**Módulo 1 -** [https://www.youtube.com/MODULO1](https://www.youtube.com/watch?v=Ejkb_YpuHWs&list=PLHz_AreHm4dkZ9-atkcmcBaMZdmLHft8n&pp=iAQB)

**Capitulo 1 – História da Internet**

Introdução ao curso e disponibilização de material para estudo, pelo link [https://github.com/Material\_Apoio](https://github.com/gustavoguanabara/html-css/tree/master/aulas-pdf)

Conteúdo contado o surgimento da internet, o famoso **WWW** (world wide web), os protocolos que utilizamos até hoje o **HTPP** para navegarmos entre os sites e também o **HTTPS** que seria da mesma forma, mas para o envio de dados de forma criptografada, tornando assim um protocolo de mais segurança para assim tornar possível transações para compra e venda de maneira mais segura.

**Capitulo 2 – Como Funciona a Internet?**

Vimos como que se conecta um computador a rede e como ele se comporta para fazer essa conexão. Como funciona as rotas até completar essa conexão de forma completa, o que são os domínios, e como que uma **URL** se comporta.

Digamos que cada máquina ao se conectar à internet utiliza o **MODEN** para fazer, que tem a responsabilidade de receber o sinal de internet da maneira que ele vier e transformar as informações em sinais de internet.

Já que falamos de MODEM, vamos entender melhor, são os famosos roteadores que são responsáveis por encontrar a melhor rota possível para realizar um acesso e envio de informações para um servidor.

Como já sabemos a internet utiliza de vários protocolos para realizar uma conexão, um deles é o TCP/IP. Supomos que toda vez que nos conectamos geramos um IP, ou seja, um endereço único que nos identifica, porem toda vez que desligamos o roteador e ligamos, esse IP muda, e como vamos fazer para decorar o endereço IP de um site de gostamos muito? Aí entra o DNS (Domam Name System), que salva esses endereços, como se fosse uma enorme lista telefônica.

**Domínio e Hospedagem**

Para conseguirmos acessar um site é preciso que saibamos sua **URL** (**Uniform** **Resource** **Locator**), e parte principal da **URL** é seu **domínio**.

O **domínio** é o nome único que vai conseguir identificar o seu servidor ou suas páginas, e para ter acesso a um domínio é preciso pagar por isso.

Exemplos de **domínios** são grandes sites que sempre mantiveram a mesma **URL** e que ninguém mais utiliza.

Ainda há aqueles domínios com varia terminações (.com, .gov.br, .io, .edu.br), essas terminações determinam se é site governamental, ONG, etc.

* **TLD** (Top Level Domain) – usado em alguns casos para identificar o país (.br)
* **GTLD** – são TLD genéricos que não identificam pais (.net, .gov, .org. .io, .info, .online).
* **ccTLD** – são TLD com designação de pais (country code) (.com.br, .edu.us, .jp, . es, .co.fr)

já que estamos falando de **domínio**, existe também o **subdomínio**, nada mais é que um atalho para uma determinada área do site:

* **www.google.com.br** - site principal do Google
* **images.google.com.b**r - Google Imagens
* **maps.google.com.br** - Acessa o Google Maps
* **mail.google.com.br** - Entra no Gmail

**NOTA**: muitos acabam que WWW faz parte do domingo, e não faz, é um subdomínio que faz conexão direta com o servidor padrão, ou página inicial no caso do Google, acima.

Além do **subdomínio** e domínio, uma **URL** também é composta pelo protocolo utilizado, que pode ser **http://** ou **https://**, dependendo se o seu servidor tem ou não segurança por **SSL** (um serviço de criptografia de dados).

**Capitulo 3 – Como funciona HTML e CSS**

**HTML** já se entrega pelo próprio nome, como **Hypertext Markup Language**, ou de forma traduzida **Linguagem de Marcação Hipertexto**, até tem quem seja muito iniciante que ache que é uma linguagem de programação, mas está erado, ela é uma linguagem para identificar o conteúdo da página, sem instruções muito complexas. **HTML** se base em marcações, ou mais conhecidas como **TAG**. Vimos complexidades em **JavaScript**, ela sim é uma linguagem de programação, que busca que p usuário tenha uma interação com a página.

Agora no caso do **CSS**, ele vai focar em **seletores**, **propriedades** ou **valores**. Sendo assim possível estilizar baseado em um seletor, alterando sua propriedade trocando por novos valores.

Para ficar mais fácil de entender:

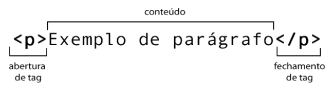
**HTML = conteúdo**

**CSS = estilo**

**JS(JavaScript) = interatividade**

**TAG’s HTML**

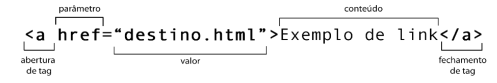
Como eu já disse anteriormente, a **HTML** funciona baseada em marcações específicas chamadas **tag’s**. Uma **tag** é um conjunto de palavras entre sinais de colchete angular, conforme representado a seguir:



\*(imagem 1)

Na imagem anterior, você consegue perceber o uso da **tag**

para a criação de um parágrafo simples. A maioria das **tag’s** possuem uma abertura e um fechamento, e você identifica isso pela presença da barra no fechamento da **tag**. Além disso, as **tag’s** também podem ter atributos e valores, que vão configurar seu comportamento:



\*(imagem 2)

Uma mesma **tag** pode ter vários parâmetros, mas algumas **tag’s** não tem a necessidade de ter um conteúdo, ou um fechamento, que seria o caso de **<br>** e **<img>**.



\*(imagem 3)

Mas tome cuidado para não usar **tag’s** obsoletas, até existem sites por aí que usam, mas com o surgimento do **HTML5**, em sua versão mais recente não podemos mais cometer o erro de usa-las.

**Chegou a vez dos seletores CSS**

Como já vimos anteriormente **CSS** é **Cascading Style Sheets** (**Folhas de Estilo em Cascata**), são usadas para configurar o resultado visual de elementos **HTML**.

Essas configurações são feitas por meio de **seletores**:



\*(imagem 4)

Na imagem o seletor vai alterar as propriedades do parágrafo em questão, se limitando a tudo que estiver dentro de chaves { }, no seletor mostrado será alterado os valores de:

* Fonte escolhida foi Arial **(font-family)**
* O tamanho da fonte é de 12pt**(font-size(pontos))**
* A cor da fonte será azul **(color)**

**Estrutura básica de um documento HTML**

Ao criar um novo documento **HTML**, existe um certo padrão para isso, e tendo isso em mente vamos analisar todas as 11 linhas desse documento base.



\*(imagem 5)

* **Linha 1**: Indica a versão do documento (no caso, HTML5).
* **Linhas 2 e 11**: Delimitam o documento HTML, que é sempre dividido em duas partes: a cabeça e o corpo(head/body). **Na linha 2**, indicamos o idioma, será no idioma Português do Brasil.
* **Linhas 3 e 7**: Local onde são realizadas algumas configurações iniciais como formatos, estilos, ícone de favoritos, etc.
* **Linha 4**: Adiciona ao documento atual o suporte a caracteres acentuados.
* **Linha 5**: Indica que o conteúdo aparecerá, por padrão, ocupando todo o espaço disponível da tela e com uma escala de 1:1.
* **Linha 6**: Configura o título da página, que aparecerá como identificação da aba do navegador, ao lado do favicon.
* **Linhas 8 e 10**: Delimitam o corpo da página, a maior porção do site, que vai aparecer na tela. É aqui onde colocaremos todo o nosso conteúdo.

**Front-end e Back-end**

**Front-end** é o profissional que trabalha com as tecnologias que são interativas para o cliente, ou seja, **HTML/CSS/JS** é claro que é também um profissional que precisa ter noções básicas em outras áreas, como Design para entender a disposição das formas das páginas, cores, tipografias, etc.

**Back-end** é o profissional que está mais interessado em fazer o código dele conversar com o servidor e com tecnologias e ferramentas que conectam com o servidor e não com a interação com o usuário. Algumas dessas tecnologia ou linguagens são:

* **PHP**, que até hoje se faz presente no mercado apesar de ser uma linguagem antiga e está na grande parcela dos sites da internet.
* **JS/JavaScript**, mas em sua versão para Back end, como NODEJS, mas não confunda com JavaScript para o Front end, porque são tecnologias totalmente diferentes e para fins distintos.
* **C#**
* **Python**
* **RUBY**
* **JAVA**, e não confunda com JavaScript, essa sim é completamente diferente uma da outra.

Outra categoria de desenvolvedor, seria o **FULL STACK**, é o profissional que cuida tanto do **FRONT END** quanto o **BACK** **END**, mas para se tornar esse tipo de profissional requer muita pratica, experiência e conhecimentos nas mais diversas linguagens e tecnologias.

Agora vamos imaginar que um cliente acesse um pagina de compras. Logo quando ele acessa a página já retorna as informações base, como **HTML**, **CSS** e **JS**, porém como que o site vai saber quais produtos me mostrar? Quais promoções vão ter? Quantas cores, tamanhos e quantidades daquele produto terão em estoque? Claro que o cliente não vai ficar inserindo isso manualmente sempre, então aí que entra uma linguagem de **BACK** **END** como o **PHP**, que quando o cliente abre a página, faz uma solicitação ao servidor e retorna uma página **HTML** criada no momento do acesso, que já contém **CSS** (estilo) e **JS** (interatividade), o mesmo acontece ao seciona um produto, assim bom base nos dados que constam no servidor, o PHP consulta e retorna de forma interativa para o usuário.

**Capitulo 4 – Primeiros passos com HTML5**

Para iniciar a criar os primeiros códigos é preciso antes escolher um editor que realmente se adeque aquilo que você deseja, ou que mesmo não sendo o melhor e usado pela maioria das pessoas, que seja fácil de usar para você.

Nessa situação eu escolhi de fato um que eu já tenho total adaptação que é o **VISUAL STUDIO CODE**, da Microsoft e distribuído gratuitamente na internet, e é bem fácil sua instalação. Mas outras opções interessantes são:

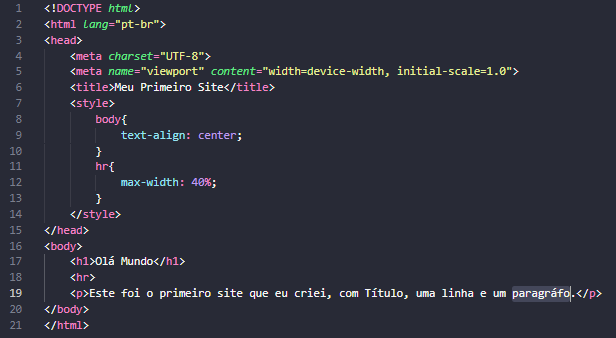
* **SUBLIME TEXT**
* **ATOM**
* **BRACKETS**
* **NOTEPAD++**
* **WEVSTORM**



\*(imagem 1)

Agora siga o passo a passo da configuração do editor escolhido no material de apoio. E logo em seguida vamos entender o exercício proposto.

Bom vamos criar uma pasta aonde vão conter todos os arquivos desse conteúdo até o final do curso, após isso vamos chamar uma pasta de EXERCICIOS, sendo assim pra o primeiro uma nova pasta EX001, certo, arrastando a pasta pai para dentro do VSCODE (VISUAL STUDIO CODE), podemos ver que temos acesso a todas as pastas/arquivos. E vamos criar um novo arquivo, será o index.html, logo em seguida vamos digitar um comando básico que cria toda a estrutura base da nossa página HTML, seria ! e após digitar pressione ENTER, e fala como eu fiz que vou explicando o que acontece na nossa página.

\*(imagem 2)

**LINHA 2**: alteramos a propriedade para o navegador entender que é uma página em português, como já vimos anteriormente.

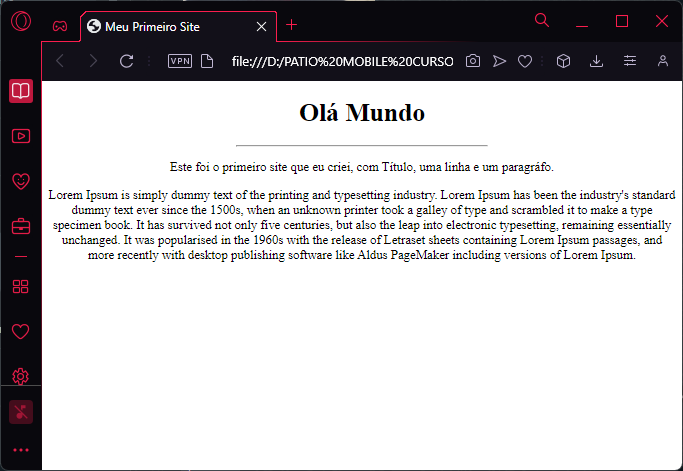
**LINHA 17**: Foi adicionado um título, que vale lembrar só de digitar **h1** e após apertar **ENTER** a **tag** de título abre e fecha automaticamente.

**LINHA 18**: foi inserido uma linha na horizontal, que é uma tag de autofechamento, já explicado que seria esperado existir **tag’s** assim.

**LINHA 19**: Foi adicionado um parágrafo com a **tag** **p**,por conta do tamanho do texto não foi necessário utilizar o **word** **wrap**, com o atalho **ALT+Z**, que faz quebra de linhas automática no texto, facilitando na leitura do código. E o que faz o **word** **wrap**: Suponhamos que a **tag** **p** comece na linha 1 e seu texto empurre até a linha 10, ou que fosse uma linha na vertical que nunca acabasse, ele faria uma quebra de linha que independente de quantas linhas fossem quebradas, a próxima **tag** estaria na linha 2.

**LINHA 7 a 14**: Foi inserida uma estilização por meio de **CSS**, em uma das maneiras de fazer isso, não explicarei o que faz, mas que tal você colocar isso no código e depois tirar, para ver o que acontece?

Bom agora está na hora de abrir nosso arquivo para ver como ficou, vamos na pasta que salvamos nosso arquivo, a pasta EX001, e executar o arquivo no navegador da sua preferência, porem recomendo que seja no GOOGLE CHROME, que é o navegador que tem maior compatibilidade. Feito isso, o navegador deve carregar como na imagem:

****

\*(imagem 3)

**1º** Notamos que a parte superior, a aba da página ganhou um título, graças a **tag <title>**

**2º** Notei que a tag **h1** fez com que o primeiro conteúdo se tornasse um título no nosso conteúdo.

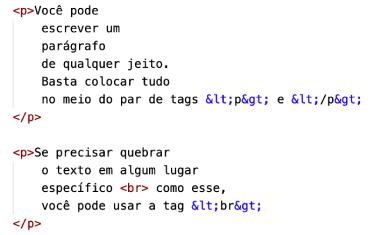
**3º** Já a **tag** **hr** fez aparecer uma linha horizontal como já prevíamos.

**4º** Ótimo, a **tag** **p**, realmente fez um parágrafo, mas vamos conferir se as quebras de linhas estão iguais para o código e o que vimos em tela?

**EXTRA**: se aumentamos e diminuirmos o tamanho da janela do navegador, notamos de forma muito básica uma certa responsividade, que será melhor trabalhada no decorrer do curso.

**Capitulo 5 – Caracteres, Parágrafos e quebras de Linhas**

Vamos começar pelos **Parágrafos**, utilizando a tag p. vamos fazer seguindo o passo a passo do material de apoio, e fazer conforme o exemplo:



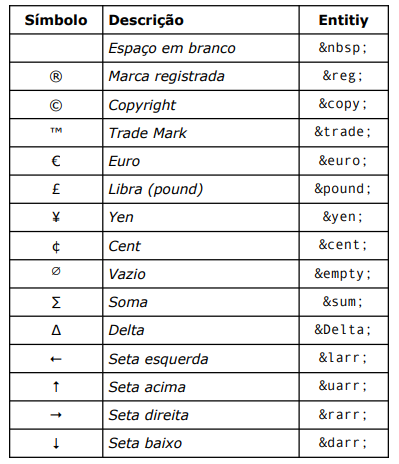
\*(imagem 1)

Vamos notar quando abrirmos o código no navegador que algumas coisas acontecendo. No primeiro parágrafo, e tanto no segundo não adianta de nada quebramos a linha do nosso código, porque ela não vai ser quebrada no navegador, já que o HTML5 não entende a tecla ENTER como uma quebra, mas sim a tag <br> que pode ser acionada no meio de um texto, ou depois de uma tag.

**IMPORTANTE:** iniciantes costumam colocar várias vezes a tag <br> para adicionar de forma maior um espaçamento no texto, mas aviso que **NÃO FAÇA ISSO**, vamos entender mais para frente que isso é trabalhado no **CSS**. Então não importa o tamanho desejado, espere um pouco até chegarmos no **CSS** para continuarmos a editar o texto.

Vamos notar também a presença de alguns caracteres especiais que eu vou deixar você os procurar no navegador e tentar adivinhar o que eles se tornaram.

Pois bem eles se tornam representações de tag de maneira visível no navegador, já que o navegador e o editor vão entender que uma tag, seja qual ela for vai se comportar sempre como uma tag e nunca como parte do texto, porém com esses tipos de caracteres especiais isso se torna possível. Pesquisando na internet podemos ter acessos a todo tipo de caractere especial de retorna em formato de texto um tag diferente, e na imagem abaixo, representa alguma delas:



\*(imagem 2)

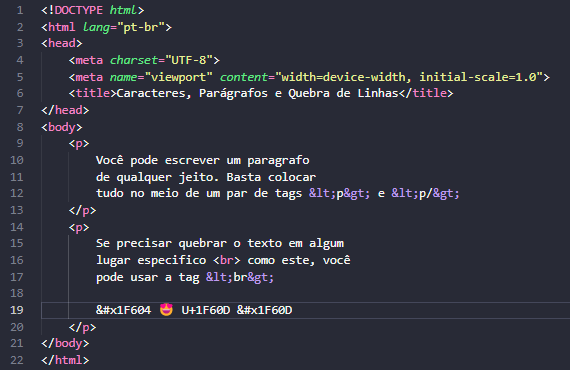
Esses caracteres são chamados de **HTML Entities**, que para achar esse basta acessar uma sessão especialmente dedicada para eles no site da W3Schols: [HTML Unicode UTF-8 (w3schools.com)](https://www.w3schools.com/charsets/ref_utf_symbols.asp)

**Extra: Inserindo Emojis**

Para realizar a adição de emojis, é bem simples pode ser feita no texto da tag p mesmo, mas é um código especial que precisa de atenção para não errar.

Primeiro vamos acessar o site <http://emojipedia.org/>, em seguida vamos selecionar um emoji qualquer do site e quando abrir uma nova guia vamos procurar por **codepoint** que do emoji selecionado é o **codepoint: U+1F60D**.

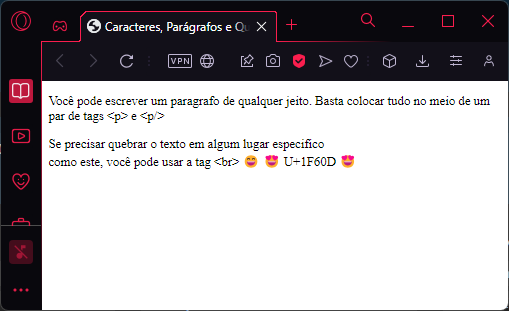
Agora para fazer com que o emoji apareça na tela SEMPRE vamos substituir o **U+** por **&#x** e assim o emoji irá aparecer no site. Mas nos dias de hoje somente copiar o emoji e copiá-lo no código já o faz aparecer, mas se por algum motivo ele não aparecer, já sabemos como fazer da maneira correta.



\*(imagem 3)

**LINHA 19**: Forma corretar de transformar um emoji por código, um emoji colado diretamente no código e que aparece no navegador, um código de emoji copiado e não substituído, e por fim o mesmo código, porém substituído.

E o resultado final seria:

 \*(imagem 4)

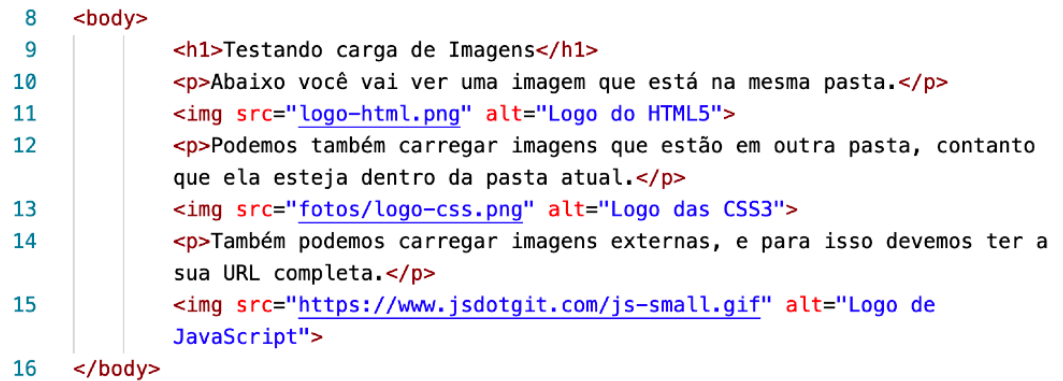
**E que comece os DESAFIOS!**

Na pasta da matéria de apoio foi colocado uma pasta de **DESAFIOS**  e com tudo mostrado até agora, podemos desenvolver uma solução para o desafio proposto, que seria o caminho **DESAFIOS > modulo-01 > d001**. Assim vamos também criar uma pasta de desafios no nosso **VSCODE** para sempre que puder acessa-lo.

**Capitulo 6 – Imagens e Favicons**

Como está tudo descrito no material de apoio, iremos criar uma pasta para exercitar chamada **EX001** e nela vamos colocar uma das imagens dispostas no material, vamos criar outra pasta **IMAGENS** e colocar a outra imagem nessa pasta. Vamos fazer isso para entendermos como colocar imagens no nosso código mesmo que estejam em pastas diferentes.

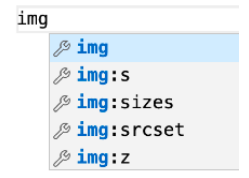
Agora vamos criar o código e vamos entender o que está acontecendo.



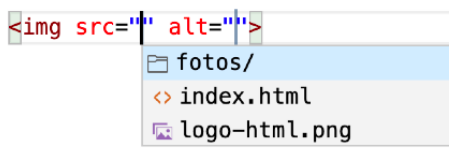
**LINHA 1 a 7**: é o código padrão que aprendemos até aqui, nessa aula vamos focar nas linhas depois disso.

**LINHA 11**: aqui criamos nossa **tag** **<img>** que é responsável por carregar imagens no nosso código, estando elas em pastas externas ao nosso projeto ou em links externos(**URL**). Ela vem acompanhada de dois parâmetros **SRC**(source, ou origem) e **ALT**(alternative, ou texto alternativo).

**DICA IMPORTANTE**, antes de criar a tag img da forma mais comum **<img>**, vamos digitar ela sem os sinas **<** **>,** e vamos perceber que o **VSCODE** vai nos dar opções para carregamos essa **tag**, e após somente precisamos apertar **ENTER**. Mas vale ressaltar que mesmo utilizando esse atalho precisamos entender o caminho, a navegação entre pastas e arquivos sempre vai ter uma barra entre eles / , para identificar que aquele arquivo está em uma determina pasta, como um caminho.



Bom após isso, vamos ver que a **tag** **img** carregou junto dos parâmetros, parecido com o que vimos no exemplo acima, mas como vamos fazer para endereçar as imagens? Simples dentro das aspas em **src**, vamos apertar **CRTL+ESPAÇO**, que vai carregar opções do que tem dentro das pastas do nosso arquivo. Quando a imagem for endereçada por link externo bastas selecionar a imagem com a opção “**Copiar endereço da imagem**”, assim a imagem vai carregar, a menos que esse link mude com o tempo.



**LINHA 11 e 13**: como explicado como e para que serve os parâmetros, dito isso, vão carregar as duas imagens salvas em pastas diferentes.

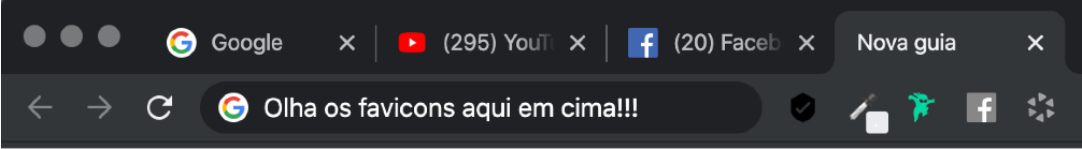
**LINHA 15**: pesquisando a imagem na internet e copiando o endereço, e desde que esse endereço não mude com o tempo, a imagem irá carregar da mesma forma que as anteriores.

**Para que servem TEXTOS ALTERNATIVOS?**

Bom suponha que quem esteja visitando nossa página tenha alguma dificuldade visual, geralmente usam aplicativos que transcrevem o que está em tela para auxiliar esses usuários, e como esses aplicativos leriam imagens? Para isso que servem textos alternativos, ou o parâmetro **alt** da **tag** **img**.

Dentro dos “ ”, vamos colocar “foto logo”, quando esse aplicativo chegar nessa imagem vai ler o texto alternativo. Mas também tem outro uso, no caso de a imagem não carregar, seja qual for o motivo, ela irá carregar um texto alternativo indicando ao usuário o que deveria estar naquele local.

**Usando Ícone de Favoritos (favicon)**

Bom você já deve ter visto um desses nos sites que costuma navega, mas não sabia do que se travava. Um favicon se encontra ao lado do **titile**, você vai entender melhor abaixo. 

Aqui você entende melhor, essas imagens ao lado do nome da guia, como em Goole, YouTube e Facebook é chamada de **favicon**, e a ultima aba não tem nada porque não carregou nenhum para, é apenas uma guia em branco.

Para usar um favicon é bastante simples, hoje há sites especializados em disponilbilizar esses arquivos normamente em formatos **ICO**. Se quiser baixar alguns icons pronto te dou duas opções:

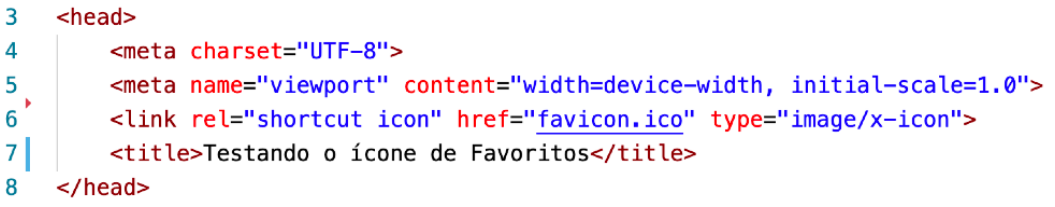
* [Icon Archive - 800,000+ free icons & stickers, PNG, SVG Downloads](https://www.iconarchive.com/)
* [Material Symbols and Icons - Google Fonts](https://fonts.google.com/icons)

Se você criar um **icon** com as próprias mão, pode entrar no site [favicon.ico Generator](https://www.favicon.cc/).

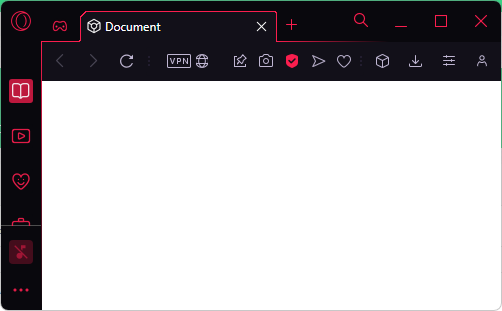
Agora se você quiser criar um **icon** com base em alguma imagem, podemos ir no site [The best Favicon Generator (completely free) - favicon.io](https://favicon.io/), que vai gerar um arquivo ZIP.

Depois de fazer algum desses passo, vamos salvar o arquivo **.ICO** na pasta do nosso projeto, e normalmente colocamos o nome do arquivo como **favicon.ico**.

Agora que você já entendeu o processo, está na hora de praticar criar uma nova pasta para exercitar, **EX004**, e da forma que criamos até aqui, vamos criar o código padrão, e na linha acima da **tag** **<title>** vamos adicionar a tag **<link>**, mas como fizemos com **img**. Digitamos link e esperamos as opções que o **VSCODE** nos dá, e vamos selecionar **link:favicon**, e se seguir esses passos, seu código deve ficar mais ou menos assim:



E só olhar no código acima, se você usou a diga que eu dei vai estar da mesma forma, e ainda por cima seu código foi criado todo “automaticamente”.



E esse foi o resultado seguindo os passos, até o salvamento.

**Quais são os formatos de FAVICONS?**

Apesar da melhor escolha ser em formato ICO, existe também a possibilidade de usar em formatos PNG e também o SVG que seria a melhor opção, já que é um vetor.

Mas mesmo SVG sendo a melhor opção nem todos os navegadores tem suportes para esse formato, então aconteceria um erro de carregamento ao acessar o site.

**VAMOS AOS DESAFIOS!**

Com todo o material absorvido até agora, já é possível realizar os desafios **d001** até **d003**. Os desafios estão em PDF em forma de resultado, mas não demonstra nenhum linha de código, temos então que resolver de maneira que visualmente fique igual ao mostrado em PDF.

**Capitulo 7 – Hierarquia de Títulos**

Para entenderemos como que funciona hierarquia de títulos, vamos lera qualquer página de notícias e vamos perceber que no topo sempre há um título com uma fonte grande e bem chamativa, e logo sem seguida vem um texto em uma fonte menor e bem menos chamativa, mas mesmo assim maior do que a fonte do corpo da notícia, isso vamos chamar de subtítulo. Mas vamos notar que em meio ao corpo da notícia há muitos outros subtítulos., isso para separar as ideias dentro do copo do texto.

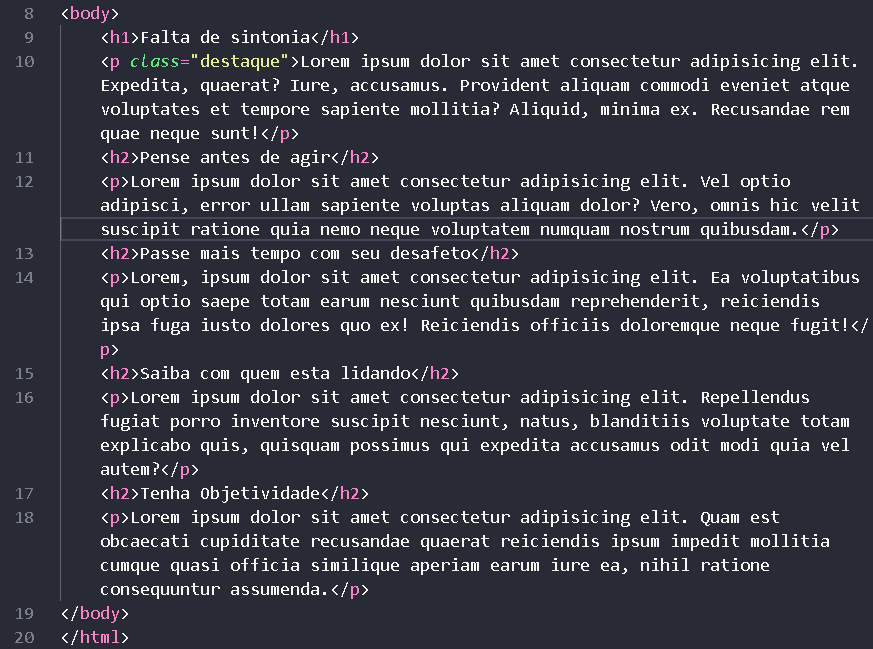
**Como criar títulos em HTML?**

Títulos em **HTML** são em **tag’s <h1>** que possuem seis níveis de hierarquia, sendo **<h1><h2><h3><h4><h5><h6>**. Muita gente acha que que **<h1>** seria um texto grande e **<h6>** seria um texto pequeno. Mas está totalmente errado.

Para entender melhor a hierarquia seria dessa forma, **<h1>** é o título e **<h2>** é subtítulo de **<h1>**, **<h3>** é subtítulo de **<h2>** e assim adiante até **<h6>**.

Vamos agora replicar o texto que do artigo que está no nosso matéria de apoio, mas não vamos ter o trabalho de digitar todos os parágrafos vamos utilizar **LOREM** **IPSUM**, que seriam textos genéricos criados automaticamente.

E como fazemos para criar? Bom digitamos a palavra **lorem** e apertamos **ENTER** logo em seguida, e o texto é gerado automaticamente, quase que da mesma forma que funciona com as **tag’s**.

Você percebe que a imagem começa na **LINHA 8**, isso porque tudo antes disso é gerado automaticamente, e vamos focar somente no conteúdo em **body**.

**LINHA 9**: Nosso título principal h1

**LINHAS 11,13, 15, 17**: são os subtítulos do título principal , como analisamos anteriormente.

**LINHA 10**: vimo uma novidade que foi a adição de uma **classe** para esse parágrafo, que serve para criarmos uma configuração visual diferente usando o **CSS**.

**Capitulo 8 – Formatação de Textos**

**Você já ouviu falar de semântica?**

Oque seria semântica? Semântica é o significado dos vocabulários, dito isso é possível analisar que **forma** e **significado**, e que **semântica** dá mais valor ao significado.

Uma das mudanças que mais diferenciam o **HTML4** para o **HTML5**, seria que buscam dar mais significado a **tag’s** que deem atenção maior para o seu significado do que para aquilo que seja mais visual.

Exemplo disso, em **HTML4** usamos a **tag** **<u>** para sublinhar uma determinada parte do texto/parágrafo, ou seja, “Rua blá blá”, se usamos a **tag** **<u>** ficara visualmente sublinhada e nada mais. Em **HTML5** temos a oportunidade de usarmos a tag **<address>**, que dá um significado aquilo que está no texto, ainda mais se estivermos navegando pelo celular é possível que de a opção de traçar uma rota até o endereço do site, tudo graças a uma tag semântica.

Tudo isso em **HTML5** se chama significado, e vimos de forma bem evidente agora **HTML** **semântico** ou **tag’s** **semânticas** ou ainda o **conteúdo** **semântico**.

**Negrito** ou *Itálico*

Vamos começar a ver uma das formatações mais usadas até os dias de hoje, que existem até **tag’s** para usar que seriam **<b>** e **<i>**, mas que não trazem significado algum para o que está sendo escrito.

Para isso, a partir de hoje vamos utilizar as **tag’s semânticas** **<strong>** e **<em>**.

A **tag <strong>** além de dar significado de força dentro da frase, ela também aparecera em **negrito**. E o mesmo acontece com a **tag <em>** que queremos **ênfase** (do inglês *emphasis*), e sendo assim dentro da tag tudo aparecerá em ***itálico***.

Vimos que ao usamos **tag’s** **semânticas** ao invés de **tag’s** convencionais, damos maior significado ao nosso código a visualmente o texto pode ser alterado, mas para maiores configurações visuais vamos mudar no **CSS**.

**Você sabe usar uma caneta MARCA TEXTO?**

Provavelmente você já utilizou uma caneta marca texto, e sabemos que quando uma parte do texto se diz importante nós queremos deixar ele marcado para ficar bem chamativo e achar de forma mais fácil.

Em **HTML5** temos uma tag que além de mudar visualmente traz um significado maior ao texto, a **tag <mark>** em meio a parte do texto que queremos demarcar.

**Mas podemos mudar a cor?** O padrão dessa tag seria amarelo, mas para alterar esse padrão vamos fazer isso em **CSS**.

**<BIG> morreu, mas <small> sobrevive**

Você já deve ter visto por aí, algum texto em tamanho menor ao restante, para isso vamos usar **<small>**, até existia uma **tag** que deixava maior, mas foi descontinuada durante as atualizações do **HTML5**.

**~~Texto deletado~~ e texto inserido**

Basicamente antes tínhamos a **tag** **<strike>** para deixar a aparecia de um texto deletado, mas hoje usamos **<del>** que traz um peso maior, significado. Aparência de texto que dá para ser lido, porém não deve ter importância.

É o oposto de texto deletado, a **tag <ins>** da aparecia de texto inserido (sublinhado), que traz um destaque, importância maior para o texto.

**Textos sobrescrito e subscrito**

Há pela internet, ainda mais em sites sobre ciência ou matemática artigos ou palavras e números escritas como 22 ou até H 2O, de maneira respectiva temos as **tag’s <sup>** e **<sub>**.

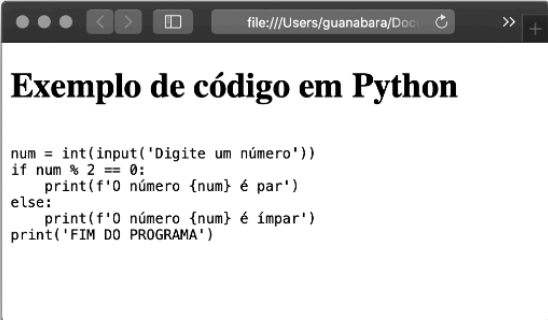
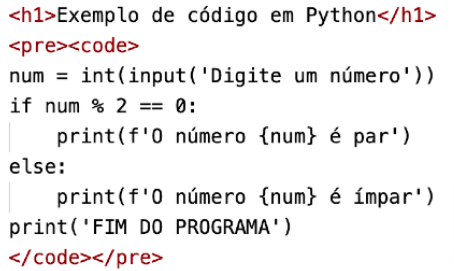
**Trechos de códigos**

Nessa vida de programador, vemos a necessidade de compartilhar nossos códigos na internet, porém para visualizarmos de maneira como código temos a **tag <code>**, uma **tag** que dá um formato totalmente diferente a essa parte do texto, se vermos o texto dentro dessa **tag**, notamos as letras monoespaçada.

Mas ainda assim o texto não fica se parecendo com um código, temos assim a adição para essa **tag**, outra **tag <pre>** que mantém o texto pre formatado da maneira que ele foi escrito.

**Tag’s unidas**

Vimos nesse exemplo se usarmos as **tag’s <pre><code>** ... **</pre></code>** dá para perceber que assim o texto fica muito parecido com um código, e qualquer outra formatação visual que achamos relevante, irá ser feita no **CSS**.



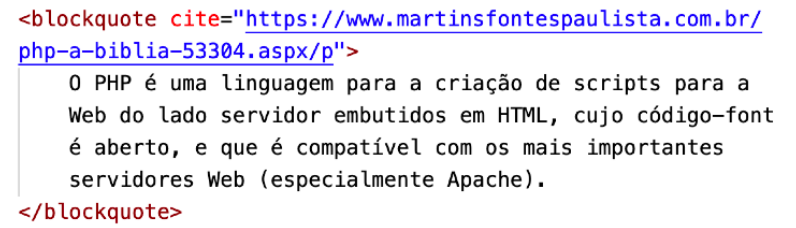
Outro exemplo seria um texto escrito em negrito e em itálico ao mesmo tempo, seria **<strong><em>**, ***ficando dessa forma***.

**Citações**

Enquanto lemos um texto qualquer existem em vários casos as citações, que são palavras para ditas por outras pessoas em meio a um texto que não seria dessa mesma pessoa.

Dito isso em um trecho na internet em meio ao seu código **HTML** podemos tem uma **tag <q>** (do inglês quote, que significa citar), que em meio a um texto faz muito sentido você usar essa **tag**.

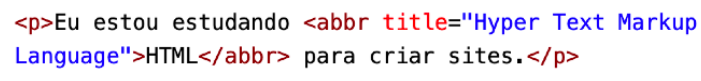
Porém existem texto aonde as citações são feitas em parágrafos próprios, e para isso temos uma **tag** especifica **<blockquote>**, e assim o texto tem a forma de uma citação e também gera um recuo especifico. Podemos também colocar um link para a citação original, mas visualmente não muda em nada, e não gera um link para um acesso externo, quem vai se aproveitar disso são os mecanismos de busca, para saberem de onde veio aquela citação.

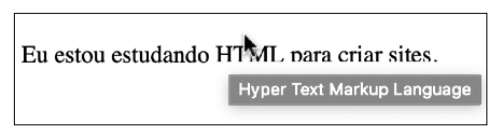


**Abreviações**

Uma das novidades apresentadas pelo **HTML5** é a possibilidade de criar abreviações, usando a **tag <abbr>**. Assim sempre que você quiser criar uma sigla e deixar claro para os mecanismos de busca o que aquela sigla significar, ou até mesmo para o usuário.

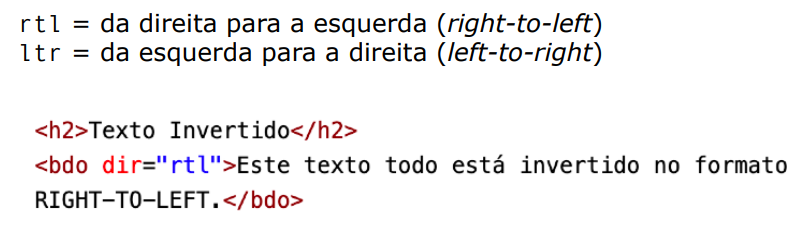
Para isso dentro da **tag <abbr>** além da **tag** vamos utilizar o parâmetro title, para assim que o usuário passar o mouse em cima, tenha alguma ação visual para que ele identifique o que aquela sigla significa.





**Texto Invertido**

Aparentemente uma das **tag’s** mais inúteis vistas até aqui. Mas para usa-la que é presentada pela **tag <bdo>**, que ao usar recomendo adicionar o parâmetro **dir**, para indicar uma direção, de duas possíveis.



Bom vimos aqui uma seleção de **tag’s** mais usadas, e não todas. Sendo assim as importantes e mais usadas teriam que estar aqui, mas trouxe uma praticamente inútil, só para te dizer que existem **tag’s** assim no meio do **HTML5**.

**HORA DE EXERCITAR**

Para essa aula vamos praticar e tentar replicar os exercícios em nosso material de apoio, sendo os respectivos exercícios **EX007** e **EX008**.

**CAPITULO 9 – Listas com HTML5**

O poder das LISTAS

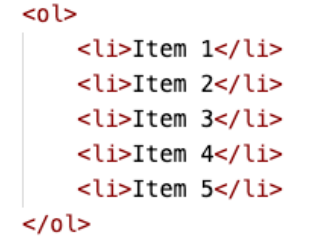
Geralmente tudo que fazemos hoje em dia, criamos uma lista para organizar ao menos as tarefas. E temos listas para tudo, para o que você imaginar, mesmo que sem querer criamos listas.

E por conta disso o HTML disponibiliza vários tipos de listas para produzirmos nosso conteúdo de forma clara e eficiente.

**Listas ORDENADAS**

A **HTML** chama de **ordened lists** todas aquelas listas aonde a ordem dos itens é importante.

Para criamos uma **ordened** **list** vamos usar a **tag <ol>** para delimitar a lista e a tag **<li>** de **list** **item** para identificar cada item.



As **tag’s** **<ol>** possuem **parâmetro type**, aonde conseguimos mudar o tipo do marcado:

* **1** é valor padrão, criar listas numeradas
* **A** cria listas alfabéticas maiúsculas
* **a** cria listas alfabéticas minúsculas
* **I** cria lista em algarismos romanos maiúsculos
* **i** cria lista em algarismos romanos minúsculos

você também pode indicar início com o parâmetro star, ficando assim:

**<ol type=”A” start=”5”>** que vai começar a partir da quinta letra do alfabeto.

**Listas NÃO ORDENADAS**

Bom se você entendeu o conceito anterior, com certeza vai entender o que são **unordered** **lists**, ou lista de marcadores. São listas que não necessariamente precisam de uma ordem ou classificação, somente precisam conter os itens.

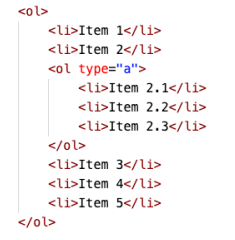
Para criar uma lista não ordenada vamos usar a **tag <ul>** para delimitar a lista, e novamente a **tag** **<li>** para seus itens.

E assim como uma lista ordenada, uma lista não ordenada também possui **parâmetro** **type** que podemos alterar seus marcadores:

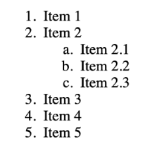
* **disc**, uma bolinha por padrão
* **circle**, uma bolinha sem preenchimento
* **square**, um quadrado totalmente preenchido

**Misturando as coisas!**

Já que vimos os tipos diferentes de listas, agora podemos dizer que podemos misturar os dois tipos para s tornar uma só, e ainda mais podemos alterar os seus parâmetros de maneira separa. Ficando assim:

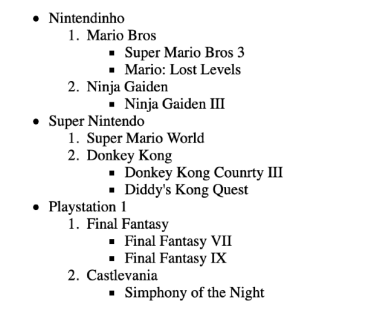


Após o código o resultado visual seria esse:



**Exercício proposto**

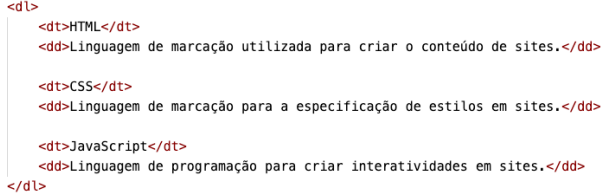
Crie seu código, para que possa conter uma lista de jogos favoritos como no exemplo:



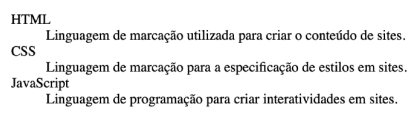
**Lista de DEFINIÇÕES**

Uma lista sem marcadores, mas muito útil em alguns casos. Pensa como um dicionário, temos termos e suas definições.

Toda lista de definições está dentro de uma **tag <dl> (definition list)**. Cada termo é **<dt> (definition term)** e cada descrição é <**dd> (definition description)**.



O resultado do código, seria esse.



E cada navegador pode exibir essa lista de maneira ligeiramente diferente, mas o que mais importa é o significado da listagem já que relaciona **TERMO** **+** **DESCRIÇÃO** diretamente com os mecanismos de busca.

**Hora de exercitar**

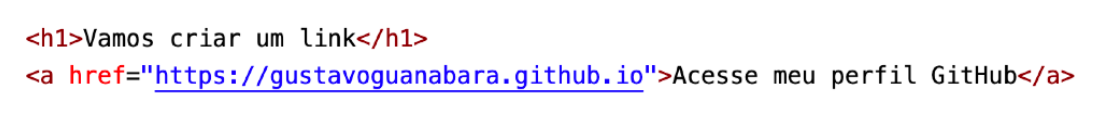
No nosso material de apoio vamos tentar replicar o conteúdo do **exercício 09**, e sendo assim vamos criar uma pasta para resolver chamada **EX009**.

**CAPITULO 10 – Ligações em toda parte**

Os **hyperlinks** são o conceito mais antigo da internet, e permite que você ligue um ponto ao outro na **Word** **Wide** **Web**. Todas as vezes que acessamos um site, seja qualquer que seja a fonte, ou até mesmo estejamos baixando algum arquivo, estamos interagindo com um **hyperlink**.

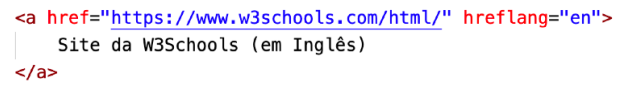
Até os mecanismos de busca se utilizam dos **hyperlinks**. O **Google** por exemplo está sempre varrendo a internet atrás de links com conteúdos novos e relevantes. Sendo assim é de extrema importância que conseguir links validos de outros site para o nosso próprio site.

Para criar hyperlinks, devemos criar ancoras com a **tag** **<a>.** que vai levar sempre seu principal parâmetro para que possamos criar uma referência de **hypertext** que é o **href**.



Note que dentro de href colocamos um link, com sua **URL** completa para outro site.

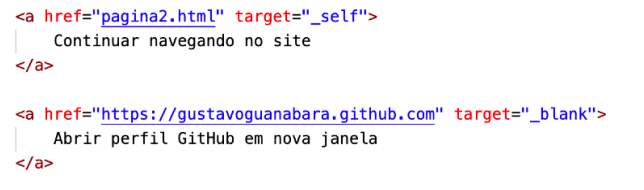
Outro atributo importante é o **hreflang**, que vai informar que a página que estamos indo é em determinado idioma. Sendo assim ajudando o usuário em uma possível tradução automática, sem necessidade de se verificar o idioma.



Agora vamos pensar, ao colocar um link externo em nosso site, nós queremos que o usuário acesse esse **link**, mas ainda tenha acesso ao nosso site. E vamos fazer isso usando uma outra parâmetro, chamada **target**.

O não uso desse parâmetro pode fazer com que a nova pagina se sobreponha a do nosso site, e assim induzindo o usuário a só continuar e não retornar mais.

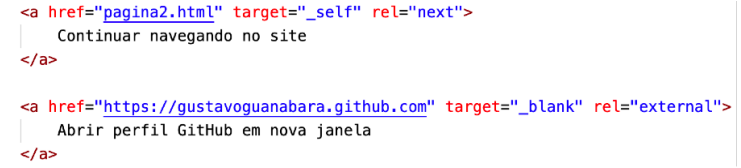
* **blank**, vai abrir o link em uma nova guia, mantendo nossa página ainda aberta
* **self**, é padrão. Faz com que o link acessado sobreponha a nossa pagina
* **top**, vai abrir o destino no navegador completo
* parente, similar ao top
* **nome**-**do**-**frame,** caso esteja usando frames



**Esse link é seu ou dos outros?**

Existem alguns recursos que podemos usar que indica qual a natureza do destino usando o atributo **rel**. Dentre os valores aceitos, estão:

* **next**, indica que o link é para a próxima página do documento atual
* **prev**, indica que o link é de uma parte anterior a atual
* **author**, indique que o link para o site autor do artigo
* **external**, indica que o link vai para um site que não faz parte do site atual
* **nofollow**, indica que é um link pago

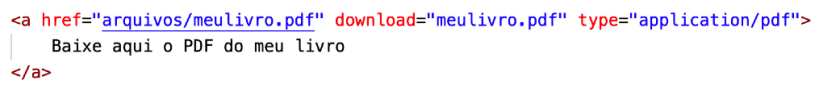


O primeiro link chamamos de **link loca** ou **link interno**, permite que o usuário continue navegando nos domínios do site, e note também que não precisa necessariamente de uma **URL** nesses casos, somente o caminho do arquivo **HTML** já resolve.

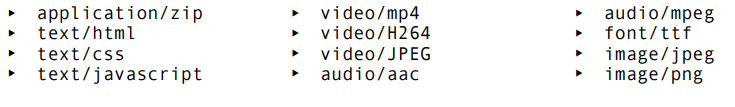
Já o segundo é um link para um site externo, chamado de **link** **externo**. Nesses caso tem necessidade de se colocar a **URL** completa, junto do seu protocolo **HTTP://** ou **HTTPS://**.

**E para fazer Downloads?**

Outra coisa bastante comum é sites conterem links para baixar arquivos em PDF, ou arquivos ZIP. Para criamos isso no nosso site, basta criar um **link** direto para nosso arquivo, adicionar a propriedade **download** com o valor configurado para dar nome ao arquivo baixado e o atributo **type** para indicar ao navegador que tipo de arquivo está sendo baixado.



Outros tipos de medias types usados no dia a dia, são:



Nesse site conseguimos ver muito mais opções de **media** **types**, ou o que colocar no atributo **type** acesse o link:

<https://www.iana.org/assignments/media-types/media-types.xhtml>

**Hora de exercitar**

Junto ao nosso material de apoio realize o **exercício** **10**, com fim de reproduzir tudo aquilo que está em tela sem cópias os códigos.

**Hora do desafio**

Com base em tudo que aprendemos até aqui, em nosso material de apoio, tente resolver o **desafio** **05** e o **desafio 06**. Esses desafios não possuem código para consulta, tente chegar o mais próximo do proposto com uma solução própria.

**Capitulo 11 – Imagens dinâmicas, Áudio e Vídeo**

**Seu site deve se adaptar ao tamanho da tela**

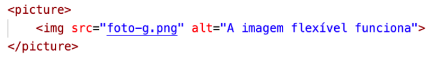
Já sabemos que o tamanho do arquivo de imagem pesa para que os mecanismos de busca achem a sua página. Dito isso temos que fazer com que nosso site conte com imagens de diferentes tamanhos para se adequar a todo tipo de dispositivo e ainda assim permanecer leve.

Bom mesmo que pagássemos uma imagem gigantesca teria sim como diminuir ela de modo forçado por **CSS**, mas na prática a imagem continua com o mesmo peso (bytes), para isso seria interessante criar imagens de tamanhos diferentes para ambos os tamanhos que se demonstrem necessários.

**IMAGENS FLEXIVEIS**

Nosso primeiro passo vai ser adaptar no código 3 imagens que vão responder a tamanhos diferentes do nosso navegador.

Para isso vamos utilizar a tag **<img>** que já conhecemos, e agora vamos utilizar ela dentro da tag **<picture>** e vamos também aprender a **tag <source>.**



Bom a nossa primeira imagem começa a ter problemas quando a janela do navegador começa a se aproximar dos 1000px, já que esse é o tamanho dela em seu tamanho total.



Dito isso vamos utilizar a tag <source> com os atributos **media**, **srcset** e **type** que basicamente para:

* **media**: indica o tamanho máximo a ser considerado para que a imagem indicada em **srcset**
* **srcset**: indica a imagem a ser carregada, porém digitando o caminho manualmente.
* **type**: indica o ***media*** ***types*** do arquivo (***media*** ***types*** foi vista no capítulo anterior).

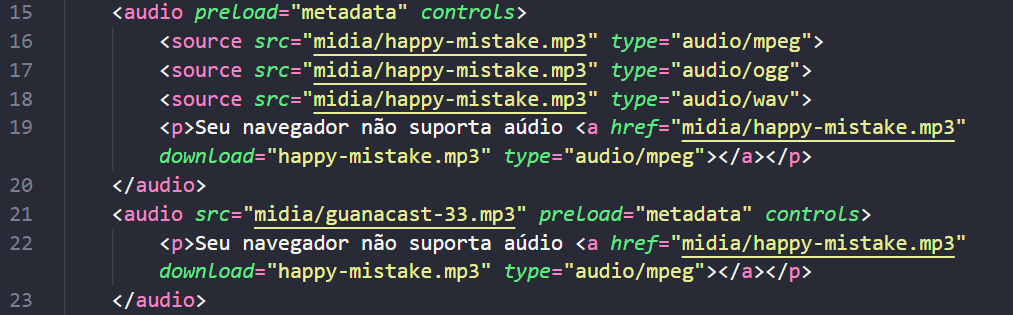


Aqui vamos entender que a **tag** **source** conta com uma certa hierarquia, onde o menor valor a ser considerado deve estar em primeiro, e seguindo uma ordem crescente. Se indicarmos um valor menor na **tag** abaixo, a imagem não vai ser carregada. E a última tag antes da **tag** **<picture>** fechar, tem que ser a nossa imagem, **tag <img>**.

**AUDIOS no HTML**

Com **HTML5**, veio a facilidade para compartilhar áudios nos nosso sites. E a partir de agora temos a **tag <áudio>** que automaticamente carrega um atributo src para localizar nosso arquivo de áudio, porem necessidade de outros atributos para se fazer visível no navegador.

Mas para se fazer necessário outras opções de áudio caso a principal não carregue, vamos usar **<source>** com a ajuda do atributo **type**.



Antes vamos entender os atributos para entender o que acontece dentro das tag’s.

* **preload**: indica se o áudio vai ser pré-carregado ou não, e aceira três valores
  + **metadadata**: vai carregar algumas informações sobre o arquivo (tamanho, tempo, etc.)
  + **none**: não vai carregar nada, até que o usuário aperte o botão para iniciar o áudio
  + **auto**(padrão): vai carregar o arquivo de áudio inteiro assim que a página carregue, mesmo que o usuário não clique em nada
* **controls**: atributo que não recebe valor, e serve para que o player carregue na tela.
* **autoplay**: vai iniciar a reprodução automática do áudio assim que a pagina carregar, sem necessidade da interação do usuário.
* **loop**: assim que o áudio terminar, ele se repete, sem interação do usuário.

Dentro da **tag <áudio>** vamos criar outros **<source>** para o mesmo arquivo de áudio, com vários formatos **type** diferentes, para garantir que se um formato não carregar, carregue em outro formato. E em último caso criamos um parágrafo que permita o download do áudio para ouvir em outros players de áudio, ou mesmo para quem quiser baixar o áudio somente clicando com o botão direito do mouse.

Agora para entendermos o que acontece no código:

**LINHA 15**: Abertura da **tag** **<áudio>** com o atributo **preload** recebendo o valor **metadata** que vai carregar assim as informações do áudio, como tempo de reprodução.

**LINHA 15 a 18**: Abertura **tag <source>** para localizar aonde o arquivo na sua máquina com o atributo **src** recebendo o valor da localização do arquivo, e o atributo **type** que recebe qual o tipo de arquivo que está sendo carregado. Importante notar que colocamos 3 opções do mesmo arquivo para garantir que ele vai carregar na página independente do formato.

**LINHA 19**: Criado um parágrafo, para se caso nenhuma das opções para carregar o áudio funcione, há a possibilidade de baixar o áudio para escutar em um player próprio. E caso queira baixar o arquivo, clicando com o botão direito do mouse tem essa possibilidade.

**LINHA 21**: A **tag** **<áudio>** automaticamente cria um atributo src que possibilita carregar o áudio e todos os seu atributos em uma única linha. Seria outra maneira de colocar um áudio na página.

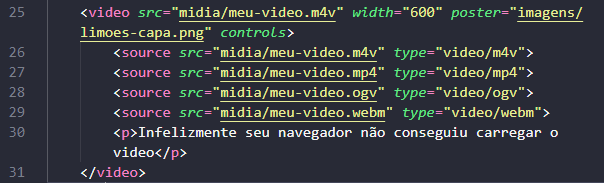
**E os VIDEOS?**

Funciona basicamente na mesma estrutura que áudio, mas vamos entender as suas tag’s, atributos e valores.

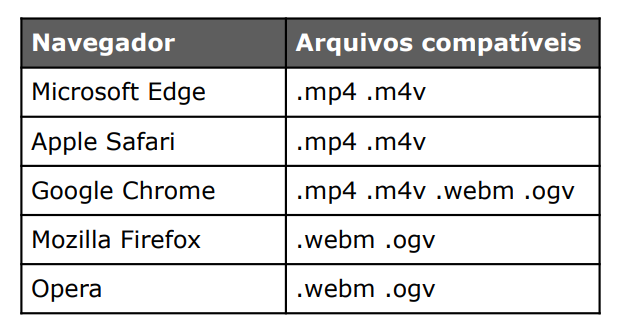
Para começarmos vamos abrir a tag <vídeo>, que recebe os atributos:

* **wdth**: que indica o tamanho que o videio exibido terá na tela
* **poster**: podemos indicar uma capa para o videio antes que ele carregue.
* **controls**: faz com que os controles do player carreguem junto do vídeo.
* **autoplay**: quando a página carregar, sem a interação do usuário o vídeo irá carregar.

Como vimos em áudio, temos as mesmas formas de colocar um videio na nossa página.



Temos que prestar atenção nos formatos que vamos subir nossos vídeos na nossa página, navegadores diferentes não podem aceitar os mesmo formatos.



Como fazemos para resolver esses problemas de subir vídeos por meio dessas tag’s e ter certeza que eles vão funcionar em quase 100% dos casos.

Bom para isso temos serviços como YouTube e Vimeo, mas cada um com suas vantagens e desvantagens.

YouTube: algoritmo rápido, servidores rápidos, fazendo assim com que nossos vídeos nunca travem, o único problema e ter a obrigação de deixar nossos vídeos públicos no YouTube, o que atrapalha a vida de escolas online aonde queremos que somente pessoas cadastradas no nosso site possam ver esse vídeo

Vimeo: para suprir essa necessidade de o vídeo só ser visível para quem tem acesso a nosso plataforma, surgiu o Vimeo as únicas desvantagem são algoritmos pouco eficientes que ocasionam travamentos nos vídeos e assinatura paga anualmente para utilizar o serviço.

**Da para incorporar vídeos dessas plataformas no HTML?**

A resposta é que dá sim para fazer isso, porem vamos frisar no YouTube, vamos em compartilhar e em seguida em incorporar.



Feito isso copie o código que aparece junto da **tag <iframe>** e basta colar no nosso editor de código e adicione ao seu **HTML**.

**HORA DE EXERCITAR**

Com os exemplos dados até agora e todo material fornecido, tente realizar os **exercícios 011** e **exercícios 012**.

**DESAFIOS**

Com base em tudo visto até agora e a até mesmo nos exercícios feitos, realize os **desafios** **007**, **008** e **009**.

**Capitulo 12 – Introdução as CSS**

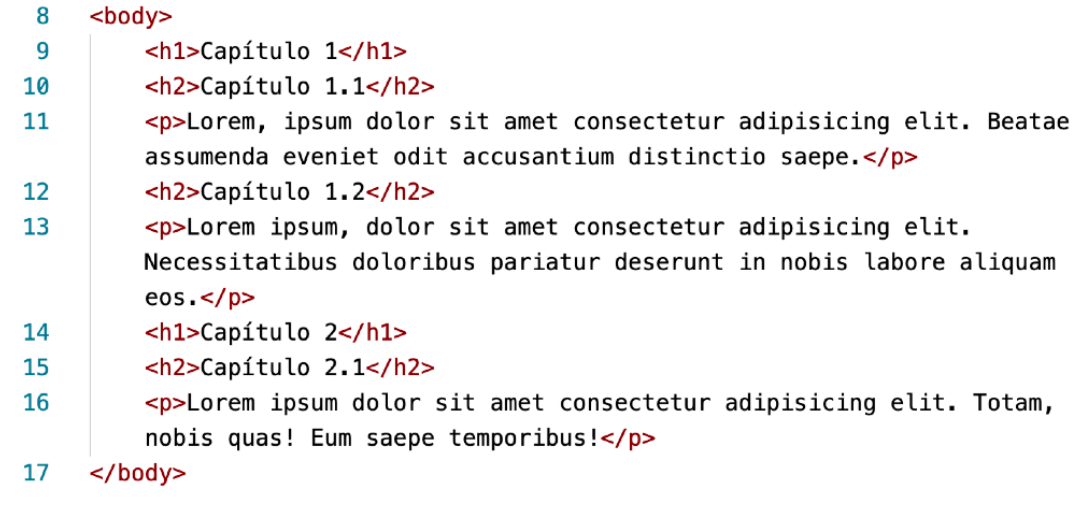
**Você se lembra do que falamos sobre CSS?**

Vimos anteriormente que o responsável pela semântica e até mesmo pela estrutura base da nossa página são as tag’s presente em HTML5, mas tudo e qualquer efeito visual é responsabilidade das CSS.

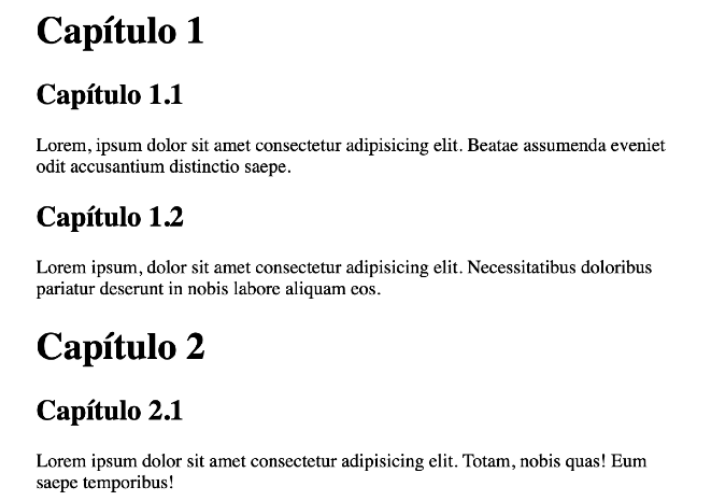
**A forma mais simples de aplicar CSS: CSS inline Style**

É a técnica mais básica que tem para aplicar um estilo, porém é a que mais bagunça nosso código, justamente porque usamos dentro de parâmetros de HTML5.

Para entender melhor, vamos criar algumas linhas de código e também criar alguns parágrafos usando Loren.



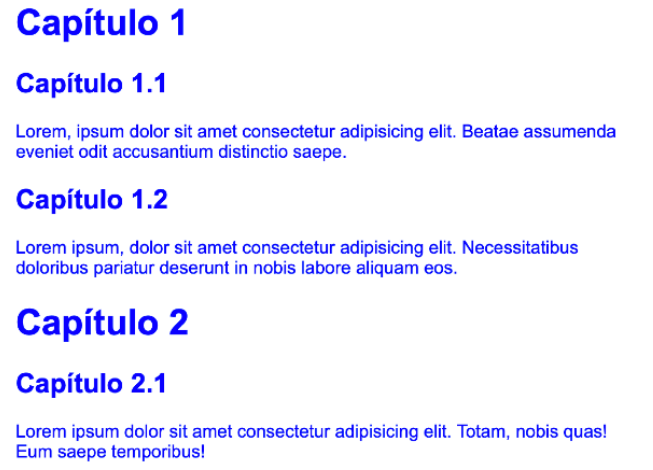
Esse código aberto no navegador, vai aparecer da seguinte forma:



Vamos nos focar na **tag <body>**, e lá que vamos colocar nosso **CSS** **inline** de maneira bem simples. Da mesma forma como parâmetro, vamos digitar **style** e logo em seguida duas declarações: **font-family** e **color**.



Muito cuidado na hora de replicar esse código, tem que ser declarado da mesma maneira que descrita aqui, com as letras maiúsculas e minúsculas. Não esqueça do ponto e vírgula no final de cada declaração. O VSCODE até faz isso de forma automática, mas sempre é bom conferir. E se deu tudo certo, o resultado deve ser esse:

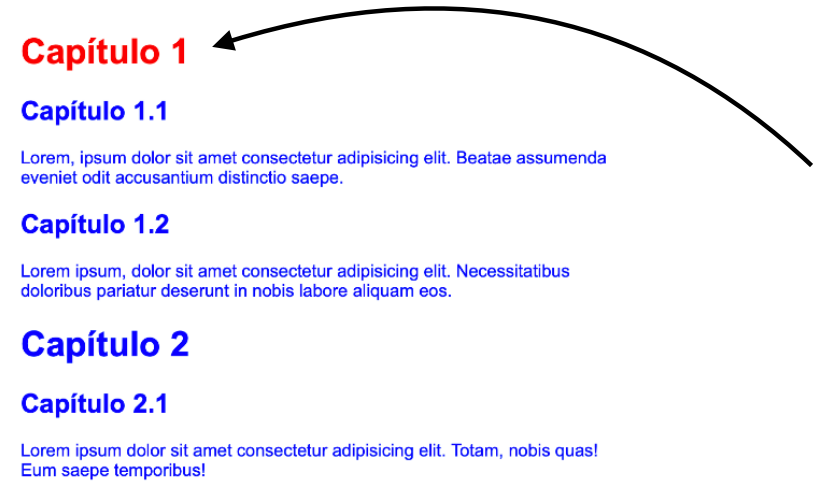


Note aqui que a fonte da letra mudou (padrão era Times) e a cor da fonte também. Mas se alguma das alterações feitas não atingiu o mesmo resultado, importante conferir o código e verificar o que está errado nele.

Agora vamos fazer mais uma alteração, mas somente no **h1**.



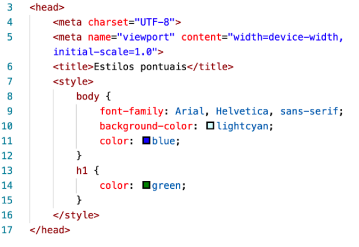
E o resultado será:



Isso acontece porque o **CSS** **inline** consegue selecionar tudo por meio de **<body>** sou somente uma **tag**, como fizemos aqui, uma **configuração pontual**.

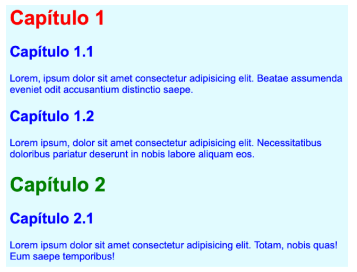
**Estilizando de maneira mais interessante: CSS internal style**

Para aplicar estilos de maneira mais dinâmica e pratica vamos criar a **CSS** agora separado do nosso código que fica no **<body>**, vamos criar uma **tag <style>** dentro de **<head>**.



**ATENÇÃO**: Sua **tag <style>** tem que estar dentro de **<head>**, mas se acontecer de colocar ela fora de **<head>** pode até ser que funcione, mas vai estar fora dos padrão estabelecidos pela W3C.

Resultado das alterações:



Mas porque mesmo dando as instruções **CSS** de estilo as **tag’s h1** não foram modificadas e estão com cores diferentes? Simples, o que o **CSS** inline se sobrepõe a todo os outros tipos de **CSS**, já que ele é uma configuração pontual o navegador entende que ela está acima das outras. E para desfazer é só apagar o **CSS** **inline** da **tag** em específica.

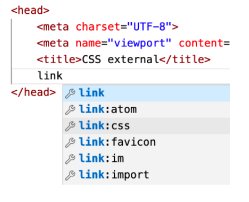
**A técnica vai versátil: CSS external style**

Deixar seu **CSS** fora do **HTML** deixa o código mais organizado e torna mais versátil estilizar, e faz com que nosso **CSS** seja reaproveitado de maneira mais eficientes por outras páginas **HTML** do nosso projeto. E para fazer esse link, vamos usar a **tag <link>** especialmente configurada para trabalhar com arquivos **CSS**.



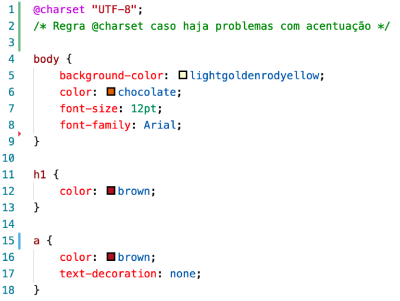
Uma dica para deixar nosso código mais eficiente de ser escrito, vamos apenas digitar link e esperar as opções que o **VSCODE** me dá, e vou escolher a única que é **link:css**, assim meu link especial para **CSS** é criado de forma automática.

Vale notar também que um caminho style.css é carregado de forma automática, se apertarmos **CRTL** e clicarmos nesse caminho, o próprio **VSCODE** pergunta se queremos que ele crie esse arquivo de forma automática.





Agora é só adicionar os seletores e todas as suas declarações no arquivo criado para que elas possam ser aplicadas ao documento.



**LINHA** **1**: declaramos o valor **@charset** **“UTF-8”,** como uma regra igual no código **HTML**, para não termos problemas com caracteres especiais.

**LINHA** **2**: adicionamos um comentário, que não vai ser levado em consideração. No **CSS** um comentário é caracterizado pelos sinais **/\*** e **\*/**, e tudo dentro desses sinais o navegador vai entender como comentário e não vai levar em consideração.

Bom de forma resumida:

**CSS** **externo** = usar sempre que puder

**CSS** **interno** = use para pequenas configurações

**CSS** **inline** = procure evitar

**HORA DE EXERCITAR**

Em nosso material de apoio procure realizar os **exercícios 013, 014, e 015**. Com base em tudo que aprendemos até aqui, devemos ser capazes de resolver. Mas lembre-se, nada de copiar o código.