Introduction to JavaScript

WEB DEVELOPMENT FUNDAMENTALS

Introduction

- A very brief history
- What is JavaScript?
- How to place script in a web page
  - Embedding
  - Linking
  - <noscript>
- Visual Studio
- Chrome Developer tools

2

A very brief history

- JavaScript has been with us since 1995
  - Originally designed for client based form validation
  - Three separate versions in IE, Netscape and ScriptEase
- Put forward to the ECMA as a proposed standard in 1997 as v1.1
  - Ratified in 1998 as ECMAScript
  - Implemented in browsers with various degrees of success ever since
- Implementation is made up of three parts
  - The Core (ECMAScript)
  - The DOM (Document Object Model)
  - The BOM (Browser Object Model)

3

ECMAScript

- All browsers should adhere to the ECMAScript standard
  - They do not (the Netscape IE browser wars were messy!)
  - ECMAScript standard 3 was mostly implemented
  - ECMAScript 4 was not
  - ECMAScript 5 was implemented correctly
    - As part of the HTML5 project
- Since 2015 new versions of the ECMAScript specification have been released yearly
  - i.e. ECMAScript 2018 is the 9th version of the specification

4

What can JavaScript do?

- JavaScript is a client side scripting language
  - It dynamically executes in the browser
- JavaScript gives HTML designers a programming tool
  - JavaScript can react to events reacting to the page or user
- With JavaScript, it is more a question of what it cannot do
  - It can read and change the content of HTML elements
  - Even create new elements
  - It can be used to validate form input
  - Can be used to detect the visitor's browser
  - Can be used to create cookies
  - Can asynchronously request data from a server

5

Some key Javascipt concepts

- JavaScript is a loosely typed dynamic programming language
  - Variables are not given a static type
  - They can change their type
  - So understanding and maintaining type matters
- JavaScript is a case sensitive programming language
  - Everything in JavaScript is case sensitive
  - There is a best practice approach as we will discover
- Code termination is optional
  - JavaScript uses a semi colon to terminate a line of code
  - Technically, this is optional but it causes serious headaches

6

Adding script to HTML – embedding

- You can either place JavaScript on a page inline
  - The closing tag is mandatory
  - Most browsers will assume JavaScript if not stated
    - It is best practice to use it, however

Inline JavaScript

```
<script type="text/JavaScript">
        … script goes here …
</script>
```

- The script can be placed in either the head or body section
  - It is executed as soon as the browser renders the script block
  - Current practice often places it just before the closing body tag

7

Adding script to HTML – linking

- You can also place JavaScript in a separate file and link to it
  - Useful if script is going to be used on multiple pages
  - Requires an additional request to the server
  - The requested file is cached by the browser
  - Preferred approach to working with script

External JavaScript

```
<script type="text/JavaScript"  src="../myScript.js">
</script>
```

- When linking to external script, there are a few things to remember
  - There must be a closing </script> tag
  - There must be a src attribute:
    - This can be relative or absolute
  - No JavaScript can occur within the script tag

8

The <noscript> element

- Client-side scripting may not be available
  - The browser may not understand or have disabled client scripting
- The <noscript> element will only render its content if
  - The browser does not understand <script>
  - The client has disabled script

The <noscript> element

```
<noscript>
        If you see this, you do not have JavaScript enabled.
</noscript>
```

9

Comments

- Commenting code is an essential part of programming
  - Single line comment

```
index = 3; // From here to end of line is a comment
```

  - Multi-line comment

```
/*
  Everything inside these delimiters is treated as
  a comment and ignored by the interpreter
*/
```

10

Review

- What is JavaScript?
  - A client side scripting language
- What is JavaScript for?
  - JavaScript is a client
- How do we use JavaScript?
  - Linked or embedded

11

Types and operators

WEB DEVELOPMENT FUNDAMENTALS

Introduction

- In this module, you will learn to
  - Declare variables
  - Understand types
    - Primitive types
      - Strings, Numbers, Booleans, Undefined, Nulls
    - Reference types
  - Operators
    - Using operators
    - Type conversion

13

Declaring variables

- Declaring variables
  - Implicit
  - Explicit
  - With assignment
- Variable names
  - Start with a letter , "_" or "$"
  - May also include digits
  - Are case sensitive
  - Cannot use reserved keywords
  - E.g. int, else, case

- Best practice is to use camelCase for variable names

```
x = 10;
var y;
var z = 10;
```

14

JavaScript types

- Dynamically typed
  - Data types not declared and not known until runtime
  - Variable types can mutate
- Interpreted
  - Stored as text
  - Interpreted into machine instructions and stored in memory as the program runs
- Primitive data types
  - Number, String, Undefined, Booleans (true/false)
- Reference types
  - Object

15

Value and reference types

- JavaScript can hold two types
- Primitive
  - Primitive values are pieces of data stored on the stack
  - Their value is stored in the location the variable accesses
  - They have a fixed length
  - Quick to look up
- Or reference
  - Reference values are objects stored on the heap
  - The value stored in the variable is a pointer
  - The heap allows the variable's size to grow:
    - Without effecting performance

16

The typeof operator

- Calling typeof on a variable or value returns one of the following
  - number
  - boolean
  - String
  - Undefined
    - If the variable has not be initialised
  - Object
    - If a null or a reference type

The typeof operator

```
var typeTest = "string value";
alert(typeof  typeTest)      //outputs string
alert(typeof 95)            //outputs number
```

17

The undefined type

- A variable that has been declared but not initialised is undefined

The undefined type

```
var age;
 console.log(age);  //returns undefined
```

- A variable that has not been declared will also be undefined
  - The typeof operator does not distinguish between the two

The undefined type

```
//var boom;
console.log(boom);  //returns undefined
```

  - It is a good idea to initialise variables when you declare them

18

Null is not undefined

- Null and undefined are different concepts in JavaScript
  - Undefined variables have never been initialised
  - Null is an explicit keyword that tells the runtime it is 'empty'

The null type

```
var userID = null;
console.log(userID);  //returns null
```

- There is a foobar to be aware of with null:
  - Undefined is the value of an uninitialised variable
  - Null is the value used to represent an object that does not exist

Null and undefined

```
var userID = null;
var password;        //implicitly undefined
console.log(userID  == password);  //returns true
```

19

The Boolean type

- Boolean can hold two values – true and false
- These are reserved words in the language:

The Boolean type

```
var loggedOn = false;
 console.log(loggedOn);  //returns false
```

- When evaluated against numbers, you can run into issues
  - false is the evaluated as 0
  - true can be evaluated to 1
  - Vice-versa as well

20

The Number type

- Always stored as 64-bit values
  - Division between any two numbers can produce fractional parts
  - If bitwise operations are performed, the 64-bit value is rounded to a 32-bit value first
  - There are a number of special values

Constant	Definition
`Number.Nan` or `Nan`	Not a number
`Number.Infinity` or `Infinity`	Greatest possible value (but no numeric value)
`Number.POSITIVE_INFINITY`	Positive infinity
`Number.NEGATIVE_INFINITY`	Negative infinity
`Number.MAX_VALUE`	Largest possible number represented in the 64-bits
`Number.MIN_VALUE`	Smallest possible number represented in the 64-bits

21

The String type

- Immutable series of zero or more Unicode characters
  - Modification produces a new string
  - Can use single (') or double quotes (")
  - Primitive and not a reference type
- String concatenation is expensive
- Forward slash (/) used for escaping special characters

Escape	Output
`\'`	'
`\"`	"
`\\`	\
`\b`	Backspace
`\t`	Tab
`\n`	Newline
`\r`	Carriage return
`\f`	Form feed
`\ddd`	Octal sequence
`\xdd`	2-digit hex sequence
`\udddd`	Unicode sequence (4-hex digits)

22

String functions

- The String type has string manipulation methods including

Method	Description
`indexOf()`	Returns the first occurrence of a character in a string
`charAt()`	Returns the character at the specified index
`toUpperCase()`	Converts a string to uppercase letters

- Method is called against the string variable

The indexOf() method

```
var str = "Hello world, welcome to the universe.";

var n = str.indexOf("welcome");
```

  - Where n will be a number with a value of 13

23

Operators – assignment and arithmetic

- Operators allow us to work with types in tasks, such as
  - Mathematic operations
  - Comparisons
- They include
  - Assignment:

Assignment	=
Shorthand Assignment	`+=, -=, *=, /=, %=`

  - Arithmetic:

Arithmetic	
Addition, subtraction	`+ , -`
Multiplication, division, modulus	`* , / , %`
Negation	`-`
Increment, decrement	`++ , --`

24

Operators – Relational and Boolean

- Relational and Boolean operators evaluate to true or false
  - Relational:

Relational	
Less than, greater than	< , >
Less than or equal, greater than or equal	<= , >=
Equals, not equals	== , !=

  - Boolean:

Boolean	
AND, OR	&& , ||
NOT	!

- The Boolean logical operators short-circuit
  - Operands of && , || evaluated strictly left to right and are only evaluated as far as necessary

25

Type checking

- JavaScript is a loosely-typed language

```
var a = 2;
var b = "two";
var c = "2";
alert(typeof a);// alerts "number"
alert(typeof b);// alerts "string"
alert(typeof c);// alerts "string"
```

- JavaScript types can mutate and have unexpected results

```
alert(a * a);// alerts 4
alert(a + b);// alerts 2two
alert(a * c);// alerts 4
alert(typeof (a * a));// alerts "number"
alert(typeof (a + b));// alerts "string"
alert(typeof (a * c));// alerts "number"
```

26

Type conversion

- Implicit conversion is risky – better to safely convert
- You can also use explicit conversion
  - **`eval( )`** evaluates a string expression and returns a result
  - **`parseInt( )`** parses a string and returns an integer number
  - **`parseFloat( )`** parses a string, returns a floating-point number

Type conversion with parseInt()

```
var s = "5";
  var i = 5;
  var total = i + parseInt(s);  //returns 10 not 55
```

- You can also check if a value is a number using `isNaN()`

The isNaN() global function

```
isNaN(s); // returns true
!isNan(i); //returns true
```

27

Exercise 12a

- Exploring operators and types
  - Arithmetic types
  - Relational operators
  - Assignment operations
  - Type mismatching and conversion

28

Review

- Primitive variables
  - Value types
- Understand types
  - There are six simple types
  - Types mutate
- Operators
  - You use operators to manipulate type

29

Flow of control

WEB DEVELOPMENT FUNDAMENTALS

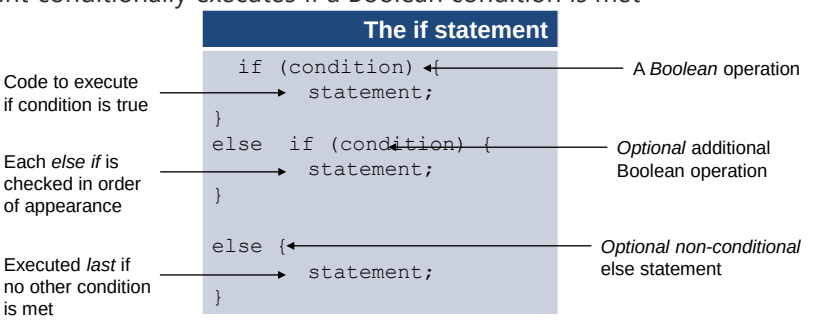
Introduction

- Understanding flow of control statements
  - The if statement
  - The switch statement
- Understanding loops
  - The while and do while loops
  - for loops

31

If statements

- The if statement conditionally executes if a Boolean condition is met



- The if statement has optional else if and else branches
  - Additional Boolean conditions executed in order
    - So specificity matters
  - Often best to use many single if statements over a mega if statement

32

The ternary if

- An alternative to the standard if statement exists in JavaScript
  - Known as a ternary operator

The if approach

```
var now = new Date();
var greeting = "Good";

if (now.getHours() > 17){
   greeting += " evening.";
}
else{
   greeting += " day.";
}
```

The ternary operator approach

```
var now = new Date();
var greeting = "Good" + ((now.getHours() > 17) ? " evening." : " day.");
```

33

The switch statement

- switch statement
  - Control passes to the case label that matches the expression
  - Carries on until hits a break statement
  - If no case labels match, control passes to the default label
    - If there is no default label, entire switch statement is skipped:

The switch statement

```
switch (expression)
{
    case label:
        statement;
        break;
    case label:
        statement;
        break;
    default:
        statement;
        break;
}
```

34

The while loop

- Loops allow a set of statements to be run more than once:
  - Either for a fixed number of iterations or until a condition is met
- The while loop has two varieties the while and do while
  - The while checks before it executes:

The while statement

```
while (condition){
        statement;
}
```

  - The do while always runs at least once

The do while statement

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

Note the semi-colon

35

The for loop

- The for loop utilises a counter until a condition is met

The for loop

```
for ([initial-expression]; [condition]; [loop-expression]) {
        statement;
}
```

- In the below example, i is incremented by 1 after each iteration
  - The loop expression can be any arithmetic operation

A for loop in action

```
 for (var i = 0; i < 10; i++) {
            i += i;
            console.log(i);
}
```

36

Review

- Flow of control and loops are the basis of programming
  - Along with operators
- If statements allow conditional logic
- Loops allow reuse of code without repetition

37

Exercise 12b

- Flow of Control

38