# 

#### Javascript

Lógica de Programação

#### Operadores Javascript

Os operadores existem para nos ajudar na execução de várias tarefas: podemos comparar variáveis, efetuar cálculos, atribuir valores, criar testes lógicos, etc.

Todas as linguagens de programação têm os seus operadores e no Javascript não poderia ser diferente, vamos conhecer os principais operadores existentes na linguagem.

#### Operadores aritméticos

Operador	Sinal	Definição
Soma	+	Executa a soma entre dois ou mais números
Subtração	-	Executa a subtração entre dois ou mais números
Multiplicação		Executa a multiplicação entre dois ou mais números
Divisão	/	Executa a divisão entre dois ou mais números
Módulo	%	Consiste na divisão entre dois números, retornando o resto obtido nessa divisão
Potência	**	Executa o cálculo da potência entre dois números, elevando o primeiro número à potência do segundo número

•

#### Operadores relacionais

•	Operador	Sinal	Definição
	Maior que	>	Retornará verdadeiro quando o primeiro valor for maior que o segundo valor
	Menor que	<	Retornará verdadeiro quando o primeiro valor for menor que o segundo valor
	Maior ou igual a	>=	Retornará verdadeiro quando o primeiro valor for maior ou igual ao segundo valor
	Menor ou igual a	<=	Retornará verdadeiro quando o primeiro valor for menor ou igual ao segundo valor
	Igual a	==	Retornará verdadeiro quando os dois valores forem iguais
	Diferente de	!=	Retornará verdadeiro quando os dois valores forem diferentes
	Estritamente igual a	===	Retornará verdadeiro quando os dois valores forem iguais e do mesmo tipo
	Estritamente diferente a	!==	Retornará verdadeiro quando os dois valores não forem iguais ou não forem do mesmo tipo

#### Operadores lógicos

Operador	Sinal	Definição
E	&&	Para retornar verdadeiro, todas as expressões devem ser verdadeiras
ου	П	Para retorna verdadeiro, basta apenas uma expressão ser verdadeira
NÃO	1	Inverte o resultado da expressão: se o resultado for verdadeiro retornará falso, se o resultado for falso retornará verdadeiro

•

#### Expressões Lógicas Compostas



Matemática:

16 ≤ **idade** < 18

idade > 70

Lógica:

idade >= 16 && idade < 18

idade > 70

(idade >= 16 && idade < 18) | | (idade > 70)

#### Operadores incremento/decremento

Operador	Sinal	Definição
Incremento	++	Adiciona 1 ao valor da variável
Decremento		Subtrai 1 do valor da variável

## Operações pré/pós-fixadas

Operador	Sinal	Definição
Pré-fixada	++variável variável	O valor da variável é incrementado ou decrementado antes de ser usado em uma expressão
Pós-fixada	variável++ variável	O valor da variável é incrementado ou decrementado após ser usado em uma expressão

#### Operadores de atribuição

• Operador ·	Sinal	Definição
Atribuição	=	Atribui um valor a uma variável
Atribuição de adição	+=	Atribui um valor efetuando uma soma
Atribuição de subtração	- <del>-</del>	Atribui um valor efetuando uma subtração
Atribuição de multiplicação	*=	Atribui um valor efetuando uma multiplicação
Atribuição de divisão	/=	Atribui um valor efetuando uma divisão
Atribuição de resto	%=	Atribui um valor efetuando um módulo
Atribuição de exponencial	**=	Atribui um valor efetuando uma potenciação

•

.

.

•

# Objeto Math

O Objeto Math permite a realização de qualquer operação aritmética. Possui uma série de métodos e propriedades que irão auxiliar em tarefas que possam envolver cálculos. Vamos dar uma olhada nos seus métodos mais comuns.

Método	O que faz
Math.ceil()	Retorna o próximo número inteiro
Math.floor()	Retorna o número inteiro anterior
	Retorna o número inteiro mais próximo, para isso ele verifica:
Math.round()	Parte decimal do número maior ou igual a 49, retornará o número inteiro anterior.
	Parte decimal do número maior ou igual a 50, retornará o próximo número inteiro
Math.max()	Retorna o maior valor encontrado no intervalo

Método	O que faz
Math.min()	Retorna o menor valor encontrado no intervalo
Math.pow()	Retorna a potência de um número.
Math.random()	Retorna um número aleatório entre 0 e 1
Math.sqrt()	Retorna a raiz quadrada de um número
Math.cbrt()	Retorna a raiz cúbica de um número

•

.

#### Sorteando números

Com o uso do método Math.random(), temos o sorteio de um valor aleatório entre 0 e 1, mas e se precisarmos de um número entre 0 e 99? Para sortearmos um número inteiro qualquer, podemos seguir os seguintes passos:

 Multiplicar o número que será obtido por 100. Qualquer número decimal multiplicado por 100, fará com que o ponto flutuante desse número, avance duas casas decimais para a direita.

Utilizar o método Math.floor() para retornar o número inteiro

anterior..

#### Exemplo de sorteio

```
//criando as variáveis e sorteando os números
const numero1 = Math.floor(Math.random() * 100);
const numero2 = Math.floor(Math.random() * 100);
const numero3 = Math.floor(Math.random() * 100);
console.log(`Número sorteado: ${numero1}`);
console.log(`Número sorteado: ${numero2}`);
console.log(`Número sorteado: ${numero3}`);
```

# Objeto String

O objeto String representa uma cadeia de caracteres que poderá ser manipulada através de uma série de métodos bem úteis.

Podemos acessar individualmente, os caracteres que formam a string usando os índices, da mesma forma que é feito nos arrays. Nesse caso o primeiro caracter da string está no índice O, o segundo no índice 1 e assim sucessivamente até o final da string. Vamos conhecer os métodos mais importantes:

Método	O que faz
Propriedade length	Retorna o número de caracteres existente na string
toUpperCase()	Retorna toda a string em letras maiúsculas
toLowerCase()	Retorna toda a string em letras minúsculas
charAt()	Retorna o caractere que está posicionado no índice declarado
indexOf()	Retorna o índice da primeira ocorrência de uma substring declarada. Caso não encontre nada retornará-1

Método	O que faz
lastIndexOf()	Retorna o índice da última ocorrência de uma substring declarada. Caso não encontre nada retornará-1
concat()	Concatena duas ou mais strings
replace()	Substitui a primeira substring encontrada por outra
replaceAll()	Substitui todas as substrings encontradas por outra
substring()	Retorna uma parte da substring. Devemos definir o índice inicial e final

Método	O que faz
slice()	Extrai uma parte da string retornando uma nova string
split()	Converte uma string em um array
trim()	Remove espaços em branco existentes no início e no fim da string. Bem útil para a manipulação de dados em formulários
includes()	Verifica se uma string existe em outra string, retornando apenas true ou false
startsWith()	Verifica se a string começa com o prefixo especificado, retornando apenas true ou false
endsWith()	Verifica se a string termina com o sufixo especificado, retornando apenas true ou false

## Objeto Date

O Objeto Date possui métodos que permitem a manipulação de datas e horas. Para isso precisamos estanciar o objeto utilizando o método construtor, ele retornará a data e hora atual.

```
//utilizando o método construtor

const dataAtual = new Date();

//exibindo o objeto

console.log(`Data atual: ${dataAtual}`);
```

Agora que possuímos um **objeto** com a data, podemos manipular as informações da forma que acharmos necessária. Só precisamos tomar cuidado com alguns detalhes:

- Os objetos instanciados serão baseados no valor de tempo composto pelo número de milissegundos transcorridos desde a data de 1º de Janeiro de 1970, até o momento de execução do script.
- Os dias da semana são representados por números no intervalo entre 0 (domingo) e 6 (sábado).
- Os meses do ano são representados por números no intervalo entre 0 (janeiro)
   e 11 (dezembro).
- Dias da semana, dias do mês, números do mês, hora, minutos e segundos
   menores que 10, serão exibidos apenas o seus números correspondentes, sem a presença do número 0 antecedendo-os.

Método	O que faz
getTime()	Retorna o número de milissegundos transcorridos entre 1 de janeiro de 1970 até a data atual
getDay()	Retorna o número do dia da semana correspondente do objeto
getDate()	Retorna o número do dia do mês correspondente do objeto
getMonth()	Retorna o número do mês correspondente do objeto

.

•

Método	O que faz
getFullYear()	Retorna o ano correspondente do objeto
getHours()	Retorna a hora correspondente do objeto
getMinutes()	Retorna os minutos correspondentes do objeto
getSeconds()	Retorna os segundos correspondentes do objeto
getMilliseconds()	Retorna os milissegundos correspondentes do objeto

#### Estruturas de decisão

Estruturas de decisão são essenciais em qualquer linguagem de programação, não poderia ser diferente em JavaScript. Essas estruturas analisam o que chamamos de testes lógicos (também chamadas de condições), e conforme o resultado obtido, ações previamente definidas são executadas pelo seu script.

# Estrutura if/else

A estrutura if / else pode usada para testar uma condição através de um teste lógico, e conforme a resposta obtida o script executará alguma funcionalidade. O teste sempre retornará duas respostas possíveis: true, e false.

Quando o teste lógico retornar true, os comandos que estiverem dentro do bloco if serão executadas, quando ele retornar false, o script executará os comandos do bloco else.

```
if (teste lógico) {

// Bloco true

}

else {

// Bloco false
}
```

# Estrutura if/else if/ else

As vezes pode acontecer de você precisar testar mais do que dois valores, e aí apenas uma estrutura if / else não dará conta.

Podemos usar a estrutura if / else if / else que permitirá a criação de quantos testes lógicos desejar. Vale ressaltar que assim que um teste retornar true, os testes abaixo dele não será verificados.

Caso todos os testes retornem false, o bloco de comandos a ser executado será o do último else.

```
if (teste lógico) {
      // Outro bloco
```



# If ternário

Podemos passar o resultado de um teste lógico diretamente a uma variável, sem precisar da instrução if.

Para isso atribuímos à variável a condição a ser verificada e usamos o operador ternário ponto de interrogação (?).

Passaremos dois valores separados pelo sinal de dois pontos (:), o primeiro para o retorno true e o segundo para o retorno false.

```
//instanciando o objeto Date
const data = new Date();
//pegando o mês contido no objeto
const mes = data.getMonth() + 1;
//criando o if ternário
const semestre = (mes <= 6 ? 'primeiro semestre' : 'segundo semestre');
console.log(`Estamos no mês ${mes}`);
console.log(`Estamos no ${semestre} do ano de
${data.getFullYear()}`);
```

# Estrutura switch/case/default

Um switch compara o conteúdo de uma variável com valores definidos dentro da instrução case. Quando temos um retorno true, será executado o bloco de comandos correspondente. Caso o retorno seja false, será testado o próximo case.

Podemos também definir um último bloco de comandos através da instrução **default**, ele será usado caso todos os cases retornem false.

Utilizamos também a instrução **break**, para encerrarmos a instrução switch.

```
//bloco de comandos
break; //saída da instrução switch
case valor: // segundo valor a ser verificado
//bloco de comandos
break; //saída da instrução switch
default: //caso nenhuma opção seja executada
```

# Estruturas de repetição

As estruturas de repetição devem ser usadas quando precisamos repetir alguma ação dentro de nossos scripts, conforme condições a serem definidas. Essas estruturas também são conhecidas por loops, sendo que as mais usuais são: for, while e do / while.

Dentro dessas estruturas, podemos utilizar uma variável que será responsável por controlar o número de repetições executadas, é com essa variável que o seu script saberá a quantidade de repetições realizadas.

### Estrutura for

É uma das estruturas mais comuns em programação, ela deve ser utilizada quando sabemos o número exato de repetições que uma determinada tarefa ou bloco de comandos deve executar.

```
for (variável = início; variável < fim; incremento++){
    //Bloco de comandos que será repetido
}
```

- Variável = início: declarar a variável que controlará o número de repetições a ser executado pelo loop, e atribuir seu valor inicial.
- Variável < fim: definir o valor final da variável, aquele que deve finalizar a estrutura de repetição.
- Incremento++: definir a quantidade de vezes que o loop será executado, para isso temos uma expressão que mudará cada vez que ocorrer a repetição.

### Estrutura while

Na maioria das vezes usamos essa estrutura quando não sabemos o número de repetições que serão executadas, mas ela também pode ser usada quando temos o número de repetições definido.

Devido a sua sintaxe, while parece muito com a estrutura if, mas com uma grande diferença: o if não consegue voltar e refazer o teste lógico, já o while consegue voltar e testar novamente a condição.

```
//Bloco de comandos a ser executado
```

## Estrutura do/while

Essa estrutura de repetição, obrigatoriamente, executará o loop pelo menos uma vez, independente da condição ser verdadeira ou falsa. Isso acontece devido a sua sintaxe, a condição só é verificada após a primeira execução do bloco de comandos, sendo verdadeira, o código será executado novamente. Isso continuará até que a condição seja falsa, e então o loop será interrompido.

Podemos também utilizar algum incremento para definir o número de vezes que a estrutura repetirá.

```
do {

//Bloco de comandos a ser executado pelo menos uma vez
}

while(condição);
```

#### BIBLIOGRAFIA BÁSICA

- DUCKETT, Jon. Javascript & Jquery: desenvolvimento de interfaces interativas. Rio de Janeiro: Alta Books, 2015.
- LEPSEN, Edécio Fernando. Lógica de programação e algoritmos com JavaScript. São Paulo:
   Novatec, 2018.TERUEL.
- JAVASCRIPT.INFO. The Modern JavaScript Tutorial. 2022. Disponível em: <a href="https://javascript.info/">https://javascript.info/</a>.
- MDN WEB DOCS. Javascript. Disponível em: <a href="https://developer.mozilla.org/pt-BR/docs/Web/JavaScript">https://developer.mozilla.org/pt-BR/docs/Web/JavaScript</a>.
- LEARN.JS. Javascript. Disponível em: <a href="https://www.learn-js.org/">https://www.learn-js.org/</a>.
- SOLOLEARN.JS. Introduction to Javascript. Disponível em: <a href="https://www.sololearn.com/learn/courses/javascript-introduction">https://www.sololearn.com/learn/courses/javascript-introduction</a>.
- BRAZILJS. Eloquente Javascript. Disponível em: <a href="https://github.com/braziljs/eloquente-javascript">https://github.com/braziljs/eloquente-javascript</a>.

#### **OBRIGADO**



Copyright © 2023 | Professor Alexandre Russ

Todos os direitos reservados. A reprodução ou divulgação total ou parcial deste documento é expressamente proibida sem o consentimento formal, por escrito, do professor/autor.

#