JAVASCRIPT: CLIENT-SIDE SCRIPTING

What is JavaScript

- JavaScript runs right inside the browser
- JavaScript is dynamically typed
  a) **HTML** to define the content of web pages
  b) **CSS** to specify the layout of web pages
  c) **JavaScript** to program the behavior of web pages
- JavaScript is object oriented in that almost everything in the language is an object

JavaScript is not Java

It's not Java

- Although it contains the word *Java*, JavaScript and Java are vastly different programming languages with different uses. Java is a full-fledged compiled, object-oriented language, popular for its ability to run on any platform with a JVM installed.
- Conversely, JavaScript is one of the world's most popular languages, with fewer of the object-oriented features of Java, and runs directly inside the browser, without the need for the JVM.

Client-Side Scripting

It's good

- There are many **advantages** of client-side scripting:
- Processing can be offloaded from the server to client machines, thereby reducing the load on the server.
- The browser can respond more rapidly to user events than a request to a remote server ever could, which improves the user experience.
- JavaScript can interact with the downloaded HTML in a way that the server cannot, creating a user experience more like desktop software than simple HTML ever could.

Client-Side Scripting

There are challenges

- The **disadvantages** of client-side scripting are mostly related to how programmers use JavaScript in their applications.
- There is no guarantee that the client has JavaScript enabled
- The idiosyncrasies between various browsers and operating systems make it difficult to test for all potential client configurations. What works in one browser, may generate an error in another.
- JavaScript-heavy web applications can be complicated to debug and maintain.

JavaScript History

- JavaScript was introduced by Netscape in their Navigator browser back in 1996.
- JavaScript is in fact an implementation of a standardized scripting language called **ECMAScript**
- JavaScript was only slightly useful, and quite often, very annoying to many users

JavaScript in Modern Times

AJAX

- JavaScript became a much more important part of web development in the mid 2000s with **AJAX**.
- **AJAX** is both an acronym as well as a general term.
- As an acronym it means **A**synchronous **J**avaScript **A**nd **X**ML.
- The most important feature of AJAX sites is the asynchronous data requests.

Adding a JS file

```
<html>
    <head>
        <meta charset="utf-8"/>
        <script src="scripts/simple.js"></script>
    </head>
    <body>
        <p>This is a paragraph</p>
    </body>
</html>
```

simplejs.html

Inside the js file

```
alert("My first javascript call!");
```

Adding two js files!

```
<html>
    <head>
        <meta charset="utf-8"/>
        <script src="scripts/simple.js"></script>
        <script src="scripts/second.js"></script>
    </head>
    <body>
        <p>This is a paragraph</p>
    </body>
</html>
```

Open Developer Tools in Browser

- Firefox – Press F12
- In Safari

https://coolestguidesontheplanet.com/safari-web-developer-tools-show-dock-browser-window/

Users Without Javascript

They do exist

- **Web crawler**. A web crawler is a client running on behalf of a search engine to download your site, so that it can eventually be featured in their search results.
- **Browser plug-in**. A browser plug-in is a piece of software that works within the browser, that might interfere with JavaScript.
- **Text-based client**. Some clients are using a text-based browser. (like Lynx, and WebIE)
- **Visually disabled client**. A visually disabled client will use special web browsing software to read the contents of a web page out loud to them.

The <noscript> tag

Mechanism to speak to those without JavaScript

- Any text between the opening and closing tags will only be displayed to users without the ability to load JavaScript.
- It is often used to prompt users to enable JavaScript, but can also be used to show additional text to search engines.
- Requiring JavaScript for the basic operation of your site will cause problems eventually and should be avoided.
- This approach of adding functional replacements for those without JavaScript is also referred to as **fail-safe design**, which is a phrase with a meaning beyond web development.

```
<script src="simple.js"></script>
<noscript>Your browser does not support
JavaScript!</noscript>
```

Where does JavaScript go?

- JavaScript can be linked to an HTML page in a number of ways.
- Inline
- Embedded
- External

Embedded JavaScript

Better

- Embedded JavaScript refers to the practice of placing JavaScript code within a <script> element

```
<script type="text/javascript">
/* A JavaScript Comment */
alert ("Hello World!");
</script>
```

LISTING 6.2 Embedded JavaScript example

External JavaScript

Better

- JavaScript supports this separation by allowing links to an external file that contains the JavaScript.
- By convention, JavaScript external files have the extension .js.

```
<html>
    <head>
        <meta charset="utf-8"/>
        <script src="scripts/simple.js"></script>
    </head>
    <body>
        <p>This is a paragraph</p>
    </body>
</html>
```

Comments in JS

```
/* This is a comment */
alert("My first javascript call!"); // This is also a comment
```

```
// second.js

alert("My second javascript call!");
/* Semicolons will separate
statements
*/
alert("this");
alert("can");
alert("get annoying");
```

JavaScript Syntax

- We will briefly cover the fundamental syntax for the most common programming constructs including
- **variables**,
- **assignment**,
- **conditionals**,
- **loops**, and
- **arrays**
- before moving on to advanced topics such as **events** and **classes**.

Variables

`var`

- **Variables** in JavaScript are **dynamically typed**, meaning a variable can be an integer, and then later a string, then later an object, if so desired.
- This simplifies variable declarations, so that we do not require the familiar type fields like *int*, *char*, and *String*. Instead we use **let**
- **Assignment** can happen at declaration-time by appending the value to the declaration, or at run time with a simple right-to-left assignment

Variables - example

```
//vars.js

"use strict"; //This is very important! Do not forget this!

let name;
let age;
let greeting;

name = "Vivek";
age = 178;
greeting = "Hello!";

alert(greeting + " " + name);
alert ("At " + age + ", you are very old");
```

vars.html

Variable Naming

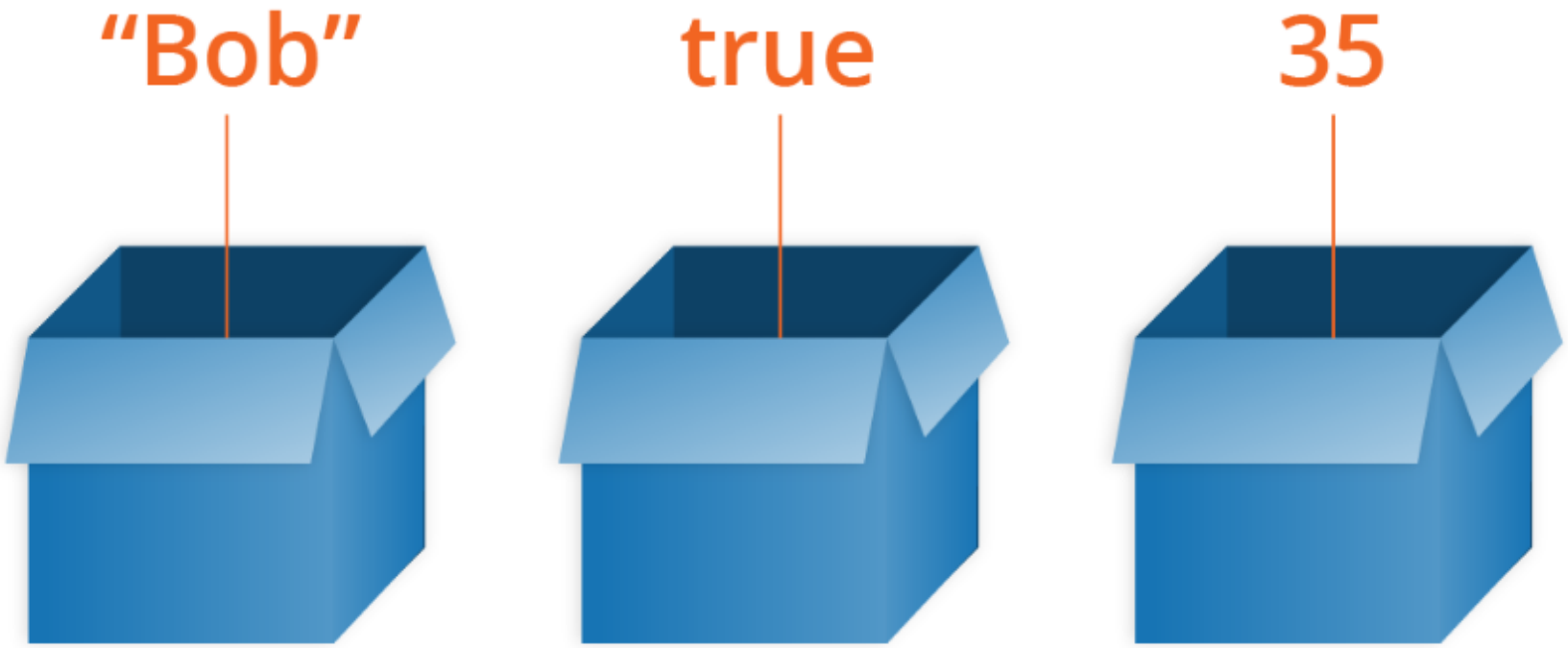
- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this lecture)
- Names are case sensitive (`y` and `Y` are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

Old Javascript & New Javascript

- `"use strict";`

- `"let"` is the modern way
- `"var"` is the old way

Dynamically Typed



Assignment Operators

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Example

```
let txt = “What a nice”;

txt += “day!!”;

console.log(txt);

>> What a nice day!!


let x = 5;

let y = “Hello”;

let z = x + y;
```

What will be the value of z?

console.html

Comparison Operators

- `let x = 9;`
- True or not True

Operator	Description	Matches (let x=9)
==	Equals	(x==9) is true (x=="9") is true
===	Exactly equals, including type	(x==="9") is false (x===9) is true
< , >	Less than, Greater Than	(x<5) is false
<= , >=	Less than or equal, greater than or equal	(x<=9) is true
!=	Not equal	(4!=x) is true
!==	Not equal in either value or type	(x!=="9") is true (x!==9) is false

Logical Operators

- The Boolean operators and, or, and not and their truth tables are listed in the Table. Syntactically they are represented with && (and), || (or), and ! (not).

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND Truth Table

A	B	A || B
T	T	T
T	F	T
F	T	T
F	F	F

OR Truth Table

A	! A
T	F
F	T

NOT Truth Table

TABLE 6.2 AND, OR, and NOT Truth Tables

Conditionals

If, else if, …, else

- JavaScript's syntax is almost identical to that of Java, or C when it comes to conditional structures such as if and if else statements. In this syntax the condition to test is contained within ( ) brackets with the body contained in { } blocks.

```
let name = prompt('What is your name?');

if (name === 'Adam') {
    alert('Hello Adam, nice to see you!');
} else if (name === 'Alan') {
    alert('Hello Alan, nice to see you!');
} else if (name === 'Bella') {
    alert('Hello Bella, nice to see you!');
} else {
    alert('Hello, I have not met you before!');
}
```

Loops

Round and round we go

- Like conditionals, loops use the ( ) and { } blocks to define the condition and the body of the loop.
- You will encounter the **while** and **for** loops
- While loops normally initialize a **loop control variable** before the loop, use it in the condition, and modify it within the loop.

```
let i=0;    // initialise the Loop Control Variable

while(i < 10){  //test the loop control variable

        i++;    //increment the loop control variable

}
```

For Loops

Counted loops

- A **for loop** combines the common components of a loop: initialization, condition, and post-loop operation into one statement.
- This statement begins with the **for** keyword and has the components placed between **( )** brackets, semicolon (**;**) separated as shown

```
for (let i = 0; i < 10; i++){

// your code comes here

}
```

Functions

- **Functions** are the building block for modular code in JavaScript, and are even used to build **pseudo-classes**, which you will learn about later.
- They are defined by using the reserved word **function** and then the function name and (optional) parameters.
- Since JavaScript is dynamically typed, functions do not require a return type, nor do the parameters require type.

Functions

Example

- Therefore a function to raise x to the yth power might be defined as:

```
function power(x,y){
        let pow=1;
        for (let i=0;i<y;i++){
                pow = pow*x;
        }
        return pow;
}
```

- And called as

```
power(2,10);
```

Undefined

Example

- A variable without a value has the value `undefined`

```
let car;
```

- The `typeof` of an undefined variable is also `undefined`

```
typeof car;

>> undefined
```

Try By Yourself

- What happens when you add

`"2" + 2`?

`0.1 + 0.1 + 0.1` ?

What happens when you try to compare

`"2"` with `2`? (two forms of comparison: `==` and `===`)

What is the simplest function, you can think of?

Can you compare them for equality?

JAVASCRIPT OBJECTS

JavaScript Objects

- Objects can have **constructors**, **properties**, and **methods** associated with them.
- There are objects that are included in the JavaScript language; you can also define your own kind of objects.

JavaScript Objects

- Objects are written as `name:value` pairs, separated by commas

```
var person = {
    firstName : "Vivek",
    lastName  : "Nallur",
    age       : 150,
    eyeColor  : "brown"
};
```

Objects - Example

```
//objects.js
"use strict";

let person = {
    firstName   : "Vivek",
    lastName    : "Nallur",
    age         : 150,
    eyeColour   : "Brown"
};

let secondPerson = {
    firstName   : "Abey",
    lastName    : "Campbell",
    age         : 15,
    eyeColour   : "Green"
};

alert ("The first object is called: " + person.firstName);
alert ("The second object has " + secondPerson.eyeColour + " eyes");
```

objects.html

Constructors

- Normally to create a new object we use the new keyword, the class name, and ( ) brackets with *n* optional parameters inside, comma delimited as follows:

```
let someObject = new ObjectName(p1,p2,..., pn);
```

- For some classes, shortcut constructors are defined

```
let greeting = "Good Morning";
```

- vs the formal:

```
let greeting = new String("Good Morning");
```

Properties

Use the dot

- Each object might have properties that can be accessed, depending on its definition.
- When a property exists, it can be accessed using **dot notation** where a dot between the instance name and the property references that property.
- *//show someObject.property to the user*

```
alert(someObject.property);
```

Methods

Use the dot, with brackets

- Objects can also have methods, which are **functions** associated with an instance of an object. These methods are called using the same dot notation as for properties, but instead of accessing a variable, we are calling a method.
- `someObject.doSomething();`
- Methods may produce different output depending on the object they are associated with because *they can utilize the internal properties of the object.*

Objects Included in JavaScript

- A number of useful objects are included with JavaScript including:
- Array
- Boolean
- Date
- Math
- String
- Dom objects

Arrays

- Arrays are one of the most used data structures. In practice, this class is defined to behave more like a linked list in that it can be resized dynamically, but the implementation is browser specific, meaning the efficiency of insert and delete operations is unknown.
- The following code creates a new, empty array named greetings:
- let greetings = new Array();

Arrays

Initialize with values

- To initialize the array with values, the variable declaration would look like the following:

```
let greetings = new Array("Good Morning",
"Good Afternoon");
```

- or, using the square bracket notation:

```
let greetings = ["Good Morning", "Good
Afternoon"];
```

Arrays

Access and Traverse

- To access an element in the array you use the familiar square bracket notation from Java and C-style languages, with the index you wish to access inside the brackets.
-
```
alert ( greetings[0] );
```
- One of the most common actions on an array is to traverse through the items sequentially. Using the Array object's **length** property to determine the maximum valid index. We have:

```
for (let i = 0; i < greetings.length; i++){

    alert(greetings[i]);

}
```

Arrays

Modifying an array

- To add an item to an existing array, you can use the **push** method.

```
greetings.push("Good Evening");
```

- The `pop()` method can be used to remove an item from the back of an array.
- Additional methods: `concat()`, `slice()`, `join()`, `reverse()`, `shift()`, and `sort()`

Math

- The **Math class** allows one to access common mathematic functions and common values quickly in one place.
- This static class contains methods such as max(), min(), pow(), sqrt(), and exp(), and trigonometric functions such as sin(), cos(), and arctan().
- Many mathematical constants are defined such as PI, E, SQRT2, and some others
- **Math.PI;** *// 3.141592657*
- **Math.sqrt(4);** *// square root of 4 is 2.*
- **Math.random();** *// random number between 0 and 1*

String

- The **String class** has already been used without us even knowing it.
- **Constructor usage**
- let greet = new String("Good"); *// long form constructor*
- let greet = "Good"; *// shortcut constructor*
- **Length of a string**
- alert (greet**.length**); *// will display "4"*

String

Concatenation and so much more

- `let str = greet.concat("Morning");` *// Long form concatenation*
- `var str = greet + "Morning";` *// + operator concatenation*
- Many other useful methods exist within the String class, such as
- accessing a single character using `charAt()`
- searching for one using `indexOf()`.
- Strings allow splitting a string into an array, searching and matching with `split()`, `search()`, and `match()` methods.

Date

- The Date class is yet another helpful included object you should be aware of. It allows you to quickly calculate the current date or create date objects for particular dates. To display today's date as a string, we would simply create a new object and use the toString() method.

```
var d = new Date();

alert ("Today is "+ d.toString());

alert("Today is " + d.toUTCString());

alert("Today is " + d.toDateString());
```

Date

- Many methods to retrieve dates in various formats

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Window

- The window object in JavaScript corresponds to the browser itself. Through it, you can access the current page's URL, the browser's history, and what's being displayed in the status bar, as well as opening new browser windows.
- In fact, the alert() function mentioned earlier is actually a method of the window object.
- Window Size

Scope

- Javascript variables have two types of scope:
  - Local scope – inside the function that creates it

```
// code here can not use carName

function myFunction() {
    var carName = "Volvo";

    // code here can use carName

}
```

Scope

- Automatically global!
  - Assigning a value to a variable that has not been declared makes it automatically global

```
myFunction();

// code here can use carName

function myFunction() {
    carName = "Volvo";
}
```

String - basics

- Single quotes, double quotes and backticks – all work

```
let hello  = "Hello";

let world  = 'world';

let helloW = `Hello World`;
```

- Can be created from other types using the `String()` function

```
let intString = String(32); // "32"

let booleanString = String(true); // "true"

let nullString = String(null); // "null"
```

toString

- `toString()` can be used to convert Numbers, Booleans or Objects to Strings

```
var intString = (5232).toString(); // "5232"

var booleanString = (false).toString(); // "false"

var objString = ({}).toString(); // "[object Object]"
```

Concatenating Strings

- Use the `+` operator or the `concat` method

```
let foo = "Foo";

let bar = "Bar";

console.log(foo + bar);          // => "FooBar"

console.log(foo + " " + bar);    // => "Foo Bar"

foo.concat(bar)                  // => "FooBar"

"a".concat("b", " ", "d")        // => "ab d"
```

Template literals

- Can be created using backticks

```
let greeting = `Hello`;
```

- string interpolation using `${variable}` inside template literals

```
let place = `World`;

let greet = `Hello ${place}!`

console.log(greet); // "Hello World!"
```

Reversing Strings

```
function reverseString(str) {

    return str.split('').reverse().join('');

}

reverseString('string');    // "gnirts“
```

- Will NOT work on characters such as diaresis (e.g., ẘ, ǻ, Ḛ̈, ẍ, etc)!

Access character at index in String

- charAt()

```
let string = "Hello, World!";

console.log( string.charAt(4) ); // "o"
```

- Use the [] notation

```
let string = "Hello, World!";

console.log( string[4] ); // "o"
```

- Get the character code using charCodeAt()

```
let string = "Hello, World!";

console.log( string.charCodeAt(4) ); // 111
```

Escaping Quotes

- Single quotes can be escaped using a literal backslash

```
let text = 'L\'albero means tree in Italian';

console.log( text ); \\ "L'albero means tree in Italian“
```

- Same goes for double quotes

```
let text = "I feel \“great\"";
```

- Quotes in HTML can be represented using ' (or ') as a single quote and " (or ") as double quotes

```
var hello = '<p class="special">I&apos;d like
to say &qout;Hi&qout;</p>';
```

That’s all, folks!

- Questions?