Nessa aula, o enfoque prático será em compreender as estruturas de dados já apresentadas de forma teórica e praticá-las para que o entendimento seja contemplado.

O Javascript não é assunto da nossa disciplina, dessa maneira você só precisa se atentar para o conceito de cada estrutura sem se preocupar com sintaxes da linguagem.

Todo o material a ser abordado aqui, está na documentação da Mozilla que mantém tudo o que precisamos saber sobre o JS: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array>

**Array**

Criação de um array vazio

O Array é uma estrutura de armazenamento de dados. Isso nós já sabemos. Também sabemos que um array tem uma dimensão previsível, ou seja, sabemos exatamente o tamanho que ele possui pois é igual à quantidade de elementos que ele possui. Então, vamos aprender a construir um array com o JS.

var frutas = []; // variável recebe um array vazio

No trecho de código acima foi criado um Array vazio. Você pode estar confuso achando que um array precisa sempre ter elementos dentro dele. Ok, vamos entender uma coisa. Nosso objetivo aqui é praticar, e para praticar, precisamos de uma linguagem de programação, e o JS foi a linguagem selecionada para o curso, mas poderia ser Java (e será, em breve), Phyton, Ruby, Php, etc … É particular da linguagem de programação a forma como ela irá lidar com as estruturas de dados. Para a nossa experiência aqui, saiba … um array pode ser criado sim vazio! E como medir a dimensão de um array no JS?

console.log(frutas.length) // retorna 0

Como criamos um array vazio anteriormente, a propriedade **length** que é muito comum nas linguagens de programação para medir o dimensionamento de um array, nos retorna a dimensão 0 (zero).

Criação de um array com a dimensão pré definida

Uma outra forma de criarmos um array seria informarmos diretamente a dimensão do array utilizando a propriedade *.length* que acabamos de aprender.

frutas.length = 10

Criação de um array com elementos pré definidos

A terceira maneira que iremos descobrir sobre como criar um array é informar diretamente os elementos que ele irá possuir.

var frutas = ["banana", "maça", "pêssego"];

console.log(frutas.length) // dimensão 3

Analisando a dimensão do array

Aprendemos a criar um array vazio especificando sua dimensão ou não. Precisamos então nos lembrarmos da teoria: Um array é estático. Possui tamanho bem definido. A quantidade de elementos é igual ao tamanho da dimensão do …. Pera aí. Então, se um array de 4 posições for criado e um elemento for inserido na sexta posição, dará erro? Veja:

O comando **push** insere um elemento ao fim do array, então:

var frutas = ["banana", "maça", "pêssego"];

console.log(frutas.length)

A dimensão do array *frutas* criado acima é 3, pois temos exatamente 3 elementos na estrutura. A dimensão desse array inicia na posição 0 e termina na posição 2. Sendo assim, 0 a primeira posição, 1 a segunda e 2 a terceira posição. Veja a saída do comando a seguir:

console.log(frutas[0])

Como o zero é o lugar do primeiro elemento, *“banana”* será impresso nesse comando. Sempre olhando da esquerda para a direita, pois é a forma *default* de se ver um array. Lembra que um array é uma lista, e qual a ordem de se ver uma lista? De cima (início) para baixo (fim). No Array, é da esquerda para a direita.

Retomando o raciocínio da indagação que nos trouxe até aqui: É possível adicionar um elemento fora das dimensões de um array? A resposta para essa pergunta é: SIM !

Vamos adicionar mais uma fruta nessa nossa lista:

frutas.push('cajá');

console.log(frutas.length) // dimensão 4

Perceba que agora a dimensão mudou de 3, para 4 pois inserimos mais 1 fruta na lista. Ao inserirmos essa nova fruta, a posição na sequência foi utilizada por *default.* Mas, observe o código abaixo:

frutas[5] = 'manga'; // Inserindo “Manga” na posição 5

console.log(frutas)

Lembre-se: Dimensão não é a mesma coisa que posição. Não esqueça disso. Nosso array tem tinha 4 elementos e o último elemento da estrutura (até então o cajá) estava na posição 3 e não na 4, pois a 4 não existia ainda. Entretanto, com o código acima que inseriu “manga” na posição 5, perceba que a posição 4 foi pulada, e como um array é estático e sua dimensão precisa acompanhar a quantidade de elementos dentro da estrutura, a dimensão desse array precisou ser ajustada e a posição 4 passou a existir, só que sem elemento algum.

["banana","maça","pêssego","Cajá",**undefined**,"manga"]

Um *“Undefined”* precisou ocupar o lugar da posição 4 para que *“manga”* pudesse existir na estrutura. Temos agora um array com 5 elementos, mas dimensão 6.

console.log(frutas[5]) // Imprime “manga”

console.log(frutas.length) // Imprime 6

Para diminuir então a dimensão do array, observamos aqui outra novidade:

var times = ["Flamengo", "Corintias", "Barcelona", "Real Madrid"];

times.length = 1

console.log(times) // Imprime "Flamengo"

Quando diminuímos a dimensão de um array, as posições que estariam além da nova dimensão são apagados automaticamente.

O primeiro e o último elemento de um array

Já aprendemos a ler um array como lemos uma lista, afinal, um array é uma lista. Sabemos que é da esquerda para direita (a forma *default* de leitura de um array). Então, podemos dizer que o elemento da posição zero é o primeiro elemento (se a posição zero for *undefined*, esse valor será retornado como primeiro elemento mesmo não sendo nenhum elemento definido). Então, acompanhe o raciocínio com o trecho de código abaixo:

var times = ["Flamengo", "Corintias", "Barcelona", "Real Madrid"];

var primeiroElemento = times[0]; // Flamengo

var ultimoElemento = times[times.length - 1]; // Real Madrid

A única novidade é que length - 1 quer dizer que a dimensão do array - 1 retorna o último elemento do array.

Retornar somente os índices que possuem elementos definidos

Sabemos que algumas posições podem receber valores não definidos: *undefined* então, se quisermos saber quais as posições que possuem elementos definidos, o *Object.Keys* informa para nós.

console.log(Object.keys(array));

Manipulação de dados em um array

Um array NÃO É uma Fila ou uma Pilha. Há quem diga que um array é uma estrutura de dados concreta e essas outras estruturas sejam estruturas de dados abstrata, pois o Array de fato existe como estrutura de armazenamento e, como bem vimos na teoria, nada se refere a uma Fila ou Pilha como uma estrutura, apenas como um modelo, uma técnica utilizada para manipulação de elementos em uma estrutura de dados. o JS disponibiliza as operações de Fila e Pilha para serem utilizadas em um Array. Para isso, basta utilizar as propriedades adequadas ao funcionamento de cada estrutura.

Pois bem, iremos dar prosseguimento ao aprendizado de Estrutura de Dados introduzindo o conceito básico sobre Filas e Pilhas.

**FILA** (Primeiro a entrar é o primeiro a sair)

Seguindo a teoria que aprendemos na apostila, podemos ver abaixo que uma fila de nome *queue* foi criada como um array vazio e então utilizou a propriedade *push* para incluir os elementos dessa fila. Foram criados 4 elementos que ocuparam posições na estrutura na ordem em que foram criados. Com isso *“João”* ocupou a primeira posição (zero) e *“Luiza”* a última posição (3). Como já sabemos sobre o funcionamento de uma fila, sabemos que o primeiro a ser retirado da estrutura será *“João”* e o último *“Luiza”*.

var queue = []

queue.push("João")

queue.push("José")

queue.push("Maria")

queue.push("Luzia")

console.log(queue) // O array com os 4 elementos

console.log(queue[0]) // Para constatar qual é o primeiro elemento da pilha

console.log(queue.shift()) // Remove e retorna o primeiro elemento da fila

console.log(queue) // A pilha sem o elemento removido

**PILHA** (Último a entrar primeiro a sair)

Nessa estrutura é muito importante salientar o seguinte: “Último a entrar e primeiro a sair” significa que, como em uma lista a gente sabe como ler seus elementos, e uma Pilha é uma lista, a gente sabe qual é o primeiro elemento e qual o último. Por isso, não podemos achar que o último elemento é também o primeiro elemento. O último elemento será retirado primeiro, mas ele permanece sendo o último da estrutura.

var pilha = [];

pilha.push("Prato 1");

pilha.push("Prato 2");

pilha.push("Prato 3");

console.log(pilha[0]); // O primeiro elemento da pilha continua sendo o que primeiro entrou

pilha.pop(); // O elemento que é removido da pilha é sempre o último que entrou

console.log(pilha) // Retorna a pilha sem seu último elemento