KAZAKH BRITISH TECHNICAL UNIVERSITY

# WEB DEVELOPMENT

Lesson 3

# JavaScript

Style guide: https://github.com/airbnb/javascript

## JavaScript

- High level
- Dynamic
- Dynamically typed
- Interpreted

**JavaScript has nothing to do with Java**

# ECMAScript — JavaScript standart

- ECMAScript 1 (1997)
- …
- ECMAScript 5 (2009) — ES5
- …
- ECMAScript 2015 — ES6 (classes, arrow functions, let/const, promises)
- ECMAScript 2017 (async/await)
- ECMAScript 2020 (optional chaining **?.**,)
- ECMAScript 2023 (Array.findLast, toSorted, toReversed)
- ECMAScript 2025 ← we are here

## Data types

**Primitive types:**

- Number
- String
- Boolean
- Null
- Undefined
- Symbol — ES6
- BigInt — ES2020

**Reference type:**

- Object (includes arrays, functions, dates, etc.)

# Good news

If you know C++, C or Java — JavaScript has similar syntax

```
i = 3;

i = i * 10 + 3 + (i / 10);

while (i >= 0) {
    sum += i*i;   // Comment
    i--;
}

for (i = 0; i < 10; i++)  {

}
/* this is a comment */
```

```
if (i < 3) {
    i = foobar(i);
} else {
    i = i * .02;
}
```

Most C operators work:

```
* / % + - ! >= <= > < && || ?:
```

```
function foobar(i) { return i;}

continue/break/return
```

# Variable scoping

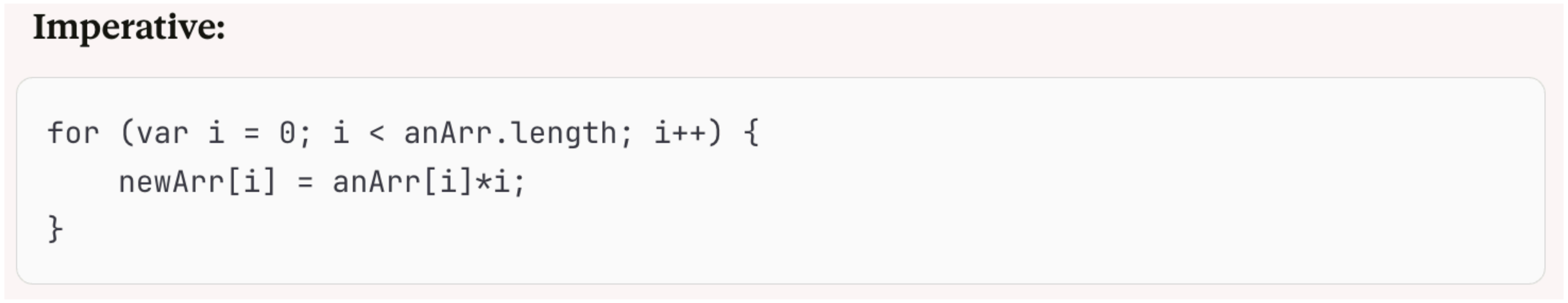
```javascript
let count = 5;         // block-scoped, reassignable
const name = "Ali";    // block-scoped, NOT reassignable

if (true) {
    let x = 10;
    const y = 20;
}
// x and y are NOT accessible here
```

```javascript
// var is function-scoped (confusing!)
if (true) {
    var leaked = "oops";
}
// leaked is accessible here — unexpected!

// Rule: always use const by default,
// use let only when you need to reassign,
// never use var
```

# ECMAScript version 6 extensions

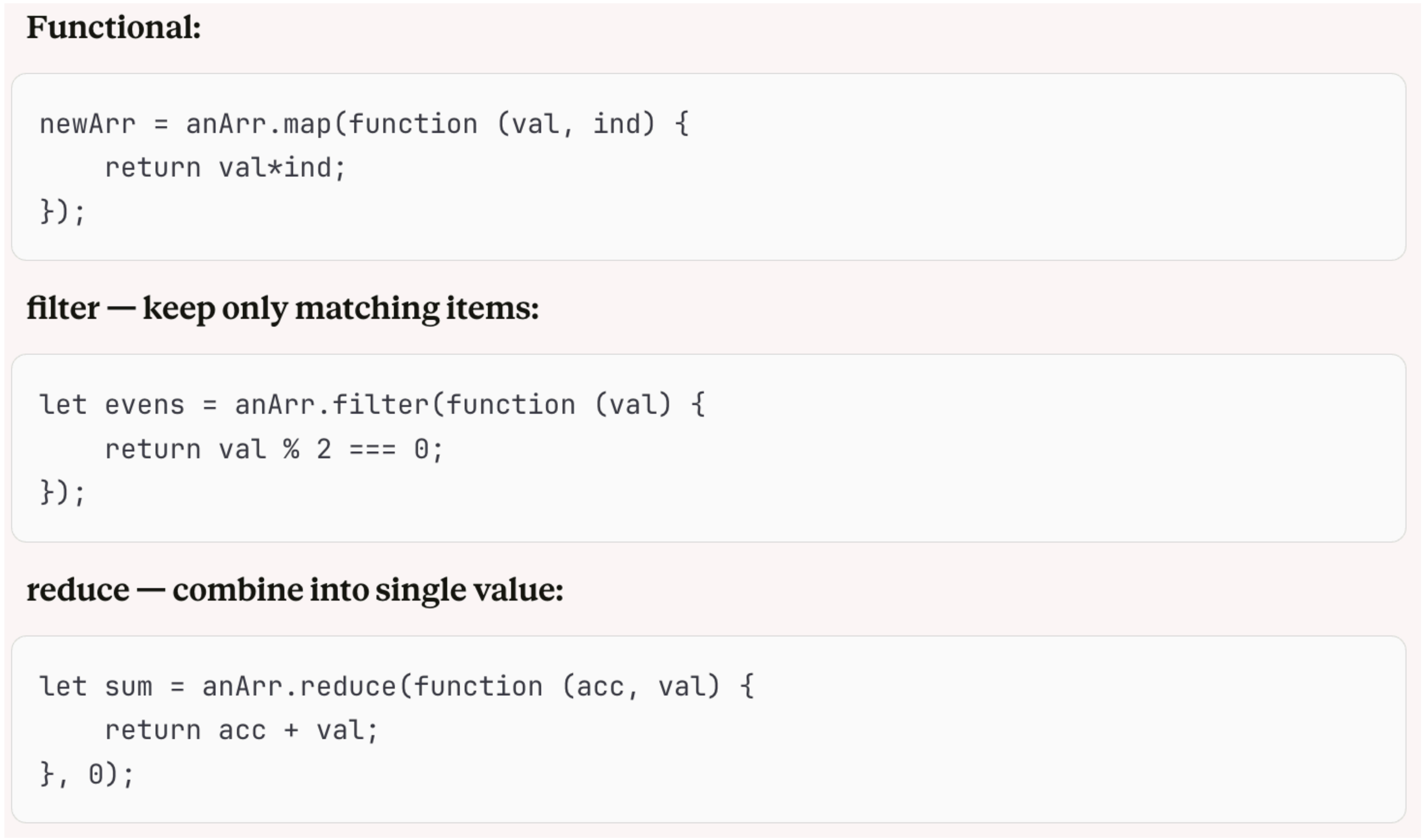
```
class Rectangle extends Shape {  // Definition and Inheritance
  constructor(height, width) {
     super(height, width);
     this.height = height;
     this.width = width;
  }
  area() {                         // Method definition
    return this.width * this.height;
  }
  static countRects() {           // Static method
     ...
  }
}

var r = new Rectangle(10,20);
```

## Imperative vs Functional Programming



**Imperative:**

```
for (var i = 0; i < anArr.length; i++) {
    newArr[i] = anArr[i]*i;
}
```



**Functional:**

```
newArr = anArr.map(function (val, ind) {
    return val*ind;
});
```

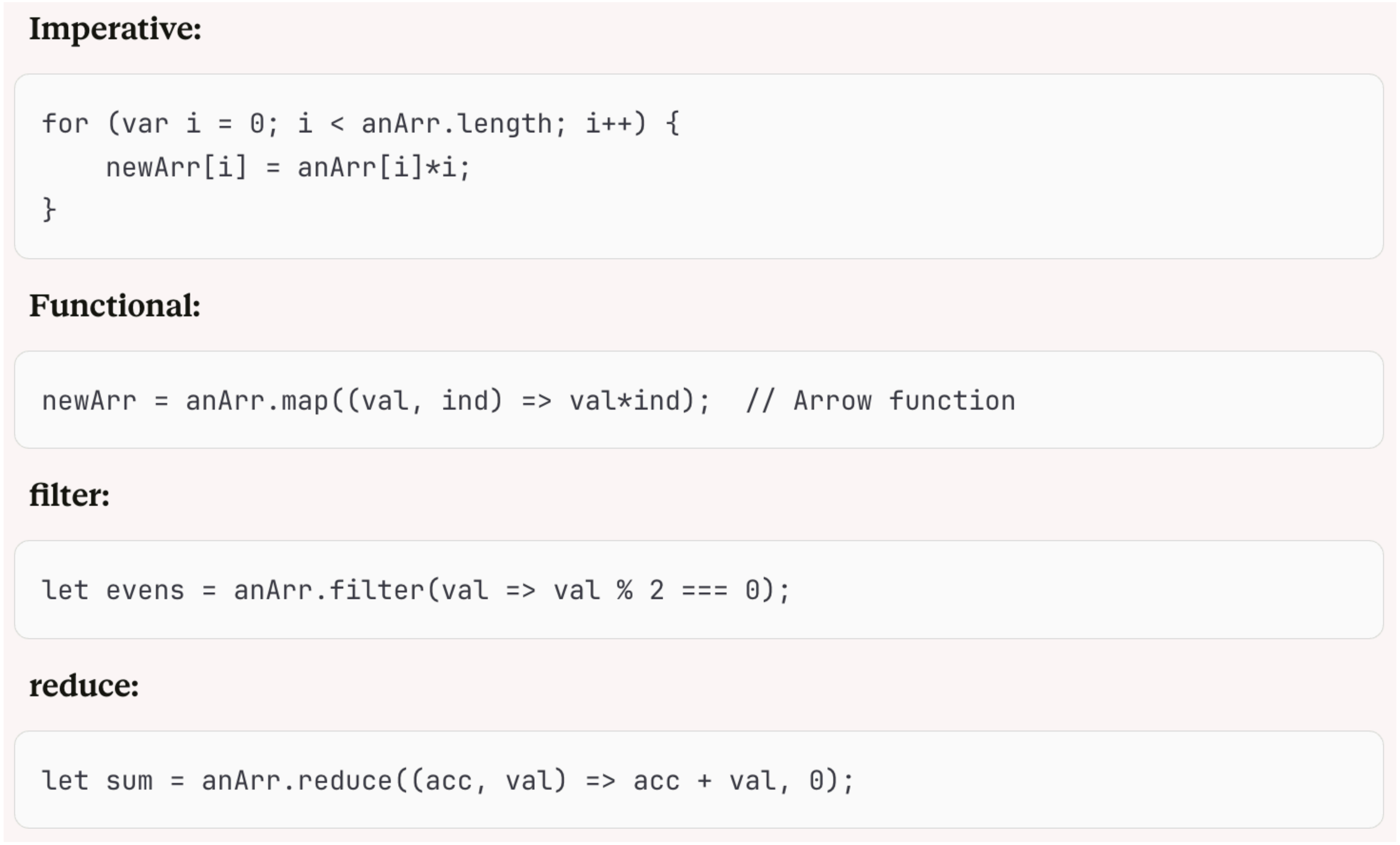
**filter — keep only matching items:**

```
let evens = anArr.filter(function (val) {
    return val % 2 === 0;
});
```

**reduce — combine into single value:**

```
let sum = anArr.reduce(function (acc, val) {
    return acc + val;
}, 0);
```

# Functional Programming - ECMAScript 6



**Imperative:**

```
for (var i = 0; i < anArr.length; i++) {
    newArr[i] = anArr[i]*i;
}
```

**Functional:**

```
newArr = anArr.map((val, ind) => val*ind);  // Arrow function
```

**filter:**

```
let evens = anArr.filter(val => val % 2 === 0);
```

**reduce:**

```
let sum = anArr.reduce((acc, val) => acc + val, 0);
```

## JavaScript Object Notation (JSON)

```
var obj = { ps: 'str', pn: 1, pa: [1,'two',3,4], po: { sop: 1}};

var s = JSON.stringify(obj) =
    '{"ps":"str","pn":1,"pa":[1,"two",3,4],"po":{"sop":1}}'
```

typeof s == 'string'
**JSON.parse**(s)  // returns object with same properties

- JSON is the standard format for sending data to and from a browser

# Some JavaScript idioms

```
// Old way (still works)
hostname = hostname || "localhost";
port = port || 80;

// Modern (ES2020) — safer with falsy values
hostname = hostname ?? "localhost";
port = port ?? 80;
// ?? only falls back on null/undefined
// || falls back on ANY falsy (0, "", false)
```

```
// Old way
var prop = obj && obj.address && obj.address.city;

// Modern (ES2020) — optional chaining
var prop = obj?.address?.city;
```

```
// Old way
var msg = "Hello, " + name + "! You are " + age;

// Modern — template literals
var msg = `Hello, ${name}! You are ${age}`;
```

## **D**ocument **O**bject **M**odel

## DOM hierarchy

- Rooted at `window.document`
- Follows HTML document structure
  - `window.document.head`
  - `window.document.body`
- DOM objects have tons (~250) of properties,

  most private

# Accessing DOM Nodes

- Walk DOM hierarchy (not recommended)
  - `element = document.body.firstChild.nextSibling.firstChild;`
- Use DOM lookup method. An example using ids:
  - `element = document.getElementById(“div1”);`
  - `getElementsByClassName(), getElementsByTagName()`
- Many: getElementsByClassName(), getElementsByTagName(), …
  - `document.body.firstChild.getElementsByTagName()`
- Modern & recommended — CSS selectors:
  - `// Select first match`
  - `element = document.querySelector("#div1");`
  - `element = document.querySelector(".menu-item");`
  - `element = document.querySelector("div > p.active”);`
  - `// Select ALL matches (returns NodeList)`
  - `elements = document.querySelectorAll(“.menu-item");`
  - `elements = document.querySelectorAll("ul li");`

# More commonly used Node properties

- `element.textContent       // get/set text only`
- `element.innerHTML         // get/set text + HTML tags`

- `element.getAttribute("src")`
- `element.setAttribute("src", “photo.jpg")`

- `element.classList.add("active")`
- `element.classList.remove("active")`
- `element.classList.toggle("done")`
- `element.classList.contains("done")   // returns true/false`

- `input.value               // get/set input field value`
- `input.value = ""          // clear input`

# DOM and CSS interactions

- `element.style.color = "#ff0000";`
- `element.style.backgroundColor = "#eee";`
- `element.style.display = “none";`

- `// define styles in CSS (toggle classes instead of inline styles)`
- `// .hidden { display: none; }`
- `// .highlight { background-color: yellow; }`

- `element.classList.add("hidden");`
- `element.classList.remove("highlight");`

## Changing the Node structure

- `let element = document.createElement("p");`
- `element.textContent = "Hello!";`
- `element.classList.add("item");`
- `parent.appendChild(element);`

- `// Insert at a specific position:`
- `parent.insertBefore(newElement, referenceElement);`

- `// Remove:`

-
  ```
  element.remove();                // modern, simple
  parent.removeChild(element);    // older way
  ```

- `// Clone:`
- `let copy = element.cloneNode(true);  // true = deep copy`

## More DOM operations

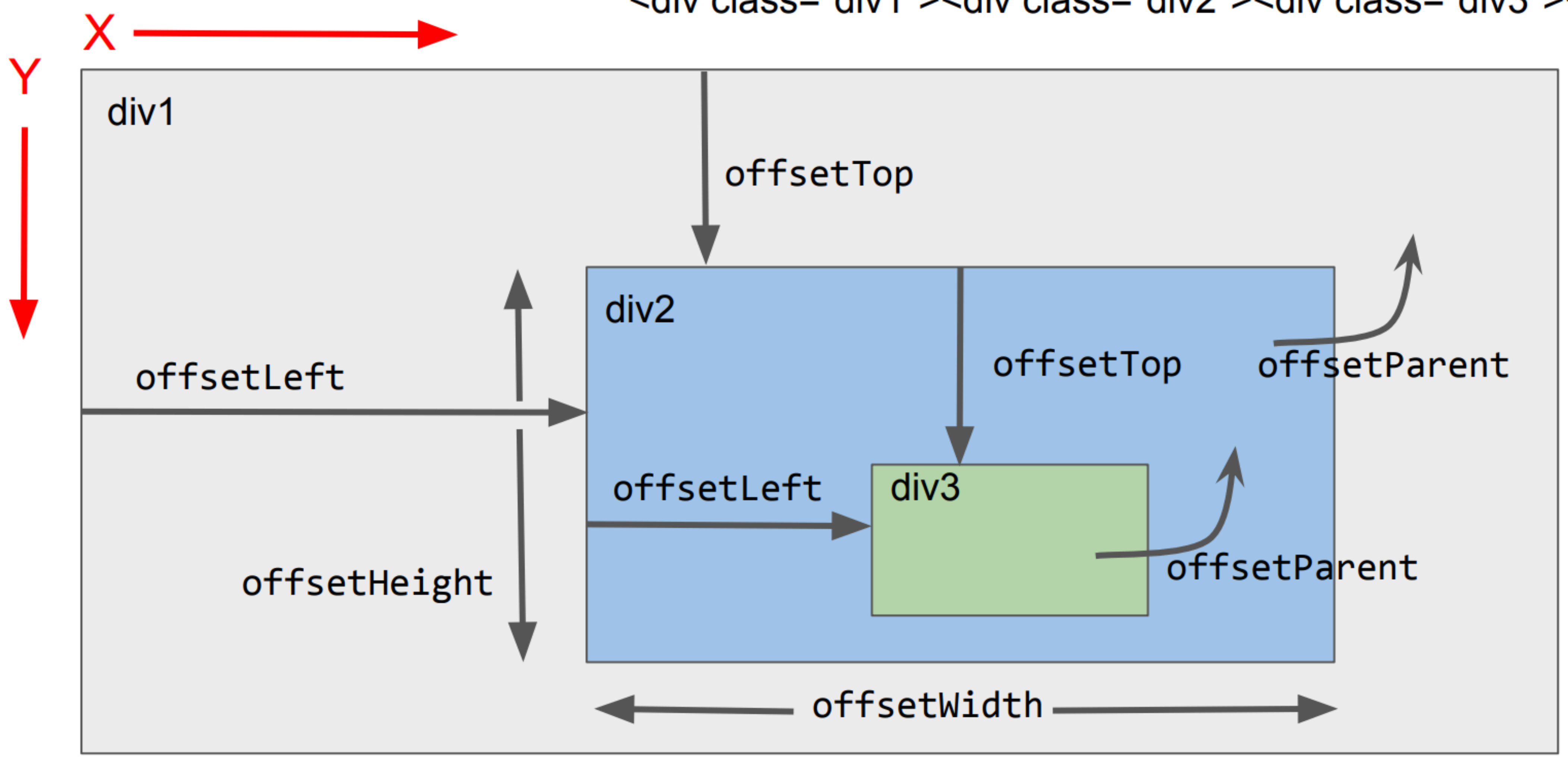
- `window.location.href = “newPage.html";`
- `console.log("Reached point A”);`
- `console.error("something broke”);`
- `alert("Wow!"); confirm("OK?");`

# DOM's Coordinate System

- The screen origin is at the upper left; y increases as you go down
- The position of an element is determined by the upper-left outside corner of its margin
- Read location with `element.offsetLeft`, `element.offsetTop`

# DOM's Coordinate System

<div class="div1"><div class="div2"><div class="div3"></div></div>/</div>



## JavaScript and DOM Events

- Mouse-related: mouse movement, button click, enter/ leave element
- Keyboard-related: down, up, press
- Focus-related: focus in, focus out (blur)
- Input field changed, Form submitted

## Event handling

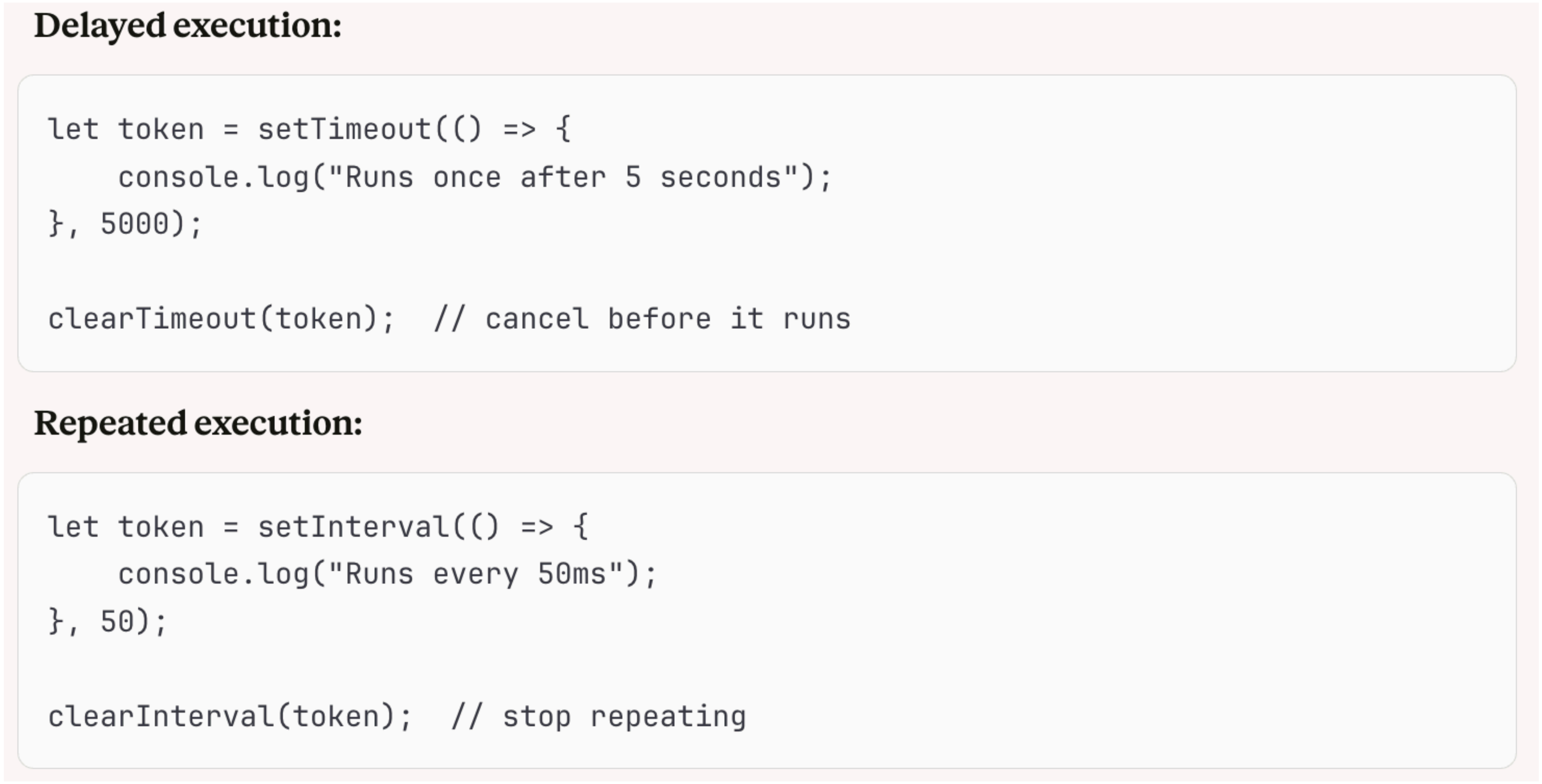
1. What happened: the event of interest
2. Where it happened: an element of interest.
3. What to do: JavaScript to invoke when the event occurs on the element.

## Specifying the JavaScript of an Event

- Option #1: in the HTML(avoid this):
  - `<div onclick=“divClicked();”>…</div>`
  - `<!-- mixes HTML and JS – hard to maintain -->`
- Option #2: addEventListener (recommended):
  - `element.addEventListener("click", mouseClick);`

  -
    ```
    // with anonymous arrow function
    element.addEventListener("click", (event) => {
    console.log(event.target);
    });
    ```

# Timer Event



**Delayed execution:**

```
let token = setTimeout(() => {
    console.log("Runs once after 5 seconds");
}, 5000);

clearTimeout(token);  // cancel before it runs
```

**Repeated execution:**

```
let token = setInterval(() => {
    console.log("Runs every 50ms");
}, 50);

clearInterval(token);  // stop repeating
```

## Questions?