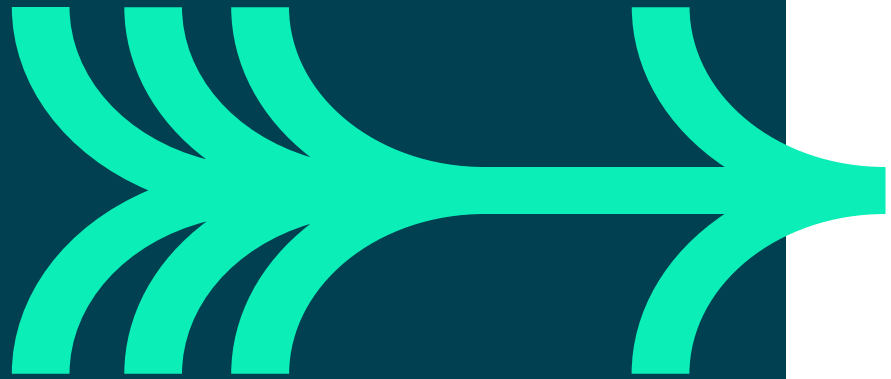
POWERING POTENTIAL

Events

→ JavaScript Fundamentals

INTRODUCTION

- Understanding JavaScript events
- Subscription models
  - Inline
  - Programmatic
  - Event listeners
- Event bubbling and capturing
- The Event object
- The ‘this’ keyword



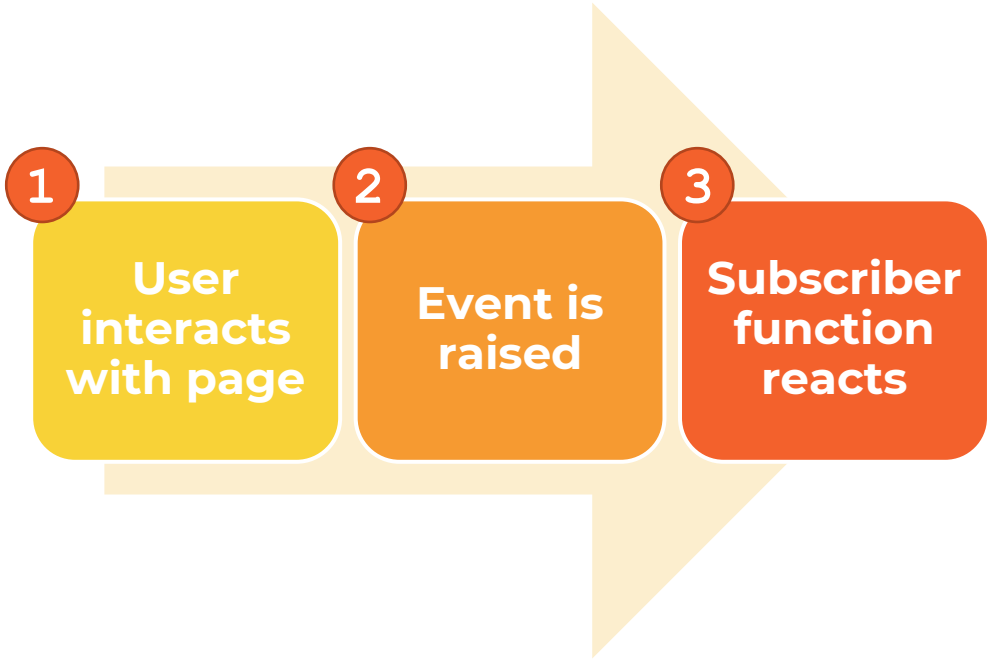
Understanding JavaScript events

- Events are the beating heart of any JavaScript page
  - JavaScript was designed to provide interactivity to web pages
  - This means our pages become responsive to users
- Events can be tricky as older browsers implemented them badly
- Can be implemented as hardcoded attributes or programmatically
  - Inline hardcoded will work everywhere, but can be a blunt instrument
    - Always on, always do the same thing
- Programmatic events are reusable and can be more sophisticated
  - Conditional events depending on browser
  - Detachable - can be switched off

The JavaScript event model

The JavaScript event model uses a publisher/subscriber model.

- Event is raised, a DOM raises countless events
- If a function has been subscribed to the event, it fires



The inline subscription model

- The inline subscription model hardcodes events in the HTML
  - It's quick, easy, and works in all browsers
- This approach is okay for testing but not recommended for release
  - It will likely lead to hard to maintain and repetitive code
  - The event is always on and always fires
- Different event models may be needed for different UI

```
<button type="button" onclick="changeClass('container', 'div2');">
```

- Choose the event you wish to subscribe to
  - Add function call code as the attribute value

Simple event registration model

- All modern browsers accept this programmatic registration approach
  - Events are properties of DOM objects
  - You can assign an event to a function

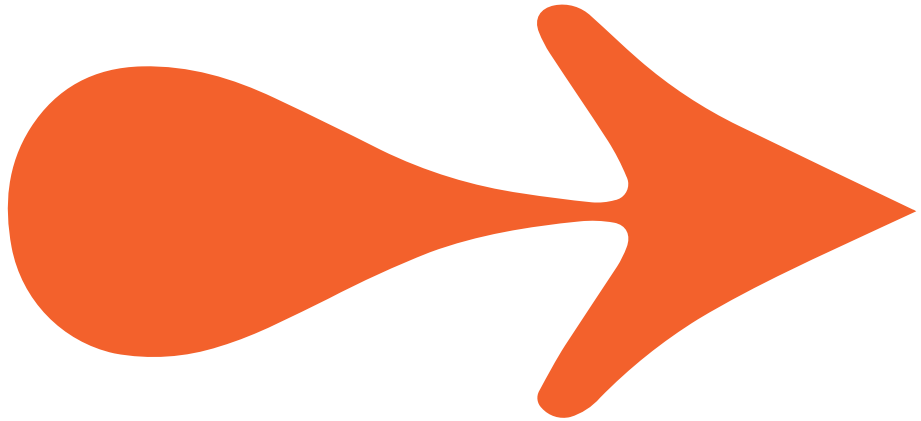
```
myObject.onclick = functionName;
```

- This can also be achieved with anonymous functions
  - Very useful when you only want one object to raise the function

```
myObject.onclick = function(){
    //code to do stuff
}
```

- This approach limits one event to one behaviour, unless you use nested function calls

Event listener registration model



- Allows multiple subscribers to the same events
- Can be detached easily during the life of the program
- Event listeners can be added to any DOM event
  - DOM object selected as usual
  - **`addEventListener`** method setup takes three parameters:

1. The event
2. The function to be raised
3. Whether event capturing should occur (optional, **`default: false`**)

Using addEventListener

- Using **`addEventListener`** is quite simple
  - Parameter 1 – is the event
  - Parameter 2 – is the function
  - Parameter 3 – is a Boolean event bubbling property

```
let e = document.getElementById('container')
e.addEventListener('click', callMe, false);
```

- Multiple events can be subscribed to the same element

```
e.addEventListener('click', callMe, false);
e.addEventListener('click', alsoCallMe);
```

addEventListener and anonymous functions

In many situations, we want to conceal event-raising functions.

- Sounds like a job for anonymous functions!

```
e.addEventListener('click', function () { alert('Do stuff'); });
```

By using the **addEventListener** approach, no parameters can be passed.

- With the exception of the event object (covered later)
- We can get around this issue with anonymous functions

```
b.addEventListener('click', function () {
changeClass(e, 'div2');

});
```

Nested function call

Removing Event Listeners

- Done using the `removeEventListener()` function
- Arguments must be exactly the same as the arguments used to add the event in the first place
  - Event type must be the same
  - Event handler function must be the same
  - Any options, including bubbling and capturing, must be the same

The event object (2)

The event object is created and passed when an event occurs.

- You can add a parameter to the event handling function to catch it

Prevents event bubbling

```
a1.addEventListener('click', stopDefault);
function stopDefault(evt) {
    evt.preventDefault();
    evt.stopPropagation();
}
```

Stops the hyperlink from redirecting

Different kinds of events allow you to capture additional information.

- Such as, key pressed or mouse button clicked

```
b.addEventListener('mousedown', mouseEvent, true);
function mouseEvent(e) {
    alert(`${e.pageX} ${e.pageY}`);
}
```

QuickLab 21 - Events

- Adding and removing event handlers from elements

REVIEW

- Understanding JavaScript events
- Subscription models
  - → Inline
  - → Programmatic
  - → Event listeners
- Event bubbling and capturing
- The Event object
- The ‘this’ keyword