Breaking Javascript

Document Object Model (DOM)

by Shaun Kriel

What is the DOM (Doucument Object Model) ?

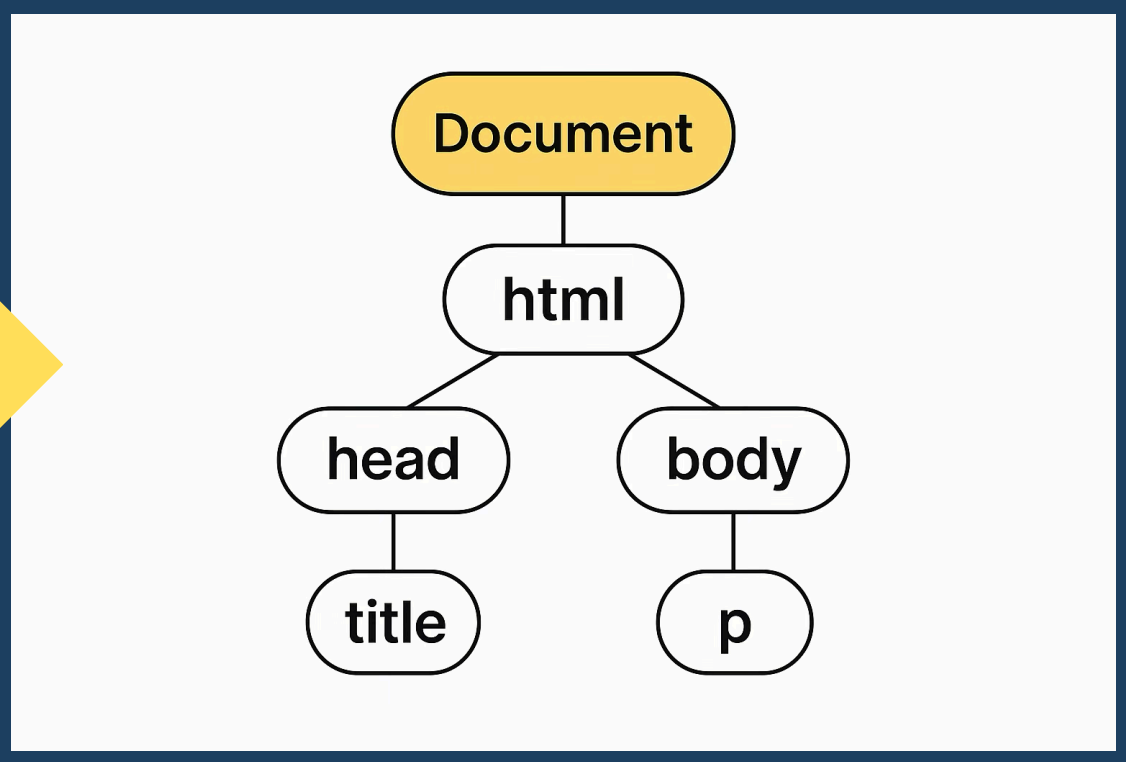
The **DOM** is like a *map* of your web page that the browser creates so it can understand and interact with it.

Think of it like this:

- Your **webpage** is a **house**
- The **HTML** is the **blueprint**
- The **DOM** is the live **3D model** of the house where each part (doors, windows, rooms) can be seen, changed or moved around - *while people are inside*

So the **DOM** turns **HTML** into **TREE** structure that Javascript can read and change

```
Document
└──── html
├──── head
│   └──── title
└──── body
  ├──── h1
  └──── p
```



How the DOM works

When a browser loads your **HTML** page, it reads the **HTML** and builds a ***tree*** like structure (like the diagram)

Each tag - **<h1>**, **<p>**, ** ** becomes a **node** in the tree

JavaScript can then use the **DOM** to:

- Change text
- Add / remove elements
- React to clicks or other user actions

The DOM is what lets a website respond and interact with the users

Where / Why the DOM is used

The DOM is used whenever you want your webpage to:

- Be interactive (buttons - "Show More" or "Submit"
- Update without reloading (editing a to-do list)
- Respond to things like:
  - Mouse clicks
  - Keyboard input
  - Page loads

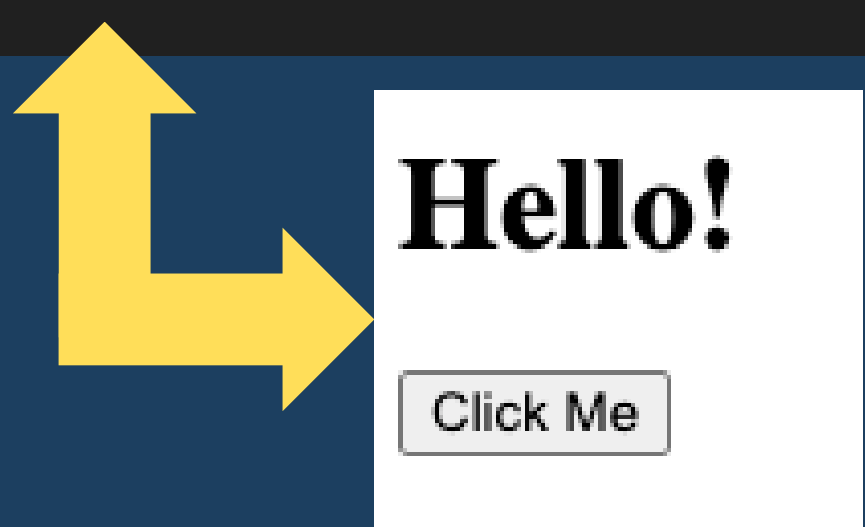
Its used in:

- Games on website
- Forms that give instant feedback
- Interactive stories, quizzes and more

Example: using DOM with Javascript

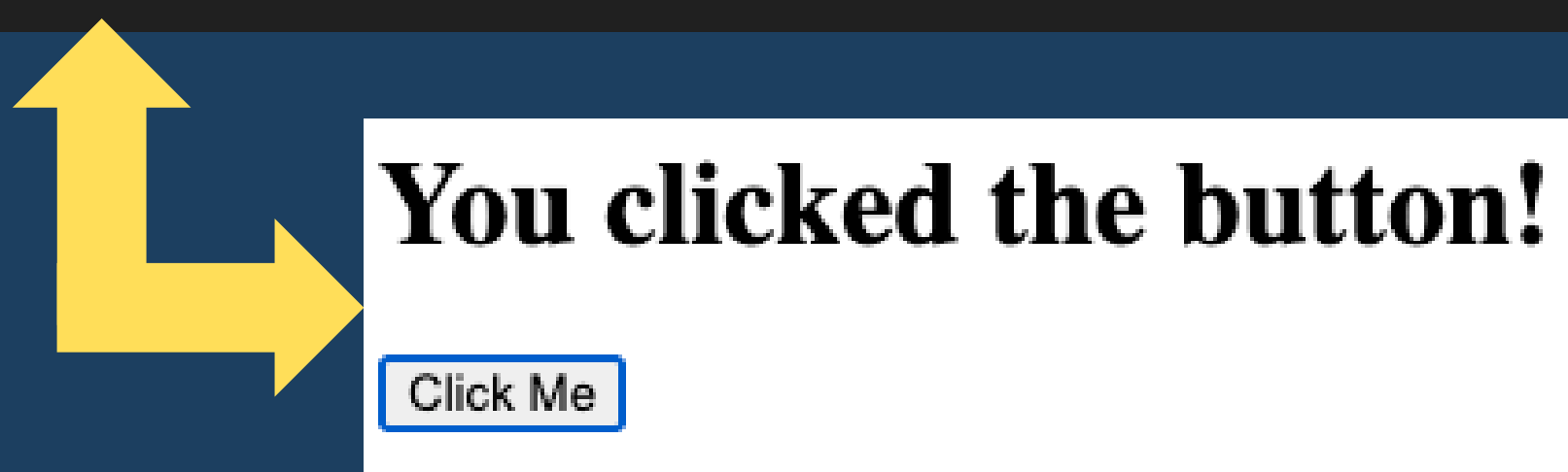
html -

```html
<body>
  <h1 id="greeting">Hello!</h1>
  <button onclick="changeText()">Click Me</button>
```



javascript-

```javascript
function changeText() {
    // find the <h1> by its ID and change its text
    document.getElementById("greeting").innerText = "You clicked the button!"
}
```



whats happening?

- **Document** refers to the whole page
- **getElementById** (**"greeting"**) finds the **<h1>**
- **innerText** = ". . ." changes the text

This is DOM manipulation in action - Using JS to change part of the page after it was already loaded

DOM Selectors

single element:

Used to grab one specific element from the page.

```
// DOM selector - single element //
const title = document.querySelector('h1');
console.log(title.innerText);
```

```
Hello!
>
```

querySelector( ) grabs the first matching element

multiple elements::

Used when you want to grab more than one element (like a list of items)

```
<h1 id="greeting">Hello!</h1>
<li>banana</li>
<li>apple</li>
<li>kiwi</li>
<button onclick="changeText()">Click Me</button>
```

```
// DOM selector - multiple elements //
const items = document.querySelectorAll('li');
items.forEach(item => console.log(item.innerText));
```

querySelectorAll() grabs all matching elements. You can loop through them.

```
banana
apple
kiwi
>
```

traversing elements:

Move up, down, or sideways in the tree

```
// Traversing DOM elements //
const list = document.querySelector('ul');
console.log(list.children); // all <li> children
console.log(list.firstElementChild); // first <li>
console.log(list.lastElementChild); // Last <li>
```

use **children**, **firstElementChild**, **lastElementChild** to navigate elements

```
▾ HTMLCollection(3) [li, li, li] i
  ▸ 0: li
  ▸ 1: li
  ▸ 2: li
    length: 3
  ▸ [[Prototype]]: HTMLCollection
▾ <li>
    ::marker
    "banana"
  </li>
▾ <li>
    ::marker
    "kiwi"
  </li>
```

traversing node:

Node inlcude text, comments and elements

```
// Traversing nodes //
const list = document.querySelector('ul');
console.log(list.childNodes); // includes text nodes
```

```
NodeList(7) [text, li, text, li, text, li, text]
  0: text
  1: li
  2: text
  3: li
  4: text
  5: li
  6: text
  length: 7
  [[Prototype]]: NodeList
```

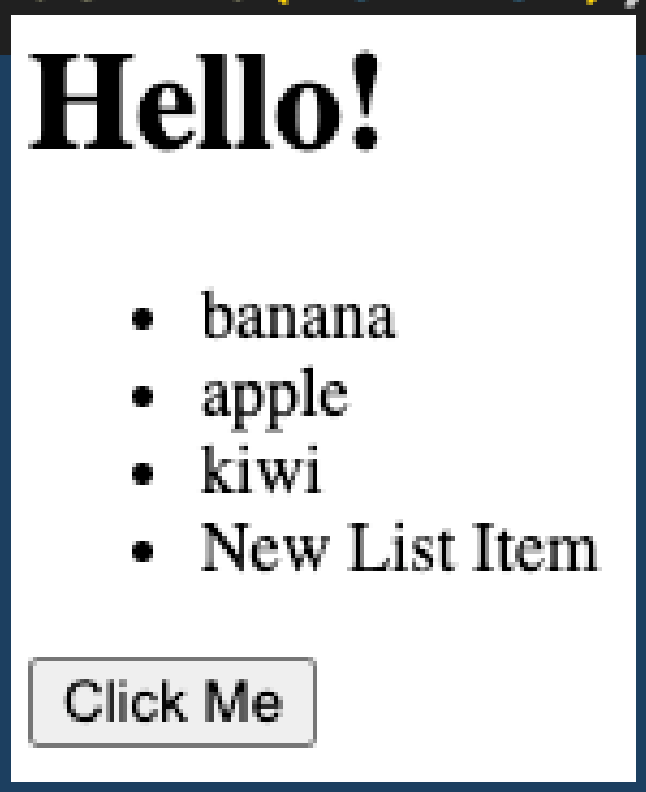
childNodes shows everything, incl line breaks & spaces

create elements:

Make new elements with javascript

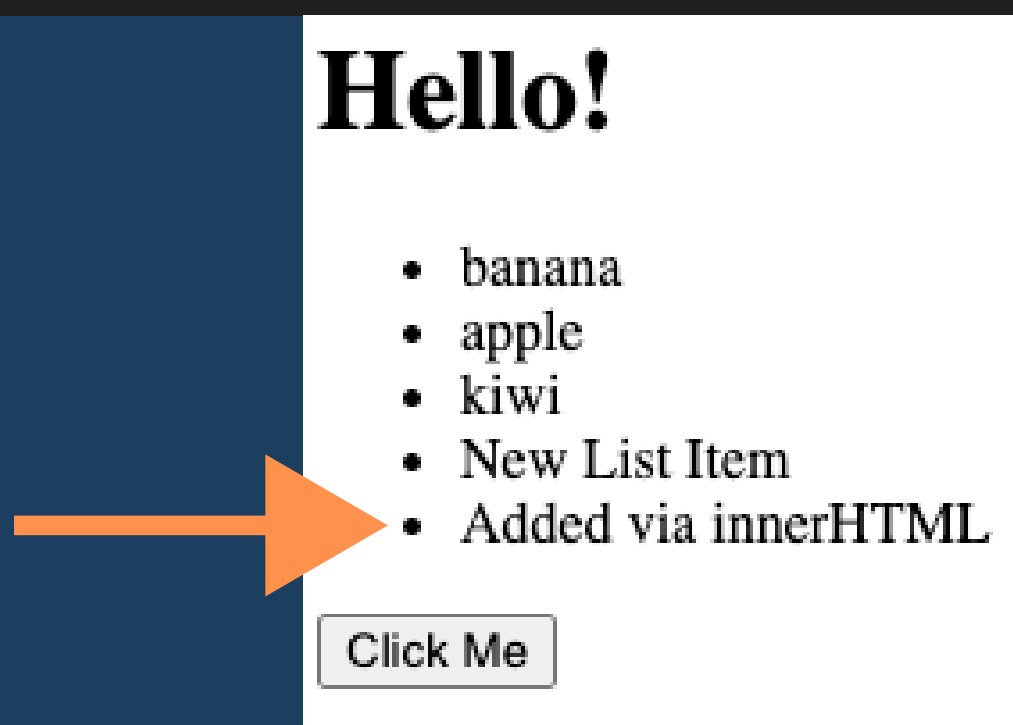
```
// create element //
const newItem = document.createElement('li');
newItem.innerText = 'New List Item';
document.querySelector('ul').appendChild(newItem);
```

createElement() creates, **appendChild()** adds it to page

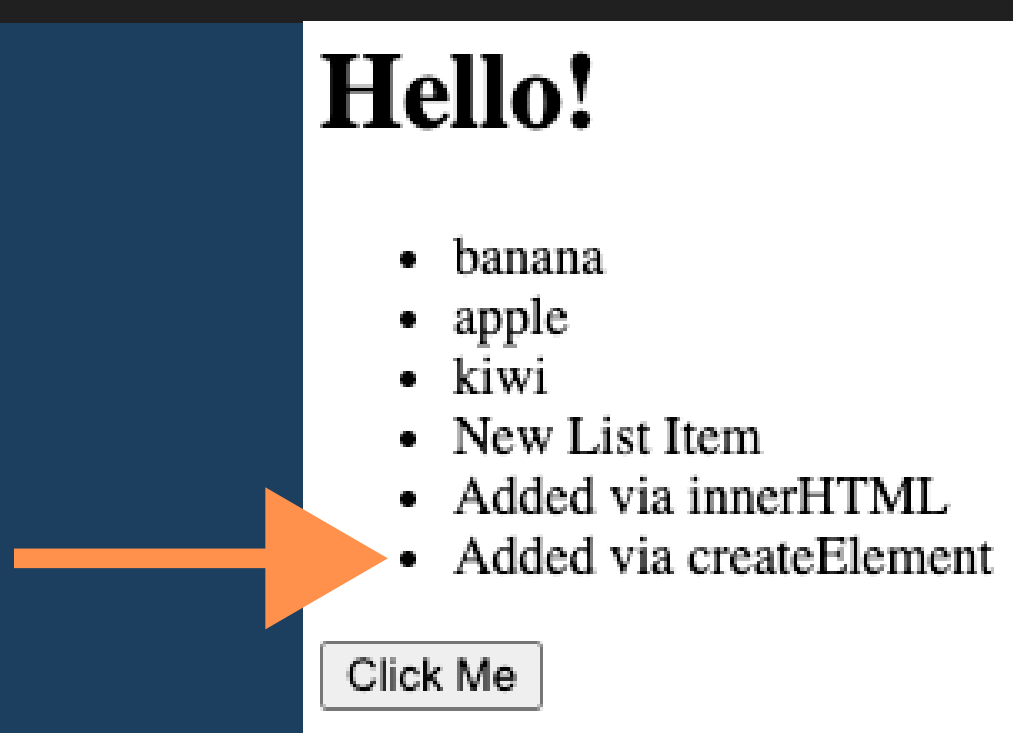


create item: innerHTML vs createElement

```
// using innerHTML (fast but less safe) //
document.querySelector('ul').innerHTML
+= '<li>Added via innerHTML</li>';
```




```
// using createElement (safer and recommended)
const li = document.createElement('li');
li.innerText = 'Added via createElement';
document.querySelector('ul').appendChild(li);
```



innerHTML rewrites the HTML & removes event listeners. Use createElement for better control

Refactor to Multiple Functions:

Break big code into smaller parts for reusability

```
function createListItem(text) {
    const li = document.createElement('li');
    li.innerText = text;
    return li;
}

function addToList(item) {
    const list = document.querySelector('ul');
    list.appendChild(item);
}

// use the functions //
const newItem = createListItem('Reusable Item');
addToList(newItem);
```

createListItem - this function creates a new **<li>** HTML element, gives it some text, returns it so you can do something with it later (like adding to list)

addToList - this function finds a <ul> in the document and adds a new item (that you pass in) to the bottom of that list.

use both functions together - step 1: create an <li> with the test "Reusable Item" Step 2: add it to the first <ul> on the page. Webpage will got from
**
**
to

  Reusable Item

remove elements:

```
// remove elements //
const itemToRemove = document.querySelector('li');
itemToRemove.remove();
```

before

after