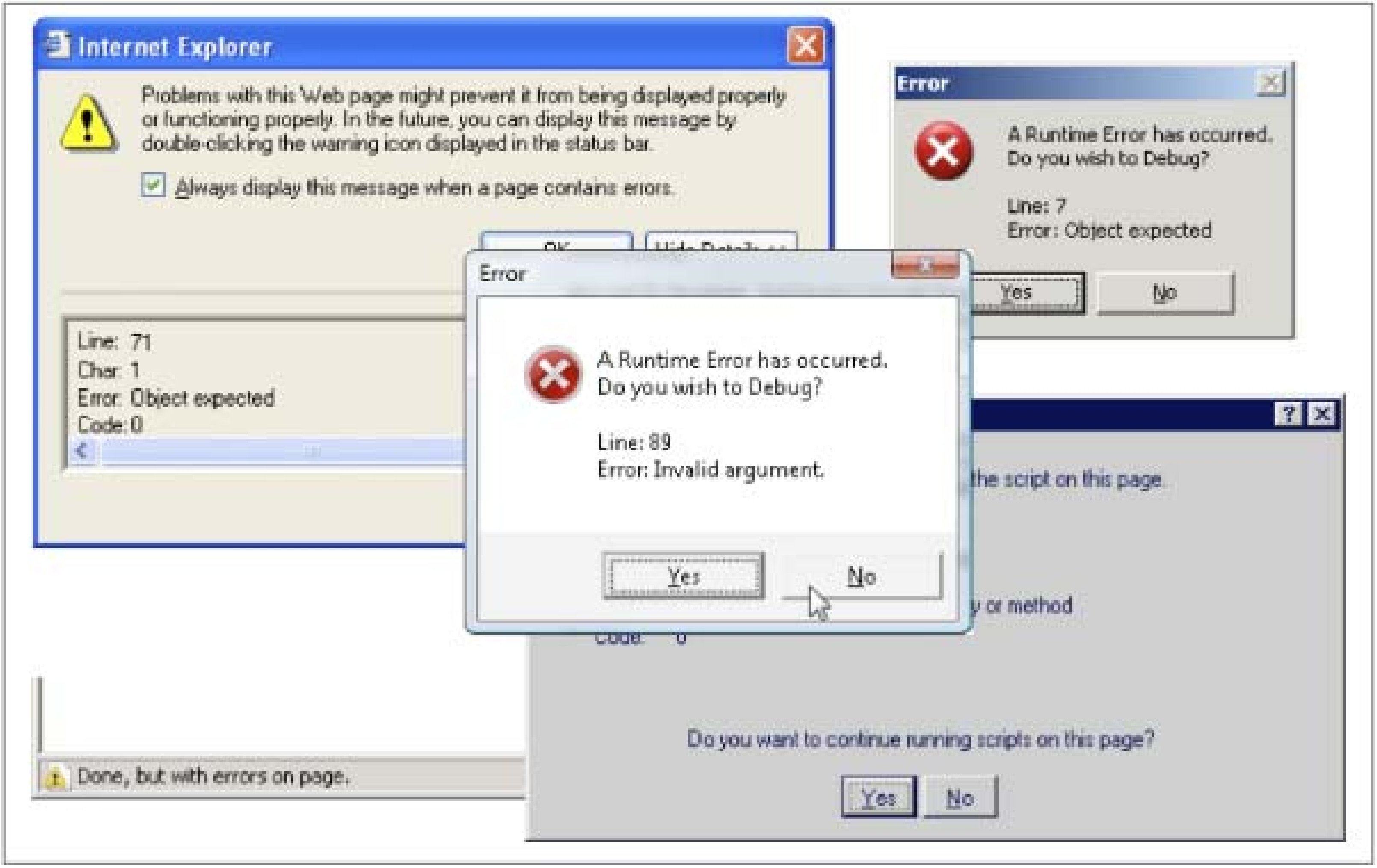
Chapter 6.1 – Java Script

HISTORY

- JavaScript has had a bad reputation for years.
- Many developers consider writing in JavaScript a pain because the programs behave so unpredictably.
- Once done coding, they open another browser to test, only to be greeted with an unhelpful error message

Some errors


Internet Explorer

Problems with this Web page might prevent it from being displayed properly or functioning properly. In the future, you can display this message by double-clicking the warning icon displayed in the status bar.

☑ Always display this message when a page contains errors.

Line: 71
Char: 1
Error: Object expected
Code: 0

Done, but with errors on page.

Error

A Runtime Error has occurred.
Do you wish to Debug?

Line: 7
Error: Object expected

Yes No

Error

A Runtime Error has occurred.
Do you wish to Debug?

Line: 89
Error: Invalid argument.

Yes No

the script on this page

y or method

Code: 0

Do you want to continue running scripts on this page?

Yes No

INTRODUCTION

- JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages.
- All modern HTML pages are using JavaScript.
- JavaScript can change HTML content, attributes, CSS, and can validate the data as well.
- JavaScript and Java are completely different languages, both in concept and design.
- JavaScript was invented by Brendan Eich in 1995, and became an ECMA(**European Computer Manufacturers Association)** standard in 1997.
- ECMA-262 is the official name. ECMAScript 5 (JavaScript 1.8.5 - July 2010) is the latest standard.

INTRODUCTION

- JavaScript is:
  - JavaScript is a lightweight, interpreted programming language
  - Designed for creating network-centric applications
  - Complementary to and integrated with Java
  - Complementary to and integrated with HTML
  - Open and cross-platform

Why Study JavaScript?

- JavaScript is one of the **3 languages** all web developers **must** learn:
  - **HTML** to define the content of web pages
  - **CSS** to specify the layout of web pages
  - **JavaScript** to program the behavior of web pages

Client-side JavaScript

- Client-side JavaScript is the most common form of the language.
- The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- The JavaScript code is executed when the user submits the form, and only if all the entries are valid they would be submitted to the Web Server.
- JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user explicitly or implicitly initiates.

Advantages of JavaScript

- **Less server interaction:** You can validate user input before sending the page off to the server.
- This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces:** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations with JavaScript

- We can not treat JavaScript as a full fledged programming language. It lacks the following important features:
- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript can not be used for Networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multi-process capabilities.
- Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Syntax

- JavaScript statements are separated by **semicolon**.
- JavaScript statements are composed of:
  - Values, Operators, Expressions, Keywords, and Comments.
- The JavaScript syntax defines two types of values: Fixed values and variable values.
- Fixed values are called **literals**. Variable values are called **variables**.
- **Numbers** are written with or without decimals: 10.50 1001
- **Strings** are text, written within double or single quotes: “abc“, ‘abc'
- **Expressions** can also represent fixed values: 5+6, 5 * 10

Case Sensitivity

- JavaScript is a case-sensitive language.
- This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.
- So identifiers *Time*, *TIme* and *TIME* will have different meanings in JavaScript.

JavaScript Syntax

- A JavaScript consists of JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.
- You can place the <script> tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the <head> tags.
- The <script> tag alerts the browser program to begin interpreting all the text between these tags as a script.
- So simple syntax of your JavaScript will be as follows:

```
<script ...> JavaScript code </script>
```

JAVASCRIPT SYNTAX

- The script tag takes two important attributes:
- **language:** This attribute specifies what scripting language you are using. Typically, its value will be *javascript*.
- Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to *"text/JavaScript"*.
- <script langage="javascript" type="text/javascript">

  JavaScript code </script>

Your First JavaScript Script:

- **first.php**

```
<html><body>
        <script   language="javascript"   type="text/javascript">
        <!-- document.write("Hello World!") //-->
        </script>
</body> </html>
```

The <script> Tag

- In HTML, JavaScript code must be inserted between <script> and </script> tags.
- **Example**

-
  ```
  <script>
  document.getElementById("demo").innerHTML = "My First JavaScript";
  </script>
  ```

JavaScript Functions and Events

- A JavaScript **function** is a block of JavaScript code, that can be executed when "asked" for.
- For example, a function can be executed when an **event** occurs, like when the user clicks a button.
- You can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JavaScript in <head>

- **Example (third.php)**

```
<html><head>
    <script>
    function myFunction() {
        document.getElementById("demo").innerHTML = "Paragraph
    changed.";
    }
    </script>
    </head>
<body>
<h1>My Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

JAVASCRIPT IN <BODY>

- **Example (forth.php)**
-
```
<html>
<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph
changed.";
}
</script>
</body>
</html>
```

External JavaScript

- Scripts can also be placed in external files.
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the **file extension .js**.
- To use an external script, put the name of the script file in the src (source) attribute of the <script> tag:
- **Example**
-
```
<html>
<body>
<script src="myScript.js"></script>
</body>
</html>
```

External JavaScript

- Advantages
  - Placing JavaScript in external files has some advantages:
  - It separates HTML and code.
  - It makes HTML and JavaScript easier to read and maintain Cached JavaScript files can speed up page loads.

Warning for Non-JavaScript Browsers

- If you have to do something important using JavaScript then you can display a warning message to the user using <noscript> tags.
- You can add a *noscript* block immediately after the script block as follows: **(second.php)**
- <html> <body>

```
        <script language="javascript" type="text/javascript">
        <!–
                document.write("Hello World!")
        //-->
        </script>
        <noscript> Sorry...JavaScript is needed to go ahead.
        </noscript>
</body> </html>
```

The Document Object

- When an HTML document is loaded into a web browser, it becomes a **document object**.
- The document object is the root node of the HTML document and the "owner" of all other nodes.
- The document object provides properties and methods to access all node objects, from within JavaScript.

WRITE() METHOD

- The write() method writes HTML expressions or JavaScript code to a document.
- The write() method is mostly used for testing: If it is used after an HTML document is fully loaded, it will delete all existing HTML.
- When this method is not used for testing, it is often used to write some text to an output stream opened by the document.open() method.
- doc6.php

HTML DOM getElementById() Method

- The getElementById() method returns the element that has the ID attribute with the specified value.
- This method is used almost every time you want to manipulate, or get info from, an element on your document.
- Returns *null* if no elements with the specified ID exists.
- An ID should be unique within a page.
- However, if more than one element with the specified ID exists, the getElementById() method returns the first element in the source code.
- doc1.php

getElementsByClassName() Method

- The getElementsByClassName() method returns a collection of all elements in the document with the specified class name, as a NodeList object.
  - Returns a NodeList object, representing a collection of elements with the specified class name.
  - The nodes can be accessed by index numbers. The index starts at 0.
  - The elements in the returned collection are sorted as they appear in the source code.
- doc2.php

getElementsByTagName() Method

- The getElementsByTagName() method returns a collection of all elements in the document with the specified tag name, as a NodeList object.
- The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.
- The parametervalue "*" returns all elements in the document.
- You can use the length property of the NodeList object to determine the number of elements with the specified tag name, then you can loop through all elements and extract the info you want.
- doc4.php

getElementsByName() Method

- The getElementsByName() method returns a collection of all elements in the document with the specified name (the **value** of the name attribute), as a NodeList object.
  - The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.
  - The elements in the returned collection are sorted as they appear in the source code.
- doc3.php

HTML DOM readyState Property

- The readyState property returns the (loading) status of the current document.
- Return Value: A String, representing the status of the current document. One of five values:
  - uninitialized - Has not started loading yet
  - loading - Is loading
  - loaded - Has been loaded
  - interactive - Has loaded enough and the user can interact with it
  - complete - Fully loaded
- doc5.php

JAVASCRIPT OUTPUT

- JavaScript Display Possibilities
  - JavaScript can "display" data in different ways:
  - Writing into an alert box, using **window.alert()**.
  - Writing into the HTML output using **document.write()**.
  - Writing into an HTML element, using **innerHTML**.
  - Writing into the browser console, using **console.log()**.

Using window.alert()

- You can use an alert box to display data: (ALERT.php, alert1.php, alert2.php)
- **Example**
-
```
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

USING DOCUMENT.WRITE()

- For testing purposes, it is convenient to use **document.write(): (write.php, write1.php)**
- **Example**
-
```
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Using document.write()

- Using document.write() after an HTML document is fully loaded, will **delete all existing HTML**:
- **Example**
-
```
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Using innerHTML

- The innerHTML property sets or returns the HTML content (inner HTML) of an element.
- The **innerHTML** property can be used along with getElementById() within your **JavaScript** code to refer to an HTML element and change its contents.
- Return the innerHTML property (string, HTML content of an element:
  - *HTMLElementObject*.innerHTML
- Set the innerHTML property:
  - document.getElementById('{ID of element}').innerHTML = '{content}';
- Example : (innerHTML.html, innerHTML.php, innerHTML1.php)

JavaScript Comments

- JavaScript supports both C-style and C++-style comments, Thus:
- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Using console.log()

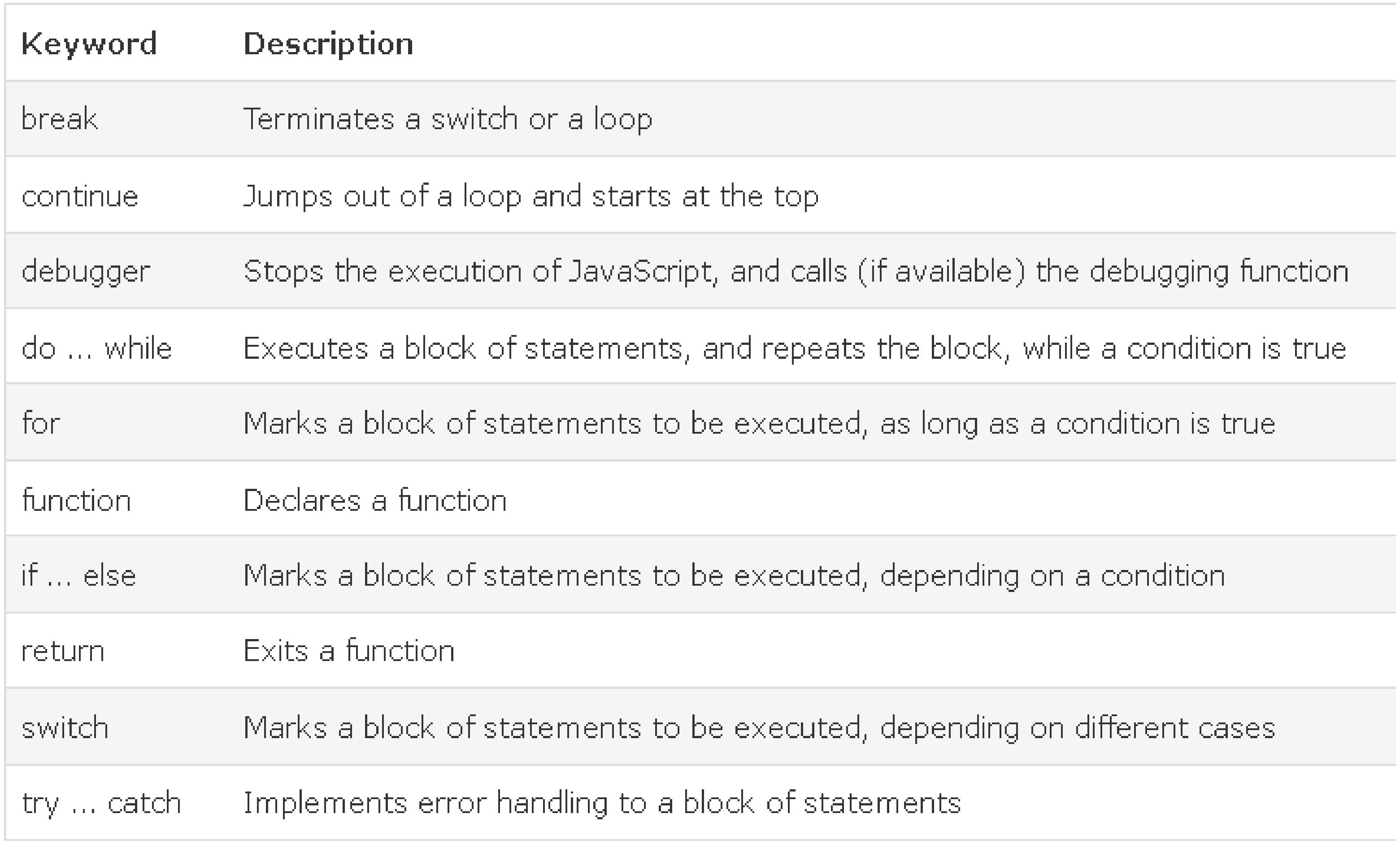
- In your browser, you can use the **console.log()** method to display data.
- Activate the browser console with F12, and select "Console" in the menu.
- log.php
- **Example:**
-
```
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

JavaScript White Space

- JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.
- The following lines are equivalent:
- var person = "Hege";
  var person="Hege";

JavaScript Keywords

- JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.



Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements

Declaring (Creating) JavaScript Variables

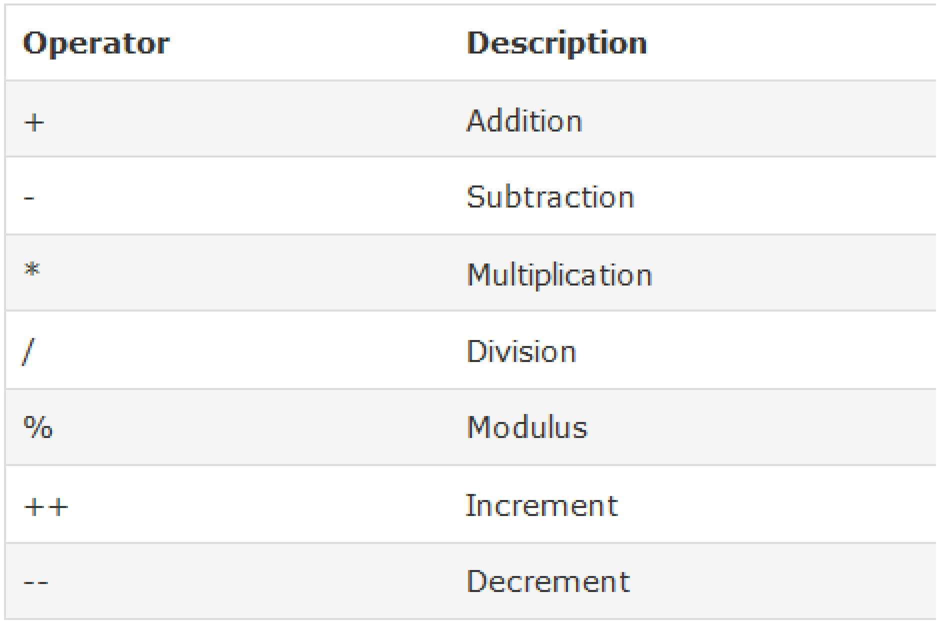
- Creating a variable in JavaScript is called "declaring" a variable.
- You declare a JavaScript variable with the **var** keyword:
  - var carName;
- To **assign** a value to the variable, use the equal sign:
  - carName = "Volvo";
- You can also assign a value to the variable when you declare it:
  - var carName = "Volvo";

Declaring (Creating) JavaScript Variables

- **Example (1.php)**
- <html> <body>
- <h1>JavaScript Variables</h1>
- <p>Create a variable, assign a value to it, and display it:</p>
- <p id="demo"></p>
- <script>var carName = "Volvo";
- document.getElementById("demo").innerHTML = carName;
- </script></body></html>

JavaScript Arithmetic Operators

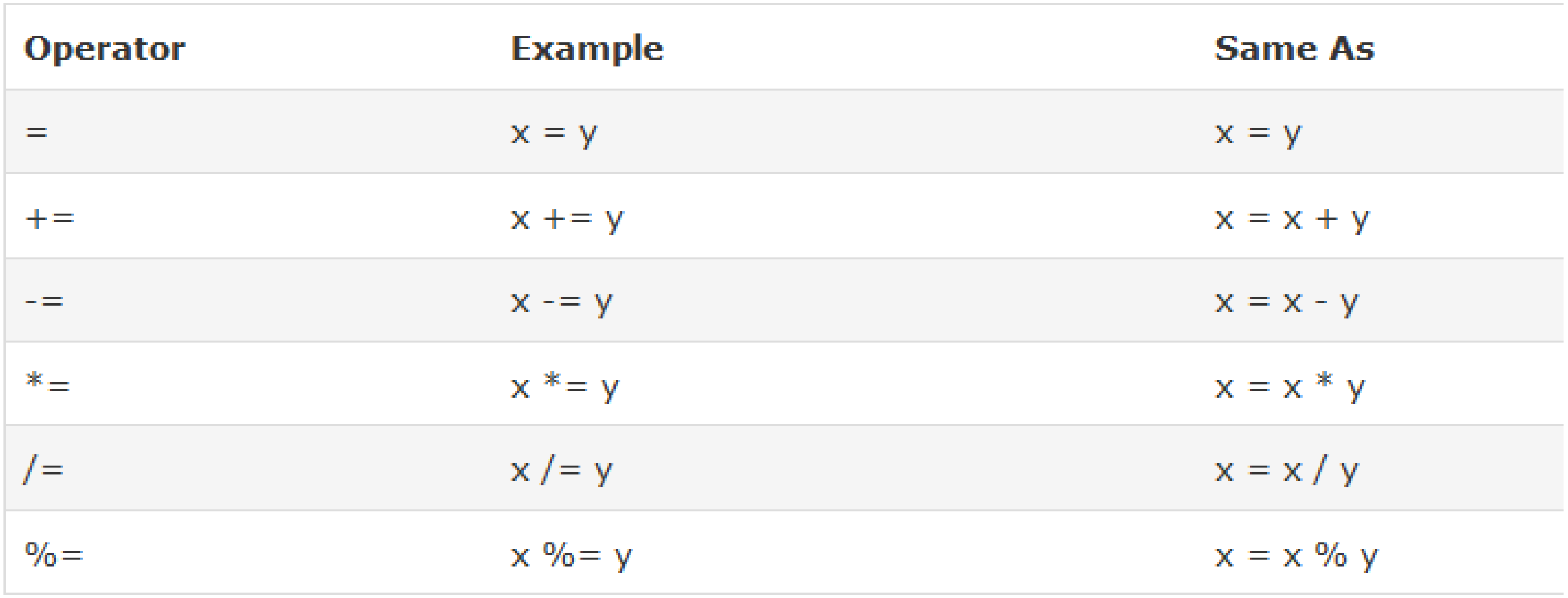
- Arithmetic operators are used to perform arithmetic between variables and/or values.



Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

JavaScript Assignment Operators

- Assignment operators are used to assign values to JavaScript variables.



Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

JavaScript Data Types

- JavaScript variables can hold many **data types**: numbers, strings, arrays, objects and more:

-
  ```
  var length = 16;                                    // Number
  var lastName = "Johnson";                           // String
  var cars = ["Saab", "Volvo", "BMW"];                // Array
  var x = {firstName:"John", lastName:"Doe"};         // Object
  ```

JavaScript Has Dynamic Types

- JavaScript has dynamic types. This means that the same variable can be used as different types:
- **Example**
- var x; // Now x is undefined
  var x = 5; // Now x is a Number
  var x = "John"; // Now x is a String

JavaScript Arrays

- You can create an array using either
  - an array initializer (array literal) or
  - the Array constructor.
- The array initializer (array literal) syntax is simple: a comma-separated list of values in square brackets. Here are some examples:
  - var myArray1 = [1,3,5,7,9] // an array with 5 elements var
  - myArray2 = [5] // an array with 1 element
  - var myArray3 = [true,'Hi',[7]] Array items are separated by commas.
- Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

JavaScript Arrays

- The Array constructor syntax has three different forms.
- If you call the constructor with two or more arguments, the arguments initialize the array elements.
- If you only supply one argument to the Array constructor, the argument initializes the *length* of the new array; the new array's elements are not initialized.
- Finally, if you call the constructor without arguments, the new array's length is set to zero, and its elements are not initialized.
- Here are examples:
- var myArray4 = new Array(1,3,5,7,9) // an array with 5 elements
- var myArray5 = new Array(100)   // an empty array of length 100
- var myArray6 = new Array()   // an empty array of length 0

Array Properties and Methods

- The **length** property of an array returns the length of an array (the number of array elements).
- The easiest way to add a new element to an array is to use the length property.
- Adding elements with high indexes can create undefined "holes" in an array.
- **array1.php, array2.php**

Arrays and objects

- The Difference Between Arrays and Objects?
  - In JavaScript, **arrays** use **numbered indexes**.
  - In JavaScript, **objects** use **named indexes**.
- When to Use Arrays? When to use Objects?
- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

Arrays and objects

- Avoid new Array()
- There is no need to use the JavaScript's built-in array constructor **new** Array().
- **Use [] instead.**
- These two different statements both create a new empty array named points:
- var points = new Array();   // Bad
  var points = [];   // Good
- These two different statements both create a new array containing 6 numbers:
- var points = new Array(40, 100, 1, 5, 25, 10) // Bad
  var points = [40, 100, 1, 5, 25, 10];   // Good
- To identify between array and object use typeof.

JAVASCRIPT OBJECTS

- JavaScript objects are written with curly braces.
- Object properties are written as name:value pairs, separated by commas.
- **Example (objVar.php)**

```
<html><body>
<p id="demo"></p>
<script>
var person = {
    firstName : "John",
    lastName  : "Doe",
    age       : 50,
    eyeColor  : "blue"
};
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body></html>
```

The typeof Operator

Example (typeof.php)

```
<html> <body>
<p>The typeof operator returns the type of a variable or an expression.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
typeof "john" + "<br>" +
typeof 3.14 + "<br>" +
typeof false + "<br>" +
typeof [1,2,3,4] + "<br>" +
typeof {name: 'john', age:34};
</script>
</body></html>
```

JavaScript Variable Scope

- The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.
- **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.
- Within the body of a function, a local variable takes precedence over a global variable with the same name.

JavaScript Variable Scope

- If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable.
- Following example explains it:

```
<script type="text/ Javascript">
<!–
var myVar = "global"; // Declare a global variable function
checkscope( ) {
        var myVar = "local"; // Declare a local variable
        document.write(myVar);
}
//-->
 </script>
```

Automatically Global

- If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.
- This code example will declare carName as a global variable, even if it is executed inside a function. **(global.php)**

```
<html><body>
<p>If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable:</p>
<p id="demo"></p>
<script>
myFunction();
document.getElementById("demo").innerHTML =
"I can display " + carName;
function myFunction() {
    carName = "Volvo";
}
</script>  </body> </html>
```

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: ***(parameter1, parameter2, ...)***
- *functionName*(*parameter1, parameter2, parameter3*) {
  code to be executed
  }
- Inside the function, the arguments are used as local variables.

Function Invocation and Return

- The code inside the function will execute when "something" **invokes** (calls) the function:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

- P.S. In JavaScript, there is a Function constructor also. Function() is a constructor that creates a Function object. E.g. var setBGColor = **new** Function(document.bgColor='blue');

Functions Used as Variables

- Accessing a function without (), will return the function definition. **(f2.php)**
- In JavaScript, functions can be used as variables. **(f4.php)**

```
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"The temperature is " + toCelsius(32) + " Centigrade";

function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
</script>
</body> </html>
```

JavaScript Events

- JavaScript's interaction with HTML is handled through events that occur when the user or browser manipulates a page.
- E.g. When the page loads, clicking a button, pressing any key, closing window, resizing window etc.
- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable to occur.
- Events are a part of the Document Object Model (DOM) Level 3 and every HTML element has certain set of events which can trigger JavaScript Code.

JavaScript Events

- HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.
- With single quotes:

  <*some-HTML-element some-event*='***some JavaScript***'>
- With double quotes:

  <*some-HTML-element some-event*="***some JavaScript***">
- **Example: (click1.php)**

```
<html> <body>
<button
onclick="getElementById('demo').innerHTML=Date()">The time
is?</button>
<p id="demo"></p>
</body>
</html>
```

JavaScript Events

- Example with a change: **(click2.php)**

```
<html>
<body>

<button onclick="this.innerHTML=Date()">The time is?</button>

</body>
</html>
```

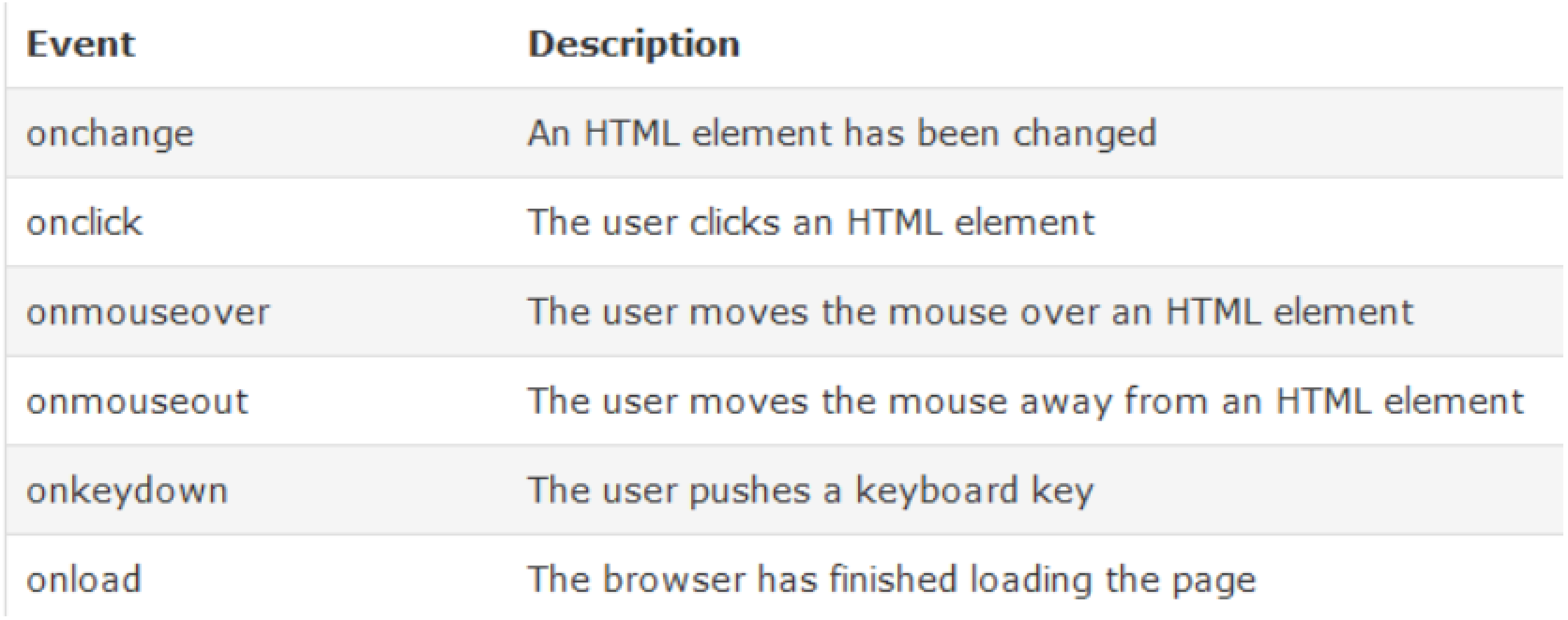
Types of events

- There are many types of events.
- DOM events, which are initiated by DOM-elements. For instance, a click event happens when an element is clicked, a mouseover - when a mouse pointer comes over an element,
- Window events. For instance, resize - when a browser window is resized,
- Other events, like load, readystatechange. They are used in AJAX and for other needs.
- DOM events connect JavaScript code with the document, providing the means for building dynamical interfaces.

HTML Events

- Most JavaScript-applications perform actions as a response to *events*.
- An *event* is a signal from the browser that something has happened.
- An HTML event can be something the browser does, or something a user does.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

Common HTML Events

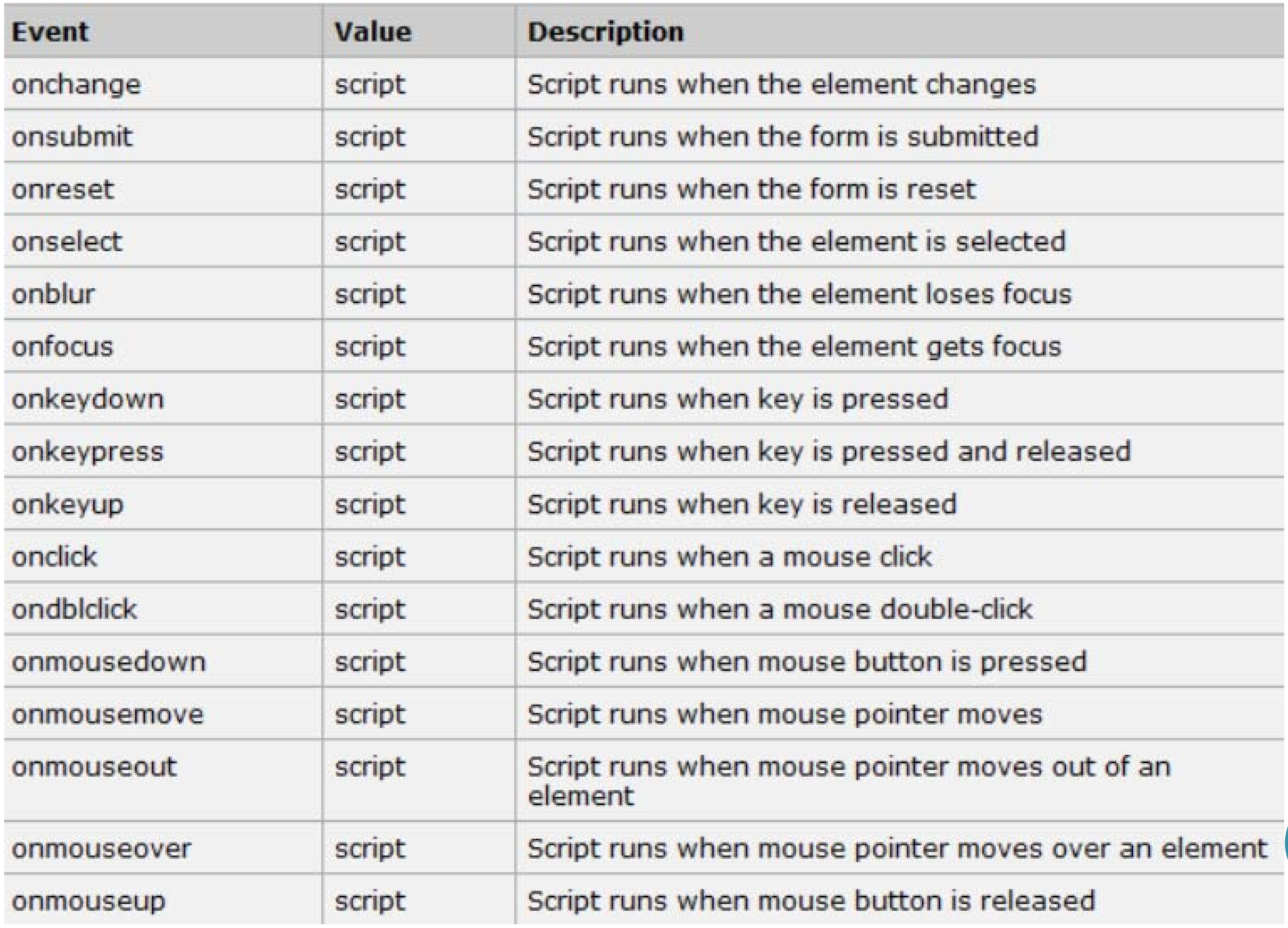


Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

What can JavaScript Do?

- Event handlers can be used to handle, and verify, user input, user actions, and browser actions:
- Things that should be done every time a page loads or closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user input data and more ...
- Many different methods can be used to let JavaScript work with events:
  - HTML event attributes can execute JavaScript code directly
  - HTML event attributes can call JavaScript functions
  - You can assign your own event handler functions to HTML elements
  - You can prevent events from being sent or being handled
  - And more ...

HTML 4 Standard Events



Event	Value	Description
onchange	script	Script runs when the element changes
onsubmit	script	Script runs when the form is submitted
onreset	script	Script runs when the form is reset
onselect	script	Script runs when the element is selected
onblur	script	Script runs when the element loses focus
onfocus	script	Script runs when the element gets focus
onkeydown	script	Script runs when key is pressed
onkeypress	script	Script runs when key is pressed and released
onkeyup	script	Script runs when key is released
onclick	script	Script runs when a mouse click
ondblclick	script	Script runs when a mouse double-click
onmousedown	script	Script runs when mouse button is pressed
onmousemove	script	Script runs when mouse pointer moves
onmouseout	script	Script runs when mouse pointer moves out of an element
onmouseover	script	Script runs when mouse pointer moves over an element
onmouseup	script	Script runs when mouse button is released

Displaying Dates

```
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = Date();
</script>
</body>
</html>
```

- The script above says: assign the value of Date() to the content (innerHTML) of the element with id="demo". **(date1.php)**

Displaying Dates

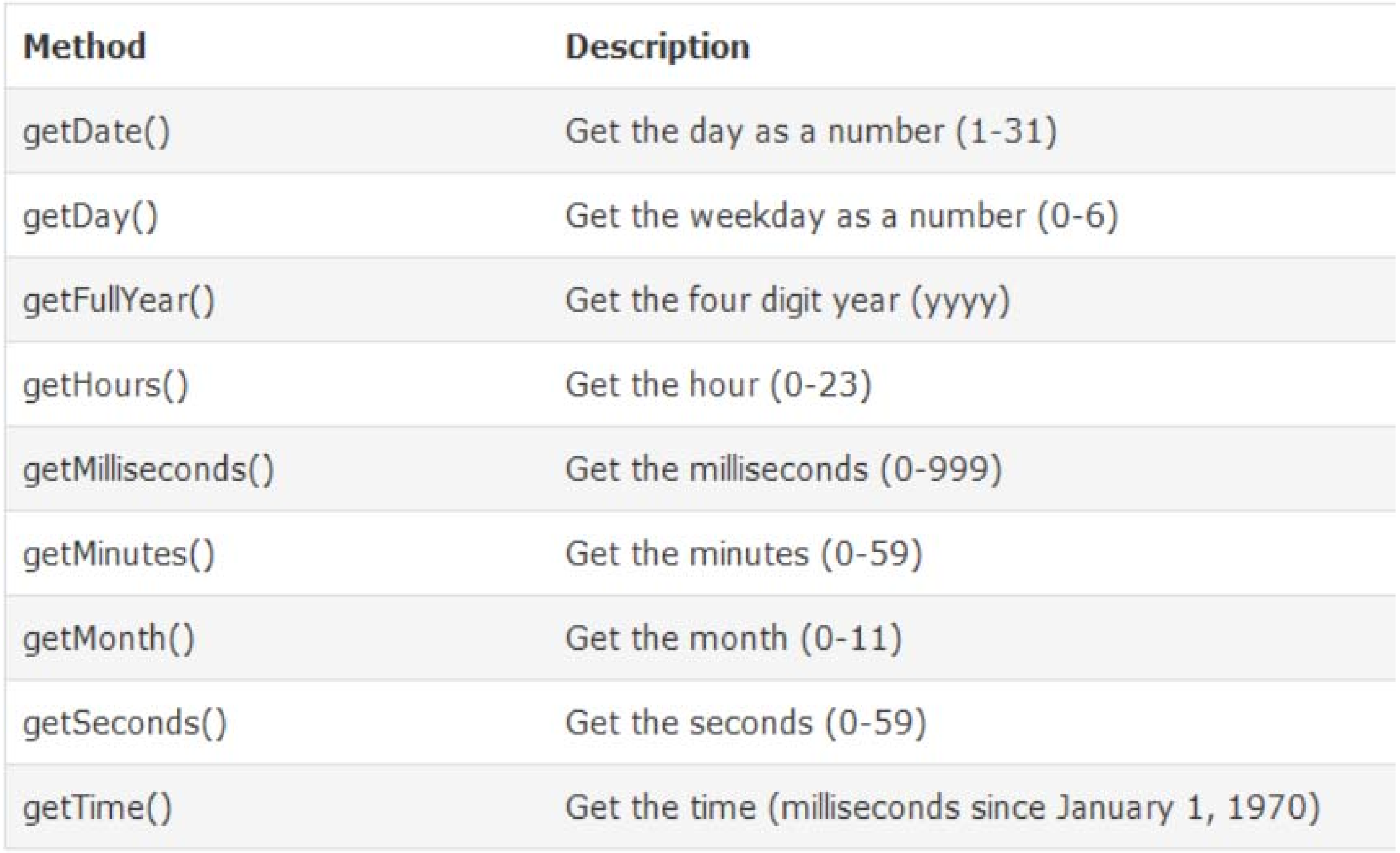
- The Date object lets us work with dates.
- A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.
- Date objects are created with the **new Date()** constructor.
- There are **4 ways** of initiating a date:
- new Date()
  new Date(milliseconds) **(date3.php)**
  new Date(dateString) **(date2.php, takes date and time)**
  new Date(year, month, day, hours, minutes, seconds, milliseconds)
  (date4.php)

Displaying Dates

- Using new Date(), creates a new date object with the **current date and time.**
- Using new Date(**date string**), creates a new date object from the **specified date and time.**
- Using new Date(**number**), creates a new date object as **zero time plus the number**.
  - Zero time is 01 January 1970 00:00:00 UTC. The number is specified in milliseconds. **(datemethod1.php)**
- Using new Date(**7 numbers**), creates a new date object with the **specified date and time**:
  - The 7 numbers specify the year, month, day, hour, minute, second, and millisecond, in that order
- Variants of the example above let us omit any of the last 4 parameters.

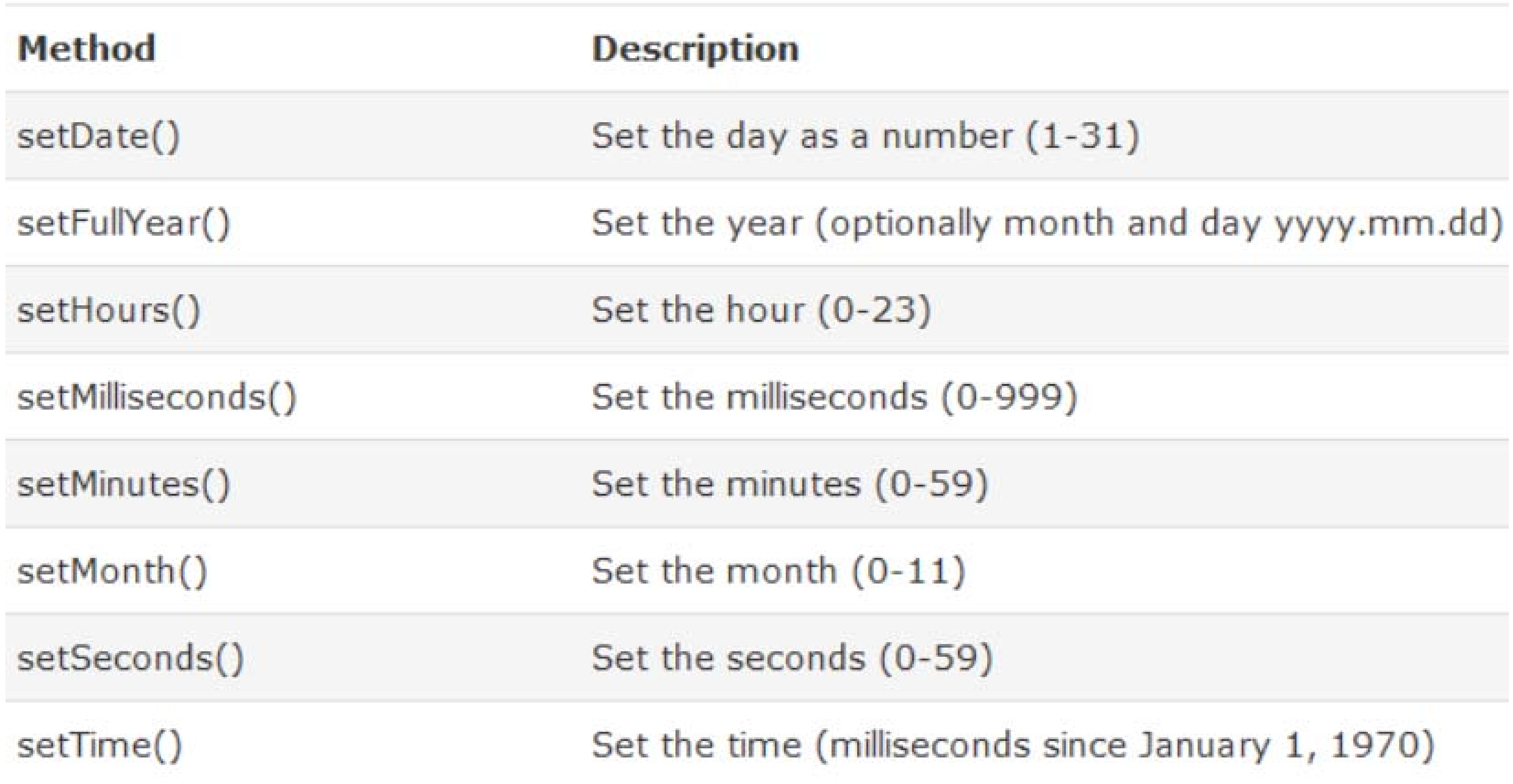
Date Get Methods

datemethod2.php, datemethod3.php



Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Date Set Methods



Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day yyyy.mm.dd)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

JavaScript Regular Expressions

- Regular expression is a sequence of characters that forms a **search pattern**.
- When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of **text search** and **text replace** operations.

JavaScript Regular Expressions

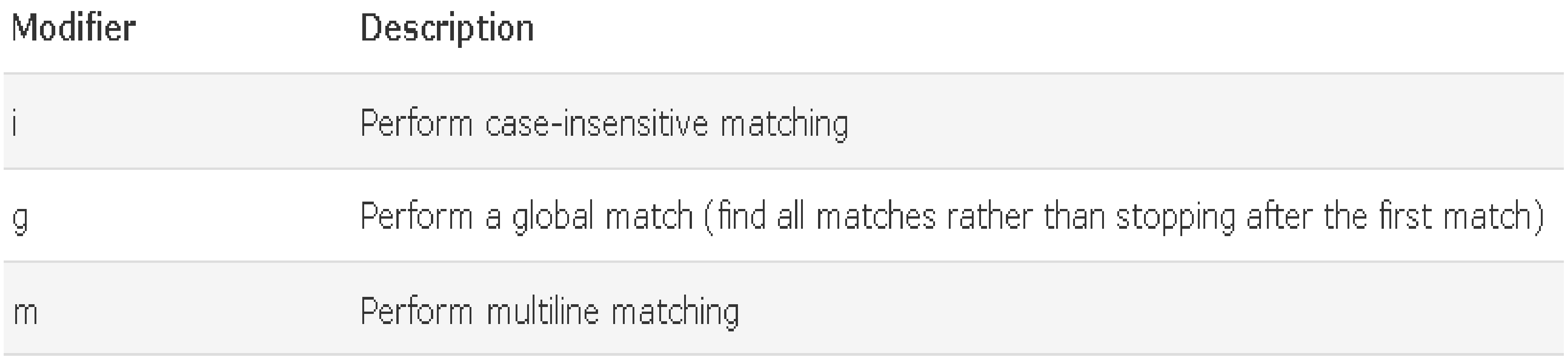
- **Using String Methods**
- In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().
- **The search() method** uses an expression to search for a match, and returns the position of the match. **(regex1.php, regex2.php)**
- **The replace() method** returns a modified string where the pattern is replaced. **(regex3.php)**

JavaScript Regular Expressions

- **Using String search() With String**
- The search method will also accept a string as search argument. The string argument will be converted to a regular expression
- **Use String replace() With a Regular Expression**
- Use a case insensitive regular expression to replace Microsoft with MCA in a string.

JavaScript Regular Expressions

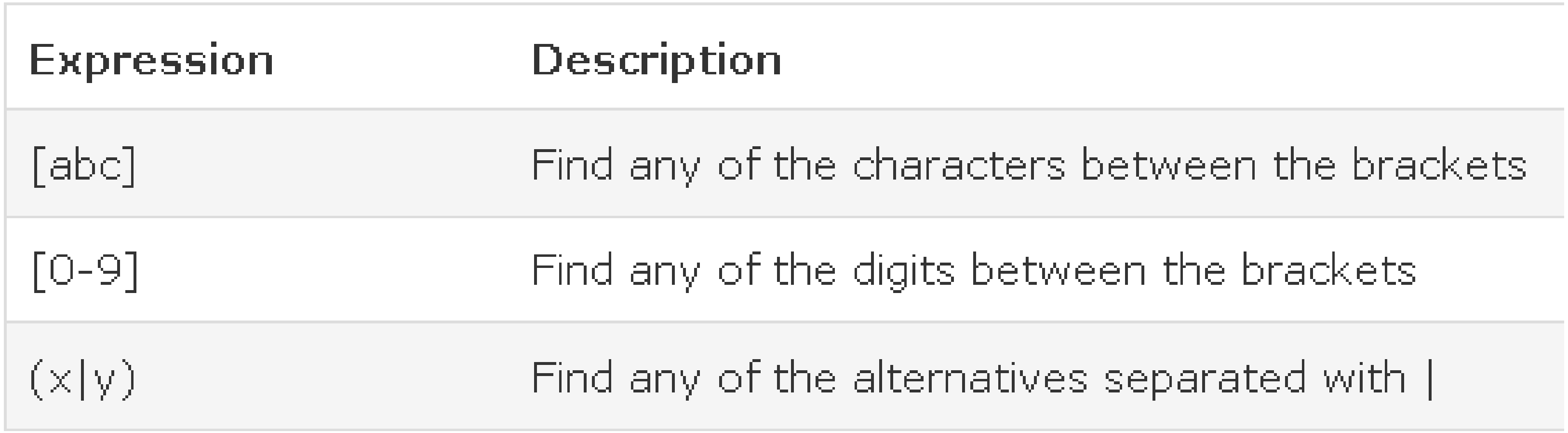
- **Regular Expression Modifiers**
- **Modifiers** can be used to perform case-insensitive more global searches:



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

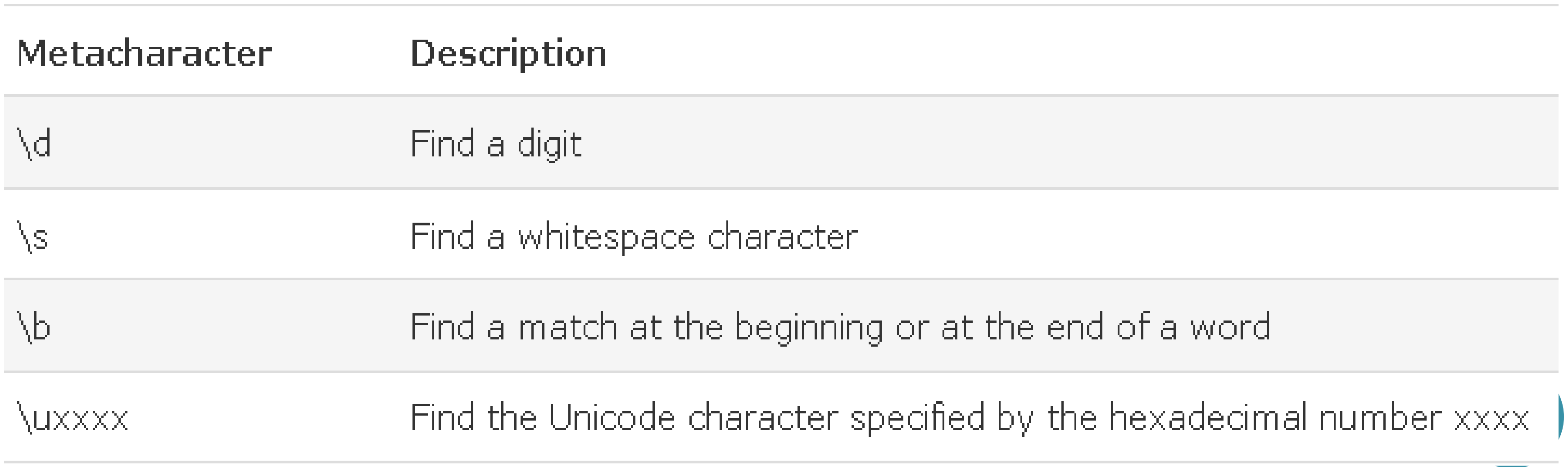
JavaScript Regular Expressions

Brackets are used to find a range of characters:



Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x|y)	Find any of the alternatives separated with |

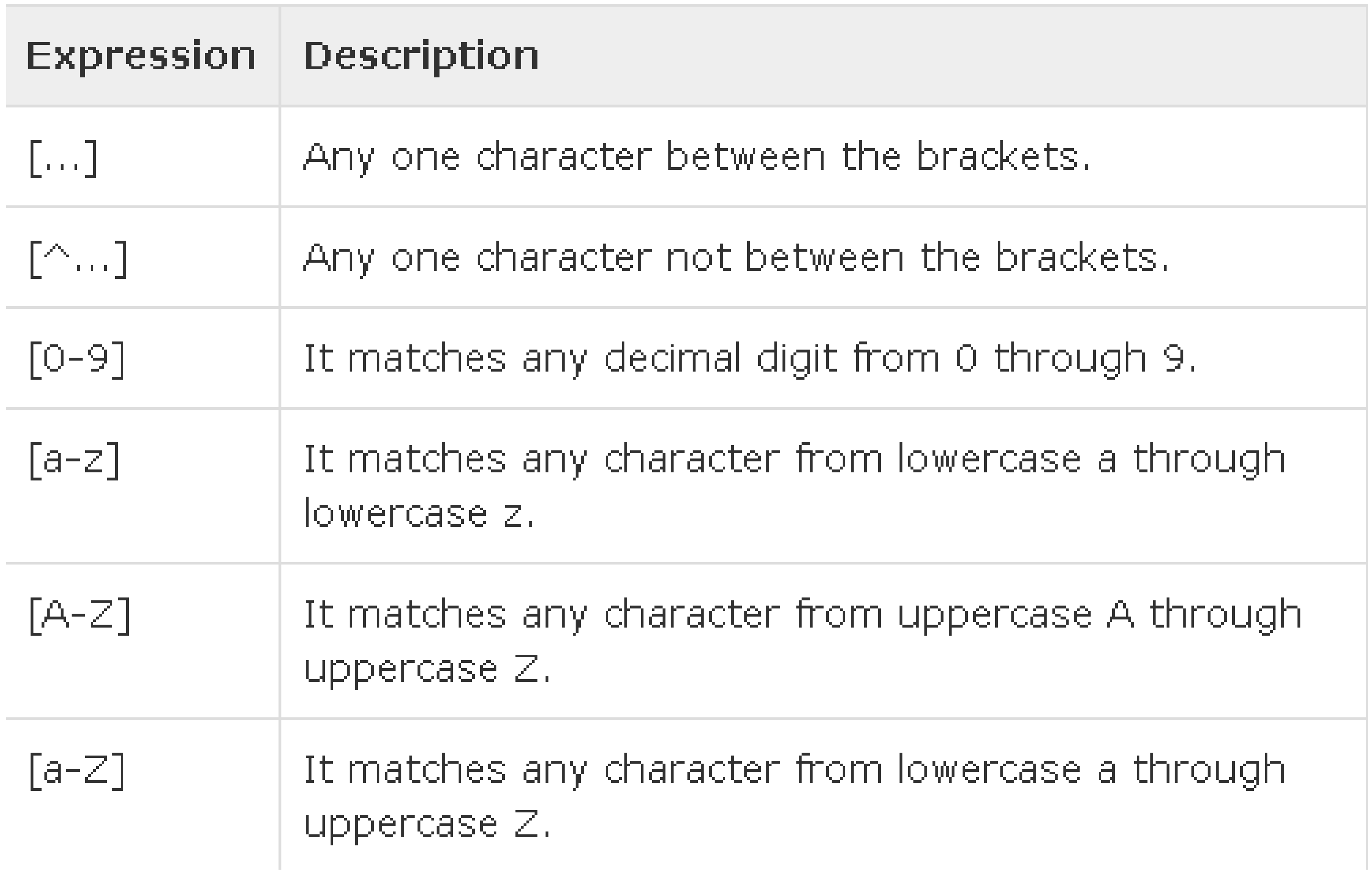
Metacharacters are characters with a special meaning:



Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning or at the end of a word
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

JavaScript Regular Expressions

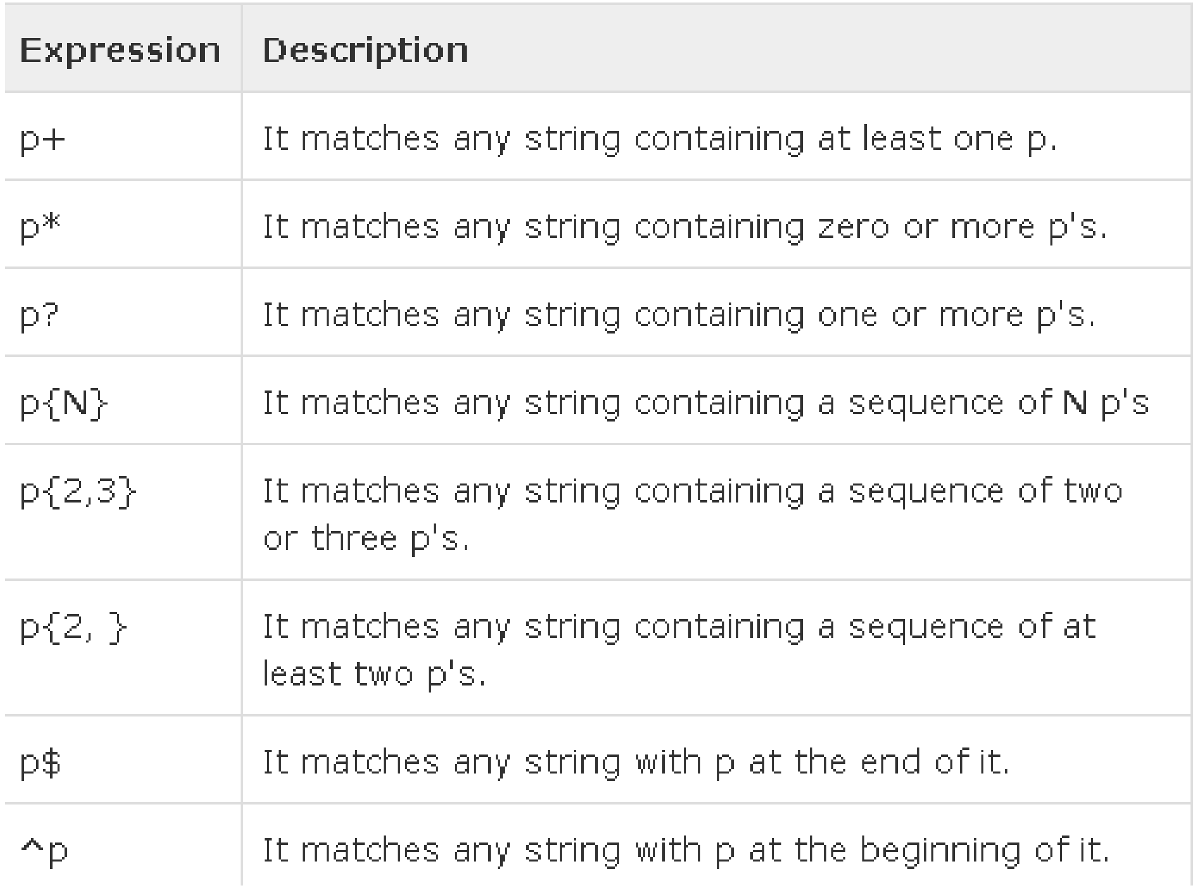
- Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.



Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets.
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

JavaScript Regular Expressions

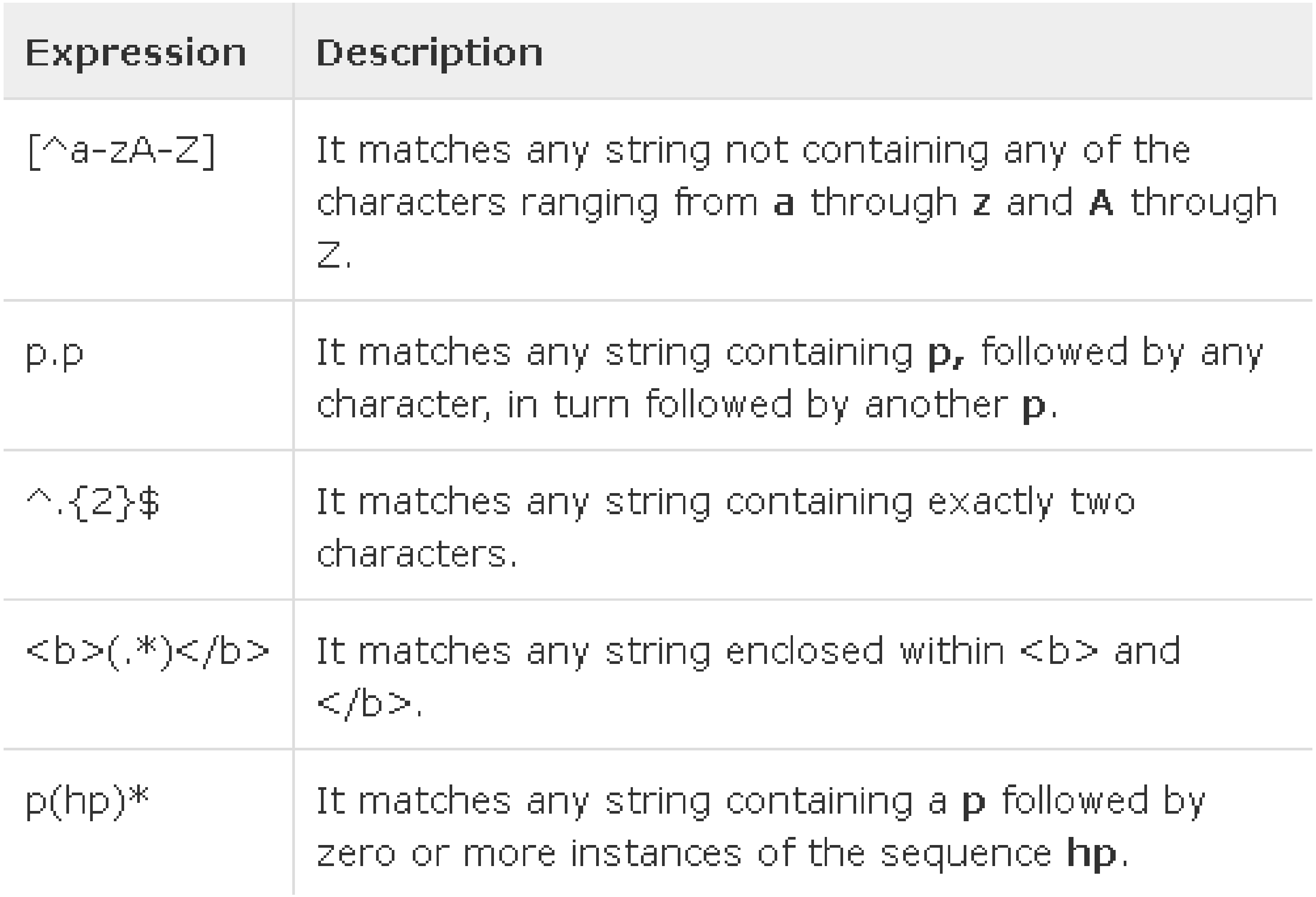
- **Quantifiers**
- The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character has a specific connotation. The +, *, ?, and $ flags all follow a character sequence.



Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing one or more p's.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

JavaScript Regular Expressions

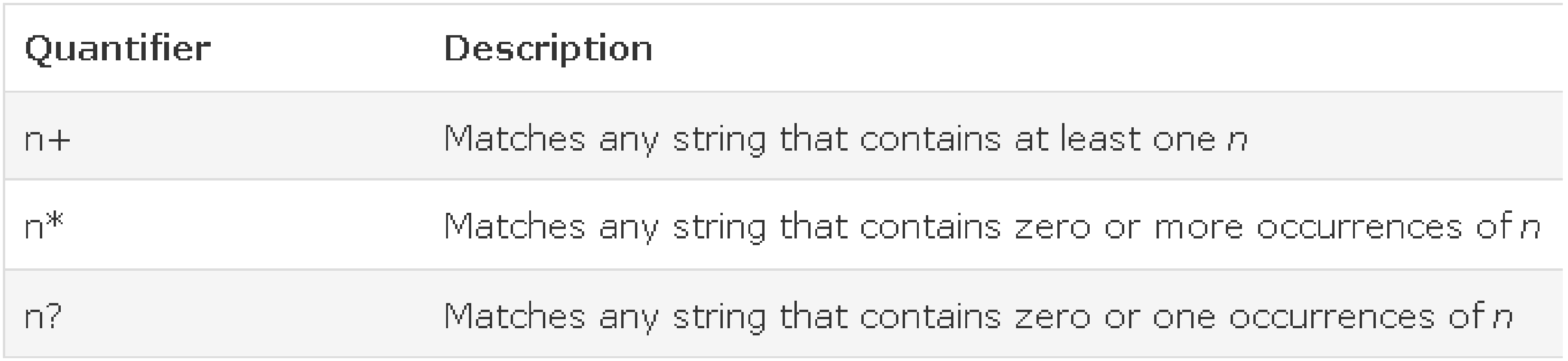
- **Examples**



Expression	Description
[^a-zA-Z]	It matches any string not containing any of the characters ranging from **a** through **z** and **A** through Z.
p.p	It matches any string containing **p**, followed by any character, in turn followed by another **p**.
^.{2}$	It matches any string containing exactly two characters.
<b>(.*)</b>	It matches any string enclosed within <b> and </b>.
p(hp)*	It matches any string containing a **p** followed by zero or more instances of the sequence **hp**.

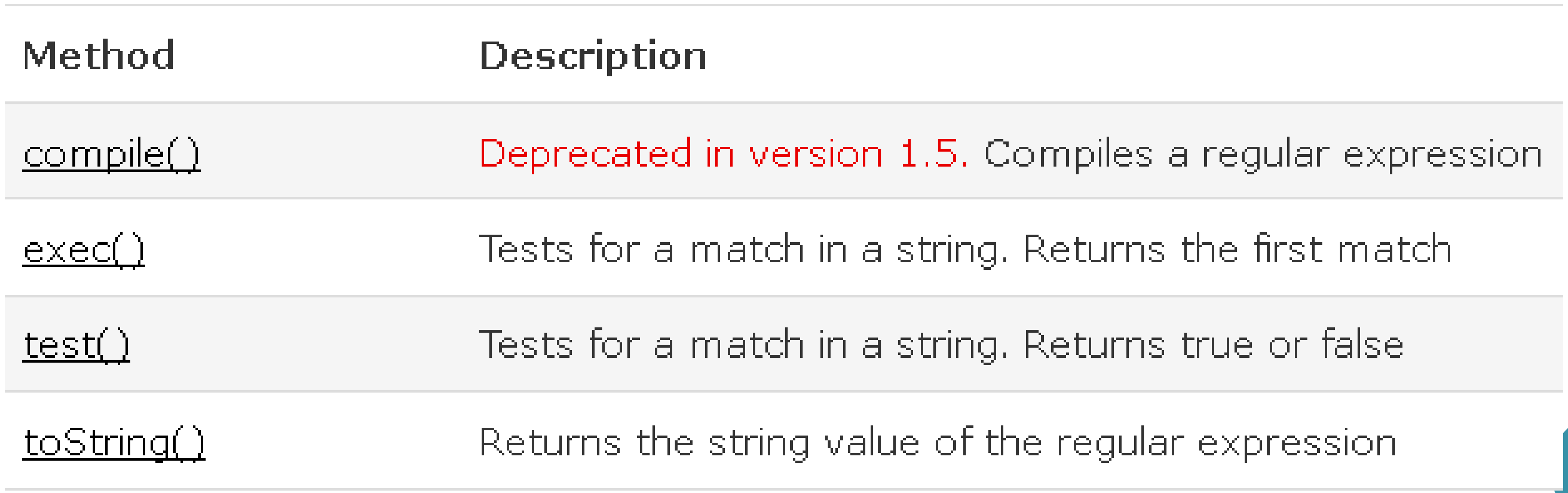
JavaScript Regular Expressions

Quantifiers define quantities:



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*

RegExp Object Methods (regex4.php, regex5.php)



Method	Description
compile()	Deprecated in version 1.5. Compiles a regular expression
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

JavaScript Regular Expressions

- **Using the RegExp Object**
- In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.
- **Using test()**
- It searches a string for a pattern, and returns true or false, depending on the result.
- **Using exec()**
- It searches a string for a specified pattern, and returns the found text.
- If no match is found, it returns *null*.

JavaScript Errors

- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result.
- **JavaScript try and catch**

JavaScript Errors - Throw and Try to Catch

- The **try** statement allows you to define a block of code to be tested for errors while it is being executed.
- The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The JavaScript statements **try** and **catch** come in pairs:

-
  ```
  try {
      Block of code to try
  }
  catch(err) {
      Block of code to handle errors
  }
  ```

JavaScript Errors

- **JavaScript Throws Errors**
- When an error occurs, JavaScript will normally stop, and generate an error message.
- The technical term for this is: JavaScript will **throw** an error.
- **The throw Statement**
- The **throw** statement allows you to create a custom error.
- The technical term for this is: **throw an exception**.
- The exception can be a JavaScript String, a Number, a Boolean or an Object

The finally Statement

- The **finally** statement lets you execute code, after try and catch, regardless of the result:

-
  ```
  try {
      Block of code to try
  }
  catch(err) {
      Block of code to handle errors
  }
  finally {
      Block of code to be executed regardless of the try / catch result
  }
  ```

JavaScript Use Strict

- **"use strict";** Defines that JavaScript code should be executed in "strict mode".
- The "use strict" directive is new in JavaScript 1.8.5 (ECMAScript version 5).
- It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
- The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
- With strict mode, you can not, for example, use undeclared variables.

Why Strict Mode?

- Strict mode makes it easier to write "secure" JavaScript and changes previously accepted "bad syntax" into real errors.
- E.g. in normal JavaScript, mistyping a variable name creates a new global variable.
  - In strict mode, this will throw an error, making it impossible to accidentally create a global variable.
- In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties.
- In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

What is not allowed?

- Using a variable (property or object) without declaring it
- Using a variable (property or object) without declaring it
- Deleting a variable, a function, or an argument
- Defining a property more than once
- Duplicating a parameter name
- Octal numeric literals and escape characters
- Writing to a read-only property
- Writing to a get-only property
- Deleting an undeletable property
- The string "eval" cannot be used
- The string "arguments" cannot be used

JAVASCRIPT - DIALOG BOXES

- JavaScript supports three important types of dialog boxes.
- These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users.
- **Alert Dialog Box:**
- An alert dialog box is mostly used to give a warning message to the users.
- Like if one input field requires to enter some text but user does not enter that field then as a part of validation you can use alert box to give warning message
- alert1.php

JAVASCRIPT - DIALOG BOXES

- **Confirmation Dialog Box:**
- A confirmation dialog box is mostly used to take user's consent on any option.
- It displays a dialog box with two buttons: **OK** and **Cancel**.
- If the user clicks on OK button the window method *confirm()* will return true.
- If the user clicks on the Cancel button *confirm()* returns false. You can use confirmation dialog box
- alert2.php

JavaScript - Dialog Boxes

- **Prompt Dialog Box:**
- The prompt dialog box is very useful when you want to pop-up a text box to get user input.
- This dialog box is displayed using a method called *prompt()* which takes two parameters
  - (i) A label which you want to display in the text box
  - (ii) A default string to display in the text box.
- This dialog box with two buttons: **OK** and **Cancel**.
- If the user clicks on OK button the window method *prompt()* will return entered value from the text box.
- If the user clicks on the Cancel button the window method *prompt()* returns *null*.
- prompt.php, prompt2.php

JavaScript - Page Printing

- JavaScript helps you to implement this functionality using **print** function of *window* object.
- The JavaScript print function **window.print()** will print the current web page when executed.
- You can call this function directly using *onclick* event.
- print.php

JavaScript - Page Refresh

- You can refresh a web page using JavaScript **location.reload** method.
- This code can be called automatically upon an event or simply when the user clicks on a link.
- You can also use JavaScript to refresh the page automatically after a given time period.

JavaScript - Page Redirection

- When you click a URL to reach to a page X but internally you are directed to another page Y that simply happens because of page re-direction.
- This concept is different from JavaScript Page Refresh.

Why redirect?

- There could be various reasons why you would like to redirect from original page, as follows:
- You did not like the name of your domain and you are moving to a new one.
- Same time you want to direct your all visitors to new site.
- In such case you can maintain your old domain but put a single page with a page re-direction so that your all old domain visitors can come to your new domain.

Why redirect?

- You have build-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server side page redirection you can use client side page redirection to land your users on appropriate page.
- The Search Engines may have already indexed your pages.
- But while moving to another domain then you would not like to lose your visitors coming through search engines.
- So you can use client side page redirection.
- redirect1.php, redirect2.php, redirect3.php

JavaScript Timing Events

- With JavaScript, it is possible to execute some code at specified time-intervals. This is called timing events.
- It's very easy to time events in JavaScript. The two key methods that are used are:
  - setInterval() - executes a function, over and over again, at specified time intervals
  - setTimeout() - executes a function, once, after waiting a specified number of milliseconds

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.
- **Syntax**
- window.setInterval("*javascript function*", *milliseconds*);
- 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.
- timeAlert1.php, timeAlert2.php

How to Stop the Execution?

- he clearInterval() method is used to stop further executions of the function specified in the setInterval() method.
- **Syntax**
  - window.clearInterval(*intervalVariable*)
- The **window.clearInterval()** method can be written without the window prefix.
- To be able to use the clearInterval() method, you must use a global variable when creating the interval method:
  - myVar=setInterval("*javascript function*", *milliseconds*);
- timeInterval1.php

The setTimeout() Method

- Syntax
- window.setTimeout("*javascript function*", *milliseconds*);
- 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.
- timeInterval2.php

JavaScript Event Listener

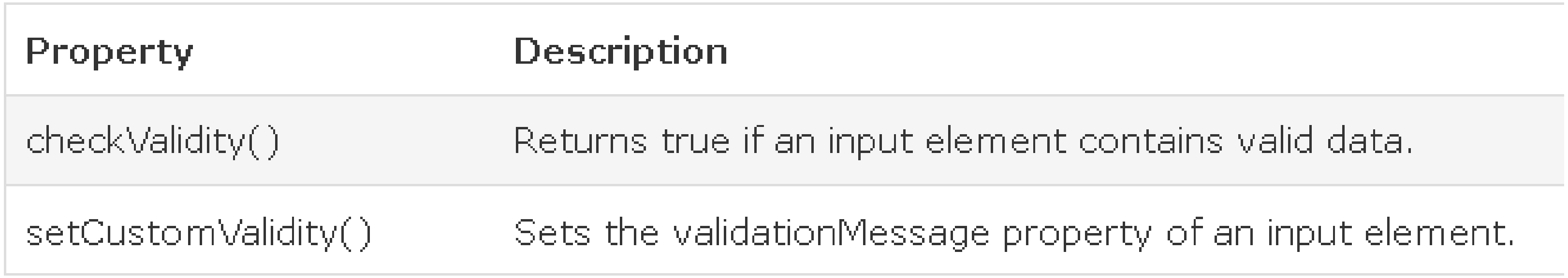
- The addEventListener() method.
- Syntax
  - *element*.addEventListener(*event, function, useCapture*);
- The first parameter is the type of the event (like "click" or "mousedown").
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a Boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

JavaScript Event Listener

- **Example**
- Add an event listener that fires when a user clicks a button.
- The addEventListener() method attaches an event handler to the specified element.
- The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
- You can add many event handlers to one element.
- listener1.php

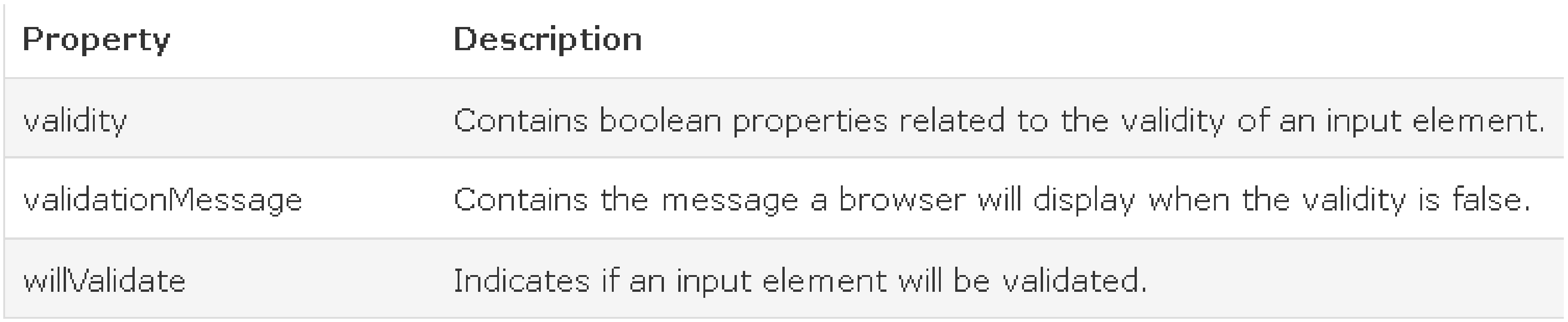
JavaScript Validation API

- **Constraint Validation DOM Methods**



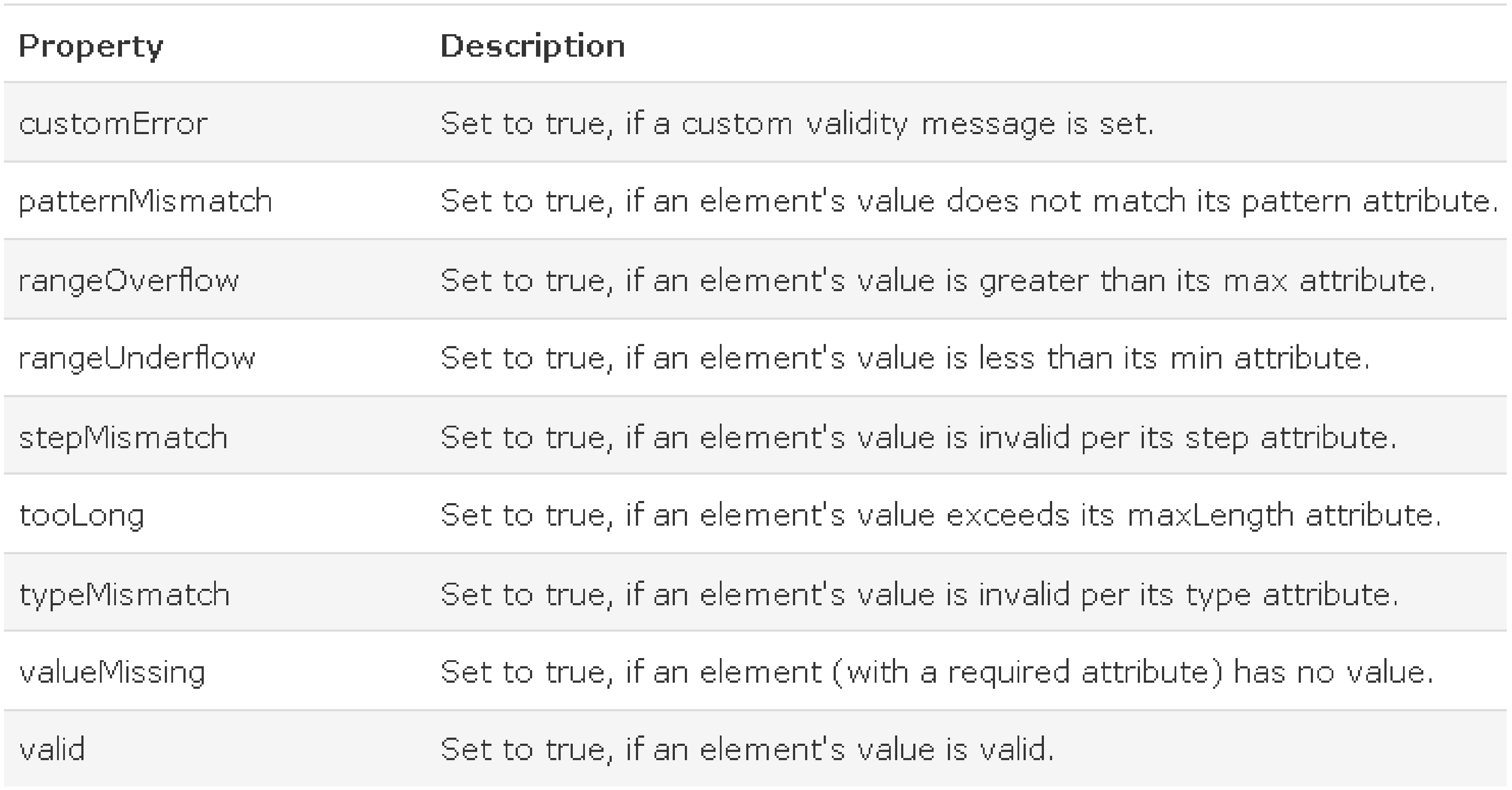
Property	Description
checkValidity()	Returns true if an input element contains valid data.
setCustomValidity()	Sets the validationMessage property of an input element.

- If an input field contains invalid data, display a message.
- **Constraint Validation DOM Properties**



Property	Description
validity	Contains boolean properties related to the validity of an input element.
validationMessage	Contains the message a browser will display when the validity is false.
willValidate	Indicates if an input element will be validated.

Validity Properties



Property	Description
customError	Set to true, if a custom validity message is set.
patternMismatch	Set to true, if an element's value does not match its pattern attribute.
rangeOverflow	Set to true, if an element's value is greater than its max attribute.
rangeUnderflow	Set to true, if an element's value is less than its min attribute.
stepMismatch	Set to true, if an element's value is invalid per its step attribute.
tooLong	Set to true, if an element's value exceeds its maxLength attribute.
typeMismatch	Set to true, if an element's value is invalid per its type attribute.
valueMissing	Set to true, if an element (with a required attribute) has no value.
valid	Set to true, if an element's value is valid.

The rangeOverflow property (validate1.php)
The rangeUnderfloww property (validate2.php)