JavaScript and jQuery Course

Instructor Teresa Pelkie

Session 2

Topic: How to Work with JavaScript Data, Variables and Operators

Chapter 2: Getting started with JavaScript pages 62–71

Chapter 3: page 93 – `parseInt()`, `parseFloat()` methods only

Chapter 3: page 95 – `toFixed()` method only

Chapter 4: How to test and debug a JavaScript application (short overview)

Variables - pages 56-57

A *variable* is an identifier that stores some kind of data.

Variables are created using the `var` keyword which is lower-case.

When a variable is created it is said to be *declared***.** You do not have to store data in the variable when you create it, but you can, and this is usually the case.

Data is stored in a variable by using the **assignment operator** (the equal sign `=`). When a variable is created and assigned a value it is said to be *initialized*. A variable that has not been assigned a value is said to be *undefined*.

Example

```
var myName="Teresa";
var myName = 'Teresa';
```

Note: spaces do not matter around the assignment operator. It is usually better to put one space before and after the assignment operator.

- The `var` keyword is used to declare a variable called `myName`. Iit is typed using camel-case. This is the typical convention, but not required.
- The assignment operator assigns a value of `Teresa` to the variable.
- `Teresa` (the value that is assigned) has quotations around it because it is a *literal*, meaning it is to be interpreted as it is. You can use single or double quotations. Numbers generally are not enclosed in quotations.
- The value on the right-hand side of the assignment operator is placed into the variable on the left-hand side.
- The variable is initialized because it is assigned a value at the time that it is created.
- Variable names must begin with a letter, `$`, or _ (underscore).
- Variable names can only contain numbers, letters, `$`, or _ (underscore).
- Variable names are case sensitive.
- A variable name cannot be a JavaScript *reserved word* (page 57)

A variable can be declared at any time, and then assigned a value later in the code.

Example

```
var myName;
myName=”Teresa”;
```

Multiple variables can be declared in the same statement. It is generally preferable to declare each variable in a separate statement.

Example

```
var  myName, myAddress, myPhoneNumber;
```

Multiple variables can be declared and initialized at once.

Example

```
var myName="Teresa", myAddress="2 Main", myPhoneNumber="7605551212";
```

Variables that contain numeric values (numbers)

Numbers are **not** enclosed in quotations.

String Variables

Strings or literals (text) **are** enclosed in quotations.

How to Name Variables - page 57

- Variable names must begin with a letter, the $ character, or the underscore (_) character
  Note: it is **NOT** a good idea to begin a variable name with the $ character, as jQuery and PHP use that character for other purposes.
- Variable names are case sensitive.
- Variable names may not begin with a number, but they can contain numbers.
- Variable names cannot contain spaces, punctuation, mathematical or logical operators.
- Variable names cannot be a JavaScript reserved words, shown here:

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

JavaScript Variable Types - pages 62-63

JavaScript is called a *loosely typed language* or *dynamically typed*. That is because the variable types are determined when the variable is assigned a value, or when a new value is assigned to the variable. In many other programming languages, you need to assign a type when you declare the variable.

JavaScript supports the following variable types:

string	any literal that is enclosed in quotations
number	any number - positive, negative, decimal, fraction
boolean	a value of true or false, stored as 1 for true and 0 for false
function	any built-in or user-defined function
object	any object

Scope - pages 104-105

Scope indicates the part of the code where the variable exists, meaning variables exist where they are created, and may not be able to used in a different location.

Note: Always use the `var` keyword to declare a variable, as it gives the variable scope. Otherwise, a variable that is created without `var` will have *global scope* which can cause problems.

To learn more and to understand "hoisting"

http://net.tutsplus.com/tutorials/javascript-ajax/quick-tip-javascript-hoisting-explained/

http://www.programmerinterview.com/index.php/javascript/javascript-hoisting/

Performing Calculations - pages 64-69

binary operator - uses two operands

unary operator - uses one operand

Arithmetic Operators

Operator	Operation	Example
+	Addition	`var n = 2+ 4;`
-	Subtraction	`var n = 4 - 2;`
*	Multiplication	`var n = 4 * 2;`
/	Division	`var n = 4 / 2;`
%	Modulus	`var n = 43 % 10; // n = 3 (the remainder)`
++	Increment	`n++; // adds one to value`
--	Decrement	`n--; // subtracts one from value`

Assignment Operators

Operator	Example	Equivalent to
=	`y = x + 3;`	
+=	`x += 3;`	`x = x + 3;`
-=	`x -= 3;`	`x = x -3;`
*=	`x *= 3;`	`x = x * 3;`
/=	`x /= 3;`	`x = x / 3;`

Comparison Operators - page 73

Operator	Description	Example
`==`	Equal To	`if (x == y) { }`
`!=`	Not Equal To	`if (x != y) { }`
`<`	Less Than	`if (x < y) { }`
`>`	GreaterThan	`if (x > y) { }`
`<=`	Less Than or Equal To	`if (x <= y) { }`
`>=`	Greater Than or Equal To	`if (x >= y) { }`

Logical Operators - page 73

Operator	Description	Example
`&&`	AND	`if (x < 10 && y > 1) { }`
`||`	OR (two "bar" characters, upper shift on the \ key)	`if (x < 10 || y > 1) { }`
`!`	NOT (exclamation character, upper shift on the 1 key)	`if( !(x == y) ) { }`

Order of Precedence - pages 64-65, 72-73

The arithmetic operators are executed in the following order. This is referred to as the *order of precedence*.

*	multiplication
/	division
+	addition
-	subtraction

To **alter the order of precedence**, use parentheses. Calculations inside parentheses are always performed first.

Example

```
var n = 4 + 5 * 10;    // result is 54 because the multiplication occurs first
var n = (4 + 5) * 10;  // result is 90 because addition is performed first
```

How to Test the Value of a Variable

After assigning a value to a variable or after performing calculations, you may want to make sure that the value is correct. There are several ways to do this.

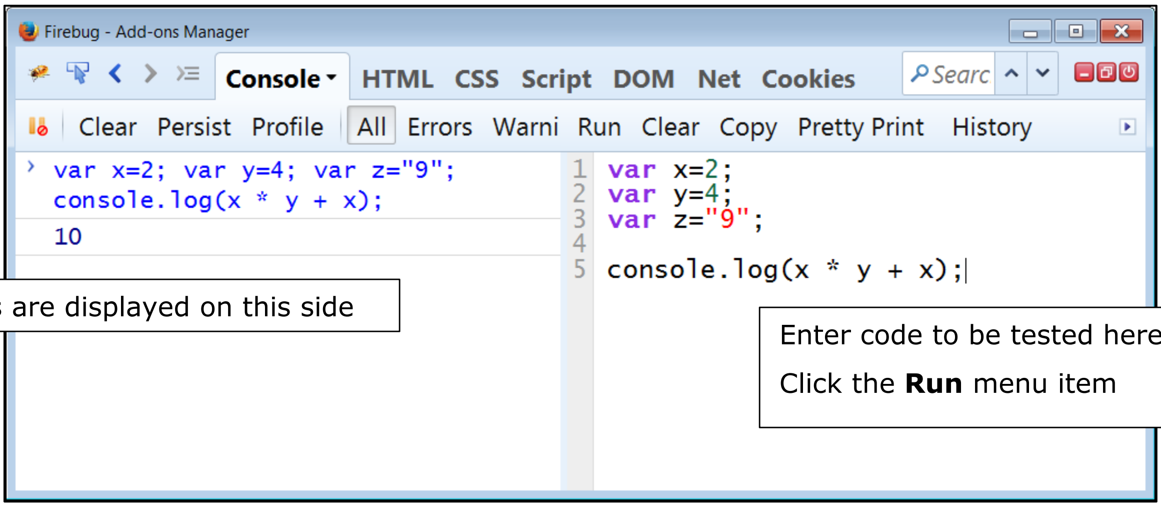
Firebug - Chapter 4

The Firebug Add-on for Firefox is an easy way to test your JavaScript Code. Use the statement shown here:

```
console.log(variable_name_or_expression);
```

Calculation Examples using Firebug

You can use Firebug in the Firefox browser to quickly perform the calculations!

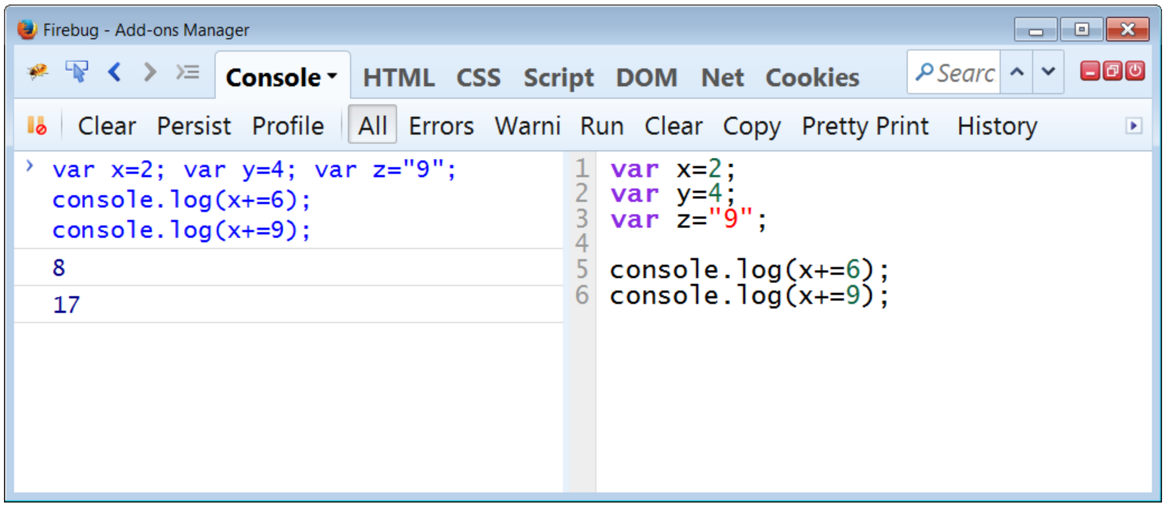

Firebug - Add-ons Manager
Console HTML CSS Script DOM Net Cookies
Clear Persist Profile All Errors Warni Run Clear Copy Pretty Print History
Results are displayed on this side
Enter code to be tested here
Click the **Run** menu item

```
> var x=2; var y=4; var z="9";
  console.log(x * y + x);
  10
```

```
var x=2;
var y=4;
var z="9";

console.log(x * y + x);
```

The figure above shows a calculation performed in the `console.log` statement in the panel on the right. The result (10) is displayed in the console panel on the left.

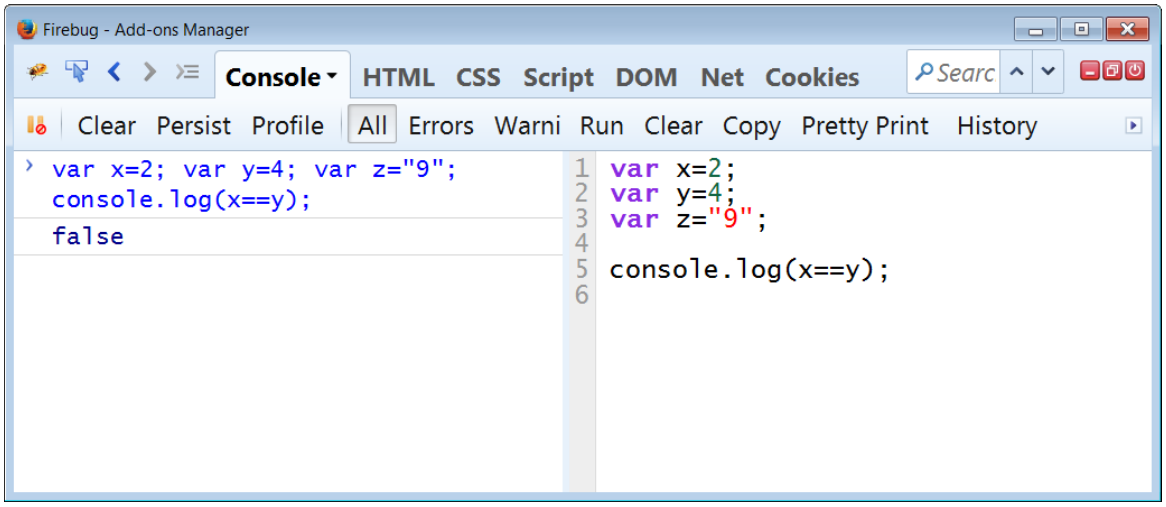

Firebug - Add-ons Manager
Console HTML CSS Script DOM Net Cookies
Clear Persist Profile All Errors Warni Run Clear Copy Pretty Print History

```
> var x=2; var y=4; var z="9";
  console.log(x+=6);
  console.log(x+=9);
  8
  17
```

```
var x=2;
var y=4;
var z="9";

console.log(x+=6);
console.log(x+=9);
```

The figure above shows two successive `console.log` statements. Inside the statements, the short-hand addition operator is used for the variable `x`.

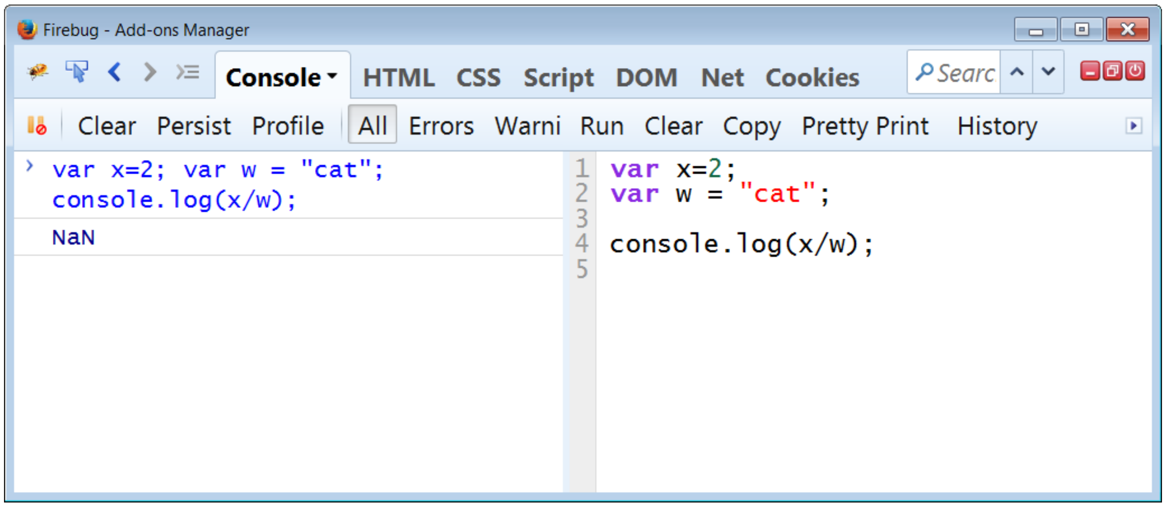

Firebug - Add-ons Manager
Console HTML CSS Script DOM Net Cookies
Clear Persist Profile All Errors Warni Run Clear Copy Pretty Print History

```
> var x=2; var y=4; var z="9";
  console.log(x==y);
  false
```

```
var x=2;
var y=4;
var z="9";

console.log(x==y);
```

The figure shown above shows a comparison operator ("equals" comparison) used in the `console.log` statement. The result of the comparison (`false`) is shown on the left.


Firebug - Add-ons Manager
Console HTML CSS Script DOM Net Cookies
Clear Persist Profile All Errors Warni Run Clear Copy Pretty Print History

```
> var x=2; var w = "cat";
  console.log(x/w);
  NaN
```

```
var x=2;
var w = "cat";

console.log(x/w);
```

The figure shown above shows the result of an invalid division operation performed in the `console.log` statement. The result (`NaN`, or "not a number") is shown on the left. When you divide a number by the text value "cat", you get "not a number".

Editor / View in Browser

You can write to the page using `document.write()` and view the output in a browser. This statement can go inside the `<head>` or `<body>`.

Note: it is OK to use `document.write()` for testing, but we usually use the Document Object Model to write to the page. You will learn how to do that later.

Example

```
<script>
    var firstName = "Maria";
    var lastName = "Smith";

    document.write(firstName + " " + lastName);
</script>
```

Formatting Numbers - page 95

The `toFixed()` method converts and rounds a number into a string, using the specified number of decimal places.

Example

```
var num = 5.56789;
var n = num.toFixed(2);  // n is set to 5.57
```

NaN - pages 71-71

This value (`NaN`) is returned if the value is not a valid number. It can be used to test if a value is a number

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

Math Object - Math() - not in book

The Math object allows you to perform mathematical tasks. **Note that this is uppercase "M".**

Method	Description

`Math.abs(x)`	Returns the absolute value of *x*
	`var n = -7.25;`
	`var x = Math.abs(n); // x is set to 7.25`

`Math.ceil(x)`	Returns *x*, rounded up to nearest integer
	`var n = Math.ceil(1.75); // n is set to 2`
	`var z = Math.ceil(0.4);  // z is set to 1`

`Math.floor(x)`	Returns *x*, rounded down to nearest integer
	`var n = Math.floor(1.75); // n is set to 1`
	`var z = Math.floor(0.4);  // z is set to 0`

`Math.max(n1, n2, n3, ... n)`	Returns the number with the highest value in a list of 1..*n* values `var n = Math.max(1, 5, 7, 9, 3, 8); // n = 9`
`Math.min(n1, n2, n3, ... n)`	Returns the number with the lowest value in a list of 1..*n* values `var n = Math.min(1, 5, 7, 9, 3, 8); // n = 1`
`Math.random()`	Returns a random number between 0 and 1 `var n = Math.random(); // possible value: 0.314159265`

`Math.round(x)`	Rounds *x* to the nearest integer
	`var n = Math.round(1.7);   // n is set to 2`
	`var n = Math.round(0.408); // n is set to 0`

`Math.pow(x, y)`	Returns the value of *x* raised to the power of *y* `var n = Math.pow(2, 4); // n is set to 16 (2*2*2*2)`
`Math.sqrt(x)`	Returns the square root of *x* `var n = Math.sqrt(81); // n is set to 9`
For other `Math` object methods, see http://www.w3schools.com/jsref/jsref_obj_math.asp	

The String Object - String() - pages 68-69

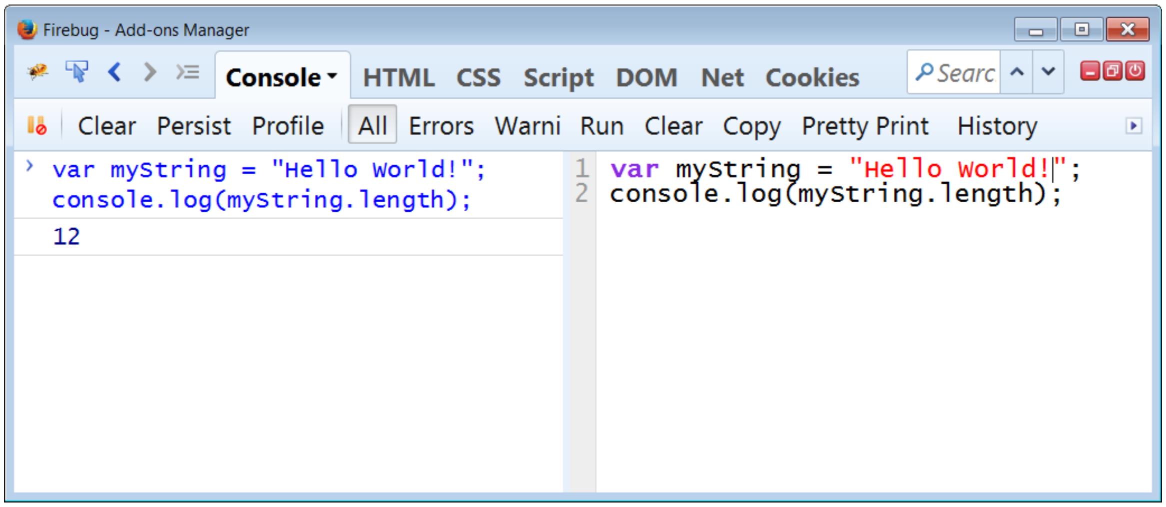
Any variable that contains text or a number that is to be treated as text is a *string*, and is also a `String` object. **Note that this is uppercase “S”.**

Any variable that contains text or a literal is automatically a `String` object, which means that you can access its properties and methods.

Example - the length property (returns the length of a string)

```
var myString = "Hello World!";
var x = myString.length;  // x is 12
```

You can try these examples in Firebug



```
> var myString = "Hello World!";
console.log(myString.length);
12
```

```
var myString = "Hello World!";
console.log(myString.length);
```

The figure above shows the `String.length` method being used to determine the length of the string value in the variable `myString`.

String Object - most not in book

The `String` object provides methods to perform operations on string variables. Although the `String` object itself is defined (with an uppercase "S"), you perform `String` methods on your string variables.

Method	**Description**

string.charAt(*x*)	Returns the character at the index value specified by *x.*
	Note: string positions start counting at zero.
	`var myString = -"Hello World";`
	`var x = myString.chartAt(0); // x is set to "H"`

string.concat(*join_string*)	Concatenates (joins) one or more strings to the target string variable.
	`var myString = "Hello";`
	`myString.concat(" World");  // myString is now "Hello World"`
	
	`myString.concat(" Big", " World"); // myString is now`
	`// "Hello World Big World"`

string.indexOf(*value*)	Returns the zero-based starting position of *value* in the string being searched, or -1 if the *value* is not found in the string being searched
	`var myString = "Hello World";`
	`var z = myString.indexOf("Wo");  // z is set to 6`
	`var q = myString.indexOf("No");  // q is set to -1`

string.lastIndexOf(*value*)	Returns the zero-based position of the last occurrence of *value* in the string being searched, or -1 if the *value* is not found in the string being searched.
	`var myString = "Hello World";`
	`var z = myString.lastIndexOf("o");  // z is set to 7`
	`var q = myString.lastIndexOf("e"); // q is set to 1`
	`var m = myString.lastIndexOf("b"); // m is set to -1`

string.replace(*match*, *replace*)	Returns a string with the value specified in *match* replaced with the value specified in *replace*. If the value specified in *match* is not found, returns the original value of the string.
	`var myString = "Hello World";`
	`var newString = myString.replace("World", "JavaScript");`
	`// newString is set to "Hello JavaScript"`
	`var sameString = myString.replace("HTML5", "JavaScript");`
	`// sameString is set to "Hello World" (match not found)`

string.slice(*start*, *end*)	Returns a string containing the characters in the original string starting at the position given in *start* through the position given in *end-1* ("up to but not including"). If *end* is not specified, returns all characters from *start* to the end of the string.
	If *start* is a negative number, get that number of characters starting at the end of the string. If *start* is negative, a negative number can also be sepcified for *end* to indicate how many characters are to be trimmed from the result.
	`var myString = "Hello World";`
	`var s1 = myString.slice(3);      // s1 is set to "lo World"`
	`var s2 = myString.slice(1, 3);   // s2 is set to "el"`
	`var s3 = myString.slice(15);     // s3 is an empty string`
	`var s4 = myString.slice(-1);     // s4 is set to "d"`
	`var s5 = myString.slice(-5);     // s5 is set to "World"`

	`var s6 = myString.slice(-5, -3);  // s6 is set to "Wo"`

`string.split(`*`separator`*`, `*`limit`*`)`	Split a string into an array of substrings. The split is performed where the string specified in *`separator`* is encounted. The optional *`limit`* specifies the maximum number of splits to perform.
	`var myString = "Hello World Big Oceans";`
	`var a1 = myString.split(" ");  // a single blank character`
	`// a1 is set to ["Hello", "World", "Big", "Oceans"]`
	`var a2 = myString.split(""); // split on empty string`
	`// a2 is set to ["H", "e", "l", "l", "o", ...]`
	`// all individual characters in the original string become`
	`// individual array elements`
	`var a3 = myString.split(" ", 2);`
	`// a3 is set to ["Hello", "World"]`

`string.substr(`*`start`*`, `*`length`*`)`	Extract from a string, beginning at *`start`* for the specified *`length`* number of characters.
	To start the extraction at the end of the string, use a negative number for *`start`*.
	`var myString = "Hello World";`
	`var s1 = myString.substr(6);      // s1 is set to "World"`
	`var s2 = myString.substr(6, 3);   // s2 is set to "Wor"`
	`var s3 = myString.substr(-1);     // s3 is set to "d"`
	`var s4 = myString.substr(-5);     // s4 is set to "World"`
	`var s5 = myString.substr(-5, 3);  // s5 is set to "Wor"`

`string.substr(`*`start`*`, `*end*`)`	Extract from a string, beginning at *`start`* up to but not including the position specified for *end*. If *end* is not specified, extract to the end of the string.
	`var myString = "Hello World";`
	`var s1 = myString.substring(4);     // s1 is set to "o World"`
	`var s2 = myString.substring(6,8);   // s2 is set to "o Wo"`
	`var s3 = myString.substring(-1, 8); // s3 is set to "Hello Wo"`
	`// negative start value treated as starting at position 0`

`string.toLowerCase()`	Converts a string to lowercase letters.
	`var myString = "Hello World";`
	`var s1= myString.toLowerCase(); // s1 is set to "hello world"`

`string.toUpperCase()`	Converts a string to uppercase letters.
	`var myString = "Hello World";`
	`var s1 = myString.toUpperCase();  // s1 is set to "HELLO WORLD"`

For other `String` object methods, see

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

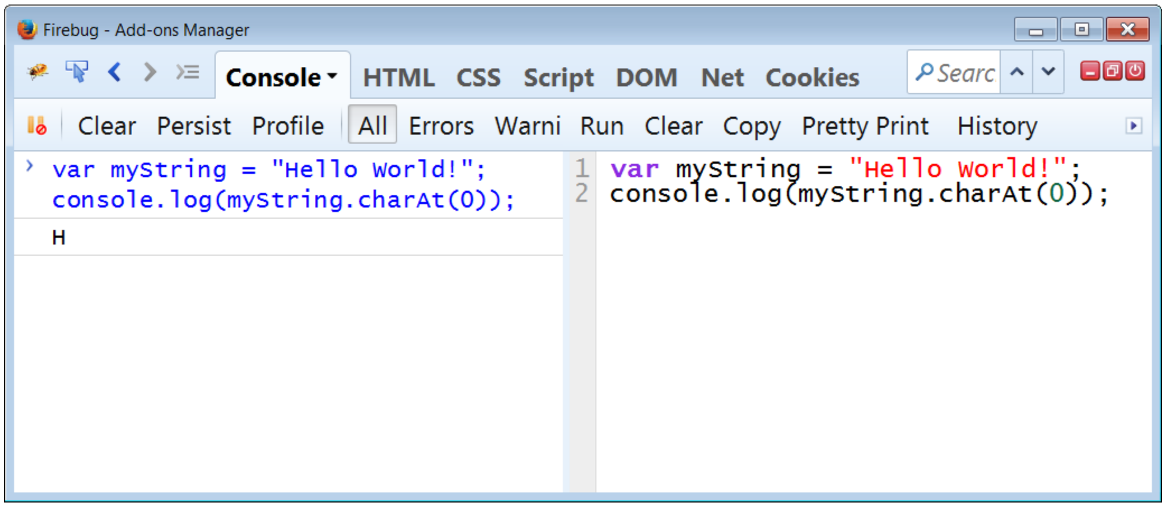
String Object Examples

These examples show how you can use Firebug to test string methods.

charAt() method - returns the character at a numeric index position

```
var myString = "Hello World!";
console.log(myString.charAt(0));
```

returns: `H`



```
var myString = "Hello World!";
console.log(myString.charAt(0));
H
```

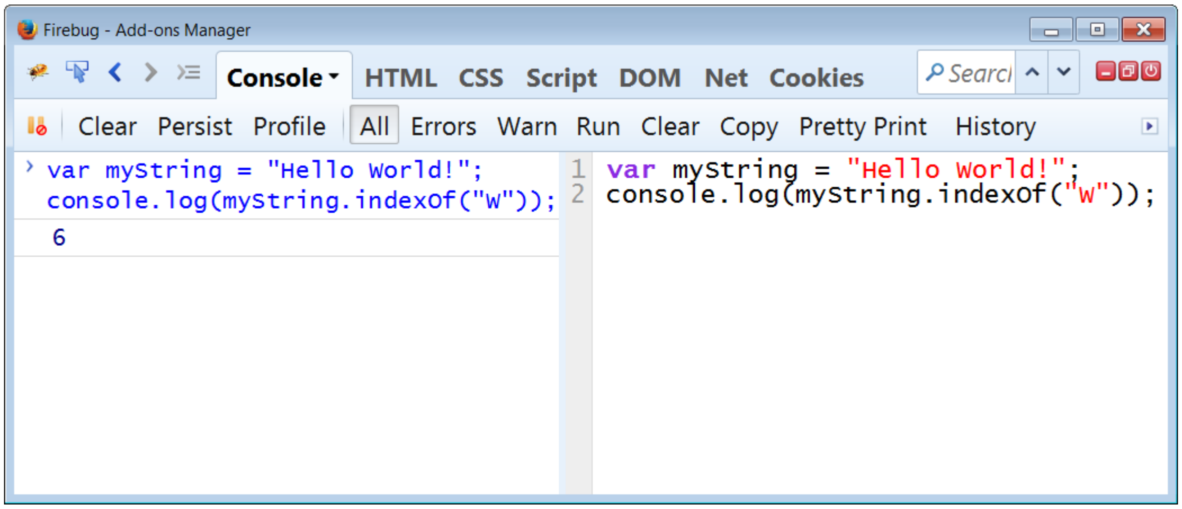
```
var myString = "Hello World!";
console.log(myString.charAt(0));
```

The figure above shows the `charAt()` method to get the value of the character at the first position of the string.

indexOf() method - returns the zero-based index (position) of a search string within the source string. The method returns -1 if the search string does not occur within the source string.

```
var myString = "Hello World!"
console.log(myString.indexOf("W"));
```

returns: 6 (the search string is at position 6 of the source string)



```
var myString = "Hello World!";
console.log(myString.indexOf("W"));
6
```

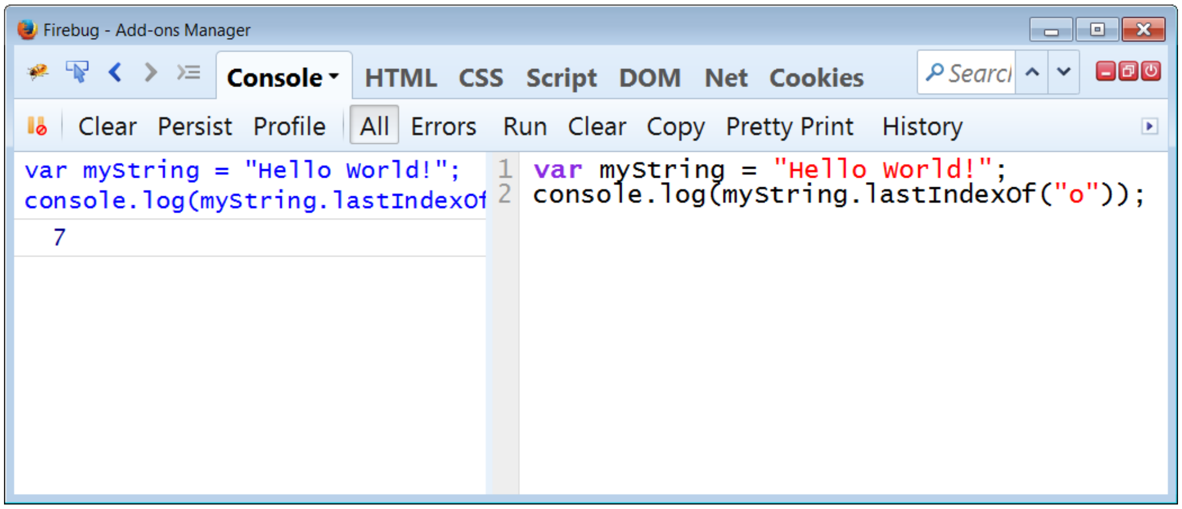
```
var myString = "Hello World!";
console.log(myString.indexOf("W"));
```

The figure above shows the `indexOf()` method to get the position of the search string "W" in the source string "Hello World!".

lastIndexOf() method - returns the zero-based index (position) of the last occurrence of a search string within the source string. This method returns -1 if the value to search for never occurs.

```
var myString = "Hello World!";
console.log(myString.lastIndexOf("o"));
```

returns: 7 (the search string is at position 7 of the source string, this is the last "o" in the string)



```
var myString = "Hello World!";
console.log(myString.lastIndexOf
   7
```

```
var myString = "Hello World!";
console.log(myString.lastIndexOf("o"));
```

The figure above show the `lastIndexOf()` method to get the position of the search string "o" in the source string "Hello World!".

Manipulating Strings, Concatenation - page 69

Strings can be *concatenated* (joined) using the concatenation operator (+), with the assignment operator (=).

```
var myString1 = “Hello”;
var myString2 = “World!”;

newString  = myString1 + “ ” + myString2;  // value of newString: Hello World!
```

Note: when concatenating two strings, you can concatenate a blank space between the strings to make the result readable.

Escape Sequences - pages 68-69

If you want to include a quotation inside a string, you can:

- use both double and single quotations
- use the escape character - the backslash - \

Example

```
console.log("Hello \"Big\" World!");
```

returns: Hello "Big" World!

Example

```
console.log('Hello "Big" World!');
```

returns: Hello "Big" World!

Other special characters that can be inserted into a string:

`\'`	single quote
`\"`	double quote
`\\`	backslash
`\n`	new line
`\r`	carriage return
`\t`	tab
`\b`	backspace
`\f`	form feed

Type Conversions - page 71

Because JavaScript is *loosely typed*, you may need to tell the browser that a string is a number and vice versa.

Convert a string to a number

`parseInt()` function - parses a string and returns an integer

`parseFloat()` function - parses a string and returns a floating point number or decimal

Convert a number to a string

`toString()` method - converts a number to a string

Sample Files for this session

File Name	Description
02_assignment_sample_document_write.html	Example showing several definitions, calculations, and writing results to the page using document.write().
02_assignment_sample_document_write_2.html	Example showing several definitions, calculations, and writing results to the page using document.write(). In this example, the <script> blocks are used for each individual item to be written to the page.
02_assignment_sample_DOM.html	Example showing several definitions, calculations, and writing results to the page using document.getElementById() - the DOM.
02_math-ceil.html	Use the Math.ceil() method to round a number up to the next integer.
02_math-floor.html	Use the Math.floor() method to round a number down to the next integer.
02_math-random.html	Use the Math.random() method to generate a number between 0 and 1 (not including 1).
02_math-random2.html	Use the Math.random() method and the Math.floor() method to generate a random number between 0 and 10.
02_math-round.html	Use the Math.round() method to round to the nearest integer (round up or round down).
02_NaN.html	Shows how the value NaN is returned if a value cannot be parsed as a number, and how to use the isNaN function to determine if a value can be parsed as a number.
02_parseInt.html	Use the parseInt() function to parse the integer value from a string value. Also shows parseFloat() function to parse a value with decimal positions from a string value.
02_string-indexOf.html	Use the indexOf() method to determine the position of the search string in the source string.
02_string-length.html	Use the length property to determine the length of a string.
02_string-slice.html	Use the slice() method to extract a stirng from the source string.
02_string-toUpperCase.html	Use the toUpperCase() method to convert a string to all upper case characters.
02_toFixed.html	Use the toFixed() method to format a numeric vlaue to a specified number of decimal places.
02_toFixed_mistake.html	Example that shows an incorrect use of the toFixed() method, used before a calculation is performed.
02_variable_expression_alert.html	How to use the alert() method to display a value that is calculated in an expression.
02_variable_expression_alert2.html	How to use the alert() method to display a value that is concatenated with a value that is calculated in an

	expression.
`02_variable_expression_document_write.html`	How to use the `document.write()` method to write a value to the page.
`02_variable_expression_document_write2.html`	How to use the `document.write()` method to write a value concatenated with an expression to the page.
`02_variable_expression_DOM.html`	How to use the `document.getElementById()` method to write a value to the page using the DOM.