# Creating custom buttons (1): Action Buttons

We take a lot of what HTML does for granted and assume that whatever we throw at the browser will do the same thing.

One of the earliest examples I remember is creating our own custom buttons.

We will use this code adapted from the ARIA Authoring Practices Guide (APG) button examples as we go through the process of turning this code:

```
<div id="action">
  Print Page
</div>
```

Into an accessible and usable button.

The first set of changes are attributes to the HTML code itself:

| Role | Attribute | Element | Usage |
|---|---|---|---|
| button | | div | Identifies the element as a button widget.<br><br>Accessible name for the button is defined by the text content of the element |
| | tabindex="0" | div | Includes the element in the tab sequence.<br><br>Needed on the a element because it does not have a href attribute. |

We add the button role to the `div` to tell assistive technology tools that the div represents a button.

The tabindex attribute represents the tab order of the current element. Using a value of 0 means that it will be at the same level as any other element that hasn't been explicitly given a tabindex attribute.

With these changes, the element now looks like this:

```
<div
  role="button"
  tabindex="0"
  id="action">
  Print Page
</div>
```

Users must be able to activate the button via the keyboard. This is done via Javascript.

We need to consider two events for each of the actions:

- **pointerDown** when a pointer device is down on this element
    - We will use pointer events rather than click
- **keydown** when a key is pressed on this element
    - The key pressed has to be the space bar or then enter key. THey will both have the same result
    - Any other keys pressed will be ignored in this button

I've broken the script by sections to make it easier to comment.

The first section defines a constant to hold the button by it's ID and a function to call from our event handlers...

```
const button = document.getElementById("action");

function printWindow() {
  window.print();
}
```

Next we define the functions that will run inside the event handlers.

The `handlePointerDown` function will be used with the `pointerDown` event and will call the `printWindow` function.

```
function handlePointerDown(event) {
  console.log("Pointer Down Event");
```

```
  printWindow();
}
```

The handleKeyDown function is a little more complicated. We only want some keys to trigger the event so we test them using an if block.

In the if block we test if they key pressed was either the space bar (represented by both the string Space or a string with a space in it) or the Enter key (represented by the string Enter) and call the printWindow function. Otherwise we do nothing.

```
function handleKeyDown(event) {
  if (
    event.key === "Space" ||
    event.key === " " ||
    event.key === "Enter") {
    console.log("Key pressed event");
    printWindow();
  }
}
```

Finally we call the events handlers on our button.

The syntax is the same for both events, the first parameter is the name of the event as a string and the second is the function that we want to execute. Both functions are already defined.

For the pointer-related events, we use the pointerdown event to capture the mouse, pen, or touch on the button. When the event triggers we want to run the handlePointerDown function.

The second function will handle the keydown event event and runs the handleKeyDown function.

```
button.addEventListener(
  "pointerdown",
  handlePointerDown
```

```
);

button.addEventListener(
  "keydown",
  handleKeyDown
);
```

The full working example, along with CSS styling, is in the Codepen below

The HTML of an equivalent `button` element woud look like this:

```
<button>
  Print Page
</button>
```

What we get when we use a button:

- `role = "button"` is implicit for the button element so we don't need to add it
- you can `tab` into buttons without adding `tabindex="0"`
- both the `space` and `enter` keys activate the button
- buttons are clickable by default

You still have to code events that will execute actions based on these events.

***Yes, we can create buttons and other components using our own custom code but it's not trivial and it requires a lot of work that native components do for you.***