Que es JavaScript?

**‘An object-oriented computer programming language commonly used to create interactive effects within web browsers.’**

JavaScript es un lenguaje de programación que permite a los desarrolladores crear acciones en sus páginas web.

JavaScript tiene la ventaja de ser incorporado en cualquier página web, puede ser ejecutado sin la necesidad de instalar otro programa para ser visualizado.

Es un lenguaje fácil de emplear, entender y trabajar.

Gran parte de la programación en este lenguaje está centrada en describir objetos, escribir funciones que respondan a movimientos del mouse, aperturas, utilización de teclas, cargas de páginas entre otros.

Cómo nació JavaScript?

JavaScript nació con la necesidad de permitir a los autores de sitio web crear páginas que permitan intercambiar con los usuarios, ya que se necesitaba crear webs de mayor complejidad. El [HTML](http://www.maestrosdelweb.com/editorial/htmlhis/)solo permitía crear páginas estáticas donde se podía mostrar textos con estilos, pero se necesitaba interactuar con los usuarios.

Gracias a su compatibilidad con todos los navegadores modernos se ha convertido en un estándar como lenguaje de programación del lado del cliente.

Del lado del cliente: las extensiones del lado del cliente permiten a una aplicación  colocar elementos en un formulario HTML y responder a los eventos del usuario como clicks del mouse, ingresos en formularios, y navegación en la página.

Del lado del servidor: las extensiones del lado del servidor permiten a las aplicaciones comunicarse con una base de datos relacional, proveyendo un flujo constante de información, o realizar movimientos de archivos en el servidor.

Diferencias entre JavaScript y Java

Java es un lenguaje de programación multiplataforma orientado a objetos. Este lenguaje, por su propiedad de multiplataforma, se puede implementar en cualquier entorno (linux, windows, unix, etc) ya que lo único necesario para que ese lenguaje sea interpretado es que el sistema sobre el cual se van a correr las aplicaciones, webs, etc. posean la JVM (Java Virtual Machine), que es el interpretador de lenguaje java. Los archivos fuente se guardan como .java y los "compilados" se guardan como .class, que son los archivos interpretados por la JVM. Otra cosa es que java es un lenguaje de clases.

Por otro lado, JavaScript es un lenguaje que se implementa sobre plataforma web, el cual sirve para diversas funciones sobre la misma. Contrariamente a lo que informaron en la respuesta, no es solo para realizar retoques en páginas web, si no que JavaScript también cuenta con programación de objetos para interactuar en las páginas (no es solo scripting interpretado). Esto hace que se puedan programar juegos en JavaScript (obviamente interpretados sobre una web). Al mismo tiempo, al igual que java, JavaScript cuenta con diversos frameworks, como prototype, jquery, moo, etc. con los cuales se pueden programar aplicaciones web con muchas utilidades y robustez.

Diferencias:

**Compilador**. Para programar en Java necesitamos UN Kit de desarrollo y UN compilador. Sin embargo, JavaScript no es UN lenguaje que necesite que sus programas se compilen, sino que éstos se interpretan por parte del navegador cuando éste lee la página.   
**Orientado a objetos**. Java es UN lenguaje de programación orientado a objetos. (Más tarde veremos que quiere decir orientado a objetos, para el que no lo sepa todavía) JavaScript no es orientado a objetos, esto quiere decir que podremos programar sin necesidad de crear clases, Tal como se realiza en los lenguajes de programación estructurada como C o Pascal.   
**Propósito**. Java es mucho más potente que JavaScript, esto es debido a que Java es un lenguaje de propósito general, con el que se pueden hacer aplicaciones de lo más variado, sin embargo, con JavaScript sólo podemos escribir programas para que se ejecuten en páginas web.   
**Estructuras Fuertes**. Java es UN lenguaje de programación fuertemente tipado, esto quiere decir que al declarar una variable tendremos que indicar su tipo y no podrá cambiar de UN tipo a otro automáticamente. Por su parte JavaScript no tiene esta característica, y podemos meter en una variable la información que deseemos, independientemente del tipo de ésta. Además, podremos cambiar el tipo de información de una variable cuando queramos.   
**Otras características**. Como vemos Java es mucho más complejo, aunque también más potente, robusto y seguro. Tiene más funcionalidades que JavaScript y las diferencias que los separan son lo suficientemente importantes Como para distinguirlos fácilmente.

JavaScript y la especificación ECMA

Netscape inventó JavaScript y JavaScript fue utilizado por primera vez en los navegadores Netscape. Sin embargo, Netscape está trabajando con [Ecma International](http://www.ecma-international.org/) - la asociación Europea para la estandarización de la información y de los sistemas de comunicación (European Computer Manufacturers Association) para la entrega de un lenguaje de programación internacional estandarizado basado en el núcleo de JavaScript. Esta versión estandarizada de JavaScript, llamada ECMAScript, se comporta de la misma manera en todas las aplicaciones que soportan el estándar. Las compañías pueden usar el estándar abierto del lenguaje para desarrollar sus implementaciones en JavaScript. El estándar ECMAScript está documentado en la especificación ECMA-262.

El estándar ECMA-262 también es provisto por la organización para la estandarización internacional [ISO](http://www.iso.ch/) (International Organization for Standardization) La especificación ECMAScript no describe el modelo objeto documento (DOM) [Document Object Model], el cual está estandarizado por el consorcio W3C [World Wide Web Consortium (W3C)](http://www.w3.org/). El DOM define la manera en la cual los objetos documentos HTML son expuestos para el diseño de su script.

#### La documentación de JavaScript versus la especificación ECMA

La especificación ECMAScript es un conjunto de requerimientos para la implementación de ECMAScript; esto es útil si desea determinar si la característica de JavaScript es soportada en otras implementaciones ECMAScript. Si planea escribir código JavaScript que utilice sólo características soportadas por ECMAScript, entonces necesita revisar la especificación ECMAScript.

Introducción a OOP

La programación orientada a objetos o POO (OOP según sus siglas en inglés) es un [paradigma de programación](http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que usa los [objetos](http://es.wikipedia.org/wiki/Objetos_(programaci%C3%B3n_orientada_a_objetos)) en sus interacciones, para diseñar aplicaciones y programas [informáticos](http://es.wikipedia.org/wiki/Inform%C3%A1tica). Está basado en varias técnicas, incluyendo [herencia](http://es.wikipedia.org/wiki/Herencia_(inform%C3%A1tica)), [cohesión](http://es.wikipedia.org/w/index.php?title=Cohesi%C3%B3n_(inform%C3%A1tica)&action=edit&redlink=1), abstracción, polimorfismo, [acoplamiento](http://es.wikipedia.org/wiki/Acoplamiento_(inform%C3%A1tica)) y [encapsulamiento](http://es.wikipedia.org/wiki/Encapsulamiento_(inform%C3%A1tica)). Su uso se popularizó a principios de la década de los años 1990.

Los objetos son entidades que tienen un determinado estado, comportamiento (método) e identidad:

-El estado está compuesto de datos o informaciones; serán uno o varios atributos a los que se habrán asignado unos valores concretos (datos).

-El comportamiento está definido por los [métodos](http://es.wikipedia.org/wiki/M%C3%A9todo_(inform%C3%A1tica)) o mensajes a los que sabe responder dicho objeto, es decir, qué operaciones se pueden realizar con él.

-La identidad es una propiedad de un objeto que lo diferencia del resto; dicho con otras palabras, es su identificador (concepto análogo al de identificador de una [variable](http://es.wikipedia.org/wiki/Variable_(programaci%C3%B3n)) o una [constante](http://es.wikipedia.org/wiki/Constante_(programaci%C3%B3n))).

Un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados [métodos](http://es.wikipedia.org/wiki/M%C3%A9todo_(programaci%C3%B3n_orientada_a_objetos)), que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separa el estado y el comportamiento.

La POO difiere de la [programación estructurada](http://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada solo se escriben funciones que procesan datos. Los programadores que emplean Programación Orientada a Objetos, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

Conceptos

[**Clase**](http://es.wikipedia.org/wiki/Clase_%28inform%C3%A1tica%29)

Definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella.

[**Herencia**](http://es.wikipedia.org/wiki/Herencia_%28inform%C3%A1tica%29)

(Por ejemplo, herencia de la clase C a la clase D) es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables públicas declaradas en C. Los componentes registrados como "privados" (private) también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Esto es así para mantener hegemónico el ideal de POO.

[**Objeto**](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29)

Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa). Es una instancia a una clase.

[**Método**](http://es.wikipedia.org/wiki/M%C3%A9todo_%28programaci%C3%B3n_orientada_a_objetos%29)

Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

**Evento**

Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.

**Atributos**

Características que tiene la clase.

**Mensaje**

Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

**Propiedad o atributo**

Contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.

**Estado interno**

Es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.

**Componentes de un objeto**

Atributos, identidad, relaciones y métodos.

**Identificación de un objeto**

Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

En comparación con un lenguaje imperativo, una "variable" no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la "función" es un procedimiento interno del método del objeto.

### Reserved keywords as of ECMAScript 6

[Break](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/break) [case](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/switch" \o "The documentation about this has not yet been written; please consider contributing!) class [catch](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/try...catch)

[const](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/const) [continue](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/continue) [debugger](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/debugger) [default](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/switch)

[Delete](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/delete) [do](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/while) [else](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/if...else) [export](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/export)

Extends [finally](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/try...catch) [for](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/for) [function](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/function)

[If](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/if...else) [import](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/import) [in](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/for...in)[instanceof](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/instanceof) [let](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/let)

[New](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/new) [return](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/return) [super](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/super) [switch](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/switch)

[This](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/this) [throw](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/throw) [try](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/try...catch) [type of](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/typeof)

[Var](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/var) [void](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/void) [while](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/while) [with](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/with)

Yield

The following are reserved as future keywords when they are found in strict mode code:

Implements package protected static

Interface private public

The following are reserved as future keywords by older ECMAScript specifications (ECMAScript 1 till 3).

Abstract Boolean by techar

Double final float goto

Int long native short

Synchronized transient volatile

Estas son también palabras reservadas, aunque no son utilizadas en este momento en JavaScript y se reservan para usos futuros:

class, enum, extends, super, const, export e import

En modo estricto ("use strict") también se consideran las siguientes palabras como reservadas:

implements, let, private, public, yield, interface, package, protected y static

V8 (motor JavaScript)

**V8** es un motor de [código abierto](http://es.wikipedia.org/wiki/C%C3%B3digo_abierto) para [JavaScript](http://es.wikipedia.org/wiki/JavaScript) creado por [Google](http://es.wikipedia.org/wiki/Google), siendo su [programador](http://es.wikipedia.org/wiki/Programador) jefe Lars Bak.[3](http://es.wikipedia.org/wiki/V8_%28motor_JavaScript%29#cite_note-3)

Está escrito en [C++](http://es.wikipedia.org/wiki/C%2B%2B) y es usado en [Google Chrome](http://es.wikipedia.org/wiki/Google_Chrome). También el "V8 JavaScript" está integrado en el navegador de internet del sistema operativo [Android 2.2](http://es.wikipedia.org/wiki/Android) “Froyo”. Implementa [ECMAScript](http://es.wikipedia.org/wiki/ECMAScript) como especifica [ECMA-262](http://es.wikipedia.org/w/index.php?title=ECMA-262&action=edit&redlink=1) 5.ª edición y corre en [Windows XP](http://es.wikipedia.org/wiki/Windows_XP), [Vista](http://es.wikipedia.org/wiki/Windows_Vista), [Mac OS X 10.5](http://es.wikipedia.org/wiki/Mac_OS_X) (Leopard) y [Linux](http://es.wikipedia.org/wiki/Linux) en procesadores [IA-32](http://es.wikipedia.org/wiki/IA-32) y [ARM](http://es.wikipedia.org/wiki/Arquitectura_ARM).

V8 puede funcionar de manera individual (standalone) o incorporada a cualquier aplicación C++.

Server-side and client-side programming

Web development is all about communication. In this case, communication between two (2) parties, over the HTTP protocol:

* The **Server** - This party is responsible for **serving** pages.
* The **Client** - This party *requests* pages from the **Server**, and displays them to the user. In most cases, the client is a **web browser**.
  + The **User** - The user *uses* the **Client** in order to surf the web, fill in forms, watch videos online, etc.

## Server-side Programming

Server-side programming, is the general name for the kinds of programs, which are run on the **Server**.

### Uses

* Process user input.
* Display pages.
* Structure web applications.
* Interact with permanent storage (SQL, files).

### Example Languages

* PHP
* ASP.Net in C#, C++, or Visual Basic.
* Nearly any language (C++, C#, Java). These were not designed specifically for the task, but are now often used for application-level web services.

## Client-side programming

Much like the server-side, Client-side programming is the name for all of the programs, which are run on the **Client**.

### Uses

* Make interactive webpages.
* Make stuff happen dynamically on the web page.
* Interact with temporary storage, and local storage (Cookies, local Storage).
* Send requests to the server, and retrieve data from it.
* Provide a remote service for client-side applications, such as software registration, content delivery, or remote multi-player gaming.

### Example languages

* JavaScript (primarily)
* HTML\*
* CSS\*

Any language running on a client device that interacts with a remote service is a client-side language.

# Scripting Language

A [high-level programming language](http://www.webopedia.com/TERM/H/high_level_language.html) that is [interpreted](http://www.webopedia.com/TERM/I/interpreter.html) by another program at [runtime](http://www.webopedia.com/TERM/R/runtime.html) rather than [compiled](http://www.webopedia.com/TERM/C/compiler.html) by the computer's processor as other programming languages (such as [C](http://www.webopedia.com/TERM/C/C.html) and [C++](http://www.webopedia.com/TERM/C/C_plus_plus.html)) are. Scripting languages, which can be embedded within [HTML](http://www.webopedia.com/TERM/H/HTML.html), commonly are used to add functionality to a Web page, such as different menu styles or graphic displays or to serve dynamic advertisements. These types of languages are [client](http://www.webopedia.com/TERM/C/client.html)-side scripting languages, affecting the data that the end user sees in a [browser](http://www.webopedia.com/TERM/B/browser.html) window. Other scripting languages are [server](http://www.webopedia.com/TERM/S/server.html)-side scripting languages that manipulate the data, usually in a [database](http://www.webopedia.com/TERM/D/database.html), on the server.

Scripting languages came about largely because of the development of the Internet as a communications tool. JavaScript, [ASP](http://www.webopedia.com/TERM/A/active_server_pages.html), [JSP](http://www.webopedia.com/TERM/J/JSP.html), [PHP](http://www.webopedia.com/TERM/P/PHP.html), [Perl](http://www.webopedia.com/TERM/P/Perl.html), [Tcl](http://www.webopedia.com/TERM/T/TCL.html) and [Python](http://www.webopedia.com/TERM/P/Python.html) are examples of scripting languages.

The origin of the term was similar to its meaning in "a movie script tells actors what to do": a scripting language controlled the operation of a normally-interactive program, giving it a sequence of work to do all in one batch. For instance, one could put a series of editing commands in a file, and tell an editor to run that "script" as if those commands had been typed interactively.

This usage was generalized under Unix to describe scripts for the shell to execute, which meant that the running of a whole series of programs was controlled, not just one. Thus "shell script".

Chapter 1.

Good Parts

Most programming languages contain good parts and bad parts. I discovered that I could be a better programmer by using only the good parts and avoiding the bad parts. After all, how can you build something good out of bad parts?

Fortunately, JavaScript has some extraordinarily good parts. In JavaScript, there is a beautiful, elegant, highly expressive language that is buried under a steaming pile of good intentions and blunders. The best nature of JavaScript is so effectively hidden that for many years the prevailing opinion of JavaScript was that it was an unsightly, incompetent toy.

WHY JAVASCRIPT?

JavaScript is an important language because it is the language of the web browser. Its association with the browser makes it one of the most popular programming languages in the world. At the same time, it is one of the most despised programming languages in the world.

The amazing thing about JavaScript is that it is possible to get work done with it without knowing much about the language, or even knowing much about programming.

Analyzing JavaScript

The very good ideas include functions, loose typing, dynamic objects, and an expressive object literal notation. The bad ideas include a programming model based on global variables.

JavaScript’s functions are first class objects with (mostly) lexical scoping. JavaScript is the first lambda language to go mainstream. Deep down, JavaScript has more in common with Lisp and Scheme than with Java. It is Lisp in C’s clothing. This makes JavaScript a remarkably powerful language.

JavaScript has a very powerful object literal notation. Objects can be created simply by listing their components. This notation was the inspiration for JSON, the popular data interchange format.

JavaScript is much maligned for its choice of key ideas. For the most part, though, those choices were good, if unusual. But there was one choice that was particularly bad: JavaScript depends on global variables for linkage. All of the top-level variables of all compilation units are tossed together in a common namespace called the global object . This is a bad thing because global variables are evil, and in JavaScript they are fundamental.

“Why should I use JavaScript?” There are two answers. The first is that you don’t have a choice. The Web has become an important platform for application development, and JavaScript is the only language that is found in all browsers.

Chapter 2.

GRAMMAR

This chapter introduces the grammar of the good parts of JavaScript, presenting a quick overview of how the language is structured.

Whitespace

JavaScript offers two forms of comments, block comments formed with /\* \*/ and line-ending comments starting with // . Take care that the comments always accurately describe the code. Obsolete comments are worse than no comments.

Names

Palabras reservadas.

Most of the reserved words in this list are not used in the language. The list does not include some words that should have been reserved but were not, such as undefined , NaN , and Infinity .

Names are used for statements, variables, parameters, property names, operators, and labels.

Unlike most other programming languages, there is no separate integer type, so 1 and 1.0 are the same value.

Negative numbers can be formed by using the – prefix operator.

The value NaN is a number value that is the result of an operation that cannot produce a normal result. NaN is not equal to any value, including itself.

Strings

A string literal can be wrapped in single quotes or double quotes. It can contain zero

or more characters.

Strings have a length property. For example, "seven".length is 5. Strings are immutable. Once it is made, a string can never be changed. But it is easy to make a new string by concatenating other strings together with the + operator.

Two strings containing exactly the same characters in the same order are considered to be the same string. So:

'c' + 'a' + 't' === 'cat'

is true.

Statements

A compilation unit contains a set of executable statements. In web browsers, each <script> tag delivers a compilation unit that is compiled and immediately executed

Statements tend to be executed in order from top to bottom. The sequence of execution can be altered by the conditional statements (if and switch), by the looping statements (while , for , and do ), by the disruptive statements (break , return , and throw), and by function invocation.

A block is a set of statements wrapped in curly braces. Unlike many other languages, blocks in JavaScript do not create a new scope, so variables should be defined at the top of the function, not in blocks.

**Switch statement**

The switch statement performs a multi way branch. It compares the expression for equality with all of the specified cases. The expression can produce a number or a string.

**Case clause**

A case clause contains one or more case expressions. The case expressions need not be constants. The statement following a clause should be a disruptive statement to prevent fall through into the next case. The break statement can be used to exit from a switch.

**While statement**

The while statement performs a simple loop. If the expression is false, then the loop will break. While the expression is truth, the block will be executed.

The conventional form is controlled by three optional clauses: the initialization, the condition, and the increment. First, the initialization is done, which typically initializes the loop variable. Then, the condition is evaluated. Typically, this tests the loop variable against a completion criterion. If the condition is omitted, then a condition of true is assumed. If the condition is false, the loop breaks. Otherwise, the block is executed, then the increment executes, and then the loop repeats with the condition

**For statement**

The other form (called for in) enumerates the property names (or keys) of an object. On each iteration, another property name string from the object is assigned to the variable.

**Do statement**

The do statement is like the while statement except that the expression is tested after the block is executed instead of before. That means that the block will always be executed at least once.

**Try statement**

The try statement executes a block and catches any exceptions that were thrown by the block. The catch clause defines a new variable that will receive the exception object..

**Throw statement**

The expression is usually an object literal containing a name property and a message property. The catcher of the exception can use that information to determine what to do.

**Return statement**

The return statement causes the early return from a function. It can also specify the value to be returned. If a return expression is not specified, then the return value will be undefined.

**Break statement**

The break statement causes the exit from a loop statement or a switch statement. It can optionally have a label that will cause an exit from the labeled statement.

**Expression statement**

An expression statement can either assign values to one or more variables or members, invoke a method, delete a property from an object.

Expressions

The simplest expressions are a literal value (such as a string or number), a variable, a built-in value (true , false , null , undefined , NaN , or Infinity ), an invocation expression preceded by new , a refinement expression preceded by delete , an expression wrapped in parentheses, an expression preceded by a prefix operator.

The operators at the top of the operator precedence list in Table 2-1 have higher precedence. They bind the tightest. The operators at the bottom have the lowest precedence.

Operator precedence

. [] ( ) Refinement and invocation

delete new typeof + - ! Unary operators

\* / % Multiplication, division, modulo

+ - Addition/concatenation, subtraction

>= <= > < Inequality

=== !== Equality

&& Logical and

|| Logical or

?: Ternary

**Prefix operator**

If the operand of! Is truth, it produces false. Otherwise, it produces true .

The + operator adds or concatenates. If you want it to add, make sure both operands are numbers.

The / operator can produce a no ninteger result even if both operands are integers.

The && operator produces the value of its first operand if the first operand is false. Otherwise, it produces the value of the second operand.

**Infix operator**

The || operator produces the value of its first operand if the first operand is truth. Otherwise, it produces the value of the second operand.

**Invocation**

Invocation causes the execution of a function value. The invocation operator is a pair of parentheses that follow the function value. The parentheses can contain arguments that will be delivered to the function.

**Refinement**

A refinement is used to specify a property or element of an object or array. This will be described in detail in the next chapter**.**

Literals

Object literals are a convenient notation for specifying new objects. The names of the properties can be specified as names or as strings.

Functions

A function literal defines a function value. It can have an optional name that it can use to call itself recursively. It can specify a list of parameters that will act as variables initialized by the invocation arguments. The body of the function includes variable definitions and statements.