CSE 106

Lecture 7 – JavaScript

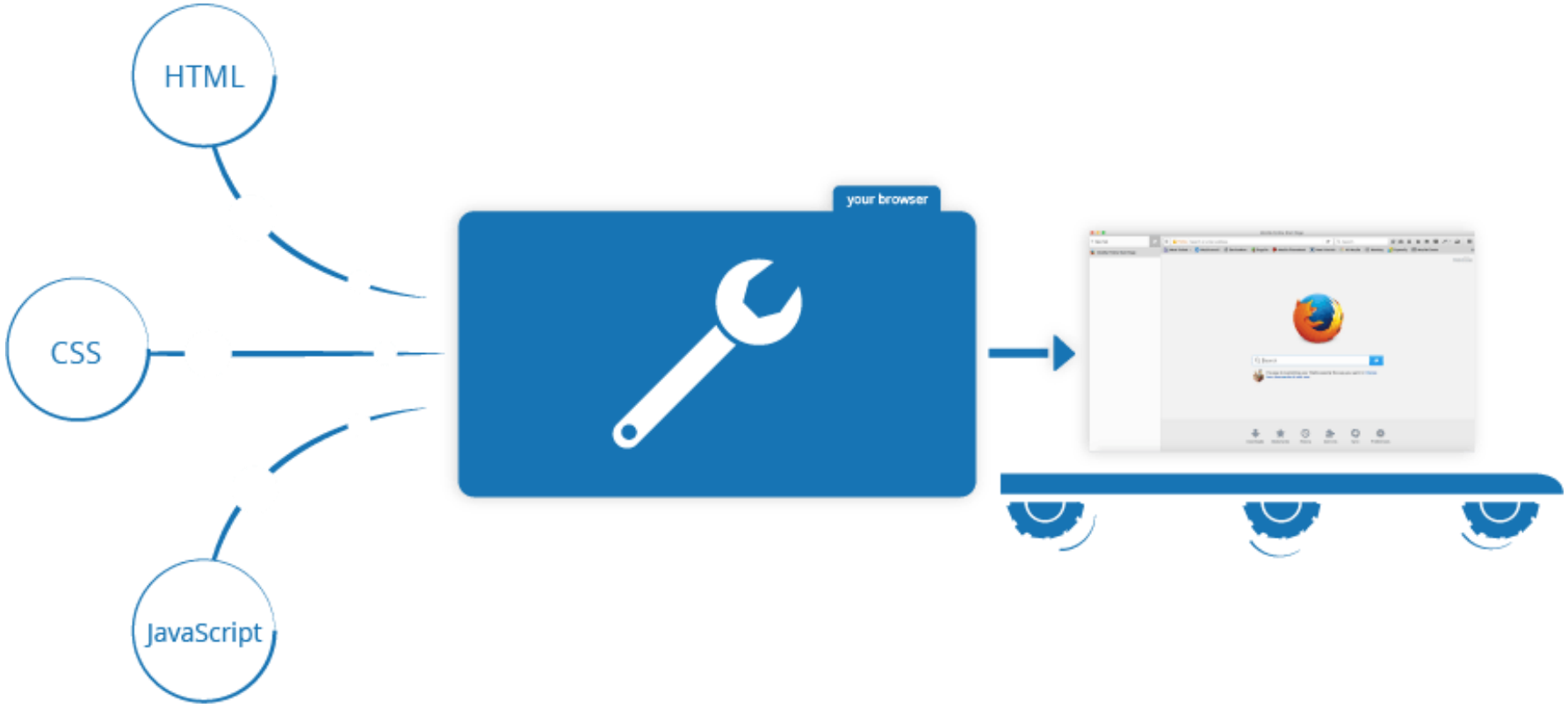
Acknowledgement: w3schools.com, developer.mozilla.org

JavaScript (JS)

- The programming language of the web
- A completely different language than Java
- Interpreted by the browser
- Enables dynamic webpages
- Primarily a frontend language until the advent of Node.js
  - Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine
  - Node.js enables JavaScript for the backend

Loading a web page in a browser

- Runs your code (HTML, CSS, JavaScript) in an execution environment



Dynamic websites

- A static website just sits there (HTML and CSS alone)
- A dynamic web page shows different things in different circumstances
- JavaScript let’s you dynamically modify HTML and CSS to update a user interface, via the Document Object Model (DOM) API
- JavaScript enables you to:
  - Update content
  - Do animation
  - Receive events (i.e. callbacks)
  - And more

Document Object Model (DOM)

- Represents the structure of an HTML document in memory
- Represents a document with a logical tree
  - Each branch of the tree ends in a node
  - Each node contains objects
  - Nodes can have event handlers attached to them
- All parts of the document (the head, tables, text, etc) are parts of the DOM for that document
- DOM methods allow programmatic access to the tree

Document Object Model (DOM)

- The DOM is not a programming language
- The DOM is not part of the JavaScript language, but is a Web API
- The DOM is independent of any particular programming language, but typically you will access it through JavaScript
- JavaScript uses the DOM to access the document and its elements

Document Object Model (DOM)

```
<html>
  <head>
    <script>
       window.onload = function() {
         const heading = document.createElement("h1");
         const heading_text = document.createTextNode("Big Head!");
         heading.appendChild(heading_text);
         document.body.appendChild(heading);
       }
    </script>
  </head>
</html>
```

Where to put JS? - Internal JavaScript

- In between <script> </script> in HTML page
- Can be placed in the <body> or <head> sections or both

```
<body>

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

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

  <script>

    function myFunction() {

      document.getElementById("demo").innerHTML = "Paragraph changed.";

    }

  </script>

</body>
```

Where to put JS? - External JavaScript

- Place JS in external file and reference it with <script> tag

```
<body>
  <p id="demo">A Paragraph.</p>
  <button type="button" onclick="myFunction()">Try it</button>
  <script src="myScript.js"></script>
</body>

// External file: myScript.js
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

External JavaScript

- Placing scripts in external files has some advantages:
  - It separates HTML and code
  - It makes HTML and JavaScript easier to read and maintain
  - Makes your JavaScript more reusable
  - Cached JavaScript files can speed up page loads
- To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

JavaScript Output

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using `innerHTML`
  - Writing into the HTML output using `document.write()`
  - Writing into an alert box, using `window.alert()`
  - Writing into the browser console, using `console.log()`

JavaScript Output Using innerHTML

- To access an HTML element, use document.getElementById(id)
  - The **id** attribute defines the HTML element
  - The **innerHTML** property defines the HTML content

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

<script>
  document.getElementById("demo").innerHTML = "Hello world";
</script>
```

JavaScript Output Using document.write()

- For testing purposes, it is convenient to use document.write()
- Using it after an HTML document is loaded deletes all existing HTML

```
<h1>My First Web Page</h1>

<p>My first paragraph.</p>

<script>

  document.write("Hello world");

</script>
```

JavaScript Output Using alert()

- Use an alert box to display data:

```
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
  alert("Hello world");
</script>
```

JavaScript Output Using console.log()

- Call the console.log() method in the browser to display data:

```
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
  console.log("Hello world");
</script>
```

Wake-up!

- https://youtu.be/6Z-y5-7Ywko

JavaScript Statements

- Semicolons separate JavaScript statements
- JavaScript ignores whitespace and line breaks
- JavaScript statements can be grouped in code blocks, with {...}

```
<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>
```

JavaScript Variables

- JavaScript is not statically typed – no need to specify type
- The general rules for constructing names for variables are:
  - Can contain letters, digits, underscores, and dollar signs
  - Names must begin with a letter
  - Names are case sensitive
  - JavaScript keywords cannot be used as name
- JavaScript use the keywords var, let and const to declare variables

```
var x = 10;

let name = "John Doe";

const PI = 3.14159265359;
```

JavaScript Let

- The let keyword was introduced in ES6 (2015)
- Before ES6, JS had only Global Scope and Function Scope
- Variables defined with **`let`**
  - have block scope
  - cannot be redeclared
  - must be declared before use

JavaScript Let

- Variables declared with `let` inside a { } block **cannot** be accessed from outside the block, not the case with `var`

```
{
  let x = 2;
}
// x CANNOT be used here

{
  var x = 2;
}
// x CAN be used here
```

JavaScript Let

- Variables defined with `let` cannot be redeclared

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has already been declared


var x = "John Doe";

var x = 0;
```

JavaScript Let

- Redeclaring a variable inside a block is different with `var` vs `let`:

```
var x = 10;
// Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is 2
```

```
let x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

JavaScript Let

- Variables defined with var are hoisted to the top and can be initialized at any time
- i.e. You can use the variable before it is declared:

```
carName = "Volvo";
var carName;
```

JavaScript Const

- Same rules as let, but const:
  - Cannot be reassigned
  - Must be assigned

```
const PI = 3.141592653589793;
PI = 3.14;      // This will give an error

const PI; // This is incorrect
PI = 3.14159265359;
```

JavaScript Const

- The keyword const is a little misleading
  - It does not define a constant value
  - It defines a constant reference to a value
- Because of this you can NOT:
  - Reassign a constant value
  - Reassign a constant array
  - Reassign a constant object
- But you CAN:
  - Change the elements of constant array
  - Change the properties of constant object

JavaScript Const

- As a general rule, always declare a variable with `const` unless you know that the value will change
- Use const when you declare:
  - A new Array
  - A new Object
  - A new Function

JavaScript Constant Array

- You can change the elements of a constant array:

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";

// You can add an element:
cars.push("Audi");
```

JavaScript Constant Object

- You can change the properties of a constant object:

```
// You can create a const object:
const car = {type:"Fiat", model:"500", color:"white"};
// You can change a property:
car.color = "red";
// You can add a property:
car.owner = "Johnson";
// Or do it this way:
car["year"] = 1982;
console.log(car);
```

JavaScript Functions

- Defined with the function keyword, followed by a name, followed by parentheses()

```
function multiply(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
let v1 = 4;
let v2 = 5;
console.log(multiply(v1, v2));
```

JavaScript Object Functions (Methods)

- Methods are actions that can be performed on objects

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
console.log(person.fullName());
```