
# how to a11y

All you need to know about building accessible web content

Tanja Ulianova

# Tanja Ulianova

inovex

Web Developer from Karlsruhe

all things web
a11y nerd
clean architecture

@techtanja

# What is a11y?

# a11y = a{ccessibilit}y

making web content usable for any people
(regardless of their abilities)

part of universal design and good usability

# Users!

| | |
|---|---|
|  low vision, blindness, color blindness<br>*Screen Reader* | people who can't see a mouse cursor<br>rely on screen reader |
|  low hearing, deafness | somebody who cannot hear well needs captions |
|  motor impairments<br>*people who can't operate a mouse* | keyboard access |
|  cognitive impairments, dyslexia, migraines | simple language, uncomplicated GUIs, reader mode |
|  seizure disorders | avoid flashing animations |

# "us" vs. "them"

not needing a11y is only temporary

natural • illness • situational • aging

# Guidelines

W3C Working Draft

TABLE OF CONTENTS


Abstract

Status of This Document

Introduction

.1 Background on WCAG 2

.2 WCAG 2 Layers of Guidance

.3 WCAG 2.2 Supporting Documents

.4 Requirements for WCAG 2.2

.5 Comparison with WCAG 2.1

.5.1 New Features in WCAG 2.2

.5.2 Numbering in WCAG 2.2

.5.3 Conformance to WCAG 2.2

.6 Later Versions of Accessibility Guidelines

1. Perceivable

1.1 Text Alternatives

1.1.1 Non-text Content

1.2 Time-based Media

1.2.1 Audio-only and Video-only (Prerecorded)

1.2.2 Captions (Prerecorded)

1.2.3 Audio Description or Media Alternative (Prerecorded)

1.2.4 Captions (Live)

1.2.5 Audio Description (Prerecorded)

1.2.6 Sign Language (Prerecorded)

1.2.7 Extended Audio Description (Prerecorded)

1.2.8 Media Alternative (Prerecorded)

1.2.9 Audio-only (Live)

1.3 Adaptable

1.3.1 Info and Relationships

1.3.2 Meaningful Sequence

1.3.3 Sensory Characteristics

1.3.4 Orientation

1.3.5 Identify Input Purpose


# Web Content Accessibility Guidelines (WCAG) 2.2

W3C Working Draft 21 May 2021


**This version:**
https://www.w3.org/TR/2021/WD-WCAG22-20210521/

**Latest published version:**
https://www.w3.org/TR/WCAG22/

**Latest editor's draft:**
https://w3c.github.io/wcag/guidelines/22/

**Previous version:**
https://www.w3.org/TR/2021/WD-WCAG22-20210513/

**Latest Recommendation:**
https://www.w3.org/TR/WCAG/



**Editors:**
Chuck Adams (Oracle)
Alastair Campbell (Nomensa)
Rachael Montgomery (Invited Expert)
Michael Cooper (W3C)
Andrew Kirkpatrick (Adobe)


**Participate:**
GitHub w3c/wcag
File a bug
Commit history
Pull requests



## Abstract


Web Content Accessibility Guidelines (WCAG) 2.2 covers a wide range of recommendations for making Web content more accessible. Following these guidelines will make content more accessible to a wider range of people with disabilities, including accommodations for blindness and low vision, deafness and hearing loss,


**W**eb **C**ontent **A**ccessibility **G**uidelines: https://www.w3.org/TR/WCAG22/

# WCAG 2.2

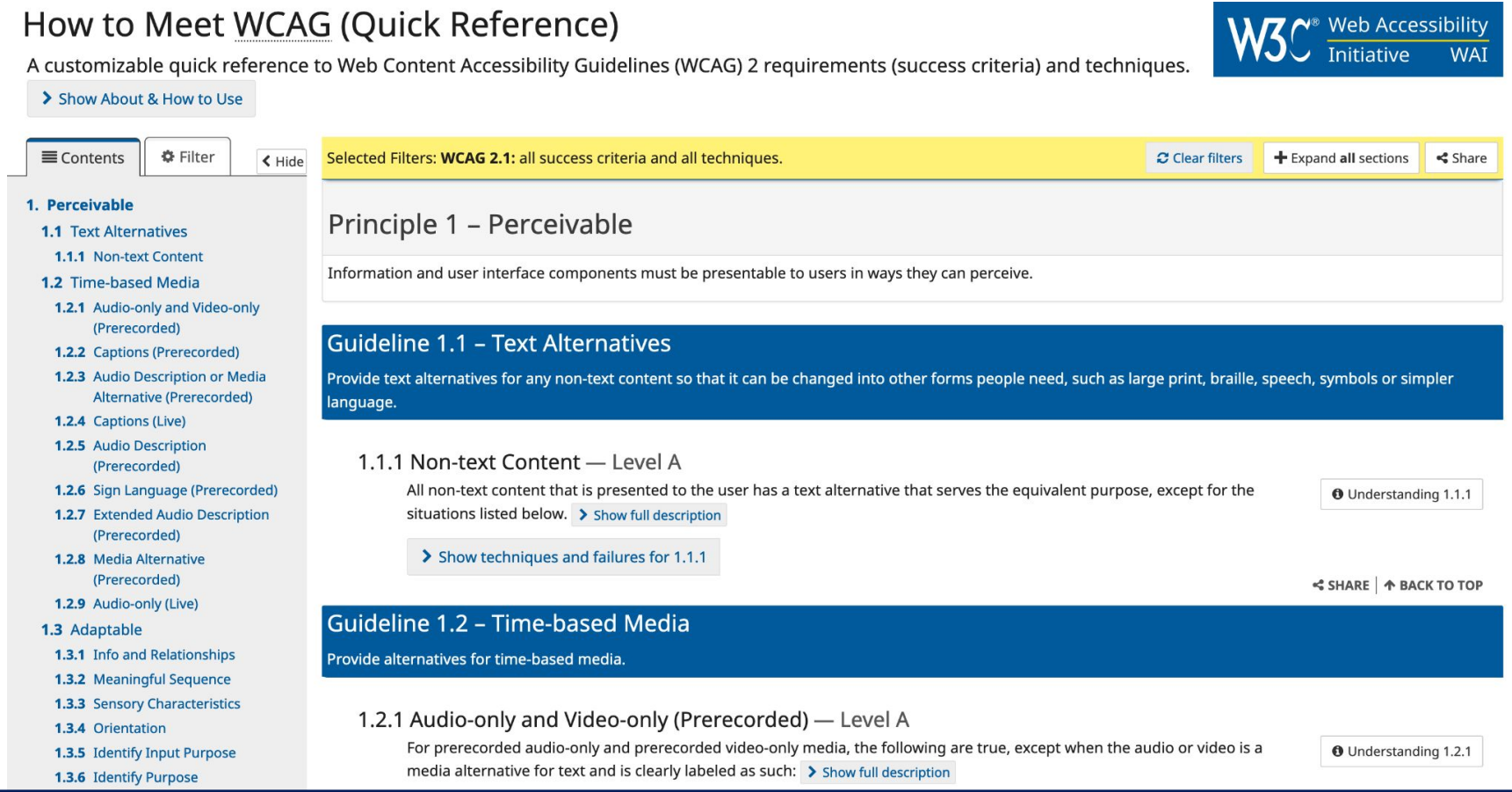
13 Guidelines based on four principles:

*perceivable, operable, understandable, robust*

Success Criteria (aka Levels of Conformance)

| | |
|---|---|
| **A** | basic a11y features |
| **AA** | worldwide legal standard (list of laws) |
| **AAA** | extended a11y features |

https://www.w3.org/TR/WCAG22/




**How to Meet WCAG:** https://www.w3.org/WAI/WCAG21/quickref/

# Testing for a11y

# Start with a checklist

- Define your most important use cases
- Monitor from the beginning
- Or begin optimizing from most important use case

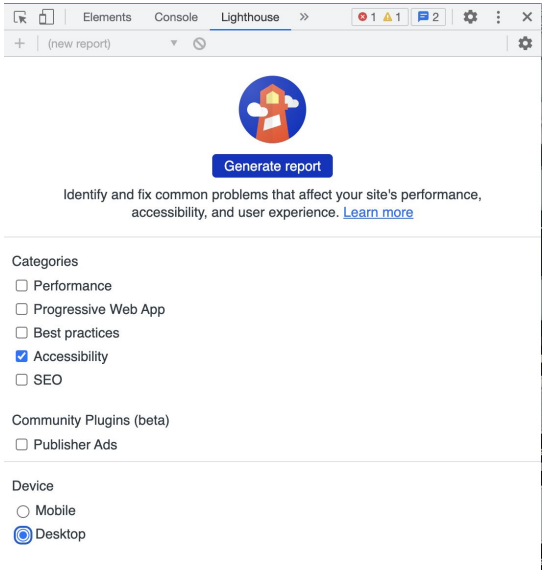
| | Use Case 1 | Use Case 2 | Use Case 3 |
|---|---|---|---|
| Acceptance Criteria 1 | ✅ | ✅ | |
| Acceptance Criteria 2 | ✅ | | |

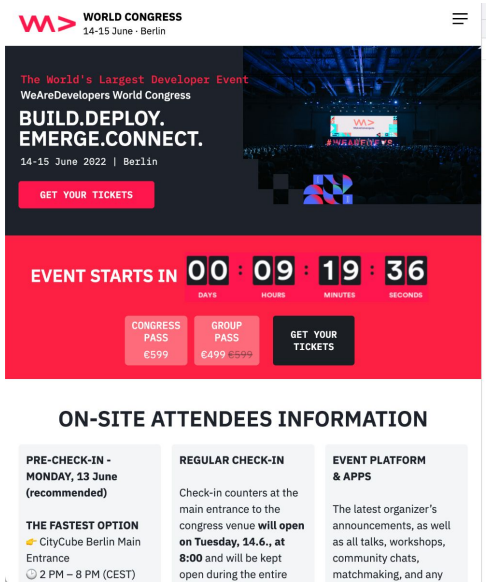
A11y Project checklist: https://www.a11yproject.com/checklist/

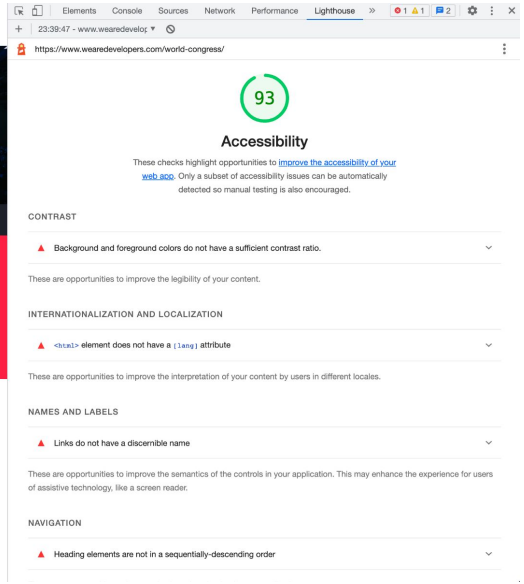
WebAim checklist: https://webaim.org/standards/wcag/checklist

inovex

# Accessibility Audits Dev Tools: Chrome



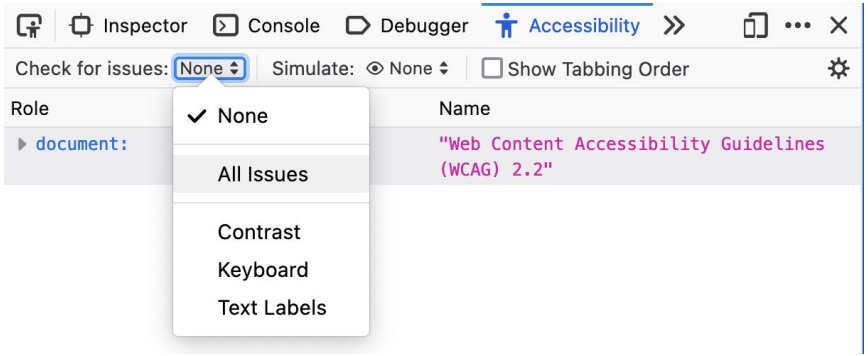




Chrome Dev Tools > Lighthouse > Accessibility

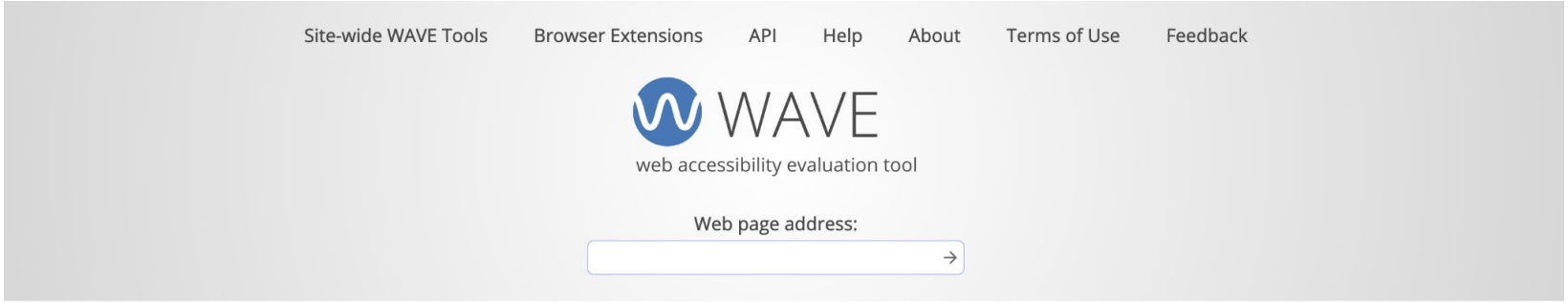
inovex

# Accessibility Audits Dev Tools




Firefox Dev Tools > Accessibility > Check for Issues

inovex




## WAVE Web Accessibility Evaluation Tool

WAVE® is a suite of evaluation tools that helps authors make their web content more accessible to individuals with disabilities. WAVE can identify many accessibility and Web Content Accessibility Guideline (WCAG) errors, but also facilitates human evaluation of web content. Our philosophy is to focus on issues that we know impact end users, facilitate human evaluation, and to educate about web accessibility.

You can use the online WAVE tool by entering a web page address (URL) in the field above. WAVE Chrome, Firefox, and Edge browser extensions are available for testing accessibility directly within your web browser - handy for checking password protected, locally stored, or highly dynamic pages. We also have a Accessibility IMpact (AIM) service, subscription WAVE API, and a stand-alone WAVE API and testing engine for easily collecting data on many pages. If you need enterprise-level reporting and tracking of accessibility, WAVE powers the Pope Tech accessibility tool.


**WAVE Audits:** https://wave.webaim.org/

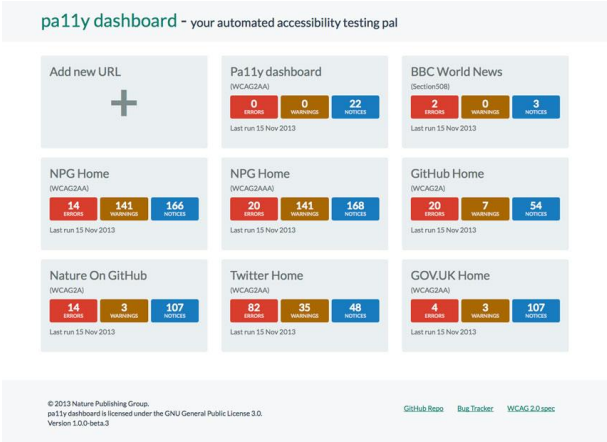
# Automated Testing

## aXe Core

- CLI
- pure JavaScript API
- Wrapper (Selenium, etc..)
- Used in Lighthouse

## pa11y

- CLI
- pure JavaScript API
- Some wrappers
- Dashboard App




BUT: not everything can be automated

manual tests + user studies are a necessity

# Manual Screen Reader Testing

```
document.body.style.clip = 'rect(0,0,1px,1px)';
document.body.style.overflow = 'hidden';
document.body.style.position = 'absolute';
```

Free Screen Readers:

*Windows*: NVDA (www.nvaccess.org), Narrator - native

*Mac*: VoiceOver - CMD + F5 (System Preferences > Accessibility > Voice Over)

*Linux*: Orca (https://help.gnome.org/users/orca/stable/)

BRLTTY (https://brltty.app/)

Voxin (https://oralux.org/)

More about screen reader and their usage: https://bit.ly/3Hli0z1

inovex

<DEMO TIME>

# Color

https://www.w3.org/TR/WCAG22/#distinguishable

## Guideline 1.4 – Distinguishable:

Make it easier for users to see and hear content including separating foreground from background.

## Technical implementation

- Contrast!
- Don't solely rely on color for meaning or information

# Contrast

## Minimal requirement (AA)

4.5 : 1
for small text

3 : 1
large text (>18p, >14p bold)

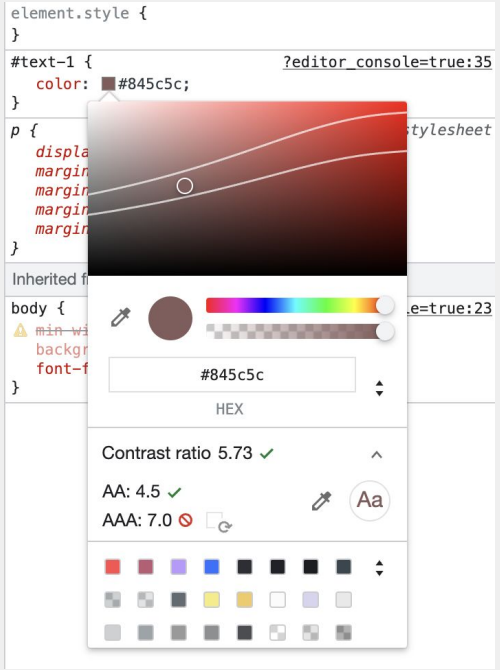
## Enhanced requirement (AAA)

7 : 1
for small text

4.5 : 1
large text (>18p, >14p bold)

# Color Contrast Checkers

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.




https://bit.ly/3OacotR

- Chromium Dev Tools
- Online Tools:
  - Web Aim Contrast Checker
  - https://contrast-ratio.com/
  - ...
- Plugins for most design tools

# Don't communicate through color only

Always use textual information as well


**Bad**


**Good**

# <DEMO TIME>

Color a11y tools in firefox

Using HTML:

# Labelling content

*https://www.w3.org/TR/WCAG22/#text-alternatives*

## Guideline 1.1 – Text Alternatives:

Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.

## Technical implementation

- Provide labels for input fields
- alt attribute for images

alt=”
A couple of sheep on a green hill looking straight into the camera“

# And if your image is just decorative...

```
<img alt="">
```

# If your image stands for a control..

## Focus on the action!

~~*"Shopping Cart"*~~

*"Add to shopping cart"*

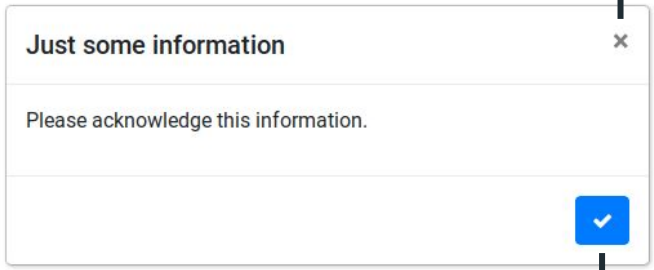
~~*"Star"*~~

*"Add to favorites"*

(Not good)

(Good)

# Also label icon buttons




```
<button>x</button>                    Bad
```

```
<button aria-label="Close">x</button>
```

```
<button>
  <i class="icon-check"></i>          Bad
</button>
```

```
<button aria-label="Confirm">
  <i class="icon-check"></i>
</button>
```

# Form Labels

Address

1234 Main St

Address 2

Apartment, studio, or floor

City

State

Choose...

Zip

Check me out

Sign in

# Label every input correctly

```
[wrong]
<div>
    <div class="label">Name of kitten</div>
    <input type="text">
</div>
```

make it clear which label belongs to which input field

```
[correct]
<div>
    <label class="label" for="kittenName">Name of kitten</label>
    <input type="text" id="kittenName">
</div>
```

# Also correct ...

wrap input in label

```
<label>Name <input type="text"></label>
```

labelling only for assistive technology

```
<input type="text" aria-label="Name" placeholder="Name">
```

multiple inputs with one label

```
<label id="links" for="firstLink">Links</label>
<input type="text" aria-labelledby="links" id="firstLink">
<input type="text" aria-labelledby="links">
```

Using HTML:

# Keyboard Support

*https://www.w3.org/TR/WCAG22/#keyboard-accessible*

## Guideline 2.1 – Keyboard Accessible:

Make all functionality available from a keyboard.

## Technical implementation

- Proper focus handling via keyboard
- Avoid keyboard traps


```
<div class=”button” onclick=”..”>
  Sign me up
</div>
```

# Antipattern!

- No focus via tab

  → `tabindex="0"`

- No keyboard events

  → *Trigger onclick on Enter keydown event*

- Not announced as a button

  → `role="button"`


```
<button class="button">
  Sign me up
</button>
```

## Good! 👌

- Extensive keyboard support for free:
  - Events
  - Focus
- Style as you wish

| | |
|---|---|
| **<button>** | for actions |
| **<a>** | for navigation & links |

# Tabindex

**`tabindex="0"`**

Make an otherwise unfocusable element focusable.

Use its position in DOM for focus order.

**`tabindex="-1"`**

Exclude an otherwise focusable element from focus order.

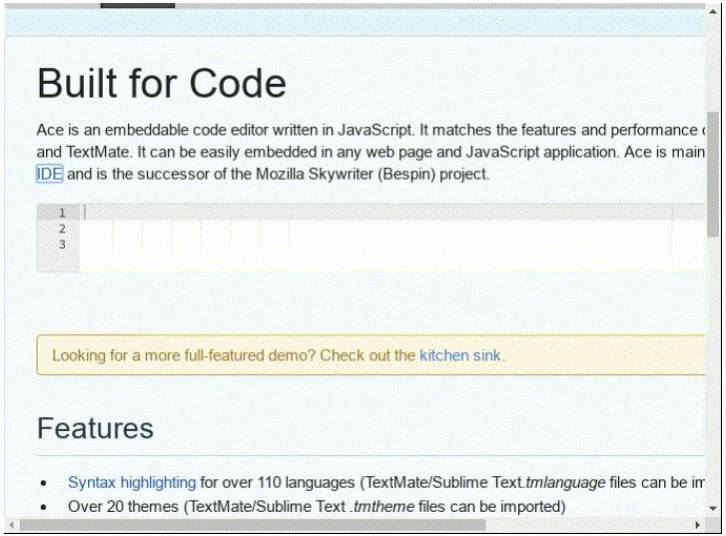
Make an unfocusable element, focusable via **`element.focus()`**

Avoid tabindex > 0

Creates a page global focus order (all >0 in their order, before all with 0).
Very hard to manage and bad for non vision impaired keyboard users.

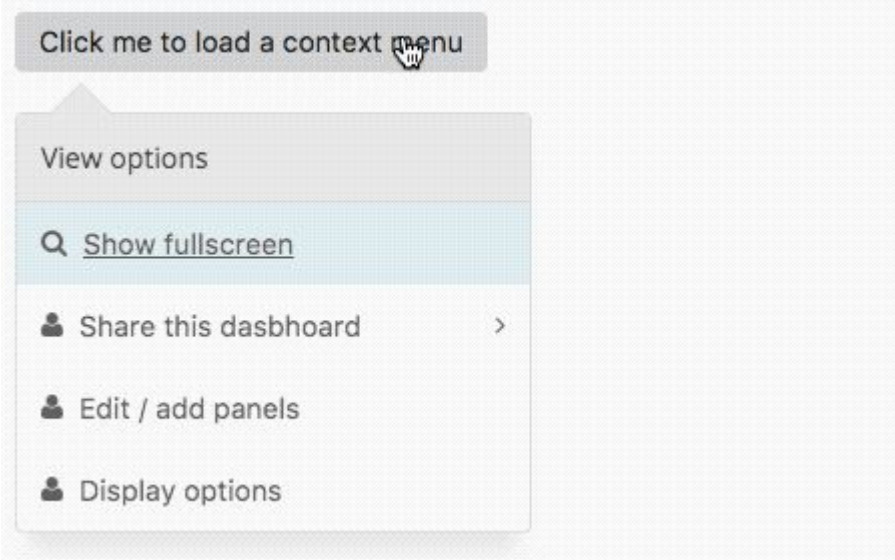
# Don't trap focus

User should be able to navigate within the page without being trapped (e.g. inside a code editor).




# unless you need to trap focus

In modal elements, e.g. dialogs, contextmenus, usually everything, that closes as soon as you click outside of it.



## Make focus ring visible!

(no `outline: none`)

# <LAB TIME>

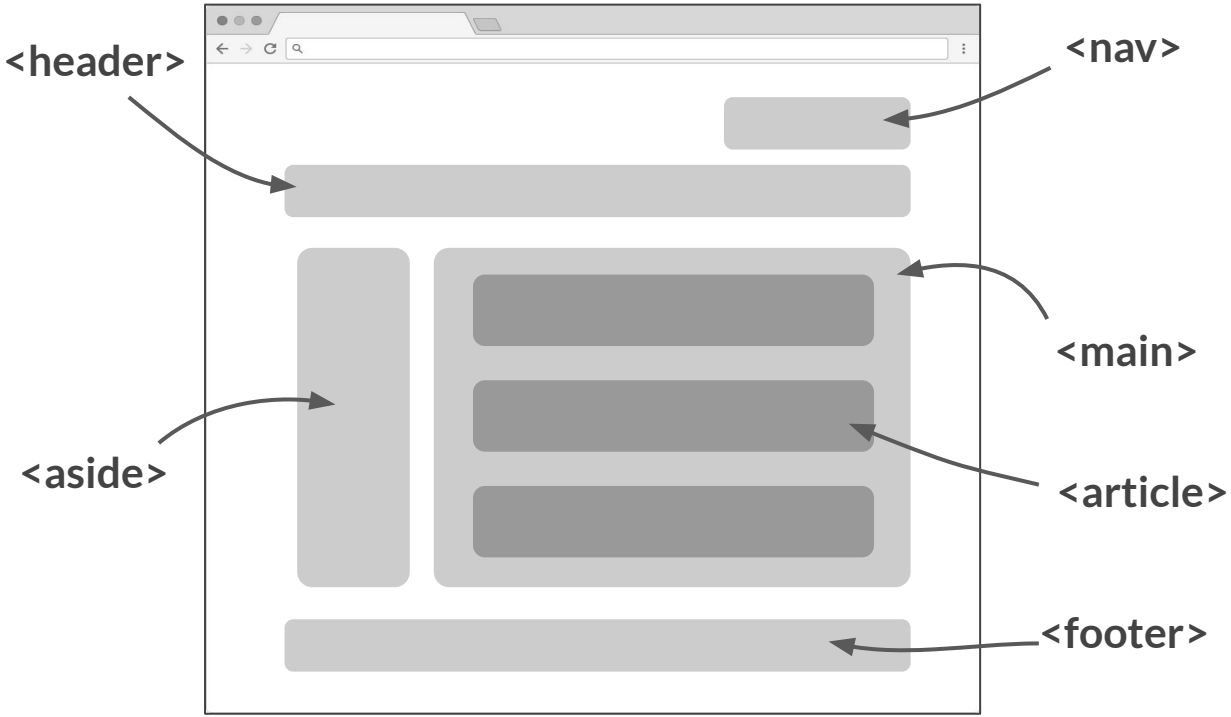
https://github.com/tanjadev/keyboard-a11y

**https://bit.ly/3NPUT2d**

**Code Pen: https://bit.ly/3NRSK6a**

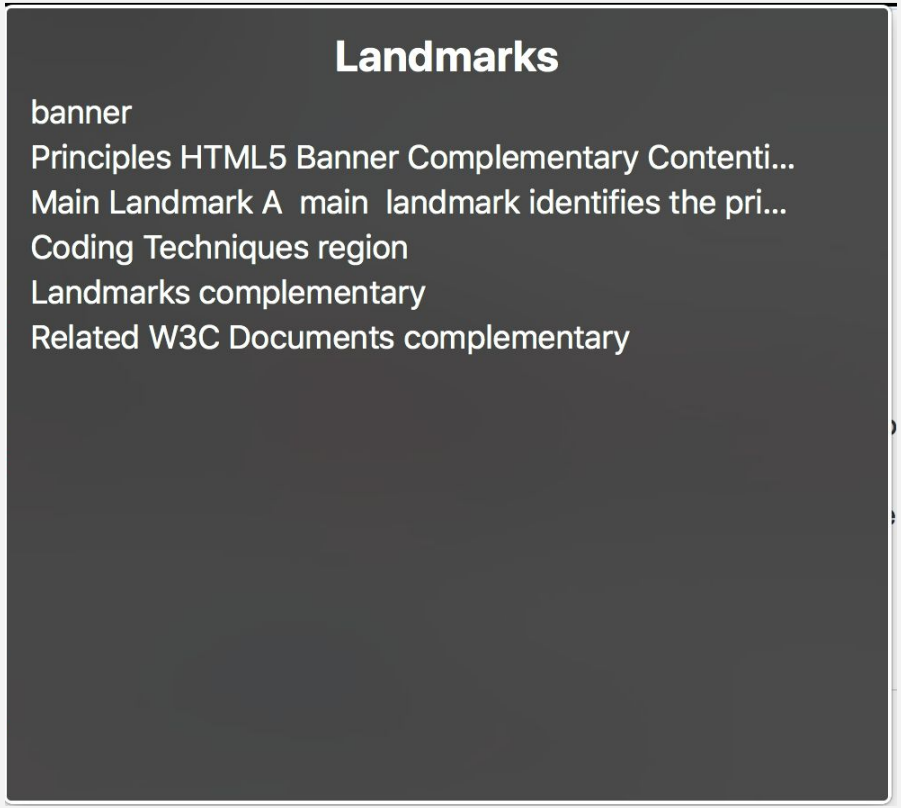
Using HTML:

# Semantic HTML


<header>
<nav>
<main>
<aside>
<article>
<footer>

# Define Landmarks

```
<aside aria-label="Related W3C Documents">
```



# Take care of your Document

- Proper Heading Structure: `<h1> .. <h6>`
- Don't skip levels!
- Max one <h1> per page

- Each Page should have a unique `<title>`

- Define your `<html lang="" >`

# Avoid styling tags

| | | |
|---|---|---|
| **`<strong>`** | | **`<b>`** |
| **`<em>`** | instead of | **`<i>`** |
| **`display: table`** (or smth. else) | | **`<table>`** `for layouting` |

Use HTML to structure your content, leave styling to CSS.

** ** **is a container!**

**** **is an *inline* container!**

Let's talk about

# aria=

W3C Candidate Recommendation Draft

TABLE OF CONTENTS


Abstract

Status of This Document

1. Introduction
1.1 Rich Internet Application Accessibility
1.2 Target Audience
1.3 User Agent Support
1.4 Co-Evolution of WAI-ARIA and Host Languages
1.5 Authoring Practices
1.5.1 Authoring Tools
1.5.2 Testing Practices and Tools
1.6 Assistive Technologies

2. Important Terms

3. Conformance
3.1 Non-interference with the Host Language
3.2 All WAI-ARIA in DOM
3.3 Assistive Technology Notifications Communicated to Web Applications
3.4 Conformance Checkers
3.5 Deprecated Requirements

4. Using WAI-ARIA
4.1 WAI-ARIA Roles
4.2 WAI-ARIA States and Properties


# Accessible Rich Internet Applications (WAI-ARIA) 1.2

W3C Candidate Recommendation Draft 08 December 2021

▼ More details about this document

**This version:**
https://www.w3.org/TR/2021/CRD-wai-aria-1.2-20211208/

**Latest published version:**
https://www.w3.org/TR/wai-aria-1.2/

**Latest editor's draft:**
https://w3c.github.io/aria

**History:**
https://www.w3.org/standards/history/wai-aria-1.2
Commit history

**Implementation report:**
https://w3c.github.io/test-results/core-aam-1.2/

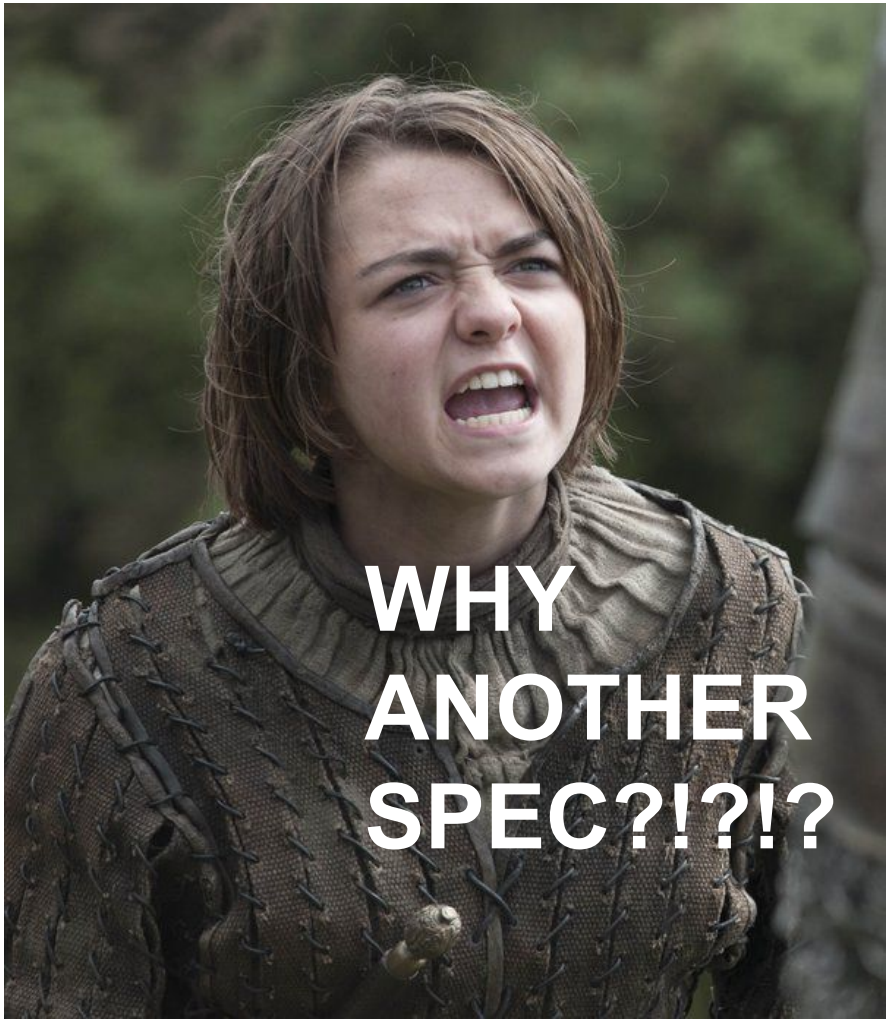
**Latest Recommendation:**
https://www.w3.org/TR/wai-aria-1.1/


**Editors:**
Joanmarie Diggs (Igalia, S.L.)
James Nurthen (Adobe)
Michael Cooper (W3C)

**Former editors:**
Shane McCarron (Spec-Ops) (Editor until 2018)
Richard Schwerdtfeger (Knowbility) (Editor until October 2017)


## Accessible Rich Internet Applications (WAI-ARIA) 1.2

https://www.w3.org/TR/wai-aria-1.2/


WHY
ANOTHER
SPEC?!?!?

Many design patterns do not have a native html equivalent

ARIA fills in these gaps

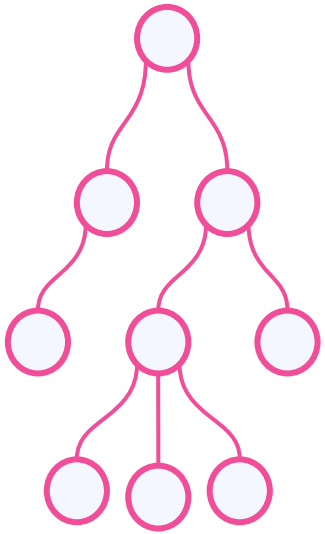
ARIA defines: **roles, states and properties**

**Provides context**

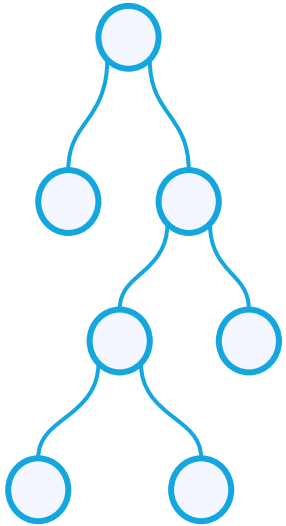
**Extends semantics**

# Where does this information go?

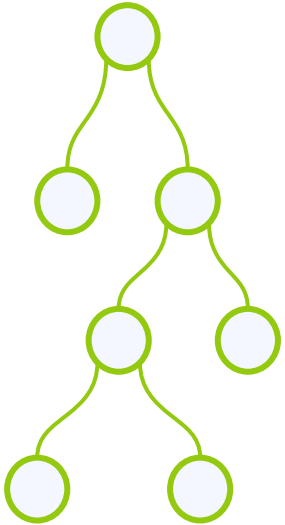
## DOM



## A11Y Tree



## A11Y OS API

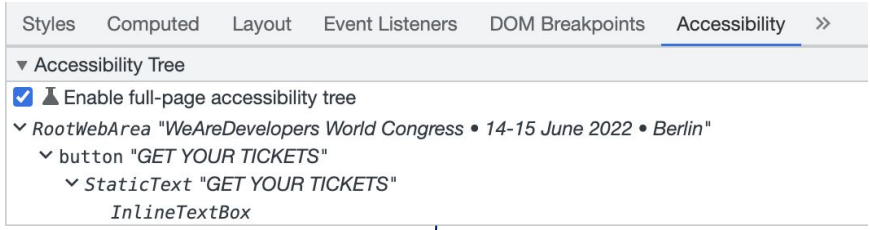


**Linux**: AT-SPI / IAccessible2

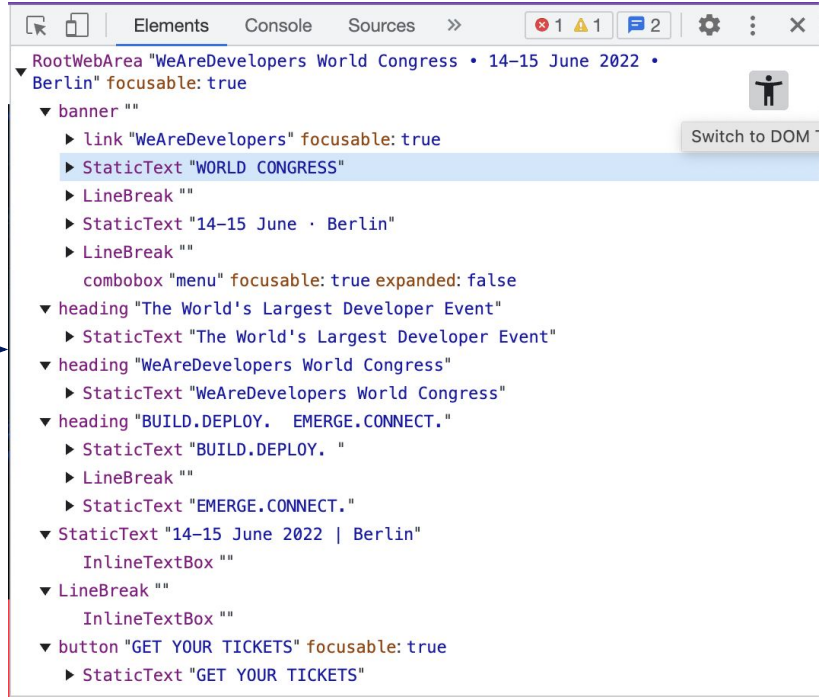
**Windows**: MSAA / IAccessible2/ UI Automation (UIA)

**Mac OS**: Ax/uiA

# Chromium Dev Tools > Accessibility > Enable full-page a11y tree




Reload Page and Click On



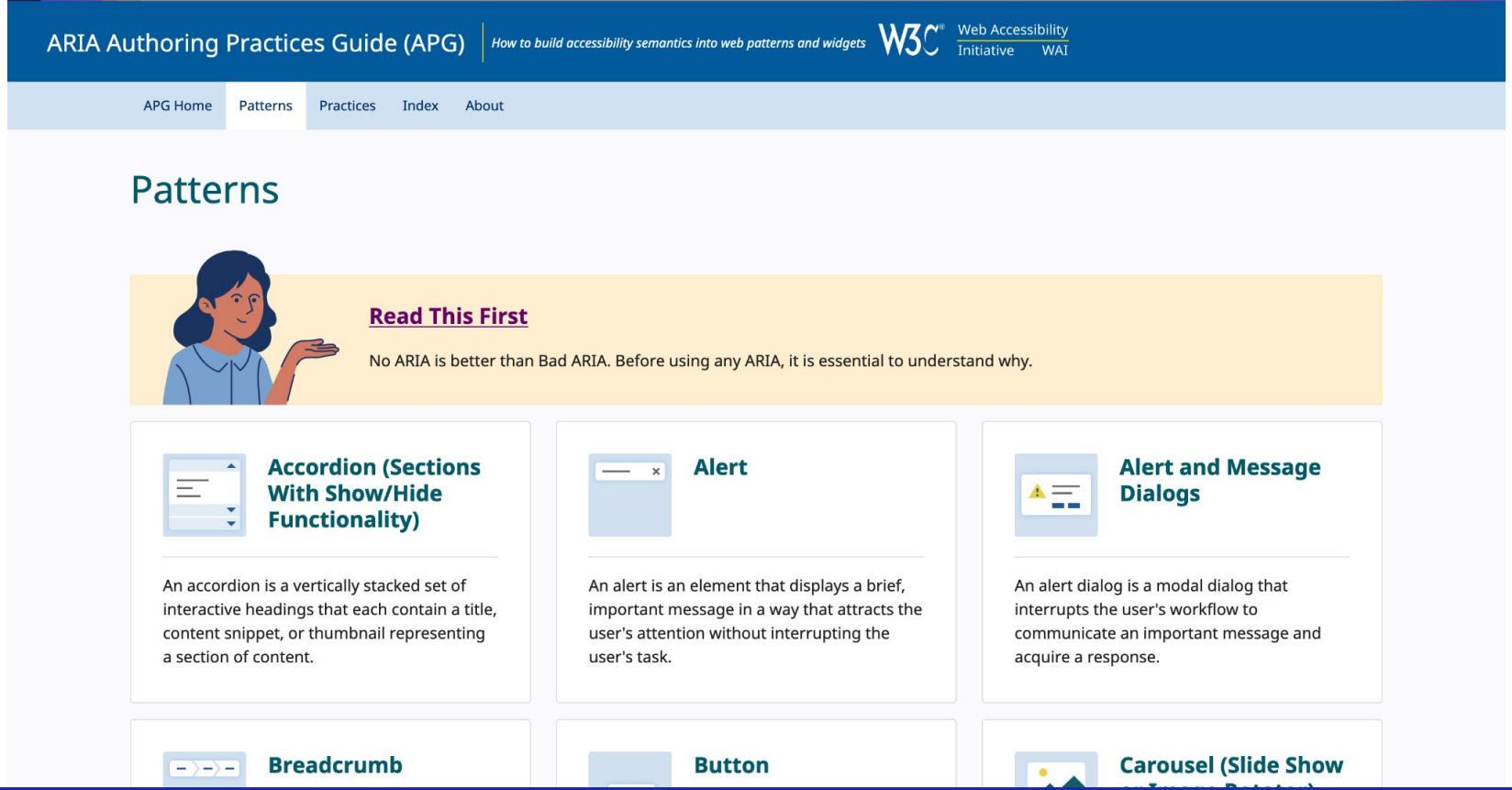
# ARIA usage

*"If you can use a native HTML element or attribute with the semantics and behavior you require, [...] then do so."*

- First Rule of ARIA

Examples:

- Don't use aria-checked, if you can use checked.
- Don't use `role="button"` if you can use `<button>`.




https://www.w3.org/WAI/ARIA/apg/practices/

# Providing Extra Description

**`aria-label="text"`**

Change the name of that element in the accessibility tree.

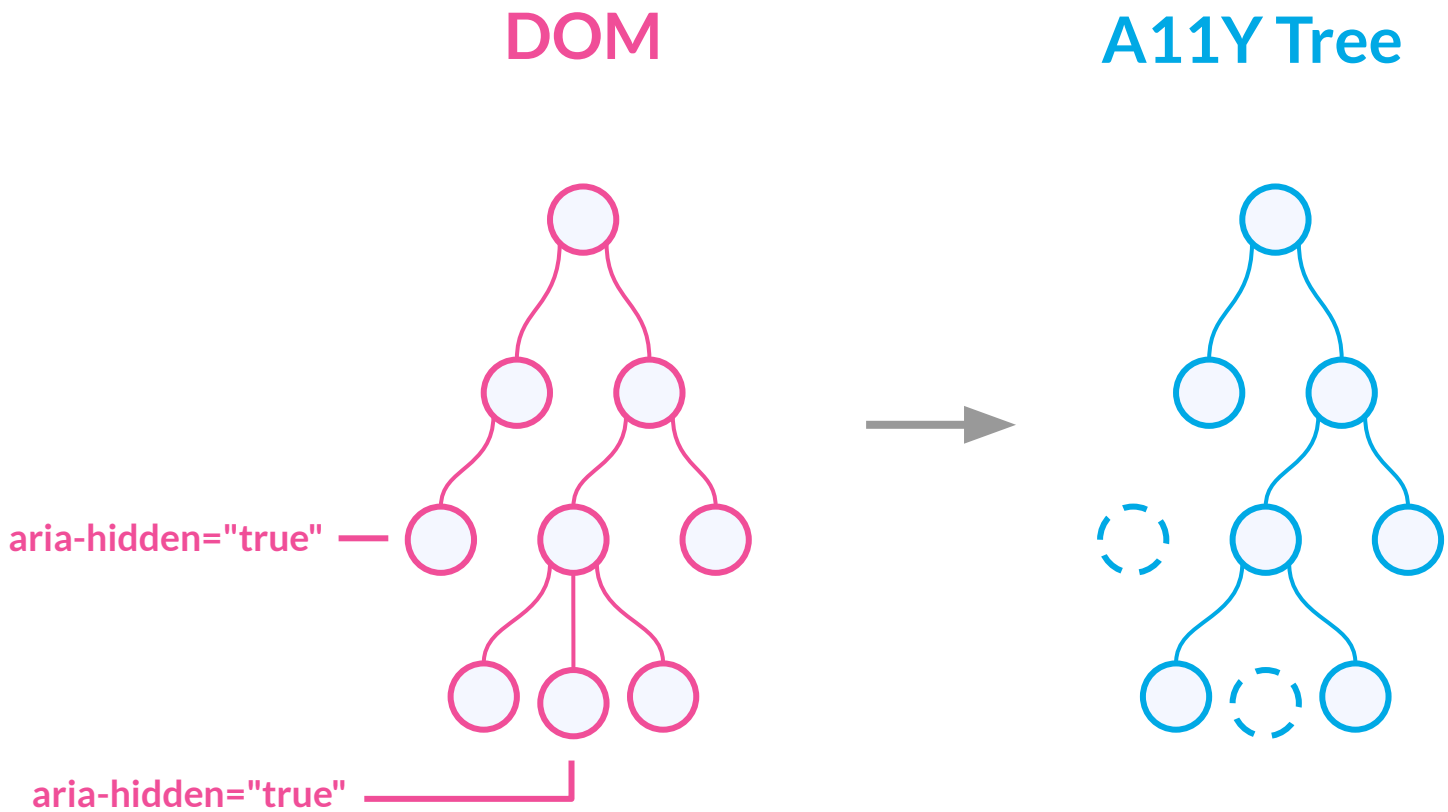
**`aria-labelledby="<idref>"`**

Change the name of that element in the accessibility tree to use the content of a references element.

**`aria-describedby="<idref>"`**

Add additional information for that element to the content of another element.

# Hiding elements from the a11y tree




inovex

# ARIA example: live regions

```
<div class="chat" role="log" aria-live="polite">
  ...
</div>
```

goo.gl/Z59dKQ

# No ARIA is better than bad ARIA

**ARIA is complex**

**Browser and Screen Reader Support of specific attributes is inconsistent:**

ARIA Browser & Screenreader Support:
https://a11ysupport.io/

**Always consult the best practices:**
https://www.w3.org/WAI/ARIA/apg/practices/

# Wrapping up

# TLDR:

✅ a11y benefits everyone

✅ Proper contrast for text and icons

✅ Text alternatives & labels

✅ Tab order is logical

✅ Every action is keyboard accessible

✅ Focus is visible when navigating via keyboard

✅ Use ARIA if you have to

✅ Include automated tests

# Find us, follow us

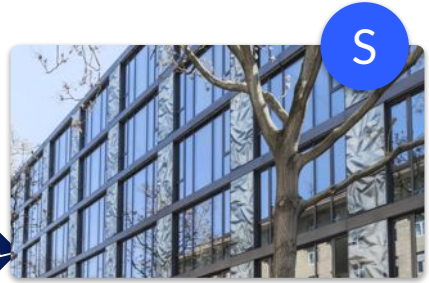
@inovexgmbh

@inovexlife

inovex

# Standorte









inovex

# Thank you for your attention!

Tanja Ulianova
Fullstack Web Developer

inovex GmbH
Ludwig-Erhard-Allee 6
76131 Karlsruhe

tanja.ulianova@inovex.de

inovex

# Extra Lab

1. Checkout **github.com/tanjadev/web-a11y-workshop**
2. Use `npm install` and `npm start` to start web application
3. Visit `localhost:3000`
4. Find "all" accessibility issues (some can be detected automatically, some can't).
5. Make the application properly accessible

If you are stuck: ISSUES.md contains a list of several accessibility issues.