the Master Course

{CODENATION}

JS & DOM

Introduction

{CODENATION}

{CODENATION}

Learning Objectives

Understand the HTML & DOM structure

To be able to apply changes to the DOM by responding to user interaction

{CN}®

DOM

Lets look at...

... Creating new Elements

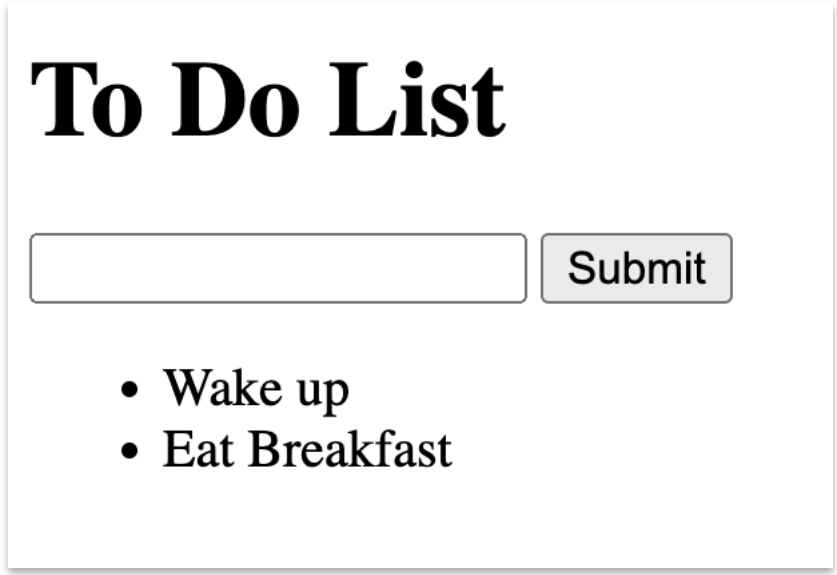
createElement

DOM

This method creates a HTML element specified by tag name

HTML

```
<h1>To Do List</h1>
<input id="toDoInput" type="text">
<button id="submitBtn">Submit</button>
<ul id="list">
    <li>Wake up</li>
    <li>Eat Breakfast</li>
</ul>
```



{CODENATION}

createElement

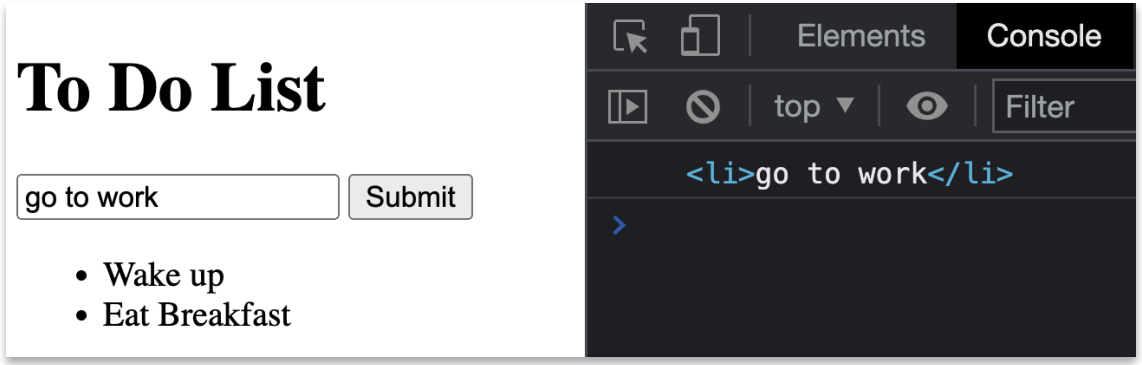
DOM

createElement creates the element but doesn't add it to the page.
For now its stored in the listItem variable

JS

```
const submitBtn = document.getElementById('submitBtn')
const input = document.getElementById('toDoInput')

submitBtn.addEventListener('click', () => {
    const listItem = document.createElement("li")
    listItem.textContent = input.value
    console.log(listItem)
})
```



{CODENATION}

appendChild

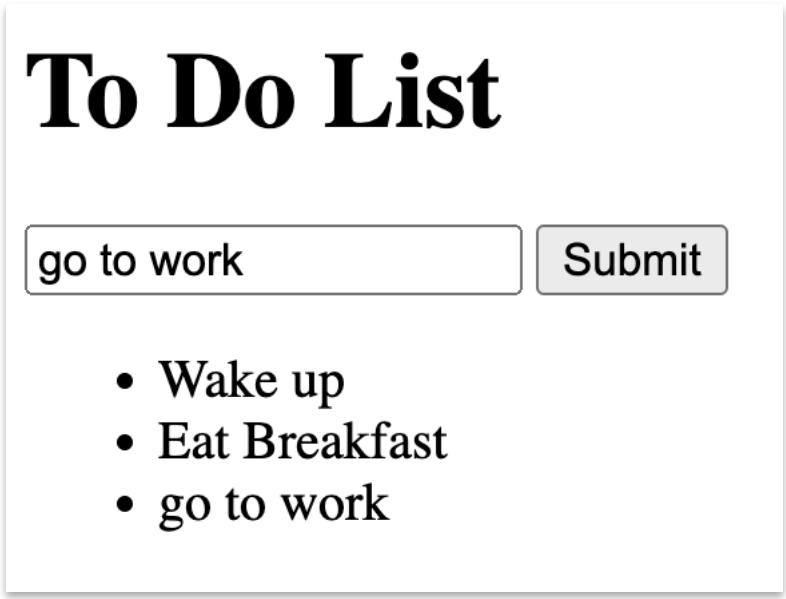
DOM

This method adds an element to the end an another element. In this case it adds the li element to the end of the ul

JS

```
const submitBtn = document.getElementById('submitBtn')
const input = document.getElementById('toDoInput')
const list = document.getElementById('list')

submitBtn.addEventListener('click', () => {
    const listItem = document.createElement("li")
    listItem.textContent = input.value
    list.appendChild(listItem)
})
```



{CODENATION}

Lets look at...

... Removing Elements

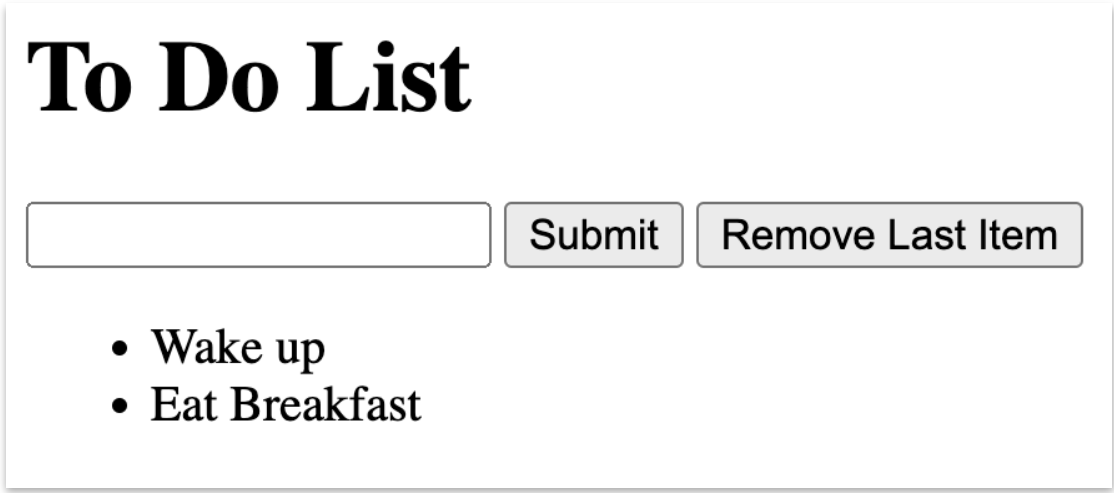
removeChild

DOM

This method removes a specified child from a specified element

HTML

```
<h1>To Do List</h1>
<input id="toDoInput" type="text">
<button id="submitBtn">Submit</button>
<button id="removeBtn">Remove Last Item</button>
<ul id="list">
    <li>Wake up</li>
    <li>Eat Breakfast</li>
</ul>
```



{CODENATION}

removeChild

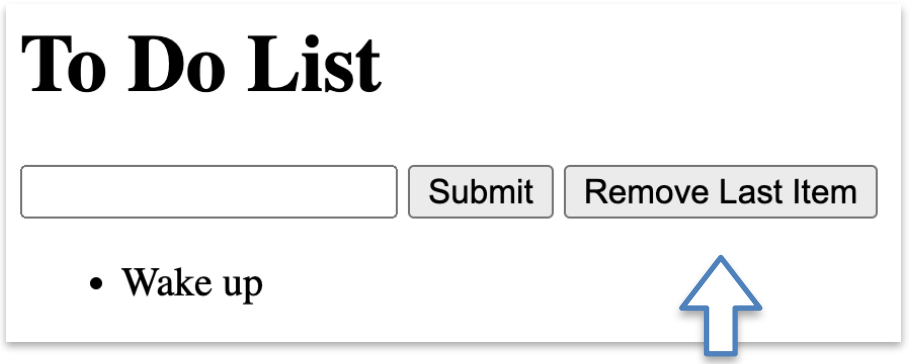
DOM

Target the element you want to delete, then target its parent element. In this case we remove the last li from the ul

JS

```
const removeBtn = document.getElementById('removeBtn')

removeBtn.addEventListener('click', () => {
    const lastLiItem = document.querySelector("li:last-child")
    list.removeChild(lastLiItem)
})
```



After remove button is clicked

{CODENATION}

{CN}®

DOM

Lets look at...

... setTimeout()

setTimeout

DOM

This method calls a function after a specified number of milliseconds. 1 second = 1000milliseconds

HTML

```
<h1 id="heading"></h1>
<button id="surpriseBtn">Click for Surprise!</button>
```

JS

```
const surpriseBtn = document.getElementById('surpriseBtn')
const heading = document.getElementById('heading')

surpriseBtn.addEventListener('click', () => {
    heading.textContent = 'SURPRISE!'
    setTimeout(() => {
        heading.textContent = ''
    }, 1000);
})
```

Text then disappears after 1 second

{CODENATION}

Lets look at...

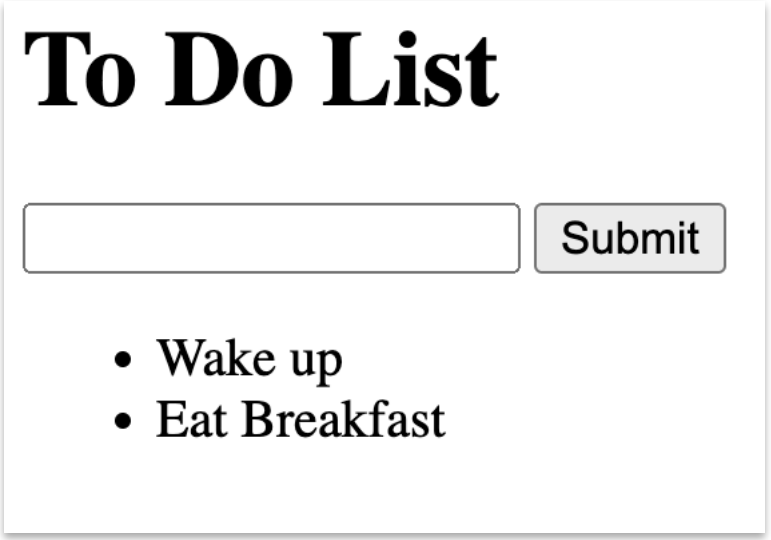
... adding event listeners to multiple elements

Multiple event listeners DOM

Targeting each li element and giving each one a separate event listener to do the same thing isn't DRY code

HTML

```
<h1>To Do List</h1>
<input id="toDoInput" type="text">
<button id="submitBtn">Submit</button>
<ul id="list">
    <li>Wake up</li>
    <li>Eat Breakfast</li>
</ul>
```



{CODENATION}

forEach

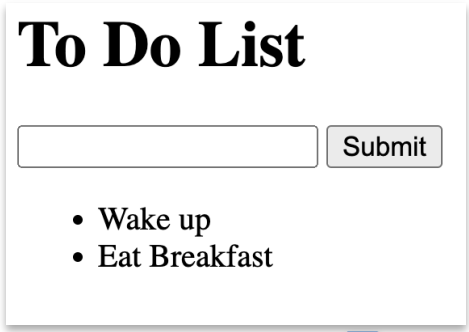
DOM

This method calls a function for each element in an array. The function takes a parameter which is a name for the current value

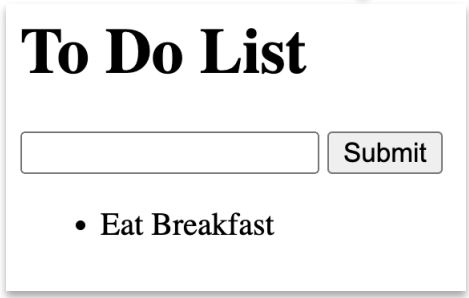
JS

```
const list = document.getElementById('list')
const allListItems = document.querySelectorAll('li')

allListItems.forEach((listItem) => {
    listItem.addEventListener('click', (event) => {
        list.removeChild(event.target)
    })
});
```



Removes li thats clicked

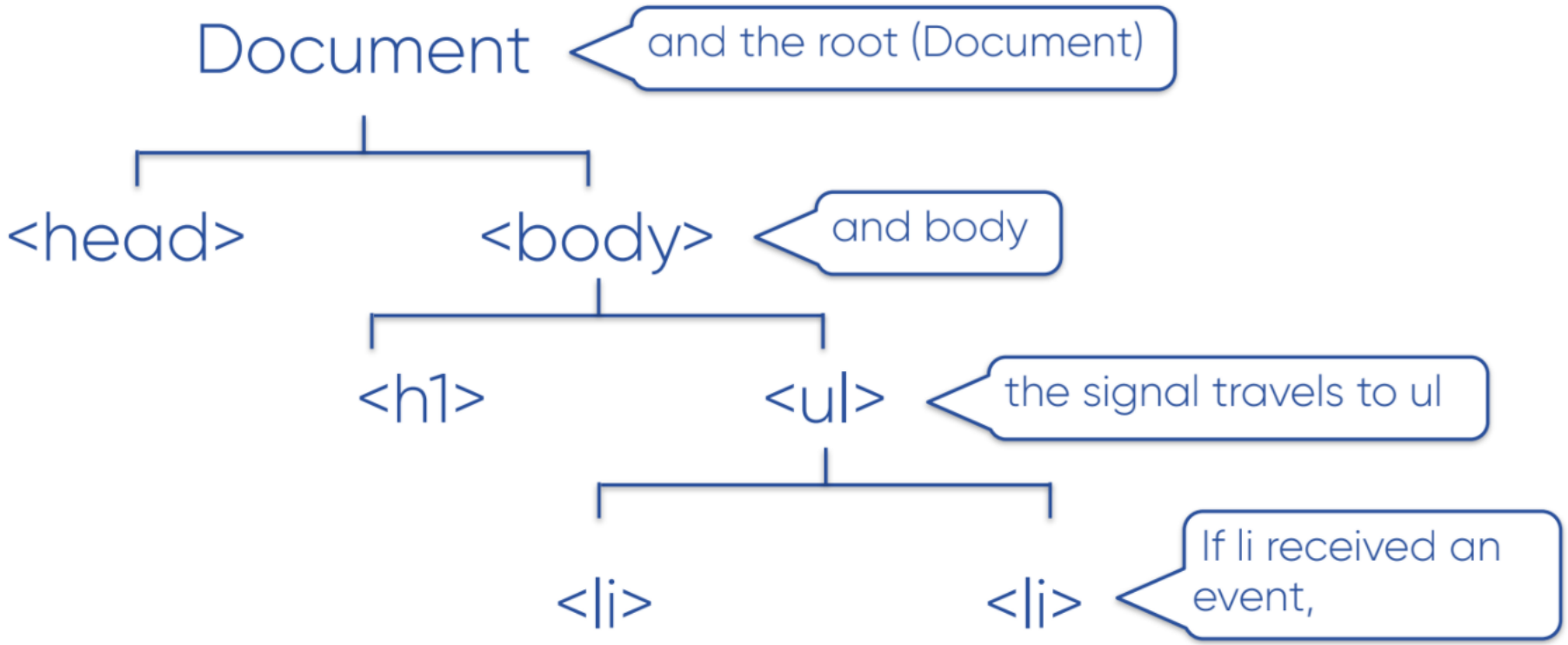


{CODENATION}

Event Bubbling

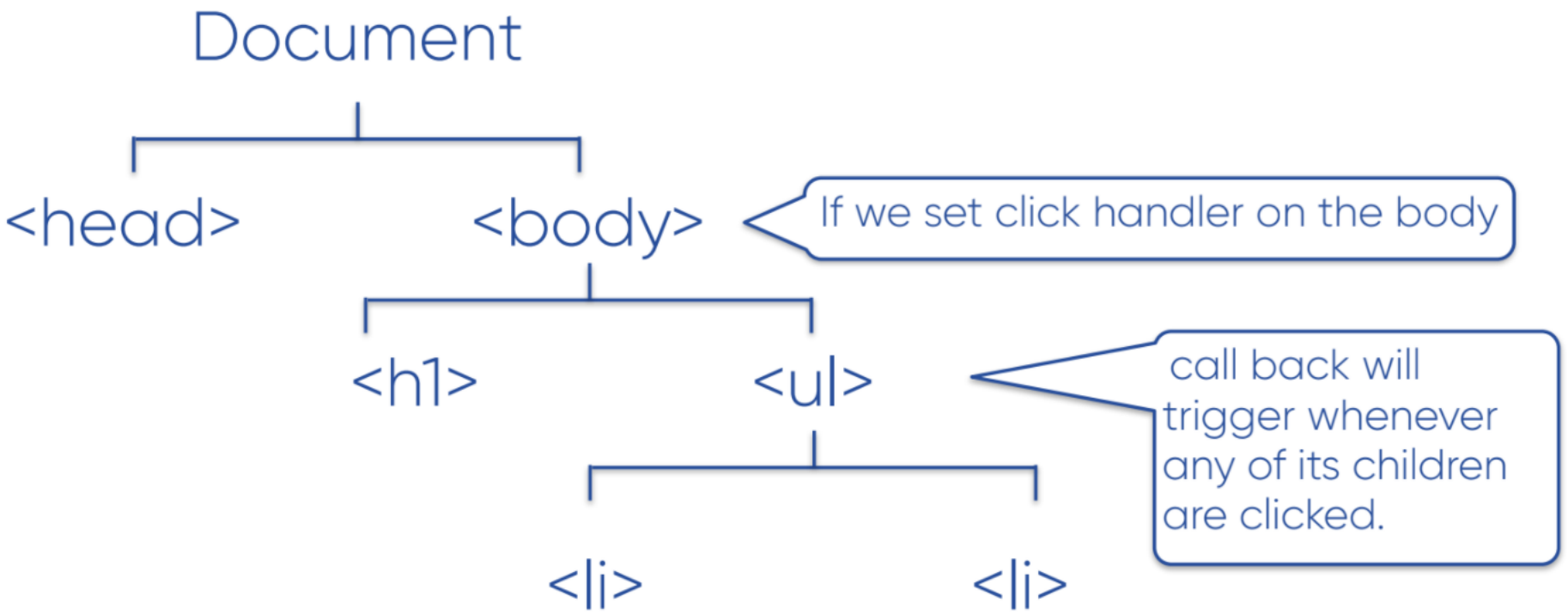
Li receiving a signal

DOM

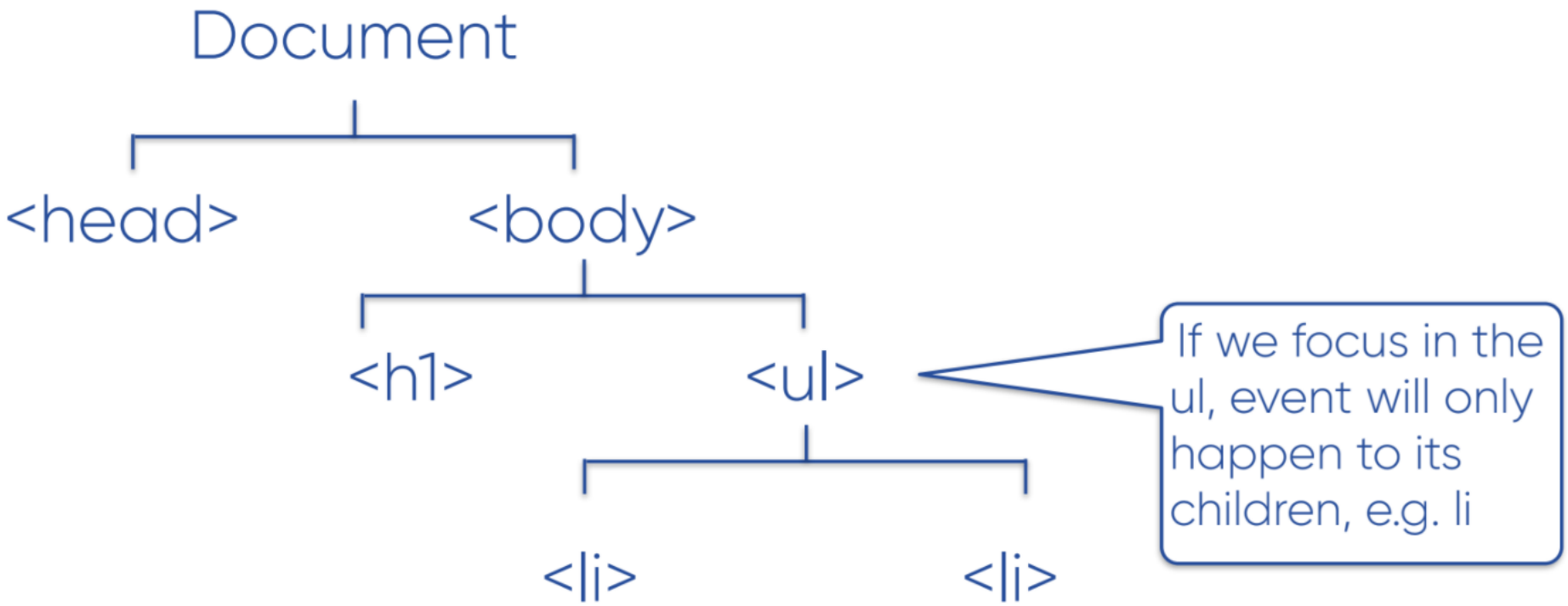


{CODENATION}

Setting click handler on the body



Setting click handler on the ul



Using event bubbling

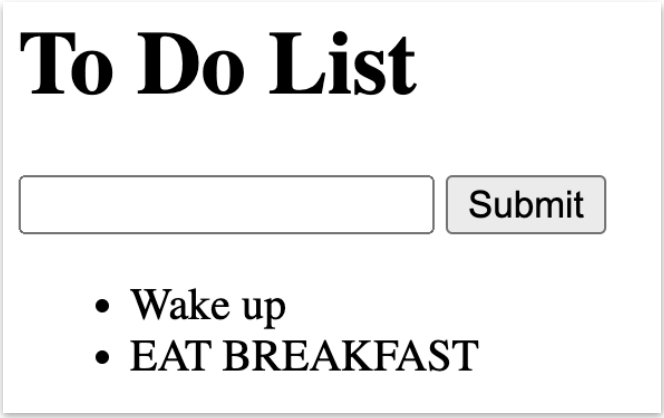
DOM

We can make use of the event.target property to identify the child element we are currently interacting with

JS

```
const list = document.getElementById('list')

list.addEventListener('mouseover', (event) => {
    event.target.textContent = event.target.textContent.toUpperCase()
})
```



Li turns uppercase on mouseover

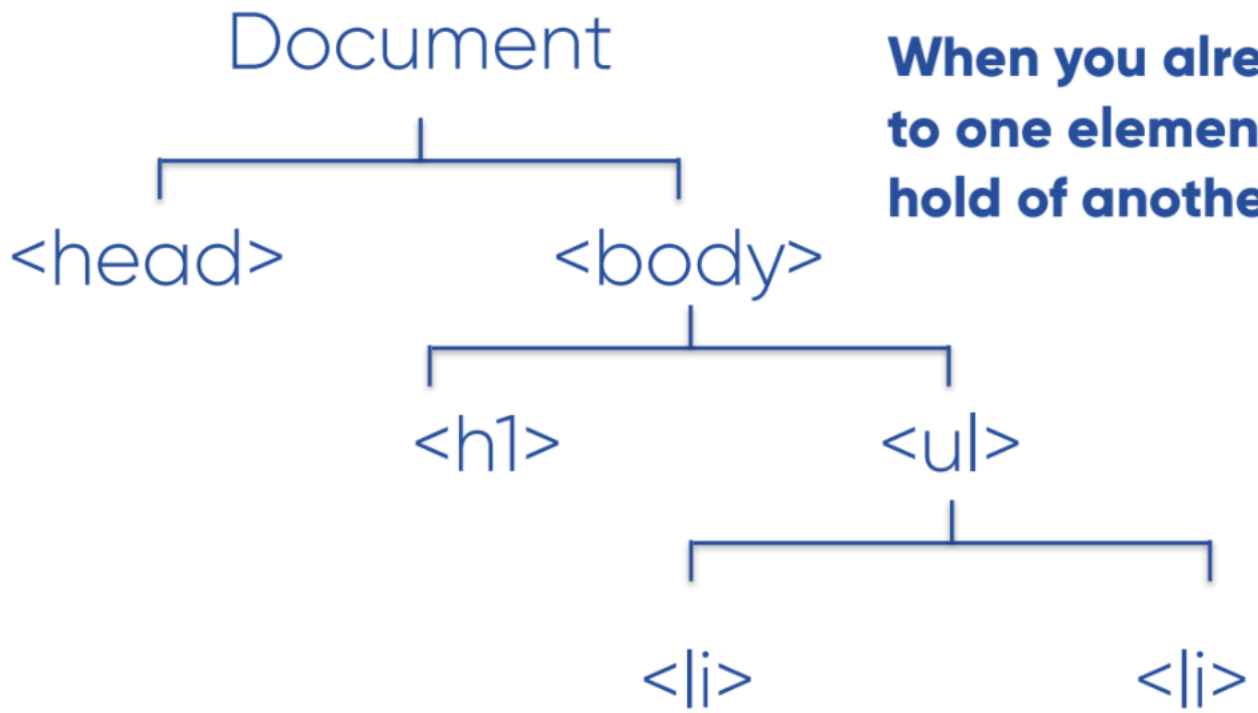
{CODENATION}

Lets look at...

... DOM Traversal

DOM Traversal

DOM



When you already have a reference to one element and you need to get hold of another element nearby.

*It is a way to move from one part of the DOM to another and select an element based on its relationship to another element

{CODENATION}

parentNode

DOM

This property returns the parent element of a specified element

HTML

```
<div>
    <h1>My Pop-Up Box</h1>
    <p>Some text</p>
    <button id="closeBtn">Close Pop-Up</button>
</div>
<button id="openBtn">Open Pop-Up</button>
```

{CODENATION}

parentNode

DOM

We can access the parent element without needing to select another HTML element. In this case the parent is the div

JS

```
const closeBtn = document.getElementById('closeBtn')

closeBtn.addEventListener('click', () => {
    closeBtn.parentNode.style.display = 'none'
})
```

Div is hidden on close button click

{CODENATION}

previousElementSibling DOM

This property returns the previous HTML element in the same tree level. In this case the open buttons previous sibling is the div

JS

```
const openBtn = document.getElementById('openBtn')

openBtn.addEventListener('click', () => {
    openBtn.previousElementSibling.style.display = 'block'
})
```

Div is visible on open button click

{CODENATION}

{CODENATION}

Learning Objectives

Understand the HTML & DOM structure

To be able to apply changes to the DOM by responding to user interaction