Anotações da Aula

Neste módulo 8, aprendemos sobre...

- A Internet
- Desenvolvimento WEB
- HTML
- CSS
- JavaScript

Quero compartilhar meu aprendizado e/ou minha dúvida...

Ir para o Discord

A Internet

Hoje daremos uma olhada na programação web, utilizando um conjunto de novas linguagens e tecnologias para criar aplicações gráficas e visuais para a internet.

A **Internet** é a rede de redes de computadores que se comunicam entre si, fornecendo a infraestrutura para enviar zeros e uns. No topo dessa base, podemos construir aplicativos que enviam e recebem dados.

Roteadores são computadores especializados, com CPUs e memória, cujo objetivo é retransmitir dados através de cabos ou tecnologias wireless, entre outros dispositivos na internet.

Os **protocolos** são um conjunto de convenções padrão, como um handshake físico, com o qual o mundo concordou para que os computadores se comuniquem. Por exemplo, existem certos padrões de zeros e uns, ou mensagens, que um computador deve usar para informar a um roteador para onde deseja enviar os dados.

TCP/IP são dois protocolos para enviar dados entre dois computadores. No mundo real, podemos escrever um endereço em um envelope para enviar uma carta a alguém, junto com nosso próprio endereço para receber uma carta. A versão digital de um envelope, ou uma mensagem com endereços de e para, é chamada de **pacote (packet)**.

IP significa protocolo de internet, um protocolo suportado por softwares de computadores modernos, que inclui uma forma padrão para os computadores se endereçarem. Os **endereços IP** são **endereços** exclusivos de computadores conectados à Internet, de forma que um pacote enviado de um computador a outro será repassado aos roteadores até chegar ao seu destino.

- Os roteadores têm, em sua memória, uma tabela que mapeia os endereços IP para os cabos, cada um conectado a outros roteadores, para que saibam para onde encaminhar os pacotes. Acontece que existem

protocolos para os roteadores se comunicarem e descobrirem esses caminhos também.

DNS , sistema de nome de domínio, é outra tecnologia que traduz nomes de domínio como cs50.harvard.edu em endereços IP. O DNS é geralmente fornecido como um serviço pelo provedor de serviços de Internet mais próximo, ou ISP.

Por fim, o **TCP** , protocolo de controle de transmissão, é um protocolo final que permite a um único servidor, no mesmo endereço IP, fornecer vários serviços por meio do uso de um **número de porta** , um pequeno número inteiro adicionado ao endereço IP. Por exemplo, HTTP, HTTPS, e-mail e até mesmo Zoom têm seus próprios números de porta para esses programas usarem para se comunicar na rede.

O TCP também fornece um mecanismo para reenviar pacotes se um pacote for perdido e não recebido. Acontece que, na internet, existem vários caminhos para um pacote a ser enviado, uma vez que muitos roteadores estão interconectados. Portanto, um navegador da web, fazendo uma solicitação para um gato, pode ver seu pacote enviado por um caminho de roteadores e o servidor de resposta pode ver seus pacotes de resposta enviados por outro.

- Uma grande quantidade de dados, como uma imagem, será dividida em pedaços menores para que os pacotes tenham todos tamanhos semelhantes. Dessa forma, os roteadores ao longo da Internet podem enviar os pacotes de todos de forma mais justa e fácil. Neutralidade da rede (net neutrality) refere-se à ideia de que esses roteadores públicos tratam os pacotes igualmente, em oposição a permitir que pacotes de certas empresas ou de certos tipos sejam priorizados.
- Quando houver vários pacotes para uma única resposta, o TCP também especificará que cada um deles seja rotulado, como “1 de 2” ou “2 de 2”, para que possam ser combinados ou reenviados conforme necessário.

Com todas essas tecnologias e protocolos, somos capazes de enviar dados de um computador para outro e abstrair a Internet para construir aplicativos no topo.

Desenvolvimento WEB

A web é um aplicativo executado sobre a Internet, o que nos permite obter páginas da web. Outros aplicativos, como o Zoom, fornecem videoconferência, e o e-mail também é outro aplicativo.

HTTP , ou protocolo de transferência de hipertexto, rege como os navegadores da web e os servidores da web se comunicam nos pacotes TCP/IP.

Dois comandos suportados por HTTP incluem **GET** e **POST**. GET permite que um navegador solicite uma página ou arquivo, e POST permite que um navegador envie dados para o servidor.

Um **URL** ou endereço da web pode ser semelhante a *https://www.example.com/*.

- ***https*** é o protocolo que está sendo usado e, neste caso, HTTPS é a versão segura do HTTP, garantindo que o conteúdo dos pacotes entre o navegador e o servidor sejam criptografados.
- ***example.com*** é o nome de domínio, onde ***.com*** é o domínio de nível superior, convencionalmente indicando o “tipo” de site da Web, como um site comercial para .com ou uma organização para .org . Agora, existem centenas de domínios de nível superior, e suas restrições variam quanto a quem pode usá-los, mas muitos deles permitem que qualquer pessoa se registre para um domínio.

- **www** é o nome do host que, por convenção, nos indica que se trata de um serviço “world wide web”. Não é obrigatório, então hoje muitos sites não estão configurados para incluí-lo.
- Finalmente, o **/** no final é uma solicitação para o arquivo padrão, como index.html , com o qual o servidor da web responderá.

Uma solicitação HTTP começará com:

```
GET / HTTP / 1.1
Host: www.example.com
…
```

- O **GET** indica que a solicitação é para algum arquivo e **/** indica o arquivo padrão. Uma solicitação pode ser mais específica e começar com **GET /index.html**.
- Existem diferentes versões do protocolo HTTP, portanto, **HTTP / 1.1** indica que o navegador está usando a versão 1.1.
- **Host: www.example.com** indica que a solicitação é para ***www.example.com*** , pois o mesmo servidor da web pode hospedar vários sites e domínios.

Uma resposta começará com:

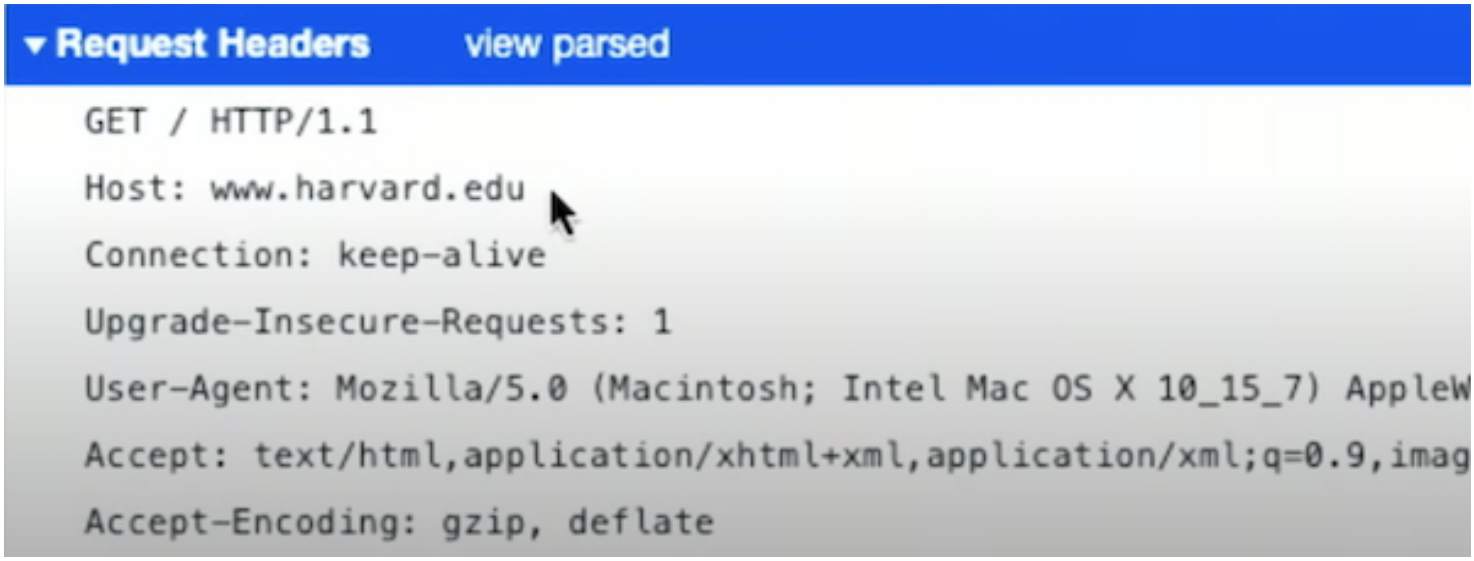
```
HTTP / 1.1 200 OK
Content-Type: text/html
…
```

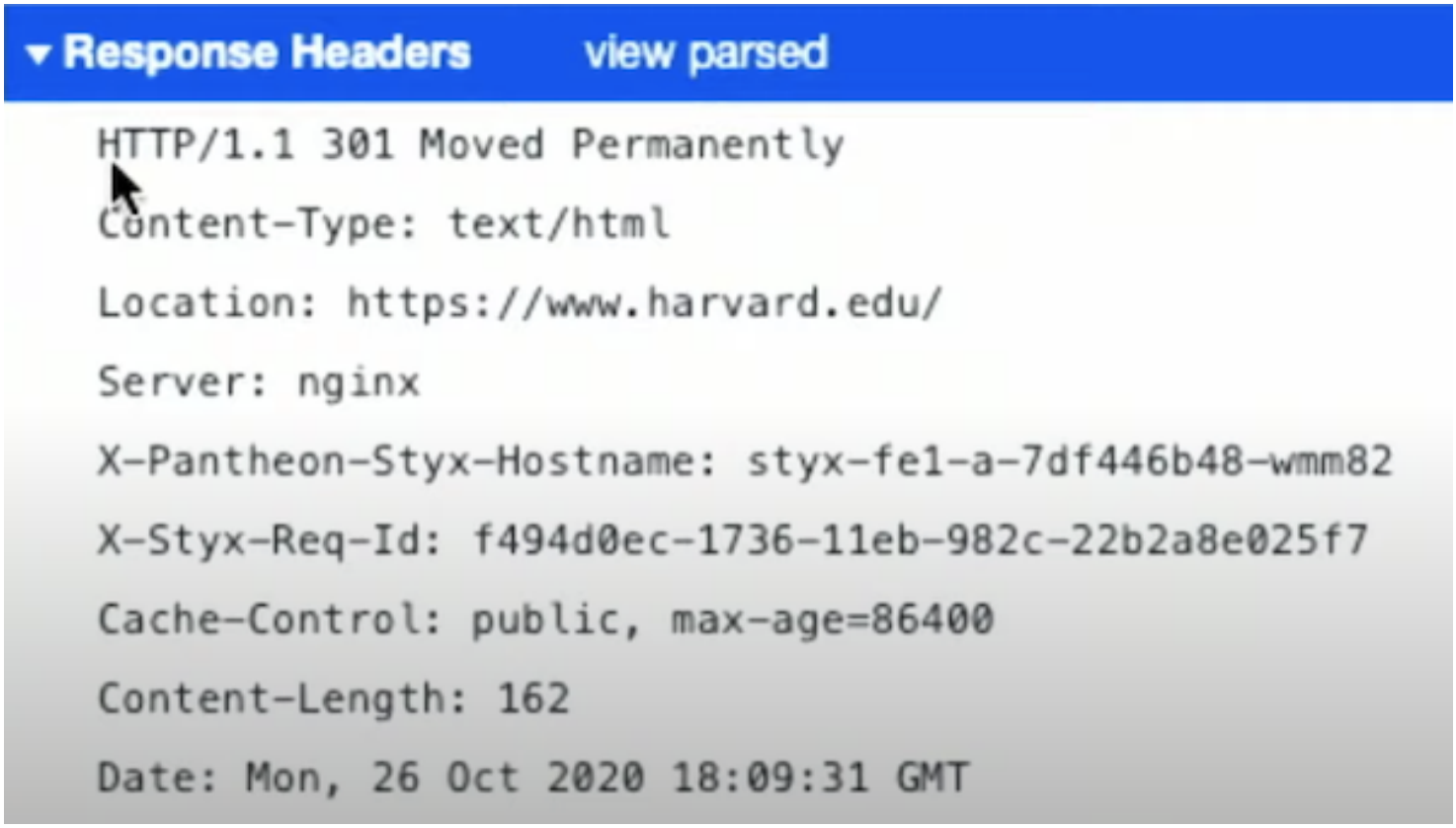
- O servidor web responderá com a versão do HTTP, seguido por um código de status, que é **200 OK** aqui, indicando que a solicitação era válida.
- Em seguida, o servidor da web indica o tipo de conteúdo em sua resposta, que pode ser texto, imagem ou outro formato.
- Finalmente, o resto do pacote ou pacotes incluirão o conteúdo.

Podemos ver um redirecionamento em um navegador digitando uma URL, como ***http://www.harvard.edu***, e olhando para a barra de endereço após o carregamento da página, que mostrará ***https://www.harvard.edu***. Os navegadores incluem ferramentas de desenvolvedor, que nos permitem ver o que está acontecendo. No menu do Chrome, por exemplo, podemos ir em Exibir> Desenvolvedor> Ferramentas do desenvolvedor, que abrirá um painel na tela. Na guia Rede, podemos ver que houve muitos pedidos de texto, imagens e outros dados que foram baixados separadamente para as páginas da web individuais.

A primeira solicitação, na verdade, retornou um código de status **301 Moved Permanently**, redirecionando nosso navegador de **http: //** ... para **https: //** ... :

A solicitação e a resposta também incluem vários cabeçalhos ou dados adicionais:





- Observe que a resposta inclui um cabeçalho Location: para qual o navegador nos redireciona.

Outros códigos de status HTTP incluem:

- 200 OK
- 301 Moved Permanently
- 304 Not Modified
  - Isso permite que o navegador use seu cache, ou cópia local, de algum recurso como uma imagem, em vez de fazer com que o servidor o envie de volta.

- 307 Temporary Redirect
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 418 I'm a Teapot
- 500 Internal Server Error
  - O código com erros em um servidor pode resultar neste código de status.
- 503 Service Unavailable
- ...

Podemos usar uma ferramenta de linha de comando, **curl** , para conectar a um URL. Podemos executar:

```
curl -I http://safetyschool.org
HTTP/1.1 301 Moved Permanently
Server: Sun-ONE-Web-Server/6.1
Date: Wed, 26 Oct 2020 18:17:05 GMT
Content-length: 122
Content-type: text/html
Location: http://www.yale.edu
Connection: close
```

- Acontece que ***safetyschool.org*** redireciona para ***yale.edu***!
- E ***harvardsucks.org*** é um site com outra pegadinha em Harvard!

Por fim, uma solicitação HTTP pode incluir entradas para servidores, como a string **q** = **cats** após o **?**:

```
GET / search?q=cats HTTP / 1.1
Host: www.google.com
…
```

Isso usa um formato padrão para passar input, como argumentos de linha de comando, para servidores da web.

HTML

Agora que podemos usar a Internet e o HTTP para enviar e receber mensagens, é hora de ver o que há no conteúdo das páginas da web. **HTML** , Hypertext Markup Language, não é uma linguagem de programação, mas sim usada para formatar páginas da web e dizer ao navegador como exibir as páginas, usando tags e atributos.

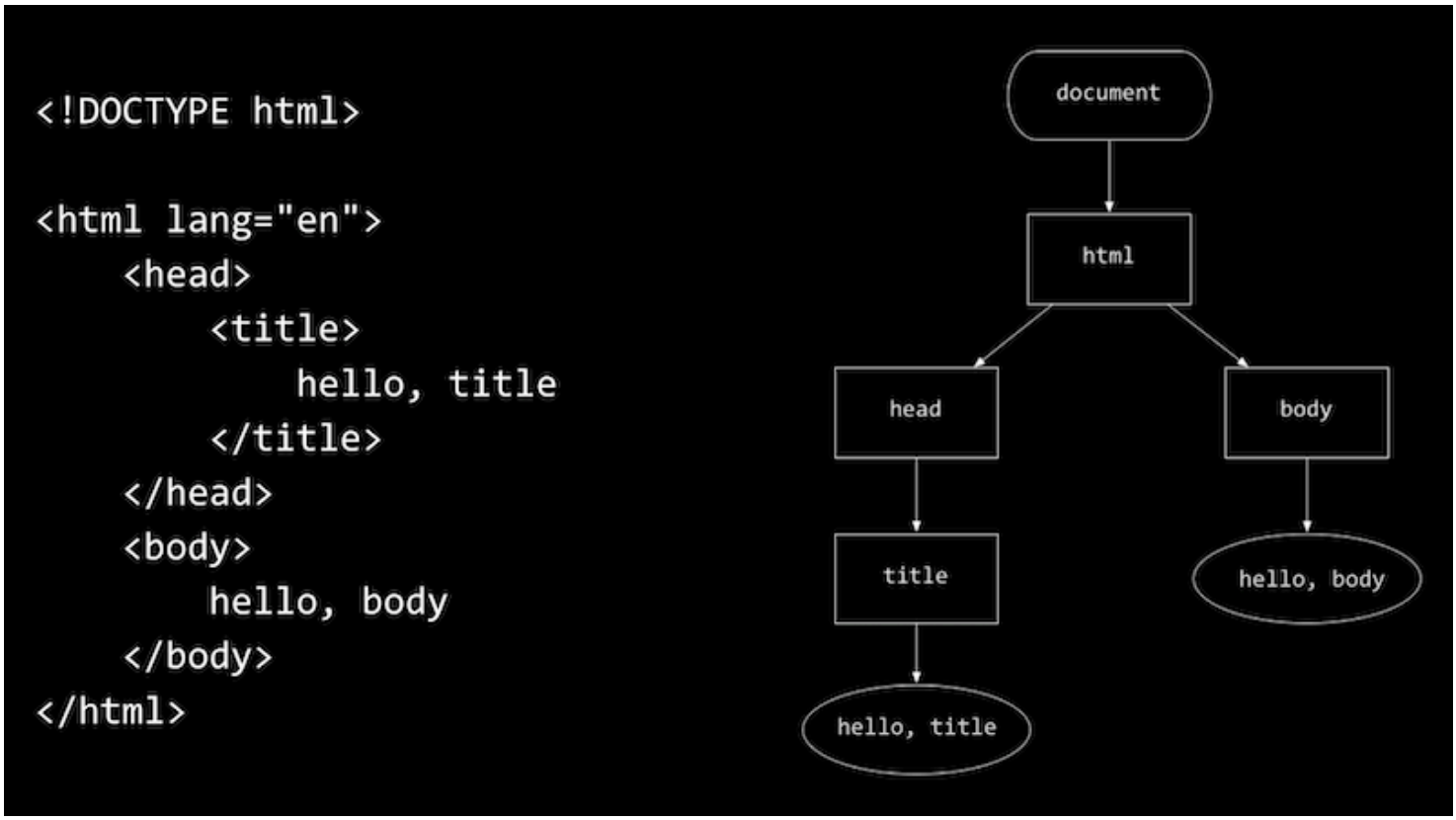
Uma página simples em HTML pode ter a seguinte aparência:

```
<! DOCTYPE html>
<html lang = "en" >
    <head>
        <title>
            hello, title
```

```
        </title>
    </head>
    <body>
        hello, body
    </body>
</html>
```

- A primeira linha é uma declaração de que a página segue o padrão HTML.
- A seguir vem uma **tag** , uma palavra entre colchetes, como **<html>** e **</html>**. A primeira é uma tag de início ou de abertura e a segunda é uma tag de fechamento. Nesse caso, as tags indicam o início e o fim da página HTML. A tag de início aqui também tem um atributo , **lang = "en"** que especifica que o idioma da página será em inglês, para ajudar o navegador a traduzir a página, se necessário. </p>
- Dentro da tag <html> há mais duas tags, **<head>** e **<body>**, que são como nós filhos em uma árvore. E dentro de **<head>** está a tag **<title>** , cujo conteúdo vemos em uma guia ou título de janela em um navegador. Em **<body>** está o conteúdo da própria página, que veremos na visualização principal de um navegador também.

A página acima será carregada no navegador como uma estrutura de dados, como esta árvore:



- Observe que há uma hierarquia mapeando cada tag e seus filhos. Nós retangulares são tags, enquanto os ovais são texto.

Podemos salvar o código acima como um HTML em nossos computadores locais, o que funcionaria em um navegador, mas apenas para nós. Com o IDE CS50, podemos criar um arquivo HTML e realmente disponibilizá-lo na Internet.

Vamos criar hello.html com o código acima, e iniciar um servidor web instalado no CS50 IDE com **http-server**, um programa que irá ouvir HTTP solicitações e responder com páginas ou outro conteúdo.

O próprio CS50 IDE já está sendo executado em algum servidor web, usando as portas 80 e 443, portanto, nosso próprio servidor web dentro do IDE terá que usar uma porta diferente, **8080** por padrão. Veremos uma URL longa, terminando em **cs50.ws** e, se abrirmos essa URL, veremos uma lista de arquivos, incluindo hello.html.

De volta ao terminal de nosso IDE, veremos novas linhas de texto impressas por nosso servidor web, um log de solicitações que ele está recebendo.

Vamos dar uma olhada em **paragraphs.html** .

- Com a tag **<p>**, podemos indicar que cada seção do texto deve ser um parágrafo.
- Depois de salvar este arquivo, precisaremos atualizar o índice no navegador da web e, em seguida, abrir paragraphs.html .

Podemos adicionar cabeçalhos com tags que começam com h e ter níveis de **1** a **6** em **headings.html**.

Também examinamos **list.html**, **table.html** e **image.html** para adicionar listas, tabelas e imagens.

- Podemos usar a tag **<ul>** para criar uma lista não ordenada, como uma lista em topicos, e **<ol>** para uma lista ordenada com números.
- As tabelas começam com uma marca **<table>** e têm marcas **<tr>** como linhas e marcas **<td>** para células individuais.
- Para image.html, podemos fazer upload de uma imagem para o IDE CS50, para incluí-la em nossa página, bem como usar o atributo **alt** para adicionar texto alternativo para acessibilidade.

Procurando documentação ou outros recursos online, podemos aprender as tags que existem em HTML e como usá-las.

Podemos criar links em **link.html** com a tag **<a>** ou âncora. O atributo **href** é para uma referência de hipertexto, ou simplesmente para onde o link deve nos levar, e dentro da tag está o texto que deve aparecer como o link.

- Poderíamos definir o **href** como ***https://www.yale.edu*** mas deixar ***Harvard*** dentro da tag, o que pode enganar os usuários ou até mesmo induzi-los a visitar uma versão falsa de algum site. **Phishing** é um ato de enganar os usuários, uma forma de engenharia social que inclui links enganosos.

Em search.html , podemos criar um formulário mais complexo que recebe a entrada do usuário e a envia para o mecanismo de pesquisa do Google:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>search</title>
    </head>
    <body>
        <form action="https://www.google.com/search" method="get">
            <input name="q" type="search">
            <input type="submit" value="Search">
        </form>
    </body>
</html>
```

- Primeiro, temos uma tag **<form>** que possui uma ação(**action**) do URL de pesquisa do Google, com um método de GET.
- Dentro do formulário, temos um **<input>** , com o nome **q**, e outro **<input>** com o tipo de envio(**submit**). Quando a segunda entrada, um botão, é clicado, o formulário irá anexar o texto na primeira entrada para o URL de ação, terminando com **search?q=...** .
- Portanto, quando abrimos **search.html** em nosso navegador, podemos usar o formulário para pesquisar através do Google.
- Um formulário também pode usar um método POST, que não inclui os dados do formulário no URL, mas em outro lugar na solicitação.

CSS

Podemos melhorar a estética de nossas páginas com CSS , Cascading Style Sheets, outra linguagem que informa ao nosso navegador como exibir tags em uma página. CSS usa **propriedades** ou pares de valores-chave, como **color: red;** para tags com seletores.

Em HTML, temos algumas opções para incluir CSS. Podemos adicionar uma tag <style> dentro da tag **<head>** , com estilos diretamente dentro, ou podemos vincular a um arquivo styles.css com uma tag **<link>** dentro da tag **<head>** .

Também podemos incluir CSS diretamente em cada tag:

```
<! DOCTYPE html>

<html lang = "en" >
    <head>
        <title> css </title>
    </head>
    <body>
        <header style = "font-size: large; text-align: center;" >
            John Harvard
        </header>
        <main style = "font-size: medium; text-align: center;" >
            Welcome to my homepagel!
        </main>
        <footer style = "font-size: small; text-align: center;" >
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

- As tags **<header>** , **<main>** e **<footer>** são como as tags **<p>** , indicando as seções em que o texto de nossa página está.
- Para cada tag, podemos adicionar um atributo de estilo(**style**), com o valor sendo uma lista de propriedades de valores-chave CSS, separadas por ponto e vírgula. Aqui, estamos definindo o tamanho da fonte(**font-size**) para cada tag e alinhando o texto no centro.
- Observe que podemos usar **©**, uma **entidade HTML**, como um código para incluir algum símbolo em nossa página da web.

Podemos alinhar o texto de uma só vez:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>css</title>
    </head>
    <body style="text-align: center;">
        <header style="font-size: large;">
            John Harvard
        </header>
        <main style="font-size: medium;">
            Welcome to my home page!
        </main>
        <footer style="font-size: small;">
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

- Aqui, o estilo aplicado à tag **<body>** é aplicado em cascata, ou se aplica a seus filhos, de forma que todas as seções internas também terão texto centralizado.

Para fatorar ou separar nosso CSS do HTML, podemos incluir estilos na tag **<head>** :

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <style>

            header
            {
                font-size: large;
                text-align: center;
            }

            main
            {
                font-size: medium;
                text-align: center;
            }

            footer
            {
                font-size: small;
                text-align: center;
            }

        </style>
        <title>css</title>
    </head>
    <body>
        <header>
            John Harvard
        </header>
        <main>
            Welcome to my home page!
        </main>
        <footer>
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

- Para cada tipo de tag, usamos um **seletor de tipo(type selector)** CSS para **definir** o estilo.

Também podemos usar um **seletor de class(class selector)** mais específico:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <style>

            .centered
            {
                text-align: center;
            }

            .large
            {
                font-size: large;
            }

            .medium
            {
                font-size: medium;
            }

            .small
            {
                font-size: small;
            }

        </style>
        <title>css</title>
    </head>
    <body>
        <header class="centered large">
            John Harvard
        </header>
        <main class="centered medium">
            Welcome to my home page!
        </main>
        <footer class="centered small">
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

- Podemos definir nossa própria classe CSS com um . seguido por uma palavra-chave que escolher, então aqui nós criamos **.large**, **.medium** e **.small**, cada um com alguma propriedade para o tamanho da fonte.
- Então, em qualquer número de tags no HTML de nossa página, podemos adicionar uma ou mais dessas classes com **class="centered large"** , reutilizando esses estilos.
- Podemos remover a redundância de centered e aplicá-la apenas à tag **<body>** também.

Finalmente, podemos pegar todo o CSS das propriedades e movê-los para outro arquivo com a tag **<link>**:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <link href="styles.css" rel="stylesheet">
        <title>css</title>
    </head>
    <body>
        <header class="centered large">
```

```
            John Harvard
        </header>
        <main class="centered medium">
            Welcome to my home page!
        </main>
        <footer class="centered small">
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

- Agora, uma pessoa pode trabalhar no HTML e outra pode trabalhar no CSS, de forma mais independente.

Com CSS, também contaremos com referências e outros recursos para descobrir como usar as propriedades conforme necessário.

Podemos usar **pseudoseletores (pseudoselectors)**, que selecionam certos estados:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <style>

            #harvard
            {
                color: #ff0000;
            }

            #yale
            {
                color: #0000ff;
            }

            a
            {
                text-decoration: none;
            }

            a:hover
            {
                text-decoration: underline;
            }

        </style>
        <title>link</title>
    </head>
    <body>
        Visit <a href="https://www.harvard.edu/" id="harvard" >Harvard</a> or <a href="https://www.yale.ed
u/" id="yale" >Yale</a>.
    </body>
</html>
```

- Aqui, estamos usando **a:hover** para definir propriedades em tags **<a>** quando o usuário passa o mouse sobre elas.
- Também temos um atributo **id** em cada tag **<a>**, para definir cores diferentes em cada um com **seletores de ID (ID selectors)** que começam com um # no CSS.

JavaScript

Para escrever um código que pode ser executado nos navegadores dos usuários ou no cliente, usaremos uma nova linguagem, **JavaScript**.

A sintaxe do JavaScript é semelhante à do C e do Python para construções básicas:

```
let counter = 0;

counter = counter + 1;
counter += 1;
counter++;

if(x < y)
{

}

if(x < y)
{

}
else
{

}

if(x < y)
{

}
else if(x > y)
{

}
else
{

}

while(true)
{

}

for(let i = 0; i < 3; i++)
{

}
```

- Observe que o JavaScript também é digitado livremente, com **let** sendo a palavra-chave para declarar variáveis de qualquer tipo.

Com o JavaScript, podemos alterar o HTML no navegador em tempo real. Podemos usar tags **<script>** para incluir nosso código diretamente ou de um arquivo **.js**.

Vamos criar outro formulário:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <script>

            function greet()
            {
```

```
                alert('hello, body');
            }

        </script>
        <title>hello</title>
    </head>
    <body>
        <form onsubmit="greet(); return false;">
            <input id="name" type="text">
            <input type="submit">
        </form>
    </body>
</html>
```

- Aqui, não adicionaremos uma ação ao nosso formulário, pois ele permanecerá na mesma página. Em vez disso, teremos um atributo onsubmit que chamará uma função que definimos em JavaScript e usará return false; para evitar que o formulário seja realmente enviado em qualquer lugar.
- Agora, se carregarmos essa página, veremos hello, body sendo mostrado quando enviarmos o formulário.

Como nossa tag de entrada, ou **elemento**, tem um ID de name , podemos usá-lo em nosso script:

```
<script>

    function greet()
    {
        let name = document.querySelector('#name').value;
        alert('hello, ' + name);
    }

</script>
```

- document é uma variável global que vem com JavaScript no navegador, e querySelector é outra função que podemos usar para selecionar um nó no DOM , Document Object Model ou a estrutura em árvore da página HTML. Depois de selecionar o elemento com o nome do ID , obtemos o value dentro do input e o adicionamos ao nosso alerta.
- Observe que JavaScript usa aspas simples para strings por convenção, embora aspas duplas também possam ser usadas, desde que correspondam a cada string.

Podemos adicionar mais atributos ao nosso formulário, para alterar o texto do espaço reservado, alterar o texto do botão, desativar o preenchimento automático ou focar o input automaticamente:

```
<form>
    <input autocomplete="off" autofocus id="name" placeholder="Name" type="text">
    <input type="submit">
</form>
```

Também podemos ouvir **eventos(events)** em JavaScript, que ocorrem quando algo acontece na página. Por exemplo, podemos ouvir o evento de **submit** em nosso formulário e chamar a função **greet**:

```
<script>

    function greet()
    {
        let name = document.querySelector('#name').value;
        alert('hello, ' + name);
```

```
    }

    function listen() {
        document.querySelector('form').addEventListener('submit', greet);
    }

    document.addEventListener('DOMContentLoaded', listen);

</script>
```

- Aqui, em **listen** , passamos a função **greet** por nome, e não a chamamos ainda. O ouvinte do evento irá chamá-lo para nós quando o evento acontecer.
- Precisamos primeiro ouvir o evento **DOMContentLoaded**, já que o navegador lê nosso arquivo HTML de cima para baixo, e o **form** não existiria até que ele lesse todo o arquivo e carregasse o conteúdo. Portanto, ao ouvir esse evento e chamar nossa função de **listen** , sabemos que form existirá.

Também podemos usar **funções anônimas** em JavaScript:

```
<script>

    document.addEventListener('DOMContentLoaded', function() {
        document.querySelector('form').addEventListener('submit', function() {
            let name = document.querySelector('#name').value;
            alert('hello, ' + name);
        });
    });

</script>
```

- Podemos passar uma função lambda com a sintaxe **function(),** então aqui passamos os dois ouvintes diretamente para **addEventListener**.

Além de **submit** , existem muitos outros eventos que podemos ouvir:

- **blur**
- **change**
- **click**
- **drag**
- **focus**
- **keyup**
- **load**
- **mousedown**
- **mouseover**
- **mouseup**
- **submit**
- **touchmove**
- **unload**
- **...**

Por exemplo, podemos ouvir o evento **keyup** e alterar o DOM assim que liberarmos uma tecla:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <script>

            document.addEventListener('DOMContentLoaded', function() {
                let input = document.querySelector('input');
                input.addEventListener('keyup', function(event) {
                    let name = document.querySelector('#name');
                    if (input.value) {
                        name.innerHTML = `hello, ${input.value}`;
                    }
                    else {
                        name.innerHTML = 'hello, whoever you are';
                    }
                });
            });

        </script>
        <title>hello</title>
    </head>
    <body>
        <form>
            <input autocomplete="off" autofocus placeholder="Name" type="text">
        </form>
        <p id="name"></p>
    </body>
</html>
```

- Observe que também podemos substituir strings em JavaScript, com **${input.value}** dentro de uma string cercada por crases, ` .

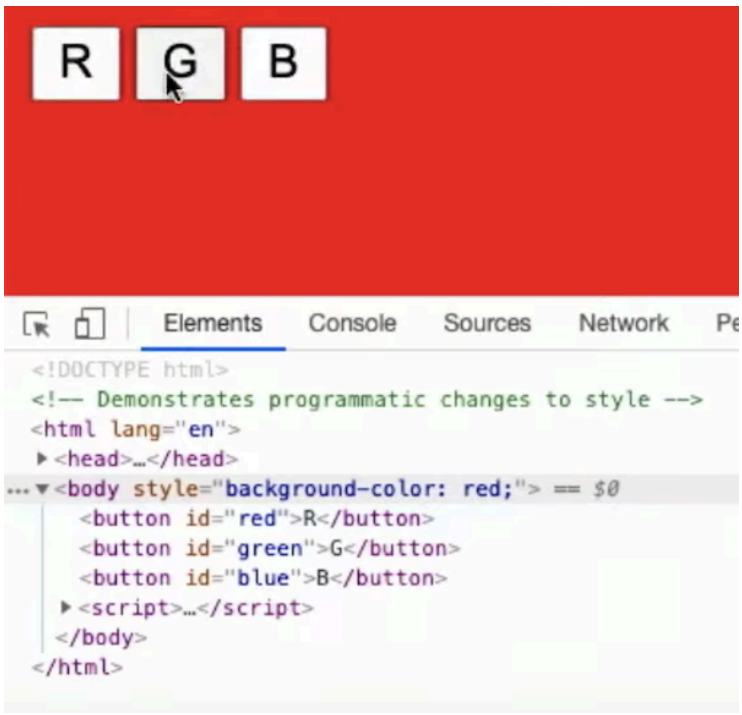
Também podemos alterar o estilo de maneira programática:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>background</title>
    </head>
    <body>
        <button id="red">R</button>
        <button id="green">G</button>
        <button id="blue">B</button>
        <script>

            let body = document.querySelector('body');
            document.querySelector('#red').onclick = function() {
                body.style.backgroundColor = 'red';
            };
            document.querySelector('#green').onclick = function() {
                body.style.backgroundColor = 'green';
            };
            document.querySelector('#blue').onclick = function() {
                body.style.backgroundColor = 'blue';
            };

        </script>
    </body>
</html>
```

Depois de selecionar um elemento, podemos usar a propriedade **style** para definir os valores das propriedades CSS também. Aqui, temos três botões, cada um dos quais com um ouvinte ***onclick*** que altera a cor de fundo do elemento **<body>**

- Observe aqui que nossa tag **<script>** está no final de nosso arquivo HTML, então não precisamos ouvir o evento **DOMContentLoaded** , uma vez que o resto do DOM já terá sido lido pelo navegador.

Também nas ferramentas de desenvolvedor de um navegador, podemos ver o DOM e todos os estilos aplicados por meio da guia Elements :



- Podemos até usar isso para alterar uma página em nosso navegador depois de carregada, clicando em algum elemento e editando o HTML. Mas essas alterações serão feitas apenas em nosso navegador, não em nosso arquivo HTML original ou em alguma página da web em outro lugar.

Em **size.html**, podemos definir o tamanho da fonte com um menu suspenso via JavaScript, e em **blink.html** podemos fazer um elemento "piscar", alternando entre visível e oculto.

Com **geolocation.html**, podemos pedir ao navegador as coordenadas GPS de um usuário e, com **autocomplete.html**, podemos preencher automaticamente algo que digitamos, com palavras de um arquivo de dicionário.

Finalmente, podemos usar Python para escrever código que se conecta a outros dispositivos em uma rede local, como uma lâmpada, por meio de uma API , interface de programação de aplicativo. A **API** da nossa lâmpada, em particular, aceita solicitações em determinados URLs:

```
import os
import requests

USERNAME = os.getenv("USERNAME")
IP = os.getenv("IP")

URL = f"http://{IP}/api/{USERNAME}/lights/1/state"
```

```
requests.put(URL, json={"on": False})
```

- Com este código, podemos usar o método PUT para enviar uma mensagem para a nossa lâmpada, desligando-a.
- Usamos variáveis de ambiente, valores armazenados em outro lugar em nosso computador, para nosso nome de usuário e endereço IP.

Agora, com um pouco mais de lógica, podemos fazer nossa lâmpada piscar:

```
import os
import requests
import time

USERNAME = os.getenv("USERNAME")
IP = os.getenv("IP")
URL = f"http://{IP}/api/{USERNAME}/lights/1/state"

while True:
    requests.put(URL, json={"bri": 254, "on": True})
    time.sleep(1)
    requests.put(URL, json={"on": False})
    time.sleep(1)
```

- Vamos montar HTML, CSS, JavaScript, Python e SQL na próxima vez!