Alliance For Digital Employability

Coding Bootcamp

Web Design and Development Fundamentals (Front End)

Lesson 2

(Common)

Alliance For Digital Employability

JavaScript Part I

4.6.1 Code in JavaScript, variables, functions

4.6.2 Code using advance java script: if conditions, loops

Contents

Learning Objectives

Alliance For Digital Employability

- Learn the basics of JavaScript
- Familiarize yourself with the syntax of JavaScript
- Understand the basic usage of JavaScript
- Practice on the fundamental concepts and features of the language
- Code in JavaScript using variables, functions
- Learn how arrays are used in JavaScript
- Understand conditionals, loops
- Familiarize with JavaScript Objects

Syllabus

4.6 JavaScript/ jQuery	4.6.1	Code in JavaScript, variables, functions
	4.6.2	Code using advance java script: if conditions, loops
	4.6.3	Improve usability of forms, validate data from the user

What is JavaScript? (1)

- Introduced by Netscape back in 1996
- A lightweight, interpreted, programming language
- Runs **inside** the browser
- Object-oriented and **prototype-based** rather than class-based as the other OOP languages such as C++, Java and C#

What is JavaScript? (2)

- JS is a prototype-based, multi-paradigm, dynamic scripting language
- It is known as the scripting language for Web pages
- It is used in non-browser environments also, like Node.js and CouchDB

developer.mozilla.org/en-US/docs/Web/JavaScript

Hello Console!

Chrome: View → Developer → JavaScript Console.

Firefox: Tools → Web Developer → Web Console.

Safari: Develop → Show Error Console.

CTRL+SHIFT+J

F12

Clear console:

Clear()

Console.clear()

Some History

Alliance For Digital Employability

JavaScript History I

Alliance For Digital Employability

Ο Παγκόσμιος Ιστός ξεκίνησε ως σελίδες κειμένου που ενώνονταν μεταξύ τους με υπερσυνδέσμους. Οι χρήστες σύντομα ήθελαν περισσότερη αλληλεπίδραση με αυτές τις σελίδες, οπότε η Netscape (πρώιμος κατασκευστής και πωλητής του ομόνυμου προγράμματος περιήγησης) ζήτησε από έναν από τους υπαλλήλους του, τον Brendan Eich, να αναπτύξουν μια νέα γλώσσα για το πρόγραμμα περιήγησης Navigator. Αυτό έπρεπε να γίνει γρήγορα λόγω του έντονου ανταγωνισμού μεταξύ της Netscape και της Microsoft την εποχή εκείνη.

Ο Eich κατάφερε να δημιουργήσει μια πρωτότυπη γλώσσα σε μόλις 10 ημέρες. Για να γίνει αυτό, δανείστηκε διάφορα στοιχεία από άλλες γλώσσες, συμπεριλαμβανομένων των AWK, Java, Perl, Scheme, HyperTalk και Self.

Αυτό ήταν ένα εντυπωσιακό επίτευγμα, αλλά μέσα στη βιασύνη να βγει πρώτη στην αγορά, προέκυψαν (και κληρονομήθηκαν) μια σειρά από ιδιορρυθμίες και σφάλματα στην γλώσσα που ποτέ δεν αντιμετωπίστηκαν πλήρως.

Η νέα γλώσσα αρχικά ονομάστηκε Mocha, στη συνέχεια άλλαξε σε LiveScript, για να αλλάξει ξανά βιαστικά και να ονομαστεί JavaScript, ώστε να μπορέσει να επωφεληθεί από τη δημοσιότητα της γλώσσας Java της Sun Microsystem, που εκείνη την εποχή ήταν σε μεγάλη άνοδο.

Αυτό το όνομα έχει συχνά προκαλέσει κάποια σύγχυση, με τη JavaScript συχνά να θεωρείται ως μια ελαφρύτερη έκδοση της Java. Ωστόσο, οι δύο γλώσσες δεν σχετίζονται - αν και η JavaScript έχει συντακτικές ομοιότητες με την Java.

JavaScript History II

Η JavaScript έκανε το ντεμπούτο της στην έκδοση 2 του προγράμματος πλοήγησης Navigator του Netscape το 1995. Το επόμενο έτος η Microsoft επεξεργάστηκε την JavaScript για να δημιουργήσει μια δική της έκδοση, την οποία αποκάλεσε JScript προκειμένου να αποφύγει ζητήματα πνευματικής ιδιοκτησίας με την Sun Microsystems, στην οποία ανήκε στο εμπορικό σήμα Java. Η JScript ενσωματώθηκε στην έκδοση 3 του Internet Explorer και ήταν σχεδόν πανομοιότυπη με τη JavaScript - είχε όλα τα σφάλματα και ιδιορρυθμίες της Javascript- όμως περιελβανε κάποιες επιπλέον δυνατότητες που αφορούσαν μόνο τον Internet Explorer.

Την ίδια περίοδο η Microsoft συμπεριέλαβε άλλη μια γλώσσα που ονομάζεται VBScript με τον Internet Explorer.

Το 1996, οι Netscape και η Sun Microsystems αποφάσισαν να τυποποιήσουν τη γλώσσα Javascript, μαζί με τη βοήθεια της Ευρωπαϊκής Ένωσης Κατασκευαστών Ηλεκτρονικών Υπολογιστών, ή αγγλιστί European Computer Manufacturers Association, ένωση η οποία θα είναι υπεύθυνη για την ανάπτυξη του προτύπου, του standard με άλλα λόγια της γλώσσας. Η τυποποιημένη γλώσσα ονομάστηκε ECMAScript για να αποφευχθεί η παραβίαση του εμπορικού σήματος Java της Sun. Αυτό προκάλεσε ακόμα μεγαλύτερη σύγχυση, αλλά τελικά το ECMAScript αφορά στις προδιαγραφές και ο όρος JavaScript ήταν (και εξακολουθεί να χρησιμοποιείται) όταν αναφερόμαστε ίδια τη γλώσσα.

What can you build with JavaScript?

Alliance For Digital Employability

- Web Applications:

Photopea: A Photoshop Clone (https://www.photopea.com/)

- Desktop Applications (Using Electron.JS // https://electronjs.org/)

- Skype (https://www.skype.com/en/)
- Slack (https://slack.com/)
- Visual Studio Code (https://code.visualstudio.com/)

- **Browser Extensions** (https://en.wikipedia.org/wiki/Browser_extension)
- **Mobile Apps** (https://facebook.github.io/react-native/)
- **Server Applications** (https://nodejs.org/en/)
- **Command Line Tools** (https://developer.atlassian.com/blog/2015/11/scripting-with-node/)
- **Electronics (Arduino)** (https://www.espruino.com/)
- **Artificial Intelligence / Machine Learning** (https://js.tensorflow.org/)

JavaScript At First Glance

Alliance For Digital Employability

- JavaScript is the programming language of HTML and the Web.
  - Easy To Use
  - Can change HTML Content (Attributes, Elements
  - Can change HTML Styles (CSS)
  - Is Case Sensitive
  - Uses the Unicode character set
- In HTML, JavaScript code must be inserted between <script> and </script> tags

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

  - Scripts can be placed **in** the **<body>,** or in the **<head>** section of an HTML page, or **in both**.

JavaScript features

- It can be used to create
  - sophisticated desktop-like applications that run within the browser
  - mobile applications
  - browsers' extensions
- It is becoming the language of the Internet of Things
- There are plenty of JavaScript frameworks, libraries and plug-ins

Client-Side Scripting (1)

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 (2)

The **disadvantages** of client-side scripting are mostly related to how programmers use JavaScript in their applications

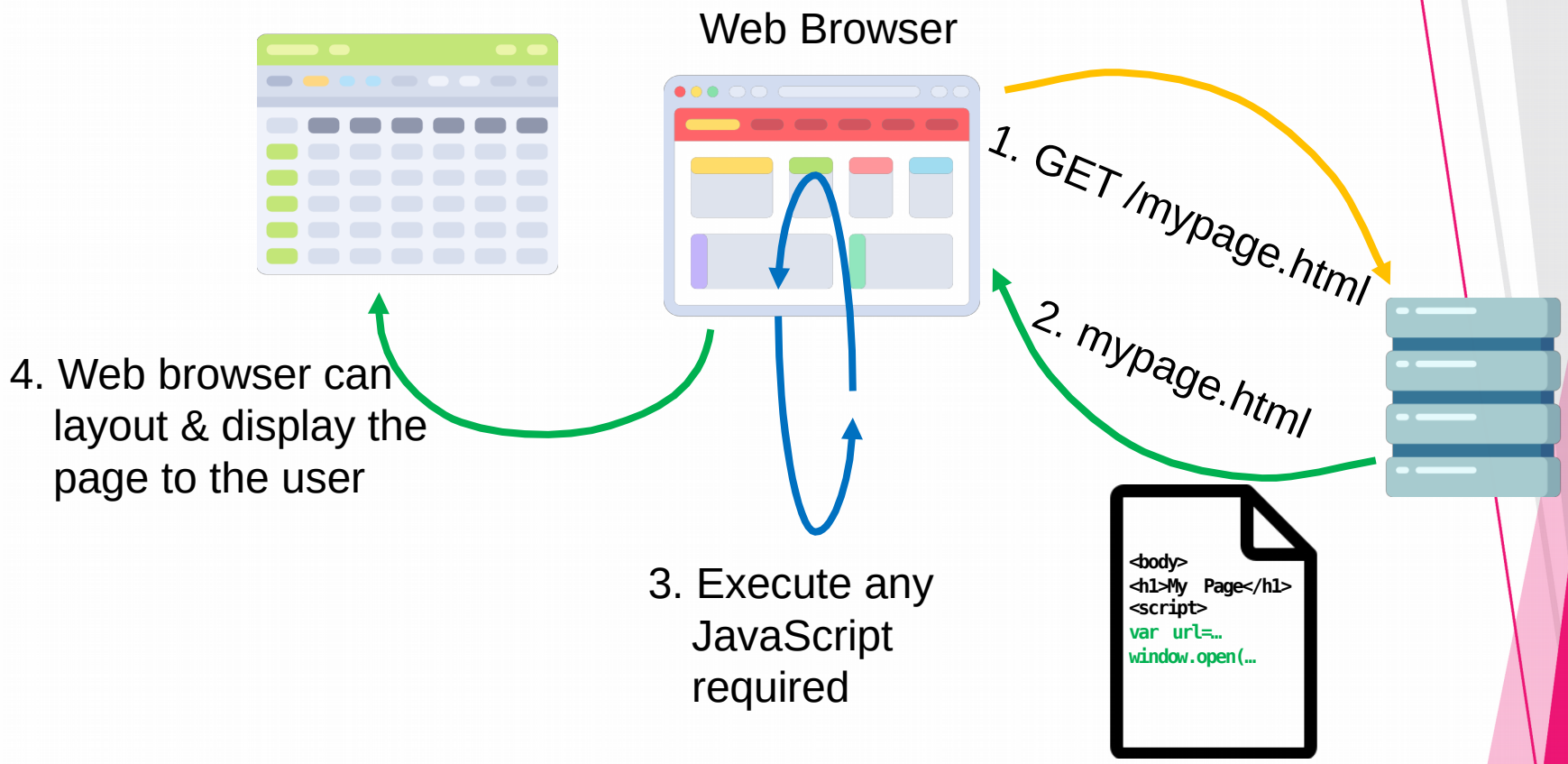
** There is no guarantee that the client has JavaScript enabled

** The differences 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

How JavaScript works?



JavaScript Example

```
const name = “Corto”;
const age = 35;
const greeting = “Hello, my name is “ + name +
“and I am” + age + “ years old.”;

console.log(greeting);
```

Class Exercise #1 <>

Alliance For Digital Employability

- Write your own first JavaScript code snippet, using your own name and age.
- Save as myFirst.js

```
var name = 'John';
var age = 28
var greeting = 'Hello, my name is ' + name + 'and I am' + age + ' years old.';

console.log(greeting);
```

JavaScript Syntax

- JavaScript uses a C-like syntax, like Java and C#.

```
const name = "Mad Max";

// Output 5 times: My name is Mad Max.

for(var i=0; i<5; i++) {
   console.log("My name is "  + name + ".");
}
```

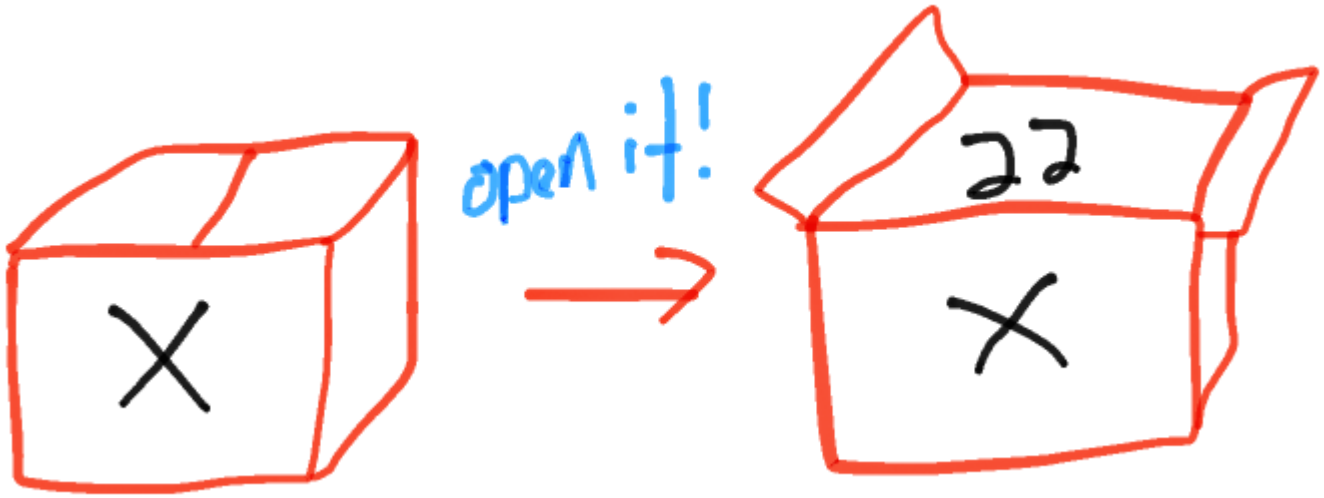
Class Exercise #2 <>

Alliance For Digital Employability

- Write your own JavaScript code snippet, like the one below
- Save as mySecond.js

```
var name = 'Mad Dog';

// Output 5 times: My name is Mad Dog.
for (var i=0; i<5; i++) {
  console.log('My name is ' + name + '.');
}
```

Variables



Variables

- Variables are symbolic names for values in your application.
- Variables are declared with the **var or let** keyword

  (e.g. var n = 5; let z = “Zoe”;).
- 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 var
- 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 naming

- All JavaScript variables must be identified with **unique** names
- These unique names are called identifiers
- General rules:
  - can contain letters, digits, underscores, and dollar signs
  - must begin with a letter (NOT a number)
  - can also begin with $ and _
  - are case sensitive
  - reserved words (like JavaScript keywords) cannot be used as names.
  - Hyphens (-) are not allowed in JavaScript.
- Examples of legal names are **Number_hits, temp99, $credit,** and **playerName.**

Variables naming – reserved words

- break
- case
- catch
- class
- const
- continue
- debugger
- default
- delete
- do
- else
- export
- extends
- finally
- for
- function
- if
- import
- in
- instanceof
- new
- return
- super
- switch
- this
- throw
- try
- typeof
- var
- void
- while
- with
- yield

Declaring Variables

You can declare a variable in three ways:

1. With the keyword var.
   Example: var x = 42.
   This syntax can be used to declare both local and global variables.
2. By simply assigning it a value.
   Example: x = 42.
   This always declares a global variable, if it is declared outside of any function. It generates a strict JavaScript warning. You shouldn't use this variant.
3. With the keyword let.
   Example, let y = 13.
   This syntax can be used to declare a block-scope local variable.

Variables examples

each line should be terminated by semicolon

```
var abc;
var def=0;
def= 4;
def= “hello”;
```

a variable named def initialized with 0

notice that the type of the value assigned to def has changed to string without any special instruction or declaration

Class Exercise #3 <>

Alliance For Digital Employability

- Variable example:

  variables.html

Evaluating Variables (1)

- A variable declared using the var or let statement with no assigned value specified has the value of undefined.
- An attempt to access an **undeclared** variable will result in a **ReferenceError** exception being thrown:

```
1  var a;
2  console.log('The value of a is ' + a); // The value of a is undefined
3
4  console.log('The value of b is ' + b); // The value of b is undefined
5  var b;
6
7  console.log('The value of c is ' + c); // Uncaught ReferenceError: c is not defined
8
9  let x;
10 console.log('The value of x is ' + x); // The value of x is undefined
11
12 console.log('The value of y is ' + y); // Uncaught ReferenceError: y is not defined
13 let y;
```

Evaluating Variables (2)

- You can use **undefined** to determine whether a variable has a value.
- In the following code, the variable input is not assigned a value, and the if statement evaluates to true.

```
1  var input;
2  if (input === undefined) {
3    doThis();
4  } else {
5    doThat();
6  }
```

Evaluating Variables (3)

Also note that:

- The undefined value behaves as false when used in a boolean context.
- The undefined value converts to NaN when used in numeric context.
- When you evaluate a null variable, the null value behaves as 0 in numeric contexts and as false in boolean contexts.

Class Discussion

Alliance For Digital Employability

Consider the following code snippets:

```
var n;
console.log('The value of n is ' + n);


n = 11;
console.log('The value of n is ' + n);
```

What is the value of n in these cases?

Class Discussion – Answer

Alliance For Digital Employability

Consider the following code:

```
var n;
console.log('The value of n is ' + n);
```

//The value of n is undefined

```
n = 11;
console.log('The value of n is ' + n);
```

//The value of n is 11

Class Discussion

Alliance For Digital Employability

Consider the following code snippets:

```
var myArray = [ ];
if (!myArray[0]) myFunction();
```

```
var n = null;
console.log(n * 32);
```

What will happen in these cases?

Class Discussion - Answer

Alliance For Digital Employability

Consider the following code snippets:

```
var myArray = [];

if (!myArray[0]) myFunction();
```

The function myFunction will be executed as myArray is undefined.

```
var n = null;

console.log(n * 32);
```

Will log 0 to the console (behave as 0 in numeric context)

Class Discussion: var VS let

Alliance For Digital Employability

Open a console (F12) and write:

```
var myName = 'Takis';
var myName = 'Anna';

 let myName = 'Takis';
  let myName = 'Anna';
```

Class Discussion: var VS let

1. Using `var`, you can declare the same variable as many times as you like.

You can't do this with `let`

2. **(No) Hoisting in let** *(we will soon see what hoisting is)*

```
yourNumber = 2;
yourNumber;
// ...is understood as:
var yourNumber;
yourNumber = 2;
```

Variable Scope

- When you declare a variable outside of any function, it is called a **global variable**, because it is available to any other code in the current document.
- When you declare a variable within a function, it is called a **local variable**, because it is available only within that function.
- JavaScript before ECMAScript 2015 does not have block statement scope; rather, a variable declared within a block is local to the function (or global scope) that the block resides within.

Global Variables

- **Global variables** are in fact properties of the global object.
- In web pages the global object is window, so you can set and access global variables using the **window.variable** syntax.
- Consequently, you can access global variables declared in one window or frame from another window or frame by specifying the window or frame name.
- For example, if a variable called phoneNumber is declared in a document, you can refer to this variable from an iframe as parent.phoneNumber.

Constants

- You can create a read-only, named constant with the **const** keyword. The syntax of a **constant** identifier is the same as for a variable identifier: it must start with a letter, underscore or dollar sign ($) and can contain alphabetic, numeric, or underscore characters.

Example: **const PI = 3.14;**

- A constant cannot change value through assignment or be re-declared while the script is running. It has to be initialized to a value.
- The scope rules for constants are the same as those for let block-scope variables.
- If the const keyword is omitted, the identifier is assumed to represent a variable.
- You cannot declare a constant with the same name as a function or variable in the same scope.

Variable Scope - Example

- The following code will log 5, because the scope of x is the function (or global context) within which x is declared, not the block, which in this case is an if statement.

```
if (true) {
    var x = 5;
}
console.log(x);  // x is 5
```

But if you use the let declaration this behavior will change:

```
if (true) {
    let y = 5;
}
console.log(y);  // ReferenceError: y is not defined
```

Variable Hoisting

- Can refer to a variable declared later, without getting an exception.
- Concept is known as **hoisting**; variables in JavaScript are in a sense "hoisted" or lifted to the top of the function or statement.
- Note that variables that are hoisted will return a value of undefined.
- So even if you declare and initialize after you use or refer to this variable, it will still return undefined.
- Because of hoisting, all var statements in a function should be placed as near to the top of the function as possible. This best practice increases the clarity of the code.

Variable Hoisting - Example

These examples will be evaluated the same:

/*Example1*/

```
console.log(x === undefined); // true
var x = 3;
```

Same as:

```
var x;
console.log(x === undefined); // true
x = 3;
```

Class Discussion ‹›

What is the evaluation of variable myvar in these cases? Is the evaluation the same in both cases?

```
/*Case1*/
var myvar = 'my value';
 (function() {
  console.log(myvar);
  var myvar = 'local value';
})();

/*Case2*/
var myvar = 'my value';
 (function() {
  var myvar;
  console.log(myvar);
  myvar = 'local value';
})();
```

Class Discussion - Answer

Same in both cases; **undefined**

```
/*Case1*/
var myvar = 'my value';
 (function() {
  console.log(myvar); // undefined
  var myvar = 'local value';
})();

/*Case2*/
var myvar = 'my value';
 (function() {
  var myvar;
  console.log(myvar); // undefined
  myvar = 'local value';
})();
```

JavaScript Types & Expressions

- JavaScript provides a number of different value types, such as number or string
  - The type can be determined using the typeof keyword
- Expressions are combinations of values, variables and operators, which compute to a value
  - For example, 5*20 evaluates to 100
  - Expressions can also contain variable values, for instance, x*100
  - Expressions can be of various types, such as numbers and strings

    Example: "John" + " " + "Doe“ evaluates to “John Doe”

Data structures and types

The latest ECMAScript standard defines **seven data** types:

- Six data types that are **primitives**:
  - Boolean. true and false.
  - null. A special keyword denoting a null value. Because JavaScript is case-sensitive, null is not the same as Null, NULL, or any other variant.
  - undefined. A top-level property whose value is undefined.
  - Number. 42 or 3.14159.
  - String. "Howdy"
  - Symbol (new in ECMAScript 2015). A data type whose instances are unique and immutable.
- **Object**
- Although these data types are a relatively small amount, they enable you to perform useful functions with your applications.
- Objects and functions are the other fundamental elements in the language.
- You can think of objects as named containers for values, and functions as procedures that your application can perform.

Data structures and types

Alliance For Digital Employability

--- **string** // let js = "javascript is everywhere!";

--- **number** // let js = 10.6;

--- **boolean (true or false)** // let z = true;

--- **array** // let myName = ['what', 'is', 'your', 'name'];

--- **object** // var user = {name: 'John', age: 24};

--- **function** // function () { return 4; }

--- **undefined / null**

- **symbol** // let symbol1 = Symbol();

Data Type Conversion

- JavaScript is a dynamically typed language
- Don't have to specify the data type of a variable when you declare it, and data types are converted automatically as needed during script execution

Example:

```
var answer = 42;
```

But later you can do

```
answer = 'Thanks for everything!';
```

Literals

- You use **literals** to represent values in JavaScript. These are **fixed values**, not variables, that you literally provide in your script.
  - Array literals
  - Boolean literals
  - Floating-point literals
  - Integers
  - Object literals
  - RegExp literals
  - String literals
- More on literals in Advanced JavaScript

Comparison Operators

Operator	Description	Matches (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

Equality Operators <>

Alliance For Digital Employability

Equality ==
Inequality !=
Identity / strict equality === (suggested)
Non-identity / strict inequality !== (suggested)

Open your browser's console and try:

```
1 == 1
7 == '7'
1 != 2
5 === 5
9 === '9'
3 !== 3
3 !== '3'
0 !=  false
0 == false
0 == null
let object1 = {'key': 'value'}, object2 = {'key': 'value'};
object1 == object2
```

Assignment Operators

Operator	Example	Is equal
=	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

JavaScript Comments

- Comments are statements that are ignored, and therefore, not executed
  - Single line comments start with //.
  - Multi-line comments start with /* and end with */.
  - Used mostly for explanatory purposes

```
<script>
var x = 5; //Declare x, then give it the value of 5
/* Declare y, then
give it the value of x + 2 */
var y = x + 2;
</script>
```

Strings (1)

- A JavaScript string simply stores a series of characters like "John Doe"
- A string can be any text inside quotes. You can use single or double quotes:

```
<script>
var x = “this is a string”;
</script>
```

- The length of a string is found in the built in property length
- In order to be able to properly display a quote same as used for the string the \ escape character can be used

```
var y = “We are the so-called \”Vikings\” from the north.”
```

(try: `y.length`)

Strings (2)

- Line break can be applied either by using backslash (\) or plus (+)
- When using the == operator, equal strings are equal
- When using the === operator, equal strings are not equal, because the === operator expects equality in both type and value
- JavaScript provides a set of functions for easier string manipulation, such as string splitting, substring searching etc.
- Strings, but also numbers, can also be defined as objects with the keyword new

Numbers (1)

- JavaScript has only one type of number. Numbers can be written with or without decimals

```
var x = 3.14; //A number with decimals
var y = 3; //A number without decimals
```

- Extra large or extra small numbers can be written with scientific (exponent) notation

```
var x = 123e5; //12300000
var y = 123e-5; //0.00123
```

- The maximum number of decimals is **17**, but floating point arithmetic is not always 100% accurate – one must multiply and divide in order to have precision
- JavaScript uses the '+' operator for both number addition and string concatenation

Numbers (2)

- Trying to do arithmetic with a non-numeric string will result to a NaN (not a number), a reserved word indicating an number is not legal
  - “100”/”Apple” will result to NaN
  - JavaScript provides the `isNan(value)` function in order to check if a value is NaN
- Infinity (or -Infinity) is a property of the global object, (or in other words, a variable in global scope). The value `Infinity` (positive infinity) is greater than any other number.

  `Infinity` equals Infinity, and any number divided by Infinity equals 0.

Numbers (3)

- JavaScript provides a set of function for operations on numbers, for instance:
  - specification of a number to exponential or fixed notation (toExponential(value)/toFixed(value))
  - conversion (i.e.: parseInt(stringValue) that converts a string to an integer number)

How can JavaScript be used?

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

Inline JavaScript

- **Inline JavaScript refers** to the practice of **including JavaScript** code directly **within certain HTML attributes**

```
<input type="button" onClick="alert('Are you sure?');" />
```

Embedded JavaScript

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

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

Class Exercise #5

Alliance For Digital Employability

- Open one of your HTML pages (i.e. CV one) and embed a script like the one below to greet your users to your website

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

External JavaScript

- **External JavaScript** files typically contain function definitions, data definitions, and entire frameworks.

```
<head>
    <script type="text/javascript"
src=”myscript.js"></script>
</head>
```

Class Exercise #5

Alliance For Digital Employability

- Create an .js file with a greeting (like variables.html);
- Open one of your HTML pages (i.e. CV one) and use the .js file create above a greet your users to your website

```
<head>
    <script type="text/javascript"
src=”myscript.js"></script>
</head>
```

No JavaScript

- JavaScript can be ignored or be deactivated in some users
  - Web crawlers
  - Text-based clients
  - Visually disabled users using screen readers

- There is the <noscript> tag as fail-safe design

  Any text between the opening and closing tags will only be displayed to users without the ability to load JavaScript

```
 <script>
document.write("You win!")
</script>
<noscript>Your browser does not support JavaScript.
Please enable JS to see our message!</noscript>
```

Booleans

- Very often, in programming, you will need a data type that can only have one of two values, like true or false
- JavaScript has a Boolean data type
  - It can only take the values true or false
- In JavaScript, you can use the **`Boolean(expressionValue)`** function to determine if an expression is true or false

Logical Operators

- The Boolean operators and, or, and not are represented with && (and), || (or), and ! (not)

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

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

A	!A
F	T
T	F

Logical Operators - Exercise

Try:

```
true && false
false && true
false || true
true || false
----------------------
let x = 6;
let y = 3;

x < 10 && y > 1
x === 5 || y === 5
x !== y
----------------------
```

Control flow

- Block Statements
- Conditional Statements
- Exception Handling Statements
- Promises

Block Statements

- The most basic statement is a **block statement** that is used to group statements. The block is delimited by a pair of curly brackets:

```
{
    statement_1;
    statement_2;
    .
    .
    .
    statement_n;
}
```

- Block statements are commonly used with control flow statements

Block Statement - Examples


```
while (x < 10) {
   x++;
}
```

Here, { x++; } is the block statement

- This outputs 2 because the var x statement within the block is in the same scope as the var x statement before the block. In C or Java, the equivalent code would have outputted 1.

```
var x = 1;
{
  var x = 2;
}
console.log(x); // outputs 2
```

Conditional Statements

- A conditional statement is a set of commands that executes if a specified condition is true.
- JavaScript supports two conditional statements:
  - **if...else** : Use the if statement to execute a statement if a logical condition is true. Use the optional else clause to execute a statement if the condition is false.

    and
  - **switch** : A switch statement allows a program to evaluate an expression and attempt to match the expression's value to a case label. If a match is found, the program executes the associated statement.

```
if (condition) {
   statement_1;
} else {
   statement_2;
}
```

```
switch (expression) {
  case label_1:
    statements_1
    [break;]
  case label_2:
    statements_2
    [break;]
    ...
  default:
    statements_def
    [break;]
}
```

Conditionals if else (1)

- JavaScript's syntax is almost identical to that of PHP, 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

```
var hourOfDay;
var greeting;
if (hourOfDay > 4 && hourOfDay < 12) {
greeting = "Good Morning";
}
else if (hourOfDay >= 12 && hourOfDay < 18) {
greeting = "Good Afternoon";
}
else {
greeting = "Good Evening";
}
```

Conditionals if else (2)

- In the case of multiple conditions only the first logical condition which evaluates to true will be executed.
- To execute multiple statements, group them within a block statement ({ ... }) .
- In general, it's good practice to always use block statements, especially when nesting if statements:

```
if (condition) {
  statement_1_runs_if_condition_is_true;
  statement_2_runs_if_condition_is_true;
} else {
  statement_3_runs_if_condition_is_false;
  statement_4_runs_if_condition_is_false;
}
```

Conditionals if else (3)

- It is advisable to not use simple assignments in a conditional expression, because the assignment can be confused with equality when glancing over the code.

For example, do not use the following code:

```
if (x = y) {
  /* statements here */
}
```

- If you need to use an assignment in a conditional expression, a common practice is to put additional parentheses around the assignment.

For example:

```
if ((x = y)) {
  /* statements here */
}
```

Truthiness in Javascript

```
let x;
x = 1;   // x is a number
x = '1'; // x is a string
x = [1]; // x is an array
```

```
// all true
1 == '1';
1 == [1];
'1' == [1];
```

```
// all false
1 === '1';
1 === [1];
'1' === [1];
```

Alliance For Digital Employability

Falsy Values

- The following values evaluate to false (also known as Falsy values):
  - false
  - undefined
  - null
  - 0
  - NaN
  - the empty string (" ")
- All other values, including all objects, evaluate to true when passed to a conditional statement.
- Do not confuse the primitive boolean values true and false with the true and false values of the Boolean object.

Truthiness in Javascript

The following values are always falsy:

- false
- 0 (zero)
- '' or "" (empty string)
- null
- undefined
- NaN

Everything else is truthy. That includes:

'0' (a string containing a single zero)
'false' (a string containing the text “false”)
[] (an empty array)
{} (an empty object)
function(){} (an “empty” function)

Truthiness in JS: Recommendations

Alliance For Digital Employability

```
1. Avoid direct comparisons

// instead of
if (x == false) // ...
// runs if x is false, 0, '', or []

// use
if (!x) // ...
// runs if x is false, 0, '', NaN, null or undefined

2. Use === strict equality

// instead of
if (x == y) // ...
// runs if x and y are both truthy or both falsy
// e.g. x = null and y = undefined

// use
if (x === y) // ...
// runs if x and y are identical...
// except when both are NaN
```

Falsy Values Example

- In the following example, the function checkData returns true if the number of characters in a Text object is three; otherwise, it displays an alert and returns false.

```
function checkData() {
  if (document.form1.threeChar.value.length == 3) {
    return true;
  } else {
    alert('Enter exactly three characters. ' +
    document.form1.threeChar.value + ' is not valid.');
    return false;
  }
}
```

Class Exercise #7

Alliance For Digital Employability

Consider the following code:

```
var b = new Boolean(false);
if (b)
if (b == true)
```

- What do the red expressions evaluate to?

Class Exercise #7 - Answer


```
var b = new Boolean(false);
if (b) // this condition evaluates to true
if (b == true) // this condition evaluates to false
```

Conditionals switch (1)

- Another conditional statement is **switch**
- The program first looks for a case clause with a label matching the value of expression and then transfers control to that clause, executing the associated statements. If no matching label is found, the program looks for the optional default clause, and if found, transfers control to that clause, executing the associated statements. If no default clause is found, the program continues execution at the statement following the end of switch. By convention, the default clause is the last clause, but it does not need to be so.
- The **optional break statement** associated with each case clause ensures that the program breaks out of switch once the matched statement is executed and continues execution at the statement following switch. If break is omitted, the program continues execution at the next statement in the switch statement.
- Its syntax is similar to other programming languages such as Java, PHP and C#

Conditionals switch (2)

- Example:

```
switch (artType) {
   case "PT":
      output = "Painting";
      break;
   case "SC":
      output = "Sculpture";
      break;
   default:
   output = "Other";
}
```

Class Exercise #8

Alliance For Digital Employability

Write a switch statement that will:

- if **fruittype** evaluates to one of the fruits in the list, it displays its price per kg (as table below)

Fruit	Price per kg
Bananas	1,30€
Oranges	0,89€
Apples	1,05€
Cherries	2,99€
Watermelon	1,20€

- When `break` is encountered, the program terminates switch and executes the statement following switch.
- If `break` were omitted, the statement for case "Cherries" would also be executed.
- Add appropriate messages where you deem fit.

Class Exercise #8 - Answer

Alliance For Digital Employability

```
switch (fruittype) {
  case 'Oranges':
    console.log('Oranges are €0.89 a kg');
    break;
  case 'Apples':
    console.log('Apples are €1.05 a kg');
    break;
  case 'Bananas':
    console.log('Bananas are €1.30 a kg.');
    break;
  case 'Cherries':
    console.log('Cherries are €2,99 a kg..');
    break;
  case ‘Watermelon':
    console.log('Watermelon are €1.20 a kg.');
    break;
default:
   console.log('Sorry, we are out of ' + fruittype + '.');
}
console.log("Is there anything else you'd like?");
```

Fruit	Price per kg
Bananas	1,30€
Oranges	0,89€
Apples	1,05€
Cherries	2,99€
Watermelon	1,20€

Exception handling statements

- You can throw exceptions using the throw statement and handle them using the try...catch statements.
  - throw statement
  - try...catch statement

More on these statements in Part II of JavaScript

Conditional Assignment

```
/* x conditional assignment */
x = (y==4) ? “y is 4” : “y is not 4”;
```

condition

value if true

value if false

```
/* equivalent to */
if (y==4) {
   x = “y is 4”;
}
else {
   x = “y is not 4”;
}
```

Loops (1)

- 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

Loops (2)

- A loop as a computerized version of repeating an action some number of times (and it's actually possible that number could be zero). The statements for loops provided in JavaScript are:
  - for statement
  - do...while statement
  - while statement
  - labeled statement
  - break statement
  - continue statement
  - for...in statement
  - for...of statement

`for` loop (1)

- for loop **repeats until** a specified condition evaluates to false. The JavaScript for loop is similar to the Java and C for loop. A for statement looks as follows:

```
for ([initialExpression]; [condition];
[incrementExpression]) statement
```

- 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` loop (2)

for ([initialExpression]; [condition]; [incrementExpression]) statement

When a for loop executes, the following occurs:

- The initializing expression initialExpression, if any, is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity. This expression can also declare variables.
- The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the for loop terminates. If the condition expression is omitted entirely, the condition is assumed to be true.
- The statement executes. To execute multiple statements, use a block statement ({ ... }) to group those statements.
- If present, the update expression incrementExpression is executed.
- Control returns to step 2.

`for` loop - Example

- The for statement counts the number of selected options in a scrolling list (a <select> element that allows multiple selections).
- The for statement declares the variable i and initializes it to zero.
- It checks that i is less than the number of options in the <select> element, performs the succeeding if statement, and increments i by one after each pass through the loop.

`for` loop - Example

https://codesandbox.io/s/for-loop-example-rqd7l?fontsize=14

`for` loop - Exercise

Write a for loop that will iterate from 0 to 20. For each iteration, it will check if the current number is even or odd, and report that to the screen (e.g. "3 is odd").

`for` loop exercise - Solution

```
for (var i = 0; i <= 20; i++) {
     if (i % 2 === 0) {
        console.log(i + ' is even');
    } else {
        console.log(i + ' is odd');
    }
}
```

`while` statement

- A while statement executes its statements as long as a specified condition evaluates to true. A while statement looks as follows:

```
while (condition)
  statement
```

- If the condition becomes false, statement within the loop stops executing and control passes to the statement following the loop.
- The condition test occurs before statement in the loop is executed. If the condition returns true, statement is executed and the condition is tested again. If the condition returns false, execution stops and control is passed to the statement following while.
- To execute multiple statements, use a block statement ({ ... }) to group those statements.

`while` example

https://bit.ly/2SOS06a

What do you think below code will output:

```
while (true) {
  console.log('Hello People of Earth!');
}
```

do…while statement (1)

- The do...while statement **repeats until a specified condition evaluates to false**. A do...while statement looks as follows:

```
do
  statement
while (condition);
```

- statement executes once before the condition is checked. To execute multiple statements, use a block statement ({ ... }) to group those statements.
- If condition is true, the statement executes again.
- At the end of every execution, the condition is checked. When the condition is false, execution stops and control passes to the statement following do...while.

while/do while

```
var count = 0; //initialise the Loop Control Variable
while (count < 10) { //test the loop control variable
        // do something
        // ...
        count++; //increment the loop control variable
}
count = 0; //initialise the Loop Control Variable
do {
        // do something
        // ...
        count++; //increment the loop control variable
} while (count < 10); //test the loop control variable
```

Class Exercise#9

- Write a do…while loop that iterates at least once, increases i by 1 each time and reiterates until I is no longer less than 7

Class Exercise#9 - Answer

- Write a do…while loop that iterates at least once, increases i by 1 each time and reiterates until I is no longer less than 7

```
var i = 0;
do {
  i += 1;
  console.log(i);
} while (i < 7);
```

Arrays

- Arrays are one of the most commonly used data structures in programming
- JavaScript does not have an explicit array data type, like Python or Java
- Use the predefined Array object and its methods to work with arrays in your code.
- JavaScript provides two main ways to define an array
  - object literal notation
  - use the Array() constructor

Arrays literal notation

- The literal notation approach is generally preferred since it involves less typing, is more readable, and executes a little bit quicker

```
var years = [1855, 1648, 1420];
var countries = ["Canada", "France",
      "Germany", "Nigeria",
      "Thailand", "United States"];
var mess = [53, "Canada", true, 1420];
```

Arrays II - array contractor

```
var students = new Array(Kostas
Evripidis, Sakis Rokas, Soula
Tripolitsioti, ..., studentN);
```

```
var students = Array(Kostas Evripidis,
Sakis Roukas, ..., studentN);
```

Arrays common features

- arrays in JavaScript are zero indexed
- [ ] notation for access
- .length – gives the length of the array
- .push() – inserts an item
- .pop() – removes an item
- concat(), slice(), join(), reverse(), shift(), and sort()

Arrays common methods

- Concat()

```
var myArray = new Array('1', '2', '3');

myArray = myArray.concat('4a', '4b', 4'c');

// myArray is now ["1", "2", "3", "4a", "4b",
"4c"]
```

- Slice()

```
Var myArray = new Array('A', 'B', 'C', 'D', 'E');

myArray = myArray.slice(1, 4); // starts at index
1 and extracts all elements
// until index 3, returning [ "B", "C", "D"]
```

Arrays common methods II

- join()

```
var myArray = new Array('Earth', 'Wind', 'Fire');
var list = myArray.join(' - '); // list is "Earth-Wind-Fire"
```

- reverse()

```
var myArray = new Array('1', '2', '3');
myArray.reverse();
// myArray = ["3", "2", "1"]
```

Arrays common methods III

- sort()

```
var myArray = new Array('Wind', 'Rain', 'Fire');
myArray.sort();
// myArray = ["Fire", "Rain", "Wind"]
```

- shift()

```
var myArray = new Array('1', '2', '3');
var first = myArray.shift();
// myArray is now ["2", "3"], first is "1"
```

Array Exercise

Create an array to hold your top 5 choices in music styles (“rock”, “jazz”, “heavy-metal”, “electronica”, “opera”).

For each choice log to the screen a string like "My 1st choice is rock, "My 2nd choice is jazz", "My 3rd choice...", picking the right suffix for the number based on what it is.

Array Exercise solution


```
var musicStyles = ['rock', 'jazz', 'electronica', 'funk', 'classical' ];

for (var i = 0; i < musicStyles.length; i++) {
  let choiceNum = i + 1;
  let choiceNumEnd;
  if (choiceNum == 1) {
        choiceNumEnd = 'st';
    } else if (choiceNum == 2) {
        choiceNumEnd = 'nd';
    } else if (choiceNum == 3) {
        choiceNumEnd = 'rd';
    } else {
        choiceNumEnd = 'th';
    }
    console.log('My ' + choiceNum + choiceNumEnd + ' choice is ' +
musicStyles[i]);
}
```

Objects in JavaScript

- JavaScript is an object-based language based on prototypes instead of classes.

```
// "Object" is a prototype, "myCar" is an object
var myCar = new Object();

// myCar.brand is a property
myCar.brand = "Ford";

// myCar.getBrand is a method
myCar.getBrand = function () {
  return this.brand; // "this" refers to the current object: myCar
}
```

- An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method.

JavaScript Objects (1)

- JavaScript is not a full-fledged object-oriented programming language
- It does not have classes per se, and it does not support many of the patterns you'd expect from an object-oriented language like inheritance and polymorphism
- The language does, however, support objects
- Objects in JavaScript, just as in many other programming languages, can be compared to objects in real life.

JavaScript Objects & Classes

- In a language implementing classical inheritance like Java, C++ you start by creating a class--a blueprint for your objects--and then you can create new objects from that class or you can extend the class, defining a new class that augments the original class.

- In JavaScript you first create an object (there is no concept of class), then you can augment your own object or create new objects from it. (Although it is no difficult, is seems a little foreign and hard to metabolize for someone used to the classical way).

JavaScript Objects (2)

- An object is a standalone entity, with properties and type.
- Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.
- 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

Objects and Properties (1)

- A JavaScript object has properties associated with it.
- A property of an object can be explained as a variable that is attached to the object.
- Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object.
- You access the properties of an object with a simple dot-notation:

  objectName.propertyName

- Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value.

Objects and Properties (2)

Alliance For Digital Employability

- An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string.
- Any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation.
- This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime).

```
// four variables are created and assigned in a single go,
// separated by commas
var myObj = new Object(),
    str = 'myString',
    rand = Math.random(),
    obj = new Object();

myObj.type              = 'Dot syntax';
myObj['date created']   = 'String with space';
myObj[str]              = 'String value';
myObj[rand]             = 'Random Number';
myObj[obj]              = 'Object';
myObj['']               = 'Even an empty string';

console.log(myObj);
```

Create New Objects

- JavaScript has a number of predefined objects.
- You can create your own objects.
- Create objects through:
  - Using an object initializer.
  - First create a constructor function and then instantiate an object invoking that function in conjunction with the new operator.
  - Using the objects.create method

Object Creation (1)


```
var objName = {
    name1: value1,
    name2: value2,
    // ...
    nameN: valueN
};
```

Note: Unassigned properties of an object are undefined (and not null).

Object Creation (2)


```
// first create an empty object
var objName = new Object();
// then define properties for this object
objName.name1 = value1;
objName.name2 = value2;
```

Using object initializers

- The syntax for an object using an object initializer is:

```
var obj = { property_1:   value_1,   // property_# may be an identifier...
            2:            value_2,           // or a number...
            // ...,
            'property n': value_n };       // or a string
```

- where obj is the name of the new object, each property_i is an identifier (either a name, a number, or a string literal), and each value_i is an expression whose value is assigned to the property_i.
- The obj and assignment is optional; if you do not need to refer to this object elsewhere, you do not need to assign it to a variable. (Note that you may need to wrap the object literal in parentheses if the object appears where a statement is expected, so as not to have the literal be confused with a block statement.)

Examples

- The following statement creates an object and assigns it to the variable x if and only if the expression cond is true:

```
if (cond) var x = {greeting: 'hi there'};
```

- The following example creates myHonda with three properties. Note that the engine property is also an object with its own properties.

```
var myHonda = {color: 'red', wheels: 4, engine: {cylinders: 4, size: 2.2}};
```

Class Exercise #10

Alliance For Digital Employability

- Create an object named myCar and give it properties named make, model, year, fuel (use your favorite car)

Class Exercise #10 - Answer

Alliance For Digital Employability

- Create an object named myCar and give it properties named make, model, and year (use your favorite car)

```
var myCar = new Object();
myCar.make = 'Ford';
myCar.model = 'Mustang';
myCar.year = 1969;
MyCar.fuel = “gasoline”
```

Create Object via Constructor function

- You can create an object with these two steps:
  1. Define the object type by writing a constructor function. There is a strong convention, with good reason, to use a capital initial letter.
  2. Create an instance of the object with new.
- To define an object type, create a function for the object type that specifies its name, properties, and methods.

Constructors

Alliance For Digital Employability

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:

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

For some classes, shortcut constructors are defined

```
var greeting = "Good Morning";
```

vs the formal:

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

Class Exercise #10b

Alliance For Digital Employability

- Suppose you want to create an object type for cars. You want this type of object to be called car, and you want it to have properties for make, model, and year. Use a constructor approach.

Class Exercise #10b - Answer


```
1  function Car(make, model, year) {
2    this.make = make;
3    this.model = model;
4    this.year = year;
5  }
```

Create the function

```
var mycar = new Car('Eagle', 'Talon TSi', 1993);
```

Create an object my car

```
var kenscar = new Car('Nissan', '300ZX', 1992);
var vpgscar = new Car('Mazda', 'Miata', 1990);
```

Create any number of car objects by calls to new

Browse objects - for loop <>

- When you want to browse objects you can use

  `for.. in`

```
var txt = “”;
var person = {
firstName: “Philip”, middleName: “Kindred”,
lastName: “Dick”, born: 1928, died: 1982
};

for (var x in person) {
   txt +=person[x] + “ ” ;
}
```

Alliance For Digital Employability

Properties

- 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);
```

Enumerate the properties of an object

There are three native ways to list/traverse object properties:

- for...in loops: This method traverses all enumerable properties of an object and its prototype chain
- Object.keys(o) : This method returns an array with all the own (not in the prototype chain) enumerable properties' names ("keys") of an object o.
- Object.getOwnPropertyNames(o) : This method returns an array containing all own properties' names (enumerable or not) of an object o.

Property enumeration example

```
function listAllProperties(o) {
    var objectToInspect;
    var result = [];

    for(objectToInspect = o; objectToInspect !== null;
objectToInspect = Object.getPrototypeOf(objectToInspect)) {
     result =
result.concat(Object.getOwnPropertyNames(objectToInspect));
    }

    return result;
}
```

Methods

- 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

Using the Object.create method

- Objects can also be created using the Object.create() method.
- This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function

Object.create method

Example


```
// Animal properties and method encapsulation
var Animal = {
  type: 'Invertebrates', // Default value of properties
  displayType: function() {  // Method which will display type of Animal
    console.log(this.type);
  }
};

// Create new animal type called animal1
var animal1 = Object.create(Animal);
animal1.displayType(); // Output:Invertebrates

// Create new animal type called Fishes
var fish = Object.create(Animal);
fish.type = 'Fishes';
fish.displayType(); // Output:Fishes
```

Object Exercise

Alliance For Digital Employability

Create an array of objects, where each object describes a book and has properties for the title (a string), author (a string), and alreadyRead (a boolean indicating if you read it yet).

Iterate through the array of books. For each book, log the book title and book author like so: "The Hobbit by J.R.R. Tolkien".

Use an if/else statement to change the output depending on whether you read it yet or not. If you read it, log a string like 'You already read "The Hobbit" by J.R.R. Tolkien', and if not, log a string like 'You still need to read "The Lord of the Rings" by J.R.R. Tolkien.'

Object Exercise solution

```
var books = [
  {title: 'The Man in the High Castle',
   author: 'Philip K. Dick',
   alreadyRead: false
  },
  {title: 'Serotonin',
  author: 'Michel Houellebecq',
  alreadyRead: true
  }];

for (var i = 0; i < books.length; i++) {
  var book = books[i];
  var bookInfo = book.title + '" by ' + book.author;
  if (book.alreadyRead) {
    console.log('You already read "' + bookInfo);
  } else {
    console.log('You still need to read "' + bookInfo);
  }
}
```

Objects Included in JavaScript

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

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
```

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();
// This outputs Today is Wed Jul 31 2019 11:32:40
GMT+0300 (Eastern European Summer Time)
alert ("Today is "+ d.toString());
```

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

References

- https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
- https://www.w3schools.com/js/
- https://javascript.info/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Variables

Bibliography

- Fundamentals of Web Development, 2nd Edition
  Randy Connolly and Ricardo Hoar
  ISBN 9780134481760
  © 2018 Pearson

Alliance For Digital Employability

4.4 JavaScript (I)

Q&A
Exercises and Review Questions

Sample Question #1

Alliance For Digital Employability

Which of the following is an invalid name for a variable in JavaScript?

A. myvariable

B. function

C. integer3

D. _carpool

Sample Question #1 - Answer

Alliance For Digital Employability

Which of the following is an invalid name for a variable in JavaScript?

A. myvariable

B. function

C. integer3

D. _carpool

Sample Question #2

When you declare a variable outside of any function, it is called a_____, because it is available to any other code in the current document.

A. constant

B. local variable

C. function

D. global variable

Sample Question #2 - Answer

When you declare a variable outside of any function, it is called a_____, because it is available to any other code in the current document.

A. constant

B. local variable

C. function

D. global variable

Sample Question #3

Alliance For Digital Employability

What would be the outcome of this loop be?

```
while (true) {
  console.log('Hello, world!');
}
```

A. Hello, world!
B. Null
C. Infinite loop
D. false

Sample Question #3 - Answer

What would be the outcome of this loop be?

```
while (true) {
  console.log('Hello, world!');
}
```

A. Hello, world!

B. Null

C. Infinite loop

D. false

Class Review Exercise #1

You are provided with two variables:

- A number
- A boolean

If the value of n equals 0, then assign the value true to isZero.

If the value of n does not equal 0, then assign the value false to isZero.

Value of n = 5

CR Exercise #1 - Answer

```
var n =5;
if (n===0) {
    isZero = true;
} else {
    isZero=false;
}
```

Class Review Exercise #2

- You are provided with a variable named **counter** and value 0
- Create either a for or a while loop that will run 5 times and will increment the value of counter by 2 during each step.
- The final value of counter should be 10.

CR Exercise #2 - Answer

```
var counter = 0;
do {
  counter += 2;
  console.log(i);
} while (counter<10);
```

Class Review Exercise #3

- Using a switch statement write a function that returns 1 for 0 and 1 and the factorial of any number <=100

CR Exercise #3 - Answer

```
function factorial(n) {
  switch (n) {
    case 0:
    case 1:
      return 1;
      break;
    default:
      if (n > 100) {
        return false;
      }

      return n*factorial(n-1);
      break;
  }
}
```

Exercises

Exercise 1: What will the following expressions execution (evaluation) print?

- 2 + 3
- “Hello ” + “World!”
- “Hello ” + 1
- “Hello”*5
- “Hello”/2
- 10/0

- **Exercise 2:** Write a function that takes as argument two strings, concatenates them and returns the result.

Exercises

Alliance For Digital Employability

Exercise 3: Create a car object of a Ferrari F430.

- This should contain the following properties: Brand, Model, Max Speed (330 km/h), Current Speed, Status of whether the car has started or not.
- User should be able to start, stop and set its speed status.
- The car should also provide a way of exposing all the data described above, after each state change.

Exercise 4: Create a function that takes as argument the three constants (a, b, c) of a polynomial and computes a second degree polynomial discriminant.

The type is (b*b) – (4*a*c).

Alliance For Digital Employability

Thank you

Alliance For Digital Employability

Coding Bootcamp

Web Design and Development Fundamentals (Front End)
Lesson 3
(Common)

Alliance For Digital Employability

JavaScript Part II

4.6.1 - 4.6.3

Contents

Learning Objectives

- Understand the JavaScript functions and the different ways that they can be defined and called
- Understand the object prototypes
- Familiarize with the exception handling
- Understand the Document Object Model and the selection process
- Familiarize with the JavaScript events
- Understand the JSON and XML model
- Learn how to validate the form data

Syllabus

4.6 JavaScript/ jQuery	4.6.1	Code in JavaScript, variables, functions
	4.6.2	Code using advance java script: if conditions, loops
	4.6.3	**Improve usability of forms, validate data from the user**

Introduction to JS Functions

- Functions are one of the fundamental building blocks in JavaScript.
- A function is a JavaScript procedure—a set of statements that performs a task or calculates a value.
- To use a function, you must define it somewhere in the scope from which you wish to call it.

Functions (1)

- A JavaScript function is a block of code designed to perform a particular task
  - This block can be reused within the code, and, if provided different arguments, can produce different results
  - Executed when "something“ invokes it (calls it – via an event call, direct call from code, or automatically)
  - A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ()

Functions (2)

- Function names can contain letters, digits, underscores, and dollar signs - same rules as variables
  - In fact, functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations
- The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: {}
- Inside a function:
  - When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement

Functions (3)

- Functions are the building block for modular code in JavaScript

```
function subtotal(price,quantity) {
   return price * quantity;
}
```

- The above is formally called a function declaration, called or invoked by using the () operator

```
var result = subtotal(10,2);
```

Defining Functions (1)

- A function definition (also called a function declaration, or function statement) consists of the `function` keyword, followed by:
  - The name of the function.
  - A list of parameters to the function, enclosed in parentheses and separated by commas.
  - The JavaScript statements that define the function, enclosed in curly brackets, { }.
- In addition to defining functions as described below, you can also use the **Function constructor** to create functions from a string at runtime, much like eval().

Defining Functions – Example 1


```
function square(number) {
  return number * number;
}
```

- Code above defines a simple function named **square**
- The function square takes **one parameter**, called **number**
- The function consists of one statement that says to return the parameter of the function (that is, number) multiplied by itself
- The return statement specifies the value returned by the function
- Primitive parameters (such as a number) are passed to functions **by value**; the value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.

Defining Functions – Example 2

Alliance For Digital Employability

- If you pass an object (i.e. a non-primitive value, such as **Array** or a user-defined object) as a parameter and the function changes the object's properties, that change is visible outside the function, as shown in the example below:

```
function myFunc(theObject) {
  theObject.make = 'Toyota';
}

var mycar = {make: 'Honda', model: 'Accord', year: 1998};
var x, y;

x = mycar.make; // x gets the value "Honda"

myFunc(mycar);
y = mycar.make; // y gets the value "Toyota"
                // (the make property was changed by the function)
```

Function Expressions (1)

- While the function declaration above is syntactically a statement, functions can also be created by a function expression.
- Such a function can be anonymous; it does not have to have a name.
- For example, the function square could have been defined as:

```
var square = function(number) { return number * number; };
var x = square(4); // x gets the value 16
```

Function Expressions (2)

```
// defines a function using a function expression
var sub = function subtotal(price, quantity) {
        return price * quantity;
};
// invokes the function
var result = sub(10,2);
```

- It is conventional to leave out the function name in function expressions

Anonymous Function Expression

```
// defines a function using an anonymous function expression
var calculateSubtotal = function (price,quantity) {
      return price * quantity;
};
// invokes the function
var result = calculateSubtotal(10,2);
```

Passing Functions

- Function expressions are convenient when passing a function as an argument to another function
- The following example shows a map function that should receive a function as first argument and an array as second argument

```
function map(f, a) {
  var result = [], // Create a new Array
      i;
  for (i = 0; i != a.length; i++)
    result[i] = f(a[i]);
  return result;
}
```

Nested Functions

- You can nest a function within a function.
- The nested (inner) function is private to its containing (outer) function. It also forms a closure. A closure is an expression (typically a function) that can have free variables together with an environment that binds those variables (that "closes" the expression).
- Since a nested function is a closure, this means that a nested function can "inherit" the arguments and variables of its containing function. In other words, the inner function contains the scope of the outer function.
- To summarize:
  - The inner function can be accessed only from statements in the outer function.
  - The inner function forms a closure: the inner function can use the arguments and variables of the outer function, while the outer function cannot use the arguments and variables of the inner function.
  - Closure is useful to create private variables or functions.

Closure example


```
function OuterFunction() {

    var outerVariable = 100;

    function InnerFunction() {
        alert(outerVariable);
    }

    return InnerFunction;
}
var innerFunc = OuterFunction();

innerFunc(); // 100
```

Nested Functions – Example 1


```
function calculateTotal(price,quantity) {
    var subtotal = price * quantity;
    return subtotal + calculateTax(subtotal);
    // this function is nested
    function calculateTax(subtotal) {
        var taxRate = 0.05;
        var tax = subtotal * taxRate;
        return tax;
    }
}
```

Nested Functions – Example 2


```
function addSquares(a, b) {
  function square(x) {
    return x * x;
  }
  return square(a) + square(b);
}
```

Class Exercise #1

Alliance For Digital Employability

Consider the following code:

```
function map(f, a) {
  var result = []; // Create a new Array
  var i; // Declare variable
  for (i = 0; i != a.length; i++)
    result[i] = f(a[i]);
      return result;
}
var f = function(x) {
   return x * x * x;
}
var numbers = [0,1, 2, 5,10];
var cube = map(f,numbers);
console.log(cube);
```

- What is happening here?
- What is the outcome of this code?

Class Exercise #1 - Answer


```
function map(f, a) {
  var result = []; // Create a new Array
  var i; // Declare variable
  for (i = 0; i != a.length; i++)
    result[i] = f(a[i]);
      return result;
}
var f = function(x) {
   return x * x * x;
}
var numbers = [0,1, 2, 5,10];
var cube = map(f,numbers);
console.log(cube);
```

- This function receives function defined by function expression and executes it for every element of array received as second argument
- Function returns: [0, 1, 8, 125, 1000].

Class Exercise #2

Alliance For Digital Employability

Consider the following code:

```
function addSquares(a, b) {
  function square(x) {
    return x * x;
  }
  return square(a) + square(b);
}
a = addSquares(2, 3);
b = addSquares(3, 4);
c = addSquares(4, 5);
```

- What will the following lines return?

a = addSquares(2, 3);

b = addSquares(3, 4);

c = addSquares(4, 5);

Class Exercise #2 - Answer


```
function addSquares(a, b) {
  function square(x) {
    return x * x;
  }
  return square(a) + square(b);
}
a = addSquares(2, 3); // returns 13
b = addSquares(3, 4); // returns 25
c = addSquares(4, 5); // returns 41
```

Defining Functions (2)

- In JavaScript, a function can be defined based on a condition.
- For example, the following function definition defines **myFunc** only if **num equals 0**:

```
var myFunc;
if (num === 0) {
  myFunc = function(theObject) {
    theObject.make = 'Toyota';
  }
}
```

Calling Functions

- Defining a function does not execute it
- Defining the function simply names the function and specifies what to do when the function is called
- Calling the function actually performs the specified actions with the indicated parameters
- Functions must be in scope when they are called

Example, if you define the function square, you could call it as follows:

square(5);

This statement calls the function with an argument of 5. The function executes its statements and returns the value 25.

Calling Functions (2)

- A function can call itself.
- For example, here is a function that computes factorials **recursively**:

```
function factorial(n) {
  if ((n === 0) || (n === 1))
    return 1;
  else
    return (n * factorial(n - 1));
}
```

- You could then compute the factorials of one through five as follows:

```
var a, b, c, d, e;
a = factorial(1); // a gets the value 1
b = factorial(2); // b gets the value 2
c = factorial(3); // c gets the value 6
d = factorial(4); // d gets the value 24
e = factorial(5); // e gets the value 120
```

Calling Functions (3)

- There are other ways to call functions
  - There are often cases where a function needs to be called dynamically
  - Call a function where the number of arguments needs to vary
  - The context of the function call needs to be set to a specific object determined at runtime
- It turns out that functions are, themselves, objects, and these objects in turn have methods (see the Function object).
- One of these, the apply() method, can be used to achieve this goal.

Callback Functions (1)

- A callback is a function that is to be executed after another function (normally asynchronous) has finished executing (hence the name ‘call back’).
- In JavaScript, functions are objects, so they can take functions as arguments, and can be returned by other functions. Functions that do this are called higher-order functions. Any function that is passed as an argument and subsequently called by the function that receives it, is called a callback function.
- Callbacks are helpful for one very important reason: JavaScript is an event driven language. This means that instead of waiting for a response before moving on, JavaScript will keep executing while listening for other events.

Callback Functions example 1

```
function codeHomework(subject, callback) {
    alert(`Starting my ${subject} coding homework.`);
    callback();
}

function alertFinished(){
  alert('Finished my coding homework');
}

codeHomework('Javascript', alertFinished);
```

Callback Functions example 2

```
var calculateTotal = function (price, quantity, tax) {
    var subtotal = price * quantity;
    return subtotal + tax(subtotal);
};

var calcTax = function (subtotal) {
    var taxRate = 0.05;
    var tax subtotal * taxRate;
    return tax;
};

var temp = calculateTotal(50,2,calcTax);
```

The local parameter variable tax is a reference to the calcTax() function

Passing the calcTax() function object as a parameter

We can say that calcTax variable here is a callback function

Alliance For Digital Employability

Callback Functions example 3

- In the previous example we can write the callback function inline

```
var temp = calculateTotal( 50, 2,
        function (subtotal) {
            var taxRate = 0.05;
            var tax = subtotal * taxRate;
            return tax;
        }
    );
```

Passing an anonymous function as a callback function parameter

Methods

- A **method** is a function that is a property of an object
- A **method** is a **function** associated with an object, or, simply put, a method is a property of an object that is a function
- Methods are defined the way normal functions are defined, except that they have to be assigned as the property of an object

```
objectName.methodname = function_name;

var myObj = {
  myMethod: function(params) {
    // ...do something
  }

  // OR THIS WORKS TOO

  myOtherMethod(params) {
    // ...do something else
  }
};
```

Using **this** for Object References

- JavaScript has a special keyword, **this**, that you can use within a method to refer to the current object
- **this** refers to the calling object in a method
- When combined with the **form** property, **this** can refer to the current object's parent form.

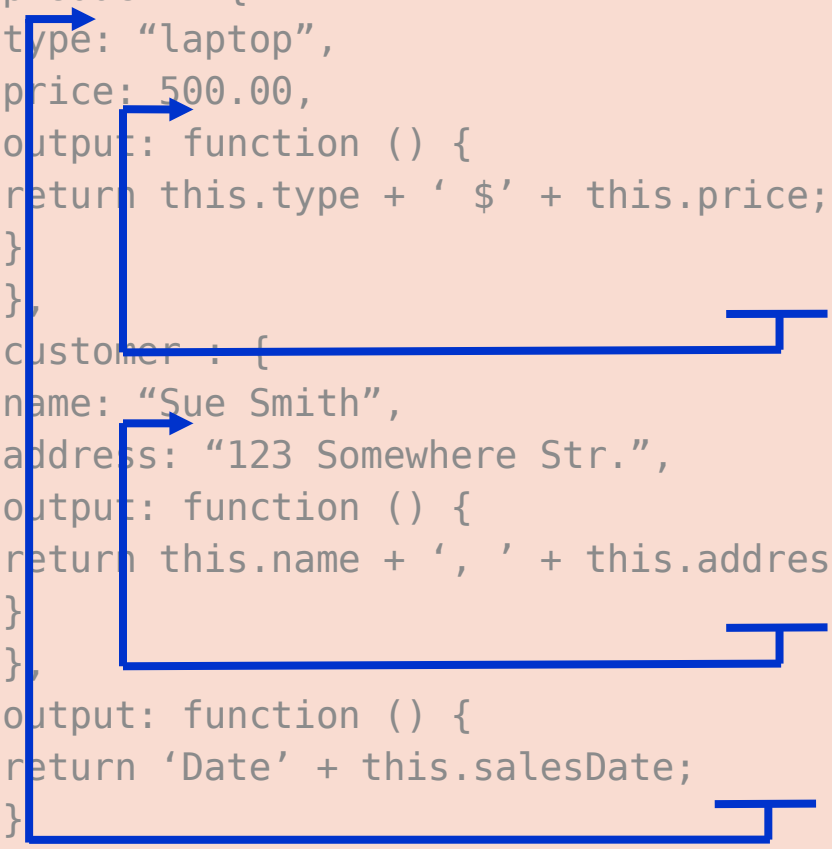
this - Example

- In the following example, the form myForm contains a Text object and a button.
- When the user clicks the button, the value of the Text object is set to the form's name.
- The button's onclick event handler uses this.form to refer to the parent form, myForm.

```
<form name = "myForm">
<p><label>Form name: <input type = "text" name = "text1"
value = "beluga"></label></p>
<p><input name="button1" type = "button" value = "Show Form Name"
    onclick="this.form.text1.value = this.form.name">
</p>
</form>
```

Objects and Functions Together

```
var order = {
salesDate : “May, 5, 2017”,
product : {
type: “laptop”,
price: 500.00,
output: function () {
return this.type + ‘ $’ + this.price;
}
},
customer : {
name: “Sue Smith”,
address: “123 Somewhere Str.”,
output: function () {
return this.name + ‘, ’ + this.address;
}
},
output: function () {
return ‘Date’ + this.salesDate;
}
};
```



Function Scope

- Variables defined inside a function cannot be accessed from anywhere outside the function, because the variable is defined only in the scope of the function
- However, a function can access all variables and functions defined inside the scope in which it is defined
- A function defined in the global scope can access all variables defined in the global scope
- A function defined inside another function can also access all variables defined in its parent function and any other variable to which the parent function has access

Scope (1)

- Scope determines the accessibility (visibility) of variables
- In JavaScript there are two types of scope:
  - Local scope
  - Global scope
- JavaScript has function scope
  - Each function creates a new scope
- Variables defined inside a function are not accessible (visible) from outside the function
- Since local variables are only recognized inside their functions, variables with the same name can be used in different functions
  - Local variables are created when a function starts, and deleted when the function is completed

Scope (2)

- A variable declared outside a function, becomes GLOBAL
  - All scripts and functions on a web page can access it
  - In HTML, the global scope is the window object
  - If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable
- Do NOT create global variables unless you intend to
  - Your global variables (or functions) can overwrite window variables (or functions)
  - Any function, including the window object, can overwrite your global variables and functions

Scope (3) – Example


```
// Function declaration
function myFunction() {
carName = “Volvo”;
var scopedCarName = “Saab”;

// Code here can use globalCarName and carName (as they are global)
// but also scopedCarName as local variable to myFunction
}

var globalCarName = “BMW”;

// Code her can use globalCarName as a global variable
// Code here can use carName as a global variable
// Code here can not use scopedCarName

// Function call
myFunction();
```

Class-based vs. prototype-based languages

- Class-based object-oriented languages, such as Java and C++, are founded on the concept of two distinct entities: **classes** and **instances**.

  A class defines all of the properties (considering methods and fields in Java, or members in C++, to be properties) that characterize a certain set of objects. A class is an abstract thing, rather than any particular member of the set of objects it describes. For example, the Employee class could represent the set of all employees.

  An instance, on the other hand, is the instantiation of a class; that is, one of its members. For example, Victoria could be an instance of the Employee class, representing a particular individual as an employee. An instance has exactly the same properties of its parent class (no more, no less).

- A **prototype-based language**, such as **JavaScript**, does not make this distinction: it simply has **objects**.
- A prototype-based language has the notion of a prototypical object, an object used as a template from which to get the initial properties for a new object.
- Any object can specify its own properties, either when you create it or at run time. In addition, any object can be associated as the prototype for another object, allowing the second object to share the first object's properties.

Comparison of class-based (Java) and prototype-based (JavaScript) object systems

Class-based (Java)	Prototype-based (JavaScript)
Class and instance are distinct entities.	All objects can inherit from another object.
Define a class with a class definition; instantiate a class with constructor methods.	Define and create a set of objects with constructor functions.
Create a single object with the new operator.	Same.
Construct an object hierarchy by using class definitions to define subclasses of existing classes.	Construct an object hierarchy by assigning an object as the prototype associated with a constructor function.
Inherit properties by following the class chain.	Inherit properties by following the prototype chain.
Class definition specifies all properties of all instances of a class. Cannot add properties dynamically at run time.	Constructor function or prototype specifies an initial set of properties. Can add or remove properties dynamically to individual objects or to the entire set of objects.

Object Prototypes

- A prototype is an object from which other objects inherit properties
- While the constructor function is simple to use, it can be an inefficient approach for objects that contain methods
- Prototypes are an essential syntax mechanism in JavaScript, and are used to make JavaScript behave more like an object-oriented language and add functions to objects

Using Prototypes to Extend Other Objects

```
String.prototype.countChars = function (c) {
        var count=0;
        for (var i=0;i<this.length;i++) {
                if (this.charAt(i) == c)
                        count++;
                }
        return count;
}
var msg = "Hello World";
console.log(msg + "has" + msg.countChars("l") + " letter l's");
```

Error Handling Tips

- Ensure your code is working generally
- Common JavaScript problems that you will want to be mindful of, such as:
  - Basic syntax and logic problems
  - Making sure variables, etc. are defined in the correct scope, and you are not running into conflicts between items declared in different places
  - Confusion about this, in terms of what scope it applies to, and therefore if its value is what you intended
  - Incorrectly using functions inside loops
  - Making sure asynchronous operations have returned before trying to use the values they return

Alert

- The alert() function makes the browser show a pop-up to the user, with whatever is passed being the message displayed
- The following JavaScript code displays a simple hello world message in a pop-up:

```
alert ( "Good Morning" );
```

- Using alerts can get tedious fast
  - When using debugger tools in your browser you can write output to a log with:
  - console.log("Put Messages Here");
  - And then use the debugger to access those logs

Tools

- You can ensure better quality, less error-prone JavaScript code using a **linter**, which points out errors and can also flag up warnings about bad practices, etc., and be customized to be stricter or more relaxed in their error/warning reporting.
- The JavaScript/ECMAScript linters recommended are:
  - JSHint
  - ESLint

  Both have VS extensions
- The Atom code editor has a JSHint plugin
  - Go to Atom's Preferences... dialog (e.g. by Choosing Atom > Preferences... on Mac, or File > Preferences... on Windows/Linux) and choose the Install option in the left-hand menu.
  - In the Search packages text field, type "jslint" and press Enter/Return to search for linting-related packages.
  - You should see a package called lint at the top of the list. Install this first (using the Install button), as other linters rely on it to work. After that, install the linter-jshint plugin.

Error Handling Statements

- You can throw exceptions using the throw statement and handle them using the try...catch statements.
  - throw statement
  - try...catch statement
- Exception Types: Just about any object can be thrown in JavaScript. Nevertheless, not all thrown objects are created equal. While it is fairly common to throw numbers or strings as errors it is frequently more effective to use one of the exception types specifically created for this purpose:
  - ECMAScript exceptions
  - DOMException and DOMError

try...catch statement

- The try...catch statement marks a block of statements to try, and specifies one or more responses should an exception be thrown.
- If an exception is thrown, the try...catch statement catches it.
- The try...catch statement consists of a try block, which contains one or more statements, and a catch block, containing statements that specify what to do if an exception is thrown in the try block. That is, you want the try block to succeed, and if it does not succeed, you want control to pass to the catch block.
- If any statement within the try block (or in a function called from within the try block) throws an exception, control immediately shifts to the catch block.
- If no exception is thrown in the try block, the catch block is skipped.
- The finally block executes after the try and catch blocks execute but before the statements following the try...catch statement.

Errors using try and catch

- When the browser's JavaScript engine encounters an error, it will throw an exception
- These exceptions interrupt the regular, sequential execution of the program and can stop the JavaScript engine altogether
- However, you can optionally catch these errors preventing disruption of the program using the try–catch block

```
try {
nonexistantfunction("hello");
}
catch(err) {
alert("An exception was caught: " + err);
}
```

try…catch example


```
function getMonthName(mo) {
  mo = mo - 1; // Adjust month number for array index (1 = Jan, 12 = Dec)
  var months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
                'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
  if (months[mo]) {
    return months[mo];
  } else {
    throw 'InvalidMonthNo'; //throw keyword is used here
  }
}

try { // statements to try
  monthName = getMonthName(myMonth); // function could throw exception
}
catch (e) {
  monthName = 'unknown';
  logMyErrors(e); // pass exception object to error handler -> your own function
}
```

Class Discussion

- What does this function do?

```
function getMonthName(mo) {
  mo = mo - 1; // Adjust month number for array index (1 = Jan, 12 = Dec)
  var months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
                'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
  if (months[mo]) {
    return months[mo];
  } else {
    throw 'InvalidMonthNo'; //throw keyword is used here
  }
}

try { // statements to try
  monthName = getMonthName(myMonth); // function could throw exception
}
catch (e) {
  monthName = 'unknown';
  logMyErrors(e); // pass exception object to error handler -> your own function
}
```

Class Discussion

Alliance For Digital Employability

- calls a function that retrieves a month name from an array based on the value passed to the function
- If the value does not correspond to a month number (1-12), an exception is thrown with the value "InvalidMonthNo"
- The statements in the catch block set the monthName variable to unknown.

```
function getMonthName(mo) {
  mo = mo - 1; // Adjust month number for array index (1 = Jan, 12 = Dec)
  var months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
                'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
  if (months[mo]) {
    return months[mo];
  } else {
    throw 'InvalidMonthNo'; //throw keyword is used here
  }
}

try { // statements to try
  monthName = getMonthName(myMonth); // function could throw exception
}
catch (e) {
  monthName = 'unknown';
  logMyErrors(e); // pass exception object to error handler -> your own function
}
```

The **catch block**

- You can use a catch block to handle all exceptions that may be generated in the try block.

```
catch (catchID) {
 statements
}
```

- The catch block specifies an identifier (catchID in the preceding syntax) that holds the value specified by the throw statement; you can use this identifier to get information about the exception that was thrown.
- JavaScript creates this identifier when the catch block is entered; the identifier lasts only for the duration of the catch block; after the catch block finishes executing, the identifier is no longer available.

catch block example

- For example, the following code throws an exception. When the exception occurs, control transfers to the catch block.

```
try {
  throw 'myException'; // generates an exception
}
catch (e) {
  // statements to handle any exceptions
  logMyErrors(e); // pass exception object to error handler
}
```

The **finally block**

- The finally block contains statements to execute after the try and catch blocks execute but before the statements following the try...catch statement.
- The finally block executes whether or not an exception is thrown. If an exception is thrown, the statements in the finally block execute even if no catch block handles the exception.
- You can use the finally block to make your script fail gracefully when an exception occurs; for example, you may need to release a resource that your script has tied up.

finally block example (1)

- The following example opens a file and then executes statements that use the file (server-side JavaScript allows you to access files).
- If an exception is thrown while the file is open, the finally block closes the file before the script fails.

```
openMyFile();
try {
  writeMyFile(theData); //This may throw a error
} catch(e) {
  handleError(e); // If we got a error we handle it
} finally {
  closeMyFile(); // always close the resource
}
```

finally block example (2)

- If the finally block returns a value, this value becomes the return value of the entire try-catch-finally production, regardless of any return statements in the try and catch blocks:

```
function f() {
  try {
    console.log(0);
    throw 'bogus';
  } catch(e) {
    console.log(1);
    return true; // this return statement is suspended
                 // until finally block has completed
    console.log(2); // not reachable
  } finally {
    console.log(3);
    return false; // overwrites the previous "return"
    console.log(4); // not reachable
  }
  // "return false" is executed now
  console.log(5); // not reachable
}
f(); // console 0, 1, 3; returns false
```

finally block example (3)

- Overwriting of return values by the finally block also applies to exceptions thrown or re-thrown inside of the catch block:

```
function f() {
  try {
    throw 'bogus';
  } catch(e) {
    console.log('caught inner "bogus"');
    throw e; // this throw statement is suspended until
             // finally block has completed
  } finally {
    return false; // overwrites the previous "throw"
  }
  // "return false" is executed now
}

try {
  f();
} catch(e) {
  // this is never reached because the throw inside
  // the catch is overwritten
  // by the return in finally
  console.log('caught outer "bogus"');
}

// OUTPUT
// caught inner "bogus"
```

Nesting try…catch statements

- You can nest one or more try...catch statements.
- If an inner try...catch statement does not have a catch block, it needs to have a finally block and the enclosing try...catch statement's catch block is checked for a match.

Throw your own

- Although try-catch can be used exclusively to catch built-in JavaScript errors, it can also be used by your programs, to throw your own messages
- The throw keyword stops normal sequential execution, just like the built-in exceptions

```
try {
    var x = -1;
    if (x<0)
       throw “smallerthanoError”;
}
catch(err) {
    alert(err + “was thrown”);
}
```

Utilizing Error objects

- Depending on the type of error, you may be able to use the 'name' and 'message' properties to get a more refined message. 'name' provides the general class of Error (e.g., 'DOMException' or 'Error'), while 'message' generally provides a more succinct message than one would get by converting the error object to a string.
- If you are throwing your own exceptions, in order to take advantage of these properties (such as if your catch block doesn't discriminate between your own exceptions and system ones), you can use the Error constructor.

Example


```
function doSomethingErrorProne() {
  if (ourCodeMakesAMistake()) {
    throw (new Error('The message'));
  } else {
    doSomethingToGetAJavascriptError();
  }
}
....
try {
  doSomethingErrorProne();
} catch (e) {
  console.log(e.name); // logs 'Error'
  console.log(e.message); // logs 'The message' or a JavaScript error message)
}
```

Alliance For Digital Employability

The Document Object Model (DOM)

Introduction

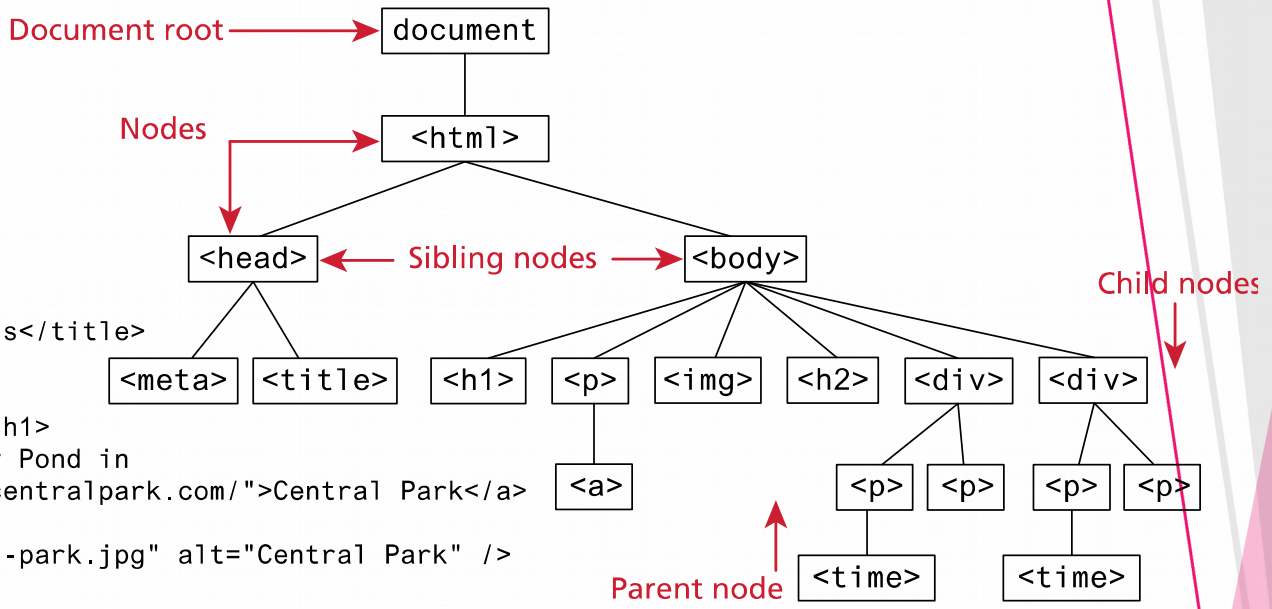
- The Document Object Model (DOM) is a programming interface for HTML.
- It provides a structured representation of the document (e.g. a web page) as a tree.
- It defines methods that allow access to the tree, so that they can change the document structure, style and content.

Document Object Model

- JavaScript is almost always used to interact with the HTML document in which it is contained
- This is accomplished through a programming interface (API) called the Document Object Model
- According to the W3C, the **DOM** is a:

  Platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.

DOM Overview




```
<html>
<head>
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>By Ricardo on <time>2016-05-23</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```

DOM example

https://codesandbox.io/s/dom-structure-ib2ud

DOM Nodes

- In the DOM, each element within the HTML document is called a node. If the DOM is a tree, then each node is an individual branch
- There are:
  - element nodes,
  - text nodes, and
  - attribute nodes
- All nodes in the DOM share a common set of properties and methods

Element Node Object

- Element Node object represents an HTML element in the hierarchy, contained between the opening <> and closing </> tags for this element
- Every node has
  - classList
  - className
  - Id
  - innerHTML
  - Style
  - tagName

Essential Node Object properties

Property	Description
attributes	Collection of node attributes
childNodes	A NodeList of child nodes for this node
firstChild	First child node of this node
lastChild	Last child of this node
nextSibling	Next sibling node for this node
nodeName	Name of the node
nodeType	Type of the node
nodeValue	Value of the node
parentNode	Parent node for this node
previousSibling	Previous sibling node for this node

Other Common Attributes

- Other common attributes include:
  - href
  - name
  - src
  - value
- However these do not exist in all objects

Document Object

- The DOM document object is the root JavaScript object representing the entire HTML document

```
// retrieve the URL of the current page
var a = document.URL;
// retrieve the page encoding, for example ISO-8859-1
var b = document.inputEncoding;
```

Selection Methods (1)

- Classic
  - getElementById()
  - getElementsByTagName()
  - getElementsByClassName()
- Newer
  - querySelector() and
  - querySelectorAll()

Selection Methods example

dom-basics.html

Accessing the DOM

Alliance For Digital Employability

- The browser provides us with access to DOM objects

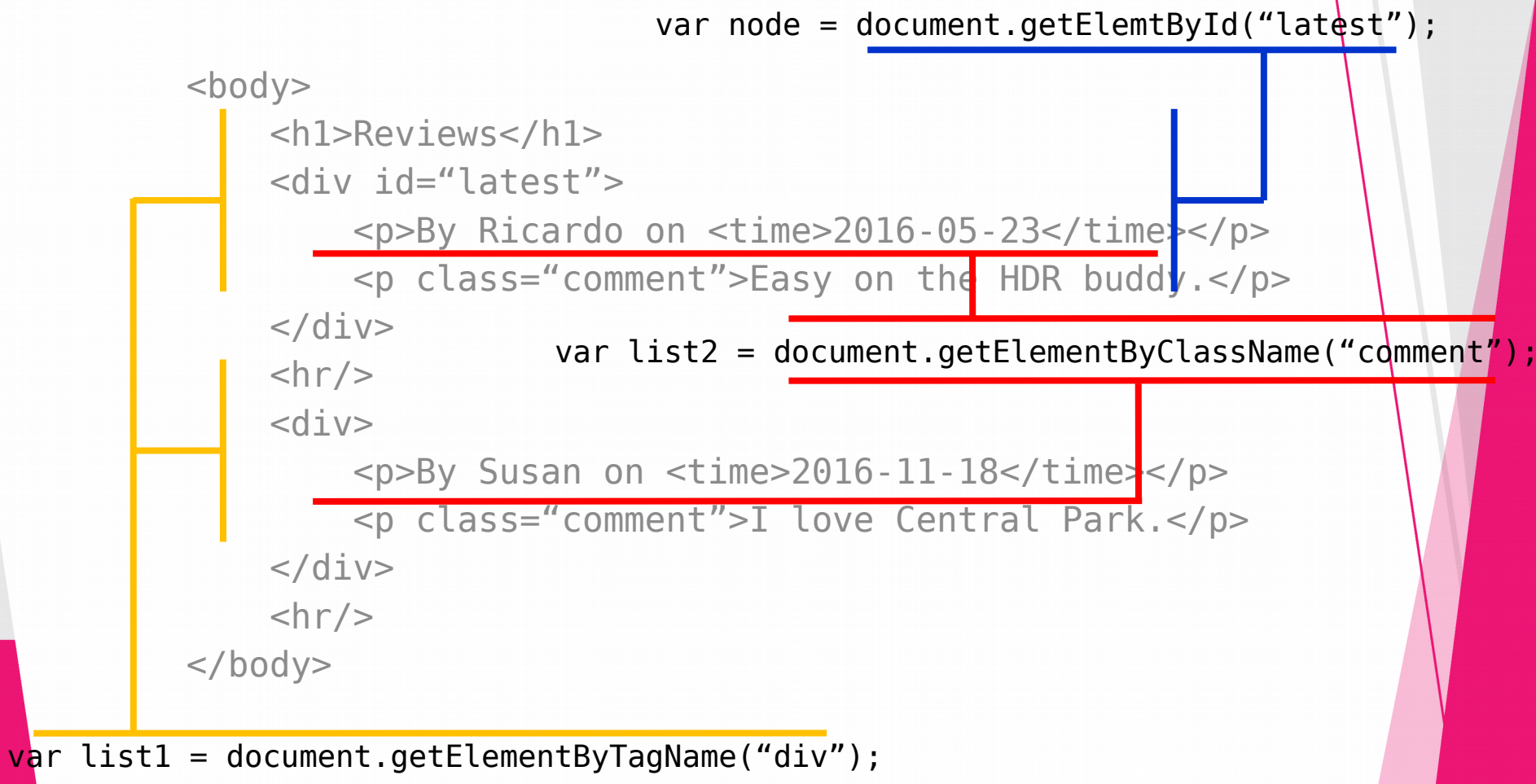
```
// Update the page's title
document.title = 'Hello!';

// Refresh the current tab
window.reload();

// Get the element with id="button"
var button = document.getElementById('button');

// Update the button's text
button.textContent = 'Push the button!'
```

Accessing HTML Elements

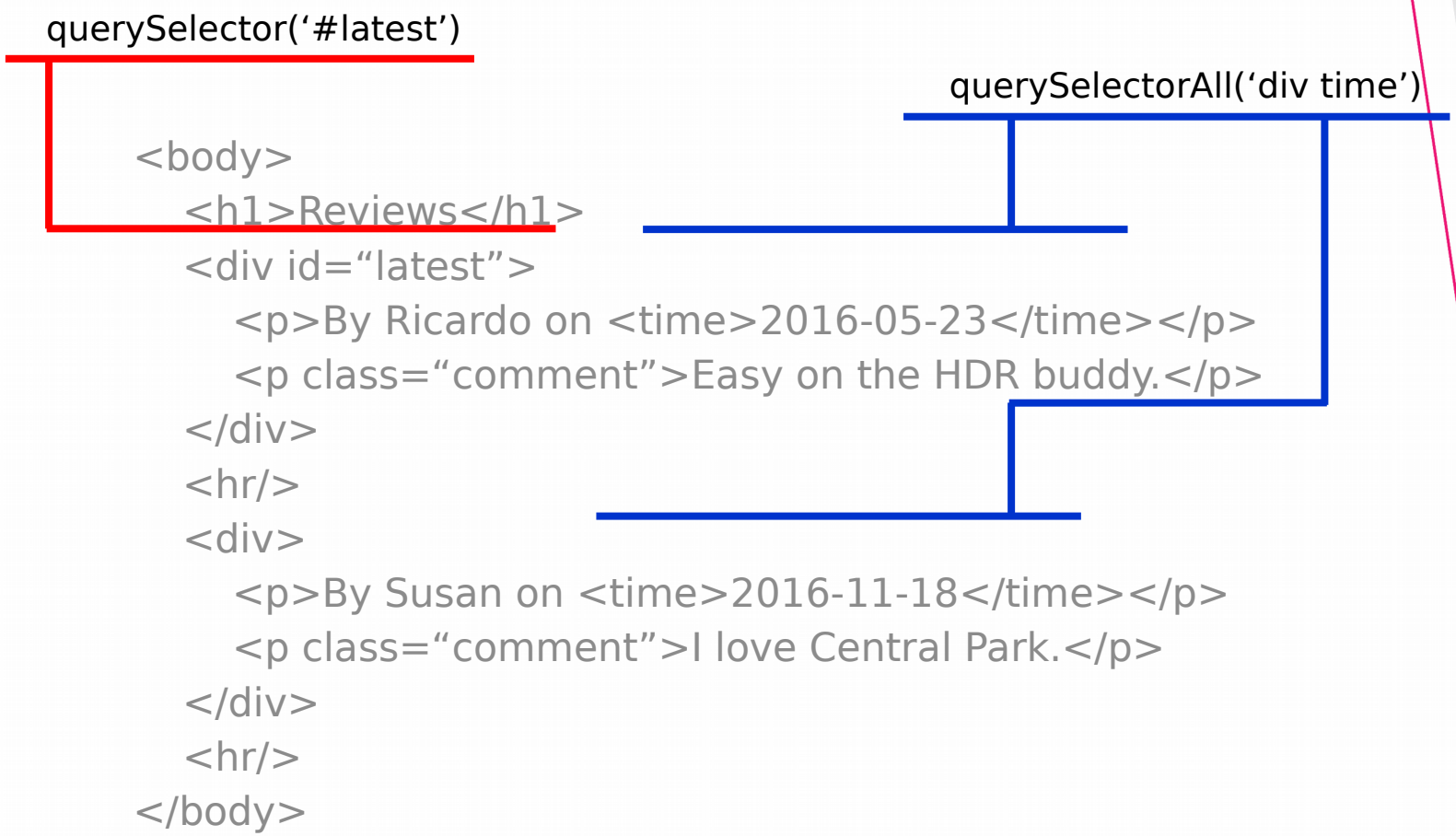
- The browser provides us with access to HTML elements

```
// Accessing an HTML element by ID
var d = document.getElementById('something');

// Accessing HTML elements by class name
var prettyElements = document.getElementsByClassName('pretty');

// Accessing HTML elements by CSS selectors
var prettyDivs = document.querySelectorAll('div.pretty');
```

Selection Methods (2)



Query Selectors



Creating HTML Elements

- In an HTML document, the **Document.createElement() method** creates the HTML element specified by tagName, or an HTMLUnknownElement if tagName isn't recognized.
- In a XUL document, it creates the specified XUL element.
- In other documents, it creates an element with a null namespace URI.
- To explicitly specify the namespace URI for the element, use document.createElementNS().
- Syntax:
  var element = document.createElement(tagName[, options]);

Example


```
// Create a div element
var d = document.createElement('div');

// Create an image element
var img = document.createElement('img');

// Add the above elements into the web page
document.body.appendChild(d);
d.appendChild(img);
```

Class Exercise #3

Alliance For Digital Employability

- Create a <**img></img> element** and append it to the element with **id parent**

Modifying HTML Elements

- Example:

```
// Modify the text of an HTML element
d.textConent = 'This is a div!';

// Modify an attribute of an HTML element
img.src = 'https://github.com/parisk.png';

// Modify the HTML contents of an element
d.innerHTML = 'This is some <strong>bold test</strong>.';
```

JavaScript Events

JavaScript Events (1)

- A JavaScript **event** is an action that can be detected by JavaScript
  - many of them are initiated by user actions
  - some are generated by the browser itself

- We say that an event is triggered and then it is handled by JavaScript functions

JavaScript Events (2)

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.
- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.
- Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

JavaScript Events (3)

- Event handlers can be used to handle, and verify, user input, user actions, and browser actions:
  - Things that should be done every time a page loads
  - Things that should be done when the page is closed
  - Action that should be performed when a user clicks a button
  - Content that should be verified when a user inputs data
- Many different methods can be used to let JavaScript work with events:
  - HTML event attributes can execute JavaScript code directly
  - HTML event attributes can call JavaScript functions
  - You can assign your own event handler functions to HTML elements
  - You can prevent events from being sent or being handled

Event-Handling Approaches (1)

- Inline Hook

```
…
<script type="text/javascript" src="inline.js"></script>
…
<form name='mainform' onsubmit="validate(this);">
<input name="name" type="text"
    onchange="check(this);"
    onfocus="highlight(this, true);"
    onblur="highlight(this, false);">
…
```

inline.js

```
function validate(node) {
    …
}
function check(node) {
    …
}
function highlight(node, state) {
    …
}
```

Event-Handling Approaches (2)

- Event Property Approach

```
var myButton = document.getElementById('example');
myButton.onclick = alert('some message');
```

Event-Handling Approaches (3)

- Event Listener Approach

```
var myButton = document.getElementById('example');
myButton.addEventListener('click', alert('some message'));
myButton.addEventListener('mouseout', funcName);
```

Event-Handling Approaches (4)

- Event Listener Approach (anonymous function)

```
myButton.addEventListener('click', function() {
        var d = new Date();
        alert("You clicked this on "+ d.toString());
});
```

Event Object (1)

- When an event is triggered, the browser will construct an event object that contains information about the event.

```
div.addEventListener('click', function(e) {
    // find out where the user clicked
    var x = e.clientX;
    ...
```

Event Object (2)

- bubbles
  - Indicates whether the event bubbles up through the DOM
- cancelable
  - Indicates whether the event can be cancelled
- target
  - The object that generated (or dispatched) the event
- type
  - The type of the event

Event Types

- Mouse Events
- Click Events
- Touch Events
- Form Events
- Frame Events

Mouse Event Types

- click
  - The mouse was clicked on an element
- dblclick
  - The mouse was double clicked on an element
- mousedown
  - The mouse was pressed down over an element
- mouseup
  - The mouse was released over an element
- mouseover
  - The mouse was moved (not clicked) over an element
- mouseout
  - The mouse was moved off of an element
- mousemove
  - The mouse was moved while over an element

Click Event Types

- keydown
  - The user is pressing a key (this happens first)
- keypress
  - The user presses a key (this happens after keydown)
- keyup
  - The user releases a key that was down (this happens last)

Touch Event Types

- Touch events are a new category of events that can be triggered by devices with touch screens
- There is limited Browser support (in 2017)
- The different events (e.g., touchstart, touchmove, and touchend) are analogous to some of the mouse events (mousedown, mousemove, and mouseup)

Form Event Types

- blur
  - When an element has lost focus
- change
  - When a value in an element has changed by the user
- focus
  - When an element has received focus
- reset
  - When a form is reset
- select
  - When some text is being selected
- submit
  - When the form is submitted

Frame Event Types

- abort
  - An object was stopped from loading
- error
  - An object or image did not properly load
- load
  - When a document or object has been loaded
- resize
  - The document view was resized
- scroll
  - The document view was scrolled
- unload
  - The document has unloaded

What is JSON?

- JSON: **J**ava**S**cript **O**bject **N**otation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.

```
{
    "name":"John",
    "age":30,
    "cars": {
        "car1":"Ford",
        "car2":"BMW",
        "car3":"Fiat"
    }
}
```

JSON (1)

- JSON is text, written with JavaScript object notation
  - When exchanging data between a browser and a server, the data can only be text
  - We can convert any JavaScript object into JSON, and send JSON to the server, also convert any JSON received from the server into JavaScript objects
  - This way we can work with the data as JavaScript objects, with no complicated parsing and translations
- JSON syntax is a subset of the JavaScript syntax

JSON (2)

- JSON data is written as name/value pairs
  - Keys (names) must be strings, written with double quotes
  - Values must be one of the following data types: string, number, object (JSON object), array, boolean, null
  - In JavaScript, keys can be strings, numbers, or identifier names
  - In JavaScript values can be all of the above, plus any other valid JavaScript expression, including function, date and undefined
- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json“
- Both JSON and XML can be used to receive data from a web server

JSON Sending Data

- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

```
var myObj = { "name":"John", "age":31, "city":"New York" };
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

JSON Receiving Data

- If you receive data in JSON format, you can convert it into a JavaScript object:

```
var myJSON = '{ "name":"John", "age":31, "city":"New York" }';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

JSON Syntax Rules

- The JSON syntax is a subset of the JavaScript syntax
- JSON syntax is derived from JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays
- In JSON, values must be one of the following data types:
  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - null

XML

- XML stands for eXtensible Markup Language.
  - Designed to store and transport data.
  - Designed to be both human- and machine-readable.
  - Does not carry any information about how to be displayed.
  - Used in many aspects of web development.
  - Used to separate data from presentation.
- XML documents form a tree structure that starts at "the root" and branches to "the leaves".

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Syntax Rules

- XML can contain the **prolog** line, that often defines XML document version and encoding.

```
<?xml version="1.0" encoding="UTF-8"?>
```

- XML documents must contain one **root** element that is the **parent** of all other elements.
  - All XML Elements Must Have a Closing Tag
  - XML Elements Must be Properly Nested
  - XML Tags are Case Sensitive
- XML elements can have attributes, just like HTML.
  - Attributes are designed to contain data related to a specific element.

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

- XML Namespaces provide a method to avoid element name conflicts.

```
<root>
  <h:table xmlns:h="http://www.w3.org/TR/html4/">
  </h:table>

  <f:table xmlns:f="https://www.w3schools.com/furniture">
  </f:table>
</root>
```

XML Validation

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".
  - The purpose of a DTD is to define the structure of an XML document.
  - It defines the structure with a list of legal elements.

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

- An XML Schema (XSD) describes the structure of an XML document, just like a DTD.
  - XML Schema is an XML-based alternative to DTD.

```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

XML: XSLT

Alliance For Digital Employability

- XSL (eXtensible Stylesheet Language) is a styling language for XML.
- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation
- An XSL style sheet consists of one or more set of rules that are called templates.

  A template contains rules to apply when a specified node is matched.

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Form Data Validation

- Sending data is not enough — we also need to make sure that the data users fill out in forms is in the correct format we need to process it successfully, and that it won't break our applications.
- We also want to help our users to fill out our forms correctly and not get frustrated when trying to use our apps.
- Form validation helps us achieve these goals.

What is Form Validation?

- Go to any popular site with a registration form, and you will notice that they give you feedback when you don't enter your data in the format they are expecting.
- You'll get messages like:
  - "This field is required" (you can't leave this field blank)
  - "Please enter your phone number in the format xxx-xxxx" (it wants three numbers followed by a dash, followed by four numbers)
  - "Please enter a valid e-mail address" (the thing you've entered doesn't look like a valid e-mail address)
  - "Your password needs to be between 8 and 30 characters long, and contain one uppercase letter, one symbol, and a number" (seriously?)

Form Validation

Alliance For Digital Employability

- Form Validation is when you enter data and the web application checks it to see if it is correct.
- If so, it allows it to be submitted to the server and (usually) saved in a database; if not, it gives you error messages to explain what you've done wrong (provided you've done it right).
- Form validation can be implemented in a number of different ways.
- We want to make filling out web forms as non-horrible as possible, so why do we insist on blocking our users at every turn? There are **three** main reasons:
  - Get the right data, in the right format — our applications won't work properly if our user's data is stored in any old format they like, or if they don't enter the correct information in the correct places, or omit it altogether.
  - Protect our users — if they entered really easy passwords, or no password at all, then malicious users could easily get into their accounts and steal their data.
  - Protect ourselves — there are many ways that malicious users can misuse unprotected forms to damage the application they are part of.

Types of Form Validation

There are different types of form validation that you'll encounter on the web:

- **Client-side validation** is validation that occurs in the browser, before the data has been submitted to the server. This is more user-friendly than server-side validation as it gives an instant response. This can be further subdivided:

  JavaScript validation is coded using JavaScript. It is completely customizable.

  Built in form validation is done with HTML5 form validation features, and generally doesn't require JavaScript. This has better performance, but it is not as customizable.

- **Server-side validation** is validation that occurs on the server, after the data has been submitted — server-side code is used to validate the data before it is put into the database, and if it is wrong a response is sent back to the client to tell the user what went wrong. Server-side validation is not as user-friendly as client-side validation, as it requires a round trip to the server, but it is essential — it is your application's last line of defense against bad (meaning incorrect, or even malicious) data. All popular server-side frameworks have features for validating and sanitizing data (making it safe).
- In the real world, developers tend to use a combination of client-side and server-side validation, to be on the safe side.

Data Validation (1)

- Data validation is the process of ensuring that user input is clean, correct, and useful. Typical cases:
  - has the user filled in all required fields?
  - has the user entered a valid date?
  - has the user entered text in a numeric field?
- Data validation can occur on both the Server and the Client.
- **Server side validation** is performed by a web server, after input has been sent to the server.
- **Client side validation** is performed by a web browser, before input is sent to a web server.

✓ NOTE: both validations should take place, for security purposes.

Data Validation (2)

In many cases, data are sent to the server via HTML Forms.

- However, validation can be crucial on data.
- Validation can (and should) occur both on the server and the client.

HTML form validation can be done by JavaScript

- JavaScript Can Validate Text Input
- JavaScript Can Validate Numeric Input
- HTML form validation can also be performed automatically by the browser

If a form field is empty, the **required** attribute prevents this form from being submitted:

```
function validateForm() {
    var x = document.forms["myForm"]["fname"].value;
    if (x == "") {
        alert("Name must be filled out");
        return false;
    }
}
```

```
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

Alliance For Digital Employability

Class Activity | HTML build in validation Example

- https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation

Validating forms using JavaScript (1)

Constraint validation API properties

Property	Description
validationMessage	A localized message describing the validation constraints that the control does not satisfy (if any), or the empty string if the control is not a candidate for constraint validation (willValidate is false), or the element's value satisfies its constraints.
validity	A ValidityState object describing the validity state of the element.
validity.customError	Returns true if the element has a custom error; false otherwise.
validity.patternMismatch	Returns true if the element's value doesn't match the provided pattern; false otherwise. If it returns true, the element will match the :invalid CSS pseudo-class.
validity.rangeOverflow	Returns true if the element's value is higher than the provided maximum; false otherwise. If it returns true, the element will match the :invalid and :out-of-range and CSS pseudo-class.
validity.rangeUnderflow	Returns true if the element's value is lower than the provided minimum; false otherwise. If it returns true, the element will match the :invalid and :out-of-range CSS pseudo-class.
validity.stepMismatch	Returns true if the element's value doesn't fit the rules provided by the step attribute; otherwise false . If it returns true, the element will match the :invalid and :out-of-range CSS pseudo-class.

Validating forms using JavaScript (2)

Constraint validation API properties

Property	Description
validity.tooLong	Returns true if the element's value is longer than the provided maximum length; else it will be false If it returns true, the element will match the :invalid and :out-of-range CSS pseudo-class.
validity.typeMismatch	Returns true if the element's value is not in the correct syntax;otherwise false. If it returns true, the element will match the :invalid css pseudo-class.
validity.valid	Returns true if the element's value has no validity problems; false otherwise. If it returns true, the element will match the :valid css pseudo-class; the :invalid CSS pseudo-class otherwise.
validity.valueMissing	Returns true if the element has no value but is a required field; false otherwise. If it returns true, the element will match the :invalid CSS pseudo-class.
willValidate	Returns true if the element will be validated when the form is submitted; false otherwise.

Validating forms using JavaScript (3)

Constraint validation API methods

Method	Description
checkValidity()	Returns true if the element's value has no validity problems; false otherwise. If the element is invalid, this method also causes an invalid event at the element.
setCustomValidity(*message*)	Adds a custom error message to the element; if you set a custom error message, the element is considered to be invalid, and the specified error is displayed. This lets you use JavaScript code to establish a validation failure other than those offered by the standard constraint validation API. The message is shown to the user when reporting the problem. If the argument is the empty string, the custom error is cleared.

Class Example

- HTML

```
<form novalidate>
  <p>
    <label for="mail">
      <span>Please enter an email address:</span>
      <input type="email" id="mail" name="mail">
      <span class="error" aria-live="polite"></span>
    </label>
  </p>
  <button>Submit</button>
</form>
```

Alliance For Digital Employability

Class Example
Using the constraint validation API

- CSS

  JS-L2_CSS_ClassExercise.cs

Alliance For Digital Employability

Class Example
Using the constraint validation API

- JavaScript code handles the custom error validation

```
// There are many ways to pick a DOM node; here we get the form itself and the email
// input box, as well as the span element into which we will place the error message.

var form  = document.getElementsByTagName('form')[0];
var email = document.getElementById('mail');
var error = document.querySelector('.error');

email.addEventListener("input", function (event) {
  // Each time the user types something, we check if the
  // email field is valid.
  if (email.validity.valid) {
    // In case there is an error message visible, if the field
    // is valid, we remove the error message.
    error.innerHTML = ""; // Reset the content of the message
    error.className = "error"; // Reset the visual state of the message
  }
}, false);
form.addEventListener("submit", function (event) {
  // Each time the user tries to send the data, we check
  // if the email field is valid.
  if (!email.validity.valid) {

    // If the field is not valid, we display a custom
    // error message.
    error.innerHTML = "I expect an e-mail, darling!";
    error.className = "error active";
    // And we prevent the form from being sent by canceling the event
    event.preventDefault();
  }
}, false);
```

Class Example
Using the constraint validation API

- Here is the live result

Class Activity

Alliance For Digital Employability

- Apply the Class example above to one of your pages (or create a new simple form page)

Summary on Form validation

- Form validation does not require complex JavaScript, but it does require thinking carefully about the user.
- Always remember to help your user to correct the data he provides. To that end, be sure to:
  - Display explicit error messages.
  - Be permissive about the input format.
  - Point out exactly where the error occurs (especially on large forms).

References

- https://www.w3schools.com/js/
- https://www.w3schools.com/xml/default.asp
- https://www.w3schools.com/xml/xsl_intro.asp

Alliance For Digital Employability

4.4 JavaScript (II)

Q&A
Exercises and Review Questions

Sample Question #1

A JavaScript _____ is a block of code designed to perform a particular task.

A. variable

B. error

C. function

D. object

Sample Question #1 – Answer

A JavaScript _____ is a block of code designed to perform a particular task.

A. variable

B. error

C. function

D. object

Sample Question #2

Consider the XML code to the right. This code:

A. Has a syntax error
B. Is error free
C. Is not standard XML code
D. Is JavaScript, not XML

```
<?xml version="1.0" encoding="UTF-8"?>
- <note>
    <to>Panos</to>
    <from>Stacey</Ffrom>
    <heading>Reminder</heading>
    <body>Don't forget we have exams this weekend!</body>
  </note>
```

Alliance For Digital Employability

Sample Question #2 - Answer

Alliance For Digital Employability

Consider the XML code to the right. This code:

A. Has a syntax error

B. Is error free

C. Is not standard XML code

D. Is JavaScript, not XML

```
<?xml version="1.0" encoding="UTF-8"?>
- <note>
    <to>Panos</to>
    <from>Stacey</Ffrom>
    <heading>Reminder</heading>
    <body>Don't forget we have exams this weekend!</body>
  </note>
```

Sample Question #3

Alliance For Digital Employability

What will the following JS code do?

```
<!DOCTYPE html>
<html>
<body>
<button
onclick="document.getElementById('dem
o').innerHTML=Date()">The time is?
</button>
<p id="demo"></p>
</body>
</html>
```

A. Will create a button, which when clicked will take us to the InnerHTML page

B. Will create a link named The time is?, which when clicked will take us to a page with the current date and time

C. Will create a link named Demo, which when clicked internally navigate to an area where a preset date is displayed

D. Will create a button, which when clicked will display the current date and time

Sample Question #3 - Answer

Alliance For Digital Employability

What will the following JS code do?

```
<!DOCTYPE html>
<html>
<body>
<button
onclick="document.getElementById('dem
o').innerHTML=Date()">The time is?
</button>
<p id="demo"></p>
</body>
</html>
```

A. Will create a button, which when clicked will take us to the InnerHTML page

B. Will create a link named The time is?, which when clicked will take us to a page with the current date and time

C. Will create a link named Demo, which when clicked internally navigate to an area where a preset date is displayed

D. Will create a button, which when clicked will display the current date and time

Class Review Exercise #1

- Write a JavaScript program that accept two integers and display the larger.

Challenge: What if the user gives as input a non-numeric value?

Class Review Exercise #1 Sample Solution

Alliance For Digital Employability

JavaScript (index.js)

```
var num1 = window.prompt("Input the First integer", "0");
var num2 = window.prompt("Input the second integer", "0");
num1 = parseInt(num1, 10);
num2 = parseInt(num2, 10);
if(isNaN(num1)) {
  alert("First input value is NaN.");
} else if (isNaN(num2)) {
  alert("Second input value is NaN.");
} else {
  if (num1 > num2) {
    alert("The larger of " + num1 + " and " + num2 + " is " + num1 + ".");
  }
  else if (num1 < num2) {
    alert("The larger of " + num1 + " and " + num2 + " is " + num2 + ".");
  }
  else {
    alert("The values " + num1 + " and " + num2 + " are equal.");
  }
}
```

HTML

```
<html>
    <head>
        <meta charset=utf-8 />
        <script src="index.js"></script>
        <title>Write a JavaScript program that accept two integers and display the larger</title>
    </head>
    <body>
    </body>
</html>
```

Class Review Exercise #2

- Write a JavaScript program that takes a positive number as argument and prints all the numbers in the console, along with either “Odd” or “Even” based on what they are.

Class Review Exercise #2 Sample Solution

Alliance For Digital Employability

JavaScript (index.js)

```
//Declare function.
function oddOrEven(limit) {
      if(limit <= 0) {
      return;
      }
  for (var i = 1; i <= limit; i++) {
    if(i % 2 == 0) {
      console.log(i + ": Even");
    } else {
      console.log(i + ": Odd");
    }
  }
}
//Call it.
oddOrEven(100);
```

HTML

```
<html>
      <head>
            <meta charset=utf-8 />
            <script src="index.js"></script>
            <title>Odd or Even program</title>
     </head>
     <body>
   </body>
</html>
```

Class Review Exercise #3

Alliance For Digital Employability

- Describe the attributes of a cell phone in both XML and JSON format:
  - Brand: Samsung,
  - Model: Galaxy S8,
  - Cores: 8,
  - Storage in GB: 64
  - Screen Size in Inches: 5.8.
  - Features: SMS, MMS, Email, Push Mail, IM

Class Review Exercise #2 Sample Solution

Alliance For Digital Employability

cellphone.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<Brand>Samsung</Brand>
<Model>Galaxy S8</Model>
<Cores>8</Cores>
<StorageGB>64</StorageGB>
<ScreenSizeInches>5.8</ScreenSizeInches>
<Features>SMS</Features>
<Features>MMS</Features>
<Features>Email</Features>
<Features>Push</Features>
<Features>Mail</Features>
<Features>IM</Features>
```

cellphone.json

```
{
        "Brand": "Samsung",
        "Model": "Galaxy S8",
        "Cores": 8,
        "StorageGB": 64,
        "ScreenSizeInches": 5.8,
        "Features": ["SMS", "MMS", "Email", "Push", "Mail", "IM"]
}
```

Exercise #1

Alliance For Digital Employability

- Write a JavaScript program where the program takes a random integer from 1 to 10, the user is then prompted to input a guess number. If the user input matches with guess number, the program will display a message "Good Work" otherwise display a message "Not matched".

Tip 1: use Math.random() that returns a random number between 0 (inclusive) and 1 (exclusive).

Tip 2: use Math.ceil(number) that returns the smallest integer greater than or equal to a given number.

Exercise #2

- Write a JavaScript program to calculate multiplication and division of two numbers (input from user).

Challenge: Are there any exceptional cases? If yes, how can we handle them?

Exercise #3

- Write a JavaScript program which iterates the integers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

- **Challenge/Discussion: What if we wanted to do this for 1000 integers as easy as possible?**

Exercise #4

Alliance For Digital Employability

Addition From Part I, Exercise 3:

- Create a car object of a Ferrari F430. This should contain the following properties: brand, model, max speed (330 km/h), current speed, status of whether the car has started or not. (….)

- Describe the car properties of Part 1 exercise 3 in both valid XML and JSON format. Additionally, add an array of its monthly max speeds reached this year:

  250km/h for January 2017

  300km/h for February 2017

  278.5km/h for March 2017.

  Note: Consider year and month as attributes.

Exercise #5

Alliance For Digital Employability

- Create appropriate form validation in one of your HTML pages using:
  - Built in form validation and HTML5 features
  - JavaScript validation

Further Reading:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#Nested_try-blocks
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch

- https://developer.mozilla.org/en-US/docs/Web/Events
- http://www.javascriptkit.com/jsref/events.shtml
- https://www.w3schools.com/js/js_htmldom_events.asp
- https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation

Alliance For Digital Employability

Thank you

Alliance
For Digital
Employability