COMP6080
Web Front-End Programming

Javascript

In the web browser

Where to put your browser-based Javscript

Javascript is a programming language, that has two main uses:

1. Javascript used to manipulate the DOM in a web browser (discussed in another lecture)
2. Javascript used to write scripts with NodeJS (discussed in another lecture)

Today we are focusing on (1).

More specifically, what are the different ways you can include your Javascript in a page run by a web browser?

HOW code is included

Code can either be included inline (part of the page) or included via external link (URL to resource)

inline

```html
<script type="text/javascript">
const a = 1 + 2;
console.log(a);
</script>
```

mypage1.html

external

```html
<script type="text/javascript" src="mywork.js">
</script>
```

mypage2.html

```javascript
const a = 1 + 2;
console.log(a);
```

mywork.js

HOW code is included

Linking javascript externally:

- Improvements performance when browser cache is utilised
- Reduces the time it takes for initial html document to be received (smaller file)

Placing Javascript inline:

- Avoids network requests (potentially costly) for script if file is not yet loaded

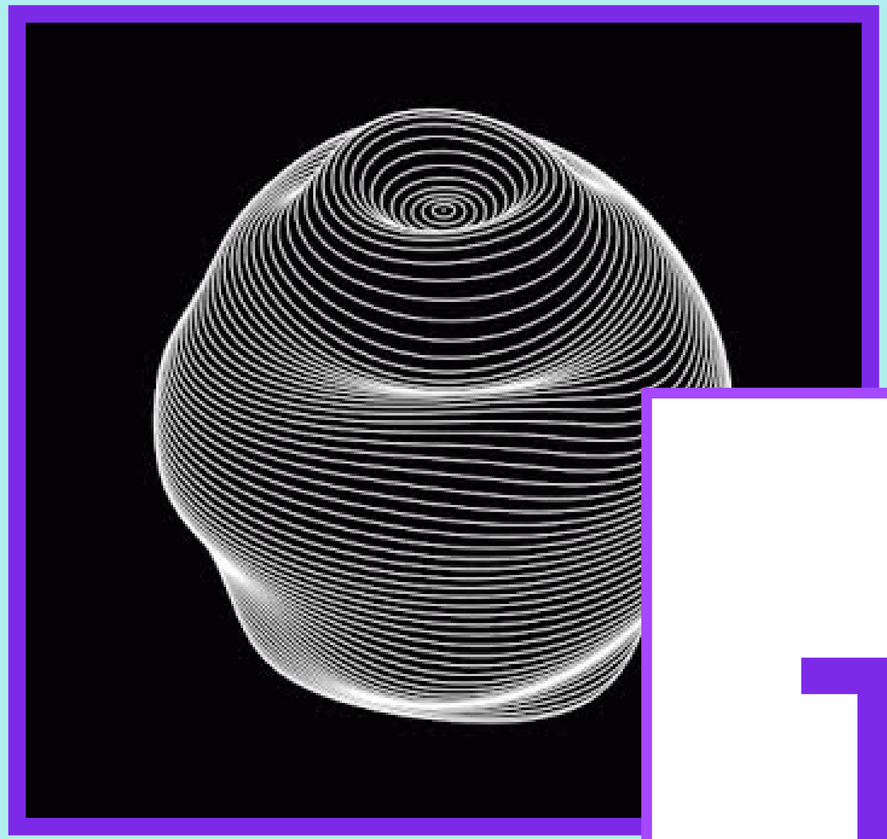
WHERE code is included

```html
<html>
  <head>
    <!-- CAN INCLUDE IN HEADER -->
  </head>
  <body>
    <!-- CAN INCLUDE IN TOP OF THE BODY -->
    <!-- MOST OF YOUR PAGE -->
    <!-- CAN INCLUDE IN BOTTOM OF THE BODY -->
  </body>
</html>
```

Generally speaking you can either:

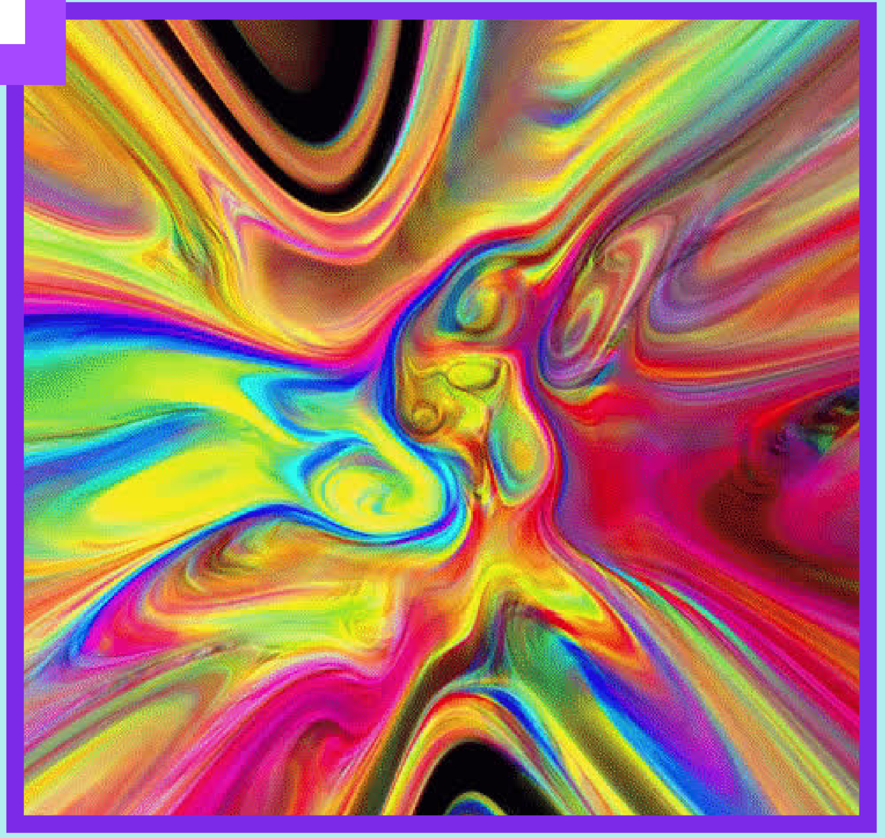
1. Include in the <head> or at the top of <body> if you **need** your javascript to run prior to your DOM elements doing initial render
2. Include at the end of <body> if you need **do not need** your javascript to run prior to your DOM elements initially rendering

We usually do (2)



The DOM

Presented by
Anna Azzam



Overview

- What is the DOM?
- Data types of DOM Elements
- Reading the DOM in JS
- Modifying the DOM in JS
- Reading and modifying live example
- Scrolling live example

HTML

Markup language creating web documents

CSS

Style sheet language for applying styles to a document

JavaScript

Scripting language to make your web page dynamic

What is the DOM?

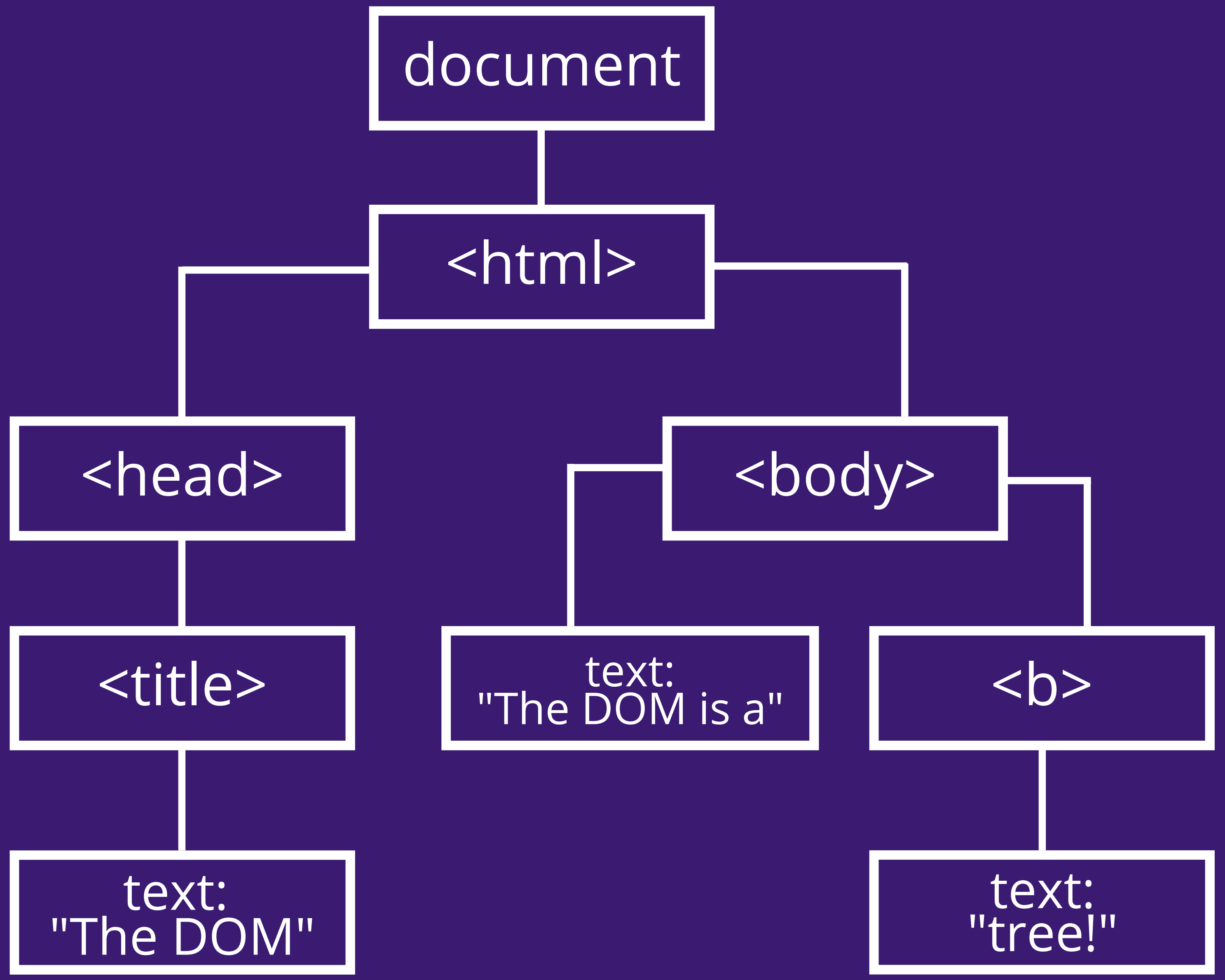
The DOM (Document Object Model) is an interface that allows JavaScript to interact with HTML through the browser.

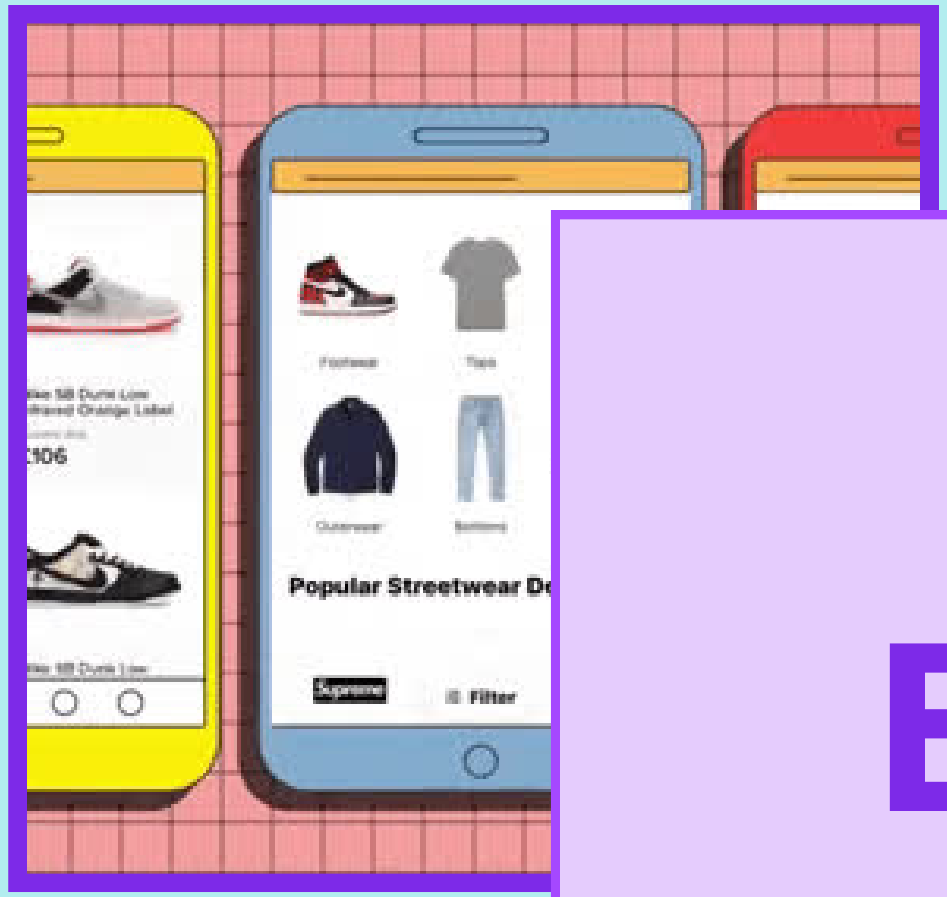
HTML **CSS** **JavaScript**

The DOM interface allows JavaScript to access and update the content, structure, and style of a document

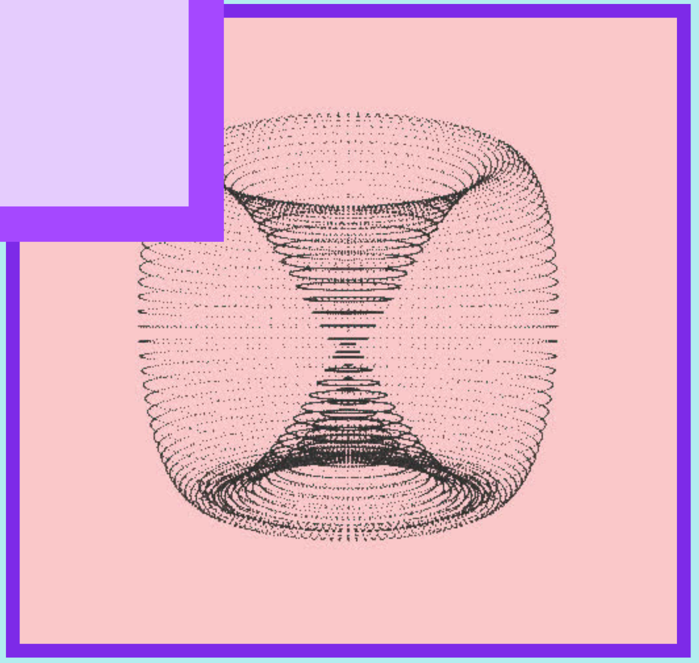
Tree-like structure of the DOM

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>The DOM</title>
  </head>
  <body>
    The DOM is a <b>tree!</b>
  </body>
</html>
```





DOM Elements and Data Types



DOM Data Types

To understand how the DOM is represented, we will introduce some new data types

Document

The type of the `document` object. Represents the root of the entire DOM.

Element

A node in the DOM tree. Objects of this type implement an interface that allows interacting with the document.

NodeList

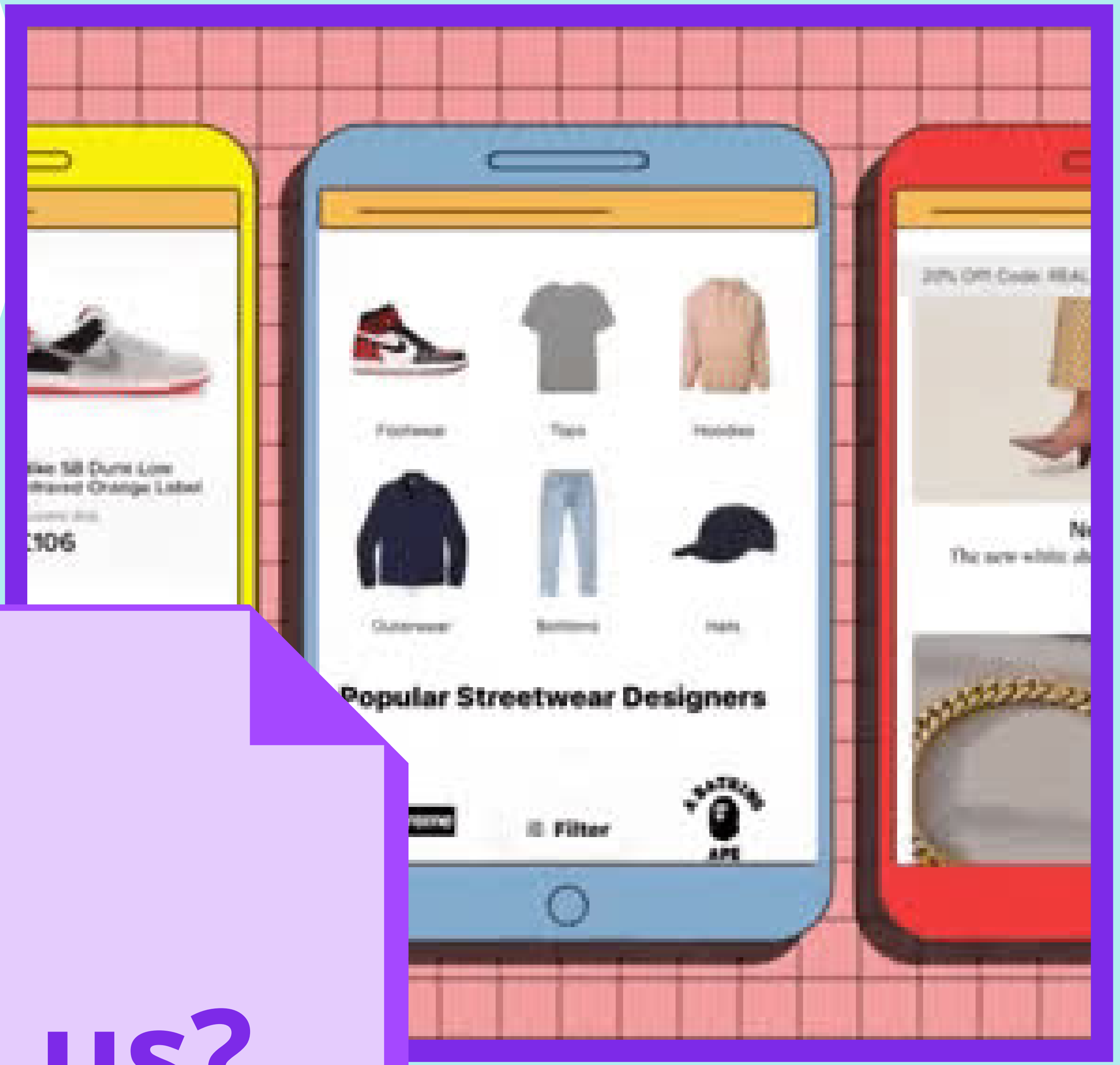
An array of elements, like the kind that is returned by the method.

Understanding DOM Elements

- Element is the base class for all types of objects in the Document
- Different HTML tags/elements correspond to different Element types in JS
- Different Element types include:

```
HTMLInputElement, HTMLSpanElement, HTMLDivElement,
HTMLScriptElement, HTMLHeadingElement,HTMLImageElement....
```

Q



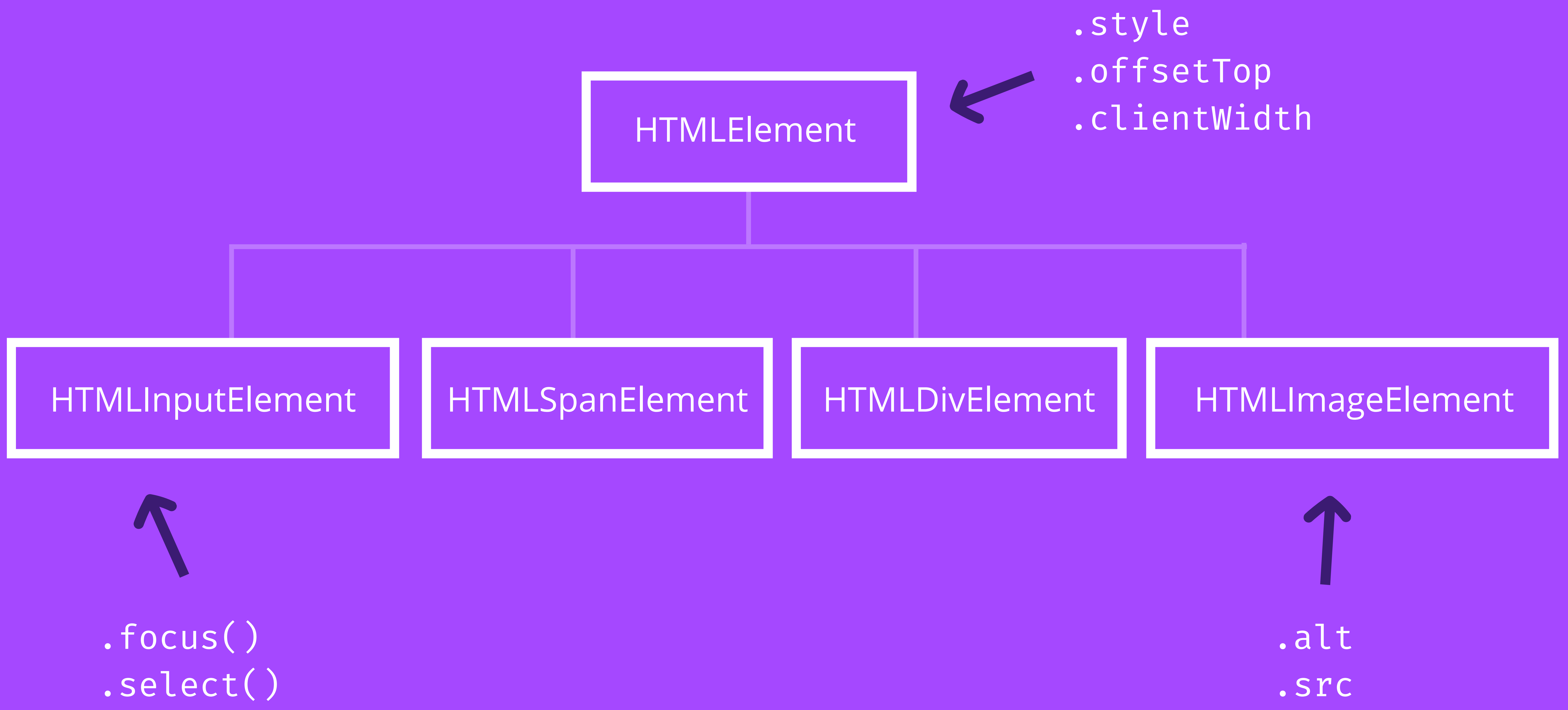
What does the Element interface give us?

Ways of *reading* things

e.g. Get the size, position, color or text of an element

Ways of *writing* things

e.g. Set an attribute, change the styling of an element


.style
.offsetTop
.clientWidth
HTMLElement
HTMLInputElement
HTMLSpanElement
HTMLDivElement
HTMLImageElement
.focus()
.select()
.alt
.src

HTMLElement

The HTMLElement interface represents any HTML element. Some elements directly implement this interface, while others implement it via an interface that inherits it.

MDN Web Docs

Reading the DOM

```
// Returns an html element with the given id
document.getElementById(id);

// Returns a DOM HTMLCollection of all matches
document.getElementsByTagName(name);
document.getElementsByClassName(classname);

// Returns the first Element that matches the selector
document.querySeletor(query);
```

Reading the DOM example

Writing to the DOM

```javascript
// Create a new div element
let element = document.createElement("div");
// Create a new text node
let textNode = document.createTextNode("Some text");

// Adding and removing elements
element.appendChild(textNode);
element.removeChild(textNode);

// Making changes to attributes
button.setAttribute("disabled", "");
```

Changing the style of an element

An element has a "style" property which corresponds to the "style" attribute of the HTML element.

This can be modified in the JavaScript.

```
// Changing element.style
element.style.left = "50px"; // Note: don't forget units!

// Adding 5px to the left value
let newLeft = parseInt(element.style.left, 10) + 5 + "px";
element.style.left = newLeft;

element.style.backgroundColor = "red"; // Note: camelCase
```

Getting the style of an element

Note: `element.style.left` will only be present on an element if the `left` property was set in inline styles, or by scripting (not if it was set in CSS).

```
// Getting computed style
let computedStyle = window.getComputedStyle(element, null)
let bgColor = computedStyle.getPropertyValue("background-
color")
```

Changing the classes of an element

Another way of modifying the style of the element is to change the classnames which exist on the element. This can be done using the "classList" property.

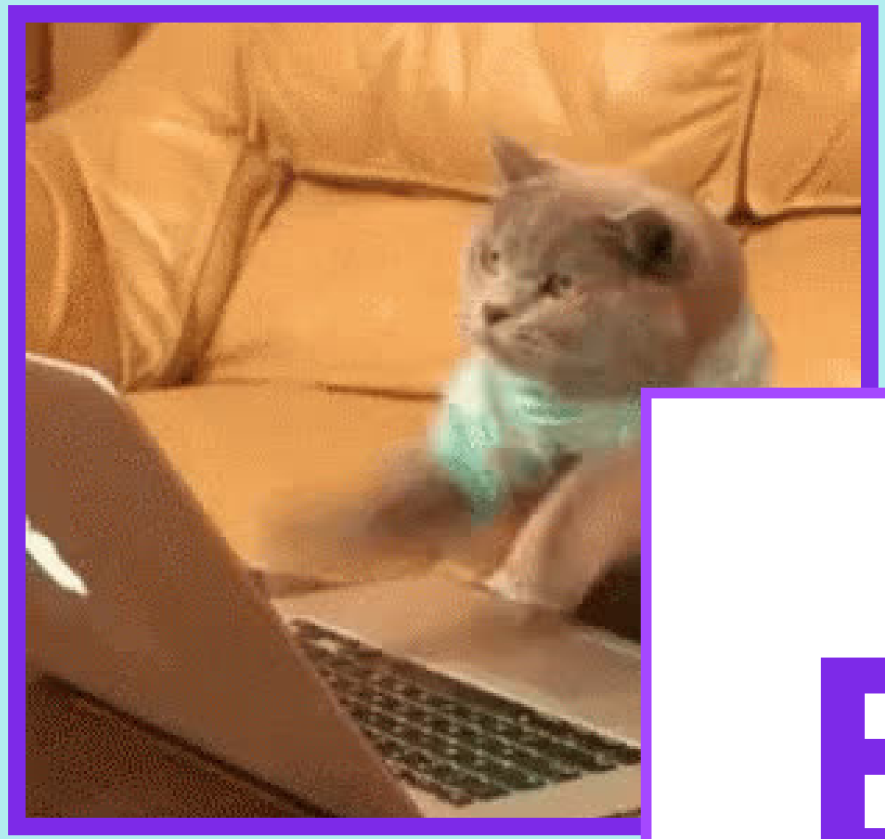
```
// Changing element.classList
element.classList.add("class");
element.classList.remove("class");
element.classList.toggle("class");
element.classList.contains("class"); // returns true if
class exists on element
```

Writing to the DOM example

Scrolling

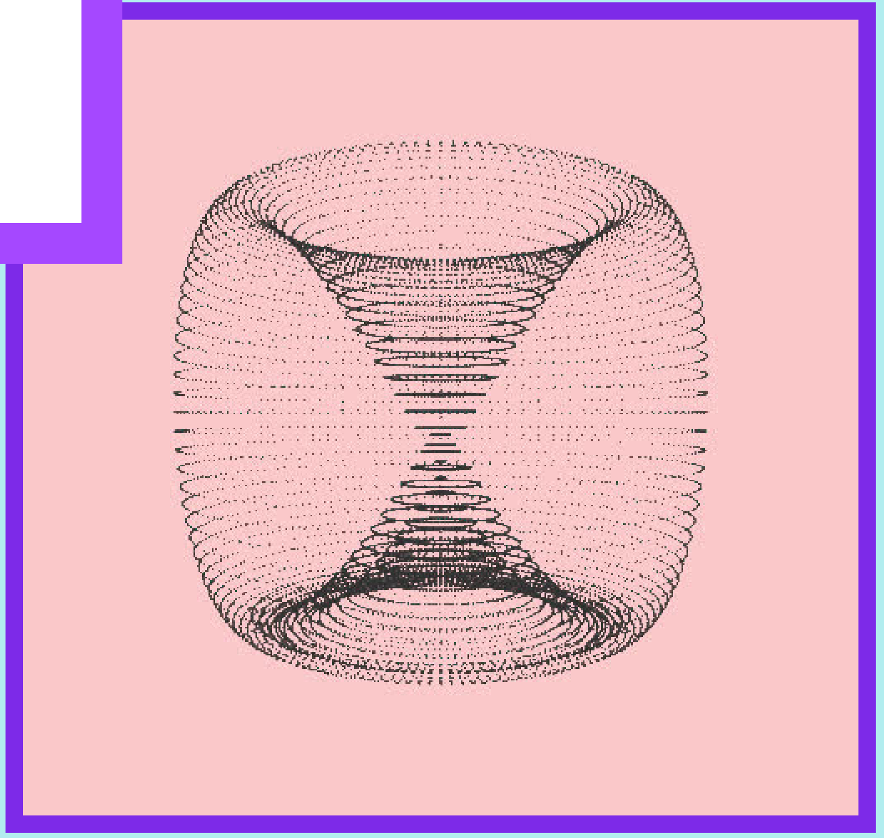
```
// Get the current scroll position of the page
console.log(window.scrollX);
console.log(window.scrollY);

// Scroll to a position on the page:
window.scrollTo({
  top: 100,
  left: 0,
  behavior: "smooth",
});
```

Scrolling to an element example



Events

Presented by
Anna Azzam



What is an Event?

An *event* is a signal that a "thing" has happened to a DOM element

This "thing" can be the element *loading*, a *click*, or a *key press*

We can use events to run JavaScript code in response to user actions

What are some examples of events?

Mouse events

```
click
dblclick
mouseup
mouseenter
mousedown
mouseleave
```

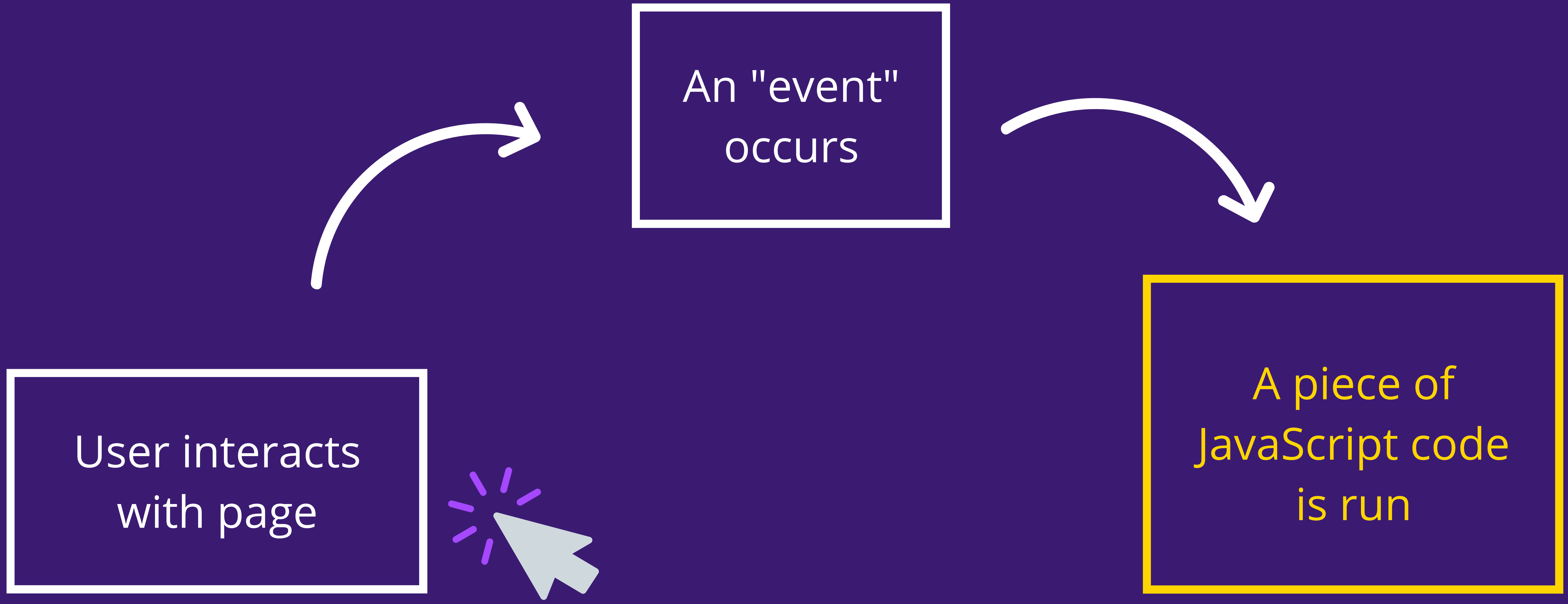
Keyboard events

```
keydown
keypress
keyup
```

...and more!

```
error
load
fullscreenchange
submit
canplay
canplaythrough
animationstart
```

Event Handlers



Adding Event Handlers

1) In HTML
2) DOM Property
3) addEventListener

```
<input
  value="Click me"
  onclick="alert('Clicked!')"
  type="button"
>
```

Adding Event Handlers

1) In HTML
2) DOM Property
3) addEventListener

```
let element = document.getElementById('btn');

element.onclick = () => {
  alert('Button was clicked!');
};
```

```
function doSomething() {
  alert('hello');
}

element.onclick = doSomething();
```

What's wrong with this code?

A

```
function doSomething() {
  alert('hello');
}

element.onclick = doSomething();
```

By adding parentheses, we are executing doSomething rather than assigning a function to onclick.

A

```
function doSomething() {
  alert('hello');
}

element.onclick = doSomething;
```

By adding parentheses, we are executing doSomething rather than assigning a function to onclick.

Adding Event Handlers

1) In HTML
2) DOM Property
3) addEventListener

```
// Definition:
target.addEventListener(
  type, // e.g. 'click', 'mousemove'
  listener, // the callback
  [options]
);

// Example:
let element = document.getElementById('btn');
let handler = () => {
  alert('button was clicked');
})

element.addEventListener('click', handler);
element.removeEventListener('click', handler);
```

Event Interface

The parameter to an event handler is an event object

```
document.addEventListener('mousemove', (event) => {
  console.log(event.clientX);
  console.log(event.clientY);
});
```

The event object represents the event that has taken place, and has properties describing details of the event

Event Interface Properties

Some of the properties on the event interface include:

```
event.currentTarget // current element handler is running on
event.timeStamp // time the event was created (in ms)
event.type // name of the event, e.g. 'click'
```

Different types of events have specific properties:

```
event.clientX // A MouseEvent has the X and Y coordinate
event.key // A KeyboardEvent has the keycode of the key that was pressed
```

Keyboard Events Example

The Event Loop

- The event loop is a single-threaded loop which runs in the browser, and manages all events.
- When an event is triggered, the event is added to the queue.

```
while (queue.waitForMessage()) {
  queue.processNextMessage();
}
```

The Event Loop

JavaScript uses a run-to-completion model, meaning it will not handle a new event until the current event has completed.

Event Capturing and Bubbling

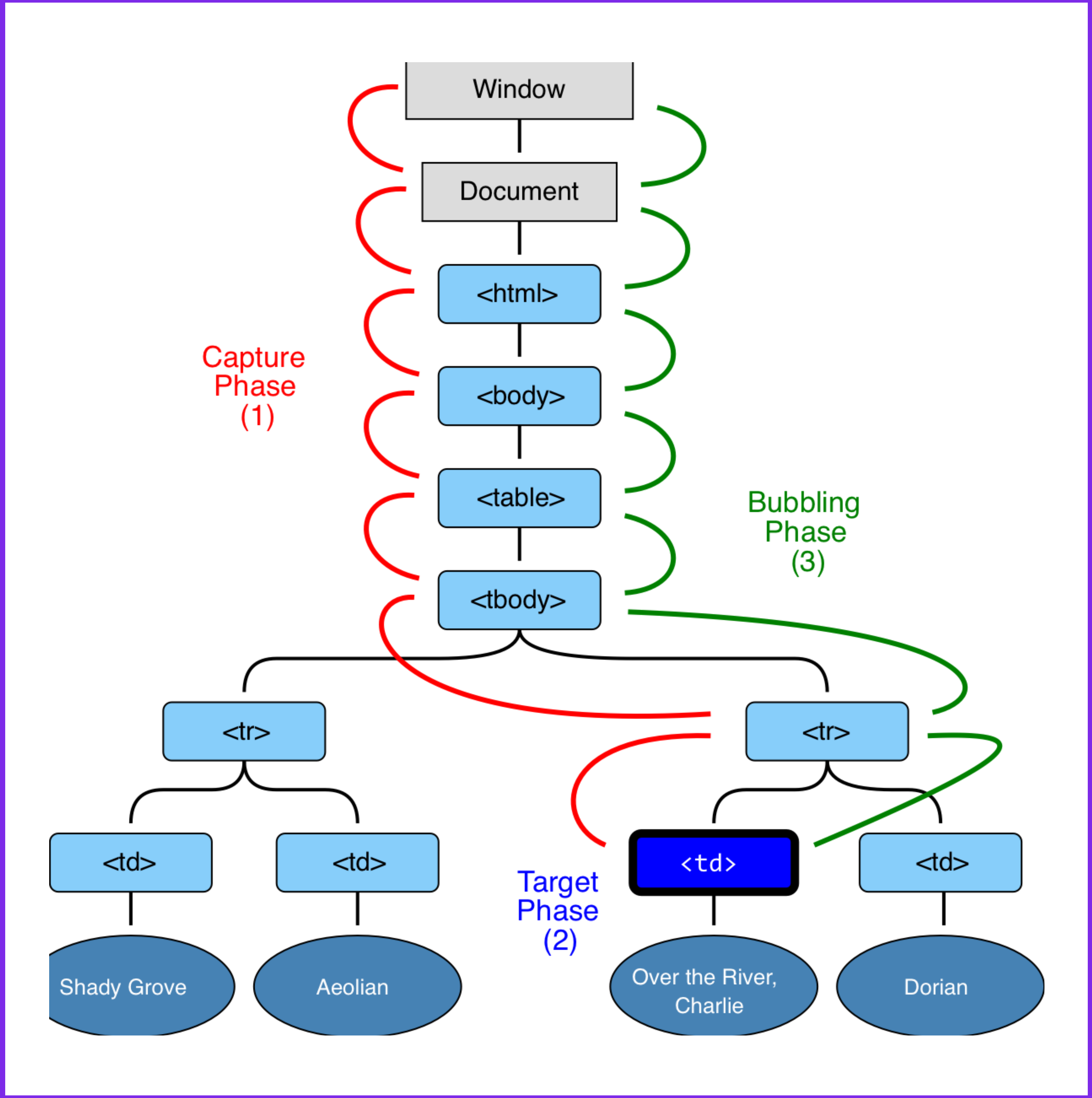


Image Source: https://javascript.info/bubbling-and-capturing

Event Capturing and Bubbling Example

Prevent Default

Some types of DOM elements have default behaviour, e.g.

1. Clicking an input checkbox toggles the checkbox
2. Images have a default drag and drop behaviour to allow you to drag them into another location
3. Key presses into a text input field has the default behaviour of entering that text into the input field

To stop the default behaviour of an event, use:

```
event.preventDefault()
```

Prevent Default Example

Basketball drag and drop game

COMP6080
Web Front-End Programming

Javascript

Closures

Closures

Javascript "closures" is a concept that a few programming languages have (e.g. Python too).

Closures allow us to create execution environments (similar to what a new object in a class does). This used to be *needed* before ES6 especially through the use of IIFE.

Since ES6 (modern javascript) they are virtually never needed, and even more rarely used. However, understanding the basics of them is still important.

We will demo some a very basic example.

Another demonstration of their use can be found here.

Feedback

Forms

Presented by
Anna Azzam

Forms

- A HTML <form> element is a way of defining a form which is used to get user input
- They consist of different types of input elements:
  - text fields
  - checkboxes
  - radio buttons
  - submit buttons
- We specify the type of input element using the type attribute:

```
<form name="user_form">
  First name:<br>
  <input type="text" name="firstname">
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

Reading forms in JavaScript

document.forms

When you have forms in your document, they can be found in a special document property called `document.forms`

This is a “named collection”, i.e. it’s both named and ordered. We can use both the name or the number in the document to get the form.

```
document.forms.test // the form with name="test"
document.forms["test"] // also the form with name="test"
document.forms[0] // the first form in the document
```

form.elements

Each form has a field `form.elements` which has all of the elements in that form.
This is also a “named collection”

HTML

```html
<form>
   <input type="text" name="fname">
  <input type="radio" name="age" value="10">
  <input type="radio" name="age" value="20">
</form>
```

JS

```js
const form = document.forms[0];

// element with the name "fname"
form.elements.fname;
// shorter notation:
form.fname;

// since there are multiple elements with
the name "age", this returns a collection
const ages = form.elements.age;
```

Backreferences

Each form element stores a backreference to the form it came from, `element.form`

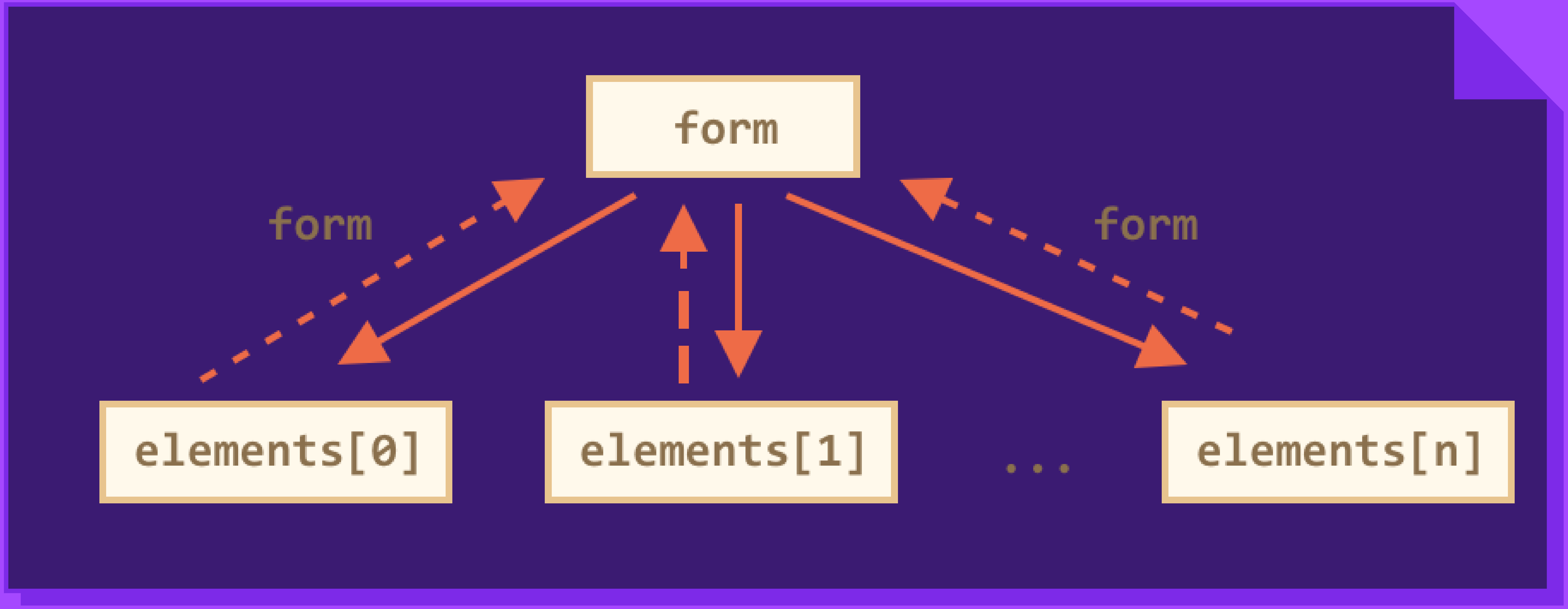


Image source: javascript.info

Backreferences

HTML

```html
<form>
  <input type="text" name="login">
</form>
```

JS

```js
const form = document.forms[0];
const login = form.login;

console.log(login.form)
```

What will this console.log?

Form Values

To get the value of a form element:

```
// To get the text for an input element or textarea:
input.value

// To get a boolean for a checkbox or radio button
input.checked

// For a <select> tag
select.options // the collection of <option>s
select.value // the value of the currently selected option,
select.selectedIndex // the index of the currently selected option
```

Submit Buttons and onsubmit

- `<input type="submit">` defines a button for submitting the form data to a form-handler
- Clicking a submit button triggers a "submit" event on the form
- We can listen to this event, and run form validation

```
form.addEventListener('submit', (event) => {
  event.preventDefault();
  // form validation can go here
});
```

Forms Example

Local Storage

Presented by
Anna Azzam

Persistence

State in React and JavaScript is ***transient***, meaning we lose it after a page refresh

Sometimes we want data to be ***persistent*** between sessions

This can be done using a ***database***, or stored client-side via ***localstorage***

What is Local Storage?

- **window.localStorage** is an API that allows you to read and write to a storage object in the document
- The stored data in this storage object is **persisted between sessions**
- This means we can save and retrieve data when a user closes their tab or browser

Pros and Cons

When should I not use Local Storage?

1. When security of data is important
2. For large amounts of data (localstorage has limited storage)
3. For complex data (only stores strings)
4. Data needed on multiple devices

When should I use Local Storage?

1. When there is no server
2. For data which is not crucial if it is lost (e.g. or remembering view options or user settings)

Localstorage API

```
// Add a data item given the key and value
localStorage.setItem(key, value);

// Retrieves an item from localstorage given a key
const value = localStorage.getItem(key);

// Remove an item with a given key from localstorage
localStorage.removeItem(key);

// Remove all items from localstorage
localStorage.clear();
```

Local storage example