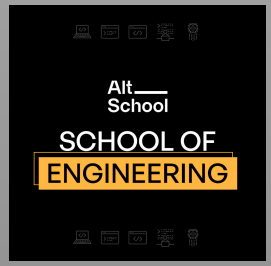

Circle - 08 • Assignment Presentation

Presentation Outline

1. JavaScript Refresh
2. Conditional Statements
3. Asynchronous JavaScript
4. DOM & Events
5. ES Modules & Form Handling
6. Node + npm & Bundlers
7. Browser Object Modules


1. JavaScript Refresh

Spread Operator and Rest Parameter

The Spread Operator [...oldCopy] creates a new copy of an array/object

```
const originals = [1, 2, 3]
const clone = [...originals]   // new copy
clone.push(4)
//OR
// const clone = [...originals, 4]

console.log(originals) // → [1,2,3]
console.log(clone)     // → [1,2,3,4]

// rest parameter
function sum(...nums) {
  return nums.reduce((a, b) => a + b, 0)
}
console.log(sum(2+3+4+5+6+4)) // 24
```

```
[1, 2, 3]
[1, 2, 3, 4]
24
```

Cont’d

The Rest Syntax allows a function accept any amount of argument and gathers everything into one variable (an array).

Key takeaways

- Spread avoids accidental mutationq
- Rest collects unknown arguments **and must be • last in the param list • only once • no default value**

Handy Array Methods

Method	What it does	Mutates?
`sort(cb)`	Arrange items, accepts compare fn	✓
`reverse()`	Flip order in place	✓
`slice(--)`	Copy a portion	✗
`push(x)`	Add to end	✓
`shift()`	Remove first	✓
`unshift(x)`	Add to front	✓

Tip: use non-mutating methods (`slice` , `map` , `filter`) in React state to avoid bugs.

2. JS Conditional Toolkit

```
if (score > 90)                  // IF
  grade = 'A'
else if (score > 75)             // ELSE-IF
  grade = 'B'
else grade = 'C'

const msg = age >= 18            // TERNARY
  ? 'Vote!' : 'Grow up'

switch(day) {                    // SWITCH
  case 'Mon': …
}
```

- **Nested switch** → rarely worth the complexity.
- Pick the construct that keeps intent obvious.

3. Asynchronous JavaScript

```
async function getGitHubUser(name) {
  const res = await fetch(`https://api.github.com/users/${name}`)
  if (!res.ok) throw new Error('Network error')
  return res.json()
}

getGitHubUser('chrisroland')
  .then(user => console.log(user.name))
  .catch(console.error)
```

```
Chris Ebube Roland
```

- Promises tame callback hell
- `async/await` reads top-to-bottom
- `fetch()` supersedes old **XMLHttpRequest** for HTTP requests.
- Always handle errors `(try/catch or .catch())`
- Callbacks still exist – `.map`, `.filter`, `.reduce` each expect one

4. DOM & Events

Events are created in Javascript using the following methods

- Html attribute
- DOM Property
- addEventListener

Html attribute

```
<p onmouseover="alert('Mouse!')">Click Me! </p>
```

DOM Property

DOM (Document Object Model) is simply Javascript representation of your Html, DOM is an important aspect of Javascript. Through the DOM, we can search and modify html elements.

continuation

```
<div id="event">Events in Javascript</div>
<script>
  event.onmouseover = function () {
    alert('Hmm, Motion!')
  }
</script>

  Creating an element Through the DOM
<script>
const newElement = document.createElement('p');
 const newText = document.createTextNode('Javascript is not for the weak');
 newElement.appendChild(newText);
 element.appendChild(newElement);
 console.log(newElement.textContent);
 </script>
```

The main difference between "innerHTML" and "textContent" is that while textContent allows you to pass in text only, innerHTML allows you to pass in html text.

addEventListener

```
<button id="btn">Clicked 0 times</button>

<script type="module">
  const btn = document.getElementById('btn')
  let count = 0

  btn.addEventListener('click', e => {
    count++
    e.currentTarget.textContent = `Clicked ${count} times`
  })
</script>
```

'click' is the event while the function represents a handler which listens or responds to the click event.

Click the button to increment the count.

Clicked 0 times

The Concept of Event Bubbling and Capturing

Bubbling - This concept simply entails firing an event on the innermost element, then on successively less nested elements. When the element is clicked, it runs the handlers on it, then up to its parent (We could refer to it as "Ascension" i.e moving upwards).

```
<div onclick="alert('Second div')">
<div onclick="alert('First div')">
 <p onclick="alert('P Element')">P Element</p>
</div>
</div>
```

In the example above, the handler on the "p" tag will run, followed by the handlers on the first and second divs respectively.

p ⟶ div 1 ⟶ div 2

Capturing - This is the reverse of Bubbling, the event fires on the least nested element, then the following nested elements until it reaches the target element (moving downwards).

Event Delegation

Event delegation signifies assigning a single handler on a common parent to handle events on multiple child elements. This approach reduces the number of event listeners required and improves efficiency.

- `addEventListener` is preferred
- Understand **bubbling** vs **capturing**
- Use delegation for long lists

5. ES Modules + Dynamic import()

Export labels what a module shares while **import** pulls that piece into another file.

```
// utils/math.js
export function add(a, b) { return a + b }
export default function mul(a, b) { return a * b }

// main.js
import mul, { add } from './utils/math.js'

(async () => {
  if (performance.now() > 5000) {
    const { sparkle } = await import('./effects/sparkle.js')
    sparkle()
  }
})();
```

Why it matters: predictable scope, tree-shaking, lazy-loading.

Form Handling with FormData

```
<form id="todoForm">
  <input name="task" required>
  <button>Add</button>
</form>

<script type="module">
  todoForm.addEventListener('submit', e => {
    e.preventDefault()
    const data = new FormData(e.target)
    console.log(Object.fromEntries(data)) // { task: "Buy Akara" }
  })
</script>
```

- `preventDefault()` stops page reload
- `FormData` quickly serialises any form

6. Node + npm & Bundlers

```
npm init -y            # generates package.json
npm i -D vite          # ultra-fast dev server
npm run dev            # HMR at localhost:5173
npm run build          # output /dist with hashed assets
vite preview           # test production build
```

Why bundlers?	Benefits
Browsers can’t import SVG/PNG or npm libs directly	Bundlers translate everything
Code-splitting & optimisation	Smaller, faster production bundles
Dev server with HMR	Instant feedback while coding

7. Browser Object Models

Layer	What it lets JS control
DOM	HTML & content structure
CSSOM	Stylesheets (classes, colors)
BOM	Browser Object Model e.g chrome – `window` , `history` , `navigator`

Note: the *window* object is global; *document* and styles live one layer below.

Confetti Demo

```
//Confetti.js
import confetti from 'canvas-confetti'

export function celebrate() {
  confetti({
    particleCount: 150,
    spread: 70,
    origin: { y: 0.6 }
  })
}
```

```
<form id="todoForm">
  <input name="task" required>
  <button onclick="celebrate()">Add</button>
</form>
<script>
  import celebrate from '/confetti.js'
</script>
<!-- click button to see confetti fx -->
```

Add to do

Could be used to celebrate. E.g call `celebrate()` after adding a new to-do.

Summary;

Skills learned	Usage/Real-world impact
Clean array/object handling	Fewer bugs, simpler state updates
Promises & `await`	Reliable API calls, loaders, error UI
DOM mastery	Interactive components without libraries
ES Modules	Maintainable, testable codebase
Bundlers & npm	Modern workflow—ready for React/Next