Resumão Javascript

Ferramentas Essenciais

1. **Um Editor de Texto/Código:**
    - Você precisa de um programa para escrever e salvar seu código.
    - **Opções populares (e gratuitas): VS Code** (Visual Studio Code), Sublime Text ou Atom.
2. **Um Navegador Web Moderno:**
    - O JavaScript é executado principalmente nos navegadores (como Chrome, Firefox, Edge, Safari).
    - Você o usará para **testar** seu código e ver os resultados. A maioria tem **Ferramentas de Desenvolvedor** (Developer Tools/DevTools) integradas, que são cruciais para depurar (encontrar erros).
3. **Noções Básicas de HTML e CSS (Recomendado):**
    - Embora não sejam *estritamente* JavaScript, a maior parte do JS básico é usada para manipular elementos de páginas web.
    - **HTML** (estrutura da página) e **CSS** (estilo/aparência) são o "trio" da web e saber o básico deles facilitará muito a visualização e aplicação do seu JS.

Conceitos Fundamentais de JavaScript

Estes são os tópicos de programação que você deve focar em aprender primeiro:

1. **Sintaxe Básica e Estrutura:**
    - Como o código é escrito e lido (ponto e vírgula, comentários).
    - **Variáveis:** Entender `var`, `let`, e `const` para armazenar informações.
2. **Tipos de Dados:**
    - Saber os diferentes tipos de informação que você pode usar: **Strings** (texto), **Numbers** (números), **Booleans** (verdadeiro/falso), **Arrays** (listas) e **Objects** (estruturas de dados mais complexas).
3. **Operadores:**
    - Como fazer cálculos (**aritméticos:** `+`, `-`, `*`, `/`) e como fazer comparações (**comparação:** `==`, `===`, `>`, `<`, `!=`).
4. **Estruturas de Controle de Fluxo:**
    - **Condicionais (`if/else` e `switch`):** Para executar código diferente dependendo de uma condição (decisões).
    - **Loops (`for` e `while`):** Para repetir um bloco de código várias vezes (repetição).
5. **Funções:**
    - Aprender a **declarar** e **chamar** funções para agrupar e reutilizar blocos de código.
6. **Introdução ao DOM (Document Object Model):**
    - Aprender como o JavaScript **interage com o HTML**.
    - Como **selecionar** elementos da página, **mudar** seu conteúdo ou estilo, e **responder** a eventos (como cliques de mouse ou digitação).

---

1. Variáveis e Tipos de Dados

Variaveis

As **Variáveis** são um conceito dentro da programação de uma estrutura que serve para armazenar valores.

Dentro do Javascript, declaramos uma variável com o uso dos termos `let` (mutável) e `const` (constante, imutável)

```
const nome = 'Danilo'
let idade = 24
```

Tipos Primitivos de Dados

De forma geral, as linguagens de programação conseguem armazenar uma série de tipos de dados, desde mais simples a mais complexos. Dentre os principais que precisamos aprender temos:

Number (números)

```
let idade = 24
let saldo = 64.90
```

Note que diferente da notação em português, nao utilizamos a `',' (virgula)`, mas sim `'.' (ponto)` para separação das casas decimais

Além disso é importante ressaltar que dentro do javascript, diferente de outras linguagens, não temos uma distinção de tipo dentro numeros inteiros (int), reais (float)

String (texto)

```
let nome = "Danilo"
let profissão = 'Educador'
let frase = " Eu acredito que às vezes são as pessoas que ninguém espera nada que fazem as coisas que ninguém consegue imaginar. - A
```

Estruturas de texto em `JS` precisam ser registradas com `' (aspas simples)` ou `" (aspas duplas)`, além disso podem armazenar multiplas palavras.

Boolean (verdadeiro/falso)

```
let acordado = true
let  = false
```

Os booleanos serão um tipo de dado fundamental quando formos ver sobre o funcionamento de condicionais e os operadores lógicos e de comparação

Arrays (Listas)

```
const cores = ['vermelho', 'azul', 'verde'];
const notas = [8.6, 7, 4.8]
```

Listas ordenadas de valores

Os elementos de um array podem ser acessados por meio de sua posição, é importante ressaltar que os Arrays começam a partir da posição `0 (zero)`

```
const frutas = ["Banana","Uva", "Pera", "Maça"]
let favorita = frutas[1] // Armazenar Uva
```

Outro recurso importante é a propriedade `length`, por meio dela podemos acessar a quantidade de elementos de um array.

```
const notas = [10, 8, 6, 7]
let quantidade = notas.length
```

Objetos

```
const pessoa = { nome: 'Bob', idade: 25 };

const carro = {
    modelo: "Uno",
    marca: "Fiat",
    ano: 1984,
    placa: "XYZ 9876"
}
```

Os Objetos são coleções de pares chave-valor, usadas para representar entidades complexas.

EXERCICIO

Exercicio 1

Crie um programa chamado `variaveis.js`, em que voce deve declarar 4 variaveis: seu nome, sua idade, se você é casado e sua profissão dos sonhos.

Exercicio 2

Crie um programa chamado `arrays.js`, em que voce deve criar 3 arrays, um contendo sua lista de filmes favoritos, outro contendo sua data de nascimento no formato `[dia, mes, ano]` (exemplo [14,5,2000]) e uma lista de compras de mercado.

Exercicio 3

A partir dos valores do exercicio 1 crie agora um arquivo agora chamado `pessoa.js` em que em vez de variaveis independentes voce crie um objeto pessoa com os mesmos atributos

2. Operadores

Os operadores são os simbolos essencias para o funcionamento de um programa. Dentre eles, cada um tem sua propria função dentro do código, entre os principais operadores temos:

Atribuição

Armazenam valores (`=` , `+=` , `-=`)

```
let x = 10
x += 5 // Incrementa o valor em 5
```

Além dos operadores `+=` e `-=` , temos também `++` e `--` incrementar e decrementar em 1 o valor de uma variavel, esses serão bem comuns ao trabalhar-mos com as estruturas de Loops

Aritméticos

Realizam operações matemáticas (`+` , `-` , `*` , `/` , `%`).

```
let soma = 5 + 3
```

```
let x = 10
let y = 20
let z = (y - x) * 2
```

Comparação

Compararam valores, retornando um booleano (`true` ou `false`).

```
let x = 5
let y = 10

let maior = x > y // maior vai ser falso
```

Dos principais operadores temos:

1. Maior `>`
2. Maior ou Igual `>=`
3. Menor `<`
4. Menor ou Igual `<=`
5. Igual `==`
6. Estritamente Igual `===`

```
2 == "2" // Verdadeiro
2 === "2" // Falso
```

Lógicos

Combinam expressões booleanas

1. `&&` (E)
2. `\|\|` (OU)
3. `!` (Negação/Inversor)

```
let x = 10
let y = 5
let z = 0

(x > y && z > y) // Falso
(x > y || z > y) // Verdadeiro
(!(y > x)) // Verdadeiro
```

Exercicios

Exercício 1: Manipulação de Variáveis e Operadores Aritméticos/Atribuição

Objetivo: Criar um arquivo `Estoque.js` que gerencie o estoque de um produto, aplicando operações de atribuição e aritméticas.

Instruções:

1. Crie uma variável chamada `estoque_atual` e atribua o valor inicial de `100`.
2. Crie uma variável chamada `vendas_dia` e atribua o valor de `35`.
3. Utilize um operador de atribuição combinado (`-=`) para diminuir o `estoque_atual` pelo valor de `vendas_dia`.
4. O gerente decide adicionar **10** novas unidades ao estoque. Utilize o operador de incremento (`++` ou `+= 1`) **dez vezes** ou o operador de atribuição combinado (`+= 10`) para realizar esta adição.
5. Crie uma variável chamada `produtos_por_caixa` e atribua o valor `12`.
6. Calcule quantos produtos "sobraram" fora das caixas completas (ou seja, o resto da divisão) e armazene este valor em uma nova variável chamada `estoque_extra`.
7. Imprima no console o valor final de `estoque_atual` e de `estoque_extra`.

Exercício 2: Condicional Simples com Operadores de Comparação Estrita

Objetivo: Criar um arquivo `Acesso.js` que valide um dado de entrada garantindo que o tipo e o valor sejam os esperados, utilizando o operador de igualdade estrita (`===`).

Instruções:

1. Crie duas variaveis `pin_informado` e `pin_correto`.
2. Defina a variável `pin_correto` com o valor numérico `4567`.
3. O argumento `pin_informado` será o valor que o usuário digitou, que pode ser uma *string* ou um *number*.
4. Utilize uma estrutura `if/else` e o operador **estritamente igual (`===`)** para verificar se o `pin_informado` é idêntico (valor e tipo) ao `pin_correto`.
5. Se a condição for verdadeira, deve retornar a mensagem: `"Acesso Liberado!"`.
6. Se a condição for falsa, deve retornar: `"Erro: PIN ou Tipo Inválido."`.

Exercício 3: Lógica Complexa com Operadores Lógicos

Objetivo: Criar um arquivo `Promocao.js` que determine a elegibilidade para uma promoção combinando múltiplas condições usando os operadores lógicos (`&&`, `||`, `!`).

Instruções:

1. Crie 3 variaveis booleanas
   - `assinatura_ativa` (Ex: `true` se tem)
   - `primeira_compra` (Ex: `true` se é a primeira)
   - `conta_suspensa` (Ex: `true` se estiver suspensa)
2. Crie um codigo que valide uma promoção quando a pessoa for elegivel. Uma pessoa é elegível se:
   - Ela tem uma `assinatura_ativa` **OU**
   - (Ela está fazendo a `primeira_compra` **E** a sua `conta_suspensa` **NÃO** é verdadeira)
3. Utilize a combinação de operadores `&&`, `||`, e `!` para expressar essa lógica e armazene o resultado em uma variável `elegivel`.
4. Imprima o valor booleano de `elegivel`, substitua os valores das variaveis iniciais.

3. Condicionais

Executa blocos de código diferentes com base em uma condição booleana.

If / Else

```
let numero = 5
if(numero > 0){
    console.log("Numero é positivo")
}else if(numero < 0){
    console.log("Numero é negativo")
}else{
    console.log("Zero")
}
```

Exercicios

Exercício 1: Classificação Simples com If / Else (Votação)

Objetivo: Crie um arquivo `Votacao.js` Praticar a decisão binária (`true` ou `false`) usando `if` e `else` diretamente no script.

Instruções:

1. Crie uma variável chamada **`idade`** e atribua a ela um valor numérico (ex: `15` ou `20`).
2. Use a estrutura **`if / else`** para verificar se a pessoa tem **16 anos ou mais** (idade mínima para votar no Brasil).
3. Utilize `console.log()` para imprimir a mensagem apropriada:
    - Se puder votar: `"Pode votar!"`
    - Caso contrário: `"Não pode votar ainda."`

Exercício 2: Múltiplas Classificações com If / Else If / Else (Notas)

Objetivo: Crie um arquivo `Notas.js` para praticar o encadeamento de condições (`if` , `else if` , `else`) para cobrir múltiplos cenários.

Instruções:

1. Crie uma variável chamada **`nota_aluno`** e atribua a ela um valor numérico entre 0 e 100.
2. Use a estrutura **`if / else if / else`** para classificar e imprimir o **Conceito** da nota de acordo com as seguintes regras:
    - Se a nota for **maior ou igual a 90**: imprima `"Conceito A"` .
    - Se a nota for **maior ou igual a 70 E menor que 90**: imprima `"Conceito B"` .
    - Se a nota for **menor que 70**: imprima `"Conceito C"` .

Exercício 3: Controle de Acesso com Condições Lógicas Compostas

Objetivo: Crie um arquivo chamado `Login.js` para praticar o uso de **operadores lógicos (`&&` ou `||`)** em conjunto com a estrutura de controle (`if / else`).

Instruções:

1. Crie duas variáveis booleanas: **`estaLogado`** e **`temAssinatura`** , e atribua a elas valores `true` ou `false` de sua escolha.
2. Use o operador lógico **`&&` (E)** para combinar as duas variáveis dentro do seu `if` .
3. O acesso Premium **só** deve ser liberado se **ambas** as variáveis forem `true` .
4. Utilize `console.log()` para imprimir a mensagem apropriada:
    - Se a condição for verdadeira: `"Acesso Premium Liberado!"`
    - Caso contrário: `"Acesso Negado. Verifique login e assinatura."`

4 Loops

Repete um bloco de código enquanto uma condição for verdadeira.

For

```
    for(let i = 0; i < 5; i++){
        console.log(i)
    }
```

Repete um bloco de código um número determinado de vezes. Entendemos a estrutura for a partir de:

```
for(Condicao Inicial; Condição de Parada; Passo){
    Codigo;
}
```

Iteração sobre Arrays

Ideal para iterar sobre Arrays.

```
let nomes = ["Danilo", "João", "Isabela", "Julia"]
for(let i = 0; i < nomes.length; i++){
    console.log(nomes[i])
}
```

```
let nomes = ["Danilo", "João", "Isabela", "Julia"]
for(let n of nomes){
    console.log(n)
}
```

While

```
let contador = 0
while(contador < 10){
    console.log(contador)
}
```

Do While

```
let dinheiro = 100
do{
    dinheiro -= 10
}while(dinheiro > 0)
```

Exercicios

Exercício 1: Contador e Decisão (Loop `for`)

Objetivo: Crie um arquivo chamado `Sorte.js` para usar o laço `for` para repetição de um número determinado de vezes e aplicar uma Estrutura de Controle de Fluxo (`if`) dentro do loop.

Instruções:

1. Escreva um laço **`for`** que conte e itere de `1` até `10` (inclusive).
2. Dentro do laço, use `console.log()` para imprimir o número da iteração.
3. Adicione uma estrutura **`if`** dentro do laço: se o número atual for igual a `7`, imprima a mensagem `"Chegamos ao número da sorte!"`.

Exercício 2: Repetição por Condição (Loop `while`)

Objetivo: Crie um arquivo chamado `Jogo.js` para usar o laço `while` para repetir um bloco de código enquanto uma condição de estado inicial permanecer verdadeira.

Instruções:

1. Crie uma variável chamada **`vida_jogador`** e atribua o valor inicial `50`.
2. Crie uma variável chamada **`dano_por_rodada`** e atribua o valor `8`.
3. Escreva um laço **`while`** que execute o bloco de código **enquanto** `vida_jogador` for maior que `0`.
4. Dentro do loop, diminua a `vida_jogador` pelo valor de `dano_por_rodada` (use um operador de atribuição combinado, como `-=`).
5. Em cada repetição, use `console.log()` para mostrar a vida restante: `"Vida restante: [valor]"`.

Exercício 3: Iteração e Formatação de Arrays (Loop `for...of`)

Objetivo: Crie um arquivo chamado `Produtos.js` para praticar a iteração direta sobre elementos de um **Array** usando o laço `for...of`.

Instruções:

1. Crie um Array chamado **`lista_produtos`** com os seguintes nomes de produtos (strings): `"Teclado"`, `"Mouse"`, `"Monitor"`, `"Webcam"`.
2. Escreva um laço **`for...of`** para percorrer **todos** os elementos do Array.
3. Dentro do laço, use `console.log()` para imprimir a mensagem: `"Produto: [Nome do Produto]"`.

5. Funções

As funções são blocos de código reutilizáveis que podem receber **parâmetros** e retornar um **valor**.

Declaração

```
function saudacao(nome) {
    return 'Olá, ' + nome;
}
```

Chamada

```
let texto = saudacao("Lima")
```

A partir da chamada de conseguimos executar o código dentro da função.

Arrow Functions

```
const somar = (a, b) => a + b;
```

Sintaxe mais concisa e moderna para declarar funções.

Exercicios

Ótimo! Agora que apresentamos as **funções**, podemos praticar a criação de blocos de código reutilizáveis.

Aqui estão 3 exercícios simples, focados na declaração e chamada de funções (incluindo *Arrow Functions*):

Exercício 1: Declaração de Função Padrão e Retorno

Objetivo: Criar um arquivo `Triplo.js` com uma função que recebe um valor, realiza um cálculo simples e retorna o resultado.

Instruções:

1. Declare uma função chamada **`triplicarNumero`** que aceite um único **parâmetro** (`num`).
2. Dentro da função, calcule e use a instrução **`return`** para devolver o valor do número multiplicado por **3**.
3. Crie uma variável chamada **`resultado`** e chame a função, passando o número `7` como argumento.
4. Imprima o valor da variável `resultado` no console.

Exercício 2: Função com Lógica Interna e Múltiplos Parâmetros

Objetivo: Criar um arquivo `Maioridade.js` com a função que utiliza a **estrutura de controle de fluxo** (`if/else`) para determinar seu retorno, baseada em dois parâmetros.

Instruções:

1. Declare uma função chamada **`checarMaioridade`** que aceite dois **parâmetros**: `idade` (número) e `idadeMinima` (número, use `18`).
2. Dentro da função, use uma estrutura **`if/else`** para verificar se a `idade` é **maior ou igual** à `idadeMinima` .
3. Se a condição for verdadeira, retorne a string `"É maior de idade."` .
4. Caso contrário, retorne a string `"É menor de idade."` .
5. Chame a função com um valor de sua escolha (ex: `17`) e imprima o valor retornado no console.

Exercício 3: Sintaxe de Arrow Function (`=>`)

Objetivo: Crie um arquivo `Media.js` Praticar a sintaxe concisa e moderna das *Arrow Functions*.

Instruções:

1. Declare uma **Arrow Function** chamada **`calcularMedia`** .
2. A função deve aceitar **três parâmetros** (`n1` , `n2` , `n3`).
3. A função deve calcular a média aritmética dos três números e **retornar** esse valor de forma concisa (em uma única linha, como no exemplo `somar`).
4. Chame a função com os valores `10` , `20` e `30` , e imprima o resultado no console.