FRONT END DESIGN ENGINEERING

FIAP

REVISÃO JS

Prof. Alexandre Carlos profalexandre.jesus@fiap.com.br

Prof. Luís Carlos lsilva@fiap.com.br

Prof. Wellington Cidade profwellington.tenorio@fiap.com.br

Operadores Aritméticos

Os operadores aritméticos realizam cálculos matemáticos com números (literais ou variáveis).

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
**	Exponenciação (ES2016)
/	Divisão
%	Módulo (Resto da divisão)
++	Incremento
--	Decremento

Operações Aritméticas

45697056

Uma operação aritmética típica envolve dois números. Esses números podem ser:

- **Literais:**

```
let x = 100 + 50;
```

- **Variáveis:**

```
let x = a + b;
```

- **Expressões:**

```
let x = (100 + 50) * a;
```

Operandos e Operadores

45697056

Em uma operação aritmética, os números são chamados de **operandos**, e a operação a ser realizada é definida por um **operador**.

Exemplo:

Operando	Operador	Operando
100	+	50

Operandos e Operadores

45697056

Adição

O operador de adição (`+`) soma números:

```
let x = 5;
let y = 2;
let z = x + y; // z = 7
```

Subtração

O operador de subtração (`-`) subtrai números:

```
let x = 5;
let y = 2;
let z = x - y; // z = 3
```

Multiplicação

O operador de multiplicação (`*`) multiplica números:

```
let x = 5;
let y = 2;
let z = x * y; // z = 10
```

Operandos e Operadores

45697056

Divisão

O operador de divisão (`/`) divide números:

```
let x = 5;
let y = 2;
let z = x / y; // z = 2.5
```

Resto da Divisão

O operador de módulo (`%`) retorna o resto da divisão:

```
let x = 5;
let y = 2;
let z = x % y; // z = 1
```

Incremento

O operador de incremento (`++`) aumenta o valor de um número em 1:

```
let x = 5;
x++;
let z = x; // z = 6
```

Operandos e Operadores

45697056

Decremento

O operador de decremento (`--`) diminui o valor de um número em 1:

```
let x = 5;
x--;
let z = x; // z = 4
```

Exponenciação

O operador de exponenciação (`**`) eleva o primeiro operando à potência do segundo:

```
let x = 5;
let z = x ** 2; // z = 25
```

O mesmo resultado pode ser obtido com `Math.pow()` :

```
let z = Math.pow(x, 2); // z = 25
```

Precedência de Operadores

45697056

A precedência de operadores define a ordem em que as operações são realizadas em uma expressão aritmética.

- Multiplicação (`*`) e divisão (`/`) têm precedência maior que adição (`+`) e subtração (`-`).
- Parênteses podem alterar a precedência, sendo calculados primeiro:

```
let x = (100 + 50) * 3; // x = 450
```

- Operações com a mesma precedência são calculadas da esquerda para a direita:

```
let x = 100 + 50 - 3; // x = 147
let y = 100 / 50 * 3; // y = 6
```

Conclusão Operadores Matemáticos

Os operadores aritméticos em JavaScript permitem realizar cálculos matemáticos de forma simples e eficiente. Compreender a precedência de operadores e o uso de parênteses é essencial para garantir que as expressões sejam avaliadas corretamente.

Tipos de Dados em JavaScript

Tipos de Dados em JavaScript

- String
- Number
- BigInt
- Boolean
- Undefined
- Null
- Symbol
- Object

O Tipo de Dado Object

O tipo de dado object pode conter tanto objetos embutidos quanto objetos definidos pelo usuário.

Tipos de objetos embutidos incluem:

- Objetos
- Arrays
- Datas
- Maps
- Sets
- Tipos de arrays específicos (intarrays, floatarrays)
- Promises, entre outros.

Exemplos:

```
// Números:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleanos:
let x = true;
let y = false;

// Objeto:
const person = {firstName:"John", lastName:"Doe"};

// Array:
const cars = ["Saab", "Volvo", "BMW"];

// Objeto de Data:
const date = new Date("2022-03-25");
```

Nota: Uma variável em JavaScript pode armazenar qualquer tipo de dado.

O Tipo de Dado Object

O tipo de dado object pode conter tanto objetos embutidos quanto objetos definidos pelo usuário.

Tipos de objetos embutidos incluem:

- Objetos
- Arrays
- Datas
- Maps
- Sets
- Tipos de arrays específicos (intarrays, floatarrays)
- Promises, entre outros.

Exemplos:

```
// Números:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleanos:
let x = true;
let y = false;

// Objeto:
const person = {firstName:"John", lastName:"Doe"};

// Array:
const cars = ["Saab", "Volvo", "BMW"];

// Objeto de Data:
const date = new Date("2022-03-25");
```

O Conceito de Tipos de Dados

Em programação, os tipos de dados são um conceito essencial. Para manipular variáveis corretamente, é importante conhecer seus tipos.

Sem tipos de dados definidos, um computador pode ter dificuldade em processar operações corretamente. Por exemplo:

```
let x = 16 + "Volvo";
```

O JavaScript interpretará essa operação como:

```
let x = "16" + "Volvo";
```

Regras do JavaScript para Operações com Strings e Números

- Se um número for somado a uma string, o JavaScript converterá o número em uma string antes de realizar a concatenação.

Exemplos:

```
let x = 16 + "Volvo";  // Resultado: "16Volvo"
let x = "Volvo" + 16;  // Resultado: "Volvo16"
```

O JavaScript avalia expressões da esquerda para a direita, o que pode gerar resultados inesperados:

```
let x = 16 + 4 + "Volvo";  // Resultado: "20Volvo"
let x = "Volvo" + 16 + 4;  // Resultado: "Volvo164"
```

Tipagem Dinâmica em JavaScript

- O JavaScript possui tipagem dinâmica, ou seja, uma mesma variável pode armazenar diferentes tipos de dados em momentos distintos.

Exemplo:

```
let x;        // x é undefined
x = 5;        // x agora é um número
x = "John";   // x agora é uma string
```

Strings em JavaScript

45697056

- Strings são cadeias de caracteres delimitadas por aspas simples ou duplas.

Exemplos:

```
let carName1 = "Volvo XC60";
let carName2 = 'Volvo XC60';
```

É possível incluir aspas dentro da string, desde que sejam diferentes das aspas que delimitam a string.

Exemplos:

```
let answer1 = "It's alright";
let answer2 = "He is called 'Johnny'";
let answer3 = 'He is called "Johnny"';
```

Números em JavaScript

- Todos os números no JavaScript são armazenados como números de ponto flutuante (decimal).

Exemplo:

```
let x1 = 34.00;
let x2 = 34;
```

Notação Exponencial

Números extremamente grandes ou pequenos podem ser escritos em notação científica.

Exemplos:

```
let y = 123e5;    // 12300000
let z = 123e-5;   // 0.00123
```

Números em JavaScript

- **BigInt em JavaScript**
- BigInt é um tipo de dado introduzido no ES2020 para armazenar números inteiros muito grandes.

Exemplo:

```
let x = BigInt("123456789012345678901234567890");
```

- **Booleanos em JavaScript**
- Booleanos podem ter apenas dois valores: true ou false.

Exemplo:

```
let x = 5;
let y = 5;
let z = 6;
(x == y)  // Retorna true
(x == z)  // Retorna false
```

Booleanos são frequentemente utilizados em testes condicionais.

Tipos Referência

- **Arrays em JavaScript**
- Arrays são coleções ordenadas de valores.

Exemplo:

```
const cars = ["Saab", "Volvo", "BMW"];
```

- Os índices de arrays começam em 0, ou seja, o primeiro item tem índice 0, o segundo 1, e assim por diante.

Tipos Referência

- **Objetos em JavaScript**
- Objetos são coleções de pares chave-valor.

Exemplo:

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

- O objeto person possui quatro propriedades: firstName, lastName, age e eyeColor.

TYPEOF

45697056

- **O Operador typeof**
- O operador typeof retorna o tipo de uma variável.

Exemplos:

```
typeof ""             // Retorna "string"
typeof "John"         // Retorna "string"
typeof 0              // Retorna "number"
typeof 3.14           // Retorna "number"
typeof (3 + 4)        // Retorna "number"
```

Undefined / Vazio

- **O valor Undefined**
- Se uma variável não tiver valor, seu tipo será undefined.

Exemplo:

```
let car;    // undefined
car = undefined;    // undefined
```

- **O valor Vazio**
- Um valor vazio ("") ainda é considerado uma string válida.

Exemplo:

```
let car = "";    // Valor vazio, mas tipo "string"
```

Conclusão Tipos de Dados

45697056

Conclusão

Os tipos de dados no JavaScript são fundamentais para o funcionamento da linguagem.

Compreender como cada tipo se comporta é essencial para evitar erros e construir aplicações robustas.

O uso correto de strings, números, arrays e objetos ajuda na escrita de códigos mais organizados e eficientes.

Além disso, o operador typeof auxilia na identificação de tipos, garantindo maior controle sobre os dados manipulados no programa.

Exercícios / Tipos de Dados

45697056

Crie um arquivo ***tipos.html*** e um arquivo ***tipos.js***, garantindo que o código siga as boas práticas de desenvolvimento.

O versionamento do código deve ser feito utilizando Git/GitHub.

Exercícios / Tipos de Dados

45697056

📌 **Exercício 1: Manipulação de Strings e Números:**

Crie uma variável chamada nome e armazene seu nome nela.

Em seguida, crie uma variável idade e atribua um número representando sua idade. Exiba no console a seguinte frase usando template literals:

"Meu nome é [seu nome] e eu tenho [sua idade] anos."

- Resolução (tipos.js):

```
const nome = "Carlos";
const idade = 25;

console.log(`Meu nome é ${nome} e eu tenho ${idade} anos.`);
```

Exercícios / Tipos de Dados

45697056

📌 Exercício 2: Operações Matemáticas

Crie duas variáveis num1 e num2, atribua valores numéricos a elas e realize as operações soma, subtração, multiplicação e divisão. Exiba os resultados no console.

🔹 Resolução (tipos.js):

```
const num1 = 10;
const num2 = 5;

console.log(`Soma: ${num1 + num2}`);
console.log(`Subtração: ${num1 - num2}`);
console.log(`Multiplicação: ${num1 * num2}`);
console.log(`Divisão: ${num1 / num2}`);
```

Exercícios / Tipos de Dados

45697056

📌 Exercício 3: Comparação de Valores (Booleanos)

Crie duas variáveis a e b com valores numéricos. Utilize operadores de comparação (==, ===, >, <, >=, <=) e exiba os resultados no console.

🔹 Resolução (tipos.js):

```
const a = 8;
const b = "8";


console.log(a == b);  // true (compara valor, ignora tipo)
console.log(a === b); // false (compara valor e tipo)
console.log(a > 5);   // true
console.log(b < 10);  // true
console.log(a >= 8);  // true
console.log(b <= 7);  // false
```

Exercícios / Tipos de Dados

45697056

📌 Exercício 4: Manipulação de Arrays

Crie um array chamado frutas contendo pelo menos 4 frutas. Adicione mais uma fruta ao final, remova a primeira fruta e exiba o array atualizado no console.

🔹 Resolução (tipos.js):

```
let frutas = ["Maçã", "Banana", "Laranja", "Uva"];
frutas.push("Pera");  // Adiciona ao final
frutas.shift();       // Remove o primeiro item

console.log(frutas);
```

Exercícios / Tipos de Dados

45697056

📌 Exercício 5: Criando e Manipulando Objetos

Crie um objeto chamado carro com as propriedades marca, modelo e ano. Exiba no console os valores dessas propriedades e adicione dinamicamente uma nova propriedade chamada cor.

🔹 Resolução (tipos.js):

```
const carro = {
  marca: "Toyota",
  modelo: "Corolla",
  ano: 2022
};

console.log(carro.marca); // Toyota
console.log(carro.modelo); // Corolla
console.log(carro.ano);  // 2022

carro.cor = "Prata"; // Adicionando dinamicamente uma nova propriedade
console.log(carro);
```

Manipulação de Array

Para darmos prosseguimento a nossa revisão e aumentar o nosso entendimento, vamos entender melhor como manipular arrays. Um array nada mais é que uma variável onde é possível armazenar vários valores, isso nos possibilita trabalhar com grandes quantidades de informações de um determinado tipo de forma mais simples, leve e performática.

```
let aluno1 = 'João'
let aluno2 = 'Carlos'
let aluno3 = 'Maria'
```

Usando variáveis simples para armazenar valores do mesmo tipo.

```
let aluno = ['joão','Carlos','Maria']
```

Usando array para armazenar valores do mesmo tipo.

Obs. Imagine se fossem dezenas ou centenas de valores....

Manipulação de Array

Neste array podemos guardar qualquer tipo de elemento, desde uma simples string, outros arrays ou até objetos.

```
let aluno = ['joão','Carlos','Maria']
```

Array de strings.

```
let grupos = [['Laura','Letícia'],['Pedro','Gustavo']]
```

Array de arrays.

```
let carros = [
    {'marca':'Honda','modelo':'Civic'},
    {'marca':'Toyota','modelo':'Corolla'},
    {'marca':'GM','modelo':'Cruze'}
]
```

Array de objetos.

Manipulação de Array PUSH( )

Para inserirmos um novo elemento a nosso array, podemos inserir alocar na próxima posição, ou pedirmos para que ele faça isso por nós utilizando o método push().

```
aluno[3] = 'Barbara'
console.log(aluno); // João, Carlos, Maria, Barbara
```

Adicionando uma nova posição ao array

```
aluno.push('Lucas')
console.log(aluno); // João, Carlos, Maria, Barbara, Lucas
```

Ultilizando o método push( ).

Manipulação de Array SORT( ), REVERSE() e POP( )

Podemos ordenar o conteúdo dos array, utilizando o método sort( ), perceba que agora está em ordem alfabética. E usando o reverse() invertemos a ordem.

```
console.log(aluno.sort()); // Barbara, Carlos, João, Lucas, Maria
console.log(aluno.sort().reverse()); // Maria, Lucas, João, Carlos, Barbara
```

Se precisarmos remover o último elemento, podemos usar o método pop( ), ao remover o último elemento, nós podemos guarda-lo em outra variável se quisermos.

```
alunoDesistente = aluno.pop()
console.log(aluno); // "Barbara", "Carlos", "Lucas", "Maria"
console.log(alunoDesistente); // João
```

Manipulação de Array UNSHIFT( ) e SHIFT( )

Podemos inserir um elemento na posição inicial do array com o método unshift( ).

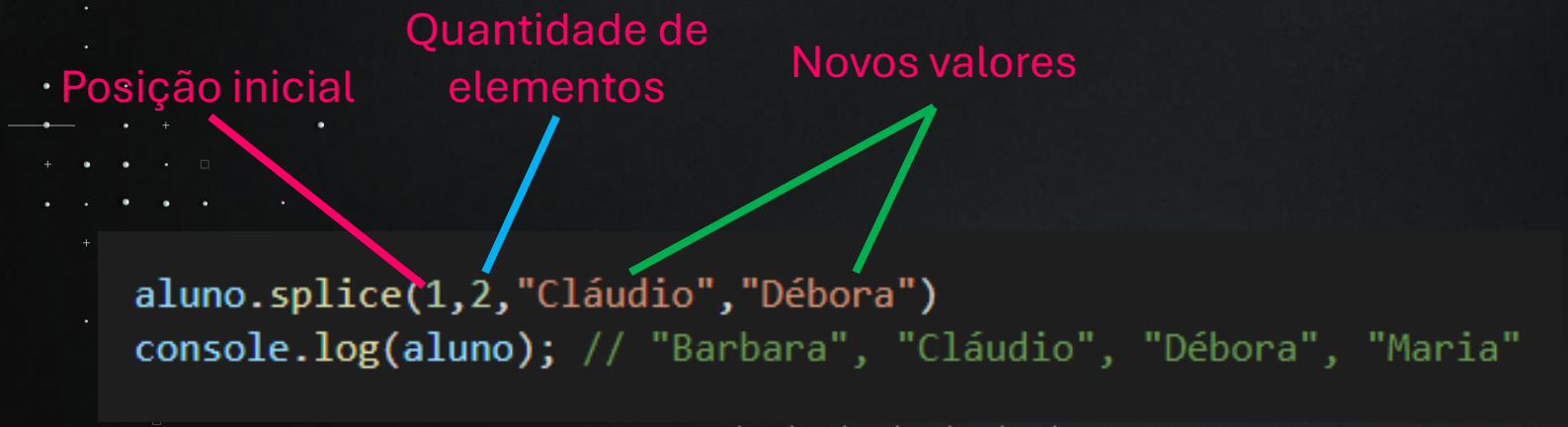
```
aluno.unshift('Igor')
console.log(aluno); // "Igor", "Barbara", "Carlos", "Lucas", "Maria"
```

E para remover o elemento da primeira posição usamos o método shift( ).

```
aluno.shift()
console.log(aluno); // "Barbara", "Carlos", "Lucas", "Maria"
```

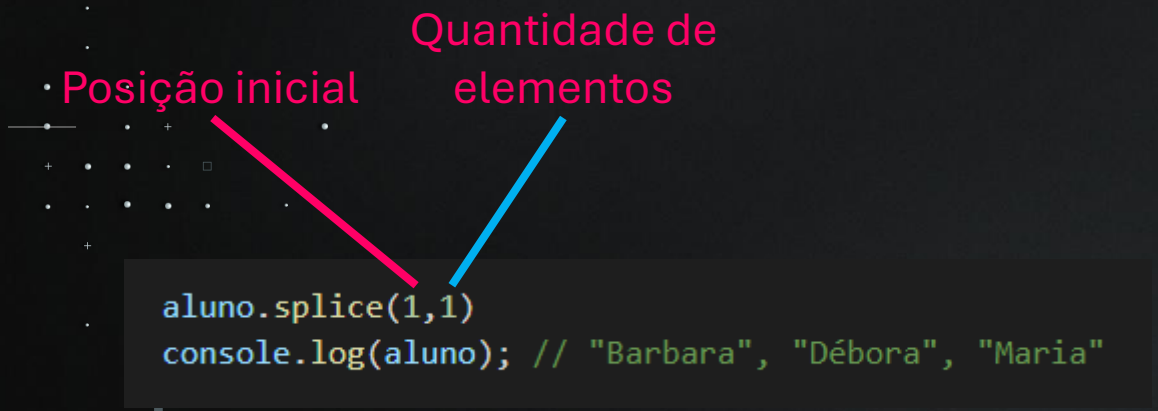
Manipulação de Array SPLICE( )

Se quisermos alterar ou remover um ou mais elementos de uma posição específica do array podemos utilizar o método splice( ).



Manipulação de Array SPLICE( )

Com splice também podemos apagar um ou mais itens do array.



Perceba que como não passamos os valores para substituir ele acaba apagando apenas

Exercício 1

45697056

Crie um projeto HTML/CSS/JS chamado ***exercicio-01-arrays***

Dentro desse projeto, crie uma página HTML nomeada index.html e adicione os seguintes elementos:

- Título da página com a identificação do aluno.
- Formulário contendo um input e um botão para adicionar novos nomes.
- Botão para ordenar os nomes em ordem alfabética.
- Botão para reverter a ordem dos nomes.
- Uma lista não ordenada (ul) para exibir os nomes inseridos no array, através do formulário (esta lista deve ser preenchida dinamicamente por um loop for).

É obrigatório o uso dos métodos de manipulação de array para adicionar, ordenar e reverter a ordem dos nomes. Utilize os seguintes métodos: PUSH, SORT, e REVERSE.

Exercício 2

45697056

Crie um projeto HTML/CSS/JS chamado ***exercicio-02-arrays***

Dentro desse projeto, crie uma página HTML nomeada index.html e adicione os seguintes elementos:

- Título da página com a identificação do aluno.
- Formulário contendo um input e um botão para adicionar novos nomes.
- Botão para ordenar os nomes em ordem alfabética.
- Botão para reverter a ordem dos nomes.
- Botão para remover o nome pesquisado.
- Uma lista não ordenada (ul) para exibir os nomes inseridos no array, através do formulário (esta lista deve ser preenchida dinamicamente por um loop for).

É obrigatório o uso dos métodos de manipulação de array para adicionar, ordenar e reverter a ordem dos nomes. Utilize os seguintes métodos: PUSH, SORT, REVERSE e SPLICE.

Manipulação de Array – método MAP( )

O método map permite criar um novo array a partir de um array já existente, podendo manipular seu conteúdo através de uma função de callback.

```
const cursos = [
    {'nome':'HTML5','duracao':'3 meses'},
    {'nome':'CSS3','duracao':'4 meses'},
    {'nome':'Javascript','duracao':'5 meses'}
]

console.log(cursos); //exibe todos os objetos do array

const nomeCursos = cursos.map(cursos => cursos.nome)

console.log(nomeCursos); // arrays apenas com os nomes dos cursos

const propgCursos = cursos.map(cursos => `O ${cursos.nome} só dura ${cursos.duracao}`)
//arrays manipulando o conteúdo
for(let cr in propgCursos) console.log(propgCursos[cr]);
```

Manipulação de Array – método MAP( )

No método map, a função de call-back também pode receber um segundo parâmetro, se é a posição do elemento no array, podendo ser usado para criar uma identificação única do elemento..

```
const indiceCursos = cursos.map((cursos,i) =>
    `O ${cursos.nome} deve ser o ${i+1}º a ser feito.`)

for(let i in indiceCursos) console.log(indiceCursos[i]);
```

Manipulação de Array – método FILTER( )

Se precisarmos criar um novo array a partir de um primeiro, mas somente com valores específicos podemos usar o método filter, que percorre o array fazendo a validação contida na função de callback.

```
const notas = [1,2,3,4,5,6,7,8,9,10]
console.log(notas); // 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
const notasAprov = notas.filter(item => item >= 6)
console.log(notasAprov); // 6, 7, 8, 9, 10

const pares = notas.filter(item => item %2 == 0)
console.log(pares); // 2, 4, 6, 8, 10
```

Manipulação de Array – método FILTER( )

Ainda conhecendo o filter podemos, por exemplo, fazer uma validação de login de uma forma bem descomplicada, comparando os valores do objeto.

```
let alunos = [
    {nome:'Luis', senha: 123},
    {nome:'Alexandre', senha: 456}
]

const logado = alunos.filter(aluno => aluno.nome === 'Luis' && aluno.senha === 123)

console.log(...logado)
```

Aqui estamos percorrendo o array e retornando apenas o objeto correto caso a condição seja satisfeita.

Manipulação de Array – método REDUCE( )

O método reduce é usado para reduzir um array a um único valor, aplicando uma função a um acumulador e cada valor do array (da esquerda para a direita) para reduzir o array a um único valor.

O método reduce executa uma função de call-back para cada interação da passagem pelo array retornando um único valor. Devemos inserir um argumento que será o responsável por acumular os valores e um segundo que representará o valor atual..

```
array.reduce(function(acumulador, valorAtual, índice, array) {
    // lógica de redução
}, valorInicial);
```

Manipulação de Array – variáveis REDUCE( )

- acumulador: O acumulador guarda o valor retornado na última invocação da função de callback, ou o valor inicial, se fornecido.

- valorAtual: O valor do elemento atual sendo processado no array.

- índice: O índice do elemento atual sendo processado no array. (opcional)

- array: O array sobre o qual o reduce foi chamado. (opcional)

- valorInicial: Um valor a ser usado como o primeiro argumento da primeira invocação da função de callback. (opcional)

Manipulação de Array – variáveis REDUCE( )

Exemplo Básico :

Vamos entender o reduce com um exemplo simples onde somamos todos os números em um array.

```
const numeros = [1, 2, 3, 4, 5];

const soma = numeros.reduce(function(acumulador, valorAtual) {
    return acumulador + valorAtual;
}, 0);

console.log(soma); // Saída: 15
```

Manipulação de Array – Contexto REDUCE( )

Passo a Passo:

Inicialização: O acumulador começa com o valor 0 (valorInicial).

Iteração 1: acumulador é 0 e valorAtual é 1. Soma: 0 + 1 = 1.

Iteração 2: acumulador é 1 e valorAtual é 2. Soma: 1 + 2 = 3.

Iteração 3: acumulador é 3 e valorAtual é 3. Soma: 3 + 3 = 6.

Iteração 4: acumulador é 6 e valorAtual é 4. Soma: 6 + 4 = 10.

Iteração 5: acumulador é 10 e valorAtual é 5. Soma: 10 + 5 = 15.

Resultado Final: O valor final do acumulador é 15.

Manipulação de Array – método EVERY( )

O método every testa se todos os elementos do array passam pelo teste implementado pela função fornecida, retornando assim um valor boleano.

```
const filaBrinquedo = [
    { ' nome': 'Sara' , 'altura' : 1.50},
    {'nome': 'Luciana', 'altura ' : 1.70},
    { ' nome': 'Kleber' , 'altura' :1.65},
    { ' nome' : 'Anderson', ' altura' :1.80}
];

const todaFilaPode = filaBrinquedo.every(pessoas => pessoas.altura >= 1.60);
console.log(todaFilaPode == true ? "Vamos lá" : "Nem todos podem");

//A saída será = Nem todos podem
```

Manipulação de Array – método SOME( )

O método some testa se ao menos um dos elementos do array passa no teste lógico, retornando um boleano.

```
const passeio = [
    {'nome':'Sara','idade':17},
    {'nome':'Luciana','idade':16},
    {'nome':'Kleber','idade':15},
    {'nome':'Anderson','idade':21}
]

const verifIdade = passeio.some(pessoa => pessoa.idade >= 18)
console.log(verifIdade);

                //A saída será = 21
```

Manipulação de Array – método FIND( )

O método find retorna o primeiro elemento do array que atender ao teste imposto pela função callback.

```
const candidatos = [
    {'nome':'Reinaldo','nota':65},
    {'nome':'Rita','nota':67},
    {'nome':'Sérgio','nota':78},
    {'nome':'Valter','nota':80}
]

const selecionado = candidatos.find( cand => cand.nota >= 70)
console.log(`${selecionado.nome} teve a nota ${selecionado.nota}!`);

//A saída será = Sérgio teve a nota 78!
```

Manipulação de Array – método INCLUDES( )

O método includes verifica se um array contém ou não um determinado elemento e retorna um boleano.

```
const convidados = ['prof Allen', 'Lucas', 'Gilberto','prof Luís','prof Alexandre']

const profConvid = convidados.filter( conv => conv.includes('prof'))
console.log(profConvid); // "prof Allen", "prof Luís", "prof Alexandre"
```

Repare que neste exemplo usamos o includes com o FILTER em uma string, que é um array de caracteres.

Exercício 3

45697056

Crie um projeto HTML/CSS/JS chamado ***exercicio-03-arrays***

Você deve criar um programa em JavaScript que manipula um array de salários de funcionários e apresente na página HTML.

1 – Crie um array contendo 10 valores de salário e utilizando o método map() atribua um aumento de 15% para salários até 2000 e um aumento de 10% para salários acima de 2000.

2 – Utilizando o array de resultado do exercício anterior, crie um novo array, usando o método filter(), contendo somente os salários superiores a 2500.

3 – Utilizando o array de resultado do exercício anterior, usando o método reduce(), some os valores.

- Título da página com a identificação do aluno.
- Uma lista não ordenada (ul) para exibir os salários com as devidas alterações.

É obrigatório o uso dos métodos de manipulação de array para o exercício que deve usar os métodos map, filter, reduce,find, opcionais, every, includes e some. Utilize de forma sequencial e integrada.

OBJETOS

45697056

Em JavaScript, objetos são estruturas de dados que permitem armazenar informações em pares de chave-valor. Eles são uma coleção de propriedades, onde cada propriedade tem um nome (chave) e um valor associado. Os valores podem ser de qualquer tipo de dado, como números, strings, booleanos, funções e até mesmo outros objetos.

```
//Exemplo criação de um objeto
    const pessoa = {
        nome: 'Luis',
        idade: 30,
        sexo: 'M',
        altura: 1.80,
        peso: 80,
        andar: function(){
            console.log('Andando...')
        }
    }
```

PROPRIEDADES

45697056

As propriedades dos objetos em JavaScript são pares de chave-valor que definem as características e valores associados a estes objetos. Essas propriedades permitem armazenar e acessar informações dentro do objeto. A chave é sempre uma string (ou símbolo) que identifica exclusivamente a propriedade, e o valor pode ser de qualquer tipo de dado válido em JavaScript, como números, strings, booleanos, funções ou até mesmo outros objetos. As propriedades podem ser adicionadas, modificadas ou removidas dinamicamente em tempo de execução.

PROPRIEDADES

45697056

```
//Acessando o atributo nome
console.log(pessoa.nome);

//Alterando o valor de um atributo
pessoa.nome = 'Alexandre';
console.log(pessoa.nome);

//Adicionando um atributo dirteto ao objeto
pessoa.cabelo = 'Castanho';
console.log(pessoa.cabelo);

//Adicionando um atributo ao objeto utilizando colchetes
pessoa['olhos'] = 'Castanhos';
console.log(pessoa['olhos']);

//Acessando um atributo utilizando colchetes
console.log(pessoa['nome']);

//Acessando um atributo utilizando colchetes e uma variável
let atributo = 'idade';
console.log(pessoa[atributo]);
```

PROPRIEDADES

45697056

```
//Deletando um atributo do objeto
delete pessoa.idade;
console.log(pessoa);

//Deletando um atributo utilizando colchetes
delete pessoa['sexo'];
console.log(pessoa);

//Deletando um atributo utilizando colchetes e uma variável
let atributo2 = 'altura';
delete pessoa[atributo2];
console.log(pessoa);
```

MÉTODOS

45697056

Métodos em objetos literais em JavaScript são funções que estão associadas às propriedades de um objeto. Eles permitem que os objetos realizem ações ou executem operações específicas relacionadas aos seus dados. Os métodos são definidos dentro do objeto como valores de propriedades, onde a chave é o nome do método e o valor é a própria função. Essas funções podem acessar e manipular os dados do objeto usando a palavra-chave "this", que se refere ao próprio objeto em que o método está sendo chamado. Os métodos podem ser chamados diretamente através do objeto e podem ter acesso a outras propriedades e métodos do mesmo objeto.

MÉTODOS

45697056

```
//Adicionando Metodo ao objeto
    pessoa.falar = function(){
        console.log('Falando...')
    }
    pessoa.falar();

//Executando a função andar
    pessoa.andar();

//Adicionando Metodo ao objeto utilizando colchetes
    pessoa['correr'] = function(){
    console.log('Correndo...')
    }

    pessoa.correr();

//Adicionando Metodo ao objeto utilizando colchetes e uma variável
    let metodo = 'pular';
    pessoa[metodo] = function(){
        console.log('Pulando...')
    }
    pessoa.pular();
```

MÉTODOS

```
//Acessando um metodo utilizando colchetes
    console.log(pessoa['falar']);

//Acessando um metodo utilizando colchetes e uma variável
    let metodo2 = 'correr';
    console.log(pessoa[metodo2]);

//Deletando um metodo do objeto
    delete pessoa.falar;
    console.log(pessoa);

//Deletando um metodo utilizando colchetes
    delete pessoa['correr'];
    console.log(pessoa);

//Deletando um metodo utilizando colchetes e uma variável
    let metodo3 = 'pular';
    delete pessoa[metodo3];
    console.log(pessoa);
```

MÉTODOS(FUNÇÕES)

```
//Exemplo de SPREAD
    const pessoa2 = {...pessoa, nome: 'Andréia', idade: 25, sexo: 'F'}
    console.log(pessoa2);

//Exemplo de REST
    const {nome, idade, ...resto} = pessoa2;
    console.log(nome, idade, resto);

//Exemplo de DESTRUCTURING
    const {nome: nome2, idade: idade2, ...resto2} = pessoa2;
    console.log(nome2, idade2, resto2);
```

Exercício 4

Crie um projeto HTML/CSS/JS chamado ***exercicio-04-arrays***

Vamos criar uma lista de tarefas. Nossa lista de tarefas será uma tabela e deve ser feita utilizando objetos com os seguintes atributos:

- ***Descrição***
- ***Autor***
- ***Departamento***
- ***Importância***

Nossa lista de tarefas deverá ter os seguintes controles:

- ***Inclusão de nova tarefa***
- ***Exclusão da tarefa concluída***
- ***Opção para adicionar o campo valor nos objetos das tarefas que serão pagas à parte***
- ***Opção para adicionar o campo duração nos objetos das tarefas que serão realizadas à parte***
- ***Opção para criação de uma lista das tarefas por ordem de importância contendo apenas a descrição***

DUVIDAS

OBRIGADO