores de eventos que foram anexados ao elemento respetivo através do método **addEventListener**

```
document.getElementById("myBtn").removeEventListener("click", myFunction)
```

onload: evento de carregamento da página HTML

- Usando manipuladores no próprio HTML

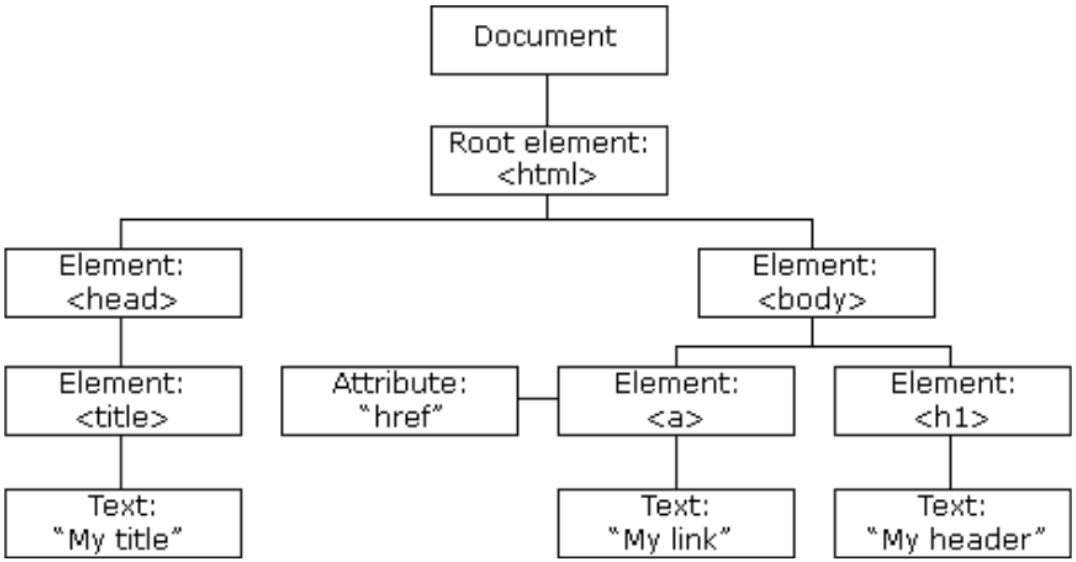
```
<!DOCTYPE html>
<html>
<head>
  <script>
    function myFunction() {
      alert("Page is loaded");
    }
  </script>
</head>
<body onload="myFunction()">
  <h1>Hello World!</h1>
</body>
</html>
```

- Usando manipuladores apenas no JavaScript

```
window.onload = function () {
  console.log("load event detected!");
}
```

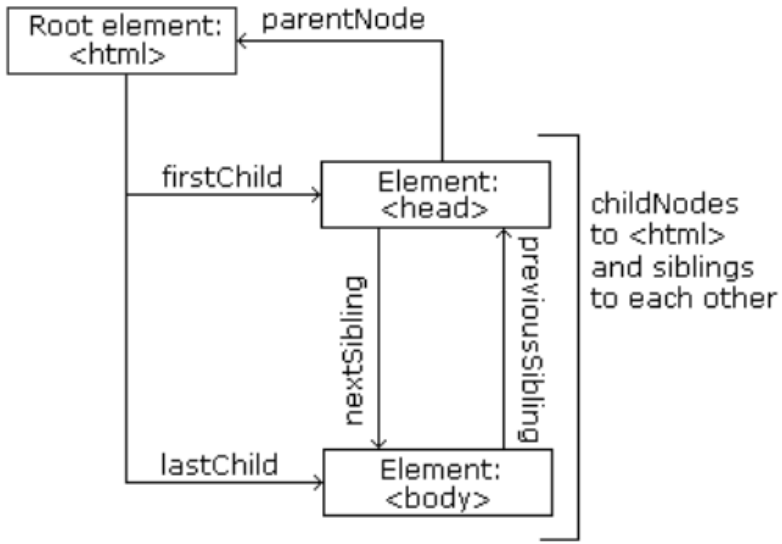
DOM - Navegação

- Com o HTML DOM, é possível navegar na árvore usando relacionamentos de nodos.
- O padrão HTML DOM do W3C define que tudo num documento HTML é um nodo:
  - O documento inteiro é um **nodo de documento**
  - Todo elemento HTML é um **nodo de elemento**
  - O texto dentro de elementos HTML são **nodos de texto**
  - Todos os comentários são **nodos de comentário**

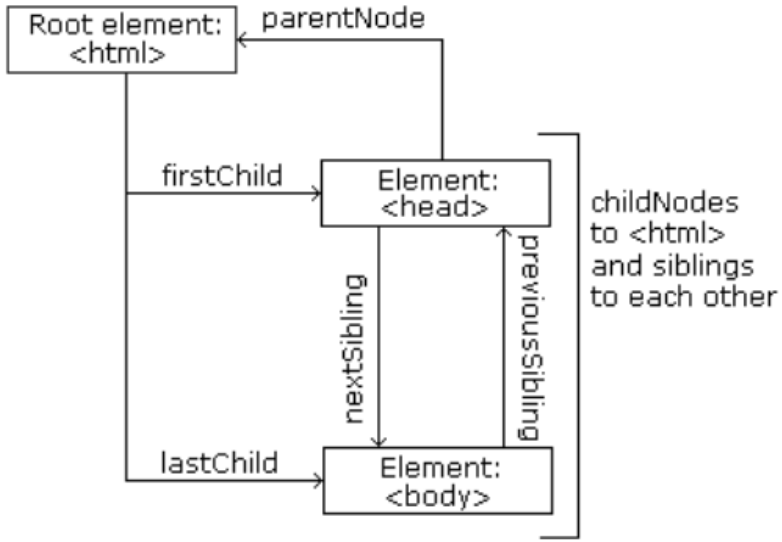


DOM - Navegação

- Os nodos na árvore têm um relacionamento hierárquico entre si
- Os termos pai, filho e irmão são usados para descrever os relacionamentos.
  - Numa árvore de nodos, o nodo superior é chamado de raiz.
  - Cada nodo tem exatamente um pai, exceto a raiz.
  - Um nodo pode ter um número de filhos
  - Irmãos são nodos com o mesmo pai



DOM - Navegação



- Obter um texto de um elemento html

```
<title id="demo">DOM Tutorial</title>
```

```
let txt = document.getElementById("demo").innerHTML
let txt = document.getElementById("demo").firstChild.nodeValue
let txt = document.getElementById("demo").childNodes[0].nodeValue
```

DOM - Navegação

Propriedades importantes de um nodo

- **nodeName**
- **nodeValue**
- **nodeType**

nodeName

- Especifica o nome de um nó
- Somente de leitura
- Obtenção de diferentes valores:
  - **nodo de elemento** é o mesmo que o nome da tag
  - **nodo de texto** é sempre **#text**
  - **nodo de documento** é sempre **#document**

```
<title id="demo">DOM Tutorial</title>
```

```
console.log(document.getElementById("demo").nodeName)
-----------------------------------------------------------------------
TITLE
```

nodeValue

- Especifica o valor de um nodo
- Somente de leitura
- Obtenção de diferentes valores:
  - nodo de elemento é **undefined**
  - nodo de texto é o próprio texto
  - nodo de atributo é o valor do atributo

```
<title id="demo">DOM Tutorial</title>
```

```
console.log(document.getElementById("demo").nodeValue)
-------------------------------------------------------------------
undefined
```

nodeType

- Especifica o tipo de um nodo
- Somente de leitura

Constant	Value	Description
`Node.ELEMENT_NODE`	`1`	An `Element` node like `<p>` or `<div>`.
`Node.TEXT_NODE`	`3`	The actual `Text` inside an `Element` or `Attr`.
`Node.CDATA_SECTION_NODE`	`4`	A `CDATASection`, such as `<!CDATA[[ … ]]>`.
`Node.PROCESSING_INSTRUCTION_NODE`	`7`	A `ProcessingInstruction` of an XML document, such as `<?xml-stylesheet … ?>`.
`Node.COMMENT_NODE`	`8`	A `Comment` node, such as `<!-- … -->`.
`Node.DOCUMENT_NODE`	`9`	A `Document` node.
`Node.DOCUMENT_TYPE_NODE`	`10`	A `DocumentType` node, such as `<!DOCTYPE html>`.
`Node.DOCUMENT_FRAGMENT_NODE`	`11`	A `DocumentFragment` node.

DOM - Gestão de Nodos

A gestão de nodos passa pela:

- **Criação de novos nodos**
- **Substituição de nodos existentes**
- **Remoção de nodos existentes**

Criação de novos nodos

- Para adicionar um novo elemento ao HTML DOM:
  - Crie o elemento (nodo de elemento) com os métodos **createElement/createTextNode**
  - Anexe-o a um elemento existente com o método **appendChild**

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>
```

```
let para = document.createElement("p")
let node = document.createTextNode("This is new.")
para.appendChild(node)

let element = document.getElementById("div1")
element.appendChild(para)
```

Substituição de nodos existentes

- Para substituir um elemento no HTML DOM, use o método **replaceChild**.

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>
```

```
let para = document.createElement("p")
let node = document.createTextNode("This is new.")
para.appendChild(node)

let parent = document.getElementById("div1")
let child = document.getElementById("p1")
parent.replaceChild(para, child)
```

Remoção de nodos existentes

- Para remover um elemento HTML, deve:
  - Conhecer o pai do elemento
  - Usar o método **removeChild**

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>
```

```
let parent = document.getElementById("div1")
let child = document.getElementById("p1")
parent.removeChild(child)
```

DOM - Validação de Formulários

- A Validação de formulários HTML **deve ser feita unicamente pelos novos atributos dos elementos de formulário do HTML!** (ver slides HTML)
- Por exemplo, tornar um elemento do formulário obrigatório.

```
Name: <input type="text" required>
```

- Contudo pode também **criar um listener** para validações extra

```
<form id="frmForm">
  nome: <input type="text" required>
  <input type="submit" value="submeter">
</form>
```

```
let myForm = document.getElementById("frmForm")
myForm.addEventListener("submit", function(event){
  // fazer as validações extra necessárias
  // ...
  // não deixar fazer a submissão
  event.preventDefault()
});
```

- Outro exemplo com **onsubmit**

```
<form name="myForm"
      action="/action_page.php"
      onsubmit="return validateForm()"
      method="post"
>
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
```

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

Informação Adicional

Referência

- MDN JavaScript[8]
- MDN DOM[9]

Tutorial

- The Modern JavaScript Tutorial[10]

Style Guide

- Airbnb JavaScript Style Guide[11]

[8] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
[9] https://developer.mozilla.org/(..)/API/Document_Object_Model
[10] https://javascript.info/
[11] https://github.com/airbnb/javascript