**Javascript I – Lógica de Programação**

***O que é uma linguagem de programação?***

Primeiramente, é necessário entender o que é uma Linguagem de Programação. A Linguagem de Programação é um conjunto de códigos que contém instruções para o computador.

Exempos: C, C++, C#, Java, PHP, Python, Ruby, Javascript, Go, Elixir, Dart e muitas outras.

Elas podem ser classificadas em 2 níveis: **Alto nível** (linguagem mais aproximada a linguagem humana), **Baixo nível** (linguagem mais aproximada a linguagem do hardware).

Podem ser **compiladas** ou **interpretadas**:

**Compilada** = O computador, através do compilador, traduz o código-fonte em uma linguagem de baixo nível(um binário( 0 e 1), assembly) e consegue executar o programa diretamente acessando esse arquivo.

Os compiladores, geram um arquivo executável. Através do compilador é possível verificar erros e otimizar o código.

**Interpretadas** = A linguagem tem um motor que interpreta o código imediatamente e transforma em algo que o computador entende. Ex: Javascript possui um motor(v8) que interpreta a linguagem. Logo, ele não precisa ser compilado antes para ser executado.

Os interpretadores traduzem e executam o código linha por linha, sem gerar uma arquivo executável.

***O que é JavaScript?***

O JavaScript foi criado em 1995 por Brendan Eich para ser usada pelo Netscape. Seu nome oficial é ECMAscript *(ECMA- É uma empresa europeia de padronização (estilo a ISO dos EUA). Logo, ECMAscript, nada mais é do que a versão padronizada do Javascript.).*

Ela é uma linguagem de alto nível e precisa de um interpretador.

O Javascript é uma linguagem multiparadigma, então é possível programar de diversas formas para obter um resultado no Javascript.

Paradigmas de programação são modelos ou abordagens para estruturar e organizar o código de um programa. Eles fornecem um conjunto de conceitos e técnicas que guiam o processo de desenvlvimento de software.

[***Imperativo***](https://www.google.com/search?sca_esv=7b647de37a4058b4&sxsrf=AE3TifNQENaORotp7Qd9CCnuKv_kjUIvmQ%3A1754766694312&q=Imperativo&sa=X&ved=2ahUKEwj3rOmrt_6OAxWuBLkGHaKRG04QxccNegUI5gEQAQ&mstk=AUtExfB5_lmFt3zPvVSITDkcwk0mIJUErFFySXxbkPxiK2vo_hddH0dnltYYe2mxHL9gC4myDAyAHLXuYqNvjIlZzJjBMtlWWKaxgC7ORyLQ2GPThFYVi5FihyFiGNDLsE7kAuG7BhD6dX0dyIp37KQeP2nQ3nLfUyT45aB__42tLFdagqEOrt0K3xCdC_ttjBQxAcBSLDL-A0JoHG_KcsNOAsmBVVKduO4dH88s9cNiu298VLv2SkmEMiVzfQwIhKJ0caA8ob5jknO3QuyIDfRAxpXz&csui=3)***:***

Foca em instruções detalhadas sobre como o computador deve executar as tarefas, passo a passo.

[***Declarativo***](https://www.google.com/search?sca_esv=7b647de37a4058b4&sxsrf=AE3TifNQENaORotp7Qd9CCnuKv_kjUIvmQ%3A1754766694312&q=Declarativo&sa=X&ved=2ahUKEwj3rOmrt_6OAxWuBLkGHaKRG04QxccNegUIjAIQAQ&mstk=AUtExfB5_lmFt3zPvVSITDkcwk0mIJUErFFySXxbkPxiK2vo_hddH0dnltYYe2mxHL9gC4myDAyAHLXuYqNvjIlZzJjBMtlWWKaxgC7ORyLQ2GPThFYVi5FihyFiGNDLsE7kAuG7BhD6dX0dyIp37KQeP2nQ3nLfUyT45aB__42tLFdagqEOrt0K3xCdC_ttjBQxAcBSLDL-A0JoHG_KcsNOAsmBVVKduO4dH88s9cNiu298VLv2SkmEMiVzfQwIhKJ0caA8ob5jknO3QuyIDfRAxpXz&csui=3)***:***

Descreve o que o programa deve fazer, sem especificar como fazê-lo explicitamente.

[***Orientado a Objetos***](https://www.google.com/search?sca_esv=7b647de37a4058b4&sxsrf=AE3TifNQENaORotp7Qd9CCnuKv_kjUIvmQ%3A1754766694312&q=Orientado+a+Objetos&sa=X&ved=2ahUKEwj3rOmrt_6OAxWuBLkGHaKRG04QxccNegUIjQIQAQ&mstk=AUtExfB5_lmFt3zPvVSITDkcwk0mIJUErFFySXxbkPxiK2vo_hddH0dnltYYe2mxHL9gC4myDAyAHLXuYqNvjIlZzJjBMtlWWKaxgC7ORyLQ2GPThFYVi5FihyFiGNDLsE7kAuG7BhD6dX0dyIp37KQeP2nQ3nLfUyT45aB__42tLFdagqEOrt0K3xCdC_ttjBQxAcBSLDL-A0JoHG_KcsNOAsmBVVKduO4dH88s9cNiu298VLv2SkmEMiVzfQwIhKJ0caA8ob5jknO3QuyIDfRAxpXz&csui=3)***:***

Organiza o código em torno de objetos que encapsulam dados e comportamentos.

[***Funcional***](https://www.google.com/search?sca_esv=7b647de37a4058b4&sxsrf=AE3TifNQENaORotp7Qd9CCnuKv_kjUIvmQ%3A1754766694312&q=Funcional&sa=X&ved=2ahUKEwj3rOmrt_6OAxWuBLkGHaKRG04QxccNegUIjgIQAQ&mstk=AUtExfB5_lmFt3zPvVSITDkcwk0mIJUErFFySXxbkPxiK2vo_hddH0dnltYYe2mxHL9gC4myDAyAHLXuYqNvjIlZzJjBMtlWWKaxgC7ORyLQ2GPThFYVi5FihyFiGNDLsE7kAuG7BhD6dX0dyIp37KQeP2nQ3nLfUyT45aB__42tLFdagqEOrt0K3xCdC_ttjBQxAcBSLDL-A0JoHG_KcsNOAsmBVVKduO4dH88s9cNiu298VLv2SkmEMiVzfQwIhKJ0caA8ob5jknO3QuyIDfRAxpXz&csui=3)***:***

Utiliza funções como blocos de construção primários, evitando modificação de estado e efeitos colaterais.

<https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o_multiparadigma>

É possível usar o JavaScript fora dos navegadores através de *runtimes* como o Node.js.

***Tipos de Dados***

***Dados são uma coleção de fatos, números, palavras, observações ou outras informações úteis.*** Por meio do processamento de dados e da análise de dados, as organizações transformam dados brutos em insights valiosos que melhoram a tomada de decisões e geram resultados de negócios melhores.

[*https://encurtador.com.br/01ODp*](https://encurtador.com.br/01ODp)

No JavaScript (e em outras linguagens de programação) dados podem ser representados em diferentes formas:

***Tipo de dados primitivos:***

**number e bigint** (serve para números inteiros muito grandes. Adicione um “n” no final.)

**string (**lista de caracteres como textos, palavras, frases ou qualquer conjunto de símbolos que podem ser lidos e interpretador por um computador.

**Boolean** (é um tipo de dado que tem valor True ou False)

**Null e undefined** (null = Valor vazio x undefined = Não foi atribuído nenhum valor.)

***Tipo de dados não primitivos:***

**Objects (Objetos):** Estruturas de dados que permitem armazenar múltiplos pares chave-valor, onde cada chave é uma string (ou símbolo) e o valor pode ser qualquer tipo de dado, incluindo outros objetos ou funções.

**Arrays (Vetor):** Listas ordenadas de valores, que podem ser de qualquer tipo de dado. Os arrays são um tipo específico de objeto em JavaScript.

**Functions (Funcões):** Objetos especiais que podem ser invocados para executar um bloco de código. Funções são consideradas objetos de primeira classe em JavaScript, podendo ser passadas como argumentos para outras funções, retornadas de outras funções e atribuídas a variáveis.

**Dates (Datas):** Objetos que representam datas e horas.

**RegEx (Expressões regulares):** Objetos que representam padrões de texto para busca e manipulação de strings.

***Variáveis***

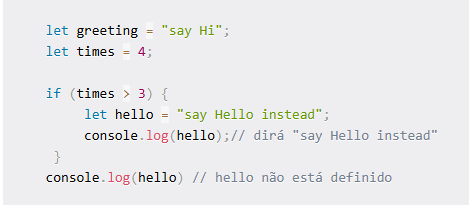
As variáveis servem para armazenar os dados para podermos usá-los no nosso código. As existem três tipos de variáveis: var, let e const. Ex:

var nome = “Levi” – let nome = “Levi” – const nome = “Levi”

Cada um desses três tipos tem seus próprios sentidos:

Var = Tem escopo global ou de função. Isso significa que elas podem ser acessadas de qualquer lugar do código; dentro da função em que são declaradas ou em todo o documento (se forem declaradas fora de qualquer função). **Elas também podem ser declaradas de novo e atualizada.**

Let = A principal diferença entre var e let é o escopo. Variáveis declaradas com var têm escopo global ou de função, enquanto variáveis declaradas com let têm escopo de bloco (ela pode ser usada no escopo global, porém não é recomendado). Diferente de Var, a variável let não pode ser declarada novamente dentro de seu escopo.



A variável let pode ser atualizada, mas não declarada novamente. Porém, se a mesma variável for definida em escopos diferentes, não haverá erro já que são tratadas como variáveis diferentes (em escopos diferentes).

Const = Variáveis declaradas com const têm escopo de bloco e mantém seus valores constantes. Isso significa que elas só podem ser acessadas no bloco de código em que são declaradas. Ela não pode ser atualizada nem declarada novamente.

* As declarações de var têm escopo global ou de função, enquanto as declarações de let e de const têm escopo de bloco.
* Variáveis de var podem ser atualizadas e declaradas novamente dentro de seu escopo. As variáveis de let podem ser atualizadas, mas não podem ser declaradas novamente. As variáveis de const não podem ser atualizadas nem declaradas novamente.
* Todas elas passam por hoisting para o topo de seu escopo. Porém, enquanto variáveis com var são inicializadas com undefined, as variáveis com let e const não são inicializadas.
* Enquanto var e let podem ser declaradas sem serem inicializadas, const precisa da inicialização durante a declaração.

***Operadores e expressões***

**Operadores aritmético:**

Alguns operadores são: +(Soma), -(Subtração), \*(Multiplicação), /(Divisão(por padrão é a numeração real 5/2= 2.5), %(Resto da divisão), \*\*(potência). Todos esses são considerados operadores binário, pois necessitam de 2 números para funcionar.

No JavaScript a ordem de precedência para operadores aritméticos é: ()Parenteses> \*\*Potência > /Divisão - \* Multiplicação - %Resto da divisão (esses 3 tem o mesmo valor de precedência, porém será priorizado da esquerda para direita) > + Soma -Subtração.

Assim como na matemática, você pode usar os parênteses para priorizar certos cálculos.

**Operadores de atribuição:**

*Var n = 3 (Exemplo)*

n = n + 4 virou 7, agora de forma simplificada: n+= 4 essa simplificação só serve para auto-atribuições

n = n – 5 virou 2, agora de forma simplificada: n-= 5

n = n \* 4 virou 8, agora de forma simplificada: n\*= 4

n = n / 2 virou 4, agora de forma simplificada: n/= 2

n = n \*\* 2 virou 16, agora de forma simplificada: n \*\*= 2

n = n % 5 virou 1, agora de forma simplificada: n%= 5

**Operadores de encadeamento:**

Ele serve para encadear uma coisa na outra:

Ex: Console.error

**Operadores de incremento:**

Os operadores de incremento servem para adicionar ou remover um(1) ao valor de uma variável.

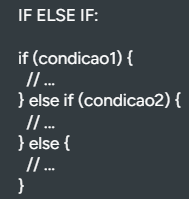
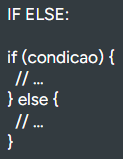
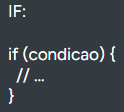
É possível usar ele pré-fixado e pós-fixado: ++x e x++ (x é um valor, ex= 5)

***Estruturas condicionais: If e Else***

Permite criar blocos de código que vão ser executados somente quando determinada operação lógica for verdadeira.

Ex: **SE(if)** condição === true **ENTÃO** faça algo

Além disso, pode ser usada com o **Else** para acrescentar um bloco que vai ser executado somente quando essa operação lógica for falsa.



**Operadores de ternário:**

Forma **abreviada** e **autoavaliada** do If Else:

const resultado = condição ? resultado\_verdadeiro : resultado\_falso

**Estrutura condicional: Switch**

Permite criar blocos condicionais como o IF, porém com uma estrutura que avalia uma expressão e testa o resultado contra vários casos, executando os casos correspondentes.

É possível definir um caso padrão, que sempre é executado.

Também é possível parar a execução após algum dos casos.

SWITCH (expressão)

CASO a: faça uma coisa

CASO b: faça outra coisa E PARE

CASO C: faça mais outra coisa

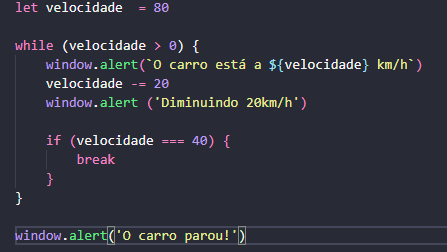
POR PADRÂO: faça algo sempre

***Estruturas de repetição: While***

Estruturas de repetição servem para executar um mesmo bloco de código várias vezes.

O while serve para criar um laço de repetição que será executado enquanto uma determinada condição for verdadeira.

O while e todas as outras estruturas de repetição que veremos aceitam o break para parar a repetição a qualquer momento

****

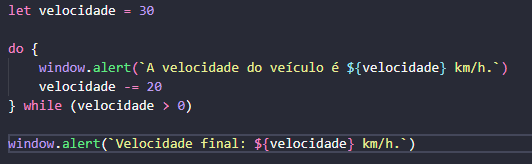
**Atenção:** *é importante que sempre exista uma forma da condição se tornar falsa, ou então o bloco ficará se repetindo para sempre.*

***Estruturas de repetição: Do while***

O Do While é muito parecido com o While, porém ele garante que o bloco de código será executado pelo menos uma vez, mesmo que a condição seja falsa.

No Do While a primeira verificação da condição só acontece depois que o bloco foi executado uma vez. Portanto, mesmo que seja falsa, ela ira executar uma vez.

**FAÇA alguma coisa ENQUANTO (condição):**



***Estruturas de repetição: For***

O **for** também serve para realizar repetições, porém com uma estrutura diferente do **WHILE** e do **DO WHILE**.

Ele existe para ***abreviar o código*** em casos que seguem essa estrutura **específica**;

Inicialização: expressão executada antes de tudo.

Condição: condição semelhante ao **while**, é testada antes da execução do bloco de repetição

Finalização: expressão executada após o bloco de repetição.

Estrutura > ***PARA inicialização; condição; finalização FAÇA alguma coisa***

*!!!Lembre-se que o resultado precisa ser booleano!!!*

For (inicialização; condição; finalizacao) {

//...

}

