

Ministerio de Hacienda

Intervención General de la Administración del Estado

Oficina de Informática Presupuestaria



Guía Rápida de Accesibilidad Web

Jueves 04 de agosto de 2022

Versión 4.0.

David Ordiales Rodríguez

Copyright © 2022 Ministerio de Hacienda. Gobierno de España



El presente documento está bajo la licencia Creative Commons Reconocimiento-Compartir Igual versión 4.0 España.

Usted es libre de:

- Compartir — copiar y redistribuir el material en cualquier medio o formato.
- Adaptar — remezclar, transformar y crear a partir del material.
- El licenciente no puede revocar estas libertades en tanto usted siga los términos de la licencia.

Bajo las condiciones siguientes:

- Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.

Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor.

Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Esto es un resumen legible por humanos del texto legal (la licencia completa) disponible en <http://creativecommons.org/licenses/by-nc-sa/4.0/deed.es>

Contenido

Guía Rápida de Accesibilidad Web	1
Introducción.....	5
HTML.....	7
DTD	7
Idioma.....	8
Viewport.....	8
Título.....	9
Secciones	10
Encabezados.....	11
Párrafos	13
Listas.....	13
Marcas estructurales vs. marcas de formato.....	14
Enlaces.....	15
Enlaces Gráficos.....	16
Imágenes	16
SVG.....	17
Font Awesome.....	17
Tablas	18
Tablas multinivel.....	19
Formularios	20
Etiquetado	21
Instrucciones.....	24
Validaciones.....	24
Notificaciones	25
Información personal	26
Multimedia	26
CSS.....	29

Separación de contenido y presentación.....	29
Tipos de hojas de estilo	30
Diseño adaptativo	31
Marco de trabajo	32
Reglas para teléfonos inteligentes	32
Grid	33
Flexbox.....	35
Contraste	36
Legibilidad	38
Foco	39
JavaScript	41
Tecnologías compatibles con la accesibilidad.....	41
Navegación mediante el teclado.....	42
Orden de navegación.....	43
Saltar al contenido principal.....	43
Componentes de usuario	44
Cambios de contexto.....	45
Componentes de interfaz de usuario.....	47
WAI-ARIA	48
Patrones de diseño	49
Patrón disclosure	50
Navegación mediante el teclado	51
Presentación del menú.....	53
Implementación mediante JavaScript	57
Consideraciones.....	65
Anexo I	67
Lectores de pantalla	67
NVDA	69
Anexo II	71

JavaScript.....	71
¿Dónde se inserta un script?	71
Eventos.....	71
Referencias.....	72
Algunas cosas que podemos hacer con una referencia.....	73
Contenido	74
Atributos	74
CSS	74
Una cuestión de familia.....	75
Crear elementos.....	78
Recorrer colecciones	79
Eventos dentro de un bucle	80

Introducción

El propósito de las **pautas de accesibilidad web**, [Web Content Accessibility Guidelines \(WCAG\) 2.1](#), es el de obtener una web donde:

- Las técnicas de desarrollo, empleadas en su construcción, no constituyan barreras a las personas con discapacidades, a la hora de navegar por ellas.
- Las tecnologías de asistencia a la accesibilidad como los lectores de pantalla, magnificadores de pantalla y otros tipos de *plugins*, no se encuentren con dificultades a la hora de poder prestar correctamente sus prestaciones.

A continuación, tienes unos cuantos ejemplos típicos a los que se enfrentan las personas con discapacidades cuando navegan por un sitio web:

- Las personas con daltonismo, a las que les cuesta reconocer ciertos tipos de colores, se encuentran con problemas, si la información que se les suministra depende exclusivamente del color.
- Las personas que tengan lesiones por esfuerzos repetitivos y que tengan problemas a la hora de navegar, por ejemplo, mediante el ratón, se encuentran con problemas si no se les proporcionan mecanismos de navegación alternativos como el teclado.
- Las personas con discapacidad auditiva se enfrentan con problemas al acceder a contenidos de audio, si no se les proporciona dicha información en otro formato como los subtítulos.
- Las personas invidentes, que emplean lectores de pantalla y transcritores braille para navegar por las páginas, se encuentran con problemas si éstas no se encuentran correctamente estructuradas y no se proporcionan alternativas textuales para los contenidos no textuales como las imágenes.
- Las personas con déficit de atención e hiperactividad se encuentran con problemas en aquellos sitios que empleen elementos que les puedan distraer (como las animaciones), o que empleen gráficos demasiado complejos o formularios incorrectamente estructurados.
- Las personas mayores se encuentran con problemas si se emplean tamaños tipográficos pequeños, se adaptan mal los textos sobre los que se haga zoom o se emplean componentes de interfaz de usuario con áreas de interacción muy pequeñas.

Las pautas de accesibilidad se constituyen sobre cuatro **principios**:

- **Perceptible**: la información y los componentes de la interfaz de usuario deben ser presentados a los usuarios de modo que ellos puedan percibirlos. Por ejemplo, dotando de alternativas textuales a las imágenes o etiquetando los controles de los formularios.
- **Operable**: los componentes de la interfaz de usuario y la navegación deben ser operables. Por ejemplo, posibilitando una correcta navegación mediante el teclado.

- **Comprensible:** la información y el manejo de la interfaz de usuario deben ser comprensibles. Por ejemplo, especificando el idioma principal de las páginas o los cambios de idioma que puedan producirse dentro de ellas.
- **Robustez:** el contenido debe ser lo suficientemente robusto para que pueda ser interpretado de manera confiable. Por ejemplo, mediante un uso correcto, tanto desde el punto de vista sintáctico como semántico, de la DTD empleada en las páginas pertenecientes a un sitio web.

Cada uno de estos principios se subdividen a su vez en una serie de **pautas**, las cuales a su vez presentan una serie de **criterios de conformidad** que se clasifican por su importancia con las letras A, AA y AAA.

Con esta estructuración en mente es muy habitual realizar una revisión enumerada de cada uno de los criterios de conformidad de las pautas de accesibilidad sobre nuestras webs.

Si bien, esta práctica puede ser recomendable desde el punto de vista de la auditoría (a la hora de obtener una *medida* del estado de accesibilidad de nuestras webs), el planteamiento de esta guía se basa más en el primer propósito de las pautas de accesibilidad web visto anteriormente. El de obtener una web donde:

- Las técnicas de desarrollo, empleadas en su construcción, no constituyan barreras a las personas con discapacidades, a la hora de navegar por ellas.

Los desarrolladores web empleamos básicamente tres tipos de lenguajes web para la implementación de una web:

- **HTML:** un lenguaje de marcas para la creación de páginas web, que permite la estructuración de éstas y la identificación de sus diferentes tipos de contenido mediante el uso de marcas.
- **CSS:** un lenguaje de presentación, que permite la separación entre el contenido (previamente estructurado por HTML), y su presentación mediante diferentes tipos de hojas de estilo en cascada.
- **JavaScript:** un lenguaje de programación de *scripts*, que permite aplicar una capa de comportamiento a los diferentes elementos y componentes que se encuentran en las páginas webs.

Por lo tanto, de lo que se trata es de añadir al conocimiento que ya tenemos de estos lenguajes una capa adicional de uso, mediante el empleo de una serie de técnicas, que nos permitan desarrollar webs accesibles.

Al igual que el estudiante que domina un idioma, adquiere una terminología específica al estudiar una nueva disciplina, nosotros como desarrolladores web, incorporaremos al conocimiento que ya tenemos de los lenguajes web, la especialización en su uso accesible.

Por último, antes de empezar con la lectura de la guía, os recomendamos que empecéis por ver el video [Web Accessibility Perspectives](#).

HTML

El objetivo de HTML es el de identificar mediante **marcas** diferentes tipos de contenido en un texto: secciones, encabezados, párrafos, imágenes, enlaces, listas, tablas, formularios, etc. Constituye pues, los cimientos de una página web.

DTD

WCAG 4.1.1

Una DTD especifica las marcas que se pueden emplear en un *lenguaje de marcas*, así como las marcas que pueden ser hijas de otras marcas.

Se recomienda el empleo de la DTD de HTML5, puesto que esta versión del lenguaje incluye diversas mejoras, relacionadas con la accesibilidad, con respecto a versiones anteriores:

```
<!DOCTYPE html>
```

Puesto que el mero uso de una DTD no asegura la corrección de una página web, se debe comprobar dicha corrección mediante el empleo de herramientas, como las ofrecidas por el servicio de validación de marcado del W3C: <https://validator.w3.org/>

En la siguiente imagen (figura 1), se puede ver un ejemplo del resultado devuelto por el servicio de validación de marcado del W3C:

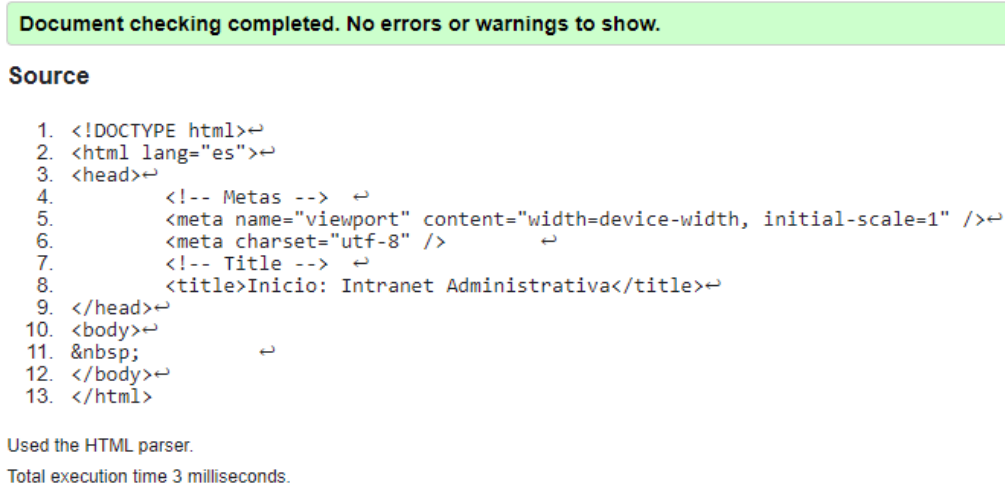


Figura 1

¿Por qué esto es importante? Para determinar el estándar de corrección y validez a emplear en el documento web. Un contenido correcto y válido, desde el punto de vista semántico, elimina y, disminuye, muchos de los impedimentos frente a los que se encuentran las *tecnologías de asistencia a la accesibilidad* (lectores de pantalla,

comandos de voz, magnificadores de pantalla y otros tipos de *plugins*) al procesar una página.

Idioma

WCAG 3.1.1

Especifica el idioma principal de las páginas mediante el atributo **lang** de la marca **HTML**:

```
<html lang="es">
```

Los valores a emplear con el atributo **lang** se pueden consultar en la página de [registro de subetiquetas de IANA](#).

Además del idioma, también es posible indicar la región al cual nuestro idioma pertenece:

```
<html lang="es-ES">
```

Si bien, siempre que sea posible, es preferible emplear el valor más simple: [Language tags in HTML and XML](#).

¿Por qué esto es importante? Para que los lectores de pantalla puedan emplear una voz y pronunciación acorde al idioma que están leyendo.

Viewport

WCAG 1.4.10

A la hora de evitar los desplazamientos horizontales o las reducciones de niveles de zoom, en dispositivos con resoluciones menores a las empleadas habitualmente en equipos de escritorio, es necesario especificar que:

- La anchura del *viewport* coincida con la anchura física del dispositivo donde se esté visualizando la página.
- El nivel del zoom sea el predeterminado.

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

En el siguiente ejemplo (figura 2), la imagen de la izquierda y, de la derecha, muestran la misma página en un dispositivo móvil. Si bien, mientras que en la imagen de la izquierda se ha reducido el nivel de zoom para ajustar la anchura de la página a la del dispositivo (dificultando su lectura), en la imagen de la derecha no se ha reducido el zoom y se ha aplicado un diseño adaptativo:

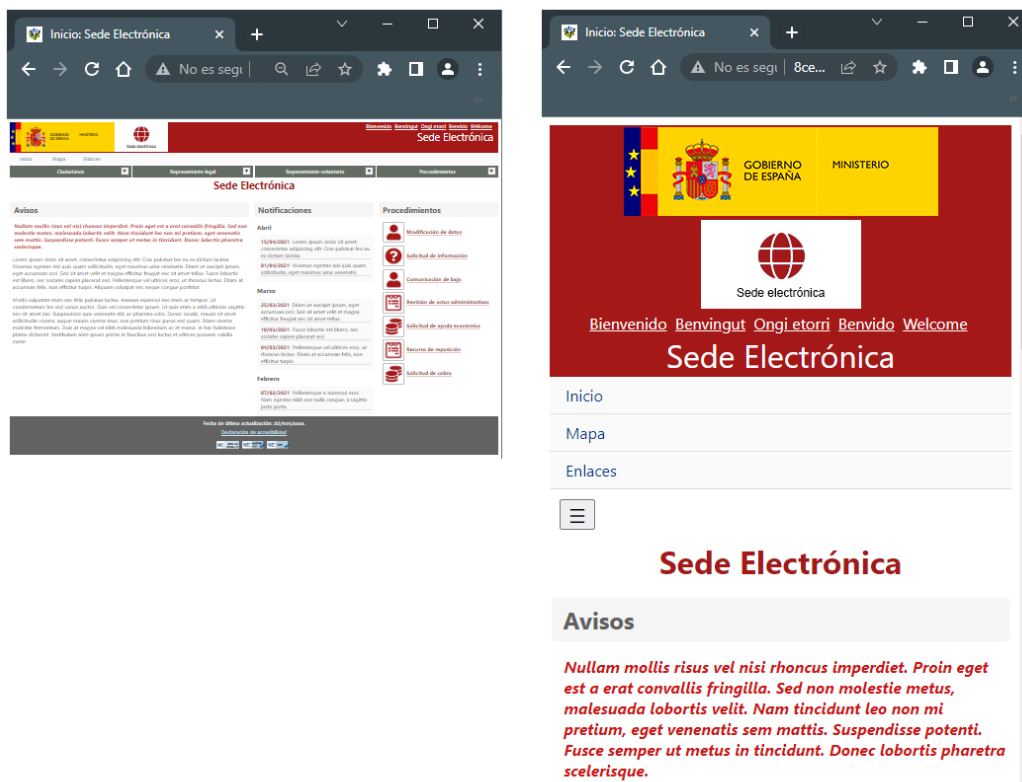


Figura 2

¿Por qué esto es importante? Para que el contenido de la página pueda ser presentado, sin pérdida de información, en diferentes tipos de dispositivos y configuraciones.

Título

WCAG 2.4.2

Incluye un título *descriptivo* del contenido de cada página que informe con claridad al usuario dónde se encuentra dentro de un sitio web:

```
<title>Título descriptivo: Sitio Web</title>
```

En el siguiente ejemplo (figura 3), podemos comprobar mediante la pestaña del navegador, que nos encontramos en la página de **Inicio** de una **Sede Electrónica**:

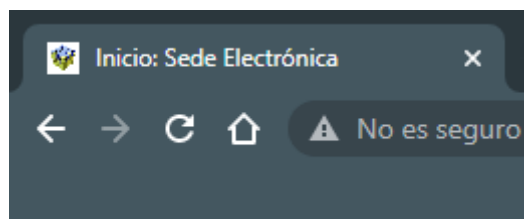


Figura 3

¿Por qué esto es importante? Para que tanto los usuarios como las tecnologías de asistencia a la accesibilidad puedan identificar correctamente la página en la que se encuentran.

Secciones

[WCAG 1.3.1](#) | [WCAG 2.4.1](#) | [WCAG 2.4.6](#) | [Page Regions \(WAI Tutorials\)](#)

Una página bien estructurada es una página que posibilita tanto una navegación eficiente por parte del usuario, como un procesamiento correcto por parte del navegador y de las tecnologías de asistencia a la accesibilidad.

Emplea las marcas de sección de HTML5 (**<header>**, **<nav>**, **<main>**, **<footer>**, etc.) para delimitar e identificar las diferentes secciones empleadas en la página:

```
<header><!-- Encabezado --></header>
<nav aria-label="Navegación secundaria"><!-- Navegación --
></nav>
<nav aria-label="Navegación principal"><!-- Navegación --></nav>
<main><!-- Contenido --></main>
<footer><!-- Pie --></footer>
```

Si existe más de una sección de un mismo tipo, deberemos etiquetarlas a la hora de poder diferenciarlas: [Labeling Regions](#).

En el siguiente ejemplo (figura 4), se han recuadrado y superpuesto los nombres de las secciones correspondientes:

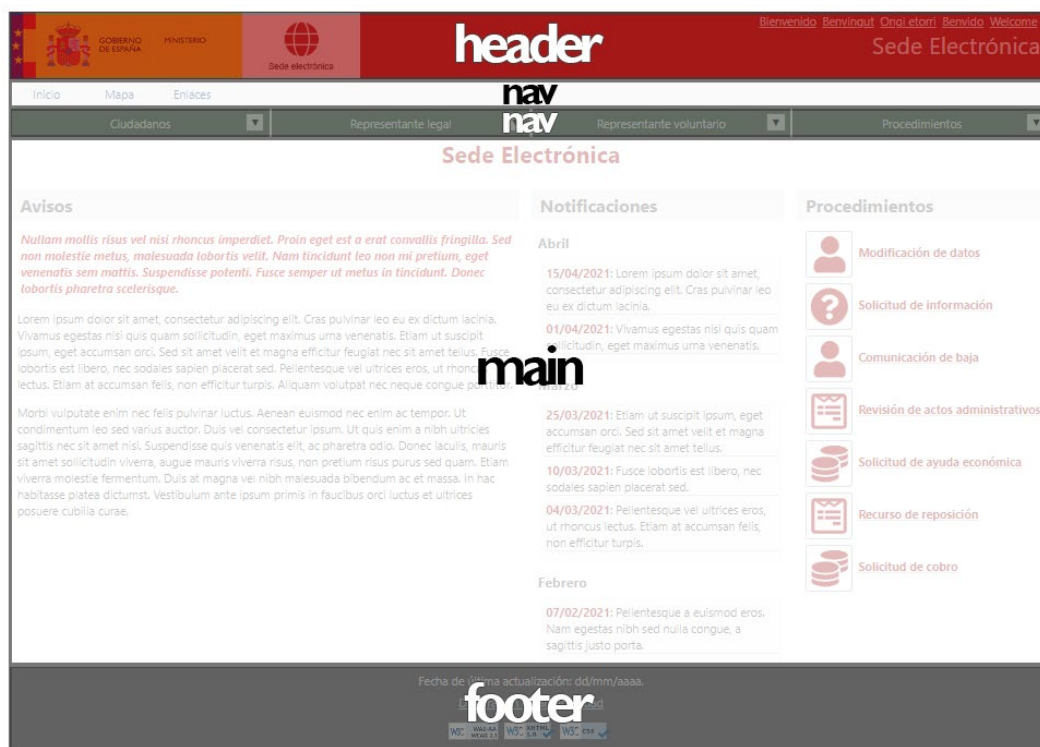


Figura 4

Nota: si necesitamos identificar secciones, dentro del contenido, que no se ajustan a ninguna de las secciones de uso más habituales, podemos emplear la marca `<section>`, para indicar que se tratan de secciones genéricas: [Using HTML5 section element](#).

¿Por qué esto es importante? Porque la estructuración del contenido mediante secciones facilita a las personas su ubicación de una manera más rápida y sencilla. Además, permite el empleo de accesos directos (por ejemplo, a la sección principal), los cuales evitan a los lectores de pantalla la lectura de los contenidos que se encuentren en secciones anteriores a la que nos interesa. Por último, mediante el empleo de ciertos *plugins*, posibilita la navegación directa mediante el teclado a cada sección, evitando así el uso reiterado de la tecla **TAB** para llegar a situar el foco sobre un elemento de una sección concreta.

Encabezados

[WCAG 1.3.1](#) | [Headings \(WAI Tutorials\)](#)

Una vez que hemos delimitado las diferentes áreas de una página mediante secciones, debemos estructurar los contenidos de cada sección mediante encabezados.

Puede haber secciones, como la sección de navegación, que no vayan a necesitar dicha estructura. Pero secciones, como la sección principal, deben incorporarlas:

```
<main>
  <h1>Daily Planet</h1>
```

```

<h2>Internacional</h2>
<h3>Wonder Woman visita el Partenón en Atenas</h3>
<!-- Contenido Noticia -->
<h3>La estación espacial recibe a Superman</h3>
<!-- Contenido Noticia -->
<!-- etc. -->

<h2>Nacional</h2>
<h3>Resultados de las elecciones en Metrópolis</h3>
<!-- Contenido Noticia -->
<!-- etc. -->
</main>

```

Ten en cuenta que:

- Una sección sólo debe incluir un encabezado de primer nivel: ej. “Daily Planet”.
- No se deben saltar niveles en los encabezados: ej. “Wonder Woman visita el Partenón en Atenas” es una noticia de nivel 3, ubicada en la sección “Internacional” de nivel 2, del periódico “Daily Planet” de nivel 1.
- Los encabezados deben tener contenido (ej. las noticias de nivel 3) o preceder a otros encabezados de nivel inferior (ej. las secciones “Internacional” y “Nacional” de nivel 2 que preceden a las noticias).

En el siguiente ejemplo (figura 5), se muestra la extensión [HeadingMap](#), que construye un sistema de navegación mediante marcadores, en base a los encabezados que encuentra en una página:

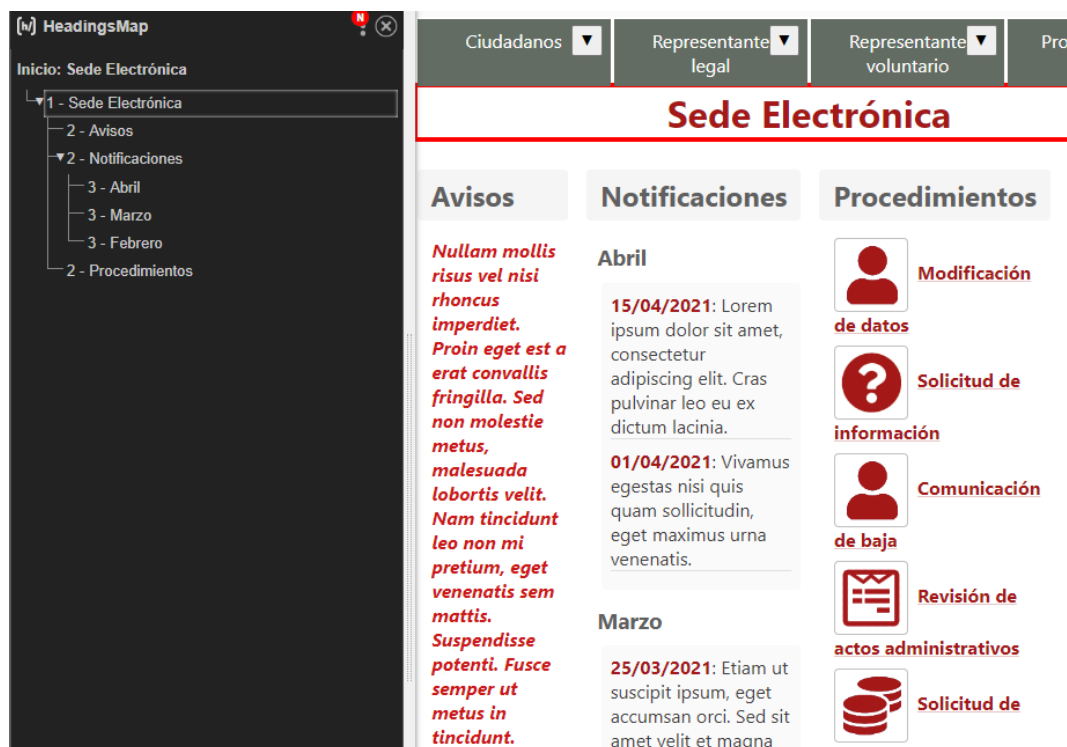


Figura 5

¿Por qué esto es importante? Porque permite una identificación clara de las diferentes partes que conforman un contenido. Y además proporciona, a las tecnologías de asistencia a la accesibilidad, una estructura en base a la cual puedan diseñar mecanismos de navegación basados en los marcadores de los encabezados.

Párrafos

[WCAG 1.3.1](#) | [Content Structure \(WAI Tutorials\)](#)

Un párrafo es un bloque coherente de texto conformado por una o más oraciones. Emplea la marca **<p>** para identificarlos y especifica los cambios de idioma que se puedan producir (con respecto al idioma principal indicado en la marca **<html>**) mediante el empleo del atributo **lang**:

```
<p>La frase <span lang="en"><em>"there is shadow hanging over me"</em></span> significa <em>"una sombra se cierne sobre mí"</em>.</p>
```

Visualmente podríamos obtener el mismo efecto con otro tipo de marcado, por ejemplo, mediante el uso de **<div>**. Pero de lo que se trata es de identificar adecuadamente un tipo de contenido específico. Y no es lo mismo, una marca blanca de bloque *sin carga semántica* como **<div>**, que el empleo de la marca **<p>**, que especifica que la información acotada por ella es un párrafo.

Además, HTML no es un lenguaje sólo pensado para su visualización: puede ser también verbalizado mediante un lector de pantalla o transcrito mediante un dispositivo Braille.

¿Por qué esto es importante? Para mejorar la legibilidad sobre el texto y proporcionar mayor control a los usuarios sobre la presentación de la información.

Listas

[WCAG 1.3.1](#) | [Content Structure \(WAI Tutorials\)](#)

Las listas nos permiten, por una parte, la descripción de enumeraciones desordenadas y ordenadas de elementos, pasos, ideas, etc. Y por otra, la definición de términos. Emplea el marcado adecuado a cada una de ellas:

```
<!-- Lista desordenada -->
<ul>
  <li>Teléfono inteligente.</li>
  <li>Tableta.</li>
  <li>Equipo de escritorio.</li>
</ul>
```

```

<!-- Lista ordenada -->
<ol>
  <li>Introduzca la tarjeta en el conector del teclado.</li>
  <li>Cumplimente en la ventana emergente la contraseña.</li>
  <li>Acepte la operación mediante el botón <em>Aceptar</em> del
cuadro de diálogo.</li>
</ol>

<!-- Lista de definición -->
<dl>
  <dt>Certificado electrónico.</dt>
  <dd>Fichero informático firmado electrónicamente por un
prestador de servicios de certificación.</dd>
  <dt>Página web.</dt>
  <dd>Fichero informático estructurado mediante HTML.</dd>
</dl>

```

¿Por qué esto es importante? Para especificar una semántica adecuada al contenido que se pretende comunicar: colecciones de elementos en los cuales el orden no es relevante; secuencias de instrucciones o pasos en los cuales el orden si es relevante; y finalmente, definiciones de términos.

Marcas estructurales vs. marcas de formato

WCAG 1.3.1

Hemos recalcado varias veces, que la finalidad de una marca es *identificar un tipo de contenido*. Sin embargo, la ausencia de un lenguaje de presentación (como CSS) en los orígenes de HTML, trajo como resultado la introducción de algunas marcas (como **** e **<i>**), cuyo propósito es meramente el de aplicar un formato específico a un texto.

Puesto que hoy en día, CSS nos permite separar la presentación del contenido, debemos emplear sólo marcas estructurales (como **** y ****) y evitar el uso de marcas que sólo aporten un mero formato visual a un contenido:

```

<p>El <strong>Real Decreto 1112/2018</strong> <em>de
accesibilidad de los sitios web y aplicaciones para dispositivos
móviles</em>, tiene como objeto <em>"garantizar los requisitos
de accesibilidad de los sitios web y aplicaciones para
dispositivos móviles de organismos del sector público y otros
obligados"</em>.</p>

```

Nota: si bien, visualmente **** y **** no se diferencian de las marcas **** e **<i>**, mientras que las primeras identifican términos y frases que deben ser resaltados o enfatizados (con independencia de que su representación sea visual o no), las segundas se limitan a aplicar un formato.

¿Por qué esto es importante? Porque mientras que marcas como **** e **<i>** se limitan a aportar una mera presentación gráfica, marcas como **** o **** especifican que sus términos deben ser resaltados o enfatizados, con independencia de cuál sea la presentación por la cual haya optado el usuario: visual, verbal, braille, etc.

Enlaces

WCAG 2.4.4

Además de ser un lenguaje de marcado, HTML es también un lenguaje de hipertexto. El hipertexto es una estructura documental en la cual los contenidos de unos documentos se relacionan/referencian con los contenidos de otros documentos mediante hipervínculos o enlaces.

Los textos empleados en los enlaces deben ser lo suficientemente descriptivos, como para que no haya lugar a dudas de cuál es su propósito:

```
<!-- Enlace incorrecto -->
<a href="https://www.boe.es/diario_boe/">Pulse aquí</a>

<!-- Enlace correcto -->
<a href="https://www.boe.es/diario_boe/">Boletín Oficial del
Estado</a>
```

Realiza el siguiente ejercicio *antropológico*. Mirando a la siguiente imagen (figura 6), imagina que escuchas la frase “pulsa aquí”:



Figura 6

¿Por qué esto es importante? Para que la *finalidad* del enlace pueda ser determinada tan solamente por lo que *comunica su texto*, y no, por un *contexto* que puede encontrarse *no disponible* para algunos grupos de usuarios.

Enlaces Gráficos

Si el enlace emplea una imagen, ésta deberá incluir un texto alternativo mediante su atributo **alt**, mediante el cual el propósito del enlace pueda ser comunicado a una tecnología de asistencia a la accesibilidad:

```
<a href="https://www.boe.es/diario_boe/"></a>
```

Si el enlace incluye tanto texto como una imagen, ambos textos deberán ser diferentes, a la hora de evitar redundancias y repeticiones que los lectores de pantalla leerían:

```
<a href="https://www.boe.es/diario_boe/"> Boletín Oficial del Estado</a>
```

¿Por qué esto es importante? Para que las tecnologías de asistencia a la accesibilidad sean capaces de: interpretar los literales de los enlaces; transcribirlos a braille; y ejecutar los enlaces mediante comandos de voz.

Imágenes

[WCAG 1.1.1](#) | [Images \(WAI Tutorials\)](#)

Las imágenes, y otros tipos de componentes, deben ofrecer mecanismos alternativos que suministren, en lo posible, la misma información a aquellas personas que puedan tener dificultad para verlas, comprenderlas o inclusive cargarlas.

En el siguiente ejemplo, incluimos un texto alternativo mediante el empleo del atributo **alt**:

```

```

Si la imagen es meramente decorativa no será necesario comunicar su propósito a una tecnología de asistencia a la accesibilidad, por lo cual, bastará con asignar la cadena vacía al atributo **alt**:

```

```

Si la imagen no aporta información adicional al texto que la precede o sucede (por ejemplo, en un enlace), también la podemos considerar como decorativa:

```
<a href="https://www.boe.es/diario_boe/"> Boletín Oficial del Estado</a>
```

En el siguiente ejemplo (figura 7), la imagen que precede al texto del enlace *Solicitud de información*, no necesita comunicar información adicional a una tecnología de asistencia (por lo cual, basta con asignarle la cadena vacía):



Figura 7

Si la complejidad expresada por la imagen es difícilmente expresable por su texto alternativo (piénsese, por ejemplo, en un gráfico o en un diagrama), tendremos que aportar (además de un texto alternativo) una descripción por separado de la imagen:

```

<p id="descripción">El gráfico muestras las ventas trimestrales que se han producido en el año en cursos en los diferentes países que conforman la Unión Europea. etc.</p>
```

¿Por qué esto es importante? Al igual que con los enlaces, para que las tecnologías de asistencia a la accesibilidad sean capaces de interpretar, transcribir a braille, etc. los textos alternativos de las imágenes.

SVG

Si incluyes gráficos vectoriales mediante SVG ([Scalable Vector Graphics](#)), identifícalos como imágenes, mediante el atributo **role** de WAI-ARIA, y añádeles un texto alternativo mediante el atributo **aria-label** de WAI-ARIA:

```
<svg role="img" aria-label="Texto alternativo">
  <!-- Marcado -->
</svg>
```

Nota: puedes consultar más información en la página [ARIA: img role](#) de MDN Plus.

Font Awesome

Si incluyes iconos de [Font Awesome](#), oculta los iconos a las tecnologías de asistencia mediante el atributo **aria-hidden** de WAI-ARIA y proporciónales una descripción

alternativa, mediante algún elemento de HTML, cuya clase no muestre en pantalla, dicho texto alternativo:

```
<i class="fas fa-camera-retro" aria-hidden="true" title="Texto alternativo"></i>  
<span class="screenreader">Texto alternativo para lectores de pantalla</span>
```

Nota: puedes consultar más información en la página de [Accesibilidad](#) de Font Awesome.

Tablas

[WCAG 1.3.1](#) | [Tables \(WAI Tutorials\)](#)

Si bien es posible emplear las tablas con fines de diseño, tal uso es igual de desaconsejable que el empleo de las marcas de formato, en lugar de las marcas estructurales.

En lo posible, debemos emplear las tablas para presentar la información categorizada mediante encabezados:

```
<table>  
  <caption>Ventas mensuales</caption>  
  <tr>  
    <!-- Encabezados -->  
    <th>Enero</th>  
    <th>Febrero</th>  
    <!-- etc. -->  
  </tr>  
  <tr>  
    <!-- Datos -->  
    <td>34.000,00€</td>  
    <td>25.000,00€</td>  
    <!-- etc. -->  
  </tr>  
  <!-- etc. -->  
</table>
```

Si bien no es obligatorio, es conveniente especificar un título, mediante la marca **<caption>**, que identifique el propósito de la tabla.

En el siguiente ejemplo (figura 8), el texto “*Acceso, formularios e información sobre los procedimientos de la sede electrónica*”, es el título que se ha asignado a la tabla mediante la marca **<caption>**:

Acceso, formularios e información sobre los procedimientos de la sede electrónica

Procedimiento	Formulario	Información	Descripción	Versión
Modificación de datos	Formulario	Información	Vestibulum a nisl vel nisl ultricies dapibus.	v1.0.
Solicitud de información	Formulario	Información	Vestibulum a nisl vel nisl ultricies dapibus. Phasellus laoreet eget nisl ac ultrices.	v1.0.
Comunicación de baja	Formulario	Información	Vestibulum a nisl vel nisl ultricies dapibus. Phasellus laoreet eget nisl ac ultrices.	v1.1.
Revisión de actos administrativos	Formulario	Información	Vestibulum a nisl vel nisl ultricies dapibus. Sed tempus tellus ac interdum egestas.	v1.0.
Solicitud de ayuda económica	Formulario	Información	Vestibulum a nisl vel nisl ultricies dapibus. Sed tempus tellus ac interdum egestas.	v2.0.
Recurso de reposición	Formulario	Información	Vestibulum a nisl vel nisl ultricies dapibus.	v1.0.
Solicitud de cobro	Formulario	Información	Vestibulum a nisl vel nisl ultricies dapibus. Sed tempus tellus ac interdum egestas.	v1.2.

Figura 8

¿Por qué esto es importante? Para que las tecnologías de asistencia a la accesibilidad sean capaces de identificar las tablas de una página y puedan comunicarle al usuario, mediante el título de la tabla, una idea de su contenido, de manera que este pueda decidir si está interesado, o no, en su lectura.

Tablas multinivel

Si la tabla incluye múltiples niveles de encabezado es necesario identificarlos mediante el atributo **id** y referenciarlos desde cada celda mediante el atributo **headers**, a la hora de que las tecnologías de asistencia a la accesibilidad sean capaces de diferenciarlos unívocamente:

```
<p id="descripción">El encabezado de la primera columna muestra el horario. El resto de encabezados los días de la semana.</p>
<table aria-describedby="descripción">
  <caption>Programación</caption>
  <tr>
    <td></td>
    <th id="d1">Lunes</th>
    <th id="d2">Martes</th>
    <!-- etc. -->
  </tr>
  <tr>
    <th id="t1">8:00 - 9:00</th>
    <td headers="d1 t1">Matemáticas</td>
    <td headers="d2 t1">Lengua</td>
    <!-- etc. -->
  </tr>
  <tr>
    <th id="t2">9:00 - 10:00</th>
    <td headers="d1 t2">Historia</td>
    <td headers="d2 t2">Matemáticas</td>
    <!-- etc. -->
  </tr>
</table>
```

```
</tr>
<!-- etc. -->
</table>
```

Nota: en las tablas con múltiples encabezados es conveniente describir su función, a la hora de facilitar su navegación: [Caption & Summary](#).

¿Por qué esto es importante? Porque permite a las tecnologías de asistencia a la accesibilidad, como los lectores de pantalla, identificar para cada celda de datos sus correspondientes celdas de encabezado. Por ejemplo, en la tabla del ejemplo anterior un lector de pantalla será capaz de determinar que el lunes (identificado como d1) de 8:00 a 9:00 (identificado como t1) se impartirá Matemáticas.

Formularios

[WCAG 1.3.1](#) | [WCAG2.5.3](#) | [WCAG 3.3.1](#) | [WCAG 3.3.2](#) | [WCAG 3.3.3](#) | [Forms \(WAI Tutorials\)](#)

Los formularios en HTML nos permiten recabar información del usuario, para su posterior procesamiento, mediante una serie de controles: campos de texto, casillas de verificación, botones de opción, listas desplegables, etc.

Al igual que hemos organizado los contenidos de las secciones mediante encabezados, debemos organizar los controles de los formularios en áreas específicas mediante el empleo de la marca **<fieldset>**:

```
<form>
  <fieldset>
    <legend>Datos Personales</legend>
    <!-- Controles del apartado "Datos Personales" -->
  </fieldset>
  <fieldset>
    <legend>Consulta</legend>
    <!-- Controles del apartado "Consulta" -->
  </fieldset>
  <!-- etc. -->
</form>
```

En el siguiente ejemplo (figura 9), se muestran las áreas del formulario con sus correspondientes leyendas:

Figura 9

¿Por qué esto es importante? Porque la agrupación de los controles facilita al usuario su comprensión y le ayuda a su correcta cumplimentación.

Etiquetado

De forma análoga a las imágenes, los controles de los formularios deberán ser correctamente etiquetados, a fin de que los usuarios puedan reconocer fácilmente su propósito y las tecnologías de asistencia a la accesibilidad puedan interactuar con ellos.

Para asociar un campo de texto con su etiqueta, identificaremos el campo de texto mediante el atributo **id** y lo referenciaremos mediante el atributo **for** de la marca **<label>** (figura 10):

```
<label for="Nombre">Nombre</label>
<input id="Nombre" type="text" name="Nombre" />
```

Figura 10

Seguiremos la misma estrategia con los botones de opción y las casillas de verificación, en los cuales identificaremos, además, su propósito general mediante el empleo de **<fieldset>** y **<legend>**:

```
<!-- Botones de opción -->
<fieldset>
  <legend>Sexo</legend>
  <input id="Mujer" type="radio" name="Sexo" value="Mujer" />
  <label for="Mujer">Mujer</label>
  <input id="Hombre" type="radio" name="Sexo" value="Hombre" />
  <label for="Hombre">Hombre</label>
</fieldset>

<!-- Casillas de verificación -->
<fieldset>
  <legend>Seleccione las semanas en las que se encuentra
  interesado</legend>
  <input id="Semana1" type="checkbox" name="Semana1"
  value="Semana1" />
  <label for="Semana1">Semana1</label>
  <input id="Semana2" type="checkbox" name="Semana2"
  value="Semana2" />
  <label for="Semana2">Semana2</label>
</fieldset>
```

Las listas desplegables también emplearán el etiquetado mediante **<label>**. Si sus valores pueden ser agrupados es conveniente hacerlo mediante la marca **<optgroup>**, a la hora de facilitar su identificación y selección:

```
<label for="Dia">Seleccione un día de la semana</label>
<select id="Dia" name="Dia">
  <option value="">Sin seleccionar</option>
  <optgroup label="Laborales">
    <option value="Lunes">Lunes</option>
    <option value="Martes">Martes</option>
    <option value="Miércoles">Miércoles</option>
    <option value="Jueves">Jueves</option>
    <option value="Viernes">Viernes</option>
  </optgroup>
  <optgroup label="Festivos">
    <option value="Sabado">Sabado</option>
    <option value="Domingo">Domingo</option>
  </optgroup>
</select>
```

Finalmente, en el siguiente ejemplo se muestra el etiquetado de un campo de texto, en el cual incluimos, mediante el atributo **placeholder**, un texto por defecto:

```
<label for="Comentario">Comentario</label>
<textarea id="Comentario" name="Comentario" placeholder="Escriba
aquí su comentario" rows="6"></textarea>
```

¿Por qué esto es importante? Para que las tecnologías de asistencia a la accesibilidad sean capaces de identificar los controles de los formularios mediante sus etiquetas asociadas. Por ejemplo, las personas que navegan por comandos de voz pueden activar un control de un formulario mediante su etiqueta a la hora de cumplimentarlo.

Los formularios disponen, además, de otros tipos de controles que les permiten: el envío de los datos y el restablecimiento del formulario a sus valores originales.

Los controles típicos que se suelen emplear son los controles **<input>** de tipo **submit** (envío) y **reset** (reestablecimiento). En este caso los valores del atributo **value** funcionan como etiquetas de dichos controles:

```
<input type="submit" value="Enviar" />
<input type="reset" value="Cancelar" />
```

Lo mismo ocurre si empleamos un control **<input>** de tipo **image**. Sólo que en este caso es el atributo **alt** el que sustituye al etiquetado de **<label>**:

```
<input type="image" src="url" alt="Enviar" />
```

Si empleamos botones, el propio texto de los botones será empleado como su etiqueta:

```
<button type="submit">Enviar</button>
<button type="reset">Cancelar</button>
```

En ocasiones el propósito de un control puede estar lo suficientemente claro en base a su contexto. Por ejemplo, en el siguiente ejemplo el botón **Buscar** funciona *visualmente* como etiquetado del campo de texto que le precede:

```
<input type="text" name="buscar">
<button type="submit">Buscar</button>
```

El problema es que las tecnologías de asistencia a la accesibilidad no son capaces de establecer dicha relación. Una de las soluciones posibles es el uso del atributo **aria-label** de WAI-ARIA, mediante el cual se pueden etiquetar elementos interactivos que carecen de un etiquetado visual: [Labeling Controls](#).

```
<input type="text" name="buscar" aria-label="Buscar" />
<button type="submit">Buscar</button>
```


Nota: el criterio de accesibilidad 2.5.3. nos dice que en estos casos el *nombre accesible* (el valor del atributo **aria-label**, en nuestro ejemplo: **Buscar**) debe ser igual o contener al texto empleado como **etiquetado visible** (en nuestro ejemplo, el texto del botón **Buscar**).

¿Por qué esto es importante? Porque evita las repeticiones visuales innecesarias de información, sin dejar de comunicar el propósito de los controles a las tecnologías de asistencia a la accesibilidad.

Una vez estructurado un formulario y etiquetado sus controles debemos tener en cuenta las siguientes cuestiones.

Instrucciones

Proporcionar al usuario **instrucciones** que le ayuden a cumplimentar el formulario (figura 11):

```
<label for="DNI">DNI</label>
<input id="DNI" type="text" name="DNI" aria-
describedby="formatodni" />
<span id="formatodni">Formato DNI: 12345678A</span>
```

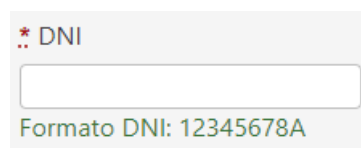


Figura 11

¿Por qué esto es importante? Porque las personas con discapacidades cognitivas y la gente mayor, pueden tener problemas intentando comprender la información que se les está solicitando.

Validaciones

Efectuar **validaciones** que informen a los usuarios de aquellos controles del formulario que son obligatorios (figura 12):

```
<label for="Nombre"><abbr title="campo obligatorio">*</abbr>
Nombre</label>
<input id="Nombre" type="text" name="Nombre" required="required"
/>
```

* Nombre

* Apellidos

Completa este campo

Figura 12

También es conveniente efectuar validaciones de aquellos campos que puedan estar sujetos a algún tipo concreto de formato (figura 13):

```
<label for="DNI">DNI</label>
<input id="DNI" type="text" name="DNI" aria-
describedby="formatodni" pattern="\d{8}[A-Z]{1}" />
<span id="formatodni">Formato DNI: 12345678A</span>
```

* DNI

AABBCQ

Utiliza un formato que coincida con el solicitado

Figura 13

¿Por qué esto es importante? Porque advierte al usuario sobre los controles obligatorios del formulario y le proporciona información sobre los posibles errores que pueda cometer.

Notificaciones

Proporcionar una **notificación** (habitualmente detrás del formulario), sobre el correcto o incorrecto envío del formulario al servidor, de manera que el usuario tenga constancia de si se ha producido o no un error en el envío:

```
<!-- Mensaje de envío correcto -->
<p class="StatusOK">El formulario ha sido enviado con éxito.</p>

<!-- Mensaje de envío incorrecto -->
<p class="StatusERROR">Se ha producido un error en el envío del
formulario.</p>
```

¿Por qué esto es importante? Porque la retroalimentación, el *feedback*, proporciona al usuario la constatación de que el proceso de envío ha sido realizado, o no, con éxito.

Información personal

Finalmente, el criterio 1.3.5. requiere la identificación del propósito de cada campo de texto que solicite información personal sobre el usuario mediante la utilización del atributo **autocomplete** y un valor que identifique dicho tipo de información (figura 14):

```
<label for="Nombre"><abbr title="campo obligatorio">*</abbr>
Nombre</label>
<input id="Nombre" type="text" name="Nombre" required="required"
autocomplete="given-name" />
```

Figura 14

Aquí tienes alguno de los valores más frecuentemente usados:

- **Nombre:** given-name
- **Apellidos:** family-name
- **Puesto de trabajo:** organization-title
- **Empresa:** organization
- **Sexo:** sex
- **Email:** email

Puedes consultar el resto de valores en el epígrafe [7. Input Purposes for User Interface Components](#).

¿Por qué esto es importante? Porque una vez que dichos valores hayan sido introducidos, al menos una vez, los navegadores serán capaces de cumplimentarlos automáticamente por el usuario. Ayudando así, a aquellas personas que puedan tener dificultades a la hora de interpretar alguna de las etiquetas empleadas en los campos de texto de los formularios.

Multimedia

Los contenidos audiovisuales o multimedia representan una barrera para muchos usuarios que no pueden escucharlos, visualizarlos o entenderlos (algunas personas procesan mejor la información escrita que audiovisual).

A la hora de facilitar el acceso a los contenidos multimedia debemos emplear algunas de las siguientes opciones:

- **Subtítulos:** en un video, los subtítulos constituyen una versión textual del audio del video en su lengua original (no confundir con los subtítulos que se emplean

para realizar traducciones a otros idiomas) y de la información visual no hablada como, por ejemplo: “un pájaro sobrevuela la ciudad” o “una mujer se sienta en un banco del parque”.

- **Descripción de audio:** en un video, las descripciones de audio aportan, mediante un segundo canal de audio, descripciones sobre aquellas partes del video en las cuales por lo general no se habla, pero suceden ciertos acontecimientos que deben ser comunicados al usuario: ej. “entra en escena un hombre que se dirige hacia el director de orquesta para saludarlo”.
- **Transcripciones:** una transcripción de un audio o un video es una representación textual, tanto de la información hablada como no hablada, que normalmente suele suministrarse mediante marcado HTML a parte del audio o del video. Su contenido es equivalente al ofrecido mediante los subtítulos. Si bien, mientras que en el caso de los subtítulos sólo podemos leer o escuchar el fragmento correspondiente al punto del video en el que nos encontremos, en el caso de las transcripciones tenemos a nuestra disposición todo el contenido hablado y no hablado del audio o del video.

Resumiendo:

- Debemos incluir subtítulos con los videos ([WCAG 1.2.2](#)) y además, si fuera necesario, descripciones de audio ([WCAG 1.2.3](#) y [WCAG 1.2.5](#)).
- Debemos incluir transcripciones tanto con los videos como con los audios ([WCAG 1.2.1](#)).

En el siguiente ejemplo se muestra el marcado para incluir un fichero de video y un fichero de audio:

```
<!-- Video -->
<video controls="controls">
  <source src="fichero.mp4" type="video/mp4" />
  <source src="fichero.webm" type="video/webm" />
</video>

<!-- Audio -->
<audio controls="controls">
  <source src="fichero.mp3" type="audio/mpeg" />
  <source src="fichero.wav" type="audio/wav" />
</audio>
```

Los subtítulos y las descripciones de audio se incluyen mediante la marca **<track>**. Suelen emplear el formato [WebVTT](#) (The Web Video Text Tracks Format).

En el siguiente ejemplo, incorporamos un subtítulo y una descripción de audio a un video mediante **<track>**:

```
<video controls="controls">
  <source src="fichero.mp4" type="video/mp4" />
  <source src="fichero.webm" type="video/webm" />
```

```

<!-- Subtítulos y descripciones de audio -->
<track src="fichero.vtt" kind="captions" srclang="es"
label="español" />
<track src="fichero.vtt" kind="descriptions" srclang="es" />
</video>

```

Nota: las descripciones de audio mediante la marca **<track>** no se encuentran actualmente soportadas por los reproductores por defecto de los navegadores.

Las transcripciones deben incluirse, a parte, mediante marcado convencional:

```

<p><strong>Alicia:</strong> "¿Me podrías indicar, por favor,
hacia dónde tengo que ir desde aquí?"</p>
<p><strong>Gato:</strong> "Eso depende de a dónde quieras
llegar."</p>
<p><strong>Alicia:</strong> "A mí no me importa demasiado a
dónde..."</p>
<!-- etc. -->

```

O alternativamente mediante el uso de algún reproductor multimedia, como [Able Player](#), capaz de mostrar y resaltar los textos de las transcripciones a medida que estos se pronuncian en los contenidos multimedia correspondientes.

En la siguiente imagen (figura 15), se muestra un ejemplo de transcripción sincronizada en [YouTube](#):

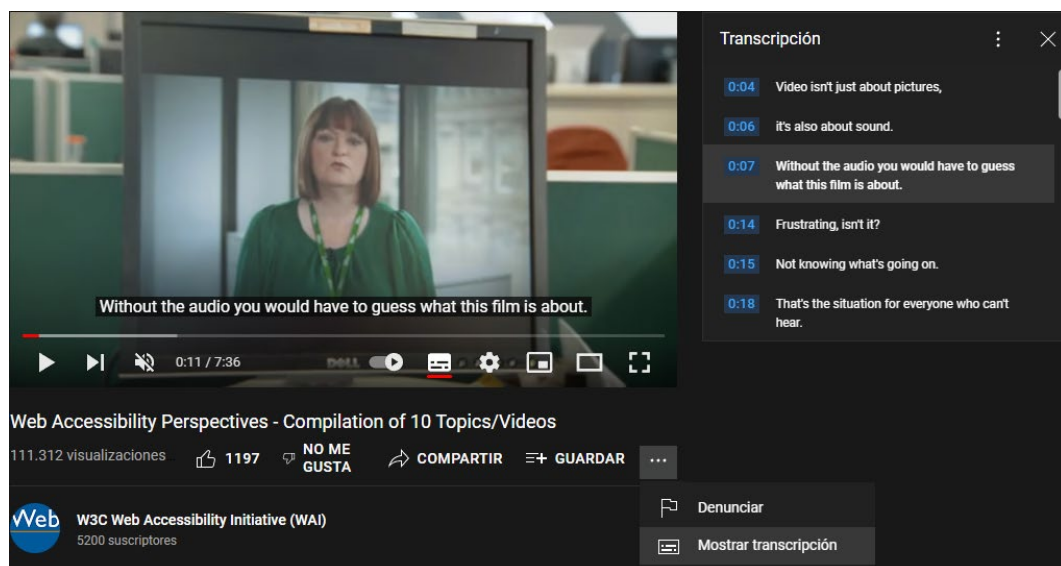


Figura 15

¿Por qué esto es importante? Para que los contenidos audiovisuales no constituyan una barrera en la navegación para aquellas personas que no puedan escucharlos, visualizarlos o comprenderlos.

El objetivo de CSS es proporcionar una o más **presentaciones** a un documento HTML. Si HTML constituyen los cimientos de una página web, CSS constituye su acabado final: pintura, molduras, tipos de materiales empleados, etc.

Separación de contenido y presentación

WCAG 1.3.1

La separación del contenido de su presentación, nos permite un tratamiento más eficiente del contenido, al encontrarse este último claramente diferenciado de su presentación: ej. no es lo mismo una fecha, que su presentación en formato anglosajón.

CSS es un *lenguaje de presentación*, que nos permite presentar el contenido previamente estructurado mediante el marcado de HTML. Por ejemplo, la siguiente regla aplica un color azul a los encabezados **<h1>** (figura 16):

```
h1 {  
  color:rgb(0,0,255);  
}
```

Intranet Administrativa

Figura 16

Ahora bien, al igual que ocurre con HTML, hay ciertos usos de CSS que, si bien sintácticamente son correctos, no lo son desde el punto de vista de la finalidad del lenguaje.

Observa la definición de la siguiente regla, en ella se define una clase que, aplicada por ejemplo a un párrafo, podría asimilarse visualmente a la anterior:

```
.encabezado1 {  
  color:rgb(0,0,255);  
  
  /* Propiedades predeterminadas de h1 */  
  display: block;  
  font-size: 2em;  
  margin-top: 0.67em;  
  margin-bottom: 0.67em;  
  margin-left: 0;  
  margin-right: 0;  
  font-weight: bold;  
}
```

Ahora bien, mientras que la primera regla aplica un estilo, una presentación, a un encabezado, a un contenido previamente correctamente estructurado. La segunda regla se limita a *simular visualmente* el aspecto de un encabezado mediante la aplicación de una clase a una marca, que no tiene por qué ser un encabezado (figura 17).

¿Soy un encabezado? ¿O soy un párrafo?

Figura 17

Si una página organiza sus contenidos mediante encabezados, una tecnología de asistencia a la accesibilidad puede proporcionar, a partir de ellos, un sistema de navegación mediante marcadores (recuerda el ejemplo con la extensión [HeadingMap](#), que vimos en el epígrafe de *Encabezados*). Si la página se limita a simularlos mediante estilos, las tecnologías de asistencia no podrán proporcionar dicha funcionalidad.

En resumen, CSS debe emplearse para aplicar estilos a contenidos que se encuentren previamente estructurados de una forma correcta, no para simular estructuras inexistentes.

¿Por qué esto es importante? Para que las tecnologías de asistencia a la accesibilidad reconozcan las diferentes partes que conforman una página (secciones, encabezados, párrafos, etc.) y puedan asistir así, al usuario en la navegación.

Tipos de hojas de estilo

Así mismo, puesto que la finalidad de CSS es la separación de la presentación del contenido, se recomienda emplear sólo hojas de estilo vinculadas o globales y evitar el uso de hojas de estilo incrustadas que sintácticamente se encuentran entremezcladas con el código HTML de la página:

```
<!-- Las hojas de estilo vinculadas se pueden aplicar a un
conjunto de páginas web -->
<link rel="stylesheet" type="text/css" href="fichero.css">

<!-- Las hojas de estilo globales se aplican sólo a la página en
la que se encuentran -->
<head>
  <style type="text/css">
    /* Reglas CSS */
  </style>
</head>

<!-- Las hojas de estilo incrustadas sólo afectan a una marca
concreta de una página. Se desaconseja su uso. -->
<p style="color:rgb(255,0,0);">¡Alerta!</p>
```

¿Por qué esto es importante? Para que la separación entre contenido y presentación sean lo más claras posibles. Es mucho más sencillo modificar la presentación de una web diseñada con hojas de estilos vinculadas y globales que con hojas de estilo en línea.

Diseño adaptativo

WCAG 1.4.10.

Durante mucho tiempo el dispositivo de salida principal para la visualización de las páginas web lo constituyó el monitor. Pero hoy en día su visualización se puede llevar a cabo en un gran número de dispositivos y resoluciones: teléfonos inteligentes, tabletas, equipos de sobremesa, etc.

En la siguiente imagen (figura 18), se muestra una página en su versión para equipo de sobremesa y versión para teléfono inteligente:

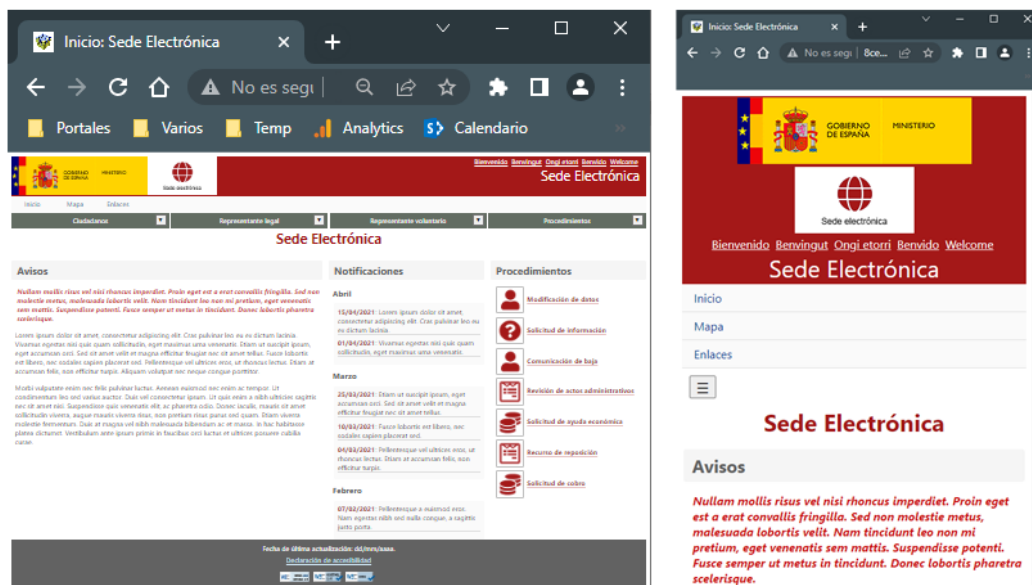


Figura 18

Con independencia del dispositivo y resolución en el que se muestre una página, el diseño adaptativo tiene como finalidad la conservación de la funcionalidad de la página, evitando, además, tanto la pérdida de información como los desplazamientos horizontales.

El diseño adaptativo se fundamenta en base a las siguientes tecnologías de CSS:

- **Media Queries:** expresiones booleanas que nos permiten definir reglas de hojas estilo en cascada que sólo se aplicarán en el caso de que la expresión sea evaluada como verdadera.
- **Grid:** modelo que nos permite definir un contenedor compuesto por un número determinado de filas y columnas.

- **Flexbox:** modelo que nos permite definir un contenedor compuesto por un número indeterminado de elementos, los cuales son capaces de ajustar *flexiblemente* sus dimensiones en una disposición horizontal o vertical dentro del contenedor.

Marco de trabajo

En lo que sigue, se va a proponer un posible marco de trabajo, a la hora de obtener un diseño adaptativo, a partir de la siguiente estructura:

```
<!-- Grid SmartPhone -->
<div id="gridSP">
  <header><!-- Encabezado --></header>
  <nav aria-label="Navegación secundaria"><!-- Navegación --
></nav>
  <nav aria-label="Navegación principal"><!-- Navegación --
></nav>
  <main>
    <h1>Lorem ipsum dolor sit amet</h1>
    <!-- Grid Tablet -->
    <div id="gridT">
      <div>
        <!-- Contenido -->
      </div>
      <div>
        <!-- Contenido -->
      </div>
    </div>
  </main>
  <footer><!-- Pie --></footer>
</div>
```

Nota: podríamos pensar en emplear `<section>` en lugar de `<div>`. Pero recuerda, que el propósito de `<section>` es identificar una sección de contenido en general, mientras que `<div>` carece de carga semántica. Por eso `<div>` es la marca *ideal* a emplear, cuando el propósito del marcado es meramente aplicar un estilo específico.

Reglas para teléfonos inteligentes

Primeramente, empezaremos definiendo las reglas para teléfonos inteligentes, que servirán también como reglas base para otros tipos de dispositivos. Y a continuación, mediante **Media Queries** definiremos las reglas que se aplicarán a otros tipos de dispositivos que empleen resoluciones mayores:

```
/* Reglas para teléfonos inteligentes */
/* Reglas CSS */
```

```

/* Reglas para tabletas */
@media only screen and (min-width: 768px) {
  /* Reglas CSS */
}

/* Reglas para equipos de escritorio */
@media only screen and (min-width: 992px) {
  /* Reglas CSS */
}

```

Nota: algunos autores, fuerzan la orientación del *viewport* a horizontal (**orientation:landscape**), puesto que es la opción empleada por muchos usuarios cuando navegan con una tableta u otro tipo de dispositivos. Sin embargo, debemos evitarlo, puesto que algunos usuarios emplean para navegar dispositivos fijos que no permiten ser rotados: [WCAG 1.3.4 Orientation](#).

Es importante reseñar que muchos teléfonos inteligentes modernos superan los 768 píxeles empleados en la consulta de medios anterior. Por lo cual podría pensarse que las reglas definidas para tabletas también les van a afectar. Pero las reglas no se evalúan teniendo en cuenta los píxeles físicos de los dispositivos, sino los píxeles lógicos de CSS. Según la densidad de la pantalla de un dispositivo, un pixel lógico de CSS, puede equivaler a dos, tres o cuatro píxeles reales del dispositivo. Por lo cual, un dispositivo, con una resolución de 1400 x 3200 píxeles (reales) y una densidad cercana a 4 píxeles, obtiene un *viewport* de 360 x 800 píxeles (CSS): [Responsive Web Design – Media Queries](#).

Grid

A continuación, definiremos los *layouts* que se van a emplear para teléfonos inteligentes y para otro tipo de dispositivos.

Para teléfonos inteligentes definiremos mediante Grid, una *estantería* de una columna por cinco filas. En cada una de estas filas (baldas de la columna), se situarán cada una de las secciones de nuestro ejemplo (**header**, **nav**, **nav**, **main** y **footer**):

```

/* Reglas para teléfonos inteligentes */
* {
  box-sizing: border-box; /* width = border+padding+content */
}

#gridSP {
  display:grid;
  grid-template-columns:auto;
  grid-template-rows:auto auto auto 1fr auto;
}

header {

```

```

    /* Propiedades CSS */
  }
  nav:nth-of-type(1) {
    /* Propiedades CSS */
  }
  nav:nth-of-type(2) {
    /* Propiedades CSS */
  }
  main {
    /* Propiedades CSS */
  }
  footer {
    /* Propiedades CSS */
  }
}

```

Para tabletas, y dispositivos con resoluciones mayores, definiremos, dentro de la balda donde se encuentra la sección principal, una nueva estantería de dos columnas por una fila:

```

/* Reglas para tabletas */
@media only screen and (min-width: 768px) {
  main #gridT {
    display:grid;
    grid-template-columns:0.6fr 0.4fr;
    grid-template-rows:auto;
    grid-column-gap:1em;
  }
}

```

De esta manera habremos conseguido, para teléfonos inteligentes, un diseño de una columna por cinco filas (una para cada sección). Y para tabletas, un diseño en el cual la balda de la sección principal se divide en dos columnas (figura 19).

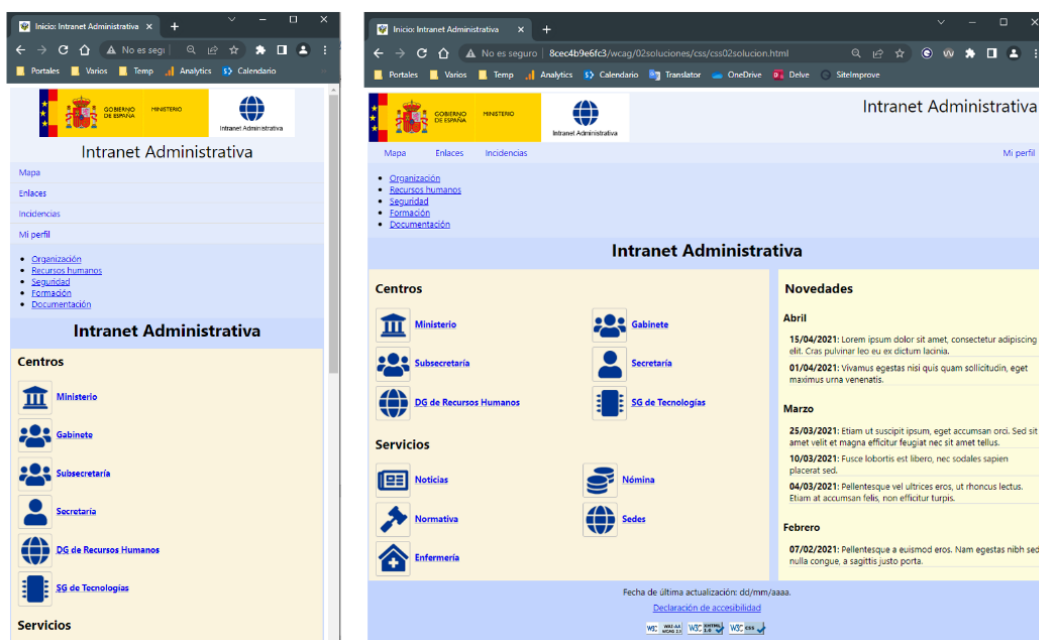


Figura 19

Si no definimos reglas, dentro de otras **Media Queries**, para dispositivos con resoluciones mayores a 768 píxeles, dichos dispositivos también emplearían el diseño especificado para las tabletas.

Nota: puedes leer más sobre el modelo Grid en la página [A Complete Guide to Grid](#).

Flexbox

Por último, puede haber baldas dentro de nuestro diseño, dentro de las cuales existen números variables de elementos. Por ejemplo, en un menú de navegación se pueden añadir o eliminar páginas con relativa frecuencia (figura 20):

```
<nav aria-label="Navegación secundaria">
  <ul>
    <li><a href="mapa.html">Mapa</a></li>
    <li><a href="enlaces.html">Enlaces</a></li>
    <li><a href="incidencias.html">Incidencias</a></li>
    <li><a class="MiPerfil" href="miperfil.html">Mi
Perfil</a></li>
  </ul>
</nav>
```

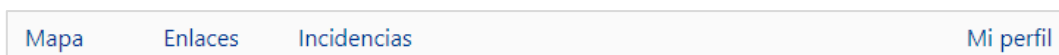


Figura 20

En este caso el uso de Grid no es adecuado, puesto que Grid está pensado para diseñar contenedores con un número determinado de filas y columnas.

Para este tipo de casuísticas es más recomendable el uso de Flexbox, puesto que Flexbox define contenedores para números indeterminado de elementos.

Mediante las siguientes reglas, en modo tableta, definimos a la lista desordenada de la sección de *navegación secundaria* de nuestro ejemplo, como un contenedor *flexible*, cuyos hijos (elementos), se disponen consecutivamente, unos detrás de otros, en una única columna:

```
/* Reglas para tabletas */
@media only screen and (min-width: 768px) {
  /* Reglas CSS */

  nav:nth-of-type(1) > ul {
    display: flex;
    flex-direction: row;
  }
  nav:nth-of-type(1) ul li {
    flex-basis: auto;
  }
  nav:nth-of-type(1) ul li.MiPerfil {
    margin-left: auto;
  }
}
```

Nota: puedes leer más sobre el modelo Flexbox en la página [A Complete Guide to Flexbox](#).

¿Por qué esto es importante? Porque permite adaptar los contenidos de una página, sin pérdida de información y funcionalidad, y sin requerir el empleo simultáneo tanto de la barra de desplazamiento horizontal como de la vertical, a diferentes tipos de dispositivos: teléfonos inteligentes, tabletas, equipos de escritorio, etc.

Contraste

WCAG 1.4.3.

El contraste de una imagen se mide mediante la diferencia existente entre sus sombras y sus luces. Una diferencia mayor entre ambas, arroja una imagen con un mayor contraste y, por ende, con una mayor definición.

En la siguiente imagen (figura 21), se puede comprobar como a medida que el círculo se acerca más hacia el blanco, el contraste con el fondo disminuye y no es más difícil de visualizar:

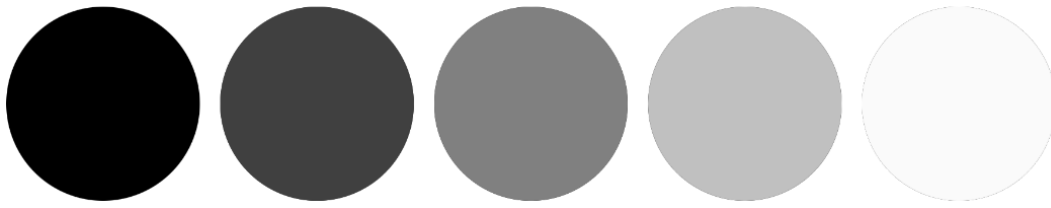


Figura 21

Si la diferencia, la ratio, entre el color de primer plano (empleado en un texto o un componente de la interfaz de usuario) y el color de segundo plano (empleado en un fondo) no tienen el suficiente contraste, muchas personas tendrán problemas de legibilidad, por lo cual, no podrán distinguir con claridad los textos.

Los criterios de las pautas de accesibilidad proponen diferentes ratios mínimas de contraste según los casos. En concreto:

- **Textos hasta 24 píxeles: 4.5:1.**
- **Textos a partir de 24 píxeles y componentes de interfaz de usuario: 3:1.**

Esto es, a partir de 24 píxeles se exige una ratio de contraste menor ($3-1 = 2$), que hasta alcanzar los 24 píxeles ($4.5 - 1 = 3.5$).

Para entender esta nomenclatura, si piensas en el número situado a la izquierda de los dos puntos como el color empleado en el texto y en el número situado a la derecha como el color del fondo, un texto blanco, por ejemplo, sobre fondo blanco (ejemplo extremo) tendría una ratio de 1:1. Esto es, no cumpliría con la ratio exigida:

```
p {
  color:rgb(255,255,255);
  background-color:rgb(255,255,255);
}
```

Sin embargo, un rojo puro igual o inferior a 238 (en notación RGB) sobre fondo blanco si la cumplimentaría:

```
p {
  color:rgb(238,0,0);
  background-color:rgb(255,255,255);
}
```

En el siguiente ejemplo (figura 22), solo los dos primeros textos (128 y 238) satisfacen la ratio exigida entre el rojo y el blanco:

Texto
128 **Texto**
238 **Texto**
255

Figura 22

Nota: desde el punto de vista de la percepción humana, es interesante reseñar, que algunas combinaciones de colores que son calificadas como erróneas por el algoritmo, sin embargo son consideradas como válidas para un gran número de usuarios: [The Myths of Color Contrast Accessibility](#).

Existen diferentes tipos de herramientas para verificar el contraste entre el color de fondo y primer plano. Una de las más conocidas es [Colour Contrast Analyser](#).

También podemos emplear extensiones de navegador como [Wave](#).

Algunos navegadores incluyen también algunas funcionalidades a la hora de analizar los contrastes. Por ejemplo, cuando inspeccionamos en Chrome un elemento que tenga definido en la pestaña **Style** (de las **Herramientas para desarrolladores**), la propiedad **color**, al pulsar sobre el cuadrado de color (que precede al valor de la propiedad), nos mostrará en una ventana emergente (dentro de la sección **Contrast ratio**), si se trata de un valor válido o no.

¿Por qué esto es importante? Para facilitar a las personas con diferentes tipos de discapacidades visuales a percibir los contenidos de la página.

Legibilidad

[WCAG 1.4.12.](#)

Algunas personas necesitan aumentar el interlineado de los párrafos, la separación entre las letras o la separación entre las palabras para aumentar su legibilidad. Para ello pueden emplear hojas de estilo personalizadas, *bookmarklets*, extensiones del navegador, etc.

Por ello, es importante no impedir que tales estilos personalizados puedan ser implementados por el usuario.

Por ejemplo, la diferencia entre las dos siguientes reglas es que, si bien ambas aplican los mismos valores a las mismas propiedades, la primera regla hace uso del modificador **!important** de CSS:

```
p:nth-of-type(2n+1) {
  color:rgb(255,0,0);
  letter-spacing:normal !important;
  word-spacing:normal !important;
  line-height:normal !important;
  margin:0 !important;
}
p:nth-of-type(2n) {
  color:rgb(0,0,255);
  letter-spacing:normal;
  word-spacing:normal;
  line-height:normal;
  margin:0;
```

```
}
```

El modificador **important** de CSS se emplea para anular la preponderancia de una regla sobre otra.

Por lo cual si un usuario emplea un *bookmarklet* (un enlace de HTML que ejecuta código JavaScript) como el siguiente, para aumentar los valores de dichas propiedades, tal aumento sólo se producirá sobre los párrafos que no sean afectados por la primera regla, los párrafos pares (figura 23):

```
<a href="javascript:(function(){var col =
document.querySelectorAll('p');col.forEach(function(el,index){el
.setAttribute('style','letter-spacing:0.12em;word-
spacing:0.16em;line-height:1.5em;margin:1em
0;'));});})();">Bookmarklet</a>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec lobortis ipsum massa, ac accumsan orci vulputate sed. Curabitur eu dui lectus. Etiam non fringilla augue. Fusce eu suscipit ex. Aenean hendrerit pellentesque lacus eu molestie. Aliquam vitae mauris sed ex iaculis luctus dictum nec ipsum. Proin malesuada condimentum scelerisque. Etiam malesuada cursus varius. Proin tincidunt vestibulum arcu, at viverra elit condimentum a. Nam ornare diam sit amet commodo rutrum. Cras et augue ut ligula rutrum faucibus sit amet quis lacus. Donec tempus vehicula malesuada.

Ut pulvinar velit ipsum, ut mattis magna auctor lacinia. Phasellus pulvinar augue sed tincidunt tempor. Aenean pellentesque, diam id interdum dapibus, nibh felis elementum metus, quis luctus leo massa vitae risus. Pellentesque mollis, mauris ut vehicula ultricies, nisl ex molestie est, in efficitur risus turpis pulvinar sapien. Vivamus id eleifend velit. Nulla pharetra mi malesuada elementum auctor. Vestibulum bibendum lacus tellus, ac fermentum lorem pretium quis. Nam at efficitur enim. Suspendisse potenti. Quisque luctus consequat fermentum. Sed ac magna sed purus egestas consequat. Etiam egestas purus enim. Nam ac diam et diam aliquam vulputate sit amet sed nibh. Mauris at maximus nibh.

Figura 23

Además, relacionado con lo anterior, deberemos evitar la asignación de alturas a contenedores de texto mediante valores absolutos, puesto que, si el usuario aumenta el tamaño del texto para mejorar su legibilidad (mediante alguna de las técnicas vistas anteriormente), posiblemente se produzcan solapamientos y desbordamientos del texto en tales contenedores.

¿Por qué esto es importante? Para que los usuarios puedan modificar, según sus necesidades, el interlineado, el espaciado entre letras y el espaciado entre palabras y, aumentar así, su legibilidad, sin que se produzcan solapamientos, ni desbordamientos.

Foco

WCAG 2.4.7.

La navegación de una página no debe ser solamente accesible mediante el empleo del ratón, sino que también debe serlo mediante el empleo del teclado para aquellos usuarios que no puedan emplear un ratón o prefieran navegar mediante el teclado.

Cuando tabulamos, el foco cambia a un elemento activo de la página, a través del cual, el usuario puede ejecutar una acción: visitar un enlace, introducir un valor en un formulario, ejecutar un comando, etc.

Como ayuda visual de navegación, los navegadores recuadran de forma automática el contorno de los elementos que tienen el foco (figura 24). Por esta razón, no se debe ocultar mediante una regla de estilo dicho recuadro:

```
<!-- Ejemplo incorrecto -->
input:focus {
  outline:none;
}
```



Figura 24

¿Por qué esto es importante? Para que los usuarios que navegan mediante el teclado obtengan un indicador visual de los elementos sobre los que pueden interactuar.

El objetivo de JavaScript es el de dotar de **comportamiento** a un documento HTML: pop-ups, rollovers, widgets, etc. Continuando con nuestro símil anterior, en un edificio JavaScript posibilita el funcionamiento de los ascensores, la detección de movimiento, el sistema automático de alumbrado, etc.

Tecnologías compatibles con la accesibilidad

WCAG 2.1.1.

Hasta la llegada de WCAG 2.0 el uso de JavaScript se encontraba desaconsejado, siempre y cuando no se aportara una alternativa que no hiciera uso de él.

Con la aparición de la versión 2.0 de las pautas de accesibilidad web apareció el concepto de **tecnología compatible con la accesibilidad**. Una *tecnología compatible con la accesibilidad* es una tecnología que permite la creación de contenidos accesibles y es soportada, nativamente o mediante la instalación de un *plugin*, por la mayoría de navegadores y productos de uso común del mercado.

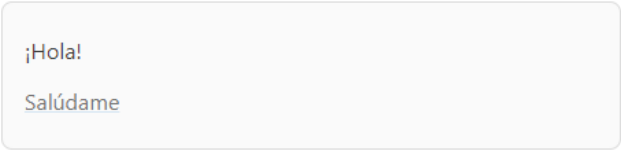
Desde este punto de vista, **JavaScript**, como estándar de facto empleado en el desarrollo web, constituye una tecnología compatible con la accesibilidad. Ahora bien, su mero uso (que puede ser totalmente correcto desde un punto de vista sintáctico), no nos asegura su conformidad con respecto a los requisitos de accesibilidad.

Veámoslo con un ejemplo. El siguiente fragmento de HTML contiene dos párrafos:

```
<p><output id="output1" aria-live="polite">&nbsp;</output></p>
<p><a id="anchor1" href="#">Salúdame</a></p>
```

Mediante el siguiente *script* cuando el usuario sitúe el ratón sobre el literal “Salúdame”, saludaremos al usuario mediante la asignación de la cadena “¡Hola!” al elemento **output1** (figura 25):

```
window.addEventListener("load",function(e){
  var anchor1 = document.querySelector("#anchor1");
  var output1 = document.querySelector("#output1");
  anchor1.addEventListener("mouseover",function(e){
    output1.innerHTML = "¡Hola!";
  });
});
```



¡Hola!
Salúdame

Figura 25

¿Cuál es el problema en el ejemplo anterior con respecto a la accesibilidad? Que una persona que emplee el teclado y tabule hacia nuestro enlace, no obtendrá ningún saludo como respuesta, puesto que no hemos gestionado también un evento para el teclado (figura 26). Esto es, no obtendrá *la misma funcionalidad*.

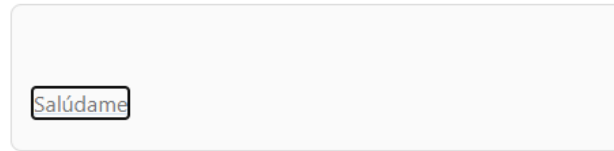


Figura 26

Por lo tanto, debemos evitar el uso de eventos dependientes de un dispositivo concreto, como por ejemplo el ratón y, proporcionar además otro tipo de eventos que cubran la navegación mediante otro tipo de dispositivos como el teclado (figura 27):

```
window.addEventListener("load", function(e) {  
  var anchor1 = document.querySelector("#anchor1");  
  var output1 = document.querySelector("#output1");  
  function hSaludo(e) {  
    output1.innerHTML = "!Hola!";  
  }  
  anchor1.addEventListener("mouseover", hSaludo);  
  anchor1.addEventListener("focus", hSaludo);  
});
```

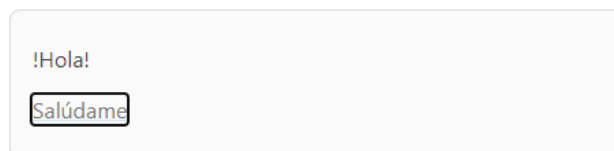


Figura 27

Nota: el atributo **aria-live** de WAI-ARIA se emplea para señalar aquellas zonas o elementos de la página, en las cuales el contenido puede cambiar de forma dinámica. Sin su señalización, puede ser que algunos usuarios y tecnologías de asistencia, no sean capaces de percibir el cambio. Cuando el cambio se produce, los lectores de pantalla lo anunciarán inmediatamente (si se asigna el valor **assertive** al atributo) o tras producirse una pausa (si se asigna el valor **polite**): [Dynamic Content Updates](#).

¿Por qué esto es importante? Para asegurarnos que ningún grupo de usuarios pierda funcionalidad o sea incapaz de navegar por nuestra web.

Navegación mediante el teclado

La navegación por teclado consiste básicamente en ir tabulando por aquellos elementos de la página con los cuales el usuario puede interactuar mediante el teclado: enlaces, controles de formularios, botones, etc.

Ya hemos visto, como dichos controles muestran un contorno visual al obtener el foco, que no debemos ocultar. Pero también debemos tener en cuenta las siguientes consideraciones:

Orden de navegación

El orden en el cual los elementos interactivos de la página van recibiendo el foco, a medida que el usuario va tabulando por la página, debe seguir un orden lógico que respete la estructura de la página. Esto es, si la página se compone de un encabezado, sección de navegación, contenido, etc., al tabular, deberemos acceder primeramente a los elementos interactivos del encabezado, después a los de la barra de navegación, etc.

Deberemos pues:

- Estructurar adecuadamente nuestras páginas mediante el empleo de secciones.
- Evitar el uso de estilos que puedan modificar el orden lógico de navegación mediante el teclado: por ejemplo, en **Flexbox** podemos modificar el orden de los elementos que se encuentran dentro de un contenedor mediante la propiedad **order** de CSS.
- Evitar el uso del atributo **tabindex** de HTML para modificar el orden predeterminado de los elementos interactivos o para aplicarle foco a elementos que no son interactivos.

Una manera de comprobar el orden de navegación de una página es desactivar su hoja de estilos y comenzar a navegar por ella tabulando mediante la tecla **TAB**.

Saltar al contenido principal

Si bien es importante cuidar la estructura y orden lógico, frente a una persona que emplea el ratón y puede interactuar directamente con un elemento interactivo concreto de la página, una persona que emplee el teclado no tiene más remedio que tabular hasta alcanzar el elemento en cuestión para interactuar con él.

Al margen de las soluciones de terceros o de los mecanismos de navegación, proporcionados por las tecnologías de asistencia a la accesibilidad, como desarrolladores web, podemos proporcionar enlaces que le permitan al usuario saltarse las secciones de navegación, a la hora de acceder directamente a la sección principal (figura 28):

```
<a id="skip" href="#main">Saltar navegación</a>
<header>
  <!-- Contenido -->
</header>
<!-- Contenido -->
<main id="main">
  <!-- Contenido -->
```

```
</main>
```



Figura 28

Puesto que tales enlaces no son necesarios para las personas que no empleen el teclado, podemos ocultarlos (hasta que obtengan el foco) mediante las siguientes reglas:

```
#skip {  
  position:absolute;  
  left:-10000px;  
  top:auto;  
  width:1px;  
  height:1px;  
  overflow:hidden;  
}  
#skip:focus {  
  left:auto;  
  width:auto;  
  height:auto;  
}
```

Nota: no debemos ocultar tales enlaces mediante el empleo de las propiedades **display** o **visibility**, puesto que, de hacerse así, ya no se encontrarían disponibles para los lectores de pantalla: [Invisible Content Just for Screen Reader Users](#).

Componentes de usuario

Un componente de usuario es un elemento de la interfaz de la aplicación que ofrece cierta funcionalidad al usuario: por ejemplo, un campo de texto, un botón, un menú, etc. HTML incorpora ciertos componentes de usuario, que se suelen emplear para la confección de formularios. Pero no nos permite la inserción directa de otros tipos de componentes como pueden ser, por ejemplo, los menús de navegación, los acordeones o las pestañas.

Para este segundo tipo de componentes deberemos utilizar una amalgama de HTML, CSS y JavaScript que les de *forma*. Con respecto a dichos componentes, deberemos tener en cuenta, las siguientes consideraciones:

- Comprobar que una vez que el control obtiene el foco mediante el teclado podemos continuar navegando al siguiente elemento interactivo de la página mediante la tecla **TAB** (y no quedarnos atrapados dentro del control). Esto es, una vez que el control ha obtenido el foco, el usuario no empleará la tecla de

tabulación para navegar por el control (sino, por ejemplo, las teclas de dirección).

- Comprobar que la navegación mediante el teclado funciona con independencia del dispositivo. Por ejemplo, si sólo hemos tenido en cuenta la navegación mediante el teclado para dispositivos de sobremesa, dejaremos de lado a los usuarios que también empleen el teclado para navegar en otros tipos de dispositivos.

¿Por qué esto es importante? Para asegurar una navegación mediante el teclado lógica, rápida y sin *trampas*.

Cambios de contexto

[WCAG 3.2.1.](#) | [WCAG 3.2.2.](#)

En lingüística el *contexto* es el entorno del que depende el sentido de una palabra o frase. Así, por ejemplo, la palabra “tío” puede diferir según la empleemos en un entorno familiar o, en un bar, en el que estemos hablando informalmente con unos cuantos amigos.

En el contexto de una página web, el envío de un formulario o la apertura de una nueva página en una nueva pestaña, constituyen cambios de contexto. Si estos cambios se producen como consecuencia de acciones conocidas por el usuario, no hay ningún problema.

Por ejemplo, un usuario envía un formulario pulsando sobre su correspondiente botón de envío. O abre una nueva página, en una nueva ventana, al pulsar sobre un enlace, previo aviso de que la apertura se va a realizar en una pestaña nueva:

```
<a href="https://www.boe.es/diario_boe/" target="_blank"
title="Abre en nueva ventana">Boletín Oficial del Estado</a>
```

El problema surge cuando ante determinadas acciones del usuario efectuamos cambios de contexto inesperados para el usuario, que pueden confundirle y desorientarle.

En el siguiente ejemplo, se muestra una lista desplegable para seleccionar el idioma de la página:

```
<select aria-label="Seleccionar idioma"
onchange="form1.submit();">
  <option lang="es" selected="selected"
onchange="form1.submit();">Bienvenido</option>
  <option lang="ca">Benvingut</option>
  <option lang="eu">Ongi etorri</option>
  <option lang="gl">Benvido</option>
```

```
<option lang="en">Welcome</option>
</select>
```

El problema es que no sólo se selecciona el idioma, sino que, además se efectúa una petición al servidor para que se cargue la página en la versión del idioma seleccionado (obsérvese la gestión de **onchange** en **SELECT**). Esto es, se produce un cambio de contexto inesperado, puesto que el comportamiento predeterminado de una lista es la selección de una de sus opciones y no la ejecución de una acción: la petición al servidor para que se cargue otra versión de la página.

Lo adecuado sería realizar este envío mediante un control que si este pensado para la ejecución de acciones (figura 29):

```
<select aria-label="Seleccionar idioma">
  <option lang="es" selected="selected"
onchange="form1.submit();">Bienvenido</option>
  <option lang="ca">Benvingut</option>
  <option lang="eu">Ongi etorri</option>
  <option lang="gl">Benvido</option>
  <option lang="en">Welcome</option>
</select>

<button type="submit">Cambiar idioma</button>
```

A diferencia del ejemplo anterior, la selección de un idioma no provoca una petición al servidor. Tenemos que pulsar sobre el botón *Cambiar idioma* para que la petición al servidor se produzca y se cargue una nueva versión de la página.

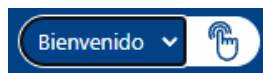


Figura 29

Nota: puedes consultar más sobre esta técnica en: [Using a button with a select element to perform an action](#).

Por lo tanto, debemos evitar los cambios inesperados que puedan confundir al usuario o, al menos, advertirle de tales cambios (como en el caso de los enlaces que abren en una nueva ventana o pestaña).

En concreto, debemos comprobar y evitar en los elementos interactivos de las páginas (enlaces y controles de formulario):

- Que no se produzcan cambios de contexto cuando obtengan el foco.
- Que no se produzcan cambios de contexto cuando su valor cambie.

Esto es, que no gestionemos los eventos **onfocus** y **onchange** de dichos controles para efectuar acciones que pueden resultar inesperadas para los usuarios.

¿Por qué esto es importante? Porque evita que las personas se desorienten o distraigan por cambios de contexto que no responden a las acciones efectuadas por ellos. Los cambios de contexto deben ser consistentes con las acciones del usuario.

Componentes de interfaz de usuario

[WCAG 4.1.2.](#) | [Menus \(WAI Tutorials\)](#)

Anteriormente hemos visto que un **componente de interfaz de usuario** es un elemento de la interfaz de una aplicación (como un campo de texto, un botón o un menú) que ofrece cierta funcionalidad al usuario. También hemos visto, que, si bien HTML incorpora ciertos componentes de usuario (sobre todo orientados a la confección de formularios), no incorpora un marcado específico para otro tipo de componentes, como, por ejemplo, un menú de navegación o un *treeview*.

Para que el usuario pueda percibir estos últimos tipos de componentes necesitamos emplear una combinación de HTML, CSS y JavaScript.

Veamos un ejemplo con un menú de navegación. Un menú de navegación se suele marcar mediante una lista anidada desordenada:

```
<nav id="nav2" aria-label="Menú principal">
  <ul>
    <li><a href="#">TopItem1</a>
      <ul>
        <li><a href="#">Item1</a>
          <ul>
            <li><a href="#">SubItem1</a></li>
            <li><a href="#">SubItem2</a></li>
          </ul>
        </li>
        <li><a href="#">Item2</a></li>
        <li><a href="#">Item3</a></li>
      </ul>
    </li>
    <li><a href="#">TopItem2</a>
      <ul>
        <li><a href="#">Item1</a></li>
        <li><a href="#">Item2</a></li>
        <li><a href="#">Item3</a></li>
      </ul>
    <!-- etc. -->
  </ul>
</nav>
```

El **problema** es que mientras una persona puede percibir el marcado anterior como un componente (figura 30), una tecnología de asistencia no. Sólo percibe una lista anidada

de enlaces y no es capaz de transmitirle al usuario que dicha lista anidada es un menú de navegación.



Figura 30

WAI-ARIA

Para paliar dicha carencia del marcado de HTML, el W3C introdujo **WAI-ARIA**, [Accessible Rich Internet Applications](#), un marco de trabajo que proporciona una serie de atributos mediante los cuales es posible comunicar dicha información a las tecnologías de asistencia.

Ya hemos visto algunos de estos atributos. Por ejemplo:

- **Aria-label:** nos permite especificar un nombre accesible para un elemento:

```
<button type="button" aria-label="Cerrar">X</button>
```

- **Aria-describedby:** nos permite aportar información adicional para un elemento.
- **Role:** nos permite especificar un *tipo* para un elemento:

```
<div role="tab"><h2>Etiqueta</h2></div>
```

En el documento [Using ARIA](#) se establecen unas cuantas reglas sobre el uso de WAI-ARIA:

Primera regla: si mediante el marcado de HTML es posible comunicar a una tecnología de asistencia el propósito de un componente, no emplees atributos de ARIA. ¿Por qué? Porque los atributos de ARIA suscriben un compromiso de comportamiento asociado, que a menudo no se implementa o se implementa de forma incorrecta.

Segunda regla: no modifiques la semántica por defecto de un elemento de HTML, a menos que sea imprescindible:

```
<h2 role="tab">Etiqueta</h2>
```

Tercera regla: todos los componentes que empleen ARIA deben ser operables mediante teclado.

Cuarta regla: no asignes a un elemento interactivo el rol **presentation**, que elimina la semántica del elemento, su tipo. Y no les apliques el atributo **aria-hidden**, puesto que entonces, no podrán ser detectados por las tecnologías de asistencia.

Quinta regla: asegúrate que todos los elementos interactivos son identificables por una tecnología de asistencia. Esto es, posee un nombre accesible.

Resumiendo hasta aquí, ARIA nos proporciona, por una parte: una serie de atributos mediante los cuales podemos comunicar a las tecnologías de asistencia la semántica de nuestros componentes; y, por otra, la indicación (mediante su tercera regla), de que dichos componentes puedan ser operables mediante el teclado.

Patrones de diseño

Ahora bien, dado un componente y varios desarrolladores, podríamos tener diferentes implementaciones para un mismo componente.

Para paliar, en lo posible, dicha diversificación en el desarrollo, el sitio web [ARIA Authoring Practices Guide \(APG\)](#), recomienda una serie de patrones para la confección de los componentes de interfaz de usuario de uso más frecuente: acordeones, cuadros de diálogo, carruseles, menús de navegación, etc.

En concreto, para cada uno de estos componentes se especifica tanto los atributos de WAI-ARIA que deben emplearse, como la operabilidad mediante el teclado que debe implementarse.

Retomando nuestro ejemplo, para la confección de un menú de navegación, deberíamos buscar dentro de la página [Patterns](#), el componente que concuerde más con dicho componente: [Menu o Menu bar](#).

Si bien, el patrón [Menu o Menu bar](#), parece haberse pensado más para la implementación de un menú de una **aplicación web**, que para la implementación de un simple menú de navegación web.

Nota: de hecho, algunos autores han defendido que no se debería emplear dicho patrón para la confección de un menú de navegación: [Don't use ARIA Menu Roles for Site nav](#).

Parece que las opiniones al respecto han sido tenidas en cuenta por el W3C y, al acceder a la página de ejemplo [Navigation Menubar Example](#), podemos leer la siguiente advertencia:

CAUTION! *Before considering use of the ARIA menubar pattern for site navigation, it is important to understand:*

- *The menubar pattern requires complex functionality that is unnecessary for typical site navigation that is styled to look like a menubar with expandable sections or fly outs.*

- *A pattern more suited for typical site navigation with expandable groups of links is the disclosure pattern. For an example, see Example Disclosure Navigation Menu.*

Esto es, por una parte, se nos indica que la funcionalidad requerida por el patrón es extremadamente compleja para la implementación de un simple menú de navegación web. Y por otra, se nos recomienda, que empleemos en su lugar el patrón [Disclosure \(Show/Hide\)](#).

Veamos, por lo tanto, cómo confeccionar nuestro menú de navegación mediante la implementación de dicho patrón.

Patrón disclosure

Un componente **disclosure** es un componente compuesto por un botón y una sección de contenido, que es mostrada u ocultada cuando presionamos sucesivamente sobre el botón (figura 31).

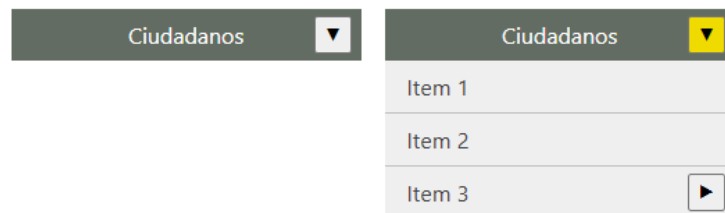


Figura 31

Si accedemos a algunos de los ejemplos para la confección de un menú de navegación mediante este componente, veremos que la estrategia a seguir consiste en añadir a cada elemento de la lista que tenga una lista anidada, un botón que sirva para mostrarla u ocultarla:

```
<nav id="nav2" aria-label="Menú principal">
  <ul>
    <li>
      <a href="#" aria-current="page">TopItem1</a>
      <button aria-label="TopItem1" aria-expanded="false" aria-controls="n0">▼</button>
      <ul id="n0">
        <li>
          <a href="#">Item1</a>
          <button aria-label="TopItem1" aria-expanded="false"
aria-controls="n1">▶</button>
          <ul id="n1">
            <li><a href="#">SubItem1</a></li>
            <li><a href="#">SubItem2</a></li>
          </ul>
        </li>
        <li><a href="#">Item2</a></li>
        <li><a href="#">Item3</a></li>
      </ul>
    </li>
  </ul>
</nav>
```

```

    </ul>
  </li>
  <li>
    <a href="#">TopItem2</a>
    <button aria-label="TopItem2" aria-expanded="false" aria-
controls="n2">▼</button>
    <ul id="n2">
      <li><a href="#">Item1</a></li>
      <li><a href="#">Item2</a></li>
      <li><a href="#">Item3</a></li>
    </ul>
    <!-- etc. -->
  </li>
  <!-- etc. -->
</ul>
</nav>

```

Detengámonos un momento para estudiar nuestro ejemplo:

- Incluimos nuestro menú de navegación dentro de una sección de navegación convenientemente etiquetada.

Empleamos los siguientes atributos de ARIA con los botones:

- **Aria-label:** etiquetamos nuestros botones con el literal del enlace al que acompañan.
- **Aria-expanded:** este atributo le comunica a una tecnología de asistencia, si una lista anidada de nuestro menú se encuentra desplegada o replegada.
- **Aria-Controls:** identifica la lista anidada que puede ser desplegada o replegada por un botón.

Empleamos el siguiente atributo de ARIA con los enlaces:

- **Aria-current:** identifica el enlace, dentro del menú de navegación, correspondiente a la página en la cual nos encontramos.

Todo esto, no expresa más que un compromiso que aun debemos implementar. Esto es, simplemente con el marcado actual, cuando pulsemos sobre los botones, sus listas asociadas no se van a replegar y desplegar. Tampoco al navegar por él, se va a cambiar la asignación del atributo **aria-current**, al nuevo enlace sobre el que pulsemos.

Ahora bien, gracias al patrón, disponemos de una estructura base desde la cual partir.

Navegación mediante el teclado

Hasta aquí acabamos de estudiar la parte semántica del patrón, pero hemos visto que los patrones también especifican el comportamiento que debe implementarse mediante el teclado para un componente.

En concreto se definen los siguientes comportamientos para las siguientes teclas:

- **ENTER o SPACE:** si el foco se encuentra sobre un botón, su lista anidada correspondiente se desplegará o replegará, sin que el botón pierda el foco. Si se encuentra sobre un enlace, navegará a él y le asignará el atributo **aria-current**.
- **ESC:** si el foco se encuentra sobre un botón, cuya lista anidada se encuentra desplegada, la replegará. Si se encuentra sobre un enlace de una lista anidada, la replegará y le aplicará el foco al botón al que se encuentra asociada.
- **FLECHA DERECHA o INFERIOR:** si el foco se encuentra sobre un enlace o un botón, cuya lista asociada se encuentra replegada, se le asignará el foco al siguiente enlace o botón existentes. Si el foco se encuentra sobre un botón, cuya lista asociada se encuentra desplegada, se asignará el foco al primer enlace de la lista asociada.
- **FLECHA IZQUIERDA o SUPERIOR:** si el foco se encuentra sobre un enlace o un botón (con independencia del estado de su lista asociada), se le asignará el foco al anterior enlace o botón existentes.
- **HOME:** si el foco se encuentra sobre un enlace o un botón, aplicaremos el foco, si es posible, al primer enlace del nivel en el que se encuentra.
- **END:** si el foco se encuentra sobre un enlace o un botón, aplicaremos el foco, si es posible, al último enlace del nivel en el que se encuentra.

¿Cómo debería funcionar la tabulación? En el punto sobre navegación mediante el teclado, hemos visto que se recomienda que cuando un componente obtenga el foco, la siguiente vez que tabulemos sobre él, apliquemos el foco al siguiente elemento activo de la página, que no se encuentre dentro del componente. Y que, dentro del componente, establezcamos una navegación personalizada para el componente:

[Keyboard Navigation Inside Components](#).

Ahora bien, como el patrón empleado por nuestro componente no emplea atributos de rol (a la hora de evitar la complejidad de implementar un simple menú de navegación, como un menú de una aplicación web), los lectores de pantalla no serán capaces de cambiar automáticamente del *modo exploración*, al *modo foco*.

Nota: el **modo exploración** de un lector de pantalla emplea un cursor, a partir del cual comienza a leer. El usuario puede cambiar la posición del cursor, mediante el teclado, para que el lector lea las partes que sean de su interés. Por ejemplo, al pulsar la flecha inferior en **NVDA**, el cursor se posicionará en la siguiente línea y comenzará a leerla. O al pulsar la tecla **H** posicionará el cursor en el siguiente encabezado y lo leerá.

Ahora bien, ¿qué ocurre, por ejemplo, cuando el cursor se encuentra situado dentro de un elemento interactivo como un campo de texto y pulsamos la letra **H**? Que no se escribirá la letra **H**, sino que se desplazará el cursor al siguiente encabezado.

Para evitar este tipo de problemas con los elementos interactivos de una página, los lectores también soportan un **modo foco**. En este modo, las teclas empleadas no son interpretadas por el lector, sino pasadas al navegador para su ejecución habitual.

El problema es que los lectores, no son siempre capaces de cambiar de un modo a otro de forma automática. Algunas veces, deberemos forzar el modo para poder interactuar con un elemento interactivo: por ejemplo, en un campo de texto, al pulsar sobre la tecla **ENTER**, **NVDA** emitirá un pitido y cambiará a modo foco.

Por último, recalcar que una de las casuísticas en las cuales si son capaces de cambiar automáticamente al modo foco es cuando tabulamos. Al tabular, el siguiente elemento interactivo obtiene el foco y el lector cambia automáticamente al modo foco.

Lo cual quiere decir, que cuando empleemos las teclas de flecha en nuestro menú, en lugar de ejecutarse los comportamientos descritos por el patrón, lo que cambiará es la posición del cursor controlado por el lector de pantalla.

Para posibilitar, por tanto, la correcta navegación con un lector de pantalla, el patrón admite el uso de la tabulación dentro del componente. En concreto:

- **TAB**: si el foco se encuentra sobre un vínculo o un botón, cuya lista asociada se encuentra replegada, se aplicará el foco al siguiente enlace o botón existente. Si el foco se encuentra sobre un botón, cuya lista asociada se encuentra desplegada, se aplicará el foco al primer enlace de la lista asociada.

¿Por qué esto es importante? Para posibilitar la interacción de las tecnologías de asistencia a la accesibilidad con los componentes de interfaz de usuario.

Presentación del menú

Para la presentación del menú emplearemos una hoja de estilos en cascada.

Empezaremos por definir dentro de nuestra hoja de estilo, las reglas para teléfonos inteligentes (figura 32) y, posteriormente, las reglas para tabletas (y otro tipo de dispositivos de resolución mayor).



Figura 32

Comenzamos definiendo el modelo de cálculo mediante la propiedad **box-sizing** y asignando cero a los márgenes, externos e internos, de los elementos que se encuentren dentro de la sección de navegación de nuestro menú:

```

/* ***** Reglas para teléfonos inteligentes ***** */
* {
  box-sizing: border-box;
}
#nav2 * {
  margin:0px;
  padding:0px;
}

```

Ocultamos las viñetas en las listas y aplicamos, a los elementos de las listas, posición relativa:

```

#nav2 ul {
  list-style-type:none;
}
#nav2 ul li {
  position:relative;
}

```

Aplicamos a nuestros enlaces una visualización de *bloque* para que ocupen la totalidad del ancho de nuestros dispositivos móviles:

```

#nav2 ul li a {
  display:block;
  color:rgb(255,255,255);
  background-color:rgb(98,108,98);
  text-decoration:none;
  padding:0.5em 1em;
  border-bottom:1px solid rgb(192,192,192);
}

```

Asignamos un tamaño a nuestros botones y les aplicamos posición absoluta, para colarlos a la derecha de nuestros enlaces:

```

#nav2 ul li button {
  width:2em;
  height:2em;
  position:absolute;
  top:0.4em;
  right:0.5em;
  min-width:0 !important;
}

```

Aplicamos estilos diferenciados para informar al usuario: cuando se encuentra sobre un enlace (en dispositivos táctiles, dicho evento no suele reconocerse de una forma adecuada); cuando está pulsando sobre él; y para indicarle, cuál es el enlace que apunta a la página actual en la cual se encuentra:

```
#nav2 ul li a:hover, #nav2 ul li a:focus {
  color:rgb(28,57,148);
  background-color:rgb(255,255,255);
}
#nav2 ul li a:active {
  background-color:rgb(232,232,255);
}
#nav2 ul li a[aria-current] {
  color:rgb(64,64,64);
  background-color:rgb(255,210,0);
}
```

Aplicamos sangrías para diferenciar visualmente los enlaces de unos niveles y de otros:

```
#nav2 ul li ul li a {
  padding-left:3em;
  color:rgb(64,64,64);
  background-color:rgb(239,239,239);
}
#nav2 ul li ul li ul li a {
  padding-left:4em;
}
```

Ocultamos, de manera predeterminada, las listas anidadas del menú, de manera que no se muestren desplegadas al cargarse la página:

```
#nav2 > ul > li > ul {
  display:none;
}
```

Ocultamos o mostramos, las listas anidadas asociadas a los botones, en base al valor asignado al atributo **aria-expanded** de los botones:

```
#nav2 ul li button[aria-expanded="false"] ~ ul {
  display:none;
}
#nav2 ul li button[aria-expanded="true"] ~ ul {
  display:block;
}
```

En modo tableta, definimos al **** raíz del menú, como un contenedor de tipo Flexbox. Y especificamos, para cada uno de sus hijos *flexibles* ****, una anchura mínima de 100 pixeles, a partir de la cual, podrán expandirse hasta ocupar horizontalmente el espacio que se encuentre disponible en su contenedor:

```
/* ***** Reglas para tabletas ***** */
@media only screen and (min-width: 768px) {
  #nav2 > ul {
```



```

    display: flex;
}
#nav2 > ul > li {
    width: 100px;
    flex-grow: 1;
    margin: 0.1em;
}

```

Aplicamos alineación centrada a los enlaces del nivel superior:

```

#nav2 > ul > li > a {
    text-align: center;
    min-height: 100%;
    border-bottom: none;
}

```

Por último, aplicamos diferentes estilos para la visualización de los menús y submenús desplegables:

```

#nav2 ul li ul {
    position: absolute;
    z-index: 1;
    width: 100%;
}
#nav2 ul li ul li {
    position: relative;
}
#nav2 ul li ul li ul {
    top: 0px;
    left: 100%;
}
#nav2 ul li:last-child ul li ul {
    left: -100%;
}
#nav2 ul li ul li a {
    padding-left: 1em !important;
}
}

```

En la siguiente imagen (figura 33), se puede apreciar la apariencia del menú en modo tableta y escritorio:



Figura 33

Implementación mediante JavaScript

Nos queda, ahora, aplicarle el comportamiento a nuestro menú mediante el empleo de JavaScript.

Para facilitar la comprensión del código resulta útil confeccionar un árbol invertido, con el marcado del menú (figura 34):

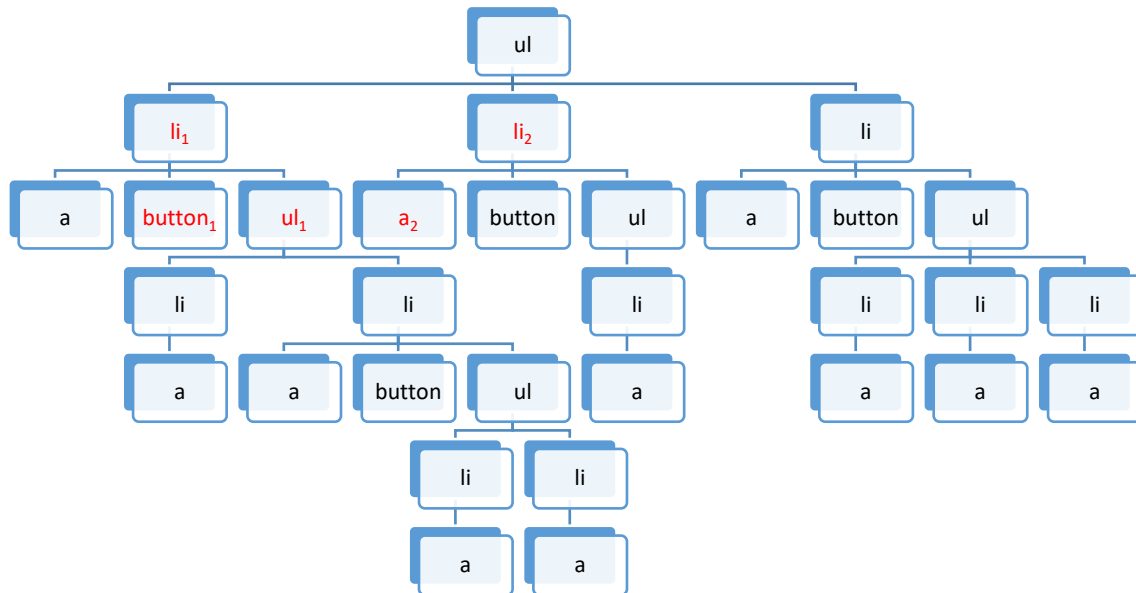


Figura 34

Y parafrasear alguno de los comportamientos vistos para el teclado en el patrón en *términos familiares*. Por ejemplo, *si pulsamos la flecha derecha o inferior, situados en un botón cuya lista asociada se encuentra replegada, le aplicaremos el foco al siguiente enlace existente*.

¿Cómo podemos parafrasear dicho comportamiento mediante el empleo de nuestro árbol? De la siguiente manera: si el foco se encuentra en el botón **button₁** y su lista asociada **ul₁** se encuentra replegada (lo cual podemos averiguar consultando el valor del atributo **aria-expanded** del botón), cuando pulsemos la flecha derecha o inferior, tendremos que:

- Buscar el elemento **li₁** padre de **button₁**.
- Luego el elemento **li₂** (su tío), hermano del padre de **button₁**.
- Y finalmente, aplicar el foco, al hijo de su tío **a₂** (su primo).

Primeramente, vamos a mostrar la estructura de nuestro *script* y después iremos estudiando cada una de sus partes:

```
window.addEventListener("load", function(e) {  
  //root  
  var root = document.querySelector("#nav2 > ul");  
});
```

```

//a
var col_a = document.querySelectorAll("#nav2 > ul a");
col_a.forEach(function(el,index) {
    //Código
});

//Keydown
var col = document.querySelectorAll("#nav2 a, #nav2 button");
col.forEach(function(el,index) {
    //Código
});
});

```

Una vez que se dispara el evento **onload**, al terminarse de cargarse la página, obtenemos una referencia del elemento **UL** raíz de nuestro menú, mediante el método **querySelector**:

```

//root
var root = document.querySelector("#nav2 > ul");

```

A continuación, obtenemos la colección de enlaces de nuestro menú, mediante el método **querySelectorAll** y la recorreremos mediante un bucle **forEach**. Observa la función anónima empleada en el bucle. El parámetro **el** recoge en cada iteración del bucle a un elemento de la colección **col_a**, esto es, a un enlace del menú. El parámetro **index** recoge el número de la iteración en la que nos encontramos, comenzando a contar desde cero:

```

//a
var col_a = document.querySelectorAll("#nav2 > ul a");
col_a.forEach(function(el,index) {
    //Código
});

```

Veamos ahora que vamos a hacer dentro del bucle en cada iteración.

Primero vamos a asignar el atributo **aria-current**, al enlace del menú que coincida con la página actual en la que nos encontramos. Para ello cotejamos, si la página a la que apunta el enlace, **el_page**, es la misma que la página en la que nos encontramos, **url_page**:

```

//aria-current
var href = el.href.split("/");
var el_page = href[href.length-1].toLowerCase();
var url = window.location.toString().split("/");
var url_page = url[url.length-1].toLowerCase();
if(el_page == url_page) {
    el.setAttribute("aria-current", "page");
}

```

```
}
```

A continuación, si el enlace en el que nos encontramos iterando en el bucle, tiene un hermano (una lista anidada), le añadiremos un botón que nos permita desplegarla y replegarla.

El condicional comprueba la existencia de la lista anidada mediante la propiedad **nextElementSibling** del enlace y, en caso de que no sea nula, crea y añade el botón.

Observa cómo se asignan los valores a los atributos del botón:

- **Aria-label:** le asignamos el valor de la propiedad **textContent** del enlace al que acompaña. Esto es, el literal del enlace.
- **Aria-expanded:** le asignamos el valor **false**, puesto que de manera predeterminada queremos que la lista asociada se encuentre replegada.
- **Aria-controls:** le asignamos un valor compuesto por la letra **n** y el número de la iteración en la que nos encontramos. Por ejemplo, para la primera iteración **n0**.

Después, le añadimos una flecha hacia abajo, **▼**, si se trata de un botón que se encuentra en el menú superior o una flecha hacia la derecha, **▶**, si se trata de un botón que se encuentra en un nivel inferior.

Asociamos la lista anidada con el botón, asignando al atributo **id** de la lista, el mismo valor que le hemos asignado al atributo **aria-controls** del botón.

Finalmente, añadimos el botón antes de su lista asociada, mediante el método **insertBefore** del padre de su enlace:

```
//Button
if(el.nextElementSibling != null){
  //button
  var button = document.createElement("button");
  button.setAttribute("aria-label",el.textContent);
  button.setAttribute("aria-expanded","false");
  button.setAttribute("aria-controls","n"+index);
  if(el.parentElement.parentElement == root){
    button.innerHTML = "&#9660;";
  }
  else{
    button.innerHTML = "&#9654;";
  }
  el.nextElementSibling.setAttribute("id","n"+index);
  el.parentElement.insertBefore(button,el.nextElementSibling);
}
```

Una vez añadido el botón, pasamos a gestionar su evento **onclick**. Cuando se pulse con el ratón sobre el botón (o cuando este tenga el foco y pulsemos las teclas **ENTER** o **SPACE**), desplegaremos o replegaremos su lista asociada:

```
//Onclick
```

```
button.addEventListener("click", function(e) {
    //Código
});
```

Si al pulsar sobre el botón su lista asociada se encuentra replegada, la desplegaremos asignando al atributo **aria-expanded** del botón el valor **true** (lo cual provocará que la regla de la hoja de estilo definida para **[aria-expanded="true"]**, le aplique a la propiedad **display** de la lista asociada el valor **block**).

Como puede darse el caso, de que otra lista anidada descendiente del abuelo del botón actual ya se encuentre desplegada, obtenemos la colección **col_button** de todos los botones descendientes del abuelo del botón actual y, nos aseguramos, de que sus listas asociadas se encuentren replegadas:

```
if(button.getAttribute("aria-expanded")==false){
    button.setAttribute("aria-expanded", "true");
    var col_button =
button.parentElement.parentElement.querySelectorAll("button");
    col_button.forEach(function(el,index){
        if(el != button){
            el.setAttribute("aria-expanded", "false");
        }
    });
}
```

Si al pulsar sobre el botón su lista asociada se encuentra desplegada, la replegaremos asignando al atributo **aria-expanded** del botón el valor **false** (lo cual provocará que la regla de la hoja de estilo definida para **[aria-expanded="false"]**, le aplique a la propiedad **display** de la lista asociada el valor **none**).

Como puede darse el caso, de que otra lista anidada descendiente de la actual ya se encuentre desplegada, obtenemos la colección **col_button** de todos los botones descendientes de la lista asociada con el botón actual y, nos aseguramos, de que sus listas asociadas se encuentren replegadas:

```
else{
    button.setAttribute("aria-expanded", "false");
    var col_button =
button.nextElementSibling.querySelectorAll("button");
    col_button.forEach(function(el,index){
        el.setAttribute("aria-expanded", "false");
    });
}
```

Una vez finalizada la iteración del bucle **forEach**, para la colección **col_a** de enlaces del menú, vamos a implementar el comportamiento de las teclas definidas en el patrón, cuando son pulsadas sobre un enlace o un botón del menú que posea el foco.

Para ello, primeramente, obtendremos la colección **col** de enlaces y botones del menú, mediante el método **querySelectorAll**:

```
//Keydown
var col = document.querySelectorAll("#nav2 a, #nav2 button");
```

A continuación, recorreremos dicha colección mediante un bucle **forEach**. Observa que en este caso el parámetro **el**, podrá referenciar tanto a un enlace como a un botón, al tratarse de una colección mixta de elementos.

Para cada elemento de nuestra colección gestionaremos el evento **onkeydown** y ejecutaremos el código correspondiente, en base al código de la tecla presionada, que obtendremos de la propiedad **keyCode** del objeto **Event**:

```
col.forEach(function(el, index) {
    el.addEventListener("keydown", function(e) {
        switch(e.keyCode) {
            //ESC
            case 27:
                //Código
                break;

            //Home
            case 36:
                //Código
                break;

            //End
            case 35:
                //Código
                break;

            //DOWN
            case 40:
            //RIGHT
            case 39:
                //Código
                break;

            //UP
            case 38:
            //LEFT
            case 37:
                //Código
                break;
        }
    });
});
```

Veamos el código para cada una de las teclas.

ENTER o SPACE: si el foco se encuentra sobre un botón, su lista anidada correspondiente se desplegará o replegará, sin que el botón pierda el foco. Si se encuentra sobre un enlace, navegará a él y le asignará el atributo **aria-current**.

Cuando se pulsa un botón con las teclas **ENTER** o **SPACE** se dispara el evento **onclick**. Como dicho evento ya lo hemos gestionado anteriormente, no es necesario que codifiquemos nada para estas teclas. La asignación del atributo **aria-current** también ha sido codificada ya.

ESC: si el foco se encuentra sobre un botón, cuya lista anidada se encuentra desplegada, la replegará. Si se encuentra sobre un enlace de una lista anidada, la replegará y le aplicará el foco al botón al que se encuentra asociada.

Para implementar esta parte del código, vamos a diferenciar cuando pulsamos sobre un botón del nivel superior y cuando lo hacemos sobre un enlace o un botón de un subnivel. Para ello, averiguamos si el abuelo del enlace o el botón actual se corresponde con el elemento **UL** raíz.

Si se trata de un botón de nivel superior, cuya lista asociada se encuentra desplegada, la replegamos.

Si no, si se trata de un botón de un subnivel, cuya lista asociada se encuentra desplegada, también la replegamos.

Y si se trata de un enlace o de un botón, cuya lista asociada se encuentra replegada, replegamos la lista en la cual se encuentra el enlace o el botón:

```
//ESC
case 27:
    //Si pulsamos ESC en el primer nivel
    if(el.parentElement.parentElement == root){
        //Sobre un botón cuyo menú este desplegado
        if(el.tagName == "BUTTON" && el.getAttribute("aria-expanded")== "true"){
            el.setAttribute("aria-expanded", "false");
        }
    }
    else{
        //Si pulsamos ESC en un subnivel cuyo menú este desplegado
        if(el.tagName == "BUTTON" && el.getAttribute("aria-expanded")== "true"){
            el.setAttribute("aria-expanded", "false");
        }
        //Si pulsamos ESC en un subnivel sobre un enlace o un botón cuyo menú este replegado
        else{
            var ul_parent = el.parentElement.parentElement;
            ul_parent.previousElementSibling.setAttribute("aria-expanded", "false");
        }
    }
}
```

```

        ul_parent.previousElementSibling.focus();
    }
}
break;

```

HOME: si el foco se encuentra sobre un enlace o un botón, aplicaremos el foco, si es posible, al primer enlace del nivel en el que se encuentra.

Para ello, a través del abuelo **UL** del enlace o botón actual, accederemos a su primer descendiente **LI** y al enlace hijo de este **A**:

```

//Home
case 36:
    e.preventDefault();
    //Seleccionamos el 1er enlace
el.parentElement.parentElement.children[0].querySelector("a").focus();
break;

```

END: si el foco se encuentra sobre un enlace o un botón, aplicaremos el foco, si es posible, al último enlace del nivel en el que se encuentra.

Para ello, a través del abuelo **UL** del enlace o botón actual, accederemos a su último descendiente **LI** y al enlace hijo de este **A**:

```

//End
case 35:
    e.preventDefault();
    //Seleccionamos el último enlace
el.parentElement.parentElement.children[el.parentElement.parentElement.children.length-1].querySelector("a").focus();
break;

```

FLECHA DERECHA o INFERIOR: si el foco se encuentra sobre un enlace o un botón, cuya lista asociada se encuentra replegada, se le asignará el foco al siguiente enlace o botón existentes. Si el foco se encuentra sobre un botón, cuya lista asociada se encuentra desplegada, se asignará el foco al primer enlace de la lista asociada.

Si se trata de un enlace **A** y existe su hermano **BUTTON**, le aplicaremos el foco.

Si no existe su hermano, comprobaremos si existe su tío posterior **LI**, en cuyo caso le aplicamos el foco a su primo **A**.

Si se trata de un botón **BUTTON**, cuya lista asociada **UL** se encuentra desplegada, accederemos a dicha lista mediante la propiedad **nextElementSibling**, obtendremos su primer descendiente **LI** y le aplicaremos el foco a su enlace **A**.

Si la lista asociada **UL** no se encuentra desplegada, comprobaremos si existe su tío posterior **LI**, en cuyo caso le aplicamos el foco a su primo **A**:

```
//DOWN
case 40:
//RIGHT
case 39:
    e.preventDefault();
    //A
    if(el.tagName == "A"){
        //Si el siguiente elemento existe (BUTTON), le
aplicamos el foco
        if(el.nextElementSibling != null){
            el.nextElementSibling.focus();
        }
        //Si no, si existe su tío posterior (LI), le
aplicamos el foco a su enlace (A) primo
        else{
            if(el.parentElement.nextElementSibling != null){
                el.parentElement.nextElementSibling.querySelector("a").focus();
            }
        }
    }
    //BUTTON
    if(el.tagName == "BUTTON"){
        //Si el menú está desplegado, le aplicamos el foco
al primer enlace (A) del menú
        if(el.getAttribute("aria-expanded") == "true"){
            el.nextElementSibling.querySelector("li").querySelector("a").focus();
        }
        //Si está replegado y existe su tío posterior (LI),
le aplicamos el foco a su enlace (A) primo
        else{
            if(el.parentElement.nextElementSibling != null){
                el.parentElement.nextElementSibling.querySelector("a").focus();
            }
        }
    }
    break;
```

FLECHA IZQUIERDA o SUPERIOR: si el foco se encuentra sobre un enlace o un botón (con independencia del estado de su lista asociada), se le asignará el foco al anterior enlace o botón existentes.

Si se trata de un enlace **A** y su tío anterior **LI** existe y tiene un botón **BUTTON**, le aplicaremos el foco al botón.

Si su tío **LI** existe, pero no tiene un botón **BUTTON**, le aplicaremos el foco al enlace **A** de su tío.

Si se trata de un botón **BUTTON**, con independencia de cómo se encuentre su lista asociada, le asignaremos el foco a su hermano anterior **A**:

```
//UP
case 38:
//LEFT
case 37:
    e.preventDefault();
    //A
    if(el.tagName == "A"){
        //Si su tio anterior (LI) existe
        if(el.parentElement.previousElementSibling != null){
            //y tiene un primo botón (BUTTON), le aplicamos el
foco
            if(el.parentElement.previousElementSibling.querySelector("button
") != null){
                el.parentElement.previousElementSibling.querySelector("button").
                focus();
            }
            //si no, le aplicamos el foco a su primo enlace (A)
        }else{el.parentElement.previousElementSibling.querySelector("a").
        focus();
        }
    }
    //BUTTON
    //Le aplicamos el foco a su enlace hermano (A)
    if(el.tagName == "BUTTON"){
        el.previousElementSibling.focus();
    }
    break;
```

¿Por qué esto es importante? Para que los usuarios que navegan mediante teclado no pierdan funcionalidad al interactuar con un componente o se queden atrapados en él.

Consideraciones

Como puede observarse, la elaboración de un componente accesible mediante la aplicación de un patrón de diseño no es una tarea fácil:

- Requiere una carga de desarrollo adicional a la del desarrollo tradicional de componentes.
- La documentación, en el momento actual, no aclara cual debería ser el funcionamiento o comportamiento de dichos componentes en navegadores para teléfonos inteligentes.
- Existe cierta confusión, entre lo que debe considerarse un menú de una aplicación web y un simple menú de navegación web.

En todo caso, mientras HTML no aporte un marcado para la inclusión de compones de interfaz de usuario, sólo dispondremos de WAI-ARIA, como único mecanismo capaz de comunicar la semántica de nuestros componentes, a las tecnologías de asistencia a la accesibilidad.

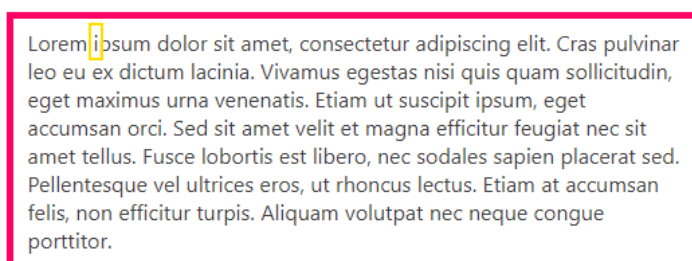
Lectores de pantalla

Un lector de pantalla es un software encargado de convertir la información textual a voz sintetizada. Por eso, necesitan conocer el idioma a emplear, para poder aplicar una entonación y pronunciación adecuadas.

Pueden emplearse tanto para la lectura de diferentes tipos de documentos, como para la lectura de páginas web.

Emplean un cursor virtual, a partir del cual comienzan a leer los contenidos de una forma lineal. Si bien, disponen de diferentes combinaciones de teclas a la hora de situar dicho cursor sobre otras partes del documento. Por ejemplo, pueden mover el cursor una posición hacia la derecha o izquierda del carácter en el que se encuentran, desplazarlo hacia la siguiente o anterior palabra, etc.

En el siguiente ejemplo (figura 35), en el cual se ha empleado el lector de pantalla NVDA, puede observarse la posición del cursor virtual recuadrado en color amarillo sobre la letra *i* de la palabra *ipsum*:



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras pulvinar leo eu ex dictum lacinia. Vivamus egestas nisi quis quam sollicitudin, eget maximus urna venenatis. Etiam ut suscipit ipsum, eget accumsan orci. Sed sit amet velit et magna efficitur feugiat nec sit amet tellus. Fusce lobortis est libero, nec sodales sapien placerat sed. Pellentesque vel ultrices eros, ut rhoncus lectus. Etiam at accumsan felis, non efficitur turpis. Aliquam volutpat nec neque congue porttitor.

Figura 35

También nos permiten desplazar el cursor sobre elementos diferenciados dentro del documento, como, por ejemplo: encabezados, listas, enlaces, imágenes, tablas, etc. Por ello, es de vital importancia que los documentos se encuentren correctamente estructurados y maquetados. Pues si nuestra página, por ejemplo, no incluye encabezados, el lector no podrá acceder a ellos y tendrá que limitarse a leer linealmente su contenido.

En el siguiente ejemplo (figura 36), también empleando NVDA, hemos desplazado el cursor al encabezado *Abril*, presionando sucesivamente la tecla **H** (*heading*). Cada vez que presionamos dicha tecla, el lector sitúa el cursor al inicio del siguiente encabezado que encuentra y lo lee:



Figura 36

En una página web, por lo general, los lectores pueden emplear dos tipos de modos: el **modo exploración** y el **modo foco**. El modo exploración es el modo que hemos visto hasta ahora, en el cual, mediante una serie de combinaciones de teclas, situamos el cursor virtual en una zona o elemento de la página, a partir del cual el lector comienza su lectura.

Pero en una página web existen también elementos interactivos (como los enlaces y los controles de los formularios), con los cuales el usuario puede interactuar. Por ejemplo, en la siguiente imagen (figura 37), nos encontramos situados en un campo de texto que nos solicita nuestro nombre:



Figura 37

Si nos encontramos en el modo exploración (imagen de la izquierda recuadrada en rojo), y pulsamos sobre la letra **D**, en lugar de escribir la letra, situará (en NVDA), el cursor al inicio de la siguiente sección de la página (header, nav, main, etc.).

Pero si nos encontramos en modo foco (imagen de la derecha recuadrada en azul), escribirá la letra **D**, de la manera habitual.

Esto es, en modo exploración (que es el modo por defecto en el que suelen funcionar los lectores de pantalla), es el lector el que gestiona las teclas que pulsamos, mientras que en modo foco, es el navegador el que las gestiona.

El problema es que los lectores, no son siempre capaces de cambiar de un modo a otro de manera automática. Por ello, algunas veces, tendremos que forzar el cambio de modo para poder interactuar con un elemento interactivo. Por ejemplo, en NVDA podemos forzar dicho cambio mediante la combinación de teclas **NVDAMod + SPACE**, donde **NVDAMod** es la tecla empleada por el lector como modificador (normalmente la tecla **Bloq Mayús**).

Si empleamos la tecla de tabulación, el comportamiento es el adecuado. Esto es, el lector cambia automáticamente al modo foco, por lo cual, el siguiente elemento

interactivo de la página obtiene el foco. En el siguiente ejemplo (figura 38), podemos apreciar un enlace que ha obtenido el foco con el lector de pantalla activo:



Figura 38

De entre los lectores de pantalla actualmente disponibles, los más populares son [JAWS](#) (Windows), [NVDA](#) (Windows), [VoiceOver](#) (Mackintosh) y [TalkBack](#) (Android).

NVDA

Lo mejor para empezar a familiarizarse con su uso, es comenzar con una página que sólo incluya párrafos. Así, podemos aprender primero a desplazar el cursor a diferentes partes de los párrafos para que el lector proceda a su lectura:

- **Para desplazar el cursor al siguiente o anterior carácter:** [LEFT|RIGHT].
- **Para desplazar el cursor a la siguiente o a la anterior palabra:** Ctrl + [LEFT|RIGHT].
- **Para desplazar el cursor a la siguiente o anterior línea:** [UP|DOWN].
- **Para desplazar el cursor al siguiente o anterior párrafo:** Ctrl + [UP|DOWN].
- **Para detener la lectura:** Ctrl.
- **Para reanudar la lectura (donde se encontrará originalmente el cursor):** ModNVDA + DOWN.
- **Para silenciar el lector:** ModNVDA + S.

A continuación, podemos cargar una página que contengan otros tipos de elementos, aparte de los párrafos, y aprender a situar el cursor en ellos para que el lector los lea:

- **Tecla D:** sitúa el cursor en la siguiente sección de la página, la identifica y lee su primer elemento.
- **Tecla H:** sitúa el cursor en el siguiente encabezado de la página y lo lee.
- **Tecla L:** sitúa el cursor en la siguiente lista de la página y lee su primer elemento.
- **Tecla K:** sitúa el cursor en el siguiente enlace de la página y lo lee.
- **Tecla G:** sitúa el cursor en la siguiente imagen de la página y lee su texto alternativo.
- **Tecla T:** sitúa el cursor en la siguiente tabla de la página y lee su título.
- **Tecla F:** sitúa el cursor en el siguiente formulario de la página y se sitúa en su primer control. Puede ser necesario cambiar al modo foco (NVDAMod + Space).

También es posible situar el cursor en un elemento anterior, combinando las teclas vistas con **SHIFT**. Por ejemplo, para situar el cursor en el encabezado anterior emplearíamos **SHIFT + H**.

O navegar por los elementos interactivos de la página (enlaces y controles de formularios), mediante la tecla de tabulación de la manera habitual.

Nota: puedes practicar y consultar el uso de NVDA y otros lectores de pantalla desde la sección **Evaluation, Testing and Tools** de la página [Articles](#) de [WebAIM](#).

JavaScript

En este anexo se repasan una serie de puntos relacionados con JavaScript, con el propósito de facilitar la comprensión de la lectura de la tercera parte, JavaScript, de esta guía.

¿Dónde se inserta un script?

Los lugares donde podemos insertar *scripts* en una página web son muy similares a los lugares donde podemos insertar una hoja de estilos en cascada.

Podemos insertarlos directamente en un atributo de evento de una marca:

```
<button onclick="alert('¡Funciona!');">Púlsame</button>
```

Podemos insertarlos en cualquier parte de la página mediante el empleo de la marca **<script>**:

```
<script>
  alert("¡Funciona!");
</script>
```

O podemos vincularlos con un fichero aparte, lo cual nos permite reutilizar el *script* con más de una página web:

```
<script src="script.js"></script>
```

Nota: está es la opción que se está empleando en esta guía.

Eventos

En programación orientada a objetos, un evento es un suceso que le acaece a un objeto. Por ejemplo, pulsar con el ratón sobre un botón o pulsar la tecla ENTER cuando tiene el foco, dispara el evento **onclick** sobre el botón.

Nota: llamar a un ascensor, pedirle una bebida a un camarero o felicitar a alguien son ejemplos cotidianos de eventos. Observa que no tienen por qué tener respuesta: el ascensor puede estar averiado, el camarero no oírte, etc.

El evento **onload** que se produce en el navegador (objeto **window**), al descargarse una página, es un buen punto de partida para la ejecución de un *script*. En el siguiente ejemplo, gestionamos dicho evento para el objeto **window**, mediante el empleo del método [addEventListener](#):


```
window.addEventListener("load", function(e) {  
    console.log("¡Funciona!");  
});
```

Si ejecutamos el *script* anterior y accedemos a las **Herramientas para desarrolladores** del navegador (normalmente pulsando F12), en la pestaña **Console** deberemos poder leer el texto **¡Funciona!** que hemos escrito mediante el método **log** del objeto **console** (figura 39):

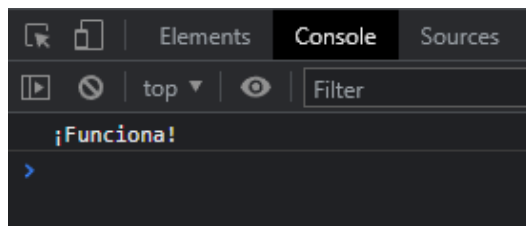


Figura 39

Nota: en el caso de haber cometido algún error sintáctico, podremos ver dicho error en la pestaña **Console**. Prueba a cometer errores para entender los distintos mensajes de error que se muestran en la consola.

Referencias

Una vez que hemos vinculado nuestro *script* y gestionado su evento **load**, normalmente obtendremos la referencia de un objeto o una colección de objetos.

Nota: piensa en una referencia como en un *nombre*. Los nombres nos permiten hacer referencias a distintos objetos o personas (nuestros propios nombres funcionan como referencias nuestras).

Partiendo del siguiente fragmento de HTML:

```
<main>  
  <h1>JavaScript</h1>  
  <div class="grid">  
    <div>  
      <ul>  
        <li>Top element 01</li>  
        <li>Top element 02</li>  
        <li>Top element 03</li>  
      </ul>  
    </div>  
  </div>  
</main>
```

Mediante el método **querySelector** de **document** obtendremos una referencia del primer elemento que satisfaga el selector de CSS:

```

window.addEventListener("load",function(e) {
    var el = document.querySelector("main > .grid > div > ul >
    li");
    console.log(el.textContent);
});

```

Como se puede comprobar por consola (figura 40), obtenemos una referencia del primer elemento ``, de la lista ``, cuyo contenido textual se corresponde con el texto *Top element 01*.

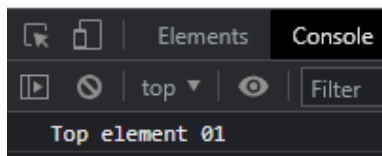


Figura 40

Si en lugar de emplear **querySelector**, empleamos [querySelectorAll](#), obtendremos la colección de todos los elementos `` de la lista `` (figura 41):

```

window.addEventListener("load",function(e) {
    var col = document.querySelectorAll("main > .grid > div > ul >
    li");
    console.log(col);
});

```

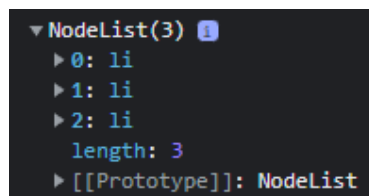


Figura 41

Si una vez obtenida la colección de elementos ``, quisiéramos referenciar su primer elemento, por ejemplo, para obtener su contenido textual, emplearíamos la notación habitual para *arrays* de JavaScript (figura 42):

```

console.log(col[2].textContent);

```

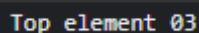


Figura 42

Algunas cosas que podemos hacer con una referencia

Una vez hemos obtenido una referencia, **el**, de un elemento, algunas de las cosas que podemos hacer son:

Contenido

Obtener su contenido textual:

```
var texto = el.textContent;
```

Modificar su contenido textual:

```
el.textContent = "Alicia en el país de las Maravillas";
```

Obtener su contenido HTML:

```
var html = el.innerHTML;
```

Modificar su contenido HTML:

```
el.innerHTML = "<h1>Alicia en el país de las Maravillas</h1>";
```

Atributos

Obtener el valor de un atributo:

```
var valor = el.getAttribute("id");
```

Modificar el valor de un atributo:

```
el.setAttribute("id", "button1");
```

CSS

Añadir una clase:

```
el.classList.add("clase1");
```

Quitar una clase:

```
el.classList.remove("clase1");
```

Una cuestión de familia

Muchas veces a partir de una referencia de un elemento, querremos obtener la referencia de otro, que puede ser su padre, hermano, hijo, etc.

Nota: al igual que en el día a día, empleamos expresiones como tu hermano, su sobrino, mi primo, etc.

Vamos a complicar un poco nuestro ejemplo anterior de HTML:

```
<main>
  <h1>JavaScript</h1>
  <div class="grid">
    <div>
      <ul id="padre">
        <li id="e1">
          <a id="hijo1" href="#">Top element 01</a>
          <ul id="hijo2">
            <li><a href="#">Item 01</a>
            <li><a href="#">Item 02</a>
            <li><a href="#">Item 03</a>
          </ul>
        </li>
        <li id="hermano1">
          <a href="#">Top element 02</a>
        </li>
        <li id="hermano2">
          <a href="#">Top element 03</a>
        </li>
      </ul>
    </div>
  </div>
</main>
```

Y vamos a contemplar su lista ``, mediante un gráfico en forma de árbol invertido (hemos añadido subíndices para mejorar la comprensión):

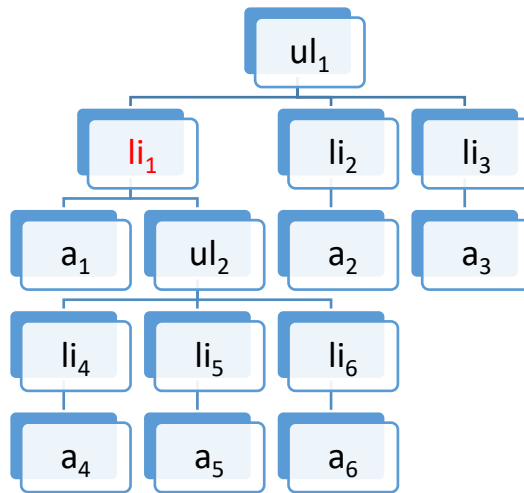


Figura 43

Si nos fijamos en el elemento **li₁** (identificado como **el** en el fragmento de HTML anterior):

- **ul₁**: es su padre.
- **li₂** y **li₃**: son sus hermanos.
- **a₁** y **ul₂**: son sus hijos.
- **a₂** y **a₃**: son sus sobrinos.
- Y el resto de elementos, junto a **a₁** y **ul₂**, son sus descendientes.

¿Cómo se obtendrían dichas referencias mediante código?

Para obtener una referencia del primer **** (figura 44):

```
var el = document.querySelector("main > .grid > div > ul > li");
console.log("LI: " + el.getAttribute("id"));
```

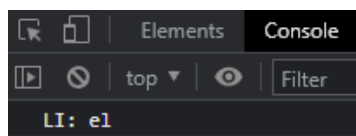


Figura 44

Nota: también podríamos haber obtenido directamente una referencia a partir de su atributo **id**, pero el objetivo del ejercicio es aprender a hacerlo con elementos que no tienen por qué estar identificados.

Para obtener una referencia del padre de **el** (figura 45):

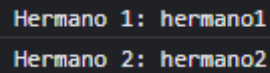
```
var padre = el.parentElement;
console.log("Padre: " + padre.getAttribute("id"));
```

Padre: padre

Figura 45

Para obtener una referencia de los hermanos de **el** (figura 46):

```
var hermano1 = el.parentElement.children[1];
console.log("Hermano 1: " + hermano1.getAttribute("id"));
var hermano2 = el.parentElement.children[2];
console.log("Hermano 2: " + hermano2.getAttribute("id"));
```



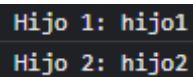
```
Hermano 1: hermano1
Hermano 2: hermano2
```

Figura 46

Nota: hemos empleado la colección **children** del padre de **el**, para obtener una referencia de sus hermanos. Observa que **el == el.parentElement.children[0]**.

Para obtener una referencia de los hijos de **el** (figura 47):

```
var hijo1 = el.children[0];
console.log("Hijo 1: " + hijo1.getAttribute("id"));
var hijo2 = el.children[1];
console.log("Hijo 2: " + hijo2.getAttribute("id"));
```

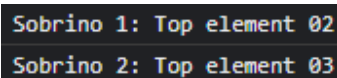


```
Hijo 1: hijo1
Hijo 2: hijo2
```

Figura 47

Para obtener una referencia de los sobrinos de **el** (figura 48):

```
var sobrino1 = el.parentElement.children[1].children[0];
console.log("Sobrino 1: " + sobrino1.textContent);
var sobrino2 = el.parentElement.children[2].children[0];
console.log("Sobrino 2: " + sobrino2.textContent);
```



```
Sobrino 1: Top element 02
Sobrino 2: Top element 03
```

Figura 48

Otras posibilidades a tener en cuenta.

Para obtener el siguiente hermano de **el**:

```
var hermano = el.nextElementSibling;
```

Para obtener el hermano anterior de **el**:

```
var hermano = el.previousElementSibling;
```

Para obtener el primer elemento **<div>** que sea ancestro de **el**:

```
var ancestro = el.closest("div");
```

Nota: un elemento **ancestro** es como un elemento **descendiente**, pero al revés. Esto es, mientras que los elementos hijos, nietos, etc. de un elemento son sus descendientes, los elementos padres, abuelos, etc. son sus ancestros.

El método [closest](#) es muy útil, a la hora de averiguar si un elemento es o no descendiente de otro. Por ejemplo, cómo podemos saber si un enlace de una página es un simple enlace o forma parte de un menú de navegación: comprobando si dicho enlace tiene o no como ancestro al elemento **<nav>**.

Crear elementos

En ocasiones será interesante poder añadir nuevos elementos que no existan en el HTML de partida.

Nota: esto es debido a que muchas veces no sabemos con antelación el número de elementos que vamos a necesitar en una página. Por ejemplo, los enlaces de un menú de navegación cambiarán a medida que se incluyan o eliminen páginas del menú.

En el siguiente ejemplo:

1. Creamos un botón con el método [createElement](#) de **document**.
2. Le asignamos el identificador **button1** a su atributo **id**.
3. Le asignamos el texto **¡Púlsame!**.
4. Y finalmente lo agregamos al HTML existente, mediante el método **prepend**, como primer hijo de **main > .grid > div** (figura 49):

```
window.addEventListener("load", function(e) {  
    var button1 = document.createElement("button");  
    button1.setAttribute("id", "button1");  
    button1.innerHTML = "¡Púlsame!";  
    document.querySelector("main > .grid > div").prepend(button1);  
});
```



Figura 49

Una vez agregado al HTML existente, podemos gestionar sus eventos. Por ejemplo, cuando alguien pulse sobre él, le cambiaremos su etiqueta en función del valor de la variable **bandera flag** (figura 50):

```
var flag = true;  
button1.addEventListener("click", function(e) {
```

```

    if(flag){
        button1.innerHTML = "¡Me has pulsado!";
        flag = false;
    }
    else{
        button1.innerHTML = "¡Púlsame!";
        flag = true;
    }
    });
});

```

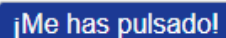


Figura 50

Recorrer colecciones

Cuando se obtiene una colección de elementos, es muy habitual recorrerlos mediante un bucle. Tomando como punto de partida nuestro fragmento de HTML original:

```

<main>
  <h1>JavaScript</h1>
  <div class="grid">
    <div>
      <ul>
        <li>Top element 01</li>
        <li>Top element 02</li>
        <li>Top element 03</li>
      </ul>
    </div>
  </div>
</main>

```

Vamos a obtener la colección de elementos `` de la lista ``:

```

window.addEventListener("load",function(e) {
    var col = document.querySelectorAll("main > .grid > div > ul >
    li");

```

Y, vamos a emplear un bucle **forEach** para iterar sobre cada uno de ellos:

```

    col.forEach(function(el,index) {
        console.log(index + ": " + el.textContent);
    });
});

```


En cada iteración, los valores de los argumentos **el** e **index**, empleados en la función anónima, cambian:

- **el**: contiene una referencia de un elemento de la colección. En nuestro ejemplo: en la primera iteración referenciará el primer elemento **** de la colección; en la segunda, el segundo elemento ****; y, en la tercera, el tercer elemento ****.
- **index**: recibe un entero, correspondiente al ordinal de cada iteración. Como se puede apreciar en la salida por consola (figura 51), el primer valor empleado por **index** es 0, el segundo 1 y el tercero 2.

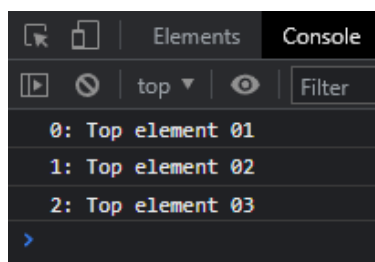


Figura 51

Nota: una **función anónima** es una función que no tiene un nombre para invocarla. Su ejecución depende del contexto donde se situé. En nuestro ejemplo, el bucle **forEach**.

Eventos dentro de un bucle

Si necesitamos gestionar un evento, para cada elemento de una colección, podemos hacerlo mediante el empleo de un bucle **forEach**:

```
window.addEventListener("load", function(e) {
  var col = document.querySelectorAll("main > .grid > div > ul >
  li");
  col.forEach(function(el, index) {
    el.addEventListener("click", function(e) {
      console.log(el.textContent);
    });
  });
});
```

Como se puede observar, el ejemplo es muy parecido al anterior: primero obtenemos la colección de elementos **** de **** y, a continuación, iteramos por dicha colección mediante un bucle **forEach**.

Dentro del bucle empleamos el parámetro **el** de la función anónima, para gestionar en cada iteración del bucle, el evento **onclick** para cada elemento **** de la colección.

Para comprobar su funcionamiento, tendremos que observar por consola, que cuando pulsamos sobre un elemento **** con el ratón, se muestra su correspondiente contenido textual.

En la siguiente imagen (figura 52), se muestra un ejemplo de salida en el cual se ha pulsado consecutivamente sobre el tercer y segundo elemento de la lista:

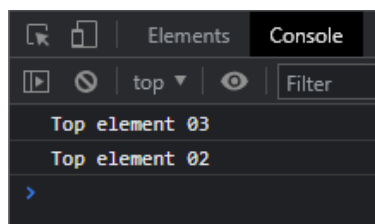


Figura 52