DESENVOLVIMENTO WEB

Objetivo de Aprendizagem

- Nesta unidade será apresentado ao aluno a estrutura e a funcionalidade de uma linguagem de programação utilizando formas de representações de problemas com construções de algoritmos.

Aplicabilidade da Programação

- Presente em todas as áreas da computação:
  - Hardware;
  - Sistemas Operacionais;
  - Análise de Sistemas;
  - Banco de Dados;
  - Desenvolvimento Web;
  - Redes de Computadores;
  - Etc.

Sequência Lógica:

- Estes pensamentos devem ser descritos como uma *sequência de instruções*, que devem ser seguidas em ordem para se cumprir uma determinada tarefa;
- *Passos* executados até se atingir um objetivo ou solução de um problema

Instrução:

- Cada um dos *passos*, cada uma das ações a tomar (obedecendo a *sequência lógica*) para ir resolvendo o problema, ou para ir executando a tarefa;
- Uma só instrução não resolve problemas.

EXEMPLO: para “fazer omelete”

- Instruções: “quebrar ovos”, “bater ovos”, “pôr sal”, “ligar fogão”, “pôr óleo na frigideira”, “pôr frigideira no fogo”, “fritar ovos batidos”, etc...
- Quanto às instruções isoladas:
  - Só “quebrar ovos”, ou só “pôr óleo na frigideira”, não é suficiente para cumprir a tarefa “fazer omelete”
- Quanto à sequência lógica:
  - Se executarmos “fritar ovos batidos” antes de “bater ovos”, ou pior, antes de “quebrar ovos”, não iremos cumprir a tarefa “fazer omelete”

Algoritmo

- Sequência finita de passos, seguindo uma sequência lógica que levam à execução de uma tarefa;
- Claro e preciso.

Exemplo de algoritmo

Quando uma dona de casa prepara um bolo, segue uma receita, que nada mais é do que um algoritmo em que cada instrução é um passo a ser seguido para que o prato fique pronto com sucesso:

1. Bata 4 claras em neve
2. Adicione 2 xícaras de açúcar
3. Adicione 2 colheres de farinha de trigo, 4 gemas, uma colher de fermento e duas colheres de chocolate
4. Bata por 3 minutos
5. Unte uma assadeira com margarina e farinha de trigo
6. Coloque o bolo para assar por 20 minutos

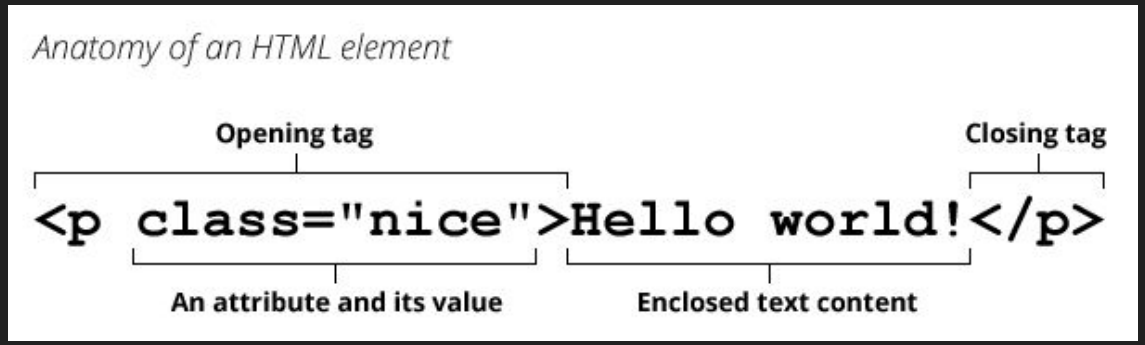
FASES para desenvolver o algoritmo:

- Determinar o problema.
- Dividir a solução nas três fases:

- Exemplo:
  - Problema: calcular a média de dois números
  - Dados de entrada: os números, N1, N2
  - Processamento: somar os dois números e dividir a soma por 2
  - Dados de saída: a média $\frac{N1 + N2}{2}$

Introdução a HTML

- HTML - HyperText Markup Language: é um documento em texto simples estruturado com elementos. Os elementos estão rodeados por marcas de abertura e fecho correspondentes. Cada marca começa e termina com parênteses angulares (<>). Existem alguns elementos vazios ou vazios que não podem incluir qualquer texto, por exemplo <img>.
- Poderá estender as marcas HTML com atributos, que fornecem informações adicionais que afetam a forma como o navegador interpreta o elemento:



- O arquivo HTML é normalmente salvo com uma extensão .html ou .htm

Introdução ao Javascript

- JavaScript é uma linguagem de programação que permite a você implementar itens complexos em páginas web — toda vez que uma página da web faz mais do que simplesmente mostrar a você informação estática.

Características da linguagem - Sintaxe

- Sintaxe
  - Javascript é uma linguagem interpretada.
    - O código é executado de cima para baixo e o resultado da execução do código é imediatamente retornado.
  - É uma linguagem case-sensitive.
    - A linguagem diferencia caracteres maiúsculos e minúsculos na declaração das variáveis.

Variável

- Representa uma posição na memória, onde pode ser armazenado um dado;
- Possui um nome e um valor;
- Durante a execução do algoritmo, pode ter seu valor alterado.

Declaração de variável - Atribuições

- Atribui o valor da direita à variável da esquerda
  - MEDIA = (N1+N2) / 4

    (Lê-se média recebe N1 +...)
  - Neste caso, estamos atribuindo o resultado da fórmula à variável média;
- Outros Exemplos:
  - a = 3;
  - a = x;

Declaração de variável - Identificadores

- É a identificação/nome da variável, exemplo: nota = 10.

- REGRAS:
  1. Pode começar com uma letra a-z, A-Z, _, $
  2. Os próximos podem ser letras ou números
  3. Não pode utilizar nenhum símbolo, exceto _, $
  4. Não pode conter espaços em brancos
  5. Não pode conter acentos
  6. Não pode utilizar palavras-reservadas

Declarações de variáveis - Tipos de variáveis

var	let	const
Declara uma variável, opcionalmente, inicializando-a com um valor.	Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor. Podem ter seu valor atualizado	Declara uma constante de escopo de bloco, apenas de leitura. Não podem ter seu valor atualizado

- Enquanto **var** e **let** podem ser declaradas sem ser inicializadas, **const** precisa da inicialização durante a declaração.

Declaração de variáveis - Escopo de variáveis

- Uma variável pode pertencer a um dos seguintes escopos:
  - Escopo global: o escopo padrão para todos os códigos em execução no modo de script.
  - Escopo da função: O escopo criado com uma função.
- As variáveis declaradas com let ou const podem pertencer a um escopo adicional:
  - Escopo do bloco: O escopo criado com um par de chaves (um bloco).

Declaração de variáveis - Tipos de Dados

Nome	Descrição
BOOLEAN	Um tipo de dado lógico que pode ter apenas um de dois valores possíveis: verdadeiro ou falso.
STRING	Em qualquer linguagem de programação, uma string é uma sequência de caracteres usados para representar texto.
NUMBER	É um tipo de dado numérico
INTEGER	São números inteiros, ex. 10, 400 ou -5.
FLOAT	Tem pontos e casas decimais, por exemplo 12.5 e 56.7786543.
NULL	O valor null é um literal em JavaScript que representa um valor nulo ou "vazio".
UNDEFINED	Uma variável que não recebeu um valor tem o valor indefinido.

Operadores Aritméticos

Operador	Nome	Exemplo
+	Adição	6 + 9
-	Subtração	20 - 15
*	Multiplicação	3 * 7
/	Divisão	10 / 5
%	Resto da divisão	10%2
**	Potência/Expoente	4**2

Hierarquia das Operações Aritméticas

1 º ( ) Parênteses
2 º Exponenciação
3 º Multiplicação, divisão (o que aparecer primeiro)
4 º + ou – (o que aparecer primeiro)

EXERCÍCIOS

- Faça um algoritmo que imprima seu nome, sua idade, e seu estado civil, colocando esse último item como um tipo booleano (true / false).
- Faça um algoritmo que leia quatro números informados e que depois mostre no console a média desses valores.
- Faça a tabuada do valor 5.
- Faça um algoritmo que leia um número e mostre seu antecessor e seu sucessor.
- Faça um algoritmo que leia um valor, e mostre esse valor com 5% de desconto.

TEMPLATE STRING && COMENTARIOS

- Como fazer comentários no JavaScript ?

```
// Texto

/*
        Texto
        Texto
*/
```

- O que é uma template String ?
  - São strings que permitem expressões embutidas.
  - *Template strings* são envolvidas por (acentos graves) (` `) em vez de aspas simples ou duplas. *Template strings* podem possuir *placeholders*. Estes são indicados por um cifrão seguido de chaves (${expression})

```
let nome = “Kelvin”
console.log(`Seu nome
é ${nome}`);
```

Estruturas de Controle

Operador	Nome	Exemplo
==	Igualdade	5 == 5 ou 5 == ”5”
!=	Diferença/Não-igualdade	30 != 0
!==	Diferença estrita/Não-igualdade estrita (não é o mesmo?)	'Chris' !== 'Ch' + 'ris'
===	Igualdade estrita (é estritamente o mesmo?)	5 === 2 + 4
<	Menor que	10 < 6
>	Maior que	10 > 20
<=	Menor ou igual que	10 <= 20
>=	Maior ou igual que	20 >= 10
&&	E	3 && 4 < 5
||	Ou	5 || 7 < 6

COMANDOS DE CONTROLE - IF ... ELSE

if (condição) afirmação 1 [else afirmação 2]

SINTAXE

EXEMPLO 1

```
if (condição) {
        código
} else {
        código
}
```

EXEMPLO 2

```
if (condição) {
        código
} else if (condição) {
        código
}
```

EXEMPLO 3

```
if (condição) {
        código
        if (condição) {
                código
} } else {
        código
}
```

COMANDOS DE CONTROLE - SWITCH ... CASE

- Variável = Expressão que será validada.
- Valor = Condição que deverá ser verificada para a execução das instruções.
- Break = Serve para encerrar a leitura do switch. Sem este comando, o código do case abaixo será executado na sequência.
- Default = executará quando nenhum case corresponder ao valor especificado.

EXEMPLO 1

```
let variável = valor;
switch (variável) {
        case:
                instrução
        break;
        case:
                instrução
        break;
        default:
                instrução
}
```

EXEMPLO 2

```
let podio = 1;
switch (podio) {
        case 1:
        console.log(`Você ficou em ${podio} lugar`);
        break;
        case 2:
                console.log(`Você ficou em ${podio} lugar`);
        break;
        case 3:
                console.log(`Você ficou em ${podio} lugar`);
        break;
        default:
                console.log(`Não subiu no pódio`);
}
```

EXERCÍCIOS

obs: os exercícios a seguir tentem utilizar template string em todos

SWITCH CASE EXERCÍCIOS

1 - Faça um código com switch/case, e para cada caso, diga se o usuário tem permissão comum, gerente, diretor.

2 - Faça uma calculadora que realize as operações de soma, subtração, adição, e divisão.

Utilize o switch/case para fazer que os usuários escolha a opção individual de cada operação

IF... ELSE EXERCÍCIOS

1 - Escreva um programa para ler 2 valores (considere que não serão informados valores iguais) e escrever o maior deles.

2 - Escreva um programa para ler o ano de nascimento de uma pessoa e escrever uma mensagem que diga se ela poderá ou não votar este ano (não é necessário considerar o mês em que ela nasceu).

3 - Escreva um programa que verifique a validade de uma senha fornecida pelo usuário. A senha válida é o número 1234.

- Devem ser impressas as seguintes mensagens:
- ACESSO PERMITIDO caso a senha seja válida.
- ACESSO NEGADO caso a senha não seja válida.4

4 - Calcule o Índice de massa corporal

kg / (altura * altura)

- Mostre o índice na tela.
- E diga ao usuário o que o índice dele corresponde.

MENOR QUE 18.5 = MAGREZA

ENTRE 18.5 E 24.9 = NORMAL

ENTRE 25.0 E 29.9 = SOBREPESO

ENTRE 30.0 E 39.9 = OBESIDADE

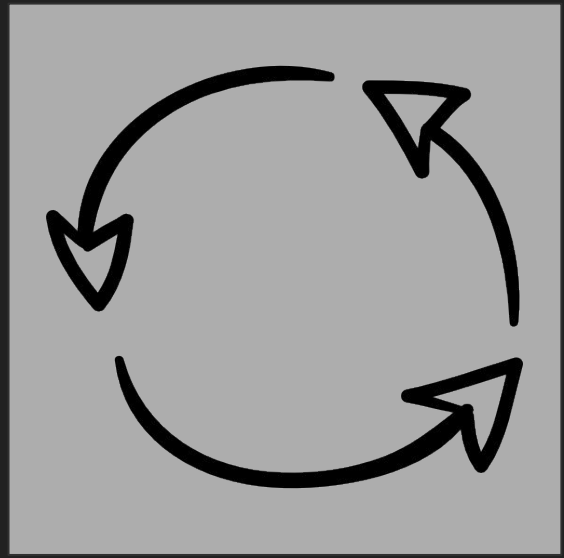
MAIOR QUE 40.0 = OBESIDADE GRAVE

5 - Escreva um programa para ler 3 valores inteiros (considere que não serão lidos valores iguais) e escrevê-los em ordem crescente.

ESTRUTURA DE REPETIÇÃO

O que é uma estrutura de repetição ?

- Dentro da lógica de programação é uma estrutura que permite executar mais de uma vez o mesmo comando ou conjunto de comandos, de acordo com uma condição ou com um contador, um loop.



UTILIDADE DA ESTRUTURA DE REPETIÇÃO

```
let num = 5;
console.log(num * 0);
console.log(num * 1);
console.log(num * 2);
console.log(num * 3);
console.log(num * 4);
console.log(num * 5);
console.log(num * 6);
console.log(num * 7);
console.log(num * 8);
console.log(num * 9);
console.log(num * 10);
```

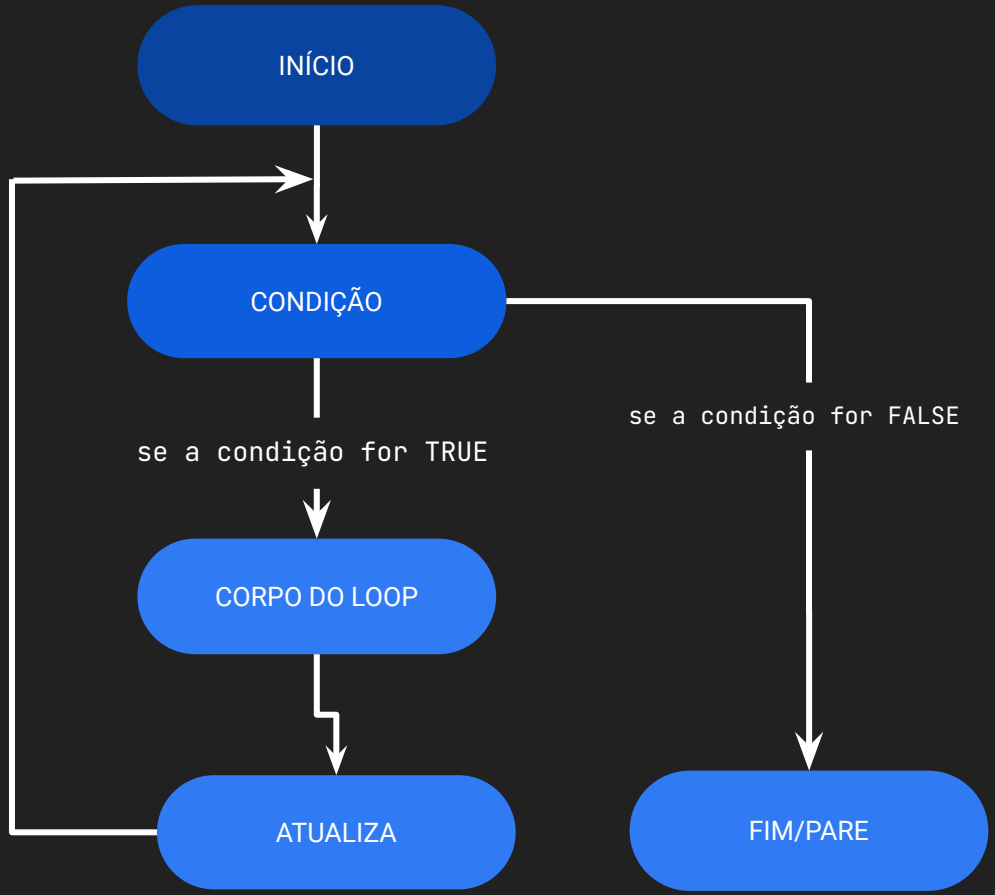
ESTRUTURAS DE REPETIÇÃO

WHILE	let contador = 0;
	
	while(contador < 10){
	  console.log(contador);
	  contador++;
	}

DO… WHILE	let contador = 0;
	do {
	  contador++;
	  console.log(contador);
	} while (contador < 10);

FOR	for(let i = 0; i < 10; i++){
	  console.log(i);
	}

ESTRUTURA DE REPETIÇÃO



EXEMPLO - ESTRUTURA DO … WHILE

```
let contador = 0;
do {
  console.log(contador);
  contador++
} while (contador < 5);    //  0
                               1
                               2
                               3
                               4
```

EXEMPLO - ESTRUTURA WHILE

```
let contador = 0;

while(contador < 5){

  console.log(contador)

  contador++;

}                          //  0
                               1
                               2
                               3
                               4
```

EXEMPLO - ESTRUTURA FOR

```
for(let i = 0; i <5; i++){
  console.log(i);
}
//  0
    1
    2
    3
    4
```

ARRAY && OBJETOS - ARRAY

O que é um array ?

- ***Array*** é uma estrutura de dados simples presente na maioria das linguagens de programação. Seu principal objetivo é ser um espaço contínuo na memória para organizar e armazenar uma coleção de elementos sequencialmente.

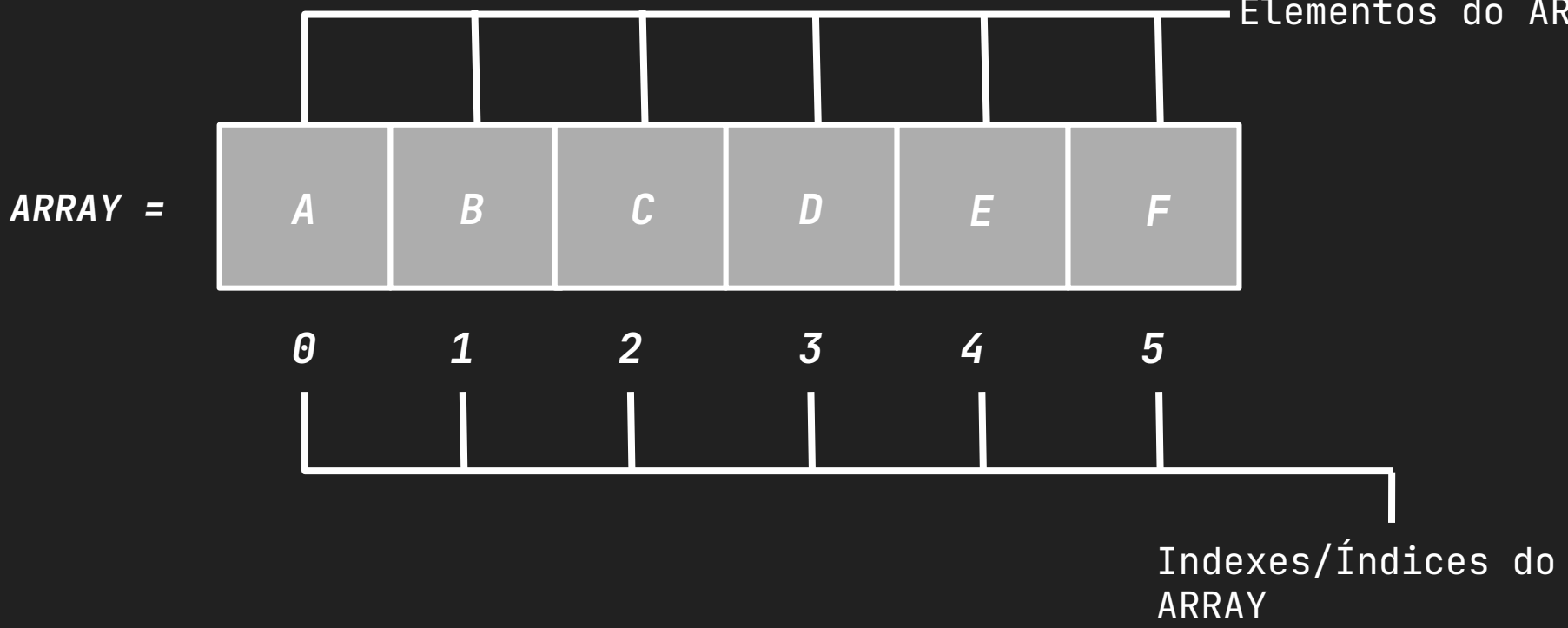
Assim, essas informações podem ser acessadas rapidamente por meio da indicação do índice da posição pretendida.

Para que serve um array?

- A principal finalidade do **array** nada mais é do que agrupar uma coleção de elementos em uma única variável, ou seja, para cada linha de programação, um dado.

ARRAY && OBJETOS - ARRAY

- ***Coleção de Elementos Sequencialmente.***



DECLARAÇÃO DE ARRAY

- ***Como declarar um array?***

  - let frutas = [“Morango”, "Maçã"];
  - let frutas = Array()
  - let frutas = [];

- ***Como acessar um ITEM (index) do array?***

  Para acessar itens dentro do array, primeiramente temos que saber a quantidade de índices que ele possui, e qual valor os elementos equivalem.

  - ***let primeiro = frutas[0];***

MÉTODOS DO ARRAY

- ***Para adicionar um item no final. push()***
  - ***frutas.push(“Goiaba”);***

- ***Para remover um item no final do array. pop()***
  - ***frutas.pop(“Goiaba”);***

- ***Para adicionar um item no começo do array. unshift()***
  - ***frutas.unshift(“Goiaba”);***

- ***Para remover um item no começo do array. shift()***
  - ***frutas.shift(“Goiaba”);***

MÉTODOS DO ARRAY

- *Procurar um índice dentro do array. indexOf()*
  - *let index = frutas.indexOf(“Morango”);*

- *Remover item pela posição. splice(pos, index)*
  - *let n = 1, pos = 2;*
    let remover = frutas.splice(pos, n);

- *Como copiar um array e suas partes. slice()*
  - *let copia = frutas.slice();*
  - *let copia_indice_zero = frutas.slice()[0];*

- *Como ver o tamanho do array. length()*
  - *let tamanho = frutas.length();*

- *Como ver se algo está incluído. includes()*
  - *let existe = frutas.includes(“Morango”);*

ARRAY && OBJETO - OBJETO

O que é um objeto ?

- Um objeto **é uma entidade independente, com propriedades e tipos**. Compare-o com uma xícara, por exemplo. Uma xícara é um objeto, com propriedades. Uma xícara tem uma cor, uma forma, peso, um material de composição.
- Um **objeto** em JavaScript tem propriedades associadas a ele. Uma propriedade de um objeto pode ser explicada como uma variável que é ligada ao objeto. Propriedades de objetos são basicamente as mesmas que variáveis normais em JavaScript, exceto pelo fato de estarem ligadas a objetos.Um objeto em JavaScript tem propriedades associadas a ele. Uma propriedade de um objeto pode ser explicada como uma variável que é ligada ao objeto.

ARRAY && OBJETOS - OBJETOS

- Sintaxe
  - nomeDoObjeto.nomeDaPropriedade

```
let pessoa = {
  nome: 'Heloysa',
  idade: 21,
  pais: "Brasil"
}
console.log(pessoa)
console.log(pessoa.nome)
//{nome: 'Heloysa', idade: 21, pais: 'Brasil'}
//Heloysa
```

ARRAY && OBJETOS - OBJETOS

```
let pessoa = {
  nome: "Heloysa",
  idade: 21,
  linguaguens_de_programacao: ["Python", "C",
"Typescript", “Javascript”],
  caracteristicas: {
    olhos: "Castanho",
    cabelo: "Castanho Escuro"
  }
}
console.log(pessoa.linguaguens_de_programacao[1]);
console.log(pessoa.caracteristicas.cabelo);
```

//C

//Castanho Escuro

ARRAY && OBJETOS - OBJETOS

```
let pessoa = {
  nome: "Heloysa",
  idade: 21,
  linguaguens_de_programacao: ["Python", "C",
"Typescript", “Javascript”],
  caracteristicas: {
    olhos: "Castanho",
    cabelo: "Castanho Escuro"
  }
}
console.log(pessoa.linguaguens_de_programacao[1]);
console.log(pessoa.caracteristicas.cabelo);
```

//C

//Castanho Escuro

ARRAY && OBJETOS - OBJETOS

```
let pessoa = {
  nome: "Heloysa",
  idade: 21,
  pais: "Brasil",
  caracteristicas : {
    olhos: "castanho",
    cabelo: "liso"
  }
}
console.log(pessoa); //nome: 'Heloysa', idade: 21, pais: 'Brasil', caracteristicas: {…}}
pessoa.nome = 'Kelvin';
pessoa.características.cabelo = 'ondulado'
console.log(pessoa); // {nome: 'Kelvin', idade: 21, pais: 'Brasil', caracteristicas: {…}}
console.log(pessoa.nome); // Kelvin
console.log(pessoa.caracteristicas.cabelo) // Ondulado
```

EXERCÍCIOS

obs: os exercícios a seguir tentem utilizar template string em todos

Exercício Estrutura de Repetição - For

- Faça uma tabuada de 10.

Exercício Array

- Faça um array com 3 valores de média, tire a média desses três valores e guarde em uma variável, no final mostre a variável com a média.

Exercício Objeto

- Faça um objeto que contenha um nome de um filme, a nota que você dá pra ele 0-10, gênero do filme, ao final faça um console personalizado, mostrando o filme, a nota, e o gênero do filme.

EXERCÍCIOS

- Faça um programa que tenha um ARRAY, contendo 3 elementos dentro, e utilizando o método SLICE, pegue a primeira letra de cada elemento.

- Faça um array que contenha 5 números, e com a estrutura de repetição FOR, e utilizando o IF/ELSE diga qual é o maior número do array.

- Faça um programa com o loop FOR, que some todos os valores do loop, e ao final mostre a média desses valores, e a soma

- Faça um programa com o loop FOR, que some todos os valores do loop, e ao final mostre a média desses valores, e a soma

- Faça um array que contenha 5 números, e com a estrutura de repetição FOR, e utilizando o IF/ELSE diga qual é o MENOR número do array.

EXERCÍCIOS

- Faça um loop FOR que encontre todos os números PARES, e todos os números ÍMPARES, e armazene respectivamente os dois em um array denominado PAR e outro ÍMPAR
  dica (Para sabermos se um número é PAR o seu resto da divisão por 2, deve ser igual a 0,
  Para sabermos se um número é ÍMPAR, seu resto da divisão por 2 deve ser diferente de 0. exemplo
  1 % 2 == 0 - Impar
  2 % 2 == 0 - Par
)

ARRAY && OBJETO - OBJETO

O que podemos extrair de um Objeto ?

- As *“keys”* de um objeto são a lista de nomes das propriedades.
- Os *“values”* de um objeto são a lista de valores das propriedades.
- As *“entries”* de um objeto são a lista de pares de nomes das propriedades e os seus valores correspondentes.

ARRAY && OBJETO - OBJETO

Vamos utilizar o seguinte exemplo de um objeto Javascript.

```
const hero = {
  nome: 'Batman',
  cidade: 'Gotham'
};
```

- As “keys” de hero são [‘nome’, ‘cidade’].
- Os “values” são [‘Batman’, ‘Gothan’].
- As “entries” são [[‘nome’,‘Batman’], [‘cidade’, ‘Gothan’]]

ARRAY && OBJETO - OBJETO

1. Object.keys() retorna as keys.

```
const hero = {
  nome: 'Batman',
  cidade: 'Gotham'
};
Object.keys(hero); // ⇒ ['nome', 'cidade']
```

ARRAY && OBJETO - OBJETO

2. Object.values() retorna os valores.

```
const hero = {
  nome: 'Batman',
  cidade: 'Gotham'
};
Object.values(hero); // ⟹ ['Batman', 'Gotham']
```

ARRAY && OBJETO - OBJETO

3. Object.entries() retorna as entradas.

```
const hero = {
  nome: 'Batman',
  cidade: 'Gotham'
};
Object.entries(hero); // ⇒ `[['name', 'Batman'],
['city', 'Gotham']]`
```

ARRAY && OBJETO - OBJETO

MÉTODOS DO ARRAY - Utilizando funções.

- forEach()
  - Permite executar uma ação para cada item de um array.
- filter()
  - O método cria um novo array filtrando as informações do array selecionado.
- findIndex()
  - O método retorna o Index de determinado item, através de uma função.

ESTRUTURA FOR - OUTRAS FORMAS

```
let cores = ['Azul', 'Amarelo', 'Ciano'];

for(let i in cores){
  console.log(cores[i]);
}
// ⇒ Azul, Amarelo, Ciano

for(let cor of cores){
  console.log(cor)
}
// ⇒ Azul, Amarelo, Ciano
```

EXERCÍCIOS

obs: os exercícios a seguir tentem utilizar template string em todos

EXERCÍCIOS

- Utilizando um dos recursos mostrado anteriormente, faça um código para detectar se o objeto está vazio ou não.
- Faça um objeto chamado ListaDeCompras, e dentro desse objeto crie uma propriedade chamada produtos, e dentro dela coloque os produtos, e seus respectivos valores, e utilizando uma dos recursos mostrados anteriormente, faça a soma total desses valores.
- Crie um objeto, e utilizando um dos recursos mostrado anteriormente, ache a propriedade que contenha 0 como valor.
  Objeto exemplo:

```
const livros = {
  'O Iluminado': 50,
  'Os sete maridos de Evelyn Hugo': 60,
  'O Hobbit': 0,
}
```

EXERCÍCIOS

- Utilizando um dos recursos mostrado anteriormente, faça um código para detectar se o objeto está vazio ou não.
- Faça um objeto chamado ListaDeCompras, e dentro desse objeto crie uma propriedade chamada produtos, e dentro dela coloque os produtos, e seus respectivos valores, e utilizando uma dos recursos mostrados anteriormente, faça a soma total desses valores.
- Crie um objeto, e utilizando um dos recursos mostrado anteriormente, ache a propriedade que contenha 0 como valor.
  Objeto exemplo:

```
const livros = {
  'O Iluminado': 50,
  'Os sete maridos de Evelyn Hugo': 60,
  'O Hobbit': 0,
}
```

EXERCÍCIOS

- Faça um objeto que contenha uma propriedade, e seu valor seja uma lista de nomes, utilizando os recursos mostrados anteriormente, mostre qual é o maior nome da lista.

- Repetindo o objeto abaixo em seu código, utilizando os recursos mostrados anteriormente, crie uma nova lista contendo apenas as palavras que começam com F.

```
const carros = {

 marcas: ["Fiat", "Chevrolet", "Ford", "Volkswagen"]

}
```

FUNÇÕES

Funções são blocos de construção fundamentais em JavaScript. Uma função é um procedimento de JavaScript - 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.

DECLARAÇÕES DE FUNÇÕES

A definição da função (também chamada de declaração de função) consiste no uso da palavra chave function, seguida por:

- Nome da Função.
- Lista de argumentos para a função, entre parênteses e separados por vírgulas.
- Declarações JavaScript que definem a função, entre chaves { }.

```
function nome_da_função(parâmetros) {
  instrução
}
```

EXEMPLO DE DECLARAÇÃO

```
function soma(n1, n2) {
  return  n1 + n2
};
soma(10,5)
console.log(soma(10,5)); // ⇒ 15
```

A declaração return especifica o valor retornado pela função.

DECLARAÇÕES DE FUNÇÕES

Parâmetros primitivos (como um número) são passados para as funções por valor; o valor é passado para a função, mas se a função altera o valor do parâmetro, esta mudança não reflete globalmente ou na função chamada.

Se você passar um objeto (ou seja, um valor não primitivo, tal como Array ou um objeto definido por você) como um parâmetro e a função alterar as propriedades do objeto, essa mudança é visível fora da função, conforme mostrado no exemplo a seguir:

EXEMPLO - PARÂMETRO NÃO PRIMITIVO

```
function minhaFuncao(objeto) {
    objeto.make = "Toyota";
  }

  var meucarro = {make: "Honda", model: "Accord", year: 1998};
  var x, y;

  x = meucarro.make;      // x recebe o valor "Honda"

  minhaFuncao(meucarro);
  y = meucarro.make;      // y recebe o valor "Toyota"
                      // (a propriedade make foi alterada pela função)
```

EXPRESSÃO DE FUNÇÃO

Embora a declaração de função acima seja sintaticamente uma declaração, funções também podem ser criadas por uma expressão de função. Tal função pode ser anônima; ele não tem que ter um nome. Por exemplo, a função soma poderia ter sido definida como:

```
let soma = function(n1,n2) {
    return n1 + n2
};
let x = soma(10, 5) //x recebe o valor 15
```

CHAMANDO UMA FUNÇÃO

A definição de uma função não a executa. Chamar a função executa realmente as ações especificadas com os parâmetros indicados. Por exemplo, se você definir a função soma, você pode chamá-la do seguinte modo:

```
soma(10, 5)
```

A declaração anterior chama a função com os argumentos 10 e 5. A função executa as instruções e retorna o valor 15.

Funções devem estar no escopo quando são chamadas. O escopo de uma declaração de função é a função na qual ela é declarada (ou o programa inteiro, se for declarado no nível superior).

Os argumentos de uma função não estão limitados a strings e números. Você pode passar objetos para uma função.

CHAMANDO UMA FUNÇÃO

Uma função pode chamar a si mesma. Por exemplo, a função que calcula os fatoriais recursivamente:

```
function factorial(n) {
  if (n === 0 || n === 1) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}

const a = factorial(1); // a recebe o valor 1
const b = factorial(2); // b recebe o valor 2
const c = factorial(3); // c recebe o valor 6
const d = factorial(4); // d recebe o valor 24
const e = factorial(5); // e recebe o valor 120
```

ESCOPO DE FUNÇÃO

Variáveis definidas dentro de uma função não podem ser acessadas de qualquer lugar fora da função, porque a variável é definida apenas no escopo da função. No entanto, uma função pode acessar todas as variáveis e funções definidas dentro do escopo em que é definida.

Em outras palavras, uma função definida no escopo global pode acessar todas as variáveis definidas no escopo global. Uma função definida dentro de outra função também pode acessar todas as variáveis definidas em sua função pai, e quaisquer outras variáveis às quais a função pai tem acesso.

RECURSIVIDADE

Uma função pode referir- se e chamar- se a si mesma. Há três formas de uma função se referir a si própria:

1. O nome da função
2. argumentos.callee() => A propriedade contém a função atualmente em execução à qual os argumentos pertencem.
3. Uma variável no escopo que se refere à função

```
const (metodo1) foo = function (metodo3) bar() {
  // declaracoes (metodo2)
};
```

FUNÇÃO RECURSIVA

Uma função que chama a si mesma é chamada de função recursiva. Em alguns casos, a recursividade é análoga a um laço. Ambos executam o código várias vezes, e ambos necessitam de uma condição (para evitar um laço infinito, ou melhor, recursão infinita, neste caso).

```
let x = 0;
while (x < 10) { // "x < 10" a condição do laço
   x++;
}
     Pode ser convertido em função recursiva em uma chamada para a função:
function loop(x) {
   if (x ⩾ 10) { // "x ⩾ 10" a condição de parada (equivalente a "!(x < 10)")
      return;
   }
   loop(x + 1); // chamada recursiva
}
loop(0);
```

USANDO OBJETOS

Os argumentos de uma função são mantidos em um objeto do tipo array. Dentro de uma função, você pode endereçar os argumentos passados para ele conforme:

```
arguments[i]
```

onde i é um número ordinal do argumento, começando com zero. Então, o primeiro argumento passado para a função seria arguments[0]. O número total de argumentos é indicado por arguments.length.

Usando o objeto arguments, você pode chamar a função com mais argumentos do que o formalmente declarado. Isso muitas vezes é útil se você não sabe de antemão quantos argumentos serão passados para a função. Você pode usar arguments.length para determinar a quantidade de argumentos passados para a função, e então acessar cada argumento usando o objeto arguments.

Por exemplo, considere uma função que concatena várias *strings*. O argumento formal para a função é uma *string* que especifica os caracteres que separam os itens para concatenar. A função definida como segue:

```
function myConcat(separador) {
  let result = "", // inicializa a lista
      i;
  // itera por meio de argumentos
  for (i = 1; i < arguments.length; i++) {
     result += arguments[i] + separador;
  }
  return result;
}
```

Você pode passar qualquer quantidade de argumentos para esta função, e ela concatena cada argumento na *string* "list":

```
myConcat(", ", "red", "orange", "blue"); // retorna "red, orange, blue, "
myConcat("; ", "elephant", "giraffe", "lion", "cheetah"); // retorna "elephant;
giraffe; lion; cheetah; "
myConcat(". ", "sage", "basil", "oregano", "pepper", "parsley"); // retorna "sage.
basil. oregano. pepper. parsley. "
```

ARROW FUNCTION (FUNÇÃO SETA)

Uma expressão de arrow function/função seta (também chamada de seta gorda para distinguir de uma sintaxe hipotética -> no JavaScript futuro) tem uma sintaxe mais curta em comparação com expressões de função e não têm this, arguments, super, ou new.target. Arrow functions são sempre anônimas.

Dois fatores influenciaram a introdução de funções de seta: funções mais curtas e o não-vinculante this.

EXERCÍCIOS

1. Implemente uma função que recebe um número e verifica se ele é par ou ímpar usando uma estrutura de controle if/else.
2. Implemente uma função que recebe um array de números e retorna a soma desses números usando uma estrutura de laço for.
3. Implemente uma função que recebe um array de strings e retorna uma nova array contendo apenas as strings que possuem mais de 5 caracteres.
4. Faça um objeto chamado ListaDeCompras, e dentro desse objeto crie uma propriedade chamada produtos, e dentro dela coloque os produtos, e seus respectivos valores, e utilizando uma dos recursos mostrados anteriormente, faça a soma total desses valores.
5. Implemente uma função chamada somaQuadradosPares que recebe um array de números inteiros como entrada e retorna a soma dos quadrados dos números pares do array. ***Utilize como entrada o array [1, 2, 3, 4, 5, 6] e Implemente utilizando arrow function***

FUNÇÕES ASSÍNCRONAS - ASYNC && AWAIT

- ***Função Assíncrona***

A função assíncrona no JavaScript é uma função que permite que outras partes do código sejam executadas, enquanto outra operação está em andamento.

Funções assíncronas são aquelas que acessam ou buscam algum tipo de recurso em um dispositivo externo, como por exemplo um banco de dados, nesse tipo de função precisamos que o nosso código espere que a resposta esteja disponível antes de executar a ação seguinte.

FUNÇÕES ASSÍNCRONAS - ASYNC && AWAIT

- ***Função Assíncrona != Função Síncrona***

A principal diferença entre uma função síncrona e uma função assíncrona é a forma como elas lidam com a execução de tarefas que podem levar algum tempo para serem concluídas.

Uma função síncrona é executada de forma sequencial, uma linha de código após a outra, e o controle não é devolvido para a próxima linha de código até que a linha atual seja concluída. Isso significa que se uma função síncrona executar uma tarefa que leva muito tempo, a execução do programa inteiro ficará presa até que essa tarefa seja concluída.

A função assíncrona é executada de forma não sequencial.

FUNÇÕES ASSÍNCRONAS - ASYNC && AWAIT

- ***FUNÇÃO SÍNCRONA***

```
function Soma(x,y){
  return x + y;
}
console.log(Soma(2,4)); // 6
```

- O exemplo acima é de uma função síncrona, ou seja, que as etapas estão sendo executadas sequencialmente.

FUNÇÕES ASSÍNCRONAS - ASYNC && AWAIT

- ***Sintaxe - Função Assíncrona***

  - Declaração com a palavra chave “async”, antes da função, ela é usada para marcar que aquela função é assíncrona.
  - Palavra chave “await” para esperar que a operação assíncrona seja concluída, para seguir para o próximo passo do código. O “await” só pode ser usado dentro de uma função “async”.

```
async function NomeDaFuncao(){
  await acao;
}
```

FUNÇÕES ASSÍNCRONAS - ASYNC && AWAIT

- ***Função assíncrona***

- O exemplo ao lado, é de uma função assíncrona, nesse tipo de função precisamos que o nosso código espere que a resposta esteja disponível antes de executar a ação seguinte.

```
function calcula(a, b) {
  return (a + b);
}

async function display() {
  console.log("INICIO");
  const result = await calcula(2, 3);
  console.log(result);
  console.log("FIM")
}

display()
```

FUNÇÕES ASSÍNCRONAS - ASYNC && AWAIT

- ***Função assíncrona com Promise***
- ***Promise***

- Uma promise é um objeto JavaScript, que representa um valor que pode estar disponível agora, no futuro, ou nunca.

```
function PrimeiraFuncao(){
  return new Promise((resolve)⇒{
    setTimeout(()⇒{
      console.log('Espera');
      resolve();
    }, 2000)
  })
}
async function SegundaFuncao(){
  console.log('Inicio');
  await PrimeiraFuncao();
  console.log('FIM')
}
SegundaFuncao();
```

FUNÇÕES ASSÍNCRONAS - ASYNC && AWAIT

- Função assíncrona com requisição de API

- A “api” é uma interface que permite que um aplicativo se comunique com outro aplicativo ou serviço para realizar determinada tarefa ou obter informações. Ex: Google Maps, PayPal, Google Drive.

```
async function getUser(userId) {
  let response = await fetch(`https://api.com/api/user/${userId}`);
  let userData = await response.json();
  return userData.name; //vai estar retornando o nome
 }
 getUser(1);
```

CALLBACK

- ***O que é uma callback ?***

Callback é uma função que é passada como argumento para outra função, e é executada após a conclusão de uma operação assíncrona, ou evento específico.

- ***Qual sua utilidade ?***

São extremamente utilizadas para lidar com tarefas assíncronas, como requisições de rede, animações ou qualquer outra operação que pode demorar para ser concluída.

CALLBACK - Callback Síncrona != Assíncrona

- A ***callback síncrona*** ela vai retornar um resultado direto, que não depende de outro processo ou serviço.
- A ***callback assíncrona*** é útil quando dependemos de um resultado para executar outra função, como esperar uma requisição ser concluída.

Exemplo Callback Síncrona:

```
function principal(argumento){
  console.log(`O resultado é de ${argumento}`);
}
function callback(a,b,callback){
  let op = a + b;
  callback(op);
}
callback(5,6,principal);
```

CALLBACK - Callback Assíncrona - Exemplo

Exemplo Callback Assíncrona:

```
async function obterDadosAssincronos(url, callback) {
  const resultados = await fetch(url);
  const dados = await resultados.json();
  callback(dados);
  });
}
function manipularDadosAssincronos(dados) {
  console.log(dados);
}
obterDadosAssincronos('https://minha.api.com/dados',
manipularDadosAssincronos);
```

CALLBACK - Exemplo && Exercicios

- *Repositório do Exemplo estará disponível no GitHub*

Exercícios Funções

1 - Faça uma função que verifica a idade, e retorne se a pessoa é maior ou menor de idade.

2 - Faça uma função chamada PalavrasComLetra, que recebe uma frase/palavra, e uma letra como parâmetro, e retorna a quantidade de vezes que a letra se repete naquela frase.

CALLBACK - Exemplo && Exercicios

Exercícios Callback

1 - Faça duas funções, uma multiplicando dois valores e outra somando dois valores. e faça uma terceira função apenas para exibir o resultado. Utilize callbacks para mostrar o resultado na função de exibir.

2 - Crie uma função que recebe um array de números, e uma função callback como argumento, e retorne um novo array com os números multiplicados por 2 usando a função de callback.

CALLBACK - Exemplo && Exercicios

Exercícios Callback

3 - Crie uma função "assíncrona" que esteja obtendo informações de uma requisição. Faça uma função que contenha um objeto, e uma função callback como parâmetros, após isso crie uma função que mostre no terminal cada informação passada.

Ex de Objeto

```
const usuário = {
    id: 1,
    nome: ‘Marcos’,
    email: ‘marcos@mail.com’
}
```

PROMISES - O que é uma Promise?

- Uma Promise é um objeto JavaScript que representa um valor que pode estar disponível agora, no futuro ou nunca. É usada para lidar com operações assíncronas, que podem demorar algum tempo para serem concluídas.
- As Promises têm três estados possíveis: pendente, realizada e rejeitada.

PROMISES - Qual sua funcionalidade ?

- Lidar com código assíncrono de uma forma mais simples e organizada.
- Evitar o uso de callbacks, que podem se tornar complicados e difíceis de manter em códigos grandes.
- Permitir encadear operações assíncronas em sequência e tratar o resultado em cada etapa do encadeamento.
- Lidar com erros de uma forma mais clara e organizada, utilizando o método catch para tratar erros de Promise rejeitadas.
- Tornar o código mais legível, organizado e fácil de manter.

ESTRUTURA DE UMA PROMISE

- Uma Promise é uma função que recebe dois parâmetros: resolve e reject.
- Resolve é uma função que é chamada quando a operação assíncrona é concluída com sucesso.
- Reject é uma função que é chamada quando ocorre algum erro na operação assíncrona.

```
const myPromise = new Promise((resolve, reject) => {
    // operação assíncrona aqui
    if (/* operação bem-sucedida */) {
      resolve(resultado);
    } else {
      reject(erro);
    }
  });

  myPromise.then(resultado => {
    // manipular o resultado aqui
  }).catch(erro => {
    // manipular o erro aqui
  });
```

ESTADOS DE UMA PROMISE

- Uma Promise tem três estados possíveis: pendente, realizada e rejeitada.
- O estado pendente é quando a Promise ainda não foi resolvida ou rejeitada.
- O estado realizada é quando a Promise foi resolvida com sucesso.
- O estado rejeitado é quando a Promise foi rejeitada devido a algum erro.

MÉTODOS THEN && CATCH

- O método then é chamado quando a Promise é resolvida com sucesso e recebe como parâmetro o valor passado para a função resolve.
- O método catch é chamado quando a Promise é rejeitada e recebe como parâmetro o erro passado para a função reject.

EXEMPLO DE USO DE UMA PROMISE

```
const promise = new Promise((resolve, reject) => {
    setTimeout(() => {
      const randomNumber = Math.random();
      if (randomNumber > 0.5) {
        resolve(randomNumber);
      } else {
        reject(new Error('Número menor que 0.5'));
      }
    }, 1000);
  });

  promise
    .then(result => console.log(result))
    .catch(error => console.error(error));
```

ENCADEAMENTO DE PROMISES && EXEMPLO

- É possível encadear várias Promises para executar várias operações assíncronas em ordem.
- O resultado de uma Promise pode ser passado para a próxima Promise usando o método then.

```
const promise1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(1);
  }, 1000);
});

const promise2 = promise1.then((result) => {
  console.log(result);
  return result + 1;
});

promise2.then((result) => console.log(result));
```

PROMISE USANDO API

EXERCÍCIOS

1- Faça uma função que calcule 2 números e utilizando uma promise faça uma comparação para saber se os valores passados são do tipo numérico, caso sejam imprima o resultado e caso não retorne um erro.

2 - ***Faça duas funções, uma multiplicando dois valores e outra somando dois valores. e faça uma terceira função apenas para exibir o resultado. Utilize callbacks e Promises para mostrar o resultado na função de exibir***