SUMÁRIO

```
//JavaScript Básico
//* Aula4 do comando console
//#obs ele cita como aula 3 no vídeo, na udemy está 4

//| diferenças ao apresentar uma string (texto)
//#nota a aspas duplas pode conter aspas simples e o aspas simples pode envolver aspas duplas
//#nota o com crase pode conter ambas as aspas e é usado para template string

console.log('Jean "Meira"');
console.log("Jean 'Meira'");
console.log(`Jean Meira`);

//| números
//#nota todos são tipo number, não muda de inteiro(int) para ponto flutuante(float)

console.log(15.85);
console.log(35);

//| Múltiplos valores

console.log(35, 15.85, 'Jean Meira de novo');

//* Aula 8 - comentários
//#nota# comentários são ignorados ao se executar (pela engine/motor)

// Isto é um comentário
console.log('Hello world'); // aqui temos outro comentário
//console.log('código não executado')

//#nota# códigos com (//) serão considerados comentários e não são executados
/*  isto é
    é um comentário
    que permite mais de uma linha,
    formando um comentário por bloco
    #importante# não deve se esquecer de fechar
*/

//* Aula 09 - navegador vs Node (HTML+JavaScript)

console.log('Este trecho será exibido no console do navegador, usando um arquivo .js')
```

```
//#nota# é possível criar uma pasta com todos scrips .js que serão usados pelo arquivo
//#nota# assim é possível devinir um lugar com blocos convenientes de arquivos .js

//* HTML A PARTIR DAQUI
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Minha primeira página HTML</title>

    <script>
        // um comentário
        console.log('olá mundo');

/*
#nota# não é uma boa prática colocar código javascrit nessa parte do    código,pois ao colocar nesse local gerará atraso no carregamento da página, devido ao fato de que ao encontrar um código o navegador tentará executar os scrips.
*/
    </script>
</head>

<body>

    <script src='index.js'>
        // console.log('Este trecho será exibido no console do navegador')

/* código JS pode ser colocada dentro da tag scrip, mas é melhor prática separar os arquivos, usando assim o apontamento do caminho (src) para o arquivo .js, da forma a seguir
            <script src='index.js'>
*/
        // Um melhor prática é adicionar os scripts no final do body
    </script>
</body>
</html>

//* Aula 10 - variáveis let e var
//#obs# aula 6 citada no vídeo

//#nota# variáveis podem ser atribuídas como let e var
```

```
//#nota# podendo ou não ser inicializadas com valor, se não receber valor
será considerado undefined
//#obs# le- se como variável do tipo let recebe o valor do tipo string
'João'.

let nome = 'João';          //? já inicializado na declaração

let nome2;                  //? declaração
nome2 = 'qualquer valor';   //? atribuição de um valor

console.log(nome, 'nasceu em 1984');
console.log('Em 2000', nome, 'conheceu Maria');
console.log(nome, 'caso-se com Maria em 2012');
console.log('Maria teve 1 filho com', nome, 'em 2015');
console.log('O filho de', nome, 'se chama Eduardo');

//#nota# var é mais antigo, o var permite redeclaração enquanto o let não,
 ambos pode serem reatribuidos
//#nota# recebendo assim outros valores
//#aviso#let não podem ser declaradas mais de uma vez enquanto var pode
var nome3 = 'Jean';
var nome3 = 'Guilherme';

console.log(nome3);

//#obs# A variável permite salvar valores que serão usados em múltiplas pa
rtes do código, além de trabalhar
//#obs# informações no decorrer do código e poder salvar informações vinda
s do usuário ou fontes externas.

//#importante# o uso de variáveis ou partes dinâmicas de código podem mais
 facilmente ser alteradas,
//#importante# o código exemplificado acima pode mudar o nome citados em t
odas as frases somente alterando
//#importante# a linha de declaração da variável nome, o que não seria pos
sível se fosse manualmente
//#importante# escrito em cada comando console.log();


//#nota# Não podemos criar variáveis com palavras reservadas ex -> let if
-- let let, e assim por diante;
//#nota# É recomendado que variáveis tenham nomes significativos ex -> let
 n = 'João';
//#nota# n é muito vago, podendo ser qualquer coisa
    //#obs# É bom que a variável tenha valor semântico.
```

```
//#nota# Não começar o nome de uma variável com um número;
//#nota# Variáveis em geral começam com letras minúsculas, (existem exceçõ
es);
//#nota# Não podem conter  espaços ou -, ex -> let nome cliente; let nome-
completo;
    //#obs# para variáveis com nomes múltiplos pode-se usar o padrão
camelCase ex -> let nomeCLiente.
//#nota# Variáveis são case-
sensitive, ou seja, nas variáveis letras maiúsculas e minúsculas diferem
//#nota# no resultado final;

//#importante#  NÃO UTILIZE VAR, UTILIZE LET

//* Aula 11 - Const

//* As mesmas regras de variáveis valem para constantes
//#nota# Não podemos criar constantes com palavras reservadas ex -> let if
 -- let let, e assim por diante;
//#nota# É recomendado que constantes tenham nomes significativos ex -> le
t n = 'João';
//#nota# n é muito vago, podendo ser qualquer coisa
    //#obs# É bom que a constante tenha valor semântico.
//#nota# Não começar o nome de uma constante com um número;
//#nota# constantes em geral começam com letras minúsculas, (existem exceç
ões);
//#nota# Não podem conter  espaços ou -, ex -> let nome cliente; let nome-
completo;
    //#obs# para constantes com nomes múltiplos pode-
se usar o padrão camelCase ex -> let nomeCLiente.
//#nota# constantes são case-
sensitive, ou seja, nas variáveis letras maiúsculas e minúsculas diferem
//#nota# no resultado final;
//#importante# Constantes não podem ser redeclaradas e nem tem novos valor
es atribuídos as mesmas.
    //#obs# não podem modificar seu valor
//#importante# devem ser inicializadas e declaradas ao mesmo tempo

const nome = 'João';

console.log(nome);

//#nota# É possível usar uma constante ou variável na declaração de outra.

const primeiroNumero = 5;
const primeiroNumeroStrg = '5';
```

```
const segundoNumero = 10;
const resultado = primeiroNumero * segundoNumero;
const resultadoDuplicado = 2* resultado;

console.log(resultado);                                    //? resposta es
perada -> 50
console.log(resultadoDuplicado);                           //? resposta es
perada -> 100

//* Verificando o tipo de uma const ou let com a função typeof();

console.log(primeiroNumero + segundoNumero);                //? resposta e
sperada -> 15
console.log(typeof(primeiroNumero + segundoNumero));        //? resposta e
sperada -> number
console.log(primeiroNumeroStrg + segundoNumero);            //? resposta e
sperada -> 510
console.log(typeof(primeiroNumeroStrg + segundoNumero));    //? resposta e
sperada -> string

//#nota# Ao receber uma string e um number o código concatena os valores e
m vez de somar,
//#nota# aprensentando a string 5 e o número 10 escritos em sequência.
//#nota# Porém, dessa forma o resultado passa a ser interpletado como uma
string.

//* Aula 15 - primeira diferença entre var e let
//#obs# no vídeo citado como aula 9

//? Var Aceita redeclaração

var nome ='Luiz';   //? Declaração
var nome = 'Otávio' //? Re-declaração

console.log(nome);  //? resultado esperado -> Otávio

//#aviso# Adendum não faça o demonstrado a seguir
nome1='Luiz';

//* Aula 16 - Tipos de dados primitivos

//| strings

const nome = 'Luiz';
const nome1 ="Luiz";
```

```
const nome2 = `Luiz`;

//| numbers

const num1 = 10;
const num2 = 10.5;

//| undfined
let nomeAluno               //? undefined -> não aponta para nenhum local
na memória

//| null
let sobernomeAluno = null   //? null -> não aponta para nenhum local na me
mória

//#nota# undefined != de null
//#nota# null é a indicação que foi escolhido que a variável não terá um v
alor, não aponta para nenhuma memória
//#nota# undefined é uma variável que não recebeu um valor.

//? boolean
const boolean = true;
const boolean2 = false;

//#nota# boolean assume valor true ou false, representando verdadeiro ou f
also, 1 ou 0;
//#obs# boolean tem peso lógico, ajuda em decisões do código. Muda o fluxo
 da aplicação, usando desvios condicionais.

//* Aula 17 - Operadores aritméticos, de atribuição e incremento.

//| Aritméticos
//_ adição          -> +
//_ subtração       -> -
//_ divisão         -> /
//_ multiplicação   -> *
//_ potenciaçã      -> **
//_ resto da divisão-> %

//#nota# a precedêndia das operações são realizadas conforme a matemática

//_ exemplo
const num1 =5;
const num2 =2;
const num3 =10;
```

```
console.log(num1 + num2 * num3);    //? resultado esperado -> 25

/*
    ! Ordem de precedência segue conforme indicado abaixo
    _ 1º ** (potenciação)
    _ 2º * (multiplicação) , / (divisão) e % (resto da divisão ou módulo d
a divisão)
    _ 3º + (adição) e - (subtração)
*/

//#obs# é possível alterar a ordem de precedência com o uso parênteses ()
//_ exemplo

console.log((num1 + num2) * num3);    //? resultado esperado -> 70

//| Incremento e decremento
//_  ++ soma um no valor (incrementa)
//_  -- subtraí um no valor (decrementa)

//_ exemplo (funcionam para incremento e decremento)

let contador =1;
contador++;
console.log(contador);

console.log(contador++)             //? realiza a ação e depois incrementa
 (pós incremento)
    //#obs# evite usar o modelo acima, com pós incremento junto com a funç
ão que irá usar de imediato, pois pode causar bugs.

console.log(++contador)             //? incrementa e depois realiza a ação
 (pré incremento)

//| Operadores de atribuição
//_ incremento de mais de uma unidade

const passo =2;
let contador1 =0;

contador1 = contador1 + passo;
console.log(contador1);

//_ De modo simplificado
```

```
//#obs# é possível usar com incremento, decremento, multiplicação, divição
 e potenciação

contador1 += passo;          //? o mesmo que digitar -> contador1 = contado
r1 + passo;
console.log(contador1);

/*
    ! cuidado com usar contas ou atribuições com variáveis, ao se usar com
 variáveis com tipagem diferentes de números
    ! pode se obter resultados adversos e inesperados.
    ! podendo até obter resultados NaN -> not a number
*/

//#importante# Quando for possível converta as variáveis para o tipo desej
ado para garantir o funcionamento

//_ Exemplo
const numTest = parseInt('5');

console.log(typeof(numTest));        //? resultado esperado -> number

//_ parseInt()      -> converte para inteiro, sem números após a vírgula
//_ partseFloat()   -> converte para float, números com valores após a vír
gula
//_ Nuber()         -> converte para número, sem distinção

//* Aula 18 - Alert, confirm e Prompt (Navegador)

//| Alert
alert('Mensagem');

//#nota# alert é um método do objeto window
//#obs# alert é um atalho para window.alert();
//_ o retorno é undefined, ou seja, não retorna valor algum.

//| Confirm
window.confirm('Deseja realmente apagar?');

//| prompt
//? sempre vai te retornar uma string
window.prompt('Digite o seu nome.');

//| é possível capturar o retorno da função
```

```
const confirma = confirm('Realmente deseja apagar?');
console.log('confirma tem valor:', confirma);

let num1 = prompt('Digite um número');
alert(`Você digitou: ${num1}`);
console.log(`Você digitou: ${num1}`);

//| A partir daqui é HTML
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Aula 18</title>
</head>
<body>
    <script src="./script.js"></script>
</body>
</html>

//* Aula 21 - Mais sobre Strings


let umaString = "Um \"texto\""; //? a barra invertida é um caractere de es
cape.
//_ dessa forma é possível inserir aspas duplas em uma string declarada co
m aspas duplas

console.log(umaString);

umaString = "Um \\texto";        //? usando duas barras é possível fazer u
ma barra aparecer no resultado final

console.log(umaString);

//#nota# uma string é indexável. Cada caractere tem um índice. Começando e
m 0 (zero);
umaString = "Um texto";
console.log("A quinta letra da váriável umaString é: "+umaString[4]);
console.log(umaString.charAt(4));   //? usando o método charAt(); para rea
lizar o mesmo.
console.log(umaString.charCodeAt(4));   //? retorna o código da tabela ASC
II.

//| função concat concatena os textos, como o + ou uso de templateString
```

```
console.log(umaString.concat(' em', ' um', ' lindo dia'));
console.log(umaString + ' em' + ' um'+ ' lindo dia');
console.log(`${umaString} em um lindo dia`);

//| Pesquisando índice que começa uma palavra
console.log(umaString.indexOf('texto'));
//#nota# retorna o índice se achar e -1 se não achar.

//#obs# é possível enviar dois parâmetros, então o segundo será de onde a
pesquisa começa
console.log(umaString.indexOf('um', 3));    //? retorno -
1 pois não há um após o índice 3.

console.log(umaString.lastIndexOf('o'));    //? começa a procura do final
para o começo.
//#obs# pode dar diferença se mandar o parâmetos de início.

console.log(umaString.search(/x/));         //? similar ao indexOf, mas ac
eita expressões regulares.
//#obs# e em realidade pode ter funcionamentos um pouco mais amplos.

//| Replace
console.log(umaString.replace('Um', 'outra')); //? substitui uma palavra p
or outra

umaString = 'O rato roeu a roupa do rei de roma';
console.log(umaString.replace(/r/, '#'));     //? substitui somente o 1º r
console.log(umaString.replace(/r/g, '#'));    //? substitui todos os r

//| Checando o tamanho

console.log(umaString.length);
//           012345
umaString = 'O rato';
console.log(umaString.length);      //? conta como 0 a 5, ou seja 6

//| dividindo uma string

umaString = 'O rato roeu a roupa do rei de roma.';

console.log(umaString.slice(2, 6));
//#nota# Indica a posição de início e final, sendo que o final não é
contado.

console.log(umaString.slice(2));
```

```
//#nota# Se não receber o segundo parametro, conta como do início indicado
 até o final.

console.log(umaString.length-3);        //? resultado esperado -> 32
console.log(umaString.slice(-3));
console.log(umaString.slice(32));
/*
    _ Usar número negativo é o mesmo que adotar o início como o tamanho to
tal do texto
    _ menos o valor indicado. Sendo assim, no exemplo acima é printado ape
nas os 3
    _ últimos caracteres. E -3 é o mesmo que indicar o início de 32.
    _ Também é possível indicar começo e final com números negativos.
*/

//? também é possível usando outro comando.
console.log(umaString.slice(-5, -1));
console.log(umaString.substring(umaString.length - 5, umaString.length -
1));

//? Baseado em algum caractere.

console.log(umaString.split(' '));
console.log(umaString.split('r'));

console.log(umaString.split(' ', 2));       //? é possível limitar o númer
o de resultados.

//| representar em maiúsculo ou minúsculo
console.log(umaString.toUpperCase());
console.log(umaString.toLocaleLowerCase());

//* Aula 23 - Mais sobre números

let num1 =1;    // numbe
let num2 =2.5;  // number

console.log(num1 + num2);
console.log(num1.toString()+ ' e ' + num2.toString());
console.log(typeof num1);

//_ É realizado a contenação, mas o tipo do número permanece inalterável.

//? para converter -> num1 = num1.toString();
```

```
num1 = 15;
console.log(num1.toString(2));  //? representa o número em base binária.

num1 = 10.872158137674;
console.log(num1.toFixed(2));   //? limita o número de casas após a vírgul
a que será mostrado
//#obs# o número é arredondado.
//#aviso# recomenda-se fazer todos os
cálculos e apenas usar esse artifício ao exibir o resultado.

console.log(Number.isInteger(num1));    //? o número é um tipo inteiro ?
//_ Retorna verdadeiro ou falso.

let temp = num1 * 'olá';
console.log(temp, Number.isNaN(temp));  //? a conta é inválida, retornando
 um NaN;

//! padrão IEEE 754-2008 -> para imprecisão

num1 = 0.7;
num2 =0.1;

console.log(num1+num2);     //? resultado -> 0.79999...

//? o mesmo que num1 + num2;
num1 += num2;  //? 0.8
num1 += num2;  //? 0.9
num1 += num2;  //? 1.0

console.log(num1);  //? mas da o resultado 0.9999...

num1 = num1.toFixed(2); //? fixa apenas duas casas;
console.log(num1);      //? o resultado da 1.00, parecendo correto.
console.log(Number.isInteger(num1), Number.isInteger(1.00));    //? aqui n
ota-se a diferença.

//| Para corrigir o problema devemos...
num1 = 0.7;
num2 =0.1;

num1 += num2;  //? 0.8
num1 += num2;  //? 0.9
num1 += num2;  //? 1.0
```

```
num1 = Number(num1.toFixed(2));     //? fixar as casas garantindo que será
 um número

console.log(num1);
console.log(Number.isInteger(num1));    //? só será inteiro se as casas ap
ós o ponto forem .00

//| outra forma seria trabalhar com contas.
//_ não somando números com vírgulas
num1 = 0.7;
num2 =0.1;

num1 = ((num1*100)+(num2*100))/100;
console.log(num1);  //? 0.8

num1 = ((num1*100)+(num2*100))/100;
num1 = ((num1*100)+(num2*100))/100;

console.log(num1);
console.log(Number.isInteger(num1));

//* Aula 24 - Objeto Math

/*
    _ Math é um objeto embutido que tem propriedades
    _ e métodos para constantes e funções matemáticas.
    _ Não é um objeto de função.
    _ Todas as propriedades e métodos de Math são estáticas.
*/

let num1 = 9.54578;

//| arredondamento
//_ Para baixo
let num2 = Math.floor(num1); //? arredondando para baixo.
console.log(num2);  //? resultado esperado -> 9;

//_ Para cima
num2 = Math.ceil(num1); //? arredonda pra cima.
console.log(num2);  //? resultado esperado ->  10;

//_ Para o que estiver mais próximo
num2 = Math.round(num1); //? arredonda para o mais próximo
console.log(num2);  //? resultado esperado -> 10
```

```
num1 = 9.44578;
num2 = Math.round(num1);
console.log(num2);  //? resultado esperado -> 9

num1 = 9.50;
num2 = Math.round(num1);
console.log(num2);  //? resultado esperado -> 10
//#obs# Se o número estiver na metade será considerado para cima

//| O maior e menor número de uma sequência
console.log(Math.max(1,2,3,4,5,-10,-
50,1500,9,8,7,6));  //? resultado esperado -> 1500
console.log(Math.min(1,2,3,4,5,-10,-
50,1500,9,8,7,6));  //? resultado esperado -> -50

//| gerando um número aleatório entre 0 e 1, sem incluir o 1;
console.log(Math.random());

//_ é possível trabalhar para gerar outros resultados
const aleatorio = Math.round(Math.random() * (10-5) + 5);
console.log(aleatorio); //? aleatório entre 5 e 10.

//| o Math tem constantes
console.log(Math.PI);   //? o pi = 3.14158...
console.log(Math.E)     //? o número de Euler = 2,71828...

//| Potenciação
console.log(Math.pow(2, 10));   //? 2 elevado a 10
//#obs# pode-se dizer que é o mesmo que
console.log(2 ** 10);      //? 2 elevado a 10

//| Raiz
num1 = 9;
console.log(num1 ** (1/2));
console.log(Math.sqrt(num1));

//! Cuidado com divisões por 0, pois no Js isso é possível.
//? ex:
console.log(100 / 0); //? o número retornará infinity

//* Aula 26 - Arrays (básico)

const alunos = ['Jean', 'Maria', 'João'];
console.log(alunos)
//_ tentar organizar com um tipo só de dados dentro, mas não é regra
```

```
//_ o JavaScript permite a adição, mas não é boa prática de programação

//_ Arrays são indexados por elementos, começando no valor 0.
console.log(alunos[0])

//| é possível editar ou adicionar
alunos[1] = 'Roberta';
alunos[3] = 'Luiza';
console.log(alunos);

// _ Se o Array tem um elemento na posição ele altera, se não ele adiciona
.

//| Se quiser adicionar no último é possível usar conforme a seguir.
console.log(alunos.length)
alunos[alunos.length] = 'Fábio';
alunos[alunos.length] = 'Luana';
alunos.push('Otávio');
console.log(alunos);

//| Se quiser adicionar no começo...

alunos.unshift('Luiza');
console.log(alunos);

//| Removendo
alunos.pop();   //? remove o último
//_ é possível salvar o elemento removido
const removido = alunos.pop();
console.log(alunos, removido);

alunos.shift(); //? remove do começo
console.log(alunos);

delete alunos[2];   //? remove o índice específico
console.log(alunos, alunos[2])
//_ O delete não muda os índices, o que ocorre com pop e shift. O local ap
agado fica como undefined para o caso do delete;

alunos[2] = 'Carlos';
//| Dividindo um Array
console.log(alunos.slice(0, 3));
console.log(alunos.slice(0, -1));

//_ Arrays retornar como objetos, pois são como obejtos indexados
```

```
console.log(typeof alunos);
//? é possível checar se é um array
console.log(alunos instanceof Array);

//#Aviso# geralmente é melhor adicionar elementos no final do array, para que não se tenha que mudar todas as possições.
//#Aviso# isso torna a performance do programa ruim.

//* Aula 28 - Funções (básico)

//#nota# funções executam ações, podendo ou não retornar algo

function saudacao(){
    console.log('Bom dia');
};

saudacao();

//| funções que recebem parâmetros

function saudacao1(nome){
    console.log(`Bom dia ${nome}`);
};

saudacao1('Jean');

//#nota# função são reutilizáveis
saudacao1('Fulano');

const variavel = saudacao1('Fulano');
console.log(variavel);
//#obs# toda função por padrão retorna undefied, e por isso é salvo undefined na variavel.

//| Funções com retorno

function saudacao2(nome){
    console.log(`Bom dia ${nome}`);
    return 123456
};

const retorno = saudacao2('Jean');
console.log(retorno);
```

```
//#aviso# Esperasse que o retorno da função seja algo semantico com o nome
, e não que seja algo bem diferente.

function saudacao3(nome){
    return `Bom dia ${nome} pelo retorno`
};

const retorno2 = saudacao3('Jean');
console.log(retorno2);

//| funções com mais uso e que permitem repricar o seu uso.

function soma(x,y){
    const resultado = x +y;
    return resultado;
}

console.log(soma(2,2));
console.log(soma(3,1));
console.log(soma(5,10));

// console.log(soma(2,2), resultado); //#nota# daria erro pois o resultado
 faz parte do escopo local da função.

const resultado = soma(2,2);    //? é possível declarar uma const resultad
o, pois a que está dentro da função está isolada.

//! Quando a função chegar em um return sairá da função e aplica o retorno
, tudo após um return será ignorado.

function soma1(x,y){
    const resultado = x +y;
    return resultado;
    console.log('olá mundo');
}

console.log(soma1(5,10));

//| Valores faltando

console.log(soma()); //? retorna NaN
//#obs# será NaN mesmo se faltar só um

//_ É possível tratar atribuindo um valor inicial para os argumentos da fu
nção
```

```
function soma2(x=1,y=1){
    const resultado = x +y;
    return resultado;
}

console.log(soma2());
console.log(soma2(5,10));

//#obs# caso não receba valores, serão usados os pré-
definidos, caso receba será adotado o recebido.

//| Outros modos de criar funções

//_ Funções anônimas -> em variáveis.
//#obs# precisa do ponto e vírgula obrigatoriamente no final.

const raiz = function(n){
    return (n ** 0.5);
};

console.log(raiz(9));
console.log(raiz(16));
console.log(raiz(25));

//_ Arrow function

const raizArrow = (n) => {
    return ( n ** 0.5);
};

console.log('Arrow', raizArrow(9));

//_ Se tiver apenas uma linha de retorno é possível simplificar ainda mais
.

const raizArrow1 = (n) =>  n ** 0.5;
console.log('Arrow resumida',raizArrow1(9));

//#Aviso# funções são basicamente iguais, mesmo com as declarações diferen
tes.
//#Aviso# Quando entrar em this terão diferenças que serão expostas
//#importante# Não é uma boa prática criar funções que executem diversas a
ções.

//* Aula 29 - Objetos (básico)
```

```
//#Nota# é possível alteral os valores dos elementos de objetos e arrays.
//#Nota# mas não é possível reatribuir o valor ou mudar o tipo.
//#Nota# ao mudar elementos internos não há alterações pra onde é apontato
 na memória.

//| declaração

const pessoa1 = {
    nome: 'Luiz',
    sobrenome: 'Miranda',
    idade: 25
};

console.log(pessoa1.nome);
console.log(pessoa1.sobrenome);
//#Nota# da pra criar atributos que são como variáveis, mas estão internos
 ao objeto.
//#Nota# os atribuitos são separados por uma , no final de cada linha
//#Nota# e num objeto utiliza par chave e valor para cada atributo, que sã
o separados por :

//| criar por funções -> function factory
// fábrica de objetos.

function criaPessoa(nome, sobrenome, idade){    //? Isso são parâmetros de
 uma função
    return{ nome, sobrenome, idade};
    /*  É o mesmo que citar como nome: nome e assim por diante,
        Pois quando a chave e o valor possuem o mesmo nome é possível
        omitir o : valor
    */
}

const pessoa2 = criaPessoa('Jean','Meira','23');    //? Aqui são argumento
s que serão passados para os parâmetros.
console.log(pessoa2);

//#Nota# argumentos são os valores que são passados para o parâmetro.

//| Criar método no objeto

const pessoa3 = {
    nome: 'Jean',
    sobrenome: 'Meira',
```

```
    idade: 23,
    fala(){
        console.log(`${this.nome} ${this.sobrenome} está falando oi...
        A minha idade é ${this.idade}`)
    },
    incrementaIdade(){
        this.idade++;
    }
}

pessoa3.fala();
pessoa3.incrementaIdade();
pessoa3.fala();

//#Nota# this referencia o contexto da função, mas será dado mais detalhes
 a frente.

//* Aula 30 - Valores primitivos e por referência

/*
    _ Primitivos (imutáveis) - string, nuber, boolean, undefined, null
    _ também existem os bigint e symbol
    _ dado é o valor, a variável é somento uma caixa que contém o valor.
*/

let nome = 'Luiz';
nome = 'Otávio';
console.log(nome);
//#Obs# apesar de trocar o que está escrito, não se altera o dado / valor
primitivo

let a = 'A';
let b = a; // é feito uma cópia
console.log(a, b);

a ='Outra coisa';
console.log(a, b);
//_ b não se altera por ser uma cópia de a, então alterando a o b não é af
etado.

/*
    _ Por Referência (mutável) - array, object , function
    ! são passados por referência
    _ O que na verdade quando são atribuidos a outras variáveis,
    _ significa que vão apontar para a mesma referência, o mesmo local
```

```
    _ na memória.
*/

let c = [1, 2, 3];
let d=c;    // d vai apontar para o mesmo lugar na memória
console.log(c,d);

c.push(4);
console.log(c,d);

//_ b é afetado, pois o locar na memória é o mesmo, então mudando por a ou
 por b, afeta ambos
//? exemplo
d.pop();
console.log(c,d);

//| para cópiar o valor e mudar o local apontado na memória

let e = [1, 2, 3];
let f= [...e];
e.push(4);
console.log(e,f);

//Lógica de programação
//* Aula 33 - Operadores de comparação

/*
    _ >     maior que
    _ >=    maior que ou igual a
    _ <     menor que
    _ <=    menor que ou igual a
    _ ==    igualdade   (checa valor)
    _ ===   igualdade estrita (checa valor e tipo)
    _ !=    diferente   (checa valor)
    _ !==   diferente estrito   (checa valor e tipo)

*/

console.log(10>5);
//? É possível salvar o valor em variáveis e constantes
const comp = 10 > 5;
console.log(comp);

//_ A funcionalidade é igual as usadas em matemática. Os estritos tem um c
onceito a mais.
```

```
//| Com variáveis

let num1 = 10;
let num2 = 11;

console.log(num1 <= num2);

//| Comparação e comparação estrita

num1 = 10;
num2 = 10;
console.log(num1 == num2);
num2 = '10';
console.log(num1 == num2);
console.log(num1 === num2);

/*
    _ A comparação normal somente compara o valor contido, enquanto
    _ a comparação estrita compara o valor e o tipo da variável ou
    _ dado em questão. Isso é válido para a igualdade e desigualdade.
    ! O uso do modo estrito é altamente recomendado para evitar
    ! Comportamentos indesejados na execução do código.
    ! EVITE O USO DA IGUALDADE E DESIGUALDADE COMUNS
*/

//* Aula 34 - Operadores Lógicos


/*
    _ Operadores lógicos
    _ && -> and -> e
    _ || -> or -> ou
    _ ! -> not -> não
*/

//| && (and)
//_ Para ser verdadeiro todos as expressões precisam ser verdadeira
console.log(true && true);
console.log(true && true && true && true);

//_ É possível salvar o valor em uma variável ou constante
const expressaoAnd = (true && true && true && true);
console.log(expressaoAnd);
```

```
//_ Se uma for falsa o resultado já é falso
console.log(true && false);
console.log(true && true && false && true);

//| || (or)
//_ Para ser verdadeiro pelo menos uma das expressões precisam ser verdade
ira
console.log(true || true);
console.log(true || false);

//_ É possível salvar o valor em uma variável ou constante
const expressaoOr = (true || true);
console.log(expressaoOr);

//_ Se uma for falsa se todas as expressões forem falsas
console.log(false || false);
console.log(false || false || false || false);
```

```
//| ! (not)
//_ Sai o oposto do que entrou
console.log("negação de true:", !true);
console.log("negação de dupla de true:", !!true);

//* Aula 35 - Avaliação de curto-circuito (short-circuit)

/*
    _ && -> false && true -> false : retorna "o valor" ao achar uma falsa

    | FALSY (VALORES QUE PODEM SER AVALIADOS COMO FALÇO)
    _ false
    _ 0
    _ '' "" `` (string vazias)
    _ null / undefined
    _ NaN

    #nota# Qualquer valor diferente dos presentes no FALSY avalia verdadei
ro
*/

//? exemplo
/*
    _ Ao se deparar com um FALSY o valor do mesmo é retornado, se em uma
    _ comparação não tiver nenhum FALSY a mesma irá retornar o último valo
r lido.
```

```
    _ Isso possibilita fazer uma redução nos circuitos para alguns casos, tornando o
    _ código mais limpo e performático.
*/
console.log("Jean" && 0);   //? retorna 0;

/*
    _ A seguir tem um caso de curto circuito, onde o código é omitido em partes
    _ Porque devido as propriedades de falsy o código é perfeitamente válido.
    _ Será testado a condição e ao perceber se vaiExecutar é verdadeiro ou falso
    _ já retornará o resultado.
*/
function falaOi (){
    return 'Oi';
}

const vaiExecutar = false;

console.log(vaiExecutar && vaiExecutar);

/*
    _ O || (or) tem comportamento ao contrário
    _ || -> false && true -> true : retorna "o valor" ao achar uma true

    | FALSY (VALORES QUE PODEM SER AVALIADOS COMO FALÇO)
    _ false
    _ 0
    _ '' "" `` (string vazias)
    _ null / undefined
    _ NaN

    #nota# Qualquer valor diferente dos presentes no FALSY  avalia verdadeiro
*/

//* Aula 36 - if, else if e else (1)

const hora = 12;

if (hora >= 0 && hora < 12){
    console.log('Bom dia');
} else if(hora >= 12 && hora < 18){
```

```
    console.log('Boa tarde');
} else if (hora >= 18 && hora <=23){
    console.log('Boa noite');
}else {
    console.log('Olá');
}

/*
    _ If é o primeiro teste de condição, que ser verdadeiro, executa o cód
igo contido nas chaves.
    _ if pode ser usado sozinho.
    _ Se a condição for falsa pode ter outras condições diversas com o els
e if, em qualquer quantidade.
    _ O else if precisa ver seguido de um if, não podendo ser usado sozinh
o.
    _ Se todas as condições forém falsas será executado o conteúdo do else
.
    _ O else vem no final e só pode ser usado um.
*/

//* Aula 37 - if, else if e else (2)

//_ No uso de if e else, os blocos são interdependentes. De modo direto
//_ Se um ocorrer o outro não ocorre.

//_ No caso de usar if, else if (em quantidade desejada) e else
//_ O else depende do if e de todos else if, se não houver verdadeiros exe
cuta o else;

//_ O bloco de if, else if e else para quando encontra um elemento verdade
iro.

//_ Cada bloco com if é independente de outro if.
//* Aula 41 - Operação ternária

//#nota# são o conjunto de ? :
//#nota# sendo (condição) ? (valor para verdadeiro) : (valor para falso)

const pontuacaoUsuario = 999;

if(pontuacaoUsuario >= 1000){
    console.log('Usuário vip');
}else{
    console.log('Usuário normal');
}
```

```
const nvUsuario = pontuacaoUsuario >= 1000 ? 'Usuário vip' : 'Usuário norm
al';
console.log(nvUsuario);

//#obs# funciona como um if e else, apenas um teste com retorno se verdaei
ro ou falso.

//* Aula 42 - Objeto Date

//| formato em branco -> momento atual
const data = new Date();    //? é contada em milésimos de segundos
console.log(data.toString());

//| fomato valor
const data0 = new Date(0); //? 01/01/1970 Timestamp unix ou época unix
//#nota# esse é o marco zero da era unix, e para datas posteriores deve se
r valores positivos
//#obs# e para anteriores negativos

console.log(data0.toString()); //? mas o resultado é:
    //? Wed Dec 31 1969 21:00:00 GMT-0300 (Horário Padrão de Brasília)
    //#obs# isso se deve ao fuso horário, se for somado 3 passar a ser a d
ata 01/01/1970

//| formato ano,mes,dia,hora,minuto,segundo,milissegundo

const dataEscolhida = new Date(2019, 3, 20, 15, 14, 27, 500); //?ano, mes,
 dia, hora, min, seg, milésimo
//#nota# O mês se conta do 0 ao 11
//_ os milesimos vão até 999, se colocar 1000 ou mais ele passa 1 para a d
iante, corrigindo.
//_ assim como segundos de 0 a 59 e assim por diante.
//_ Se omitir algo ele aceitará como zero
//_ Não é possível omitir  um valor e colocar o mais a direita dele.
//_ ex: se colocar o ano, omitir o mês e colocar o dia, os dias serão cont
ados como meses.
//_ mas é possível omitir os mais a direita, até no máximo os meses, sendo
 necessário indicar o ano.
console.log(dataEscolhida.toString()) //? Sat Apr 20 2019 15:14:27 GMT-
0300 (Horário Padrão de Brasília)

//| formato datastring

const dataString = new Date('2019-04-20 20:20:59.100');
```

```
const dataString1 = new Date('2019-04-20T10:10:59.599');
console.log(dataString.toString());
console.log(dataString1.toString());

//|obter dia
console.log('Dia', data.getDate()); //? -> dia de domingo (0) a sábado (6)
console.log('Mês', data.getMonth()+1);
console.log('Ano', data.getFullYear());
console.log('Hora', data.getHours());
console.log('Min', data.getMinutes());
console.log('Seg', data.getSeconds());
console.log('ms', data.getMilliseconds());
console.log('Dia semana', data.getDay());

//| obter os milésimos de segundo de agora sem new Date();
console.log(Date.now()); //? -> retorna em milésimos.

//| formatando data

function zeroAEsquerda(num){
    return num >= 10 ? num : `0${num}`
}

function formataData(data){
    const date = zeroAEsquerda(data.getDate());
    const month = zeroAEsquerda(data.getMonth()+1);
    const year = zeroAEsquerda(data.getFullYear());
    const hours = zeroAEsquerda(data.getHours());
    const minutes = zeroAEsquerda(data.getMinutes());
    const seconds = zeroAEsquerda(data.getSeconds());

    return `${date}/${month}/${year} ${hours}:${minutes}:${seconds}`
}

const dataBrasil = new Date();
const dataAgora = formataData(dataBrasil);
console.log(dataAgora);
//* Aula 43 - Switch/Case

//_ estrutura condicional switch case

const data = new Date('1987-04-21 00:00:00');
const diaSemana = data.getDay();
let diaSemanaTexto;
```

```
if (diaSemana === 0) {
    diaSemanaTexto = 'Domingo';
} else if (diaSemana === 1) {
    diaSemanaTexto = 'Segunda';
} else if (diaSemana === 2) {
    diaSemanaTexto = 'Terça';
} else if (diaSemana === 3) {
    diaSemanaTexto = 'Quarta';
} else if (diaSemana === 4) {
    diaSemanaTexto = 'Quinta';
} else if (diaSemana === 5) {
    diaSemanaTexto = 'Sexta';
} else if (diaSemana === 6) {
    diaSemanaTexto = 'Sábado';
} else {
    console.log('Erro');
}

console.log(diaSemanaTexto);

const dataSC = new Date('1987-04-22 00:00:00');
const diaSemanaSC = dataSC.getDay();
let diaSemanaTextoSC;

switch (diaSemanaSC) {
    case 0:
        diaSemanaTextoSC = 'Domingo';
        break;
    case 1:
        diaSemanaTextoSC = 'Segunda';
        break;
    case 2:
        diaSemanaTextoSC = 'Terça';
        break;
    case 3:
        diaSemanaTextoSC = 'Quarta';
        break;
    case 4:
        diaSemanaTextoSC = 'Quinta';
        break;
    case 5:
        diaSemanaTextoSC = 'Sexta';
        break;
    case 6:
        diaSemanaTextoSC = 'Sábado';
```

```
        break;
    default:
        console.log('Erro');
}

console.log(diaSemanaTextoSC);

//| dentre de função

function getDiaSemanaTexto(diaSemana) {
    let diaSemanaTextoSC

    switch (diaSemanaSC) {
        case 0:
            diaSemanaTextoSC = 'Domingo';
            return diaSemanaTextoSC;
        case 1:
            diaSemanaTextoSC = 'Segunda';
            return diaSemanaTextoSC;
        case 2:
            diaSemanaTextoSC = 'Terça';
            return diaSemanaTextoSC;
        case 3:
            diaSemanaTextoSC = 'Quarta';
            return diaSemanaTextoSC;
        case 4:
            diaSemanaTextoSC = 'Quinta';
            return diaSemanaTextoSC;
        case 5:
            diaSemanaTextoSC = 'Sexta';
            return diaSemanaTextoSC;
        case 6:
            diaSemanaTextoSC = 'Sábado';
            return diaSemanaTextoSC;
        default:
            return console.log('Erro');
    }
}

console.log(getDiaSemanaTexto(diaSemanaSC));

// ou

const diaFunction = getDiaSemanaTexto(diaSemanaSC);
console.log(diaFunction);
```

```
//* Aula 45 - Mais diferenças entre var, let e const

const verdadeira = true;

let nome = 'Jean';
var nome2 = 'Carlos';
console.log(nome, 'e', nome2);

//_ No mesmo escopo não se redeclara let, mas var pode.
//_ let tem escopo de bloco {... bloco}
//_ var só tem escopo de função

if(verdadeira){
    let nome = 'outro nome';
    var nome2 = 'nome redeclarado';
//_ pode declarar com o mesmo nome, pois no bloco let tem outro escopo, po
rtanto dentro é outra variável
    console.log(nome, 'e', nome2);
}
//_ se o bloco se depara com uma variável let, tentará buscar no bloco, e
irá voltando até o escopo global.
//_ já o var, mesmo se for usado em blocos, estará redeclarando

console.log(nome, 'e', nome2);

/*
    _ funções tem blocos especiais que são isolados. Suas informações não
podem ser vistas de fora.
    _ mesmo var se torna isolada, mas a função pode acessar variáveis e da
dos de fora.
*/

//! hoisting -> elevação

console.log(hoistingVar);   //? resultado ->  undefined

var hoistingVar = 1;
//_ aqui ele eleva a declaração, mas sem o valor definido dela.

console.log(hoistingLet);   //? erro
//let hoistingLet = 1;      //? resultado -> is not defined
//_ não ocorre a elevação de declaração nesse caso
//* Aula 46 - Atribuição via desestruturação (Arrays)
```

```
//? ex:
let a = 'A';    //B
let b = 'B';    //C
let c = 'C';    //A

const letras = [b, c, a];
[a, b, c]=letras;

console.log(a, b, c);

//| mais detalhes

const num = [1, 2, 3, 4, 5, 6, 7, 8, 9];
//const primeiroNum = num[0];

const [primeiro, , terceiro, ...resto] = num;
console.log(primeiro, terceiro, resto);
/*
    #nota#
    _ A desestruturação pega como se fosse as possições, comendando no pri
meiro indíce.
    _ e atribui a cada elemento contido no array a esquerda.
    _ se quiser pular valores precisa deixar espaços vazios.
    _ O operador ...(nomeDaVarivel) faz com que tudo que não foi atribuind
o ainda
    _ seja salvo na variável.
    _ Todos os elementos citados no array a esquerda terão o tipo que foi
usado para declarar,
    _ podendo ser let ou const (no caso todos são const)
    _ o operador ... tem o nome de rest (rest operator),
    _ se usar em contexto diferente pode se chamar spread
*/


//| atribuição por desestruturação com arrays multiplos

const num2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
console.log(num2[1][2]) ;   //? resultado -> 6
//_ Para acessar elementos de um array dentro de um array.

const [, [, , seis]] = num2;
console.log(seis);

//* Aula 47 - Atribuição via desestruturação (objetos)
```

```
const pessoa ={
    nome: 'Jean Carlos',
    sobrenome: 'De Meira',
    idade: 23,
    endereco:{
        rua: 'Rua legal',
        numero: 4,
    },
}

//_ Atribuição via desestruturação
//_ É possível setar valor padrão. Se não tiver valor é usado o padrão.
//_ Pode-
se usar uma atribuição com nome diferente usando o exemplo de nome abaixo.
//_ também é possível usar o rest ...
const {nome: teste = 'Não encontrado', sobrenome, idade, ...rest } = pesso
a;
console.log(teste, sobrenome, idade, rest);

const {endereco: {rua, numero:num}, endereco}= pessoa;
console.log(rua, num, 'e', endereco);

//* Aula 48 - for clássico - estrutura de repetição

//#Nota# tais estruturas fazem ações repetidas, para evitar repetição de c
ódigo.

console.log('Linha 0');
console.log('Linha 1');
console.log('Linha 2');
console.log('Linha 3');
console.log('Linha 4');
console.log('Linha 5');

for(let i=0; i<=5; i++){
    console.log(`Linha ${i} com repetição for`);
}
/*
    _ Estruturas de repetição precisam de uma variável de controle, normal
mente usado o i
    _ Pois irá representar a palavra index, que significa indíce. Ela prec
isa ter valor inicial
    _ Então  é preciso usar uma comparação, pois a estrutura repete o códi
go enquanto a pergunta
    _ retornar true (verdadeiro).
```

```
    _ E por fim precisa de um incremento ou decremento, pois precisa ter m
udanças no valor da variável
    _ de controle, se não a repetição é infinita.
    _ o i ou variável de controle pode começar de qualquer valor e a alter
ação pode ser de um em um ou de qualquer valor
    _ em qualquer valor.
    _ é possível iniciar i com valores negativos.
*/

const frutas = ['maçã', 'pêra', 'uva', 'Manga', 'laranja', 'Mamão'];

for(let i=0; i<frutas.length; i++){
    console.log(`Indíce ${i}: `,frutas[i]);
}

//* Aula 50 - DOM e a árvore do DOM

/*
    _ window é o elemento pai de todos. -> é a tela
    _ depois temos o document   -> é o documento html
    _ então tem o HTML em si
    _ que contém o head e body por exemplo
    _ depois tem diversos elementos filhos
    _ Tendo os mais diversos elementos possíveis.
*/

//* Aula 51 - For in - estrutura de repetição

const frutas = ['Pêra', 'Maça', 'uva'];
const pessoa = {
    nome: 'Jean',
    sobrenome: 'Meira',
    idade: 23,
}

//#nota# o for in lê os índices ou chaves.

for(let indices in frutas){
    console.log(frutas[indices])
}

for(let chave in pessoa){
    console.log(chave,':', pessoa[chave]);
}
```

```
//* Aula 52 - for of - estrutura de repetição

const nome = ['Jean','Carlos', 'De Meira'];

for(let i=0; i<nome.length; i++){
    console.log('for clássico: ',nome[i]);
}

console.log('################');

for(let i in nome){
    console.log('for in:', nome[i]);
}

console.log('################');

//_ Já acessa diretamente o valor
for(let valor of nome){
    console.log('for of:', valor)
}

console.log('################');

//#obs# os acima podem funcionar com uma string, iterando sobre as letras ou elementos da string
//#obs# o forEch não consegue iterar sobre uma string
nome.forEach(function(valor, indice, array){
    console.log(valor, indice, array);
})

//_ para objetos pode diferenciar um pouco o for in funciona e quanto aos demais será falado
//_ mais detalhes adiante. Pode avaliar como não iterável, pois não associa com indíces.
//_ For clássico - geralmente para iteráveis (arryas e strings)
//_ fo in - retorna o índice ou chave (string, arrays e objetos)
//_ for of - retorna o valor em si (iteráveis, como arryas e strings)

//* Aula 54 - while e do while - estruturas de repetição

//| while
/*
    _ no while a variável de controle é criada fora do laço, a condição é colocada na criação do while e
```

```
    _ e a atualização da variável de controle é feita dentro do corpo do c
ódigo do while.
    ! Não esquecer de atualizar, pois se não cria condições de laços infin
itos. Onde a repetição nunca acaba;
*/

let i = 0;

while(i <= 10){
    console.log(i);
    i++;
}

function random(min, max){
    const r = Math.random() * (max - min) + min;
    return Math.floor(r);
}

const min = 1;
const max = 30;
let rand = random(min, max);
console.log(rand);
//? rand = 10;  //comente e descomente essa linha para ver o efeito de dif
erença do while e do while

while(rand !== 10){
    rand = random(min, max);
    console.log('While - Número aleatório é:',rand);
}

//#nota# usado geralmente quando não se sabe quantas vezes o laço deve ser
 executado.

//| do while
/*  //#nota#
    _ a diferença para o while é que o while checa a condição e depois exe
cuta o código se a condição for verdadeira.
    _ podendo assim não executar nenhuma vez se a condição for falça logo
de cara.
    _ já o do while executa primeiro e depois testa a condição, se a condi
ção for verdadeira vai executar novamente e
    _ testar ao final de cada execução, assim sendo mesmo que a condição s
eja verdadeira de cara, o código irá executar pelo
    _ menos uma vez.
*/
```

```
console.log('##########################################');
do{
    rand = random(min, max);
    console.log('do while - Número aleatório é:',rand);
}while(rand!==10);

/*  //#nota#
    _ Comente e descomente a linha indicada acima, então o comportamento d
e while e do while serão mais evidentes.
    _ Ao descomentar a linha, o valor inicial de rand será 10, que fará co
m que a condição seja inicialmente verdadeira
    _ então do while não irá executar;
    _ Mas o do while sim, e dentro do mesmo o rand será chamado, fazendo o
s valores seres aleatórios novamente; e assim executando
    _ o código até encontrar um valor igual a 10.
*/

//* Aula 55 - Break e continue

//#nota# funciona em todos os laços, ajuda a permitir o controle de quando
 pular um elemento ou
//_ quebrar o laço e sair a qualquer momento que quiser.

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

for(let numero of numeros){

    if(numero === 2 || numero === 5){ //? pode ter mais condições
        console.log('Pulei o número');
        continue;
        //#nota# sempre que achar continue ele pula para próxima interação
 do laço
    }
    if(numero === 8){
        console.log('Pulei o número');
        continue; //? pode ter mais de um continue
    }
    console.log(numero);
    if(numero === 7){
        console.log('encontrei o número 7, saindo.');
        break;
      //#nota# o break força a saída do laço, ou da estrutura sendo usada.
    }
}
```

```
//#nota# Cuidado com while e do while, pois a atualização da variável pode
 acabar sendo interrompida
//#nota# por que o continue ia para próxima iteração, então o valor não mu
daria mais, caindo em laço
//#nota# infinito. Portanto, se deve atuazar a variável também antes de um
 continue.

//Exercícios
// Escreva  uma função que recebe dois números e retorna o maior deles

function maiorEntreDoisNumeros(num1, num2){
    return num1 > num2 ? num1 : num2
}

console.log(maiorEntreDoisNumeros(2,10));

//? usando arrow function de retorno de uma linhas
const max2 = (num1, num2)=>  num1 > num2 ? num1 : num2;
console.log(max2(21,1));

/*
    _ Escreva uma função chamada ePaisagem que recebe dois argumentos, lar
gura e altura
    _ de uma imagem (number).
    _ Retorne true se a imagem estiver no modo paisagem.
*/

const ePaisagem = (largura, altura)=>  largura > altura ? true : false;
console.log(ePaisagem(21,1));

//? por retornar apenas true e false é possível omitir o ternário,
//? então ele checa a condição e retorna o resultado.

const ePaisagem2 = (largura, altura)=>  largura > altura;
console.log(ePaisagem2(21,1));

/*
    Escreva uma função que recebe um número
    retorne o seguinte:
    Número é divisível por 3 = Fizz
    Número é divisível por 5 = Buzz
    Número é divisível por 3 e 5 = FizzBuzz
    Número não é divisível por 3 e por 5 = Retorna o próprio número
    Checar se o número é realmente um número = retorna o próprio número
    Use a função com número de 0 a 100
```

```
*/
    const result = num => {
        if(num === 0) return num;

        if(num> 100 || num < 0){
        return 'fora dos limites';
        }else{ if(typeof(num) === "number"){
        if(num % 3 === 0 && num % 5 ===0) return 'FizzBuzz';
        if(num % 3 ===0) return 'Fiz';
        if(num % 5 ===0) return 'Buzz';
            return num;
    }else{
    return 'Não é um número'
    }
    }
}

for(let i=0; i<=100; i++){
    console.log(result(i));
}

//* Aula 59 - Tratando e lançando erros (try, catch e throw)

//_ try é como um tente, e se der erro cairá no catch

// try{
//     console.log(naoExisto); //? contém códigos que podem dar erros.
// }catch(err){
//     console.log('naoExisto não existe');
//     console.log(err);
// }
//#Obs# a execusão para no 1º erro, então comentar e descomentar o código
para analisar as ocorrências.

//#Nota# não é recomendado exibir o erro para o usuário, pois isso é poten
cialmente perigoso para sua aplicação

//| Lançando erros;

function soma(x, y){
    if(typeof x !== 'number' || typeof y !== 'number'){
        throw('X e y precisam ser números')
    }
    return x + y;
}
```

```
function soma2(x, y){
    if(typeof x !== 'number' || typeof y !== 'number'){
        throw new Error('X e y precisam ser números')
    }
    return x + y;
}

try{
    console.log(soma2(1,2));
    //console.log('com tryCatch',soma('a',2));   //? descomentar para usar
 função soma, então o erro só retornará a string.
    console.log(soma2('a',2));
}catch(error){
    console.log(error);
}
//#nota# Ao usar a soma2 também é criado um erro, então esse log de erro v
olta ao apresentar o erro, diferente da função soma.
//#nota# se criar um ReferenceError, será retornado no console um Referenc
eError. Existem vários tipos de classes de error.
//#Importante# Não retorne o erro para o usuário, aqui está sendo retornad
o por ser ambiente de estudo.

//* Aula 60 - Tratando e lançando erros (try, catch e finally)

try{
    console.log('Abri um arquivo');
    console.log('Manipulei o arquivo');
    console.log('gerou erro');
    console.log(a);
    console.log('Fechei o arquivo');
    //? Executada quando não há erros.
}catch(e){
    console.log('Tratando o erro');
    //? executada quando há erros.
}finally{
    console.log('Fechar o arquivo, Finally');
    //? sempre é executado.
}

//#Nota# podem ter tryCatchs aninhados (um dentro do corpo de código do ou
tro.)


//| parte 2
```

```
function retornaHora(data){
    if( data && !(data instanceof Date)){
        throw new TypeError('Esperando instância de Date.');
    }

    if(!data){
        data = new Date();
    }
    return data.toLocaleTimeString('pr-Br',{
        hour: '2-digit',
        minute:'2-digit',
        second:'2-digit',
        hour12: false,
    });
}

try{
    const data = new Date('01-01-1970 12:58:12');
    const hora = retornaHora();
    console.log(hora);
}catch(erro){
    // Tratar error
    console.log(erro)
}finally{
    console.log('Tenha um bom dia');
}

/*
    #Nota#
    _Nesse programa se receber a const data, irá executar com a data unix
e o finally com o
    _tenha uma bom dia. Se for inviado vazio retornará a hora atual e o fi
nally.
    _ Se for enviado formatos de dados diferentes será gerado um erro com
o throw e cairá no catch,
    _que nesse cado mostra o erro, e então cairá no finally também.

*/

//* Aula 61 - setInterval e setTimeout

function mostraHora(){
    let data = new Date();
    return data.toLocaleTimeString('pt-Br',{
        hour12: false,
```

```
    })
}


const timer = setInterval(function(){
    console.log(mostraHora());
},1000);
/*
    #Nota#
    _ A função setInterval vai configurar o intervalo de tempo para que al
guma função
    _ seja executada em determinado tempo.
    _ setInteval recebe dois parâmetros, o primeiro é qual função será exe
cutada
    _ O segundo é de quanto em quanto tempo será executada em miliSegundos
 (mS).
    _ A função executada para setInterval não pode retornar valor.
*/

setTimeout(function(){
    clearInterval(timer)
}, 5000);

/*
    #Nota#
    _ A função setTimeout executará uma vez só, assim que o tempo setado p
assar.
    _ Também recebe dois parâmetros, cujo o primeiro é a função que será e
xecutada, e o segundo
    _ É o tempo para ser executado.
    _ Também precisa de uma função sem retorno.
    _ clearInterval vai interromper a execução.
*/

//Funções avançadas

//* Aula 64 - Maneiras de declarar funções

//| declaração
function falaOi(){
    console.log('Oi');
};
//_ É uma declaração mais literal.
//_ Ocorre o function hoisting -> eleva a declaração.
//_ declaração mais clássica.
```

```
falaOi();

//| functions expressions
//_ função (todas) são first-class objects (objetos de primeira classe)
    //_ podem ser tratadas como dado.

    const souUmDado = function(){
        console.log('Sou um dado');
    }

    souUmDado();
//_ Pode executar a constante ou variável como uma função.
//_ Permite passar funções como parâmetros.

function executaFuncao(funcao){
    funcao();
}

executaFuncao(souUmDado);

//| Arrow function -> function expression resumida na declaração

const arrow = ()=> console.log('Sou uma arrow function');

arrow();

//| Dentro de objetos

const obj = {
    //falar: function(){    //? abaixo tem o modo mais novo de declarar um
 método do objeto.
    falar(){
        console.log('Estou falando...');
    }
}

obj.falar();

//* Aula 65 - Parâmetros da função

function funcao(){
    console.log(arguments);
}
/*
```

```
    #Nota#
    _ Funções criadas com a palavra function tem um parâmetro com nome arg
uments, que guarda(sustenta) todos os argumentos.
    _ Mesmo que não crie parâmetros a função vai salvar os argumentos invi
ados para suprir os parâmentros.
    _ arguments é um objeto.
*/

funcao(1, 2, 3, 4, 5, 6, 7);

function funcao2(a=0, b=0, c=0, d=0,e=0){
    console.log(arguments);
    console.log(a, b, c);
}

funcao2(1, 2, 3,undefined, 5, 6, 7);
/*
    #Nota#
    _ Isso ainda é válido caso existam argumentos criados. arguments ainda
 sustentará todos os argumentos.
    _ Se tiver parâmetros, mas não for passado argumentos para todos, o pa
râmetro será criado
    _ mas tera valor de undefined.
    _ É possível usar valores pré definidos para os parâmetros, e se os me
smos não receberem argumentos
    _ adoraram o valor inicial indicado.
    _ Se mandar undefined ele também adotará o valor padrão, Apesar de iss
o não ser muito indicado.
    _
    _ arguments não funciona com arrow functions.
    _ É possível usar a desestrruturação de objetos ou arrays dentro dos p
arâmetros de uma função
    _ Rest operator se for usado tem que ser sempre o último parâmetro da
função;
*/

//* Aula 66 - Retorno de uma função

/*
    #Nota#
    _ return retora algum valor
    _ encerra a função
    _ existe um retorno undefined por padrão
    _ se não existir a palavra return terá um retorno undefined
    _ funções podem ou não retornar valores. (além do undefined)
```

```
*/

function soma(a, b){
    return a+b;
}

console.log(soma(5,2));

//| retornando uma função sem executar imediatamente

function criaMultiplicador(multiplicador){
    //multiplicador está aqui

    /*
        _ Abaixo poderia ser de duas formas
        _ poderia ser criado a função e depois
        _ retornar a mesma sem () e argumentos,
        _ para que assim se retorne a função,
        _ que será executada em outra hora.
        _ ou retornar diretamente a função
        _ que assim também será executada depois.
    */
    // function multiplica(n){
    //     return n * multiplicador;
    // }
    // return multiplica;

    return function(n){
        return n * multiplicador;
    };
}

const duplica = criaMultiplicador(2);
const triplica = criaMultiplicador(3);
const quadriplica = criaMultiplicador(4);

console.log(duplica(2));
console.log(triplica(2));
console.log(quadriplica(2));

//* Aula 67 - Escopo léxico

const nome = 'Jean';

function falaNome(){
```

```
    console.log(nome);;
}
falaNome();
/*
    #nota#
    _ A função reconhece o que está declarado em torno dela,
    _ voltando escopos até chegar no global
*/
```

```
//* Aula 68 - Closures

/*
    #Nota#
    _ relacionado ao escopo léxico
    _ a const funcao tem acesso a três escopos.
    _ o dela, o da função mão (retornaFuncao) e o global.
    ! O closure é a habilidade da função de acessar ao seu escopo léxico
    _
*/

function retornaFuncao(){
    const nome = 'Luiz';
    return function(){
        return nome;
    }
}


const funcao = retornaFuncao();
console.log(funcao);    //? tem uma função anônima
console.dir(funcao);    //? no browser terá diferênças, inclusive mostra
os escopos acessíveis.
```

```
//* Aula 69 - Funções de callback

function rand(min = 1000, max = 3000) {
    const num = Math.random() * (max - min) +
        min;
    return Math.floor(num);
}

function f1(callback) {
    setTimeout(function () {
        console.log('f1');
        if (callback) callback();
```

```
    }, rand());
}

function f2(callback) {
    setTimeout(function () {
        console.log('f2');
        if (callback) callback();
    }, rand());
}

function f3(callback) {
    setTimeout(function () {
        console.log('f3');
        if (callback) callback();
    }, rand());
}

f1(f1Callback);

function f1Callback() {
    f2(f2Callback);
}

function f2Callback() {
    f3(f3Callback);
}

function f3Callback() {
    console.log('Olá mundo!');
}

/*
    #Nota#
    _ A função rand() e o setTimeout estão simulando interações com a web,
 onde algo demoraria
    _ um tempo indeterminado para realizar a ação.
    _ Se não existisse o callback as funções poderia ser executadas em ord
ens aleatórias, e
    _ isso pode comprometer o funcionamento de sistemas. Portanto é enviad
o um callback como argumento
    _ para as funções.
    _ e colocado uma condições, para que, se existir um callback nos parâm
etros recebidos o mesmo
    _ será executado, independente do que seja.
```

```
    _ Para melhor organização cria-
se funções f_Callback(), para que não seja colocadas aninhadas uma
    _ dentro da outra. Dessa forma se garante a ordem de execução das cois
as com o callback mesmo que
    _ cada função demore tempos diferentes para serem executadas.
*/
```

```
//* Aula 70 - Funções imediatas (IIFE)

//? Ou funções auto invocadas
//_ IIFE -> Immediately Invoked Function Expression

/*
    #Nota#
    _ Quando criamos algo que roda na web ou em contextos as variáveis e f
unções que
    _ criamos pque podem tocar o escopo global podem acabar sendo usadas,
alteradas
    _ ou mesmo conflitar com já existentes no escopo global, é preferível
evitar que isso
    _ ocorra. Desse modo é possível proteger o nosso escopo o envolvendo e
m uma função.
    _ porém essa função vai todar o escopo global, e isso pode não ser a m
elhor opção.
    _ As funções IIFE são anônimas e auto invocadas, dessa forma não é pos
sível de
    _ serem chamada, não tocando o escopo global assim.
    _ A estrutura da IIFE é envolvida em () e após terminar é colocado ()
para chamar a mesma.
    !   Exemplo da estrutura
    !       (function(){
    !           CORPO DA FUNÇÃO
    !       })();
*/

(function (idade) {
    const nome = 'Jean Meira'
    console.log(nome, ':', idade);
})(23);

const nome = 'Qualquer coisa';
```

```
//* Aula 71 - Funções fábrica (factory functions)
```

```
function criaPessoa(nome, sobrenome, altura, peso){
    return{
        nome,
        sobrenome,
        //getter
        get nomeCompleto(){
            return `${this.nome} ${this.sobrenome}`
        },
        //setter
        set nomeCompleto(valor){
            valor = valor.split(' ');
            this.nome = valor.shift();
            this.sobrenome = valor.join(' ');
        },
        fala(assunto){
            return `${this.nome} está falando sobre ${assunto}`
        },
        altura,
        peso,
        get imc(){
            const indice = this.peso/(this.altura**2);
            return indice.toFixed(2);
        },
    };
}

const p1 = criaPessoa('Jean', 'Meira', '1.72', '60');
console.log(p1.fala('Js'));
console.log(p1.imc);
p1.nomeCompleto = 'Jean Carlos De Meira' //? com o setter podesse alterar diretamente na chave.
console.log(p1.nomeCompleto);
/*
    #Nota#
    _ o this referência quem chamou a função,
    _ no contexto acima o this.nome é quase o mesmo que p1.nome
    _ colocar o get imc(), faz parecer que ele é um atributo normal.
    _ o get vem de getter e set ver sde setter.
    _ Factory function criam e retorna os objetos
    _ isso permite diminuir o código,
*/

//* Aula 73 - Funções construtoras (constructor functions)

/*
```

```
    #Nota#
    _ Funções construtoras retornam objetos (constroem)
    _ são iniciadas com letras maiúsculas. e obrigatóriamente contém a pal
avra new.
    _ new cria um novo objeto vazio, faz o this apontar para o objeto vazi
o, e retorna
    _ implicitamente.
    _ É possíveç ter atributos, métodos e variáveis privadas, que estão ac
essíveis apenas
    _ dentro da função construtora.
    _ Ao usar o this. cria-
se atributos e métodos públicos que podem ser acessados pelo notação
    _ de . fora da função construtora.
    _ A exemplo de p1.nome, p2.nome e p1.metodo demonstrados abaixo.
*/

function Pessoa(nome, sobrenome){
    //_ Atributos ou métodos privados
    const ID = 123456;
    const metodoInterno = function(){

    };

    //_ Atributos ou métodos públicos
    this.nome= nome;
    this.sobrenome = sobrenome;
    this.metodo= function(){
        console.log(this.nome + ': Sou um método');
    }
}

const p1 = new Pessoa('Jean', 'Meira');
const p2 = new Pessoa('João', 'André')
console.log(p1, p2);
console.log(p1.nome, p2.nome);
p1.metodo();

//* Aula 75 - Funções recursivas

/*
    #Nota#
    _ É uma função que ela mesma se chama.
    _ cuidado com o máximo de recursividade, em certos valores altes vai d
ar erro
    _ quase como um loop infinito.
```

```
*/

function recursiva(max){
    if(max < 0) return;
    if(max >10)  return;
    console.log(max);
    max++;
    recursiva(max);
}

recursiva(0);
```

```
//* Aula 76 - Funções geradoras

/*
    #Nota#
    _ Funções geradoras não vão retornar todos os valores de uma vez.
    _ é quase como se houvesse um pause ao longo de seu código
    _ usa um prencípio de laze evaluation (avaliação preguiçosa)
    _ e pode ser bom por causa de performance em alguns casos.
    _ Tem um método incluso, o next(), que retorna o valor (value)
    _ e se já acabou o gerador (done);
    _ A função é iterável;
    _ A função geradora normalmente usa p yield para retornar valores
    _ que funciona como um retorno fracionado, a cada vez que chama retorn
a a
    _ parte que está na lista dentro do corpo da função, repetindo isso at
é o último
    _ yield disponível. Se usar o return irá retornar o valor, mas também
irá quebrar
    _ a sequência disponível, fazendo com que a próxima chama apresente er
ros.
*/

function* geradora1() {
    //código qualquer ...
    yield 'Valor 1';
    //código qualquer ...
    yield 'Valor 2';
    //código qualquer ...
    yield 'Valor 3';
}

const g1 = geradora1();
console.log(g1.next().value);
```

```
console.log(g1.next().value);
console.log(g1.next().value);

const g2 = geradora1();
for(let valor of g2){
    console.log('For: ',valor);
}

function* geradora2(){
    let i = 0;
    while(true){
        yield i;
        i++;
    }
}
const g3 = geradora2();
console.log(g3.next().value);
console.log(g3.next().value);
console.log(g3.next().value);

function* geradora3(){
    yield 0;
    yield 1;
    yield 2;
}

function* geradora4(){
    yield* geradora3();
    yield 3;
    yield 4;
    yield 5;
}

const g4 = geradora4();
for(let valor of g4){
    console.log('delegando função', valor);
}

function* geradora5(){
    yield function(){
        console.log('Vim de y1');
    };

    yield function(){
        console.log('Vim de y2');
```

```
    }
}

const g5 = geradora5();
const func1 = g5.next().value;
const func2 = g5.next().value;
func1();
func2();
```