

The background features abstract, organic shapes in red, blue, and yellow against a black space-like backdrop with small white dots representing stars.

JavaScript

Arrays



JS

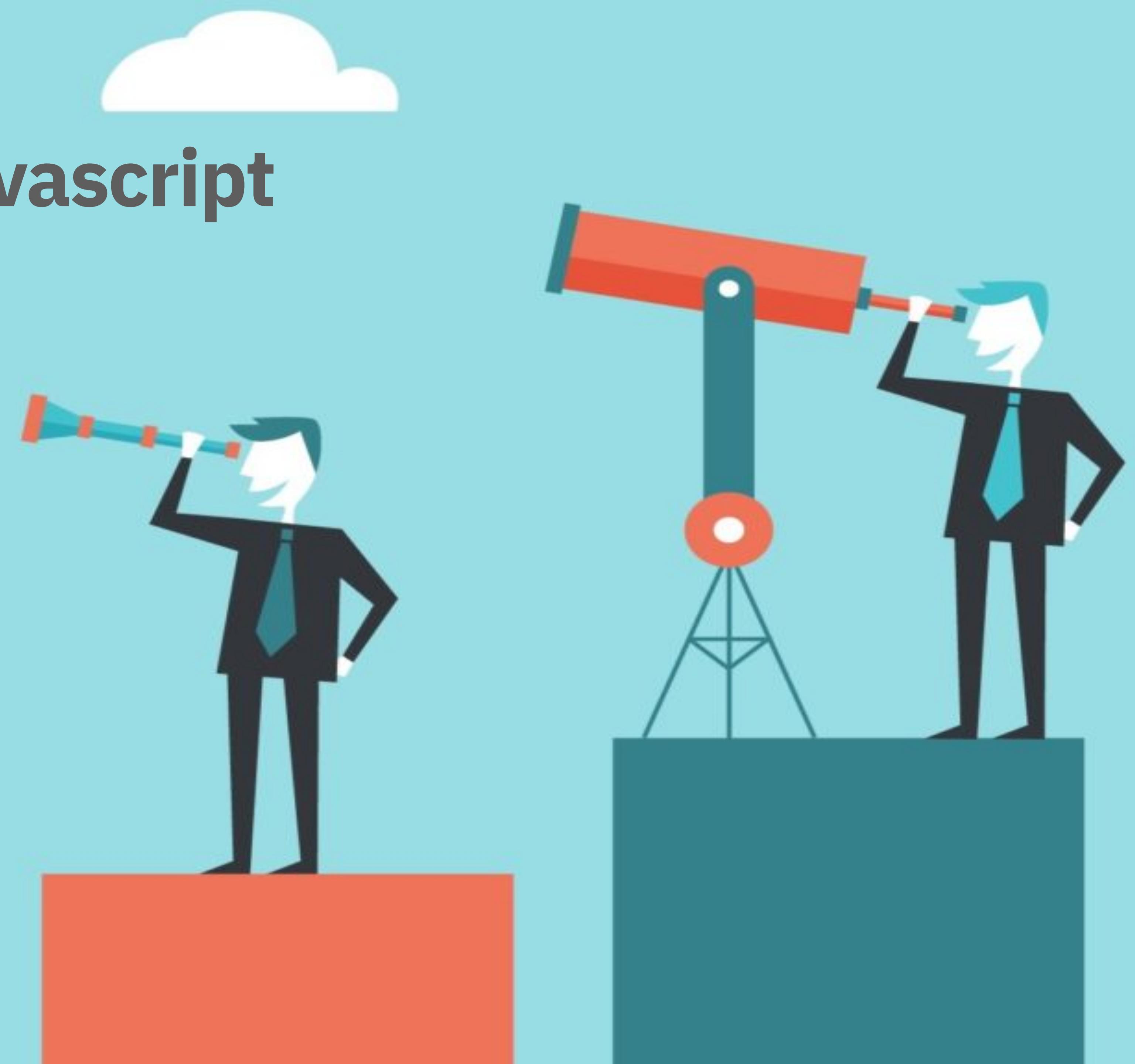
JavaScript

parte 9

aprendendo Javascript

agenda

os **arrays** são estruturas de dados que permitem trabalhar com vários elementos em uma única variável. Eles são amplamente utilizados com JavaScript e trazem consigo um conjunto não menos interessante de métodos que bem usados podem trazer muita aceleração aos seus programas.





disclaimer

arrays não são tipos
primitivos, mas
objetos JavaScript.

o que são arrays

estruturas de dados

até agora aprendemos que uma variável pode conter um tipo primitivo em especial. Ainda que ela possa mudar de tipo sem representar um sério problema a minha lógica de programação, ela só pode conter um valor por vez.

os *arrays* são um tipo especial, de fato um objeto javascript que permite em uma única variável termos uma complexa estrutura de dados com tamanho limitado a memória do computador e de quaisquer tipos desejados, inclusive um array pode conter vários outros arrays dentro dele mesmo, tornando a estrutura multidimensional.

um array é declarado basicamente inserindo os seus valores entre colchetes como apresentado a seguir.



```
let pets = ['Stella', 'Egídio', 'Luly'];
```

ou ainda pode ser declarado completamente vazio, sem nenhuma ocorrência de valores.



```
let devs = [];
```

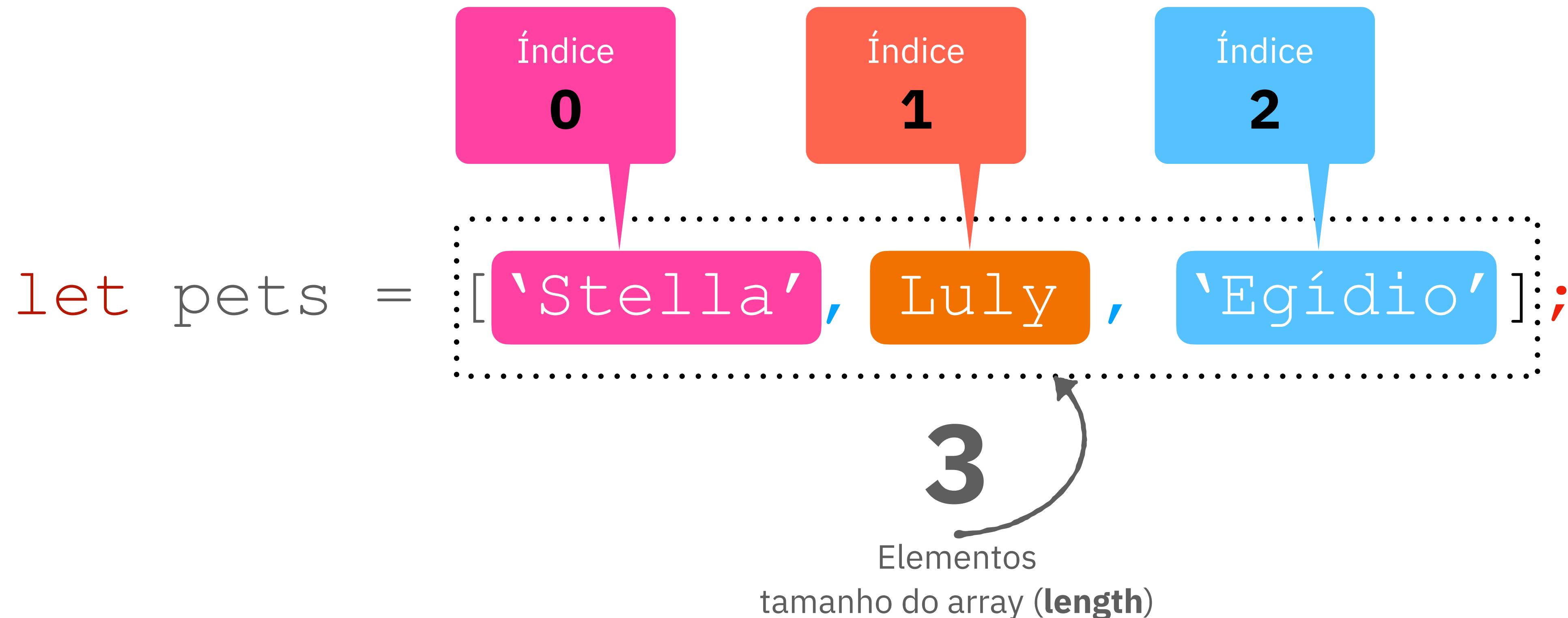
e ainda conter tipos diferentes.



```
let familia = ['Alan', 53, true, [ 'Diane', 'Laurinha'], [ 'Stella' ]];
```

estrutura do array

índice do array



como acessar um array ?

como ele é interpretado pelo JS



```
// criando um array explicitamente.  
const pets = new Array('Stella', 'Luly', 'Egídio', 'Thor');  
// acessando a primeira ocorrência do array  
console.log(pets[0]); // Stella  
// acessando a terceira posição do array  
console.log(pets[2]); // Egídio  
// acessando uma posição inexistente no array  
console.log(pets[10]); // undefined
```

você pode acessar qualquer um dos elementos do array usando o índice. Lembrando que a primeira posição de um array é a **0** (zero). E quando acidentalmente você tentar acessar uma posição que não existe, vai retornar um *undefined*.

qual o tamanho de um array ?

total de elementos do array



```
// criando um array implicitamente.  
const pets = ['Stella', 'Luly', 'Egídio', 'Thor'];  
// quantos elementos tem nosso array?  
console.log(pets.length); // 4
```

acessando a propriedade `length` do array vai retornar a quantidade de elementos atual. É especialmente útil quando formos iterar com os elementos, sejam usando as instruções de repetições como `for`, `while`, `do` como `forEach`.

acrescentando elemento no final do array

JavaScript tem método para isto **push**

adicionar novo elemento no **final** de um array é uma ação bem simples. Basta chamar seu array e consumir um método público do objeto global array chamado **push**.

```
● ● ●  
  
// criando um array implicitamente.  
const pets = ['Stella', 'Luly', 'Egídio'];  
// quantos elementos tem nosso array?  
console.log(pets.length, pets);  
// adicionando um elemento ao final com push  
pets.push('Thor');  
// quantos elementos tem nosso array agora?  
console.log(pets.length, pets);
```

ao exibir o array **pets**, vai retornar cada um dos elementos ordenados pelo seu **index**.

```
>_ Console (beta) ⓘ 2 ⓘ 0 ⚠ 0 ⓘ 0 Clear console Minimize  
  
Cloud "Running fiddle"  
  
3, ["Stella", "Luly", "Egídio"]  
4, ["Stella", "Luly", "Egídio", "Thor"]  
  
>-
```

acrescentando elemento no inicio do array

JavaScript tem método para isto **unshift**

adicionar novo elemento no **início** de um array também é uma ação bem simples. Basta chamar seu array e consumir um método público do objeto global array chamado **unshift**.

```
● ● ●  
  
// criando um array implicitamente.  
const pets = ['Stella', 'Luly'];  
// quantos elementos tem nosso array?  
console.log(pets.length, pets);  
// adicionando um elemento ao final com push  
pets.push('Egídio', 'Thor');  
// quantos elementos tem nosso array agora?  
console.log(pets.length, pets);  
// adicionando um elemento ao inicio com unshift  
pets.unshift('Lord');  
// quantos elementos tem nosso array agora?  
console.log(pets.length, pets);
```

ao exibir o array **pets**, vai retornar cada um dos elementos ordenados pelo seu **index** sendo que o **unshift** vai empurrar todos para uma posição posterior.

```
>_ Console (beta) ⓘ 3 ⓘ 0 ⓘ 0 ⓘ 0 ⓘ 0 Clear console Minimize  
Cloud "Running fiddle"  
2, ["Stella", "Luly"]  
4, ["Stella", "Luly", "Egídio", "Thor"]  
5, ["Lord", "Stella", "Luly", "Egídio", "Thor"]  
>
```

removendo elemento no final do array

JavaScript tem método para isto

pop

remover um elemento no **final** de um array é uma ação que o Javascript também implementa. Basta chamar seu array e consumir um método público do objeto global array chamado `pop`.



```
// criando um array implicitamente.  
const pets = ['Stella', 'Egídio', 'Luly'];  
// quantos elementos tem nosso array?  
console.log(pets.length, pets);  
// eliminando o último elemento do array  
pets.pop();  
// quantos elementos tem nosso array agora?  
console.log(pets.length, pets);
```

ao exibir o array `pets`, vamos perceber que o último elemento do array que continha Luly, agora não faz parte do array.

The screenshot shows a browser's developer tools console. At the top, there are three colored dots (red, yellow, green). Below them, the code from the previous block is pasted. The console output shows:

```
>_ Console (beta) ① 2 ① 0 ④ 0 ① 0 Clear console Minimize  
Cloud "Running fiddle"  
3, ["Stella", "Egídio", "Luly"]  
2, ["Stella", "Egídio"]  
>_
```

The first line shows the state before `pop()` is called. The second line shows the state after `pop()` has been called, and the last line is a prompt for the next input.

removendo elemento no início do array

JavaScript tem método para isto

shift

remover um elemento no **início** de um array é uma ação que o Javascript também implementa. Basta chamar seu array e consumir um método público do objeto global array chamado shift.



```
// criando um array implicitamente.  
const pets = ['Egídio', 'Stella', 'Thor', 'Luly'];  
// quantos elementos tem nosso array?  
console.log(pets.length, pets);  
// eliminando o primeiro elemento do array  
pets.shift();  
// quantos elementos tem nosso array agora?  
console.log(pets.length, pets);
```

ao exibir o array pets, vamos perceber que o primeiro elemento do array que continha Egídio, agora não faz parte do array.

The screenshot shows a browser's developer tools console. At the top, there are three colored dots (red, yellow, green). Below them is the code from the previous block. The console output shows:

- A message indicating "Running fiddle".
- The result of `console.log(pets.length, pets)` which is 4, followed by the array `["Egídio", "Stella", "Thor", "Luly"]`.
- The result of `console.log(pets.length, pets)` after `shift()` is called, which is 3, followed by the array `["Stella", "Thor", "Luly"]`.
- An empty line starting with '>_'.
The status bar at the top of the console shows: >_ Console (beta) ⓘ 2 ⓘ 0 ⓘ 0 ⓘ 0 ⓘ 0 Clear console Minimize

básico do básico nos arrays

manipulação simples

`push`

insere um novo elemento no final do array.

`unshift`

insere um novo elemento no início do array.

`pop`

excluiu um elemento no final do array.

`shift`

excluiu um elemento no início do array.

JavaScript sendo JavaScript

o que seria do JS se não fosse diferente

há um comportamento perigoso no JS que permite expandir um array sem o uso dos métodos `push` e `unshift` mas não é recomendado você usar este formato nos seus códigos. Vejamos

```
● ● ●  
  
const pets = [ 'Stella', 'Egídio', 'Luly' ];  
console.log(pets.length, pets);  
// Criando um elemento na posição 10  
pets[10] = 'Thor';  
console.log(pets.length, pets);
```

simplesmente acessando um índice maior que o tamanho do array, o JS vai automaticamente expandi-lo até aquele valor, associando o dado e criando índices sem valores entre o tamanho anterior até o novo tamanho. Cuidado com isto.

ao exibir o array `pets` agora depois de ir até o índice 10 quando tinha apenas 3, nosso array criou uma série de novos índices com valores vazios.

```
> console.log(pets);  
  
▼ (11) ["Stella", "Egídio", "Luly", empty × 7, "Thor"] VM330:1  
  0: "Stella"  
  1: "Egídio"  
  2: "Luly"  
  10: "Thor"  
  length: 11
```



na saída da console você pode observar que ele criou uma faixa de 7 novos índices porém todos vazios.

acessando um valor do meu array

todo o acesso é feito pelo índice

digamos que agora que eu tenho um array, eu quero acessar cada uma das posições, exibir seu conteúdo e alterar quando necessário. Como posso fazer isto de uma maneira fácil e ágil ?



```
const pets = ['Egídio', 'Stella', 'Thor', 'Luly'];
console.log(pets[1]); // Stella
```

basta acessa o array através do seu índice desejado que o valor contido naquela posição será exibida na saída padrão do **console.log** bem como nós podemos alterar o conteúdo desta posição apenas associando a ele um novo valor. Acredite, isto é possível mesmo com uma constante, mas é assunto para mais adiante.



```
const pets = ['Egídio', 'Stella', 'Thor'];
console.log(pets[1]); // Stella
pets[1] = 'Mel';
console.log(pets[1]); // Mel
```

no exemplo acima nós criamos o array contendo **Stella** na posição 2 (índice 1), depois de exibirmos nós acessamos diretamente este índice e associamos o conteúdo **Mel**. Com isto perdemos a informação anterior e fizemos literalmente uma alteração no conteúdo do índice.

Se você acessar um índice que é maior que o tamanho do array já sabe que ele irá automaticamente criar valores vazios até aquela posição - 1.

iterando com array

usando o modelo tradicional com `for` e `while`

podemos percorrer todo os dados do array usando uma simples estrutura de repetição com `for`. Vejamos.



```
const pets = ['Stella', 'Egídio', 'Luly'];
console.log('Meus pets\n-----');
for (let i = 0; i < pets.length; i++) {
    console.log(pets[i]);
}
```

Meus pets

Stella
Egídio
Luly



```
const pets = ['Stella', 'Egídio', 'Luly'];
let i = 0;
console.log('Meus pets\n-----');
while (i < pets.length) {
    console.log(pets[i]);
    i++;
}
```

Meus pets

Stella
Egídio
Luly

iterando com array

usando o modelo tradicional com `for of` e `forEach`

podemos percorrer todo os dados do array usando uma simples estrutura de repetição com `for of`. Vejamos.



```
const pets = ['Stella', 'Egídio', 'Luly'];
console.log('Meus pets\n-----');
for (const pet of pets) {
  console.log(pet);
}
```

Meus pets

Stella
Egídio
Luly



```
const pets = ['Stella', 'Egídio', 'Luly'];
console.log('Meus pets\n-----');
pets.forEach(pet => {
  console.log(pet);
});
```

Meus pets

Stella
Egídio
Luly

DANGER

HIGH
VOLTAGE



perigo

não altere o valor da propriedade `length`

se você alterar o valor da propriedade `length` do array poderá criar um comportamento indesejado no seu código.



```
const pets = ['Stella', 'Egídio', 'Luly'];
console.log(`Array de ${pets.length} posições.`);
console.log('Meus pets\n-----\n', pets);
pets.length = 6; // NÃO FAÇA ISTO.
console.log(`Agora o Array de ${pets.length} posições.`);
console.log('Meus pets\n-----\n', pets);
```

Array de 3 posições.

Meus pets

► (3) ["Stella", "Egídio", "Luly"]

Agora o Array tem 6 posições.

Meus pets

► (6) ["Stella", "Egídio", "Luly", empty × 3]

[index-01.html:15](#)

[index-01.html:16](#)

[index-01.html:19](#)

[index-01.html:20](#)

neste segundo exemplo, nós acessamos `length` e reduzimos o valor do array para um tamanho menor do atual e isto resultará em perda de dados.



```
const pets = ['Stella', 'Egídio', 'Luly', 'Thor'];
console.log(`Array de ${pets.length} posições.`);
console.log('Meus pets\n-----\n', pets);
pets.length = 2; // NÃO FAÇA ISTO.
console.log(`Agora o Array de ${pets.length} posições.`);
console.log('Meus pets\n-----\n', pets);
```

Array de 4 posições.

Meus pets

► (4) ["Stella", "Egídio", "Luly", "Thor"]

Agora o Array tem 2 posições.

Meus pets

► (2) ["Stella", "Egídio"]

[index-01.html:15](#)

[index-01.html:16](#)

[index-01.html:19](#)

[index-01.html:20](#)



VAMOS PRATICAR

revendo a prática

princípio básico

1. aprendemos que os tipos **arrays** são **objetos** e não tipos primitivos em JavaScript.
2. é possível dar valores para posições limitado a capacidade de **memória**, e que estes valores podem inclusive serem de tipos primitivos diferentes.
3. há **métodos** globais de arrays para usarmos nos programas.
4. arrays são estruturas **poderosas**.
5. vimos o perigo que é acessar diretamente a propriedade **length** dos arrays e como isto pode gerar um comportamento errôneo.

