Esto es CS50x

OpenCourseWare

Donar (https://cs50.harvard.edu/donate) (https://community.alumni.harvard.edu/give/59206872)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan) (https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/) (https://orcid.org/0000-0001-5338-2522) (https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan) (https://twitter.com/davidjmalan)

Clase 8

- La Internet
- desarrollo web
- HTML
- CSS
- JavaScript

La Internet

- Hoy echaremos un vistazo a la programación web, utilizando un conjunto de nuevos lenguajes y tecnologías para crear aplicaciones gráficas y visuales para Internet.
- El **Internet** es la red de redes de ordenadores que se comunican entre sí, que proporciona la infraestructura para enviar ceros y unos. Sobre esa base, podemos crear aplicaciones que envían y reciben datos.
- **Los enrutadores** son computadoras especializadas, con CPU y memoria, cuyo propósito es transmitir datos a través de cables o tecnologías inalámbricas, entre otros dispositivos en Internet.
- **Los protocolos** son un conjunto de convenciones estándar, como un apretón de manos físico, que el mundo ha acordado para que las computadoras se comuniquen. Por ejemplo, hay ciertos patrones de ceros y unos, o mensajes, que una computadora debe usar para decirle a un enrutador a dónde quiere enviar datos.
- **TCP / IP** son dos protocolos para enviar datos entre dos computadoras. En el mundo real, podríamos escribir una dirección en un sobre para enviar una carta a alguien, junto con nuestra propia dirección para recibir una carta a cambio. La versión digital de un sobre, o un mensaje con direcciones de origen y destino, se denomina **paquete** .
- **IP** significa protocolo de Internet, un protocolo compatible con el software de las computadoras modernas, que incluye una forma estándar para que las computadoras se dirijan entre sí. **Las direcciones IP** son **direcciones** únicas para las computadoras conectadas a Internet, de modo que un paquete enviado de una computadora a otra se pasará a través de los enrutadores hasta que llegue a su destino.
    - Los enrutadores tienen, en su memoria, una tabla que asigna direcciones IP a cables, cada uno de ellos conectado a otros enrutadores, para que sepan a dónde reenviar los paquetes. Resulta que existen protocolos para que los enrutadores se comuniquen y descubran estas rutas también.
- **DNS** , sistema de nombres de dominio, es otra tecnología que traduce nombres de dominio como cs50.harvard.edu a direcciones IP. El DNS generalmente es proporcionado como un servicio por el proveedor de servicios de Internet más cercano, o ISP.
- Finalmente, **TCP** , protocolo de control de transmisión, es un protocolo final que permite que un solo servidor, en la misma dirección IP, brinde múltiples servicios mediante el uso de un **número de puerto** , un pequeño entero agregado a la dirección IP. Por ejemplo, HTTP, HTTPS, correo electrónico e incluso Zoom tienen sus propios números de puerto para que esos programas los utilicen para comunicarse a través de la red.
- TCP también proporciona un mecanismo para reenviar paquetes si un paquete se pierde y no se recibe de alguna manera. Resulta que, en Internet, hay varias rutas para enviar un paquete, ya que hay muchos enrutadores que están interconectados. Por lo tanto, un navegador web, que realiza una solicitud para un gato, puede ver su paquete enviado a través de una ruta de enrutadores, y el servidor que responde puede ver sus paquetes de respuesta enviados a través de otra.
    - Una gran cantidad de datos, como una imagen, se dividirá en trozos más pequeños para que los paquetes sean todos de un tamaño similar. De esta manera, los enrutadores de Internet pueden enviar los paquetes de todos de manera más justa y sencilla. **La**

neutralidad de la red se refiere a la idea de que estos enrutadores públicos tratan los paquetes por igual, en lugar de permitir que se prioricen los paquetes de ciertas empresas o de ciertos tipos.

- Cuando hay varios paquetes para una sola respuesta, TCP también especificará que cada uno de ellos sea etiquetado, como con “1 de 2” o “2 de 2”, para que se puedan combinar o reenviar según sea necesario.

- Con todas estas tecnologías y protocolos, podemos enviar datos de una computadora a otra y podemos abstraer Internet para crear aplicaciones en la parte superior.

desarrollo web

- La web es una aplicación que se ejecuta en Internet, lo que nos permite obtener páginas web. Otras aplicaciones como Zoom proporcionan videoconferencias y el correo electrónico también es otra aplicación.
- **HTTP** , o Protocolo de transferencia de hipertexto, gobierna cómo los navegadores web y los servidores web se comunican dentro de los paquetes TCP / IP.
- Dos comandos compatibles con HTTP incluyen **GET** y **POST** . GET permite que un navegador solicite una página o un archivo, y POST permite que un navegador envíe datos *al* servidor.

- Una **URL** , o dirección web, podría verse así `https://www.example.com/`.
  - `https` es el protocolo que se utiliza y, en este caso, HTTPS es la versión segura de HTTP, lo que garantiza que el contenido de los paquetes entre el navegador y el servidor esté cifrado.
  - `example.com` es el nombre de dominio, donde `.com` es el dominio de nivel superior, que indica convencionalmente el "tipo" de sitio web, como un sitio web comercial para el `.com` o una organización `.org`. Ahora hay cientos de dominios de nivel superior y varían en cuanto a las restricciones sobre quién puede usarlos, pero muchos de ellos permiten que cualquiera se registre para obtener un dominio.
  - `www` es el nombre de host que, por convención, nos indica que se trata de un servicio de “world wide web”. No es obligatorio, por lo que hoy en día muchos sitios web no están configurados para incluirlo.
  - Finalmente, `/` al final hay una solicitud para el archivo predeterminado, como `index.html`, con el que responderá el servidor web.
- Una solicitud HTTP comenzará con:

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

  - El `GET` indica que la solicitud es para algún archivo e `/` indica el archivo predeterminado. Una solicitud podría ser más específica y comenzar con `GET /index.html`.
  - Existen diferentes versiones del protocolo HTTP, por lo `HTTP/1.1` que indica que el navegador está usando la versión 1.1.
  - `Host: www.example.com` indica que la solicitud es para `www.example.com`, ya que el mismo servidor web podría albergar varios sitios web y dominios.
- Una respuesta comenzará con:

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

  - El servidor web responderá con la versión de HTTP, seguida de un código de estado, que está `200 OK` aquí, indicando que la solicitud fue válida.
  - Luego, el servidor web indica el tipo de contenido en su respuesta, que puede ser texto, imagen u otro formato.
  - Finalmente, el resto del paquete o paquetes incluirá el contenido.
- Podemos ver una redirección en un navegador escribiendo una URL, como `http://www.harvard.edu`, y mirando la barra de direcciones después de que se haya cargado la página, que se mostrará `https://www.harvard.edu`. Los navegadores incluyen herramientas de desarrollo, que nos permiten ver lo que está sucediendo. En el menú de Chrome, por ejemplo, podemos ir a Ver> Desarrollador> Herramientas de desarrollo, que abrirá un panel en la pantalla. En la pestaña Red, podemos ver que hubo muchas solicitudes de texto, imágenes y otros datos que se descargaron por separado para las páginas web individuales.
- La primera solicitud en realidad devolvió un código de estado de `301 Moved Permanently`, redirigiendo nuestro navegador de `http://...` a `https://...`:

- La solicitud y la respuesta también incluyen una serie de encabezados o datos adicionales:

```
GET / HTTP/1.1
Host: www.harvard.edu
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleW
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,imag
Accept-Encoding: gzip, deflate
```

Response Headers view parsed

```
HTTP/1.1 301 Moved Permanently
Content-Type: text/html
Location: https://www.harvard.edu/
Server: nginx
X-Pantheon-Styx-Hostname: styx-fe1-a-7df446b48-wmm82
X-Styx-Req-Id: f494d0ec-1736-11eb-982c-22b2a8e025f7
Cache-Control: public, max-age=86400
Content-Length: 162
Date: Mon, 26 Oct 2020 18:09:31 GMT
```

- Tenga en cuenta que la respuesta incluye un `Location:` encabezado para que el navegador nos redirija.

- Otros códigos de estado HTTP incluyen:
  - `200 OK`
  - `301 Moved Permanently`
  - `304 Not Modified`
    - Esto permite que el navegador use su caché, o copia local, de algún recurso como una imagen, en lugar de que el servidor lo devuelva.
  - `307 Temporary Redirect`
  - `401 Unauthorized`
  - `403 Forbidden`
  - `404 Not Found`
  - `418 I'm a Teapot`
  - `500 Internal Server Error`
    - El código defectuoso en un servidor puede resultar en este código de estado.
  - `503 Service Unavailable`
  - …
- Podemos usar una herramienta de línea de comandos `curl`, para conectarnos a una URL. Podemos ejecutar:

```
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
```

  - ¡Resulta que `safetyschool.org` redirige a `yale.edu`!
  - ¡Y `harvardsucks.org` es un sitio web con otra broma sobre Harvard!
- Finalmente, una solicitud HTTP puede incluir entradas a servidores, como la cadena `q=cats` después de `?`:

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

- Utiliza un formato estándar para pasar entradas, como argumentos de línea de comandos, a servidores web.

HTML

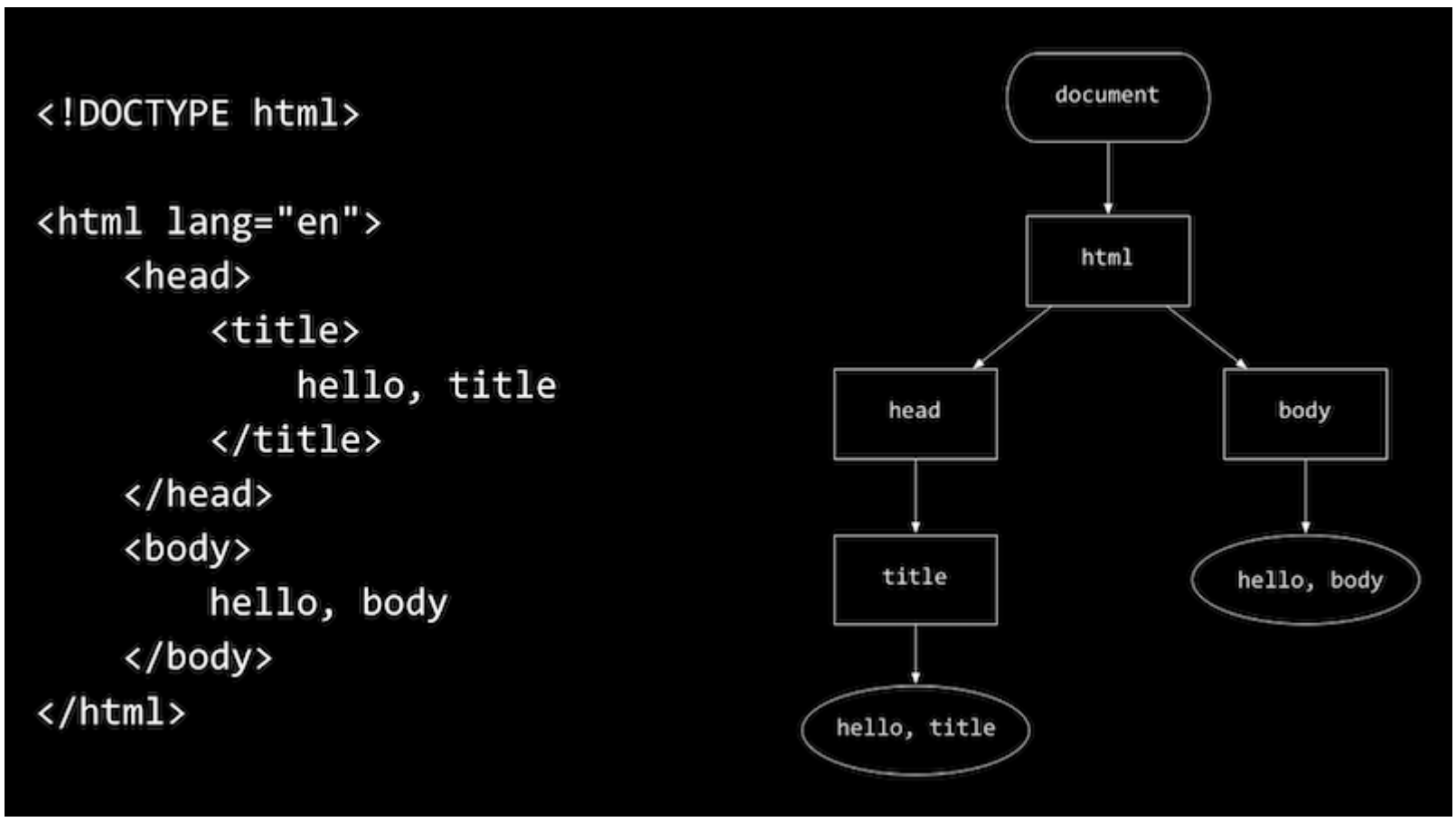
- Ahora que podemos usar Internet y HTTP para enviar y recibir mensajes, es hora de ver qué hay en el contenido de las páginas web. **HTML** , lenguaje de marcado de hipertexto, no es un lenguaje de programación, sino que se utiliza para formatear páginas web y decirle al navegador cómo mostrar las páginas, utilizando etiquetas y atributos.
- Una página simple en HTML podría verse así:

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>
            hello, title
        </title>
    </head>
    <body>
        hello, body
    </body>
</html>
```

- La primera línea es una declaración de que la página sigue el estándar HTML.
- Lo siguiente es una **etiqueta** , una palabra entre paréntesis como `<html>` y `</html>`. La primera es una etiqueta de inicio o apertura y la segunda es una etiqueta de cierre. En este caso, las etiquetas indican el inicio y el final de la página HTML. La etiqueta de inicio aquí también tiene un **atributo** , `lang="en"` que especifica que el idioma de la página será en inglés, para ayudar al navegador a traducir la página si es necesario.
- Dentro de la `<html>` etiqueta hay dos etiquetas más, `<head>` y `<body>` , que son como nodos secundarios en un árbol. Y dentro `<head>` está la `<title>` etiqueta, cuyo contenido vemos en el título de una pestaña o ventana en un navegador. Dentro `<body>` está el contenido de la página en sí, que también veremos en la vista principal de un navegador.

- La página anterior se cargará en el navegador como una estructura de datos, como este árbol:



- Tenga en cuenta que existe una jerarquía que asigna cada etiqueta y sus hijos. Los nodos rectangulares son etiquetas, mientras que los ovalados son texto.

- Podemos guardar el código anterior como HTML en nuestras computadoras locales, lo que funcionaría en un navegador, pero solo para nosotros. Con CS50 IDE, podemos crear un archivo HTML y ponerlo a disposición en Internet.
- Crearemos `hello.html` con el código anterior e iniciaremos un servidor web instalado en el IDE CS50 con `http-server` un programa

que escuchara las solicitudes HTTP y respondera con paginas u otro contenido.

- El CS50 IDE ya se está ejecutando en algún servidor web, usando los puertos 80 y 443, por lo que nuestro propio servidor web dentro del IDE tendrá que usar un puerto diferente, `8080` por defecto. Veremos una URL larga que termina en `cs50.ws`, y si abrimos esa URL, veremos una lista de archivos, incluidos `hello.html`.
- De vuelta en la terminal de nuestro IDE, veremos nuevas filas de texto impresas por nuestro servidor web, un registro de solicitudes que está recibiendo.
- Echaremos un vistazo `paragraphs.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/paragraphs0.html?highlight).
  - Con la `<p>` etiqueta podemos indicar que cada sección de texto debe ser un párrafo.
  - Después de guardar este archivo, necesitaremos actualizar el índice en el navegador web y luego abrirlo `paragraphs.html`.
- Podemos añadir títulos con etiquetas que comienzan con `h`, y tienen niveles de `1` a través `6` de `headings.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/headings.html?highlight).
- Echamos un vistazo a `list.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/list.html?highlight), `table.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/table.html?highlight)y `image.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/image.html?highlight)también para agregar listas, tablas e imágenes.
  - Podemos usar la `<ul>` etiqueta para crear una lista desordenada, como viñetas, y `<ol>` para una lista ordenada con números.
  - Las tablas comienzan con una `<table>` etiqueta y tienen `<tr>` etiquetas como filas y `<td>` etiquetas para celdas individuales.
  - Para `image.html`, podemos subir una imagen al CS50 IDE, para incluirla en nuestra página, así como usar el `alt` atributo para agregar texto alternativo para accesibilidad.
- Al buscar documentación u otros recursos en línea, podemos aprender las etiquetas que existen en HTML y cómo usarlas.
- Podemos crear enlaces `link.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/link0.html?highlight)con la `<a>` etiqueta o ancla. El `href` atributo es para una referencia de hipertexto, o simplemente a dónde debe llevarnos el enlace, y dentro de la etiqueta está el texto que debería aparecer como enlace.
  - Podríamos configurar el `href` to `https://www.yale.edu`, pero dejar `Harvard` dentro de la etiqueta, lo que podría hacer bromas a los usuarios o incluso engañarlos para que visiten una versión falsa de algún sitio web. **El phishing** es un acto de engañar a los usuarios, una forma de ingeniería social que incluye enlaces engañosos.
- En `search.html`, podemos crear un formulario más complejo que toma la entrada del usuario y la envía al motor de búsqueda de 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>
```

  - Primero, tenemos una `<form>` etiqueta que tiene una `action` URL de búsqueda de Google, con un método de GET.
  - Dentro del formulario, tenemos uno `<input>`, con el nombre `q`, y otro `<input>` con el tipo de `submit`. Cuando se hace clic en la segunda entrada, un botón, el formulario agregará el texto de la primera entrada a la URL de la acción, finalizándola con `search?q=...`.
  - Entonces, cuando abrimos `search.html` en nuestro navegador, podemos usar el formulario para buscar a través de Google.
  - Un formulario también puede usar un método POST, que no incluye los datos del formulario en la URL, sino en otra parte de la solicitud.

CSS

- Podemos mejorar la estética de nuestras páginas con **CSS** , Cascading Style Sheets, otro lenguaje que le dice a nuestro navegador cómo mostrar etiquetas en una página. CSS utiliza **propiedades** o pares clave-valor, como `color: red;` etiquetas con selectores.
- En HTML, tenemos algunas opciones para incluir CSS. Podemos agregar una `<style>` etiqueta dentro de la `<head>` etiqueta, con estilos directamente adentro, o podemos vincular a un `styles.css` archivo con una `<link>` etiqueta dentro de la `<head>` etiqueta.
- También podemos incluir CSS directamente en cada etiqueta:

```
<!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 home page!
        </main>
        <footer style="font-size: small; text-align: center;">
            Copyright &#169; John Harvard
        </footer>
    </body>
</html>
```

- `<header>`, `<main>` y las `<footer>` etiquetas son como `<p>` etiquetas, que indican las secciones en las que se encuentra el texto de nuestra página.

- Para cada etiqueta, podemos agregar un `style` atributo, siendo el valor una lista de propiedades clave-valor de CSS, separadas por punto y coma. Aquí, estamos configurando `font-size` para cada etiqueta y alineando el texto en el centro.
- Tenga en cuenta que podemos usar `&#169;`, una **entidad HTML** , como código para incluir algún símbolo en nuestra página web.
- Podemos alinear todo el texto a la 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>
```

  - Aquí, el estilo aplicado a la `<body>` etiqueta se aplica en cascada, o se aplica, a sus elementos secundarios, por lo que todas las secciones del interior también tendrán texto centrado.
- Para factorizar, o separar nuestro CSS de HTML, podemos incluir estilos en la `<head>` etiqueta:

```
<!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 etiqueta, usamos un **selector de tipo** CSS para darle estilo.
- También podemos usar un **selector de clases** más 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 nuestra propia clase CSS con un `.` seguido de una palabra clave que elegimos, por lo que aquí hemos creado `.large`, `.medium` y `.small`, cada uno con alguna propiedad para el tamaño de la fuente.
  - Luego, en cualquier número de etiquetas en el HTML de nuestra página, podemos agregar una o más de estas clases con `class="centered large"`, reutilizando estos estilos.
  - Podemos eliminar la redundancia `centered` y aplicarla también solo a la `<body>` etiqueta.
- Finalmente, podemos tomar todo el CSS de las propiedades y moverlas a otro archivo con la `<link>` etiqueta:

```
<!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>
```

- Ahora, una persona puede trabajar en HTML y otra puede trabajar en CSS, de forma más independiente.

- Con CSS, también nos basaremos en referencias y otros recursos para buscar cómo usar las propiedades cuando las necesitemos.
- Podemos usar **pseudoelectores** , que selecciona ciertos 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.edu/"
 id="yale" >Yale</a>.
    </body>
</html>
```

- Aquí, estamos usando `a:hover` para establecer propiedades en `<a>` etiquetas cuando el usuario pasa el mouse sobre ellas.
- También tenemos un `id` atributo en cada `<a>` etiqueta, para establecer diferentes colores en cada una con **selectores de ID** que comienzan con una `#` en CSS.

JavaScript

- Para escribir código que pueda ejecutarse en los navegadores de los usuarios o en el cliente, usaremos un nuevo lenguaje, **JavaScript** .
- La sintaxis de JavaScript es similar a la de C y Python para construcciones 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++)
{

}
```

- Tenga en cuenta que JavaScript también se escribe libremente, `let` siendo la palabra clave para declarar variables de cualquier tipo.

- Con JavaScript, podemos cambiar el HTML en el navegador en tiempo real. Podemos usar `<script>` etiquetas para incluir nuestro código directamente o desde un `.js` archivo.
- Crearemos otro formulario:

```
<!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>
```

- Aquí, no agregaremos un `action` a nuestro formulario, ya que este permanecerá en la misma página. En su lugar, tendremos un `onsubmit` atributo que llamará a una función que hemos definido en JavaScript y lo usaremos `return false;` para evitar que el formulario se envíe a cualquier lugar.
- Ahora, si cargamos esa página, veremos que `hello, body` se muestra cuando enviemos el formulario.

- Dado que nuestra etiqueta de entrada, o **elemento** , tiene un ID de `name` , podemos usarlo en nuestro script:

```
<script>

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

</script>
```

- `document` es una variable global que viene con JavaScript en el navegador, y `querySelector` es otra función que podemos usar para seleccionar un nodo en el **DOM** , Document Object Model o la estructura de árbol de la página HTML. Después de seleccionar el elemento con el ID `name` , obtenemos el `value` interior de la entrada y lo agregamos a nuestra alerta.
- Tenga en cuenta que JavaScript usa comillas simples para cadenas por convención, aunque se pueden usar comillas dobles siempre que coincidan con cada cadena.

- Podemos agregar más atributos a nuestro formulario, para cambiar el texto del marcador de posición, cambiar el texto del botón, deshabilitar el autocompletar o enfocar automáticamente la entrada:

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

- También podemos escuchar **eventos** en JavaScript, que ocurren cuando algo sucede en la página. Por ejemplo, podemos escuchar el `submit` evento en nuestro formulario y llamar a la `greet` función:

```
<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>
```

- Aquí, `listen` pasamos la función `greet` por nombre, y aún no la llamamos. El oyente de eventos lo llamará por nosotros cuando ocurra el evento.
- Primero debemos escuchar el `DOMContentLoaded` evento, ya que el navegador lee nuestro archivo HTML de arriba a abajo, y `form` no existiría hasta que lea todo el archivo y cargue el contenido. Entonces, al escuchar este evento y llamar a nuestra `listen` función, sabemos `form` que existirá.

- También podemos usar **funciones anónimas** en JavaScript:

```
<script>

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

</script>
```

    - Podemos pasar una función lambda con la `function()` sintaxis, así que aquí hemos pasado ambos listeneres directamente a `addEventListener`.
- Además `submit`, hay muchos otros eventos que podemos escuchar:
    - `blur`
    - `change`
    - `click`
    - `drag`
    - `focus`
    - `keyup`
    - `load`
    - `mousedown`
    - `mouseover`
    - `mouseup`
    - `submit`
    - `touchmove`
    - `unload`
    - `...`
- Por ejemplo, podemos escuchar el `keyup` evento y cambiar el DOM tan pronto como liberemos una 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>
```

    - Tenga en cuenta que podemos sustituir cadenas en JavaScript, así, con el `${input.value}` interior de una cadena rodeada de acentos abiertos, `` ` ``.
- También podemos cambiar el estilo mediante programación:

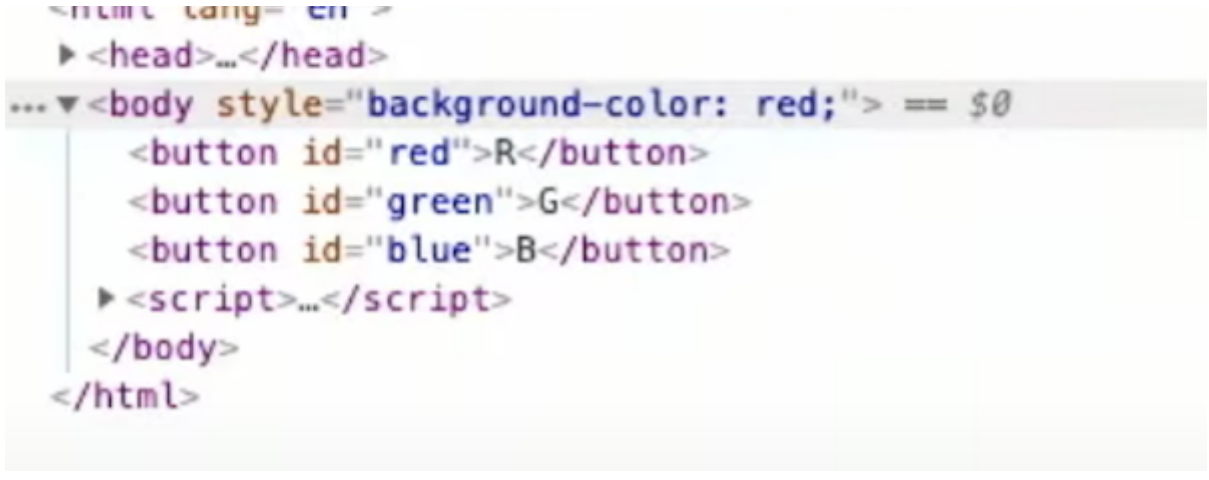
```
<!DOCTYPE html>

<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>
```

- Después de seleccionar un elemento, también podemos usar la `style` propiedad para establecer valores para las propiedades de CSS. Aquí, tenemos tres botones, cada uno de los cuales tiene un `onclick` oyente que cambia el color de fondo del `<body>` elemento.
    - Observe aquí que nuestra `<script>` etiqueta está al final de nuestro archivo HTML, por lo que no necesitamos escuchar el `DOMContentLoaded` evento, ya que el navegador ya habrá leído el resto del DOM.
- En las herramientas de desarrollo de un navegador, también podemos ver el DOM y cualquier estilo aplicado a través de la `Elements` pestaña:



  - Incluso podemos usar esto para cambiar una página en nuestro navegador después de que se cargue, haciendo clic en algún elemento y editando el HTML. Pero estos cambios solo se realizarán en nuestro navegador, no en nuestro archivo HTML original o en alguna página web en otro lugar.
- En `size.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/size.html?highlight), podemos establecer el tamaño de fuente con un menú desplegable a través de JavaScript, y en `blink.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/blink.html?highlight)podemos hacer que un elemento "parpadee", alternando entre visible y oculto.

- Con `geolocation.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/geolocation.html?highlight), podemos pedirle al navegador las coordenadas GPS de un usuario, y con `autocomplete.html` (https://cdn.cs50.net/2020/fall/lectures/8/src8/autocomplete.html?highlight), podemos autocompletar algo que escribimos, con palabras de un archivo de diccionario.
- Finalmente, podemos usar Python para escribir código que se conecte a otros dispositivos en una red local, como una bombilla, a través de una **API** , interfaz de programación de aplicaciones. La API de nuestra bombilla, en particular, acepta solicitudes en ciertas URL:

```
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})
```

  - Con este código podemos utilizar el método PUT para enviar un mensaje a nuestra bombilla, apagándola.
  - Usamos variables de entorno, valores almacenados en otro lugar de nuestra computadora, para nuestro nombre de usuario y dirección IP.
- Ahora, con un poco más de lógica, podemos hacer que nuestra bombilla parpadee:

```
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)
```

- ¡Reuniremos HTML, CSS, JavaScript, Python y SQL la próxima vez!