**ANOTAÇÕES LÓGICA DE PROGRAMAÇÃO**

**Javascript:**

<script> entender que a linguagem utilizada é o javascript

String é o termo técnico quando falamos de texto no Javascript

F12= abre o debbuger no navegador; selecionar depois console para verificar o erro do code

Ctrl+N = criar novo arquivo no sublime

Crtrl+S = salvar arquivo no sublime

Crtrl+R = recarregar navegador

Ctrl+O = abre arquivo para carregar link do code

document.write(“ “); para escrever um texto no mundo Javascript

alert(‘texto’); para criar um pop-up no navegador com o texto inserido

var x = ; para variável

function x() { } função para criar algum atalho ou uma função

// significa um comentário, sendo ignorado pelo código.

Calcula(x,y) atribui valores as variáveis da função

Return é pra retornar ao valor da fórmula (lado esquerdo da equação)

Prompt gera um pop-up permitindo ao usuário inserir um valor (Sempre retorna como um string)

parseInt( ); converte um número em formato de texto no PROMPT para um número real

== é utilizado como símbolo de igualdade, já o símbolo (=) significa “recebe” algo

&& remete a junção de duas condições (se isso e aquilo)

!= significa que é diferente na linguagem javascript (algumas linguagens utilizam <>)

If ( ) { } para utilizar uma condição

Else { } para o “se não”

Math.random ( ); gera um número aleatório.

Math.trunc( ); desconsidera os números decimais.

Math.PI retorna o número de *pi* (3.14).

Math.pow(X, Y); faz a exponenciação de 2 números. Ex: math.pow (4, 2); = 4^2 = 16.

Math.sqrt(X); retorna a raíz quadrada de X.

Math.round( ); para arredondar o número.

* Math.floor( ); para arredondar o número para baixo.
* Math.ceil( ); para arredondar o número para cima.

Math.min( ); retorna o menor valor dentre os números informados.

* Math.max( ); retorna o maior valor dentre os números informados.

Console.log(parametro); ajuda quando fazer um teste pois mostra o resultado no console do navegador;

While(operação) { } repete o que deseja enquanto a operação descrita for verdadeira

isNaN( ) é uma função que recebe um parâmetro e retorna true ou false caso o valor seja NaN ou não;

for (espaco1; espaco2; espaco3) { }

\*espaco1 é a variável a ser incrementada;

\*espaco2 é a condição de repetição;

\*espaco3 é o pós incremento;

Switch(var){ } para verificar diversas possibilidades até encontrar a true.

numero++ é a mesma coisa de numero = numero + 1 (para aumentar uma variável, isso se chama pós incremento)

parseFloat( ); converte um texto em número, assim como o parseInt( ) porém mantem as casas decimais.

querySelector(“parâmetro”) traz um termo do HTML para o JAVASCRIPT

onClick; significa “no clicar”, ou seja, executa algo quando clica no botão.

onChange; é utilizado para realizar uma ação após alguma mudança.

onLoad; é utilizado quando queremos carregar algo simultaneamente ao carregamento da página. OBS: Exclusivo para ser utilizado dentro da tag <body>.

onKeyUp; permite identificar qual tecla está pressionada. Primeiro exibe o caractere e depois executa a ação.

onKeyDown; permite identificar qual tecla está pressionada. Primeiro executa a ação e depois exibe o caractere.

* onKeyPress; tem a mesma função do OnKeyDown, porém não interpreta as teclas F1 até F12, shift, alt, ctrl...

var.innerHTML; retorna o conteúdo de dentro da variável, sem levar em consideração fonte de texto, etc.

input.focus( ) faz com que já selecione a caixa de texto sem o usuário precisar clicar

pincel.fillStyle define a cor a ser pintada;

pincel.fillRect (Xo, Yo, distancia X, distancia Y) define as corrdenadas do retângulo a ser pintado;

pincel.fillStroke define a cor da linha do perímetro;

pincel.strokeRect (Xo, Yo, distancia X, distancia Y) define o perímetro a ser desenhada a linha;

pincel.beginPath( ) utilizada para mexer no ponto do pincel;

pince.moveTo move a coordenada do pincel;

pincel.lineTo para demarcar a área a ser pintada;

pincel.fill ( ) quando finalizar as coordenadas, irá pintar;

pincel.fiilStyle para escolher a cor a ser pintada

pincel.arc (Xo, Yo, Raio, 0, ângulo do círculo (2 \* Math.PI)) para pintar um arco/círculo;

pincel.font para escolher tamanho e fonte da texto no canvas;

pincel.fillText para desenhar o texto;

pincel.clearRect para limpar a tela (intuito de causar uma animação na tela);

evento.pageX e evento.pageY identificam as coordenadas clicadas com o mouse no *canvas;*

evento.shiftKey para saber se a tecla SHIFT está sendo pressionada;

evento.altKey para saber se a tecla ALT está sendo pressionada;

onmousedown é o evento disparado quando o botão esquerdo do mouse é pressionado;

onmouseup é o evento disparado quando o botão esquerdo do mouse é solto;

setInterval (x, y); X é a função que desejamos que o JS chame em um intervalo de tempo e Y é o intervalo de tempo em milissegundos;

setTimeout ( ); define um intervalo de tempo para executar uma função somente uma vez, assim que o intervalo acabar;

símbolo ! altera a função boolean (o que é true vira false);

keyIsDown remete a uma tecla pressionada no teclado;

*str*.toUpperCase( ); retorna o valor da string original em letras Maiúsculas.

* *str*.toLowerCase( ); retorna o valor da string original em letras minúsculas.

String.indexOf(elementoBuscado); retorna a posição do elemento buscado na string

String.lastIndexOf(elementoBuscado); retorna a última posição do elemento buscado na string (para o caso de haver o mesmo elemento na string, como letras repetidas).

str.slice(posição); retorna a string da posição informada até o final

* str.slice(posição1, posição2); retorna a string contida entre a posição1 e posição2

nomeDaVariavel.push(elementoDaLista); para inserir o “elementoDaLista” na array.

nomeDaVariavel.splice(X, Y); para remover um item da array.

* X é a posição do elemento na lista
* Y é a quantidade de elementos que se deseja remover na lista

As classes em JS são utilizadas quando queremos repetir um mesmo conceito com dados diferentes (como se fosse um molde). Ex: Inserir um conjunto de informações de clientes em um banco (nome, CPF, agencia, conta...)

* Atributos/propriedades são as “variáveis” dentro da classe. Ex: Nome, CPF, agencia, conta...

Em orientação a objetos, funções também são chamadas de métodos.

* Parâmetros/argumentos são os valores recebidos dentro da função.

Quando um atributo de uma classe está com o símbolo “\_” antes do nome, significa que é um atributo privado e não deve ser mexido. Ex: \_saldo.

Utilizamos “set” quando queremos atribuir um valor e “get” quando queremos ler um valor, sem ser possível alterá-lo.

* Ambos recebem um return para retornar o valor.

typeof variável; para saber qual o tipo da variável (number, string, boolean, undefined, object).

Number(variavel); transforma a variável que está em formato String, para um formato de número. Outra opção é +variável, no qual o símbolo de “+” realiza a função de alterar a String em um número.

* String(variável); transforma a variável que está em formato number, para um formato de string. Outra opção é variável.toString( );

Categorias de erros apresentados:

* RangeError; quando o código recebe um tipo de dado, porém não no formato aceitável. Ex: deve receber números maiores que zero, mas recebe -1.
* ReferenceError; normalmente quando o código tenta acessar algo que não existe. Ex: variável que não foi definida, erros de digitação ou confusão nos nomes utilizados...
* SyntaxError; na maior parte dos casos ocorre quando há erros no programa e o JS não consegue executá-lo. Os erros podem ser propriedades ou métodos escritos ou utilizados de forma incorreta. Ex: esquecer de fechar chaves ou colchetes.
* TypeError; indica que o código esperava receber um tipo de dado, mas recebeu outro. Ex: deveria receber uma string mas recebeu um número ou booleano.
* Utilizando operador ternário:

No exemplo abaixo, utilizando o IF normalmente

if (idadeCliente >= idadeMinima) {

Console.log(“cerveja”) }

Else { console.log(“suco”) }

Com o operador ternário, fica

Console.log (idadeClinete >= idadeMinima ? “cerveja” : “suco”)

* Utilizando Template String:

Const nome = “Caio”;

Const idade = 26;

Const cidade = “Recife”;

Const apresentacao = `meu nome é ${nome}, minha idade é $(idade} e nasci na cidade de $(cidade}`;

* Utilizando Arrow Function:

Um exemplo de função normalmente escrita fica

Function apresentar (nome) {

Return `meu nome é ${nome}`; }

Utilizando no modelo Arrow Function, fica

const apresentarArrow = nome => `meu nome é ${nome}`;

Outro exemplo de Arrow Function. OBS: Quando a função possui somente uma instrução, não é necessário utilizar as chaves { }

const soma = (num1, num2) => num1 + num2;

Um exemplo de Arrow Function com mais de 1 instrução, fica

const somaNumerosPequenos = (num1, num2) => {

if (num1 || num2 > 10) {

return “somente números de 1 a 9” }

else {

return num1 + num2; } }

Array (ou vetor) é definido como uma lista ordenada de valores enumerados em que cada valor é chamado de elemento e cada elemento se localiza numa posição na lista chamada de índice. São definidas através dos colchetes [ ]; As arrays possuem métodos que são funções que servem como uma propriedade para a própria array. Os principais são:

* .length; para contar quantos elementos há dentro da array
* .push( ); para “empurrar” um elemento para dentro da array. No caso, insere um novo elemento para a array ficando na última posição.
* .pop( ); retira o último elemento do array. OBS: Altera o array original removendo o elemento.
* .concat( ); junta dois arrays, colocando o array passado como argumento logo depois do primeiro. OBS: Não altera o array que foi chamado, sendo preciso salvar o resultado em um novo array.
* .filter( ); retorna uma lista contendo todos os elementos que passaram em um teste, ou seja, em uma função escrita por nós previamente. OBS: Não altera o array que foi chamado, sendo preciso salvar o resultado em um novo array.
* .find( ); funciona de forma similar ao filter, porém retorna apenas o primeiro valor que satisfizer o teste, podendo ser string ou número.
* .indexOf( ); localiza e retorna o índice referente à primeira ocorrência de determinado valor em um array. OBS: Caso não exista nenhuma ocorrência do valor, retornará -1.
* .lastIndexOf( ); funciona da mesa forma que o indexOf, porém retorna o índice referente à ultima ocorrência de um valor em um array, varrendo o array de trás para frente.
* .forEach( ); executa uma função em cada elemento do array de forma individual. Não altera o array original e nem retorna um valor, deixando esse trabalho a cargo da função escolhida.
* .shift( ); retira o primeiro elemento do array. OBS: Altera o array original removendo o elemento e trocando o índice de todos os elementos para menos um do que eram.
* .unshift( ); funciona igual ao push porém adiciona na primeira posição e acaba trocando o índice de todos os elementos. Altera o array original com o novo valor.
* .reduce( ); utiliza uma função que foi definida previamente pelo usuário em cada um dos elementos, guardando o resultado em uma variável que pode ser acessada dentro dessa função que foi definida, retornando um único valor no final, reduzindo a array para um único valor.
* .reduceRight( ); funciona igual ao reduce porém começa do final da array até o início.
* .reverse( ); inverte a ordem dos elementos do array.
* .slice( ); copia uma parte do array para outro array.
* .sort( ); organiza de acordo com a classificação Unicode, onde os números vêm antes das letras. OBS: Não funciona corretamente para ordenar os números, sendo necessário definir uma função para auxiliar o comando.
* .splice( ); consegue remover um ou mais elementos consecutivos caso o segundo parâmetro tenha um valor maior que 0. Também inclui um ou mais elementos a partir de um índice escolhido.
* .includes( ); recebe uma array e busca o parâmetro informado dentro da array. Basicamente verifica se existe o elemento (parâmetro informado) dentro da array.

Em uma tag <áudio> escreva controls, dentro da tag, para fazer aparecer a tag no html (aparecer o botão de play, pause, etc).

document.querySelector(‘#id\_da\_tag\_audio).play( ); neste exemplo estamos chamando a função *play* na tag áudio, ou seja, estamos querendo que o navegador reproduza o som que chamamos pelo *id*.

querySelectorAll( ); utilizamos quando queremos selecionar todos os elementos do seletor desejado, reduzindo a quantidade de repetição de código.

“use strict” é o conhecido strict mode no qual traz erros que ficam escondidos no JS. As veses alguns erros aparecem embutidos e não são acusados pelo navegador, e com o “use strict” ele acusa no próprio JS o erro.

.classList[X]; utilizamos essa propriedade no JS quando queremos chamar uma classe de índice X em uma tag. Ex: algumas tags possuem mais de 1 classe, sendo assim, utilizamos o *classList* para informar qual classe que queremos chamar.

**Html:**

<br> é uma tag html para pular linha; br é abreviação de break

<hr> tag para gerar uma linha de separação

<h1> heading (utilizado para o título), existem 6 níveis de heading;

<p> utilizado para parágrafo;

</> ao final do fechamento da tag

<a> tag ancora;

<strong> utilizada para dar importância ao texto demarcado (deixa em negrito, dando ênfase);

<em> deixa o texto demarcado em *itálico*;

<html> para demarcar o texto que está dentro da estrutura html;

<!DOCTYPE html> no início do programa para passar informações ao navegador sobre qual versão do html estamos usando (html5 a mais moderna);

<meta charset="UTF-8"> resolver problemas de acentuação;

<lang = “pt-br”> para informar que o texto é na língua portuguesa;

<title> utilizado para mudar o título da aba html;

<input/> gera uma caixa de texto para o usuário escrever;

Tipos de inputs:

* <input type=’name’/> para escrever um nome qualquer
* <input type=’email’/> para escrever email. É obrigatório inserir @ na caixa de texto
* <input type=’tel’/> para escrever número telefônico
* <input type=’number’/> para escrever números como uma idade. Só é permitido números
* <input type=’url’/> para inserir um link
* <input type=’range’/> para inserir uma barra de urgência por exemplo
* <input type=’checkbox’/> para inserir uma caixa de seleção. É permitido o usuário selecionar mais de uma opção.
* <input type=’radio’/> para inserir uma caixa de seleção. Deve-se colocar o mesmo name para que o usuário só consiga selecionar uma opção.

<input type=’text’ placeholder=’Nome’/> faz com que apareça a palavra Nome na caixa de texto

<button> texto </button> aparecerá um botão com o texto dentro;

Tipos de button:

* <button type=’button’>Texto</button> não executa nada. É somente um botão
* <button type=’reset’>Texto</button> Limpa as informações descritas nas caixas de texto e caixas de seleção
* <button type=’submit’>Texto</button> Envia as informações inseridas para algum lugar

<canvas> </canvas> é uma tag para desenhar com um pincel e escrever dentro do HTML;

Width é a largura;

Height é a altura;

<head> utilizado para inserir informações que queremos passar ao navegador;

<body> utilizado para inserir informações que queremos exibir na página;

Existem as listas ordenadas <ol> e as listas não ordenadas <ul> para listar algo. Além disso é necessário inserir o <li> em cada item para identificar os itens da lista (list iten);

Utiliza-se a tag <div> para dividir os conteúdos em blocos;

<header> tem a mesma função da <div> porém é utilizado como tag para o cabeçalho da pág;

<main> é a tag utilizada para o conteúdo principal da pág;

<footer> é a tag para o rodapé da pág;

<form> é a tag para inserir formulários;

<span> tag usada para agrupar textos, e não tem nenhuma alteração visual por padrão.

<label> tag que está sempre em par com o <input> (o usuário clica no label e a caixa de texto do input é selecionada, facilitando a UX).

<nav> é uma tag utilizada para demarcar os links de navegação;

<textarea></textarea> é uma tag de conteúdo, diferentemente do <input> (que não precisa ser fechado) na qual escreve-se em mais de uma linha

<textarea cols=”00” (número de colunas) rows=”00” (número de linhas)><textarea>

<select></select> para inserir uma caixa com itens a serem selecionados

<option>Iten a ser selecionado dentro da caixa</option>

Dentro de um <form>, semanticamente utiliza-se <fieldset> no lugar da <div> e <legend> no lugar do <p>

<table> tag usada para uma tabela

<caption> tag usada para o título da tabela

Dentro da tabela, inserir a tag <tr></tr> para as linhas e <td></td> para células

<thead></thead> para definir o cabeçalho da tabela

<th></th> para definir a célula do cabeçalho da tabela

<tbody></tbody> para definir o corpo da tabela

<tfoot></tfoot> para definir o rodapé da tabela

<td colspan=’X’> para mesclar a célula em X colunas da mesma linha.

<td rowspan=’X’> para mesclar a célula em X linhas da mesma coluna.

<section> é como a <div> porém é utilizada para um bloco com conteúdo semântico. Mesmo significado, mesmo sentido

<figure> é a tag usada quando se quer mostrar algo. OBS: Não necessariamente utiliza-se a tag <img> dentro do <figure>.

* <figcaption> é a tag usada para colocar legenda abaixo da imagem ou do conteúdo do <figure>. Ex: <figurecaption> Foto de Caio Lagreca</figurecaption>

<Picture> é a tag similar ao <figure> porém é utilizada quando quer inserir uma <img>

<meta name=’viewport’ contente=’width=device-width’> dentro do <head> para informar ao servidor que o site também é para ser visualizado em dispositivo mobile

<meta name=’viewport’ contente=’width=device-width initial-scale=1.0’> para informar o zoom que a página irá ter no mobile. OBS: Sempre colocar 1.0 para garantir que a escala da página estará sempre proporcional ao tamanho da tela no dispositivo.

Para inserir a funcionalidade de um número telefônico dentro de um <a>, deve-se inserir <a href=’tel:+55....’></a>

Para inserir a funcionalidade de um email dentro de um <a>, deve-se inserir <a href=’mailto:...></a>

Para colocar um vídeo baixado e inserir os botões de play, pause etc, deve-se <video src=’img/nome-do-video.formato’ controls></video>

**CSS:**

OBS - há 3 formas de configurar o CSS:

* A 1ª é através do chamado CSS *inline*, no qual você insere as propriedades dentro da tag html, na própria linha do código.
* A 2ª forma é inserindo a tag <style> dentro do <head> e colocando a propriedade desejada.
* A 3ª forma é através da criação de um novo arquivo (geralmente chamado de style.css) no qual você irá inserir as informações no arquivo e dentro do código principal, na parte do <head> você irá inserir <link rel=”stylesheet” href=”style.css”>

A representação das cores pode ser feita de 3 maneiras:

* Através da linguagem hexagonal (0123456789ABCDEF): # \_ \_ \_ \_ \_ \_ (no qual cada 2 espaços representa as cores principais: red, green, blue)
* Através da linguagem RGB: rgb (\_, \_, \_) no qual cada espaço representa o nível das cores red, green e blue, indo do nível 0 até 255.
* Através do nome das cores

rgba(\_, \_, \_, \_) agora é possível colocar opacidade nas cores, através do “a”

Opacity: X; para deixar o elemento opaco (mais transparente). O número vai de 1 (100% do objeto) até 0 (objeto invisível).

width: 00px ou 00%; para aumentar a largura do objeto

height: 00px ou 00% para aumentar a altura do objeto

Tipos de Fontes:

* font-size: 00px; para aumentar o tamanho da letra
* font-weight: bold; deixa as letras em negrito
* font-family: serife, sans-serife, cursive, fantasy; São os 4 tipos de fontes existentes em todo navegador. São utilizadas como fonte secundária (fallback) caso o navegador não possua a fonte informada.
* font-face:
* font-variant: caps; para mudar o tamanho e variação das letras.
* font-style: italic, oblique, normal; São os 3 tipos para mudar o estilo da fonte.

Tipos de Texto:

* text-align: start, end, center, justify; para alinhar o texto.
* text-indent: 0rem; para definir a identação do texto (Ex: para um espaçamento de parágrafo).
* text-decoration: none; para não colocar nenhuma decoração no texto (ex: tirar o sublinhado do texto)
  + Text-decoration: underline; para deixar o texto sublinhado
* text-transform: uppercase, lowercase, capitalize; para torna as letras maiúsculas, minúsculas, somente a primeira letra maiúscula...
* text-shadow: X Y color; para criar sombras no texto.

Letter-spacing: 0rem; para inserir um espaçamento entre as letras.

Tipos de Background:

* Background-repeat: no-repeat; para a imagem de fundo não ficar repetindo, caso ela não caiba toda na tela desejada.
* Background-position: center; para definir a posição da imagem de fundo (no caso ficará no centro).
* Background-size: cover; para fazer a imagem ocupar toda a tela desejada.
* Background-image: url(‘exemplo.png’) para definir a imagem de fundo.
* Background-color: #\_ \_ \_ \_ \_ \_; para mudar a cor do fundo da tela.
  + Background-color: unset; para deixar o fundo transparente.

Background: linear-gradiente(color1, color2); para realizar um degradê no fundo da tela indo da cor1 para a cor2.

Margin: \_ \_ \_ \_ ; (define a margem para o objeto, primeiro pra cima, depois direita, depois pra baixo e por ultimo pra esquerda) OBS: Não coloca vírgula entre os números.

Display: inline; deixa os itens da lista lado a lado

Display: block faz com que tenha somente o item determinado na linha

Margin é o espaçamento externo

Padding é o espaçamento interno

Border: 1px solid black; para definir a espessura, estilo e cor da borda simultaneamente.

* Border-width: 0rem; para definir a espessura da borda
* Border-style: solid, dashed, doted, double; para definir o estilo da borda.
* Border-radius: 00px; para colocar bordas arredondadas.
* Border: none; para tirar a borda do objeto.

Transition: 0s xxxx; para realizar alguma ação na transição (ex: mouse passar por cima do objeto)

Cursor: pointer; mouse muda para mãozinha clicável

Transform: scale(00); aumenta o objeto de acordo com a escala

Transform: rotate (00deg) rotaciona o objeto de acordo com os graus informados (OBS: deg=degrees que é grau em inglês)

Hierarquias do CSS: Qual prioridade o CSS vai dar

* ID é o mais forte com força 100
* Class tem força 10
* Tag tem força 1

Line-height: 00; para aumentar o espaçamento entre as linhas

box-shadow: X Y color; para criar sombras no elemento

* box-shadow: inset X Y color; para criar sombra interna no elemento

list-style: none; é a propriedade para tirar as bolinhas ao lado dos links que estão contidos na tag <a>

word-wrap; essa propriedade é utilizada para especificar se o navegador pode ou não quebrar linhas dentro das palavras, afim de prevenir o vazamento quando uma sequencia de caracteres é muito longa para caber na caixa que está contida.

* Word-wrap: normal; indica que as linhas só podem quebrar em pontos de quebra normais de palavras.
* Word-wrap: break-word; indica que as palavras normalmente inquebráveis podem ser quebradas se não houver pontos de quebra de outra forma aceitável.

Word-break; essa propriedade é utilizada para especificar se o navegador deve inserir ou não quebras de linha, onde, normalmente, o texto vazaria de seu container.

* Word-break: normal; usa a regra de quebra de linha padrão.
* Word-break: break-all; quebras de linha podem ser inseridas entre quaisquer caracteres de texto.
* Word-break: keep-all; não permite quebra de linha.
* Word-break: break-word; para evitar vazamento, palavras que normalmente não seriam quebradas podem ser quebradas em pontos arbitrários se não houver pontos de quebra válidos na linha.

Pseudo-elementos:

* :hover { } para quando o mouse passa por cima do objeto
* :active { } para quando o mouse é pressionado no objeto
* :visited { } para quando um link já for visitado/clicado anteriormente
* :nth-child( ); para editar a linha informada no ( )
* :first-child { } para editar somente a primeira linha
* :first-letter{ } para editar somente a primeira letra da frase
* :before{ contente: “XX” } para inserir a informação XX antes do conteúdo
* :after{ contente: “XX” } para inserir a informação XX antes do conteúdo

Design responsivo: adaptar o site para desktop e celular

<meta name='viewport' content='width=device-width'>

@media screen and (max-width: 480px) { }

@media screen and (min-width: 000px) and (max-width: 000px) { } para definir propriedades em um dispositivo especifico.

* Tamanho mínimo de um tablet: 768px;

Display: none; faz com que o seletor não apareça na tela (Ex: ocultar uma imagem)

No código CSS:

* --nomeDaVariavel: valor; (inserir uma variável no CSS. Ao invés de ficar colocando o código da cor, você pode atribuir uma variável para identificar qual a cor. Ex –azulClaro: #DFE3E5; )

Display: grid; faz com que o container fique dividido em grids (células).

Grid-template-columns: N; faz com que o cotainer se divida em N colunas (informar antes display:grid). Ex grid-template-columns: 1fr 1fr; (divide o container em um grid com 2 colunas). OBS: “fr” é a unidade de medida fração.

* Fazendo por exemplo grid-template-columns: auto 1fr; a primeira coluna irá se ajustar automaticamente com o tamanho do maior elemento. É interessante para uma célula que está em constante alteração, como um contador de minutos ou visualizações.
* Grid-template-columns: repeat(n, 1fr); irá repetir “1fr” n vezes. Por exemplo grid-template-columns: repeat(3, auto); tem a mesma funcionalidade que grid-template-columns: auto auto auto;

Grid-template-rows: N; tem a mesma função acima, porém para as linhas.

Grid-column: span N; faz com que o elemento mescle as células (similar a funcionalidade do excel de mesclar células). O “N” é o número de células que serão mescladas.

Grid-row: span N; tem a mesma função acima, porém para mesclar as linhas.

Gap: 0px; para dar um espaçamento padrão entre os elementos do grid. Essa propriedade deve ser inserida na classe do container, fazendo com que os elementos filho tenham margens de 0px. OBS: Essa propriedade é similar ao justify-content: space-between no sentido de não dar espaço entre o elemento e as bordas do container, dando espaçamento somente entre os elementos.

* Row-gap: 0px; para dar um espaçamento padrão de 0px no grid somente entre as linhas.
* Column-gap: 0px; para dar um espaçamento padrão de 0px no grid somente entre as colunas.

**Flexbox:**

* Display: flex; deixa os elementos filho um do lado do outro e com a mesma altura
* Align-items: center; alinha os elementos filho dentro do flex no centro (verticalmente)
* Justify-content: space-between; insere o espaço vazio entre os elementos filho (deixa os objetos afastados. Ex: cada objeto em um lado extremo da tela)
* Justify-content: space-around; insere o espaço vazio em volta dos elementos filho (tanto para a esquerda, quando para a direita)
* Flex-direction: row; dispõe os elementos filhos um ao lado do outro (como ocorre normalmente)
* Flex-direction: column; dispõe os elementos filhos um embaixo do outro. OBS: É necessário primeiro inserir o display: flex;
* Flex-wrap: wrap; quebra os elementos filhos para a próxima coluna, não extrapolando o tamanho do elemento pai
* Flex-flow: XXX YYY; une as propriedades flex-direction (XXX) com flex wrap (YYY)
* Order: 0; para alterar a ordem do elemento (por padrão todos os elementos estão em ordem 0; caso queira colocar um item primeiro, a ordem deve ser -1; caos queira colocar o item depois de todos a ordem deve ser 1). OBS: Propriedade utilizada nos filhos.
* Flex-grow: 1; faz com que o item cresça o tamanho que sobrou na página. OBS: Propriedade utilizada nos filhos.
* Flex-shrink: 1; faz com que o item diminua o tamanho que sobrou na página. OBS: Propriedade utilizada nos filhos.
* Flex-basis: \_; tem a mesma função do width.
* Flex: \_ \_ \_; união das propriedades flex-grow, flex shrink e flex-basis.
* Propriedades utilizadas nos pais (flex-container):
  + Display: flex; flex-direction; justify-content; flex-wrap; flex-flow; align-items; align-content.
* Propriedades utilizadas nos filhos (flex-items):
  + Order; flex-grow; flex-shrink; flex-basis; flex; align-self.

Para juntar todos os documentos css em um único arquivo css, utiliza-se:

@import url(nomeDosArquivos.css);

**Grid:** Para fazer grids (blocos) no layout, utilizamos o display grid.

* Aplica-se o *display: grid* no elemento pai.
* Ainda no elemento pai, insere-se a propriedade *grid-template-areas:* e recebe os nomes das áreas que estarão em blocos. Ex: cabecalho, conteudo, rodape...
* Para definir o tamanho das colunas, insere a propriedade *grid-template-columns:* e recebe o tamanho (quando for um tamanho automático, colocar *auto*)
* No caso das linhas, a propriedade é *grid-template-rows*.
* Quando se há mais de 1 linha ou mais de 1 coluna, deve-se colocar os tamanhos de cada coluna/linha.
  + Ex: grid com 4 colunas e 3 linhas. grid-template-columns: auto auto auto auto; grid-template-rows: 50px auto auto;
* Nos elementos filhos, inserir a propriedade *grid-area:* no qual recebe o nome específico de cada elemento filho. Ex: cabecalho, conteudo, rodape...
* Para definirmos o tamanho dos blocos (ex: uma imagem ficar preenchida em 3 colunas e 2 linhas; um bloco for mesclado em 2 linhas...) utilizamos grid-column-start e grid-column-end (para colunas); grid-row-start e grid-row-end (para as linhas). OBS: a posição final deve ficar no ultimo bloco+1. Exemplo: querer fazer com que uma imagem ocupe 3 colunas e 2 linhas de um grid 4x4.
  + Grid-column-start: 1;
  + Grid-column-end: 4;
  + Grid-row-start: 1;
  + Grid-row-end: 3;
    - Ou utilizar os atalhos grid-column: 1 / 4; grid-row: 1 / 3.
    - Caso o bloco seja somente em 1 coluna ou linha, não é necessária a propriedade *end,* ou seja, basta somente invocar o grid-column-start ou grid-row-start e informar o número da coluna/linha.
* Grid-gap é uma propriedade que utilizamos para definir espaçamentos entre os blocos. (utilizada no elemento mãe)

**GIT e GITHUB:**

Antes de qualquer interação com o git, é necessário informar quem sou eu, para armazenar corretamente os dados e as alterações no código. Faça isso através dos comandos:

git config --local user.name "Seu nome aqui"

git config --local user.email "seu@email.aqui"

Em seguida para acessar a pasta que deseja versionar, bastar escrever “cd”+espaço e arrastar a pasta para dentro do Git Bash e depois apertar ENTER.

* Escreva “cd ..” para voltar uma pasta.

Escreva “ls” para ver os arquivos que foram para o Git que estão dentro da pasta.

Escreva “git init” para fazer o Git iniciar um repositório.

Escreva “git status” para ver o estado atual do repositório (se há algum arquivo para ser comitado).

Escreva “git add .” para adicionar todos os arquivos da pasta para dentro do Git.

Escreva “git commit -m “mensagem” para salvar o arquivo comitado e informar brevemente em “mensagem” qual foi a alteração realizada.

* Commit significa que houve alguma modificação no repositório.

Caso já tenha realizado o primeiro commit (ou seja, esteja fazendo um update do projeto), após fazer o git commit, escreva “git push” para enviar ao github.

Escreva “git log” para ver todas as alterações que foram feitas e comitadas no Git.

* “git log –oneline” para mostrar os comits de forma resumida (cada comit em uma linha).
* “git log -p” para mostrar os comits de forma mais detalhada (mostrar também quais alterações foram realizadas). OBS: Para sair dessa opção, bastar apertar a tecla Q

Escreva “git config –local” para alterar as configurações locais do Git.

* “git config –global” para alterar as configurações globais do Git.

Salve um arquivo com o nome “.gitignore” para o Git ignorar esse arquivo.

Escreva “git remote add (nome do repositório. Ex: local ou remoto) caminho/para/repositório

* Escreva “git remote -v” para ver o local do repositório adicionado

Escreva “mkdir” para utilizar o comando de criar uma nova pasta

Escreva “git init –bare” para chamar um arquivo que só contem as alterações, não contendo uma cópia do arquivo.

Escreva “rm -rf .git” para remover um repositório adicionado.

Escreva “git clone endereço/documento/” para clonar o repositório. Ex: trazer pela primeira vez os dados de um repositório remoto para repositório local.

Escreva “git push +(nome do repositório)+(branch)” para enviar todas as mudanças para o repositório. Ex: git push local master.

Escreva “git push +(nome do repositório)+(branch)” para enviar todas as mudanças para o repositório. Ex: git pull local master.

“git remote rename” para alterar o nome do repositório.

“git branch -M “main” para alterar o nome da branch (geralmente a branch vem com o nome master, e nesse exemplo está alterando para o nome main).

“git checkout -b “nome-nova-branch” para alterar a branch pra nova branch desejada, além de também criar essa nova branch (-b).

Para desfazer alterações:

* “git checkout --<arquivos>” para desfazer a alteração antes de adicioná-la (antes do git add)
* “git reset HEAD <arquivos>” para desfazer a alteração depois de adicioná-la mas antes de ser commitada (caso tenha realizado o git add).
* “git revert” para desfazer a alteração caso já tenha realizado o commit.

“git stash” para salvar arquivos temporariamente no Git.

* “git stash list” para ver a lista de arquivos salvos temporariamente.
* “git stash apply X” para aplicar a alteração número X registrada no git stash.
* “git stash drop” para remover a alteração da lista de arquivos salvos temporariamente.
* “git stash pop” para aplicar a última alteração e removê-la da lista simultaneamente.

Git log para ver os últimos commits e a “chave”

Git reset –hard <chave> para voltar o código ao commit desejado.

Como puxar atualizações do github/bitbucket para o projeto sem precisar clonar novamente:

* Acessar a Branch master, dar um git pull para puxar as atualizações para o projeto local.
* Mudar de Branch para a Branch desejada (para não ficar utilizando a master), em seguidaa dar um git pull origin master

**ANOTAÇÕES**

Unidades de medida: REM e EM

* EM é uma unidade de medida relativa ao font-size (especificamente na tag mãe)
* REM é relativa a tag de maior hierarquia no código (<html>)

SVG são imagens criadas através de instruções de um computador.

Conceito de Mobile First: Elaboração de um site pensando primeiro para ser utilizado em um dispositivo mobile.

Para fazer com que uma imagem utilize 100% da tela, é mais interessante utilizar a unidade de medida width: 100vw (viewport width) ao invés de utilizar width:100%, pois neste ultimo caso a imagem irá ficar restrita caso haja um padding porém com o VW a imagem irá utilizar 100% da largura da tela independente se há padding. OBS: Caso queira utilizar 100% do comprimento vertical da tela, utiliza-se a unidade de medida VH (viewport height).

Para arquitetar melhor a esturtura CSS, existe a metodologia BEM (Block Element Modifier) que faz com que haja um padrão para nomenclatura das classes. Ocorre da seguinte forma:

* Block. Ex: class=’cabecalho, menu, navegador, banner, conteudo’
* Block\_\_element. Ex: class=’menu\_\_link, menu\_\_lista, banner\_\_imagem, sobre\_\_titulo’
* Block\_\_element—modificador

Inserir as propriedades CSS em ordem alfabética é uma boa pratica. Em casos quando há muitas propriedades em determinado seletor, irá facilitar a localização das propriedades por exemplo.

Dicas sublime:

* Ctrl+enter – cria nova linha abaixo.
* Ctrl+shift+enter - cria nova linha acima.
* Ctrl+shift+; - para inserir um comentário.
* Ctrl+; - para comentar uma linha inteira.
* Ctrl+J – para unir o final de uma linha ao inicio de outra, eliminando o espaço entre elas.
* Ctrl+KB – para ocultar ou exibir a barra de pastas lateral.
* Ctrl+P – para exibir a barra de pesquisa de arquivos.
* Shift+alt+8 – para inserir uma nova aba de sublime abaixo, dividindo a tela em 2 abas.

Meu Kit’s Code (Font Awesome):

<script src="https://kit.fontawesome.com/c92604a6d9.js" crossorigin="anonymous"></script>

* OBS: Ter atenção se o ícone não é pago (pois não irá funcionar caso seja)

Rodando o NODEJS no VSCode

1. Abrir a pasta do arquivo (new folder) no VSCode.
2. Escrever no terminal “node nome-do-arquivo.js”

Para tirar a fonte (Fira Code), basta escrever "Consolas, 'Courier New', monospace” no editor.fontLigatures dentro do arquivo settings.json.

**SEO** (Search Engine Optimization ou organização para mecanismos de busca) são um conjunto de estratégias que, aplicadas em um site, podem melhorar o posicionamento orgânico nas paginas de buscadores como o google, garantindo mais acesso ao site.

**API (Application Programming Interface)** é um conjunto de padrões de programação para acesso a uma plataforma web. É criada quando uma empresa de software tem a intenção que outros programadores desenvolvam produtos associados ao seu serviço. Ex: Quando uma pessoa acessa uma página de um hotel, é possível visualizar dentro do próprio site o mapa do Google Maps. Esse procedimento é realizado por meio de uma API.

**Como fazer Deploy no sistema Agile – Tecomat:**

1. Acessar o diretório ssh no próprio terminal do projeto e inserir o comando: ssh -i "TecomatAgile.pem" ubuntu@ec2-18-191-17-255.us-east-2.compute.amazonaws.com
   1. Isso irá fazer a conexão com o servidor
2. Após a conexão ser feita com o servidor, entrar na pasta através do comando: cd sgqp-tecomat
3. Comando: git pull
   1. Irá solicitar um Password: YE3yQeD9Hk9zzwxU3sPF (senha git Danilo)
4. Comando: npm run build
   1. Irá buildar a aplicação
5. Comando: forever stop 0
6. Novamente comando: forever stop 0
7. Comando: npm run stayAlive
8. Pra finalizar: npm run cronJobs

**DÚVIDAS**

1. Porque usar Styled-components? Quais vantagens?
   * 1. Elimina bugs em relação a nomes de classes, impedindo a utilização de um mesmo nome para classes diferentes.
     2. Permite trabalhar com as Props (propriedades do React) no CSS. Ou seja, permite uma estilização dinâmica, integrando JS no CSS.
     3. Facilidade na manutenção.
     4. Trabalha somente com o escopo do componente. No caso, cada componente irá identificar somente as classes de seu escopo.
2. Diferença entre Classe, Objeto, Atributo e Método?

As classes são basicamente os moldes utilizados para serem criados os objetos, no qual contém os atributos (suas características) e métodos (suas funcionalidades/ações).

Um exemplo prático seria: Uma classe de cachorro, com atributos de cor, raça e altura, e métodos de “crescer” e “latir”. Através dessa Classe, pode ser criado um Objeto chamado Scooby, com a cor marrom, raça Dog Alemão e Altura 60cm, com métodos crescer = altura + 10cm e latir = au forte.

1. O que é SOLID?
   1. S (SRP = Princípio da Responsabilidade Única) – Uma Classe deve ter apenas um motivo para ser alterada.
   2. O (OCP = Open/Close Principle) – Entidades de software devem ser abertas para extensão e fechadas para modificação. Ao invés de inserir condicionais na classe (if/else ou switch/case), o ideal é estender a classe através de heranças/polimorfismos.
   3. L (LSP = Principio da Substituição de Liskov) – Classes derivadas devem poder ser substituídas por suas classes bases. Serve para identificar se classes derivadas realmente devem ser geradas de uma classe base, ou se ela mesma deve ser uma classe base.
   4. I (ISP = Principio da Segregação de Interface) – Classes herdadas de uma classe base não deve ser forçada a depender de métodos que não usam. Serve para identificar se o método inutilizado pela classe herdada realmente faz sentido em existir na classe base ou se deveria ser gerado outra classe base com este método.
   5. D (DIP = Principio da Inversão de Dependência) - Dependa de uma abstração e não de uma implementação.
2. O que é MVC?

MVC é um padrão de arquitetura de software, seprarando sua aplicação em 3 camadas. A camada de interação com o usuário (view), a camada de manipulação dos dados (model) e a camada de controle (controller).