# Helsingin malli

## Accessibility: Implementation in Code, part 1/2

Tero Pesonen / Siteimprove

## Contents

- Tailor-built elements
  - Aria—definitions
  - Navigation order
- Requirements for specific element types
  - Button → Accordion/popup → Menu button
  - Tab list
  - Modal
  - Form controls
- Demo page: http://tpesonen.net/Demo/

Siteimprove

## Additional material

- Menu buttons and menu implementations
  - Document: Module 4 / ”Menu Button Implementation Notes”
  - Demo page → ”Buttons” tab → Menu Button
- Tab List:
  - HDS –notes (link)
  - Document: Modules 3 & 4 / ”Tab List Pattern: Design and Implementation”
    - Additional discussion on the design pattern that supplements HDS and this document
  - Demo page → Tab list and Page Setup
- Modal:
  - Document: Modules 3 & 4 / ”Modal Pattern"
  - Demo page → ”Buttons” tab → Modal Button

## Tailored HTML elements

When implementing non-standard elements, developers must ensure that the element:

1. Describes its **name, role** and **value/state** to assistive technology
2. Is keyboard focusable, if interactive, and
3. Can be activated and interacted with by keyboard
4. Presents either the system default focus indicator or one that meets WCAG 1.4.11 & 2.4.7 contrast requirements.
5. Remember, too: Should the element trigger dynamic content changes (pop-up, menu button), or be a complex element with sub parts (calendar, navigation menu), its internal focus and navigation order is set according to the service design specification, that is, is not random.

## Aria techniques

- Assign the HTML tag with
  - Name
  - Role
  - Value
- Used for non-standard HTML elements
  - HTML tags like <button>, <select>, <img>, … need no Aria definitions
  - But Aria can be used to override standard tag’s properties if necessary
- Aria add descriptions, no programmatic characteristics
  - Recognized and respected (only) by assistive technology
  - Won’t impact visual presentation, or mouse use or keyboard use

# Accessible name

- SC-visible “label” for the element
- Derived from **Author** and/or **Contents** names
  1. If Author label is given by aria-labelling, Contents labels are substituted for it
  2. Aria-label or aria-labelledby name supersede title and ALT
- Author: ARIA labelling, or title or ALT.
- Contents: Text nodes of a DOM element, combined into one

```
<button>This is a Contents type name</button>
```

Author-type naming:

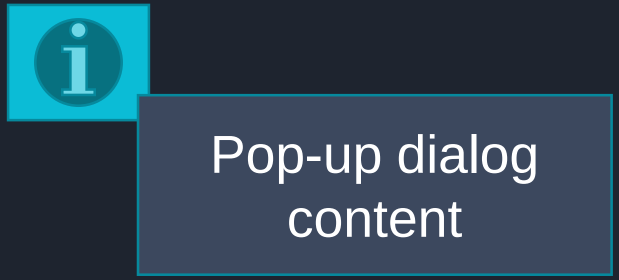
```
<input type=”text” id=”input1” aria-labelledby=”input1_label”></input>
```

## Accessible name

- Aria-label=”…”
  - Attributed directly to the tag; overrides textContent on interactive elements
- Aria-labelledby=”ID”
  - The name is derived from another element’s accessible name (compare <label for>)
- Aria-describedby=”ID”
  - The name is supplemented with an additional description, derived from another element’s name
- Div or span with only textContent cannot be aria-labelled

```
<a href=”…” aria-label=”Read article XYZ (opens a PDF)”>Read this
article<img class=”PDF_Glyph” alt=”PDF document”></a>
```

```
<a href=”…” aria-describedby=”Heading_article_XYZ”>Read the srticle
      <img class=”PDF_Glyph” alt=”PDF document”></a>
```

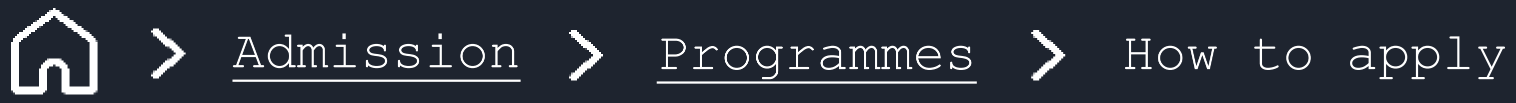


Pop-up button: Requires Author type name

```
<button aria-label=”Help about XYZ”><img
src=“…” alt></button>
```

```
<button><img src=“…” alt=“Help about
XYZ”></button>
```

NOTE: This won’t work!



```
<a href=“…”><img src=“…” alt=“Home page”></a>
```

```
<a href=“…” aria-label=“Home page”><img
src=“…” alt=“”></a>
```

If an image link has ALT, SC will announce the element as “image link”. If aria-label is used with an empty ALT, the presence of an image is not exposed.

```
<span aria-label=“Current
page:“ class=“…”>How to
apply</span>
```

1. Element has only text node(s) → aria-label has no effect
2. Aria-label will **replace** text nodes when applied

## Role

- Role=”…”
  - Button
  - Tab, tablist, tabpanel
  - Listbox, combobox, option
  - Radio, radiogroup
  - Region, main, banner, contentinfo, navigation
  - Jne.

```
<div id="radiogroup1" role="radiogroup" aria-labelledby="radiogroup1-label"
class="RadioGroup">
      <span id="radiogroup1radio0" role="radio" class="RadioButton" aria-
      checked="true" tabindex="0">First Radio Button</span>
      <span ...>
</div>
```

## State / value

- Required for instance in
  - Buttons that open/close a menu, accordion, pop-up
  - Radio button and checkbox checked status
  - Tab, in a tab list, is selected/open or not-selected
  - Etc.
- Aria techniques
  - Aria-expanded
  - Aria-selected
  - Aria-checked
  - Aria-pressed
  - Aria-current
  - jne…

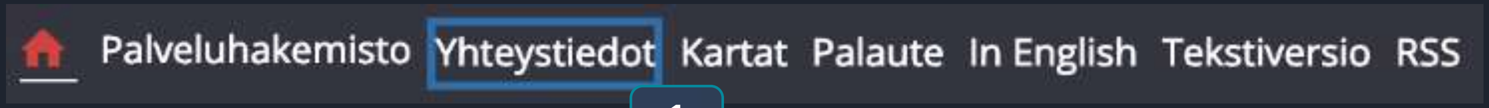
```
<button aria-pressed=”true” class=”ToggleButton”>
Toggle button</button>

<span role=”checkbox” aria-checked=”false” ...
</span>
```

## Keyboard navigation and tabindex

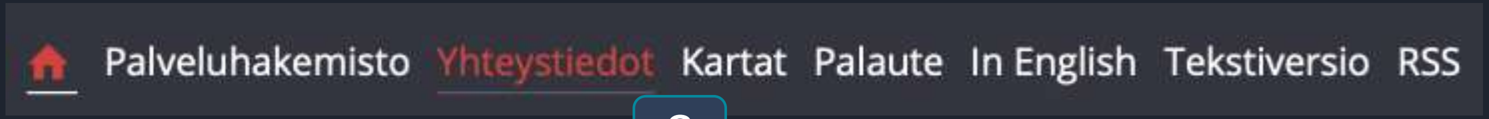
- Tabindex attribute
  - 0 : the tag is focusable
  - -1 :
    - Unfocusable by keyboard, but
    - The tag can be focused programmatically (JS: DOMNode.focus();)
  - >1 : Altered focus order (not usually needed or recommended)
- If the tag is 1) interactive, but 2) has been created by a span, div or other non-interactive HTML tag, it must be assigned tabindex=0.
- Adjust navigation and focus order by DOM order, not by tabindexes

# Visualizing keyboard focus

- If the element or focus indicator is tailored, developers must ensure it is shown and has sufficient contrast (3:1) *WCAG 2.4.7, WCAG 1.4.11*
  - TAB key → focus hops along the page (1)
  - Default/system native indicator always passes WCAG
  - Mouseover—”focus” (2) not mandatory; contrast unspecified by WCAG
- CSS
  - `:focus` state
  - E.g. `outline` or `text-decoration` properties



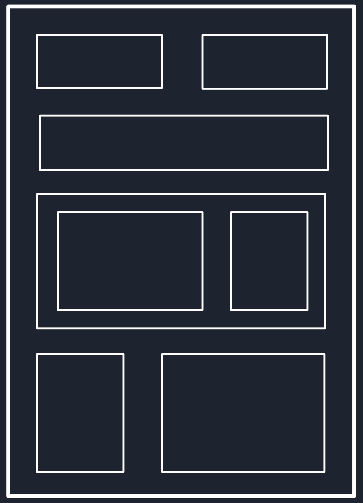
1

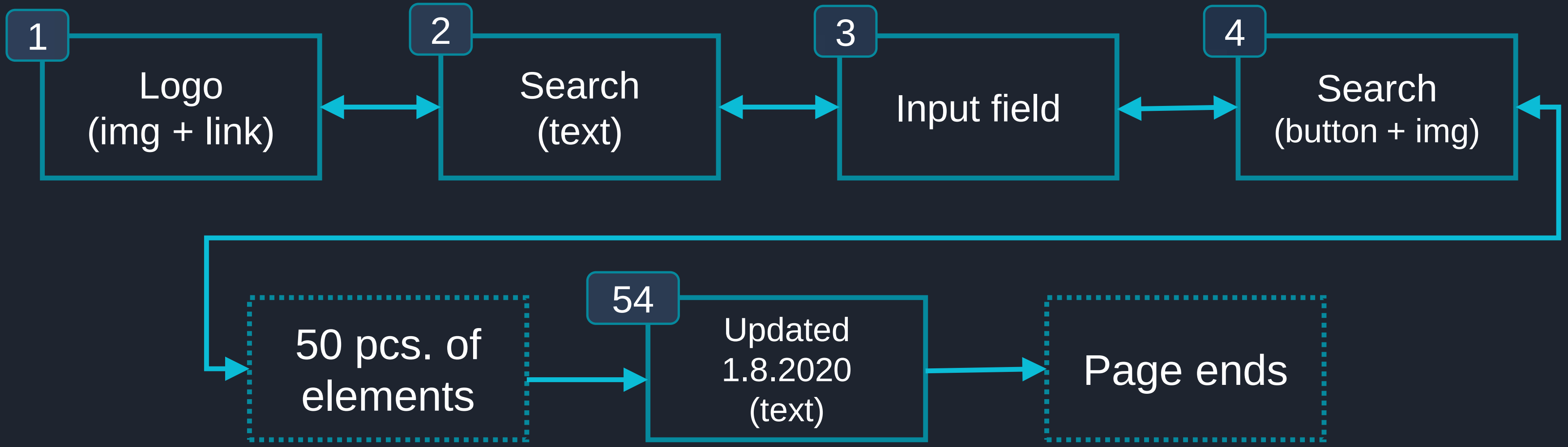


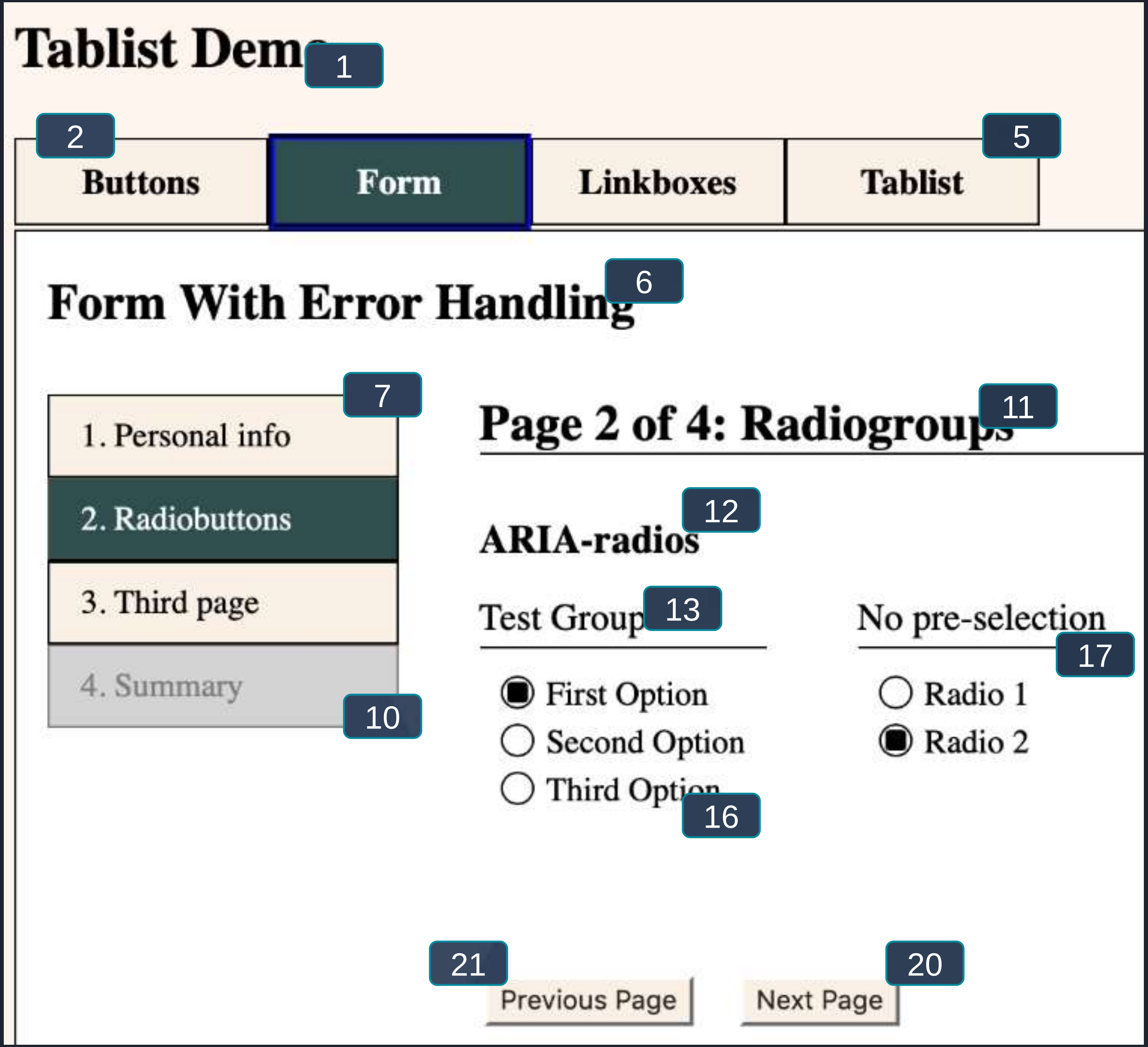
2

# Reading/focus order

- Determines the order in which individual elements on the page are observed by keyboard navigation and assistive technology
  - Keyboard navigation: WCAG 2.4.3 Focus order
  - Screen reader: Additionally WCAG 1.3.2 Meaningful sequence






Tablist Demo 1

2 Buttons | Form | Linkboxes | Tablist 5

Form With Error Handling 6

7
1. Personal info
2. Radiobuttons
3. Third page
4. Summary
10

Page 2 of 4: Radiogroups 11

ARIA-radios 12

Test Group 13
- ● First Option
- ○ Second Option
- ○ Third Option 16

No pre-selection 17
- ○ Radio 1
- ● Radio 2

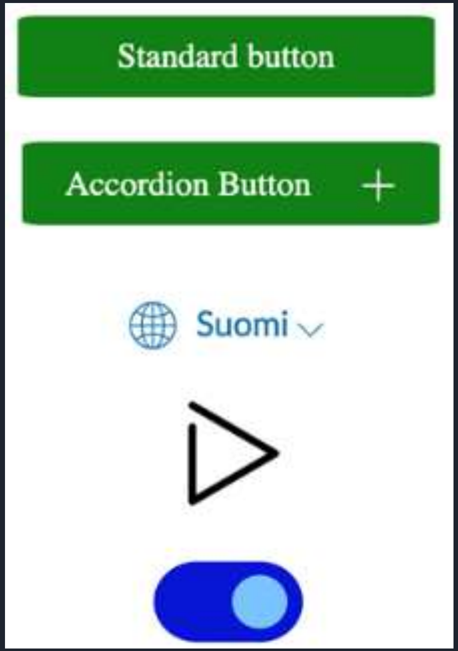
21 Previous Page | Next Page 20

- Pay attention to, for instance:
  - When using columns
  - Link and navigation segments that lie parallel to main content
  - Submit and other buttons at the end of a form (no relevant content should be placed after)
  - Tabs and tabpanels
  - Menus, pop-ups placed in navigation order immediately following their trigger buttons
  - If a form presents dynamically added/removed controls, they should appear after the trigger point, if possible

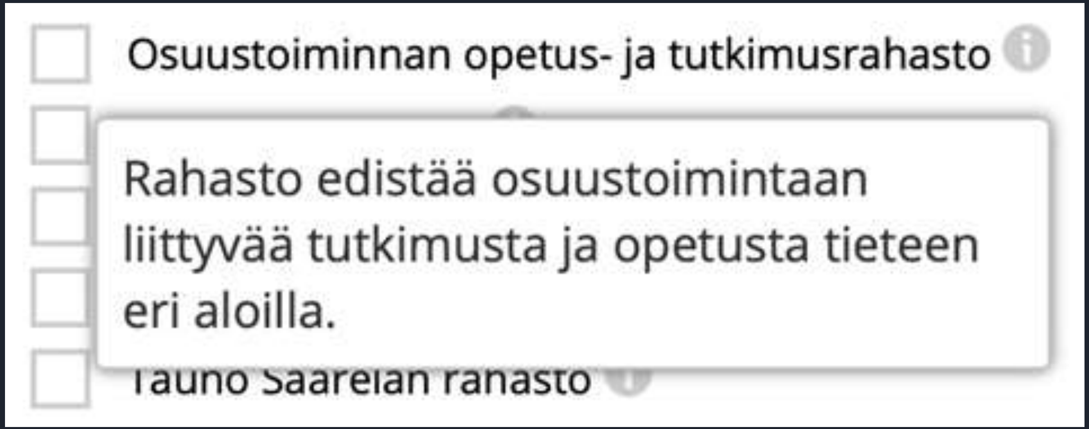
## Button

- Corresponds to: <button> tag
  - Submits a form or transits between form pages
  - Shows/hides a menu, accordion, pop-up, etc.
  - Triggers a modal
- Compare to links (anchor tag), which
  - Transfers focus
  - Opens a new URL
- The purpose of the element dictates its role as button or link
  - Visual styling should be applied independently
- Exception: Tabs in a tab list contain “tab” elements, not buttons.

Buttons may comprise text, text and graphics, or only graphics (glyphs, symbols)



The icon when focused or clicked dynamically alters the page content (shows a pop-up) → The icon functions as a button, not as a link or mere image

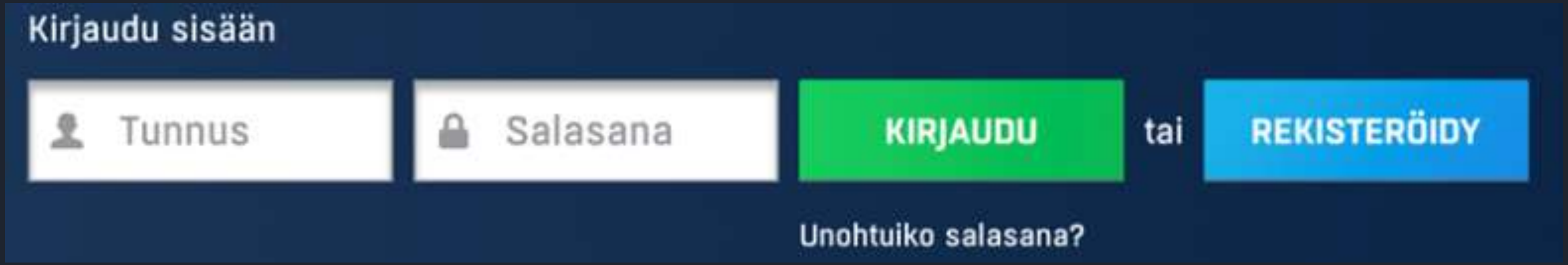

Osuustoiminnan opetus- ja tutkimusrahasto

Rahasto edistää osuustoimintaan liittyvää tutkimusta ja opetusta tieteen eri aloilla.

Tauno Saarelan rahasto

Log in = button, Register = link (opens a new page)

Ghost--button



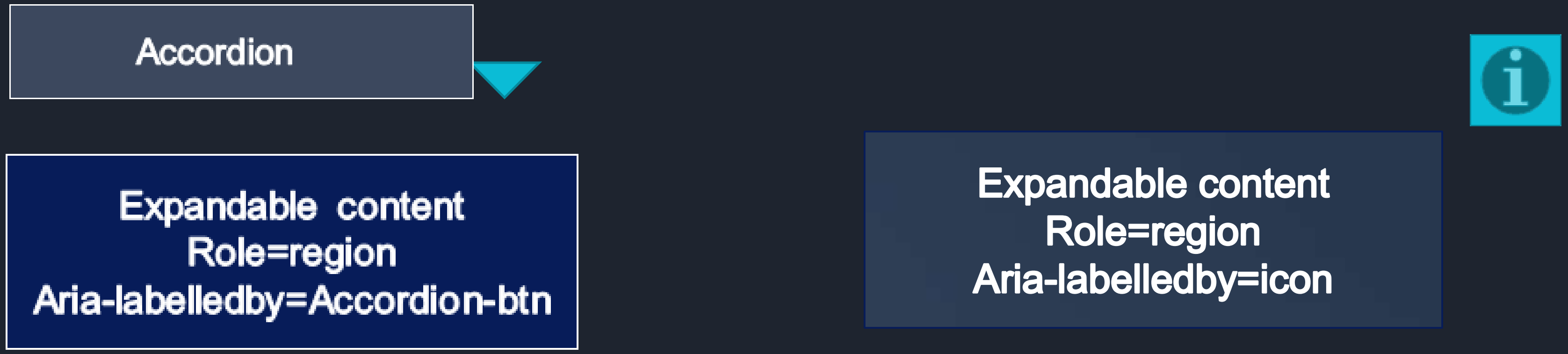
# Basic button (“action button”)

- Role=”button”
- Tabindex=”0”
- KeyboardEvent handler (keypress) → can be activated by Enter (event.keyCode === 13 || event.key === Enter)
- Screenreader: keys (e.g. “space”) need no keyboard handlers, as screenreaders also send a click event.

```
<span id=”…" class=”…" role="button" tabindex="0">Simple button</span>
```

## Accordion

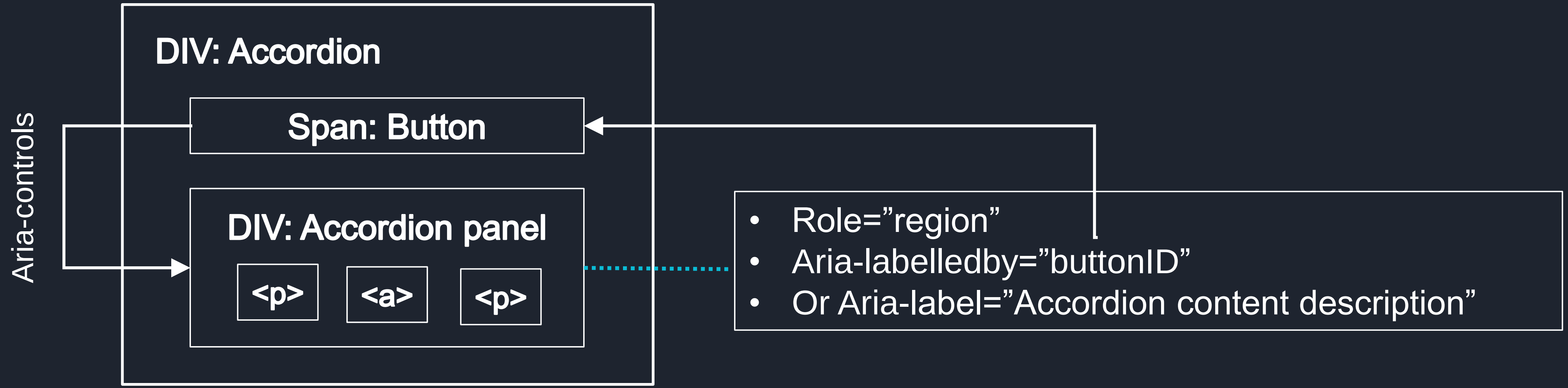
- Standard button is supplemented with
  - aria-expanded=”true”/”false”
    - NOTE: Expanded status is assigned to the button, not the content that is expanded!
  - Aria-controls=”ID”, where ID refers to the expanded content
- Accordion == Pop-up != Modal (see later slides)



```
<span id=”…" class=”…" role="button" tabindex="0” aria-expanded=”true”
aria-controls=”accordion-panel”>Accordion button</span>
```

# Accordion: DOM Structure

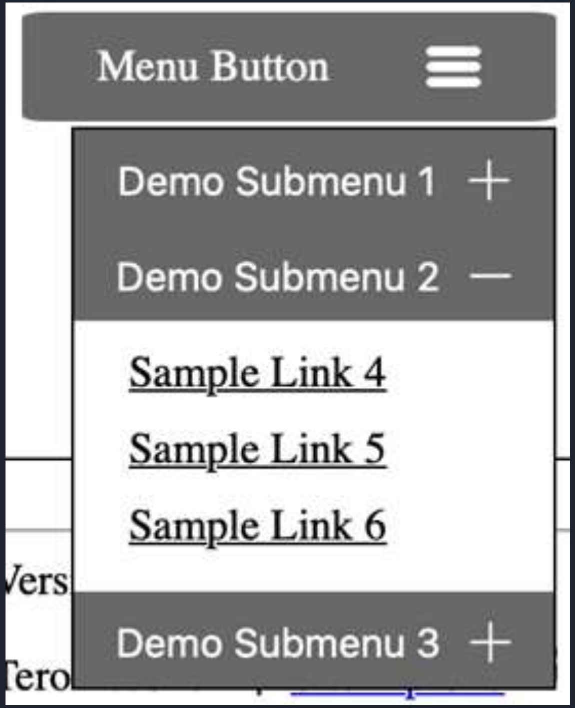
- Content typically presented as div or section
  - Sibling to the button, not a child
  - Follows immediately in navigation (DOM) order to the button
  - Recommended: Has a matching CSS *display* value (e.g. *block*) as its trigger button
- Accordion content can be demarcated by role=region or <section>



## Menu button

- Implementation similar to Accordion
- Additionally: Give the tag Aria-haspopup=”true”
  - Signals: “This is a menu kind of dynamic content”

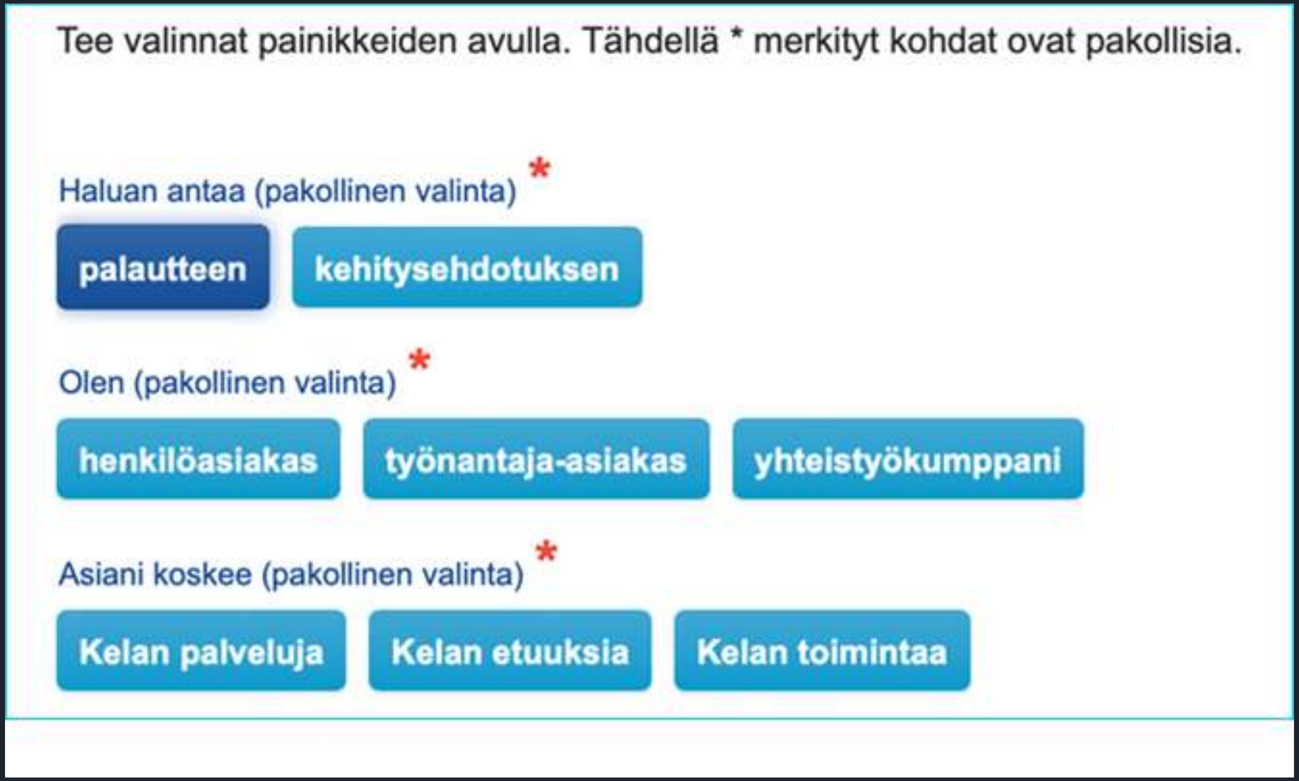
  → Screen readers describe the value in a standard fashion
  - Value remains unchanged when the menu is open/closed
- Menu can encompass submenu(buttons)/accordions



```
<span id="demobutton3" role="button" aria-haspopup=”true" tabindex="0"
aria-expanded="true" class=”…" style=”…" aria-
controls="demobutton3_panel"> Menu Button
      <img class="MenuGlyph" alt="" src=”…” width="20" height="20">
</span>
```

# Toggle button

- Button, which has Aria-pressed=”true”/”false”
- Corresponds semantically to a checkbox.



Tee valinnat painikkeiden avulla. Tähdellä * merkityt kohdat ovat pakollisia.

Haluan antaa (pakollinen valinta) *

Olen (pakollinen valinta) *

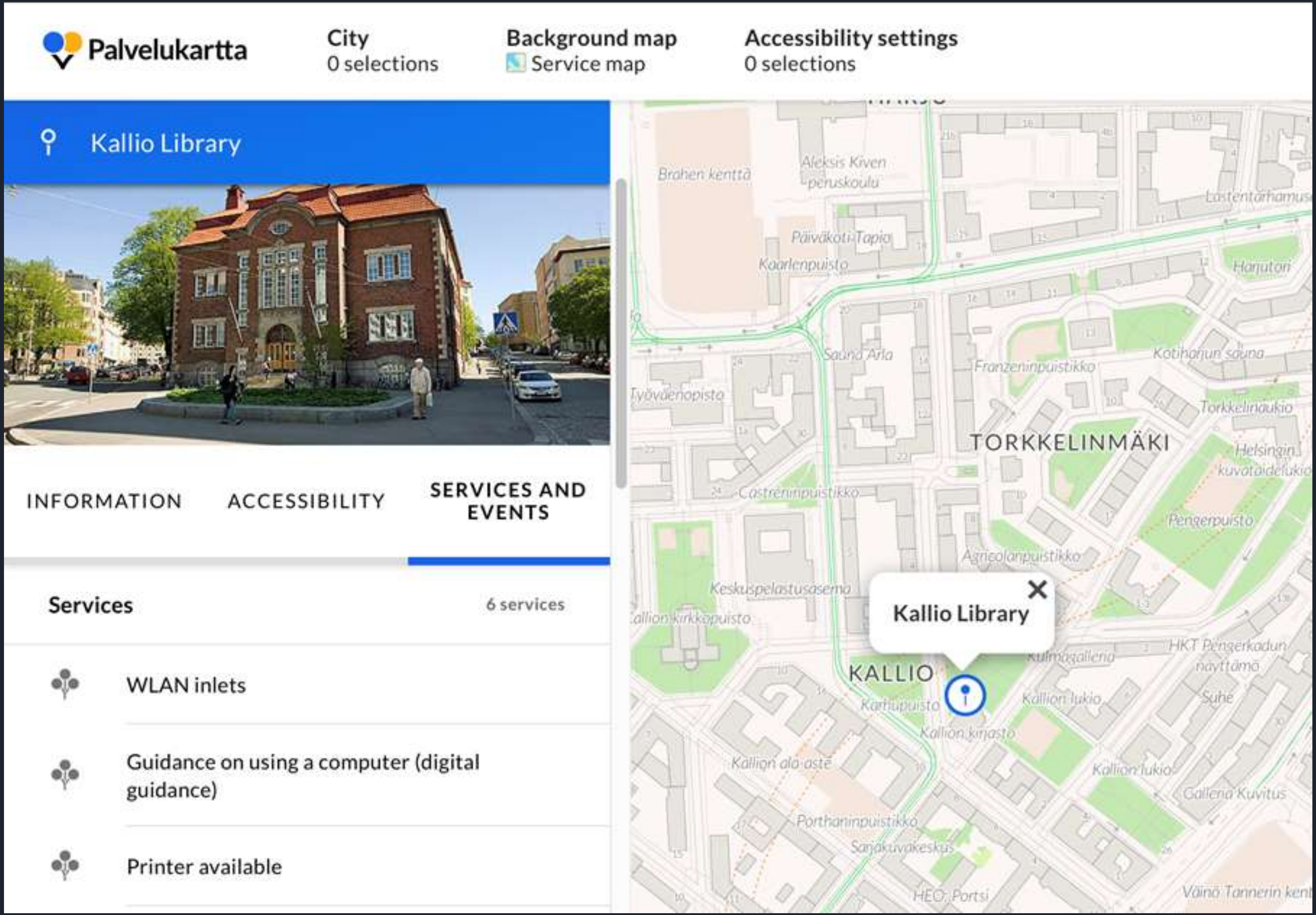
Asiani koskee (pakollinen valinta) *

# Tabs

- Divides content into parallel but optional segments both semantically and visually

  => Has to be defined correctly for assistive technology
- Proerties
  - All of the tabs are active and can be opened
  - Each tab provides categorically similar content
  - Opening a tab will only alter content, not context → not a link
  - Only one tab can be open at a time, and one tab is always open
- A section can be a tab set even though it appears visually not like so; conversely, content that look like tabs can, in fact, function as links.
  - WCAG 1.3.1 Info and relationships

## Information about a location: Tabs segment the information into categories that when displayed occupy a shared space



Palvelukartta

City
0 selections

Background map
Service map

Accessibility settings
0 selections

Kallio Library

INFORMATION | ACCESSIBILITY | SERVICES AND EVENTS

Services 6 services

WLAN inlets

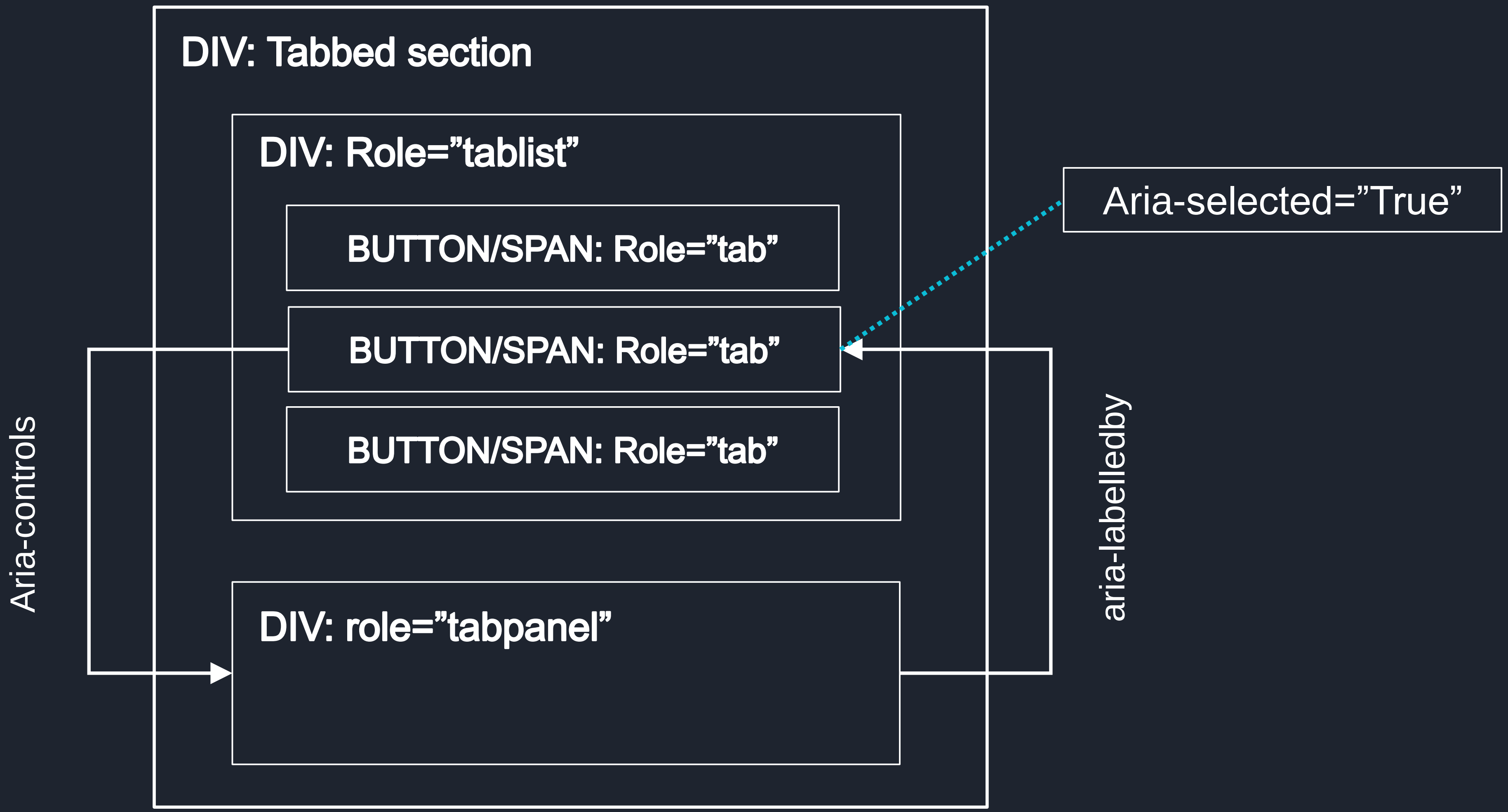
Guidance on using a computer (digital guidance)

Printer available

## Tabs: Aria definitions

- Tab list: role=”tablist”
  - Aria-label can be added
  - Individual tabs: (n pcs.): role=”tab” (NOT role=”button”)
    - Aria-controls=tab panel ID
    - Aria-selected=”true”/”false”: Is the tab currently “open”?
- Tab panel: role=”tabpanel”
  - Aria-labelledby=tab, which presently has aria-selected=true

## Tabs: Aria definitions



## Tabs: Focus order and keyboard navigation

Three approaches

1. Tabs as independent, focusable “buttons”
   - Each tab is focusable individually by TAB key in the DOM
   - Implemented as if a list of buttons
   - Many users find this intuitive on the web
2. Only tablist is focusable (W3C recommended)
   - The tablist is TAB-key-focusable, not individual tabs
   - User moves focus within the tablist with left/right arrow keys once tablist has been focused with a TAB key
   - Activating a tab will open it

## Tabs: Focus order and keyboard navigation

3. Selection follows focus (also W3C recommended)
   - Like method #2, but:
   - When the user moves focus within the tablist with arrow keys, each tab opens automatically when focused
   - Implemented like a radiogroup (see later slides)

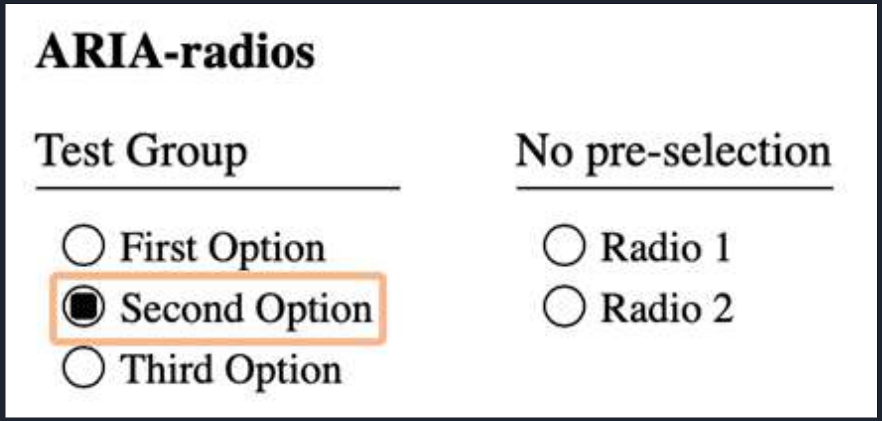
See: http://tpesonen.net/Demo/Tablist/

- Main navigation as a tablist method #2
- Tablist tab shows also method #3

# Tailor-built form controls

- Checkbox
  - Role=”checkbox”
  - Aria-checked=”true”/”false”
- Radiobutton group
  - Role=”radiogroup”
- Radio button
  - Role=”radio”
  - Aria-checked=”true”/”false”
- Remember to add where necessary
  - Tabindex=”0”
  - Keyboard focus indicator
  - Keyevent handlers

Native input type=”radio” with fieldset and legend is significantly easier to implement than an aria radio group. Native HTML elements are accessible “out of the box”.



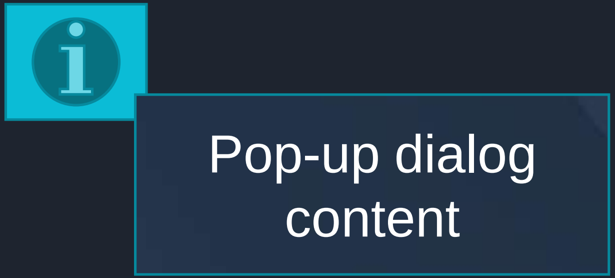
## Radiogroup: Focus management

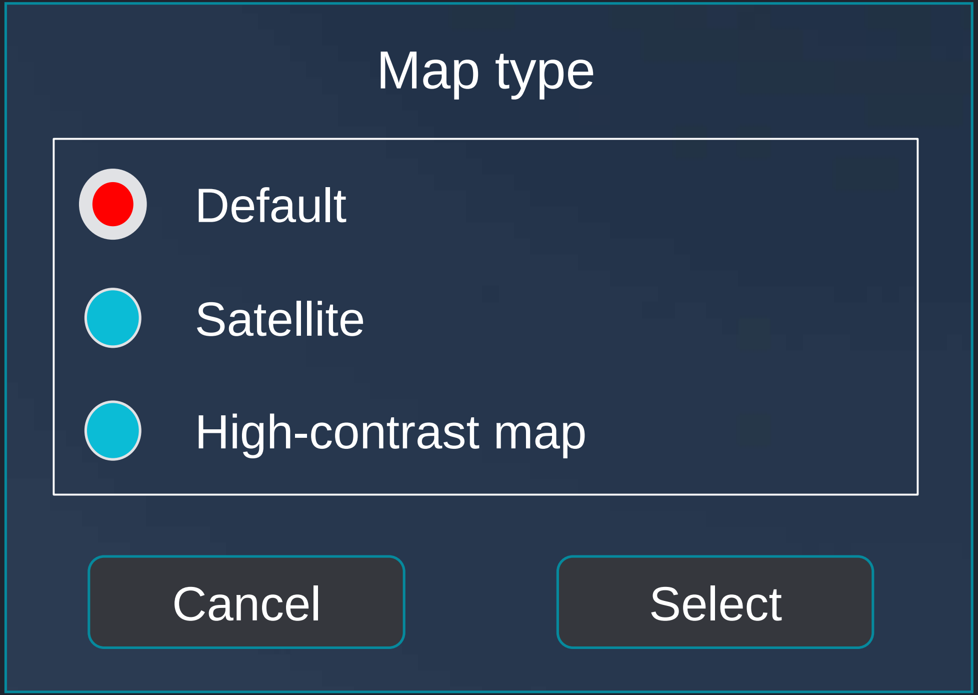
- When a radio is in a checked state, it has
  - aria-checked=“true”, otherwise aria-checked=“false”
  - Tabindex=“0”, otherwise tabindex=“-1”
  - If the group does not contain a pre-checked radio, the first radio has tabindex=“0”.
- When a radio is currently focused (CSS :focus state from onfocus event), and the user presses up or down key
  - Assign the current radio aria-checked=“false”
  - Assign the current radio tabindex=“-1”
  - Assign the targeted radio aria-checked=“true”
  - Assign the targeted radio tabindex=“0”
  - Move focus to the target radio (JS DOMNode.focus())
- If a click event occurs, repeat the above
  - Group retains keyboard-navigatable, coherent state

# Modals and pop-ups

- Dynamic content can be implemented as either a pop-up or modal
- Distinction is crucial. Impacts
  - Keyboard navigation
  - Screen reader use

  => Use-scenario design
- Must be heeded in implementation
  - Designer decrees which type the implementation should be





## Pop-up

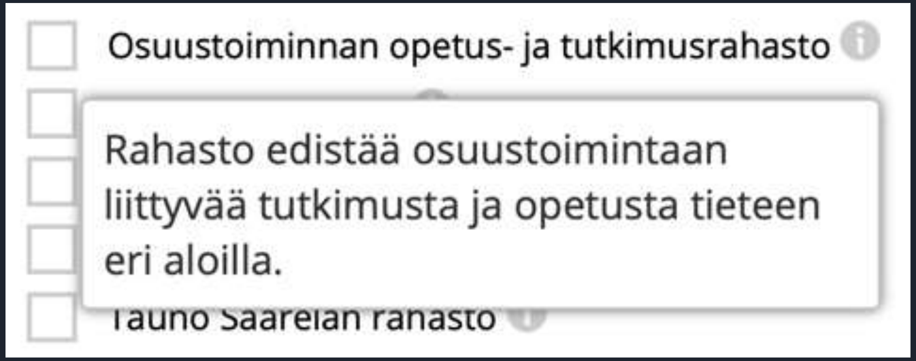
- Corresponds logically to an accordion
  - Part of the page normal navigation order
  - Pop-up can be closed or open at any given time
  - If the button that triggered the pop-up remains visible, the pop-up requires no closing button
- Often triggered by an icon
  - Pop-up opens when the icon is clicked or moused over
- Heading may suit poorly: Consider, at any rate, adding a region definition

## Pop-up & WCAG 1.4.13

- If an element triggers a pop-up when focused or hovered with a mouse:
  - The pop-up can be closed without removing focus from the trigger
    - Exception: Error messages, and situations where the pop-up will not cover or replace other content
  - The pop-up area can be focused without the pop-up disappearing
    - That is, even if the trigger loses focus/hover, in this case the pop-up remains visible
  - The pop-up stays visible till the user closes ii or its content is no longer relevant

Example: Popups to which the criterion applies:

Link: University of Helsinki donations form



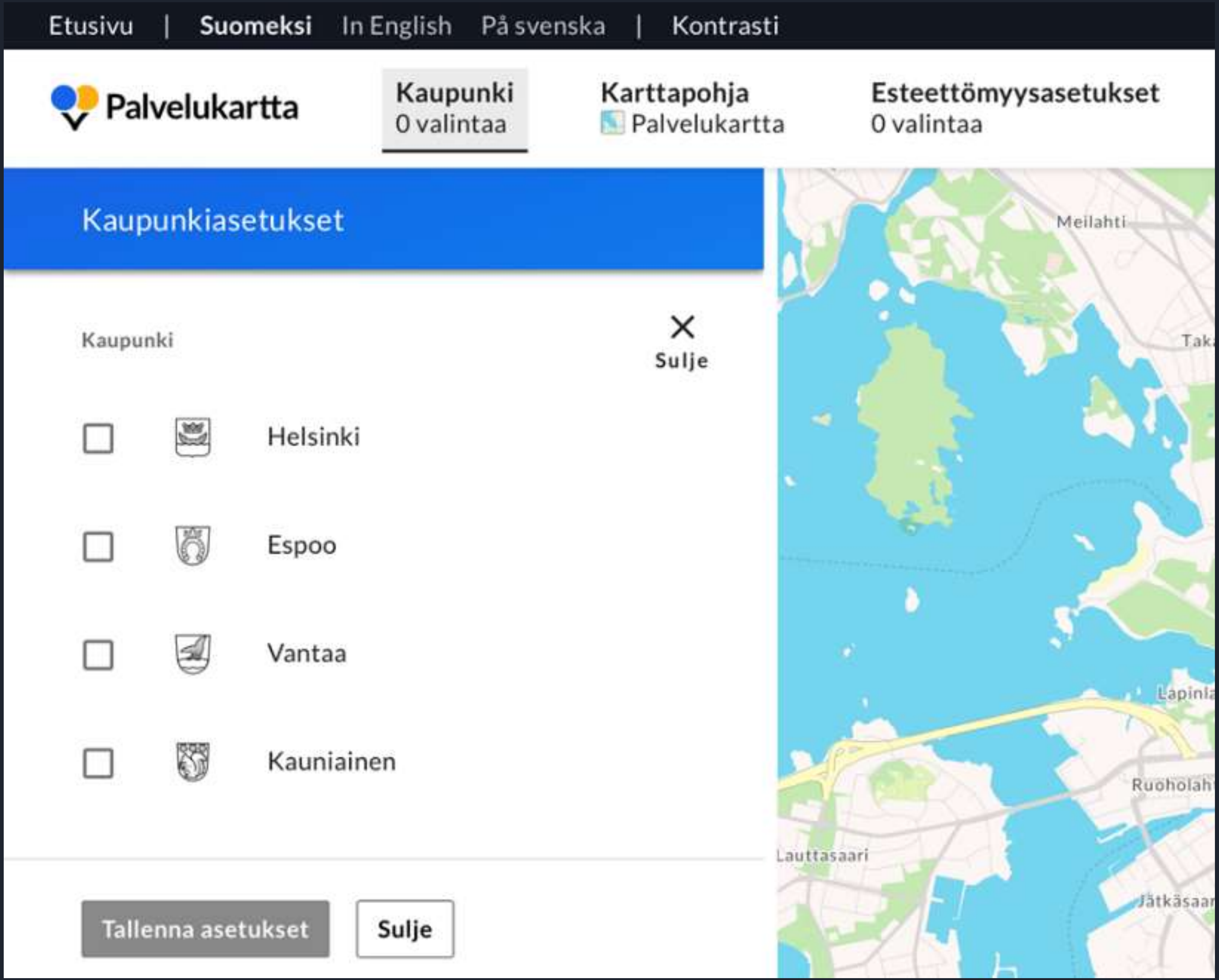
Osuustoiminnan opetus- ja tutkimusrahasto

Rahasto edistää osuustoimintaan liittyvää tutkimusta ja opetusta tieteen eri aloilla.

Tauno Saarelan rahasto

# Modal

- Captures focus within the modal
  - Focus is transferred automatically inside the modal when the modal opens
  - Focus cannot be removed from the modal area
  - The modal relinquishes focus back to the page DOM as the user closes the modal
- Modal constrains the user → A mechanism for guided use scenarios
  - The user should acknowledge information or complete a task
  - Limiting the amount of content on a page (pop-ups can remain open, modals not)
  - Preventing focus order issues
- Implementation requires the correct techniques
  - Both keyboard and SC focus must be bounded by the modal region
- Also, modal description and labelling requires attention



Etusivu | **Suomeksi** In English På svenska | Kontrasti

Palvelukartta

Kaupunki
0 valintaa

Karttapohja
Palvelukartta

Esteettömyysasetukset
0 valintaa

Kaupunkiasetukset

Kaupunki

Sulje

☐ Helsinki

☐ Espoo

☐ Vantaa

☐ Kauniainen

Tallenna asetukset

Sulje

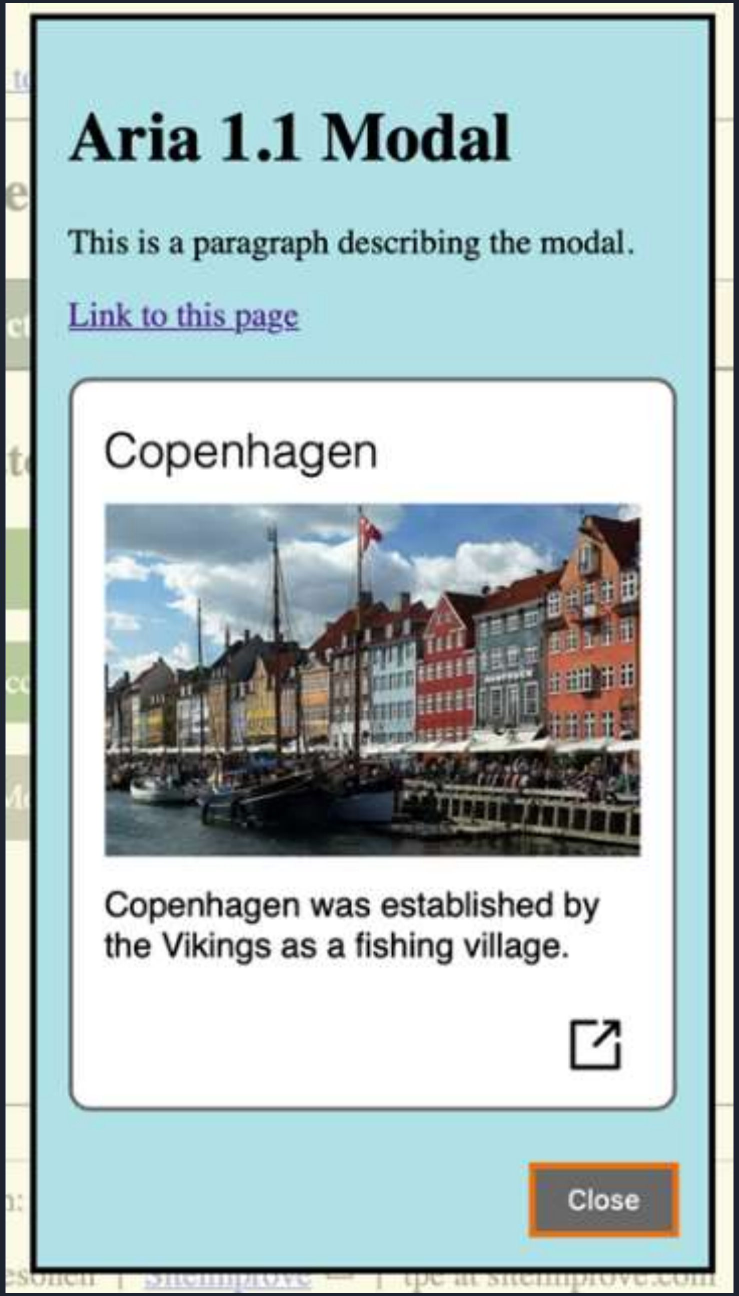
# Opening a modal

The modal trigger button is a simple action button (because context change, not content change) and so needs no state attributes (No: aria-expanded or aria-pressed)

1. Assign the modal wrapper (e.g. div) the following attributes (WCAG 1.3.1)
   - Role=”dialog”
   - Aria-modal=”true”
   - Aria-label=”Modal name”, or aria-labelledby=element within the modal that names the modal (e.g. heading)
   - Optional: Aria-describedby=a longer description of the modal’s role or function: useful especially if focus is not sey at the beginning/”top” of the modal
2. Set focus either on
   - The first interactive element of the modal, or
   - tabindex=-1 attributed first non-focusable element in the modal
3. Make sure that focus cannot escape the modal
4. Remember that the user must be able to close the modal from a control inside the modal (e.g. a button) (related: WCAG 2.1.2)

## Modal: Confining keyboard focus

- A simple solution example: Barrier divs
- Insert empty-of-content div tags as the first and last elements of the modal
- Set for each div
  - `Tabindex=”0”`, so that the div can accept a keyboard focus
  - `Onfocus` event, which transfers focus to the first or last proper interactive element inside the modal
  - `aria-hidden=”true”` attribute (to hide the “barrier” from screen readers)
  - Optional: Set `:focus` → `outline: none` to avoid visual flicker
- When the modal closes, return focus to the element (often a button) that triggered the modal.
  - Note: Browsers will not do this automatically!


# Aria 1.1 Modal

This is a paragraph describing the modal.

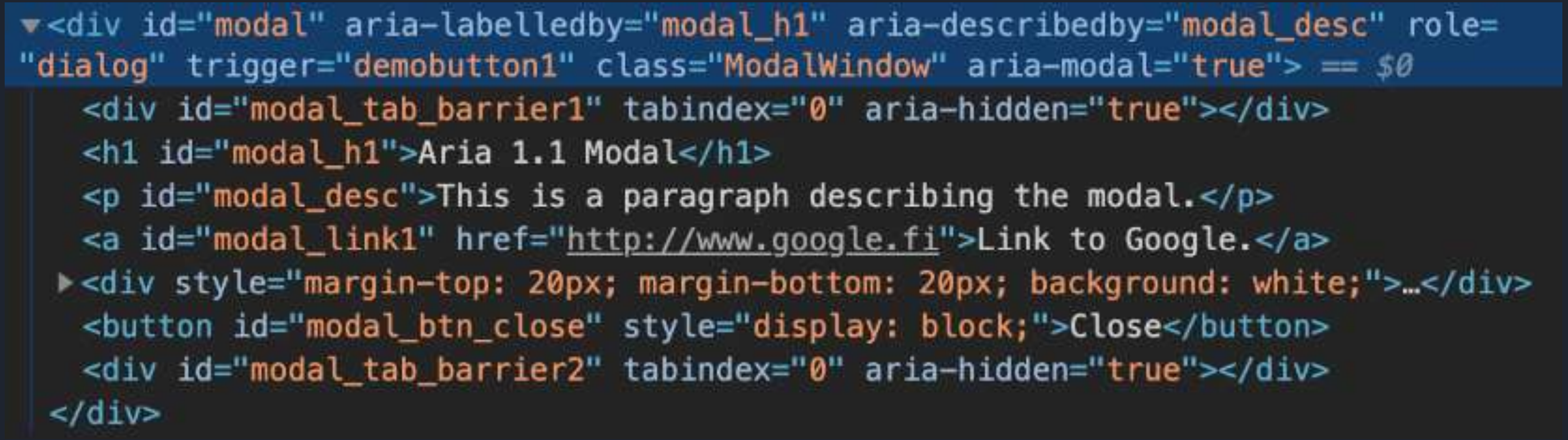
Link to this page

Copenhagen

Copenhagen was established by the Vikings as a fishing village.

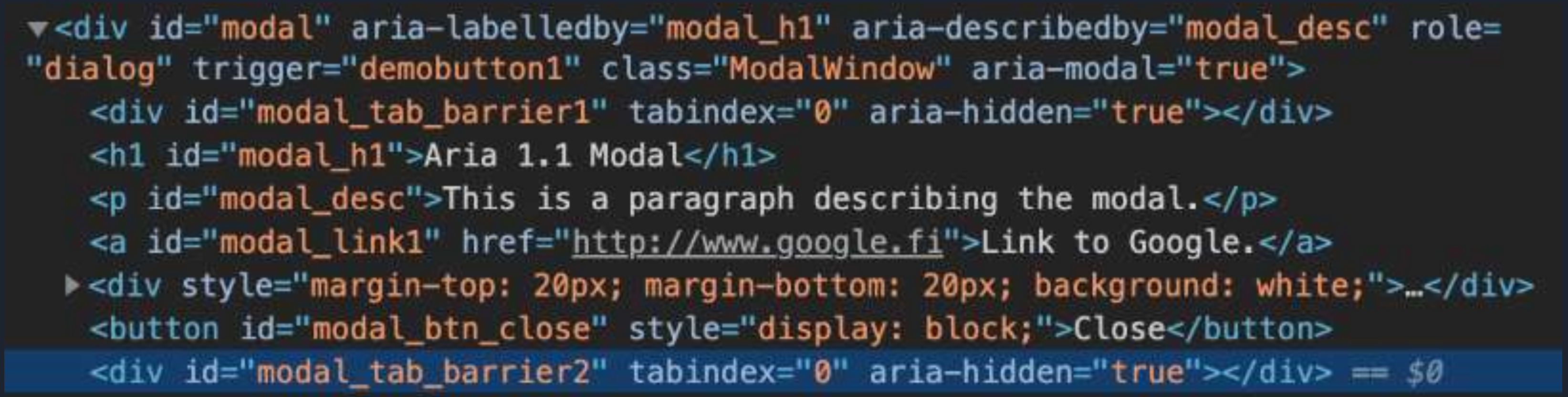
Close

## Modal HTML



```
▼<div id="modal" aria-labelledby="modal_h1" aria-describedby="modal_desc" role=
"dialog" trigger="demobutton1" class="ModalWindow" aria-modal="true"> == $0
    <div id="modal_tab_barrier1" tabindex="0" aria-hidden="true"></div>
    <h1 id="modal_h1">Aria 1.1 Modal</h1>
    <p id="modal_desc">This is a paragraph describing the modal.</p>
    <a id="modal_link1" href="http://www.google.fi">Link to Google.</a>
  ▶<div style="margin-top: 20px; margin-bottom: 20px; background: white;">…</div>
    <button id="modal_btn_close" style="display: block;">Close</button>
    <div id="modal_tab_barrier2" tabindex="0" aria-hidden="true"></div>
  </div>
```

## ”Tab barrier” –elementit



```
▼<div id="modal" aria-labelledby="modal_h1" aria-describedby="modal_desc" role=
"dialog" trigger="demobutton1" class="ModalWindow" aria-modal="true">
    <div id="modal_tab_barrier1" tabindex="0" aria-hidden="true"></div>
    <h1 id="modal_h1">Aria 1.1 Modal</h1>
    <p id="modal_desc">This is a paragraph describing the modal.</p>
    <a id="modal_link1" href="http://www.google.fi">Link to Google.</a>
  ▶<div style="margin-top: 20px; margin-bottom: 20px; background: white;">…</div>
    <button id="modal_btn_close" style="display: block;">Close</button>
    <div id="modal_tab_barrier2" tabindex="0" aria-hidden="true"></div> == $0
```

## Kiitos!