Unit 2 - HTML5, JQuery And Ajax

Class notes by Vibha Masti

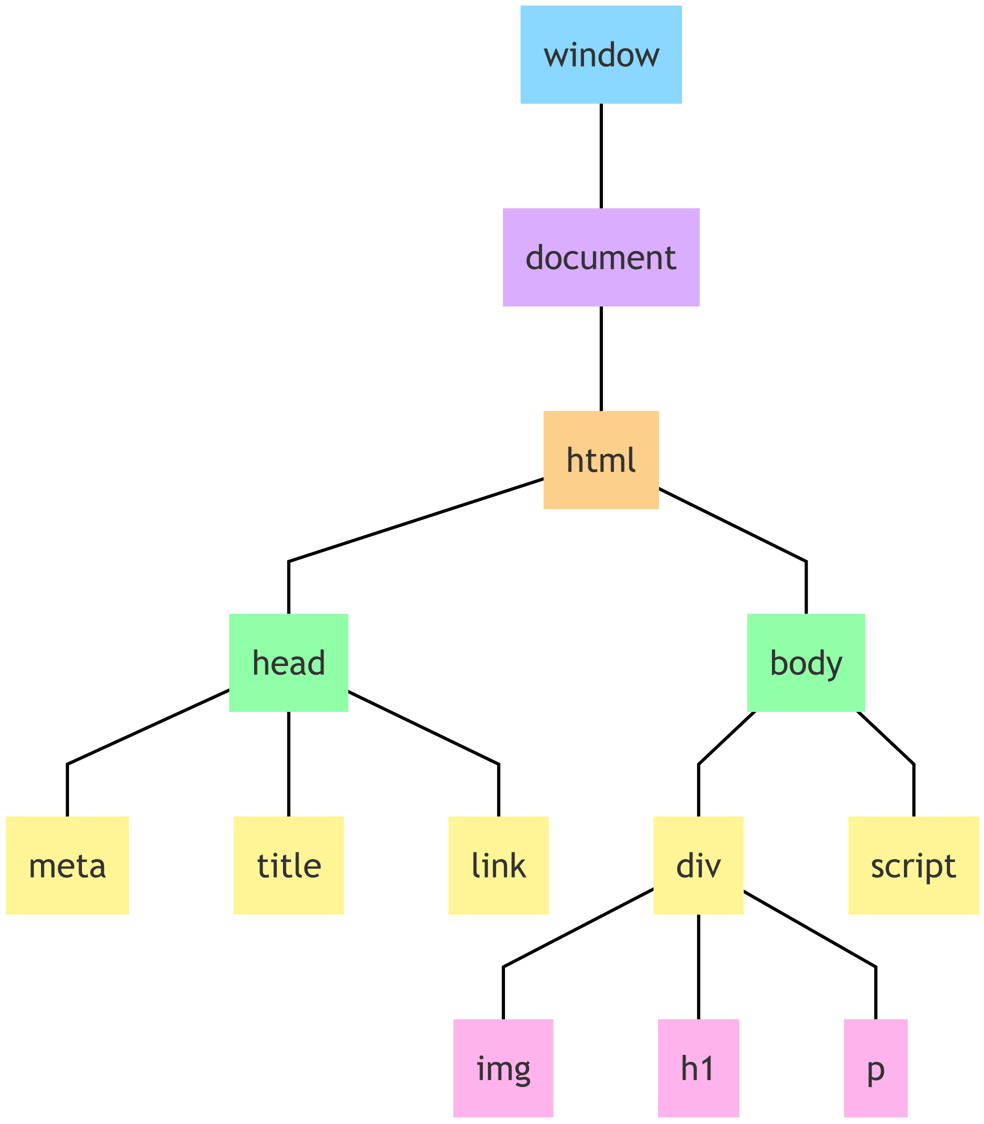
Feedback/corrections: vibha@pesu.pes.edu

0.0 Table of Contents

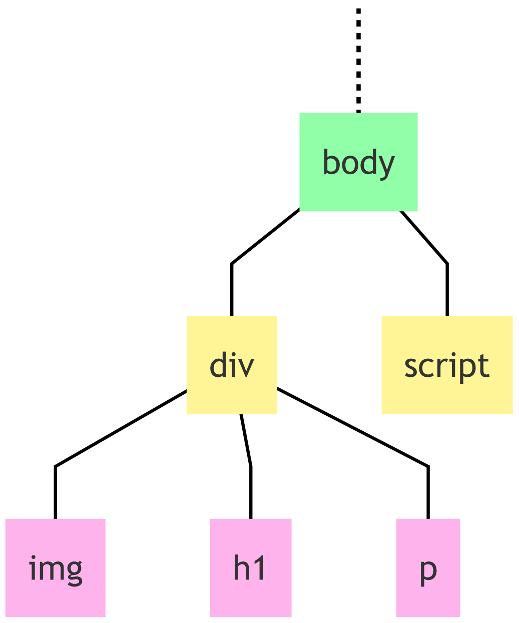
1.0 Document Object Model (DOM)

- Drawbacks of `document.write()`: executed after the page has finished loading and overwrites the page
- Practically only appends
- The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript

HTML Document structure



- DOM elements are relative to the `body`

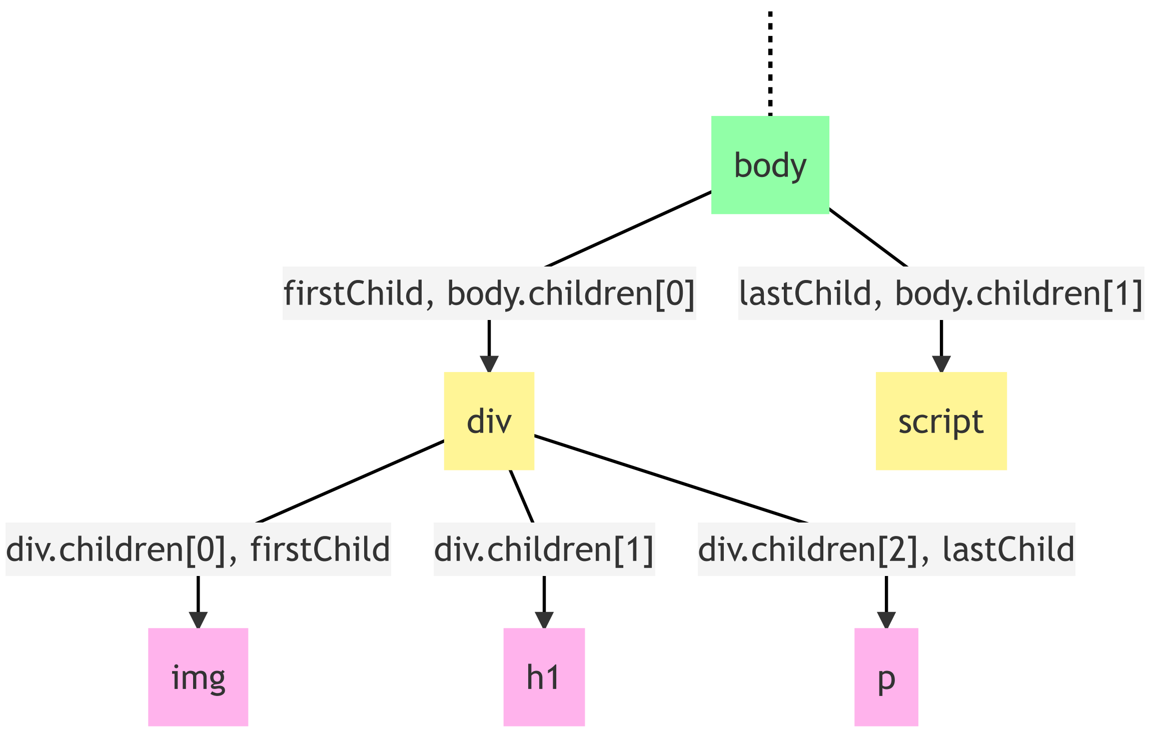


Accessing DOM Elements

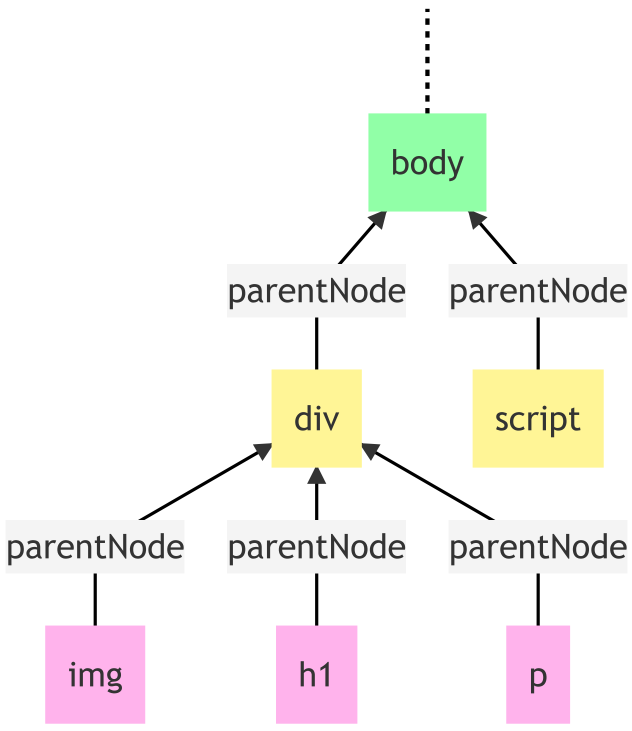
Access Element By	Equivalent Selector	Method
ID	`#demo`	`getElementByID('demo')`
Class	`.demo`	`getElementsByClassName('demo')`
Tag	`<tag_name>` like `<p>`	`getElementsByTagName('p')`
Selector (single)	Any CSS Selector	`querySelector('selector')`
Selector (all)	Any CSS Selector	`querySelectorAll('selector')`

Traversing the DOM

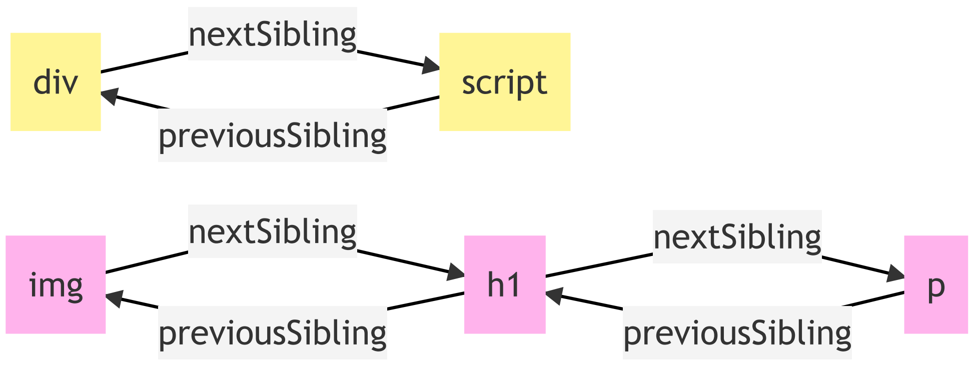
- Child relationship


body
firstChild, body.children[0]
lastChild, body.children[1]
div
script
div.children[0], firstChild
div.children[1]
div.children[2], lastChild
img
h1
p

- Parent relationship



- Sibling relationship



Creating Element Objects

- `document.createElement()` - create a new element using tag
- `document.createTextNode()` - create a new text node

Node properties and methods

- `node.textContent` or `node.innerText` - get or set the text content of an element node (without HTML tags)
- `node.innerHTML` - get or set the HTML content enclosed in the element tag
- `node.appendChild()`, `node.insertBefore()`, `node.replaceChild()`, `node.removeChild()`, `node.remove()` etc (read docs)

Code Example

- Note: `<script>` tags in the head of the document: the document isn't yet populated with the hierarchy of DOM objects yet
- Solution 1: add `<script>` tags after elements - not elegant
- Solution 2: add an `init()` function `onload`
- Must define the `init()` function in `<script>`

In the HTML file

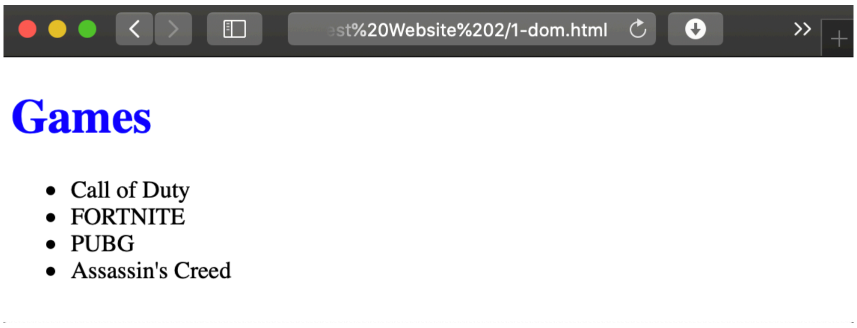
```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM Manipulation</title>
    <script type="text/javascript" src="1-dom.js">

    </script>
</head>
<!-- Add onload="init()" for DOM access in script tags
Once the page has been loaded, the hierarchy is available
(must define function init() in script)
-->
<body onload="init()">
    <h1>Games</h1>
    <ul>
        <li>Call of Duty</li>
        <li class="g1">Fortnite</li>
        <li class="g1">PUBG</li>
    </ul>

</body>
</html>
```

In the JavaScript file (try it out by un-commenting)

```
function init() {
    h1 = document.querySelector("h1");
    h1.style.color = "blue";

    list = document.querySelectorAll("li.g1");
    for (i=0; i<list.length; i++) {
        /* Convert to uppercase */
        list[i].innerText = list[i].innerText.toUpperCase();
    }

    /* Add new element to DOM */
    new_li = document.createElement('li');
    new_li.innerText = "Assassin's Creed";

    /* Select the ul element */
    ul1 = document.querySelector('ul');

    /* Add element to end (appendChild) - uncomment this */
    // ul1.appendChild(new_li);

    /* Can also use index to insert before - uncomment this */
    // ul1.insertBefore(new_li, list[0])

    /* Can also do - uncomment this */
    // ul1.insertBefore(new_li, ul1.firstChild);

    /* To remove an element - uncomment this */
    // list[0].remove();

    /* OR - uncomment this */
    // ul1.removeChild(ul1.children[1]);

    /* To replace an element - uncomment this */
    // list[0].parentNode.replaceChild(new_li, ul1.children[2]);
}
```

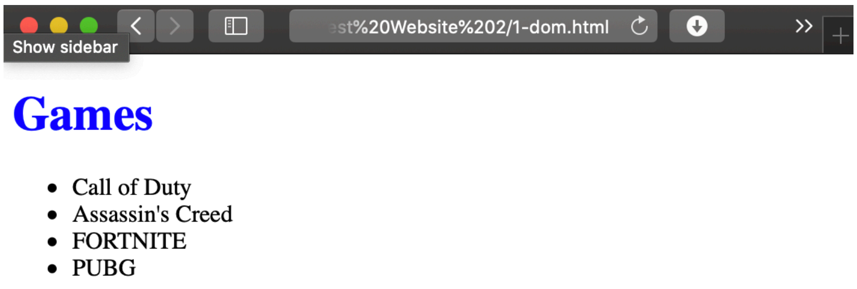
Append to end



Games

- Call of Duty
- FORTNITE
- PUBG
- Assassin's Creed

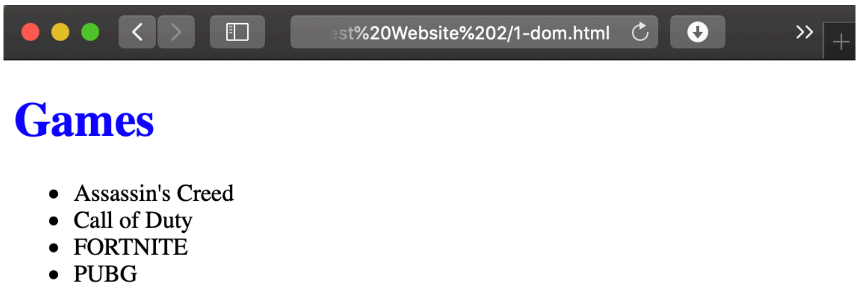
Insert before the first `g1`



Games

- Call of Duty
- Assassin's Creed
- FORTNITE
- PUBG

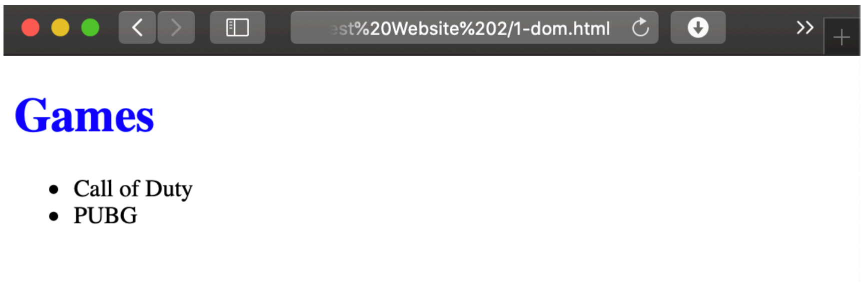
Insert before the first child



Games

- Assassin's Creed
- Call of Duty
- FORTNITE
- PUBG

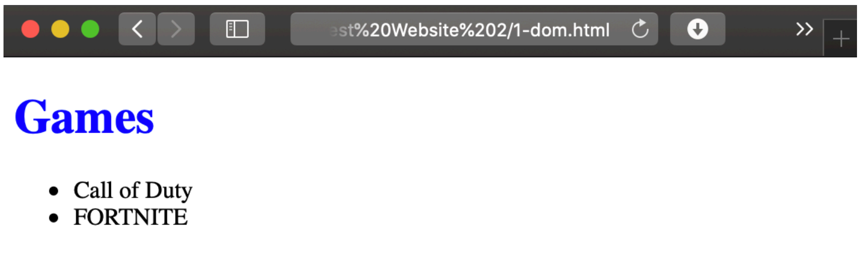
Remove `list[0]`



Games

- Call of Duty
- PUBG

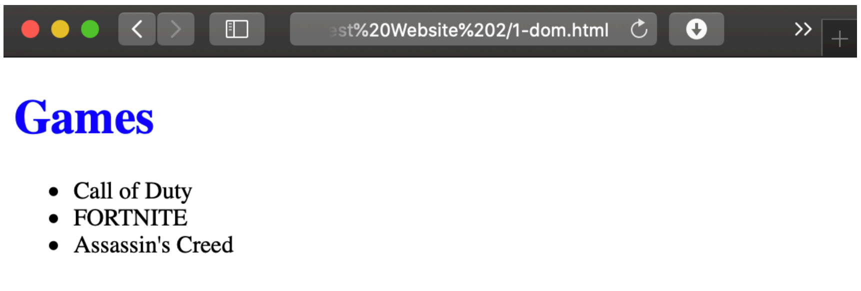
Remove the child at index 2



Games

- Call of Duty
- FORTNITE

Replace child at index 2 with the new element



Games

- Call of Duty
- FORTNITE
- Assassin's Creed

2.0 Events

- Events are created by activities associated with specific HTML elements
- The process of connecting an event handler to an event is called registration
- There are distinct approaches to event handler registration:
  - Inline event handlers
  - Event handler property
  - Event listeners

Inline Event Handlers

- Callback function

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Events</title>

    <script type="text/javascript" src="2-js-events.js">

    </script>
</head>

<body onload="init()">

    <div>
        <!-- Add in HTML -->
        <p onclick="handler(event)">Popular games</p>
        <ul>
            <li>Call of Duty</li>
            <li>Fortnite</li>
            <li>PUBG</li>
        </ul>
    </div>

</body>
</html>
```

Event Property

- Callback function
- Can also be an anonymous function

```javascript
function init() {
    list = document.querySelectorAll("li");

    for (let i in list) {
        list[i].onclick = handler;
    }

    console.log(document.body.children[0].children[0].onclick);
}

```

```
function handler(event) {
    // For when parameters are not passed - depends on browser
    ev = event || window.event;
    console.log(ev.target.innerHTML);
    ev.target.style.color = "blue";
    ev.preventDefault();
}
```

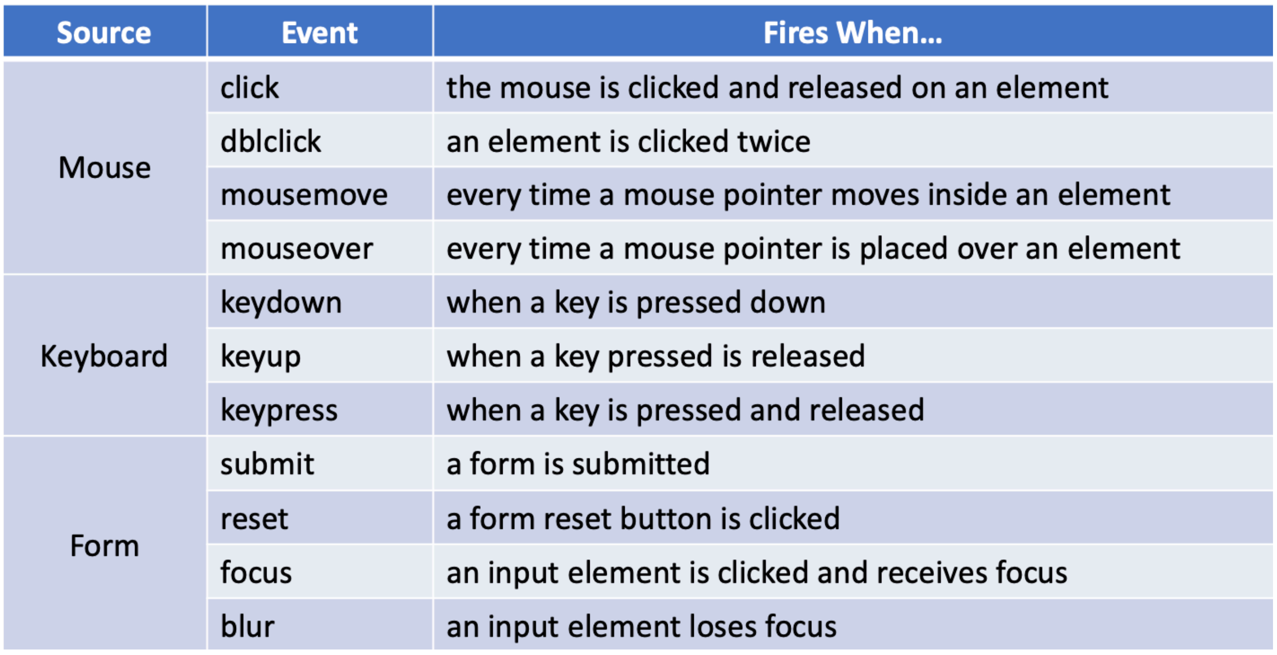
Event Listener

- An event listener watches for an event on an element
- `element.addEventListener(event, handler)`

```
function init() {
    list = document.querySelectorAll("li");

    document.querySelector("p").addEventListener("click", function(event)
{
        event.target.style.color = 'green';
        event.target.innerHTML = "I was clicked";
    })

    console.log(document.body.children[0].children[0].onclick);
}

function handler(event) {
    // For when parameters are not passed - depends on browser
    ev = event || window.event;
    console.log(ev.target.innerHTML);
    ev.target.style.color = "blue";
    ev.preventDefault();
}
```

Event Sources and Events

- The `event` object holds context such as `event.target`, `event.type` etc (read docs)



Source	Event	Fires When...
Mouse	click	the mouse is clicked and released on an element
	dblclick	an element is clicked twice
	mousemove	every time a mouse pointer moves inside an element
	mouseover	every time a mouse pointer is placed over an element
Keyboard	keydown	when a key is pressed down
	keyup	when a key pressed is released
	keypress	when a key is pressed and released
Form	submit	a form is submitted
	reset	a form reset button is clicked
	focus	an input element is clicked and receives focus
	blur	an input element loses focus

3.0 Event Propagation

- When an element on the page is clicked (for eg, a button), not only the button is being clicked but also the button’s container, the div, and the whole webpage
- Event flow explains the order in which events are received on the page from the element where the event occurs and propagated through the DOM tree
- There are two main event models: **event bubbling** and **event capturing**

Event Bubbling

- In the event bubbling model, the click event first occurs on the element that was clicked
- The click event then goes up the DOM tree, firing on each node along its way until it reaches the document object

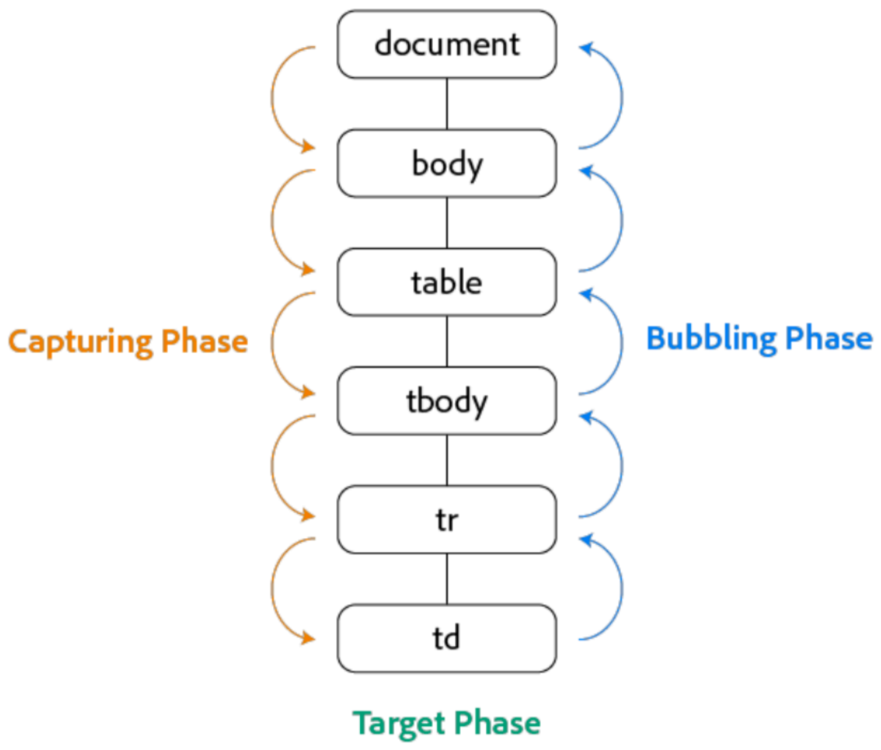
Event Capturing

- In the event capturing model, an event starts at the least specific element and flows downward toward the most specific element (element that was clicked)

DOM Level 2 Event Flow

- DOM level 2 events specify that event flow has three phases
- The three phases in which an event can propagate to handlers defined in parent elements are
    - Capturing phase
    - Target phase
    - Bubbling phase
- `element.addEventListener("event", func_ref, flag);`

- if `flag` = `true`, handler registered for capturing phase
- if `flag` = `false`, handler registered for bubbling phase

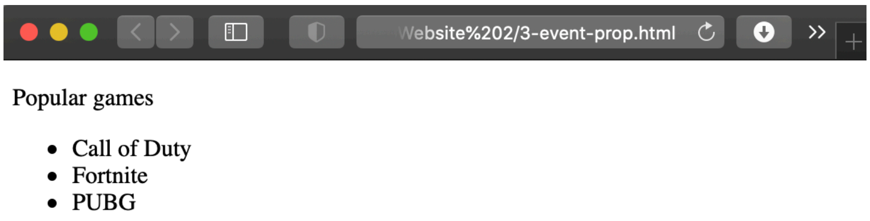


Example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Event Propagation</title>

    <script type="text/javascript" src="3-event-prop.js">

    </script>
</head>

<body onload="init()">

    <div id='div'>

        <!-- Click event propagates from div to li -->
        <p onclick="handler(event)">Popular games</p>
        <ul id='ul'>
```

```
            <li id='li'>Call of Duty</li>
            <li>Fortnite</li>
            <li>PUBG</li>
        </ul>
    </div>

</body>
</html>
```

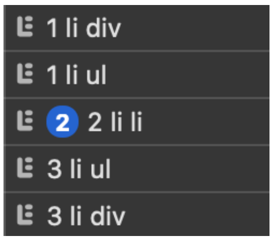
```
/*
Capturing phase, Bubbling phase, Target phase

div -> p -> ul -> li
*/

function init() {
    // false by default (bubbling)

    // Will be called first - true for capturing phase
    document.querySelector("#div").addEventListener("click", handler, true);
    document.querySelector("#ul").addEventListener("click", handler, true);

    // target - true or false
    document.querySelector("#li").addEventListener("click", handler, true);

    // called in the end - false for bubbling
    document.querySelector("#div").addEventListener("click", handler, false);
    document.querySelector("#ul").addEventListener("click", handler, false);

    // target - true or false
    document.querySelector("#li").addEventListener("click", handler, false);

}

function handler(event) {
    // event.eventPhase -> 0, 1 or 2
    console.log(event.eventPhase + ' ' + event.target.id + ' ' + event.currentTarget.id);
    // event.stopPropagation();
    // event.cancelBubble = true;
}
```



Popular games

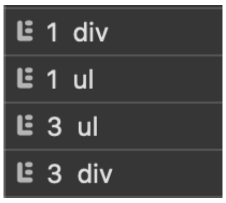
- Call of Duty
- Fortnite
- PUBG

When Call of Duty is clicked



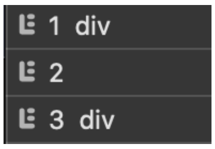
```
1 li div
1 li ul
2 2 li li
3 li ul
3 li div
```

When Fortnite is clicked



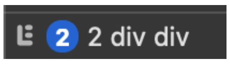
```
1  div
1  ul
3  ul
3  div
```

When Popular games is clicked



```
1  div
2
3  div
```

When an area in the `<div>` is clicked



```
2 2 div div
```

4.0 XML and JSON

- XML - extensible markup language
- JSON - JavaScript object notation
- JSON is just a data format whereas XML is a markup language

XML

```xml
<empinfo>
    <employees>
        <employee>
            <name>Daenerys Targaryen</name>
            <age>17</age>
        </employee>
        <employee>
            <name>Jon Snow</name>
            <age>20</age>
        </employee>
        <employee>
            <name>Robert Baratheon</name>
            <age>46</age>
        </employee>
    </employees>
</empinfo>
```

JSON

```json
{
    "empinfo": {
        "employees": [
            {
                "name": "Daenerys Targaryen",
                "age": 17
            },
            {
                "name": "Jon Snow",
                "age": 20
            },
            {
                "name": "Robert Baratheon",
                "age": 46
            }
        ]
    }
}
```

XML

Converting to Object Hierarchy

In the HTML file

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>XML and JSON</title>
</head>
<body>
    <h1>XML and JSON</h1>
    <p id="xml-demo"></p>
    <script src="4-xml-json.js">
    </script>
</body>
</html>
```

In the `4-xml-json.js` file

```javascript
var xmlText;

xmlText = "<bookstore>" +
            "<book>" +
                "<title>Everyday Italian</title>" +
                "<author>Giada Laurentiis</author>" +
                "<year>2005</year>" +
            "</book>" +
        "</bookstore>";

/* Convert to object hierarchy - DOM parser */

var parser = new DOMParser();

/* Convert string to object hierarchy */
xmlDOM = parser.parseFromString(xmlText, "text/xml");
xmlTitle = xmlDOM.getElementsByTagName("title")[0];
xmlTitle.childNodes[0].nodeValue += " - Modified";

document.getElementById("xml-demo").innerText =
xmlTitle.childNodes[0].nodeValue;

/* Convert object hierarchy to string - serializer for req/res */
var xmlTextSerializer = new XMLSerializer();
xmlString = xmlTextSerializer.serializeToString(xmlDOM);

```

```
console.log(xmlString);
```

Simple Server

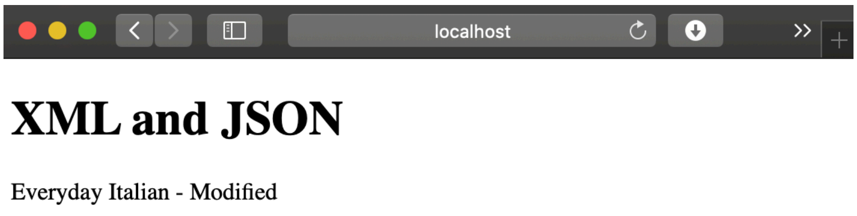
- Note: to start a simple server using python, run the following command (Terminal on Mac, Command Prompt on Windows)
- By default, it opens on port `8000`
- Go to http://localhost:8000 to find the files on the server

```
python -m http.server
```

- To quit the server, execute `CTRL-C` (Windows and Mac)
- You could also use XAMPP

Output

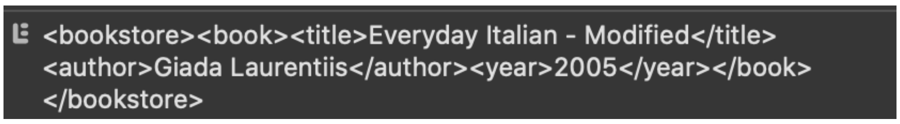
Rendered in a browser



XML and JSON

Everyday Italian - Modified

Console



```
<bookstore><book><title>Everyday Italian - Modified</title>
<author>Giada Laurentiis</author><year>2005</year></book>
</bookstore>
```

JSON

Converting to Object Hierarchy

In the HTML file

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>XML and JSON</title>
```

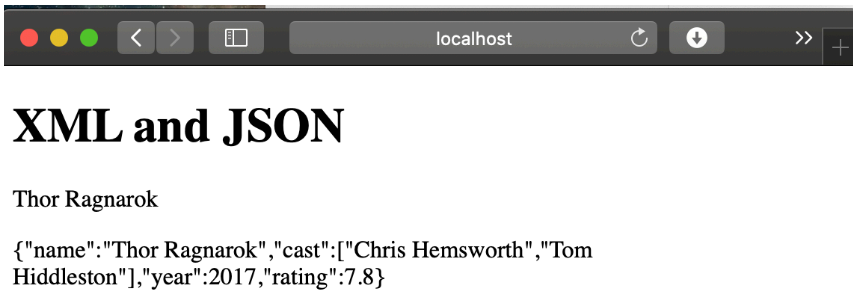
```html
</head>
<body>
    <h1>XML and JSON</h1>
    <p id="json-demo-1"></p>
    <p id="json-demo-2"></p>
    <script src="4-xml-json.js">
    </script>
</body>
</html>
```

In the `4-xml-json.js` file

```javascript
// JSON text always uses double quotes for the keys and
// values, not single quotes
var jsonText;
jsonText = '{"name": "Thor Ragnarok", "cast": ["Chris Hemsworth", "Tom Hiddleston"]}';

// Using JSON.parse() to convert string to obj
jsonObj = JSON.parse(jsonText);
document.querySelector("#json-demo-1").innerText = jsonObj.name;

console.log(jsonObj.cast)
jsonObj.year = 2017;
jsonObj.rating = 7.8;

// Using JSON.stringify() to convery obj to string

var newJsonStr = JSON.stringify(jsonObj);
document.querySelector("#json-demo-2").innerText = newJsonStr;

// jsonObj.toString() -> can be sent to server
```

Output

Rendered in a browser

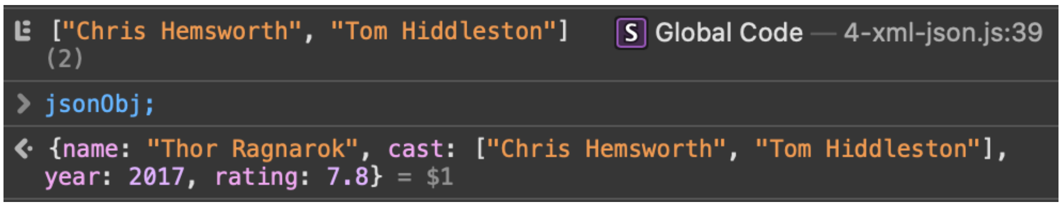


XML and JSON

Thor Ragnarok

```
{"name":"Thor Ragnarok","cast":["Chris Hemsworth","Tom
Hiddleston"],"year":2017,"rating":7.8}
```

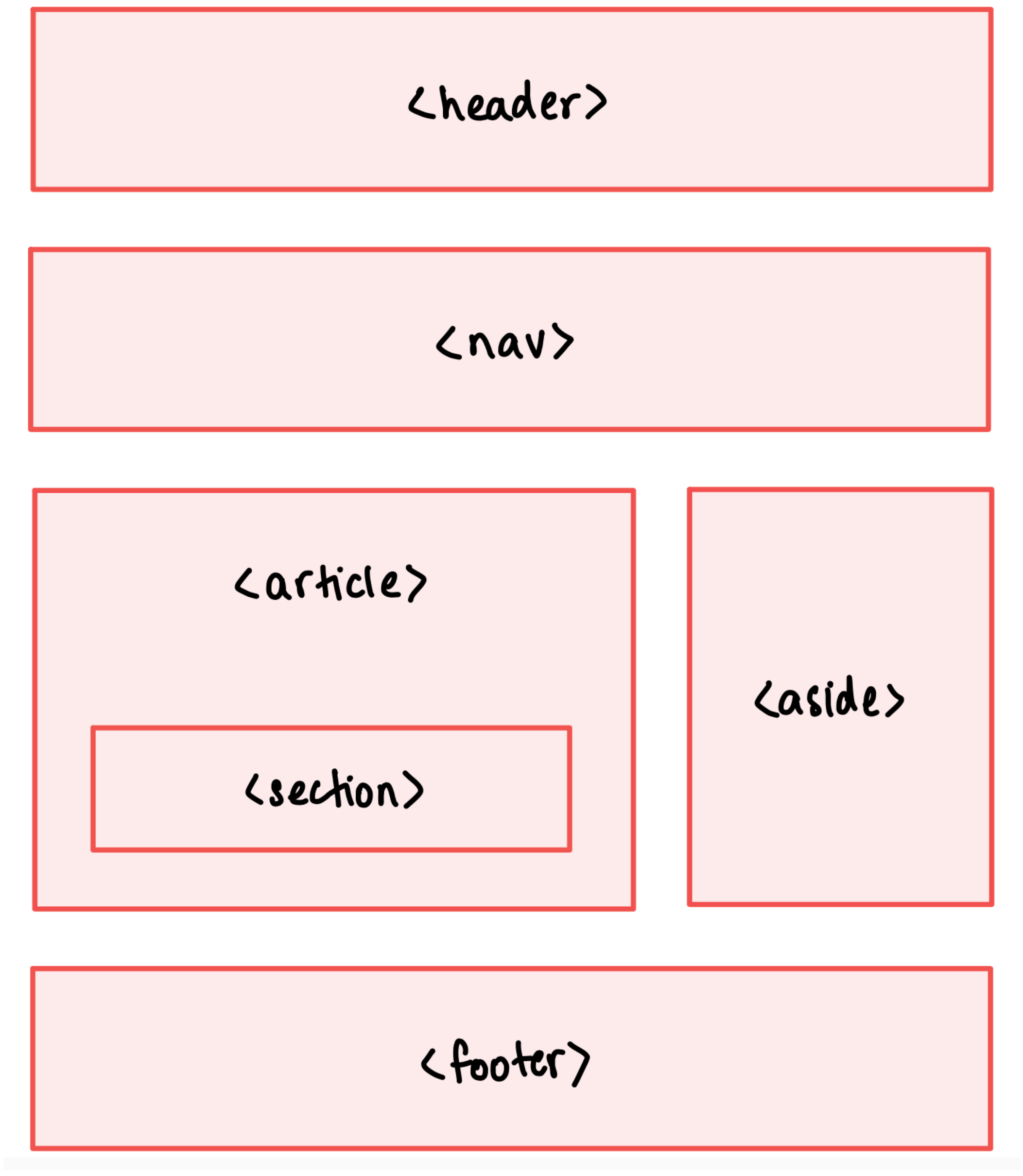
Console



```
["Chris Hemsworth", "Tom Hiddleston"]        Global Code — 4-xml-json.js:39
(2)
> jsonObj;
< {name: "Thor Ragnarok", cast: ["Chris Hemsworth", "Tom Hiddleston"],
  year: 2017, rating: 7.8} = $1
```

5.0 HTML 5

- New input types and properties
- HTML5 has added a little meaning (semantic) and identifiers to its elements, so web developers can use them wisely in their web pages
- **New input types**
  - `email`: email address
  - `number`: spinbox
  - `range`: slider
  - `url`: web addresses
  - `color`: color pickers
  - `search`: search boxes
  - `date`: date
  - `time`: time
  - `file`: input file selection
- **New input properties**
  - `placeholder`
  - `required`
  - `pattern`
  - `autofocus`
- **New structural HTML5 tags**



Basic HTML5 Document

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML Tags</title>
</head>
<body>
    <!-- Semantic tags - easier for
        programs/humans to understand-->
    <header>
        <h1>Logo Here</h1>
    </header>

    <nav><ul>
```

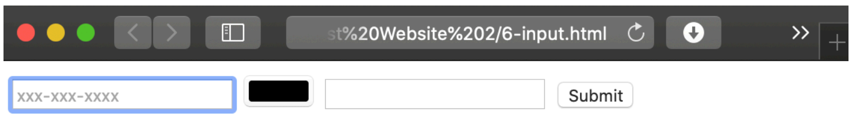
```html
            <li>Link 1</li>
            <li>Link 2</li>
            <li>Link 3</li>
        </ul></nav>

        <article>
            <section></section>
        </article>

</body>
</html>
```

Input tags

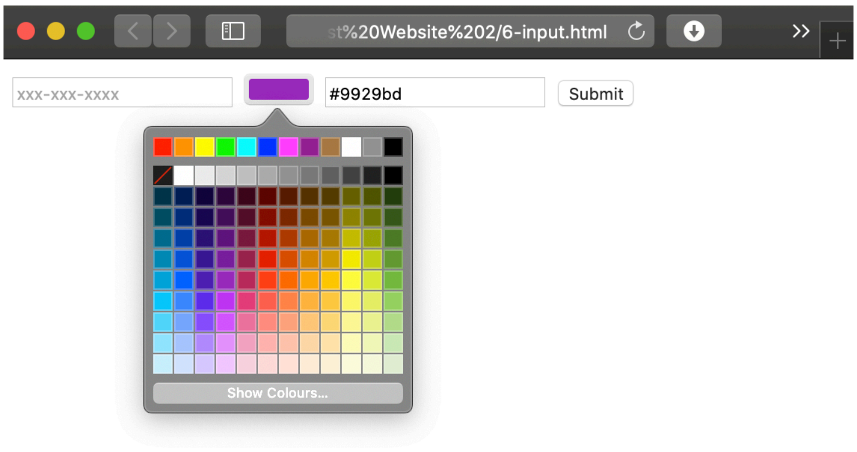
- `tel` for telephone numbers
- Placeholders, regex, name
- Color pickers

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML5 Inputs</title>
    <script>
        function setText() {
            let col = document.querySelector("#colcode").value;
            document.querySelector("#coltext").value = col;
        }

        function setColor() {
            let text = document.querySelector("#coltext").value;
            document.querySelector("#colcode").value = text;
        }
    </script>
</head>
<body>
    <form action="save.py" method="GET">
        <!-- Placeholders, required, regex, name -->

        <!-- Telephone number -->

        <input type="tel" autofocus required placeholder="xxx-xxx-xxxx"
pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}" name="tel"></input>

        <!-- Color picker -->

        <input id="colcode" onchange="setText()" type="color"
name="ucolor"></input>

        <!-- Text element -->
        <input id="coltext" onchange="setColor()" type="text"
id="coltext"></input>
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

Placeholders with format



Color picker



Audio

- Standard src - use .ogg and .mp3
- Avoid unnecessary download, use `<source/>` tags, instead of putting under `<audio></audio>` tags
- `<source src="song.mp3" type="audio/mp3"/>`
- Default message for no support

```html
<audio preload controls="controls">
    <source src="audio/Flip.mp3" type="audio/mp3"/>
</audio>
```

Video

- Standard src - .ogg, .mp4 and webm
- Use `<video></video>` and `<source/>` tags
- Controls for pause, play
- Default message
- width, height

```html
<video id="v1" autoplay controls="controls" height="200px" width="400px"
muted="muted">
    <source type="video/mp4" src="video/infinity.mp4" type="video.mp4"/>
</video>
```

Progress

- Progress bar for task completion (no need extensive JS and CSS)

```html
<label for="prog">Progress</label>
<progress id="prog" value="0" max="100"></progress>
```

6.0 Canvas and SVG

Canvas

- Uses JavaScript to draw graphics on a web page
- Rectangular area of specified dimensions

Syntax

```html
<canvas id="myCanvas" width="200" height="100">
    Canvas is not supported
</canvas>
```

- The canvas element has no drawing abilities of its own
- All drawing must be done inside a JavaScript using the context object

Code Implementation

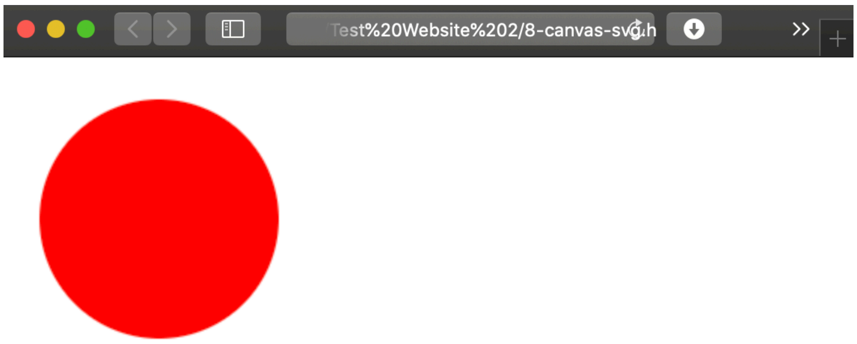
HTML file

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Canvas and SVG</title>
    <!-- Canvas, Scalable vector graphics (can zoom) -->

    <script type="text/javascript" src="8-canvas-svg.js"></script>
</head>
<body onload="init()">
    <canvas id="c1" width="400px" height="200px">
        No canvas support
    </canvas>
</body>

</html>
```

JS file

```javascript
function init() {
    c1 = document.querySelector("#c1");

    /* Context object for 2D animation */
    ctx = c1.getContext("2d");

    ctx.fillStyle = "#FF0000";

    // Syntax
    // ctx.arc(centerx, centery, radius, startangle, endangle, direction);

    // Clockwise semicircle
    // ctx.arc(100, 100, 80, 0, Math.PI, 0);
    // Anticlockwise semicircle
    // ctx.arc(100, 100, 80, 0, Math.PI, 1);

    ctx.arc(100, 100, 80, 0, 2*Math.PI, 0);
    ctx.fill();

    // For stroking, no filling
    // ctx.strokeStyle = "#FF0000";
    // ctx.rect(100, 100, 100, 100);
    // ctx.stroke();
}
```

Rendered in a browser



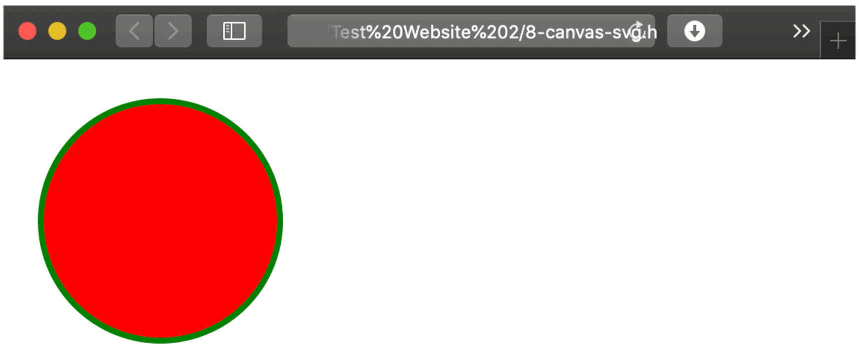
SVG

- Scalable Vector Graphics
- Vector-based graphics using HTML elements
- Do not lose any quality if they are zoomed or resized

Syntax

Circle

```
<svg id="s1" width="400px" height="200px" border="1px solid black">
        <circle cx="100" cy="100" r="80" stroke="green" stroke-width="4"
fill="red"/>
</svg>
```



Rectangle

```
<rect width="300" height="100" style="fill:rgb(0,0,255); stroke- width:3;
stroke:rgb(0,0,0)" />
```

Ellipse

```
<ellipse cx="200" cy="80" rx="100" ry="50" style="fill:yellow;
stroke:purple; stroke-width:2" />
```

Polygon

```
<polygon points="200,10 250,190 160,210" style="fill:lime; stroke:purple;
stroke-width:1" />
```

Text

```
<text x="0" y="15" fill="red" transform="rotate(30 20,40)">I love SVG</text>
```

7.0 Geolocation

- The Geolocation API of HTML5 helps in identifying the user's location
- Only if user grants permission
- Accessed via JavaScript, through the `navigator.geolocation` object

Methods

- `getCurrentPosition(success_callback [, error_callback, geo_loc_options])`
- `watchPosition(success_callback [, error_callback, geo_loc_options])`

Associated Objects

- Success callback function receives `position` object with these read only properties: `double latitude, longitude, accuracy, altitude, altitudeAccuracy, heading (direction), speed`
- Error callback function receives `error` object with these two properties:
  - `short code`
    - 1 - `PERMISSION_DENIED`
    - 2 - `POSITION_UNAVAILABLE`
    - 3 - `TIMEOUT`
  - `DOMString` message

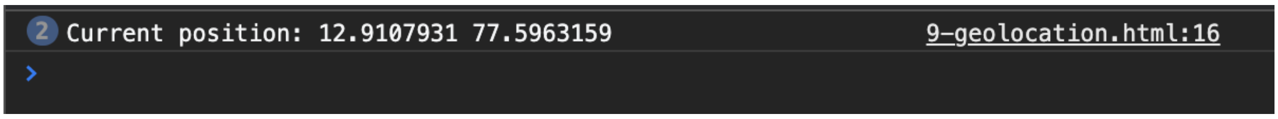
JavaScript

```javascript
function getPos() {
    navigator.geolocation.getCurrentPosition(show, error);
}

function show(position) {
    // Global for console
    ps = position;
    console.log("Current position: " + position.coords.latitude + " " + position.coords.longitude);
}

function error(error) {
    console.log(error);
}
```

HTML

```html
<body onload="getPos()">
    <button onclick="getPos()">Get current position</button>
</body>
```

Rendered in browser (only Chrome seemed to work for me; Firefox and Safari failed)



```
2 Current position: 12.9107931 77.5963159          9-geolocation.html:16
>
```

8.0 Web Worker

- A web worker is a thread executing a JavaScript file
- Asynchronous and autonomous
- A web worker does not have access to the DOM of the page that creates the web worker
- It can only listen for and post messages from and to the page
- The worker thread can perform tasks without interfering with the user interface
- Web worker performs tasks in the background, independent of other scripts and thus not affecting their performance
- The process is also called threading, i.e. separating the tasks into multiple parallel threads
- During the time, the user can browse normally, as the page stays fully responsive

Implementation

- A new `Worker` object in the main page must be created. The constructor takes the name of the worker script (eg: `my-worker.js`)
- If the specified file exists, the browser will spawn a new worker thread, which is downloaded asynchronously

```javascript
var worker = new Worker('my-worker.js');
```

- The worker will not begin until the file has completely downloaded and executed
- If the path to the file returns `404`, it will fail silently

Message Passing

Main JS file

```javascript
// Create new thread
var worker = new Worker("9-worker.js");

// Posts message to worker
worker.postMessage('Hello, world!');

```

```
// Listens for messages coming from worker
worker.onmessage = (event) => {
    if (typeof(event.data) != "object") {
        console.log("Reply: " + event.data);
    }
    else {
        console.log("Message: " + event.data.message + " " +
                    new Date().setTime(event.data.tstamp));
    }
}
```

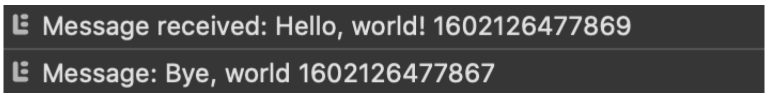
The worker file

```
this.onmessage = (event) => {
    console.log("Message received: " + event.data + ' ' + new
Date().getTime());
}

// Sends message to main thread
postMessage({"message": "Bye, world", "tstamp": new Date().getTime()});
```

Rendered in a browser

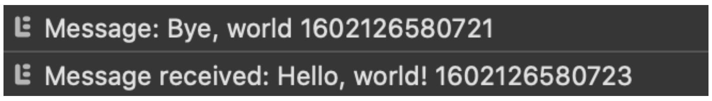
- Needs to be run on a local server [(how to setup a simple Python server)]

Sometimes received first (observe timestamp is still later)



```
Message received: Hello, world! 1602126477869
Message: Bye, world 1602126477867
```

Sometimes sent first (usually this)



```
Message: Bye, world 1602126580721
Message received: Hello, world! 1602126580723
```

9.0 jQuery

- JavaScript library that simplifies DOM manipulation and JS programming
- Basic syntax for selecting

```
$(selector).action()
```

JavaScript vs jQuery

Hide an element with id `textbox`

JavaScript

```
document.getElementById('textbox').style.display = 'none';
```

jQuery

```
$('#textbox').hide();
```

Create a `<h1>` tag with 'my text'

JavaScript

```
var h1 = document.CreateElement("h1");
h1.innerHTML = "my text";
document.getElementsByTagName('body')[0].appendChild(h1);
```

jQuery

```
$(body).append($("<h1/>").html("my text")) ;
```

Code Example

Body of HTML

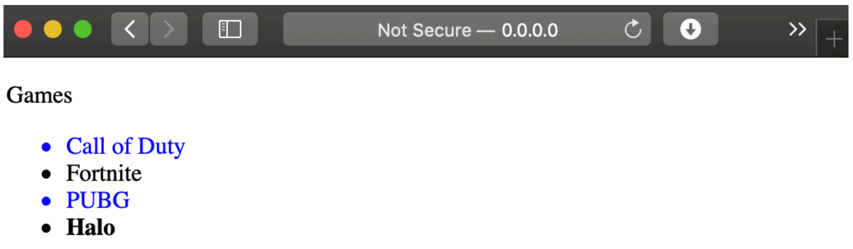
```
<body>
    <p>Games</p>
    <ul>
        <li>Call of Duty</li>
        <li class="g1">Fortnite</li>
        <li class="g1">PUBG</li>
    </ul>
</body>
```

jQuery inside the jQuery tags

- `ready` function waits for document to load
- Unlike `onload`, doesn't wait for images to load

```
<!-- Must include -->
<script type="text/javascript" src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

<script>
    // Instead of using an init() function for event onload
    $(document).ready(function () {
        $("li:even").css("color", "blue");
        console.log($("li.g1").html());        // only Fortnite

        newli = document.createElement("li");

        $(newli).html("<b>Halo</b>");
        $("li:last").after(newli);

        // Either of the two
        $(newli).attr("id", "li2");
        // newli.id = "li1";
    });
</script>
```

Rendered in browser



Games

- Call of Duty
- Fortnite
- PUBG
- **Halo**

Cascading

- Chaining Methods, also known as Cascading, refers to repeatedly calling one method after another on an object, in one continuous line of code
- Example 1

```
$("#wrapper").fadeOut().html("Welcome, Sir").fadeIn();
```

- Example 2

```
str.replace("k", "R").toUpperCase().substr(0,4);
```

Events

- Register an event handler in one of two ways

```
$("span#message").click(function(event) {});
```

```
$("span#message").on("click", function(event) {});
```

- Without the function reference argument, the event methods are treated like a manual firing of event

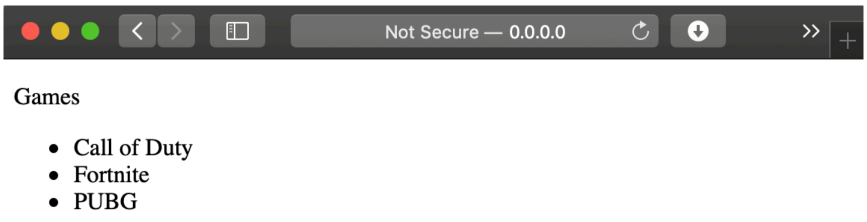
```
$("span#message").click();
```

Code Example

- Within any event handler function `this` element refers to the element for which the handler is called

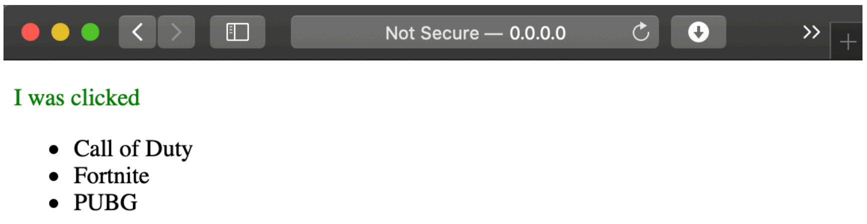
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>jQuery Events</title>
    <script
      type="text/javascript"
      src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    </script>
    <script>
      $(document).ready(function () {
        // also $('p').on('click', function()) - multiple
        $('p').click(function () {
          $(this).css('color', 'green').html('I was clicked');
        });

        $('p').mouseover(function() {
            // $('ul').slideToggle('slow');
            // $('button').toggle().html('hidden');
            if ($('button').attr('visible')) {
                $('button').show().html('hidden');
            }
            else {
                $('button').hide().html('click');
            }
        })

        // animate
        $('li.g1').click(function() {
            if ($(this).css('opacity') == 1) {
                $(this).animate({
                    left: '100px',
                    opacity: 0.4,
                    fontSize: '3em'
                }, 1000);
            }
            else {
                $(this).animate({
                    left: '0px',
                    opacity: 1,
                    fontSize: '1em'
                }, 1000);
            }
        })
      });
    </script>
  </head>
  <body>
    <!-- More than one event; chaining -->
    <p>Games</p>
    <ul>
      <li>Call of Duty</li>
      <li class="g1">Fortnite</li>
      <li class="g1">PUBG</li>
    </ul>
    <button>Click</button>
  </body>
</html>

```

Rendered in browser


Not Secure — 0.0.0.0

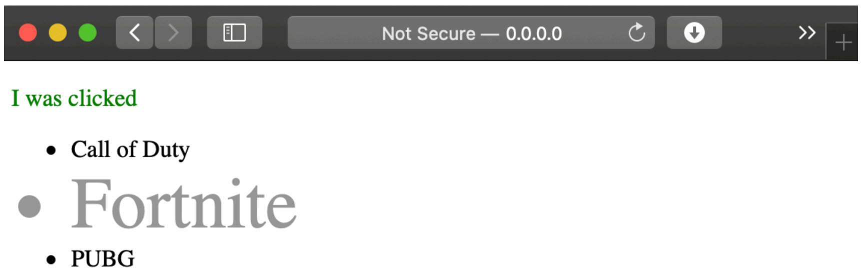
Games

- Call of Duty
- Fortnite
- PUBG


Not Secure — 0.0.0.0

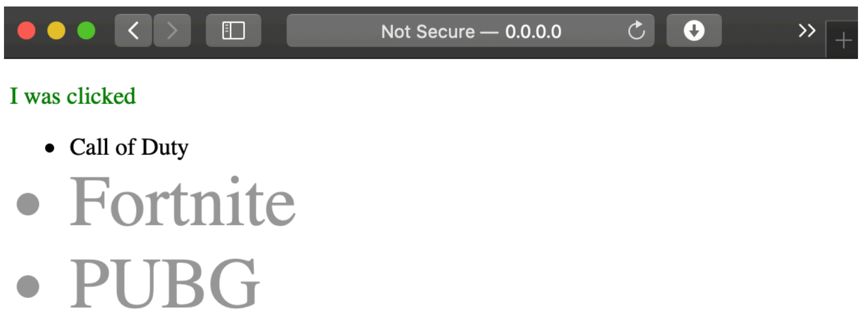
I was clicked

- Call of Duty
- Fortnite
- PUBG


Not Secure — 0.0.0.0

I was clicked

- Call of Duty
- Fortnite
- PUBG


Not Secure — 0.0.0.0

I was clicked

- Call of Duty
- Fortnite
- PUBG

- Lots of animation/styling effects can be accomplished using the effects methods like hide, show, toggle, fadein, fadeout etc (read docs)

10.0 Callbacks and Promises

Callbacks

- Function references
- Passed as arguments to other functions

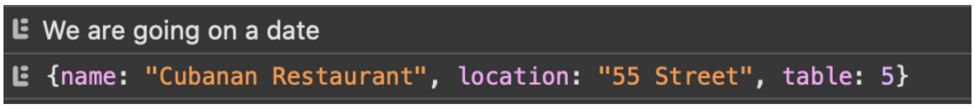
Promises

- jQuery
- A promise is used to handle the asynchronous result of an operation
- With Promises, we can defer execution of a code block until an async request is completed
- The primary way of interacting with a promise is through its then method, which registers callbacks to receive either a promise's eventual value or the reason why the promise cannot be fulfilled
- The `then()` method accepts two functions: one to execute in the event that the promise is fulfilled and the other if the promise is rejected
- If a promise is neither fulfilled nor rejected (for example, still waiting for the response of a server calculation), it's pending
- A promise may be in one of the three states: unfulfilled, fulfilled, and failed
- The promise may only move from unfulfilled to fulfilled, or unfulfilled to failed

Code Example

```
/*
1. Callbacks - function refs accepted as argument, asynchronous
2. Promises - used to handle async result of
operation (conditonal execution of callbacks)
*/

var weather;
const date = new Promise(

    // Attach a callback based on then and catch
    function(resolve, reject) {
        // Usually an API call - with delay
        setTimeout(function () {
            weather = true;

            if (weather) {
                const dateDetails = {
                    name: 'Cuban Restaurant',
                    location: '55 Street',
                    table: 5
                };
                resolve(dateDetails);
            } else {
                reject(new Error('Bad weather'));
            }
        }, 2000);
    }
)

// Status will be resolved or rejected, or pending
date
.then(function(details) {
    console.log('We are going on a date');
    console.log(details);
})
.catch(function(error) {
    console.log(error.message);
})
```

Console



```
We are going on a date
{name: "Cubanan Restaurant", location: "55 Street", table: 5}
```

11.0 Single Page Applications

- Instead of the default method of the browser loading entire new pages, a single-page application (SPA) interacts with the web browser by dynamically rewriting the current web page with new data from the web server
- Can be built using AJAX, JS frameworks etc
- The page does not reload at any point in the process, nor does it transfer control to another page

AJAX

- Using XHR or `XMLHTTPRequest`

```
var xhr = new XMLHttpRequest();
```

XHR Object Properties

- `open(method, url [, asynchronous])`
    - Initialises the request in preparation for sending to the server
    - `method` - HTTP method like GET, POST etc
    - `url` - relative or absolute URL the request will be sent to
    - `asynchronous` - boolean
- `onreadystatechange`
    - Function to call whenever the readyState changes
- `send([body])`
    - Initiates the request to the server
    - The body parameter should contain the body of the request
    - a string containing `fieldname=value&fieldname2=value2`... for POSTs
    - a `null` value for GET request
- `readyState` - int indiciating state of the request
    - 0 - uninitialized
    - 1 - loading
    - 2 - response headers received
    - 3 - some response body received
    - 4 - request complete
- `status` - HTTP status code
- `responseText`, `responseXML` and `response`
- And more (read docs)

Code Example

- Needs to be run on a local server (how to setup a simple Python server)
- Note: simple http server doesn't support POST methods

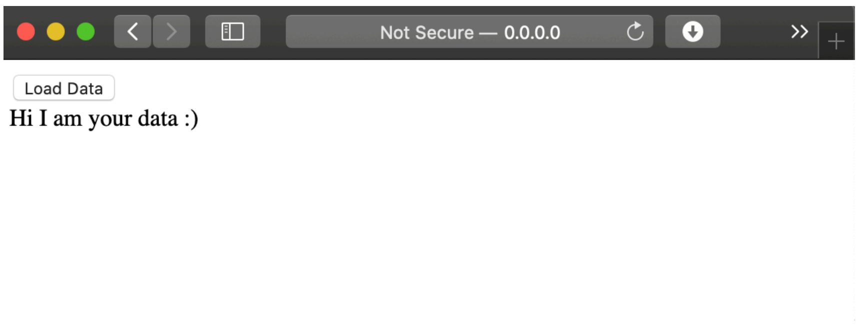
JavaScript

```javascript
function loadData() {
    let xhr = new XMLHttpRequest();

    // true - asynchronous
    xhr.open("get", "sample.txt", true);
    xhr.onreadystatechange = showData;
    // Default - text
    xhr.responseType = 'text';
    xhr.send(null);
}
```

```

function showData() {
    // this refers to xhr object
    if (this.readyState == 4 && this.status == 200) {
        // this.response or responseText or responseXML
        document.querySelector('#container').innerHTML =
this.responseText;
    }
}
```

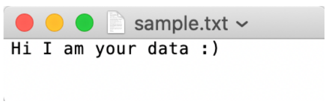
HTML

```
<button onclick="loadData()">Load Data</button>
<div id="container"></div>
```



Hi I am your data :)

sample.txt



```
Hi I am your data :)
```

12.0 jQuery `ajax` and `fetch()` methods

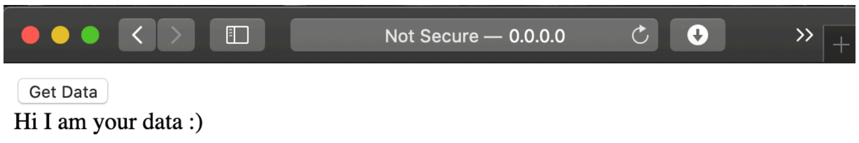
- jQuery provides methods that use XMLHttpRequest internally to make AJAX requests

Syntax

```
$.ajax({name:value, name:value})
```

Code Example

```
function getData() {
    $.ajax({
        url: 'sample.txt',
        method: 'get',
        success: function (result) {
            $("#container").html(result);
        },
        error: function (xhr, textstatus, errMsg) {
            console.log("Error: " + errMsg);
        }
    })
}
```



Hi I am your data :)

- Note: simple http server doesn't support POST methods
- To implement post, you might need to install XAMPP

Fetch Method

```
const mydiv = document.querySelector('.my-div');

fetch('resp.html')
    .then(function(response) {
        return response.text();
    })
    .then(function(text) {
        mydiv.innerHTML= text;
    });
```