

Programação I

Javascript



JavaScript



Professor
Marcus Aurelius

Organização da linguagem

- Sentença de código (linha de código), opcionalmente encerrada com ";"
- Blocos de código, delimitados por "{" e "}";

```
function arredondarNota(notaAluno) {  
  let indice  
  let notaArredondada = notaAluno  
  //const parametros = [40,50,60,70,80,90,100]  
  const parametros = [100,95,90,85,80,75,70,65,60,55,50,45,40]  
  for (indice = 0; indice < parametros.length; indice++) {  
    if ((parametros[indice] > notaAluno) && ((parametros[indice] - notaAluno) < 3)) {  
      notaArredondada = parametros[indice]  
      break  
    }  
  }  
  //console.log(notaAluno)  
  return notaArredondada  
}
```

Comentários

- `//` - única linha;
- `/* */` - múltiplas linhas;

```
/*
01) Crie uma função que dado dois valores (passados como parâmetros) mostre no console a
soma, subtração, multiplicação e divisão desses valores
*/

let valor1 = 6;
let valor2 = 3;

function operacoes(a, b) {
    console.log(`Soma.....: ${a} + ${b} = ${a + b}`);
    console.log(`Subtracao....: ${a} - ${b} = ${a - b}`);
    console.log(`Multiplicação: ${a} * ${b} = ${a * b}`);
    console.log(`Divisão.....: ${a} / ${b} = ${a / b}`);
}

// escrevendo os valores gerados
console.log(`Valores: ${valor1} e ${valor2}`);
operacoes(valor1, valor2);
```

Variáveis e Constantes

- Tipagem fraca
 - JavaScript é uma linguagem fracamente tipada;
 - Possui uma tipagem dinâmica, atribuindo valores de tipos diferentes para a mesma variável no decorrer do programa.
- Var
 - Declara variáveis visíveis fora do bloco no qual foi declarada (exceto em funções devido ao escopo da função, que é local);
 - Declara uma variável global, usada em todos os blocos do programa;
 - No navegador essa variável será acessada no objeto 'window';
 - Deve-se evitar o uso de variável global;
 - Podem ter seus valores sobrescritos;
 - Variáveis declaradas sem 'var', 'let', ou 'const', são globais;
- Let
 - Tem escopo de bloco para a variável;
 - A variável só é visível dentro do bloco onde foi declarada;
 - Não permite que uma variável seja re-declarada;

```
...  
let horasTrabalhadas = 100  
let salarioHora = 10  
...  
var valor  
var contador  
var caracteres = ''  
...  
const meuArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
...
```



JavaScript



Tipos

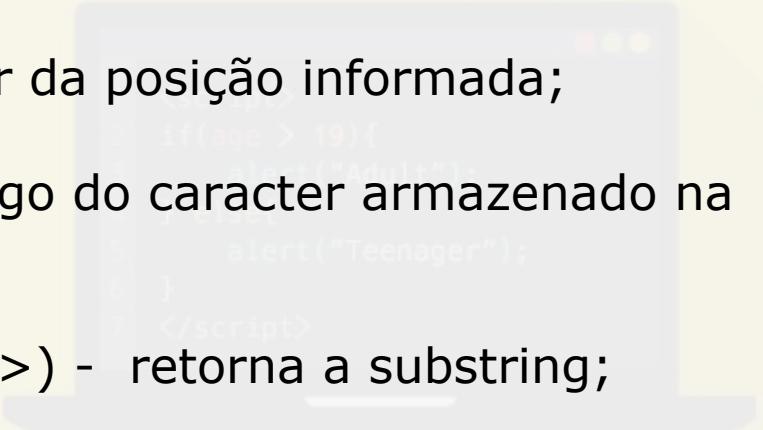
- **Number**

- A função Number tem vários métodos interessantes;
- Principais métodos:
 - `.isFinite()` - verifica se é um número finito.
 - `.isInteger()` - verifica se é um número inteiro.
 - `.isNaN()` - verifica se o valor é um `Number.NaN` (Not a Number).
 - `.toFixed(x)` - formata o número para ficar com x caracteres.
 - `.valueOf()` - retorna o valor primitivo do número.
 - `.toFixed(<casas decimais>)` - define o número de casas decimais do número.
 - `.toString(<valor>)` - converte o valor em string.

Tipos

- **String**

- Delimitado por:
 - Áspas;
 - Apóstofros;
 - Crase;
- `.charAt(<posição>)` - retorna o caracter da posição informada;
- `chaCodeAt(<posição>)` - retorna o código do caracter armazenado na posição informada;
- `.substring(<índice inicial>,<índice final>)` - retorna a substring;
- `.concat(<valor>)` - concatena um valor á string, retornando a nova string;
- `.replace(<texto>,<novo texto>);`
- `"+"` - Concatena strings;



```
if (age > 19) {  
    // ...  
    alert("Teenager");  
}
```

Tipos

- **String**

- `.split(<separador>[, <limite de divisões>])` - quebra uma string transformando-a em array, e retorna esse array;
- `.template string`
 - Estrutura delimitada por crase;
 - Valor = ``<texto> ${<variáveis/expressões matemáticas>}``

- **boolean**

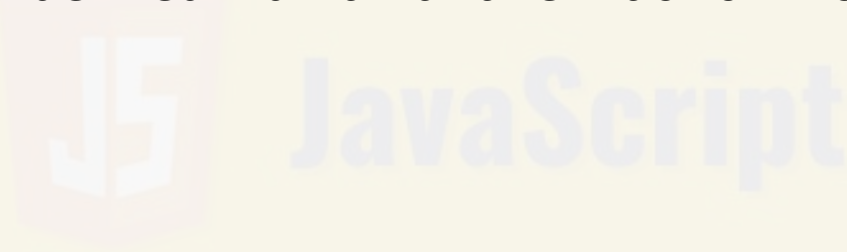
- Retorna:
 - `true`;
 - `false`;



```
1 if (age > 19) {  
2     alert("Adult");  
3 } else {  
4     alert("Teenager");  
5 }  
6  
7 </script>
```


null x undefined

- Para os tipos primitivos os valores são atribuídos por valor;
- Nos objetos e funções os valores são passados por referência;
- Null - não aponta para nenhum objeto na memória;
- Undefined - uma variável não foi inicializada;



Operadores de atribuição

=	a = b + c	
+=	a += 5	a = a + 5
-=	a -= 3	a = a - 3
*=	a *= 9	a = a * 9
/=	a /= 4	a = a / 4
%=	a %= 2	a = a % 2 (para valor par retorna '0', para valor ímpar retorna '1')

Operadores aritméticos

+	$a = b + c$	Soma
-	$a = b - c$	Subtração
*	$a = b * 9$	Multiplicação
/	$a = b / 4$	Divisão
%	$a = b \% 2$	Resto da divisão

Operadores relacionais

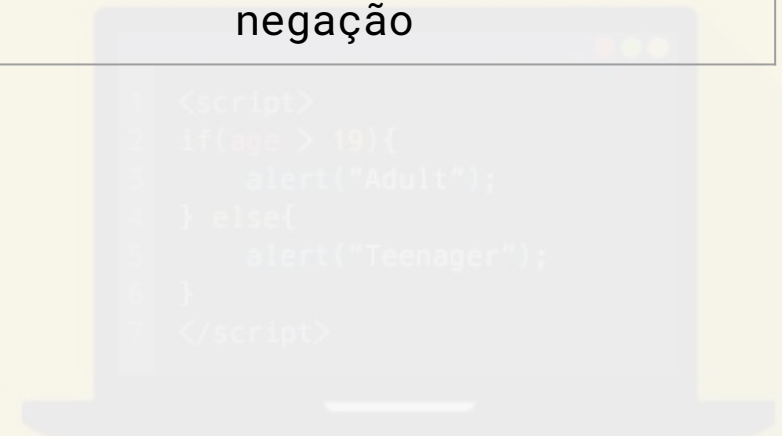
<	menor que
>	maior que
<=	menor ou igual
>=	maior ou igual
==	igualdade entre dois valores
===	compara valor e tipo (estritamente igual)
!==	compara valor e tipo (estritamente diferente)
!=	diferente

Operadores lógicos

&&	e
	ou
xor	ou exclusivo
!	negação



JavaScript



Operadores unários

++	++a	incrementa e usa a variável
	a++	usa a variável e incrementa
--	--a	decrementa e usa a variável
	a--	usa a variável e decrementa



JavaScript

```
1 <script>
2 if(age > 19){
3     alert("Adult");
4 } else{
5     alert("Teenager");
6 }
7 </script>
```

Operadores ternários

- Tem três operandos
 - Expressão relacional;
 - Instrução executada caso a condição seja verdadeira;
 - Instrução executada caso a condição seja falsa;

[<variável> =]<condição> ? instrução verdade : instrução falso

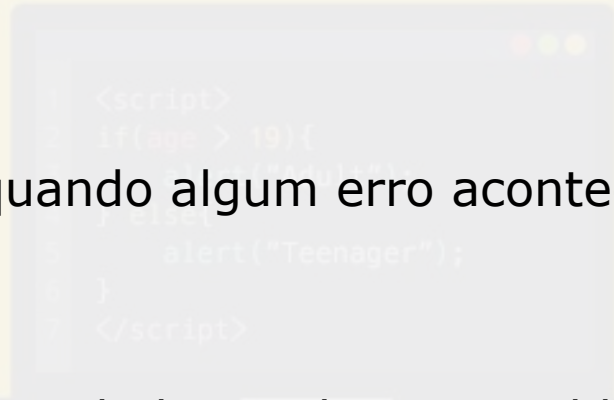
Exemplo:

```
const resultado = nota >= 7 ? 'Aprovado' : 'Reprovado'
```

```
<script>  
if (age > 19) {  
  alert("Adult");  
} else {  
  alert("Teenager");  
}  
</script>
```

Operadores try...catch...finally

- Bloco que trata erros no programa;
 - try
 - Bloco com os comandos que serão executados e que podem gerar erros;
 - Se houver algum erro o fluxo de execução é transferido para a cláusula catch;
 - catch
 - Bloco de comandos que é executado quando algum erro acontece no bloco try;
 - finally
 - Bloco de comandos que pode ser executado havendo erro no bloco try ou não;
 - throw
 - Comando que permite lançar um erro no código de forma intencional;



Operadores try...catch...finally

```
...  
try {  
    <bloco de comandos que serão executados>  
}  
catch {  
    <bloco de comandos que serão executados caso algum erro aconteça no bloco try>  
}  
finally {  
    <Bloco de comando opcional que será executado independente de erro no bloco>  
}  
...
```