Gabriel Guimarães Antunes

Desenvolvedor desde 2010
Fullstack focado em front-end
Formado em Processamento de dados

Linguagens atuais

JavaScript Moderno (Vuejs e Jquery)

CSS 3.0 (SASS)

PHP (Laravel)

Banco de dados (Oracle e PostgreSQL)

Já desenvolvi...

JavaScript (Angular 1.5)

C# - PLSQL - ColdFusion

O que veremos?

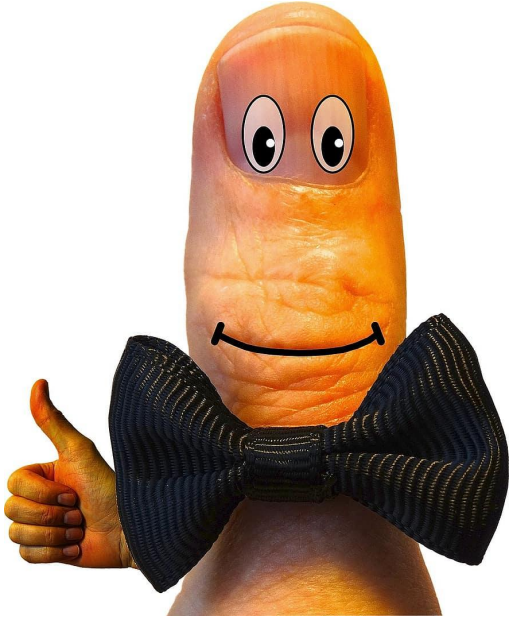
Conceito de linguagem
Tipos de desenvolvimento (desktop, web, mobile)
Algoritmo
Variáveis (int, string, bool...)
Estruturas condicionais (if, else if, else)
Operadores aritméticos, relacionais e lógicos
Estruturas de repetição (for, while...)
Contadores (loop infinito)
Funções e métodos (parâmetros)
Array
O que é Programação orientada a objetos
IDEs mais utilizadas

Desenvolvimento web
- Browser (F12)
- Protocolo HTTP
- Requisições e Respostas (AJAX)
- HTML
- JavaScript (Angular js)
- CSS
- JSON
- Jquery

Bootstrap
Noções de GIT (versionamento)
Boas práticas de código
Performance (Cache, Gulp, Assíncrono)
Banco de dados (SQL)

Antes de começar...

Deixem aberta a página do slack, por lá que eu mandarei links e materiais durantes as aulas!



Vamos interagir!

Vou fazer algumas perguntas para saber qual o nível de conhecimento da classe para poder direcionar melhor o conteúdo programático ;)

Respondam para mim no chat do Zoom:

1 para sim

0 para não

Conceito de linguagem de programação

É uma linguagem formal que, através de uma série de instruções, permite que um programador escreva um conjunto de ordens, ações e algoritmos para criar programas que controlam o comportamento físico (automação industrial) ou lógico de uma máquina.

Nós nos comunicamos com a máquina através dessa linguagem, permitindo especificar:

- quais dados um software deve operar;
- como esses dados devem ser armazenados ou transmitidos;
- quais ações o software deve executar, de acordo com cada circunstância.

Tipos de desenvolvimento

Desktop 

Web 

Mobile 

Desktop

Uma aplicação desktop é qualquer software que pode ser instalado em um computador e usado para executar tarefas específicas. Algumas aplicações desktop também podem ser usadas por vários usuários em um ambiente com rede.

Exemplos: Word, Excel, etc (isso antes de virarem aplicações web)

Desktop

Exemplo de linguagens usadas para aplicações desktop:

C++

Visual Basic

Delphi

Web

Aplicações web usam um navegador web como Mozilla Firefox, Edge, Google Chrome, dentre outros, como interface. Todo computador que possui esses navegadores e internet pode acessar a mesma aplicação de qualquer sistema operacional (Linux, Windows, MaCOS), o que não ocorre com aplicações Desktop, que normalmente funcionam em apenas um ou outro sistema operacional.

Web

Exemplo de linguagens usadas para aplicações web:

PHP

.NET

JAVA

Mobile

Aplicações Mobile são aquelas que rodam em Smartphone (celular com internet). A cada dia vemos mais e mais apps que nos ajudam a fazer tudo sem sair de casa ou até mesmo depender de um computador.

Hoje podemos fazer uma compra online, pagar contas, realizar transferências bancárias, pedir um lanche ou marcar aquele rolê de solteiro...

Mobile

Exemplo de linguagens usadas para aplicações Mobile:

Android

IOS (Apple)

React Native (baseado em React, um framework javascript), desenvolvido pela equipe do Facebook, que possibilita o desenvolvimento de aplicações mobile, tanto para Android, como para iOS, utilizando apenas JavaScript.

Algoritmo

Conjunto das regras e procedimentos lógicos definidos que levam à solução de um problema dividido em etapas finitas. Em outras palavras:

Algoritmo é simplesmente uma receita para executarmos uma tarefa ou resolver algum problema.

Exemplo de algoritmo

TAREFA CASEIRA:

Fazer uma macarronada

Fritar um ovo 

Lavar roupa

ROTINA DE TRABALHO:

Cadastrar um usuário

Abrir um chamado

Pagar um boleto

05:00

Quantos passos você realizou?

O algoritmo pode ter muitos ou poucos passos.

Vamos ver se ficaram parecidos?

Algoritmo - Frite um ovo

1- Pegue uma frigideira
2- Coloque o oleo
3- Acenda o fogo
4- Pegue o ovo
5- Quebre o ovo na frigideira
6- Coloque sal a gosto
7- Retire o ovo frito
8- Coma =)

Variáveis

Uma variável é um espaço na memória do computador destinado a um dado que é alterado durante a execução do algoritmo. Para funcionar corretamente, as variáveis precisam ser definidas por nomes e tipos (dependendo da linguagem de programação).

Veja os tipos de dados mais utilizados:

TypeScript

Vamos utilizar a linguagem TypeScript para os exemplos a seguir.

Exemplo de estrutura:

nomeDaVariavel: tipo = valor;

String

Para armazenar uma frase ou um texto qualquer.

Exemplo: a variável nome será do tipo string e receberá o valor Roberto.

nome: string = “Roberto”;

Obs: strings sempre ficam entre aspas!

Boolean

Para armazenar um valor Verdadeiro ou Falso.

Exemplo: a variável adulto será do tipo boolean e receberá o valor true.

adulto: boolean = true;

Obs: em vez de utilizar true ou false, podemos utilizar os números 1 e 0 que também representam um valor booleano

adulto = true seria o mesmo que **adulto = 1**

Number

Para armazenar um número.

Exemplo: a variável idade será do tipo number e receberá o valor 20.

idade: number = 20;

Array

Para armazenar diversos itens, seja string, number, boolean, objeto entre outros.

Exemplo: a variável cores será do tipo array e receberá os valores verde, amarelo e azul.

cores: array = [“verde”, “amarelo”, “azul”];

Não se preocupem, falaremos muito mais sobre array nas próximas aulas!

Outros tipos

Podemos ver outros tipos de variáveis em outras linguagens, digamos que são variações dos tipos primitivos vistos até agora, por exemplo:

float: é um Number, mas pode conter casas decimais => 10.5
int: inteiro, ou seja, sem casas decimais => 10
char: uma string com um único caractere => “A”

O que são variaveis?

Labenu_

Estruturas condicionais!

Condicional Simples if (se)

SE (a condição for verdadeira)

    ENTÃO {executar algum comando}

FIM-SE

Algoritmo - Frite um ovo

1- Pegue uma frigideira
2- Coloque o oleo
3- Acenda o fogo
4- Pegue o ovo
SE (o ovo estiver estragado)
    ENTÃO {Jogue fora e pegue outro}
FIM-SE
5- Quebre o ovo na frigideira
6- Coloque sal a gosto
7- Retire o ovo frito
8- Coma =)

Exemplo:

Idade: number = 18;

SE (idade >= 18)

    ENTÃO { imprima a mensagem “você é um adulto” }

FIM-SE

Exemplo com JavaScript

```
var idade = 18;
If (idade >= 18) {
        alert(“você é um adulto”);
}
```

Bora ver na prática??

Acesse essa página https://codepen.io/

To mandando o link no grupo material-didático do slack

CODEPEN

Search CodePen...

Sign Up

Log In

TRY OUR ONLINE EDITOR

Start Coding

Trending

Challenges

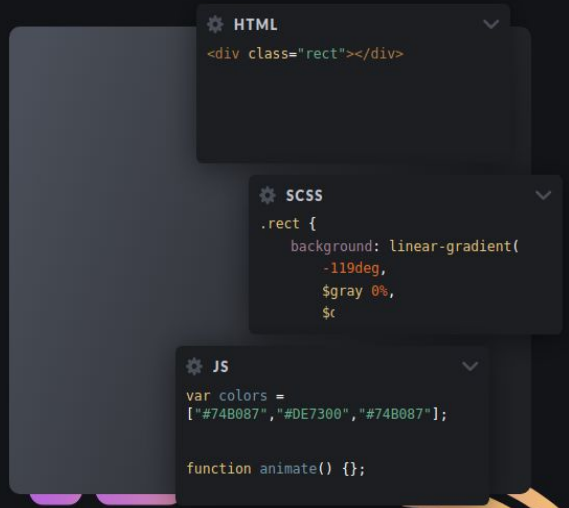
Spark

CodePen PRO

The best place to build, test, and discover front-end code.

CodePen is a **social development environment** for front-end designers and developers. Build and deploy a website, show off your work, build test cases to learn and debug, and find inspiration.

Sign Up for Free



Build & Test

Get work done quicker by building out **entire projects** or isolating code to **test features and animations**. Want to keep it all under wraps? **Upgrade to a PRO account** to keep your work private.

Try the Editor

Learn & Discover

Just starting out? Browsing, forking, and playing with Pens is a great way to understand how they were built and how code works. Teaching a class? **Professor mode** lets your students to see and comment on real-time Pen updates.

Learn More

Share Your Work

Become a part of the **most active front-end community** in the world by sharing work. Presenting at a conference? Show your code directly in the browser with **Presentation Mode**.

Explore Trending

Untitled
Captain Anonymous

Save | Settings | Change View | Sign Up | Log In

HTML

```
1
```

CSS

```
1
```

JS

```
1
```

What did the HTML say to the CSS? I like your style.

Condicional Composta else (senão)

SE (a condição for verdadeira)

    ENTÃO {executar algum comando}

SENÃO {executar outro comando}

FIM-SE

Exemplo:

Idade: number = 18;

SE (idade >= 18)

    ENTÃO { imprima a mensagem “você é um adulto” }

SENÃO { imprima a mensagem “você não é um adulto}

FIM-SE

Exemplo com JavaScript

```
let idade = 18;
If (idade >= 18) {
        alert(“você é um adulto”);
}else {
        alert(“você não é um adulto”);
}
```

Praticar!

Partiu codepen!

Exercício 1

Escreva um algoritmo em JavaScript que contenha as variáveis nome, idade e mensagens.

Inicie as variáveis com seu nome e sua idade.

Para a variável mensagens, insira as frases a seguir: (dica: use array)

“se prepare, sua vida começará a passar mais rápido!”

“depois dos 20 é um pulo pra chegar nos 30!”

Se a idade for menor ou igual a 20, exiba um alerta com a mensagem : “seuNome, se prepare, sua vida começará a passar mais rápido!”

Se a idade for maior que 20, exiba um alerta : “seuNome, depois dos 20 é um pulo pra chegar nos 30!”

Else if

Podemos juntar as condições if e else a fim de analisar uma nova possibilidade.

Se (...)

    Execute {...}

Senão Se (...)

    Execute {...}

Senão

    Execute {...}

Exemplo

```
let idade = 18;
If (idade >= 100) {
        alert(“você é um ancião”);
} else if(idade >= 18) {
        alert(“você é um adulto”);
} else {
        alert(“você é um adolescente”);
}
```

Exrercício 2

Escreva um algoritmo em JavaScript que contenha as variáveis nome, idade e mensagens.

Inicie as variáveis com seu nome e sua idade.

Para a variável mensagens, insira as frases a seguir: (dica: use array)

“se prepare, sua vida começará a passar mais rápido!”

“depois dos 20 é um pulo pra chegar nos 30!”

“temos inveja de você =(”

Se a idade for maior que 30, exiba um alerta : “Seunome, bem vindo ao time, não vai demorar para descermos pela porta da frente!”

Se a idade for maior ou igual a 20, exiba um alerta : “Seunome, se prepare, sua vida começará a passar mais rápido!”

Se a idade for menor que 20, exiba um alerta : “Seunome, temos inveja de você =(”

Switch

A condicional switch avalia uma expressão, combinando o valor da expressão para uma cláusula case, e executa as instruções associadas ao case.


WTF !

Para que serve?

Basicamente utilizamos o switch case quando há a necessidade de verificar muitas condições.

Em vez de utilizar vaaarios if’s, usamos o switch case que terá o mesmo resultado final, com uma escrita reduzida e mais legível.

Estrutura

```
Switch(variavel) {
case “valor X”:
        executar algo ;
break;
case “valor Z”:
        executar algo ;
break;
default:
        executar algo;
}
```

Var

Quando declaramos uma variável com a chave var, seu escopo será global, ou seja, ela será reconhecida em qualquer escopo (bloco) de seu contexto.

Em JavaScript, toda variável é “elevada” (hoisting) até o topo do seu contexto de execução. Esse mecanismo move as variáveis para o topo do seu escopo antes da execução do código.

Codepen

Vamos ao exemplo prático:

Var

No exemplo anterior, como a variável está dentro de uma function, a declaração da mesma é elevada (hoisting) para o topo do seu contexto, ou seja, para o topo da function.

É por esse mesmo motivo que “é possível usar uma variável antes dela ter sido declarada”: em tempo de execução a variável será elevada (hoisting) e tudo funcionará corretamente.

Problemas com var

Às vezes, queremos declarar variáveis que serão utilizadas apenas dentro de um pequeno trecho do nosso código. Ter que lidar com o escopo de função das variáveis declaradas com var (escopo abrangente) pode deixar o código muito confuso e difícil de entender.

Sabendo das "complicações" que as variáveis declaradas com var podem causar, o que podemos fazer para evitá-las?

Let

Foi pensando em trazer o escopo de bloco que o ECMAScript 6 disponibilizou essa mesma flexibilidade para a linguagem.

Através da palavra-chave let podemos declarar variáveis com escopo de bloco.

Booooooora testar!

Const

Em questão de escopo, const é idêntica a let.

Como o próprio nome já diz, seu valor é constante, ele nunca será trocado, essa tentativa vai causar um erro no javascript.

Outra particularidade é que variáveis constantes sempre devem ser inicializadas.

Operadores aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão
++	Incremento
--	Decremento

Operadores de igualdade

==	Retorna verdadeiro se os valores forem iguais
===	Retorna verdadeiro se os valores forem iguais e do mesmo tipo
!=	Retorna verdadeiro se os valores forem diferentes
!==	Retorna verdadeiro se os valores não forem iguais e/ou não forem do mesmo tipo.
>	Retorna verdadeiro se o valor da esquerda for maior que o da direita.
<	Retorna verdadeiro se o valor da esquerda for menor que o da direita.
<=	Retorna verdadeiro se o valor da esquerda for menor ou igual ao da direita.

Operadores lógicos

&& (E) Retorna verdadeiro se os dois valores forem verdadeiros

|| (OU) Retorna verdadeiro se um dos valores for verdadeiro

! (NÃO) Retorna verdadeiro se a ocorrência for falsa, e retorna falso se a ocorrência for verdadeira


OXITENO

Ter o **elemento seguro** significa...

Exercício

Crie um algoritmo que descubra sua idade (pode ser aproximado);

Armazene em uma variável o mês de seu nascimento (em forma numérica);

Exiba a mensagem “sua idade é 123, esse número é par/impar”;

Exiba a descrição do mês de seu nascimento;

10:00

Stop

Estruturas de repetição

O JS (javascript) possibilita diferentes maneiras de executar uma ação repetidas vezes, conhecidos como laços de repetição ou loop.

Veremos os tipos mais comuns:

while()

for()

.forEach()

While (enquanto)

Veremos como escrever uma estrutura de repetição while:

```
while (condição){
        execute comandos aqui;
}
```

Vamos ver na prática, abra o codepen, VsCode ou qualquer outra ferramenta!

Loop infinito

Cuidado com o loop infinito!

Se você não disser ao loop (repetição) que ele deve sair em algum momento, ele ficará executando os passos até que seu browser trave!

Como resolver?


DEDOFF
F12

for (para)

O laço for é mais utilizado que while no mundo javascript.

Sua escrita é um pouco diferente por não ter um contador dentro de seu escopo.

```
for (let i = 0; i < 5; i++) {
        execute comandos aqui;
}
```

Veremos na prática:

break/continue

Esses dois comandos auxiliam nos loops que acabamos de ver;

break: finaliza o loop assim que uma condição é atendida;

continue: pula a iteração corrente assim que uma condição é atendida;

Codemos então!

Faremos um teste utilizando o loop for:

Exercício 1

Desenvolva um algoritmo que faça iteração em um array que possua nomes e números.

Imprima somente os nomes, os números devem ser ignorados;

10:00

Stop

for aninhado

Podemos criar um array multidimensional (conhecido como matriz em outras linguagens):

[[1,2,3], [4,5,6], [7,8,9]];

Para conseguir acessar todas as suas posições, precisamos aninhar 2 laços de repetição do tipo for (por exemplo).

Aninhar é quando colocamos uma estrutura dentro da outra.

Array multidimensional

[[1,2,3], [4,5,6], [7,8,9]];

Priscila	1	2	3
Pedro	4	5	6
João	7	8	9

Array multidimensional

[[1,2,3], [4,5,6], [7,8,9]];

Posição 0	1	2	3
	P0	P1	P2
Posição 1	4	5	6
	P0	P1	P2
Posição 2	7	8	9
	P0	P1	P2

Exercício 2

Crie um array que conterá outros arrays com valores que variam entre par e ímpar.

let numeros = [[1,2,3], [4,5,6], [7,8,9]];

Com laços de repetição for aninhados, percorra este array com seus sub-arrays, busque os valores pares e imprima no console.

20:00


6ª
MAIOR
INDÚSTRIA QUÍMICA
DO MUNDO

forEach

Apesar de ser um laço de repetição, o comando forEach é um método de array.

Em comparação com o loop for, ele é mais limpo, legível e de fácil manutenção.

```
array.forEach(function(elemento){
    console.log(elemento);
})
```

forEach

O forEach é um método muito útil quando precisamos acessar todos os elementos de um Array (um por um) para depois fazer algo com eles.

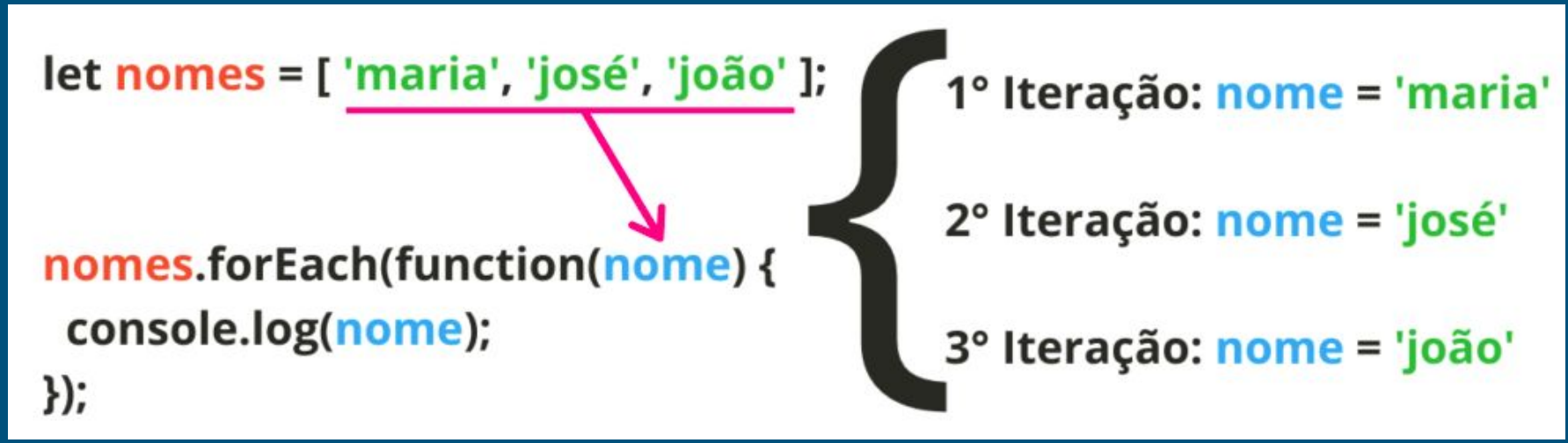
Por exemplo, desejamos mostrar no console todos os itens de um array de nomes. Ao utilizar o loop for, temos isso:

forEach

```
let nomes = ['maria', 'josé', 'joão'];
for(var i = 0; i < nomes.length; i++) {
  console.log(nomes[i]);
}

let nomes = ['maria', 'josé', 'joão'];
nomes.forEach(function(nome) {
  console.log(nome);
});
```

Exemplo forEach



forEach

Apenas imprimimos os valores no console, mas podemos fazer qualquer coisa com o valor da variável dentro da função callback, inclusive passar como parâmetro de outros métodos.

A função callback não precisa ser anônima. Podemos criar a função e depois passá-la como parâmetro ao forEach():

forEach

```
let nomes = ['maria', 'josé', 'joão'];

function imprimeNome(nome) {
  console.log(nome);
}

nomes.forEach(imprimeNome);
```

forEach

Apesar de parecido, o forEach se comporta de maneira diferente e possui menos flexibilidade que o loop for:

Não suporta break

Não suporta continue

forEach

Vamos lá :D

Codepen!

function

Definição

Funções são blocos de construção em JavaScript. Uma função é um conjunto de instruções que executa uma tarefa ou calcula um valor. Para usar uma função, você deve defini-la em algum lugar no escopo do qual você quiser chamá-la.

Definindo funções

A definição/declaração de função consiste no uso da palavra-chave **function**, seguida por:

- Nome da função;
- Lista de parâmetros/argumentos, entre parênteses separados por vírgula;
- Corpo da função, delimitada por chaves { … }

Definindo funções

Por exemplo, o código a seguir define uma função simples chamada **metade**:

```
function metade(num){
    return num / 2;
}
```

A função **metade** recebe um argumento(ou parâmetro) chamado **num**. A função consiste em uma instrução que indica para retornar o argumento da função (isto é, **num**) dividido por 2. A declaração return especifica o valor retornado pela função.

Expressão de função

Embora a declaração da função **metade** seja sintaticamente uma declaração, funções também podem ser criadas por uma expressão de função. Ela pode ser anônima, não tem que ter um nome. Por exemplo, a função **metade** poderia ter sido definida como:

```
let metade =  function(num) { return num/2; }

let x = metade(10) // x recebe 5
```

Expressão de função

As expressões de função são convenientes ao passar uma função como um argumento para outra função. O exemplo a seguir mostra uma função **dobro** sendo definida e, em seguida, chamada com uma função anônima como seu primeiro parâmetro:

Expressão de função

```
function dobro(funcao, array) {
 let result = [ ];
 let i;
 for (i = 0; i != array.length; i++)
   result[i] = funcao(array[i]);
 return result;
}
```

Vamos ver na prática:

Chamando funções

A definição de uma função não a executa. Definir a função é simplesmente nomear a função e especificar o que fazer quando a função é chamada. Chamar a função executa realmente as ações especificadas com os parâmetros indicados. Por exemplo, se você definir a função **mensagem**, você pode chamá-la do seguinte modo:

mensagem(‘hello’);

Funções

Uma função pode ser executada mesmo antes de sua declaração? Depende!

Dessa forma tudo funcionará corretamente:

```
console.log(soma(2));
function soma(num) { return num+2; }
```

Dessa forma não funcionará:

```
console.log(soma(2));
let soma = function(num) { return num+2; }
```

Exercício

Crie uma função que receba dois parâmetros:

- Array contendo siglas de Estados e nomes de Cidades;
- Função anônima que adicione a informação de quantos caracteres tem essa cidade/estado;

A função deverá percorrer o array recebido como argumento e printar os novos valores do array.

A contagem não deve ser informada caso a cidade/estado tiver menos de 3 caracteres;

Array

Já vimos bastante sobre arrays até aqui, para que servem e como funcionam.

Criando um array:

```
let letras = [‘a’, ‘b’, ‘c’]
```

Acessando um array:

```
console.log( letras[0] ) resultado: ‘a’
```

Iterar um array:

```
letras.forEach(function(letra){
        console.log(letra)
        resultado: ‘a, b, c’
})
```

Array - Métodos modificadores

Veremos a seguir os métodos modificadores de array, ou seja, modifica diretamente seus elementos.

Array - push

Utilize o comando push para adicionar um elemento ao final do array:

let letras = [‘a’, ’b’, ‘c’];

letras.push(‘d’)

[‘a’, ’b’, ‘c’, ‘d’]

Array - pop

Utilize o comando pop para remover um elemento do final do array:

let letras = [‘a’, ’b’, ‘c’];

letras.pop()

[‘a’, ’b’]

Array - shift

Utilize o comando shift para remover um elemento do início do array:

let letras = [‘a’, ’b’, ‘c’];

letras.shift()

[’b’, ‘c’]

Array - unshift

Utilize o comando unshift para adicionar um elemento ao início do array:

let letras = [’b’, ‘c’];

letras.unshift(‘a’)

[‘a’, ’b’, ‘c’]

Array - splice

Utilize o comando splice para remover/editar ou adicionar um elemento em uma determinada posição.

Dependendo da quantidade de argumentos passados para esse método, ele pode executar uma remoção/ edição ou adição de um ou mais itens. O terceiro argumento sempre servirá para adicionar um elemento ao array. A omissão dele fará com que uma remoção seja efetuada.

splice(indice, quantidade a deletar, item a ser adicionado)

Array - splice - adicionando

Para adicionar um elemento, devemos passar 3 argumentos para o método splice:

let letras = [‘a’, ‘c’];

letras.splice(1, 0, ‘b’)

Se traduzirmos isso, fica mais fácil de entender. O comando acima está dizendo:

Antes da posição 1, quero que remova 0 item e adicione o valor b

Array - splice - editando

Para “editar” um elemento, devemos passar 3 argumentos para o métido splice:

let letras = [‘a,’bbb’, ‘c’];

letras.splice(1, 1, ‘bbb’)

Porém dessa vez substituímos o valor do segundo parâmetro por 1. Essa edição na verdade é uma remoção seguida de inserção.

Traduzindo: Antes da posição 1, quero que remova 1 item e adicione o valor bbb

Array - splice - removendo

Para apenas remover um elemento, devemos passar 2 argumentos para o métido splice:

let letras = [‘a’, ‘b’, ‘c’];

letras.splice(1, 1)

Traduzindo: Na posição 1, quero que remova 1 item.

Array - sort

Ordena os itens de um array, de acordo com a tabela Unicode. Essa ordem é seguida de alguns símbolos, números, letras maiúsculas, etc.

let letras = [‘c’, ‘b’, ‘a’]

letras.sort()

[‘a’, ‘b’, ‘c’]

Esse método pode receber uma função como argumento, dentro dessa função é possível manipular os valores a fim de criar ordenações customizadas.

Array - sort

let letras = [1,2,3,4,5,6,7,8,9,10]

letras.sort()

Vamos ver o resultado?

Array - sort

Por esse motivo no exemplo anterior o “10” veio antes do “2” porque “1”, que é o primeiro caractere de “10”, vem antes do “2”.

Resolvemos isso desse modo:

```
let letras = [1,2,3,4,5,6,7,8,9,10];

letras.sort(function(a, b){
        return a - b;
})
```

Array - sort

```
letras.sort(function(a, b){
    return a - b;
})
```

- se a comparação for menor que zero, a é posicionado antes de b
- se a comparação for maior que zero, a é posicionado depois de b
- se a comparação for igual a zero, a e b permanecem com as posições inalteradas

Array - Métodos de acesso

Esses métodos não modificam o array, retornam alguma representação desse array.

Array - concat

Concatena (junta) dois arrays distintos e retorna o restultado em um único array:

let arr1 = [‘a’, ‘b’, ‘c’]

let arr2 = [‘d’, ‘e’, ‘f’]

let arr3 = arr1.concat(arr2)

Array - includes

Determina se um array contém um determinado elemento, retornando true ou false apropriadamente.

let arr1 = [‘a’, ‘b’, ‘c’];

arr1.includes(‘a’)

true

arr1.includes(‘f’)

false

Array - indexOf

Utilize o comando indexOf para encontrar o índice de um elemento no array:

lestras.indexOf(‘valor’, 2)

let letras = [‘a, ’b’, ‘c’];

letras.indexOf(‘a’)

0

Lembrando que o indexOf vai retornar a primeira ocorrência encontrada, ou seja:

[‘a’, ‘b’, ‘a’] -> o índice retornado será o primeiro encontrado, 0 nesse caso.

Caso o elemento pesquisado não exista, o método retonará -1

Array - join e toString

Junta todos os elementos de um array em uma string e retorna esta string, separando os elementos por vírgula.

A diferença é que com join com podemos especificar o separador:

let array = [‘a’, ‘b’, ‘c’]

array.join(‘|’)

‘a|b|c’

Array - filter

Cria um novo array com todos os elementos que passaram no teste implementado pela função fornecida.

```
let array = [‘a’, ‘b’, ‘a’, ‘c’, ‘d’, ‘a’];

function encontraLetra(letra) {
        return letra == ‘a’
}

let encontrados = array.filter(encontraLetra)
```

Array - map

Invoca a função callback passada por argumento para cada elemento do Array e devolve um novo Array como resultado.

let numeros = [2, 5, 7];

numeros.map((element => element * 2))

[4, 10, 14]

Vamos ver um exemplo percorrendo um array de objetos:

[{nome: ‘nome1’, idade: 22}...]

Array - some

Testa se ao menos um dos elementos no array passa no teste implementado pela função atribuída e retorna um valor true ou false.

```
function maiorQue10(element) {
 return element > 10;
}

[12, 5, 8, 1, 4].some(maiorQue10);
```

true

Array - reduce

Executa uma função (fornecida por você) para cada elemento do array, resultando num único valor de retorno.

Parâmetros:

1. Acumulador
2. Valor Atual
3. Index Atual
4. Array original

Array - reduce

O valor de retorno da sua função reducer é atribuída ao acumulador. O acumulador, com seu valor atualizado, é repassado para cada iteração subsequente pelo array, que por fim, se tornará o valor resultante, único, final.

```
[0, 1, 2, 3, 4].reduce(( acumulador, valorAtual) => acumulador + valorAtual)
```

10

Exercício

Com os métodos aprendidos até o momento, encontre na lista seguinte todos os valores ‘1’ e insira seu index em um novo array:

array = [1, 2, 3, 1, 4, 5, 1, 6, 1]

NÃO USE IF


IDENTIDADE
CULTURA
VALORES

Exercício

Crie uma função que receba o seguinte array como parâmetro:

[6,1,3,2,4,7,9,8,0]

Imprima a saída:

0 - 2 - 4 - 6 - 8 - 1 - 3 - 7 - 9

pares 0 2 4 6 8

impares 1 3 7 9