Advanced Web Programming

Event Programming

JS with the Document Object Model (DOM) and UI Events

Grzegorz Rogus
rogus@agh.edu.pl

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
AGH University of Science and Technology

Assumptions and prerequisites

- I assume (I’m sure) you know the basics of JS (variables, types, instructions, functions and you know the rules of DOM manipulation).
- The lecture refers to these basics and presents modern ways of using JS as a means of delivering responsiveness to a web application.
- Due to time constraints, I am not discussing now the issues that extend the possibilities of the language, known as ES6 and later versions (maybe letter in Vue section)

Agenda

- JS – reminder
- DOM
- DOM manipulacje (selectors i pseudo class)
- Event
- Dynamic change of DOM model

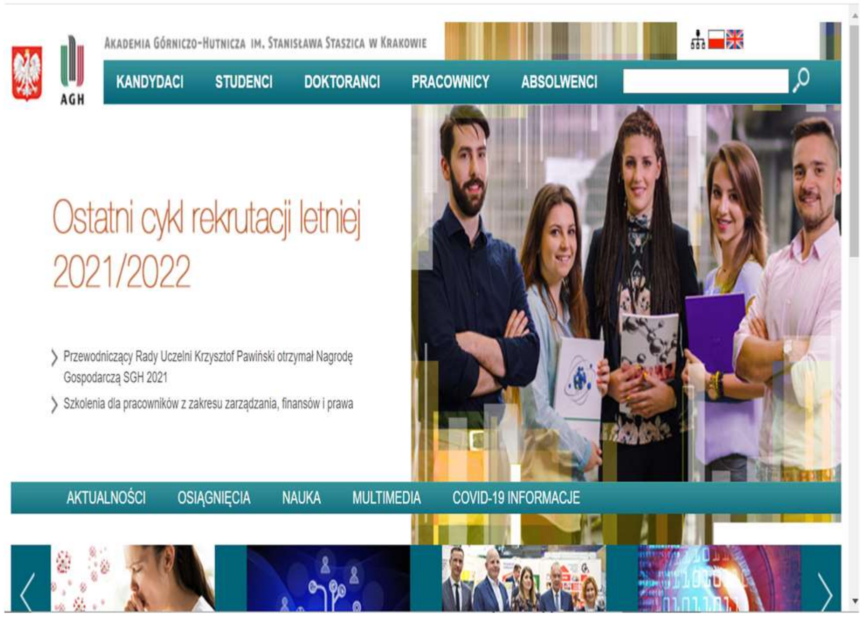
- Object + Collection
- JSON
- Read file – FetchAPI

Interactive Webpages with JS

After lab1 we know HTML5 & CSS3 but

+

produces



Ostatni cykl rekrutacji letniej 2021/2022

- Przewodniczący Rady Uczelni Krzysztof Pawiński otrzymał Nagrodę Gospodarczą SGH 2021
- Szkolenia dla pracowników z zakresu zarządzania, finansów i prawa

Describes the content and structure of the page

Describes the appearance and style of the page

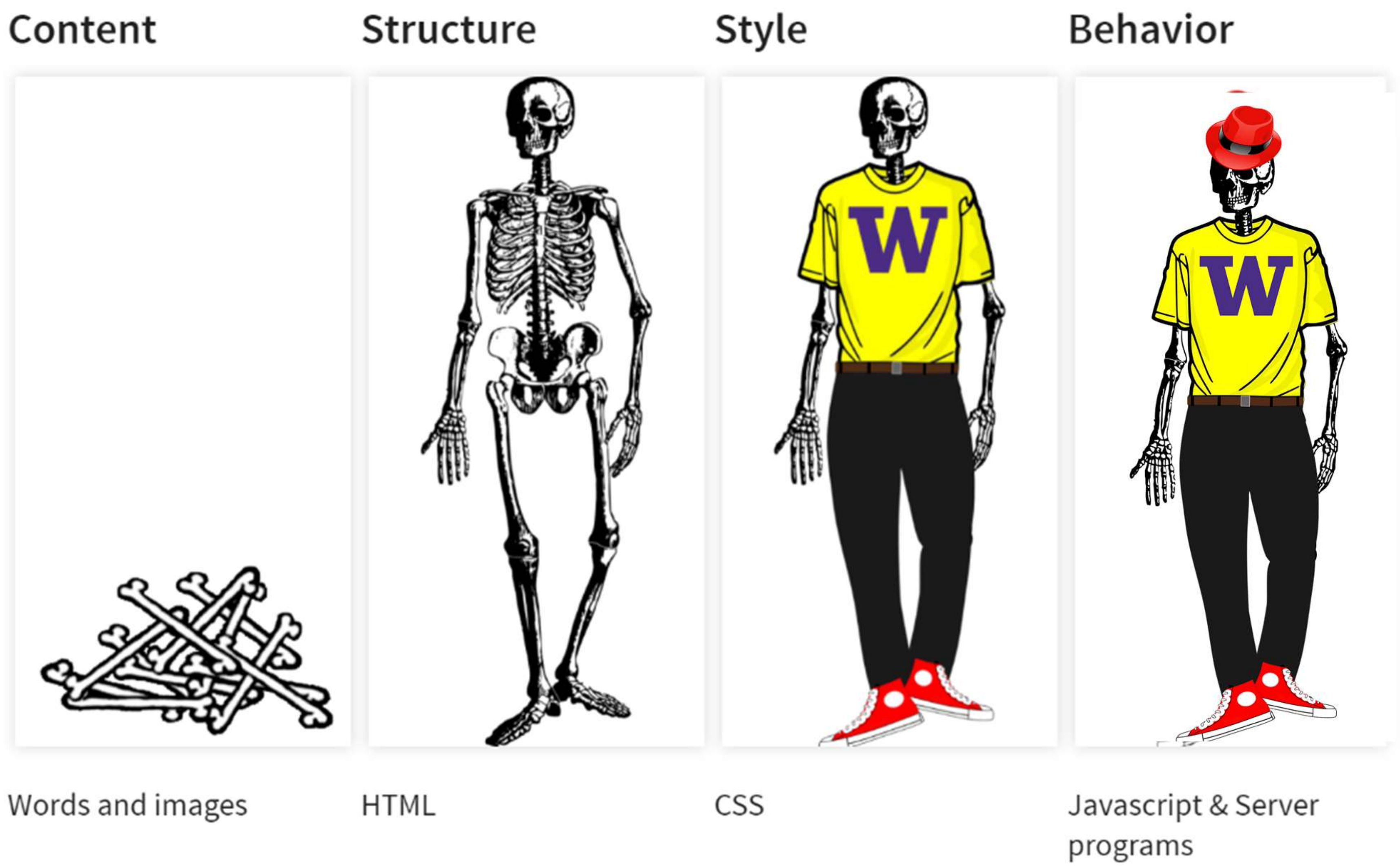
A web page…
that looks the same every time
(Static web pages)

Now that we know how to add content and styles to a web page, lets explore how to add responsive behavior

Enter JavaScript!

front-end development triad

Web Design



Web Application

JS reminder - What is JavaScript?

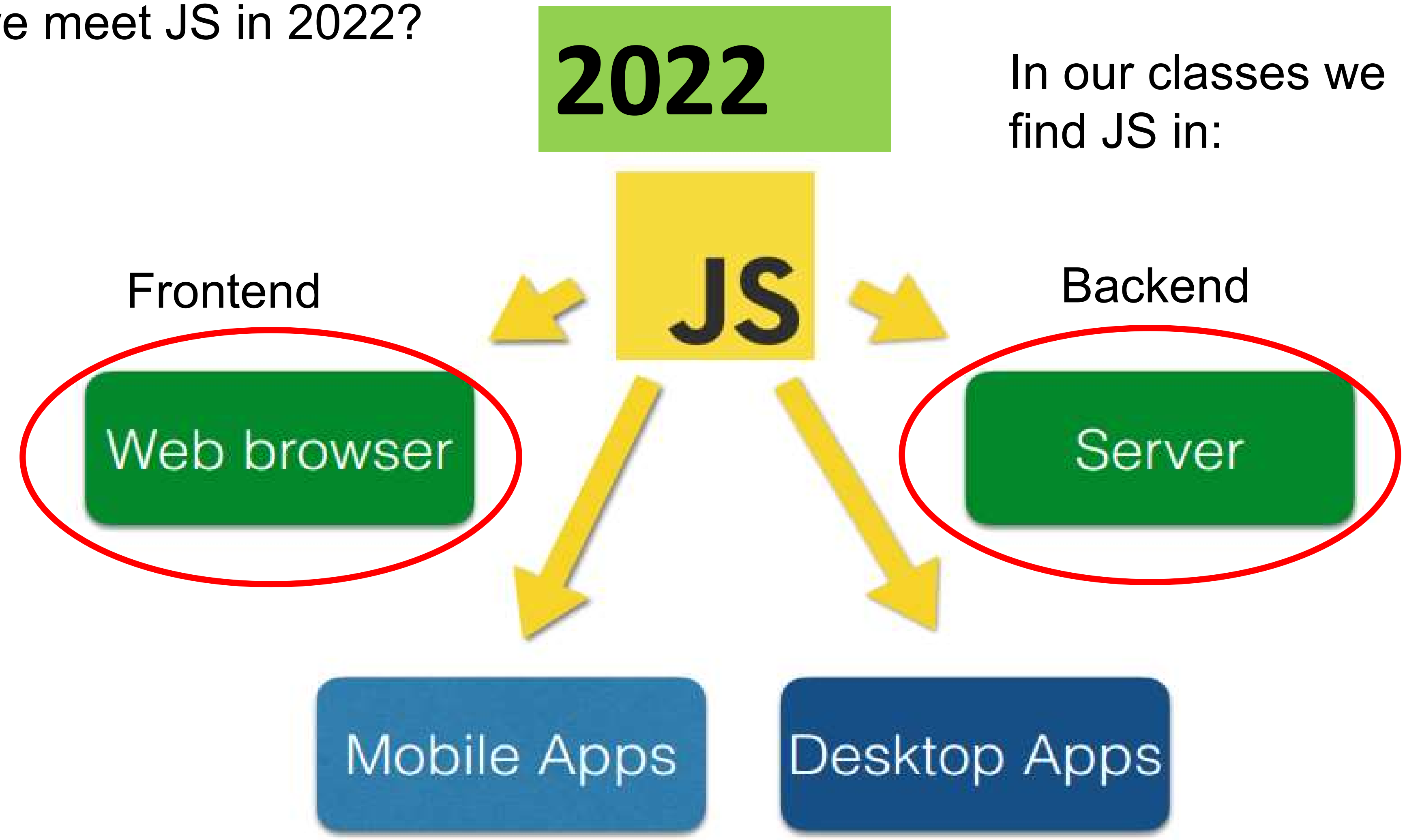
- A lightweight "scripting" programming language
- Used to make web pages interactive and dynamic:
    - Respond to user interactions with the website
    - Insert dynamic text into HTML (ex: username)
    - React to events (ex: page load, user's mouse click)
    - Get information about a user's computer (ex: what browser they are using)
    - Perform calculations on user's computer (ex: form validation)
    - Pull data from an external API (sourcedata)

- A web standard (but not supported identically by all browsers)

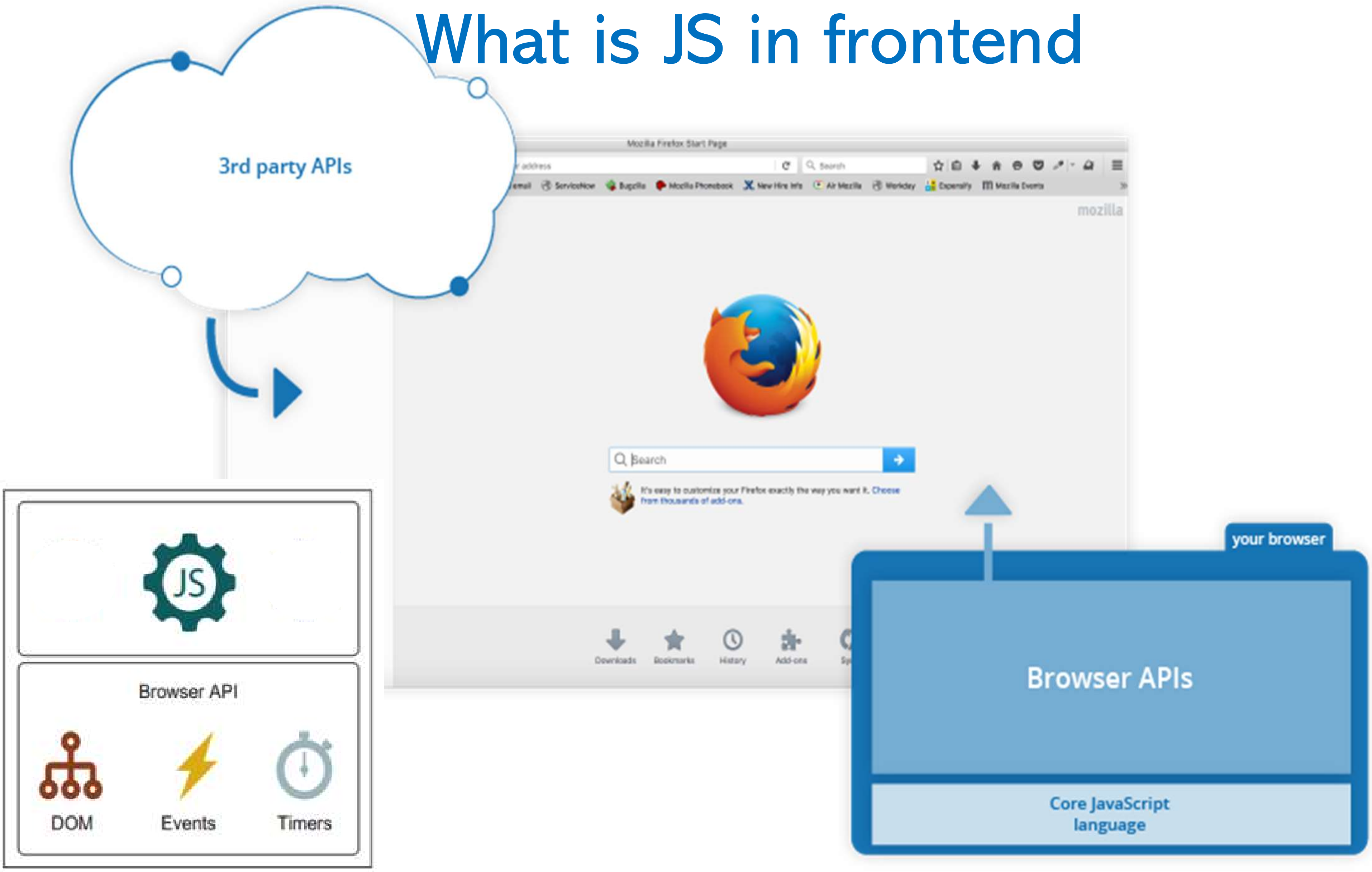
JS in details

Where we meet JS in 2022?

In our classes we find JS in:



What is JS in frontend



JavaScript (in Frontend) = ECMAScript implementation + BrowserAPI

ECMAScript Language Specification defines the *ECMAScript Language*, or just **ECMAScript** (aka **JavaScript**). ECMA-262 only specifies language syntax and semantics of the core API, such as `Array`, `Function`, and `globalThis`, while valid implementations of JavaScript add their own functionality like input/output or filesystem handling.

This standard specifies:

- language syntax - parsing rules, keywords, statements, declarations, operators etc.
- types - logical, numeric, string, object etc.
- prototypes and rules of inheritance
- standard library of built-in objects and functions - JSON, Math, methods of the Array object, methods of introspection called on objects, etc.

ECMAScript, however, does not define any aspects of HTML, CSS, or web APIs such as the DOM (Document Object Model).

Browser API contains for example:

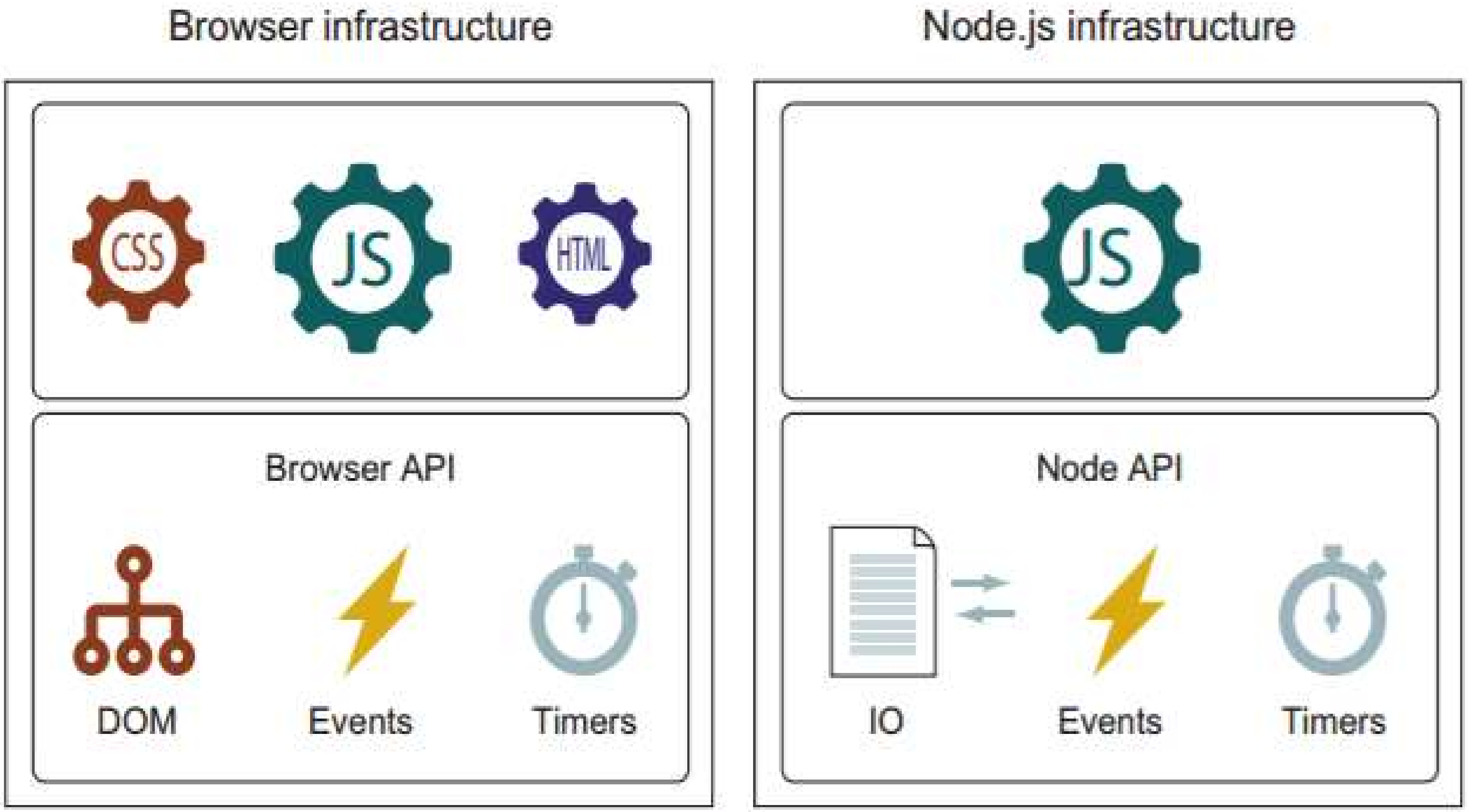
- Document manipulation API (DOM (Document Object Model) API)
- API for retrieving data from the server (XMLHttpRequest or Fetch API)
- API for drawing and editing graphics (CANVAS, WebGL)
- Audio & Video APIs (HTMLMediaElement, Web Audio API, WebRTC)
- Device API (Notifications API, Vibration API, Geolocation API)
- Client-side storage API (Web Storage API, IndexedDB API)

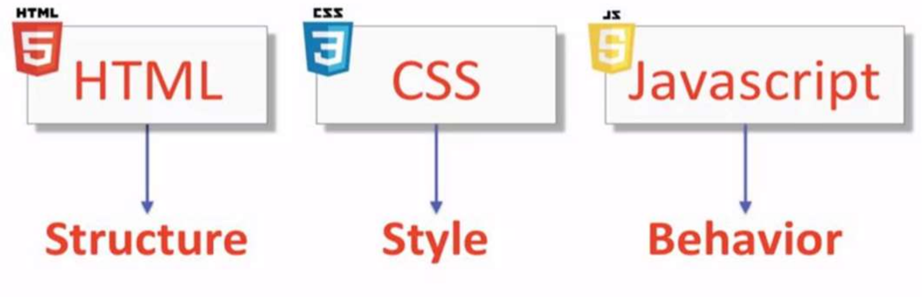
API external :

TwitterAPI,, GoogleMapsAPI FacebookAPI YouTubeAPI AmazonS3

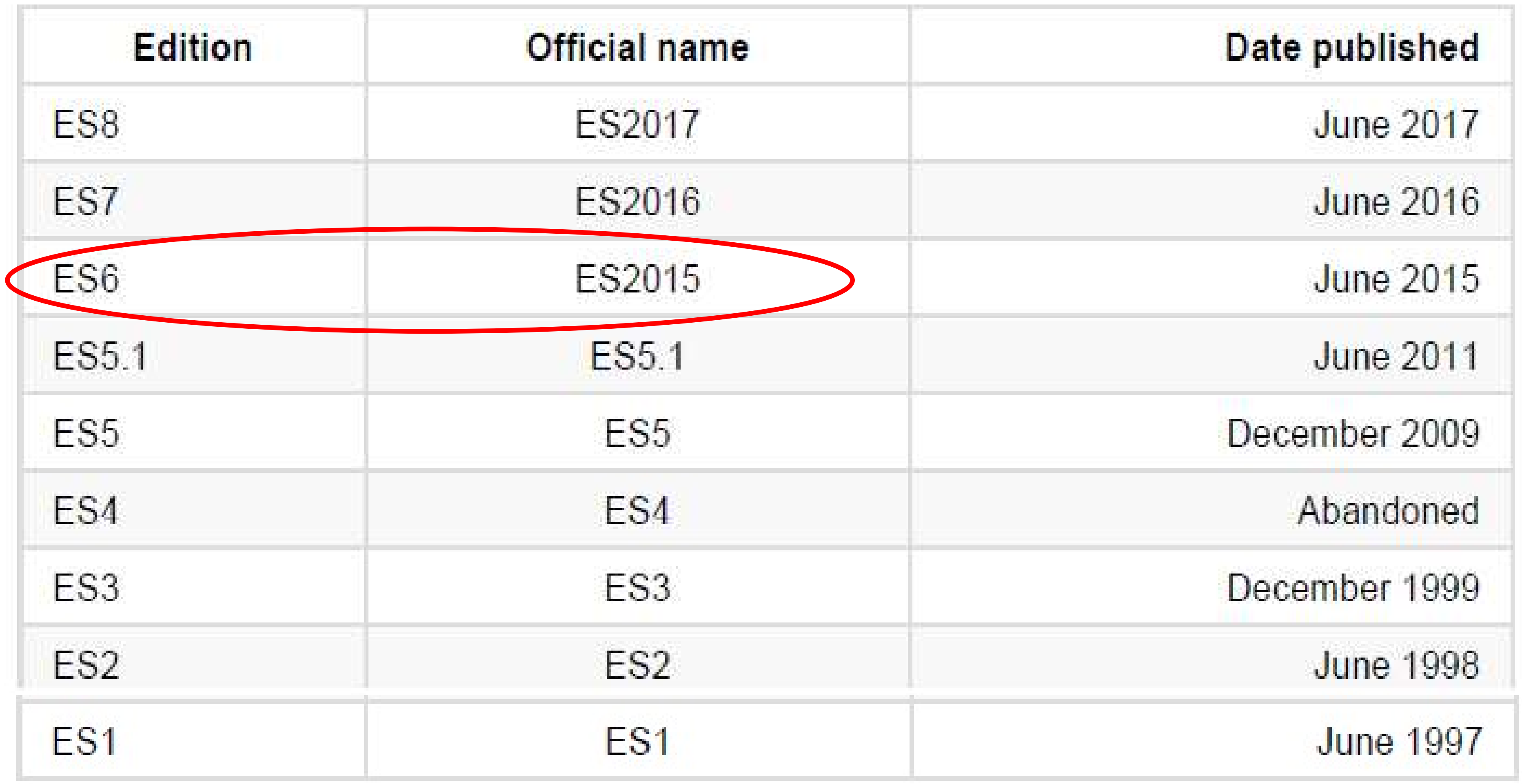
more -> https://www.programmableweb.com/category/all/apis

JS in different environments





JavaScript version



Edition	Official name	Date published
ES8	ES2017	June 2017
ES7	ES2016	June 2016
ES6	ES2015	June 2015
ES5.1	ES5.1	June 2011
ES5	ES5	December 2009
ES4	ES4	Abandoned
ES3	ES3	December 1999
ES2	ES2	June 1998
ES1	ES1	June 1997

JavaScript (JS) - ECMA6

ECMA6 = ECMAScript 2016 = ES6

WHY ES6 ?

Contrary to ES5, the new version is not only a slight improvement of its predecessor (facelifting) but introduces a completely new qualitative approach to writing JS code.

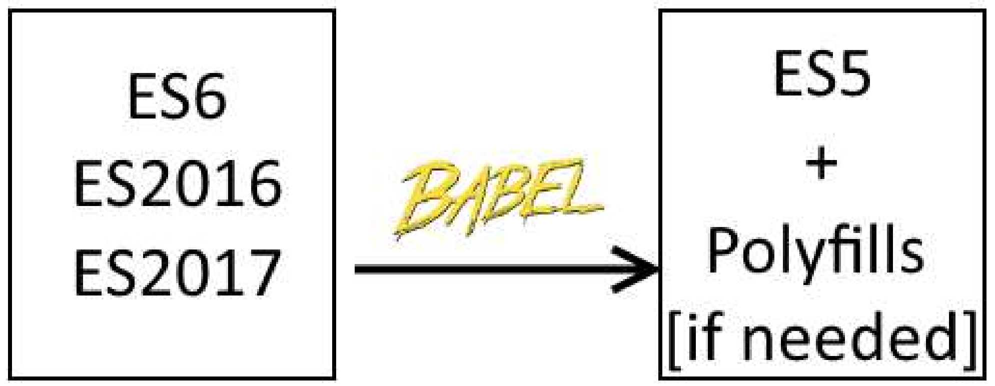
Contains:

- new syntax forms,
- new forms of code organization,
- supports many new APIs that make it easier to handle different types of data

ES6 what’s new

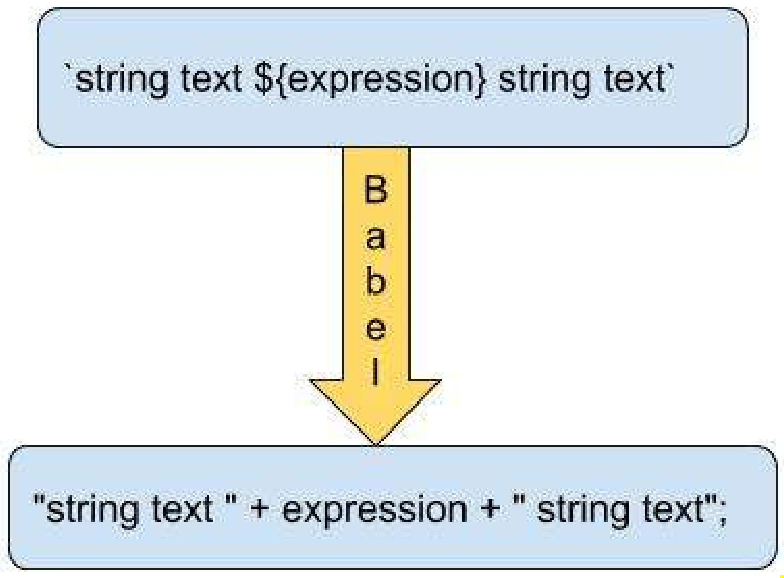
- let/const
- template strings
- new ways to declare objects
- classes
- map, filter, reduce (ES5)
- arrow functions
- for … of
- Promises
- Modules

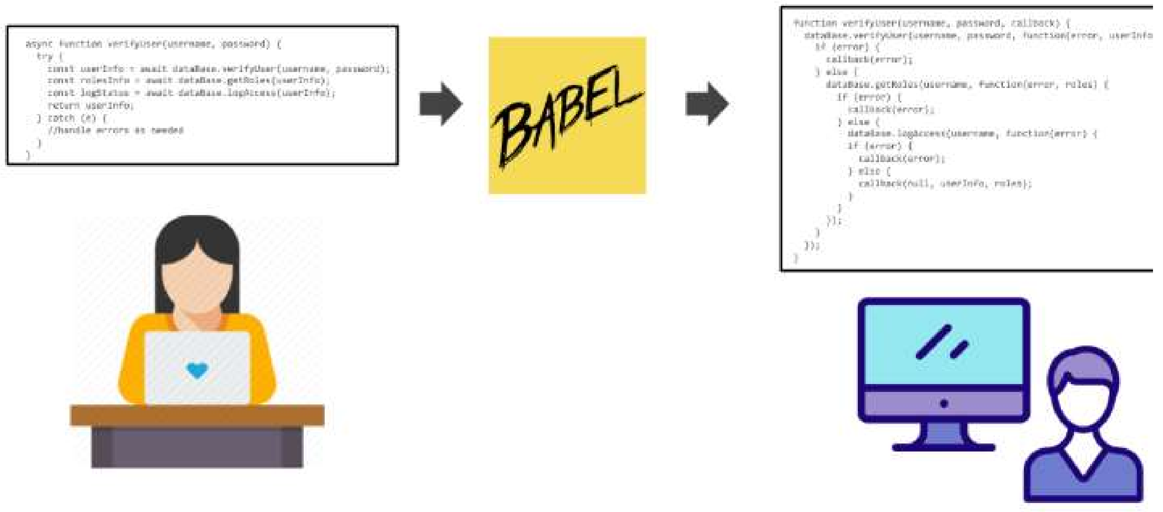
- Proxy
- Iterators
- Generators
- Symbols
- Map/Set, WeakMap/WeakSet
- extended standard library (Number, Math, Array)



```
[1, 2, 3].map((n) => n + 1)
```

```
[1, 2, 3].map(function(n) { return n + 1 })
```


`string text ${expression} string text`
Babel
"string text " + expression + " string text";



```
async function verifyUser(username, password) {
  try {
    const userInfo = await dataBase.verifyUser(username, password);
    const rolesInfo = await dataBase.getRoles(userInfo);
    const logStatus = await dataBase.logAccess(userInfo);
    return userInfo;
  } catch (e) {
    //handle errors as needed
  }
}
```

```
function verifyUser(username, password, callback) {
  dataBase.verifyUser(username, password, function(error, userInfo
    if (error) {
      callback(error);
    } else {
      dataBase.getRoles(username, function(error, roles) {
        if (error) {
          callback(error);
        } else {
         dataBase.logAccess(username, function(error) {
         if (error) {
           callback(error);
         } else {
           callback(null, userInfo, roles);
         }
        }
      });
    }
  });
}
```

wsparcie danych funkcjonalności

- http://kangax.github.io/compat-table/es6/
- https://caniuse.com/

Interactive Webpages with JS

- Now that we know how to add ***content*** and ***styles*** to a web page, lets explore how to add responsive ***behavior***

How do we interact with the page?

How to link JS to HTML

```
<script>
...kod...
</script>

Or better

<head>
    <script src="./js/script.js" defer></script>
</head>
```

Remember separation of concerns from CSS and HTML?
The same is in JS and HTML

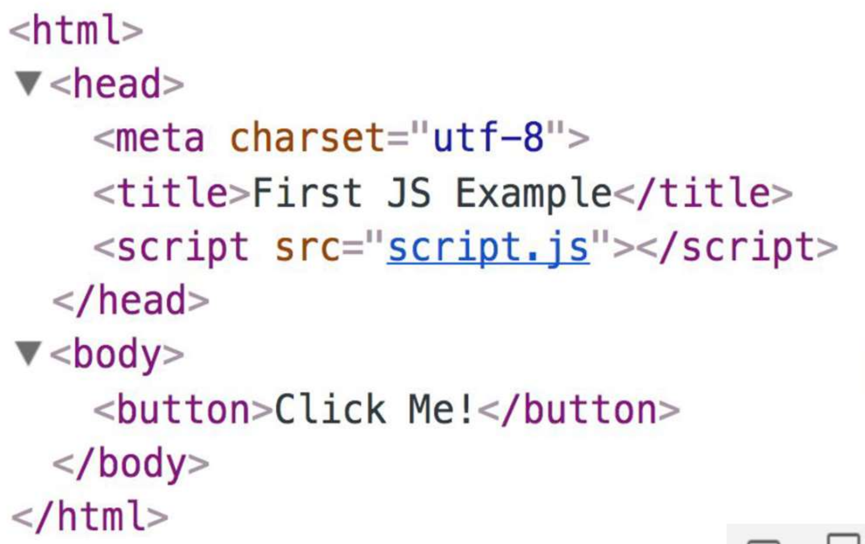
How does JavaScript get loaded?

All JavaScript code used in the page should be stored in a separate .js file
JS code *can* be placed directly in the HTML file's body or head, **but this is poor code quality. You should always separate content, presentation, and behavior**

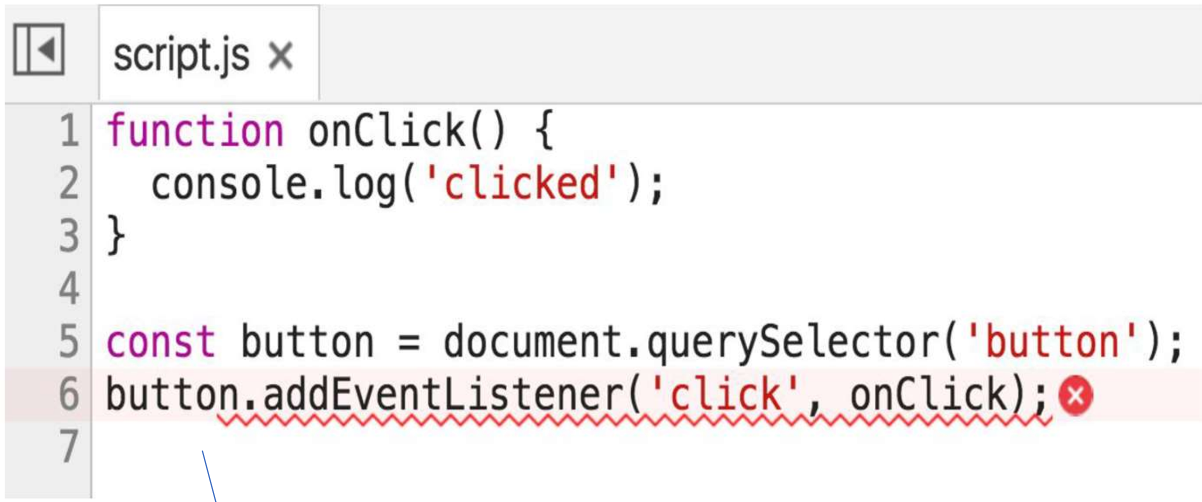
When the browser loads HTML and comes across a <script>...</script> tag, it can’t continue building the DOM. It must execute the script right now.

The same happens for external scripts <script src="..."></script>: the browser must wait for the script to download, execute the downloaded script, and only then can it process the rest of the page.

JavaScript get loaded -problems

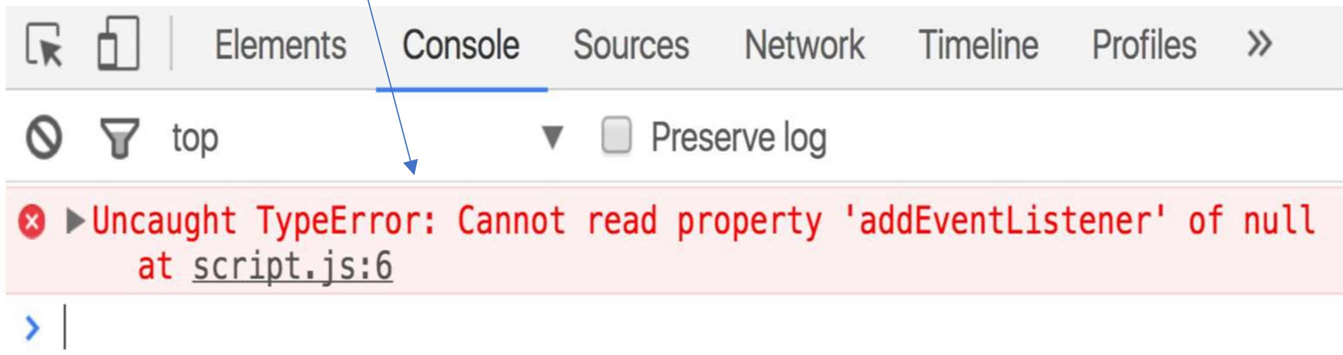


```html
<html>
▼<head>
    <meta charset="utf-8">
    <title>First JS Example</title>
    <script src="script.js"></script>
  </head>
▼<body>
    <button>Click Me!</button>
  </body>
</html>
```



script.js ×

```javascript
1 function onClick() {
2   console.log('clicked');
3 }
4
5 const button = document.querySelector('button');
6 button.addEventListener('click', onClick);
7
```



```
Uncaught TypeError: Cannot read property 'addEventListener' of null
    at script.js:6
```

Error! Why?

Error descripion:

We are only at the <script> tag, which is at the top of the document… so the <button> isn't available yet.
Therefore querySelector returns null, and we can't call addEventListener on null.

Solution

```html
<head>
    <script src="./js/somescript.js" defer</script>
</head>
```

HTML

You can add the defer attribute onto the script tag so that the JavaScript doesn't execute until after the DOM is Loaded

The defer attribute tells the browser not to wait for the script. Instead, the browser will continue to process the HTML, build DOM. The script loads “in the background”, and then runs when the DOM is fully built.

Other old-school ways of doing this (**don't do these**):

- Put the <script> tag at the bottom of the page
- Listen for the "load" event on the window object

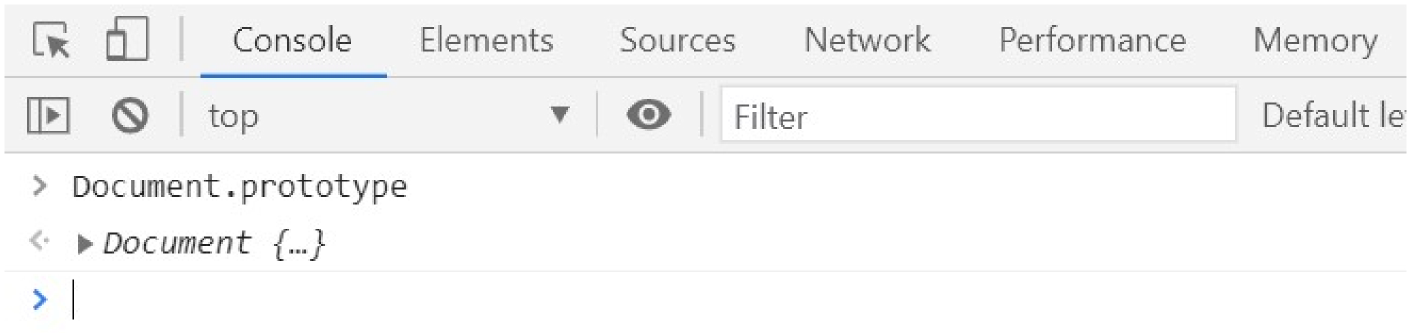
You will see tons of examples on the internet that do this. They are out of date. defer is widely supported and better.

JavaScript execution

- There is **no "main method"**
  - The script file is executed from top to bottom.
- There's **no compilation** by the developer
  - JavaScript is compiled and executed on the fly by the browser

How to debug and inspect code

The **console** is a tool to **inspect** JavaScript code.



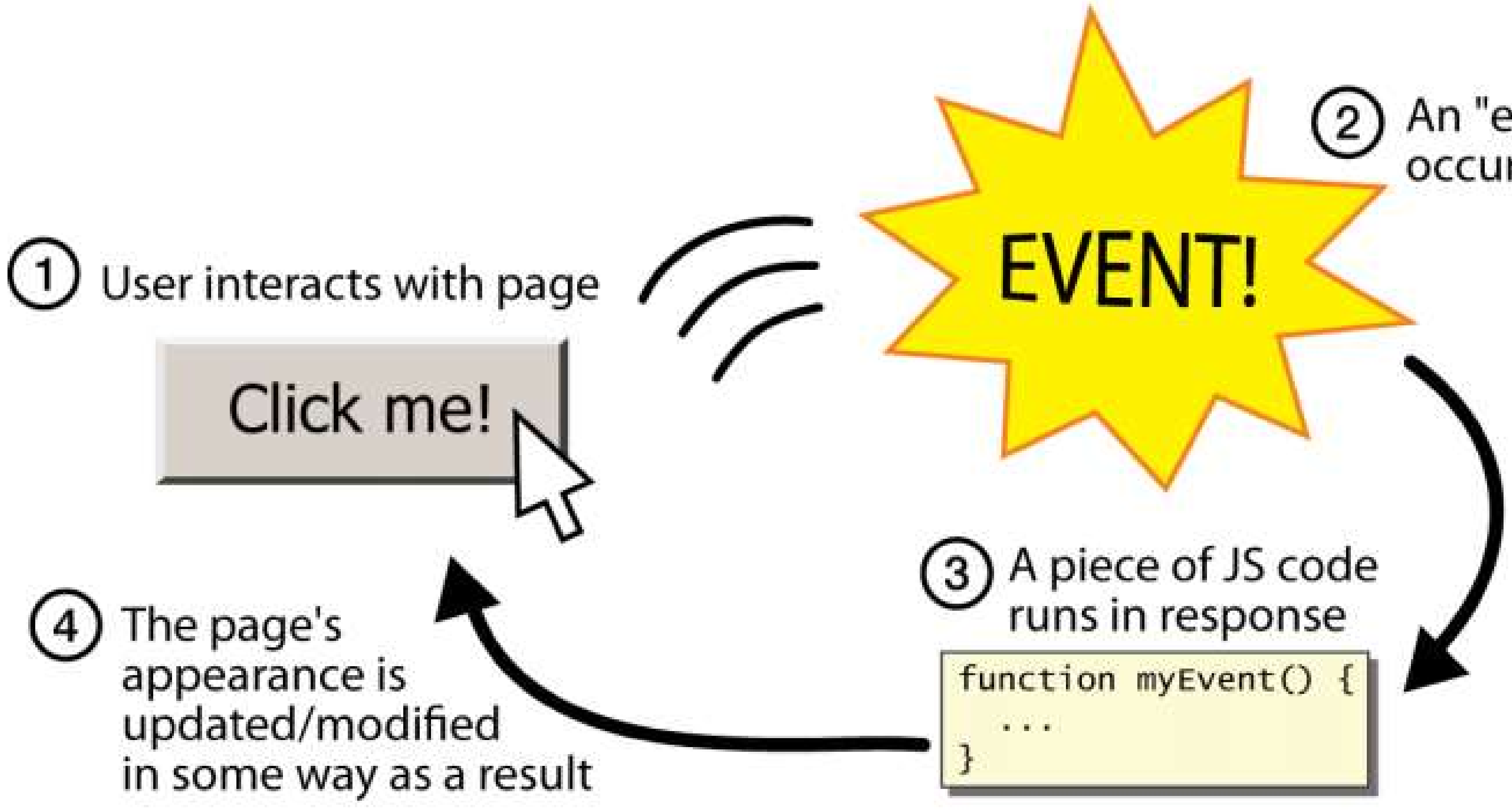
```
> Document.prototype
< ▶ Document {…}
> |
```

Use `console.log()` to output values to the browser console, most often used to debug JS programs.

Event-driven programming

Most JavaScript written in the browser is **event-driven**: The code doesn't run right away, but it executes after some event fires.

Event-Driven Programming: writing programs driven by user events


1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of JS code runs in response
4 The page's appearance is updated/modified in some way as a result

```
function myEvent() {
  ...
}
```

An event **handler** or **listener** is a **function** to be executed when a given event occurs

Common Types of JavaScript Events

An event is an **action** on a web page.

Name	Decription
click	A pointing device button (e.g. mouse) has been pressed and released on an element
dblclick	A pointing device button is clicked twice on an element
keydown	Any key is pressed down
keyup	Any key is released
mouseenter	A pointing device is moved onto an element that has the attached
mouseover	A pointing device is moved onto the element that has the listener attached to itself or one of its children
mousemove	A pointing device is moved over an element
mousedown	A pointing device button is pressed on an element

What we need to event Programming?

- Getting DOM objects
- Defined even executor (JS function)
- Bind Event with JS function

Using event listeners

Let's print "Clicked" to the Web Console when the user clicks the given button:

```
<!DOCTYPE html>
<html>
<body>
<button>Click Me!</button>
</body>
</html>
```

We need to add an event listener to the button...

Old-fashion method (don’t do this)

```
<a href="doc.html"
onMouseOver="document.status='test';return true">
   Test test Test
</a>
```

mixing technology

How to add new function to event ?

Click -> fufu1();
Leter Additionally fufu2();
Maybe update fufu1()? But what when we want to delete fufu2() from fufu1() dynamic?

Handling Events with addEventListener

Each DOM object has the following function:

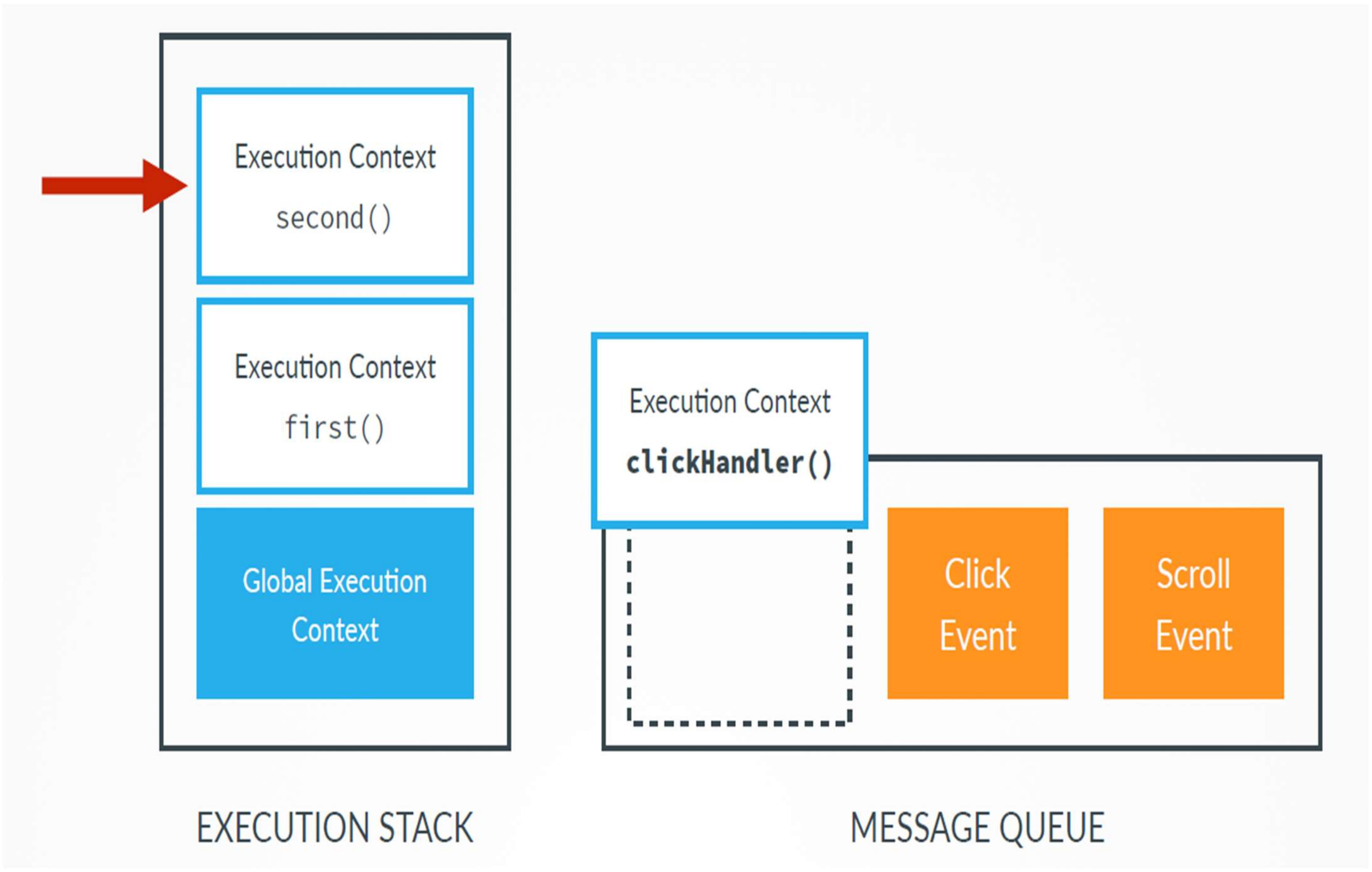
addEventListener(***event name***, ***function name***);

- ***event name*** is the string name of the JavaScript event you want to listen to
Common ones: click, focus, blur, etc

- ***function name*** is the name of the JavaScript function you want to execute when the event fires

```
// attaching a named function
button1.addEventListener("click", handleFunction);
```

HOW EVENTS ARE PROCESSED



addEventListener

The newer addEventListener is preferred as it lets you easily remove a listener to a specific event on an element, and it also lets you add multiple functions to an event listener.

```
var element = document.getElementById('Przycisk');
element.addEventListener('click', startDragDrop, false);
element.addEventListener('click', wypiszCos, false);
element.addEventListener('click', function()
{this.style.color = 'red'; }, false);
```

This method also allows you to remove the event handler using **removeEventListener**

```
element.removeEventListener('click', startDragDrop, false);
element.removeEventListener('click', wypiszCos, false);
```

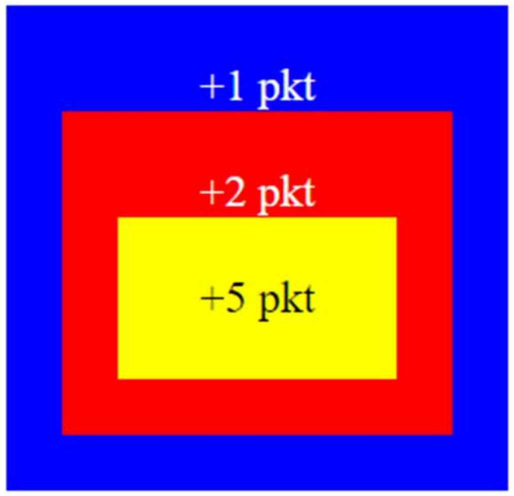
Demo: Start/example1.html

Multiple event listeners

If you put a „click” event listener on an element, what happens if the user clicks a child of that elements?

What If you have event listener set on both an element and a child of that element?

- Do both fire?
- Which fires first?

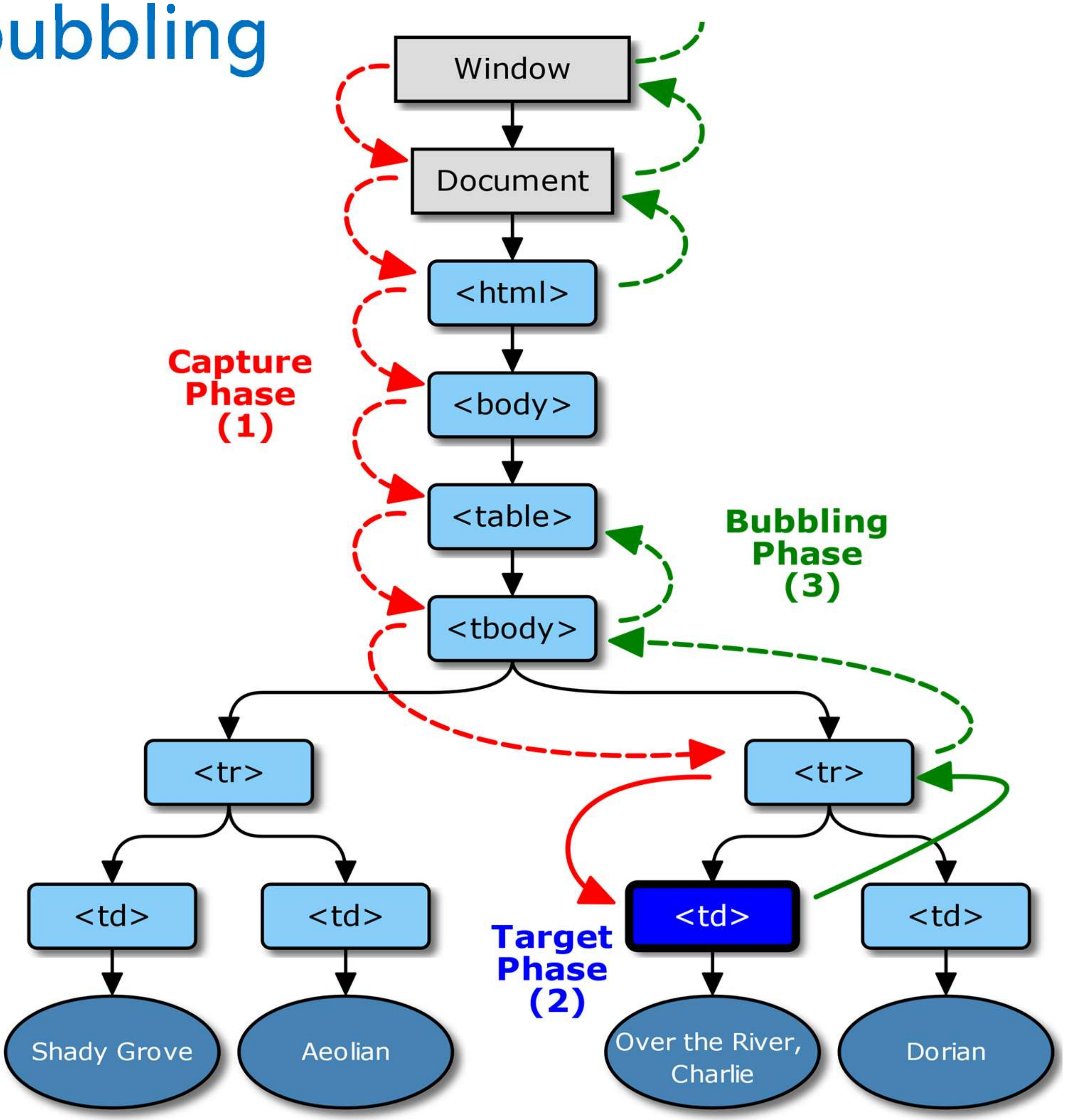


Both events fire if you click the inner element
By default , the event listener on
the inner –most element fires first

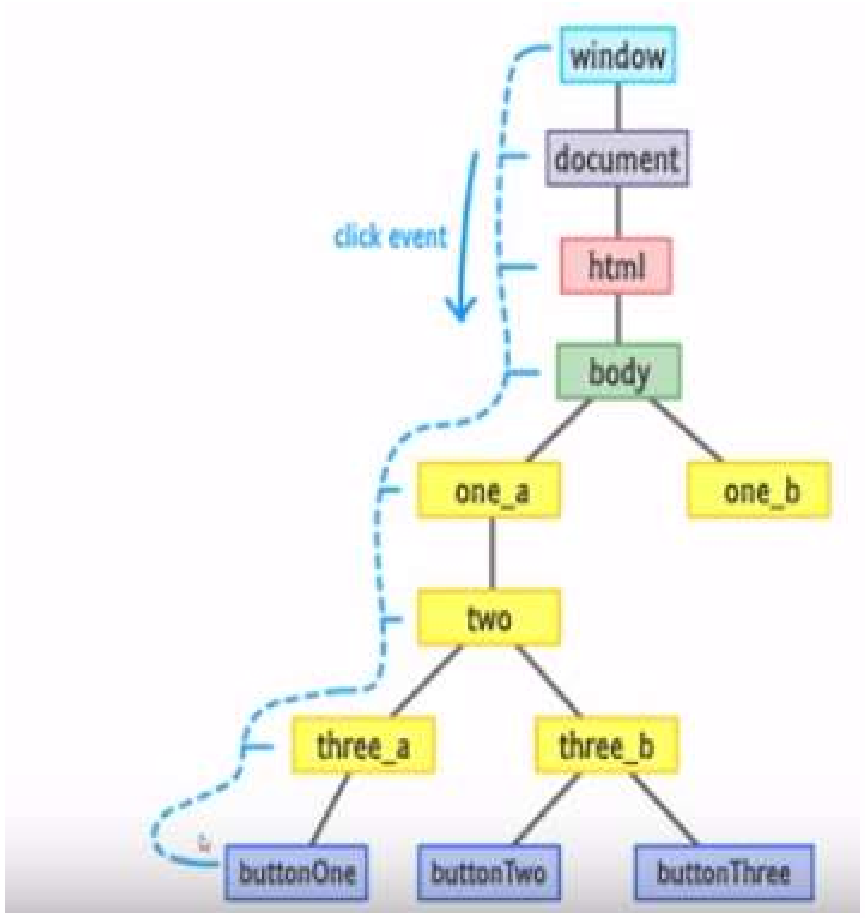
Demo: 7_EventBubbling

Event bubbling

Event ordering (inner-most to outer-most) is known as bubbling.

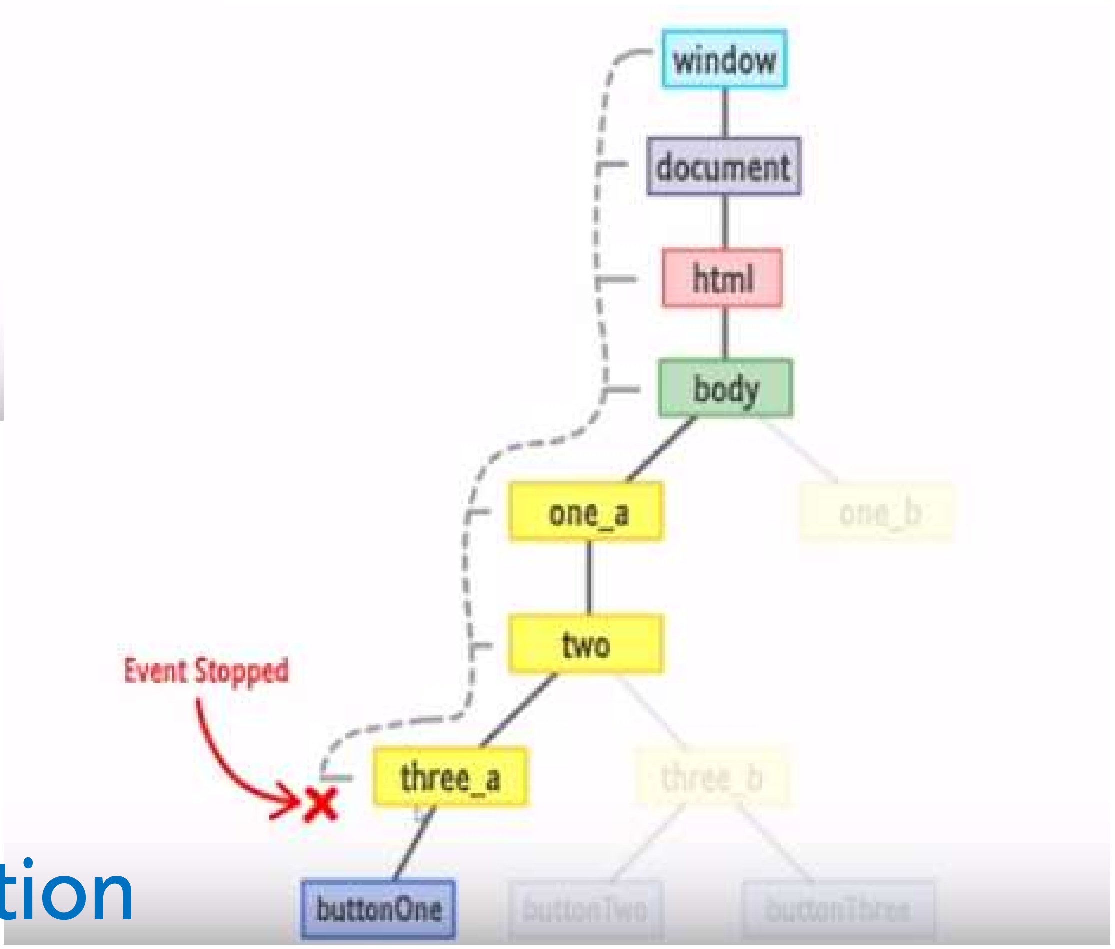


We can stop the event from bubbling up the chain of ancestors by using **event.StopPropagation()**



Normal bubbling events

Stop bubbling events



Demo: 8_StopPropagation

Event objects

Though we have ignored it so far, event handler functions are passed an argument: the event object. This object holds additional information about the event.
When the event occurs, an Event object is created and passed to the event listener. You can optionally "catch" this argument as an optional parameter to get more information about the event.

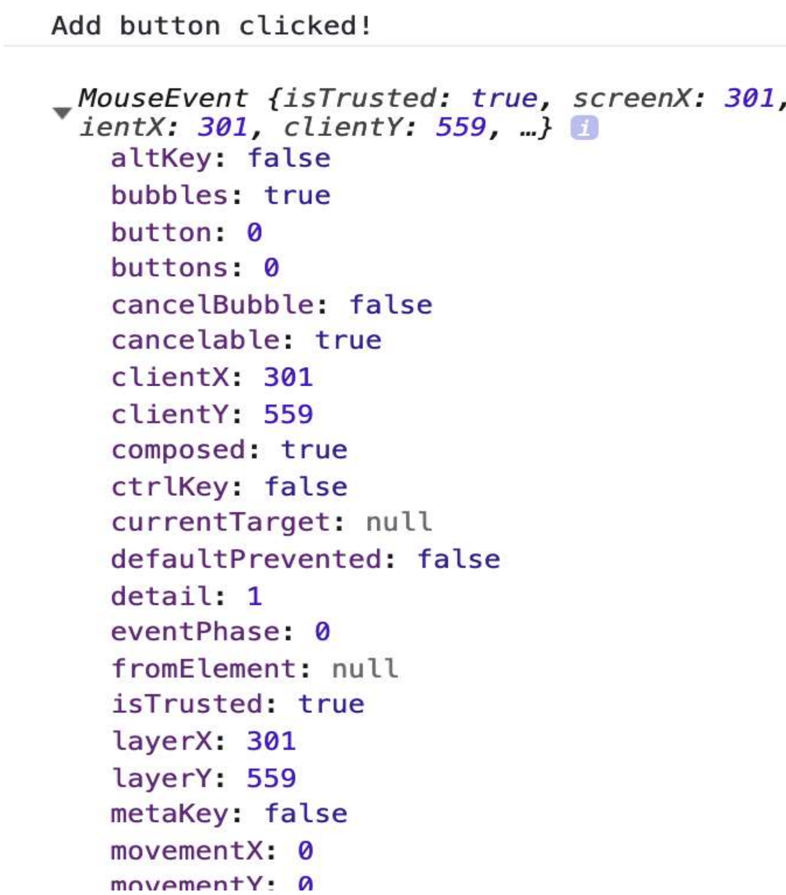
```
function responseFunction(e) {
    // we can access the click Event object here!
}
```

Event Object Properties

Event objects contain properties about an event that occurred

What types of properties do you think an Event object has?

```
addBtn.addEventListener("click", addGroup);

function addGroup(e) {
  console.log("Add button clicked!");
   console.log(e);
}
```



```
Add button clicked!

▼MouseEvent {isTrusted: true, screenX: 301,
 ientX: 301, clientY: 559, …} i
   altKey: false
   bubbles: true
   button: 0
   buttons: 0
   cancelBubble: false
   cancelable: true
   clientX: 301
   clientY: 559
   composed: true
   ctrlKey: false
   currentTarget: null
   defaultPrevented: false
   detail: 1
   eventPhase: 0
   fromElement: null
   isTrusted: true
   layerX: 301
   layerY: 559
   metaKey: false
   movementX: 0
   movementY: 0
```

Mouse Event example

```
let button = document.querySelector("button");
button.addEventListener("mousedown", event => {
    if (event.button == 0) {
        console.log("Left button");
    } else if (event.button == 1) {
                console.log("Middle button");
            } else if (event.button == 2) {
                    console.log("Right button");
              }
});
```

which mouse button was pressed, we can look at the event object’s button property.

Key events

```
window.addEventListener("keydown", event => {
   if (event.key == "v") {
     document.body.style.background = "violet";
   }
 });
 window.addEventListener("keyup", event => {
   if (event.key == "v") {
     document.body.style.background = "";
   }
 });
```

Is key „v” keypress? .

Pointer events

Mouse motion

```
let lastX; // Tracks the last observed mouse X position
let bar = document.querySelector("div");

bar.addEventListener("mousedown", event => {
  if (event.button == 0) {
    lastX = event.clientX;
    window.addEventListener("mousemove", moved);
    event.preventDefault(); // Prevent selection
  }
});

function moved(event) {
  if (event.buttons == 0) {
    window.removeEventListener("mousemove", moved);
  } else {
    let dist = event.clientX - lastX;
    let newWidth = Math.max(10, bar.offsetWidth + dist);
    bar.style.width = newWidth + "px";
    lastX = event.clientX;
  }
}
```

Default actions

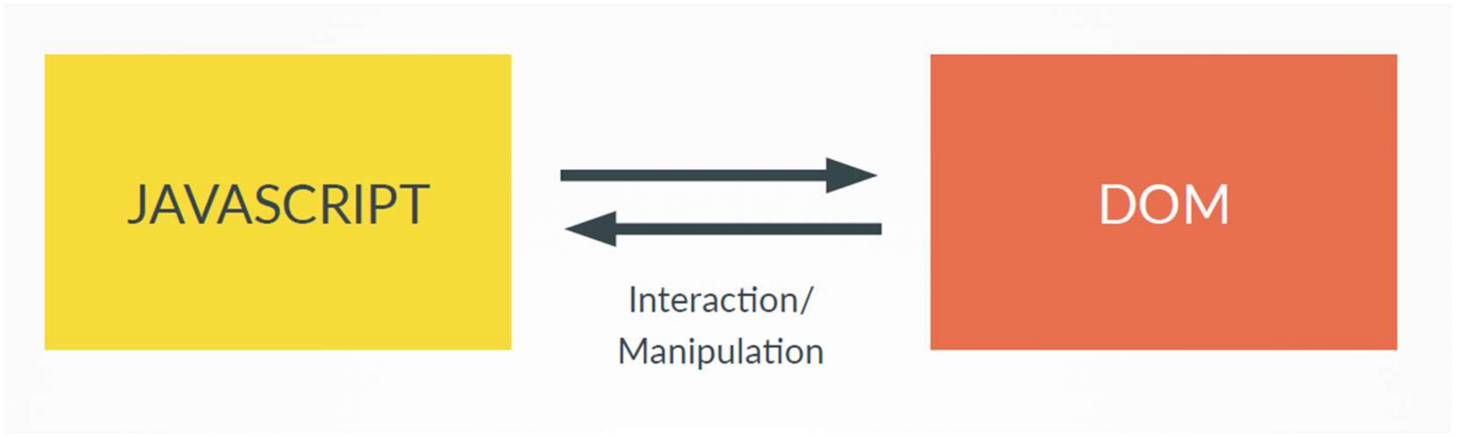
Many events have a default action associated with them. If you click a link, you will be taken to the link’s target. If you press the down arrow, the browser will scroll the page down. If you right-click, you’ll get a context menu. And so on.

For most types of events, the JavaScript event handlers are called before the default behavior takes place. If the handler doesn’t want this normal behavior to happen, typically because it has already taken care of handling the event, it can call the preventDefault method on the event object.

```
form.addEventListener('submit', (e) => {
e.preventDefault(); input =
form.querySelector("input").value;
alert( " sending " + input);
});
```

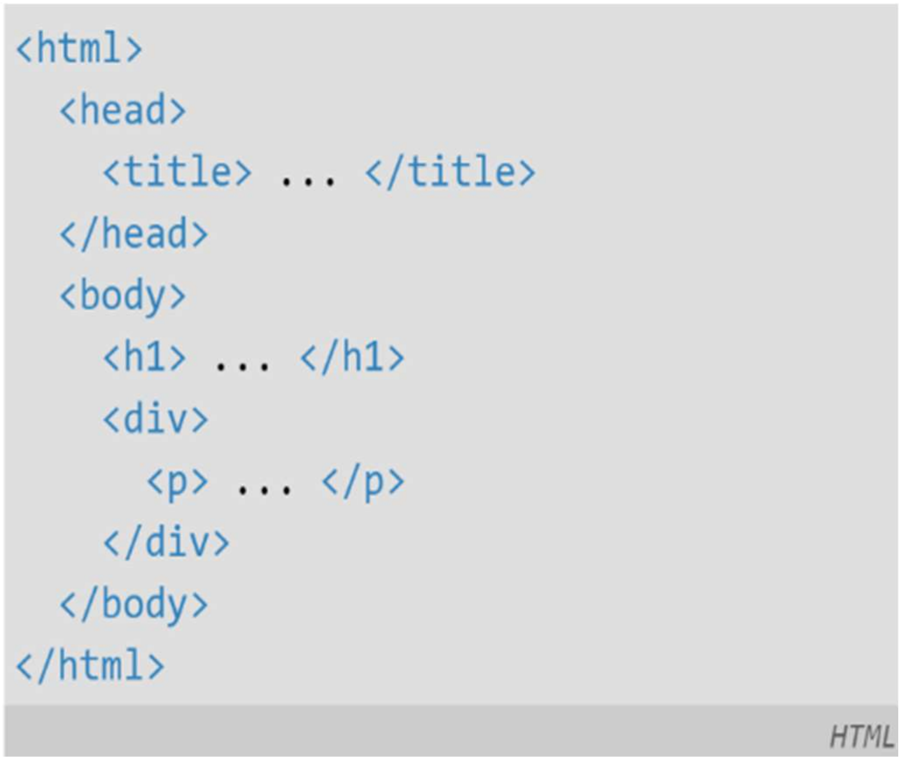
Demo: 9_PreventDefault

How do we access a DOM object from JavaScript?



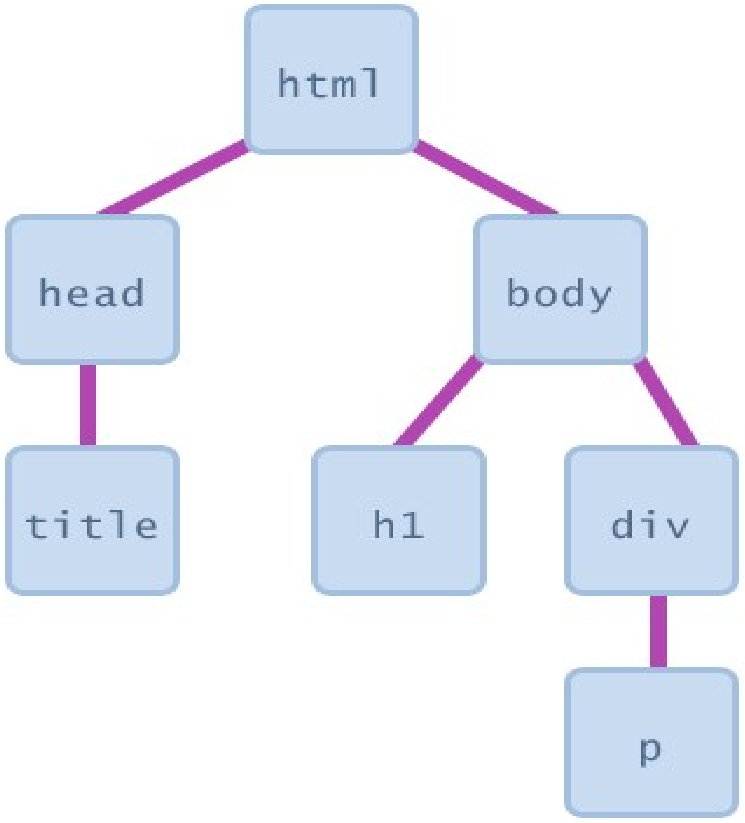
Reminder: The DOM (Document Object Model)

- Every element on a page is accessible in JavaScript through the **DOM**: **Document Object Model**
- The DOM is the tree of nodes corresponding to HTML elements on a page.
- Can modify, add and remove nodes on the DOM, which will modify, add, or remove the corresponding element on the page.



```html
<html>
  <head>
    <title> ... </title>
  </head>
  <body>
    <h1> ... </h1>
    <div>
      <p> ... </p>
    </div>
  </body>
</html>
```

HTML



$$\begin{tikzcd}
 & html \arrow[dl, dash] \arrow[dr, dash] & & \\
head \arrow[d, dash] & & body \arrow[dl, dash] \arrow[dr, dash] & \\
title & h1 & & div \arrow[d, dash] \\
 & & & p
\end{tikzcd}$$

DOM

The DOM is a tree of node objects corresponding to the HTML elements on a page.

- JS code can examine these nodes to see the state of an Element   (e.g. to get what the user typed in a text box)

- JS code can edit the attributes of these nodes to change the attributes of an element (e.g. to toggle a style or to change the contents of an <h1> tag)

- JS code can add elements to and remove elements from a web page by adding and removing nodes from the DOM

How do we access a DOM object from JavaScript?

We can access an HTML element's corresponding DOM node in JavaScript via the querySelector function:
document.querySelector('***css selector***');
- Returns the **first** element that matches the given CSS selector.

And via the querySelectorAll function:
document.querySelectorAll('***css selector***');
- Returns **all** elements that match the given CSS selector.

Another finding methods

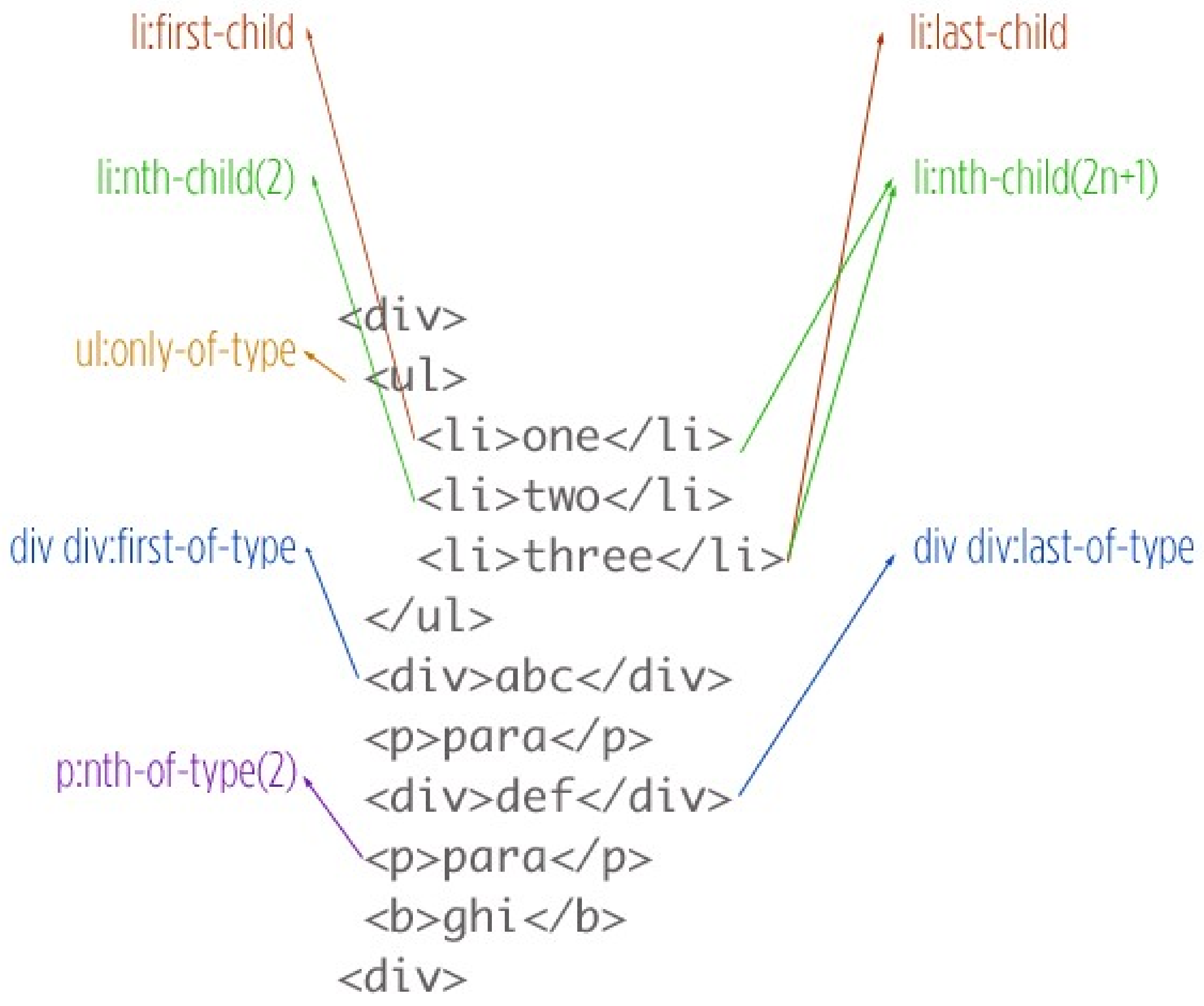
Method name	looks for	result
querySelector	CSS-selector	The first finding
querySelectorAll	CSS-selector	collection
getElementById	id	element
getElementsByName	name	element
getElementsByTagName	Tag name lub „*”	collection
getElementsByClassName	class	collection

Get object - example

```
// Returns the DOM object for the HTML element
// with id="button", or null if none exists.
let element = document.querySelector('#button’);

// Returns a list of DOM objects containing all
// elements that have a "quote" class AND all
// elements that have a "comment" class.
let elementList = document.querySelectorAll('.quote, .comment’);

// Returns a checked chceckboxs from forms
let checkedElements = document.querySelectorAll('"input:checked"');

// Using Attribute Selectors to get a Selected Radio Button
let checkedBtn = qs("input[name='color-count']:checked");
```

Getting DOM objects – use pseudo-class

More: https://www.w3schools.com/css/css_pseudo_elements.asp



```
<div>
 <ul>
   <li>one</li>
   <li>two</li>
   <li>three</li>
 </ul>
 <div>abc</div>
 <p>para</p>
 <div>def</div>
 <p>para</p>
 <b>ghi</b>
<div>
```

Used to selection in collection or forms

let checkedElements = document.querySelectorAll('.checkbox:checked');

DOM object properties

You can access **attributes** of an HTML element via a property (field) of the DOM object:

```
const image = document.querySelector('img');
image.src = 'new-picture.png’;
```

Some exceptions:
- Notably, you can't access the class attribute via object**.class**

How to update existing element?

HTML Update

```
let elem = document.querySelector('#myElem');
elem.innerHTML = ’<h1> GR  </h1>';
```

Text Content Update

```
let elem = document.querySelector('#myElem');
elem.innerText = ’GR ';
```

Adding and removing classes

You can control **classes** applied to an HTML element via classList.add and classList.remove:

```
const image = document.querySelector('img’);

// Adds a CSS class called "active".
image.classList.add('active’);

// Removes a CSS class called "hidden".
image.classList.remove('hidden');
```

Add elements via DOM

We can create elements dynamically and add them to the web page via createElement and appendChild:

document.createElement(***tag string***)
element.appendChild(***element***);

Technically you can also add elements to the webpage via innerHTML, but it poses a security risk.

```
// Try not to use innerHTML like this:
element.innerHTML = '<h1>Hooray!</h1>';
```

Demo

Remove elements via DOM

We can also call remove elements from the DOM by calling the remove() method on the DOM object:

```
element.remove();
```

And actually setting the innerHTML of an element to an empty string is a fine way of removing all children from a parent node:

```
// This is fine and poses no security risk.
element.innerHTML = '';
```

Demo

How to update HTML document - summary

Several strategies for updating HTML elements in JS:

1. Change content of existing HTML elements in page:
- Good for simple text updates

2. Add elements via createElement and appendChild
- Needed if you're adding a variable number of elements

3. Put all "views" in the HTML but set inactive ones to hidden, then update display state as necessary.

DOM traversing

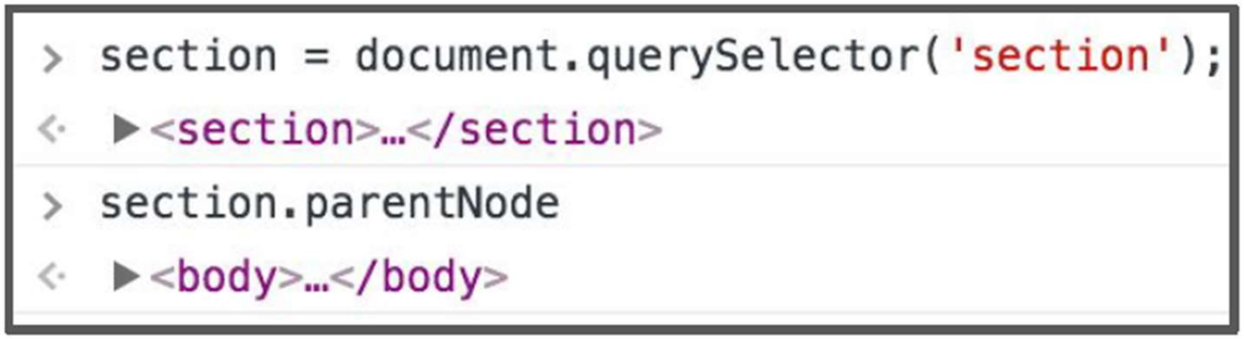
Node properties

Property	Description
textContent	The text content of a node and its descendants. (This property is writeable)
childNodes	An array of this node's children (empty if a leaf)
parentNode	A reference to this node's parent Node

```
<body>
    <h1>My favorites</h1>
    <section>
        <p>Strawberries</p>
        <p>Chocolate</p>
    </section>
</body>
```

What's the **parentNode** of **<section>**?

parentNode



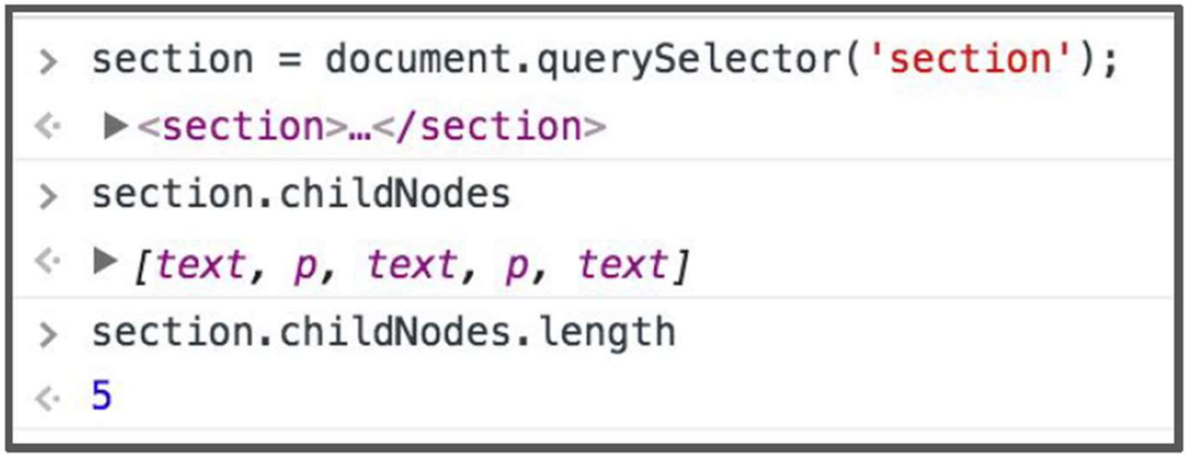
```
> section = document.querySelector('section');
<· ▶<section>…</section>
> section.parentNode
<· ▶<body>…</body>
```

```
<body>
    <h1>My favorites</h1>
    <section>
        <p>Strawberries</p>
        <p>Chocolate</p>
    </section>
</body>
```

The **parentNode** of **`<section>`** is **`<body>`**.

What are the **childNodes** of **`<section>`**?

Child Node



```
> section = document.querySelector('section');
<· ▶<section>…</section>
> section.childNodes
<· ▶[text, p, text, p, text]
> section.childNodes.length
<· 5
```

???

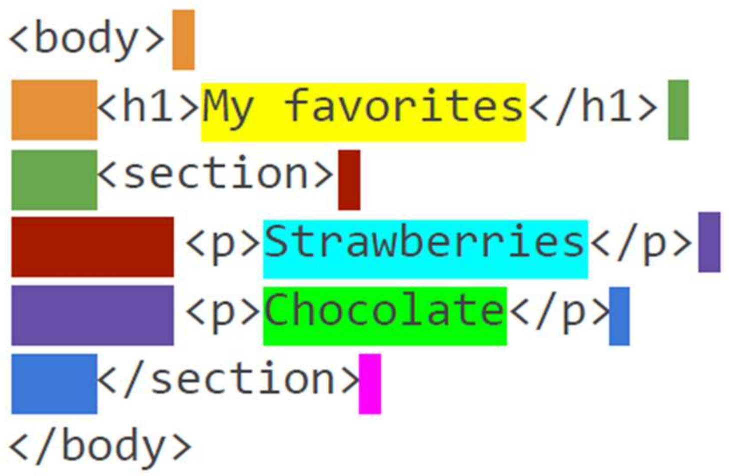
```
<body>
    <h1>My favorites</h1>
    <section>
        <p>Strawberries</p>
        <p>Chocolate</p>
    </section>
</body>
```

Why does `section` have 5 children, not 2?!

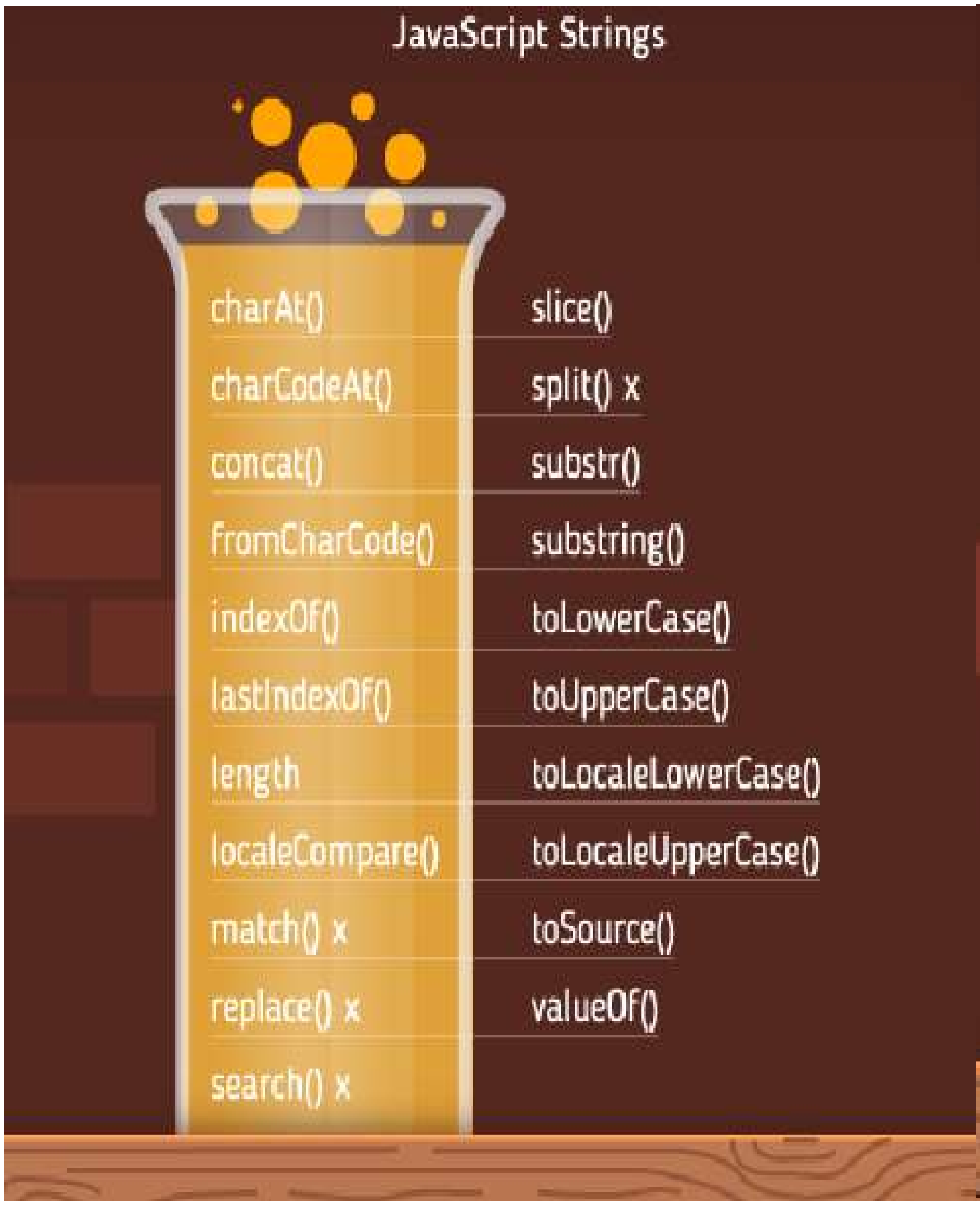
TextNodes

In addition to `Element` nodes, the DOM also contains `Text` nodes. All text present in the HTML, **including whitespace**, is contained in a text node:

```
<body>
    <h1>My favorites</h1>
    <section>
        <p>Strawberries</p>
        <p>Chocolate</p>
    </section>
</body>
```

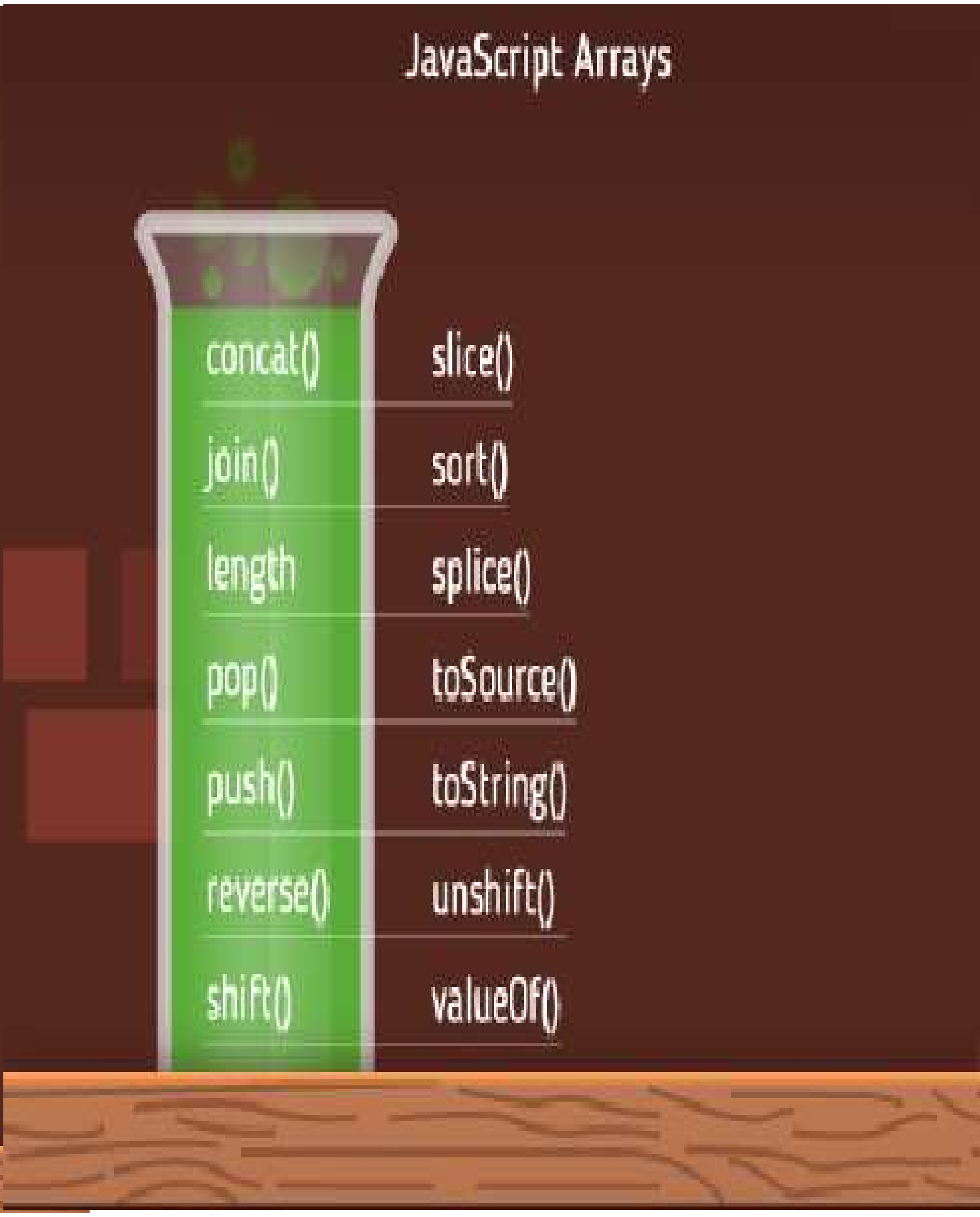


```
<body>
<h1>My favorites</h1>
<section>
    <p>Strawberries</p>
    <p>Chocolate</p>
</section>
</body>
```



JavaScript Strings

charAt()	slice()
charCodeAt()	split() x
concat()	substr()
fromCharCode()	substring()
indexOf()	toLowerCase()
lastIndexOf()	toUpperCase()
length	toLocaleLowerCase()
localeCompare()	toLocaleUpperCase()
match() x	toSource()
replace() x	valueOf()
search() x	

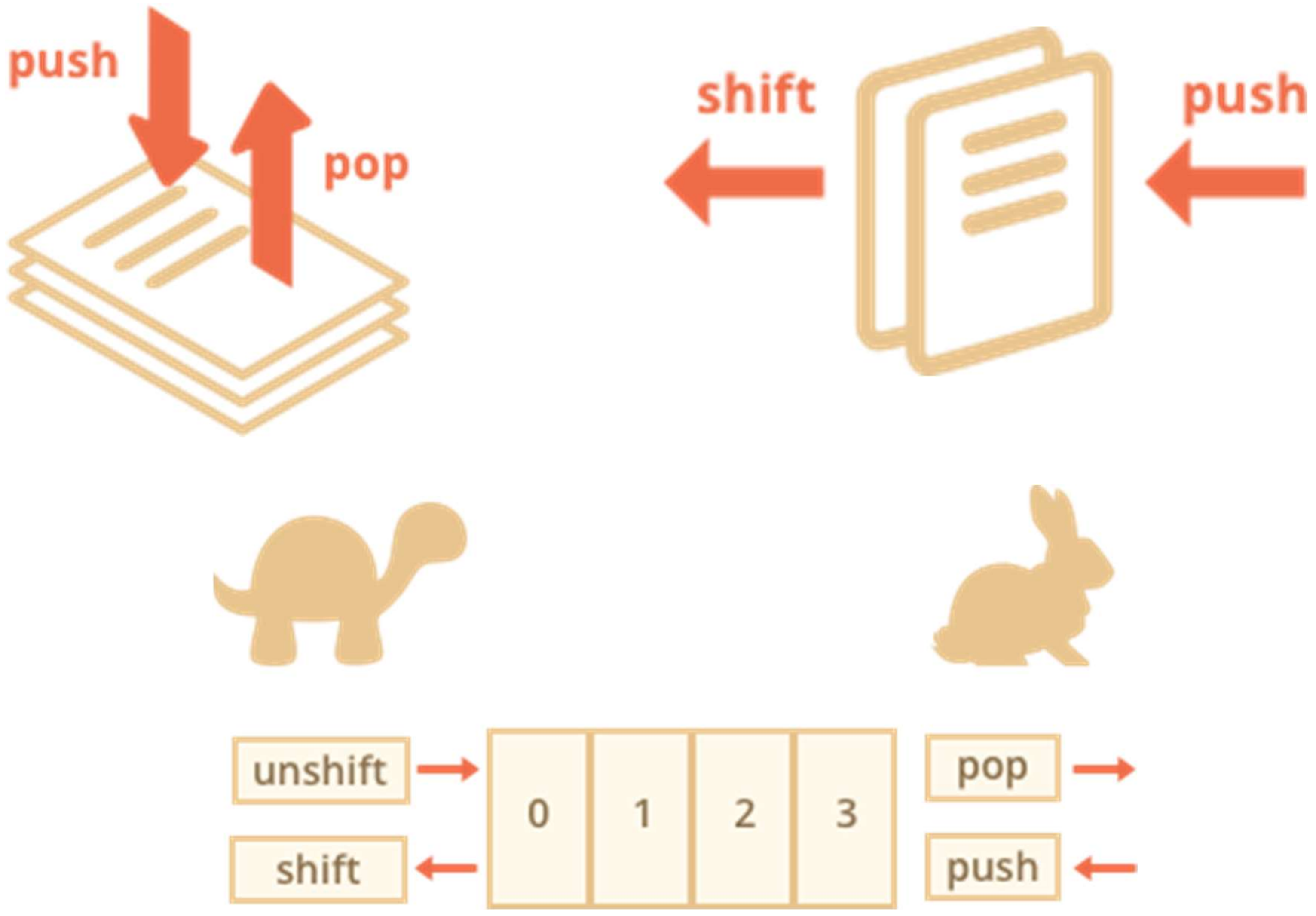


JavaScript Arrays

concat()	slice()
join()	sort()
length	splice()
pop()	toSource()
push()	toString()
reverse()	unshift()
shift()	valueOf()

Wiecej   http://www.w3schools.com/js

Przydatne metody dla obiektu Array



Method	Description
`list.push(`*`element`*`)`	Add ***element*** to back
`list.unshift(`*`element`*`)`	Add ***element*** to front

Method	Description
`list.pop()`	Remove from back
`list.shift()`	Remove from front

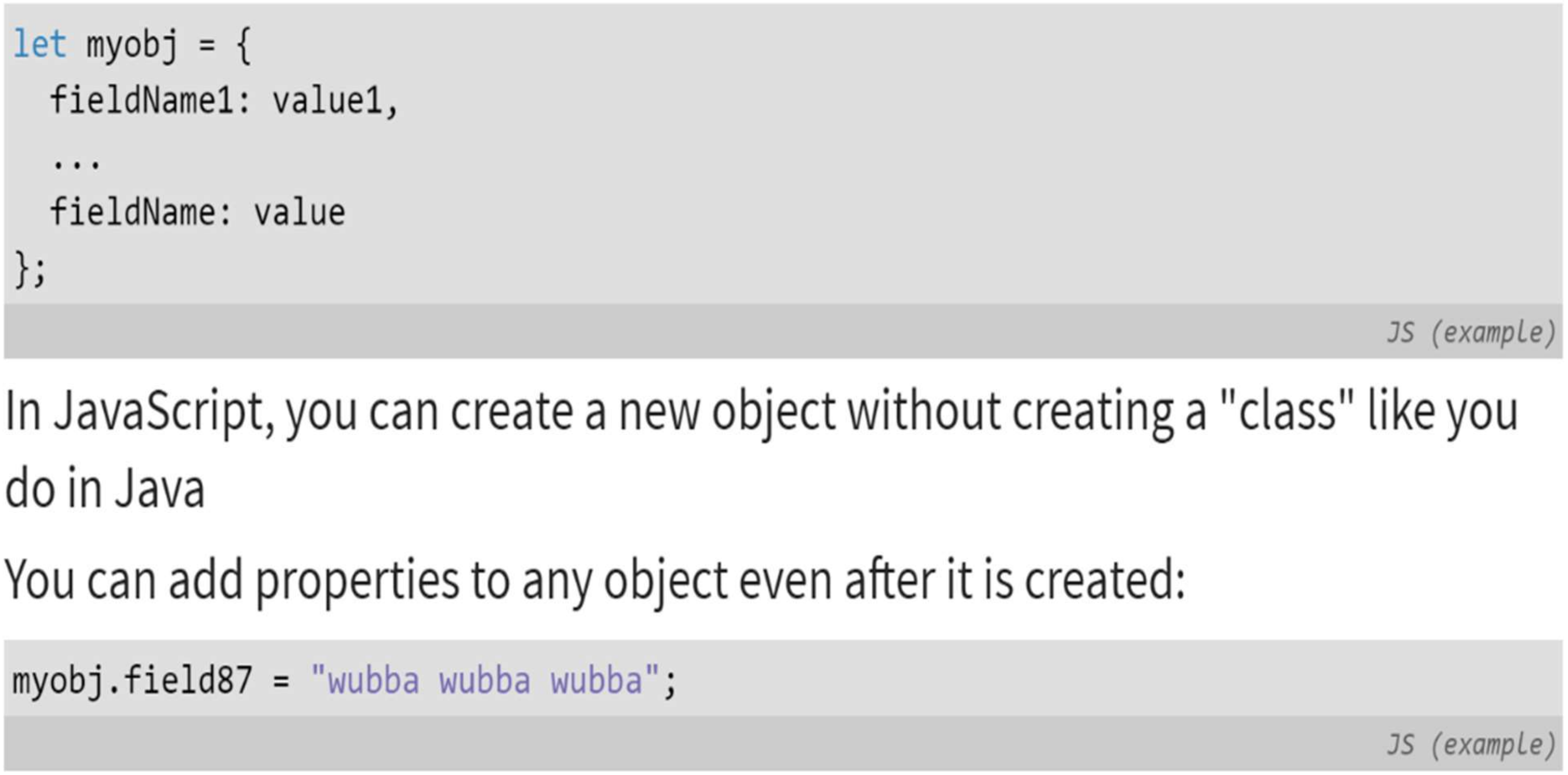
Method	Description
`list.indexOf(`*`element)`*	Returns numeric index for ***element*** or -1 if none found

JSON

JavaScript Object Notation

A data format that represents data as a set of JavaScript objects natively supported by all modern browsers (and libraries to support it in old ones)

But first… JavaScript objects



```
let myobj = {
  fieldName1: value1,
  ...
  fieldName: value
};
```

JS (example)

In JavaScript, you can create a new object without creating a "class" like you do in Java

You can add properties to any object even after it is created:

```
myobj.field87 = "wubba wubba wubba";
```

JS (example)

Example

```
let fidgetSpinner = {
  owner: " GR",                          // string
  grade: "Junior",                       // string
  "zip-code": 90210,                     // number
  price: 3.95,                           // number
  colors: ["red", "green", "purple"],    // array
  getChoice: function() { return this.owner + " bought " + this.colors[1]; }
};
console.log(fidgetSpinner.price);         // 3.95
console.log(fidgetSpinner.colors[2]));    // purple
console.log(fidgetSpinner.getChoice());   // GR bought green
console.log(fidgetSpinner["zip-code"]);   // 90210
console.log(fidgetSpinner.zip-code]);     // error
```

An object can have methods (function properties) that refer to itself as this can refer to the fields with .fieldName or ["fieldName"] syntax

JavaScript Objects and JSON

JSON is a way of saving or storing JavaScript objects.
(The technical term is "serializing" which is just a fancy word for turning an object into a savable string of characters)

Browser JSON methods:

- `JSON.parse( /* JSON string */ )` -- converts JSON string into Javascript object

- `JSON.stringify( /* Javascript Object */ )` -- converts a Javascript object into JSON text

JSON stringify/parse example

```
let point = {x: 1, y: 2, z: 3}
> point > {x: 1, y: 2, z: 3}
> let s = JSON.stringify(point);
> s
>  "{"x":1,"y":2,"z":3}"
> s = s.replace("1", "4");
> "{"x":4,"y":2,"z":3}"
> let point2 = JSON.parse(s);
> point2
> {x: 4, y: 2, z: 3}
```

What is JSON used for?

JSON data comes from many sources on the web:

- web services use JSON to communicate
- web servers store data as JSON files
- databases sometimes use JSON to store, query, and return data

JSON is the de facto universal format for exchange of data