JÖNKÖPING UNIVERSITY

School of Engineering

DOCUMENT OBJECT MODEL

Web Development with JavaScript and DOM

TWJK14 Spring 2017

Peter Larsson-Green

JÖNKÖPING UNIVERSITY
School of Engineering

EVENT DRIVEN PROGRAMMING

- Ordinary programs:
  - Execute statements from top to bottom.
  - Then the program has finish.
- Event driven programs:
  - Register listeners for events.
  - When event happens → call listeners for that event.
  - Typically used for GUIs.
  - The program is over when…?

JAVASCRIPT IS SINGLE THREADED

Only one part of our program is executed at a time.

- Heavy/long running computations will make our program lag ☹

Asynchronous programming to the rescue!

SYNCHRONOUS VS ASYNCHRONOUS

Synchronous

Wait for the result to be computed until you proceed.

AKA blocking.

Asynchronous

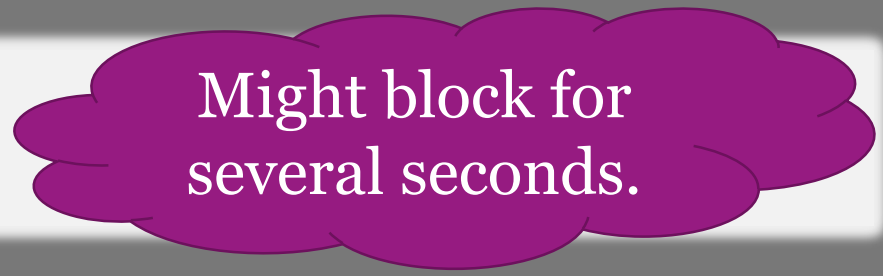
Do not wait for the result to be computed, just proceed.

The result is computed in the background.

Provide a callback that should be called when the result has been computed.

SYNCHRONOUS VS ASYNCHRONOUS

```
var content = getPage("ju.se")
// Do something with content.
```




```
getPage("ju.se", function(content){
  // Do something with content.
})
// Do other stuff while we wait for result.
```

SYNCHRONOUS VS ASYNCHRONOUS

```
var add = function(x, y){
  return x + y
}
var sum = add(4, 7)
console.log(sum)
```

```
var add = function(x, y, callback){
  callback(x + y)
}
add(4, 7, function(sum){
  console.log(sum)
})
```

BOM AND DOM

Browser Object Model.

- All resources the browser gives us access to (via JavaScript).
- No specification exists.
  - Some features specified along with HTML5.
- Includes the Document Object Model.
  - Access via the variable `document`.

SOME BOM FUNCTIONS

```
var timeoutId = setTimeout(function(){ }, 1000)
```

Browser will call the function after 1 second.

```
clearTimeout(timeoutId)
```

Cancel a `setTimeout` call.

```
var intervalId = setInterval(function(){ }, 1000)
```

Browser will call the function each second from now.

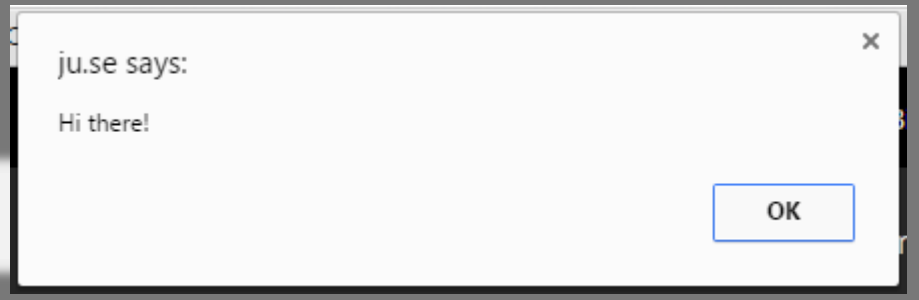
```
clearInterval(intervalId)
```

Cancel a `setInterval` call.

SOME BOM FUNCTIONS

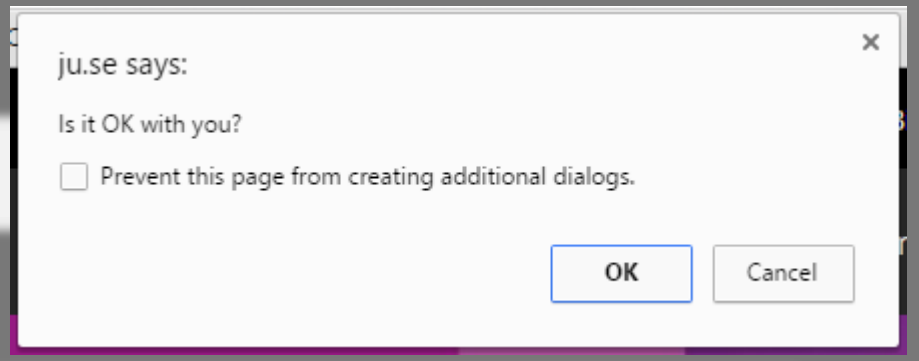
```
alert("Hi there!")
```

Shows a small box with a message.



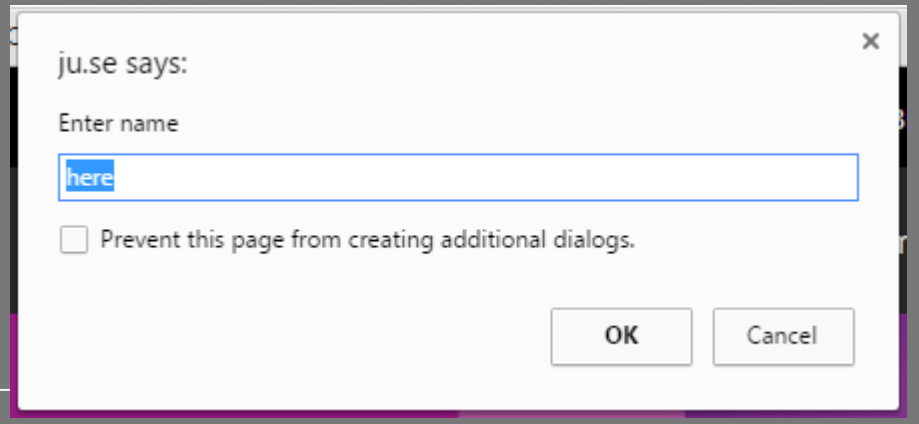

```
var ok = confirm("Is it OK with you?")
```

Shows a small box with a message.




```
var entered = prompt("Enter name", "here")
```

Lets the user enter a text.



These are all blocking!

DOM LEVELS/VERSIONS

1998: Document Object Model Level 1

2000: Document Object Model Level 2

2004: Document Object Model Level 3

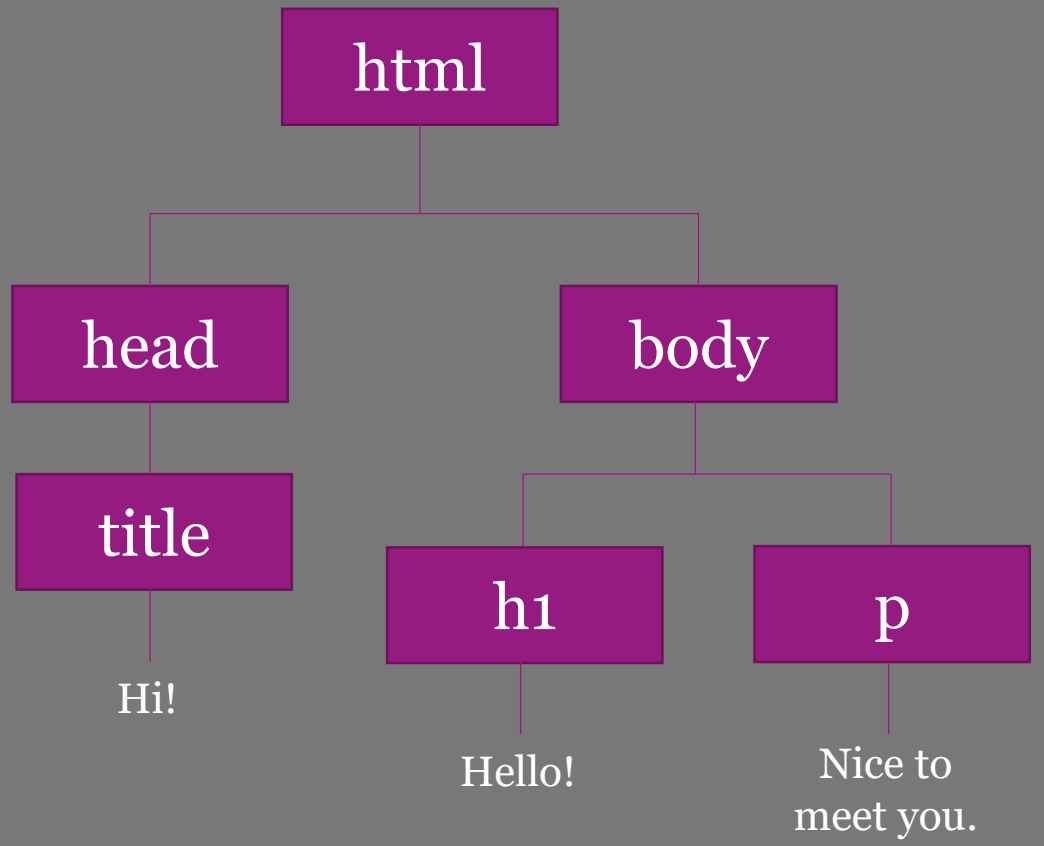
2015: W3C DOM4

Specification: https://www.w3.org/TR/domcore

Additional HTML features: https://www.w3.org/TR/html5/dom.html#dom

THE DOM TREE

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hi!</title>
  </head>
  <body>
    <h1>Hello!</h1>
    <p>Nice to meet you.</p>
  </body>
</html>
```



THE DOM TREE

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hi!</title>
  </head>
  <body>
    <h1>Hello!</h1>
    <p>Nice to meet you.</p>
  </body>
</html>
```

```
html
 |
 +-head
 |   |
 |   +-title - Hi!
 |
 +-body
     |
     +-h1 - Hello!
     |
     +-p - Nice to meet you.
```

THE `document` OBJECT

Contains information about the entire document.

Implements the `Document` interface: https://www.w3.org/TR/domcore/#interface-document

```
document.URL
document.documentElement
document.getElementsByTagName('tagName')
document.createElement('tagName')
```

The `Document` interface extends the `Node` interface.

THE NODE INTERFACE

Contains information about one node in the tree: https://www.w3.org/TR/domcore/#node

```
theNode.parentNode
theNode.childNodes
theNode.firstChild
theNode.lastChild
theNode.previousSibling
theNode.nextSibling
theNode.appendChild(newNode)
theNode.removeChild(oldNode)
```

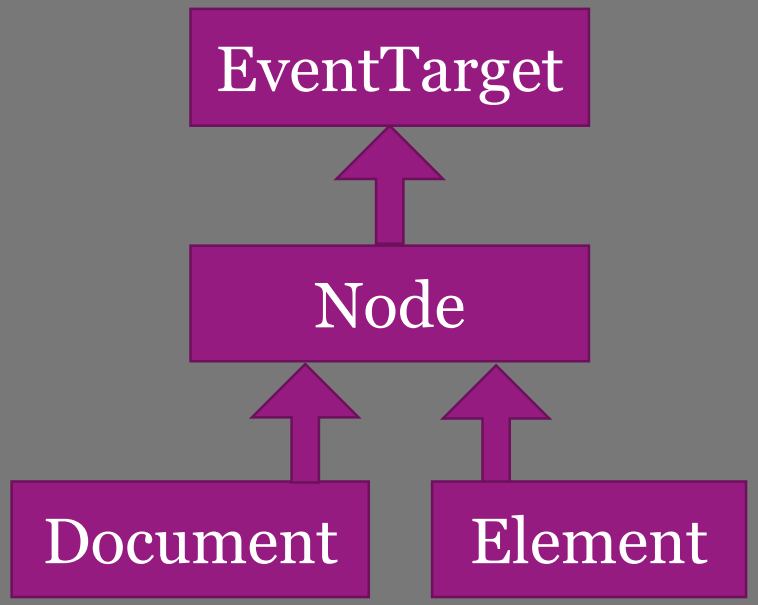
The `Node` interface extends the `EventTarget` interface.

THE EVENTTARGET INTERFACE

Used for adding event listeners: https://www.w3.org/TR/domcore/#eventtarget

```
theTarget.addEventListener("nameOfEvent", ...)
theTarget.removeEventListener("nameOfEvent", ...)
```

THE INTERFACES



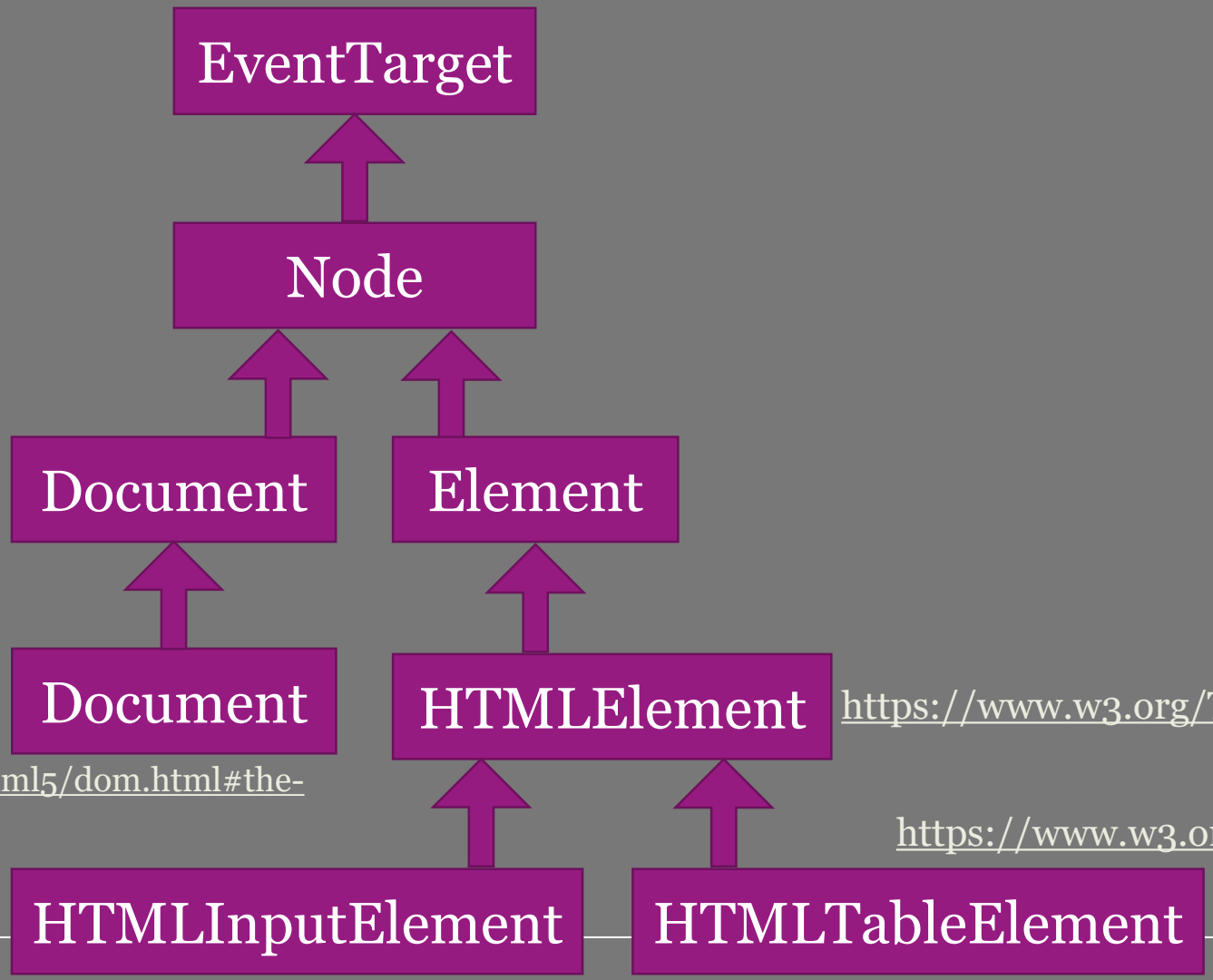
THE `ELEMENT` INTERFACE

Contains information about elements: https://www.w3.org/TR/domcore/#interface-element

```
theElement.tagName
theElement.id
theElement.getAttribute("name")
theElement.setAttribute("name", "value")
theElement.removeAttribute("name")
theElement.hasAttribute("name")
theElement.getElementsByTagName("tagName")
theElement.getElementsByClassName("className")
```

The `Element` interface extends the `Node` interface.

THE INTERFACES



https://www.w3.org/TR/html5/dom.html#htmlelement

https://www.w3.org/TR/html5/dom.html#the-document-object

https://www.w3.org/TR/html5

EXAMPLE

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hi!</title>
    <script src="my-file.js">
    </script>
  </head>
  <body>
    <p id="p">Boring text.</p>
  </body>
</html>
```

my-file.js

```
var p = document.getElementById("p")
p.innerText = "Fun text!"
```

The code is executed before the `<body>` element has been parsed.

```
document.addEventListener(
  "DOMContentLoaded",
  function(){
    var p = document.getElementById("p")
    p.innerText = "Fun text!"
  }
)
```

EXAMPLE

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hi!</title>
    <script src="my-file.js">
    </script>
  </head>
  <body>
    <button id="b"></button>
  </body>
</html>
```

my-file.js

```
var counter = 0
document.addEventListener(
  "DOMContentLoaded",
  function(){
    var b = document.getElementById("b")
    b.innerText = counter
    b.addEventListener(
      'click',
      function(){
        counter++
        b.innerText = counter
      }
    )
  }
)
```

EXAMPLE

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hi!</title>
    <script src="my-file.js">
    </script>
  </head>
  <body>
    <button id="b"></button>
  </body>
</html>
```

my-file.js

```
var counter = 0
document.addEventListener(
  "DOMContentLoaded",
  function(){
    var b = document.getElementById("b")
    b.innerText = counter
    setInterval(
      function(){
        counter++
        b.innerText = counter
      },
      1000
    )
  }
)
```

MORE ABOUT EVENTS

There exists many of them:

- https://developer.mozilla.org/en-US/docs/Web/Events#Standard_events

Not all elements support all events, but some common:

- `DOMContentLoaded` (for the `document` object).
- `keydown`, `keypress`, `keyup` (for elements that can have focus).
- `click`, `mousemove`, `mouseenter` (for elements that are shown).
- `reset`, `submit` (for `<form>`).

NESTING EVENT LISTENERS

```
<p id="p">
  How
  <span id="s">are</span>
  you?
</p>
```

What happens when "are" is clicked?

- The click event is fired on both elements.

In which order?

- IE<9: Bubbling.
- Netscape: Capturing.

```
document.getElementById('p').addEventListener('click', function(){
  console.log("Click on paragraph!")
})
document.getElementById('s').addEventListener('click', function(){
  console.log("Click on span!")
})
```

BUBBLING VS CAPTURING

Bubbling

The event is fired on the innermost element first, and then propagates up the element tree to the outermost element.

Capturing

The event is fired on the outermost element first, and then propagates down the element tree to the innermost element.

W3's event model

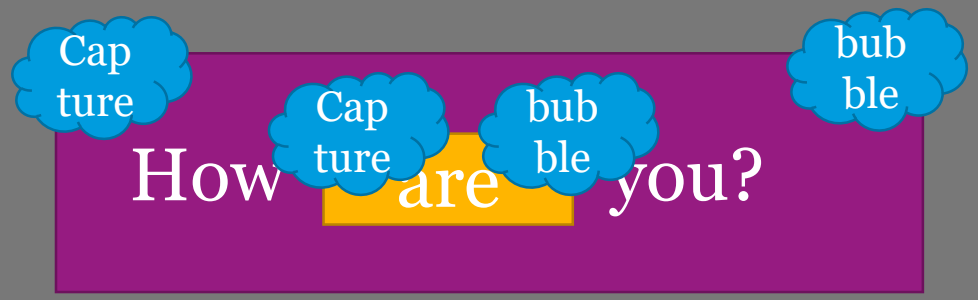
Support both:

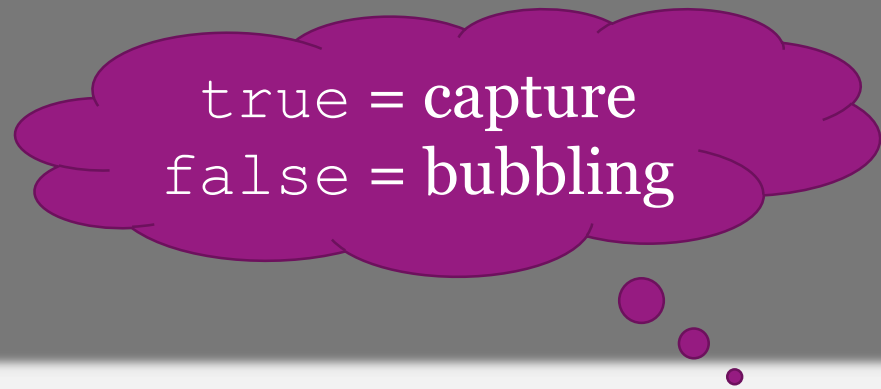
First capture phase, then bubble phase.

Specify phase in `addEventListener`.

NESTING EVENT LISTENERS

```
<p id="p">How <span id="s">are</span> you?</p>
```






```
theElement.addEventListener('click', function(){ }, true)
```

NESTING EVENT LISTENERS

```
<p id="p">How <span id="s">are</span> you?</p>
```

```
document.getElementById('p').addEventListener('click', function(){
  console.log("Click on paragraph (capture).")
}, true)
document.getElementById('p').addEventListener('click', function(){
  console.log("Click on paragraph (bubbling).")
}, false)
document.getElementById('s').addEventListener('click', function(){
  console.log("Click on span (capture).")
}, true)
document.getElementById('s').addEventListener('click', function(){
  console.log("Click on span (bubbling).")
}, false)
```

THE EVENT OBJECT

Event callbacks are called with an event object as argument:

```
theElement.addEventListener('click', function(event){
  // Do something!
})
```

The event object contains:

- information about the event.
- methods to control the event.

THE EVENT OBJECT

Event callbacks are called with an event object as argument:

```
theElement.addEventListener('click', function(event){
  // Do something!
})
```

Some information:

`event.clientX`, `event.clientY` and `event.button` (for mouse events).

`event.altKey`, `event.ctrlKey`, `event.keyCode` (for keyboard events).

`event.target` (the innermost element)

`event.currentTarget` (the element you called `addEventListener` on).

THE EVENT OBJECT

Event callbacks are called with an event object as argument:

```
theElement.addEventListener('click', function(event){
  // Do something!
})
```

Some methods:

`event.preventDefault()` (prevent forms from being submitted and links from being followed).

`event.stopPropagation()` (stop this event instance from firing on any more elements).

STYLING THROUGH JAVASCRIPT

Each HTML element object has a `style` property.

- Can be used to change CSS properties.

CSS	**JavaScript**
`property-name: value`	`theElement.style.propertyName = "value"`

```
<p id="p">Click me!</p>
```

```
var p = document.getElementById('p')
p.addEventListener('click', function(){
  p.style.fontSize = (5+Math.random()*30)+"px"
})
```

STYLING THROUGH JAVASCRIPT

Each HTML element object has a `style` property.

- Can be used to change CSS properties.

CSS	**JavaScript**
`property-name: value`	`theElement.style.propertyName = "value"`

JavaScript is not about styling; avoid using it. Instead:

- Write CSS rules with the class selector.
- Use JavaScript to dynamically add/remove classes.

```
theElement.classList.add("classNameToBeAdded")
theElement.classList.remove("classNameToBeRemoved")
```

COOKIES

Can be used to store data on the client (between page visits).

- Can be created by the server.
- Are sent to the server with each request.
- Can be created by the client:

```
document.cookie = "name=value"
```


```
document.cookie = "name=value;expires=Thu, 17 May 2016 00:00:00 GMT"
```

- Can be read by the client:

```
var cookieString = document.cookie
// cookieString = "name=value; name2=value2; ..."
```

COOKIES EXAMPLES

```
document.cookie = "a=x" // Create first cookie.
document.cookie = "b=y" // Create second cookie.

var cookieString = document.cookie // "a=x; b=y"
var cookies = {} // {a: 'x', b: 'y'}

var cookieStrings = cookieString.split("; ")
for(var i=0; i<cookieStrings.length; i++){
  var cookieParts = cookieStrings[i].split("=")
  cookies[cookieParts[0]] = cookieParts[1]
}
```

THE COOKIE LAW

An EU directive to protect visitors privacy.

- The user must approve you storing/retrieving information on the client.
- Does not only apply to cookies!

More information about the directive:

- https://cookiepedia.co.uk/eu-cookie-law

More information from Post- och telestyrelsen in Sweden:

- http://www.pts.se/sv/Bransch/Regler/Lagar/Lag-om-elektronisk-kommunikation/Cookies-kakor/Fragor-och-svar-om-kakor-for-webbplatsinnehavare

SESSIONSTORAGE & LOCALSTORAGE

HTML5 features to store data on the client.

- `localStorage` stores data forever.
- `sessionStorage` stores data only for the session.

```
theStorage.setItem("key", "value")
var value = theStorage.getItem("key")
theStorage.removeItem("key")
theStorage.clear()
```

THE `DATA-*` ATTRIBUTE

Can we add custom attributes to HTML elements?

- Traditional answer: No.
- HTML5 answer: Yes, by using the `data-*` attribute.

```
<p data-the-console="NES">Some text about a video game...</p>
```

Access it from JavaScript:

```
var p = document.querySelector("p")
var console = p.dataset.theConsole
```

```
var console = p.getAttribute("data-the-console")
```

RECOMMENDED READING

W3Schools:

- JavaScript Forms:
  - https://www.w3schools.com/js/js_validation.asp
- JS HTML DOM:
  - https://www.w3schools.com/js/js_htmldom.asp
- JS Browser BOM:
  - https://www.w3schools.com/js/js_window.asp

W3C DOM4 Specification:

- https://www.w3.org/TR/domcore

The Cookie Law Explained:

- https://www.cookielaw.org/the-cookie-law/