

DIPLOMADO

Front End

Introducción
Desarrollo de
Componentes Web Nativos

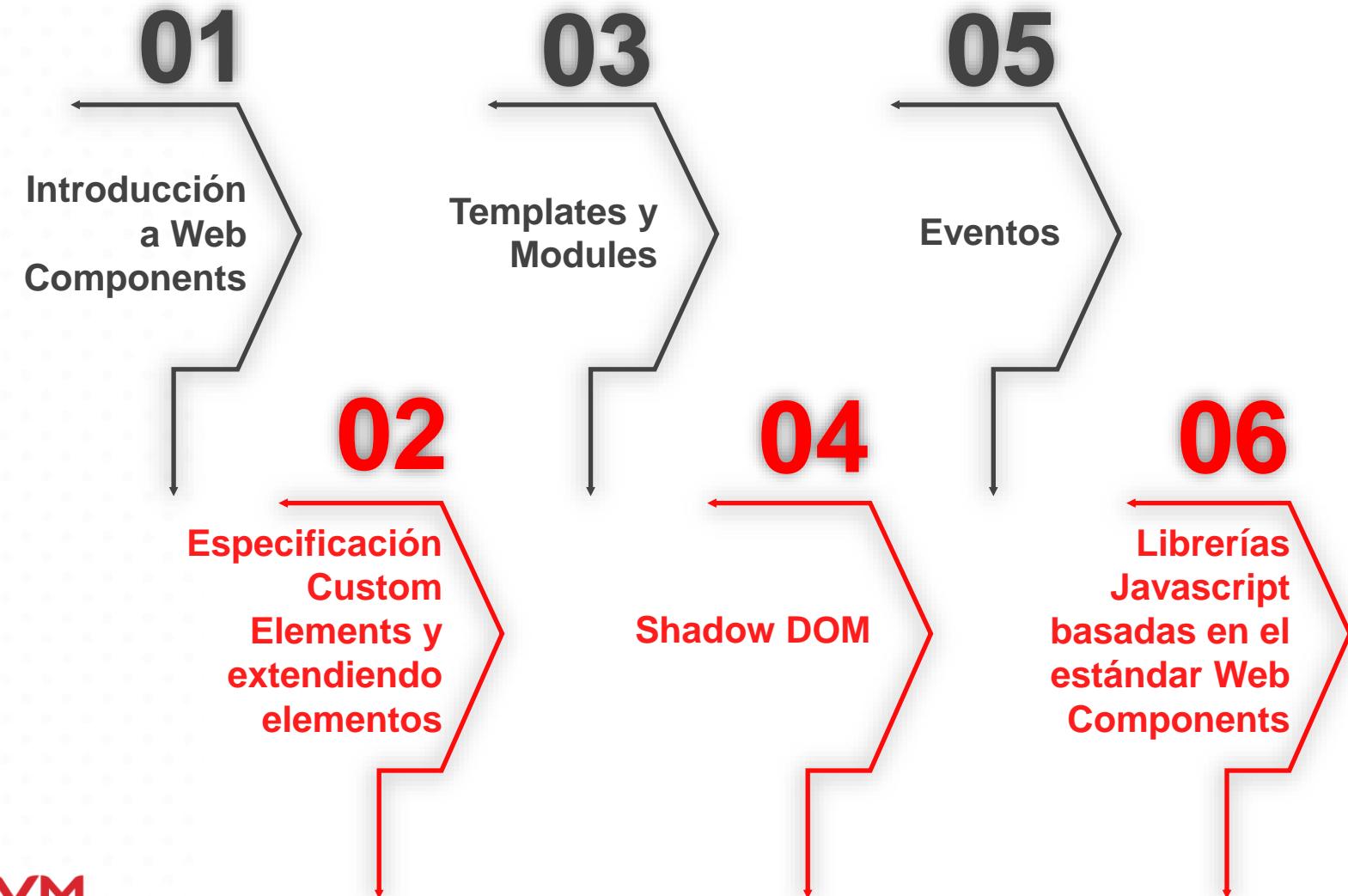
UVM

Objetivo

Aprender qué es un componente web, así como qué son el soporte del navegador web, los sistemas de diseño y el desarrollo impulsado por componentes (CDD).

Conocer diversas herramientas, tecnologías, diseños y conceptos de desarrollo que necesitas para crear tu primer componente web.

Temario



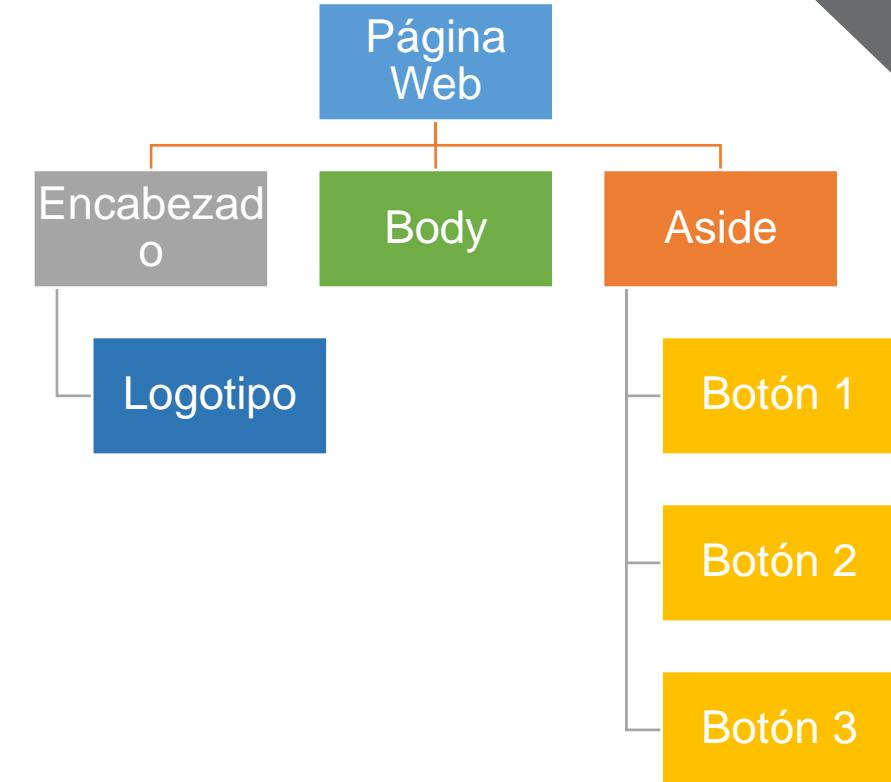
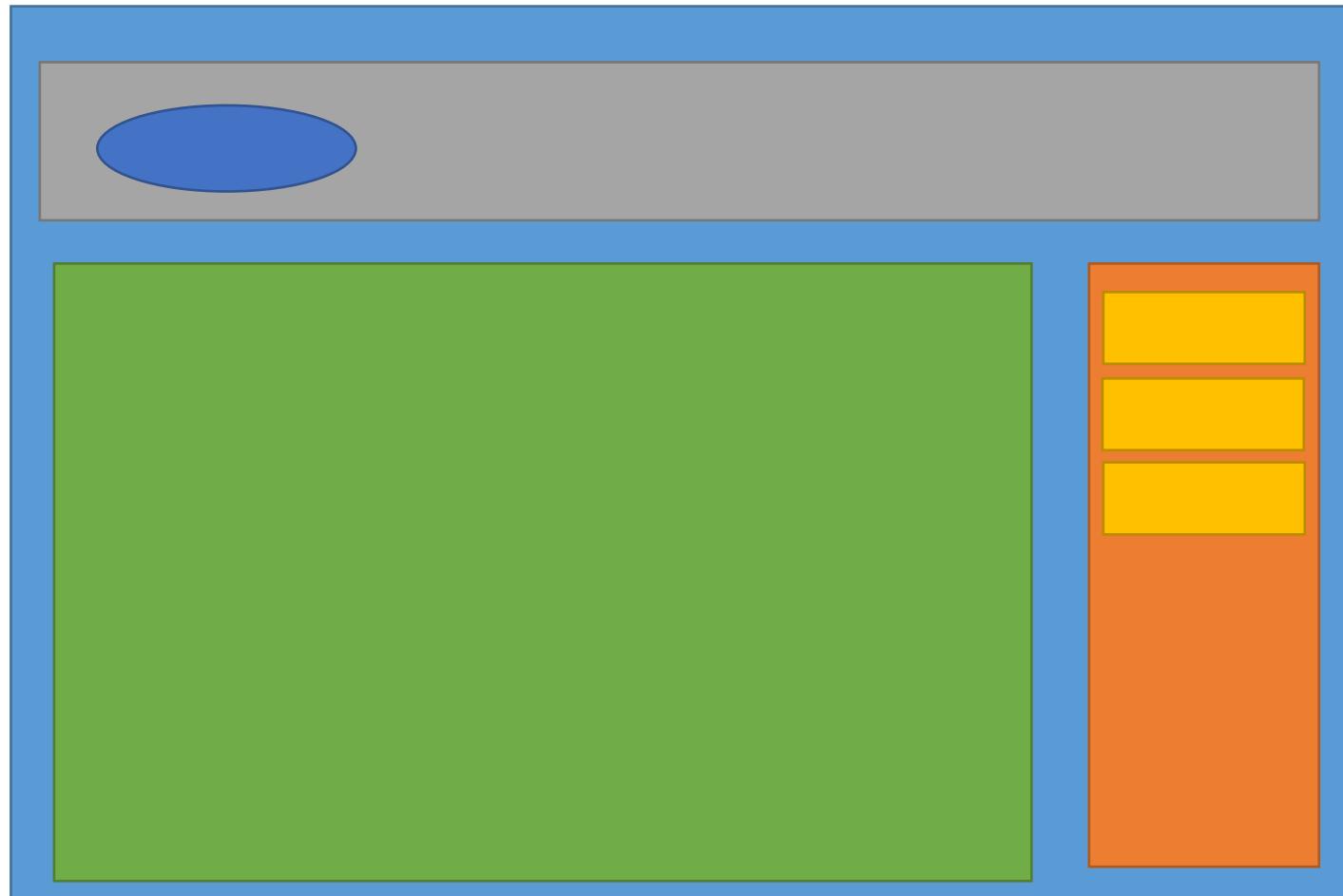
DIPLOMADO

Front End

TEMA 1

Introducción a
Web Components

¿Qué son los Web Components?



¿Qué son los Web Components?



Los Web Components pueden definirse como un conjunto de tecnologías para desarrollar interfaces y funcionalidades en el lado del cliente utilizando para ello las tecnologías nativas que soporta cualquier navegador. Esto es: HTML, CSS y JavaScript.

Si bien la idea de Web Components (o Componentes Web) nació hace ya unos años (2012), su avance a sido paulatino pero muy aprovechado por algunas interfaces gráficas.

Afortunadamente, las empresas tecnológicas se están sumando a desarrollar Web Components para conseguir ganar tiempo de desarrollo y, por qué no, ganar más dinero gracias a la facilidad de implementación que este conjunto de tecnologías tiene.

¿Qué son los Web Components?

El modelo de componentes web que se publicó en 2012 establece principalmente cuatro especificaciones referidas a la creación de estos prácticos elementos HTML. Estas son las siguientes:

- **Custom elements:** conjuntos de API de JavaScript para definir elementos personalizados por el usuario.
- **Shadow DOM:** conjunto de API de JavaScript para añadir elementos DOM.
- **ES Modules:** módulos para integrar y reutilizar documentos de JavaScript.
- **HTML templates:** plantillas HTML que no se muestran en la página web final y que pueden servir de base para ciertos elementos definidos por el usuario.

Todos los navegadores convencionales ya son compatibles con los componentes web estándares. Para trabajar con los códigos HTML encapsulados, pueden usarse todos los frameworks o bibliotecas de JavaScript que trabajan con HTML.

¿Por qué utilizar Web Components?

La arquitectura basada en componentes se ha demostrado las más adecuada para el desarrollo de aplicaciones frontend.

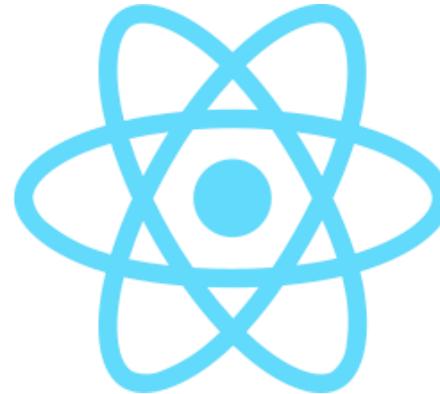
- **Encapsulación.**
 - La complejidad del componente se queda dentro del componente mismo.
- **Reusabilidad.**
 - Bajo o nulo acoplamiento, se pueden utilizar varias veces.
- **Escalabilidad.**
 - Se pueden ir desarrollando componentes de forma acotadas pero que en conjunto pueden ir creciendo la funcionalidad en los proyectos.
- **Portabilidad.**
 - Se puede utilizar en aplicaciones web o móviles.
- **Mantenibilidad.**
 - Es más fácil de hacer el mantenimiento de las aplicaciones y más comprensible.

¿Por qué utilizar Web Components?

Algunas **bibliotecas y frameworks**, como Angular o [jQuery](#), son desde hace años herramientas de trabajo esenciales para los programadores web. Si bien estas estructuras básicas de codificación son prácticas, versátiles y ahorran mucho trabajo en el desarrollo de proyectos, resultan con frecuencia inflexibles a la hora de usarlas **en más de un proyecto**. Así, no es raro para un desarrollador tener que volver a escribir un código cada vez que hay un cambio de framework, por ejemplo. Para tratar de solucionar este problema, el Consorcio W3C introdujo los web components, creando así un marco universal para poder reutilizar códigos HTML, CSS y de JavaScript de una forma simple y común a los tres lenguajes.

Puesto que los elementos web universales se caracterizan por una **sintaxis sencilla y fácil de aprender**, los programadores con menos experiencia también se benefician de este estándar del W3C. Desde hace algunos años, Google trabaja en el llamado [Polymer Project](#) para desarrollar **bibliotecas y plantillas de programación de componentes web** para ponerlas a disposición del público general.

Frameworks o Librerías

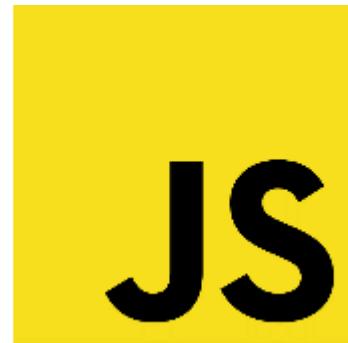


Estándar Javascript



←
Desarrollo de componentes

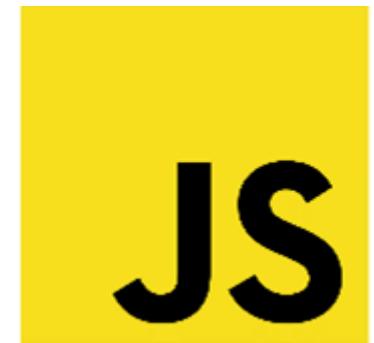
<calendario-eventos>
<botón-aceptar>
<botón-cancelar>



¿Para qué utilizar el Estándar Javascript?



- Evitas cargar librerías pesadas en el navegador
 - Ahorras tiempo de procesamiento
 - Ahorras transferencia
 - Ahorras batería
- Más relacionado al estándar
 - Tu código durará más
 - Funcionará en más plataformas
 - Lo podrás usar en cualquier framework



¿Dónde usar Web Components?

Aplicaciones web

- Sitios estáticos
- PHP
- Frameworks
 - Laravel
 - Angular
 - Vue
- CMS
 - WordPress
- SPA/PWA



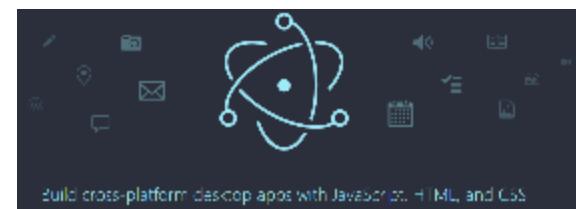
Aplicaciones móviles

- Ionic
- Cordova
- MeteorJS



Aplicaciones escritorio

- ElectronJS
 - Windows
 - Linux
 - Mac



Browser support

 CHROME OPERA SAFARI FIREFOX EDGE HTML TEMPLATES STABLE STABLE STABLE STABLE STABLE CUSTOM ELEMENTS STABLE STABLE STABLE STABLE STABLE SHADOW DOM STABLE STABLE STABLE STABLE STABLE ES MODULES STABLE STABLE STABLE STABLE STABLE

¿Depende del Tipo de Proyecto?

Existen una mutitud de herramientas frontend útiles para la etapa de desarrollo y llevar un sitio a producción.

- Webpack
- Snowpack
- Rollup
- Lo que tu framework te proporcione...

Entorno de desarrollo

Editor de Código

- Visual Studio Code

Entorno de Ejecución de JavaScript para escritorio

- Node.js

Servidor de páginas en local

- HTTP-Server

DIPLOMADO

Front End

```
<head>
  <div style="background-image:url('/pix/samples/bg1.gif');background . text- todos;
  <title>Fixed Width 2 Blue</title>
  <style type="text/css">
    <div style="background-image:url('/pix/samples/bg1.gif');background . text- todos;
    height : text - 1200px;><p>The image can be tiled across the background,
    while the text runs across the top.</p> </div>
    /* Logo */
    <body style="background-color:yellowgreen;color:white;">
      <div id="todolistid" = data.todolistid>
        <html>
          <header { background:#eee; }
            <header-inner { margin: auto; padding:10px; width:970px;background:#fff; }
            <div style="background-image:url('/pix/samples/bgl.gif');background . text- todos;
            height : text - 200px;><p>The image can be tiled across the background,
            while the text runs across the top.</p> </div>
            /* Feature */
            <div style="background-image:url('/pix/samples/bgl.gif');background . text- todos;
            height : text - 200px;><p>The image can be tiled across the background,
            while the text runs across the top.</p> </div>
            /* Menu */
            <p>You can make <span style="font-style:italic">some</span> the HTML 'span' tag.
            <p>You can bold <span style="font-weight:bold">parts</span> of your text using the HTML tag.</p>
            #top-nav ul li { margin: 0 0px; float:left; }
            #top-nav ul li a { display:block; margin:0; padding:0px 20px; color:blue; text-decoration:none; }
            <body style="background-color:yellowgreen;color:white;"> { color:#d3d3ff; }
            /* Content */
            <p>You can make <span style="font-style:italic">some</span> of your text using the HTML 'span' tag.
            <p>You bold <span style="font-weight:bold">parts</span> of your text using the HTML tag.</p>
            <div style="background-image:url('/pix/samples/bgl.gif');background . text- todos;
            height : text - 200px;><p>The image can be tiled across the background,
            while the text runs across the top.</p> </div>
            <body style="background-color:yellowgreen;color:white;">
              #content #sidebar .widget dl { margin:0; padding:0; list-style:none; color:#959595; }
              #content #sidebar .widget dl li { margin:0; padding:0px 16px; }
              #content #sidebar .widget dl li a { color:blue; text-decoration:none; margin-left:-16px; padding:4px 0px #px 16px; }
              while the text runs across the top.</p> </div>
            /* Footer */
            #footer { background:#eee; }
            #footer-inner { margin: auto; text-align:center; padding:12px; width:970px; }
            <html> <p style="font-weight:bold;">
            >HTML font code is done using CSS</p>
            >todolistid = data.todolistid
          </html>
        </div>
      </body>
    </html>
  </style>
</head>
```

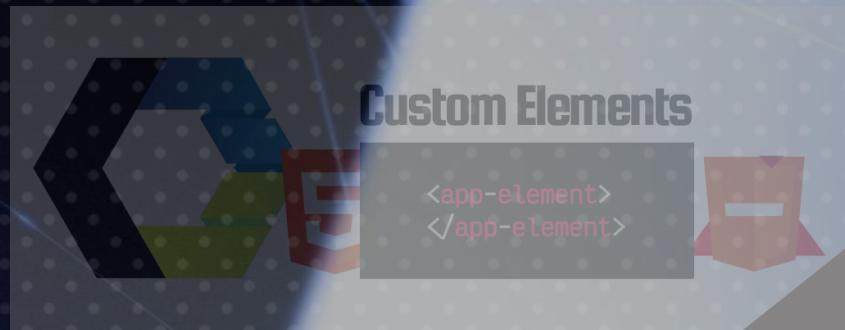


TEMA 1

Video
de Instalación del ambiente

DIPLOMADO

Front End



UVM

TEMA 2

Especificación
Custom Elements
y Extendiendo Elementos

Estructura Básica de HTML

Veamos el siguiente video para preparar la estructura de directorios que estaremos utilizando en el módulo, y generaremos el primer archivo con el cual haremos los ejemplos de los Custom Elements.

DIPLOMADO

Front End

UVM

TEMA 2

Ejercicio

Preparación de Directorios

Custom Elements

Los custom elements (elementos definidos por el usuario) son etiquetas HTML que encapsulan contenido HTML, incluyendo directrices CSS y scripts. Se alistan en el CustomElementRegistry y sus características más importantes son las siguientes:

- Extienden de los elementos HTML.
- Se definen con JavaScript y se crean al ejecutar el código.
- Son personalizables.
- Pueden incluir estilos propios.
- Acaban con una etiqueta de cierre.
- Su nombre es una secuencia DOM y siempre contiene un guion.
- Su nombre solo puede aparecer una vez en el CustomElementRegistry.



Custom Elements

La especificación de "Custom Elements", que quizás sea la más representativa de las especificaciones de los Web Components, porque es la que nos permite hacer nuevos elementos del HTML, que realizan funcionalidades personalizadas o presentan información de una nueva manera.

Además, estos Custom Elements los puedes usar directamente en tu página, sin necesidad de programación. Para ser correctos el desarrollador que crea el custom element generalmente sí necesitará realizar tareas de programación, aunque todos aquellos que lo usen, lo harán mediante la expresión de una etiqueta, de manera declarativa.

Primer Ejemplo de Custom Elements

Antes de empezar con los ejemplos estudiemos la estructura de los custom elements a partir de una clase.

Objetivo:

Crear una clase que despliegue un cuadro de alerta cuando se de clic en una etiqueta atomizada creado como Custom Element.

Primer Ejemplo de Custom Elements

```
<script>
  class CustomButton extends HTMLElement {
    constructor() {
      super();
      this.addEventListener('click', function () {
        alert('Botón Pulsado');
      })
    }
  }
</script>
```

Crear la clase
CustomButton y
extiende de la
interface
HTMLElement

Primer Ejemplo de Custom Elements

```
<script>
  class CustomButton extends HTMLElement {
    constructor() {
      super();
      this.addEventListener('click', function () {
        alert('Botón Pulsado');
      })
    }
  }
</script>
```

Crear la clase CustomButton y extiende de la interface HTMLElement

Esto es una clase de tipo interface que representa un elemento genérico de HTML

Primer Ejemplo de Custom Elements

Luego se define el constructor

```
<script>
  class CustomButton extends HTMLElement {
    constructor() {
      super();
      this.addEventListener('click', function () {
        alert('Botón Pulsado');
      })
    }
  }
</script>
```

Primer Ejemplo de Custom Elements

```
<script>
  class CustomButton extends HTMLElement {
    constructor() {
      super();
      this.addEventListener('click', function () {
        alert('Botón Pulsado');
      })
    }
  }
</script>
```

Tenemos que utilizar la palabra reservada `super` para hacer referencia al constructor de la clase padre

Primer Ejemplo de Custom Elements

```
<script>
  class CustomButton extends HTMLElement {
    constructor() {
      super();
      this.addEventListener('click', function () {
        alert('Botón Pulsado');
      })
    }
  }
</script>
```

Se define un listener del evento 'click' para que abra un 'alert' al dar clic en la etiqueta atomizada que crearemos

Primer Ejemplo de Custom Elements

```
<script>
  class CustomButton extends HTMLElement {
    constructor() {
      super();
      this.addEventListener('click', function () {
        alert('Botón Pulsado');
      })
    }
  }
  window.customElements.define('custom-button', CustomButton);
</script>
```

Se define dentro de window una etiqueta que define a 'custom-button' y hace referencia a la clase CustomButton

Primer Ejemplo de Custom Elements

```
<custom-button>Pulsar</custom-button>
<script>
  class CustomButton extends HTMLElement {
    constructor() {
      super();
      this.addEventListener('click', function () {
        alert('Botón Pulsado');
      })
    }
  }
</script>
window.customElements.define('custom-button', CustomButton);
```

Se crea una
etiqueta con el
custom element
antes del script

Primer Ejemplo de Custom Elements

Para crear un custom element de una etiqueta en específico se necesita el método define. El siguiente componente web muestra un custom element mediante el cual se puede integrar a la etiqueta button:

```
customElements.define('my-button', MyButton, { extends: 'button' });
```

Para poder usar ahora este elemento en una aplicación web, basta con el siguiente código:

```
<my-button></my-button>
```

DIPLOMADO

Front End

UVM

TEMA 2

Ejercicio

Custom Elements

DIPLOMADO

Front End



UVM

TEMA 3

Templates y Modules

Templates

Las llamadas HTML templates son plantillas de archivos HTML. Los elementos que contienen se mantienen inactivos y sin renderizar (sin ser representados gráficamente) hasta que sean solicitados explícitamente. Esta característica hace que no perjudiquen el tiempo de carga de las páginas web. Son, por lo tanto, una buena alternativa a los métodos de JavaScript tradicionales.

Con la etiqueta `<template>` se define una plantilla HTML. En el siguiente ejemplo, la plantilla creada lleva el nombre my element.

```
<template id="my-element">
<p>My element</p>
</template>
```

Para utilizar la plantilla en una página web, actívala con los métodos de JavaScript `getElementById` y `content` e intégrala en el DOM.

```
let template = document.getElementById('my-element');
let templateContent = template.content;
document.body.appendChild(templateContent);
```

DIPLOMADO

Front End

TEMA 3

Ejercicio

Template

DIPLOMADO

Front End

TEMA 3

Ejercicio

Segundo ejercicio de Templates

Integración de un template dentro de un archivo con custom elements

DIPLOMADO

Front End

TEMA 3

Ejercicio

Utilizando Custom Elements y Template

ES Modules

Los ES modules son módulos que exportan objetos, funciones o variables desde un archivo de JavaScript. Al hacerlo, pueden dividir variables en grupos dentro de un mismo archivo y referenciarlas. Actualmente, existen dos sistemas de ES modules. Mientras que CommonJS originalmente pertenecía a Node.js, el sistema ya está incluido en JavaScript ES6.

Para exportar una función desde una biblioteca de JavaScript, utiliza el método `export`. En el siguiente ejemplo, se exporta una función que devuelve un input string dos veces.

```
// ? lib.bib1
export const repeat = (string) => `${string} ${string}`;
}
```

Con `import` se puede volver a solicitar la función exportada tantas veces como se quiera.

```
main.mjs
import {repeat} from './lib.mein';
repeat ('Buenos días');
// → 'Buenos días Buenos días'
```

ES Modules

```
var MyModule = (function() {
    var privateProperty = 'Esta es una propiedad privada';

    var privateMethod = function() {
        return 42;
    };

    return {
        publicProperty: 'Esta es una propiedad pública',
        publicMethod: function() {
            return privateProperty + ' ' + privateMethod();
        }
    };
})();

// Use example
var log = console.log;

log(MyModule.publicProperty); //=> 'this is a public property'
log(MyModule.publicMethod()); //=> 'this is a private property 42'
log(MyModule.privateProperty); //=> undefined
log(MyModule.privateMethod()); //=> Uncaught TypeError: MyModule.privateMethod is not a function
```

ES Modules

Este enfoque, aunque permite definir una API pública y agrupar una serie de funcionalidad dentro de un único namespace, tiene una serie de carencias:

- Todos los módulos están referidos por una variable almacenada en el scope global. Si no elegimos correctamente los nombres para cada módulo, podremos sobrescribirlos sin darnos cuenta (el intérprete de JavaScript lo entenderá como una nueva asignación).
- No existe un mecanismo para indicar qué dependencias tiene nuestro módulo. Si, por ejemplo, el módulo anterior tuviese una dependencia con jQuery, tendríamos que ser nosotros quienes nos aseguremos que jQuery esté correctamente cargado antes de ejecutar nuestro código.
- No permite una importación granular. Con este enfoque, siempre tendremos que trabajar con todos los valores exportados por el módulo independientemente de que los necesitemos o no.

ES Modules

Esta ausencia de soporte por parte de la especificación del lenguaje hizo que la propia Comunidad tuviese que crear mecanismos para obtener esta funcionalidad por vías adicionales. Con la llegada de Node al ecosistema de JavaScript y su sistema de módulos basado en CommonJS aparecieron herramientas como Browserify y sistemas de módulos alternativos como AMD (vía RequireJS).

*Leer el artículo

DIPLOMADO

Front End

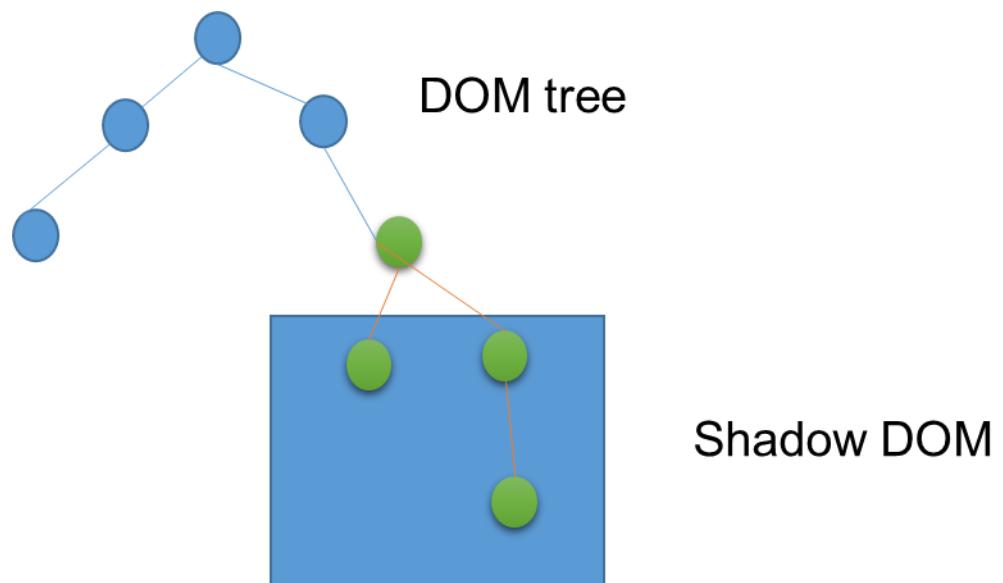
UVM

TEMA 4

Shadow DOM

Shadow DOM

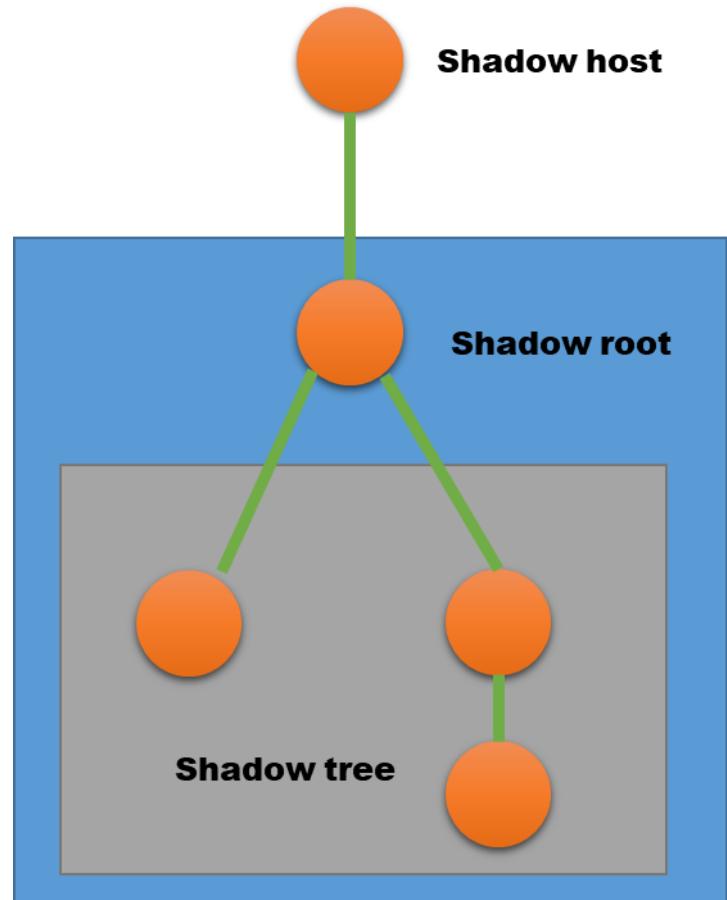
La principal característica de los componentes web es su capacidad para encapsular elementos HTML. La API Shadow DOM facilita esta tarea porque permite añadir árboles DOM ocultos a un árbol de documentos, de forma que solo sea visible la etiqueta HTML del Shadow DOM. De esta forma, pueden añadirse elementos HTML al DOM oculto sin tener que alterar cada vez también el DOM principal.



Shadow DOM

Partes diferenciales.

- **Shadow host.**
 - El nodo regular del DOM al que es atado el Shadow DOM
- **Shadow tree.**
 - El árbol DOM dentro del Shadow DOM
- **Shadow root.**
 - El nodo raíz del árbol Shadow



Cómo utilizar Shadow DOM

```
let shadow = elementRef.attachShadow({mode: 'open'});
```

- **mode: open** significa que puede acceder al shadow DOM usando JavaScript en el contexto principal de la página. Por ejemplo, usando la propiedad Element.shadowRoot:

```
let myShadowDom = elementRef.shadowRoot;
```

Cómo utilizar Shadow DOM

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Elemento con Shadow DOM</title>
</head>
<body>
    <h1>Shadow DOM en un Custom Element</h1>
    <h2>Esto es un titular H2</h2>
    <!-- Cargo mi Web Component -->
    <dw-hishadow></dw-hishadow>

    <script>
        //prototipo de elemento, a partir del elemento base
        var prototipo = Object.create(HTMLElement.prototype);

        // manejador del evento createdCallback,
        //ocurre cuando se instancia el componente
        prototipo.createdCallback = function() {
            var shadowDOM = this.createShadowRoot();
            shadowDOM.innerHTML = '<h2>Titular H2 en Shadow DOM</h2>';
        };

        //Registro el componente "dw-hishadow"
        document.registerElement('dw-hishadow', {
            prototype: prototipo
        });
    </script>
</body>
</html>
```

DIPLOMADO

Front End



TEMA 4

Ejercicio

Implementando Shadow DOM

En el siguiente ejercicio vamos a realizar un componente nativo de un botón con animación, que además es capaz de atender a diversos estados. Básicamente es un botón que, cuando se pulsa, crea una pequeña animación y que además tiene un atributo llamado "status" que permite actualizar el aspecto del botón, de modo que de un poco de feedback visual al usuario.

Podemos aprovechar una de las herramientas más útiles de ES6 como son los [template strings](#) para la creación del template del componente. Esto nos permite interpolar variables o propiedades del componente de una manera muy sencilla, que además produce un código muy legible.

Para facilitar la utilización del template dentro del componente nos apoyaremos en un [método getter](#) de Javascript, lo que me permitirá usar este template dentro de la clase del componente, tal como se usaría una propiedad común.

```
get template() {
  return `
    <style>
      div {
        display: inline-block;
        color: #fff;
        border-radius: 3px;
        padding: 10px;
        cursor:pointer;
        outline:none;
        animation-duration: 0.3s;
        animation-timing-function: ease-in;
        background-color: #000;
      }
      div:active{
        animation-name: anim;
      }
      @keyframes anim {
        0% {transform: scale(1);}
        10%, 40% {transform: scale(0.7) rotate(-1.5deg);}
        100% {transform: scale(1) rotate(0);}
      }
      .neutral {
        background-color: #888;
      }
      .danger {
        background-color: #d66;
      }
      .success {
        background-color: #3a6;
      }
    </style>
    <div class="${this.status}"><slot></slot></div>
  `;
}
```

La parte más interesante del template lo tenemos en la línea siguiente:

```
<div class="${this.status}"><slot></slot></div>
```

Aquí está interesante apreciar como se ha embeddo el valor de la propiedad status del componente. Esta propiedad pertenece al objeto botón, una vez instanciado. Además en breve veremos cómo poblar esa propiedad con el valor introducido en el atributo "status", indicado al usar el componente.

Además, muy interesante también es la etiqueta SLOT, que sirve para colocar en este punto el contenido que tenga la etiqueta del componente.

Por ejemplo, al usar el componente podemos tener algo como esto:

```
<boton-status>Haz clic aquí</boton-status>
```

El texto "Haz clic aquí" será lo que se introduzca dentro del template gracias a la etiqueta SLOT.

DIPLOMADO

Front End

UVM

TEMA 4

Ejercicio

Custom Elements y Shadow Dom

DIPLOMADO

Front End



UVM

TEMA 5

Eventos

Import HTML

Importa un archivo html en otro archivo html

Se realiza mediante la etiqueta link

Se accede mediante el protocolo http

Mediante los import html podemos distribuir los web components y se debe colocar el la sección de head del documento donde se va a utilizar el web component

```
<link rel="import" href="./mi-componente.html">
```

Import HTML

Importa un archivo html en otro archivo html

Se realiza mediante la etiqueta link

Se accede mediante el protocolo http

Mediante los import html podemos distribuir los web components y se debe colocar el la sección de head del documento donde se va a utilizar el web component

<link rel="import" href="./mi-componente.html">

DEPRECATED

Import JavaScript

Importa un archivo JavaScript desde un html

Se realiza mediante la etiqueta script

Sólo podemos importar código JavaScript

Los templates se cargan utilizando AJAX o literales

DIPLOMADO

Front End



TEMA 5

Ejercicio

Imports en Javascript

Atributos

- Propiedades que pueden definirse de forma declarativa desde HTML
- Sus valores son de tipo String
- Pueden conectar al ciclo de vida del componente

```
let titulo = element.getAttribute('titulo');
```

```
element.setAttribute('titulo','Hola');
```

Propiedades

- Propiedades definidas de forma programática
- Su valor puede ser de cualquier tipo
- Se utilizan set y get para acceder, modificar y conectar con el ciclo de vida del componente.

```
set titulo(titulo) {  
    this._titulo = titulo;  
}
```

```
get titulo() {  
    return this._titulo;  
}
```

DIPLOMADO

Front End

UVM

TEMA 5

Ejercicio

Atributos y Propiedades

Ciclo de Vida

Es un conjunto de estados en el que se puede encontrar dicho componente desde que se crea hasta que se destruye.

Bloques:



Ciclo de Vida

Creación

- Constructor
 - Lanzado cuando el componente se crea
- connectedCallback
 - Lanzado cuando el componente se adjunta al DOM

Ciclo de Vida

Actualización

- observedAttributes
 - Definir atributos que serán observados
- attributeChangeCallbak
 - Define la funcionalidad que se ejecuta cuando alguno de los atributos observados cambia su valor

Ciclo de Vida

Destrucción

- disconnectedCallback
 - Se lanza cuando el componente se elimina del DOM

DIPLOMADO

Front End



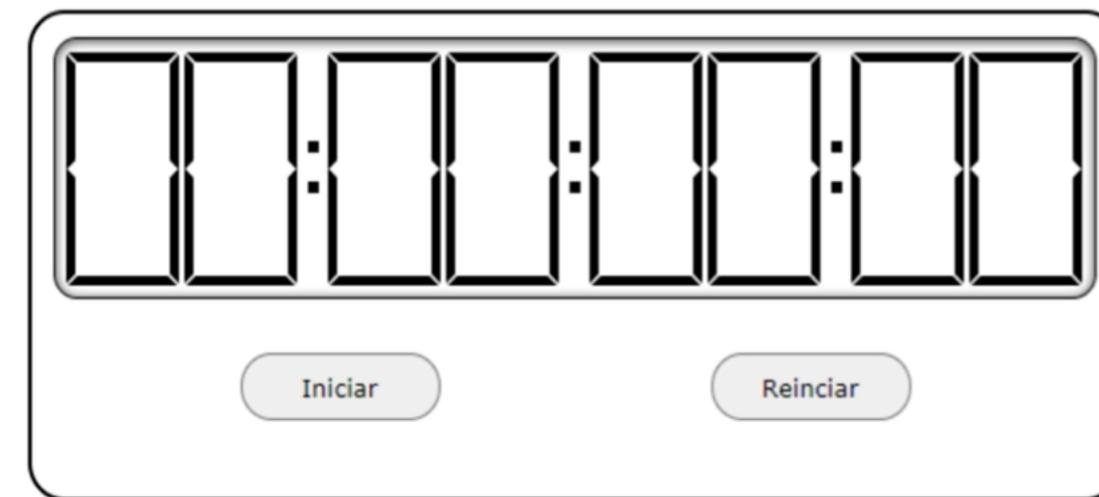
TEMA 5

Ejercicio

Aplicando el Ciclo de Vida

Ejemplo de Cronómetro

Cronómetro



Composición de Componentes

- Se basa en la encapsulación de elementos donde cada uno aporta una nueva funcionalidad o extiende la que ya tenía el elemento.
- Esto nos ayuda de forma intrínseca a crear componentes reutilizables.
- Cuando mejor se defina el componente menor acoplamiento tendrá con el medio y mayor será su capacidad para ser reutilizado.
- Un componente debe existir para aportar una funcionalidad nueva. Aunque sólo fuere visual para enriquecer el aspecto del elemento mediante CSS.

Composición de Componentes

Vamos a ver cómo definir los componentes y vamos a utilizar una regla que es la de dentro hacia fuera.

Para ello, y lo más fácil es tener siempre un boceto o un prototipo, algo donde podamos ir realizando nuestros apuntes sobre los elementos que vamos encontrando.

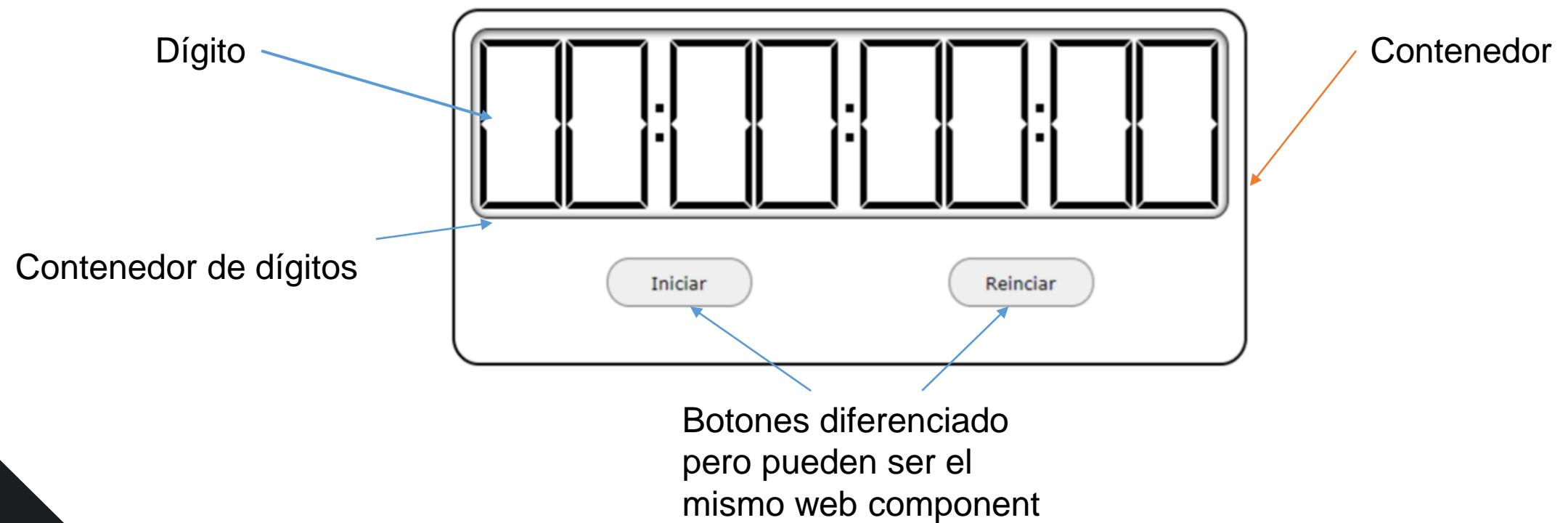
Para identificar los componentes, lo que debemos hacer es ir buscando los elementos atómicos con un función.

La funcionalidad mínima que sea mayor a la funcionalidad de una etiqueta HTML.

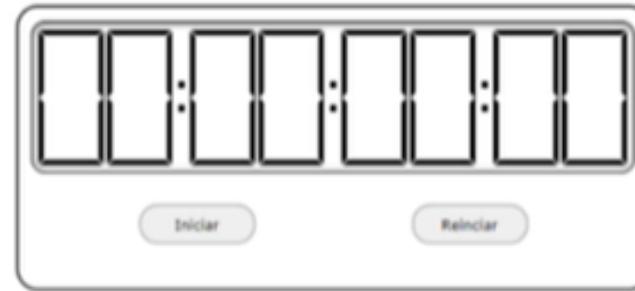
Composición de Componentes

- Tener un boceto, prototipo
- Identificar los elementos atómicos con funcionalidad mínima que sea mayor a la funcionalidad de una etiqueta HTML
- Establecer la responsabilidad

Cronómetro



- Display



- Botón



- Contenedor dígitos

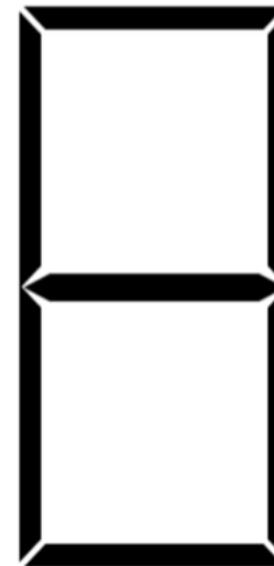


- Dígitos



Composición de Dígito

- Dígitos



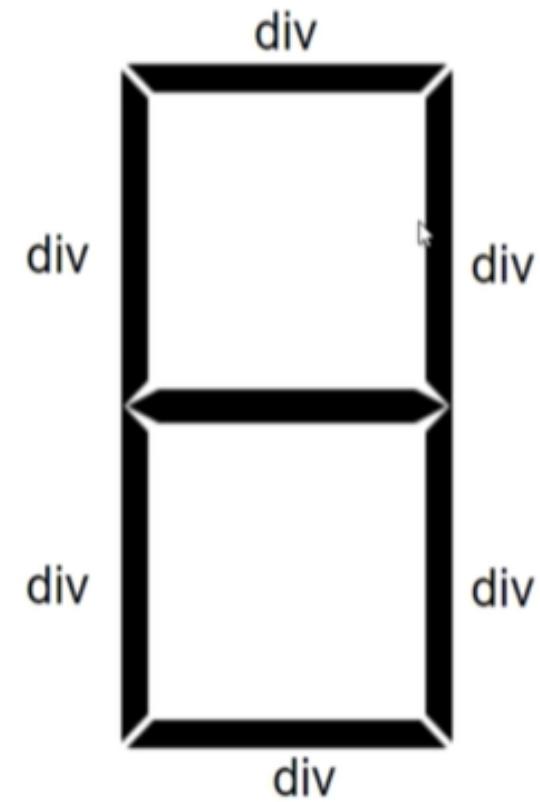
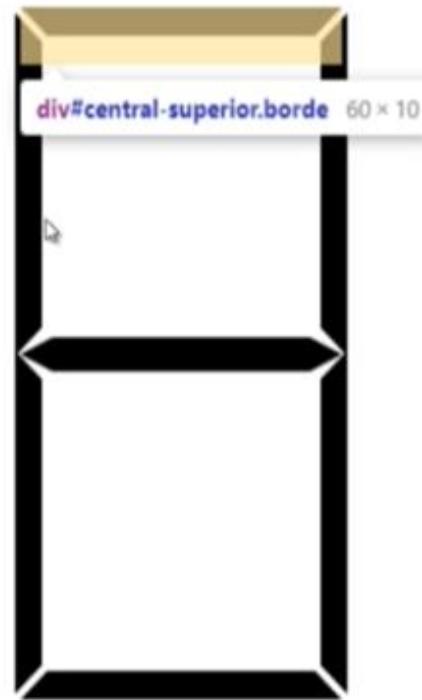
Composición de Dígito

- Dígitos



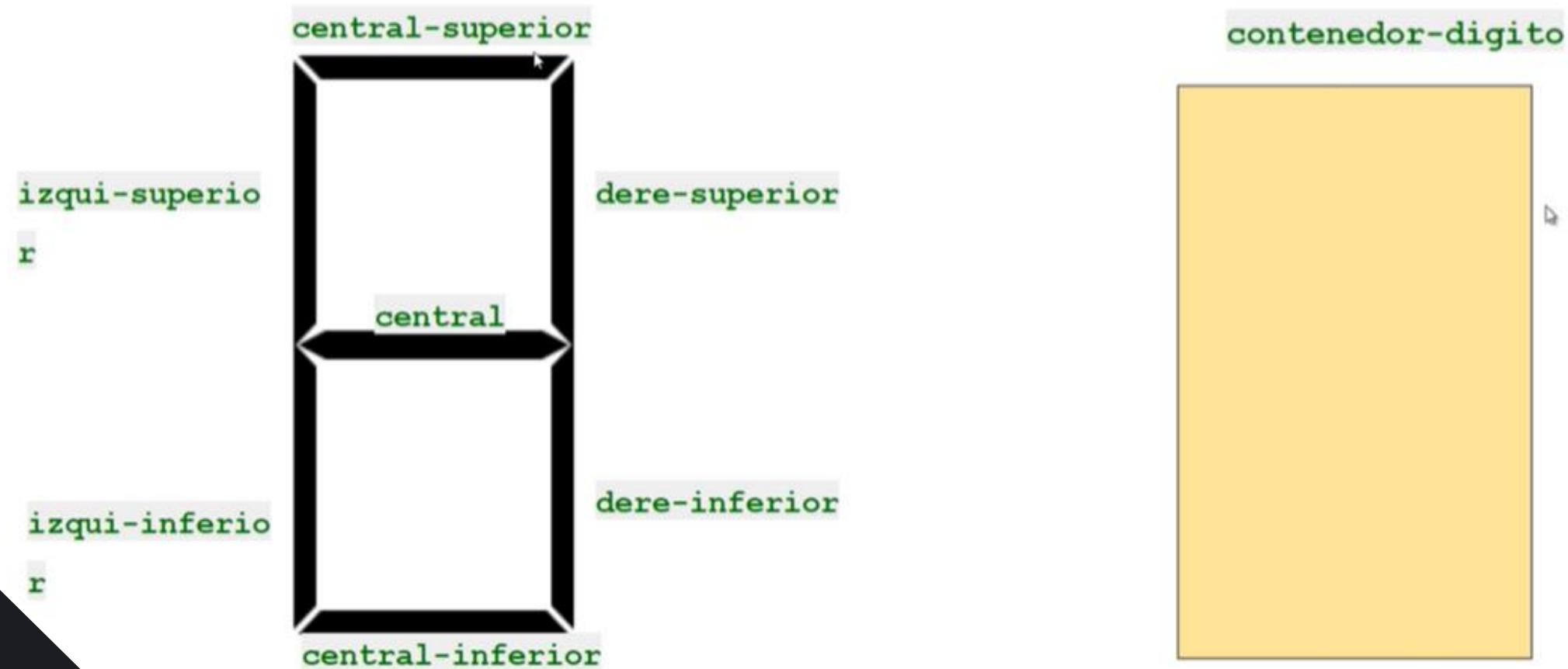
Composición de Dígito

- Dígitos



Los número se formarán con 7 etiquetas div y un contenedor para referenciar su posicionamiento

Composición de Dígito



Composición de Dígito

El template html:

```
<div id="contenedor-digito">  
    <div id="central-superior" class="borde"></div>  
    <div id="izqui-superior" class="borde"></div>  
    <div id="dere-superior" class="borde"></div>  
    <div id="central" class="borde"></div>  
    <div id="izqui-inferior" class="borde"></div>  
    <div id="dere-inferior" class="borde"></div>  
    <div id="central-inferior" class="borde"></div>  
</div>
```

Clase css borde:



#fffff00 ⇒ Transparente

```
.borde{  
    border-style: solid;  
    border-right-color: #fffff00;  
    border-left-color: #fffff00;  
    border-bottom-color: #fffff00;  
    border-width: 5px;  
}
```

Composición de Dígito

Selector css contenedor-digito

```
#contenedor-digito{  
    display: inline-block;  
    position: absolute;  
    height: 126px;  
    width: 65px;  
}
```



Composición de Dígito

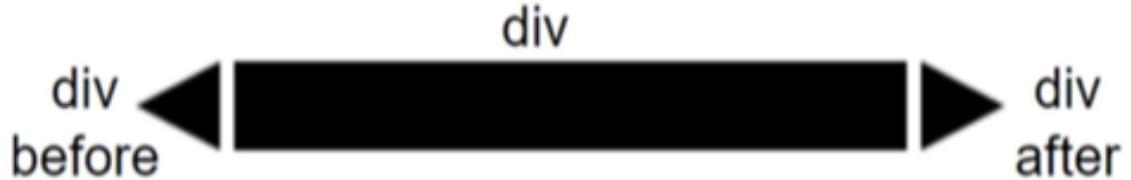
Selector css sobre cada uno de los div

```
#central-superior{  
    position: absolute;  
    width: 50px;  
    left: 2px;  
    top: 2px;  
}
```

```
#izqui-superior{  
    transform: rotate(-90deg);  
    position: absolute;  
    top: 28px;  
    left: -24px;  
    width: 50px;  
}
```

Composición de Dígito

```
#central{  
    position: absolute;  
    border-bottom-color: black;  
    border-width: 3px;  
    top: 59.5px;  
    width: 42px;  
    left: 8px;  
    border-color: black;  
}  
  
.central:before {  
    content: " ";  
    position: absolute;  
    display: block;  
    width: 0;  
    height: 0;  
    border-style: solid;  
    left: -9px;  
    border-width: 3px 6px 3px 0;  
    border-color: transparent black transparent transparent;  
    bottom: -3px;  
}  
  
.central:after {  
    content: " ";  
    position: absolute;  
    display: block;  
    width: 0;  
    height: 0;  
    border-style: solid;  
    right: -9px;  
    bottom: -3px;  
    border-width: 3px 0 3px 6px;  
    border-color: transparent transparent transparent black;  
    bottom: -3px;  
}
```



DIPLOMADO

Front End



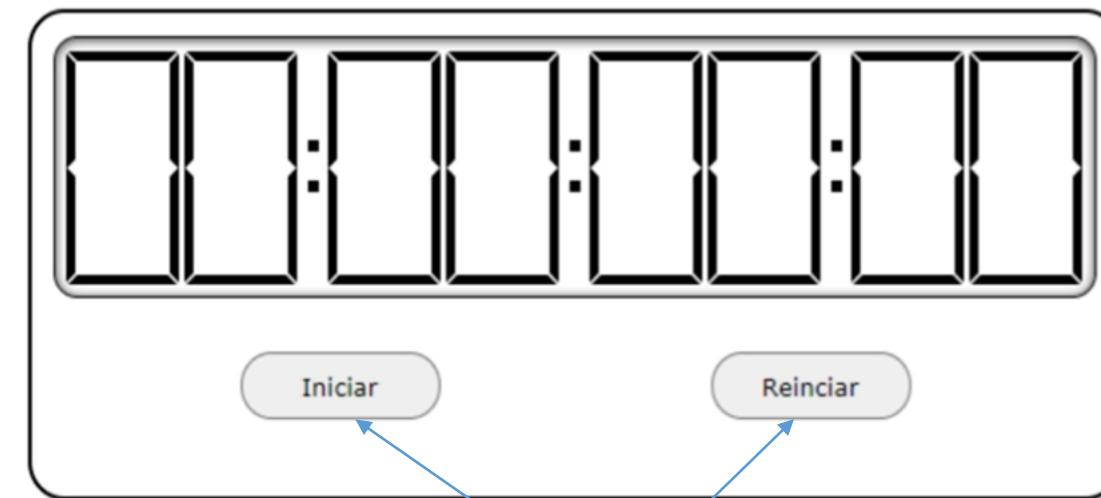
TEMA 5

Ejercicio

Preparando el elemento Dígito

Composición de Componentes

Cronómetro



Botones diferenciado
pero pueden ser el
mismo web component

Composición de Componentes

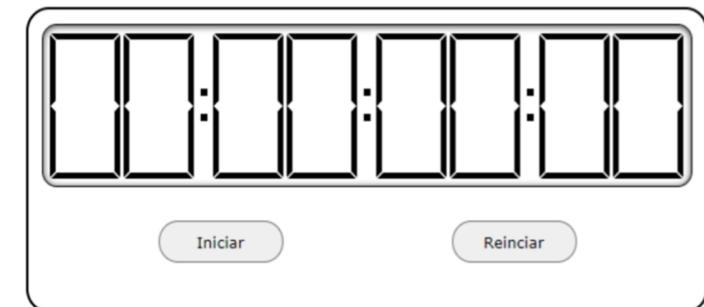
El botón Iniciar actúa de la siguiente manera:

- En el momento en que se pulsa empieza el contador a funcionar y el texto que muestra son los número representando el tiempo transcurrido.
- Si pulsamos nuevamente nos detiene el contador.
- Y si volvemos a presionarlo, inicia otra vez el contador, le deja que siga contando.

El botón Reiniciar actuá de la siguiente manera:

- si lo pulsamos cuando está contando nos reinicia sin parar de contar.
- Si le damos a pausa, reiniciamos a cero, sería como un reinicio total y el botón de iniciar volvería a activarlo, como hemos hecho hasta ahora.

Cronómetro



DIPLOMADO

Front End



TEMA 5

Ejercicio
Completando Botones
y funcionalidad

DIPLOMADO

Front End

UVM

TEMA 6

Librerías Javascript
Basadas en el Estándar
Web Components

Librerías JavaScript

Especialmente en los comienzos, programar componentes web puede resultar complicado. En la web, sin embargo, existen numerosas **librerías** con plantillas, funciones estándares y ejemplos prácticos de web components que te harán el trabajo más fácil:

- [**Lit Element**](#): una base sencilla para crear componentes web.
- [**Polymer Project**](#): dentro del marco del Polymer Project, Google ofrece diferentes herramientas para trabajar con componentes web. Entre muchos otros elementos listos para usar, hay un kit de iniciación para programar apps con componentes web y una biblioteca de plantillas HTML para JavaScript, por ejemplo.
- [**Hybrids**](#): sencilla biblioteca de UI para crear componentes web.
- [**Slim.js**](#): biblioteca con propiedades ampliadas para web basada en la **herencia de clases de JavaScript ES6**.

Lit Element

LitElement es una clase base simple para crear componentes web rápidos y ligeros que funcionan en cualquier página web con cualquier framework (Angular, Reactjs, Vuejs)

LitElement usa lit-html para renderizar en Shadow DOM y agrega una API para administrar propiedades y atributos. Las propiedades se observan de forma predeterminada y los elementos se actualizan de forma asincrónica cuando cambian sus propiedades.

A la hora de desarrollar se puede utilizar tanto TypeScript como JavaScript Vanilla.

Se puede utilizar para crear tanto componentes reutilizables para ser compartidos como componentes específicos de aplicación

LitElement es la evolución de Polymer. Los componentes de Polymer pueden utilizarse con LitElement y viceversa.

Lit Element

Vamos a aprender cómo utilizar LitElement creando una aplicación conceptual que tendrá un router reutilizable declarativo y componentes propios de la aplicación.

LitElement necesita un gesto de módulos para consumir los módulos sin hacer referencia a la ruta relativa de su ubicación. Esté será Webpack.

DIPLOMADO

Front End

UVM

TEMA 6

Ejercicio

Introducción a Lit Element

DIPLOMADO

Front End

UVM

TEMA 6

Ejercicio

Botones en Lit Element

DIPLOMADO

Front End

UVM

TEMA 6

Ejercicio

Tablas en Lit Element

DIPLOMADO

Front End



UVM

TEMA 6

Ejercicio

Registro en Lit Element