JAVASCRIPT

Introduction to Javascript

- JavaScript is the programming language of the Web

- Web developers must learn

  HTML to specify the content of web pages

  CSS to specify the presentation of web pages

  JavaScript to perform action on the web pages.

Comments

- Two styles of comments.
  - Any text between a // and the end of a line.
- Any text between the characters /* and */

Literals

- A literal is a data value that appears directly in a program. The following are all literals
  - 12 // The number twelve
  - 1.2 // The number one point two
  - "hello world" // A string of text
  - 'Hi' // Another string
  - true // A Boolean value
  - false // The other Boolean value
  - /javascript/gi // A "regular expression" literal (for pattern matching)
  - null // Absence of an object

Identifiers

- An identifier is simply a name
  - identifiers are used to name variables, functions and certain loops in javascript code.
- A JavaScript identifier must begin with a letter, an underscore (_), or a dollar sign ($).
- Subsequent characters can be letters, digits, underscores, or dollar signs.
- Digits are not allowed as the first character so that JavaScript can easily distinguish identifiers from numbers

Reserved Words

- JavaScript reserves a number of identifiers as the keywords of the language itself.
- Don’t use these words as identifiers in your program.

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

Javascript Types

- The data types of a language describe the basic elements that can be used within that language
  - ■ Numbers
  - ■ Strings
  - ■ Booleans
  - ■ Null
  - ■ Undefined
  - ■ Objects

Working with numbers

- All these are perfectly valid numbers in JavaScript:

```
var h = 0xe;
var i = 0x2;
var j = h * i;
alert(j);
```

Commonly used numeric function

- *isNaN() function to determine whether a number is legal or valid according to the ECMA-262 specification.*
- *NaN is an abbreviation for Not a Number, and it represents an illegal number*
- the string “four” is not a number to the *isNaN() function, whereas the string “4” is.*
- document.write("Is Not a Number: " + isNaN("four"));

Working with String-
String methods and properties

- Length property
  - The length property on a string object gives the length of a string, not including the enclosing quotation marks.

```
alert("This is a string.".length);
```

much more common to call the length property on a variable, like this

```
var x = "This is a string.";
alert(x.length);
```

Commonly used String methods

- substring, slice, substr, concat, toUpperCase, toLowerCase, match, search, and replace .

- **There are 3 methods for extracting a part of a string:**
  - slice(start, end)
  - substring(start, end)
  - substr(start, length)

```
var myString = "This is a string.";
alert(myString.substring(3)); //Returns "s is a string.“
alert(myString.substring(3,9)); //Returns "s is a“
alert(myString.slice(3)); //Returns "s is a string.“
alert(myString.slice(3,9)); //Returns "s is a“
```

```
var str = "Hello world!";
var res = str.substring(1, 4);
```

The difference between slice/substring() is that substring cannot accept negative indexes.

Finding a String in a String

- Finding a String in a String
  - The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a String.

    ```
    var str = "Please locate where 'locate' occurs!";
    alert(str.indexOf("locate"));
    ```

    The lastIndexOf() method returns the index of the last occurrence Of a specified text in a string:

    ```
    var str = "Please locate where 'locate' occurs!";
    var pos = str.lastIndexOf("locate");
    ```

Searching for a String in a String

The **search()** method searches a string for a specified value and returns the position of the match:

```
var str = "Please locate where 'locate‘ occurs!";
alert(str.search(“locate”));
```

Replacing String Content

- The **replace()** method replaces a specified value with another value in a string.

```
str = “Hello good morning";
alert(str.replace (“morning",“afternoon"));
```

Converting to Upper and Lower Case

- A string is converted to upper case with **toUpperCase()**

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();
```

- A string is converted to lower case with **toLowerCase()**

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();
```

Concat method

The *concat method concatenates two strings together:*

```
var firstString = "Hello ";var finalString =
firstString.concat("World");alert(finalString);
//Outputs "Hello World"
```

charAt()/charCodeAt()

- The **charAt()** method returns the character at a specified index (position) in a string.

  ```
  var str = "HELLO WORLD";
  str.charAt(0);   // returns H
  ```

- The **charCodeAt()** method returns the unicode of the character at a specified index in a string.

  ```
  var str = "HELLO WORLD";
  str.charCodeAt(0);        // returns 72
  ```

Undefined

- Undefined is a state, sometimes used like a value, to represent a variable that hasn’t yet contained a value.
- This state is different from *null.*
- undefined is a type itself while null is an object

Objects

- Objects in JavaScript are a collection of properties, each of which can contain a value.
- Each value stored in the properties can be a value, another object, or even a function.
- You can define your own objects with JavaScript, or you can use the several built-in objects.

Objects cont.

- creates an empty object called *myObject.*

  var myObject = {};

- Object with several properties.

  // person object declaration

```
 var person =
{
firstName:"John",
lastName:"Doe",
age:50,
eyeColor:"blue“
};
```

Accessing Object Properties/Methods

- You can access the object properties in two ways
  - **person.lastName;**
  - **person["lastName"];**

  An **object method** is a **function definition** stored as an object property.

  You can call an object method with the following
  Syntax

  objectName.methodName()

  name = person.fullName();

Arrays

- JavaScript arrays are used to store multiple values in a single variable.

- creating an array named cars.

Using an array literal is the easiest way to create a JavaScript Array.

var cars = ["Saab", "Volvo", "BMW"];

Alternate

var cars = new Array("Saab", "Volvo", "BMW");

Access the Elements of an Array

- You refer to an array element by referring to the **index number**.

  var name = cars[0];
  This statement modifies the first element in cars:
  cars[0] = "Opel";

  Note: You Can Have Different Objects in One Array

  You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

  myArray[0] = Date.now;
  myArray[1] = myFunction;
  myArray[2] = myCars;

Array Properties and Methods

var x = cars.length;   // The length property returns the number of elements in cars'

var y = cars.sort();   // The sort() method sort cars in alphabetical order

Adding Array Elements

- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits[fruits.length] = "Lemon";

  // adds a new element (Lemon) to fruits

Adding elements with high indexes can create undefined "holes"

in an array.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[10] = "Lemon";
// adds a new element (Lemon) to fruits
```

Looping Array Elements

- The best way to loop through an array is using a standard for loop.

```
var index;
var fruits =["Banana", "Orange", "Apple", "Mango"];
for (index = 0; index < fruits.length; index++)
 {
   text += fruits[index];
}
```

JavaScript typeof operator

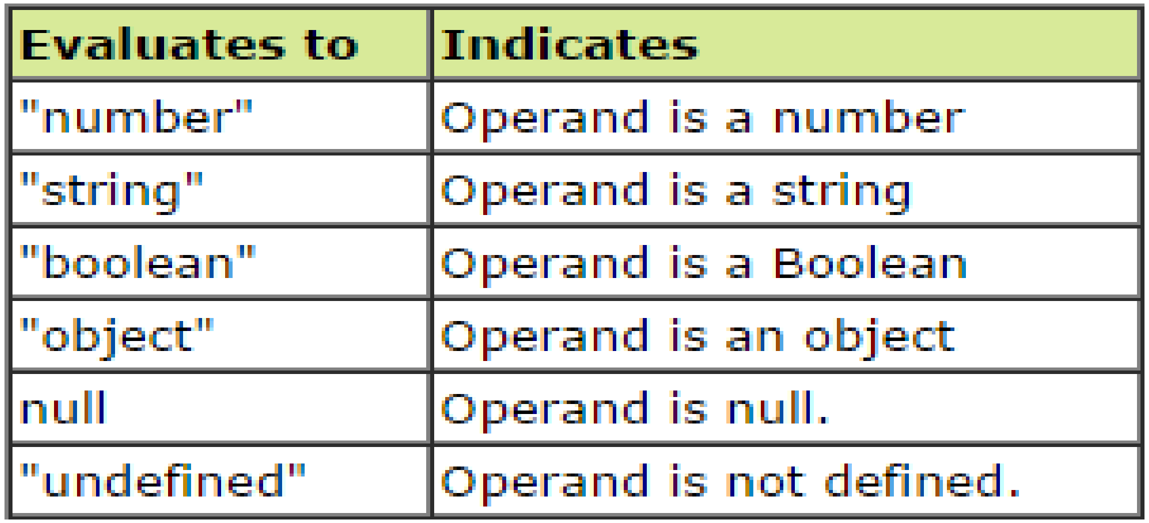
- The "typeof" operator in JavaScript allows you to probe the data type of its operand.

var myvar=5

alert(typeof myvar) //alerts "number“

Here's a list of possible values returned by the typeof operator.

typeof operator evaluates to



Evaluates to	Indicates
"number"	Operand is a number
"string"	Operand is a string
"boolean"	Operand is a Boolean
"object"	Operand is an object
null	Operand is null.
"undefined"	Operand is not defined.

JavaScript Array Methods
Converting Arrays to Strings

- In JavaScript, all objects have the valueOf() and toString() methods.

```
var fruits=["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.valueOf());
```

```
var fruits=["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.toString());
```

The **join()** method also joins all array elements into a string.
It behaves just like toString(), but you can specify the separator

```
var fruits = ["Banana", "Orange","Apple", "Mango"];
fruits.join(" * ");
```

Popping items out of an array

- The **pop()** method removes the last element from an array.
- The pop() method returns the string that was "popped out".

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();  // Removes the last element ("Mango") from fruits
```

pushing items into an array

- pushing items into an array using push() method.
- The push() method returns the new array length.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");     //  Adds a new element ("Kiwi") to fruits
```

Shifting/Unshifting Elements

- The **shift()** method removes the first element of an array, and "shifts" all other elements one place down.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();  // Removes the first element "Banana“ from  fruits
```

- The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

Changing Elements/Changing Elements

- Array elements are accessed using their **index number**.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";       // Changes the first element of fruits to "Kiwi“
```

- The length property provides an easy way to append a new element to an array.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";         // Appends "Kiwi" to fruit
```

- Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator **delete**.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];          // Changes the first element in fruits to undefined
```

Splicing an Array

- The **splice()** method can be used to add new items to an array.

```
var fruits =["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

array.splice(index, howMany, [element1][, ..., elementN]);

- **index** : Index at which to start changing the array.
- **howMany** : An integer indicating the number of old array elements to remove. If howMany is 0, no elements are removed.
- **element1, ..., elementN** : The elements to add to the array. If you don't specify any elements, splice simply removes elements from the array.

Sorting/Reversing an Array

- The **sort()** method sorts an array alphabetically.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();            // Sorts the elements of fruits
```

- The **reverse()** method reverses the elements in an array.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();            // Sorts the elements of fruits
fruits.reverse();         // Reverses the order of the elements
```

Joining Arrays

- The **concat()** method creates a new array by concatenating two arrays.

- The concat() method can take any number of

array arguments.

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias","Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);     // Concatenates arr1
with arr2 and arr3
```

Defining and using variables

- Variables are declared in JavaScript with the *var keyword .*
- Variables in JavaScript are not strongly typed.
- It’s not necessary to declare whether a given variable will hold an integer, a floating point number, or a string.
- You can also change the type of data being held within a variable through simple reassignment.

```
var x = 4;
x = "Now it's a string.";
```

Variable scope

- Variables are *globally scoped when they are used outside a function.*
- A globally scoped variable can be accessed throughout your JavaScript program.
- Variables defined within a function are scoped solely within that function and cannot be accessed outside the function.
- Function parameters are scoped locally to the function

The *Date object*

- The Date object includes many methods that are helpful when working with dates in JavaScript.

// **4 ways** of initiating a date.

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

// **new Date(), without parameters.**

```
var myDate = new Date();
document.write(myDate);
```

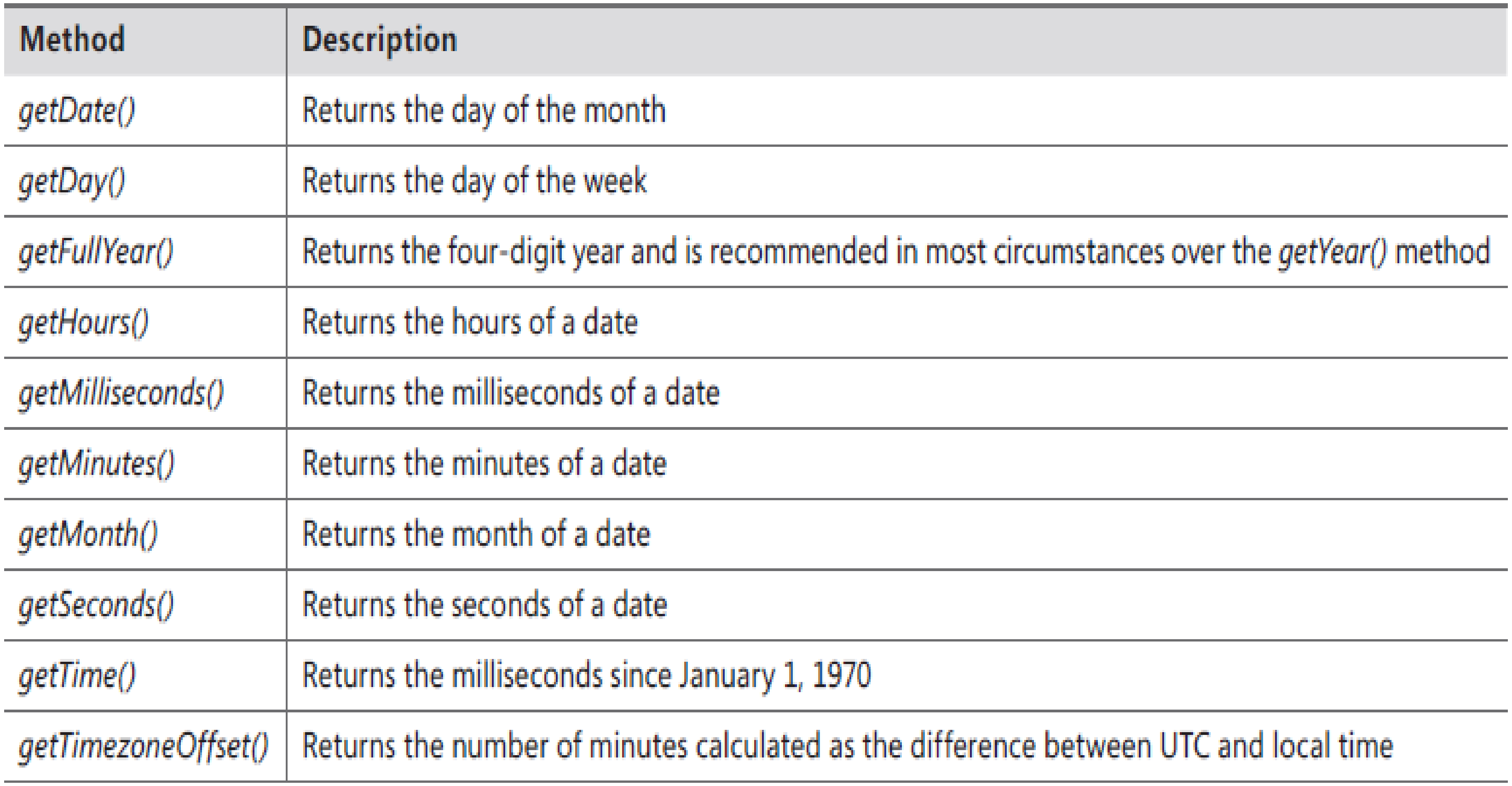
// **new Date(), with 7 numbers**

```
var myDate = new Date(99,5,24,11,33,30,0);
document.write(myDate);
```

Points to be remembered when using a *Date object*

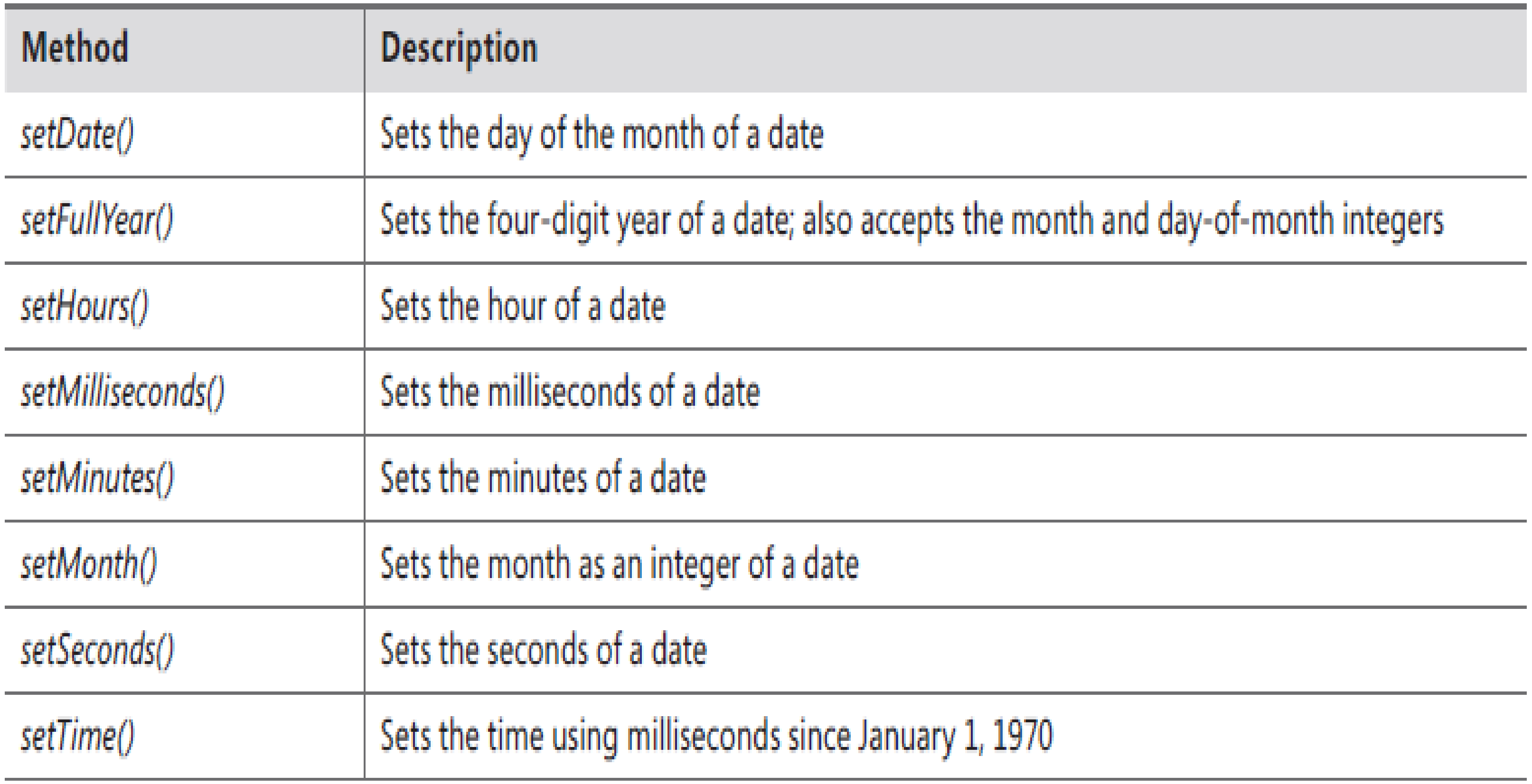
- The year should be given with four digits unless you want to specify a year between the year 1900 and the year 2000, in which case you'd just send in the two-digit year, 0 through 99, which is then added to 1900. So, 2008 equals the year 2008, but 98 is turned into 1998.
- The month is represented by an integer 0 through 11, with 0 being January and 11 being December.
- The day is an integer from 1 to 31.
- Hours are represented by 0 through 23, where 23 represents 11 P.M.
- Minutes and seconds are both integers ranging from 0 to 59.
- Milliseconds are an integer from 0 to 999.

The get methods of the Date object



Method	Description
getDate()	Returns the day of the month
getDay()	Returns the day of the week
getFullYear()	Returns the four-digit year and is recommended in most circumstances over the *getYear()* method
getHours()	Returns the hours of a date
getMilliseconds()	Returns the milliseconds of a date
getMinutes()	Returns the minutes of a date
getMonth()	Returns the month of a date
getSeconds()	Returns the seconds of a date
getTime()	Returns the milliseconds since January 1, 1970
getTimezoneOffset()	Returns the number of minutes calculated as the difference between UTC and local time

The set methods of the Date object



Method	Description
setDate()	Sets the day of the month of a date
setFullYear()	Sets the four-digit year of a date; also accepts the month and day-of-month integers
setHours()	Sets the hour of a date
setMilliseconds()	Sets the milliseconds of a date
setMinutes()	Sets the minutes of a date
setMonth()	Sets the month as an integer of a date
setSeconds()	Sets the seconds of a date
setTime()	Sets the time using milliseconds since January 1, 1970

JavaScript Regular Expressions

- A regular expression is a sequence of characters that forms a search pattern.
- The search pattern can be used for text search and text replace operations.
- Syntax : /*pattern*/*modifiers*;
- In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().
- **The search() method** uses an expression to search for a match, and returns the position of the match.
- **The replace() method** returns a modified string where the pattern is replaced.

Using String search()/replace() With a Regular Expression

- Use a regular expression to do a case-insensitive search for “pirates" in a string.

```
var str = “The pirates of the carribean";
var n = str.search(/pirates/i);
```

- Use a case insensitive regular expression to replace pirates with players in a string.

```
var str = " The pirates of the carribean";
var res = str.replace(/players/i, ”pirates");
```

Using String search()/replace() With String

- The search method will also accept a string as search argument. The string argument will be converted to a regular expression.

  var str = " The pirates of the carribean";
  var n = str.search(“pirates");

- The replace() method will also accept a string as search argument.

  var str = " **The pirates of the carribean**";
  var res = str.replace(“players", " **pirates** ");

Using the RegExp Object

- The test() method is a RegExp expression method.
- Using test()
  - It searches a string for a pattern, and returns true or false, depending on the result.

```
var patt = /e/;
patt.test("The best things in life are free!");
// output : true
```

Using the RegExp Object - cont

- The exec() method is a RegExp expression method.
- It searches a string for a specified pattern, and returns the found text.
- If no match is found, it returns *null.*

```
/e/.exec("The best things in life are free!");
Output: e
```

Modifiers

Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Brackets

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any digit between the brackets
[^0-9]	Find any digit NOT between the brackets
(x|y)	Find any of the alternatives specified

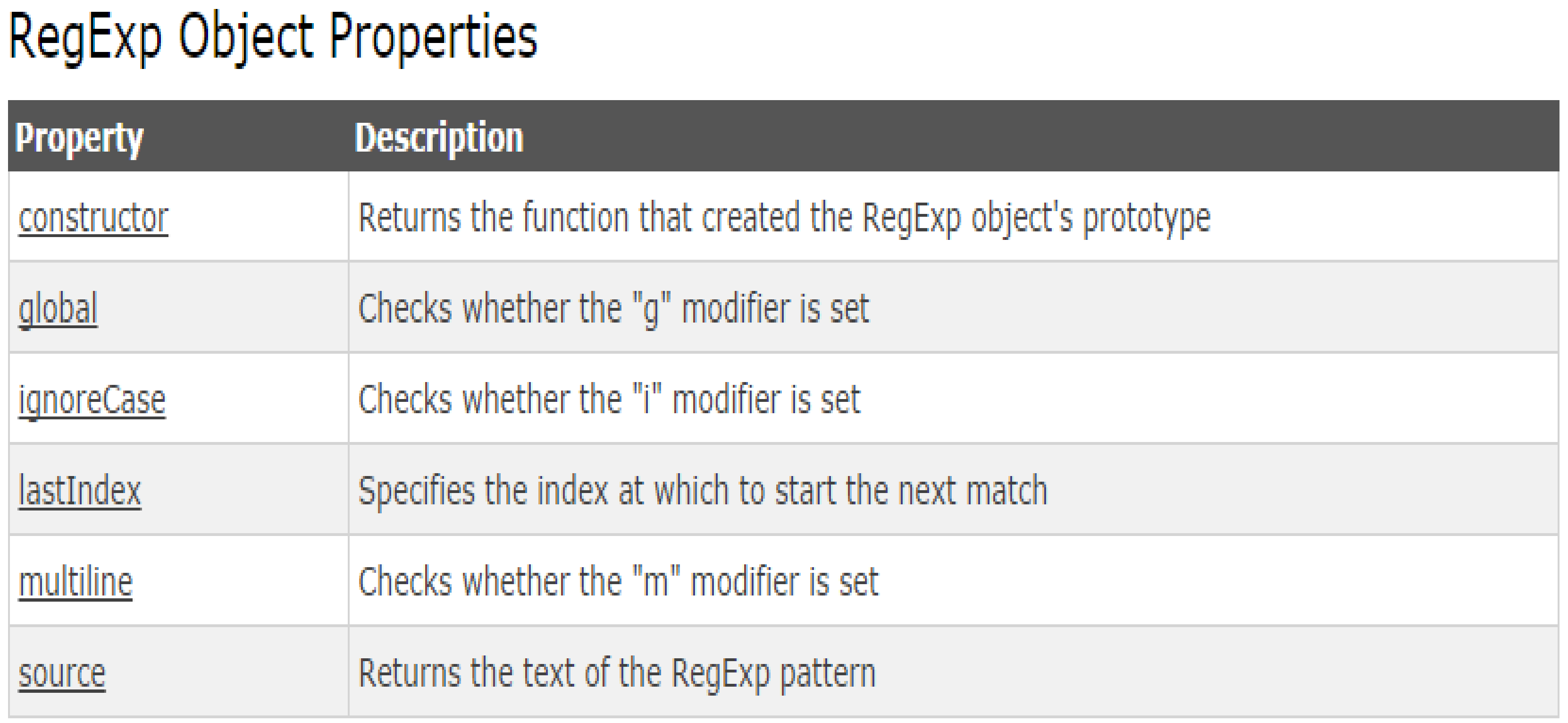
Metacharacters

Metacharacters are characters with a special meaning:

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

Quantifiers

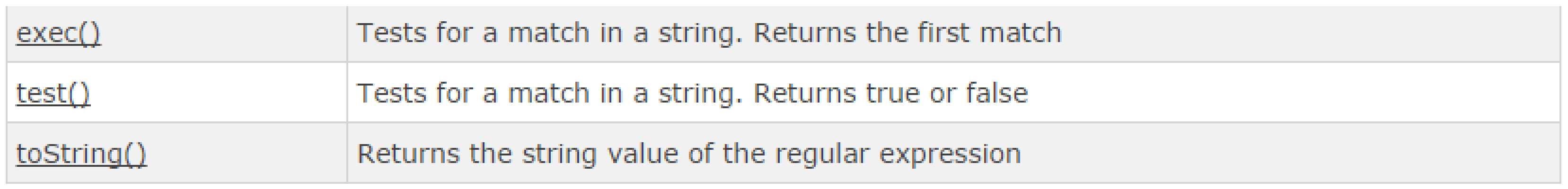
Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X* *n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*



RegExp Object Properties

Property	Description
constructor	Returns the function that created the RegExp object's prototype
global	Checks whether the "g" modifier is set
ignoreCase	Checks whether the "i" modifier is set
lastIndex	Specifies the index at which to start the next match
multiline	Checks whether the "m" modifier is set
source	Returns the text of the RegExp pattern

RegExp Object Methods



exec()	Tests for a match in a string. Returns the first match
test()	Tests for a match in a string. Returns true or false
toString()	Returns the string value of the regular expression

Using operators and expressions

- Additive operators
- Multiplicative operators
- Bitwise operators
- Equality operators
- Relational operators
- Unary operators
- Assignment operators
- The comma operator

Additive operators

- The addition operator operates in different ways, depending on the types of the values being added.
- When adding two strings, the addition operator concatenates the left and right arguments.

```
var a = 947;
var b= "Rush";
var c= 53;
var d = "43";
var result1 = a+ b; // result1 will be the string "947Rush";
var result2 = a + c; // result2 will be the number  1000;
var result3 = a + d; // result3 will be 94743;
```

Multiplicative operators

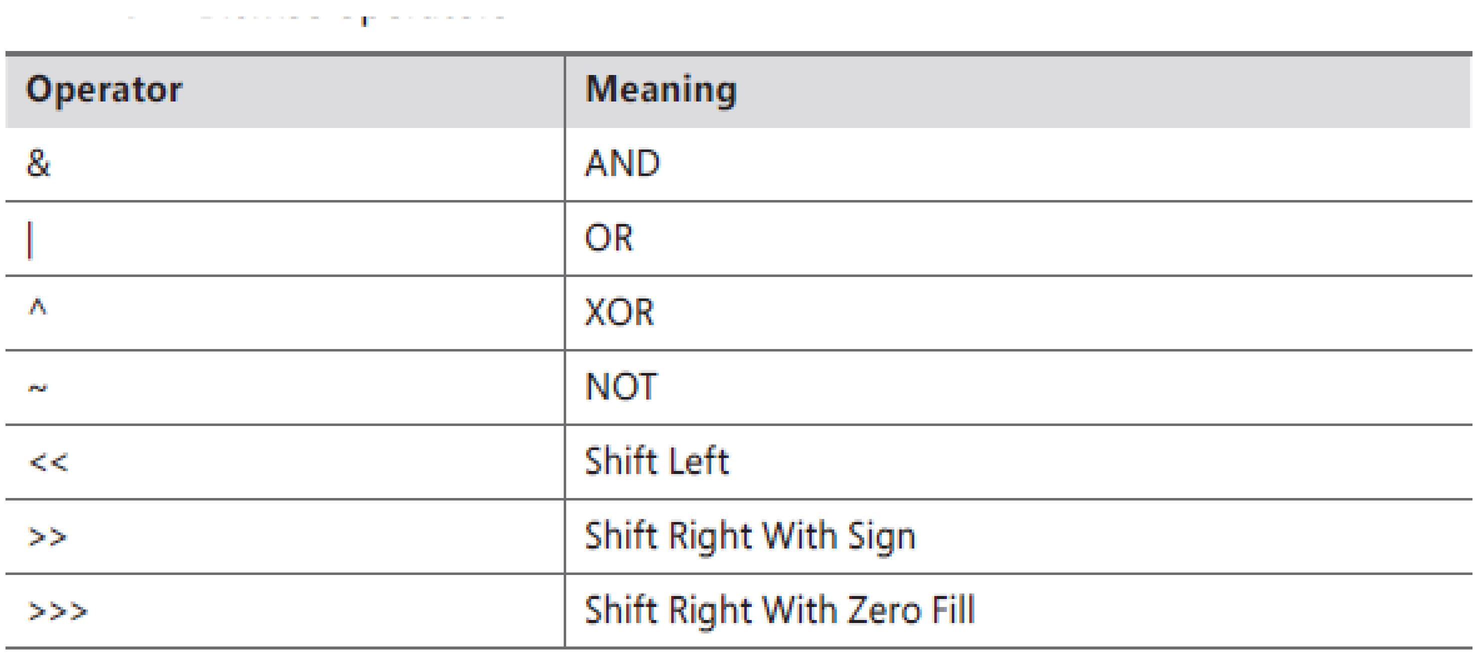
- multiplication operator (*)

  - var mult = 2 * 2;

  - var divisi= 4/2;

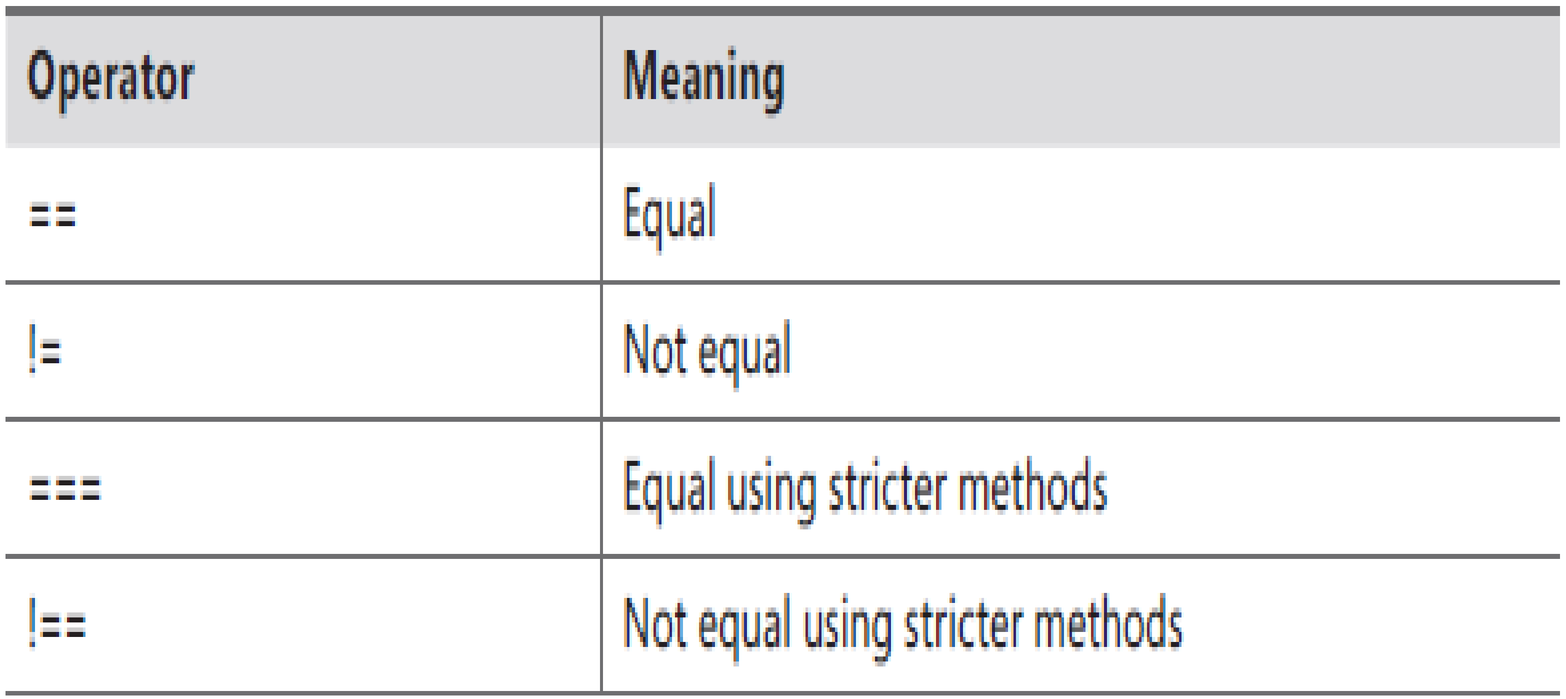
  - var mod= (4%3);

Bitwise operators



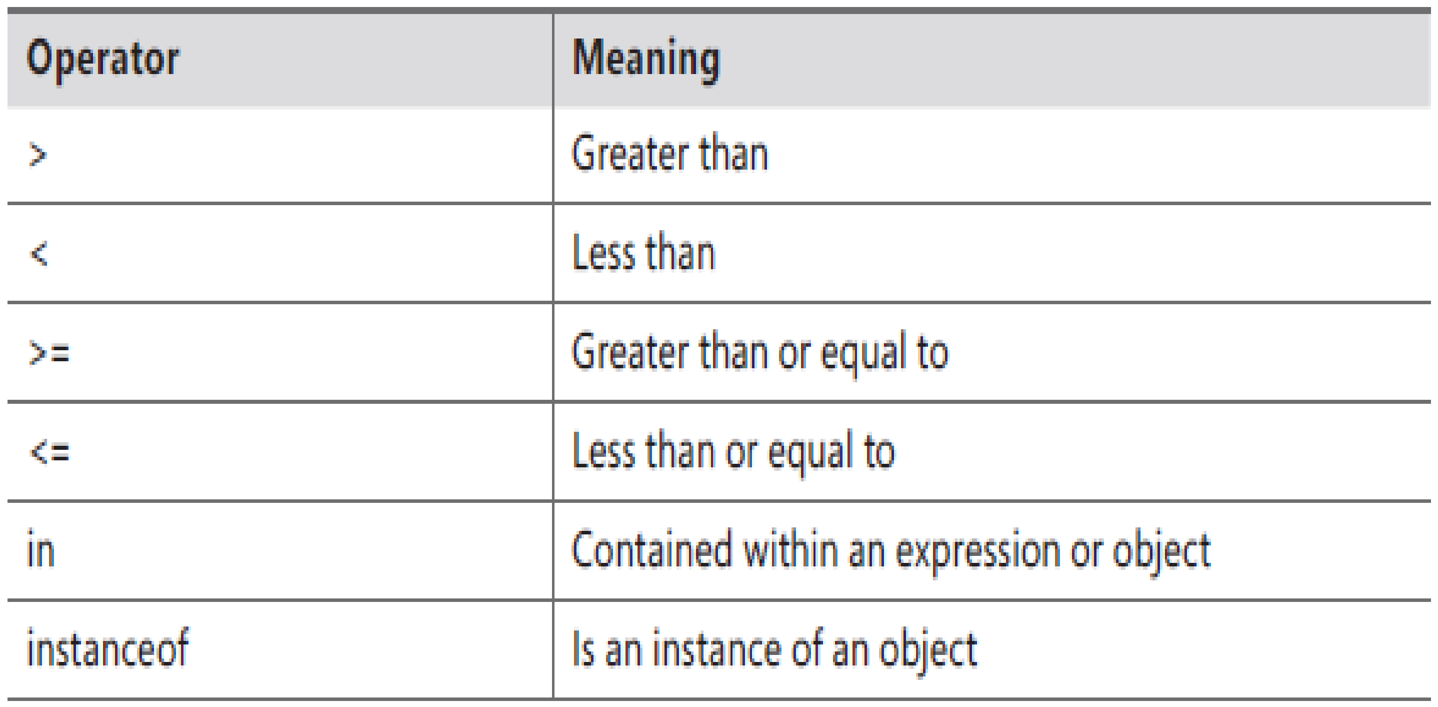
Operator	Meaning
&	AND
|	OR
^	XOR
~	NOT
<<	Shift Left
>>	Shift Right With Sign
>>>	Shift Right With Zero Fill

Equality operators



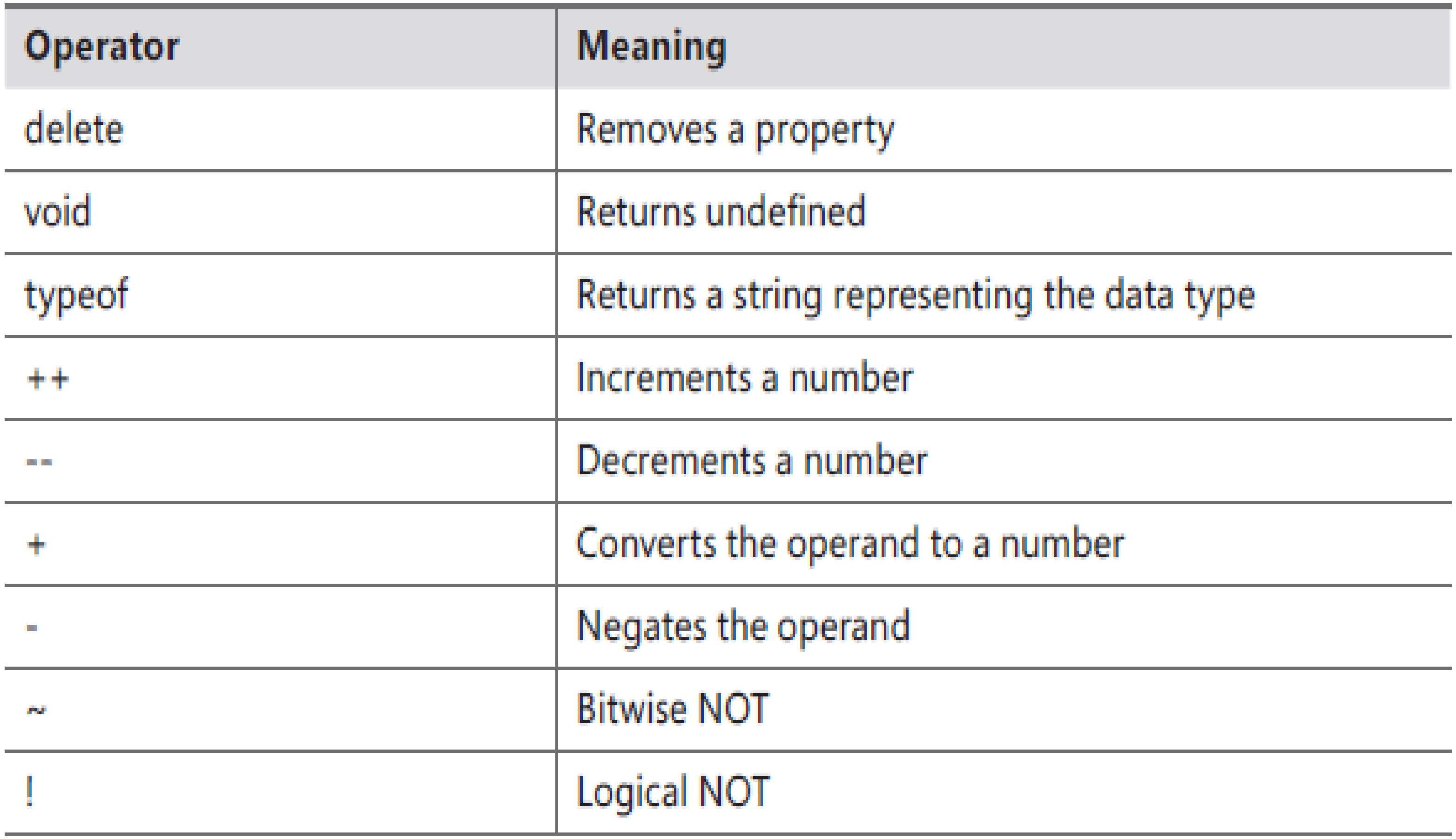
Operator	Meaning
==	Equal
!=	Not equal
===	Equal using stricter methods
!==	Not equal using stricter methods

Relational operators



Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
in	Contained within an expression or object
instanceof	Is an instance of an object

Unary operators



Operator	Meaning
delete	Removes a property
void	Returns undefined
typeof	Returns a string representing the data type
++	Increments a number
--	Decrements a number
+	Converts the operand to a number
-	Negates the operand
~	Bitwise NOT
!	Logical NOT

Conditional Statements

- Conditional Statements
- Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

```
if (condition) {
    block of code to be executed if the condition is true
}
```

Conditional Statements - cont

- Use the **else** statement to specify a block of code to be executed if the condition is false.

if (*condition*) {
    block of code to be executed if the condition is true
} else {
    block of code to be executed if the condition is false
}

Conditional Statements - cont

- Use the **else if** statement to specify a new condition if the first condition is false.

```
if (condition1) {
    block of code to be executed if condition1 is true
} else if (condition2) {
    block of code to be executed if the condition1 is false and
condition2 is true
} else {
    block of code to be executed if the condition1 is false and
condition2 is false
}
```

JavaScript Switch Statement

- Use the switch statement to select one of many blocks of code to be executed.

```
switch(expression) {
      case n:
         code block
         break;
      case n:
         code block
         break;
      default:
         default code block
   }
```

JavaScript For Loop

- JavaScript Loops

```
for (statement 1; statement 2; statement 3) {
    code block to be executed
}
```

```
var myArray = ["Vega","Deneb","Altair"];
var arrayLength = myArray.length;
for (var count = 0; count < arrayLength; count++ )
 {
alert(myArray[count]);
}
```

Javascript For/In Loop

- The JavaScript for/in statement loops through the properties of an object.

```
var person = {fname:"John", lname:"Doe", age:25};
var txt="";
var x;
for (x in person) {
                    txt += person[x] + " ";

               }
document.write(txt);
```

Javascript – While/(Do/While) loop

- Syntax

```
while (condition) {
        code block to be executed
}
```

- Syntax

```
do {
     code block to be executed
 }
 while (condition);
```

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- Function **parameters** are the **names** listed in the function definition.
- Function **arguments** are the real **values** received by the function when it is invoked.
- When JavaScript reaches a **return statement**, the function will stop executing.

Event	Description
onAbort	An image failed to load.
onBeforeUnload	The user is navigating away from a page.
onBlur	A form field lost the focus (User moved to another field)
onChange	The contents of a field has changed.
onClick	User clicked on this item.
onDblClick	User double-clicked on this item.
onError	An error occurred while loading an image.
onFocus	User just moved into this form element.
onKeyDown	A key was pressed.
onKeyPress	A key was pressed OR released.
onKeyUp	A key was released.
onLoad	This object (iframe, image, script) finished loading.
onMouseDown	A mouse button was pressed.
onMouseMove	The mouse moved.
onMouseOut	A mouse moved off of this element.
onMouseOver	The mouse moved over this element.
onMouseUp	The mouse button was released.
onReset	A form reset button was pressed.
onResize	The window or frame was resized.
onSelect	Text has been selected.
onSubmit	A form's Submit button has been pressed.
onUnload	The user is navigating away from a page.

What is the DOM?

- The DOM is a W3C (World Wide Web Consortium) standard.
- *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.“*
- The W3C DOM standard is separated into 3 different parts.

- Core DOM - standard model for all document types.

- XML DOM - standard model for XML documents.

- HTML DOM - standard model for HTML documents

What is the HTML DOM

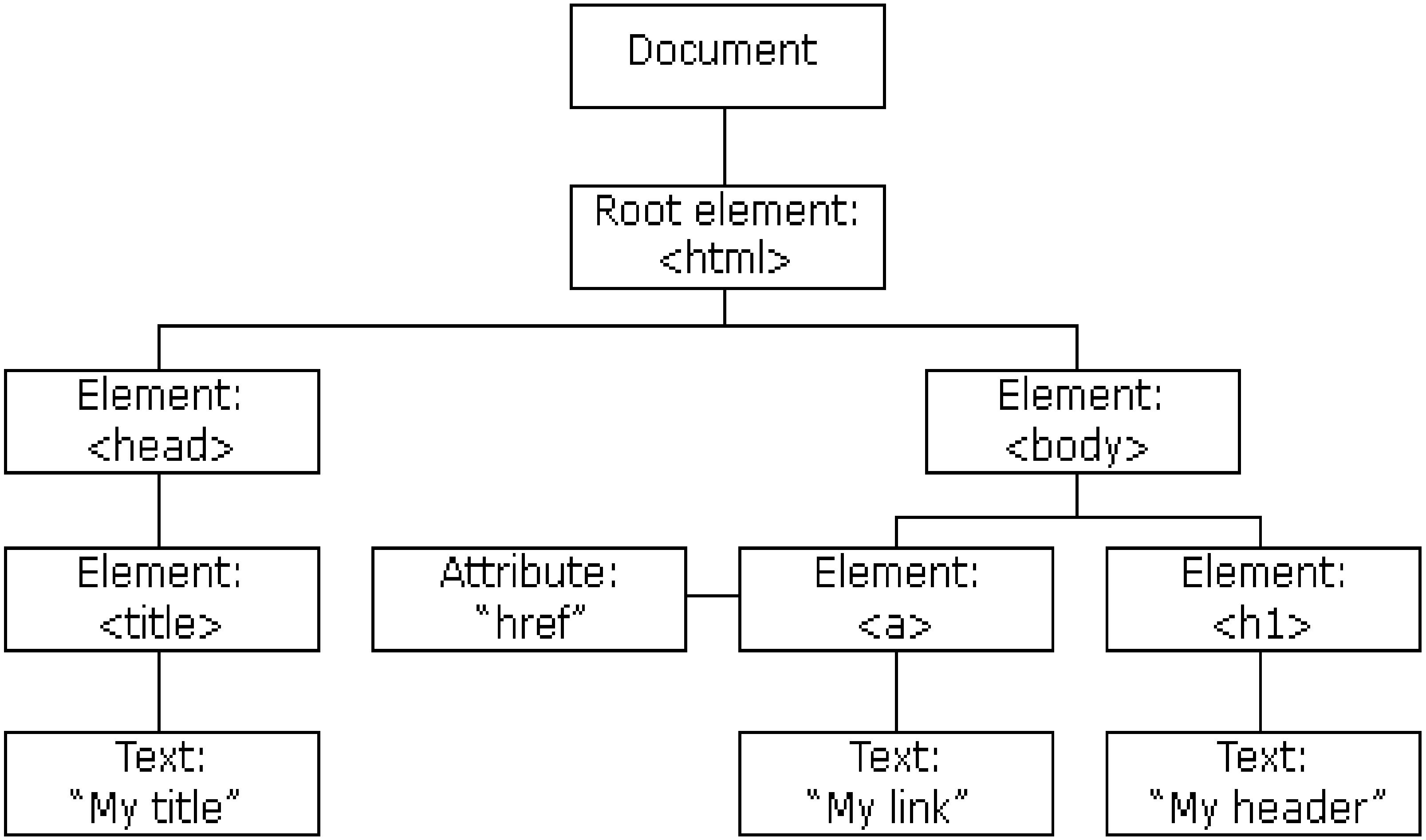
- The HTML DOM is a standard **object** model and **programming interface** for HTML.
- The HTML elements as **objects.**
- The **properties** of all HTML elements.
- The **methods** to access all HTML elements.
- The **events** for all HTML elements.

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

DOM Programming Interface

- In the DOM, all HTML elements are defined as **objects**.
- A **property** is a value that you can get or set (like changing the content of an HTML element).
- A **method** is an action you can do (like add or deleting an HTML element).

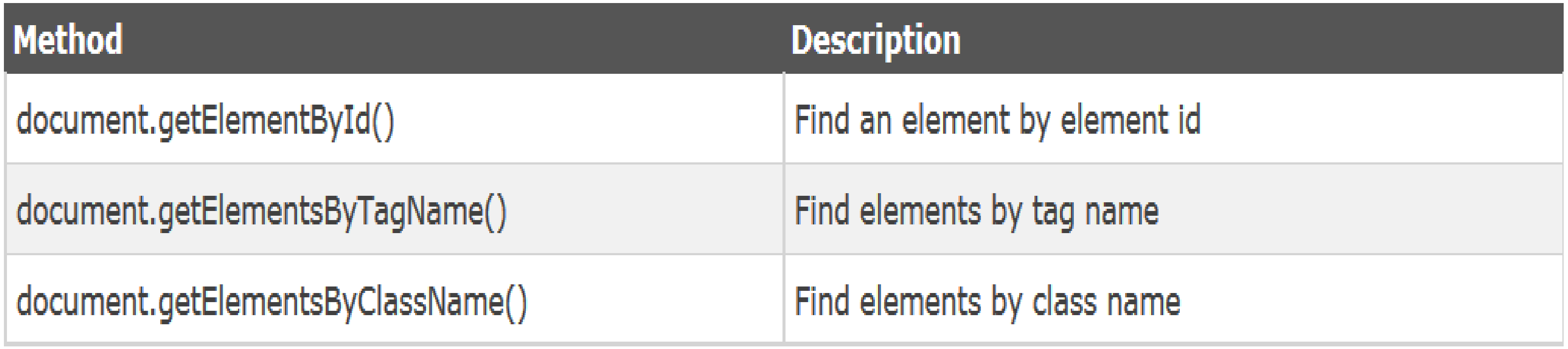
JavaScript HTML DOM



HTML DOM Document

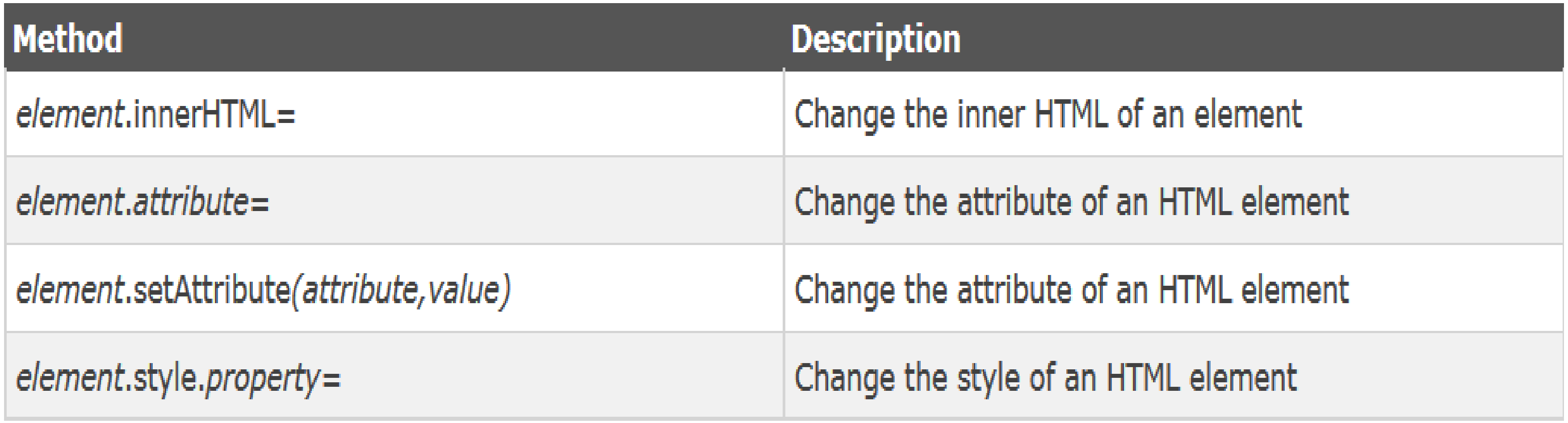
- HTML DOM object model, the document object represents your web page.
- Document object is the owner of all other objects in your web page.
- Always start with accessing the document object to access objects in an HTML page.

Finding HTML Elements



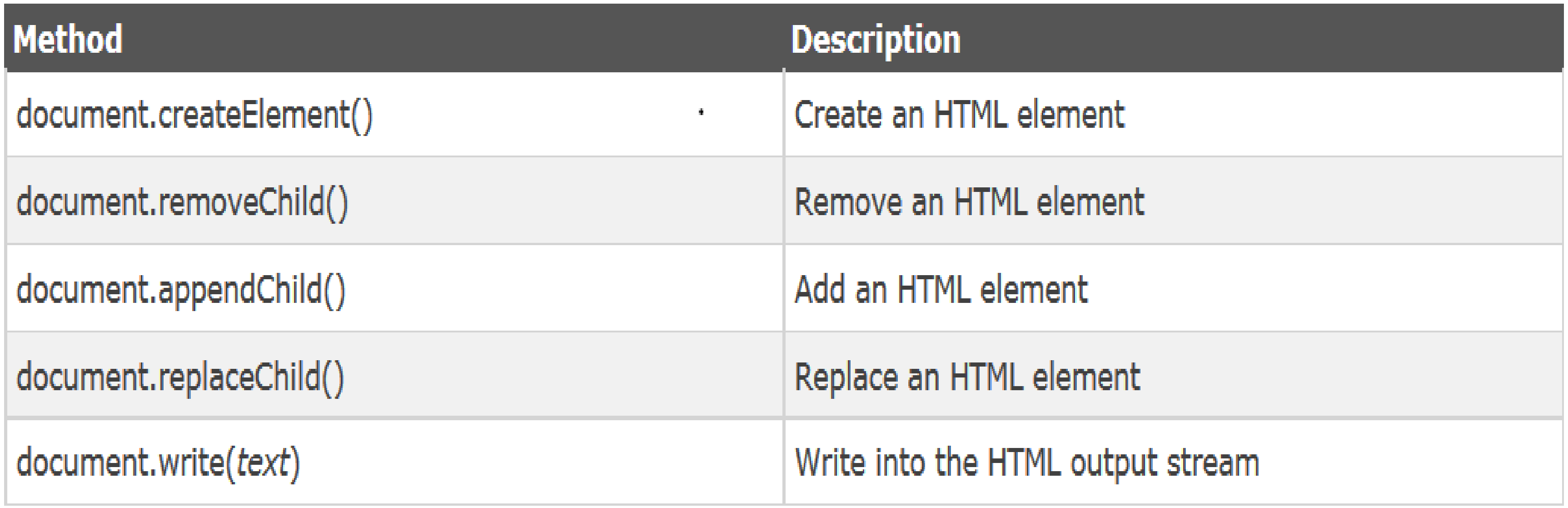
Method	Description
document.getElementById()	Find an element by element id
document.getElementsByTagName()	Find elements by tag name
document.getElementsByClassName()	Find elements by class name

Changing HTML Elements



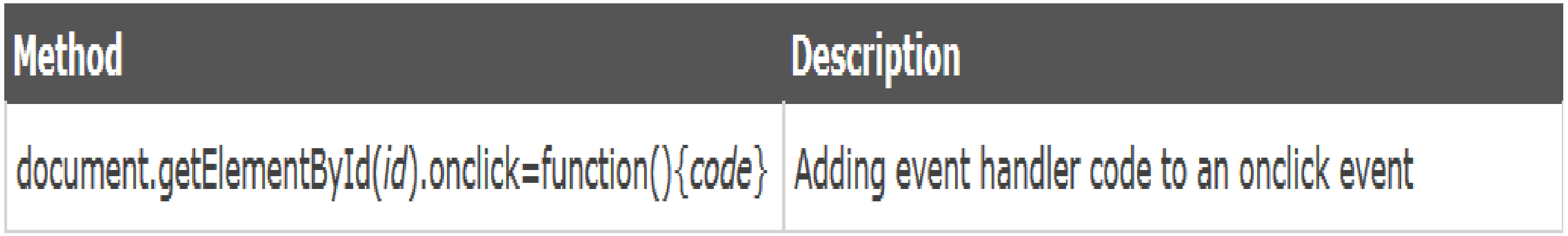
Method	Description
element.innerHTML=	Change the inner HTML of an element
element.attribute=	Change the attribute of an HTML element
element.setAttribute*(attribute,value)*	Change the attribute of an HTML element
element.style.*property*=	Change the style of an HTML element

Adding and Deleting Elements



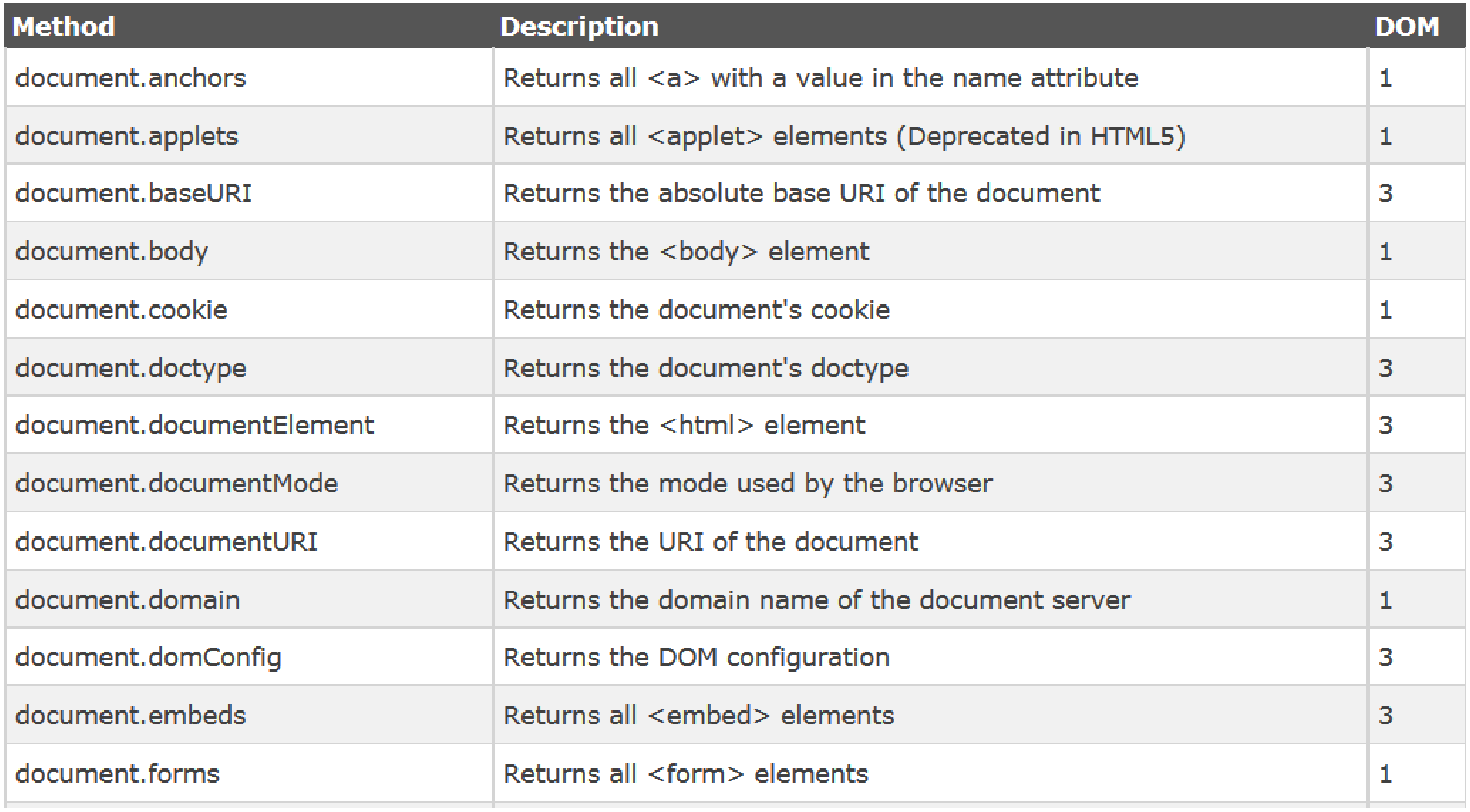
Method	Description
document.createElement()	Create an HTML element
document.removeChild()	Remove an HTML element
document.appendChild()	Add an HTML element
document.replaceChild()	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers



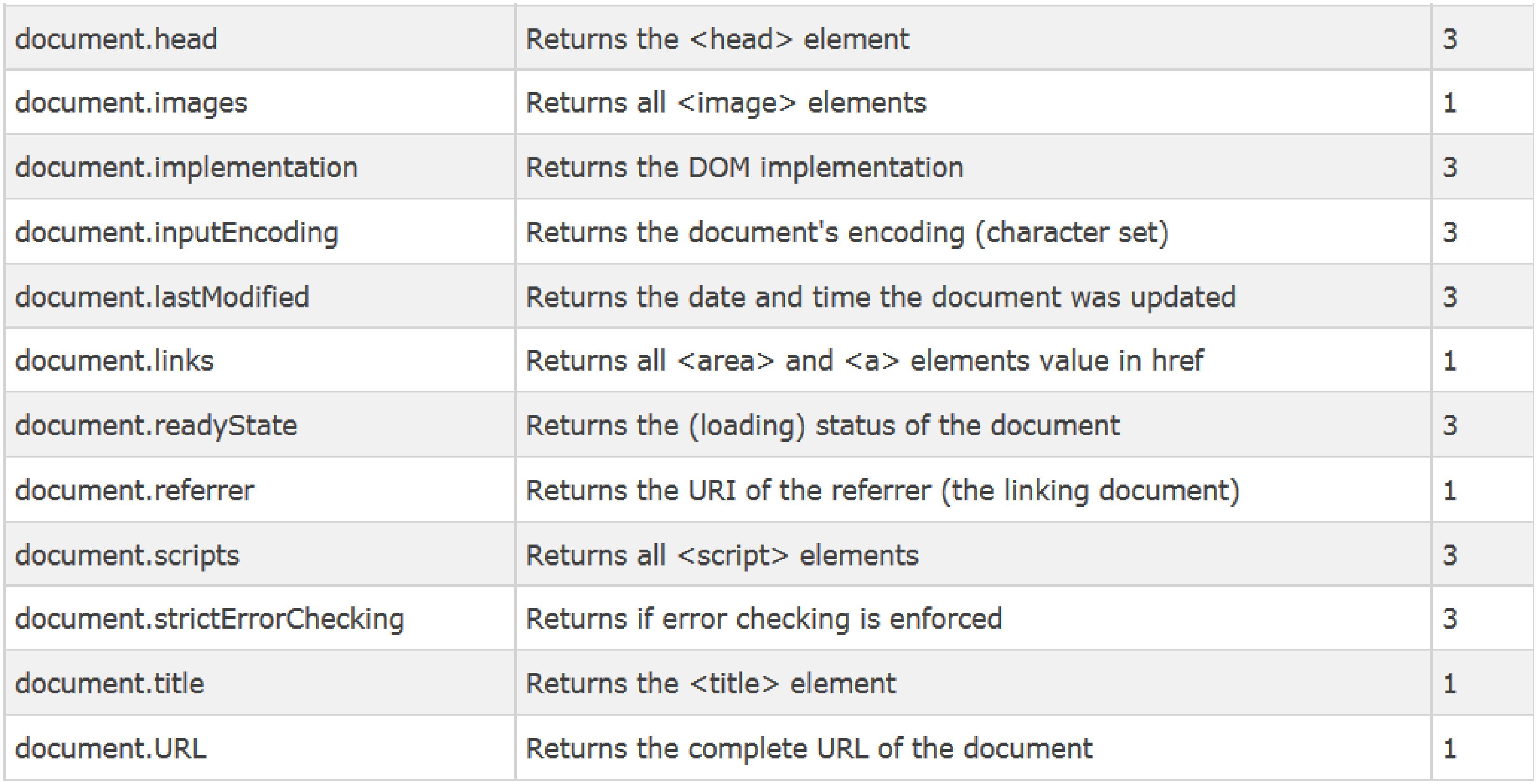
Method	Description
document.getElementById(*id*).onclick=function(){*code*}	Adding event handler code to an onclick event

Finding HTML Objects



Method	Description	DOM
document.anchors	Returns all <a> with a value in the name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1

Finding HTML Objects - cont



document.head	Returns the <head> element	3
document.images	Returns all <image> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements value in href	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

JavaScript Window - The Browser Object Model

- The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.
- The **window** object is supported by all browsers. It represent the browser's window.
- Even the document object (of the HTML DOM) is a property of the window object.

```
window.document.getElementById("header");
```

Same as

```
document.getElementById("header");
```

JavaScript Window Size

window.innerHeight - the inner height of the browser window

window.innerWidth - the inner width of the browser window

- window.open() - open a new window
- window.close() - close the current window
- window.moveTo() -move the current window
- window.resizeTo() -resize the current window

JavaScript Window Screen

- The **window.screen** object can be written without the window prefix.
- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth
- screen.pixelDepth

Javascript Window Location

- location.href returns the href (URL) of the current page
- location.hostname returns the domain name of the web host
- location.pathname returns the path and filename of the current page
- location.protocol returns the web protocol used (http:// or https://)
- location.assign loads a new document

JavaScript Window History

- The **window.history** object can be written without the window prefix.
- To protect the privacy of the users, there are limitations to how JavaScript can access this object.
- history.back() - same as clicking back in the browser
- history.forward() - same as clicking forward in the browser

JavaScript Window Navigator

- The window.navigator object contains information about the visitor's browser.
- The **window.navigator** object can be written without the window prefix.

Examples

navigator.appName

navigator.appCodeName

navigator.platform

JavaScript Popup Boxes

- JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

window.alert("*sometext*");

Confirm Box

window.confirm("*sometext*");

Prompt Box

window.prompt("*sometext*","*defaultText*");

JavaScript Timing Events

- With JavaScript, it is possible to execute some code at specified time-intervals. This is called timing events.
- setInterval() - executes a function, over and over again, at specified time intervals
- setTimeout() - executes a function, once, after waiting a specified number of milliseconds

JavaScript Timing Events – cont
The setInterval() Method

- The setInterval() method will wait a specified number of milliseconds, and then execute a specified function, and it will continue to execute the function, once at every given time-interval.
- The **window.setInterval()** method can be written without the window prefix.
- The first parameter of setInterval() should be a function.
- The second parameter indicates the length of the time-intervals between each execution.

Syntax:

window.setInterval("*javascript function*", *milliseconds*);

JavaScript Timing Events – cont
The setTimeout() Method

- The **window.setTimeout()** method can be written without the window prefix.
- The setTimeout() method will wait the specified number of milliseconds, and then execute the specified function.
- The first parameter of setTimeout() should be a function.
- The second parameter indicates how many milliseconds, from now, you want to execute the first parameter.

Syntax:

window.setTimeout("*javascript function*", *milliseconds*);

JavaScript Timing Events – Stop the execution

- The clearTimeout() method is used to stop the execution of the function specified in the setTimeout() method.
- The **window.clearTimeout()** method can be written without the window prefix.
- To be able to use the clearTimeout() method, you must use a global variable when creating the timeout method.

Syntax:

window.clearTimeout(*timeoutVariable*)

Thank You!