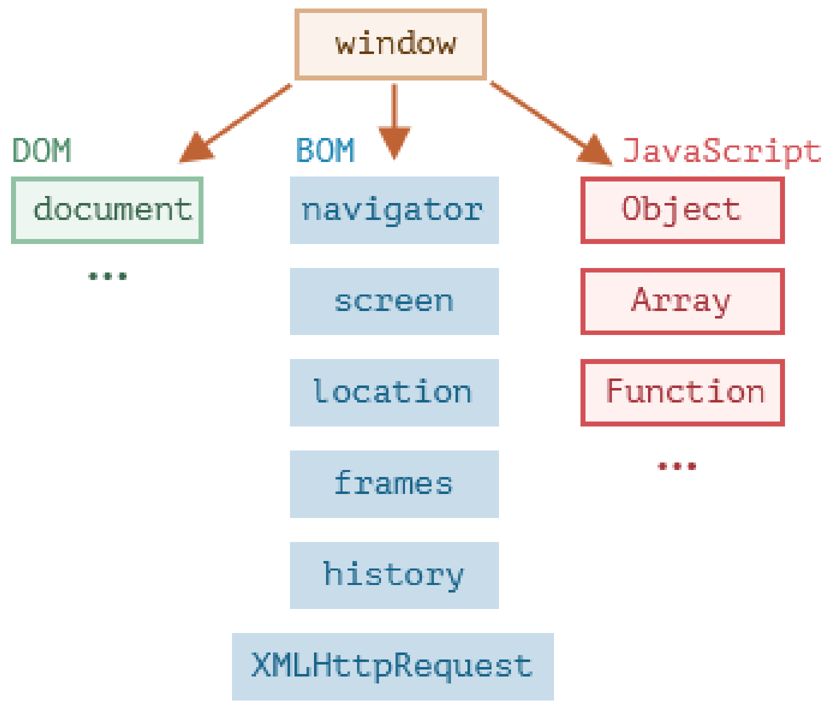
Document Object Model (DOM)

Sridhar Alagar

Browser env

Browser (host) env provides its own object and functions

By using them webpage can be controlled



Window – root object

1. It is a global obj for JS code

```javascript
function sayHi() {
  alert("Hello");
}

// global functions are methods of the global object:
window.sayHi();
```

2. Represents browser window and provides methods to control it

```javascript
alert(window.innerHeight); // inner window height
```

Document Object Model (DOM)

Backbone of HTML page?

- Tags (elements)

In a DOM, all tags are objects

- Nested tags are the children of the enclosing tag
- Text inside a tag is an object (leaf) as well

All DOM objects are accessible using JS and can be modified

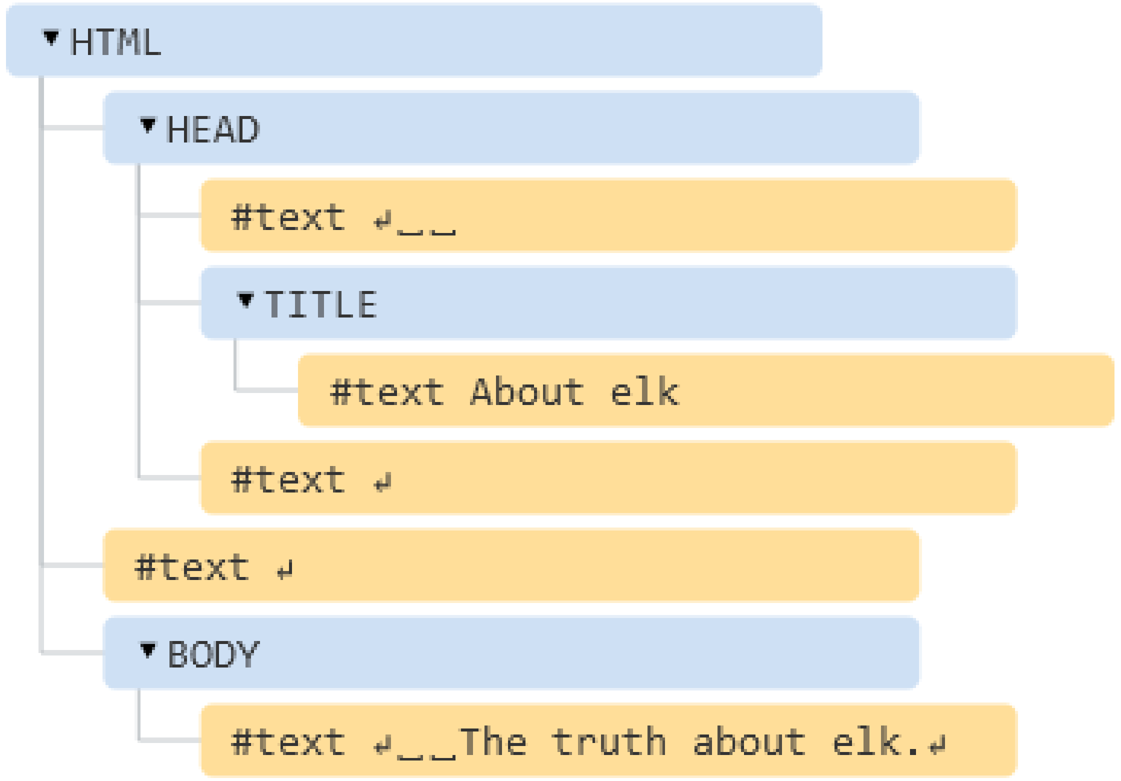
DOM tree

Every html tag is an object

Nested tags are children of the enclosing one

Text inside a tag is an object (leaf)

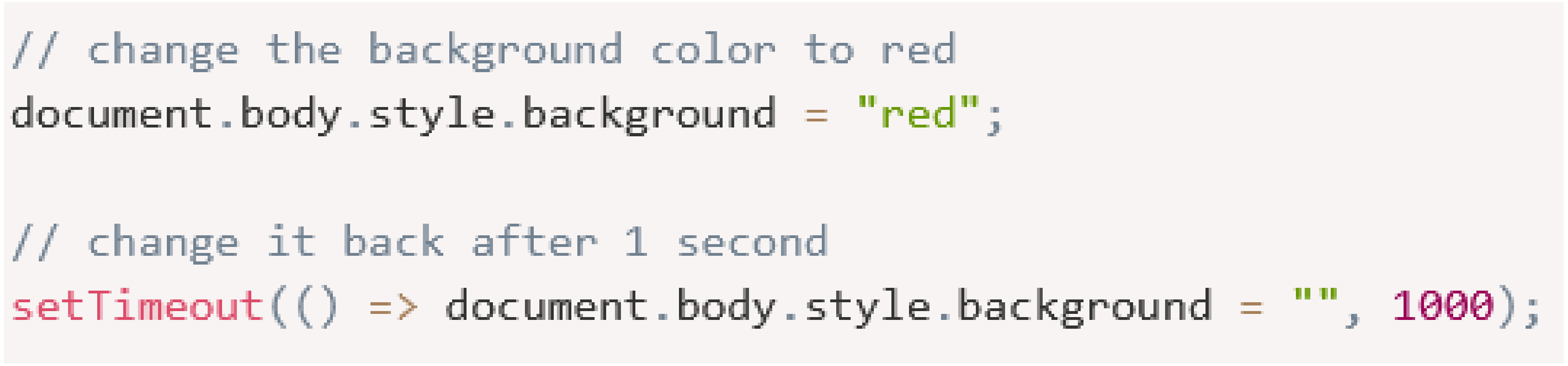
```
<!DOCTYPE HTML>
<html>
<head>
  <title>About elk</title>
</head>
<body>
  The truth about elk.
</body>
</html>
```



Accessing DOM

‘document’ object is the main entry point to the page

Can change/create anything on the page using it



```
// change the background color to red
document.body.style.background = "red";

// change it back after 1 second
setTimeout(() => document.body.style.background = "", 1000);
```

DOM Creation – who, how, and when?

DOM is created by the browser

Browser parses the document and creates the DOM tree

DOM is created as the browser parses the HTML document

- Fully constructed when the *DOMContentLoaded* event is fired

Everything in HTML is part of DOM

Even comments

There are 12 node types (https://dom.spec.whatwg.org/#node)

But only 4 are mainly used

1. document – the entry point to DOM
2. element node – tags, the tree building blocks
3. text node – contains text (leaf node)
4. comment - JS can read it from DOM

https://software.hixie.ch/utilities/js/live-dom-viewer/

Walking the DOM

To modify an element, we need to get its DOM object starting from document

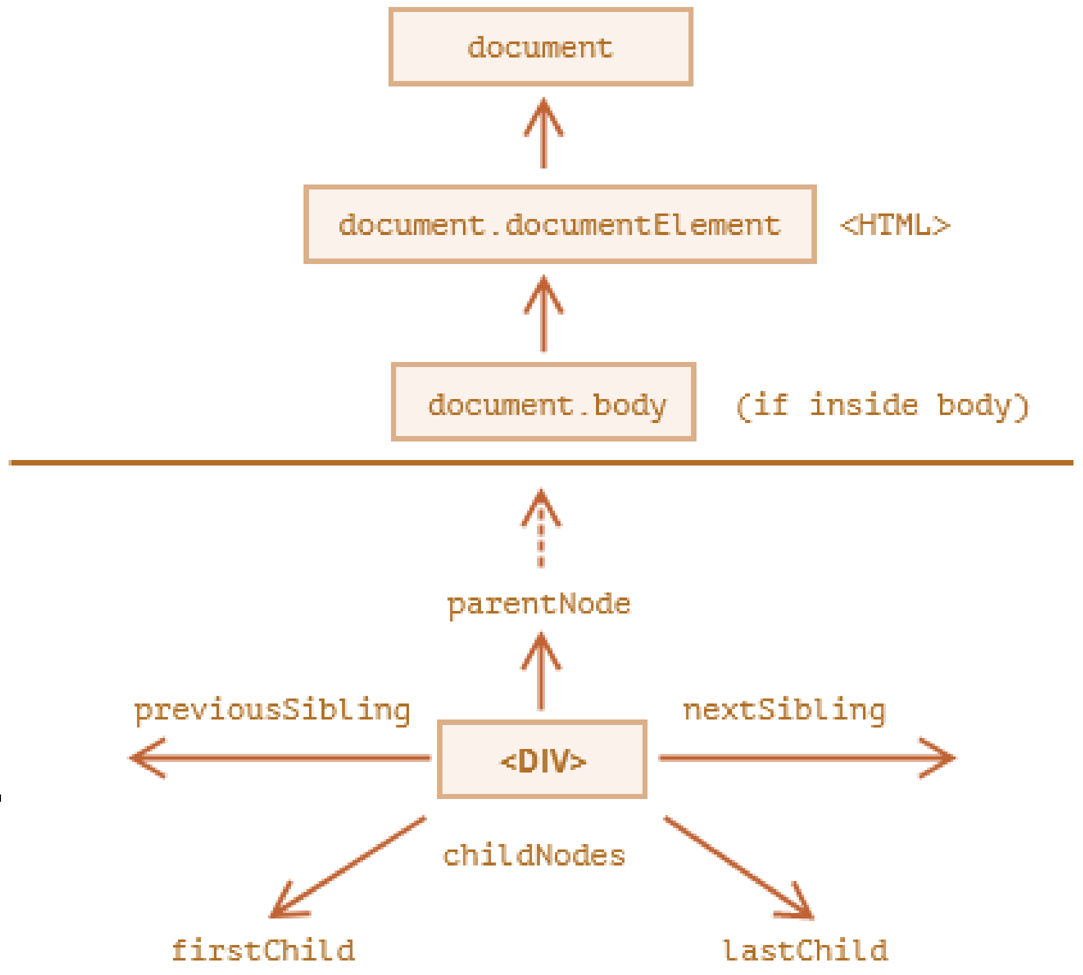
Topmost tree nodes are available as document properties

document.body

document.head

Children – directly nested elements

Descendants – children, their children…



Iterating Children

Use childNodes collection

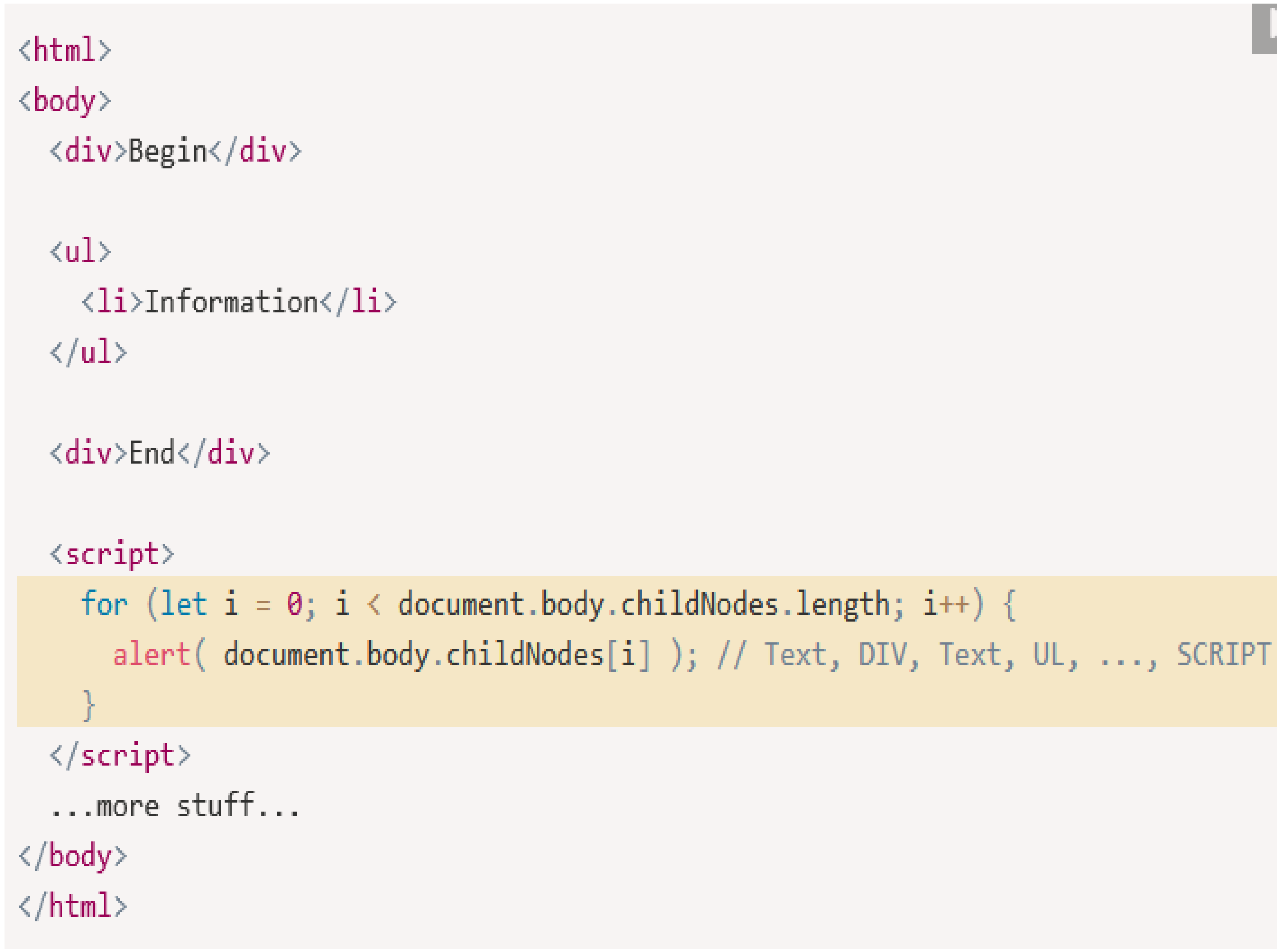
Can use for…of to iterate

childNodes not an array

Use Array.from() to create real array

firstChild and lastChild give fast access

parentNode, nextSibling, previousSibling are properties of a node

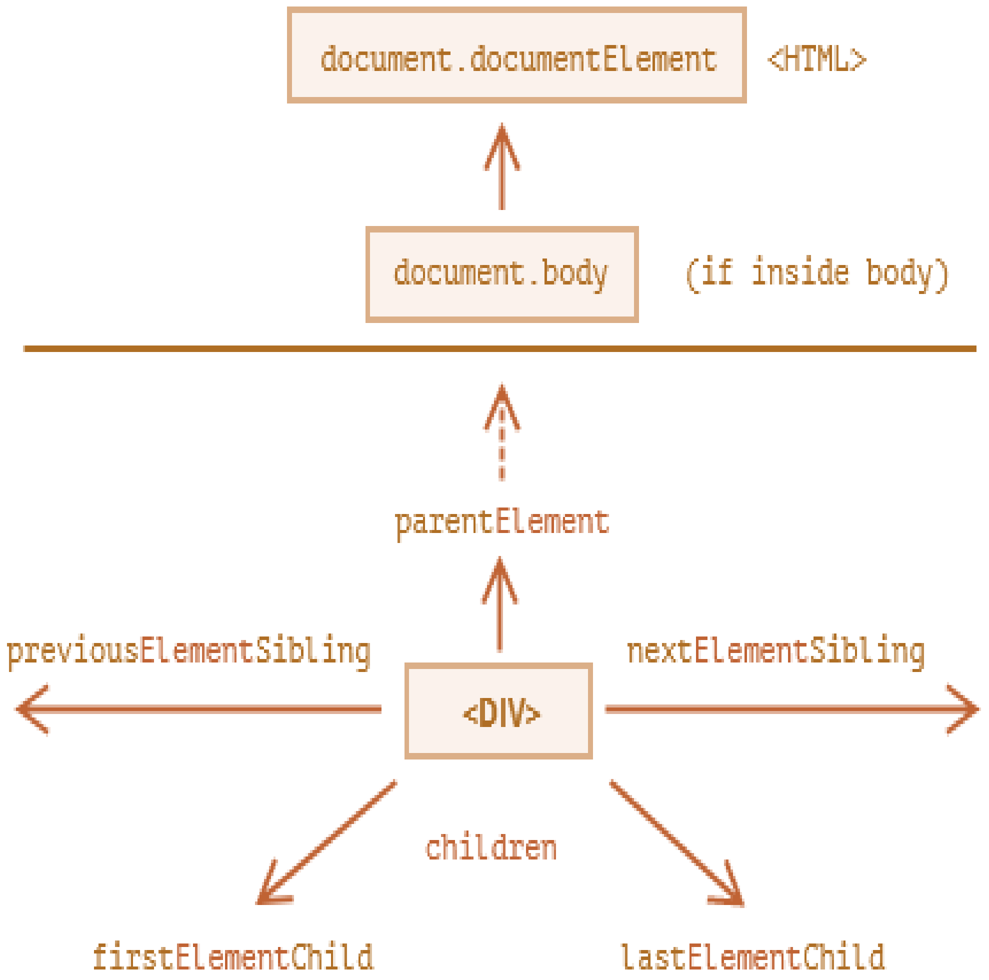


```
<html>
<body>
  <div>Begin</div>

  <ul>
    <li>Information</li>
  </ul>

  <div>End</div>

  <script>
    for (let i = 0; i < document.body.childNodes.length; i++) {
      alert( document.body.childNodes[i] ); // Text, DIV, Text, UL, ..., SCRIPT
    }
  </script>
  ...more stuff...
</body>
</html>
```

Element only traversal

childNodes contain text nodes, element nodes, comment nodes

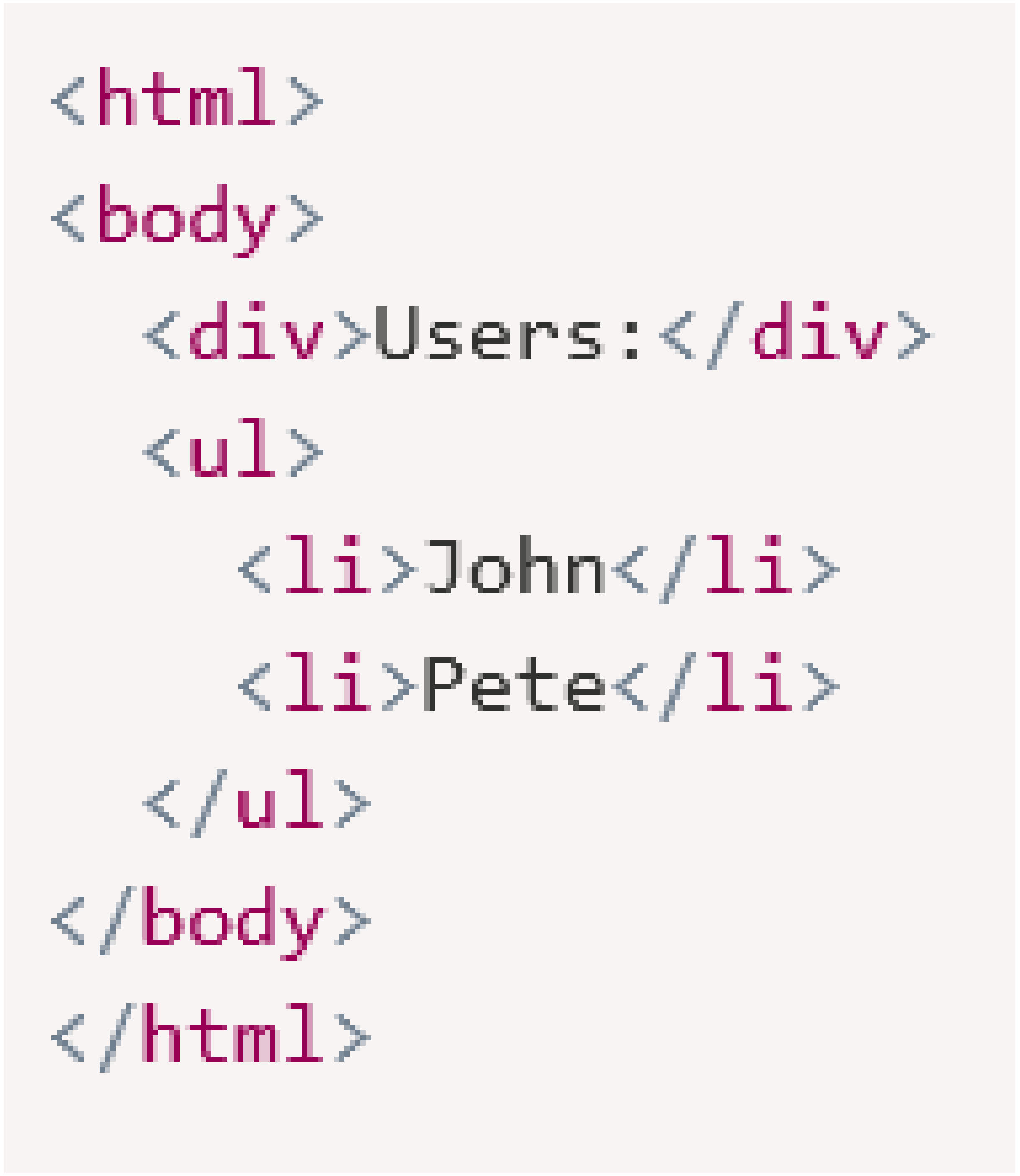
- children – only those children that are element nodes
- firstElementChild, lastElementChild – first and last element children
- previousElementSibling, nextElementSibling – neighbor elements
- parentElement – parent element



Quiz

For each of the following, give at least one way of how to access them:

1. The <div> DOM node?
2. The <ul> DOM node?
3. The second <li> (with Pete)?

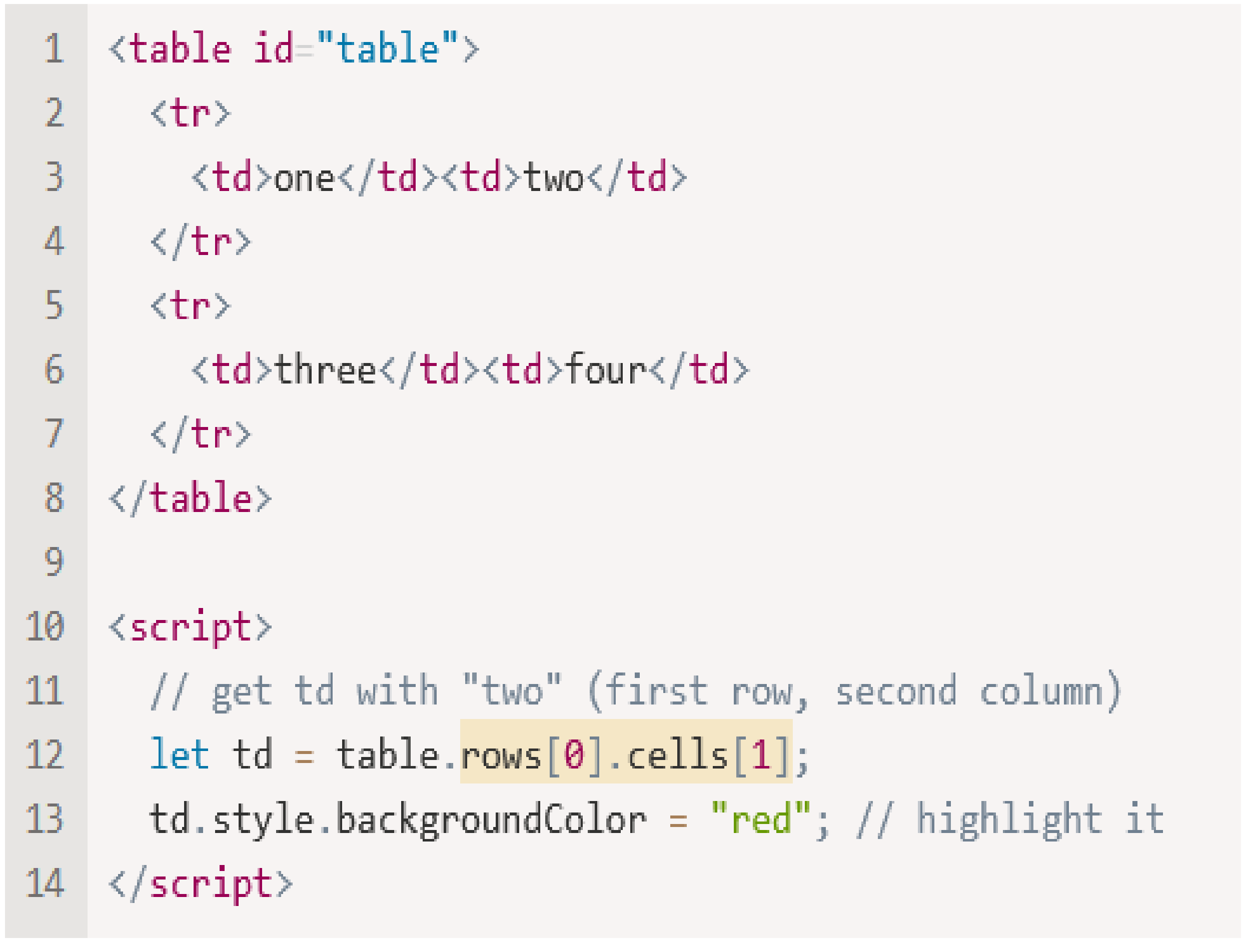


```
<html>
<body>
   <div>Users:</div>
   <ul>
      <li>John</li>
      <li>Pete</li>
   </ul>
</body>
</html>
```

Element specific properties

Certain types of DOM elements may provide additional properties, specific to their type.

Table element has many useful properties specific to table



```
<table id="table">
  <tr>
    <td>one</td><td>two</td>
  </tr>
  <tr>
    <td>three</td><td>four</td>
  </tr>
</table>

<script>
  // get td with "two" (first row, second column)
  let td = table.rows[0].cells[1];
  td.style.backgroundColor = "red"; // highlight it
</script>
```

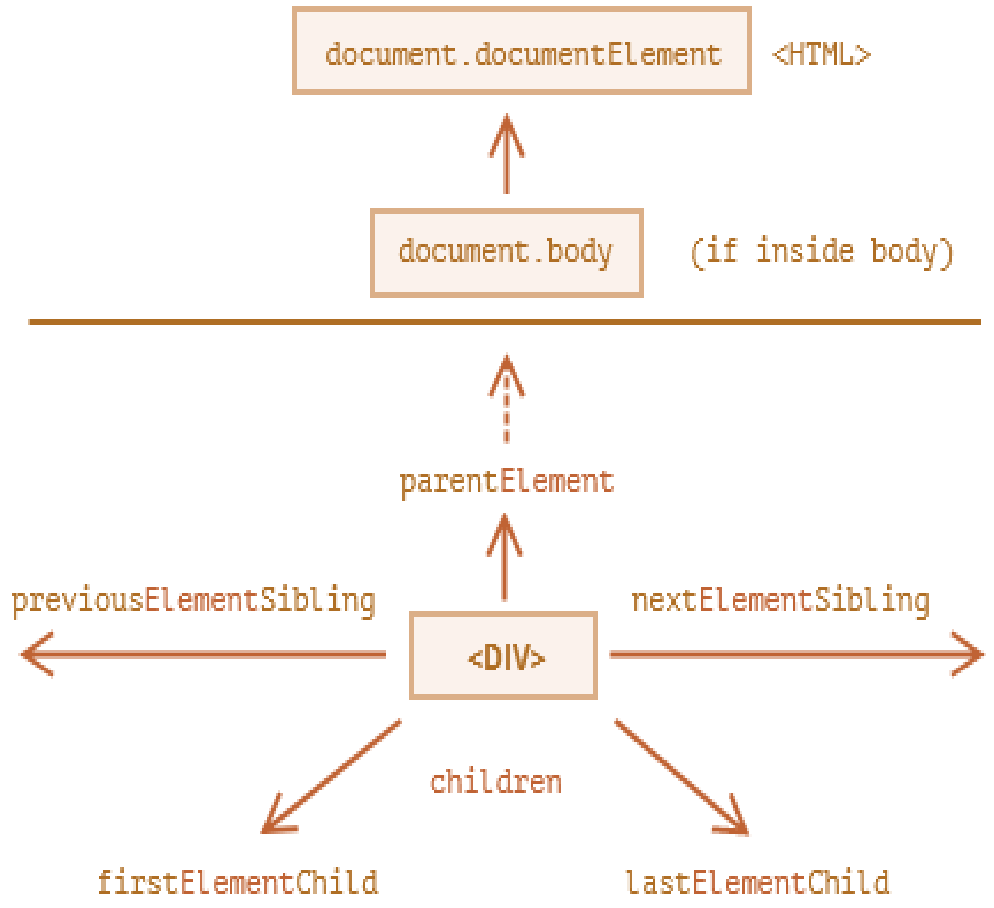
How to get an arbitrary element on a page?

Navigation properties can used when elements are close to each other

Need additional methods

Most used methods

- getElementById()
- querySelector()
- querySelectorAll()



document.getElementByID() or just id

If an an element has the id attribute, we can get the element by using

document.getElementById(id)

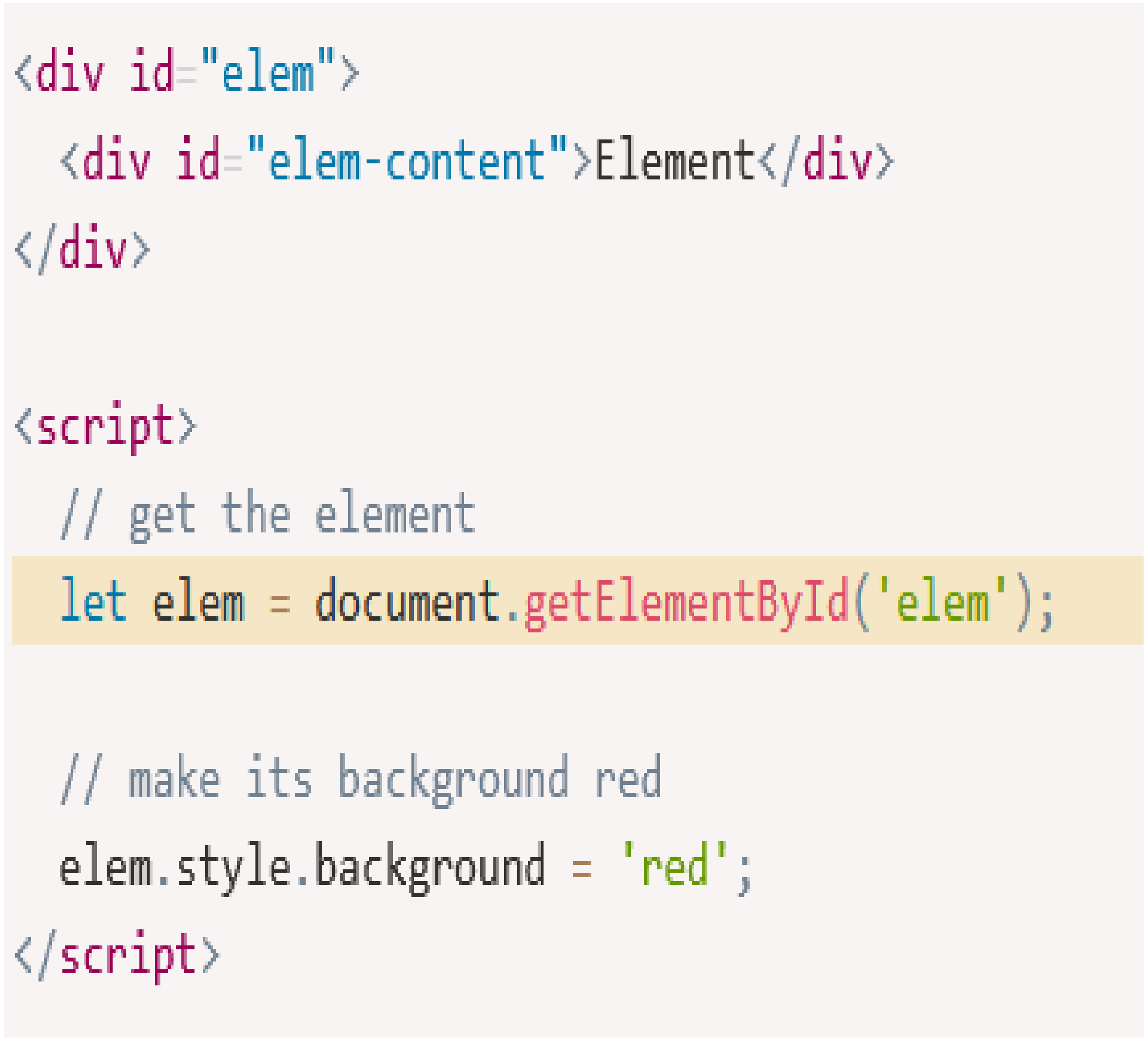
Named id is also a global variable – don't use

elem.style.background = 'red'

id must be unique

getElementByID() may return random element if id is not unique

Only document.getElementById() is valid



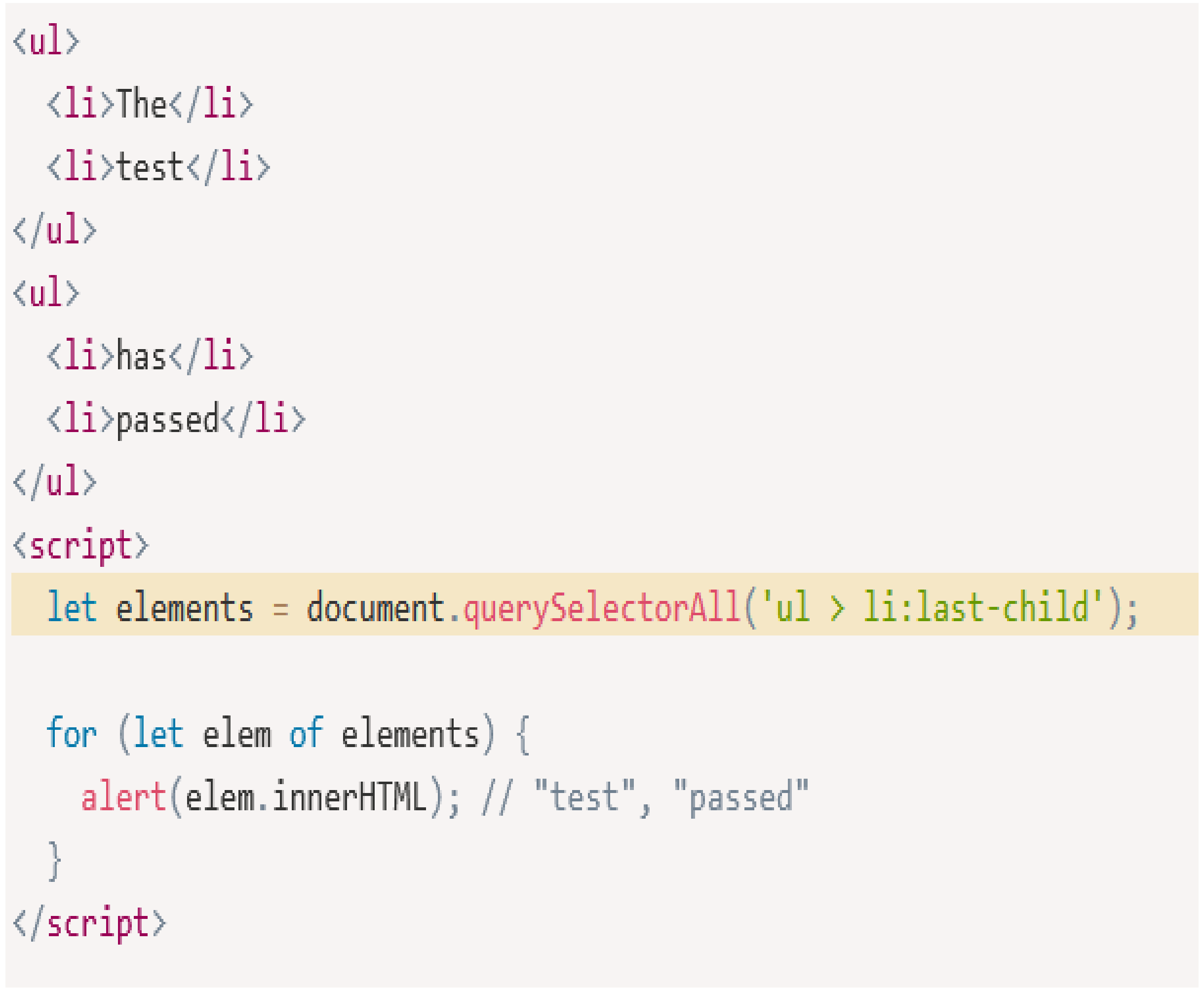
```
<div id="elem">
  <div id="elem-content">Element</div>
</div>

<script>
  // get the element
  let elem = document.getElementById('elem');

  // make its background red
  elem.style.background = 'red';
</script>
```

querySelectorAll() – most versatile method

elem.querySelectorAll(css) returns all elements inside elem matching the given CSS selector

Powerful method – any CSS selector can be used

Pseudo classes like ':hover' can used



```
<ul>
  <li>The</li>
  <li>test</li>
</ul>
<ul>
  <li>has</li>
  <li>passed</li>
</ul>
<script>
  let elements = document.querySelectorAll('ul > li:last-child');

  for (let elem of elements) {
    alert(elem.innerHTML); // "test", "passed"
  }
</script>
```

querySelector()

elem.querySelector(css) returns the first element inside elem matching the given CSS selector

same as elem.querySelectorAll(css)[0]

But querySelector() is faster

Preferred over getElementByID()

getElementsBy*

There are methods to get elements by a tag, class etc.

- elem.getElementsByTagName(tag) looks for elements with the given tag and returns the collection of them
- elem.getElementsByClassName(className) returns elements that have the given CSS class.
- document.getElementsByName(name) returns elements with the given name attribute, document-wide

querySelector is more powerful and shorter to write

DOM node classes

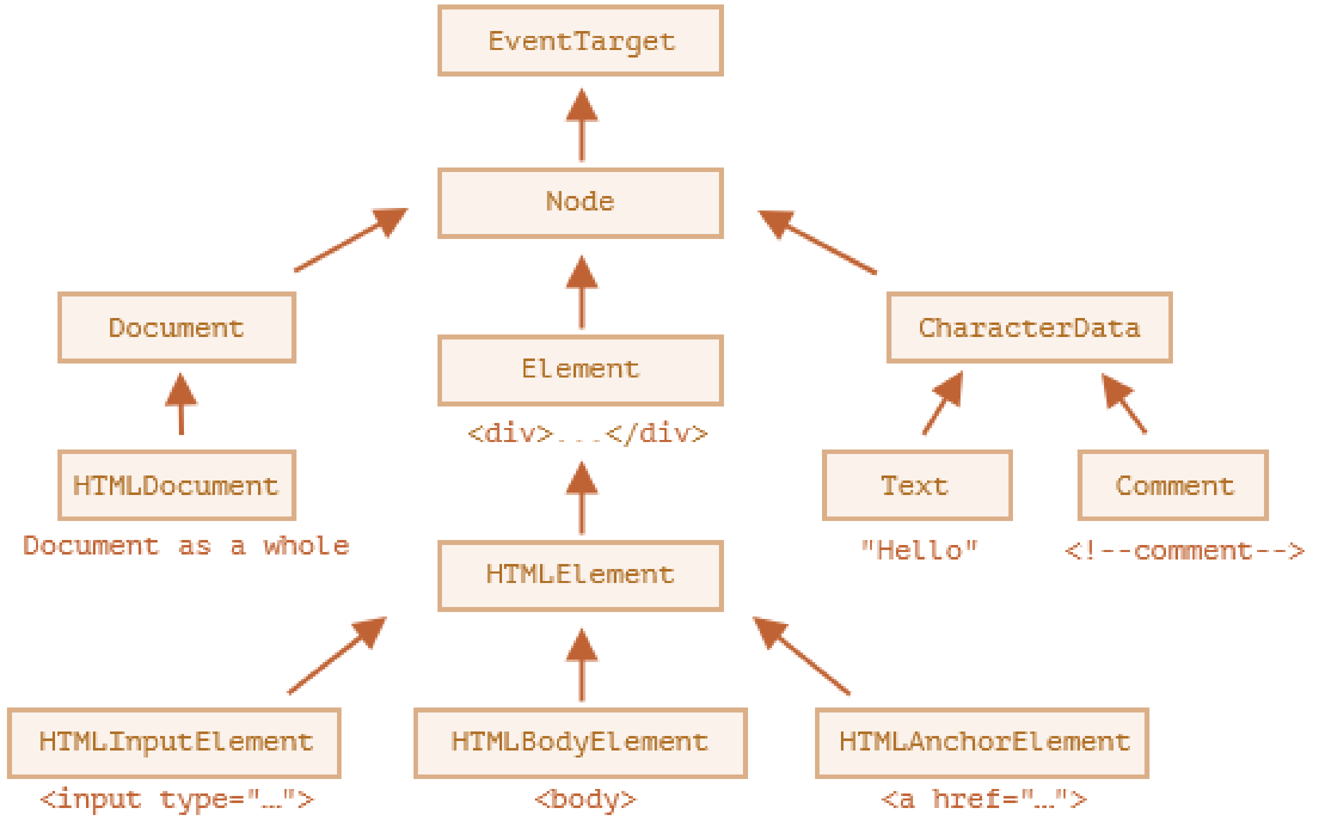
Different DOM nodes have different properties

Each DOM node corresponds to its built-in class

DOM nodes also share properties

All classes form a hierarchy

DOM nodes are regular javaScript objects



Node property: nodeType

Provides the type of the node object

Numeric value

element : 1

text: 3

document: 9

Read-only

```
<body>
  <script>
  let elem = document.body;

  // let's examine: what type of node is in elem?
  alert(elem.nodeType); // 1 => element

  // and its first child is...
  alert(elem.firstChild.nodeType); // 3 => text

  // for the document object, the type is 9
  alert( document.nodeType ); // 9
  </script>
</body>
```

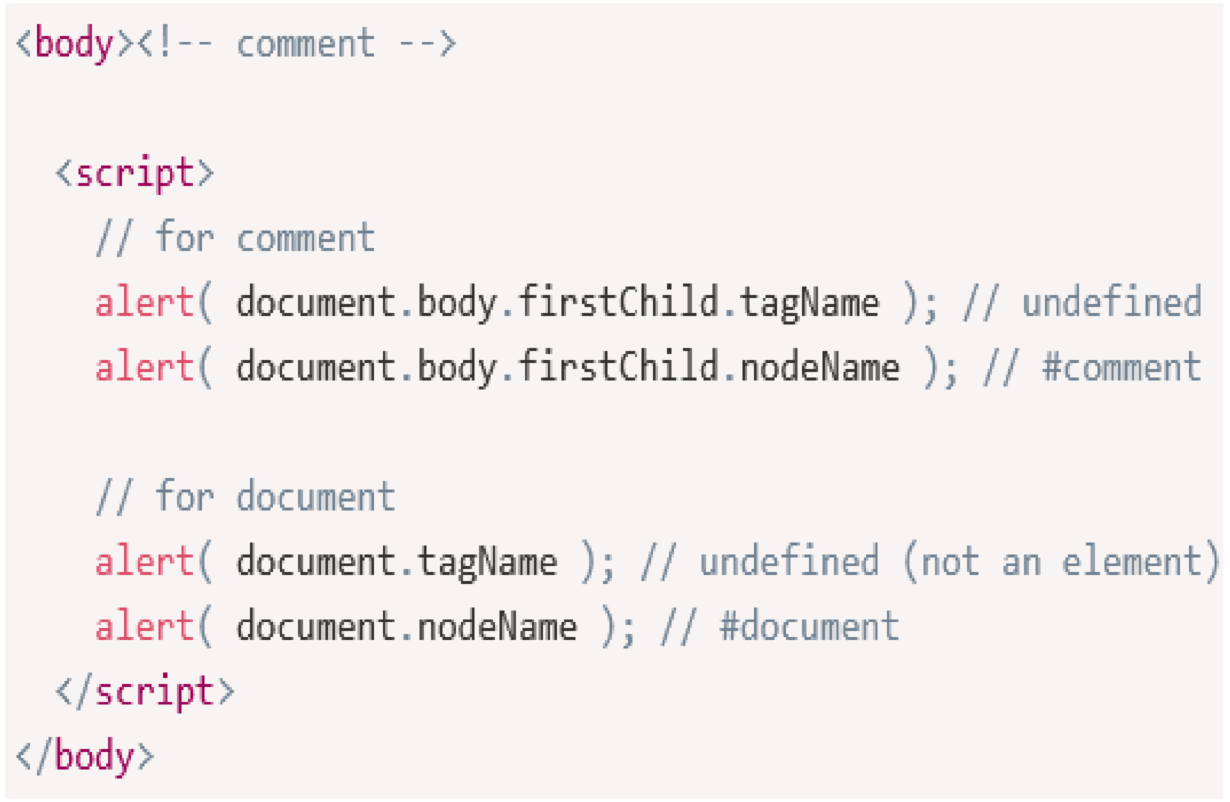
Node properties: nodeName and tagName

Provides tag name of a DOM node

tagName property exists only for Element nodes

nodeName is defined for any Node:

- for elements same as tagName
- for other node types (text, comment, etc.) it has a string with the node type.



```
<body><!-- comment -->

  <script>
    // for comment
    alert( document.body.firstChild.tagName ); // undefined
    alert( document.body.firstChild.nodeName ); // #comment

    // for document
    alert( document.tagName ); // undefined (not an element)
    alert( document.nodeName ); // #document
  </script>
</body>
```

Node property: innerHTML

Provides the HTML inside the element as a string

Can modify. Dangerous!

Beware: “innerHTML+= …” does a full overwrite

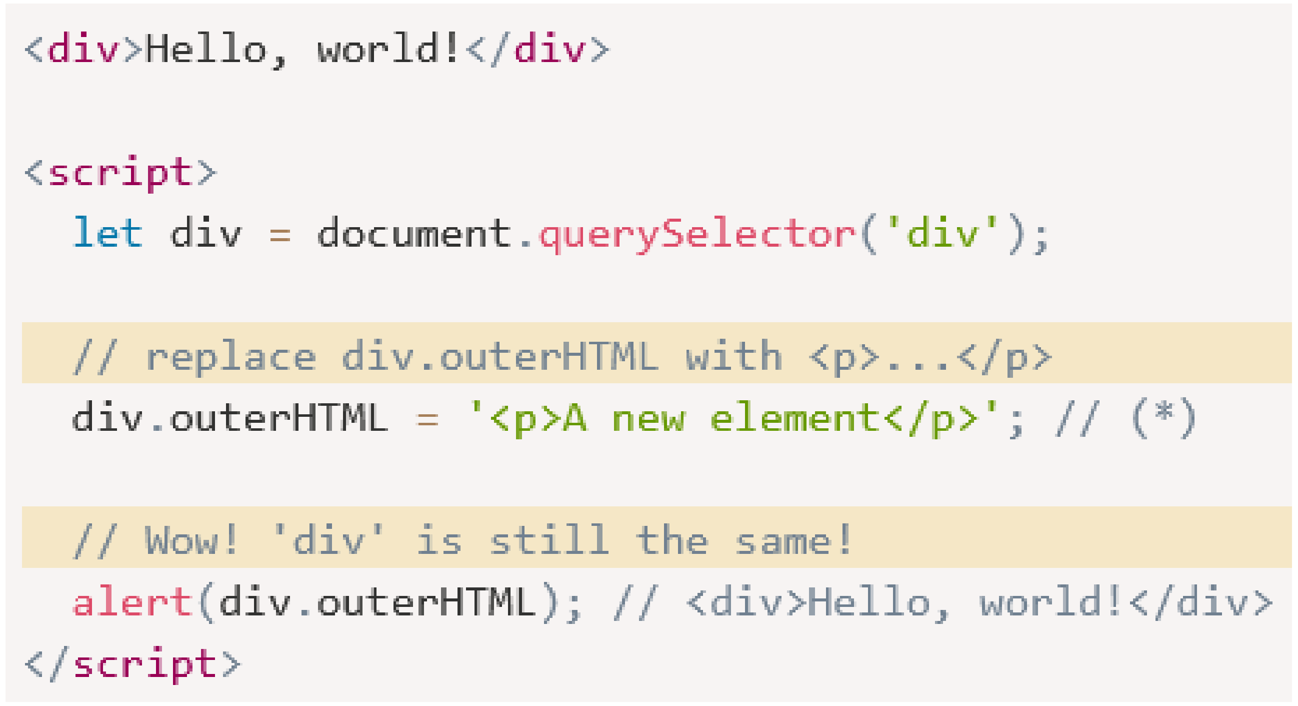
All images and other resources will be reloaded

Node property: outerHTML

Provides the full HTML of the element as a string

Can modify

Beware: unlike innerHTML, writing to outerHTML does not change the element. Instead, it replaces it in the DOM.

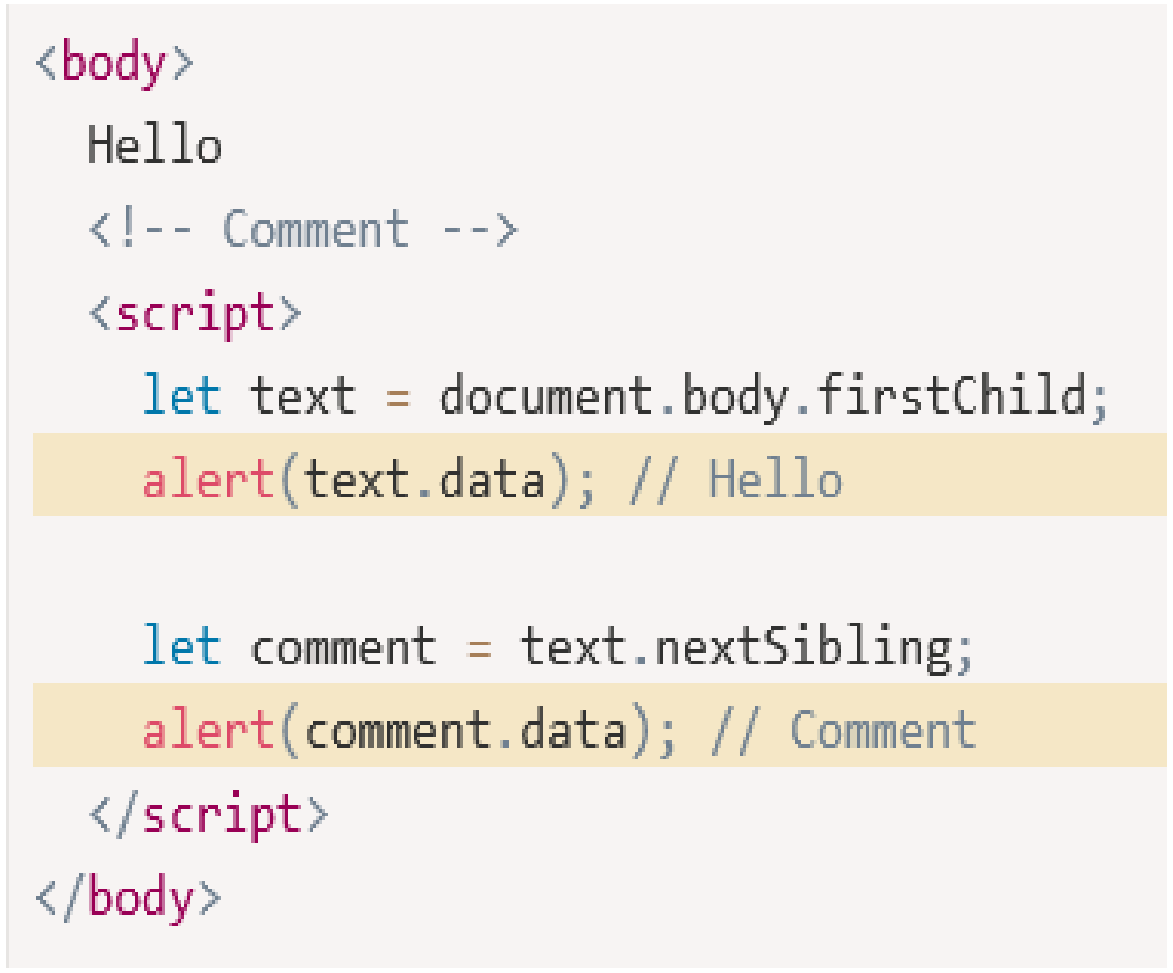


```
<div>Hello, world!</div>

<script>
  let div = document.querySelector('div');

  // replace div.outerHTML with <p>...</p>
  div.outerHTML = '<p>A new element</p>'; // (*)

  // Wow! 'div' is still the same!
  alert(div.outerHTML); // <div>Hello, world!</div>
</script>
```

Node property: nodeValue/data

Provides the content of non-element nodes

nodeValue and data are almost same



```
<body>
  Hello
  <!-- Comment -->
  <script>
    let text = document.body.firstChild;
    alert(text.data); // Hello

    let comment = text.nextSibling;
    alert(comment.data); // Comment
  </script>
</body>
```

Node property: textContent

Provides the pure text content of element nodes

Only text, no tags

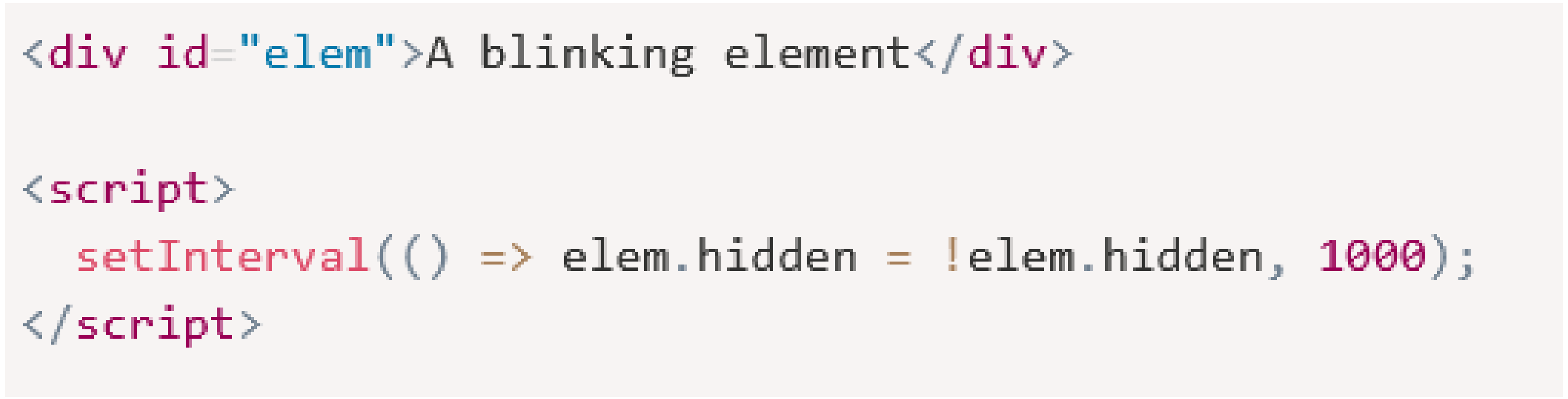
Safe way to write text

```
<div id="elem1"></div>
<div id="elem2"></div>

<script>
  let name = prompt("What's your name?", "<b>Winnie-the-Pooh!</b>");

  elem1.innerHTML = name;
  elem2.textContent = name;
</script>
```

Node property: hidden

Specifies whether the element is visible or not

How to make an element blink?

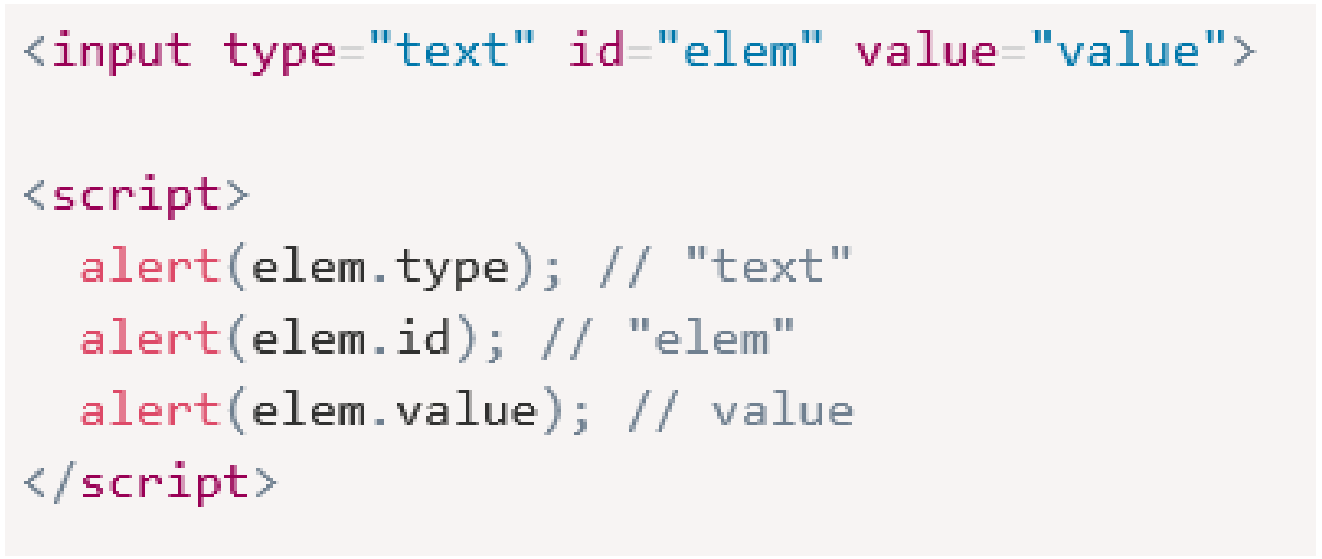


```
<div id="elem">A blinking element</div>

<script>
  setInterval(() => elem.hidden = !elem.hidden, 1000);
</script>
```

More node properties

DOM elements have additional properties, particularly those that depend on the class:

- value – the value for <input>, <select> and <textarea> (HTMLInputElement, HTMLSelectElement…).
- href – the “href” for <a href="..."> (HTMLAnchorElement).
- id – the value of “id” attribute, for all elements (HTMLElement).
- …and much more…



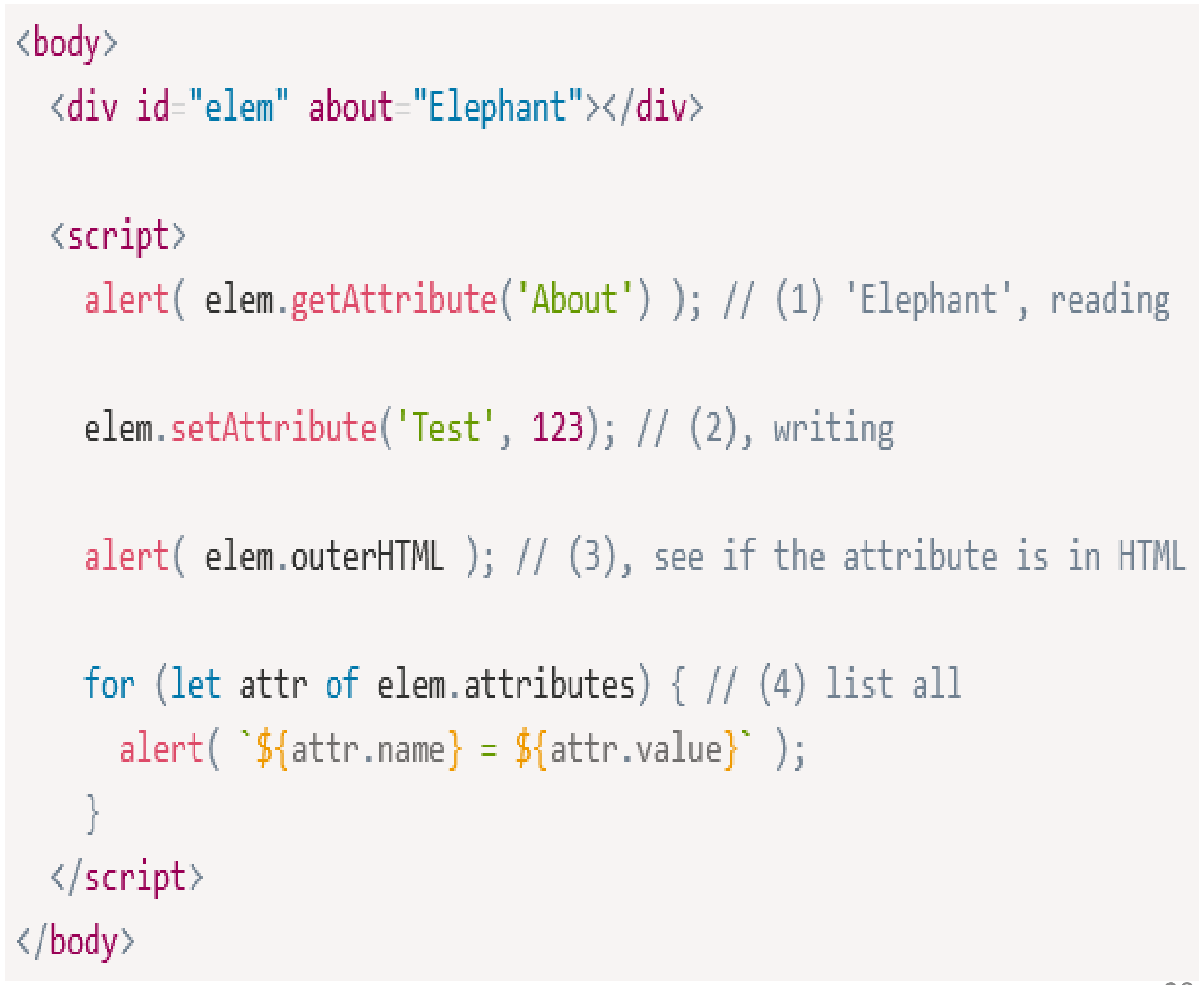
```
<input type="text" id="elem" value="value">

<script>
  alert(elem.type); // "text"
  alert(elem.id); // "elem"
  alert(elem.value); // value
</script>
```

Most standard HTML attributes have the corresponding DOM property,

Attributes methods

All attributes (even non-standard) are accessible by using the following methods:

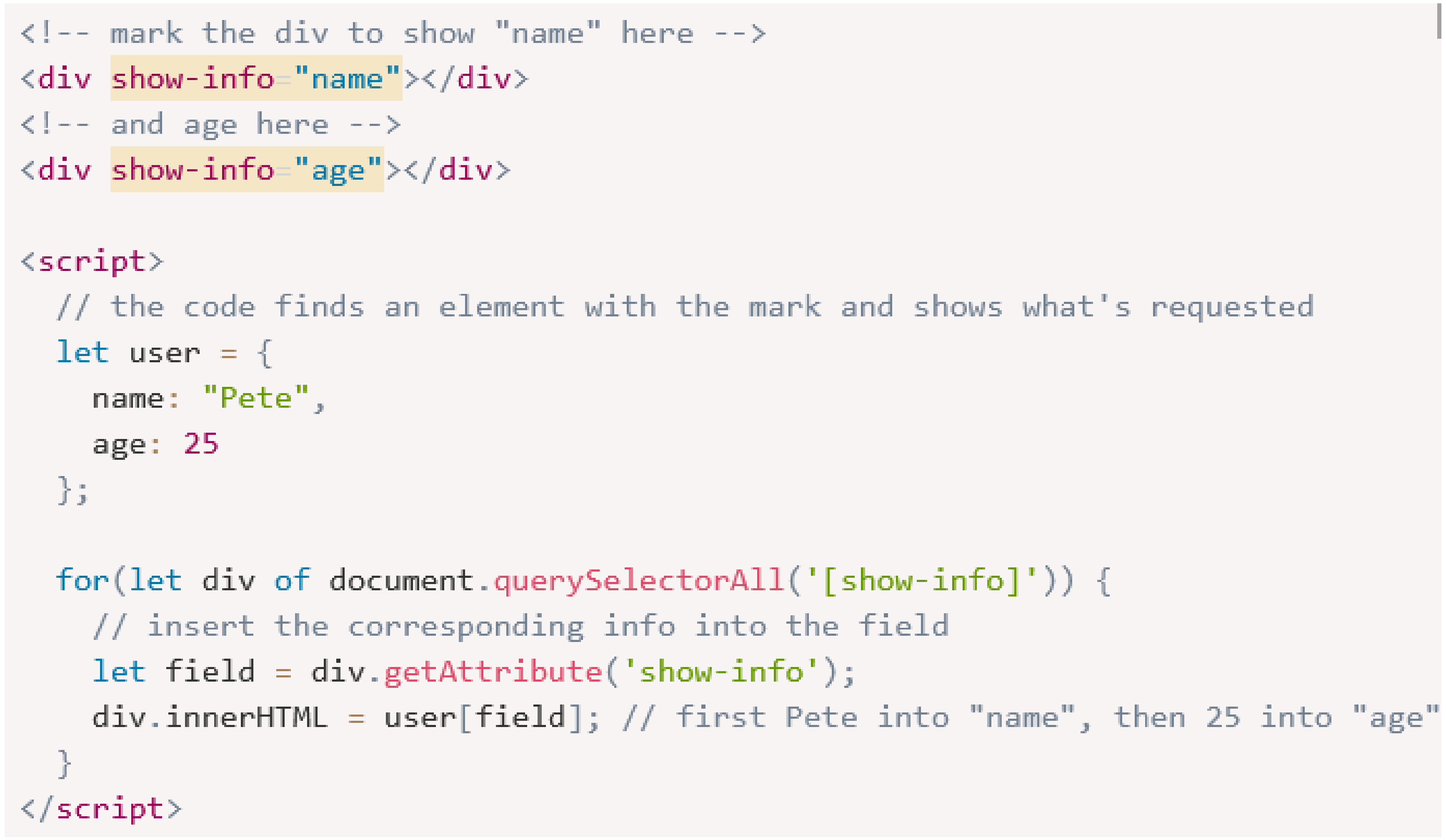
- elem.hasAttribute(name) – checks for existence
- elem.getAttribute(name) – gets the value
- elem.setAttribute(name, value) – sets the value
- elem.removeAttribute(name) – removes the attribute



```
<body>
  <div id="elem" about="Elephant"></div>

  <script>
    alert( elem.getAttribute('About') ); // (1) 'Elephant', reading

    elem.setAttribute('Test', 123); // (2), writing

    alert( elem.outerHTML ); // (3), see if the attribute is in HTML

    for (let attr of elem.attributes) { // (4) list all
      alert( `${attr.name} = ${attr.value}` );
    }
  </script>
</body>
```

Non-standard attributes

Used to pass data from HTML to JavaScript, or to “mark” HTML-elements for JavaScript



```
<!-- mark the div to show "name" here -->
<div show-info="name"></div>
<!-- and age here -->
<div show-info="age"></div>

<script>
  // the code finds an element with the mark and shows what's requested
  let user = {
    name: "Pete",
    age: 25
  };

  for(let div of document.querySelectorAll('[show-info]')) {
    // insert the corresponding info into the field
    let field = div.getAttribute('show-info');
    div.innerHTML = user[field]; // first Pete into "name", then 25 into "age"
  }
</script>
```

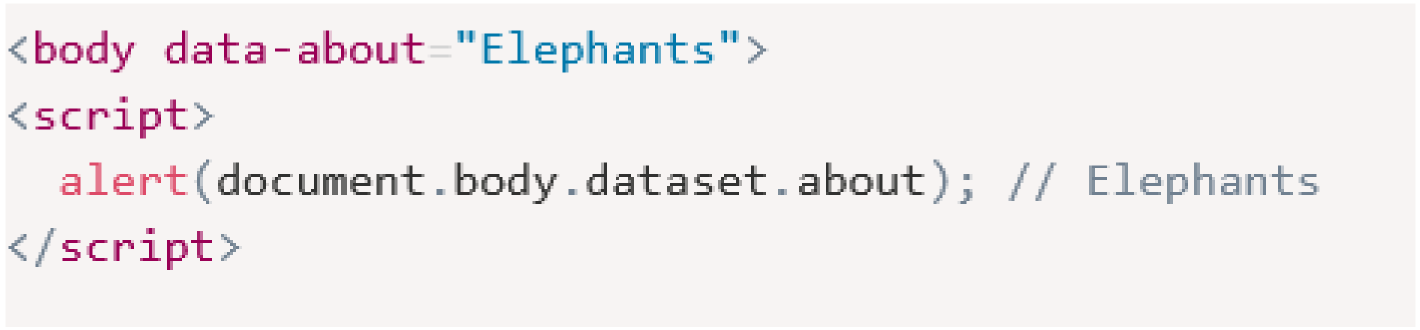
dataset

What happens if non-standard attribute becomes a standard attribute?

All attributes starting with “data-” are reserved for programmers’ use

They are available in the dataset property

Multiword attributes like data-about-us become camel-cased: dataset.aboutUs



```
<body data-about="Elephants">
<script>
  alert(document.body.dataset.about); // Elephants
</script>
```

Creating an element

document.create(tag) - Creates a new element node with the given tag

```html
<div class="alert">
  <strong>Hi there!</strong> You've read an important message.
</div>
```

```javascript
// 1. Create <div> element
let div = document.createElement('div');

// 2. Set its class to "alert"
div.className = "alert";

// 3. Fill it with the content
div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";
```

Not yet in the page

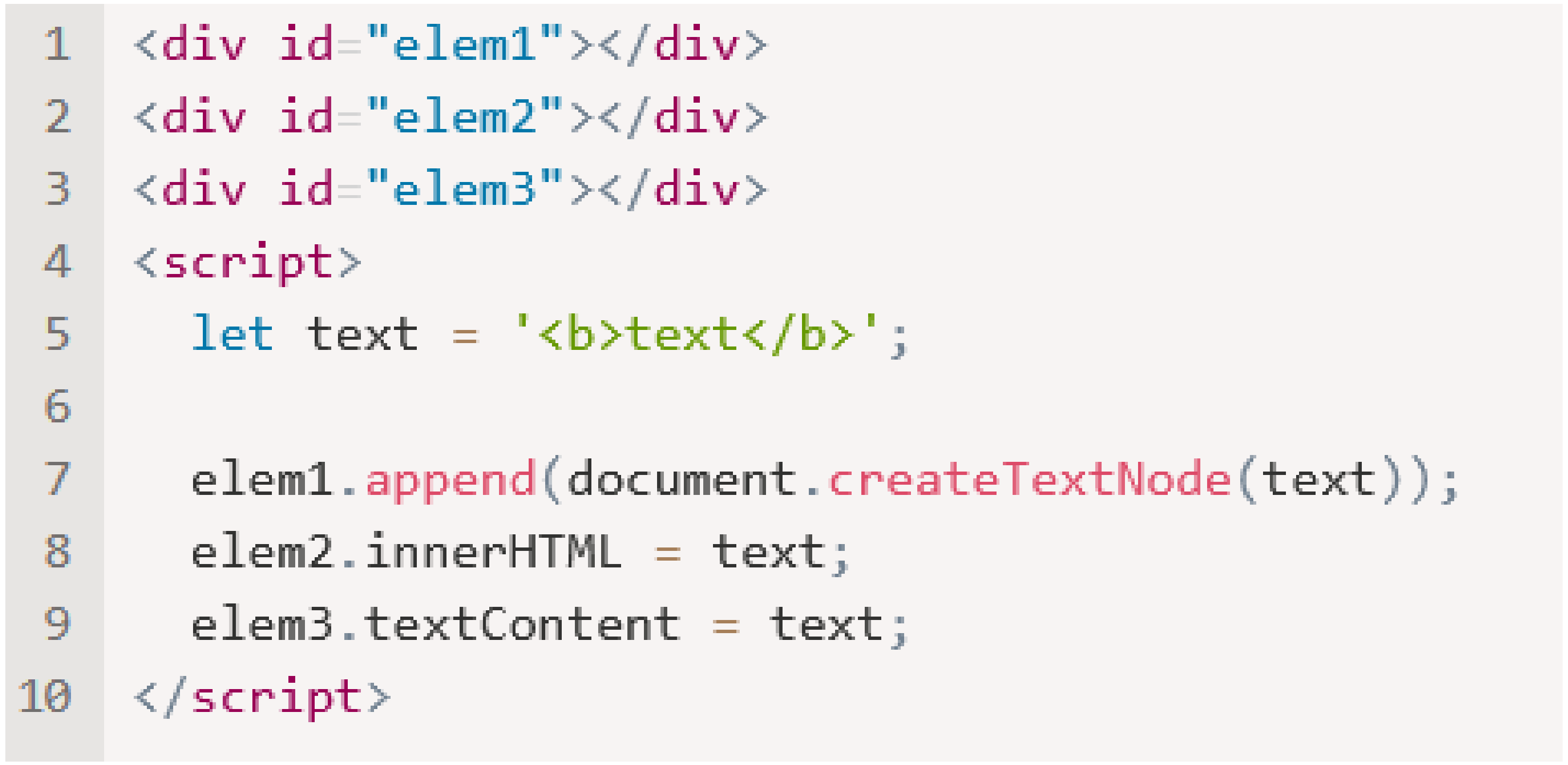
Inserting an element

elem.append(node) – inserts node at the end of elem

```
<script>
  let div = document.createElement('div');
  div.className = "alert";
  div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";

  document.body.append(div);
</script>
```

Quiz - createTextNode vs innerHTML vs textContent

Which of these 3 commands (lines 7, 8, and 9) will do the same?



```
1  <div id="elem1"></div>
2  <div id="elem2"></div>
3  <div id="elem3"></div>
4  <script>
5    let text = '<b>text</b>';
6
7    elem1.append(document.createTextNode(text));
8    elem2.innerHTML = text;
9    elem3.textContent = text;
10 </script>
```

Insertion methods

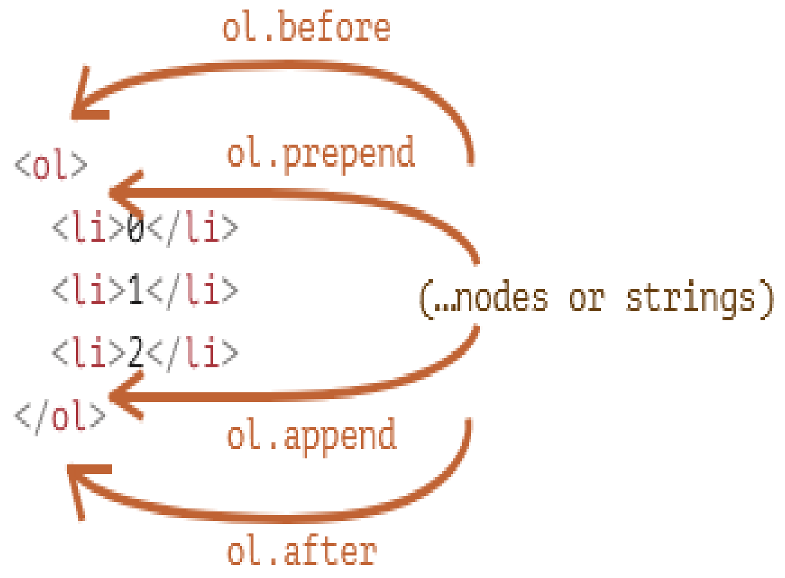
node.append(...nodes or strings) – append nodes or strings at the end of node,

node.prepend(...nodes or strings) – insert nodes or strings at the beginning of node,

node.before(...nodes or strings) – insert nodes or strings before node,

node.after(...nodes or strings) – insert nodes or strings after node,

node.replaceWith(...nodes or strings) – replaces node with the given nodes or strings



Insertion methods usage

```
<ol id="ol">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>

<script>
  ol.before('before'); // insert string "before" before <ol>
  ol.after('after'); // insert string "after" after <ol>

  let liFirst = document.createElement('li');
  liFirst.innerHTML = 'prepend';
  ol.prepend(liFirst); // insert liFirst at the beginning of <ol>

  let liLast = document.createElement('li');
  liLast.innerHTML = 'append';
  ol.append(liLast); // insert liLast at the end of <ol>
</script>
```

```
before
<ol id="ol">
  <li>prepend</li>
  <li>0</li>
  <li>1</li>
  <li>2</li>
  <li>append</li>
</ol>
after
```

insertAdjacentHTML/Text/Element

Inserts an HTML string “as html”, in the same manner as elem.innerHTML

elem.insertAdjacentHTML(where, html):

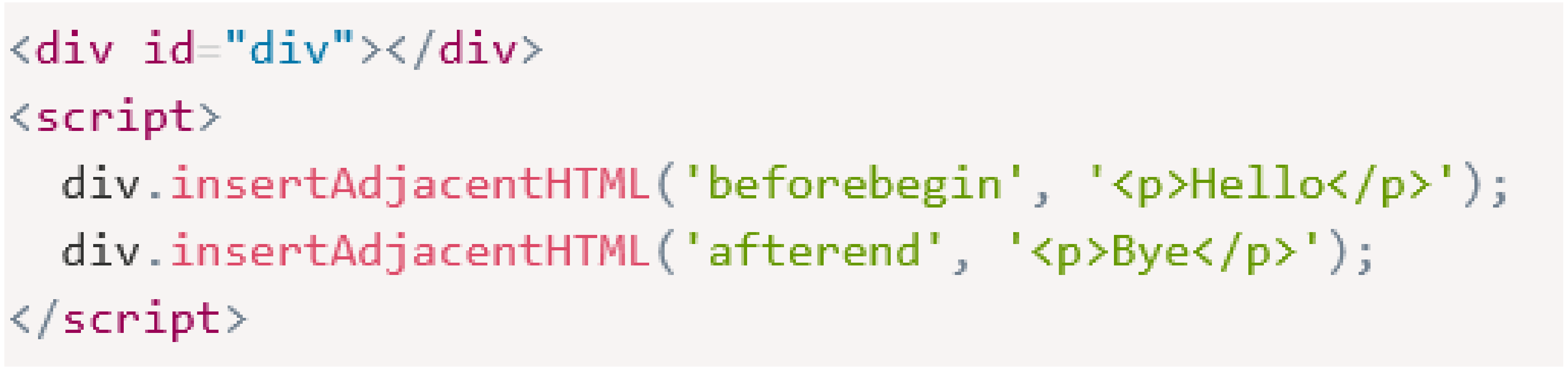
The first parameter is a code word, specifying where to insert relative to elem

Must be one of the following:

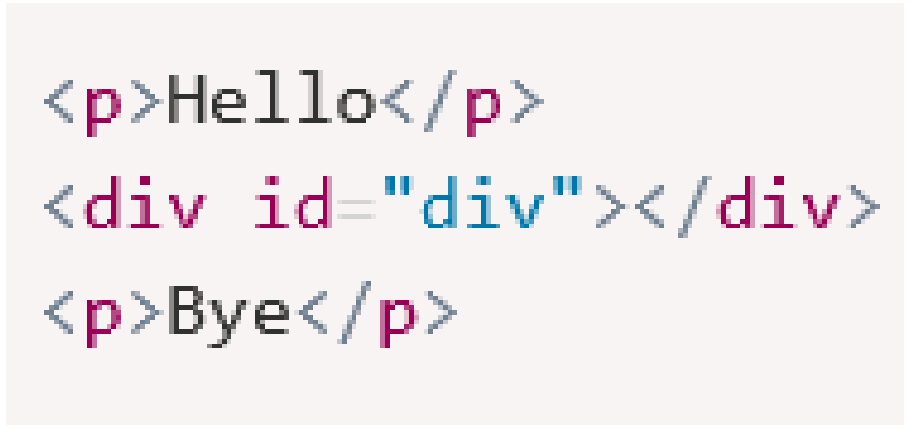
- "beforebegin" – insert html immediately before elem,
- "afterbegin" – insert html into elem, at the beginning,
- "beforeend" – insert html into elem, at the end,
- "afterend" – insert html immediately after elem.

The second parameter is an HTML string, that is inserted “as HTML”.

insertAdjacentHTML/Text/Element



```
<div id="div"></div>
<script>
  div.insertAdjacentHTML('beforebegin', '<p>Hello</p>');
  div.insertAdjacentHTML('afterend', '<p>Bye</p>');
</script>
```



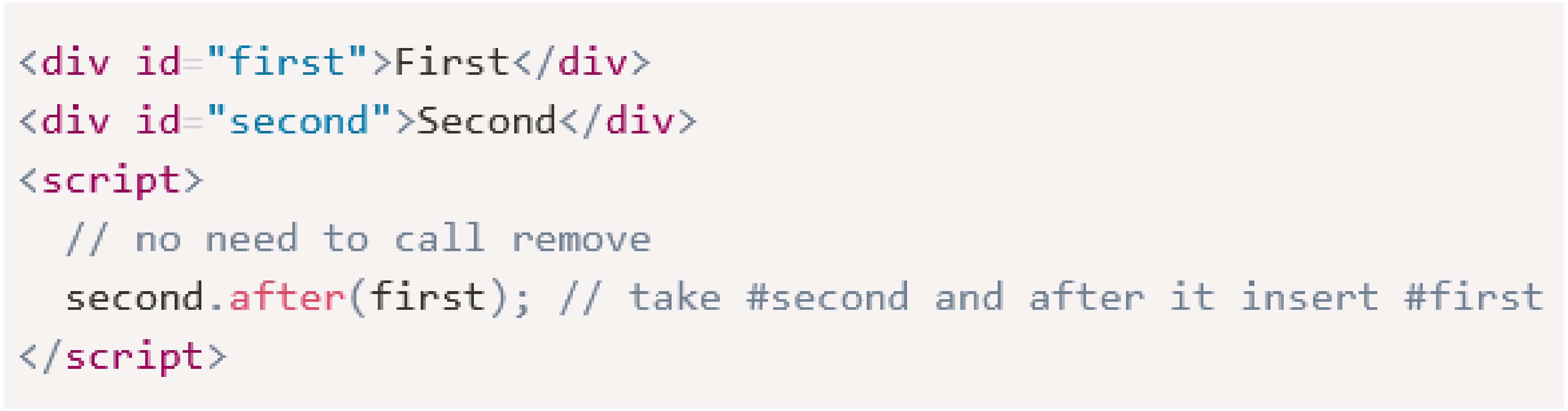
```
<p>Hello</p>
<div id="div"></div>
<p>Bye</p>
```

Remove a node - node.remove()

```
<script>
  let div = document.createElement('div');
  div.className = "alert";
  div.innerHTML = "<strong>Hi there!</strong> You've read an important message.";

  document.body.append(div);
  setTimeout(() => div.remove(), 1000);
</script>
```

Moving a node

All insertion methods automatically remove the node from the old place



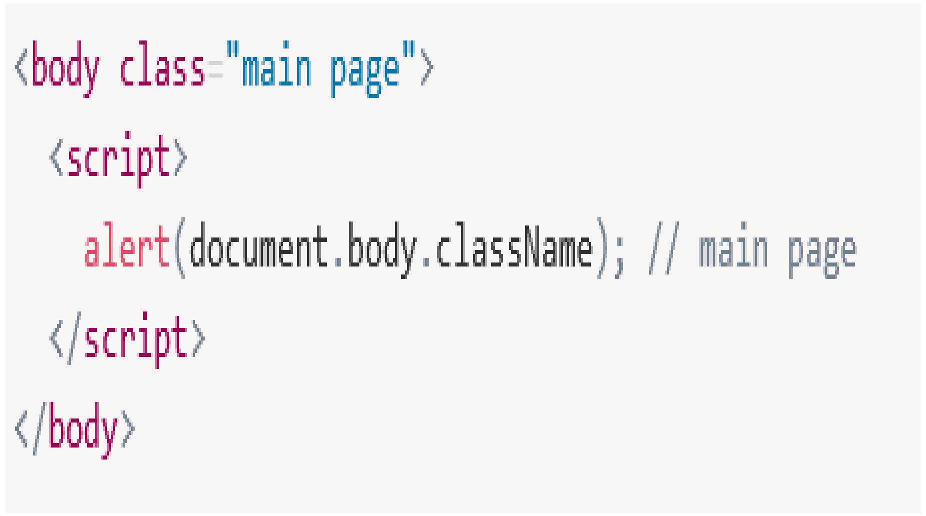
```
<div id="first">First</div>
<div id="second">Second</div>
<script>
  // no need to call remove
  second.after(first); // take #second and after it insert #first
</script>
```

Quiz

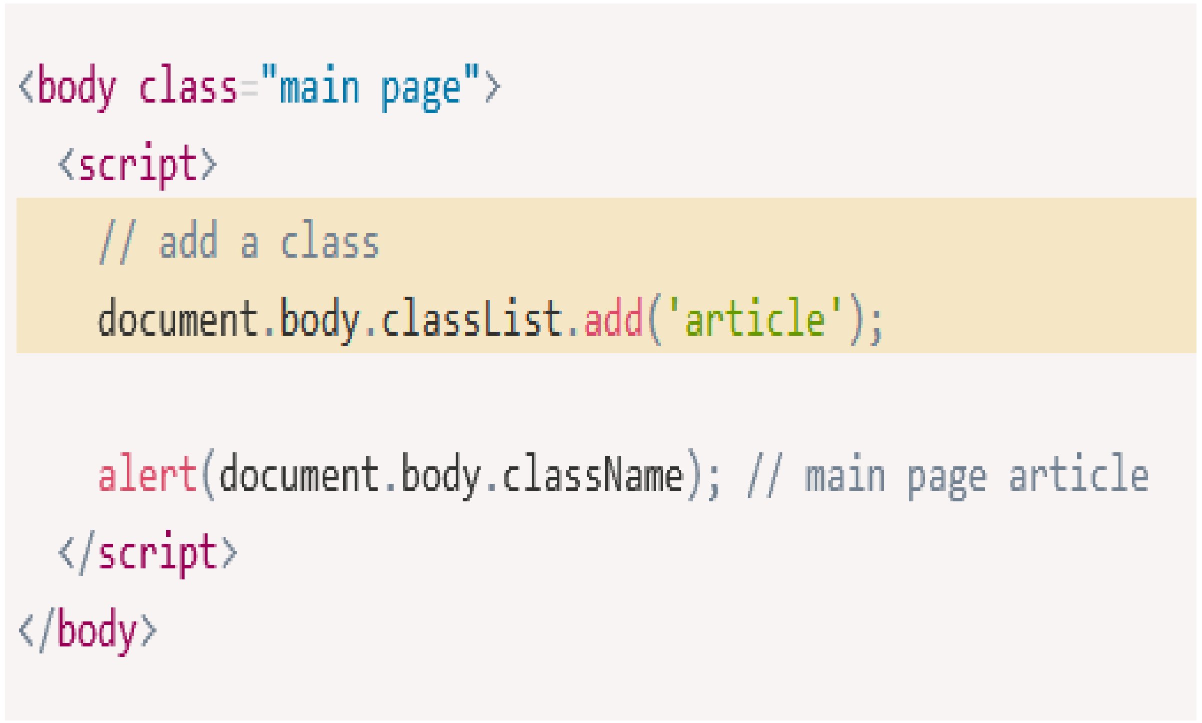
?

className and classList

className – the string value, good to manage the whole set of classes
classList – the object with methods add/remove/toggle/contains, good for individual classes



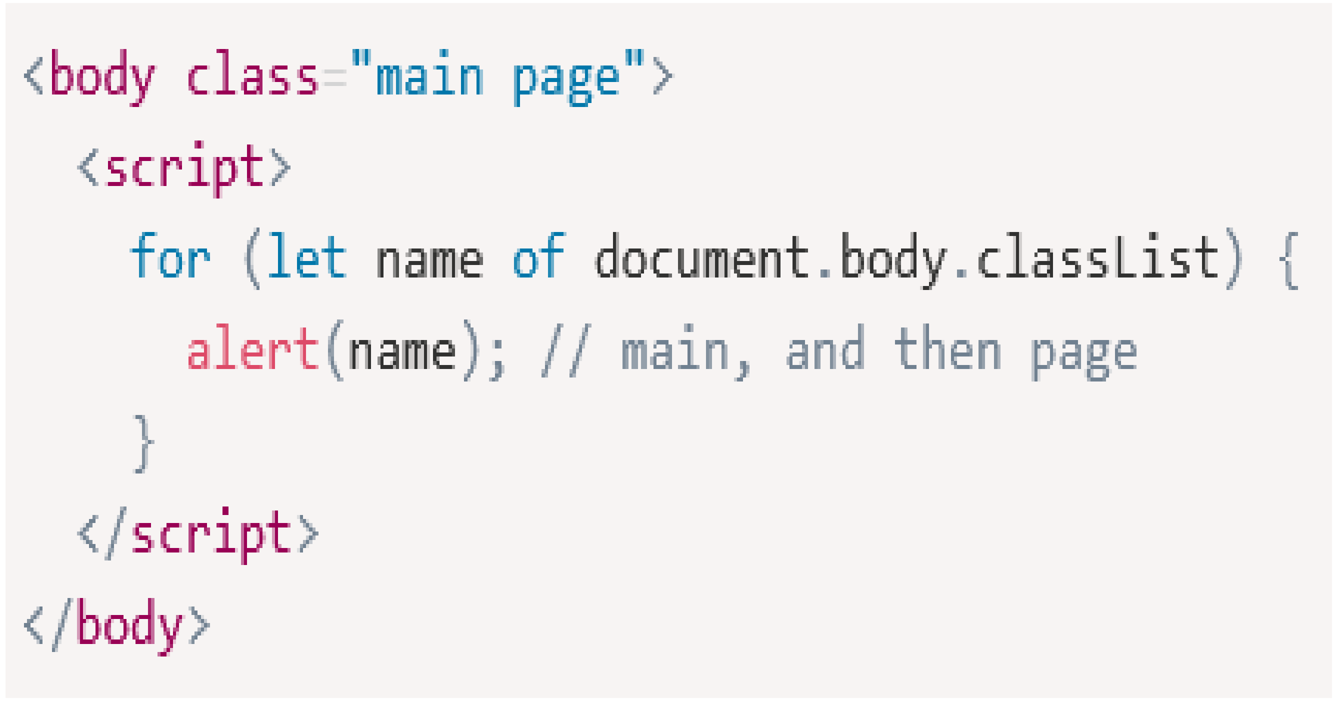
```
<body class="main page">
  <script>
    alert(document.body.className); // main page
  </script>
</body>
```



```
<body class="main page">
  <script>
    // add a class
    document.body.classList.add('article');

    alert(document.body.className); // main page article
  </script>
</body>
```

className and classList

classList is iterable



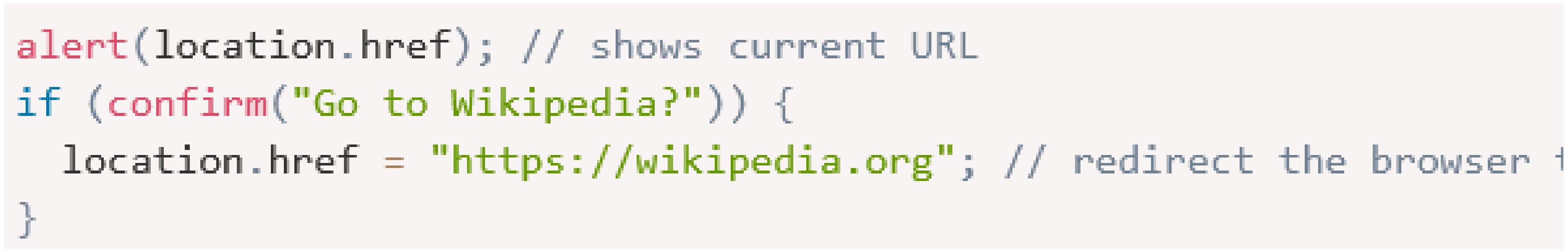
```
<body class="main page">
  <script>
    for (let name of document.body.classList) {
      alert(name); // main, and then page
    }
  </script>
</body>
```

Browser Object Model (BOM)

BOM represents object models provided by the browser

'navigator' object provides info about browser and OS

'location' object provides info about URL



```
alert(location.href); // shows current URL
if (confirm("Go to Wikipedia?")) {
  location.href = "https://wikipedia.org"; // redirect the browser
}
```

Sources

1. https://javascript.info/
2. MDN - Manipulating documents