**Nome:** Guilherme Fabiano Terra da Silva

1. **Seção 1**

* Introdução ao conteúdo do curso;
* HTML
* CSS
* JavaScript
* React
* Introdução a a TypeScript

1. **Seção 2**
2. **HTML**

* Não é linguagem de programação;
* É linguagem de marcação;
* Pois não possui if else, loops e outras capacidades padrões das linguagens de programação;
* É o esqueleto / estrutura do site;

1. **CSS**

* É uma linguagem de estilização;
* Faz o polimento do HTML;

1. **IDE usada no curso**

* VS Code;
* Instalação e configuração;

1. **Início HTML**
2. **Anatomia de tags**

* Tags possuem propósitos;
* Tags são envolvidas com <> e fechamento </>;
* Toda tag possui um nome e pode possuir atributos;
* Dentro da tag vai o conteúdo;

1. **Estrutura básica**

* DOCTYPE: Declara versão do HTML;
* Html: Para envolver todo html;
* Head: Para configurações;
* Body: Para elementos visíveis;

1. **Títulos**

* h1 a h6;

1. **Parágrafos**

* p;

1. **Quebra de linha (Não se usa mais)**

* br;

1. **Linha horizontal (Não se usa mais)**

* hr;

1. **Comentários**

* <!--Texto -->;

1. **Links**

* **Tag:** a;
* **Direcionar para link:** Atributo – href=”link”;
* **Abrir em uma nova aba:** Atributo – target=”\_blank”;

1. **Imagens**

* **Tag:** img;
* **Selecionar imagem:** Atributo – src=”local/caminho/link”;

1. **Listas não ordenadas**

* <ul> </ul>;
* Dentro da tag ul se coloca as linhas <li> </li>;

1. **Listas ordenadas**

* <ol> </ol>;
* Dentro da tag ol se coloca as linhas <li> </li>;

1. **Tabela**

* <table> </table>;
* Dentro de table vai tr, e dentro dos tr’s vai um th e vários td’s;
* **Linhas:** <tr> </tr>;
* **Colunas:** <th> </th> e <td> </td>;
* Th é só para primeira linha;

1. **Início CSS**
2. **Como adicionar?**

* Inline (Com atributo style);
* Internal (Na tag head);
* External (Em arquivo externo);

1. **Estrutura padrão**

* Nome da propriedade: Valor;

1. **ID**

* #nomeDoID;
* É único no site, não pode ter mais de 1 coisa com o mesmo id;
* Diferente do class (.nomeDaClasse) que é usada várias vezes;

1. **Cores**

* Os mais comuns são: Hexadecimal, RGB e RGBA;
* Hexadecimal é o padrão;
* Hexadecimal é um código no formato #xxxxxx ou #xxx (X: 0 a F);
* Os valores do RGB vão de 0 a 255 para cada tonalidade;
* Os valores de RGBA são 0 a 255 para RGB e 0 a 1 para A;
* A é a opacidade / nível de visibilidade;

1. **Color e Background-color**

* Regra: cor (em Hex, RGB, RGBA);
* RGBA é útil para deixar apenas um elemento com transárência;
* Um outro jeito de aplicar transparência é usando:
* opacity: (0 a 1);

1. **Auxiliar uso de imagem**

* Background-position;
* Background-size;

1. **Dimensões**

* Width;
* Height;

1. **Padding**

* **Todas as direções:** padding: valor;
* **Direção específica:** padding-direção: valor;
* **Todas com valores diferentes:** padding: topo direita inferior esquerda;
* **Laterais e extremidades sup e inf:** padding: laterais supEinf;

1. **Box-sizing: Border-box**

* Faz com que o tamanho do padding, seja incluso no tamanho máx do width da div;
* Exemplo: Se vc coloca uma div de width: 100px e coloca um padding de 25px, a div final fica com 150px (100px inicial + 25px de cada lado de padding). Com o Box-sizing: Border-box o resultado final é 100px, sendo 50px de padding e 50px para o conteúdo;

1. **Alinhamento de texto**

* Text-align: valor;
* Center, right;

1. **Decoração de textos**

* Text-decoration: underline;
* Text-decoration: line-through;
* EX: texto
* EX: ~~texto~~

1. **Transformação de textos**

* Não se escreve texto do HTML em UPPERCASE;
* O CSS se encarrega de transformar;
* Text-transform: uppercase;
* Text-transform: lowercase;

1. **Espaçamento de letras**

* Letter-spacing: Npx;

1. **Fontes**

* Font-family: nomeDaFonte;
* Sans serif é mais comum;

1. **Estilo da fonte**

* Font-style: nomeDoEstilo;
* Italic, oblique e etc;

1. **Grossura de letras**

* Font-weight: valor;
* Valor vai de 100 a 900;

1. **Tamanho da fonte**

* Font-size: valor;
* Normalmente é 16px;
* Tem múltiplas unidades;

1. **Display**

* Relacionada ao comportamento dos blocos / estruturas;
* Podem ser block ou inline;
* **Block:** Ocupa linha toda;
* **Inline:** Fica um do lado do outro;
* Display: inline-block;
* Display: block;

1. **Escondendo elementos com Display**

* Para esconder basta usar none no Display;
* Display: none;

1. **Posições**

* Relative, fixed, absolute, stick e etc;
* Position: nomeDaPosicao;
* A posição padrão é static;
* **Posição relative:** Move o elemento pela tela. Inicialmente não muda nada, mas depois de usar é possível aplicar top/left/right/bottom: valor; para desloca-lo da posição atual para outra;
* **Posição absolute:** Move o elemento pela tela. Inicialmente não muda nada, mas depois de usar é possível aplicar top/left/right/bottom: valor; para desloca-lo dentro de um container que possui posição relative. Se não existir o container ele leva em consideração o body;
* **Posição fixed:** Trava o elemento na tela, não deixando sair nunca, independentemente do scroll. É obrigatório aplicar o top/left/right/bottom: valor, se não ele nãi fica travado;
* **Posição stick:** É que nem o fixed, mas ele só fica fixo na tela a partir do momento que o scroll passa dele, de forma que ele ficaria para cima do site. Se o stick estiver mais para baixo ele não fica fixo na tela;

1. **Z-Index**

* Determina quem vai ficar na frente de quem, caso ocorra colisão;
* Z-index: valor;
* Não usar valores absurdos, somente o necessário;

1. **FORMULÁRIOS**
2. **Tag form**

* Dentro dela vão as tags de formulário;
* Possui 2 atributos normalmente;
* Action – Arquivo ou página em que os dados serão enviados;
* Method – GET ou POST (Ou tá recebendo dados, ou enviando);

1. **Atributo name**

* Importante para trabalhar com Backend;
* O nome se refere ao dado daquele campo;

1. **Atributo for**

* Serve para “linkar” um label com um input;
* O atributo for recebe o name do seu respectivo input;

1. **Envio de dados com botão submit**

* Usa-se um input;
* Quando clica envia os dados;
* Type: submit;
* Atributo value para colocar texto no botão;

1. **Tag select**

* Tag de seleção de uma única opção de varias outras;
* Pode possuir label;
* Dentro dela vão as tags option;
* Option não tem name, só value;
* Pode-se colocar o atributo selected, para um dos options iniciarem como selecionado;

1. **Select com múltiplas seleções**

* Basta adicionar atributo multiple;

1. **Área de textos**

* Tag textarea

1. **Fieldset e legend**

* São tags usadas para semântica;
* A tag Fieldset engloba tags que se relacionam, como nome e sobrenome;
* A tag legend é uma legenda que vai dentro da tag do fieldset;

1. **Datalist**

* Permite adicionar um autocomplete a uma tag de texto;
* Em um input de type: text, pode-se adicionar o atributo list;
* O atributo list recebe o mesmo valor do atributo id do datalist;
* O datalist é uma tag que recebe id;
* Dentro desse datalist vão as tags option que recebem APENAS o atributo value, sem texto na tag option;
* Permite que o usuário digite, ou selecione um texto já existente;

1. **Senha**

* Input com type: password;

1. **Limpar formulário**

* Input de botão que limpa todo o formulário;
* Input type: reset;
* Value para controlar texto que vai dentro do botão;

1. **Radio**

* Input de type: radio;
* Só 1 pode ser selecionado;

1. **PROJETO 2**
2. **RESPONSIVIDADE**
3. **Media query**

* É um recurso que concentra regras de CSS para cada resolução de cada dispositivo diferente;
* Adapta a tela de acordo com breakpoints, que são tamanhos de pixel da largura da tela;

1. **Tag de responsividade**

<meta name="viewport" content="width=device-width, initial-scale=1.0">

* Já usamos quando iniciamos o projeto com a estrutura básica;

1. **Estrutura**

@media screen and (max-width: 500px){

    h1{

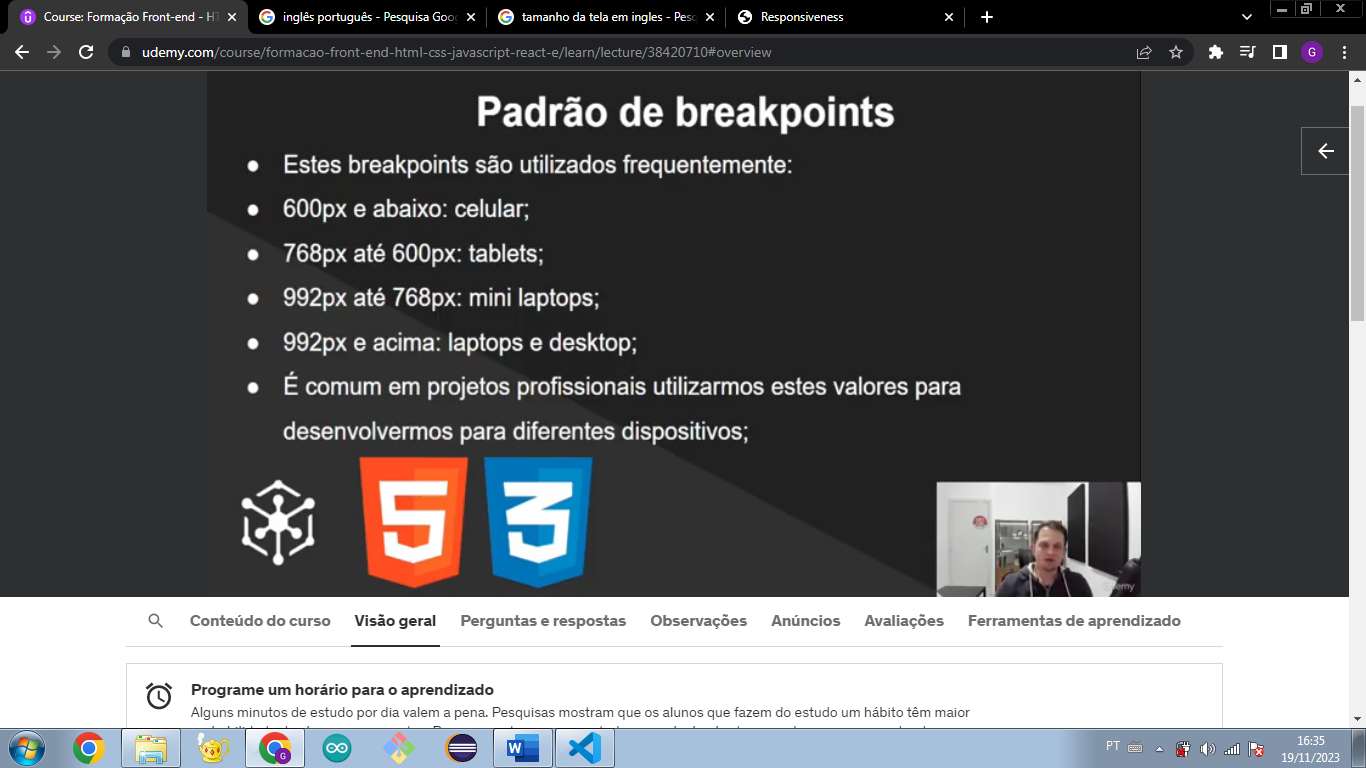
        color: blue;

    }

}

* Da pra usar max e min;

1. **Padrões**



1. **Visão landscape**

* Quando usuário vira celular / table de lado;
* Da pra alterar regras somente para estas situações;
* Estrutura:

@media screen and (orientation: landscape){

    h3{

        display: block;

    }

}

1. **HDC Host – Projeto**
2. **Recomendação de site para pegar ícones**

* CdnJs;
* Estrutura de uso: <i class=”fas <nomeDoIcone>”></i>
* Link:

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css" integrity="sha512-z3gLpd7yknf1YoNbCzqRKc4qyor8gaKU1qmn+CShxbuBusANI9QpRohGBreCFkKxLhei6S9CQXFEbbKuqLg0DA==" crossorigin="anonymous" referrerpolicy="no-referrer" />

1. **Tags não respeitando / estourando tamanho das divs pai**

* Algumas vezes as tags filhas não respeitam o tamanho da tag pai;
* Pode-se usar box-sizing: border-box para corrigir o problema (Algumas vezes);
* Coloca-se isso na configuração de reset do site;

/\* Reset \*/

\*{

    margin: 0;

    padding: 0;

    font-family: Helvetica;

    box-sizing: border-box;

}

1. **FLEXBOX**
2. **É um valor da propriedade display**

* Auxilia no posicionamento de elementos;
* Substitui outras propriedades como o position;
* O elemento PAI é quem recebe o flex;

1. **Os elementos que estão dentro do flex, se comportam como “inline”, começa como comportamento de row**
2. **Flex-direction**

* Row (Padrão);
* Column;

1. **Flex wrap**

* O flex, por padrão, tenta sempre colocar todos os elementos dentro dele na mesma linha. Para resolver este problema existe a propriedade flex-wrap: wrap, que faz a linha ser quebrada no momento de “estouro da tela”;
* **OBS:** Sem wrap o flex altera o comprimento dos elementos para que todos fiquem na mesma linha;

1. **Justify-content**

* Alinha horizontalmente o conteúdo do flex;
* Center, space-around, space-between e end;
* **Center:** Centraliza;
* **Space-around:** Distribui elementos com distância iguais e espaços livres antes do primeiro e depois do último elemento;
* **Space-between:** Distribui elementos com distâncias iguais entre eles;
* **End:** Joga elementos pro lado direito – final da tela;

1. **Align-items**

* Alinha verticalmente o conteúdo do flex;
* Center, flex-end e stretch (faz altura ser = pai, se não tiver h fixo);

1. **Gap**

* Põe espaçamento entre os elementos dentro do flex;
* Gap: Npx;

1. **Order**

* Altera a ordem dos filhos do flex;
* Geralmente usa id para determinar a ordem;
* No css usa-se: #nomeDoID{ order: número }

1. **Grow**

* É possível alterar o tamanho dos elementos flex;
* Crescendo ou diminuindo-os individualmente;
* Basta colocar a propriedade flex: proporção em relação ao padrão;

.grow{

    display: flex;

}

.grow .box{

    flex: 1;

}

.grow .big{

    flex: 2;

}

A caixa big tem 2x mais comprimento q as caixas box

1. **Basis**

* Configura a largura base do elemento;
* Flex-grow: Npx;

1. **Shrink**

* Oposto de grow;
* Faz o elemento reduzir de acordo com a alteração do tamanho da tela;

1. **Flex-shorthand**

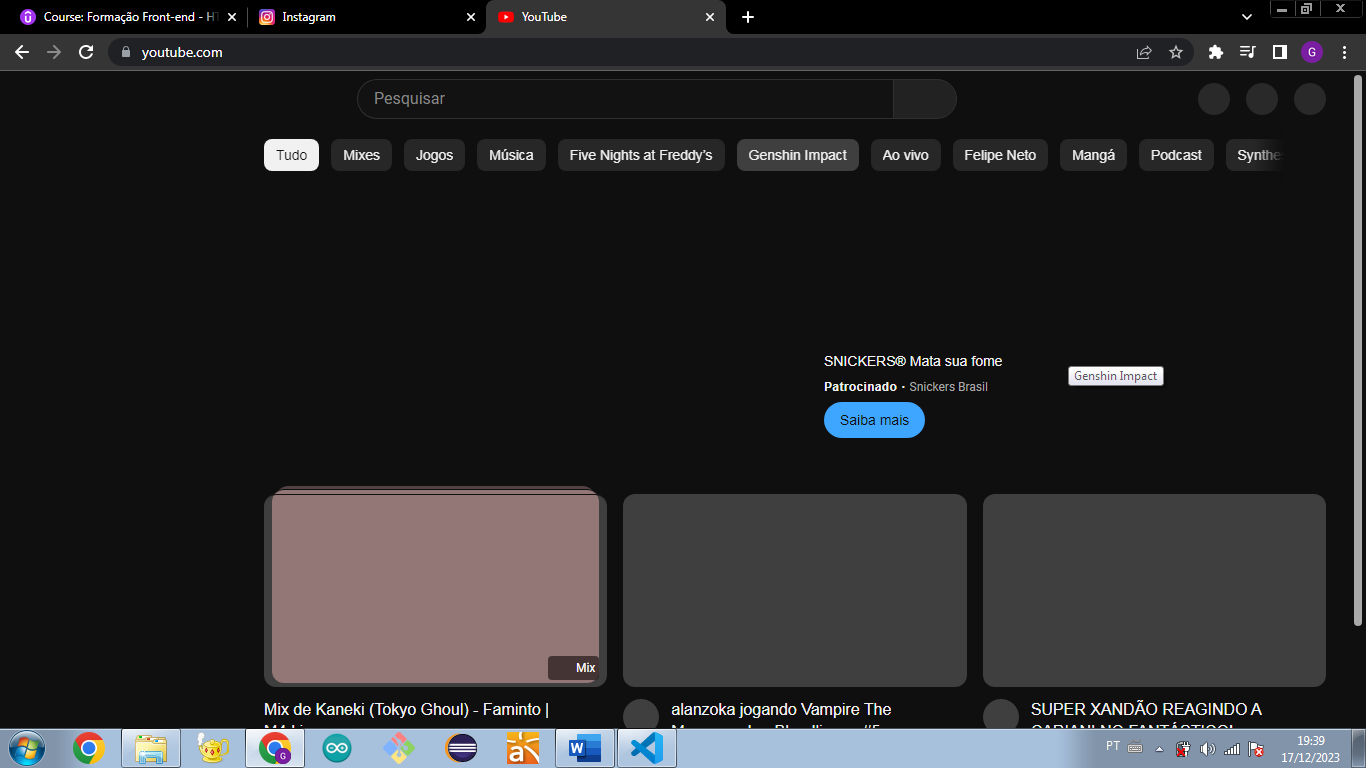
* Permite definir grow, shrink e basis ao mesmo tempo;
* Usado com o flex: Valor\_Grow Valor\_Sh Valor\_Basis\_px;

1. **Auto alinhamento**

* Permite que um elemento específico fuja do alinhamento imposto pela div pai;

1. **CLONE HOME DO INSTAGRAM**
2. **Favicon**

* Todo site possui um ícone que fica lá em cima na aba, junto do título do site;



* Para trocar usa-se a tag link, que nem o link do css, contudo, muda-se alguns atributos;
* O rel e o type sempre é a mesma coisa, o href (imagem) muda;

<link rel="shortcut icon" href="img/favicon.png" type="image/x-icon">

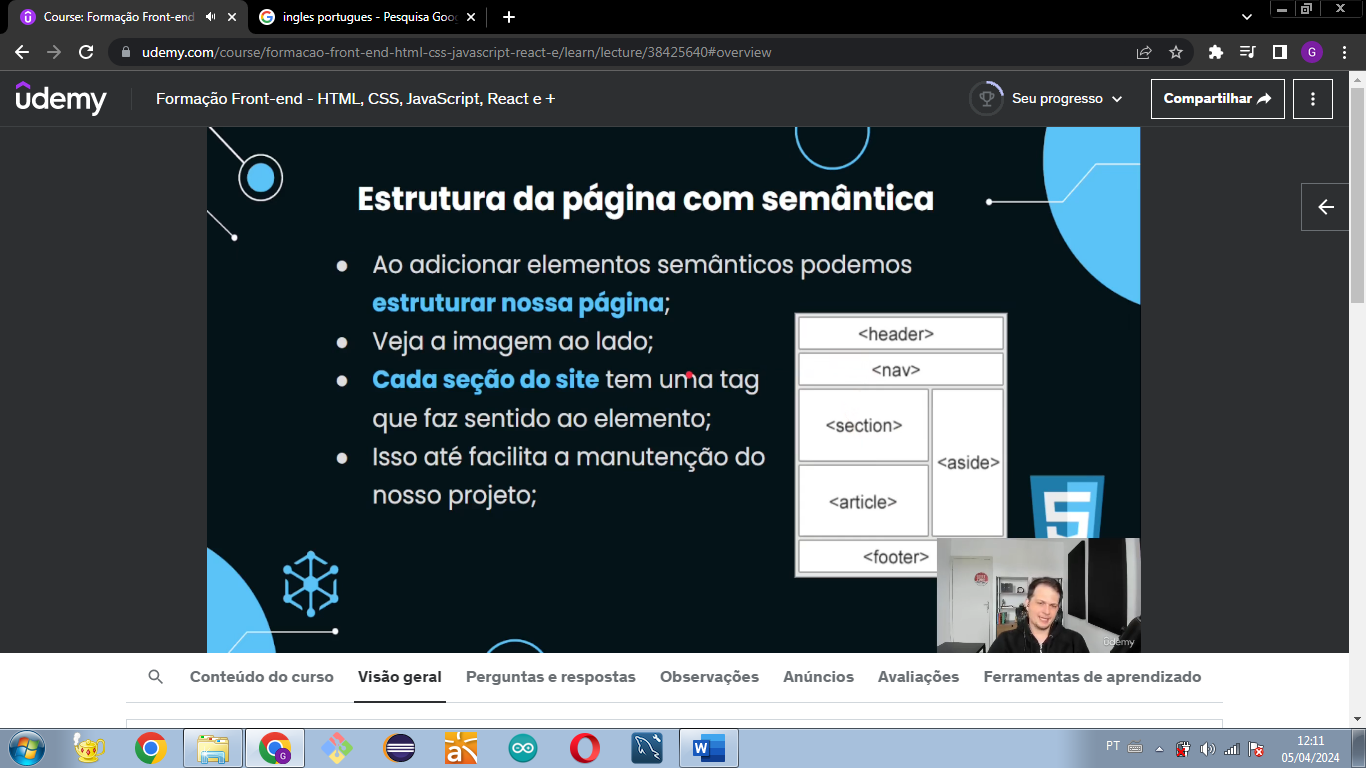
1. **FLEW WRAP**

* **Faz a linha ser quebrada quando os elementos que estão em linha estouram o tamanho da tela;**
* **Flex-wrap: wrap;**

1. **HTML SEMÂNTICO**
2. **É a atribuição de significado aos elementos**

* É um meio de facilitar o entendimento do código através de tags;
* Isso ocorre através da implementação de tags com nomes diferentes para coisas diferentes (Ex: Tag para menu, texto e etc)
* Influencia / Contribui para o SEO;

1. **Estruturação de página com semântica**



1. **Section**

* Geralmente define um agrupamento por categorias;
* Ex: Seção de produtos, contato e etc;
* Normalmente usado várias vezes em projeto;

1. **Article**

* Geralmente usada para elementos informativos;
* Ex: Posts de blog, comentários, card de produtos;

1. **Header**

* Cabeçalho;
* Geralmente tem headings, logo, informações do autor do post/página;
* Pode ter mais de um header;

1. **Footer**

* Rodapé;
* Geralmente usada para informações sobre a empresa, copyright, dados de contato e sitemap;
* Também pode ter mais de um. Podendo ser utilizada em outras tags para fazer o encerramento;

1. **Nav**

* Bloco de navegação;
* Espera que dentro dele venha links;
* Nem todo link precisa estar dentro de uma nav;
* O exemplo mais comum é a barra de navegação principal;

1. **Aside**

* Geralmente usado para criar conteúdo ao lado de um conteúdo principal;
* O conteúdo dentro de um aside relaciona indiretamente ao conteúdo principal;

1. **Figure + Figcaption**

* Figure é como um elemento pai para exibição de imagens;
* Ficaption = Legenda;
* O figcaption representa o atributo alt de um img;

1. **Main**

* É uma tag que engloba a parte mais importante da página;
* Só pode usar uma única tag main por página;

1. **Mark**

* É uma tag que substitui o span. Dentro dela deve conter um texto;
* Ela define um conteúdo que precisa estar em evidência;

1. **JAVASCRIPT**
2. **JS**

* É uma linguagem de programação de alto nível;
* Sua função é dar vida as páginas;

1. **JS é Case sensitive**
2. **JS - CONCEITOS**
3. **JS TIPOS DE DADOS E OPERADORES**
4. **Os mais comuns são:**

* Number;
* String;
* Boolean;
* Empty values(Null, undefined);
* Object;

1. **Como verificar o tipo do dado?**

* Typeof;

1. **Operadores**

* + , - , \* , / ;

1. **Special Numbers (Geralmente são resultados de operações)**

* Infinity: QUANDO RESULTADO TENDE A INFINITO;
* - Infinity: QUANDO RESULTADO TENDE A - INFINITO;
* Nan (Not a Number): QUANDO A OPERAÇÃO É FEITA COM NUMBER E ALGUMA OUTRA COISA;

1. **String**

* Em JS tem 3 maneiras de criar;
* Usando aspas simples, duplas ou crase;

1. **Interpolação**

* É uma possibilidade do JS de escrever variáveis dentro da string, sem ter que ficar concatenando;
* É tipo um printF;
* **Estrutura:** console.log(` ${ var } `);
* SÓ FUNCIONA SE USAR A STRING COM CRASE;

1. **Comparações**

* Os padrões e +2;
* Comparador Idêntico ( === );
* Comparador Diferente ( !== );
* O JS compara o valor da var, como por exemplo 5 e “5” e aponta igualdade mesmo sendo int e String. Contudo, por tipo eles são diferentes;
* Estes operadores comparam e levam em consideração o tipo também;

1. **Operadores lógicos ( && , || e ! )**
2. **Null (O programador normalmente atribui) – Undefined (Valor ainda não definido)**
3. **Conversões de tipo automática (“Bug por não set tipado”)**



1. **Declaração de variáveis**

* Let nomeDaVar = valor;
* O ; é opcional;
* Const nomeDaVar = valor;
* O const é mais comum;
* É case sensitive;
* O const não pode ser alterado;

1. **Função prompt (‘É o scanner’)**

* Estrutura: prompt(“Texto”);
* É uma função que exibe um texto e espera um input no log;
* Não utilizado, pois a caixa de entrada não é editável e ele fica esperando a entrada e não carrega o resto da página;

1. **Função alert**

* Exibe um pop up na tela com uma mensagem;
* Não utilizado, também trava a execução de código até clicar no OK do pop up;

1. **Objeto Math**

* Tem funções muito úteis;
* Math.max() – retorna maior valor de n valores;
* Math.floor() – Arredonda valor para baixo;
* Math.ceil() – Arredonda valor para cima;

1. **Objeto Console**

* Normalmente usado para exibir mensagens no log (principalmente de erro);
* Console.log(“Mensagem”);
* Console.error(“Mensagem”);
* Console.warn(“Mensagem”);

1. **ESTRUTURAS DE PROGRAMAÇÃO**
2. **Estruturas de Controle e Repetição**

* Tem todas as padrões;
* For, while, if, ...;

1. **Continue**

* Funciona como break, mas n sai do laço, apenas pula a iteração atual;
* Usado normalmente no for;

1. **FUNÇÕES**
2. **Mesma estrutura de outras linguagens**

* Não precisa indicar qual tipo será retornado;
* Não precisa indicar tipo dos argumentos recebidos;
* Uma variável pode receber uma função (const);
* Normalmente se usa const por segurança;

**Const nomeVar = function nomeDaFuncao( a , b ){**

**}**

1. **Arrow Function**

* É um modo simplificado de fazer uma função;
* Usado em variáveis;

**Cont nomeVar = ( a ) => {**

**}**

1. **Arrow Function de uma linha**

* Omite as chaves e retorna o valor da operação;
* É um arrow mais simplificado;

**Const nomeVar = ( a ) => a \* a;**

1. **Undefined**

* Quando se chama uma função q pede parâmetros e não se passa um parâmetro o valor considerado é undefined;
* !Undefined = true;
* Undefined não é considerado 100% = false;

const multiplication = ( a , b ) => {

    if( a === undefined || b === undefined){

        console.log("Undefined parameter")

    }else{

        console.log(`${a} \* ${b} = ${a \* b}`);

    }

}

const greeting = ( name ) => {

    if(!name){

        console.log("Ola!");

        return

    }

    console.log(`Ola ${name}!`);

}

1. **Default**

* Em JS é possível atribuir um valor padrão para o parâmetro;
* Ele só é usado se o usuário não passar um parâmetro;
* Para atribuir o default é só atribuir o valor nos parênteses da função;

const r = ( text = "Default text" , qt = 5 ) => {

    for( let i = 1; i <= qt ; i++ )

        console.log(text);

}

r();

r("test", 10);

1. **Closure**

* É criar uma função dentro de uma função;
* A função que está dentro fica limitada somente aquele escopo;

const multpClosure = (n) => {

    return (m) => {

        return n \* m;

    }

}

let c1 = multpClosure(3);

console.log(c1);

console.log(c1(10));

1. **ARRAYS E OBJETOS**
2. **Declaração de array**

* Let / const nomDaVar = [ valor , valor , valor ];

1. **O tipo de um array é object**
2. **Aqui o array não tem tipo especificado e pode receber vários tipos**
3. **Objetos literais**

* Para se criar objetos literais em JS se usa as chaves { };
* Os atributos não recebem atribuição por =, mas sim por :

const person = {

    name: "Matheus",

    age: 11,

    job: "Developer"

};

console.log(person);

console.log(person.name);

console.log(person.age);

console.log(person.job);

1. **Adicionar e deletar atributo de um objeto**

* **Add:** É só colocar nomeDoObjeto.novoAtributo = valor;
* **Deletar:** delete nomeDoObjeto.atributo;

const car = {

    engine: 2.0,

    brand: "VW",

    model: "Tigan",

    km: 20000

};

console.log(car);

car.doors = 4;

console.log(car);

delete car.km;

console.log(car);

1. **Object.assign – Faz uma cópia de um objeto**

const obj = {

    a: "test",

    b: true

};

const obj2 = {

};

Object.assign(obj2, obj);

console.log(obj2);

1. **Object.keys – Retorna o nome dos atributos do objeto em um vetor**

console.log(Object.keys(obj2));

1. **Object.entries – Retorna uma matriz com o nome do atributo na posição 0 e o valor daquele atributo na posição 1**

console.log(Object.entries(obj));

1. **Mutação**

* O valor da variável do objeto é uma referência;
* Caso se faça const x = { };
* E depois z = x;
* Mudanças em x afetarão z, pois ambos estão apontando para o mesmo objeto;

1. **Adicionar e remover valor do Array – PUSH, POP, SHIFT e UNSHIFT**

* nomeDoArray.unshift(valor) – Adiciona valor no início do array;
* nomeDoArray.push(valor) – Adiciona valor no final do array;
* nomeDoArray.shift() – Remove o primeiro valor e o retorna;
* nomeDoArray.pop() – Remove o último valor e o retorna;

1. **Encontrar elementos no vetor – INDEXOF e LASTINDEXOF**

* nomeDoArray.indexOf(valorProcurado);
* nomeDoArray.lastIndexOf(valorProcurado);
* **Se não existir retorna -1;**

1. **Cortar parte do array - SLICE**

* É tipo um split para array;
* Corta um pedaço do vetor e o retorna sem alterar o array original;
* Se o intervalo não existir ele retorna um vetor vazio { }

1. **ForEach**

* O foreach em JS é diferente, ele é formado pela estrutura:
* nomeDoArray.forEach(( nomeDoElemento ) => { corpo });

array2.forEach((number) => {

    console.log(number);

});

1. **Verificar se um vetor tem um valor – INCLUDES**

* nomeDoArray.includes(valor);
* Retorna true ou false;

1. **Inverter as posições do array / deixar ele de tras pra frente – REVERSE**

* nomeDoArray.reverse();
* Altera o vetor diretamente e não retorna nada;

1. **Métodos de String**

* Trim – Remove “ “ e os \n \t e etc (Retorna uma string);
* Split – Corta a String se baseando em um separador e retorna um array (retorna array). **Ex: nomeDaString.split(“ ”);**
* Join – Junta os elementos de um array em uma string e entre eles coloca um separador (retorna String). **Ex: nomeDoArray.join(“ ”);**

1. **Rest Operator / Rest Parameters**

* Usado para receber indefinidos argumentos em uma função;
* **Estrutura:** ...nomeDoConjunto;
* **O rest** converte os argumentos recebidos em um array com nome nomeDoConjunto;

1. **For of**

* É o for each das outras linguagens;
* No lugar dos : vai a palavra ‘of’;
* for(element of args)

1. **Destructuring**

* **Para obj:** Consiste em quebrar os atr do obj em várias variáveis sem ter q ficar especificando;
* **Para array:** Consiste em quebrar os índices do array em várias variáveis sem ter q ficar especificando;
* **Estrutura:** const / let { nomeAtr: nomeDaVar , nomeAtr } = obj;
* **Estrutura:** const / let [ nomeDaVar , nomeDaVar ] = array;

1. **JSON**

* É um dado em formato de texto;
* Usado para comunicação entre API e frontend;
* É obrigatório usar “ “;
* **Ex:**
* const myJSON = '{"name": "Matheus", "age": 11, "skills": [HTML"","CSS","PHP"]}';

1. **CONVERTER JSON p OBJETO e OBJETO p JSON**

* **Json para Object**
* const myObject = JSON.parse(myJSON);
* **Object para Json**
* const myNewJson = JSON.stringify(myObject);

1. **JS OO**
2. **Métodos**

* Para se declarar um método em um objeto usa-se a seguinte estrutura: nomeDoMetodo: function() { corpo }

const person = {

    name: "Alexandro",

    getName: function() {

        return this.name;

    },

    setName: function( name ){

        this.name = name;

    }

}

1. **Prototype**

* É um recurso do JS que permite a herança;
* Seu principal recurso é o **fallback;**
* **Ele faz com que se um recurso não é encontrado no objeto, ele o busque em seu pai;**

1. **Classe**

* Não é de fato uma classe;
* Aqui é criar um objeto de modelo e a partir dele criar novos objetos;
* A estrutura é a seguinte:
* Const novoObjeto = Object.create( modelo );

1. **Construtor**

* Criar uma função que crie um objeto vazio e atribua a ele as características desejadas;

function createPerson( name , age ){

    const person1 = Object.create({});

    person1.name = name;

    person1.age = age;

    return person1;

}

const bob = createPerson("Bob",21);

1. **Construtor moderno**

* Uso de new para criar o objeto;
* A classe é criada com function;

function Dogs( name , race ){

    this.name = name;

    this.race = race;

}

const dog1 = new Dogs("Toto","buldog");

1. **Adicionar atributos e métodos à classe depois de criada**

* nomeDaClasse.prototype.nomeDoMetodo = function(){
* **corpo**
* }

1. **Classe e construtor modernos ES6**

* Mais recente;
* Uso de class e constructor;
* Para declarar métodos não precisa usar function(){ };

class Book{

    constructor( title , author ){

        this.title = title;

        this.author = author;

    }

showDetails(){

        console.log(`Book: ${this.title}, Author: ${this.author}`);

    }

}

1. **Getters e Setters**

* Uso da palavra get e set;
* OBS: Em JS eles não são métodos, então na hora de usa-los não tem ();
* O set é feito com = e o get não possui os parênteses;

class Post{

    constructor( title , desciption ){

        this.title = title;

        this.description = desciption;

    }

    get getTitle(){

        return this.title;

    }

    get getDesciption(){

        return this.description;

    }

    set setTitle( title ){

        this.title = title;

    }

    set setDescription( description ){

        this.description = description;

    }

}

const post1 = new Post();

post1.setTitle = "Alice in Wonderland";

post1.setDescription = "It's a nice book!";

console.log(post1.getTitle);

console.log(post1.getDesciption);

1. **Herança**

* Em JS também se usa extends;
* Contudo, aqui é obrigatório a chamada de super() dentro do filho;

1. **DEPURAÇÃO EM JS**
2. **Strict**

* É um modo de desenvolvimento de JS que torna as suas regras mais rigorosas;
* Pode ser declarado no topo do arquivo ou em funções específicas;
* Basta digitar: “use strict”;
* OBS: Com os “ ”

1. **O console.log() é muito usado para debug**
2. **Debugger**

* É possível para o código em certas partes digitando **debugger;**
* O código congela na linha debugger até o programador avançar;
* O código pode ser acompanhado na guia source;
* Passando o mouse em cima dar variáveis da pra ver o valor;

1. **Validação de tipo**
2. **Exceptions**

* Pode ser lançada em uma situação especifica pelo programador;
* Throw new Error(“Mensagem”);

1. **Try Catch**

* Bloco try e catch;
* No catch vai () e dentro dele uma variável que guarda o erro indentificado;

try{

    let a = b;

}catch(e){

    throw new Error("Erro! Description: " + e);

}

1. **Se movendo pelo DOM**
2. **Document.body**

* Permite a seleção de elementos estruturais do HTML;

**// 1- Moving by DOM**

**console.log(document.body);**

**console.log(document.body.childNodes[1]);**

**console.log(document.body.childNodes[1].childNodes[1]);**

**console.log(document.body.childNodes[1].childNodes[1].textContent);**

1. **Seletor de elementos**

* Da pra selecionar os elementos por tag, id, classe e seletor de CSS;
* getElementsByTagName;
* getElementByID;
* getElementsByClassName;
* querySelector;

**// 2- getElementsByTagName**

**const listItens = document.getElementsByTagName("li");**

**console.log(listItens);**

**// 3 - getElementById**

**const title = document.getElementById("title");**

**console.log(title);**

**// 4 - getElementByClassName**

**const products = document.getElementsByClassName("product");**

**console.log(products);**

**// 5 - QuerySelectorAll**

**const productsQuery = document.querySelectorAll(".product");**

**console.log(productsQuery);**

**const mainContainer = document.querySelector("#main-container");**

**console.log(mainContainer);**

1. **Alterando o HTML usando o DOM**

* insertBefore, appenChild e replaceChild;
* Necessidade de seleção de elemento pai do elemento que vai receber a alteração;

**// 6 - Insert Before**

**const createP = document.createElement("p");**

**const header = title.parentElement;**

**console.log(header);**

**header.insertBefore( createP , title );**

**// 7 - AppendChild**

**const navLinks = document.querySelector("nav ul");**

**const createLi = document.createElement("li");**

**navLinks.appendChild( createLi );**

**// 8 - replaceChild**

**const h2 = document.createElement("h2");**

**h2.textContent = "Lorem Ipsum Test";**

**header.replaceChild( h2 , title );**

1. **Element.textContent**

* Coloca conteúdo dentro da tag;
* NomeDoElemento.textContent = “Texto”;

1. **SetAttribute**

* Altera ou coloca um atributo na tag;
* NomeDoElemento.setAttribute( “atributo” , “valor” );

1. **Pegando posição do elemento na tela**

* nomeDoElemento,offsetWidth;
* nomeDoElemento.offsetHeigth;

1. **Aplicando Styles em elementos**

* Elemento.style.’nomeDoEstilo’ = “valor”;

1. **Eventos**
2. **Adicionando eventos**

* addEventListener;
* Sempre se adicionar evento em alguma coisa, podendo ser **window, elemento ou document**;
* Coisa.addEventListener( “meioDeAtivação” , função );

**const btn = document.querySelector("#my-btn");**

**btn.addEventListener("click" , function() {**

**console.log("Hello!");**

**});**

1. **Removendo eventos**

* removeEventListener;

**secondBtn.removeEventListener("click", imprimeTest);**

1. **Evento de Teclas (Keyup e Keydown)**

* e pega informações da função;
* Nesse caso e pega a tecla pressionada;
* Usado em document, pois não possui elemento de uso;

**document.addEventListener("keyup", (e) => {**

**console.log("Released the key: " + e.key);**

**})**

1. **Eventos por pressionar mouse**

* Evento de click é em um objeto específico e leva em consideração o momento do click;
* Evento de pressionar mouse pode ser click na hora, click após soltar e double click;

**h4.addEventListener("mousedown" , () => {**

**console.log("Pressed the mouse!");**

**});**

**h4.addEventListener("mouseup" , () => {**

**console.log("Released the mouse!");**

**});**

**h4.addEventListener("dblclick" , () => {**

**console.log("Double Click!");**

**});**

1. **Mousemove e Scroll Event**

* Mousemove pra qualquer movimentação de mouse na tela;
* Scroll identifica o movimento vertical na tela por Scroll do mouse ou seta;
* Usado em document pois não possui elemento de uso;

**// 6 - Mousemove**

**document.addEventListener("mousemove", (e) => {**

**console.log("No Eixo X: " + e.x);**

**console.log("No Eixo Y: " + e.y);**

**})**

**// 7 - Scroll Event**

**window.addEventListener("scroll", (e) => {**

**if( window.pageYOffset > 200 ){**

**console.log("Passou de 200 em Y");**

**}**

**})**

1. **Evento de seleção e saída de objeto (usado em input)**

* Focus e blur;

**myInput.addEventListener("focus", (e) => {**

**console.log("In input!");**

**})**

**myInput.addEventListener("blur", (e) => {**

**console.log("Out input!");**

**})**

1. **Evento de carregamento de tela**

* Load;
* Usado em window, ele que é o elemento;

**window.addEventListener("load", () => {**

**console.log("Window loaded");**

**})**