DESARROLLO WEB FULLSTACK

con Python y JavaScript

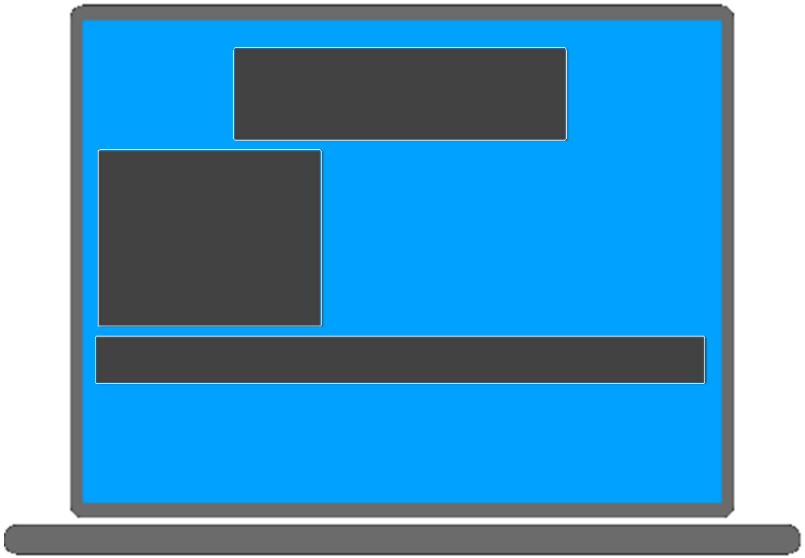
polotic misiones

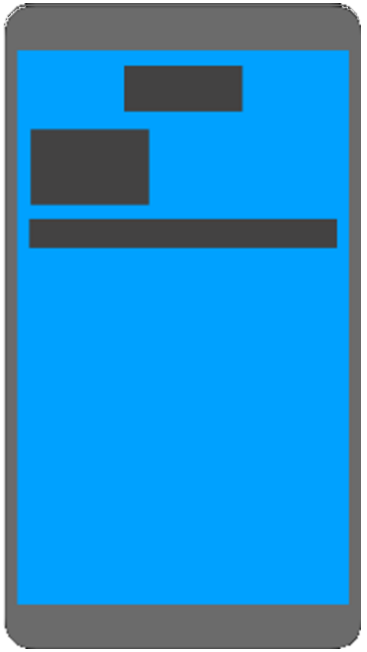
Diseño Responsivo

Diseño Responsivo

- `viewport`
- Media Queries
- `flexbox`
- `grid`

Sin Diseño Responsivo



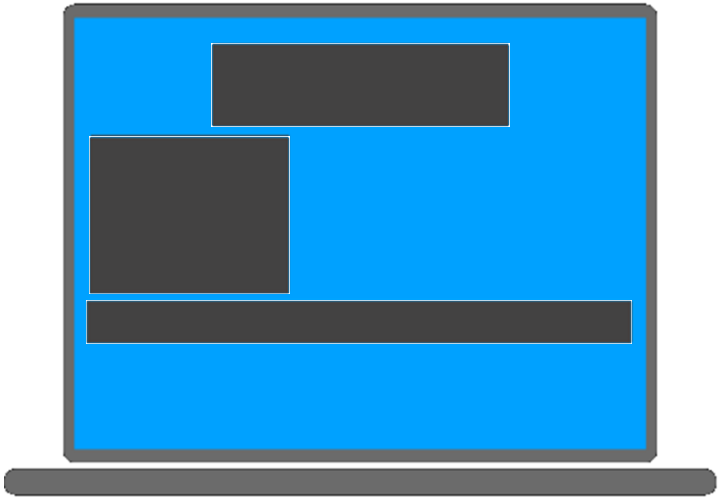


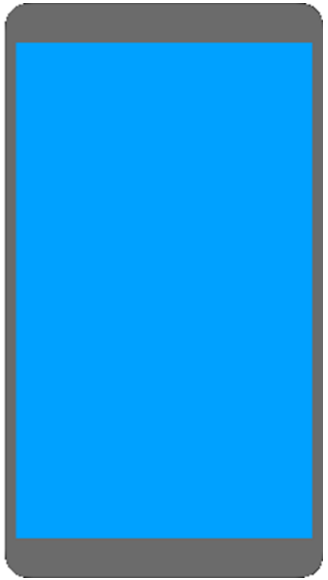
Diseño Responsivo - Viewport

- Viewport es el área visible del usuario de una página web. Ésta varía según el dispositivo y será más pequeña en un teléfono móvil que en una pantalla de computadora.

- Antes las páginas web estaban diseñadas solo para pantallas de computadora, y era común que las páginas web tuvieran un diseño estático y un tamaño fijo.
- Luego, cuando comenzamos a navegar por Internet usando tabletas y teléfonos móviles, las páginas web de tamaño fijo eran demasiado grandes para caber en el viewport.
- Para solucionar esto, los navegadores en esos dispositivos redujeron la página web completa para adaptarse a la pantalla. Si bien no fue la mejor de las soluciones, fue una solución rápida.

Usando Viewport





Diseño Responsivo - Viewport

HTML5 introdujo un método para permitir que los diseñadores web tomen el control del viewport, a través de la etiqueta `<meta>`.

Debes incluir el siguiente elemento de viewport `<meta>` en todas tus páginas web:

```
<meta name = "viewport" content = "width = device-width, initial-scale = 1.0">
```

Esto le da al navegador instrucciones sobre cómo controlar las dimensiones y la escala de la página.

- La parte `width = device-width` establece el ancho de la página para seguir el ancho de la pantalla (que variará según el dispositivo).
- La parte `initial-scale = 1.0` establece el nivel de zoom inicial cuando el navegador carga la página por primera vez.

Viewport – Tamaño del Contenido

Los usuarios están acostumbrados a desplazarse por los sitios web verticalmente en dispositivos móviles y de escritorio, ¡pero no horizontalmente! Por lo tanto, si el usuario se ve obligado a desplazarse horizontalmente o alejarse, para ver toda la página web, la experiencia del usuario es deficiente. Algunas reglas adicionales a seguir:

1. **NO utilices elementos grandes de ancho fijo:** por ejemplo, si una imagen se muestra con un ancho más ancho que la ventana, puede hacer que la ventana se desplace horizontalmente. Recuerde ajustar este contenido para que se ajuste al ancho del viewport.
2. **NO permitas que el contenido dependa de un ancho de ventana en particular para renderizarse bien:** dado que las dimensiones y el ancho de la pantalla en píxeles CSS varían ampliamente entre dispositivos, el contenido no debe depender de un ancho de ventana en particular para renderizarse bien.
3. **Usa media queries CSS para aplicar diferentes estilos para pantallas pequeñas y grandes:** establecer anchos CSS absolutos en elementos grandes hará que éste sea demasiado ancho para el viewport en un dispositivo más pequeño. Usa valores de ancho relativo, como `width: 100%`.

Media Queries

Otra forma en que podemos lidiar con diferentes dispositivos es a través de media queries o consultas CSS para los elementos.

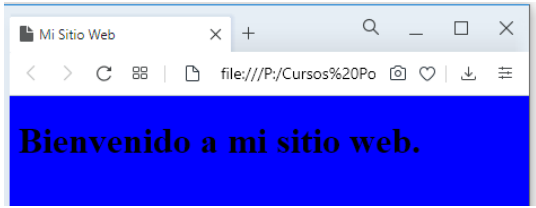
Las media queries son formas de cambiar el estilo de una página en función de cómo se ve la página. Si ésta tiene un tamaño x puedo adaptar un estilo para ese tamaño x.

Podemos tener distintos tipos y características de media queries:

- Media Types: `print, screen, ...`
- Media Features: `height, width, orientation, ...`

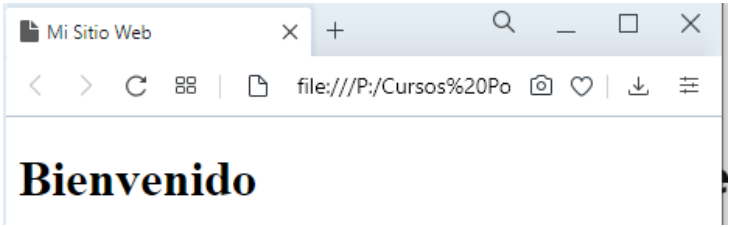
Media Queries (Ej.)

```html
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Mi Sitio Web</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <style>
            @media (min-width: 500px) {
                body {
                    background-color: red;
                }
            }

            @media (max-width: 499px) {
                body {
                    background-color: blue;
                }
            }
        </style>
    </head>
    <body>
        <h1>Bienvenido a mi sitio web.</h1>
    </body>
</html>
```

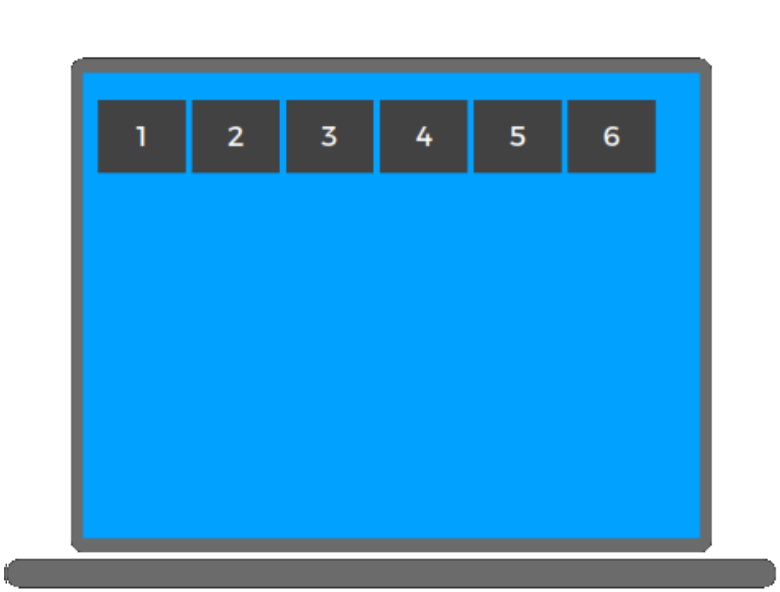


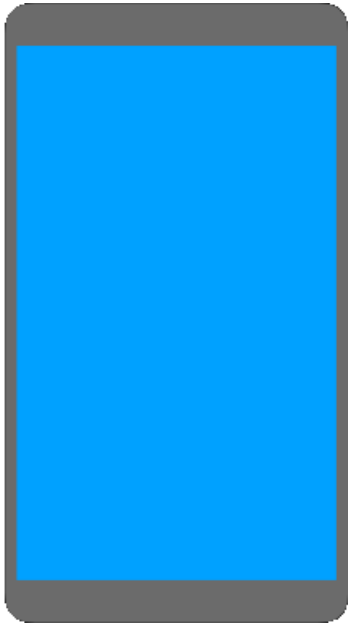
Media Queries (Ej.)

```html
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Mi Sitio Web</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <style>
            @media (min-width: 500px) {
                h1::before {
                    content: "Bienvenido a Mi Sitio Web";
                }
            }

            @media (max-width: 499px) {
                h1::before {
                    content: "Bienvenido";
                }
            }
        </style>
    </head>
    <body>
        <h1></h1>
    </body>
</html>
```

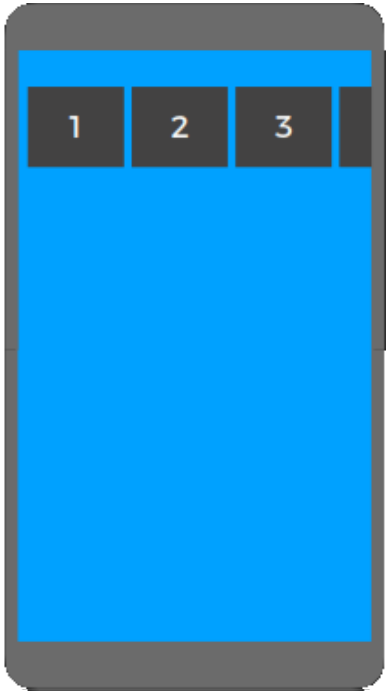


Diseño Responsivo - Flexbox





Diseño Responsivo - Flexbox



Diseño Responsivo - Flexbox

Otra forma de lidiar con los diferentes tamaños de pantalla es usar un nuevo atributo CSS conocido como `flexbox`.

Esto nos permite hacer que los elementos pasen fácilmente a la siguiente línea si no encajan horizontalmente.

Hacemos esto poniendo todos nuestros elementos en un `div` que llamaremos nuestro contenedor. Luego agregamos algo de estilo a ese `div` especificando que queremos usar una pantalla `flexbox` para los elementos dentro de ella. También agregamos un estilo adicional a los `div` internos para ilustrar mejor lo que ocurre aquí.

Diseño Responsivo - Flexbox

```html
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Mi Sitio Web</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <style>
            .container {
                display: flex;
                flex-wrap: wrap;
            }

            .container > div {
                background-color: springgreen;
                font-size: 20px;
                margin: 20px;
                padding: 20px;
                width: 200px;
            }
        </style>
    </head>
    <body>
        <div class="container">
            <div>1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>2. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>3. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>4. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>5. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>6. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>7. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>8. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>9. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasell
            <div>10. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasel
            <div>11. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasel
            <div>12. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasel
        </div>
    </body>
</html>
```

Diseño Responsivo - Grid

Otra forma popular de diseñar una página es usar un `grid` HTML o cuadrícula.

En este `grid`, podemos especificar atributos de estilo como anchos de columna y espacios entre columnas y filas.

Ten en cuenta que cuando especificamos anchos de columna, podemos decir que la última es `auto`, lo que significa que debería llenar el resto de la página.

Diseño Responsivo - Grid

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Mi sitio web</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <style>
            .grid {
                background-color: green;
                display: grid;
                padding: 20px;
                grid-column-gap: 20px;
                grid-row-gap: 10px;
                grid-template-columns: 200px 200px auto;
            }

            .grid-item {
                background-color: white;
                font-size: 20px;
                padding: 20px;
                text-align: center;
            }
        </style>
    </head>
    <body>
        <div class="grid">
            <div class="grid-item">1</div>
            <div class="grid-item">2</div>
            <div class="grid-item">3</div>
            <div class="grid-item">4</div>
            <div class="grid-item">5</div>
            <div class="grid-item">6</div>
        </div>
    </body>
</html>
```

polotic misiones

Bootstrap

Bootstrap

Resulta que hay muchas liberías que otras personas ya han escrito que pueden simplificar aún más el estilo de una página web.

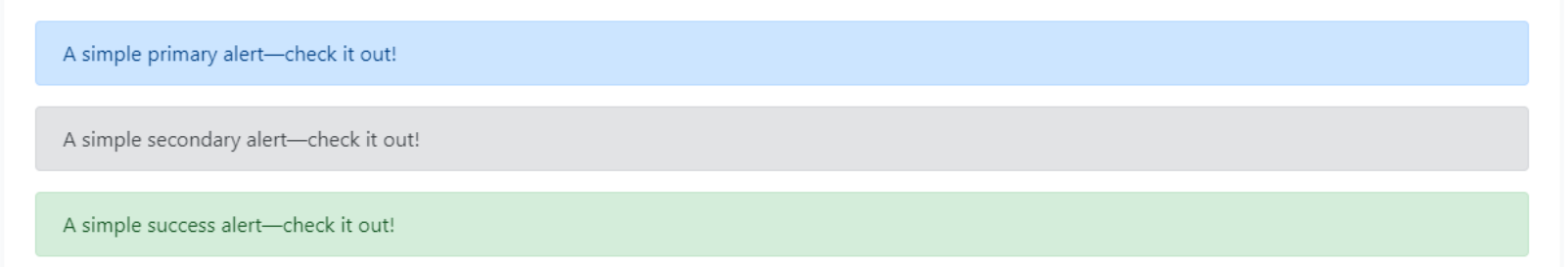
Una librería popular que usaremos a lo largo del curso se conoce como bootstrap.

Podemos incluir bootstrap en nuestro código agregando una sola línea al encabezado de nuestro archivo HTML:

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">
```

Bootstrap

En la documentación de Bootstrap podemos encontrar todas las funcionalidades adicionales que podemos agregar a nuestro sitio web como por ejemplo mensajes de alerta.




```
<div class="alert alert-primary" role="alert">
  Mensaje de alerta azul!
</div>
```

Bootstrap

- Otra característica popular de bootstrap es su sistema de grids.
- Bootstrap divide automáticamente una página en 12 columnas, y podemos decidir cuántas columnas ocupa un elemento agregando la clase `col-x` donde x es un número entre 1 y 12.
- Por ejemplo, en la página siguiente, tenemos una fila de columnas de igual ancho, y luego una fila donde la columna central es más grande:

PARTE 1

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Mi sitio web</title>
        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.mi
n.css" integrity="sha384-
Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">
        <style>
            .row > div {
                padding: 20px;
                background-color: teal;
                border: 2px solid black;
            }
        </style>
    </head>
```

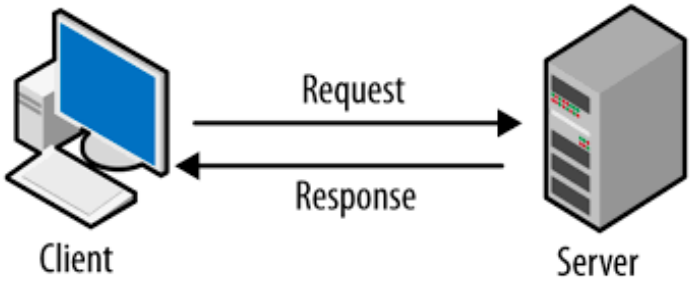
PARTE 2

```
  <body>
        <div class="container">
            <div class="row">
                <div class="col-4">
                    Esta es una seccion.
                </div>
                <div class="col-4">
                    Esta es otra seccion.
                </div>
                <div class="col-4">
                    Esta es una tercera seccion.
                </div>
            </div>
        </div>
        <br/>
        <div class="container">
            <div class="row">
                <div class="col-3">
                     Esta es una seccion.
                </div>
                <div class="col-6">
                     Esta es otra seccion.
                </div>
                <div class="col-3">
                      Esta es una tercera seccion.
                </div>
            </div>
        </div>
    </body>
</html>
```

polotic misiones

¿Dónde interviene JavaScript?

Código del lado del Cliente



En la mayoría de las interacciones online, tenemos una relación entre un cliente/usuario con un servidor mediante un HTTP Request, que envía una respuesta HTTP Response.
Todo el código Python que escribimos hasta ahora corre del lado del servidor mientras que JavaScript nos permite correr código del lado del cliente lo que significa que no se requiere interacción con el servidor mientras está corriendo, permitiendo que los sitios sean mas interactivos.

Como agregar JavaScript al HTML

Para poder añadir código JavaScript en nuestra página web agregamos las etiquetas <script></script> en cualquier parte de nuestra pagina HTML.
Por ejemplo:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Hola</title>
        <script>
            alert('¡Hola mundo!');
        </script>
     </head>
     <body>
            <h1>Hola!</h1>
     </body>
</html>
```

Donde alert es una instrucción que permite colocar un mensaje en pantalla.

JavaScript

polotic misiones

Eventos

Una de las características notables de JavaScript es que nos ayuda a implementar un estilo de programación conocido como Programación Orientada a Eventos.

La programación orientada a eventos es un paradigma que se centra en la detección de eventos y acciones que deben realizarse cuando un evento es detectado.

Un evento puede incluir cualquier cosa, desde el clic de un botón, mover el cursor, una respuesta al tipeo, o la carga de una página.

Prácticamente todo lo que haga el usuario interactuando con la página puede considerarse como evento.
En JavaScript utilizamos Event Listeners que esperan un cierto evento y luego ejecutan código.

JavaScript

Eventos

Comencemos transformando nuestro código JavaScript anterior en una función llamada `hola`:

```
function hola() {
    alert('¡Hola mundo!')
}
```

Ahora hagamos que cada vez que se cliquee un botón se llame a la función `hola`. Para hacer esto creamos un botón `HTML` con un atributo `onclick` que le da las instrucciones necesarias al navegador para saber que hacer cuando el botón es clickeado.

```
<button onclick="hola()">Clic Aquí</button>
```

Eventos

Hay muchos otros eventos que podemos detectar en JavaScript, incluidos los comunes a continuación:

- `onclick`
- `onmouseover`
- `onkeydown`
- `onkeyup`
- `onload`
- `onblur`
- …

JavaScript

Variables

JavaScript es un lenguaje de programación como Python, C o cualquier otro lenguaje con el que haya trabajado antes, lo que significa que tiene muchas de las mismas características que otros lenguajes, incluidas las variables. Hay tres palabras clave que podemos usar para asignar valores en JavaScript:

- `var`: se usa para definir una variable globalmente

```
var edad = 20;
```

- `let`: se utiliza para definir una variable que tiene un alcance limitado al bloque actual, como una función o un bucle

```
let contador = 1;
```

- `const`: se usa para definir un valor que no cambiará

```
const PI = 3.14;
```

JavaScript

Variables

Para ver un ejemplo de cómo podemos usar una variable, echemos un vistazo a una página que realiza un seguimiento de un contador:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Contador</title>
        <script>
            let contador = 0;
            function conteo() {
                contador ++;
                alert(contador);
            }
        </script>
    </head>
    <body>
        <h1>Hola</h1>
        <button onclick="conteo()">Contar</button>
    </body>
</html>
```

JavaScript

querySelector

Además de permitirnos mostrar mensajes a través de alertas, JavaScript también nos permite cambiar elementos en la página. Para hacer esto, primero debemos introducir una función llamada document.querySelector. Esta función busca y devuelve elementos del DOM. Por ejemplo, usaríamos:

```
let cabecera = document.querySelector('h1');
```

para extraer un encabezado. Luego, para manipular el elemento que hemos encontrado recientemente, podemos cambiar su propiedad innerHTML:

```
cabecera.innerHTML = `Adios`;
```

JavaScript

querySelector

Al igual que en Python, también podemos aprovechar las condiciones en JavaScript. Por ejemplo, digamos que en lugar de cambiar siempre nuestro encabezado a ¡Adiós !, deseamos alternar entre ¡Hola! ¡y adiós!.

Nuestra página podría tener un aspecto similar a la siguiente. Tenga en cuenta que en JavaScript, usamos === como una comparación más sólida entre dos elementos que también verifica que los objetos sean del mismo tipo.

Normalmente queremos usar === siempre que sea posible.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Contar</title>
        <script>
            function hola() {
                const cabecera = document.querySelector('h1');
                if (cabecera.innerHTML === 'Hola') {
                  cabecera.innerHTML = 'Adios';
                }
                else {
                  cabecera.innerHTML = 'Hola';
                }
            }
        </script>
    </head>
    <body>
        <h1>Hola</h1>
        <button onclick="hola()">Click Aqui</button>
    </body>
</html>
```

JavaScript

Manipulación del DOM

Usemos esta idea de manipulación DOM para mejorar nuestra página de contador:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Contar</title>
        <script>
            let contador = 0;
            function contar() {
                contador ++;
                document.querySelector('h1').innerHTML = contador;
            }
        </script>
    </head>
    <body>
        <h1>0</h1>
        <button onclick="contar()">Contar</button>
    </body>
</html>
```

JavaScript

Manipulación del DOM

Podemos hacer que esta página sea aún más interesante mostrando una alerta cada vez que el contador llega a un múltiplo de diez.

En esta alerta, queremos formatear un string para personalizar el mensaje, lo que en JavaScript podemos hacer usando template literals. Los template literals requieren que haya comillas invertidas (`) alrededor de toda la expresión y un $ y llaves alrededor de cualquier sustitución. Por ejemplo, cambiemos nuestra función de conteo

```
function conteo() {
    contador++;
    document.querySelector('h1').innerHTML = contador;

    if (counter % 10 === 0) {
        alert(`Conteo es ahora ${contador}`)
    }
}
```

JavaScript

Manipulación del DOM

Ahora, veamos algunas formas en las que podemos mejorar el diseño de esta página. Primero, así como intentamos evitar el estilo en línea con CSS, queremos evitar JavaScript en línea tanto como sea posible.

Podemos hacer esto en nuestro ejemplo de contador agregando una línea de secuencia de comandos que cambia el atributo onclick de un botón en la página y eliminando el atributo onclick dentro de la etiqueta del botón.

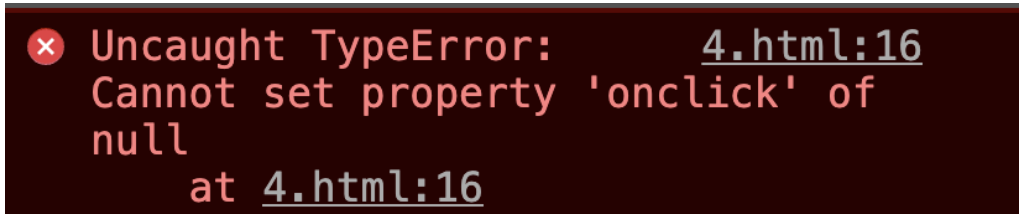
```
document.querySelector('button').onclick = conteo;
```

JavaScript

Manipulación del DOM

Una cosa a tener en cuenta sobre lo que acabamos de hacer es que no llamamos a la función de recuento agregando paréntesis después, sino que simplemente nombramos la función. Esto especifica que solo deseamos llamar a esta función cuando se hace clic en el botón. Esto funciona porque, como Python, JavaScript admite la programación funcional, por lo que las funciones pueden tratarse como valores en sí mismas.

Sin embargo, el cambio anterior por sí solo no es suficiente, como podemos ver al inspeccionar la página y mirar la consola de nuestro navegador:



Manipulación del DOM

Este error surgió porque cuando JavaScript buscó un elemento usando document.querySelector ('botón'), no encontró nada. Esto se debe a que la página tarda un poco en cargarse y nuestro código JavaScript se ejecutó antes de que se procesara el botón. Para dar cuenta de esto, podemos especificar que el código se ejecutará solo después de que la página se haya cargado usando la función addEventListener. Esta función toma dos argumentos:

- Un evento para escuchar (por ejemplo: `'clic'`)
- Una función para ejecutar cuando se detecta el evento (por ejemplo: `hola`)

Podemos usar la función para ejecutar el código solo una vez que se haya cargado todo el contenido:

```
document.addEventListener('DOMContentLoaded', function() {
    // Algo de código aquí
});
```

JavaScript

Manipulación del DOM

En el ejemplo anterior, usamos una función anónima, que es una función a la que nunca se le da un nombre. Poniendo todo esto junto, nuestro JavaScript ahora se ve así:

```
let contador = 0;

function conteo() {
    contador ++;
    document.querySelector('h1').innerHTML = contador;

    if (contador % 10 === 0) {
        alert(`Contador ahora es ${contador}`)
    }
}

document.addEventListener('DOMContentLoaded', function() {
    document.querySelector('button').onclick = conteo;
});
```

JavaScript

Archivos Externos

Otra forma en que podemos mejorar nuestro diseño es moviendo nuestro `JavaScript` a un archivo separado.
La forma en que hacemos esto es muy similar a cómo colocamos nuestro `CSS` en un archivo separado para diseñar:

1. Escribes todo el código JavaScript en un archivo separado que termine en `.js`, por ejemplo: `contador.js`.
2. Agregas un atributo `src` a la etiqueta `<script>` que apunta a este nuevo archivo.

Para nuestra página de contador, podríamos tener un archivo llamado `contador.html` que se ve así con su correspondiente archivo js:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Contador</title>
        <script src="contador.js"></script>
    </head>
    <body>
        <h1>0</h1>
        <button>Contador</button>
    </body>
</html>
```

```
let contador = 0;

function conteo() {
    contador++;
    document.querySelector('h1').innerHTML = contador;

    if (contador % 10 === 0) {
        alert(`Contador ahora es ${contador}`)
    }
}

document.addEventListener('DOMContentLoaded', function() {
    document.querySelector('button').onclick = conteo;
});
```

JavaScript

Archivos Externos

Tener JavaScript en un archivo separado es útil por varias razones:

- **Atractivo visual**: nuestros archivos HTML y JavaScript individuales se vuelven más legibles.
- **Acceso entre archivos HTML:** ahora podemos tener varios archivos HTML que comparten el mismo JavaScript.
- **Colaboración:** ahora podemos hacer que una persona trabaje fácilmente en JavaScript mientras que otra trabaja en HTML.
- **Importación:** podemos importar bibliotecas JavaScript que otras personas ya han escrito. Por ejemplo, Bootstrap tiene su propia biblioteca de JavaScript que puede incluir para que su sitio sea más interactivo.

JavaScript

Comencemos con otro ejemplo de una página que puede ser un poco más interactiva. A continuación, crearemos una página en la que un usuario puede escribir su nombre para recibir un saludo personalizado.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Hola</title>
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            document.querySelector('form').onsubmit = function() {
                const nombre = document.querySelector('#nombre').value;
                alert(`Hola, ${nombre}`);
            };
        });
    </script>
</head>
<body>
    <form>
        <input autofocus id="nombre" placeholder="Nombre" type="text">
        <input type="submit">
    </form>
</body>
</html>
```

Cambiar el CSS

Podemos hacer más que simplemente agregar HTML a nuestra página usando JavaScript, ¡también podemos cambiar el estilo de una página! En la página siguiente, usamos botones para cambiar el color de nuestro encabezado.

```
<!DOCTYPE html>
<html lang="es">
<head>
     <title>Colores</title>
     <script>
         document.addEventListener('DOMContentLoaded', function() {
            document.querySelectorAll('button').forEach(function(button) {
                button.onclick = function() {
                    document.querySelector("#hola").style.color = button.dataset.color;
                }
            });
         });
     </script>
</head>
<body>
    <h1 id="hola">Hola</h1>
    <button data-color="red">Rojo</button>
    <button data-color="blue">Azul</button>
    <button data-color="green">Verde</button>
</body>
</html>
```

La Consola

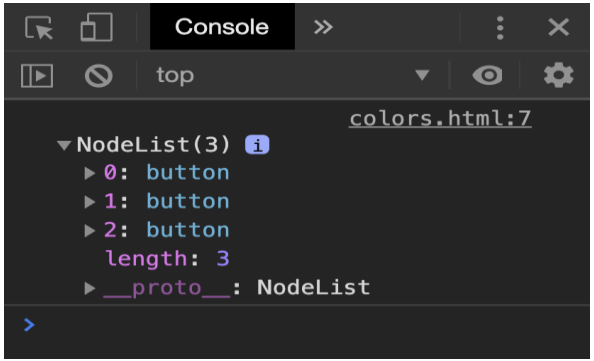
La consola es una herramienta útil para probar pequeños fragmentos de código y depurar.

Puedes escribir y ejecutar código JavaScript en la consola, que se puede encontrar inspeccionando el elemento en su navegador web y luego haciendo clic en consola. (El proceso exacto puede cambiar de un navegador a otro).

Una herramienta útil para depurar es imprimir en la consola, lo que puede hacer usando la función `console.log`.

Por ejemplo, en la página `colores.html` anterior, puedo agregar la siguiente línea:

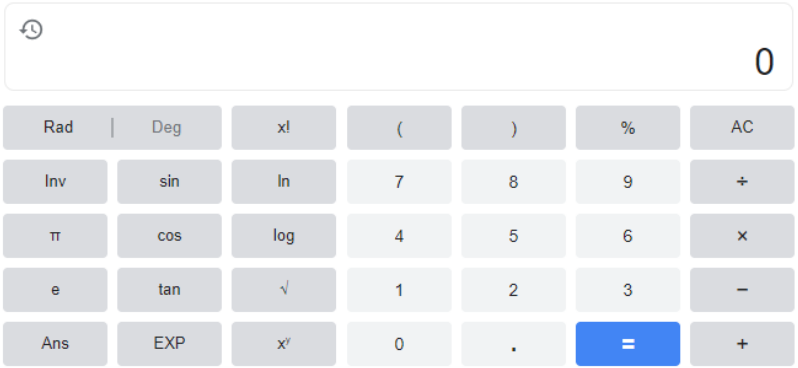
```
console.log(document.querySelectorAll('button'));
```



Trabajemos Juntos

Llegó la hora del desafío:

- Intenta construir una calculadora (que incluya las cuatro operaciones básicas, raíces, porcentajes y trigonometría) con diseño responsivo y JavaScript intentando replicar el modelo de Google:



¡HASTA LA *próxima!*

www.polotic.misiones.gob.ar

/poloticmisiones

polotic misiones

Gobierno de Misiones