OPERADORES

+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	MOD (resto de uma divisão) MOD igual a 0 significa que o primeiro número é múltiplo do segundo: • Ex: "10 % 5" é igual a 0, porque o resto da divisão é 0, ou seja, 10 é múltiplo de 5
**	Potenciação
<	Menor
>	Maior
=== ou ==	Igual
!== ou !=	Diferente
=	Atribuição
&&	E lógico • Ex: isso E aquilo
||	OU lógico • Ex: isso OU aquilo
Parênteses ()	Prioridade • Ex: "(2 + 2) / 4", vai fazer a soma primeiro que a divisão
Exclamação !	Negação • Ex: "true" é verdadeiro, mas "!true" é falso, porque é a "negação de true"
+=	Adição e atribuição • Ex: "x += 5" é igual a "x = x + 5"
-=	Subtração e atribuição • Ex: "x -= 5" é igual a "x = x – 5"
++	Incremento de uma unidade • Ex: "x++" é igual a "x = x + 1"
--	Decremento de uma unidade • Ex: "x--" é igual a "x = x – 1"
If ternário ?:	CONDIÇÃO ? SE_VERDADE : SE_FALSO `let ehMaior = 2 > 1 ? "2 é maior" : "2 não é maior";` • não é maior que 1" ○ CONDIÇÃO: 2 > 1 ○ SE_VERDADE: "2 é maior" ○ SE_FALSO: "2 não é maior"

CONDICIONAIS

IF ELSE

```
if (CONDICAO) {
  // se CONDICAO for true
} else {
  // se CONDICAO for false
}

if (2 === 10) {
  console.log(true);
} else {
  console.log(false);
}
// SAÍDA: false
```

IF TERNÁRIO

```
const RESPOSTA = CONDICAO ? true : false;

const RESPOSTA = 2 === 10 ? true : false;
// SAÍDA: false

const RESPOSTA = 2 === 10 ? "aaaaa" : "bbbbb";
// SAÍDA: bbbbb
```

SWITCH CASE

```
switch (key) {
  case value:
    break;
  default:
    break;
}

let disciplina = "Java";

switch (disciplina) {
  case "JavaScript":
    console.log("JavaScript");
    break;
  case "Java":
    console.log("Java");
    break;
  default:
    console.log("DISCIPLINA DESCONHECIDA");
    break;
}
// SAÍDA: Java
```

BIBLIOTECA MATH

Math.floor ()

- Arredonda um número para baixo e sem casas decimais (floor = chão)

```
let numero1 = 2.987;

let numero2 = Math.floor(numero1);
// Saída: 2
```

Math.ceil()

- Arredonda um número para cima e sem casas decimais (ceiling = teto)

```
let numero1 = 2.123;

let numero2 = Math.ceil(numero1);
// Saída: 3
```

Math.max()

- Retorna o maior número de uma lista de números

```
let listaNumeros = [1, 2, 3, 0, 7, -1];
let maiorNumero1 = Math.max(...listaNumeros);
let maiorNumero2 = Math.max(1, 2, 3, 0, 7, -1);
// Saída: 7
```

Math.min()

- Retorna o menor número de uma lista de números

```
let listaNumeros = [1, 2, 3, 0, 7, -1];
let menorNumero1 = Math.min(...listaNumeros);
let menorNumero2 = Math.min(1, 2, 3, 0, 7, -1);
// Saída: -1
```

Math.pow()

- Retorna a potência de dois números (base, expoente)

```
let potenciacao = Math.pow(2, 3);
// Saída: 8
```

Math.sqrt()

- Retorna a raiz quadrada de um número (sqrt = square root = raiz quadrada)

```
let raizQuadrada = Math.sqrt(25);
// Saída: 5
```

Math.random()

- Retorna um número aleatório (de 0 a 1)

```
let numeroAleatorio = Math.random();
// Saída: 0.29341468341317123
// Saída: 0.07737434429480872
```

- Para determinar o limite (mínimo ou máximo) do número aleatório, deve-se fazer:

```
limiteMin = 1;
limiteMax = 5;
// `Aleatório de 1 a 5
let numAleatorio1 = Math.random() * (limiteMax - limiteMin) + limiteMin;
// Saída: 4.284767619016935
// Saída: 2.34835451446026

limiteMin = 8;
limiteMax = 10;
// Aleatório de 8 a 10
let numAleatorio2 = Math.random() * (limiteMax - limiteMin) + limiteMin;
// Saída: 8.331675615450784
// Saída: 9.490376208540667
```

Classe/Objeto Number

Number()

- Cria um objeto Number. Usado para transformar outro tipo, como string e boolean, em número
    - Exemplo 1:

```
let numero;

let stringNumero = "1";
numero = Number(stringNumero);

console.log(stringNumero + 1);
// Saída: 11

console.log(numero + 1);
// Saída: 2

console.log(typeof stringNumero);
// Saída: string

console.log(typeof numero);
// Saída: number
```

    - Exemplo 2:

```
let booleanTrue = true;
let booleanFalse = false;
let numeroTrue = Number(booleanTrue);
let numeroFalse = Number(booleanFalse);

console.log(booleanTrue);
// Saída: true

console.log(booleanFalse);
// Saída: false

console.log(numeroTrue);
// Saída: 1

console.log(numeroFalse);
// Saída: 0

console.log(typeof booleanTrue);
// Saída: boolean

console.log(typeof numeroTrue);
// Saída: number
```

toFixed()

- Arredonda um número para a quantidade de casas decimais desejadas

```
let numero1 = 1.512349;

console.log(numero1.toFixed(0));
// Saída: 2

console.log(numero1.toFixed(1));
// Saída: 1.5

console.log(numero1.toFixed(5));
// Saída: 1.51235
```

Classe/Objeto String

String()

- Cria um objeto String. Usado para transformar outro tipo, como number e boolean, em número
    - Exemplo 1:

```
let numero = 1;
let strNumero = String(numero);

console.log(numero + 0 + 0);
// Saída: 1

console.log(strNumero + 0 + 0);
// Saída: “100”

console.log(typeof numero);
// Saída: number

console.log(typeof strNumero);
// Saída: string
```

chatAt()

- Retorna um caractere do string pela posição

```
let asdf = "abcde";

console.log(asdf.charAt(1));
// Saída: b
```

concat()

- Concatena/junta uma string a outra string ou número

```
let str = "abcde";

console.log(str.concat("AAA"));
// Saída: abcdeAAA

console.log(str + "AAA");
// Saída: abcdeAAA

console.log(str.concat("111"));
// Saída: abcde111

console.log(str + "111");
// Saída: abcde111
```

endsWith()

- Verifica se uma string termina com uma determinada sequência de caracteres (ends with = termina com)

```
let str = "abcdeAAAf";
```

```
console.log(str.endsWith("AAA"));
// Saída: false

console.log(str.endsWith("AAAf"));
// Saída: true
```

includes()

- Verifica se uma string contém uma determinada sequência de caracteres

```
let str = "abcdeAAAf";

console.log(str.includes("Ea"));
// Saída: false

console.log(str.includes("eA"));
// Saída: true
```

indexOf()

- Retorna a primeira posição que uma determinada sequência de caracteres aparece em uma string (-1: não existe)

```
let str = "abcdeAAAf";

console.log(str.indexOf("Ae"));
// Saída: -1

console.log(str.indexOf("eA"));
// Saída: 4
```

lastIndexOf()

- Retorna a última posição que uma determinada sequência de caracteres aparece em uma string (-1: não existe)

```
let str = "abcdeAAAf";

console.log(str.lastIndexOf("AB"));
// Saída: -1

console.log(str.lastIndexOf("A"));
// Saída: 7 (última vez que aparece)

console.log(str.indexOf("A"));
// Saída: 5 (primeira vez que aparece)
```

length

- Retorna o tamanho da string

```
let str = "abcdeAAAf";

console.log(str.length);
// Saída: 9
```

replace()

- Substitui uma sequência de caracteres de uma string

```
let str = "abcdef";

console.log(str.replace("cd", "CD"));
// Saída: abCDef
```

slice()

- Extrai uma sequência de caracteres de uma string

```
let str = "abcdef";

console.log(str.slice(2, 4));
// Saída: cd (começa na posição 1, termina na posição 4)

console.log(str.slice(2));
// Saída: cdef (começa na posição 1 e vai até o final da string)

console.log(str.slice(-2));
// Saída: ef (pega 2 posições, de trás pra frente)
```

split()

- Separa uma string em vetor/lista/array de acordo que uma sequência de caracteres aparece na string

```
let str = "31/12/2000";

let arr = str.split("/");

console.log(arr);
// Saída: [ '31', '12', '2000' ] (removeu as barras "/" e o que era separado por elas foi para o array)
```

startsWith()

- Verifica se uma string começa com uma determinada sequência de caracteres (starts with = começa com)

```
let str = "abcdeAAAf";

console.log(str.startsWith("Abc"));
// Saída: false

console.log(str.startsWith("abc"));
// Saída: true
```

substring()

- Extrai uma sequência de caracteres de uma string

```
let str = "abcdeAAAf";

console.log(str.substring(2, 4));
// Saída: cd
```

```
console.log(str.substring(4, 2));
// Saída: cd (não importa a ordem dos parâmetros)

console.log(str.substring(4));
// Saída: eAAAf (só aceita números positivos)
```

toLowerCase() e toUpperCase()

- Retorna a string toda em minúscula/maiúscula

```
let str = "abcdeAAAf";

console.log(str.toLowerCase());
// Saída: abcdeaaaf

console.log(str.toUpperCase());
// Saída: ABCDEAAAF
```

trim()

- Corta espaços vazios no início e final da string (trim = cortar/aparar)

```
let str = "   abcdeAAAf   aa   ";

console.log(str.trim());
// Saída: “abcdeAAAf   aa”
```

VETOR/LISTA/ARRAY

filter()

- Filtra uma lista de acordo com uma condição

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];

let listaFiltrada = lista.filter((num) => num > 4);
// Saída: [ 5, 6, 7, 8 ]
```

find()

- Encontra o primeiro item de uma lista que satisfaça a condição

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];

let item = lista.find((num) => num > 4);
// Saída: 5
```

findIndex()

- Retorna o index/posição do primeiro item de uma lista que satisfaça a condição

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];

let item = lista.findIndex((num) => num > 4);
// Saída: 4
```

includes()

- Verifica se existe um determinado item na lista

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];

let existeItem = lista.includes(10);
// Saída: false
```

indexOf()

- Retorna o primeiro index/posição do item que está procurando

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];

let index = lista.indexOf(10);
// Saída: -1
```

lastIndexOf()

- Retorna o último index/posição do item que está procurando

```
let lista = [1, 5, 3, 4, 5, 5, 7, 8];

let index = lista.lastIndexOf(5);
// Saída: 5
```

length

- Retorna o tamanho da lista

```
let lista = [1, 5, 3, 4, 5, 5, 7, 8];
```

```
let tamanhoLista = lista.length;
// Saída: 8
```

push()

- Adiciona um item na última posição da lista

```
let lista = [1, 5, 3, 4, 5, 5, 7, 8];

lista.push(1000);
// Saída: [1, 5, 3, 4, 5, 5, 7, 8, 1000]
```

pop()

- Remove o item da última posição da lista

```
let lista = [1, 5, 3, 4, 5, 5, 7, 8];

lista.pop();
// Saída: [1, 5, 3, 4, 5, 5, 7]
```

reverse()

- Inverte a lista

```
let lista = [1, 5, 3, 4, 5, 5, 7, 8];

lista.reverse();
// Saída: [8, 7, 5, 5, 4, 3, 5, 1]
```

shift()

- Remove o item da primeira posição da lista

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];

lista.shift();
// Saída: [2, 3, 4, 5, 6, 7, 8]
```

splice()

- Remove itens de acordo com a posição de um item na lista e pela quantidade de itens a serem removidos, e retorna uma lista com os itens removidos

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];
let itensRemovidos = lista.splice(2, 2);

// Saída (lista): [ 1, 2, 5, 6, 7, 8 ]
// Saída (itensRemovidos): [ 3, 4 ]
```

- Remove os itens e insere outros no lugar

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];
let itensRemovidos = lista.splice(2, 2, "teste", "teste2”);
// Saída (lista): [ 1, 2, 'teste', 'teste2', 5, 6, 7, 8 ]
```

- Insere itens em uma posição específica sem precisar remover

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];
let itensRemovidos = lista.splice(2, 0, "teste", "teste2");
```

```
// Saída (lista): [ 1, 2, 'teste', 'teste2', 5, 6, 7, 8 ]
```

toString()

- Converte lista para string

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];

let str = lista.toString();
// Saída: “1,2,3,4,5,6,7,8”
```

unshift()

- Insere um item na primeira posição da lista

```
let lista = [1, 2, 3, 4, 5, 6, 7, 8];

lista.unshift(999);
// Saída: [999, 1, 2, 3, 4, 5, 6, 7, 8]
```

OBJETOS

CLASS

```
class ObjetoPai {
  atributoPai1: number;
  atributoPai2: string | undefined;

  constructor(param1: number, param2?: string) {
    this.atributoPai1 = param1;
    this.atributoPai2 = param2;
  }
}

class ObjetoFilho extends ObjetoPai {
  atributoFilho1: boolean;
  atributoFilho2: any[] | undefined;

  constructor(paramPai1: number, paramPai2: string, paramFilho1: boolean)
{
    super(paramPai1, paramPai2);
    this.atributoFilho1 = paramFilho1;
  }
}

const objetoPai: ObjetoPai = new ObjetoPai(1, undefined);

console.log(objetoPai);
// SAÍDA: ObjetoPai { atributoPai1: 1, atributoPai2: undefined }

objetoPai.atributoPai2 = "atributo pai 2";
console.log(objetoPai);
// SAÍDA: ObjetoPai { atributoPai1: 1, atributoPai2: 'atributo pai 2' }

const objetoFilho: ObjetoFilho = new ObjetoFilho(1, "atributo pai 2",
true);
objetoFilho.atributoFilho2 = ["atributo filho 2"];

console.log(objetoFilho);
/**
  Saída:
  ObjetoFilho {
    atributoPai1: 1,
    atributoPai2: 'atributo pai 2',
    atributoFilho1: true,
    atributoFilho2: [ 'atributo filho 2' ]
  }
 */
```

INTERFACE

```typescript
interface ObjetoPai {
  atributoPai1: number;
  atributoPai2: string | undefined;
}

interface ObjetoFilho {
  atributoFilho1: boolean;
  atributoFilho2: any[] | undefined;
  objetoPai: ObjetoPai;
}

const objetoPai: ObjetoPai = {
  atributoPai1: 0,
  atributoPai2: undefined,
};

console.log(objetoPai);
// SAÍDA: { atributoPai1: 0, atributoPai2: undefined }

objetoPai.atributoPai2 = "atributo pai 2";
console.log(objetoPai);
// SAÍDA: { atributoPai1: 0, atributoPai2: 'atributo pai 2' }

const objetoFilho: ObjetoFilho = {
  atributoFilho1: false,
  atributoFilho2: undefined,
  objetoPai: {
    atributoPai1: 0,
    atributoPai2: undefined,
  },
};
objetoFilho.atributoFilho2 = ["atributo filho 2"];
objetoFilho.objetoPai.atributoPai2 = "atributo pai 2";

console.log(objetoFilho);
/**
  Saída:
  {
    atributoFilho1: false,
    atributoFilho2: [ 'atributo filho 2' ],
    objetoPai: { atributoPai1: 0, atributoPai2: 'atributo pai 2' }
  }
 */
```

CONSOLE.LOG()

Concatenação quando for printar/mostrar no console:

- **Exemplo 1**:
  - Código:

```
console.log("isso é 1 exemplo");
```

  - Resultado:

```
isso é 1 exemplo
```

- **Exemplo 2**:
  - Código:

```
const eh = "é";
const um = 1;
const exemplo = "exemplo";
const impressao = `isso ${eh} ${um} ${exemplo}`;
console.log(impressao);
```

  - Resultado:

```
isso é 1 exemplo
```

- **Exemplo 3**:
  - Código:

```
const eh = " é ";
const um = 1;
const exemplo = " exemplo";
const impressao = "isso" + eh + um + exemplo;
console.log(impressao);
```

  - Resultado:

```
isso é 1 exemplo
```

- **Exemplo 4**:
  - Código:

```
const eh = " é ";
const um = 1;
const exemplo = " exemplo";
console.log("isso" + eh + 1 + exemplo);
```

  - Resultado:

```
isso é 1 exemplo
```

- **Exemplo 5**:
  - Código:

```
const eh = "é";
const um = 1;
const exemplo = "exemplo";
console.log(`isso ${eh} ${um} ${exemplo}`);
```

  - Resultado:

```
isso é 1 exemplo
```

LAÇOS/ITERADORES (FOR/WHILE/DO-WHILE/FOR TERNÁRIO)

Comentado [F1]: Explicar escopo

for

```
for (INICIALIZA_VARIAVEIS;CONDIÇÃO;EXECUTA_NO_FIM_DE_CADA_ITERACAO) {
  // código
}
```

```
for (INICIALIZA_VARIAVEIS; CONDIÇÃO; INCREMENTO) {
  // código
}
```

```
INICIALIZA_VARIAVEIS;
for (; CONDIÇÃO; ) {
  // código
  INCREMENTO;
}
```

while

```
while (CONDIÇÃO) {
  // código
}
```

do-while

```
do {
  // código
} while (CONDIÇÃO);
```

for ternário

```
for (const ITEM_DA_LISTA of LISTA) {
  // código
}
```