FIAP

GRADUAÇÃO

45697056

# TDS
# FRONT-END DESIGN ENGINEERING

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

Prof. Luís Carlos lsilva@fiap.com.br

# Manipulação de Array

45697056

Para darmos procedimento e aumentar as possibilidades de uso do nosso código, 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

45697056

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

Array de strings.

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

Array de arrays.

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

Array de objetos.

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

# Manipulação de Array PUSH( )

45697056

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 POP( )

45697056

Já para deletar a última posição do nosso array, usamos o método pop().

```
aluno.pop()
console.log(aluno);   //João,Carlos,Maria,Barbara
```

Apagou a última posição do array “Lucas”

Podemos também apagar a última posição e reservar em outra variável.

```
let ultimoNome = aluno.pop()
console.log(ultimoNome); //Barbara
console.log(aluno); //João,Carlos,Maria
```

Apagou a última posição e reservou na variável “ultimoNome”

# Manipulação de Array SORT( ) e REVERSE( )

45697056

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

```
aluno.sort()
console.log(aluno) //Carlos,João,Maria
```

Se usarmos o método reverse(), logo após o sort(), ele deixa a ordem inversa.

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

# Manipulação de Array UNSHIFT( ) e SHIFT( )

45697056

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

```
aluno.unshift('Israel')
console.log(aluno); //Israel,Maria,João,Carlos
```

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

```
aluno.shift()
console.log(aluno) //Maria,João,Carlos
```

# Manipulação de Array SPLICE( )

45697056

Com o SPLICE( ) podemos inserir, substituir ou remover um ou mais elementos do array da posição que quisermos, ele é mais completo e por isso espera até 3 argumentos em seu construtor.

Vamos inserir dois novos nomes após o nome Maria:

Posição inicial

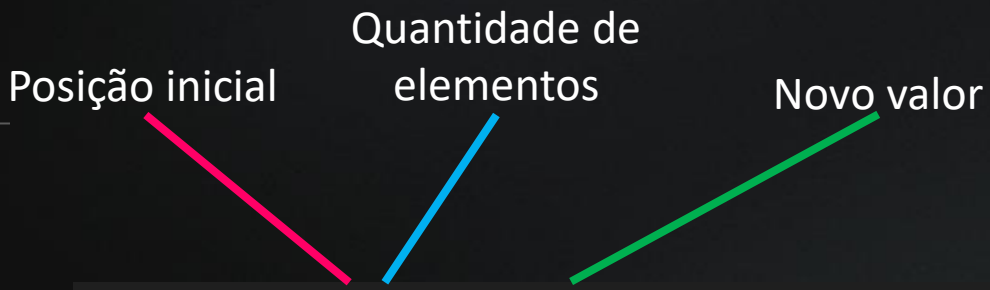
Quantidade de elementos

Novos valores

```
aluno.splice(1,0,'Junior','Julio')
console.log(aluno) //Carlos,Junior,Julio,João,Maria,Denis
```

# Manipulação de Array SPLICE( )

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

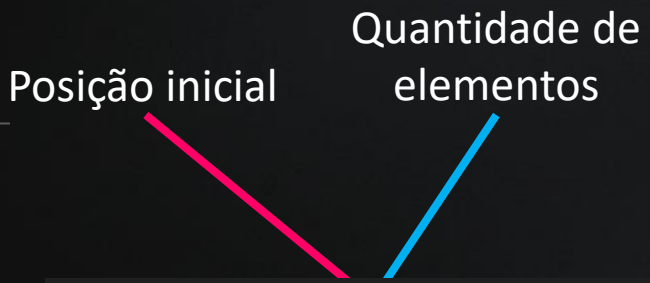



```
aluno.splice(2,1,'Juliana')
console.log(aluno) //Carlos,Junior,Juliana,João,Maria,Denis
```

Perceba que no segundo argumento passamos quantos itens deveriam ser substituídos, por isso ele saiu do array para a entrada no novo nome.

# Manipulação de Array SPLICE( )

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




```
aluno.splice(1,1)
console.log(aluno) //Carlos,Juliana,João,Maria,Denis
```

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

# Manipulação de Array – Método MAP( )

45697056

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( )

45697056

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( )

45697056

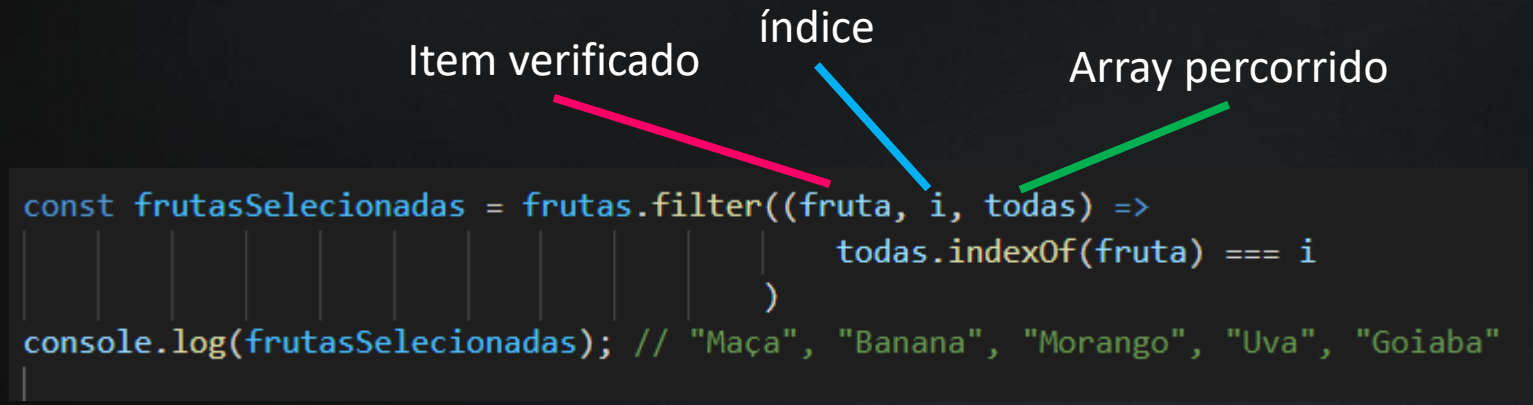
Se precisarmos criar um novo array a partir de um primeiro, mas somente com valores específicos podemos usar o método filter, que percorre o array fazendo a validação contida na função de callback.

```
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, ele pode receber 3 valores como parâmetro: o item do array, o índice do array e o próprio array.




Aqui estamos percorrendo o array, e pegando apenas a primeira fruta de cada tipo.

# Manipulação de Array – método REDUCE( )

45697056

O método reduce executa uma função de call-back para cada interação da passagem pelo array retornando um único valor. Na função de callback ele espera receber até 4 parâmetros: valor Anterior ou acumulador, valor atual, índice e o array percorrido. No exemplo abaixo só estaremos usando os dois primeiros. Devemos também passar o valor inicial.

```
const vendedores = [
    {'nome':'Janaina','vendas':5},
    {'nome':'Vitória','vendas':7},
    {'nome':'Marcelo','vendas':3},
    {'nome':'Henrique','vendas':9},
]

const totalVendas = vendedores.reduce((valorAnt,vendAtual)=> valorAnt + vendAtual.vendas,0)
console.log(totalVendas); // 24
```

Valor acumulado

Elemento atual

Valor inicial acumulado

# Manipulação de Array – método REDUCE( )

45697056

Vamos ver agora como é possível fazer o Reduce nos retornar vários resultados. Podemos ter vários resultados no valor inicial acumulado.

```
const vendedores = [
    {nome: 'Janaina', idade: 25, vendas: 5},
    {nome: 'Vitória', idade: 30, vendas: 7},
    {nome: 'Marcelo', idade: 35, vendas: 3},
    {nome: 'Henrique', idade: 40, vendas: 9}
]

const dadosVendas = vendedores.reduce(
    (acc, item) => {

        const maisNovo = acc.maisNovo < item.idade ? acc.maisNovo : item.idade
        const maisVelho = acc.maisVelho > item.idade ? acc.maisVelho : item.idade

        return {totalVendas: acc.totalVendas + item.vendas,
                maisNovo: maisNovo,
                maisVelho: maisVelho
        }
    },
    {totalVendas: 0, maisNovo: undefined, maisVelho: undefined}
)

console.log(dadosVendas);
```

Retorno dos valores que irão para o acumulado

Objeto de valores acumulados

# Manipulação de Array – método EVERY( )

45697056

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");
```

# Manipulação de Array – método SOME( )

45697056

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);
```

# Manipulação de Array – método FIND( )

45697056

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}!`);
```

# Manipulação de Array – método INCLUDES( )

45697056

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 em uma string, que é um array de caracteres.

# Manipulação de Array – Spread Operator

45697056

O Spread Operator é um a das opção mais usadas na nova versão do JS. Ela nos permite literalmente espalhar o conteúdo de um array ou objeto dentro de uma variável. Isso nos traz várias possibilidades

```
const frutas = ['maça','banana','uva']
const verduras = ['couve','alface','agrião']

const feira = [...frutas, ...verduras]

console.log(feira); //maça,banana,uva,couve,alface,agrião
```

Repare que neste exemplo unimos os dois primeiros arrays dentro do terceiro, sem o spread teríamos dois arrays fechados dentro do terceiro.

# Manipulação de Array – Spread Operator

45697056

Neste outro exemplo vamos mudar um valor de atributo do array usando também o spread

```
let pessoa = {
    nome: 'João',
    idade: 18,
    altura: 1.70
}

pessoa = {...pessoa, altura: 1.80}

console.log(pessoa); //{nome: 'João', idade: 18, altura: 1.8}
```

Também podemos inserir novos atributos e até unir atributos de dois ou mais objetos.

# Manipulação de Array – Rest Parameter

45697056

O Rest Parameter nos permite receber um número indefinido de parâmetros em uma função e nos retorna eles organizados em um array.

```
function someAll(...args){
    return args.reduce((acc, actual)=> acc += actual)

}

console.log(someAll(1,2,3));
```

Neste exemplo, nossa função pode somar quantos números quisermos.

**Obs.** Se houverem outros parâmetros, ele deve ser sempre o último.

# Manipulação de Array – Destructuring

45697056

O Destructuring nos permite desestruturar, ou seja, separar arrays e objetos, facilitando bastante quando não precisamos de todos os valores de uma só vez.

```
let alunos = ['Adriano', 'Bianca', 'Carolina']

let [aluno1,aluno2,aluno3] = alunos

console.log(aluno1); //Adriano
console.log(aluno2); //Biança
console.log(aluno3); //Carolina
```

Repare que criamos praticamente um array de variáveis para pegar os valores.

# Manipulação de Array – Destructuring

45697056

Podemos atribuir vários valores de uma só vez.

```
let car1, car2, car3, car4

[car1, car2, car3, car4] = ['Civic','Cruze','Corolla','Sentra']

console.log(car1);//Civic
console.log(car2);//Cruze
console.log(car3);//Corolla
console.log(car4);//Sentra
```

Se não tivermos o mesmo número dos dois lados, ou ele não aproveita o valor que sobrou, ou se for a variável, deixa ela como undefined.

# Manipulação de Array – Destructuring

45697056

Podemos também pré-atribuir valores à variável e ela só mantem se não receber o novo valor.

```
let car1, car2, car3, car4

[car1 = 'X1', car2 = 'A4', car3 = 'Corvetti', car4 = 'Mustang'] = ['Civic', ,'Corolla']

console.log(car1);//Civic
console.log(car2);//A4
console.log(car3);//Corolla
console.log(car4);//Mustang
```

Neste caso os carros 2 e 4 não foram atualizados porque não receberam novos valores.

# Manipulação de Array – Destructuring

45697056

Vamos ver agora como ele funciona com objetos, acaba sendo até mais simples, pois o objeto guarda atributos, que quase sempre já tem seus valores.

```
const filmes = {
    ficcao : 'Vingadores',
    terror: 'O Exorcista',
}

const {ficcao, terror} = filmes

console.log(ficcao);
console.log(terror);
```

Fica mais fácil de trabalhar com os valores em separado.

# Criando Elementos com JS – CreateElement

45697056

Para complementarmos nossos conhecimentos e fazer a próxima atividade, vamos ver um método que nos permite criar e inserir um elemento HTML na página. Primeiro devemos criar o elemento:

```
const paragrafo = document.createElement('p')
```

(Aqui estamos criando um parágrafo)

# Criando Elementos com JS – CreateElement

45697056

Agora precisamos inserir o conteúdo no elemento.

```
const texto = document.createTextNode('A frase do parágrafo!')
```

CreateTextNode cria o texto

```
paragrafo.appendChild(texto)
```

O appendChild insere o texto dentro do elemento

# Criando Elementos com JS – CreateElement

45697056

Por fim, vamos inserir o parágrafo na página.

```
document.querySelector('body').appendChild(paragrafo)
```

Usamos o appendChild para inserir no elemento já existente na página. No nosso caso como não tínhamos nenhum, colocamos no body.

Também podemos inserir atributos nele, usando o método setAttribute

```
paragrafo.setAttribute('id','paragr1')
```

Com o setAttribute podemos colocar um atributo e seu valor no elemento.

# Exercício

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.
- 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 ).

***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.***

# Exercícios

45697056

Agora que já sabemos manipular objetos, vamos melhorar a nossa lista de tarefas. Nossa nova lista 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