Apuntes para Examen: Integración de Contenido Interactivo con WebComponents

1. Introducción a WebComponents

- **WebComponents** permiten crear elementos HTML personalizados y reutilizables.
- Compuestos por:
  - **Custom Elements**: Nuevas etiquetas HTML.
  - **HTML Templates**: Plantillas de contenido ocultas hasta ser usadas.
  - **Shadow DOM**: Encapsula estilos y estructura.
  - **Módulos ES6**: Organiza el código en archivos reutilizables.

Ejemplo de estructura básica:

```
<mi-componente></mi-componente>
<script>
  class MiComponente extends HTMLElement {
    constructor() {
      super();
      this.innerHTML = '<p>¡Hola, soy un WebComponent!</p>';
    }
  }
  customElements.define('mi-componente', MiComponente);
</script>
```

2. Custom Elements

- Se crean extendiendo `HTMLElement`.
- Métodos importantes:
  - `constructor()`: Inicializa el componente.
  - `connectedCallback()`: Se ejecuta al insertarse en el DOM.
  - `disconnectedCallback()`: Se ejecuta al eliminarse del DOM.
  - `attributeChangedCallback(attr, oldVal, newVal)`: Reacciona a cambios de atributos.
  - `observedAttributes`: Especifica los atributos que observa el componente.

Ejemplo:

```
class MiBoton extends HTMLElement {
```

```
  constructor() {
    super();
    this.innerHTML = '<button>Haz clic</button>';
    this.addEventListener('click', () => alert('¡Clic!'));
  }
}
customElements.define('mi-boton', MiBoton);
```

3. HTML Templates

- Permiten definir contenido reutilizable sin renderizarlo inmediatamente.

Ejemplo:

```
<template id="mi-template">
  <p>Este es un contenido de template</p>
</template>

<script>
  const template = document.getElementById('mi-template').content.cloneNode(true);
  document.body.appendChild(template);
</script>
```

4. Shadow DOM

- Crea un DOM encapsulado para evitar conflictos de estilo y estructura.

Ejemplo:

```
class CajaEncapsulada extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.innerHTML = `
      <style>
        div { color: red; border: 1px solid black; padding: 10px; }
      </style>
      <div>Contenido encapsulado</div>
    `;
  }
}
customElements.define('caja-encapsulada', CajaEncapsulada);
```

5. Módulos de ES6

- Organizan el código en archivos independientes.

Ejemplo de exportación:

```
// archivo.js
export const mensaje = 'Hola desde el módulo';
export default function saludar() { console.log(mensaje); }
```

Ejemplo de importación:

```
import saludar, { mensaje } from './archivo.js';
console.log(mensaje);
saludar();
```

6. Buenas prácticas

- Usa **Shadow DOM** para encapsular estilos.
- Define atributos observables con `observedAttributes`.
- Organiza el código en **módulos ES6**.
- Reutiliza **templates** para reducir repetición.

7. Librerías útiles para WebComponents

- **LitElement** → Para crear WebComponents modernos.
- **Stencil.js** → Compilador de WebComponents.
- **Vue.js, React, Angular** → También permiten arquitecturas basadas en componentes.

Consejo: Evita dependencias innecesarias si puedes usar **WebComponents** nativos.

Con estos apuntes y ejemplos tienes una guía práctica para implementar WebComponents. ¡Éxito en el examen! 🚀