**Iframes - Frames inline (“quadros em linha”):** São janelas com frames de sites que podem ser colocados dentro da nossa página web, no entanto, não são todos os sites que permitem sua exibição em iframes por questões de segurança. Por isso existem situação em que o iframe **não é indicado**, mas existem situações em que ele **é indicado.**

Um iframe é um “quadro em” seu site, inclusive por padrão ele vem com dimensões pequenas (300px 150px) e por isso tem rolagem vertical. Ele não é compatível com todos os navegadores, por isso, uma estratégia que podemos usar é colocar uma **tag <a> dentro do <iframe>,** assim se o iframe não for compatível, existe um link lá que o usuário pode usar para ir para o site.

**Display <iframe>:** O display padrão do iframe é inline, ou seja, entrará imediatamente após ou antes de um texto na mesma linha, porem ocupando as medidas que forem passadas. Para isolar um <iframe> e centralizá-lo na tela, por exemplo, temos que passar no css os parâmetros display: block e margin: auto;.

**Parâmetros iframe:** src =”endereço do site” e frameborder=”valor” que aceita 0 (sem borda) e 1 (com borda). Inclusive pode ser preferível deixar frameborder em 0 e fazer configurações de borda no css.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ISSO AQUI É MUITO IMPORTANTE: os exercícios estão** exercícios > ex024 > iframe03.html

**VANTAGENS x DESVANTAGENS <IFRAME>:**

* Quando busco relação de **sentido** entre um <p> (conteúdo anterior ou posterior) e o conteúdo que quero exibir e escrollar, devemos usar esse conteúdo dentro de uma <div> e passar overflow-x e/ou overflow-y
* Quando **NÃO** busco relação de **sentido** entre um <p> (conteúdo anterior ou posterior) e o conteúdo que quero exibir e escrollar, devemos usar <iframe> pois assim conseguimos estilizar separadamente os conteúdos.
* **Exibição de várias páginas dentro de um iframe**. É possível criar uma lista <ul> com vários links <a> de documentos de páginas que quando clicadas podem ser exibidas dentro de um <iframe>. Para dar certo é preciso passar um parâmetro **name** dentro do <iframe> e o valor que for lá dentro deve ser passado como valor do parâmetro **target** do link <a>.
* **Exibição automática quando página for carregada**: Podemos usar o parâmetro src=”colocar algum documento inicial” ESTÁTICO que só irá aparecer de novo se a página for regarregada. ou outro parâmetro srcdoc= esse parâmetro permite passar html e tags dentro dele. O recomendado é usar um outro documento mesmo, por exemplo: src=”arquivo-separado.html”
* **DESVANTAGEM**: O próprio Google falou que existem algumas dificuldades para que o google bot (robô algoritmo do google) entre nos <iframes> e consiga indexar os conteúdos exibidos lá dentro; Além disso existem problemas de acessibilidade com iframes; Outro problema é a confusão em navegadores que podem não entender o botão voltar (devo voltar a página ou conteúdo do iframe?). Outro problema é a segurança, se colcoar um site dentro do meu iframe e aquele site “rouba informações”, meu site começa a fazer isso para ele. Como resolver isso: Não usar muitos iframes, principalmente para conteúdos que são altamente indexáveis (títulos, conteúdos importantes e sensíveis, entre outros).

A indexação é o processo pelo qual os motores de busca, como o Google, analisam, categorizam e armazenam o conteúdo das páginas web em seus índices. Esses índices são bancos de dados complexos que organizam informações sobre milhões de páginas, facilitando a recuperação de conteúdo relevante quando um usuário realiza uma busca.

**Como funciona a indexação:** Quando os "crawlers" ou "bots" (robôs) de busca visitam uma página, eles leem seu conteúdo e seguem links para outras páginas. Durante essa varredura, o conteúdo e a estrutura da página são analisados, e informações como título, descrição, texto, imagens e links são registradas. Com base nos dados obtidos, os motores de busca decidem quais páginas incluir no índice e como categorizá-las.

**Boas práticas de SEO incluem:** relevância de conteúdo, palavras-chave, estrutura de URL, usabilidade das páginas e tags semânticas do HTML5 com meta descrições claras e sitemaps bem estruturados.

**TORNANDO IFRAMES SEGUROS – “NÃO ABRA A PORTA DA SUA CASA P/ QUALQUER PESSOA ENTRAR”:** Para tornar <iframe> seguro, nós precisamos bloquear todas as ações e ou capturas de dados que são possivelmente maliciosas dentro do site que queremos exibir no iframe

* **Sandbox=”sandbox”:** Impede que informações que poderiam ser enviadas em formulários sejam enviadas (tira a funcionalidade do botão submit).
* **Sandbos=”allow-same-origin allow-forms allow-scripts”:** Essas configurações permitem o conteúdo dentro do <iframe> carregar formulários, dados e scripts da mesma origem que a dele (que provavelmente é seguro). Esses comandos permitem bloquear **scripts maliciosos**
* **Referrerpolicy=”no-referrer”:** Torna o conteúdo do site carregado dentro do <iframe> um pouco mais limitado

**\*IMPORTANTE: USOS DE IFRAME (VÍDEOS, MAPAS, GOOGLE DOCUMENTS):** HTML5-CSS3 > exercícios > ex024 > iframes006

**Formulários:** São campos onde declaramos determinados dados como: nome, idade, endereço, senha, cpf, número de telefone, campos de pesquisa, entre outros.

**Anatomia <form>:** Em HTML todos os campos do formulários devem vir dentro da tag <form> </form>. Dentro dessa tag virão inputs que são tags simples e que possuem um type=”text” para indicar o tipo de dado que está sendo colocado dentro dele. **\*Importante:** Os formulários, geralmente possuem um text indicando para os usuários sobre o que se trata o dado e um campo aonde o usuário coloca o valor desse dado, no entanto precisamos colocar uma **ligação / significado** para indexar esse formulário semanticamente para os motores de busca usando tag **<label>**.

Usa-se isso para relacionar: **QUAL CAIXA ESTÁ RELACIONADA COM QUAL LABEL – FACILITA PARA MECANISMOS DE BUSCA.**

Além disso, no campo action=” “ da tag form, precisamos colocar o lugar para onde vão esses dados (quando forem preenchidos pelo usuário); Para enviar esses dados são usadas linguagens de programação como PHP e JS, que usam os atributos name=”sobrenome” e id=”sobrenome” (PHP = name) (JS = id). Usaremos métodos de envio para enviar dados de formulários.

<http://127.0.0.1:5500/exercicios/ex025/cadastro.php?nome=Matheus+Luiz+&Sobrenome=Massuda>

**Métodos de envio de formulários:** Para enviar dados com HTML existem 2 métodos. Para tornar o envio de dados seguros como **senhas, dados de cartão de crédito, pagamentos, endereço, cep, cpf e informações sensíveis** é preciso usar protocolos de segurança como http e linguagens de programação para fazer criptografia de dados. No HTML existem os method=”get ou post” nenhum dos dois métodos é 100% seguro, mas o get deixa estampado os dados na URL e o post precisa usar o DEV TOOLS.

1. **Method=”get”:** Para dados não sensíveis (nome, busca google, URLs compartilháveis) e para dados com tamanho de até 3.000 bites (3.000 letras).
2. **Method=”post”:** Para dados sensíveis, que não apareçam na URL, dados com mais de 3.000 bites (3.000 letras) e para envio de arquivos (.pdf .xsl .jpg .mov).

**Atributos de <form>:**

1. **Method:** Método de envio “get” ou “post”
2. **Action:** linguagem de programação que fara gestão “cadastro.php”
3. **Autocomplete:** “on” ou “off”

**Atributos de <input>:**

1. **Required** (requerido) – coloca essa campo (input) como obrigatório dentro do formulário.
2. **Min / Maxlength**=”5” – coloca um mínimo e/ou máximo de caracteres que devem existir no campo.
3. **Min / Max –** Atributos de valor mínimo e máximo usado para input:number.
4. **Size** – Quantas letras determinarão o tamanho da caixa do input.
5. **Placeholder** – Deixa uma dica do que o usuário deve colocar no campo.
6. **Value** – Coloca o valor dentro desse atributo como valor para startar o campo do formulário, e por padrão ele retira o placeholder.
7. **Autocopmplete**: Existe um valor para cada tipo de input “username” “new-password” “currente-password”
8. **Step**: Usado para inputs do type=”number” ele padroniza o salto de um valor para outro, por exemplo, média escolar step=”0.1”. por padrão o input:number só aceitaria números inteiros.

* **<input type=”text>:** atalho input:text 🡪 aceitam dados do tipo texto “string”
* **<input type=”password”>:** atalho input:password 🡪 aceitam dados do tipo string e censuram com \*\*\*\*
* **<input type=”tel”>:** input:tel 🡪 aceitam dados de telefone e tem compatibilidade com dispositivos móveis para puxar um número direto dos contatos.

**TAG <fieldset> e <legend>:** É possível agrupar dados de um cadastro tanto semanticamente (quanto posteriormente, visualmente via CSS), usando a tag <fieldset> e <legend>. Podemos separar o cadastro por tipos de dados semelhantes. Aqui vou pedir dados pessoais (Nome, sobrenome, idade, telefone, e-mail, idade, sexo), nesse outro fielset vou pedir dados financeiros (cartão de crédito, cvc, validade, bandeira do cartão). Nesse outro fieldset vou pedir dados de escolaridade: ensino médio, superior, pós, foramação, cursos de extensão.

**CHECKBOX & RADIO:** Checkbox são maneiras de colocar mais de uma resposta em um determinado formulário. Checkbox é usada para quando permitimos mais de uma resposta. Pode ser trabalhoso pois além de estar dentro da tag <form> também é preciso relacionar cada palavra com uma <lael for=”id-da-checkbox”> para quando clicar no texto também selecionar a caixa.

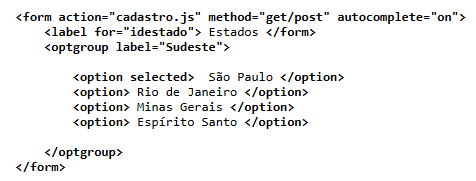
**CHECKBOX** – ASSINALAR CAIXA. Permite mais de uma alternativa assinalada

**RADIO –** ÚNICA OPÇÃO. Permite apenas uma única opção, mas **para isso é necessário no atributo name colocar uma categoria só para todos os rádios de mesmo grupo.** Por exemplo se pedirmos para assinalar Masculino ou Feminino. Devemos passar o atributo name com mesmo valor, por exemplo, sexo. Mas cada label deve se referia apenas ao seu id correto.

**Outros inputs – color / range / file:** Outros tipos de inputs são: color (permite ao usuário escolher uma cor se colcoar o código hexadecimal em atributo value=”#00ff00” ele virá com essa cor de padrão). Range (alcance) é um tipo de input que permite ao usuário selecionar uma área de alcance (o range padrão dele é de 0 até 100, mas podemos personalizar com atributos min=”1” e max=”5”, ai ele andará de um em um até 5).

<**Input:file>** São inputs que permitem ao usuário enviar arquivos como fotos, vídeos, pedfs entre outros .jpg / .png. Como provavelmente um arquivo possui um tamanho maior do que 3.000 bites, o método de envio deve sser mudado para method=”post”.

**OPTION > OPTGROUP > DATALIST:** São entradas (que fazem parte de um formulário) para o usuário colocar dados. Podemos criar um campo de <option> para um lugar com possibilidades finitas (estados do Brasil), por exemplo; Ainda por cima, podemos envolver essas opções em <optgroups> para criar um grupo dentro desse grande grupo de opções

 Perfeito quando temos alternativas finitas e com subgrupos. Ex: estados (sudeste e lista de estados do sudeste). Ex: Dps (lista de dps). Ex: Homens (de 10 a 20 anos). Ex: Camisetas (P, M, G, GG).

Já as **<DATALIST>** são usadas quando temos que escolher opções que podem ser infinitas, ou seja, criadas com uma frequência muito fácil ou que estão em constante atualização. No lugar de envolver as options com optgroups, envolveremos as options com uma datalist. Além disso é preciso colocar o campo aonde essas opções irão aparecer <input:text> e passar op atributo list=”id-da-datalist”.

**<outpu>:** Tag de forms usada para mostrar coisas na tela.

**TEXTAREA:** Cria um campo para o usuário digitar uma mensagem em texto, ela geralmente é infinita, mas a parte visual que aparece pode ser definida nos atributos da tag de acordo com colunas cols=”30” e linhas rows=”10”. Obs: quando se usa a tag <textarea> o ideal é colocar o método de envio form como method=”post” para não exc eder 3k bites.

**MEDIA QUERY (SINGULAR) E MEDIA QUERIES (PLURAL):** Medias Queries são maneiras **(em estilos)** diferentes de “imprimir” seu site em diferentes dispositivos ou maneiras, são feitos com CSS e servem para adaptar nosso conteúdo a vários tipos de mídia e de tela. Exemplos onde usamos Media Queries são: Site tradicional no navegador **no monitor** (pode conter vídeos de background, por possuir, geralmente mais banda); Site sendo exibido na tela de um **tablet** (pode limitar a quantidade de informações: textos em menus laterais, quantidade de conteúdo exibido tumnail de vídeos, menu hamburguer); Site sendo exibido na **tela de um** **celular** (limita-se ainda mais a quantidade de conteúdos, tamanho de fontes maiores, links em botões e textos, exibição de conteúdos mias leves por limitação de banda); Site sendo exibido para **impressão** (existem alguns conteúdos que não fazem sentido para ser exibidos quando quero imprimir uma página como: menus, rodapés e links que não poderão ser clicados, propagandas e vídeos que não poderão ser clicados, inclusive é recomendado que se coloque a fonte do conteúdo link escrito no topo e além disso o tipo de fonte recomendado para leitura em tela é diferente do tipo de fonte recomendado para leitura em papel).

A primeira coisa que devemos fazer para criar Medias Queries é criar arquivos CSS e linká-los dentro do index.html. A estrutura de um link é usada na parte do <head>, geralmente em baixo do <title> na parte de metadados e é:

<link *rel*="stylesheet" *href*="../styles/tela.css" *media*="screen">

O atributo media=”screen” pode aceitar screen (tela) // print (impressão) Podemos inclusive colocar vários links de css dentro do index. E apenas mudar o media=”o que quero”, dessa forma ele não vai sobrepor ao outro, mas irá funcionar apenas na media especificada

Esses atributos são chamados de **media types, MAS** SCREEN (TELA) PODE ACABAR SENDO GENÉRICO DEMAIS E FUNCIONANDO PARA QUALQUER TIPO DE TELA (monitor pc, celular, tablet, relógio, telões, cinema), POR ISSO EXISTEM AS MEDIA FEATURES (Dentro da tela dizer o tamanho e criar versões para diferentes tipos de tela).

**MEDIA FEATURE:** Media Query = media type + media feature.Características de mídia. Para indicar que estamos usando uma media feature, passamos dentro do atributo media da tag link media=”screen **and (orientation: portait)”.** As features são passadas depois de and ().

A grande sacada dos **MEDIA FEATURE** é que nós podemos passar **configurações gerais** que sempre (faça chuva ou faça sol) funcionarão de uma determinada maneira (por exemplo: background-color, font-famyli, color, font-weight). E podemos passar features que funcionarão em situações específicas. A grande complexidade em se usar essas configurações de feature é entender o que é uma **configuração específica** e o que é uma **configuração geral.**

**REUNINDO CSS:** Podemos passar todas as configurações de media e CSS em um único lugar, reunindo em configurações gerais, depois em baixo das gerais colocar configurações específicas

Nesse caso as declarações de @media screen, são passadas dentro do CSS e se fossem colocadas em um documento.css nem precisariam ser declaradas na tag <link>, bastando apenas linkar o estilo css.

Em projetos **grandes e imensos** pode ficar muito grande (verticalmente) o arquivo css e confuso para o desenvolvedor, por isso o indicado é separar arquivos de estilos de acordo com mídias.

**MOBILE FIRST:** É uma proposta desenvolvida por um ex-yahoo (Luke Wroblewski) que hoje é gerente do Google que propõe a criação / desenvolvimento dos sites **começando pelo mobile,** ou seja, no lugar de criar uma versão para pc e ir removendo funcionalidades e com isso o usuário perde experiências; Crie primeiro o Mobile (mobile first) e depois adapta as funcionalidade, menus entre outras coisas que pesariam para o mobile para a versão tradicional.

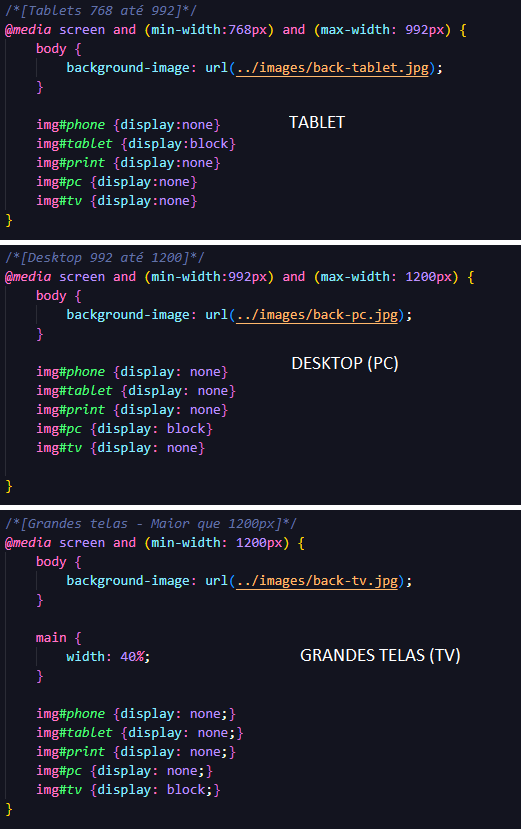
Segundo ele, o algoritmo (mecanismos de busca) do **Google vai valorizar** sites desenvolvidos com mobile first pois melhora a o carregamento e eficiência e automaticamente a experiência do usuário. A grande vantagem de desenvolver um site pelo método **“mobile first”** é a **performance.** Quando se desenvolve para mobile first você ganha uma valorização do Google.



**TYPICAL DEVICE BREACKPOINTS:** É basicamente a padronização do site para os diferentes tamanhos de tela. No entanto é muito complicado definir um padrão pois eles estão constantemente em mudança (existem diferentes tamanhos de telas com diferentes combinações de resolução).

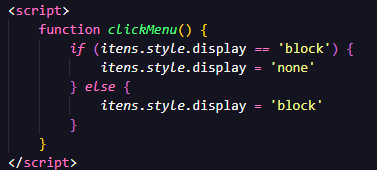
|  |  |
| --- | --- |
| **Dispositivo** | **Tamanho em Pixels** |
| Pequenas telas (relógio) | < 600px |
| Celulares | 600px até 768px |
| Tablets | 768px até 992px |
| Desktop (PC de mesa) | 992px até 1200px |
| Grandes Telas (TV) | > 1200PX |

No documento CSS, o ideal é que esses *Typical Device Breackpoints* sejam divididos entre os documentos para facilitar leitura e manutenção de código, mas em um mesmo documento ficaria algo parecido com:



**MENUS RESPONSIVOS (hamburguer):** A criação de menus deve ser pensada para ser otimizada em diferentes tamanhos de tela. Um menu que costuma funcionar bem para celulares é o **menu hamburguer** que é basicamente um botão que abre mais botões para baixo ou para o lado, geralmente representado por um ícone com três linhas uma em cima da outra.

Para criar o menu hamburguer, podemos importar ícones de menu no Google icons com link e <i>. Depois disso será necessário colocar o display a tag <menu> como “none” para que por padrão ele venha encolhido. Por último é necessário incluir o seguinte JS:

“itens” é um id=”itens” da tag <menu> que contém todos os elementos clicáveis do menu. Esse JS considera a seguinte sentença: Se o modo de visualização do menu for block, então “onclick” coloque como none, senão coloque o display como block.

Existem configurações de CSS que geralmente são usadas como padrão quando acabamos de criar um documento CSS (estilo) 🡪

* @charset “UTF-8”;
* \* { margin:0; paddin:0; }
* Html { font-size: 1em; }