

# COMP309

## Web-based Technology

---

Dr. N. B. Gyan

Central University, Miotso. Ghana

# Recap

Discussion of *possible* solution to Project 2

# Web client-side Programming

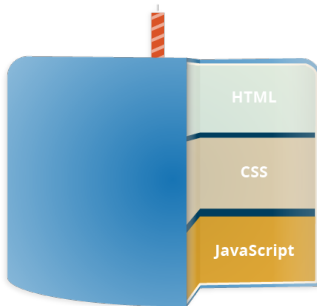
---

# What is JavaScript?

- JavaScript *is* a programming language that allows you to implement complex things on web pages.
- Every time a web page does more than just sit there and display static information for you to look at—displaying timely content updates, or interactive maps, or animated 2D/3D graphics, or scrolling video jukeboxes, etc.—you can bet that JavaScript is probably involved.

# A High-level Definition

- JavaScript is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS).



# The Different Layers

The three layers build on top of one another nicely.

- Let's take a simple text label as an example. We can mark it up using HTML to give it structure and purpose (`index.html`):

```
<!DOCTYPE html>
...
  <p>Player 1: Chris</p>
...
```

# The Different Layers

- Then we can add some CSS into the mix to get it looking nice (`index.css`):

```
1 p {  
2   font-family: 'helvetica neue', helvetica ,  
3               sans-serif;  
4   letter-spacing: 1px;  
5   text-transform: uppercase;  
6   text-align: center;  
7   border: 2px solid rgba(0, 0, 200, 0.6);  
8   background: rgba(0, 0, 200, 0.6);
```

```
9   color: rgba(0, 0, 200, 0.6);
10  box-shadow: 1px 1px 2px
11              rgba(0, 0, 200, 0.4);
12  border-radius: 10px;
13  padding: 3px 10px;
14  display: inline-block;
15  cursor: pointer;
16 }
```



# The Different Layers

PLAYER 1: CHRIS

# The Different Layers

- And finally, we can add some JavaScript to implement *dynamic* behaviour (**script.js**):

```
1 var para = document.querySelector('p');  
2  
3 para.addEventListener('click', updateName);  
4  
5 function updateName(){  
6     var name = prompt('Enter a new name');  
7     para.textContent = 'Player 1: ' + name;  
8 }
```

## How do you add JavaScript to your Page?

- JavaScript is applied to your HTML page in a similar manner to CSS.
- Whereas CSS uses `<link>` elements to apply external stylesheets and `<style>` elements to apply internal stylesheets to HTML, JavaScript only needs *one* friend in the world of HTML—the `<script>` element.
- This, just like an external CSS file, should also be place in the `<head>` element.

# External JavaScript

- Put the following in your `index.html` file and refresh your browser:

```
<script src='myScript.js' defer></script>
```

# JavaScript Running Order

- When the browser encounters a block of JavaScript, it generally runs it in order, from top to bottom.
- This means that you need to be careful what order you put things in.
- For example, let's return to the block of JavaScript we saw earlier on slide 9.

# JavaScript Running Order

- Here we are selecting a text paragraph (line 1), then attaching an **event listener** to it (line 3) so that when the paragraph is clicked, the **updateName( )** code block (lines 5–8) is run.
- The **updateName( )** code block (these types of *reusable* code block are called “functions”) asks the user for a new name, and then inserts that name into the paragraph to update the display.

# JavaScript Running Order

- If you swapped the order of the first two lines of code, it would no longer work — instead, you'd get an error returned by the browser developer console — **`TypeError: para is undefined`**.
- This means that the `para` object does not exist yet, so we can't add an event listener to it.

# Internal JavaScript

- First, find the file `apply-javascript.html`. Save it in a directory you can locate.
- Open the file in your web browser to see a simple web page containing a clickable button.
- Next, go to your text editor and add the following just before your closing `</body>` tag:

```
<script >  
  // JavaScript goes here  
</script >
```



# Internal JavaScript

- Now add some JavaScript inside our `<script>` element—add the following code just below the *“// JavaScript goes here”* line:

```
1 function createParagraph() {  
2   var para = document.createElement('p');  
3   para.textContent = 'You clicked the  
4                       button!';  
5   document.body.appendChild(para);  
6 }
```

```
7
8 var buttons = document.querySelectorAll(
9     'button'
10    );
11
12 for (var i = 0; i < buttons.length; i++) {
13     buttons[i].addEventListener(
14         'click',
15         createParagraph
16     );
17 }
```

Refresh your browser and click on the button.

# Inline JavaScript Handlers

Occasionally, you'll come across bits of actual JavaScript code living inside HTML. It might look something like this:

```
<button onclick = 'createParagraph()'>  
    Click me!  
</button>
```

# Inline JavaScript Handlers

It is *bad practice* to pollute your HTML with JavaScript, and it is inefficient—you'd have to include the `onclick="createParagraph()"` attribute on every button you wanted the JavaScript to apply to! 🚔

# Interpreted versus Compiled Code

- JavaScript is an **interpreted language**—meaning the result of running the code is immediately returned. You don't have to transform the code into a different form before the browser runs it.
- **Compiled languages** on the other hand are transformed (compiled) into another form before they are run by the computer.
- For example **C/C++** are compiled into assembly language that is then run by the computer.

# Server-side versus Client-side Code

- **Client-side code** is code that is run on the user's computer — when a web page is viewed, the page's client-side code is downloaded, then run and displayed by the browser. The example we have seen so far is explicitly a *client-side* JavaScript.
- **Server-side code** on the other hand is run on the server, then its results are downloaded and displayed in the browser.

# Server-side versus Client-side Code

- Examples of popular server-side web languages include **PHP**, **Python**, **Ruby**, and **ASP.NET**...etc., and **JavaScript** (yes!)
- JavaScript can also be used as a server-side language, for example in the popular **Node.js** environment.

# Dynamic versus Static

- The word **dynamic** is used to describe both client-side JavaScript, and server-side languages.
- It refers to the ability to update the display of a web page/app to show different things in different circumstances, generating new content as required.



# Dynamic versus Static

- Server-side code dynamically generates new content on the server, e.g. pulling data from a database, whereas client-side JavaScript dynamically generates new content inside the browser on the client, e.g. creating a new HTML table, inserting data requested from the server into it, then displaying the table in a web page shown to the user.
- A web page with no dynamically updating content is referred to as **static**—it just shows the same content all the time.

# Comments

- A single line comment is written after a double forward slash (*//*), e.g.

```
// I am a comment
```

- A multi-line comment is written between the strings */\** and *\*/*, e.g.

```
1 /*  
2   I am also  
3   a comment  
4 */
```

# Web Browser Developer Tools

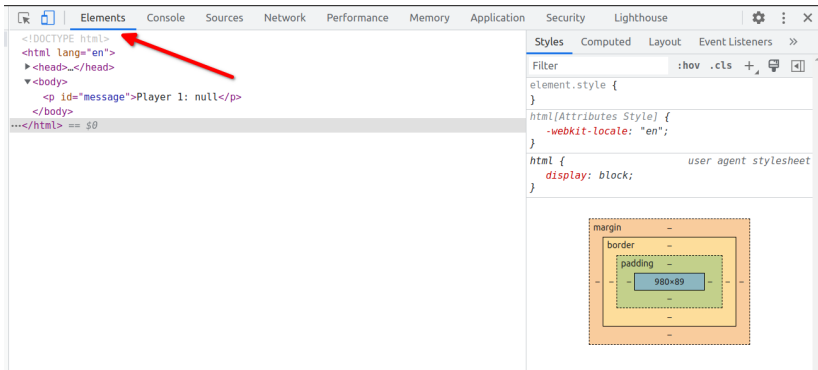
---

# Web Browser Developer Tools

- Browsers are built to abide by standards such as HTML and CSS.
- However, there are many differences in both the interpretation of these standards and in the tooling available with each major web browser.
- When writing JavaScript for the web browser, it is important to know how to access and use the browser developer tools—*especially* the JavaScript console tab.
- On modern browsers access is with key combination **Ctrl+Shift+I**.

# The Elements View

- The primary view that you'll be presented with when exploring browser developer tools for the first time will most likely be the Elements view.
- This view is super useful as it presents all the elements of a web document and the associated content and attributes in a very structured way.
- You will also notice that the various *styles* and *event listeners* will be available for you to explore within this view.



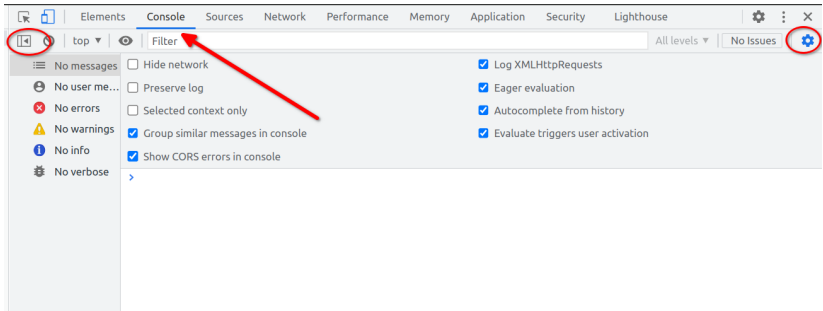
# The Elements View

- One also has access to the entire DOM structure here and can monitor this structure and the associated attributes to verify and explore changes made via code.

# The Console View

- This is the most important view when writing and testing JavaScript code.
- Any errors and warnings will be displayed within this view, and you can also get the output on whatever data you wish as your code executes within the document.





# The Console View

- Using a JavaScript method such as `console.log()` will display output of all sorts of useful data for you to explore within the Console view, and you can even customize exactly the sort of data that is shown through various options associated with the view itself.
- Every web browser has a Console view and even though specific use of this view may differ between browsers, the fundamental usage remains the same.

# The Console View

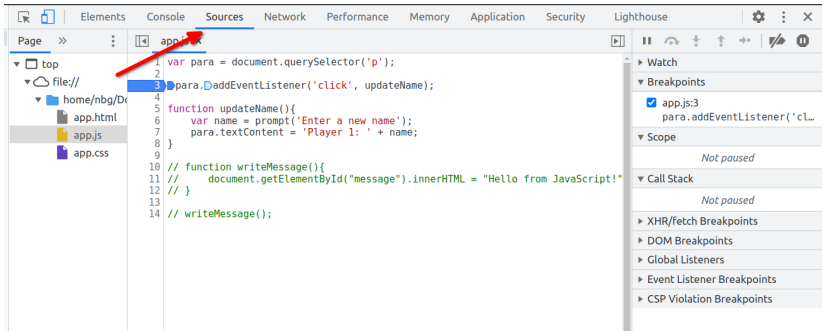
**Exercise:** Track execution of example code.

# The Sources View

- When it comes to any sort of programming, the ability to set breakpoints to effectively pause code execution and debug your program at a certain state is critical.
- Using the source view, we can do this effectively, right within the web browser itself.

# The Sources View

- This view provides a way for us to choose to view the source code for any HTML or JavaScript files that are currently running and set breakpoints at specific lines in order to cause the runtime to pause when the breakpoint is encountered.
- Once paused, we can then use additional tools within the source's view to examine our code in certain ways:



# The Sources View

- In the preceding screenshot, we have set a breakpoint at **line 3** of the JavaScript file.
- With the code execution paused at this specific line, we can examine the state of our program in a very detailed way.

# The Network View

- This allows you to keep tabs on everything being transferred as part of your application.
- HTML documents, JavaScript files, CSS files, and even invisible content such as **XMLHttpRequests** (XHR) and other behind the scenes data transmissions are all logged and measured here for inspection.
- If you want to see a specific type of network activity and hide all the others, there is even a handy filter along the top:



The screenshot shows the Chrome DevTools Network tab. The 'Disable cache' checkbox is circled in red, and a red arrow points to it from the right. The 'Preserve log' checkbox is also circled in red. The 'Filter' input field is empty. The 'Invert', 'Hide data URLs', and 'All' filters are selected. The 'Fetch/XHR', 'JS', 'CSS', 'Img', 'Media', 'Font', 'Doc', 'WS', 'Wasm', 'Manifest', and 'Other' filters are also visible. The 'Has blocked cookies', 'Blocked Requests', and '3rd-party requests' checkboxes are unchecked. The timeline shows two requests: 'app.css' (435 B, 12 ms) and 'app.js' (331 B, 15 ms). The 'Waterfall' view is visible on the right.

Name	Status	Type	Initiator	Size	Time	Waterfall
app.css	200	stylesheet	app.html	435 B	12 ms	
app.js	200	script	app.html	331 B	15 ms	

# The Network View

- One of the important aspects of the Network view that you'll want to note is that **Disable cache** is a tool option.
- Disabling the browser cache is an especially good idea if you are making many changes to externally loaded `.js` files while testing your program as it will prevent these files from being cached by the browser while testing.

# HTML & JavaScript



---

# HTML Element Manipulation with JavaScript

**Exercise:** Add to List.

# JavaScript Frameworks

---

 Remote jobs

**Experienced TYPO3 Developer (m/w/d) - REMOTE**  
signundsinn GmbH  
📍 No office location  
📶 REMOTE  
html typo3

**Senior React Frontend Engineer**  
Namaste Technologies 📍 Toronto, ON, Canada  
\$45K - \$80K 📶 REMOTE  
javascript reactjs

**Web Development Course Mentor**  
Thinkful Inc. 📍 No office location  
\$10K - \$32K 📶 REMOTE  
javascript reactjs

**Senior Front End Developer - United States**  
Mobiquity 📍 No office location  
📶 REMOTE  
javascript html

Stack Overflow, 19/10/2019

# What are JavaScript Frameworks?

- Frameworks are used to help write JavaScript.
- They are tools that you use to help perform common JavaScript tasks and sometimes to simplify tasks that are difficult to perform using JavaScript alone.

# What are JavaScript Frameworks?

- There are quite a number of JavaScript frameworks available today.
- Some include **jQuery**, **AngularJS**, **React**, **Vue.js** the **Yahoo! User Interface** (YUI) library, etc.



See you next week, God willing 🙏