

**WAI-ARIA**

**17/02/2021**

Cuestiones

**Cristian Fernandez Tirado**

Contenido

[1. Lee el apartado "ARIA, el aliado (casi) desconocido", página 197, del libro "Accesibilidad Web - WCAG 2.1 de forma sencilla" y responde a las siguientes cuestiones: 3](#_Toc64188113)

[1.1 ¿Cuál es la idea principal por la que el W3C desarrolló WAI-ARIA? 3](#_Toc64188114)

[1.2 ¿Cuáles son los tres elementos fundamentales de los que se compone ARIA? 3](#_Toc64188115)

[1.3 Analiza el Ejemplo 1: "Cambiar el funcionamiento de un objeto: una capa que se comporta como un botón". 3](#_Toc64188116)

[1.4 ¿En qué casos se recomienda utilizar ARIA en lugar de elementos HTML nativos? 4](#_Toc64188117)

[1.5 ¿Qué es un rol en ARIA? 5](#_Toc64188118)

[1.6 ¿Qué es un landmark role"? 5](#_Toc64188119)

[1.7 ¿Qué es una "live region"? 6](#_Toc64188120)

[1.8 ¿Qué diferencia hay entre los estados y las propiedades ARIA? 6](#_Toc64188121)

[1.9 ¿Qué usos tiene el atributo *aria-label*? 6](#_Toc64188122)

[1.10 ¿Y aria-labelledby? 7](#_Toc64188123)

[1.11 ¿Y aria-describedby? 9](#_Toc64188124)

[1.12 Analiza el Ejemplo 2: "Mejorar el uso accesible: navegación en pestañas". 10](#_Toc64188125)

[1.13 Revisa la lista de Buenas prácticas. 12](#_Toc64188126)

[1.14 Analiza el Ejemplo 3: "Validar un campo obligatorio". 13](#_Toc64188127)

[2. Instala la extensión *Web Developer Toolbar* (Chrome o Firefox) 14](#_Toc64188128)

[3. Realiza un análisis de accesibilidad de la web https://www.mcarmendealba.com/roles-aria-regiones-de-puntos-de-referencia-y-nombres-accesibles/ . Observa el uso que hace de atributos ARIA. 15](#_Toc64188129)

# Lee el apartado "ARIA, el aliado (casi) desconocido", página 197, del libro "[Accesibilidad Web - WCAG 2.1 de forma sencilla](https://ciclos.iesruizgijon.es/pluginfile.php/6030/mod_page/content/9/AA_Accesibilidad_Web_WCAG_ARIA_19_10_2018.pdf)" y responde a las siguientes cuestiones:

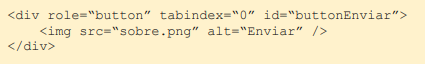
## ¿Cuál es la idea principal por la que el W3C desarrolló WAI-ARIA?

Permitir añadir información semántica a cualquier elemento de la interfaz, consiguiendo que los navegadores y productos de apoyo comprendan el comportamiento de las etiquetas pudiendo ser diferente al habitual

## ¿Cuáles son los tres elementos fundamentales de los que se compone ARIA?

Roles, estados y propiedades

## Analiza el Ejemplo 1: "Cambiar el funcionamiento de un objeto: una capa que se comporta como un botón".



Al contendor de la imagen se le asigna el **rol** de “botón” y mediante JavaScript se le asocia un evento.

Además, para que también sea accesible usando la tecla **TAB** se le asigna un **tabindex**

## ¿En qué casos se recomienda utilizar ARIA en lugar de elementos HTML nativos?

Los cuatro casos en los que se recomienda usar ARIA en lugar de elementos HTML, nativos son:

* La característica no está disponible en HTML
* La característica esta disponible en HTML, pero no esta implementada en los agentes de usuario
* La característica esta disponible e implementada en HTML, pero el agente de usuario no proporciona el soporte para la accesibilidad de ese elemento
* El diseño visual “obliga” a determinado estilo, pero no podríamos decorar un elemento nativo con ese diseño visual

## ¿Qué es un rol en ARIA?

Los **Roles** en aria son los encargados de permitir indicar la función de un elemento de la interfaz. La función se asigna con el atributo **rule**.

Actualmente hay más de 81 roles categorizados. Unos ejemplos serian:

* **Abstract:** *widget, window, section, input,* etc. No se usan en el contenido, si no para definir dentro de la ontología tipos de roles generales
* **Widget**: *menu, menuitem, tree, treeitem, tablist, tab, tabpanel, button, grid,* etc.
* **Document Structure**: *beading, table, img, tooltip, list, presentation,* etc. No suelen ser interactivos
* **Landmark:** *banner, main, navigation, contentinfo,* etc. Permiten definir grandes zonas de la pagina principal igual que las etiquetas semánticas de HTML5. Los usuarios de lector de pantalla tienen atajos para saltar de zona en zona o sacar un árbol de la estructura de la página generado a partir de esta información.
* **Live Region:** *alert, log, marquee, status* y *timer*. Definen la función de las zonas “vivas” de la página, es decir, que cambian automáticamente sin la intervención del usuario
* **Window**. *Alertdialog* y *dialog*, para las capas que abrimos a modo de ventanas

Para más roles: <http://www.w3.org/TR/wai-aria-1.1/#role_definitions>

## ¿Qué es un landmark role"?

Son los roles que permiten definir las grandes zonas de la pagina igual que las etiquetas semánticas HTML5

## ¿Qué es una "live region"?

Son los roles que definen la función de las zonas “vivas” de la página, es decir, que cambian automáticamente sin intervención del usuario

## ¿Qué diferencia hay entre los estados y las propiedades ARIA?

La diferencia entre los estados y las propiedades es que las propiedades suelen cambiar menos (aunque no siempre) que los estados, que cambian con frecuencia debido a la interacción del usuario.

Pero en la practica no es necesario diferenciarlos y todos ellos comienzan con *aria-*

## ¿Qué usos tiene el atributo *aria-label*?

El atributo *aria-label* permite indicar directamente **el contenido de la etiqueta del elemento.**

En el siguiente código:



El lector de pantalla leerá “Botón Cerrar”

También se puede usar para **ampliar la información** de un enlace que en principio podría ser confuso.



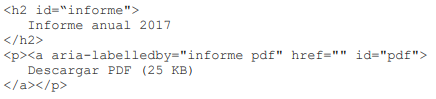
Otro uso seria para **distinguir zonas de la página**.



## ¿Y aria-labelledby?

El atributo aria-labelledby se usa para indicar **el contenido de la etiqueta del elemento**, cuando el nombre de la etiqueta esta visible en la página.

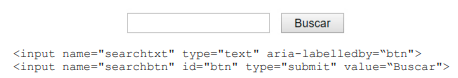
Por ejemplo, en el siguiente código:



El lector anunciara el enlace como “Informe actual 2017 Descargar PDF” en lugar de “enlace PDF”

Otro uso se da en los **formularios de búsqueda**.

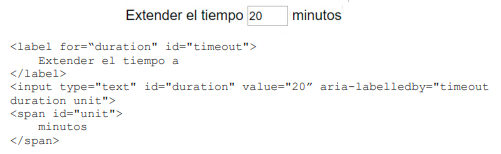
Por ejemplo, en el siguiente código.



El lector anunciara el campo como de texto como “Buscar”

También se puede usar con el **valor de los campos de un formulario**.

Por ejemplo, en el siguiente código



El lector anunciara el campo como “Extender el tiempo a 20 minutos”

También se puede usar para **etiquetar zonas de la pagina** o para etiquetar **imágenes**.

Por ejemplo, en el siguiente código

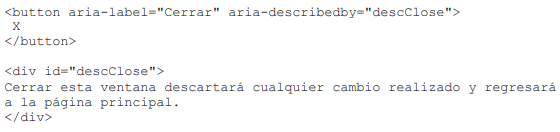


El lector ignorara las estrellas porque tienen alt=”” y anunciara la capa como una imagen con la etiqueta asociada “puntuación de 4 de 5”

## ¿Y aria-describedby?

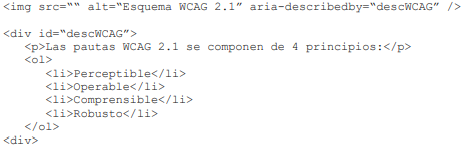
El atributo aria-describedby se usa para indicar **una información adicional** a la de su etiqueta.

Por ejemplo, en el siguiente código:



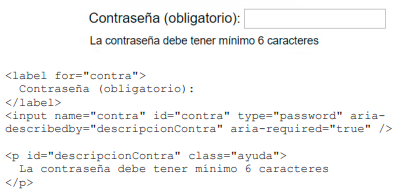
El lector de pantalla anunciara: “Botón Cerrar. Cerrar esta ventana descartara cualquier cambio realizado y regresara a la página principal”

También puede usarse para la **descripción de imágenes:**



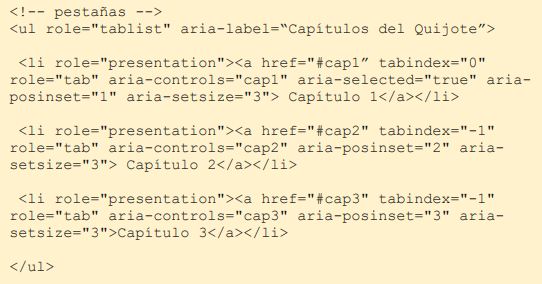
Además, se puede usar para **asociar un campo de texto a su ayuda contextual:**

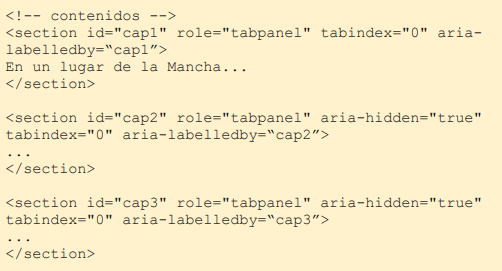
Por ejemplo, en el siguiente código:



El lector anunciara el campo con la etiqueta “Contraseña (obligatorio)” y le añade la descripción “La contraseña debe tener como mínimo 6 caracteres”

## Analiza el Ejemplo 2: "Mejorar el uso accesible: navegación en pestañas".





Se indica que cada elemento de la lista cumple el **rol** de **presentation**.

A los enlaces se les añade varios atributos

* Primero **el rol** de pestaña **tab**
* Luego, que contenido controla **aria-controls=cap1**
* Si esta seleccionado o no **aria-selected=true** que deberá cambiarse dinámicamente con JavaScript cada vez que el usuario pulse una pestaña
* Por último, **aria-posinset** y **aria-setsize** para indicar que posición ocupa y cuantas pestañas hay

Por último, a cada sección se le añaden los siguientes atributos

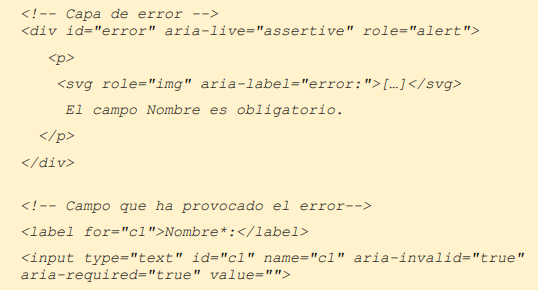
* Primero cuál es su **rol role=tabpanel**
* Luego se hace que pueda coger el foco con **tabindex=0**, para facilitar a los usuarios de productos de apoyo moverse fácilmente hasta ella, especialmente si dentro del panel no hay elemento que pueda coger el foco
* Si e método utilizado para ocultar las pestañas solo las oculta visualmente (por ejemplo, con *text-indent: -1000px)* necesitamos el atributo **aria-hidden=true** que indica que se oculte también al lector de pantalla y deberá cambiarse también dinámicamente su valor por JavaScript a medida que el usuario pulse las pestañas
* Por último, tendremos que añadir le atributo **aria-labeledby** para etiquetarla con el nombre de su pestaña
* Además, a los enlaces de las pestañas se le añadirán dos **tabindex** quedando como **tabindex=0** y **tabindex=-1**, la razón es por la manera en que deben comportarse las pestañas en el acceso mediante teclado. Por eso a la primera pestaña se le da **tabindex=0** para que coja el foco y a las demás se le elimina de la secuencia del tabulador con **tabindex=-1**. A medida que el usuario selecciona las pestañas con las flechas se va cambiando dinámicamente por JavaScript el atributo de **tabindex=-1** a **tabindex=0**. A esta técnica se le llama **roving tabindex**

## Revisa la lista de Buenas prácticas.

**Mal usado, ARIA es peligroso,** por lo que hay que W3C recomienda

* **No utilices ARIA si no es necesario**: Utiliza siempre que puedas etiquetas HTML estándar
* **Un rol es una promesa:** Si se indica que un *<div>* es un botón ls navegadores de darán el comportamiento de un botón.
* **Usa los roles y las propiedades según la especificación:** Recuerda que un rol no debe cambiar dinámicamente, solo se cambian las propiedades y estados.
* **Evitar los conflictos:** No añadas ARIA a etiquetas si pueden entrar en conflicto con su propiedad semántica, como podría ser indicar que un radiobuton tenga el rol de checkbox
* **Evita la redundancia:** No añadas ARIA a los controles nativo con el mismo valor
* **Cambia los estados y las propiedades en respuesta a los eventos:** Si indicamos que un árbol esta abierto con *aria-expanded=true* cuando el usuario lo cierre debemos cambiar la propiedad mediante JavaScript y ponerla en *false*
* **Accesible por teclado:** En HTML no todos los elementos cogen el foco automáticamente, para ello se usa el atributo *tabindex=0* o *tabindex=-1* si solo queremos que coja el foco por programación
* **Sincroniza la interfaz visual con la interfaz accesible:** Si un elemento cambia su estado de seleccionado a no seleccionado, cambia también su estilo.
* **Usa JavaScript no intrusivo:** Para tener un código HTML puro es recomendable usar JavaScript aparte para conseguir una **mejora progresiva**
* **Revisión de ARIA:** Si se usa ARIA siempre hay que revisarlo con algunas herramientas dedicadas para ello

## Analiza el Ejemplo 3: "Validar un campo obligatorio".



La capa de error del formulario el **rol=”alert”** y el atributo **aria-live=”asertive”**, lo que provoca que en cuanto se modifique el contenido de la capa el lector anuncie **“Alerta”** y lea su contenido (Sin que el foco se mueva de la capa)

El mensaje de error va precedido de un icono SVG. Este elemento tiene los atributos **rol=”img”** y **aria-abel=”error”**. Esto hará que el icono sea anunciado como imagen con la etiqueta error

En cuanto al campo, vemos que tiene los atributos **aria-required=”true”** y **aria-invalid=”true”** para que cuando coja el foco, el lector anuncio que el campo es obligatorio y que su entrada es invalida

# Instala la extensión *Web Developer Toolbar* (Chrome o Firefox)

# Realiza un análisis de accesibilidad de la web <https://www.mcarmendealba.com/roles-aria-regiones-de-puntos-de-referencia-y-nombres-accesibles/> . Observa el uso que hace de atributos ARIA.