
ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
PROGRAMACIÓN INTEGRATIVA DE COMPONENTES

CARLOS POGO

# TAREA 1.7: PRESENTACIONES DE COMPONENTES WEB

NRC: 16496

DOCENTE: JOSÉ SANCHO

2024

# 11 TEMPLATE

## index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
   <title>Custom Elements</title>
   <style>
      .title {
         color: blue;
      }
   </style>
</head>
<body>
   <h2 class="title">SECCION HTML - Pagina Principal</h2>
   <template>
    <h2>SECCION etiqueta template de la Pagina principal</h2>
    <div>
     <p>div parrafo de la Pagina Principal</p>
    </div>
   </template>
   <elemento-template></elemento-template>
   <script type="module" src="./my-element.js"></script>
</body>
</html>
```

# CÓDIGO

- `<!DOCTYPE html>`: Declara que el documento es un HTML5.
- `<html lang="en">`: Inicia el elemento raíz del documento HTML y establece el idioma del contenido a inglés.
- `<meta charset="UTF-8" />`: Define la codificación de caracteres como UTF-8.
- `<meta name="viewport" content="width=device-width, initial-scale=1.0" />`: Hace que la página sea responsiva ajustando el tamaño de la ventana gráfica.
- `<title>Custom Elements</title>`: Define el título del documento que se muestra en la pestaña del navegador.
- `<style>`: Incluye estilos CSS internos. Aquí, el selector `.title` establece el color del texto a azul.
- `<h2 class="title">SECCION HTML - Pagina Principal</h2>`: Un encabezado de nivel 2 con la clase `title`, que hará que el texto sea azul.
- `<template>`: Define un fragmento de HTML que no se renderiza automáticamente en la página. Puede ser utilizado posteriormente con JavaScript. Contiene un encabezado `<h2>` y un `<div>` con un párrafo.
- `<elemento-template></elemento-template>`: Un elemento personalizado (custom element). La funcionalidad de este elemento se define en el archivo `my-element.js`.
- `<script type="module" src="./my-element.js"></script>`: Importa un módulo JavaScript desde el archivo `my-element.js`. El uso de `type="module"` permite el uso de la sintaxis de módulos ES6.

# 11 TEMPLATE

my-element.js

```
class myElement extends HTMLElement {
   constructor() {
      super();
   }
   getTemplate() {
      const template = document.createElement("template");
      template.innerHTML = `
     <section>
       <h1>TEMPLATE Web Component</h1>
       <h2 class="title">Mensaje desde template dentro del
WC</h2>
       <div id="div_section_1">
         <p>Parrafo_1 dentro del Template dentro del
WC</p>
       </div>
       <div id="div_section_2">
         <p>Parrafo_2 metodo getTemplate() dentro del
WC</p>
       </div>
     </section>
     ${this.getStyles()}
   `;
      return template;
   }
   getStyles() {
      return `
     <style>
       h2 {
         color: red;
         background-color:yellow;
       }
     </style>
   `;
   }
```

```
   render() {

      alert('metodo render ');
      this.appendChild(this.getTemplate().content.cloneNode(tr
ue));
   }
   connectedCallback() {
      alert('metodo connectedCallback ');

      this.render();
   }
}
customElements.define("elemento-template", myElement);
```

- **`myElement`**: Clase que define el comportamiento del nuevo elemento.
- **`getTemplate`**: Método que crea un template con HTML y CSS.
- **`getStyles`**: Método que define estilos CSS específicos para el componente.
- **`render`**: Método que clona y añade el contenido del template al componente.
- **`connectedCallback`**: Método que se ejecuta al insertar el componente en el DOM y llama al render.
- **`customElements.define`**: Registra el nuevo componente con el nombre **`elemento-template`**.

Este código crea un nuevo elemento HTML personalizado que muestra contenido específico y estilos cuando se añade al DOM.

# 11 TEMPLATE

CORRIDA

# 12 SD CÓDIGO

## index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Custom Elements</title>
    <style>
      h2 {
        color: blue;
      }
    </style>
  </head>
  <body>
    <h2 class="title">Soy un titulo</h2>
    <template>
      <h2>Hola mundo again</h2>
      <div>
        <p>Soy más texto ejemplo de la clase</p>
      </div>
    </template>
    <my-element></my-element>
    <script type="module" src="./my-element.js"></script>
  </body>
</html>T
```

Estructura HTML:

- Define un título con clase **`title`** que se muestra en azul.
- Contiene un elemento **`<template>`** que no se renderiza automáticamente.
- Incluye el elemento personalizado **`<my-element`**
- **`<my-element></my-element>`**: Un elemento personalizado (custom element) que se define en el archivo **`my-element.js`**.

# 12 SD CÓDIGO

## my-element.js

```
class myElement extends HTMLElement {
 constructor() {
  super();
  this.attachShadow({ mode: "open" });
 }
 getTemplate() {
  const template = document.createElement("template");
  template.innerHTML = `
    <section>
      <h2 class="title">Hola mundo!</h2>
      <div>
        <p>Soy más texto ejemplo</p>
      </div>
    </section>
    ${this.getStyles()}
  `;
  return template;
 }
 getStyles() {
  return `
    <style>
      h2 {
        color: red;
      }
    </style>
  `;
 }
 render() {
this.shadowRoot.appendChild(this.getTemplate().content.cloneNode(true));
  }
  connectedCallback() {
   this.render();
  }
}
customElements.define("my-element", myElement);
```

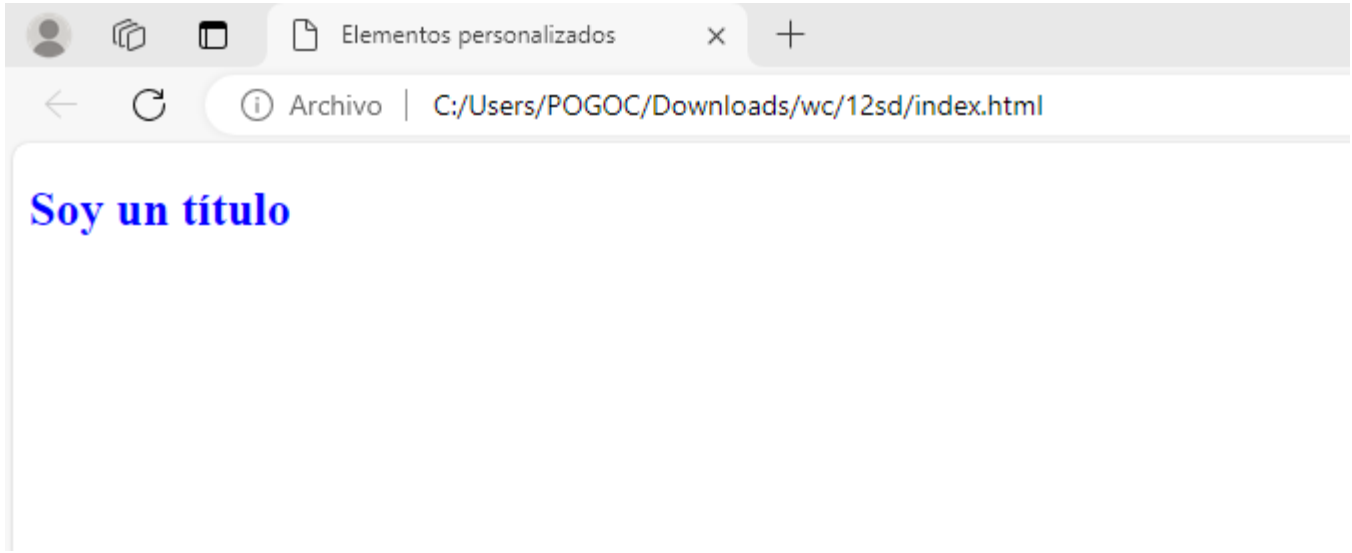
- **`myElement`**: Clase que define el comportamiento del nuevo elemento.
- **`getTemplate`**: Método que crea un template con HTML y CSS.
- **`getStyles`**: Método que define estilos CSS específicos para el componente.
- **`render`**: Método que clona y añade el contenido del template al Shadow DOM del componente.
- **`connectedCallback`**: Método que se ejecuta al insertar el componente en el DOM y llama al render.
- **`customElements.define`**: Registra el nuevo componente con el nombre **`my-element`**.

Funcionamiento del Código
Cuando se añade el elemento **`<my-element>`** al DOM:

- Se ejecuta el constructor, que adjunta un Shadow DOM al componente.
- Se llama a **`connectedCallback`**, que invoca el método **`render`**.
- El método **`render`** clona el contenido del template (incluyendo los estilos) y lo añade al Shadow DOM.
- El contenido del template, que incluye un encabezado **`<h2>`** con texto rojo y un párrafo, se renderiza dentro del Shadow DOM, encapsulando los estilos y el contenido de otros elementos del DOM.

# 12 SD

CORRIDA



# 15 SLOT CÓDIGO

## Index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-
width, initial-scale=1.0" />
  <title>Custom Elements</title>
 </head>
 <body>
  <h2 class="title">Soy un titulo</h2>
  <my-element>
   <span slot="title">Soy el titulo</span>
   <span slot="parrafo">Soy el texto del
parrafo</span>
  </my-element>
  <script type="module" src="./my-
element.js"></script>
 </body>
</html>
```

- Contiene la estructura básica de un documento HTML5 con una metaetiqueta para la codificación de caracteres y una metaetiqueta para la vista responsiva.
- `<span slot="title">Soy el titulo</span>`: Define el contenido que se insertará en el slot llamado **`title`** dentro del Shadow DOM de **`<my-element>`**.
- `<span slot="parrafo">Soy el texto del parrafo</span>`: Define el contenido que se insertará en el slot llamado **`parrafo`**.
- Dentro del elemento personalizado **`<my-element>`**, hay dos elementos **`<span>`** con atributos **`slot`**. Estos atributos **`slot`** permiten que el contenido sea proyectado en el Shadow DOM del componente personalizado.

# 15 SLOT CÓDIGO

## my-element.js

```
class myElement extends HTMLElement {
 constructor() {
  super();
  this.attachShadow({ mode: "open" });
 }
 getTemplate() {
  const template = document.createElement("template");
  template.innerHTML = `
    <section>
      <h2>
        <slot name="title"></slot>
      </h2>
      <div>
       <p>
        <slot name="parrafo"></slot>
       </p>
      </div>
    </section>
    ${this.getStyles()}
  `;
  return template;
 }
```

```
getStyles() {
    return `
      <style>
       h2 {
         color: red;
       }
      </style>`;
  }
  render() { this.shadowRoot.appendChild(this.getTemplate().content.cloneNode(true)); }
  connectedCallback() {
    this.render();
  }
}
customElements.define("my-element", myElement);
```

- **myElement**: Clase que define el comportamiento del nuevo elemento.
- **getTemplate**: Método que crea un template con HTML y CSS.
- **getStyles**: Método que define estilos CSS específicos para el componente.
- **render**: Método que clona y añade el contenido del template al Shadow DOM del componente.
- **connectedCallback**: Método que se ejecuta al insertar el componente en el DOM y llama al render.
- **customElements.define**: Registra el nuevo componente con el nombre my-element.

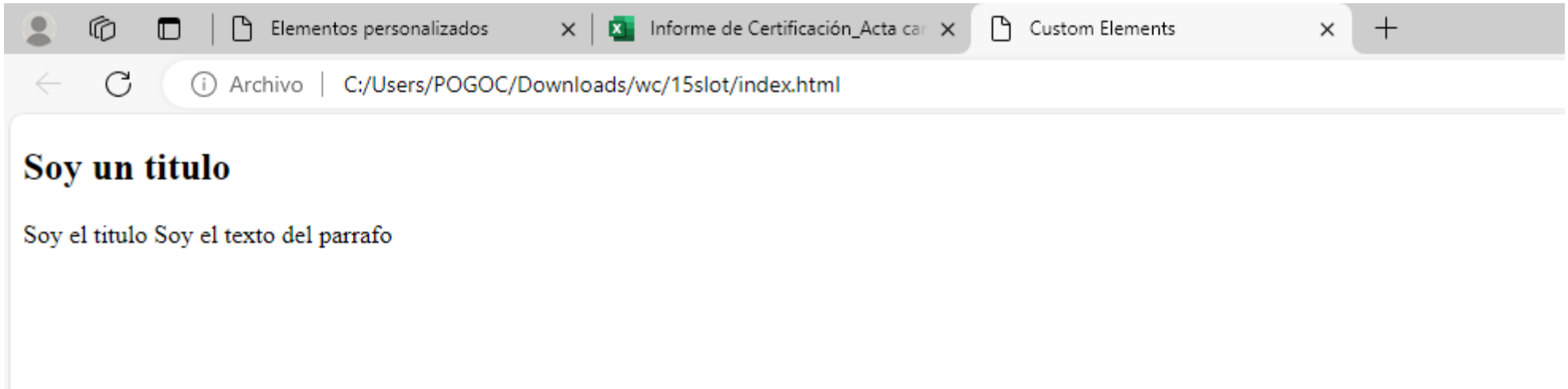
**Funcionamiento del Código**

Cuando se añade el elemento <my-element> al DOM:

- El constructor adjunta un Shadow DOM al componente.
- connectedCallback invoca el método render.
- render añade el contenido del template (incluyendo los slots) al Shadow DOM.
- Los elementos <slot name="title"> y <slot name="parrafo"> permiten que el contenido definido en el DOM principal se proyecte en el Shadow DOM del componente, reemplazando los slots con el contenido correspondiente.

# 15 SLOT

CORRIDA



# 16 SLOTTED CÓDIGO

Index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-
width, initial-scale=1.0" />
  <title>Slotted</title>
 </head>
 <body>
  <my-element>
   <span slot="title">Hola desde una etiqueta
h1</span>
   <span class="text" slot="parrafo"
    >Hola desde una etiqueta de parrafo</span
   >
  </my-element>
  <script type="module" src="./my-
element.js"></script>
 </body>
</html>
```

- Contiene dos elementos **`<span>`** con atributos **`slot`**.
- El primer **`<span slot="title">Hola desde una etiqueta h1</span>`** define el contenido que se proyectará en el slot llamado **`title`**.
- El segundo **`<span class="text" slot="parrafo">Hola desde una etiqueta de parrafo</span>`** define el contenido que se proyectará en el slot llamado **`parrafo`**.

# 16 SLOTTED

## my-element.js

```
class myElement extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" });
  }
  getTemplate() {
    const template =
document.createElement("template");
    template.innerHTML = `
    <section>
      <h1>
        <slot name="title"></slot>
      </h1>
      <p>
        <slot name="parrafo"></slot>
      </p>
      <slot></slot>
    </section>

    ${this.getStyles()}
    `;
    return template;
  }
  getStyles() {
    return `
      <style>
        ::slotted(span) {
          font-size: 30px;
          color: red;
        }
```

## CÓDIGO

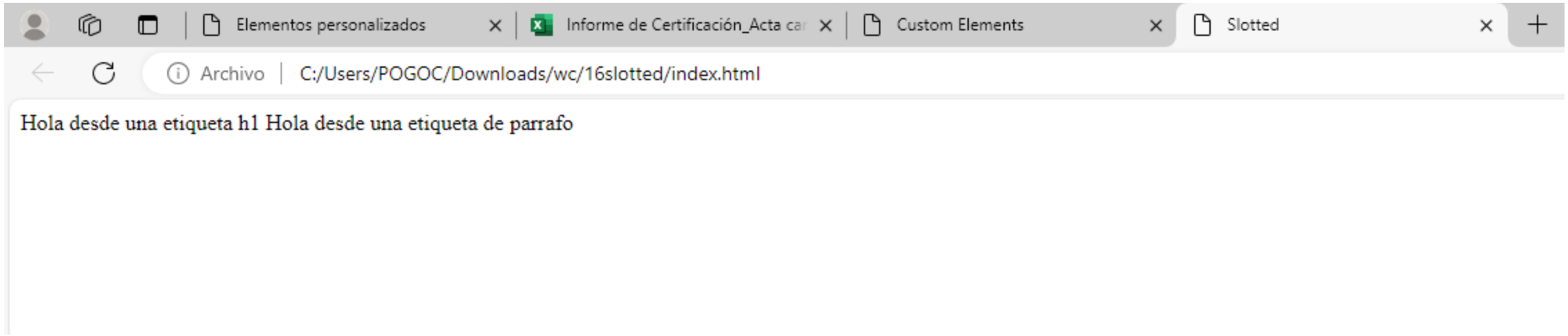
```
        ::slotted(.text) {
          color: blue;
        }
      </style>
    `;
  }
  render() {
    this.shadowRoot.appendChild(this.getTemplate().content.cloneNode(true));
  }
  connectedCallback() {
    this.render();
  }
}
customElements.define("my-element", myElement);
```

## Resumen del Código JavaScript

- Clase **`myElement`**:
  - Define un nuevo elemento personalizado `<my-element>` que hereda de `HTMLElement`.
  - Usa el Shadow DOM para encapsular el contenido y los estilos.
- Método **`getTemplate`**:
  - Crea una plantilla HTML con slots:
    - `<slot name="title"></slot>` para insertar contenido en el encabezado `<h1>`.
    - `<slot name="parrafo"></slot>` para insertar contenido en el párrafo `<p>`.
    - Un slot sin nombre (`<slot></slot>`) para cualquier contenido adicional.
- Método **`getStyles`**:
  - Define estilos específicos para los elementos proyectados (slotted elements):
    - `::slotted(span)`: Establece el tamaño de fuente en 30px y el color rojo para cualquier `<span>`.
    - `::slotted(.text)`: Establece el color azul para cualquier elemento con la clase `.text`.
- Método **`render`**:
  - Añade el contenido del template al Shadow DOM del componente.
- Método **`connectedCallback`**:
  - Se ejecuta cuando el componente se añade al DOM y llama al método **`render`** para renderizar el contenido.
- Definición del elemento personalizado:
  - `customElements.define("my-element", myElement)` registra el nuevo elemento personalizado `<my-element>`.

# 16 SLOTTED

CORRIDA



# 17 ATRIBUTE CÓDIGO

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-
width, initial-scale=1.0" />
    <title>Custom Elements</title>
  </head>
  <body>
    <h2 class="title">Soy un titulo</h2>
    <my-element
      title="Soy un titulo!"
      parrafo="Soy el texto del parrafo"

img="https://avatars3.githubusercontent.com/u/19057
08?s=280&v=4"
    ></my-element>
    <script type="module" src="./my-
element.js"></script>
  </body>
</html>
```

- El documento HTML5 define un elemento personalizado (my-element).
- Se incluyen atributos personalizados (title, parrafo, img) en el elemento my-element.
- Un script (my-element.js) se importa como módulo para definir el comportamiento del elemento personalizado.
- La página también contiene un encabezado (<h2>) con texto.

- Este documento muestra cómo integrar un elemento web personalizado con atributos específicos y cómo importar su definición desde un archivo JavaScript externo.

# 17 ATTIBUTE

CÓDIGO

## my-element.js

```
class myElement extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" });

    this.title = this.getAttribute("title");
    this.parrafo = this.getAttribute("parrafo");
    this.img = this.getAttribute("img");
  }
  getTemplate() {
    const template =
document.createElement("template");
    template.innerHTML = `
      <section>
        <h2>${this.title}</h2>
        <div>
         <p>${this.parrafo}</p>
         <img src=${this.img}/>
        </div>
      </section>
      ${this.getStyles()}
    `;
    return template;
  }
  getStyles() {
    return `
      <style>
        h2 {
          color: red;
        }
      </style>
    `;
  }
  render() {
    this.shadowRoot.appendChild(this.getTemplate().content.cloneNode(true));
  }
  connectedCallback() {
    this.render();
  }
}
customElements.define("my-element", myElement);
```

1. Constructor: En el constructor, se adjunta un Shadow DOM abierto y se obtienen los atributos title, parrafo, e img del elemento.

2. getTemplate: Crea y devuelve una plantilla HTML con el contenido del elemento, incluyendo un título (<h2>), un párrafo (<p>), y una imagen (<img>). También incluye los estilos definidos en el método getStyles.

3. getStyles: Devuelve una cadena con estilos CSS, en este caso, que define que los <h2> dentro del Shadow DOM deben ser de color rojo.

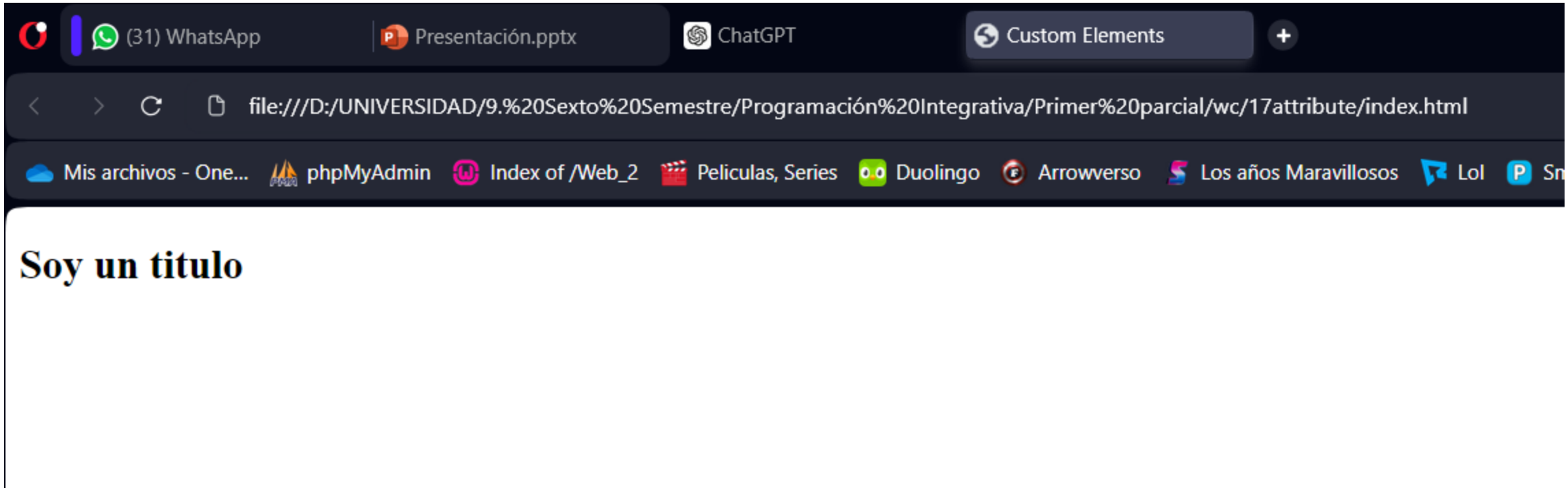
4. render: Adjunta la plantilla al Shadow DOM del elemento.

5. connectedCallback: Se ejecuta cuando el elemento es añadido al DOM, llamando al método render para insertar la plantilla en el Shadow DOM.

6. customElements.define: Define el nuevo elemento my-element usando la clase myElement.

# 17 ATTRIBUTE

CORRIDA



# 18 attributeChangedCallback

CÓDIGO

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-
width, initial-scale=1.0" />
    <title>Custom Elements</title>
  </head>
  <body>
    <h2 class="title">Soy un titulo</h2>
    <my-element
      title="Soy un titulo!!!!!"
      parrafo="Soy el texto del parrafo"

img="https://avatars3.githubusercontent.com/u/19057
08?s=280&v=4"
    ></my-element>
    <script type="module" src="./my-
element.js"></script>
  </body>
</html>
```

- Se define un documento HTML5 con un título de página "Custom Elements".
- En el cuerpo, hay un encabezado <h2> con el texto "Soy un titulo".
- Se usa un elemento web personalizado <my-element> con atributos title, parrafo e img para establecer su contenido.
- Se importa un script de módulo my-element.js que define el comportamiento del elemento personalizado.

# 18 attributeChangedCallback

CÓDIGO

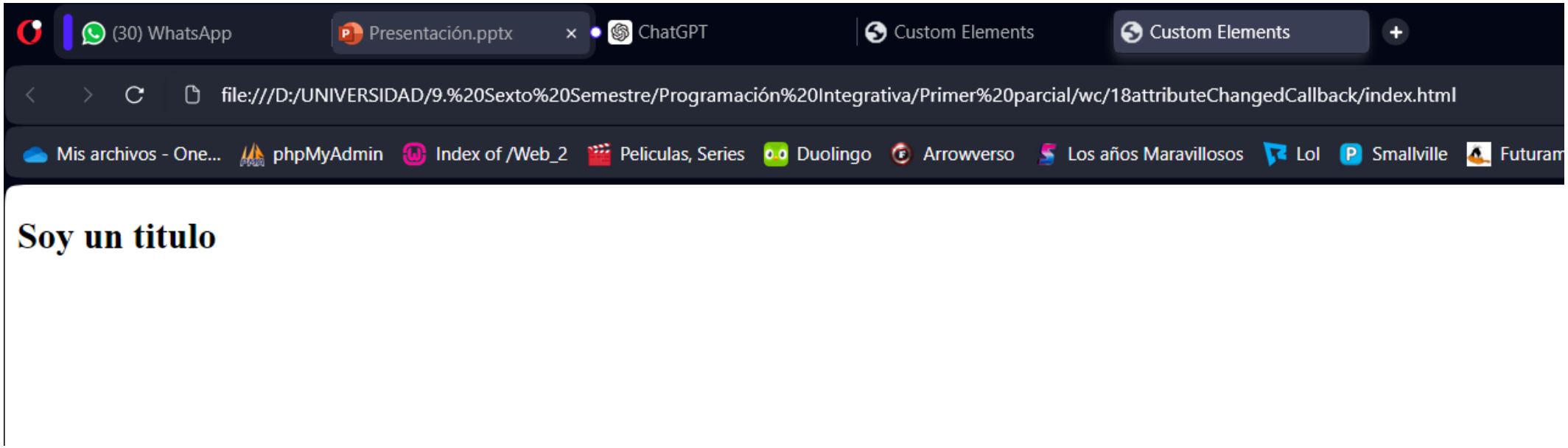
## my-element.js

```
class myElement extends HTMLElement {
 constructor() {
  super();
  this.attachShadow({ mode: "open" });
 }
 static get observedAttributes() {
  return ["title", "parrafo", "img"];
 }
 attributeChangedCallback(attr, oldVal, newVal) {
  if (attr === "title") {
   this.title = newVal;
  }
  if (attr === "parrafo") {
   this.parrafo = newVal;
  }
  if (attr === "img") {
   this.img = newVal;
  }
 }
 getTemplate() {
  const template = document.createElement("template");
  template.innerHTML = `
   <section>
    <h2>${this.title}</h2>
    <div>
     <p>${this.parrafo}</p>
     <img src=${this.img}/>
    </div>
   </section>
 ${this.getStyles()}
  `;
  return template;}
 getStyles() {
  return `
   <style>
    h2 {        color: red;
    }
   </style>   `;
 }
 render() {
  this.shadowRoot.appendChild(this.getTemplate().content.cloneNode(true));  }
 connectedCallback() {
  this.render();  }}
customElements.define("my-element", myElement);
```

1. Clase myElement: Define un elemento personalizado my-element que extiende HTMLElement.
2. Constructor: Inicializa el elemento y adjunta un Shadow DOM abierto.
3. observedAttributes: Método estático que devuelve una lista de atributos que el elemento observará para cambios ("title", "parrafo", "img").
4. attributeChangedCallback: Método llamado cuando cambia alguno de los atributos observados. Actualiza las propiedades title, parrafo, y img según los nuevos valores.
5. getTemplate: Crea una plantilla HTML que incluye el contenido del elemento (<h2>, <p>, <img>) utilizando las propiedades actualizadas.
6. getStyles: Retorna estilos CSS que se aplican al Shadow DOM, como hacer que los <h2> sean rojos.
7. render: Añade la plantilla al Shadow DOM del elemento.
8. connectedCallback: Método llamado cuando el elemento se conecta al DOM. Llama al método render para inicializar el contenido.

# 18 attributeChangedCallback

CORRIDA



# 19disconnectedCallback

## Index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-
width, initial-scale=1.0" />
  <title>Document</title>
 </head>
 <body>
          <h1>disconnectedCallback</h1>
  <my-custome-element></my-custome-element>
  <script type="module" src="./app.js"></script>
 </body>
</html>
```

## CÓDIGO

1. <h1>disconnectedCallback</h1>: Un encabezado principal que dice "disconnectedCallback".

2. <my-custome-element></my-custome-element>: Un elemento personalizado <my-custome-element> que será definido y manipulado en el archivo JavaScript app.js.

3. <script type="module" src="./app.js"></script>: Importa un script de módulo JavaScript desde el archivo app.js, que probablemente define el comportamiento del elemento personalizado y cómo interactúa con él en el DOM.

# 19disconnectedCallback

CÓDIGO

## my-element.js

```
class MyCustomElement extends HTMLElement {
  constructor() {
    super();
    console.log("Hola desde el constructor - Memoria");
  }
  connectedCallback() {
    console.log("Hola desde el DOM");
  }
  disconnectedCallback() {
    console.log("Adios del DOM");
  }
}
customElements.define("my-custome-element",
MyCustomElement);

document.querySelector("my-custome-element").remove();
```

Clase MyCustomElement:

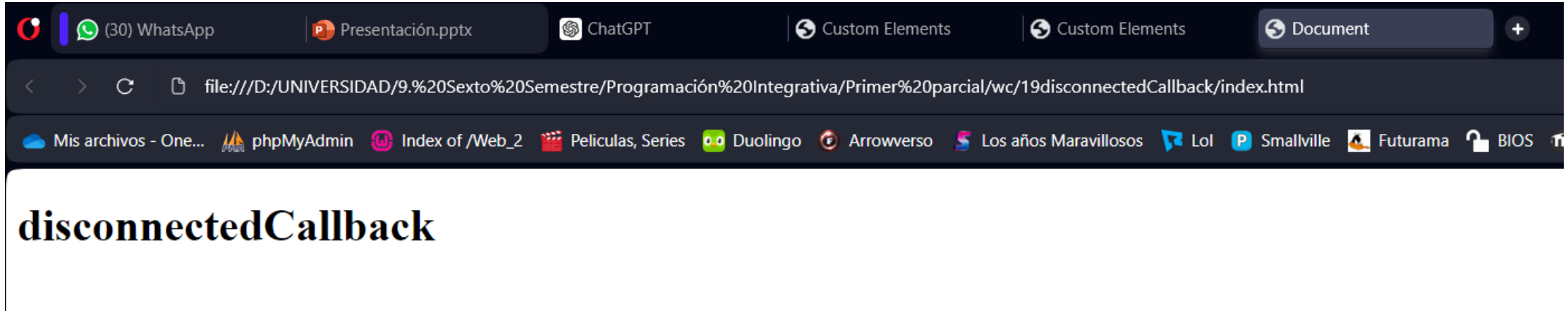
1. Constructor: Se ejecuta al crear una instancia del elemento personalizado. Imprime en la consola "Hola desde el constructor - Memoria".
2. connectedCallback(): Se llama automáticamente cuando el elemento es insertado en el DOM. Imprime "Hola desde el DOM" en la consola.
3. disconnectedCallback(): Se llama automáticamente cuando el elemento es eliminado del DOM (en este caso, cuando se elimina con document.querySelector("my-custome-element").remove();). Imprime "Adios del DOM" en la consola.

customElements.define("my-custome-element", MyCustomElement): Registra la clase MyCustomElement como un elemento personalizado my-custome-element, permitiendo su uso en el HTML.

document.querySelector("my-custome-element").remove();: Selecciona el primer elemento my-custome-element en el documento y lo elimina del DOM. Esto activa el método disconnectedCallback() de la clase MyCustomElement, imprimiendo "Adios del DOM" en la consola.

# 19disconnectedCallback

CORRIDA



# 20 host

## Index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>Host</title>
 </head>
 <body>
  <my-element>
   <span slot="title">Hola desde un componente 1</span>
   <span slot="parrafo">Soy texto ejemplo</span>
  </my-element>
  <my-element class="blue">
   <span slot="title">Hola desde un componente 2</span>
   <span slot="parrafo">Soy texto ejemplo</span>
  </my-element>
  <my-element yellow>
   <span slot="title">Hola desde un componente 3</span>
   <span slot="parrafo">Soy texto ejemplo</span>
  </my-element>
  <article class="card">
   <my-element yellow>
    <span slot="title">Hola desde un componente 3</span>
    <span slot="parrafo">Soy texto ejemplo</span>
   </my-element>
  </article>
  <script src="./my-element.js"></script>
 </body>
</html>
```

# CÓDIGO

1. Se utilizan varios elementos personalizados <my-element>, cada uno con contenido insertado dentro de slots ( con atributo slot).

2. Se muestran tres instancias de <my-element> con diferentes configuraciones:
    1. El primer <my-element> no tiene clases adicionales.
    2. El segundo <my-element> tiene la clase blue.
    3. El tercer <my-element> tiene un atributo yellow.

    Además, hay un <my-element> dentro de un <article class="card">, también con el atributo yellow.

    Se importa un script my-element.js que define el comportamiento y la estructura de los elementos <my-element>.

# 20 host

# CÓDIGO

## my-element.js

```
class myElement extends HTMLElement {
 constructor() {
  super();
  this.attachShadow({ mode: "open" });
 }
 getTemplate() {
  const template = document.createElement("template");
  template.innerHTML = `
  <section>
   <h1>
    <slot name="title"></slot>
   </h1>
   <p>
    <slot name="parrafo"></slot>
   </p>
   <slot></slot>
  </section>

  ${this.getStyles()}
  `;

  return template;
 }
 getStyles() {
  return `
   <style>
    :host {
     display: inline-block;
     width: 100%;
     min-width: 300px;
     max-width: 450px;
     font-size: 20px;
     background: grey;
```

```
}
      :host(.blue) {
       background: blue;  }
      :host([yellow]) {
       background: yellow;       }
      :host([yellow]) h1{
       color: grey;}
      :host([yellow]) p {
       color: red; }
      :host-context(article.card) {
       display: block;
       max-width: 100%;}
    </style> `; }
 render() {
  this.shadowRoot.appendChild(this.getTemplate().content.cloneNode(true)); }
 connectedCallback() {
  this.render();
 }
}
customElements.define("my-element", myElement);
```

1. Clase myElement: Define un elemento HTML personalizado my-element.
2. Constructor: Configura un Shadow DOM para encapsular el contenido interno del elemento.
3. getTemplate(): Crea una plantilla con slots (<slot>) para insertar dinámicamente contenido dentro del elemento (title, parrafo y contenido adicional).
4. Estilos (getStyles()):
    1. Define estilos para el elemento base (:host).
    2. Cambia los estilos basados en la presencia de clases (:host(.blue)) y atributos (:host([yellow])).
    3. Aplica estilos específicos cuando el elemento está dentro de un contexto particular (:host-context(article.card)).
5. render(): Coloca la plantilla generada dentro del Shadow DOM para mostrar el contenido en la página.
6. connectedCallback(): Se activa cuando el elemento se inserta en el DOM, llamando al método render() para mostrar su contenido.

# 20 host

CORRIDA