

# JavaScript and Document Object Model (DOM)

Web Development and Security (ZEIT3119)

Week 4

Dr. Reza Rafeh

# Revision

What is progressive enhancement?

Answer:

A three layered approach to web design which separates the content, presentation and behavior

What are the important aspects for planning a website?

Answer:

- Content to be displayed
- Targeted Audience
- Targeted Devices

# Revision

What are the benefits of Progressive enhancement?

Answer:

- Better performance: All layers are separate from each other but still they are dependent
- Better Scaling: Adding feature and changing web design is easy and quick
- Responsive design: Adjust according to the screen size

How many types of CSS are there

- Answer:
- Inline CSS
- Embedded CSS – Preferred for a single page website
- External CSS – Preferred for entire website (multiple pages)

# Revision

What is Nielsen Heuristic evaluation?

Answer:

- Visibility of system status
- Match between system and real world
- User control and freedom
- Consistency and standards
- Error prevention
- Good error messages and recovery from errors
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Documentation and Help

# Revision

## Prototyping

*Never Go to a Client Meeting without a Prototype*

Michael Schrage

# Outline

- **JavaScript**
- Advantage
- Syntax
- Functions
- Examples
- **Document Object Model (DOM)**
- DOM Example
- **Client-Side Controls**
- HTML forms
- Data Transmission
- Hacking Steps
- Capturing User Data
- Security

# Javascript

- **HTML:** Content of Web pages
- **CSS:** Layout of Web pages
- **JavaScript:** Control the behavior of web pages
- **JavaScript**
  - programming language for Web designing
  - It is case sensitive. A variable named “MyVariable” is different than “myvariable”
  - Object-based, client-side scripting language
    - Work with objects associated with a Web page (i.e. browser window, images, links)
    - Scripting language interpreted by a browser
    - Luxury adding to the HTML content
- Features like:
  - animated graphics
  - photo slideshows
  - autocomplete text suggestions
  - interactive forms

# What JavaScript is Used for?

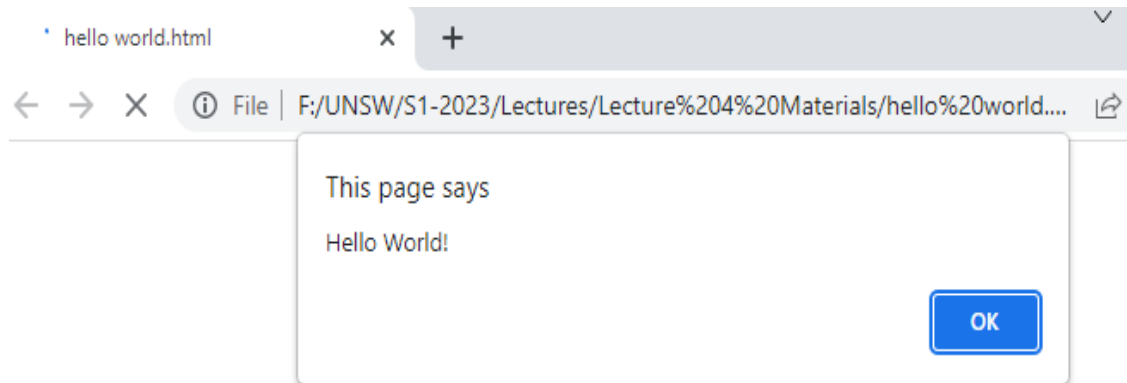
- **Adding interactivity to websites**—if you want a website to be more than a static page of text, you'll need to do some Java Scripting
- **Developing mobile applications**—JavaScript isn't just for websites...it's used to create those apps you have on your phone and tablet as well
- **Creating web browser based games**—Ever played a game directly from your web browser? JavaScript probably helped make that happen
- **Back end web development**—JavaScript is MOSTLY used on the front end of things, but it's a versatile enough scripting language to be used on back end infrastructure, too.
- Request content and information from the server and inject it into the current document as needed, without reloading the entire page—this is commonly referred to as “**Ajax.**”
- Show and hide content based on a user clicking on a link or heading, to create a “collapsible” content area



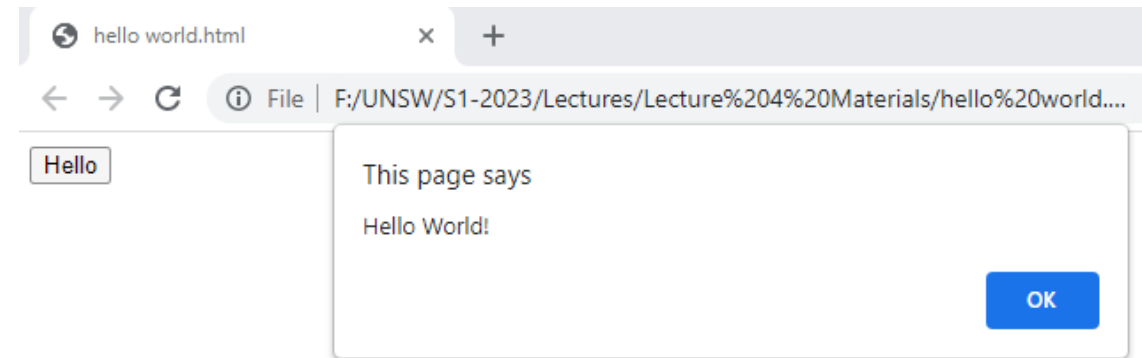


# Hello World Example

```
<!DOCTYPE html>
<html>
  <script>
    alert("Hello World!");
  </script>
</html>
```



```
<!DOCTYPE html>
<html>
  <button onclick="alert('Hello World!')">
    Hello
  </button>
</html>
```



# JavaScript Codes

## ➤ Script tags

- The `<script>` tag is used to define a client-side script (JavaScript)
- To select an HTML element, JavaScript most often uses the `getElementById()` method.
- Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

## ➤ External JavaScript

- Scripts can also be placed in external files:
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the file extension `.js`

## ➤ No Script tags

- The `<noscript>` tag is used to provide an alternate content for users that have disabled scripts in their browser or have a browser that doesn't support client-side scripts.

```
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>(myFunction is stored in an external file called "myScript.js")</p>

<script src="myScript.js"></script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

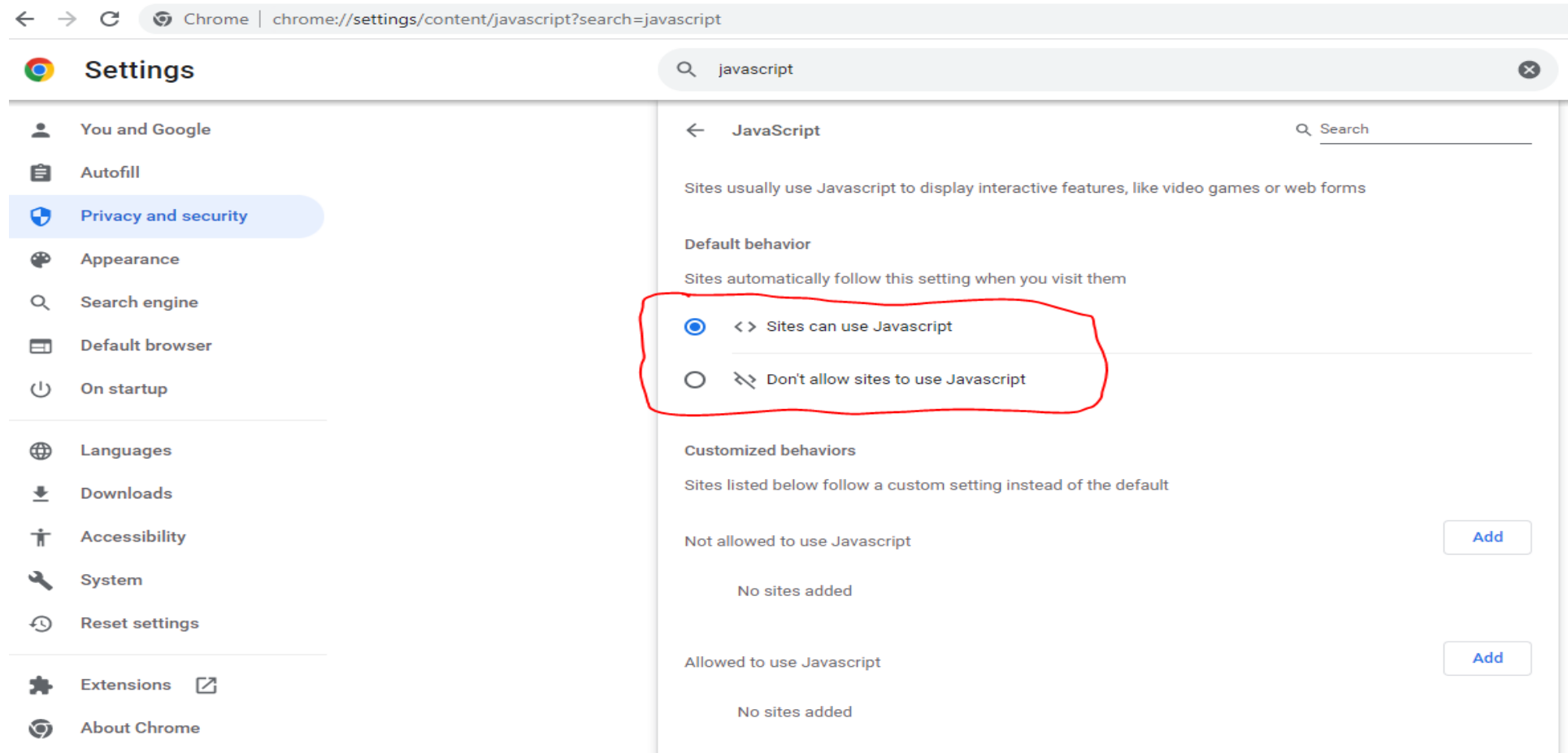
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>

<noscript>Sorry, your browser does not support JavaScript!</noscript>

<p>A browser without support for JavaScript will show the text written inside the noscript element.
</p>

</body>
</html>
```

# Turn on/off Javascript on Browser



```
<!DOCTYPE html>
<html>
  <button onclick="alert('Hello World!')"> Hello</button>
  <noscript> JavaScript is not allowed by your browser</noscript>
</html>
```

← → ↻ ⓘ File | F:/UNSW/S1-2023/Lectures/Lecture%204%20Materials/hello%20world.html

Hello JavaScript is not allowed by your browser

# JavaScript Codes

## ➤ Pop Up boxes

➤ JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box

### ➤ Alert Box

➤ An alert box is often used if you want to make sure information comes through to the user.

➤ When an alert box pops up, the user will have to click "OK" to proceed.

➤ Syntax: `window.alert("sometext")`. The `alert()` method can be written without the window prefix

### ➤ **Confirm Box**

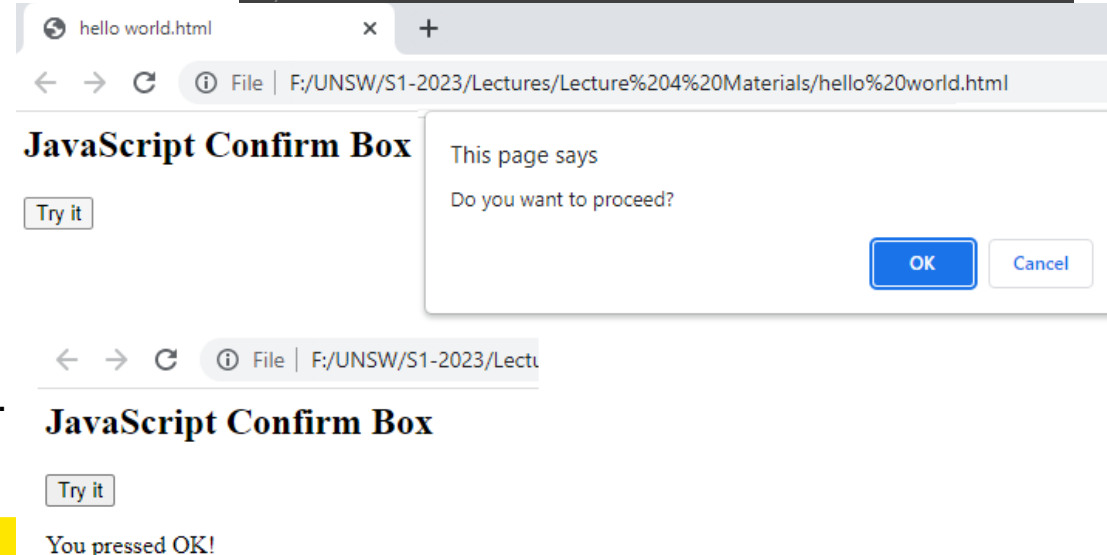
➤ A confirm box is often used if you want the user to verify or accept something.

➤ When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

➤ Syntax; `window.confirm("sometext")`;

➤ The `confirm()` method can be written without the window prefix.

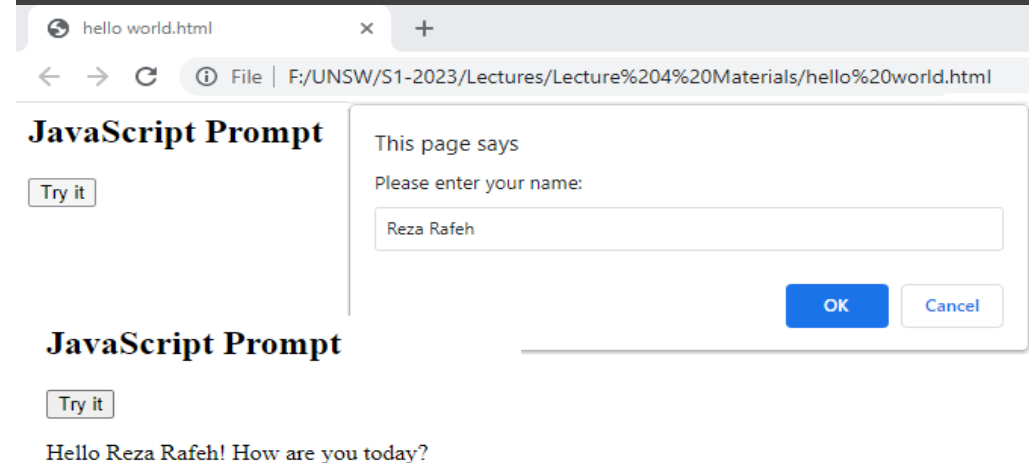
```
<html>
<body>
<h2>JavaScript Confirm Box</h2>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var txt;
  if (confirm("Do you want to proceed?")) {
    txt = "You pressed OK!";
  } else {
    txt = "You pressed Cancel!";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
</html>
```



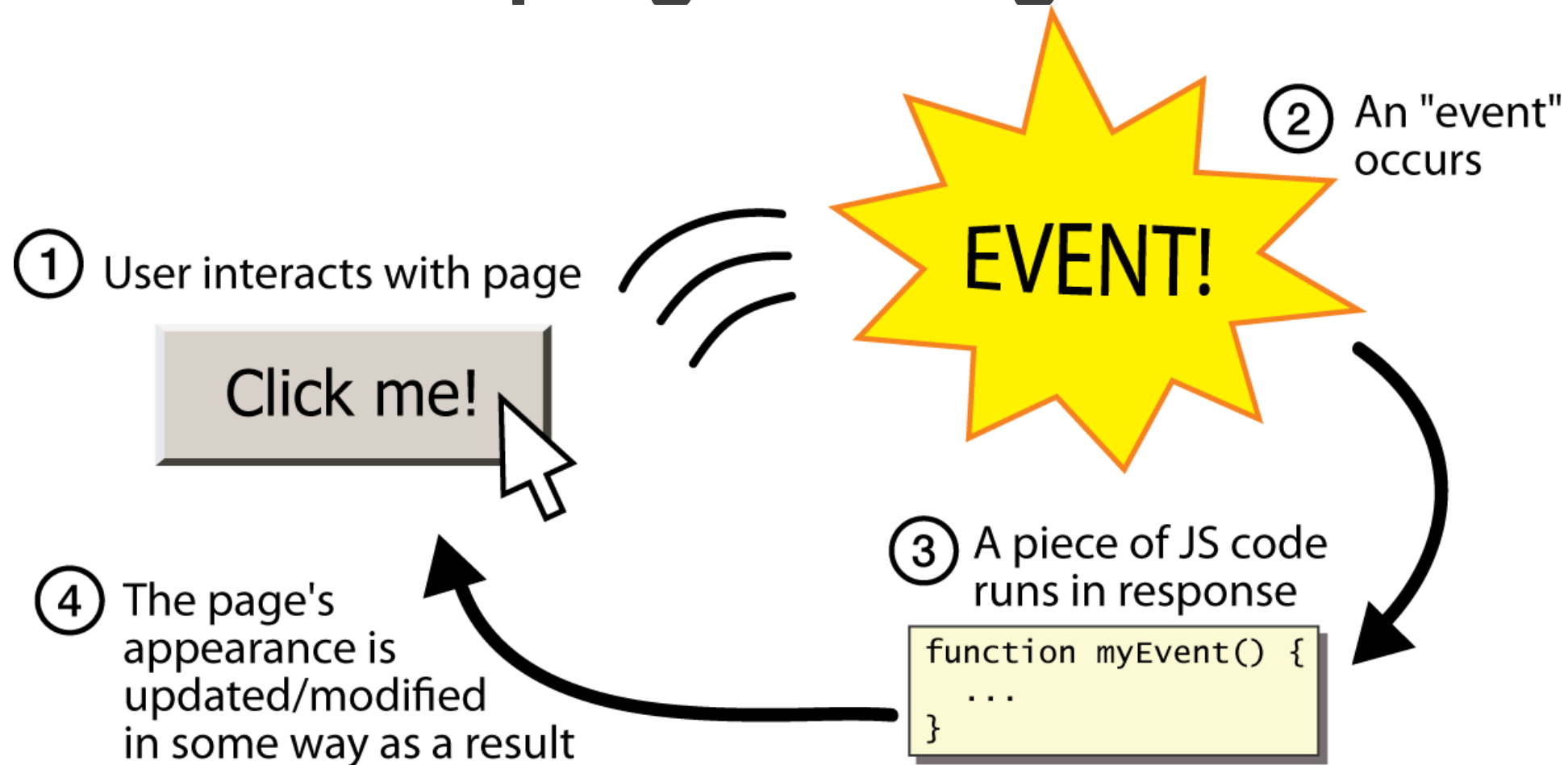
# JavaScript Codes

- **Prompt Box**
- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.
- Syntax: `prompt("sometext","defaultText");`

```
<html>
<body>
<h2>JavaScript Prompt</h2>
<button onclick="myEvent()">Try it</button>
<p id="demo"></p>
<script>
function myEvent() {
    var txt;
    var person = prompt("Please enter your name:", "Harry Potter");
    if (person == null || person == "") {
        txt = "User cancelled the prompt.";
    } else {
        txt = "Hello " + person + "! How are you today?";
    }
    document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
</html>
```



# Event-driven programming



# Document Object Model (DOM)

- It is a programming interface (API) for HTML and XML
- It was released in 1998
- DOM manipulate the contents of a document
- It can be used with any XML language (JavaScript, PHP, Ruby and more)
- Provides a structured map of document
- Translates the markup into a format that JavaScript can understand
- Without DOM, JavaScript cannot understand document's content
- The DOM is a collection of nodes:
  - Element nodes
  - Attribute nodes
  - Text nodes



# Document Object Model (DOM)

most JS code manipulates elements on an HTML page

we can examine elements' state

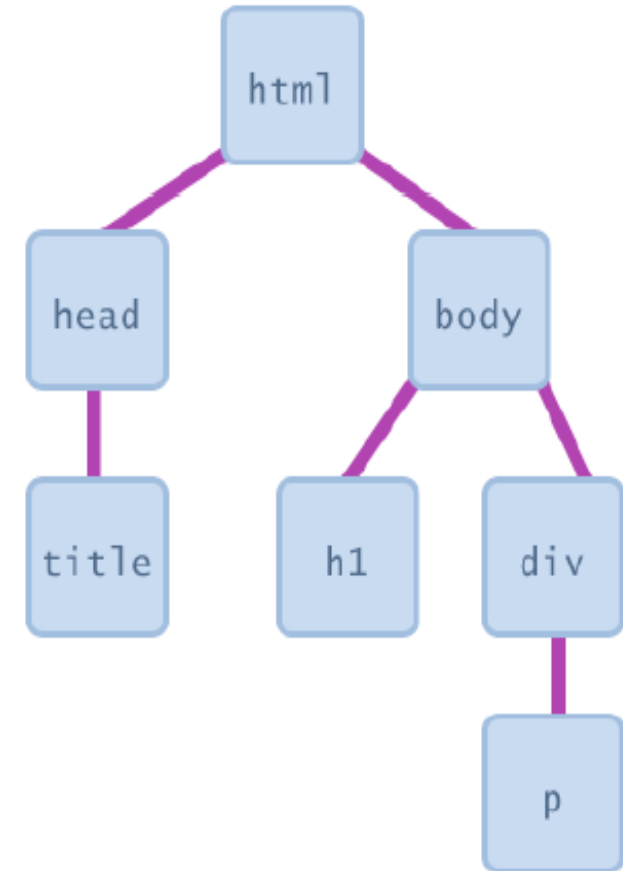
- e.g. see whether a box is checked

we can change state

- e.g. insert some new text into a div

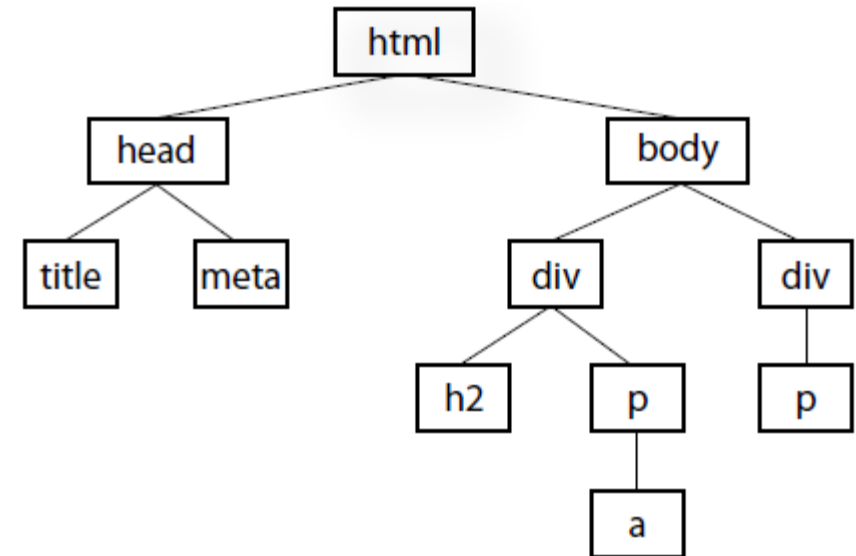
we can change styles

- e.g. make a paragraph red



# Document Object Model (DOM)

```
<html>
<head>
<title>Document title</title>
<meta charset="utf-8"/>
</head>
<body>
<div>
<h2>Subhead</h2>
<p>Paragraph text with a <a
href="foo.html">link</a> here.</p>
</div>
<div>
<p>More text here.</p>
</div>
</body>
</html>
```

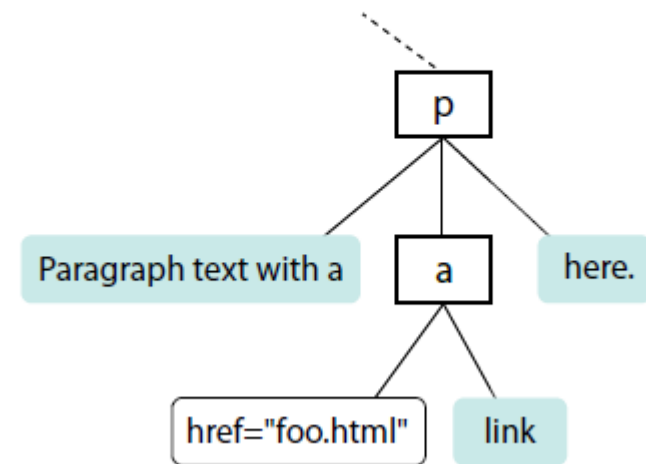


A simple document tree

# Document Object Model (DOM)

- It shows the structure of the first **p** element
- The element, its attributes, and its contents are all nodes in the DOM's node tree
- DOM also provides a standardized set of methods and functions through which JavaScript can interact with the elements on our page.

```
<p>Paragraph text with a <a href="foo.html">link</a> here.</p>
```



# DOM element objects

HTML

```
<p>  
  Look at this octopus:  
    
  Cute, huh?  
</p>
```

DOM Element Object	
Property	Value
tagName	"IMG"
<u>src</u>	"octopus.jpg"
alt	"an octopus"
id	"icon01"

JavaScript

```
var icon = document.getElementById("icon01");  
icon.src = "kitty.gif";
```

# Accessing elements:

## `document.getElementById`

```
var name = document.getElementById("id");
```

JS

```
<button onclick="changeText();">Click me!</button>  
<span id="output">replace me</span>  
<input id="textbox" type="text" />
```

HTML

```
function changeText() {  
    var span = document.getElementById("output");  
    var textBox = document.getElementById("textbox");  
  
    textBox.style.color = "red";  
  
}
```

JS

# Accessing elements:

## `document.getElementById`

- `document.getElementById` returns the DOM object for an element with a given id
- can change the text inside most elements by setting the `innerHTML` property
- can change the text in form controls by setting the `value` property

# Document Object Model (DOM)

There are several ways to use DOM and find what do you want

➤ **By id attribute value - `getElementById()`**

- This method returns a single element based on that element's ID

```

var photo = document.getElementById("lead-photo");
```

➤ **By tag name - `getElementsByTagName()`**

- This method retrieves any element or elements you specify as an argument.

- **`document.getElementsByTagName("p")`** returns every paragraph on the page

```
var paragraphs = document.getElementsByTagName("p");
for( var i = 0; i < paragraphs.length; i++ ) {
  // do something
}
```

# Document Object Model (DOM)

There are several ways to use DOM and find what do you want

- **By class attribute value** - `getElementsByClassName()`
- Access nodes in the document based on the value of a **class** attribute

```
var firstColumn = document.getElementsByClassName("column-a");
```

- **By selector** - `querySelectorAll()`
- access nodes of the DOM based on a CSS-style selector.

```
var sidebarPara = document.querySelectorAll(".sidebar p");
```

- **Access:**

```
var bigImage = document.getElementById("lead-image");  
alert( bigImage.getAttribute("src") );
```
- It is use to get value of an attribute attached to an element node



# Document Object Model (DOM)

- There are several ways to use DOM and find what do you want
- **innerHTML**
- simple method for accessing and changing the text and markup inside an element.
- **setAttribute**
- To set the value of the attribute
- It can be used to update the title attribute
- It can be used to update the checked attributes of the checkboxes and radio buttons

```
var bigImage = document.getElementById("lead-image");  
bigImage.setAttribute("src", "lespaul.jpg");
```

```
<!DOCTYPE html>  
<html>  
  <body>  
    <p id="demo">This is a test for changing innerHTML</p>  
    <button onclick="myFunction()"> Hello</button>  
  </body>  
  
  <script>  
    function myFunction(){  
      document.getElementById("demo").innerHTML = "I have  
changed!";  
    }  
  </script>  
</html>
```

← → ↻ ⓘ File | F:/UNSW/S1-2023/Lecture

This is a test for changing innerHTML

Hello

← → ↻ ⓘ File | F:/UN

I have changed!

Hello

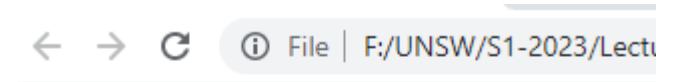
# Changing element style: `element.style`

Attribute	Property or style object
color	color
padding	padding
background-color	backgroundColor
border-top-width	borderTopWidth
Font size	fontSize
Font famiy	fontFamily

# Preetify

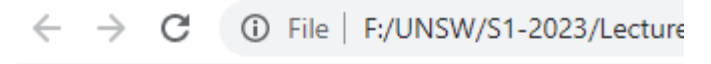
```
<!DOCTYPE html>
<html>
  <body>
    <p id="demo">This is a test for changing
innerHTML</p>
    <button onclick="changeText()"> Hello</button>
  </body>

  <script>
    function changeText() {
      var text = document.getElementById("demo");
      text.style.fontSize = "13pt";
      text.style.fontFamily = "Comic Sans MS";
      text.style.color = "red";
    }
  </script>
</html>
```



This is a test for changing innerHTML

Hello



This is a test for changing innerHTML

Hello

# JavaScript – Browser Object

- In JavaScript, the browser is known as the window object. The window object has a number of properties and methods that we can use to interact with it.
- In fact, alert() is one of the standard browser object methods.

Property/method	Description
event	Represents the state of an event
history	Contains the URLs the user has visited within a browser window
location	Gives read/write access to the URI in the address bar
status	Sets or returns the text in the status bar of the window
alert()	Displays an alert box with a specified message and an OK button
close()	Closes the current window
confirm()	Displays a dialog box with a specified message and an OK and a Cancel button
focus()	Sets focus on the current window

# JavaScript – document Object

- The document object represents your web page.
- Any element in an HTML page can be accessed through the document object.

Property	Description
<code>document.anchors</code>	Returns all <a> elements that have a name attribute
<code>document.baseURI</code>	Returns the absolute base URI of the document
<code>document.body</code>	Returns the <body> element
<code>document.cookie</code>	Returns the document's cookie
<code>document.doctype</code>	Returns the document's doctype
<code>document.documentElement</code>	Returns the <html> element
<code>document.documentMode</code>	Returns the mode used by the browser
<code>document.documentURI</code>	Returns the URI of the document
<code>document.domain</code>	Returns the domain name of the document server
<code>document.embeds</code>	Returns all <embed> elements
<code>document.forms</code>	Returns all <form> elements
<code>document.head</code>	Returns the <head> element
<code>document.images</code>	Returns all <img> elements
<code>document.implementation</code>	Returns the DOM implementation
<code>document.inputEncoding</code>	Returns the document's encoding (character set)
<code>document.lastModified</code>	Returns the date and time the document was updated
<code>document.links</code>	Returns all <area> and <a> elements that have a href attribute
<code>document.readyState</code>	Returns the (loading) status of the document
<code>document.referrer</code>	Returns the URI of the referrer (the linking document)
<code>document.scripts</code>	Returns all <script> elements
<code>document.strictErrorChecking</code>	Returns if error checking is enforced
<code>document.title</code>	Returns the <title> element
<code>document.URL</code>	Returns the complete URL of the document

# JavaScript – document Object

Property / Method	Description		
<a href="#">activeElement</a>	Returns the currently focused element in the document	<a href="#">getElementsByName()</a>	Returns an <a href="#">HTMLCollection</a> containing all elements with the specified class name
<a href="#">addEventListener()</a>	Attaches an event handler to the document	<a href="#">getElementsByName()</a>	Returns an live <a href="#">NodeList</a> containing all elements with the specified name
<a href="#">adoptNode()</a>	Adopts a node from another document	<a href="#">getElementsTagName()</a>	Returns an <a href="#">HTMLCollection</a> containing all elements with the specified tag name
<a href="#">baseURI</a>	Returns the absolute base URI of a document	<a href="#">hasFocus()</a>	Returns a Boolean value indicating whether the document has focus
<a href="#">body</a>	Sets or returns the document's body (the <body> element)	<a href="#">head</a>	Returns the <head> element of the document
<a href="#">charset</a>	<a href="#">Deprecated</a>	<a href="#">images</a>	Returns a collection of all <img> elements in the document
<a href="#">characterSet</a>	Returns the character encoding for the document	<a href="#">implementation</a>	Returns the DOMImplementation object that handles this document
<a href="#">close()</a>	Closes the output stream previously opened with document.open()	<a href="#">importNode()</a>	Imports a node from another document
<a href="#">cookie</a>	Returns all name/value pairs of cookies in the document	<a href="#">lastModified</a>	Returns the date and time the document was last modified
<a href="#">createAttribute()</a>	Creates an attribute node	<a href="#">links</a>	Returns a collection of all <a> and <area> elements in the document that have a href attribute
<a href="#">createComment()</a>	Creates a Comment node with the specified text	<a href="#">normalize()</a>	Removes empty Text nodes, and joins adjacent nodes
<a href="#">createDocumentFragment()</a>	Creates an empty DocumentFragment node	<a href="#">open()</a>	Opens an HTML output stream to collect output from document.write()
<a href="#">createElement()</a>	Creates an Element node	<a href="#">querySelector()</a>	Returns the first element that matches a specified CSS selector(s) in the document
<a href="#">createEvent()</a>	Creates a new event	<a href="#">querySelectorAll()</a>	Returns a static <a href="#">NodeList</a> containing all elements that matches a specified CSS selector(s) in the document
<a href="#">createTextNode()</a>	Creates a Text node	<a href="#">readyState</a>	Returns the (loading) status of the document
<a href="#">defaultView</a>	Returns the window object associated with a document, or null if none is available.	<a href="#">referrer</a>	Returns the URL of the document that loaded the current document
<a href="#">designMode</a>	Controls whether the entire document should be editable or not.	<a href="#">removeEventListener()</a>	Removes an event handler from the document (that has been attached with the <a href="#">addEventListener()</a> method)
<a href="#">doctype</a>	Returns the Document Type Declaration associated with the document	<a href="#">scripts</a>	Returns a collection of <script> elements in the document
<a href="#">documentElement</a>	Returns the Document Element of the document (the <html> element)	<a href="#">title</a>	Sets or returns the title of the document
<a href="#">documentURI</a>	Sets or returns the location of the document	<a href="#">URL</a>	Returns the full URL of the HTML document
<a href="#">domain</a>	Returns the domain name of the server that loaded the document	<a href="#">write()</a>	Writes HTML expressions or JavaScript code to a document
<a href="#">embeds</a>	Returns a collection of all <embed> elements the document	<a href="#">writeln()</a>	Same as write(), but adds a newline character after each statement
<a href="#">forms</a>	Returns a collection of all <form> elements in the document		
<a href="#">getElementById()</a>	Returns the element that has the ID attribute with the specified value		

# Document Example

```
<!DOCTYPE html>
<html>
  <body>
    <button onclick="writeText()"> Hello</button>
  </body>

  <script>
    function writeText() {
      document.write("Hello World!");
      document.write("<br>");
      document.write(document.URL);
    }
  </script>
</html>
```

← → ↻ ⓘ File | F:/UNSW/S1-2023/Lectures/Lecture%204%20Materials/docu

Hello World!

file:///F:/UNSW/S1-2023/Lectures/Lecture%204%20Materials/document.html

# JavaScript Codes

## Code Blocks, Number and Strings

### ➤ Code Blocks

- JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.
- The purpose of code blocks is to define statements to be executed together

### ➤ Strings

- String method helps to work with strings
- In this example it returns the length of the string

```
<html>
<body>
<p>JavaScript code blocks are written between { and }</p>
<button type="button" onclick="myFunction()">Click Me!</button>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
</script>
</body>
</html>
```

### JavaScript Statements

JavaScript code blocks are written between { and }

Click Me!

Hello Dolly!

How are you?

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript String Properties</h2>
<p>The length property returns the length of a string:</p>
<p id="demo"></p>
<script>
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
document.getElementById("demo").innerHTML = sln;
</script>
</body>
</html>
```

### JavaScript String Properties

The length property returns the length of a string:



# Variables

```
var name = expression;
```

*JS*

```
var clientName = "Connie Client";  
var age = 32;  
var weight = 127.4;
```

*JS*

variables are declared with the var keyword (case sensitive)

types are not specified, but JS does have types ("loosely typed")

- Number, Boolean, String, Array, Object, Function, Null, Undefined
- can find out a variable's type by calling `typeof`

# Number type

```
var enrollment = 99;  
var medianGrade = 2.8;  
var credits = 5 + 4 + (2 * 3);
```

*JS*

integers and real numbers are the same type (no int vs. double)

same operators: + - \* / % ++ -- = += -= \*= /= %=

many operators auto-convert types: "2" \* 3 is 6

# Comments

```
// single-line comment  
/* multi-line comment */
```

*JS*

# Math object

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);  
var three = Math.floor(Math.PI);
```

JS

- **methods:** abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan
- **properties:** E, PI

# Special values: null and undefined

```
var ned = null;  
var benson = 9;  
// at this point in the code,  
// ned is null  
// benson's 9  
// caroline is undefined
```

JS

- `undefined` : has not been declared, does not exist
- `null` : exists, but was specifically assigned an empty or null value
- Why does JavaScript have both of these?

# Logical operators

- `> < >= <= && || ! == != === !==`
- most logical operators automatically convert types:
  - ▣ `5 < "7"` is true
  - ▣ `42 == 42.0` is true
  - ▣ `"5.0" == 5` is true
- `===` and `!==` are strict equality tests; checks both type and value
  - ▣ `"5.0" === 5` is false

# if/else statement (same as Java)

```
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

JS

JavaScript allows almost anything as a condition

# Boolean type

```
var iLikeJS = true;  
var ieJsGood = "JS" > 0; // false  
if ("web devevelopment is great") { /* true */ }  
if (0) { /* false */ }
```

JS

- any value can be used as a Boolean
  - ▣ "falsey" values: 0, 0.0, NaN, "", null, and undefined
  - ▣ "truthy" values: anything else
- converting a value into a Boolean explicitly:
  - ▣ `var boolValue = Boolean(otherValue);`
  - ▣ `var boolValue = !! (otherValue);`



# for loop

```
var sum = 0;  
for (var i = 0; i < 100; i++) {  
    sum = sum + i;  
}
```

JS

```
var s1 = "hello";  
var s2 = "";  
for (var i = 0; i < s.length; i++) {  
    s2 += s1.charAt(i) + s1.charAt(i);  
}  
// s2 stores "hheelllloo"
```

JS

# while loops (same as Java)

```
while (condition) {  
    statements;  
}
```

JS

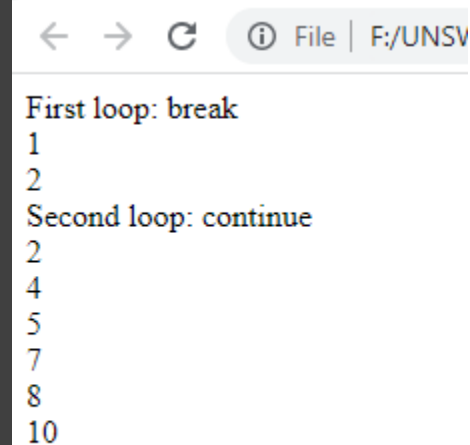
```
do {  
    statements;  
} while (condition);
```

JS

- break keyword breaks the loop (even if the condition is true)
- Continue keyword jumps to the beginning of the loop

# break and continue

```
<!DOCTYPE html>
<html>
  <script>
    var i = 1;
    document.write("First loop: break"+"<br>");
    while (i<10) {
      if (i%3 == 0)
        break;
      document.write(i+"<br>");
      i++;
    }
    i = 1;
    document.write("Second loop: continue"+"<br>");
    while (i<10) {
      i++;
      if (i%3 == 0)
        continue;
      document.write(i+"<br>");
    }
  </script>
</html>
```



First loop: break  
1  
2  
Second loop: continue  
2  
4  
5  
7  
8  
10

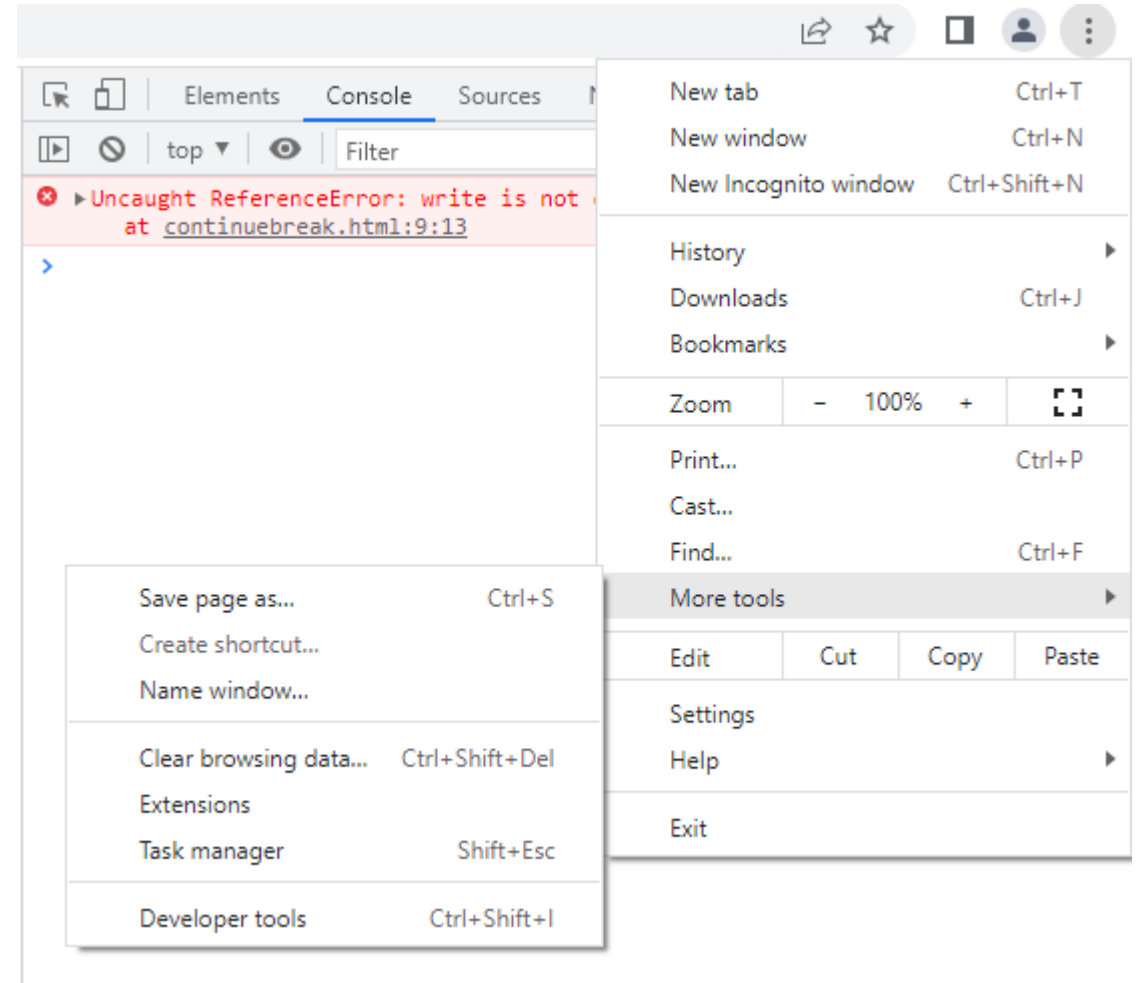
# Infinite Loop

```
while (i<10) {  
    if (i%3 == 0)  
        break;  
    document.write(i+"<br>");  
    _____  
}
```

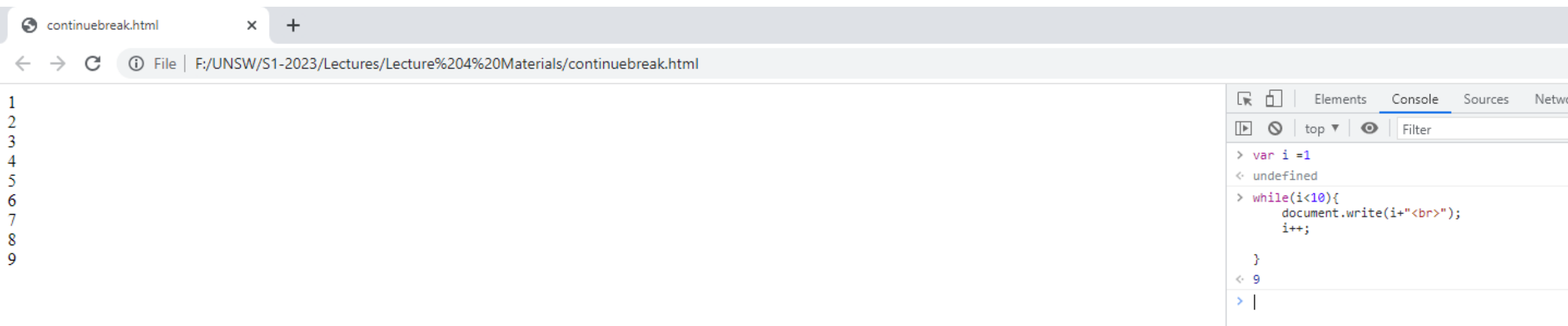
i++ is missing

# Debugging JS in Browser

```
while (i<10) {  
    if (i%3 == 0)  
        break;  
    write(i+"<br>");  
    i++;  
}
```



# Writing and Running JS in Browser



# Arrays

```
var name = []; // empty array  
var name = [value, value, ..., value]; // pre-filled  
name[index] = value; // store element
```

JS

```
var ducks = ["Huey", "Dewey", "Louie"];  
var stooges = []; // stooges.length is 0  
stooges[0] = "Larry"; // stooges.length is 1  
stooges[1] = "Moe"; // stooges.length is 2  
stooges[4] = "Curly"; // stooges.length is 5  
stooges[4] = "Shemp"; // stooges.length is 5
```

JS

# Array methods

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

JS

- **methods:** concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift
  - ▣ push and pop: add / remove from back
  - ▣ unshift and shift: add / remove from front
  - ▣ shift and pop: return the element that is removed



# String type

```
var s = "Connie Client";  
var fName = s.substring(0, s.indexOf(" ")); // "Connie"  
var len = s.length; // 13  
var s2 = 'Melvin Merchant';
```

JS

**methods:** charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase

- charAt returns a one-letter String (there is no char type)

length property (not a method as in Java)

Strings can be specified with "" or ''

concatenation with + :

- 1 + 1 is 2, but "1" + 1 is "11"

# More about String

- escape sequences behave as in Java: `\' \'\" \& \n \t \\\`
- converting between numbers and Strings:

```
var count = 10;  
var s1 = "" + count; // "10"  
var s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah  
ah ah!"  
var n1 = parseInt("42 is the answer"); // 42  
var n2 = parseFloat("booyah"); // NaN
```

JS

accessing the letters of a String:

```
var firstLetter = s[0];  
var firstLetter = s.charAt(0); // does work in IE  
var lastLetter = s.charAt(s.length - 1);
```

JS

# Splitting strings: split and join

```
var s = "the quick brown fox";  
var a = s.split(" "); // ["the", "quick", "brown", "fox"]  
a.reverse(); // ["fox", "brown", "quick", "the"]  
s = a.join("!"); // "fox!brown!quick!the"
```

JS

- split breaks apart a string into an array using a delimiter
  - ▣ can also be used with regular expressions (seen later)
- join merges an array into a single string, placing a delimiter between them

# JavaScript Codes

## Date

- **Date**
- Date can be displayed using new Date() constructor.
- Different ways of displaying a date
  - new Date()
  - new Date(*year, month, day, hours, minutes, seconds, milliseconds*)
  - new Date(*milliseconds*)
  - new Date(*date string*)

```
<html>
<body>

<h2>JavaScript new Date()</h2>

<p id="demo"></p>

<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

### JavaScript new Date()

Mon May 24 2021 13:42:44 GMT+1000 (Australian Eastern Standard Time)

```
<html>
<body>

<h2>JavaScript new Date()</h2>

<p>3 numbers specify year, month, and day:</p>

<p id="demo"></p>

<script>
var d = new Date(2018, 11, 24);
document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

### JavaScript new Date()

3 numbers specify year, month, and day:

Mon Dec 24 2018 00:00:00 GMT+1100 (Australian Eastern Daylight Time)

# Client- Side Control

## ➤ **Client-Side**

- Refers to operations that are performed by the client in a client–server environment
- Typically, web browser, that runs on a user's local computer
- The user has complete control over the client

## ➤ **Client-Side Control**

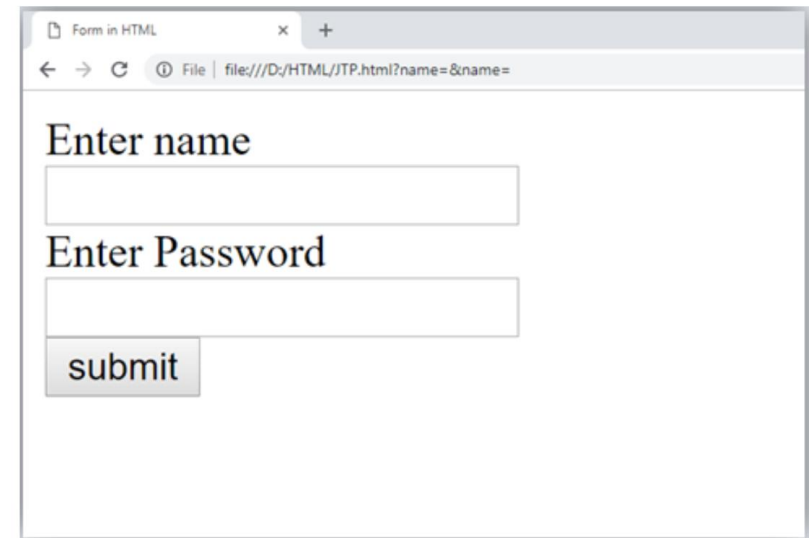
- An application may rely on client-side controls to restrict user input in two broad ways.
  - Transmitting data via the client component
  - Implementing measures on the client side

# Client-Side Control: HTML Forms

- Simplest and most common mechanism for capturing input from the user and submitting it to the server

- Example of an HTML form

```
<form>  
  <label for="name">Enter name</label><br>  
  <input type="text" id="name" name="name"><br>  
  <label for="pass">Enter Password</label><br>  
  <input type="Password" id="pass" name="pass"><br>  
  <input type="submit" value="submit">  
</form>
```

A screenshot of a web browser window titled "Form in HTML". The address bar shows the file path "file:///D:/HTML/TP.html?name=&name=". The form contains two text input fields. The first field is labeled "Enter name" and the second is labeled "Enter Password". Below the second field is a "submit" button.

# Client-Side Controls – Transmitting Data

## Disabled Elements/Hidden Fields

- **Hidden HTML form fields** are a common mechanism for transmitting data via the client in an unmodifiable way.
- If a field is flagged as hidden, it is not displayed on-screen.
- The field's name and value are stored within the form and are sent back to the application when the user submits the form.
- Element on an HTML form is flagged as disabled, it appears on-screen but is grayed out and is not editable or usable

```
<form action="order.asp" method="post">
```

```
<p>Product: <input disabled="true"  
name="product" value="Sony VAIOA217S"></p>
```

```
<p>Quantity: <input size="2" name="quantity">  
<input name="price" type="hidden"  
value="1224.95">  
<input type="submit" value="Buy!"></p>  
</form>
```



Please enter your order quantity:

Product:

Quantity:

# Client-Side Controls – Transmitting Data

## HTTP Cookies

- Another common mechanism for transmitting data via the client is HTTP cookies.
- They can be modified using an intercepting proxy, by changing either the server response that sets them or subsequent client requests that issue them. (This is not the case using hidden field)
- If the application trusts the value of the DiscountAgreed cookie when it is submitted back to the server, customers can obtain arbitrary discounts by modifying its value.

HTTP/1.1 200 OK Set-Cookie:

DiscountAgreed=25  
Content-Length: 1530

POST /shop/92/Shop.aspx?prod=3  
HTTP/1.1 Host: mdsec.net Cookie:

DiscountAgreed=25 Content-Length: 10  
quantity=l



# Client-Side Controls – Transmitting Data

## URL Parameters

- Applications frequently transmit data via the client using preset URL parameters.
- User can modify the parameters without using any particular tool
- In some cases user is not able to modify the URL parameters:
  - Where embedded images are loaded using URLs containing parameters
  - Where URLs containing parameters are used to load a frame's contents

# Client-Side Control

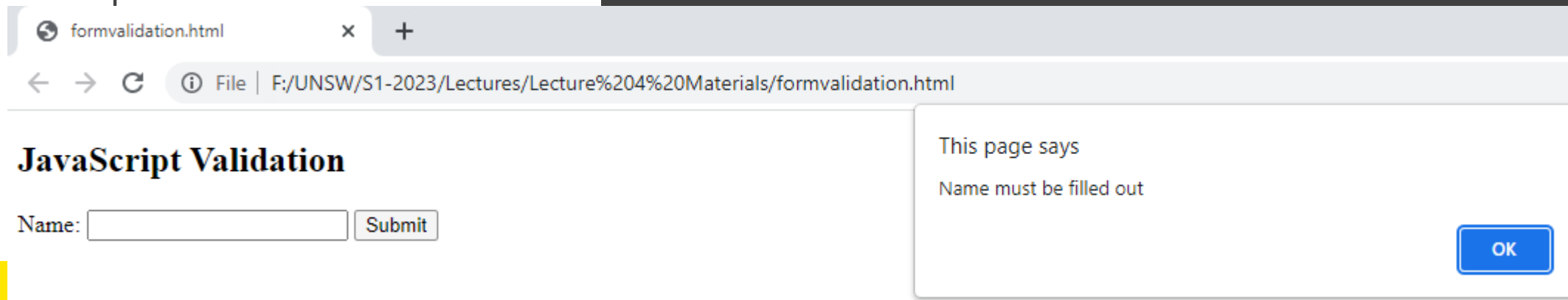
## Hack Steps

- Locate all instances within the application where hidden form fields, cookies, and URL parameters are apparently being used to transmit data via the client.
- Look for form elements containing a max-length attribute.
- Submit data that is longer than this length
- If the application accepts the overlong data, you may infer that the client-side validation is not replicated on the server.
- Identify any cases where client-side JavaScript is used
- Submit data to the server by blocking the validation steps
- Determine whether the client-side controls are replicated on the server
- And if not, whether this can be exploited for any malicious purpose.
- The above security flaws if exists, can lead to possibilities of other vulnerabilities such as SQL injection, cross-site scripting, or buffer overflows.

# Script Based Validation

- HTML form validation can be done through JavaScript
- Input validation mechanisms built into HTML forms are simple and fine-grained to perform relevant validation for many kinds of input
- Therefore, common to see customized client-side input validation implemented within scripts

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function validateForm() {
        let x = document.forms["myForm"]["fname"].value;
        if (x == "") {
          alert("Name must be filled out");
          return false;
        }
      }
    </script>
  </head>
  <body>
    <h2>JavaScript Validation</h2>
    <form name="myForm" action="/action_page.php" onsubmit="return
validateForm()" method="post">
      Name: <input type="text" name="fname">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```



The screenshot shows a web browser window with the address bar displaying the file path: F:/UNSW/S1-2023/Lectures/Lecture%204%20Materials/formvalidation.html. The page title is "formvalidation.html". The main content of the page is titled "JavaScript Validation" and contains a form with a text input field labeled "Name:" and a "Submit" button. An alert dialog box is displayed on the right side of the browser window, with the message "This page says Name must be filled out" and an "OK" button.

# Capturing User Data: Thick-Client Components

- Besides HTML forms, the other main method for capturing, validating, and submitting user data
- **Java Applet:**
  - Popular for implementing thick-client components
  - Cross-platform and run in a sandboxed environment
  - Main use: to capture user input or other in-browser information
- **ActiveX Control:**
  - Heavyweight technology compared to Java
  - ActiveX controls are written in C and C++
    - Can't be decompiled back to source code easily
  - It's possible for a user to hack ActiveX, but too complicated

# Capturing User Data: Thick-Client Components

## ➤ Shockwave Flash Objects

- Most common use of Flash is for an application context for online games
- Flash objects are contained within a compiled file that the browser downloads from the server and executes in a virtual machine (Flash player)
- SWF file contains bytecode that can be decompiled to recover the original source

# Handling Client-Side Data Securely

- Security problems with web applications arise because client-side components and user input are outside of the server's direct control
- Encryption techniques can be used to prevent tampering by the user
- Logging and Alerting
- Integration of server-side intrusion detection defenses
- Anomalies should be logged and administrators should be alerted in real time to take action

# Handling Client-Side Data Securely

- Security problems with web applications arise because client-side components and user input are outside of the server's direct control
- Encryption techniques can be used to prevent tampering by the user
- Logging and Alerting
- Integration of server-side intrusion detection defenses
- Anomalies should be logged and administrators should be alerted in real time to take action

# Final Note

- **Quiz 1 in labs this week:**
  - Monday lab: 16:20
  - Wednesday lab: 15:20
- **Feedback on others designs (evaluation) this week, due on 22nd of March (individual)**
- **Please participate in course survey – part 1:**

<https://www.surveymonkey.com/r/6WRYPNJ>