INFO 151
Web Systems and Services

Week 4 (T1)

Dr Philip Moore

Dr Zhili Zhao

Course Overview

Weeks 1 – 3

- Introduction to Web Systems and Services
- Creating Web-Pages and Web-Sites with a Markup Language
- Introductory HTML 4 and HTML 5 with CSS

Weeks 4 – 6

- Client-Side Web Programming
- Introductory JavaScript

Weeks 7 – 9

- Server-side Programming
- Introductory PHP
- Introduction to Database, SQL, and MySQL

Client-Side Web Programming

Scripting with JavaScript

Sources of Resources

- The sources of information and resources for JavaScript may be found at the:
  - w3schools.com web-site: (url: https://www.quanzhanketang.com/)
- The w3schools.com web-site available to you has limited resources for PHP
  - For information on PHP see the recommended course text book:
    - Sams Teach Yourself PHP, MySQL & JavaScript All-in-One Sixth Edition

Overview

- In this tutorial we will introduce
  - Scripting environments with an overview of client-side and server-side scripting
  - An introduction to scripting languages
  - The JavaScript, JavaScript programming, and JavaScript syntax
  - JavaScript expressions, operators, operands, and operator precedence
  - JavaScript identifiers, datatypes, strings, and variables
  - String escaping with string and variable concatenation

Scripting Environments

- Scripting is located and executed using:

- Client-side environments
  - Implemented using a local web-browser (a thin client) using a *localhost*

- Server-side environments
  - A *remote host* using a web-server accessed over the Internet

Client-Side Scripting

- Client-side environments
  - In a client-side environment scripts are embedded in HTML
  - Scripts are run in a web browser (a thin-client) located on an end-users computer
  - The source code is transferred from the web server to the users computer over the internet and run directly in the browser
  - JavaScript is used with HTML to create *stateless* websites
- The JavaScript must *enabled* on the client computer
  - Where a user disables JavaScript (for security) a warning message (a dialogue) appears alerting the user when script is attempting to run

Served-side Scripting

- Server-side environments
  - In a *server-side* environment *scripts* are located on a web-server
  - The web-server responds to a user (client) requests to a web-site
  - In server-side scripting scripts are located and run within the web-server
  - A primary function of server-side scripting is to enable dynamic (*stateful*) web-sites as opposed to static (*stateless*) web sites
- In practice web-sites generally use both client-side and server-side scripting in a combined web-system

Scripting Languages

Scripting Languages

- A *scripting language* is a programming language that supports *scripts*
  - *Scripts* are (generally) small programs written to function in *'real-time'* environments
  - *Scripts* are (generally) written to control and automate the execution of tasks and events
- *Scripting* languages (such as *JavaScript* and *php*) are (generally) *interpreted*
- High-level programming languages (such as 'Java' and 'C') are *compiled*
- A *scripting language* enables (generally) small programs to be combined into more complex programs

Scripting Languages

- JavaScript (client-side) and PHP (server side) are frequently used together
  - PHP can naturally work with MySQL whereas JavaScript can not
- Environments to which a *scripting* can be applied include:
  - Software applications
  - Web-systems (web-pages and web-sites) running in a web-browser
  - Shell programming (in a Unix – or Linux) operating system
  - Embedded systems
  - Online games

JavaScript

JavaScript Overview

- JavaScript is an important web-systems language
  - The association with web-browsers mane it the most popular web-programming language in the world
- JavaScript is a lightweight *object-oriented* scripting language which can be embedded within HTML (web-pages)
  - Object orientation will be considered later in the course
- The syntax is modelled on Java syntax which in turn is modelled on C syntax and C++ (an object-oriented C) syntax
  - However, JavaScript is a lambda language and it has much in common with Lisp and Scheme.

JavaScript

- JavaScript is generally used as client-side scripting language
- It is mainly used to enable *validation* and other simple tasks
- JavaScript has limitations:
  - JavaScript can only connect to a database (MySQL) using ‘bridging’
  - JavaScript provides limited interaction(s) with computer systems and resources (for security reasons)
  - Java script is useful for form interactions
  - JavaScript is useful in the control of the display of data to the user

JavaScript Programming

Computer Programming

- When writing computer program code there are two considerations:
  1. The code must be machine-readable
  2. The code will be human-readable (to understand the code)
- JavaScript ignores ‘whitespace’ between tokens
- In the processing of JavaScript the interpreter is not concerned with the readability of the code
- Spaces, tabs, and newlines may be used by the programmer to improve the human-readability of the program code.

JavaScript Syntax

- JavaScript is *case-sensitive*
  - All *variables*, *function names*, and other *identifiers* must be typed in lowercase without capitalization
  - The use of capitals for *GLOBAL* variables is controversial but it is a common practice (in JavaScript and other programming languages)
  - Capitalization helps to identify *GLOBAL* variables
    - For example: a constant such as ‘PI’ may be written in UPPERCASE
  - The difference between *local* and *GLOBAL* variables lies in their scope
    - We will consider variable scope later in this course

Statements

- A `statement` is a compilation unit which contains a set of executable statements
  - In web-browsers each `<script>…<\script>` tag delivers a compilation unit that is interpreted and immediately executed
  - JavaScript processes all the statements in a common global `namespace`
  - When use in a function the var statement defines the function `private` variables and their value
- Statements can implement all the programming operations:
  - Sequential processes / Selection operations / Iterative operations
  - These operations will be introduced later in this course

JavaScript Statements

- JavaScript statements are terminated by a semicolon (**;**)
- However:
  - When a statement is followed by a newline (the carriage return)
  - The terminating semicolon (**;**) may be omitted
    - This restricts where the programmer may legally break lines in JavaScript
    - A line of JavaScript code may not be broken over two lines if the first line is a legal JavaScript statement
- It is good programming practice to insert a semicolon (**;**) at the end of complete statements to improve human-readability

JavaScript Syntax

- The documentation of computer program code using *comments* (ignored by the JavaScript interpreter at runtime) is very important
- To insert *comments* into JavaScript there are 2 methods:

```
/*
 * This is a block multi-line Java, C, and C++ style comment
 * With a second line
 */
/* Another comment*/
// This is a single line Java, C, and C++ style comment
```

An HTML / JavaScript Template File

```
<!DOCTYPE html>
<!- comment -->
<html>
<head>
      <title></title>
</head>
   <body>
      <!- enter HTML here -->
      <script>
            //enter JavaScript program code here
      </script>
   </body>
</html>
```

An HTML-JavaScript File

```
<!DOCTYPE html>
<!--HTML_JS_1-->
<html>
<head>
      <title>HTML_JS_1</title>
</head>
   <body>
      <h2>This text is created under HTML</h2>
      <script>
            document.write('This text is created under JavaScript’);
      </script>
   </body>
</html>
```

HTML_JS_1

localhost:8383/HTML_JS_1/index.html

This text is created under HTML

This text is created under JavaScript

Expressions and Operators

Operator and Operand

- In computer programming there are two terms used in relation to data processing:
  - The *operator*
  - The *operand*
- The *operator*
  - Is a method which operates on the data: examples include *addition* (+), *comparison* (<), logical operators such as (*AND* (&&) / *OR* (||))
- The *operand*
  - Is the data being operated on such as variables or strings

Operator and Operand Example

- Consider the following arithmetic example: (7 + 13 = 20)
- In this example
  - *Operator*
    - The *operator* is the symbol (+) for the operation called addition
  - *Operand*
    - The *operand* (7) is one of the inputs (quantities) followed by the 'addition' *operator*
    - The *operand* (13) is the other input necessary for the operation

Operators

Operator	Syntax	Example	Definition
addition	+	x + y	Sum of x and y
subtraction	-	x - y	Difference of x and y
multiplication	*	x * y	Product of x and y
division	/	x / y	Quotient of x and y
modulo	%	x % y	Remainder of x / y
exponent	**	x ** y	x to the y power
increment	++	x++	x plus one
decrement	--	x--	x minus one

Expressions and Operators

- A JavaScript expression is formed by combining values which can be any combination of the following:
  - Literals / variables / object properties / array elements / function invocations
  - Parenthesis **(** … **)** can be used to group sub-expressions
    - This can be used to change the default evaluation (processing) order this is important because: while the JavaScript syntax may be correct the logic may be incorrect resulting in the wrong answer
- In JavaScript (in all programming languages) there is an ***operator precedence***

Operator Precedence Examples

- Arithmetic computation is processed from left to right
  - `100 + 50 * 3 = 250`
    - multiplication has precedence over addition
  - `(100 + 50) * 3 = 450`
    - multiplication has precedence over addition (but) parenthesis has precedence over multiplication
  - `100 + 50 / 3 = 116.66666666666667`
    - division has precedence over addition
  - `(100 + 50) / 3 = 50`
    - division has precedence over addition (but) parenthesis has precedence over division
  - `100 + 50 - 3 = 147`

Precedence_Test - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- Precedence_Test
  - Site Root
    - index.html
  - Unit Tests
- Precedence test

Navigator

- JavaScript
  - n1 : Number
  - n2 : Number
  - n3 : Number
  - r1 : Number
  - r2 : Number
  - r3 : Number
  - r4 : Number
  - r5 : Number
  - r6 : Number
  - r7 : Number
  - s1 : String
  - s2 : String
- HTML
  - html
    - head
      - title
      - meta
      - meta
    - body
      - div
        - script

PrecedenceTest.java | index.html

Source | History

```
12      <div>
13      <script>
14          var n1 = 100;
15          var n2 = 50;
16          var n3 = 3;
17          var r1 = n1 + n2 * n3;
18          var r2 = (n1 + n2) * n3;
19          var r3 = n1 + n2 / n3;
20          var r4 = (n1 + n2) / n3;
21          var r5 = n1 + n2 - n3;
22          var r6 = (n1 + n2) - n3;
23          var r7 = n1 + (n2 - n3);
24          var s1 = "This is a test of operator precedence";
25          var s2 = "The results confirm resuly of the precedence";
26          document.write(s1 + "<br>" + s2 + "<br>");
27          document.write("r1 = " + r1 + "<br>");
28          document.write("r2 = " + r2 + "<br>");
29          document.write("r3 = " + r3 + "<br>");
30          document.write("r4 = " + r4 + "<br>");
31          document.write("r5 = " + r5 + "<br>");
32          document.write("r6 = " + r6 + "<br>");
33          document.write("r7 = " + r7 + "<br>");
34      </script>
```

Variables | Call Stack | Breakpoints | Output - Browser Log

Name	Type	Value
<Enter new watch>		

18:1 INS

NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- Precedence_Test
  - Site Root
    - index.html
  - Unit Tests
- Precedence test

Navigator | Browser DOM

http://localhost:8383/Precedence_Te...

- html
  - head
    - title
    - meta
    - meta
  - body
    - div
      - script
      - br
      - br
      - br
      - br
      - br
      - br
      - br
      - br
      - br

PrecedenceTest.java | index.html

Source | History

```
10      </head>
11      <body>
12          <div>
13          <script>
14              var n1 = 100;
15              var n2 = 50;
16              var n3 = 3;
```

Variables | Call Stack | Breakpoints | Output - Browser Log

Name	Type	Value
<Enter new watch>		

Operator Precedence

http://localhost:8383/Precedence_Test/index.html

100%

This is a test of operator precedence
The results confirm resuly of the precedence
r1 = 250
r2 = 450
r3 = 116.66666666666667
r4 = 50
r5 = 147
r6 = 147
r7 = 147

CSS Styles

<No Element Selected>

No Rule Selected

<No Properties>

18:1 INS

Precedence test - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Function_Example
- Precedence_Test
- Precedence test
  - Source Packages
    - precedence.test
      - PrecedenceTest.java
  - Test Packages
  - Libraries
  - Test Libraries
- Return_Function

main - Navigator

Members <empty>

- PrecedenceTest
  - main(String[] args)

PrecedenceTest.java

Source History

```java
package precedence.test;
/**
 * @author philip
 */
public class PrecedenceTest {

    public static void main(String[] args) {
        System.out.println("Examples of operator precedence");
        int n1 = 100;
        int n2 = 50;
        int n3 = 3;
        int r1 = n1 + n2 * n3;
        int r2 = (n1 + n2) * n3;
        int r3 = n1 + n2 / n3;
        int r4 = (n1 + n2) / n3;
        int r5 = n1 + n2 - n3;
        int r6 = (n1 + n2) - n3;
        int r7 = n1 + (n2 - n3);
        String s1 = "This is a test of operator precedence";
        String s2 = "The results confirm the operator precedence";
        System.out.println(s1 + "\n" + s2);
        System.out.println("r1 = " + r1);
        System.out.println("r2 = " + r2);
        System.out.println("r3 = " + r3);
        System.out.print("r4 = " + r4 + "\n");
        System.out.println("r5 = " + r5);
        System.out.println("r6 = " + r6);
        System.out.println("r7 = " + r7);
    }
}
```

Variables | Call Stack | Breakpoints

21:25 INS

Precedence test - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Function_Example
- Precedence_Test
- Precedence test
  - Source Packages
    - precedence.test
      - PrecedenceTest.java
  - Test Packages
  - Libraries
  - Test Libraries
- Return_Function

main - Navigator

Members <empty>

- PrecedenceTest
  - main(String[] args)

PrecedenceTest.java

Source History

```
 * @author philip
 */
public class PrecedenceTest {

    public static void main(String[] args) {
        System.out.println("Examples of operator precedence");
        int n1 = 100;
        int n2 = 50;
        int n3 = 3;
        int r1 = n1 + n2 * n3;
        int r2 = (n1 + n2) * n3;
        int r3 = n1 + n2 / n3;
        int r4 = (n1 + n2) / n3;
        int r5 = n1 + n2 - n3;
        int r6 = (n1 + n2) - n3;
```

Output

Browser Log | Precedence test (run)

```
run:
Examples of operator precedence
This is a test of operator precedence
The results confirm the operator precedence
r1 = 250
r2 = 450
r3 = 116
r4 = 50
r5 = 147
r6 = 147
r7 = 147
BUILD SUCCESSFUL (total time: 0 seconds)
```

Output | Variables | Call Stack | Breakpoints

22:36 INS

JavaScript Syntax

Naming Rules and Conventions

- In JavaScript there are *reserved keywords* which have a special meaning to the interpreter:
  - The *keywords* must not be used as *identifiers*
  - Trying to use a keyword in JavaScript program code will throw an *error*
- Additionally:
  - Creating variables that have the same name as *GLOBAL* properties and methods should be avoided
- HTML Event Handlers:
  - Using the name of all HTML event handlers in JavaScript program code should be avoided

Naming Rules and Conventions

- In JavaScript there are types of reserved word (keyword)
  - JavaScript *Reserved* Words
  - *Removed* Reserved Words
  - JavaScript *Objects*
  - JavaScript *Properties*
  - JavaScript *Methods*
  - Java *Reserved* Words
  - Other *Reserved* Words

Reserved Keywords

- Keywords (or reserved words) cannot be used for
  - Variables
  - Labels
  - function names
  - Objects
  - built-in objects
  - Properties
  - methods
- A list of keywords (and removed keywords) may be found in the course resources

Naming Rules and Conventions

- We have seen the use of JavaScript keywords must not be used as identifiers
- using a keyword in JavaScript program code will throw an error
- There are two types of variable: a *Global* variable and a *local* variable
  - Creating variables that have the same name as *Global* variables, properties, and methods should be avoided
  - We will consider variables later in more detail later the course

Variables

- In computer programming the aim is to process and manipulate data to achieve a desired result
- A computer program uses variables to hold data
- We have introduced *local* and *GLOBAL* variables
  - In this part of the course we are focusing on *local* variables
  - In later sessions we will introduce *local* variable with local function *scope*
- In JavaScript variables are *untyped*
  - This means that a variable can contain values of any datatype
  - Java has variables that are *strongly typed*

JavaScript Variables

- In JavaScript global variables are part of a *global object*
  - This is a good program design feature as it simplifies the code writing
  - It is also a potentially bad program design feature as variables used in functions can be modified from outside the function
  - As we shall see later, we can create *static* variables within functions which have *local* scope
- In JavaScript variables are *untyped*
  - This means that a variable can contain values of any datatype
    - A numerical (a = 1) can be changed to (a = “a name”) which may be a problem
  - Java, C, C++ has variables that are *strongly typed*

Identifiers

Identifiers

- Identifiers
  - *Variables*, *functions*, and *label names* are JavaScript *identifiers*
  - Identifiers are composed od any number of letters and digits and $ characters
- The first character of an identifier must not be a digit
- The following are legal identifiers (variables):
  - `v`
  - `my_variable_name` //no spaces allowed use underscore (_)
  - `V13`
  - `$str`

Datatypes

- JavaScript supports three *primitive* data types:
  - *numbers*
  - *booleans* (*true* or *false*)
  - *strings*
- In JavaScript are two *compound* data types
  - *objects*
  - *arrays*
- JavaScript defines **specialized** types of objects to represent
  - *functions*
  - *dates*
  - *regular expressions* (not included in this course)

Numbers

- In JavaScript numbers are represented in 64-bit floating-point format and makes no distinction between integers and floating-point numbers
  - The 64-bit format is the same as a *'double'* in Java
- Numeric literals appear in JavaScript using the usual syntax of digits with an optional decimal point. For example:
  - 1
  - 3.14
  - 0001
  - 6.02e23
- Errors in numeric data processing result in a value that is 'not-a-number' (NaN)

Booleans

- In JavaScript we may need to represent if a statement is *true* or *false*
- The truth of falsity is represented by *Boolean* values
- The Boolean values can measure
  - `Truth` (or) `on`
  - `False` (or) `off`
- Boolean values are also measured numerically using an integer value
  - `Truth is [1]`
  - `False by [0]`

Strings

Literals

- There are two types of *literal*:
  - *String literals*: in JavaScript string literals appear between single quotes (‘ … ‘) or double quotes (“ … “)
    - There is no difference between the two approaches
    - This is not always the case with other programming languages (e.g., PHP)
  - *Object literals*: are used to specify new objects
    - Objects and object literals will be covered in later sessions

String

- In JavaScript, a *string* is a sequence characters enclosed in single or double quotes:
  - Using single quotation marks (‘ … ‘)
  - Using double quotation marks (“ … ”)
- The choice of quoting style is up to the programmer as there is no type for a single character in JavaScript (everything is always a string)
  - `‘abc’ === “abc”` (the output will be the same)
- Note: there are different rules for PHP

HTML_JS_1 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- HTML_JS_1
  - Site Root
    - index.html
  - Unit Tests
  - Important Files

Navigator

- html
  - head
    - title
    - meta
    - meta
  - body
    - h2
    - script

.bowerrc | package.json | gulpfile.js | Gruntfile.js | bower.json | index.html

Source | History

```
<!DOCTYPE html>
<!--
HTML_JS_1
-->
<html>
    <head>
        <title>HTML_JS_1</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h2>This text is created under HTML</h2>
        <script>
            document.write('This text is created under JavaScript');
        </script>
    </body>
</html>
```

Single quotes

html > body > script

14:67 INS

19:06 21/07/2018

HTML_JS_1 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- HTML_JS_1
  - Site Root
    - index.html
  - Unit Tests
  - Important Files

Navigator

- html
  - head
    - title
    - meta
    - meta
  - body
    - h2
    - script

.bowerrc | package.json | gulpfile.js | Gruntfile.js | bower.json | index.html

Source | History

```
<!DOCTYPE html>
<!--
HTML_JS_1
-->
<html>
    <head>
        <title>HTML_JS_1</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h2>This text is created under HTML</h2>
        <script>
            document.write("This text is created under JavaScript");
        </script>
    </body>
</html>
```

Double quotes

18:1 INS

19:06 21/07/2018

Escaping in Strings

- In JavaScript when a backslash character (\) appears within a string literal it changes (or escapes) the meaning of the character that follows it
- There are cases where it is necessary to ‘escape’ a character.
- Consider the following strings:

'This text, its’ created under JavaScript’ (This creates an error)

'This text, its\’ created under JavaScript’ (This is correct)

HTML_JS_1 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- HTML_JS_1
  - Site Root
    - index.html
  - Unit Tests
  - Important Files

Navigator

- html
  - head
    - title
    - meta
    - meta
  - body
    - h2
    - script

.bowerrc | package.json | gulpfile.js | Gruntfile.js | bower.json | index.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  HTML_JS_1
4  -->
5  <html>
6      <head>
7          <title>HTML_JS_1</title>
8          <meta charset="UTF-8">
9          <meta name="viewport" content="width=device-width, initial-scale=1.0">
10     </head>
11     <body>
12         <h2>This text is created under HTML</h2>
13         <script>
14             document.write('This text, its' created under JavaScript');
15         </script>
16     </body>
17 </html>
18
```

this symbol indicates an error or warning

shows an error

html > body > script

14:43 INS

20:02 21/07/2018

HTML_JS_1 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- HTML_JS_1
  - Site Root
    - index.html
  - Unit Tests
  - Important Files

Navigator

- html
  - head
    - title
    - meta
    - meta
  - body
    - h2
    - script

.bowerrc | package.json | gulpfile.js | Gruntfile.js | bower.json | index.html

Source | History

```
<!DOCTYPE html>
<!--
HTML_JS_1
-->
<html>
    <head>
        <title>HTML_JS_1</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h2>This text is created under HTML</h2>
        <script>
            document.write('This text, its\' created under JavaScript');
        </script>
    </body>
</html>
```

Escaping corrects the error

html > body > script

14:44 INS

20:02 21/07/2018

Concatenation

- In JavaScript when two strings are joined
  - We say that the two strings have been *concatenated*
- We may also join strings and variables into a single string output
- The following slides show the NetBeans IDE and demonstrate
  - The string *concatenation* process program code
  - Strings are concatenated into a *single string*
  - A *string* and a *variable* are *concatenated into a single string*
  - The resulting output strings in the NetBeans embedded web kit

String Concatenation

- In JavaScript two strings may be joined forming a single string
- Strings and variables may also be concatenated into an output string. For example:

```
document.write("a string" + "another String")
var n = 5;
document.write("a string" + "another String" + n);
```

- In JavaScript strings are *immutable*
  - The string can not be changed – methods that operate on a string typically return a copy

JavaScript String Code

```
<body>
      <h2 style="color: blue>This text is created under HTML5</h2>
      <script>
            var n = 5;
            document.write("This text is created under JavaScript");
            document.write("a string" + "<br>" + "another String" + ": "
            + n + "<br>");
            document.writeln("a string" + "another String" + ": " + n);
      </script>
</body>
```

Note: the error in the code*
Whitespace ignored

HTML_JS_1 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- HTML_JS_1
  - Site Root
    - index.html
  - Unit Tests
  - Important Files

Navigator

- JavaScript
  - n : Number
- HTML
  - html
    - head
      - title
      - meta
      - meta
    - body
      - h2
      - script

.bowerrc | package.json | gulpfile.js | Gruntfile.js | bower.json | index.html

Source | History

```
<!DOCTYPE html>
<!--
HTML_JS_1
-->
<html>
    <head>
        <title>HTML_JS_1</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h2>This text is created under HTML</h2>
        <script>
            var n = 5;
            document.write("a string" + " " + "another String" + ": " + n);
        </script>
    </body>
</html>
```

creating spaces in a text string

html > body > script

15:42 INS

20:27 21/07/2018

This text is created under HTML

a string another String: 5

The concatenated strings with spaces and the concatenated variable (n)

HTML_JS_1 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

HTML_JS_1

Navigator

JavaScript

HTML

bower.json | .bowerrc | package.json | gulpfile.js | Gruntfile.js | index.html

Source | History

```
<!DOCTYPE html>
<!--
HTML_JS_1
-->
<html>
    <head>
        <title>HTML_JS_1</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h2>This text is created under HTML</h2>
        <script>
            var n = 5;
            document.write("a string" + "<br>" + "another String" + ": " + n + "<br>");
            document.writeln("a string" + "another String" + ": " + n);
        </script>
    </body>
</html>
```

Note: the error in the code *

breaking the line of text into 2 lines using

20:1 INS

08:48 23/07/2018

HTML_JS_1

localhost:8383/HTML_JS_1/index.html

This text is created under HTML

a string
another String: 5
a stringanother String: 5

the output using

Note: whitespace ignored *

HTML5_JS_1 - Apache NetBeans IDE 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- HTML5_JS_1
  - Site Root
    - index.html
  - Unit Tests

Navigat...

- JavaScript
  - n : Number
- CSS
  - Elements
    - SELECTOR
- HTML
  - html
    - head
      - title
      - meta
      - meta
    - body
      - h2
      - script

index.html

Source History

```
<!DOCTYPE html>
<!--
HTML5_JS_1
-->
<html>
    <head>
        <title>HTML5_JS_1</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h2 style="color: blue"> This text is created under HTML5</h2>
        <script>
            var n = 5;
            document.write("This text is created under JavaScript");
            document.write("a string" + "<br>" + "another string" + ": " + n + "<br>");
            document.write("*a string" + "another string" + ": " + n + "<br>");
            document.write("#a string" + " " + "another string" + ": " + n);
        </script>
    </body>
</html>
```

22:1 INS

HTML5_JS_1

localhost:8383/HTML5_JS_1/in...

This text is created under HTML5

This text is created under JavaScripta string
another string: 5
*a stringanother string: 5
#a string another string: 5

Conclusions

Review (1)

- In this tutorial we have considered:
  - Scripting environments with an overview of client-side and server-side scripting
  - An overview of client-side and server-side scripting and scripting languages
  - The JavaScript, JavaScript programming, and JavaScript syntax with worked examples
  - Basic HTML and JavaScript template files
  - Scripting environments and scripting with JavaScript and statements including:
    - *Whitespace* / *variables* / *datatypes* / *numbers* / *Booleans*
  - JavaScript identifiers, datatypes, strings, and variables
  - String escaping with string and variable concatenation

Review (2)

- Datatypes and variables (identifiers):
  - *Primitive* and *compound* datatypes
  - *Specialised* types of *object*
- JavaScript expressions, operators, and operands
- Working with strings in JavaScript with output to browsers using:
  - `document.write()` and `document.getElementById()`
  - In worked examples you will see both methods used
  - `document.write()` is a simple method which works for the purposes of the course
  - `document.getElementBy()` is the optimal approach when developing 'real-world' web sites