

Java Script

1

→ Saída de dados no console

* `console.log(dado);`

→ Comentários

* `// comentário` : comentário de linha.

* `/* comentário */` : comentário de bloco.

→ Para criar variáveis utilizar let e não var.

* `let variavel = valor;`

* `let variavel;`

→ Constantes

* `const constante = valor;`

→ Tipos de dados primitivos

* `typeof dado` : informa o tipo de dado.

* strings: podem ser representadas por aspas simples, aspas duplas ou crase.

- `let variavel = 'texto';`
- `let variavel = "texto";`
- `let variavel = `texto`;`

* Number

- `let variavel = numero;`

* Boolean: pode ser true ou false.

- `let variavel = true;`

`let variavel = false;`

* **Undefined**: ocorre sempre que a variável não é inicializada, ou seja, não recebe valor inicial. Não aponta para nenhum local na memória. Q

- **let variavel;**

* **Null**: significa nulo. Não aponta para nenhum local na memória.

- **let variavel = null;**

→ Operadores

* Operadores Aritméticos

- **+** : adição / concatenação
- **-** : subtração
- **/** : divisão
- ***** : multiplicação
- ****** : exponenciação
- **%** : resto da divisão

* Operadores de atribuição

- **let x = Valor;**
- **x += Valor;**
- **x -= Valor;**
- **x /= Valor;**
- **x *= Valor;**
- **x **= Valor;**
- **x %= Valor;**

* Operadores de incremento

- **x++;** : incrementa em 1
- **x--;** : decrementa em 1

(3)

* Operadores Relacionais

- $x == y$: Valores iguais retornam true.
- $x === y$: Valores e tipos de dado iguais retornam true.
- $x != y$: Valores diferentes retornam true.
- $x !== y$: Valores ou tipos de dado diferentes retornam true.
- $x > y$: maior que
- $x < y$: menor que
- $x >= y$: maior ou igual a
- $x <= y$: menor ou igual a
- $?$: operador ternário

* Operadores Lógicos

- $\&&$: And
- $||$: Or
- $!$: Not

→ Conversão de string para number:

- * $x = parseInt(texto);$
- * $x = parseFloat(texto);$
- * $x = Number(texto);$

→ Saída de dados no navegador

- * $window.alert(mensagem);$: mostra um alerta.
- * $window.confirm(mensagem);$: abre uma janela de confirmação que retorna true ou false.
- * $window.prompt(mensagem);$: abre uma janela com um input de texto.

→ Atribuição simultânea de variáveis

- * $[a, b, c] = [valor1, valor2, valor3];$

→ Strings

* Caractere de escape: barra invertida (\)

- \' : imprime aspas simples
- \" : imprime aspas duplas
- \\ : imprime barra invertida
- \n : nova linha

* Métodos e Funções

- `texto.charAt(indice);`: retorna o caractere na posição índice.
- `texto.charCodeAt(indice);`: retorna o código ASCII do caractere na posição índice.
- `texto.indexOf(termo);`: retorna o índice da primeira ocorrência de termo.
- `texto.indexOf(termo, indice);`: retorna o índice da primeira ocorrência de termo a partir da posição índice. Retorna -1 quando termo não é encontrado.
- `texto.lastIndexOf(termo);`: similar ao `indexOf` porém busca da direita para a esquerda.
- `texto.replace(original, novo);`: substitui um termo.
- `texto.length;`: retorna o tamanho da string.
- `texto.slice(inicio, fim);`: retorna os caracteres a partir da posição inicio até a posição fim não inclusiva.
- `texto.split(termo);`: gera um array utilizando termo como um separador.
- `texto.toUpperCase();`: torna todos os caracteres maiúsculos.
- `texto.toLowerCase();`: torna todos os caracteres minúsculos.

→ Numbers

* Conversão para string
- `numero.toString();`

* Conversão para binário
- `numero.toString(2);`

* Conversão para hexadecimal
- `numero.toString(16);`

* Arredondamento

- `numero.toFixed(nº de casas decimais);`

* Checagem de inteiro

- `Number.isInteger(numero);`

* Checagem de NaN (Not a Number)

- `Number.isNaN(numero);`

→ Objeto Math

* `Math.floor(numero);`: arredonda para baixo.

* `Math.ceil(numero);`: arredonda para cima.

* `Math.round(numero);`: arredonda normalmente.

* `Math.max(numeros);`: retorna o valor máximo em uma sequência de números.

* `Math.min(numeros);`: retorna o valor mínimo em uma sequência de números.

* `Math.random();`: gera um número pseudo aleatório entre 0 e 1 não inclusivo.

* `Math.PI;`: valor de π .

* `Math.pow(numero, expoente);`: exponenciação.

* `Math.sqrt(numero);`: raiz quadrada

* `Math.abs(numero);`: módulo.

* `Math.trunc(numero);`: retorna a parte inteira de numero.

→ Arrays

- * `vetor = [elemento1, elemento2, ...];`
- * `vetor.length;`: tamanho do vetor.
- * `vetor.push(elemento);`: adiciona elemento ao fim do array.
- * `vetor.unshift(elemento);`: adiciona elemento no inicio do array.
- * `vetor.pop();`: remove o ultimo elemento e o retorna.
- * `vetor.shift();`: remove o primeiro elemento e o retorna.
- * `vetor.slice(início, fim);`: separa os elementos de um array de inicio a fim não incluso.
- * Checagem se é vetor
 - `vetor instanceof Array;`
- * `vetor2 = [...vetor1];`: deep copy de vetor1.

→ Estruturas Condicionais

* If, else if e else

```

- if(condição) { : Se
  comandos
}

else if(condição2) { : Senão se
  comandos2
}

else { : Senão
  comandos3
}

```

* Operação Ternária

- Condição ? comando1 : comando2;

Caso a condição seja verdadeira, executa comando1 e caso seja falsa executa comando2.

* Switch/case

- switch (variavel) {

 case valor1:

 comandos1;

 break;

 case valor2:

 comandos2;

 break;

 ...

 default:

 comando padrão;

}

→ Objeto Date

* const data = new Date();

* Métodos

- data.getDate(): retorna dia

- data.getMonth(): retorna mês (de 0 a 11)

- data.getFullYear(): retorna ano

- data.getHours(): retorna as horas

- data.getMinutes(): retorna os minutos

- data.getSeconds(): retorna os segundos

- data.getMilliseconds(): retorna os milisegundos

- data.getDay(): retorna os dias da semana (de 0 a 6)

- data.toString(): retorna data completa

→ Estruturas de Repetição

* For (clássico)

- **for (let i = inicio; condição; incremento) {**
 comandos;
}

* For in

- **for (let i in iterável) {** : i itera sobre os índices
 comandos;
} ou chaves do iterável, ou
 objeto.

* For of

- **for (let i of iterável) {** : i itera sobre os valores
 comandos;
} do iterável.

* While

- **while (condição) {**
 comandos;
}

* Do while : executa pelo menos uma vez.

- **do {**
 comandos;
} while (condição);

* Comando continue : pula para a próxima iteração.

* Comando break : interrompe o laço de repetição.

→ Tratamento de Erros

* throw

- `throw new tipo de erro (texto);` : cria um erro.

* try / catch / finally

```
- try {
    comandos;
} catch (erro) {
    comandos;
} finally {
    comandos;
}
```

<code>try {</code> <code>comandos;</code> <code>}</code> <code>catch (erro) {</code> <code>comandos;</code> <code>}</code> <code>finally {</code> <code>comandos;</code> <code>}</code>	<code>try:</code> executa caso não haja erro. <code>catch:</code> executa caso haja erro. <code>finally:</code> executa sempre.
---	---

→ Intervalos de tempo

* setInterval

- `const timer = setInterval(funcao, x);` : repete a função a cada `x` milissegundos.

* clearInterval

- `clearInterval(timer);` : interrompe o intervalo.

* setTimeout

- `setTimeout(funcao, x);` : executa a função após `x` segundos.

→ Funções

* Declaração de função

- `function funcao (parâmetros) {`
 `comandos;`
`}`

* Arrow Function

- `const funcao = (parâmetros) => {`
 `comandos;`
`}`

* Immediately Invoked Function Expression (IIFE)

10

- `(function (parâmetros) {
 comandos;
}) (argumentos);`

* Factory Function

- `function função (parâmetros) {
 return {
 chaves,
 métodos
 };
}`

* Constructor Functions

- `function Função (parâmetros) {
 this.atributo = valor;
 this.metodo = function (parâmetros2) {
 comandos;
 };
}`

* Generator Functions

- `function* função (parâmetros) {
 comandos;
 yield retorno1;
 Yield retorno2;
 Yield retorno3;
 ...
 return retorno final;
}`

→ Arrays (Avançado)

* Método Splice

- `vetor.splice(indice, quantidade, elementos);`

remove uma certa quantidade de elementos a partir do índice informado e adiciona elementos. Retorna um array com os elementos removidos.

* Concatenação de arrays

- `vetor1.concat(vetor2);`
- `Vetor = [...vetor1, ...vetor2];`

* Método filter

- `vetor.filter(function(valor, indice, vetor));`

Utiliza uma função que recebe o valor, índice e array completo e retorna true ou false para iterar sobre os elementos e gerar um array filtrado que será retornado

* Método map

- `vetor.map(function(valor, indice, vetor));`

Utiliza uma função que recebe o valor, índice e array completo e retorna o novo valor de cada elemento para iterar sobre os elementos e gerar um novo array.

* Método reduce

- `vetor.reduce(function(acumulador, valor, indice, vetor));`
- `vetor.reduce(function(acumulador, valor, indice, vetor), inicial);`

Utiliza uma função com acumulador para retornar o valor final do acumulador, reduzindo assim o array em um único elemento. Também pode simular filter e map.

→ Objetos

- * Representados por chaves.
- * Compostos por chaves e valores ou métodos.
- * Para impedir mudanças no objeto utiliza-se `freeze`.

- `Object.freeze(objeto);`

* Propriedades dos objetos:

- `Object.defineProperty(objeto, 'chave', { enumerable: true, value: valor, writable: true, configurable: true })`;
 - `enumerable: true`: mostra chave
 - `value: valor`: define valor
 - `writable: true`: permite alterar o valor
 - `configurable: true`: permite configurar e apagar a chave.

* Métodos

- `const objeto2 = Object.assign({}, objeto1);`: faz uma cópia de `objeto1` em `objeto2`.
- `const objeto2 = {...objeto1};`: faz uma cópia de `objeto1` em `objeto2`.
- `Object.keys(objeto);`: cria array com as chaves do objeto.
- `Object.getOwnPropertyDescriptor(objeto, 'chave');`: retorna um objeto com as propriedades da chave escolhida.
- `delete objeto.chave;`: apaga a chave escolhida.
- `Object.values(objeto);`: cria array com os valores do objeto.
- `Object.entries(objeto);`: cria array com as chaves e valores do objeto.

* Prototypes: todos os objetos herdam atributos e métodos de um protótipo.