COS10011/60004
Creating Web Applications

Lecture 5
JavaScript Part 1

Separate behaviour from content and presentation

Behaviour / Interactivity
Use **scripting** to control content behaviour

✂ Separate

Presentation
Use **CSS** to present the content

✂ Separate

Structured Content
Use **HTML** to describe the content

Increased Richness of the User Experience
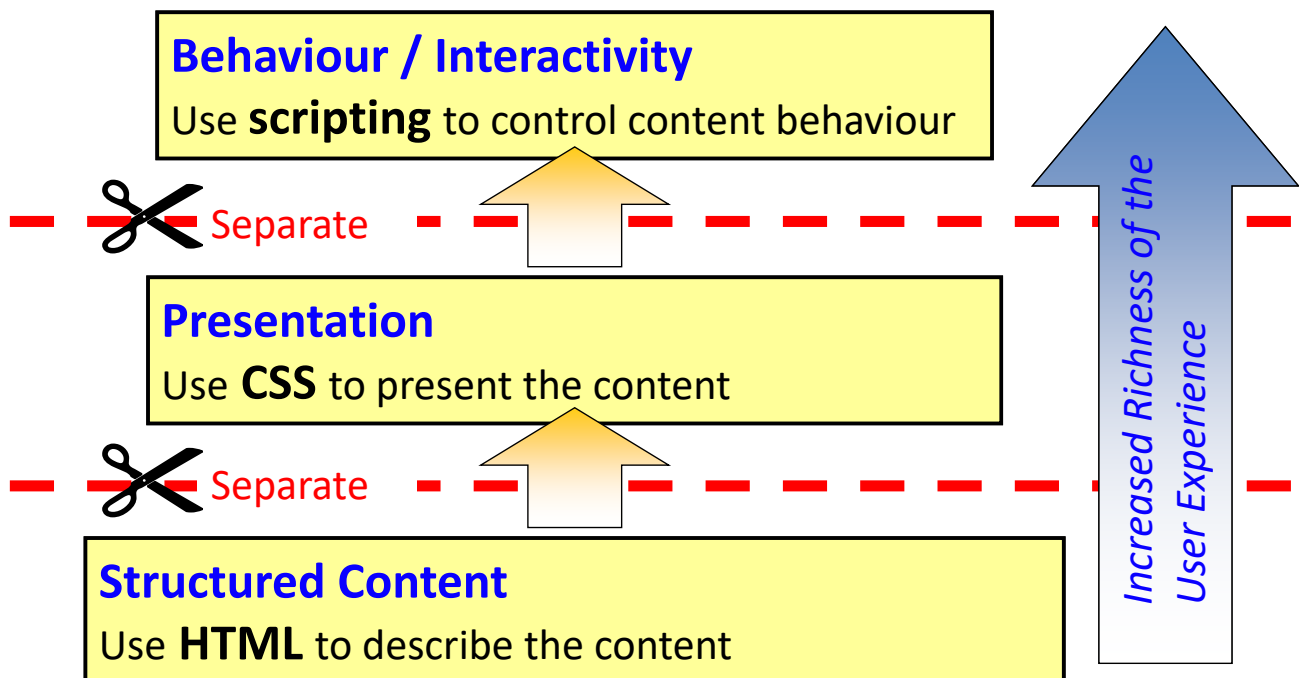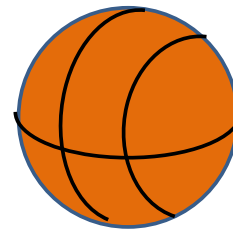
Work from the bottom up !

Programming Objects

- For example, imagine an object, say a ball.

  With this ball we can ask a couple of generalised "what" questions
    - What *attributes* does the ball have?
    - What can the ball *do*?
    - What *events* can happen to a ball?

Programming Objects (continued)

We can make the questions more specific:

- **What attributes does our ball have?**
  - What colour?
  - What size?
  - What weight?

  > Referred to as **properties.**
  > *things = nouns*

- **What can the ball do?**
  - What can we do with it - throw

  > Referred to as **methods**
  > *actions = verbs*

- **What events can happen to a ball?**
  - Is dropped
  - Is thrown

  > Referred to as **events**
  > *events = (intransitive) verbs*

JavaScript Objects

In JavaScript, we say

- **What properties does a ball have?**
  ```
  ball.colour;
  ball.size;
  ball.weight;
  ```
  Dot notation

- **What methods does a ball have?**
  ```
  ball.bounce();
  ball.hit();
  ```
  Note parenthesis

- **What events can happen to a ball?**
  ```
  ball.isDropped
  ball.isThrown
  ```
  Note *no* parenthesis

Document Object – Example

This example is a HTML document. But this applies to any XML document.
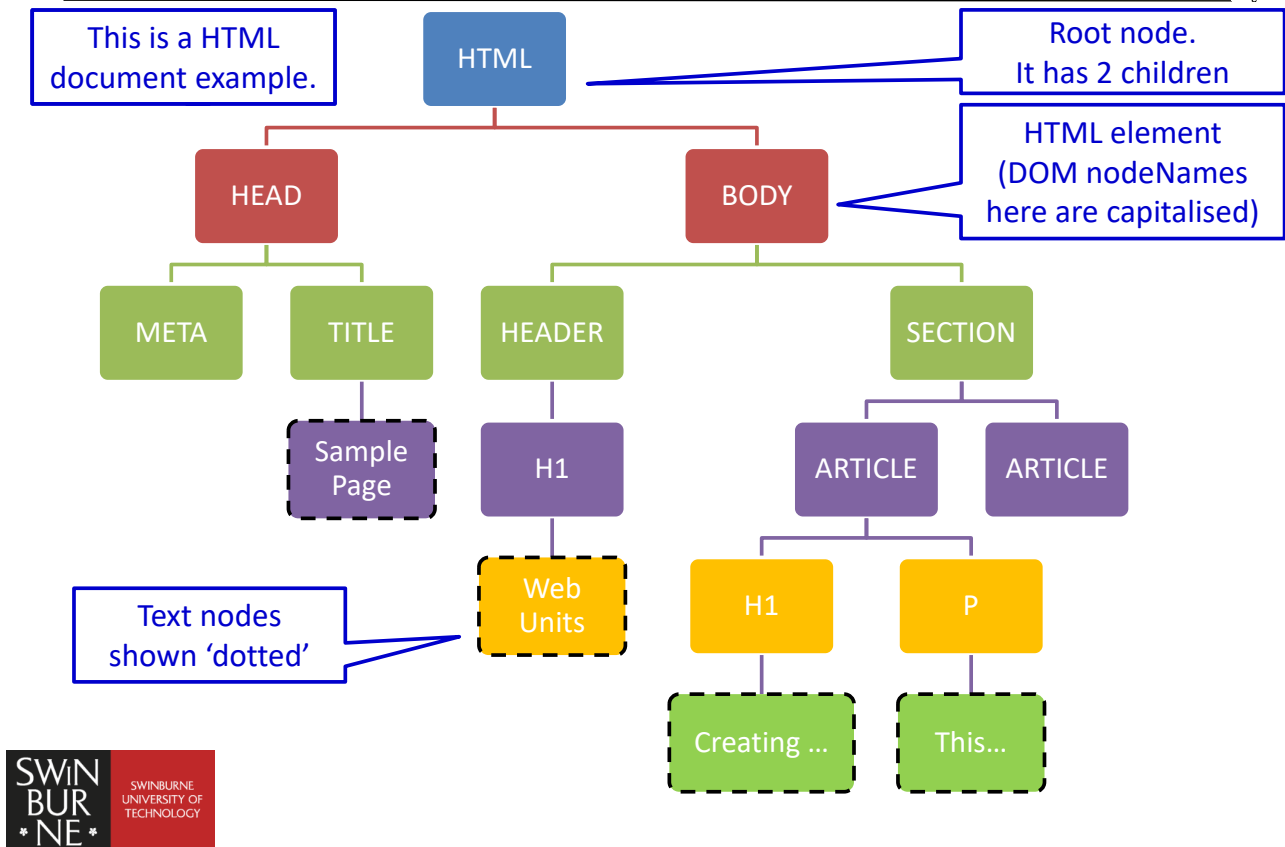
- A **document** is represented as a tree of nodes
- The first node is referred to as the **root node**
- Each node can have **children**
- A node with no children is referred to as **leaf node**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Sample Page</title>
</head>
<body>
  <header>
    <h1 id="pgHead">Web Units</h1>
  </header>
  <section>
    <article>
      <h1>Creating Web Apps</h1>
      <p>This unit covers … </p>
    </article>
    <article> …
    </article>
  <section>
</body>
</html>
```

Document Object – Tree Structure

This is a HTML document example.

HTML

Root node.
It has 2 children

HEAD

BODY

HTML element
(DOM nodeNames here are capitalised)

META

TITLE

HEADER

SECTION

Sample Page

H1

ARTICLE

ARTICLE

Text nodes shown 'dotted'

Web Units

H1

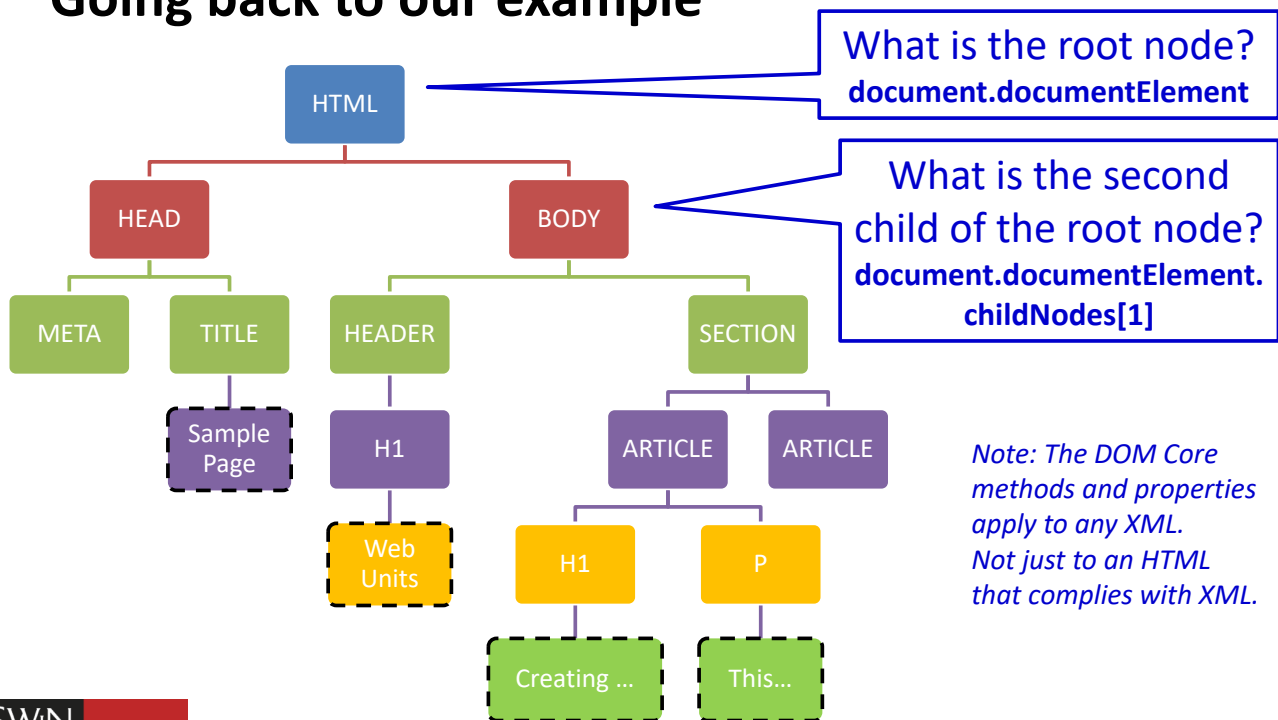**P**

Creating …

This…

Document Object

Where are the objects?

- The entire HTML page is made up of **objects**
- Using the tree representation,
  each node is an **object.**
- In our example, we have
  16 nodes or 16 objects
- We can using either
  - HTML objects or
  - **DOM Core** properties and methods
  (more on this later …)

Document Object – Tree Structure

Going back to our example

What is the root node?
document.documentElement

What is the second child of the root node?
document.documentElement.childNodes[1]

*Note: The DOM Core methods and properties apply to any XML.
Not just to an HTML that complies with XML.*

DOM - Document Object Model

- A cross-platform and language-independent API (Application Programming Interface) that treats an HTML, XHTML, or XML document as a tree structure, wherein each node is an object representing a part of the document.

- The objects can be manipulated programmatically and any visible changes occurring as a result may then be reflected in the display of the document

JavaScript and DOM

- JavaScript originally developed as an HTML manipulation scripting language by Netscape

- Java-like syntax

- Standardised as ECMAScript

- DOM is not part of core JavaScript, but JavaScript uses the DOM to interact with the Web browser. This technique is referred to as **DOM manipulation**

- DOM uses JavaScript's Core Objects
  such as Array, Boolean, Date, Math, Number, RegExp, String, …

DOM - Document Object Methods

- Some useful methods of **document** object

  ```
  getElementById()

  getElementsByName()

  getElementsByTagName()

  createElement()

  createTextNode()

  createAttribute()
  ```

Pre-defined object

HTML DOM Events

Document object can be *registered* to respond to events that happen in the browser.
An "event handler" is created for the object

- Mouse events
  - onclick, ondblclick, onmouseup, onmousedown, onmouseover, onmousemove, …
- Keyboard events
  - onkeydown, onkeyup, onkeypress, …
- Form events
  - onblur, onchange, onfocus, oninvalid, onsubmit, …
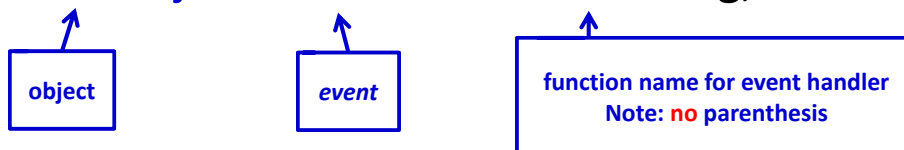- Drag events, animation, clipboard, print, media, transition, …

http://www.w3schools.com/jsref/dom_obj_event.asp

Creating an event listener in JavaScript

Listener for when the mouse clicks on an HTML object:
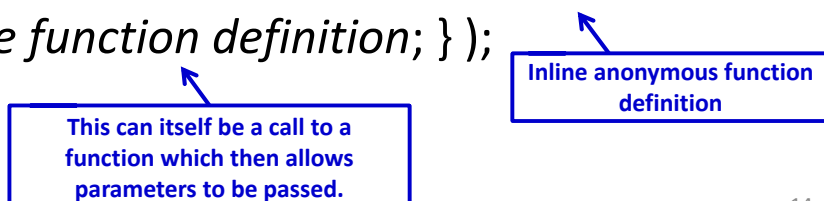
myDOMobject.**onclick** = doSomething;

object	event	function name for event handler
		Note: **no parenthesis**

OR

method

myDOMobject.**addEventListener**("click", doSomething);

Note: **no** *on-* prefix

OR

myDOMobject.**addEventListener**("click", function(){
  some function definition; });

This can itself be a call to a function which then allows parameters to be passed.

Inline anonymous function definition

A very simple example - HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Lecture 5 Demo</title>
    <meta charset="utf-8" />
    <meta name="description" content="Reading and writing to an HTML doc" />
    <script src="lect5_html_io.js"></script>
</head>
<body>
    <h1>Input and Output using JavaScript</h1>
    <p id="clickme" >Click me - to run 'prompt' and display 'alert'</p>
    <p><span id="mymessage"></span></p>
</body>
</html>
```

Create reference to JavaScript file from your HTML

Note <script> element has a closing tag </script>

Identify parts of the HTML that will respond to JS

JavaScript: Adding to HTML

- **Embedded in HTML element (Inline)** ✗

    `<input type="button" value="Back" onclick="clickme( )" />`
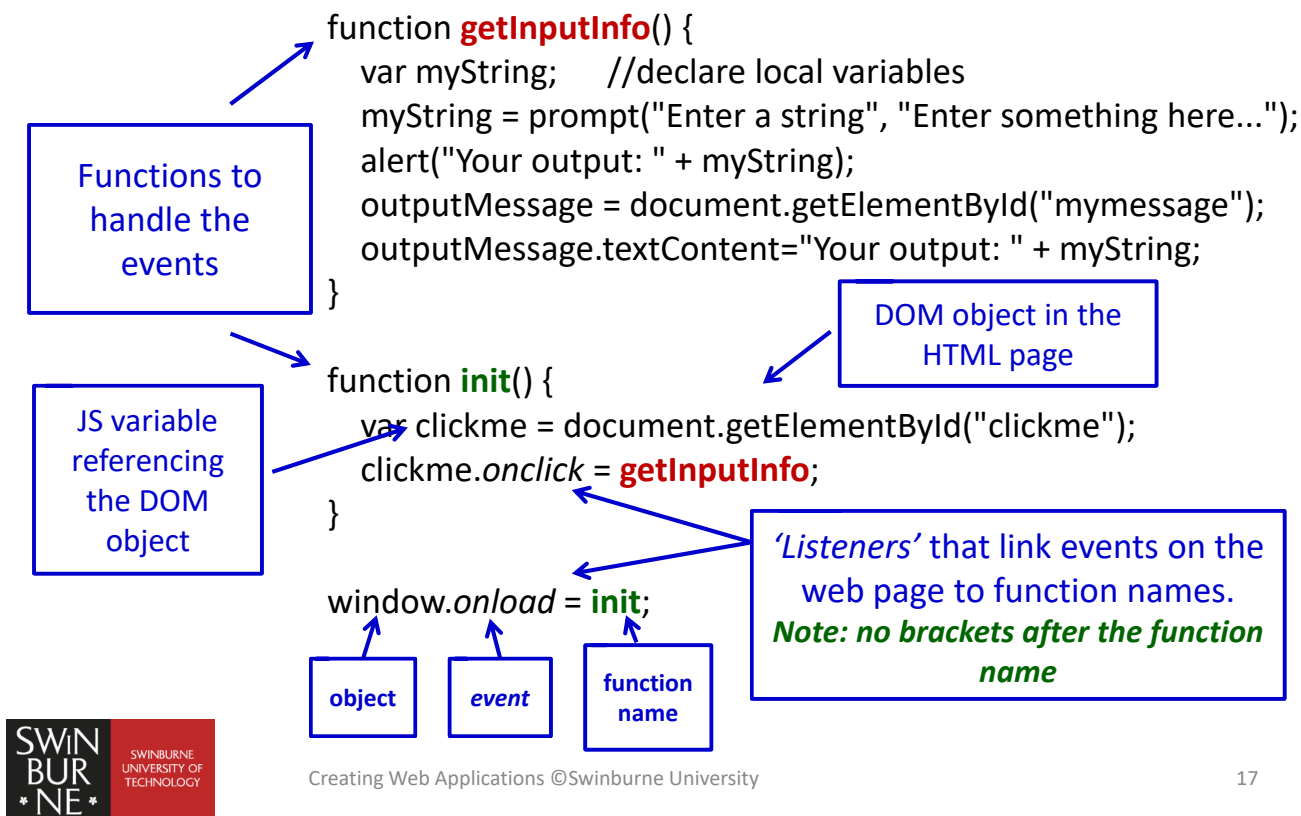
- **Embedded within HTML header** ✗

    `<script>` … `</script>`

- Include reference to an external file ✓

    `<script src="lect5_html_io.js"></script>`

> **CWA best practice, mandated approach:**
> - Separates behaviour from content
> - Can be *cached* by user's web browser, if needed by multiple webpages, it is only downloaded once.

A very simple example - JavaScript

```javascript
function getInputInfo() {
    var myString;     //declare local variables
    myString = prompt("Enter a string", "Enter something here...");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}

function init() {
    var clickme = document.getElementById("clickme");
    clickme.onclick = getInputInfo;
}

window.onload = init;
```

Functions to handle the events

JS variable referencing the DOM object

DOM object in the HTML page

'Listeners' that link events on the web page to function names.
Note: no brackets after the function name

object

event

function name

In Summary…

1. **HTML file**
    i. Create reference to JavaScript file from your HTML
        ```html
        <script src="myscript.js"></script>
        ```
    ii. Identify parts of the HTML that will respond to JS
        e.g. have id attributes on elements that will be referenced

2. **JavaScript file**
    i. Define 'listeners' that link events on the Web page to function names
        e.g. window.onload = init;   *or*   button.onclick = do_something;
    ii. Write the functions to handle the events
        ```javascript
        function do_something() {
            alert("This displays an alert box");
        }
        ```
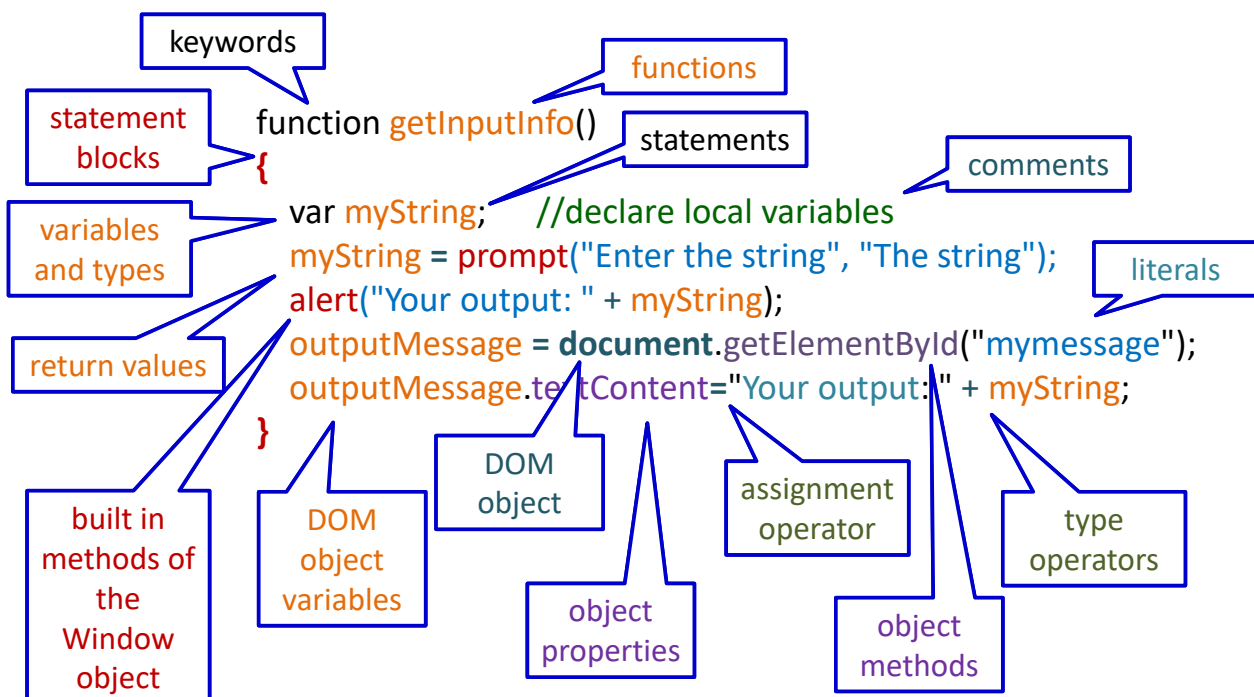
A JavaScript Template

```
/* Filename: [ name of this file…].js
   Target html: [ what is the html file(s) linked to this js…]
   Purpose : [ a html file may have multiple js files. What does this one do?…]
   Author: [ your name…]
   Date written: [ …]
   Revisions:  [ your name, what, when…]
*/

// [ brief comment on what the function does…]
function init() {
}

window.onload = init;
```

JavaScript – Language Syntax

JavaScript – This lecture

- Comments

- Statement, blocks and naming rules

- Variables

- Data types

- Operators and expressions

- Functions and scope

JavaScript – Language Syntax

Comments

Any text between /* and */ will be ignored by JavaScript.

```
/*use multiline comments for header comments  at the start of your file
or to explain a function*/
function getInputInfo()
{
    var myString;     //Explain any code that may not be obvious
    myString = prompt("Enter the string    "The string");
    alert("Your output: " + myString);
    outputMessage = document.getEle        Id("mymessage");
    outputMessage.textContent="Your            + myString;
}
```

Any text between // and the end of the line will be ignored by JavaScript.

JavaScript – Language Syntax

Statements, Blocks and Naming rules

Reserved JavaScript words

Programmer defined names

A script is a sequence of statements terminated with **;**

```
function getInputInfo( )
{
    var myString;     //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " +        myString;
}
```

Statements can be grouped together using braces **{ }**

Ignores whitespace, tabs, and newline characters except when part of string constants. They can be added as needed for readability.

JavaScript - keywords

- *Keywords* (reserved words) that have special meanings within the language syntax, such as

abstract boolean break byte case catch char class const continue debugger default delete do double else enum export extends false final finally float for function goto if implements import in instanceof int interface long native new null package private protected public return short static super switch synchronized this throw throws transient true try typeof undefined var void volatile while with

JavaScript - Language

Variables

```
var anotherGlobalVariable = "";    //global because outside a function

function getInputInfo()
{
    var myString;    //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    var outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}
```

Variable local to the function

JavaScript – Naming variables

- JavaScript is case sensitive
  - use camelCase or under_score for compound names (be consistant)
- Use meaningful names for variable *identifiers*
- Identifiers must start with either *any letter of the alphabet* or *underscore*
- Can include any letter of the alphabet, digits 0-9, and underscore
- Cannot include spaces, or punctuation characters such as comma, full stop
- By convention:
  - variables start with lower case letter e.g. var **myString**="Hello";
  - constants are upper case e.g. const **MAX_LENGTH** = 7;

Variable Declaration

- Specifying and creating a variable name is called **declaring** the variable

- Assigning a first value to a variable is called **initialising** the variable

- The way a variable is declared defines which statements can see the variables. These are
  - Global can be seen anywhere in the file
  - Local can only be seen by within a scope
    - **var** scope is the function
    - **let** scope is the block in which it is declared {let … }

A 'scope' defines from where the variable is accessible

Local Variable Declaration - **var**

- Declared within a function

- Local variable can be declared using the **var** keyword
  - declaring one variable
    `var firstName;`
  - declaring multiple variables
    `var firstName, lastName;`
  - declaring and assigning one variable
    `var firstName = 'Java';`
  - declaring and assigning multiple variables
    `var firstName = 'Java', lastName = 'Script';`

A **var** local variable's lifetime is the same as its the function

Understanding Variable Scope

- **Variable scope** is 'where in your program' a declared variable can be used

- A variable's scope can be either **global** or **local**

- A **global variable** is one that is declared inside or outside a function and is available to all parts of your program

- A **local variable** is one that is declared using the **var** keyword *inside a function* and is *only available within that function*

JavaScript – Variable Scope

```
var myGobalVariable = "";                Global
                                          Scope

function getInputInfo() {
    var myString;      //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}
                                          Local scope

function init() {
    var clickme = document.getElementById("clickme");
    clickme.onclick = getInputInfo;
}

window.onload = init;
```

Not so good ☹ *Why?*

JavaScript - Language

Variables

var anotherGlobalVariable = "";   //global because outside a function

Variable local to the function

Functions are also variables in JavaScript!

```
function getInputInfo()
{
    var myString;     //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}
```

Careful!! This is a global variable

Variable Declaration (Global)

- You must **declare and initialise a _global_ variable in the same statement** `variable_name = value;`

    e.g.   x = 3;

- Best explicitly declared outside functions

    e.g.   var x = 3;

        function myFunction(x) { ... }

Lifetime of a global variable: until the page closes

- You can change the global variable's value at any point _from anywhere_ ☹

    e.g. **function** myOtherFunction( ) {x = 4;}

This can lead to unintended "side effects"

Global variables should be avoided if possible.

Recommend using Strict Mode:
Declare `"use strict";` at the start of your JS file
    => Variables must be explicitly declared
    e.g. x=3 will cause an error, so declare with var: var x=3;
http://www.w3schools.com/js/js_strict.asp

Constants

- Used to contain information that does not change during the course of program execution

- Declared with the **const** keyword.

- must start with a letter or underscore and can contain alphabetic, numeric, or underscore characters

- By convention use letters in uppercase.

> Block scope

const PI = '3.14';

> Note: Style convention is that constants are in UPPER CASE

What's the output?

```javascript
// all functions usually grouped together
// in one location
function testScope() {
  var y;

  x = "Changed";
  y = "Changed";
}

x = "Original";
y = "Original";
testScope();
alert (x + " " + y);
```

What's the output?

- A local variable only exists *within* the function where it is declared
- Thus only the global variable is changed

Output

Changed Original
OK
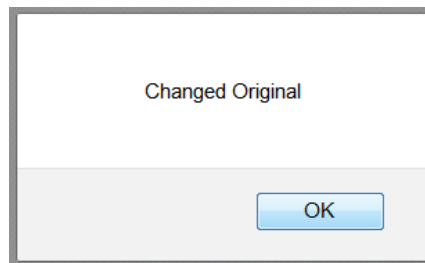
How many variables are declared here?

```
// all functions usually grouped together
// in one location
function testScope() {
  var x;


  x = "Changed";
  y = "Changed";
}

x = "Original";
y = "Original";
testScope();
alert (x + " " + y);
```

Answer: 3

Any errors here?

```
// variables must be declared before they can be
   used
function testScope() {
  var y;

  x = "Changed";
  y = "Changed";
}

x = "Original";

testScope();
alert (x + " " + y);
```

No Output

Error, as **y** does not exist outside the function

JavaScript - Language

Variables, Data types

JavaScript is a *dynamically-typed* language

We don't know the type yet because nothing is assigned to it

```
function getInputInfo()
{
    var myString;    //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
```

Now it's a String

This is a reference to an object on an HTML page

JavaScript has *dynamic* data types

JavaScript dynamically determines the type of a variable from what is assigned to it, unlike strongly typed languages such as C and Java, .

```
var x;                  // Now x is undefined
x = 5;          // Now x is a Number
x = "John";     // Now x is now a String
```

Primitive Data Types

- String

- Number
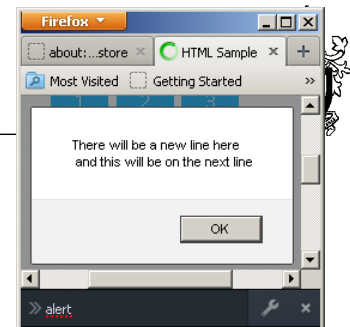
- Boolean

- Null

- Undefined

- Symbol - ECMAScript 6

String

- Is a sequence of characters
- created directly by placing the series of characters between double or single quotes, for example
  - "This is a string"
  - 'This is also a string'

String

- Uses embedded control characters
- For example,

```
alert("There will be a new line here\n and this
will be on the next line");
```

Seq	Usage	Seq	Usage
\b	backspace	\\	backslash
\t	horizontal tab	\"	double quote
\n	newline	\'	single quote
\r	carriage return	\###	Octal encoded character
\f	formfeed	\uHHHH	Unicode encoded character

Number

- **Integers** can be positive, 0, or negative
- **Integers** can be expressed in
  - **Decimal -** integer literal consists of a sequence of digits without a leading 0 (zero)

    example: 255
  - **Octal – (**digits 0-7) A leading 0 (zero) on an integer literal indicates it is in octal

    example: 0377 $= 255_{10}$
  - **Hexadecimal -** Hexadecimal integers can include digits (0-9) and the letters a-f and A-F.

    A leading 0x (or 0X) indicates hexadecimal.
    example: 0xFF $= 255_{10}$

Number

- A **floating-point number** can contain either
  - a decimal point
  - an "e" (uppercase or lowercase) which is used to represent "ten to the power of" in scientific notation
  - or both
- exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-")

  1.025e3 = 1.025 x 10^3 = 1025.0

  130e-3 = 130 x 10^{-3} = 0.130

Boolean

- **Boolean values** are **true** and **false**
- These are special values, and are not usable as 1 and 0.
- In a comparison,
  - any expression that evaluates to **0** is taken to be **false**, and
  - any expression that evaluates to a number other than **0** is taken to be **true**

Null and Undefined

- **Null** - Not the same as zero - no value at all. A null value is one that has no value and means nothing
- **Undefined** - A value that is undefined is a value held by a variable after it has been created, but before a value has been assigned to it

JavaScript – Language Syntax

Operators and Expressions

```
function getInputInfo()
{
    var myString;      //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent = "Your output: " + myString;
}
```

Operators combine *operands* (literals / variables/ constants) into *expressions* for *evaluation*

an *expression* can be *assigned* after it is *evaluated*

assignment operator

String literal

String type operator *concatenation*

Variable of type *String*

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators

An operator is associated with a type

Operator Type	Description
String	Performs operations on strings
Arithmetic	Performs mathematical calculations
Assignment	Assigns values to variables
Comparisons	Compares operands and returns a Boolean value
Conditional	Assigns values to variables based on the condition
Logical	Performs Boolean operations on Boolean values

- A **binary operator** requires *both* an operand before and after the operator, e.g. `x = 2 + 3;`

- A **unary operator** requires a *single* operand either before or after the operator, e.g. `x++;`

String Operator

- **String operator** is used to concatenate two string

"Your output: " + myString;

Operator	Name	Description
+	Concatenation	Joins two operands

"This text" + "That Text"

- Results to

"This TextThat Text"

No blank space will be inserted

Arithmetic Operators (Binary)

- **Arithmetic operators** are used to perform mathematical calculations

Operator	Name	Description
+	Addition	Adds two operands
-	Subtraction	Subtracts one operand from another operand
*	Multiplication	Multiplies one operand from another operand
/	Division	Divides one operand by another
%	Modulus	Divides one operand by another and returns the remainder

Mixed types

- As JavaScript is weakly typed, numbers are automatically converted to string when displayed

- `alert(4+"You");`
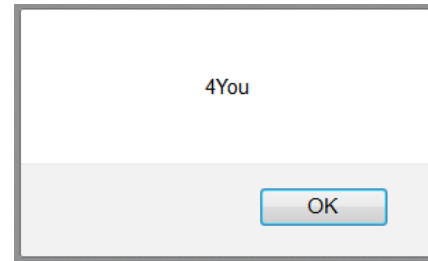
Assignment Operators

- **Assignment operators** are used for assigning a value to a variable:

  `myFavoriteSuperHero = "Batman";`

- **Compound assignment operators** perform mathematical calculations on variables and literal values in an expression, and then assign a new value to the left operand

Arithmetic Assignment Operators (Unary)

- The increment (++) and decrement (--) unary operators can be used as prefix or postfix operators
- A **prefix operator** is placed before a variable
- A **postfix operator** is placed after a variable

Operator	Name	Description
++	Increment	Increases an operand by a value of one x++; is same as x= x+1;
--	Decrement	Decreases an operand by a value of one x--; is same as x= x-1;

Assignment Operators

- Some operators are created to allow the use of fewer characters of code

```
x = 100;
y = 200;
x += y;    same as x = x + y;


x = 2;
y = 6;
x *= y;    same as x = x * y;
```

Assignment Operators

Operator	Name	description
=	Assignment	Assigns the value of the right operand to the left operand
+=	Compound addition assignment	Adds the value of the right operand to the value of the left operand and assigns the sum to the left operand x +=y same as x = x + y;
-=	Compound subtraction assignment	Subtracts the value of the right operand to the value of the left operand and assigns the difference to the left operand x -=y same as x = x - y;
*=	Compound multiplication assignment	Multiplies the value of the right operand to the value of the left operand and assigns the product to the left operand x *=y same as x = x * y;
/=	Compound division assignment	Divides the value of the right operand to the value of the left operand and assigns the quotient to the left operand
%=	Compound modulus assignment	Divides the value of the right operand to the value of the left operand and assigns the remainder (modulus) to the left operand x %=y same as x = x % y;

Comparison Operators

- **Comparison operators** are used to compare two operands and determine how one operand compares to another
- A **Boolean value** of **true** or **false** is returned after two operands are compared
- The comparison operator compares values, whereas the assignment operator assigns values
- Comparison operators are used with **conditional statements** and **looping statements.**

equals comparison operator

evaluates to *true* or false

```
if (guess == secret){
    msg = "Correct number: "+secret;}
```

We will discuss conditional control structures next week…

Comparison Operators

Operator	Name	Description
==	Equal	Returns true if the operands are equal
===	Strict equal	Returns true if the operands are equal and of the same type
!=	Not equal	Returns true if the operands are not equal
!==	Strict not equal	Returns true if the operands are not equal or not of the same type
>	Greater than	Returns true if the left operand is greater than the right operand
<	Less than	Returns true if the left operand is less than the right operand
>=	Greater than or equal	Returns true if the left operand is greater than or equal to the right operand
<=	Less than or equal	Returns true if the left operand is less than or equal to the right operand

---

Conditional (ternary) Operators

- **Conditional Operators** are used to return one of two values, based on the results of a condition. *(Note this has 3 operands)*

- The syntax for the conditional operator is:
  ```
  condition ? expression1 : expression2
  ```

- If the `condition` evaluates to true, `expression1` is returned, else `expression2` is returned. e.g.
  ```
  var ableToDrive =
    (age < 18) ? "Too young":"Old enough";
  var theFee = isMember ? "$2.00":"$10.00";
  ```

Logical Operators

- **Logical operators** are used for comparing two Boolean operands for equality

- A Boolean value of true or false is returned after two operands are compared

Operator	Name	Description
&&	And	Returns true if both the left operand and right operand return a value of true; otherwise, it returns a value of false
||	Or	Returns true if either the left operand or right operand returns a value of true; if neither operand returns a value of true, it returns a value of false
!	Not	Returns true if an expression is false and returns false if an expression is true

AND logical operator

```
if ((guess == secret) && (guess < 7)){…}
```

Question ???

Does the expression

```
((guess == secret) && (guess < 7))
```

evaluate the same as

```
(guess == secret && guess < 7)
```

- *Sometimes it helps to add brackets to expressions to make **order of precedence** clear to the reader (even if they are not strictly needed)*

Operator Precedence

- **Operator precedence** determines the order in which operators are evaluated.

- Starting from the highest precedence with the operators presented, we have
  - Arithmetic operators (unary)
  - Arithmetic operators (binary - *, /, % then +, -)
  - Comparison operators
  - Logical operators
  - Assignment operators

Evaluation of Expression

Consider the following examples:

- 25 + 100 * 4;

   Is it 425 or 500?

- 4*2+4;

   Is it 24 or 12?

- 4*(2+4);

   Is it 24 or 12?

- 7 % 5   (Modulus: What is the remainder left over when 7 is divided by 5)

   How about this?

Evaluation of Expression

Given that x = 6 and y = 3

What is the value of x in the following statements after assignment?

- x = x + y;

- x = x % y;

- x++;

What is the result returned after evaluating the following expression?

- (x < 10 && y > 1)

   (true AND true) → true

- (x==5 || y==5)

   (false OR false) → false

- !(x==y)

   not(false) → true

- x===5

JavaScript - Functions

```
function getInputInfo()
{
   var myString;     //declare local variables
   myString = prompt("Enter the string", "The string");
   alert("Your output: " + myString);
   outputMessage = document.getElementById("mymessage");
   outputMessage.textContent="Your output: " + myString;
}
```

Defining Functions

- **Functions** are groups of statements that you can execute as a single unit

- **Function definitions** are the lines of code that make up a function

- The syntax for *defining* a function is:

```
function nameOfFunction(parameters) {
    function statements;
}
```

```
function getInputInfo()
{
    ….
}
```

"Formal" parameters in function definition

Not all functions have parameters

Defining Functions (continued)

- **Function statements** do the actual work of the function and must be contained within the function braces

```
function showName(name1,name2) {
    alert (name1+name2);
}
```
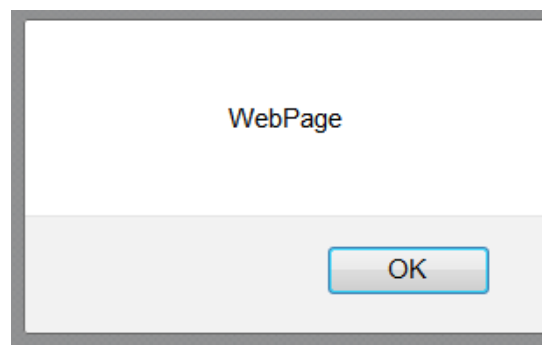
+ concatenates string values

Calling Functions

- Function must be **called** in order to be executed

- Use the function name with () to execute the function

```
showName("Web","Page");
```

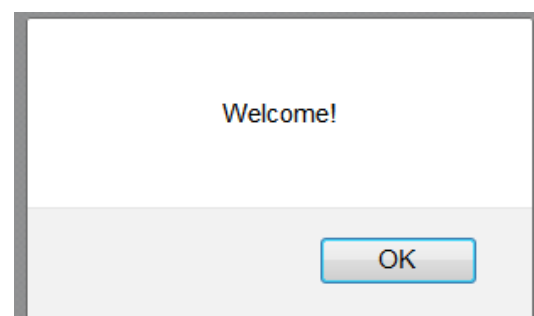> "Actual" parameters passed in call to function. Can be Literals or variables

WebPage

OK

Calling Functions

() is required even if the function has no parameters

```
function printWelcome() {
    alert ("Welcome!");
}
printWelcome();
```

Welcome!

OK

Returning Values

- A **return statement** is a statement that returns a value to the statement that called the function

- A function does not necessarily have to return a value

```
function averageNumbers(a, b, c) {
    var sum, result;
    sum = a + b + c;
    result = sum / 3;
    return result;
}
```

+ adds numbers

Returning Values (continued)

- Functions that return values work like an expression, usually with an assignment operator.

    For example

    Returned value assigned to x

    ```
    x = averageNumbers(3, 4, 5);
    ```

    To display the result, the alert function may be used

    ```
    alert(x);
    ```

    Or

    ```
    alert(averageNumbers(3, 4, 5));
    ```

Our example – return values

- Which functions calls have return values?

```
function getInputInfo()
{
    var myString;      //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}
```

Where are the function calls?

```
function getInputInfo() {
    var myString;      //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}

function init() {
    var clickme = document.getElementById("clickme");
    clickme.onclick = getInputInfo;
}

window.onload = init;
```

?

?

Event-driven: the function call is generate by the event-handler in the browser in response the user action

What's Next?
 - JavaScript Part 2

 - more about DOM