SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10011
Creating Web Applications

Lecture 5 – JavaScript (Part 1)

Acknowledgement: The contents in this document was adopted from learning material prepared by Alan Colman (Convener, SUT)

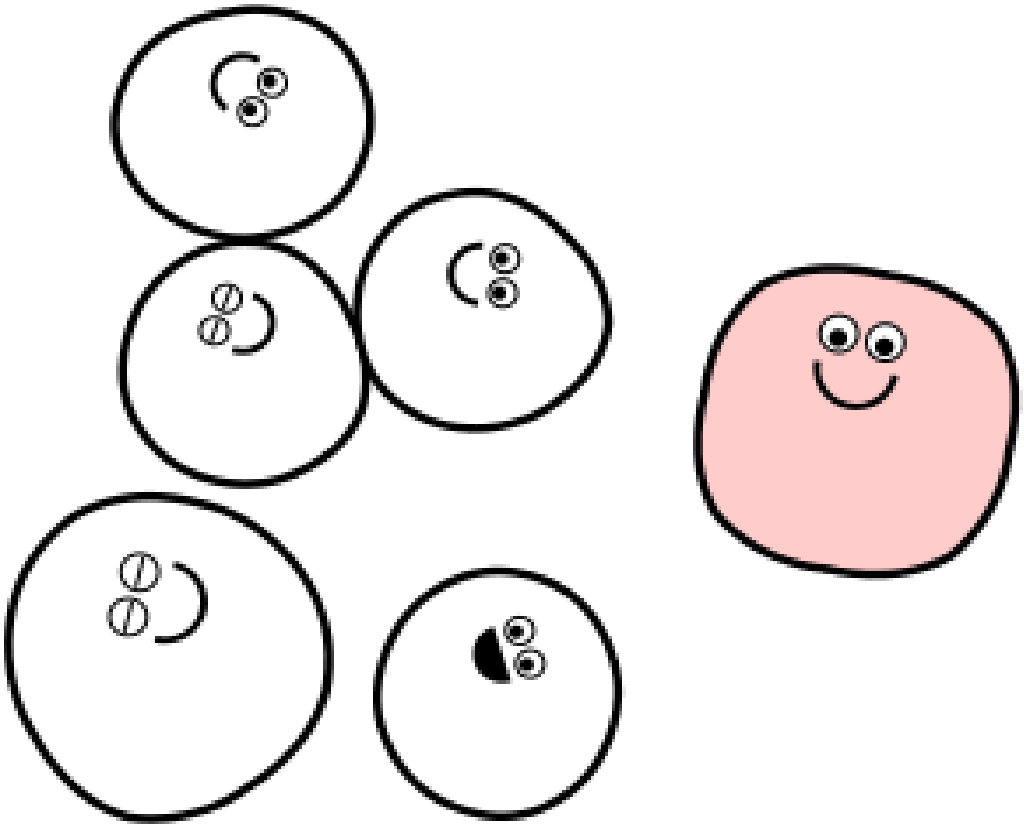
Contents

JavaScript

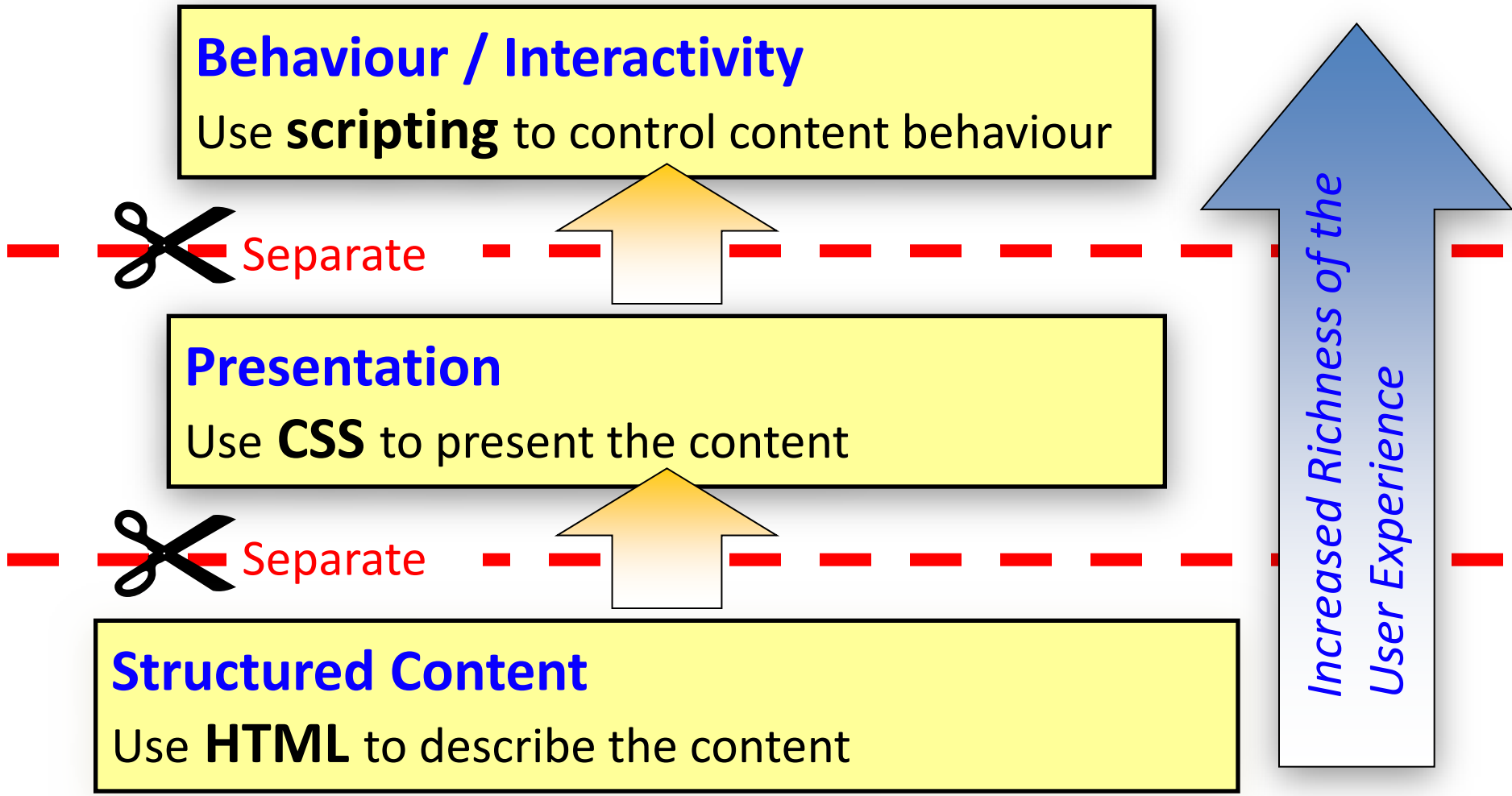
- Introduction - JavaScript
- Comments
- Statement, blocks and naming rules
- Variables
- Data types
- Operators and expressions
- Functions and scope

JavaScript = Behaviour

- eg. JavaScript HMTL +SVG
  http://www.blobsallad.se/



Separate behaviour from content and presentation



Work from the bottom up !

A very simple example - 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

Identify parts of the HTML that will respond to JS

JavaScript: Adding to HTML

- Embedded in HTML element

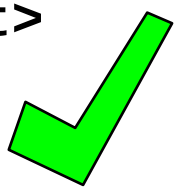
```
<input type="button" value="Back"onclick="clickme( )">
```

- Embedded within HTML header

```
<script>   …   </script>
```

- Include reference to an external file

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



CWA mandated approach:

- Separates behaviour from content
- Can be cached by user's web browser, downloaded once only if needed by multiple webpages.

A very simple example - 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;
}

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

window.onload = init;
```

Functions to handle the events

DOM object in the HTML page

JS variable referencing the DOM object

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

object

event

function name

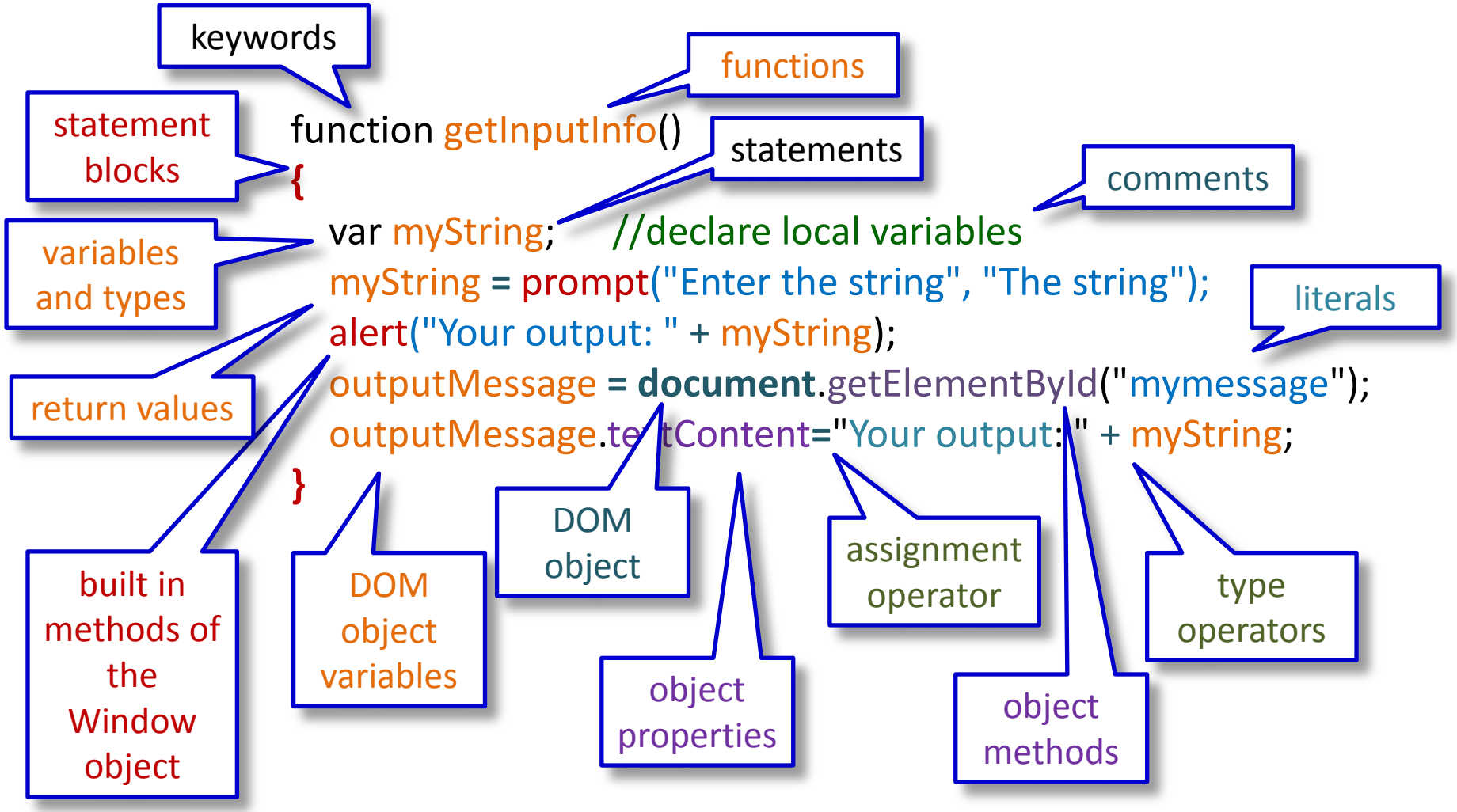
In Short…

1. HTML file
   i. Create reference to JavaScript file from your 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.load = init; *or* button.click = do_something;
   ii. Write the functions to handle the events
   ```
   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



Contents

JavaScript

- Introduction - JavaScript
- 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
   alert("Your output
   outputMessage =
   outputMessage.te
}
```

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

Contents

JavaScript

- Introduction - JavaScript
- Comments
- Statement, blocks and naming rules
- Variables
- Data types
- Operators and expressions
- Functions and scope

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

Contents

JavaScript

- Introduction - JavaScript
- Comments
- Statement, blocks and naming rules
- Variables
- Data types
- Operators and expressions
- Functions and scope

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
- By convention:
  - variables start with lower case letter e.g. var **myString**="Hello";
  - constants are upper case e.g. const **MAX_LENGTH** = 7;
- Can include any letter of the alphabet, digits 0-9, and underscore
- Cannot include spaces, or punctuation characters such as comma, full stop

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);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}
```

Variable local to the function

Functions are also variables in JavaScript!

This is a global variable

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 any 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

Local Variable Declaration - **let**

- **let** variable declaration is similar to a var but the scope is the block { } in which it is declared rather than the function
- **let** keyword introduced in ECMAScript 6
- Only introduced in 2015 so may not be supported by browser

ECMAScript is the standard JavaScript in browsers are *supposed* to follow

var vs let

```
function simpleLoop(){
      /* i is visible here*/
       for(var i=0;i<10;i++){
             alert(i);  // i is available here
       }
        alert(i);
       /* i is available as well */
}
function simpleLoop(){
      /* i is NOT visible here*/
       for(let i=0;i<10;i++){
             alert(i);
             // i is only visible in this scope {}
       }
        alert(i);
 }
```

i is *undefined* here

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) { ... }

- You can change the global variable's value at any point *from anywhere* ☹

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

Global variables should be avoided if possible.

Lifetime of a global variable: until the page closes

This can lead to unintended "side effects"

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 x not declared with var

"use strict"
"use strict"

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

Contents

JavaScript

- Introduction - JavaScript
- Comments
- Statement, blocks and naming rules
- Variables
- Data types
- Operators and expressions
- Functions and scope

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

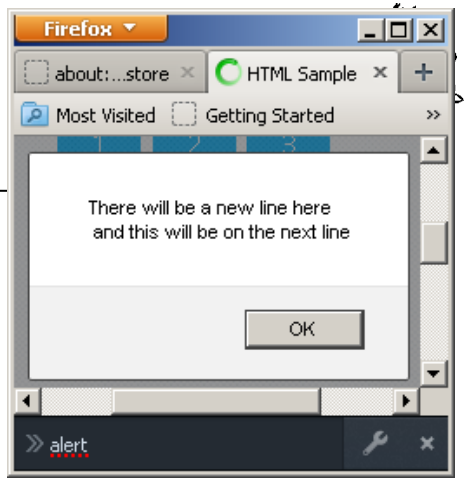
Primitive Data Types

- String
- Number
- Boolean
- Null
- Undefined

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

Number

- **Integers** can be positive, 0, or negative
- **Integers** can be expressed in
  - decimal (base 10)
  - hexadecimal (base 16)
  - octal (base 8)

Number

- **Decimal**
  - integer literal consists of a sequence of digits without a leading 0 (zero)

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

    example: 0377 = 255_{10}
  - Octal integers can include only the digits 0-7.
- **Hexadecimal**
  - A leading 0x (or 0X) indicates hexadecimal.

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

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 "-")

$1.025e3 = 1.025 \times 10^{3} = 1025.0$

$130e\text{-}3 = 130 \times 10^{-3} = 0.130$

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

Contents

JavaScript

- Introduction - JavaScript
- Comments
- Statement, blocks and naming rules
- Variables
- Data types
- Operators and expressions
- Functions and scope

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
- A **unary operator** requires a ***single*** operand either before or after the operator

String Operator

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

```
"Your output: " + myString;
```

Operator	Name	Description
+	Concatenation	Joins two operands

- Results to

“This text” + “That Text”

“This TextThat Text”

No blank space will be inserted

Arithmetic Operators (Binary)

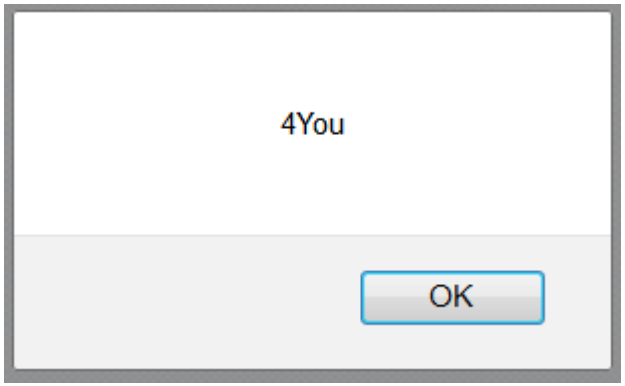
- **Arithmetic operators** are used to perform mathematical calculations

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

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;
```

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

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 wit*h* **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

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

Contents

JavaScript

- Introduction - JavaScript
- Comments
- Statement, blocks and naming rules
- Variables
- Data types
- Operators and expressions
- Functions and scope

JavaScript - Language

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)

- A **parameter** is a variable that is used within a function
- Parameters are placed within the parentheses that follow the function name
- Functions do not have to contain parameters
- The set of curly braces (called **function braces**) contain the function statements
- Functions enables reusability of code

Defining Functions (continued)

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

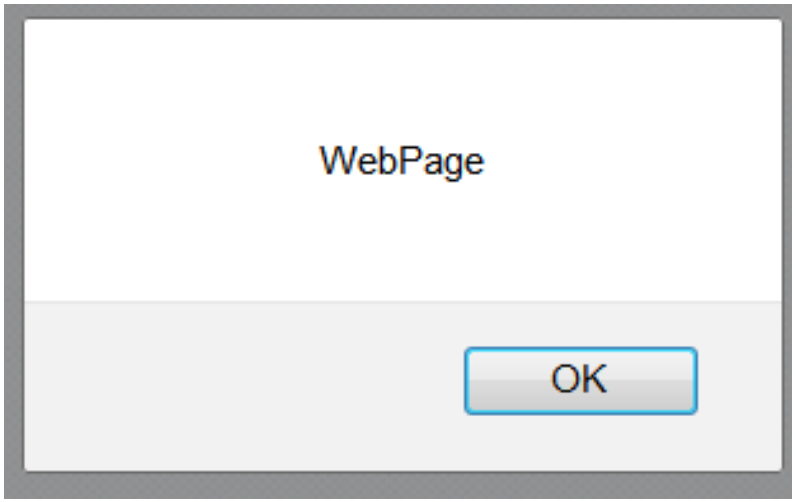
```
function showName(name1,name2) {
    alert (name1+name2);
}
```

+ 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");
```

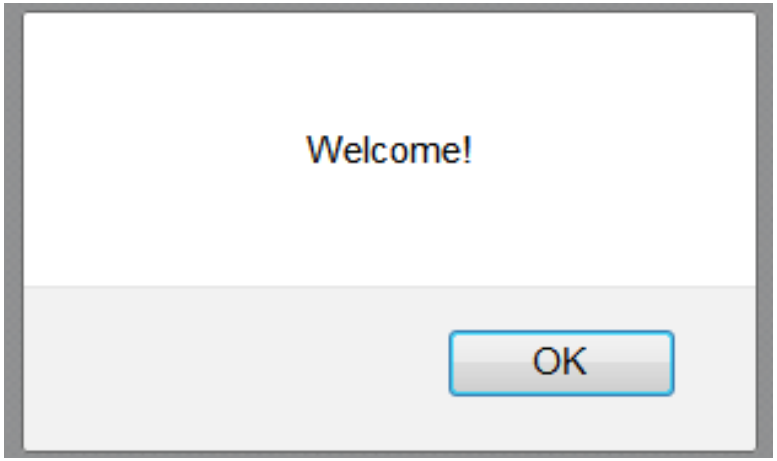


"Actual" parameters passed in call to function. Can be Literals or variables

Calling Functions

() is required even if the function has no parameters

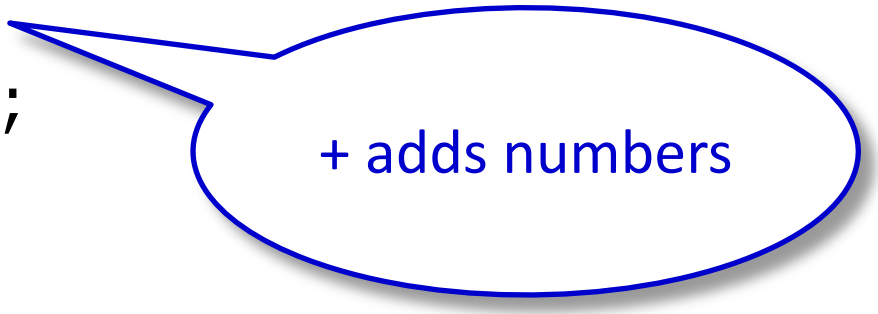
```
function printWelcome() {
    alert ("Welcome!");
}
printWelcome();
```



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;
}
```



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;
}
```

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));
```

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

JavaScript – Variable Scope

```
var myGobalVariable = "";

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;
```

Not so good ☹
Why?

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*

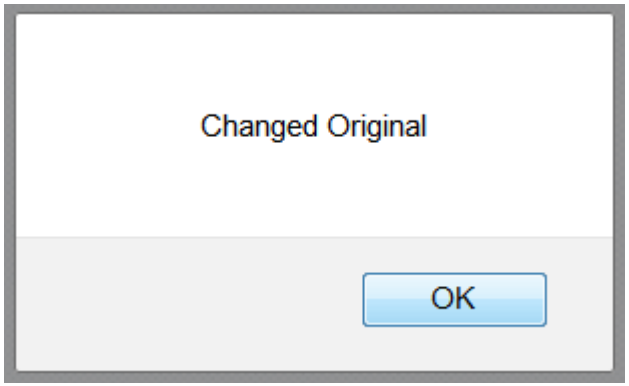
What's the output?

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

  localVariable = "Changed";
  globalVariable = "Changed";
}

globalVariable = "Original";
localVariable = "Original";
testScope();
alert (globalVariable + " " + localVariable);
```

What's the output?

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

Output



Any errors here?

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

  localVariable = "Changed";
  globalVariable = "Changed";
}

globalVariable = "Original";

testScope();
alert (globalVariable + " " + localVariable);
```

No Output

Error, as localVariable does not exist outside the function

SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10011
Creating Web Applications

What's Next?

- JavaScript Part 2
- more about DOM

Acknowledgement: The contents in this document was adopted from learning material prepared by Alan Colman (Convener, SUT)