

Domine JavaScript

4ª edición



Pablo E. Fernández Casado



Ra-Ma®

edU

Domine JavaScript

4a Edición

Pablo E. Fernández Casado



Ra-Ma[®]

edü[®]

Conocimiento a su alcance

BOGOTÁ - MÉXICO, D.F.

Fernández Casado, Pablo E.

Domine JavaScript/ Pablo E. Fernández Casado / 4a. Edición--. Bogotá: Ediciones de la U, 2021

438 p. ; 24 cm

ISBN 978-958-792-301-8 e-ISBN 978-958-792-302-5

1. Programación 2. Variables 3. Tipos de datos 4. Operadores 5. Objetos 6. Funciones

Tít.

621.39 ed.

*Edición original publicada por © Editorial Ra-ma (España)
Edición autorizada a Ediciones de la U para Colombia*

Área: Informática

Cuarta edición: Bogotá, Colombia, septiembre de 2021

ISBN. 978-958-792-301-8

- © Pablo E. Fernández Casado
- © Ra-ma Editorial. Calle Jarama, 3-A (Polígono Industrial Igarsa) 28860 Paracuellos de Jarama
www.ra-ma.es y www.ra-ma.com / E-mail: editorial @ra-ma.com
Madrid, España
- © Ediciones de la U - Carrera 27 #27-43 - Tel. (+57-1) 3203510 -3203499
www.edicionesdelau.com - E-mail: editor@edicionesdelau.com
Bogotá, Colombia

Ediciones de la U es una empresa editorial que, con una visión moderna y estratégica de las tecnologías, desarrolla, promueve, distribuye y comercializa contenidos, herramientas de formación, libros técnicos y profesionales, e-books, e-learning o aprendizaje en línea, realizados por autores con amplia experiencia en las diferentes áreas profesionales e investigativas, para brindar a nuestros usuarios soluciones útiles y prácticas que contribuyan al dominio de sus campos de trabajo y a su mejor desempeño en un mundo global, cambiante y cada vez más competitivo.

Coordinación editorial: Adriana Gutiérrez M.

Carátula: Ediciones de la U

Impresión: DGP Editores SAS

Calle 63 #70D-34, Pbx (57+1) 3203510

Impreso y hecho en Colombia

Printed and made in Colombia

No está permitida la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro y otros medios, sin el permiso previo y por escrito de los titulares del Copyright.

A mi familia

ÍNDICE

PRÓLOGO	17
CAPÍTULO 1. VARIABLES Y AMBITOS	19
1.1 DECLARACIÓN DE VARIABLES	19
1.2 ÁMBITO DE LAS VARIABLES	20
CAPÍTULO 2. TIPOS DE DATOS	23
2.1 TIPO STRING	23
2.1.1 Propiedades	24
2.1.2 Métodos	24
2.1.3 Conversión de Strings	29
2.1.4 Formateado de Strings	30
2.2 TIPO NUMBER	31
2.2.1 Propiedades	32
2.2.2 Métodos	32
2.2.3 Conversión de números	33
2.2.4 Formateado de números	34
2.2.5 Operaciones con números	38
2.3 TIPO BOOLEAN	41
2.3.1 Propiedades	41
2.3.2 Métodos	41
2.4 LITERAL NULL	42
2.5 LITERAL UNDEFINED	42
CAPÍTULO 3. OPERADORES Y EXPRESIONES	43
3.1 OPERADOR DE ASIGNACIÓN	43
3.2 OPERADOR DE CONCATENACIÓN	43
3.3 OPERADORES ARITMÉTICOS	44
3.3.1 Operador de adición	44
3.3.2 Operador de sustracción	44

3.3.3	Operador de multiplicación	44
3.3.4	Operador de división	45
3.3.5	Operador de resto, módulo o residuo	45
3.3.6	Operador de exponenciación	45
3.3.7	Operador de incremento	45
3.3.8	Operador de decremento	46
3.3.9	Orden de los operadores	46
3.4	OPERADORES LÓGICOS	47
3.4.1	Operador AND	47
3.4.2	Operador OR	47
3.4.3	Operador NOT	48
3.5	OPERADORES CONDICIONALES	48
3.5.1	Operador de igualdad	48
3.5.2	Operador de desigualdad	48
3.5.3	Operador de superioridad e inferioridad	49
3.5.4	Operador ternario	49
3.6	OPERADORES BIT A BIT	50
3.6.1	Operador AND	50
3.6.2	Operador OR	50
3.6.3	Operador XOR	50
3.6.4	Operador de complementación	50
3.6.5	Operadores de desplazamiento	51
3.7	OPERADOR COMA	51
CAPÍTULO 4. CONTROL DE FLUJO Y GESTIÓN DE ERRORES		53
4.1	ESTRUCTURA IF	53
4.2	ESTRUCTURA IF...ELSE	53
4.3	ESTRUCTURA SWITCH	54
4.4	CONTROL DE ERRORES POR TIPO DE DATO	54
4.5	CONTROL DE ERRORES POR PRESENCIA	55
4.6	MANEJO DE EXCEPCIONES	56
4.6.1	Sentencia Try...catch	56
4.6.2	Sentencia finally	57
4.6.3	Sentencia throw	57
CAPÍTULO 5. BUCLES Y LA ITERACIÓN		59
5.1	ESTRUCTURA FOR	59
5.2	ESTRUCTURA FOR...IN	60
5.3	ESTRUCTURA FOR...OF	61
5.4	ESTRUCTURA FOREACH	62
5.5	ESTRUCTURA DO...WHILE	64
5.6	ESTRUCTURA WHILE	65
5.7	SENTENCIA BREAK	66

5.8	SENTENCIA CONTINUE	66
CAPÍTULO 6. INTERNACIONALIZACIÓN		67
6.1	INTERNACIONALIZACIÓN DE NÚMEROS.....	67
6.1.1	El objeto NumberFormat.....	67
6.2	INTERNACIONALIZACIÓN DE FECHAS.....	71
6.2.1	El objeto Date.....	71
6.2.2	El objeto DateTimeFormat.....	75
6.3	COMPARACIÓN DE CADENAS SENSIBLES AL LENGUAJE.....	80
6.3.1	El objeto Collator	80
CAPÍTULO 7. OBJETOS.....		85
7.1	TIPOS DE OBJETO	85
7.2	PROPIEDADES	86
7.3	MÉTODOS	86
7.3.1	Método assign	86
7.3.2	Método create.....	87
7.3.3	Método entries.....	87
7.3.4	Método getOwnPropertyDescriptors.....	87
7.3.5	Método getOwnPropertyDescriptor	88
7.3.6	Método getOwnPropertyNames.....	88
7.3.7	Método hasOwnProperty.....	88
7.3.8	Método keys	88
7.3.9	Método values	89
7.4	ARRAYS.....	89
7.4.1	Propiedades	89
7.4.2	Métodos.....	89
7.4.3	Creación de arrays.....	93
7.4.4	Acceso a elementos de un array	93
7.4.5	Inserción y almacenamiento de elementos en un array.....	94
7.4.6	Eliminación de elementos de un array	94
7.5	JSON.....	95
7.5.1	Sintaxis.....	96
7.5.2	Creación de JSON.....	96
7.5.3	Acceso a elementos de un JSON.....	97
7.5.4	Inserción y almacenamiento de elementos en un JSON.....	97
7.5.5	Eliminación de elementos de un JSON.....	98
7.5.6	Envío y recepción de JSON	98
7.6	ESPECIALES	100
7.6.1	El objeto window	100
7.6.2	El objeto document.....	102
7.6.3	El objeto Screen	104
7.6.4	La interfaz Navigator.....	104
7.6.5	La interfaz Location	105
7.6.6	La interfaz HTMLInputElement.....	106

7.6.7	El objeto History	112
7.6.8	El objeto this.....	115
7.6.9	El objeto globalThis	117
7.6.10	El objeto prototype.....	117
7.7	OTRAS COSAS QUE SABER SOBRE LOS OBJETOS DE JAVASCRIPT ...	119
7.7.1	La herencia.....	119
7.7.2	Sentencias get y set	121
CAPÍTULO 8. FUNCIONES.....		123
8.1	CREACIÓN DE FUNCIONES	123
8.1.1	Diferencia entre modo estricto o modo no estricto	124
8.2	PASO DE PARÁMETROS.....	125
8.2.1	Por asignación directa	125
8.2.2	El objeto arguments	125
8.3	FUNCIONES ANÓNIMAS.....	127
8.3.1	Ventajas e inconvenientes.....	128
8.4	FUNCIONES CLAUSURA.....	129
8.4.1	Ventajas e inconvenientes.....	130
8.5	FUNCIONES FLECHA.....	130
8.5.1	Ventajas e inconvenientes.....	132
8.6	FUNCIONES ESPECIALES.....	132
8.6.1	Función de prototipo bind	132
8.6.2	Función de prototipo call.....	133
8.6.3	Función de prototipo apply	134
8.6.4	Diferencias entre call y apply.....	135
8.7	CONTEXTOS Y ENCAPSULAMIENTO	136
CAPÍTULO 9. CLASES.....		139
9.1	CREACIÓN DE CLASES	139
9.2	INSERCIÓN DE MÉTODOS.....	140
9.2.1	Sentencias get y set	141
9.3	EXTENSIÓN DE CLASES.....	143
9.3.1	Extensión a través de species	144
9.3.2	Extensión a través de super	145
9.4	CLASES ABSTRACTAS Y MIXINS	147
CAPÍTULO 10. EXPRESIONES REGULARES		149
10.1	DEFINICIÓN DE PATRONES	149
10.2	EL OBJETO REGEXP.....	151
10.3	USO Y FUNCIONAMIENTO DE LOS MÉTODOS PRINCIPALES.....	153
10.3.1	Método exec	153
10.3.2	Método test.....	153
10.3.3	Método match.....	153
10.3.4	Método replace.....	153

10.3.5	Método search	154
10.3.6	Método split.....	154
CAPÍTULO 11. EVENTOS.....		155
11.1	PRINCIPIO FUNDAMENTAL DE PROPAGACIÓN	156
11.2	EL OBJETO EVENT	156
11.2.1	Propiedades más frecuentes	157
11.3	LA INTERFAZ TOUCHEVENT.....	160
11.3.1	El objeto Touch.....	161
11.4	LA INTERFAZ KEYBOARDEVENT	162
11.4.1	Propiedades más importantes	162
11.5	LA INTERFAZ MOUSEEVENT	163
11.5.1	Propiedades más importantes	163
11.6	PRINCIPALES MANEJADORES DE EVENTOS.....	164
11.6.1	Eventos de ratón	164
11.6.2	Eventos de formulario	165
11.6.3	Eventos de HTML.....	165
11.6.4	Eventos de tratamiento táctil	166
11.7	OYENTES O LISTENERS	166
11.7.1	Método addEventListener	166
11.7.2	Método removeEventListener	167
11.7.3	Otras formas de establecer listeners	167
11.8	PRINCIPALES EVENTOS DEL DOM	168
11.8.1	Document DOMContentLoaded	168
11.8.2	Window load	168
11.8.3	Window resize	169
11.8.4	El evento scroll.....	169
CAPÍTULO 12. EL DOM.....		171
12.1	PROCESO DE CARGA	171
12.2	LOS NODOS Y SUS TIPOS	172
12.3	SELECCIÓN DE ELEMENTOS	172
12.3.1	Interfaz NodeList.....	173
12.3.2	Los selectores	174
12.3.3	Métodos para acceder a los nodos y elementos	175
12.4	CREACIÓN DE NODOS Y ELEMENTOS.....	176
12.4.1	Interfaz DOMTokenList	176
12.4.2	Método createElement	177
12.4.3	Propiedad id	177
12.4.4	Propiedad innerHTML	178
12.4.5	Propiedad value	178
12.4.6	Método setAttribute	178
12.4.7	Propiedad classList.....	179
12.4.8	Propiedad previousElementSibling	179

12.4.9	Propiedad nextElementSibling	180
12.4.10	Propiedad parentElement	180
12.4.11	Método appendChild	180
12.4.12	Método insertBefore	181
12.5	ELIMINACIÓN DE NODOS Y ELEMENTOS	182
12.5.1	Método remove	182
12.5.2	Método removeChild	182
12.6	DEFINICIÓN DE ESTILOS	183
12.6.1	La interfaz CSSStyleDeclaration	183
12.6.2	La interfaz CSSStyleSheet	184
12.6.3	Propiedad style	186
12.6.4	Método insertRule	187
12.6.5	Método deleteRule	187
CAPÍTULO 13. OBSERVADORES DE MUTACIÓN		189
13.1	EL OBJETO MUTATIONRECORD	189
13.1.1	Propiedades	190
13.2	EL OBJETO MUTATIONOBSERVER	191
13.2.1	Métodos	192
CAPÍTULO 14. GESTIÓN DE GRÁFICOS.....		195
14.1	LA INTERFAZ HTMLIMAGEELEMENT	195
14.1.1	Propiedades	195
14.1.2	Métodos y eventos.....	196
14.2	LA API CANVAS	197
14.2.1	Métodos y propiedades para dibujar	198
14.2.2	Métodos para transformaciones	206
14.2.3	Métodos y propiedades para textos y sombras.....	208
14.2.4	Métodos y propiedades para control de imágenes.....	211
14.2.5	Otras propiedades y métodos de interés	222
CAPÍTULO 15. ALMACENAMIENTO WEB.....		227
15.1	LA INTERFAZ STORAGE	227
15.1.1	Propiedad localStorage.....	227
15.1.2	Propiedad sessionStorage	228
CAPÍTULO 16. BASES DE DATOS WEB.....		231
16.1	LA API WEBSQL.....	232
16.1.1	Terminología y aclaraciones	232
16.1.2	Creación de bases de datos	232
16.1.3	Creación de tablas	233
16.1.4	Adición de datos	233
16.1.5	Consulta de datos	234
16.1.6	Actualización de datos	235
16.1.7	Eliminación de datos	236
16.1.8	Otras posibilidades	236

16.1.9	Límites de almacenamiento.....	238
16.2	A LA API INDEXEDDB	239
16.2.1	Terminología y aclaraciones.....	239
16.2.2	Creación de bases de datos.....	240
16.2.3	Creación de tablas	241
16.2.4	Adición de datos.....	243
16.2.5	Consulta de datos	244
16.2.6	Número total de registros	247
16.2.7	Actualización de datos	248
16.2.8	Eliminación de datos	249
16.2.9	Actualización de la base de datos.....	250
16.2.10	Eliminación de la base de datos	251
16.2.11	Límites de almacenamiento.....	252
CAPÍTULO 17. DRAG & DROP		255
17.1	LISTADO DE EVENTOS DISPONIBLES.....	255
17.2	EL OBJETO DATATRANSFER.....	256
17.2.1	Método setData	256
17.2.2	Método setData	256
17.2.3	Método clearData	256
17.2.4	Propiedad types	256
17.3	EJEMPLO DE USO.....	256
17.4	ACLARACIONES CON RESPECTO A LOS NAVEGADORES.....	258
17.4.1	Para Firefox	258
17.4.2	Para Google Chrome y Safari.....	258
CAPÍTULO 18. GESTIÓN DE FICHEROS.....		259
18.1	LECTURA DE ARCHIVOS.....	259
18.1.1	La interfaz File	259
18.1.2	El objeto FileReader.....	260
18.1.3	Ejemplo de FileReader	262
18.2	ESCRITURA DE ARCHIVOS	263
18.2.1	La API FileSystem	263
18.2.2	La API FileHandle.....	270
18.2.3	El objeto FileSystemObject de ActiveX.....	270
CAPÍTULO 19. ATRIBUTOS PERSONALIZADOS		271
19.1	LA PROPIEDAD DATASET.....	272
19.1.1	Adicción de atributos personalizados.....	273
19.1.2	Eliminar atributos personalizados	273
CAPÍTULO 20. GESTIÓN Y VALIDACIÓN DE FORMULARIOS		275
20.1	PROPIEDADES DE LOS FORMULARIOS	275
20.2	PROPIEDADES DE LOS ELEMENTOS DE FORMULARIO	277
20.3	CREACIÓN Y ENVÍO DE FORMULARIOS	278

20.4	VALIDACIÓN DE FORMULARIOS	279
20.4.1	La interfaz ValidityState	280
20.4.2	Propiedades y métodos	281
20.4.3	Eventos	282
20.4.4	Ejemplo de validación	283
CAPÍTULO 21. JAVASCRIPT ASÍNCRONO		285
21.1	EL ESTÁNDAR CORS	285
21.1.1	Encabezados de solicitud HTTP	286
21.1.2	Encabezados de respuesta HTTP	286
21.2	CONEXIONES HTTP	287
21.2.1	Objeto XMLHttpRequest	288
21.2.2	Propiedades	289
21.2.3	Métodos	290
21.2.4	Eventos	292
21.2.5	Ejemplo sencillo de XMLHttpRequest	293
21.3	PROMESAS	293
21.3.1	Objeto Promise	294
21.3.2	La API fetch	298
CAPÍTULO 22. WEB SOCKETS		305
22.1	EL OBJETO WEBSOCKET	306
22.1.1	Evento onopen	306
22.1.2	Evento onclose	307
22.1.3	Evento onerror	307
22.1.4	Evento onmessage	307
22.1.5	Método close	308
22.1.6	Método send	308
22.1.7	Propiedad readyState	309
22.1.8	Propiedad bufferedAmount	309
22.1.9	Extensiones y subprotocolos	310
22.2	EJEMPLO DE WEBSOCKET	311
CAPÍTULO 23. WEB WORKERS		313
23.1	LA INTERFAZ WORKER	314
23.1.1	Evento onmessage	315
23.1.2	Evento onerror	315
23.1.3	Evento onmessageerror	316
23.1.4	Método postMessage	316
23.1.5	Método terminate	316
23.2	EJEMPLO DE WORKER DEDICADO	316
23.3	LA INTERFAZ SHAREDWORKER	317
23.3.1	Propiedad port	318
23.3.2	Evento onmessage	318
23.3.3	Evento onerror	319
23.3.4	Evento onmessageerror	319

23.3.5	Método postMessage.....	320
23.3.6	Método terminate	320
23.4	EJEMPLO DE WORKER COMPARTIDO.....	320
CAPÍTULO 24. GEOLOCALIZACIÓN.....		323
24.1	LA INTERFAZ GEOLOCATION	323
24.1.1	Método getCurrentPosition	323
24.1.2	Método watchPosition.....	326
24.1.3	Método clearWatch.....	327
24.1.4	Opciones de configuración	328
24.2	EJEMPLO CON GOOGLE MAPS	329
CAPÍTULO 25. NOTIFICACIONES WEB.....		331
25.1	LA INTERFAZ NOTIFICATION.....	332
25.1.1	Detectando la presencia de la API Notification.....	332
25.1.2	Solicitando permiso	333
25.1.3	Propiedades de las notificaciones	334
25.1.4	Eventos	338
25.1.5	Métodos.....	339
CAPÍTULO 26. DISEÑO DE COMPONENTES WEB.....		341
26.1	INTRODUCCIÓN A LOS COMPONENTES.....	341
26.2	DEFINICIÓN POR DECLARACIÓN	341
26.2.1	Formas básicas de crear componentes.....	342
26.2.2	El paso de parámetros	343
26.2.3	La interfaz de usuario	344
26.2.4	La personalización.....	347
26.3	EXTENSIÓN DE ELEMENTOS NATIVOS	347
26.3.1	Método registerElement	348
26.3.2	Adición de propiedades y métodos	348
26.3.3	Ciclo de vida de un elemento personalizado	349
26.3.4	Adición del Shadow DOM.....	351
26.3.5	La interfaz Custom Elements	353
26.3.6	Shadow DOM.....	355
26.3.7	HTML Templates	356
26.3.8	Métodos utilizados para la definición de clases	356
26.3.9	Ejemplo de componente Web	357
26.3.10	Compatibilidad con los navegadores.....	359
26.4	EJEMPLOS DE COMPONENTES	360
CAPÍTULO 27. LIBRERÍA ISITOOLS.....		415
27.1	DESCARGA E INSTALACIÓN	415
27.2	COMO FUNCIONA ISITOOLS	416
27.3	UTILIDADES GRÁFICAS.....	418
27.3.1	Componente Alert	418
27.3.2	Componente Autocomplete	419

27.3.3	Datepicker	419
27.3.4	NState	420
27.3.5	Password.....	420
27.3.6	Selectpicker	420
27.3.7	Treeview	421
27.3.8	Validator	421
APÉNDICE 1. RESUMEN DE SELECTORES DE CSS.....		423
APÉNDICE2. RESUMEN DE ELEMENTOS FORMULARIO DE HTML.....		425
APÉNDICE 3. RESOLUCIÓN A LOS PROBLEMAS PROPUESTOS		427
A3.1	DETECTAR SI EL DISPOSITIVO ES MÓVIL O DE ESCRITORIO.....	427
A3.2	CONTADOR DE CARACTERES PARA ELEMENTOS DE FORMULARIO	427
A3.3	CREAR BOTONES DE TIPO TOGGLE SWITCH	428
A3.4	CARGA ASÍNCRONA DE SCRIPTS.....	430
A3.5	CREACIÓN DE UN LECTOR RSS CON LA API FETCH.....	431
A3.6	ADICIÓN DE UN LINECHART A NUESTRO COMPONENTE CHARTS	433
A3.7	VISOR DE IMÁGENES MEDIANTE EXTENSIÓN DE ELEMENTOS NATIVOS	435
REFERENCIAS.....		437

PRÓLOGO

JavaScript es un lenguaje de programación interpretado, basado en el estándar ECMAScript (European Computer Manufacturer's Association Script). Se caracteriza por ser un lenguaje de programación orientado a eventos y basado en prototipos, dinámico y no demasiado tipado.

Sus orígenes se sitúan en 1995 y su nombre original era Mocha. Sin embargo, no tardó mucho en ser renombrado a LiveScript hasta que, finalmente, fue bautizado como JavaScript. La razón de este último cambio fue porque Sun Microsystems (propietaria de Java) compró Netscape y, como estrategia de marketing, decidió llamarlo como su “perla” más preciada. En resumen, que JavaScript no es el lenguaje script de Java.

Cabe destacar que ya, en el año 2012, todos los navegadores soportaban el estándar ECMAScript 5.1, con alguna excepción. No obstante, fue en el año 2015 cuando JavaScript alcanzó casi todo su potencial, con la llegada de ECMAScript 6.

El uso que se le da a JavaScript está, básicamente, en el lado del cliente y son los navegadores quienes lo implementan como parte de su potencial. Es por esta razón que muchas sentencias, métodos y eventos no funcionan igual, dependiendo de en qué navegador estemos trabajando y puede que, incluso, algunas funcionalidades ni si quiera, funcionen. Por suerte parece que, no tardando mucho, esto va a cambiar.

También existe, como muchos sabrán, un JavaScript que trabaja en el lado del servidor, aunque su uso está más encaminado a la programación orientada a eventos, desarrollo de microservicios y diseño de aplicaciones con alta carga de computación.

En lo referente a su sintaxis, JavaScript resulta tener un cierto parecido con Java, sin embargo, fue construido basándose en la sintaxis de C.

Por último, sólo aclarar que, este libro está pensado para trabajar de la manera más compatible posible, es decir, que la mayoría de las cosas que aquí se explican, deberían funcionar en todos los navegadores, incluyendo Internet Explorer 11.

1

VARIABLES Y AMBITOS

A continuación, se realiza una visión inicial sobre las características básicas de JavaScript.

1.1 DECLARACIÓN DE VARIABLES

Como en casi todos los lenguajes de programación, los identificadores de variables sólo pueden empezar por una letra mayúscula, minúscula, guion bajo o símbolo dólar. No se permiten nombres de variables que empiecen por otros símbolos o dígitos y no admiten ningún tipo de operador lógico o matemático.

Para declarar una variable podemos recurrir a tres palabras reservadas, dependiendo de la versión de ECMAScript que tengamos disponible en el navegador.

La más frecuentemente utilizada es la palabra reservada **var**, ya que es compatible con todas las versiones de ECMAScript y es la menos restrictiva y más compatible entre navegadores, incluyendo Internet Explorer 11.

```
var fechaActual = new Date();
```

Otra de las formas de realizar la declaración de una variable podría ser a través de la palabra reservada **let**, sin embargo, este tipo de declaración sólo es compatible con los navegadores más modernos que contemplan ECMAScript 2015 (ES6), lo que no incluye a Internet Explorer 11.

```
let fechaActual = new Date();
```

Mientras que el uso de **var** permite la redefinición o sobrescritura de variables, este tipo de declaración no. Una vez que se haya realizado la primera

definición, no se permitirá que el nombre de la variable pueda volver a ser definida dentro del mismo contexto o bloque, no obstante, esta limitación puede ayudar a evitar errores debidos a la sobreescritura accidental.

La tercera forma que disponemos para realizar la declaración de variables es a través de la palabra reservada **const**, sin embargo, al igual que pasa con la palabra reservada **let**, este tipo de declaración sólo es compatible con los navegadores más modernos que contemplan ECMAScript 2015 (ES6), lo que no incluye a Internet Explorer 11.

```
const fechaActual = new Date();
```

En este caso, la principal diferencia es que, mientras que **var** y **let** permiten la reasignación de valores, **const** define el identificador como una declaración de constante y prohíbe su reasignación.

1.2 ÁMBITO DE LAS VARIABLES

El ámbito de las variables es un tema, a veces, complicado. Sea cual sea el lenguaje de programación siempre se producen confusiones sobre su origen y cómo afectan las variables, por ello, empezaremos por lo básico.

El ámbito de una variable, también conocido como *scope*, es el bloque o parte del código donde, esa variable, se define y está accesible. En JavaScript, los ámbitos sólo pueden ser dos: global y local.

Aunque es un poco más complejo, podríamos decir que, el ámbito local es la parte que está definida dentro de un bloque delimitado por llaves y, el ámbito global es la parte que está fuera de dichas llaves.

Dicho de otra manera. Cuando se accede o utiliza un identificador de variable, primeramente, se busca en la parte del código o bloque que está delimitado por las llaves (el ámbito local). Más tarde, si se no encuentra la declaración de esa variable en ese ámbito, se busca en los ámbitos locales de sus bloques padre hasta llegar al ámbito global, que, como ya veremos, es el objeto global (*window*).

Imaginemos una situación sencilla en la que tenemos unas funciones que operan con una variable externa a las funciones.

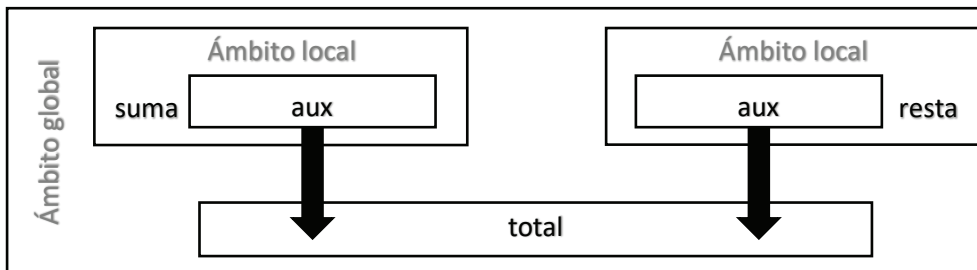
```
var total = 0;

function suma(a, b) {
  var aux = a + b;
  total = aux;
}
```

```
}  
  
function resta(a, b) {  
    var aux = a - b;  
    total = aux;  
}
```

Si observamos el código anterior, podremos ver que, la variable **aux**, se ha definido dentro de los ámbitos locales (los delimitados por las llaves de las funciones) y que, la variable **total** se ha definido en el ámbito global, lo que permite que pueda ser accedida y actualizada desde las funciones suma y resta (las cuales generan un ámbito local dependiente del ámbito global, que es el ámbito padre).

Gráficamente, podríamos decir que es como una pila que va añadiendo elementos y que se caracteriza porque, los elementos que están definidos dentro de un cuadro o bloque, pueden acceder a los elementos que los engloban. Es decir, algo como:



Como la variable **total** ha sido definida en el ámbito global, las funciones suma y resta pueden acceder a la variable y actualizarla. Por tanto, una forma simple de definir el ámbito global es “aquel que puede ser accedido desde cualquier punto del script o programa”.

Como las variables definidas como **aux** están declaradas en bloques delimitados por llaves, su ámbito será local y no podrán ser accedidas o utilizadas desde fuera de su propio ámbito.

En resumen, la declaración y uso de variables se establece de forma jerarquizada en dirección ascendente, es decir, lo que no esté en el nivel actual, será buscado en los niveles superiores y, si no lo encuentra, es cuando se producirá un error de referenciación.

2

TIPOS DE DATOS

JavaScript dispone de dos tipos de datos, primitivos y objeto.

Los tipos de datos primitivos son los que representan un único dato, son inmutables y no tienen métodos.

Los tipos de datos objeto son los que representan una o varias colecciones de datos primitivos y permiten su manipulación a través de propiedades y/o métodos.

En JavaScript, como se verá más adelante, todo son objetos.

2.1 TIPO STRING

El objeto **String** se utiliza para el tratamiento de cadenas de texto. Este tipo, además, provee de un constructor asociado que permite realizar conversiones explícitas.

```
String(4);      // Devuelve "4"
String(0.7);   // Devuelve "0.7"
String(2,4);   // Devuelve "2"
String(true);  // Devuelve "true"
String(String(5)); // Devuelve "5"
String(var);   // Devuelve error de sintaxis
```

2.1.1 Propiedades

El objeto String tiene, esencialmente, tres propiedades:

Propiedad	Descripción
constructor	Devuelve la función constructora nativa.
length	Devuelve la longitud de la cadena
prototype	Permite añadir nuevas propiedades y métodos al objeto.

2.1.2 Métodos

Los valores primitivos también son considerados objetos. Por esta razón, cualquier literal como “Esto es una cadena” puede ejecutar un método o una propiedad.

El número de métodos disponibles para este objeto es elevado, por lo que, a continuación, se muestran los más utilizados:

Método `charAt`

Devuelve el carácter correspondiente a la posición proporcionada por parámetro. Por defecto, la posición es 0.

```
“Hola”.charAt(0); // devuelve “H”
```

Método `charCodeAt`

Devuelve el código Unicode del carácter que corresponda a la posición proporcionada por parámetro. Por defecto, la posición es 0.

```
“Hola”.charCodeAt(0); // devuelve 72
```

Método `concat`

Devuelve otro String que tiene, como resultado, la unión entre de todas las cadenas proporcionadas por parámetro y la actual. El separador de parámetros es el símbolo coma.

```
“Hola”.concat(“ “, “mundo”); // devuelve “Hola mundo”
```


Método `endsWith`

Devuelve un booleano que indica si la cadena termina con la subcadena proporcionada por parámetro.

```
“Hola mundo”.endsWith(“do”); // devuelve true
```

Método `indexOf`

Devuelve la primera posición en la que aparezca la subcadena proporcionada por parámetro. Si el resultado de la búsqueda fue infructuoso, el resultado será -1.

Tiene un segundo parámetro opcional que indica desde qué posición se debe empezar a buscar y que, por defecto, es 0.

NOTA

Este método es sensible a mayúsculas y minúsculas.

```
“Hola mundo”.indexOf(“o”, 0); // devuelve 1
```

Método `lastIndexOf`

Devuelve la última posición en la que aparezca la subcadena proporcionada por parámetro. Si el resultado de la búsqueda fue infructuoso, el resultado será -1.

El método de búsqueda es al revés que el método `indexOf`, es decir, que busca desde el final hasta el principio.

Tiene un segundo parámetro opcional que indica desde qué posición se debe empezar a buscar y que, por defecto, es la longitud de la cadena.

NOTA

Este método es sensible a mayúsculas y minúsculas.

```
“Hola mundo”.lastIndexOf(“o”); // devuelve 9
```

Método match

Permite encontrar coincidencias en una cadena mediante expresiones regulares.

i NOTA

Las expresiones regulares se verán en otro capítulo más adelante.

```
“Hola mundo”.match(/ho/i);  
// devuelve un array con:  
[“Ho”, index: 0, input: “Hola mundo”, groups: undefined]
```

Método normalize

Permite convertir a la forma normal Unicode una cadena pasada como argumento.

La forma normal a elegir tiene cuatro posibles valores:

- ▀ NFC: Forma de Normalización de Composición Canónica.
- ▀ NFD: Forma de Normalización de Descomposición Canónica.
- ▀ NFKC: Forma de Normalización de Composición de Compatibilidad.
- ▀ NFKD: Forma de Normalización de Descomposición de Compatibilidad.

A parte de lo evidente, este método es útil, por ejemplo, para reemplazar acentos.

Por defecto, su valor es NFC.

```
“Ambigüedad inherente”.normalize(‘NFD’).replace(/[\u0300-\u036f]/g, ‘’);  
// devuelve “Ambigüedad inherente”
```

i NOTA

Este método no funciona en Internet Explorer 11.

Método repeat

Devuelve la concatenación de la cadena repetida las veces que se indique por parámetro.

```
“Hola. “.repeat(2); // devuelve “Hola. Hola.
```

Método replace

Permite realizar reemplazos en una cadena a través de otra cadena o una expresión regular.

NOTA

Las expresiones regulares se verán en otro capítulo más adelante.

```
“Hola mundo”.replace(“ mundo”, “”); // devuelve “Hola”  
“Palabra”.replace(/a/ig, “0”); // devuelve “P0l0br0”
```

Método search

Devuelve la posición de la primera aparición de la cadena proporcionada por parámetro.

Aunque este método acepta Strings como parámetro. Lo que utiliza son expresiones regulares. Por esta razón, si se introduce un String, será transformado de forma automática a una expresión regular.

NOTA

Las expresiones regulares se verán en otro capítulo más adelante.

```
“Hola mundo”.search(“mundo”); // devuelve 5
```

Método slice

Devuelve el fragmento de la cadena que esté comprendido entre las posiciones proporcionadas por parámetro.

i NOTA

Este método es muy similar al método `substring`, sin embargo, los resultados pueden ser muy diferentes.

```
“Hola mundo”.slice(0, 4); // devuelve “Hola”
```

Método split

Devuelve un array con todos los fragmentos de cadena que resulten de dividir la cadena origen a través otra cadena o expresión regular proporcionada por parámetro.

```
“Hola mundo”.split(“ “); // devuelve [“Hola”, “mundo”]
```

Método startsWith

Devuelve un booleano que indica si la cadena empieza por el valor proporcionado por parámetro.

Acepta un segundo parámetro que indica dónde se debe empezar a realizar la búsqueda. Por defecto es 0.

```
“Hola mundo”.startsWith(“mundo”); // devuelve false  
“Hola mundo”.startsWith(“mundo”, 5); // devuelve true
```

Método substr

Devuelve el fragmento de cadena que empieza por la posición indicada en el primer parámetro y cuya longitud es el valor proporcionado por el segundo.

```
“Hola mundo”.substr(1,6); // devuelve “ola mu”
```

Método substring

Devuelve el fragmento de cadena que se encuentre entre las posiciones proporcionadas por los parámetros.

i NOTA

Este método es muy similar al método `substring`, sin embargo, los resultados pueden ser muy diferentes.

```
“Hola mundo”.substring(1,6); // devuelve “ola m”
```

Método `toLowerCase`

Devuelve la cadena convertida a minúsculas.

```
“Hola mundo”.toLowerCase(“”); // devuelve “hola mundo”
```

Método `toUpperCase`

Devuelve la cadena convertida a mayúsculas.

```
“Hola mundo”.toUpperCase(“”); // devuelve “HOLA MUNDO”
```

Método `trim`

Devuelve la cadena sin los espacios en blanco que puedan existir en los extremos.

```
“Hola mundo”.trim(); // devuelve “Hola mundo”  
“ Hola mundo”.trim(); // devuelve “Hola mundo”
```

2.1.3 Conversión de Strings

Además de poder realizar conversiones a través de su constructor, el tipo **String** también permite hacer conversiones mediante otras funciones como, por ejemplo, **parseInt** y **parseFloat**, las cuales permiten hacer transformaciones de tipo Strings a tipo número.

```
parseInt(“4”) // Devuelve 4  
parseInt(“hola”) // Devuelve NaN (no es un número)  
parseInt(“21 calles”) // Devuelve 21  
parseInt(“1e3”) // Devuelve 1  
parseFloat(“1.5”) // Devuelve 1.5  
parseFloat(“1,5”) // Devuelve 1  
  
String(new Date()) // Devuelve la fecha actual en formato GMT
```

2.1.4 Formateado de Strings

JavaScript dispone de varias opciones para formatear texto, desde construcciones a través de literales de cadena, hasta secuencias escapadas en hexadecimal o Unicode.

```
/* Literales de cadena */
'Esto es un literal de cadena'
"Esto es otro literal de cadena"

/* Secuencia escapada en hexadecimal */
"\x41" // Devuelve "A"

/* Secuencia escapada en Unicode */
"\u0041" // Devuelve "A"
```

Como se puede apreciar, los literales de cadena no tienen nada de especial, no obstante, el escapado puede ser interesante en varios ámbitos como, por ejemplo, en situaciones donde se necesita mostrar símbolos especiales o iconos.

En ECMAScript 6 existe una forma adicional de escapar texto, mediante el uso de puntos de escape. Esta anotación permite que, cualquier carácter, pueda ser escapado utilizando valores hexadecimales comprendidos entre 0x000000 y 0x10FFFF, o lo que es lo mismo, entre 0 y 1048576. Además, resulta interesante porque evita tener que escribir códigos Unicode dobles.

```
console.log('\u{1F440}', "\uD83D\uDC40");
```

La línea de código anterior, muestra el icono de ojos Emoji de Unicode (👁). La anotación de la izquierda está representada con codificación HTML Entity hexadecimal. La anotación de la derecha está representada con codificación C/C++/Java.

Todos los ejemplos anteriores representan valores en una única línea, sin embargo, también existe la posibilidad de trabajar en modo multilínea.

El modo multilínea se puede realizar de dos formas, con ayuda del símbolo de barra invertida, o a través de literales de plantilla.

```
/* Literales de cadena multilínea (sólo con ES5.1 e inferiores) */
console.log('Nombre: Pablo\n\
Apellidos: Fernández');

/* Literales de plantilla (sólo con ES6 y superiores) */
```