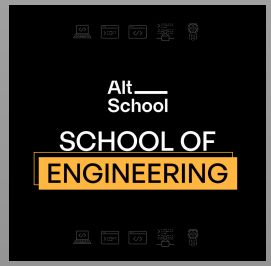


# Circle - 08 • Assignment Presentation

# 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

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

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

## Key takeaways

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

# 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
- Always handle errors (`try/catch` or `.catch()`)
- Callbacks still exist – `.map`, `.filter`, `.reduce` each expect one

# 4. DOM & Events

```
<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 the button to increment the count.

Clicked 0 times

- `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 |