

INFORMÁTICA Y
COMUNICACIONES

UF1305. PROGRAMACIÓN CON LENGUAJES DE GUIÓN EN PÁGINAS WEB

Contenidos basados en los Certificados de profesionalidad



Certia
editorial

**UF1305. PROGRAMACIÓN CON LENGUAJES DE GUIÓN EN
PÁGINAS WEB**

DATOS DEL AUTOR

Ariel Santiago Castro Álvarez es técnico superior en Desarrollo de Aplicaciones Informáticas por el Instituto Politécnico de Vigo, y obtuvo el Certificado de Aptitud Pedagógica en la Universidad de Santiago de Compostela.

Ha dedicado gran parte de su vida profesional a la impartición de cursos sobre lenguajes de programación, desarrollo de páginas web, diseño gráfico, diseño y posicionamiento de blogs, *e-commerce*, *social media marketing*, ofimática, etc. En los últimos años ha elaborado numerosos materiales didácticos para la enseñanza de las TIC a todos los niveles.

El autor participa asiduamente en acciones formativas sobre distintas tecnologías de la información y la comunicación con el fin de mantenerse al día en estas materias, siempre en evolución. En la actualidad desarrolla su actividad profesional dentro del ámbito de las redes informáticas.

FICHA

Programación con lenguajes de guión en páginas web. Informática y comunicaciones

1ª Edición

Certia Editorial, Pontevedra, 2015

Autor: Ariel Santiago Castro Álvarez

Formato: 170 x 240 mm • 301 páginas.

PROGRAMACIÓN CON LENGUAJES DE GUIÓN EN PÁGINAS WEB. INFORMÁTICA Y COMUNICACIONES.

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 u otros métodos, sin el permiso previo y por escrito de los titulares del Copyright.

Derechos reservados 2015, respecto a la primera edición en español, por Certia Editorial.

ISBN: 978-84-16019-98-4

Depósito legal: PO 701-2015

Impreso en España - Printed in Spain

Certia Editorial ha incorporado en la elaboración de este material didáctico citas y referencias de obras divulgadas y ha cumplido todos los requisitos establecidos por la Ley de Propiedad Intelectual. Por los posibles errores y omisiones, se excusa previamente y está dispuesta a introducir las correcciones pertinentes en próximas ediciones y reimpresiones.

Fuente fotografía portada: MorgueFile, autoriza a copiar, distribuir, comunicar públicamente la obra y adaptar el trabajo.

Familia profesional: INFORMÁTICA Y COMUNICACIONES

Área profesional: Desarrollo

FICHA DE CERTIFICADO DE PROFESIONALIDAD (IFCDO110) CONFECCIÓN Y PUBLICACIÓN DE PÁGINAS WEB (RD 1531/2011, de 31 de octubre modificado por el RD 628/2013, de 2 de agosto)

Correspondencia con el Catálogo Modular de Formación Profesional			
H. Q	Módulos certificado	H.CP	Unidades formativas
210	MF0950_2: Construcción de páginas web	210	UF1302: Creación de páginas web con el lenguaje de marcas
			UF1303: Elaboración de hojas de estilo
			UF1304: Elaboración de plantillas y formularios
	MF0951_2: Integración de componentes software en páginas web	180	UF1305: Programación con lenguajes de guión en páginas web
90	MF0952_2: Publicación de páginas web	90	UF1306: Pruebas de funcionalidades y optimización de páginas web
	MP0278: Módulo de prácticas profesionales no laborales	80	
510	Duración horas totales certificado de profesionalidad	560	Duración horas módulos formativos
			480

ÍNDICE

• INTRODUCCIÓN.....	17
• UNIDAD DIDÁCTICA 1. Metodología de la programación	19
1.1. Lógica de la programación	22
1.1.1. Descripción y utilización de operaciones lógicas.....	22
1.1.2. Secuencias y partes de un programa	24
1.2. Ordinogramas.....	26
1.2.1. Descripción de un ordinograma.....	26
1.2.2. Elementos de un ordinograma	27
1.2.3. Operaciones en un programa.....	28
1.2.4. Implementación de elementos y operaciones en un ordinograma.....	32
1.3. Pseudocódigos.....	34
1.3.1. Descripción de pseudocódigo.....	34
1.3.2. Creación del pseudocódigo	39
1.4. Objetos	41
1.4.1. Descripción de objetos.....	42
1.4.2. Funciones de los objetos.....	44
1.4.3. Comportamientos de los objetos	45
1.4.4. Atributos de los objetos.....	45

1.4.5. Creación de objetos	46
1.5. Ejemplos de códigos en diferentes lenguajes	49
1.5.1. Códigos en lenguajes estructurales.....	49
1.5.2. Códigos en lenguajes orientados a objetos	50
1.5.3. Códigos en lenguajes scripts.....	51
RESUMEN.....	53
ACTIVIDADES	55
 • UNIDAD DIDÁCTICA 2. Lenguaje de guión.....	57
2.1. Características del lenguaje	59
2.1.1. Descripción del lenguaje orientado a eventos.....	59
2.1.2. Descripción del lenguaje interpretado	60
2.1.3. La interactividad del lenguaje de guión.....	61
2.2. Relación del lenguaje de guión y el lenguaje de marcas	63
2.2.1. Extensión de las capacidades del lenguaje de marcas.....	64
2.2.2. Adición de propiedades interactivas.....	64
2.3. Sintaxis del lenguaje de guión.....	65
2.3.1. Etiquetas identificativas dentro del lenguaje de marcas.....	65
2.3.2. Especificaciones y características de las instrucciones.....	67
2.3.3. Elementos del lenguaje de guión.....	69
2.3.4. Objetos del lenguaje de guión	76

2.4. Tipos de scripts: inmediatos, diferidos e híbridos.....	82
2.4.1. Script dentro del cuerpo del lenguaje de marcas.....	83
2.4.2. Script dentro del encabezado del lenguaje de marcas	85
2.5. Ejecución de un script.....	87
2.5.1. Ejecución al cargar la página	89
2.5.2. Ejecución después de producirse un evento.....	89
2.5.3. Ejecución del procedimiento dentro de la página.....	89
2.5.4. Tiempos de ejecución.....	89
2.5.5. Errores de ejecución.....	91
RESUMEN.....	93
ACTIVIDADES	95

• UNIDAD DIDÁCTICA 3. Elementos básicos del lenguaje de guión	97
3.1. Variables e identificadores	99
3.1.1. Declaración de variables	99
3.1.2. Operaciones con variables.....	100
3.2. Tipos de datos	102
3.2.1. Datos booleanos	102
3.2.2. Datos numéricos	103
3.2.3. Datos de texto.....	103
3.2.4. Arrays.....	105

3.2.5. Valores nulos y globales	105
3.3. Operadores y expresiones.....	107
3.3.1. Operadores de asignación.....	107
3.3.2. Operadores aritméticos	108
3.3.3. Operadores de comparación.....	110
3.3.4. Operadores sobre bits.....	111
3.3.5. Operadores lógicos	117
3.3.6. Operadores de cadenas de caracteres.....	117
3.3.7. Operadores especiales.....	118
3.3.8. Expresiones de cadena	120
3.3.9. Expresiones aritméticas.....	120
3.3.10. Expresiones lógicas.....	120
3.3.11. Expresiones de objeto.....	121
3.3.12. Precedencia de los operadores	122
3.4. Estructuras de control.....	123
3.4.1. Sentencia IF	124
3.4.2. Sentencia WHILE.....	126
3.4.3. Sentencia FOR.....	127
3.4.4. Sentencia BREAK	130
3.4.5. Sentencia CONTINUE	131
3.4.6. Sentencia SWITCH	132

3.5. Funciones	134
3.5.1. Definición de funciones.....	134
3.5.2. Creación de funciones.....	135
3.5.3. Particularidades de las funciones en el lenguaje de guión....	136
3.5.4. Sentencia RETURN	136
3.5.5. Propiedades de las funciones	137
3.5.6. Funciones predefinidas del lenguaje de guión.....	138
3.6. Instrucciones de entrada/salida.....	145
3.6.1. Descripción y funcionamiento de las instrucciones de entrada y salida.....	145
3.6.2. Sentencia PROMPT	145
3.6.3. Sentencia DOCUMENT.WRITE	146
RESUMEN	149
ACTIVIDADES	151
• EVALUACIÓN 1.....	155
• UNIDAD DIDÁCTICA 4. Desarrollo de scripts	157
4.1. Herramientas de desarrollo, utilización	159
4.1.1. Crear scripts con herramientas de texto	159
4.1.2. Crear scripts con aplicaciones web.....	161
4.1.3. Recursos en la web para la creación de scripts	161

4.2. Depuración de errores: errores de sintaxis y de ejecución	163
4.2.1. Definición de los tipos de errores	163
4.2.2. Escritura del programa fuente	165
4.2.3. Corrección de errores de ejecución.....	165
4.2.4. Corrección de errores de sintaxis.....	168
4.2.5. Compilación del programa fuente.....	169
4.3. Mensajes de error.....	170
4.3.1. Funciones para controlar los errores	171
RESUMEN.....	175
ACTIVIDADES	177

• UNIDAD DIDÁCTICA 5. Gestión de objetos del lenguaje de guión	179
5.1. Jerarquía de objetos.....	181
5.1.1. Descripción de los objetos de la jerarquía.....	183
5.1.2. Propiedades compartidas de los objetos	184
5.1.3. Navegar por la jerarquía de los objetos	186
5.2. Propiedades y métodos de los objetos del navegador.....	189
5.2.1. El objeto superior window.....	190
5.2.2. El objeto navigator	194
5.2.3. URL actual (location)	196
5.2.4. URL visitada por el usuario	197

5.2.5. Contenido del documento actual.....	197
5.3. Propiedades y métodos de los objetos del documento.....	198
5.3.1. Propiedades del objeto document.....	198
5.3.2. Métodos de document	199
5.3.3. Ejemplos de propiedades de document	200
5.3.4. Flujo de escritura del documento.....	204
5.3.5. Métodos open() y close() de document.....	205
5.4. Propiedades y métodos de los objetos del formulario.....	206
5.4.1. Propiedades principales del objeto form.....	206
5.4.2. Métodos del objeto form.....	208
5.5. Propiedades y métodos de los objetos del lenguaje.....	214
5.5.1. Document	215
5.5.2. Window (open).....	220
5.5.3. History (go).....	220
5.5.4. Location (servidor).....	221
5.5.5. Navigator (nombre, versión y detalles del navegador).....	221
RESUMEN.....	223
ACTIVIDADES	225
 • UNIDAD DIDÁCTICA 6. Los eventos del lenguaje de guión	229
6.1. Utilización de eventos	231

6.1.1.	Definición de eventos.....	232
6.1.2.	Acciones asociadas a los eventos.....	233
6.1.3.	Jerarquía de los eventos desde el objeto window.....	239
6.2.	Eventos en elementos de formulario	244
6.2.1.	Onselect.....	247
6.2.2.	Onchange.....	248
6.3.	Eventos de enfoque.....	249
6.4.	Eventos de formulario	250
6.4.1.	Onsubmit.....	251
6.4.2.	Onreset.....	252
6.5.	Eventos de ratón, eventos de teclado	253
6.5.1.	Eventos de ratón.....	253
6.5.2.	Eventos de teclado.....	256
6.6.	Eventos de ventana.....	257
6.6.1.	Onmove.....	258
6.6.2.	Onresize.....	258
6.7.	Otros eventos	259
6.7.1.	Onload.....	259
6.7.2.	Onunload.....	259
6.7.3.	Ondragdrop	260
6.7.4.	Onerror.....	260

6.7.5. Onabort	261
RESUMEN	263
ACTIVIDADES	265
• EVALUACIÓN 2 267	
• UNIDAD DIDÁCTICA 7. Búsqueda y análisis de scripts 271	
7.1. Búsqueda en sitios especializados.....	273
7.1.1. Páginas oficiales.....	273
7.1.2. Tutoriales.....	273
7.1.3. Foros	274
7.1.4. Bibliotecas	275
7.2. Operadores booleanos	275
7.2.1. Funcionamiento de los operadores booleanos.....	276
7.2.2. Utilización en distintos buscadores.....	276
7.3. Técnicas de búsqueda.....	277
7.3.1. Expresiones.....	277
7.3.2. Definiciones de búsqueda.....	278
7.3.3. Especificaciones	279
7.4. Técnicas de refinamiento de búsquedas	281
7.5. Reutilización de scripts.....	283

7.5.1. Scripts gratuitos.....	283
7.5.2. Generalización de códigos.....	285
RESUMEN.....	287
ACTIVIDADES	289
• RESUMEN FINAL.....	293
• EVALUACIÓN FINAL	295
• BIBLIOGRAFÍA/WEBGRAFÍA	297

INTRODUCCIÓN

Esta obra pretende ser una introducción a la programación en lenguajes de guión para páginas web. Se dirige a un tipo de alumnado sin conocimientos previos en lenguajes de programación, aunque sí con ciertas competencias relacionadas con el manejo básico del ordenador y los editores de texto.

Se compone de 7 capítulos y empieza con una aproximación a la metodología de la programación cuyo objetivo es ajustar la perspectiva del alumno al modo de trabajar de un programador. Se trata de conocer fundamentalmente los conceptos de algoritmo, lenguaje de programación, y orientación a objetos y a eventos.

Se describen la sintaxis y los elementos básicos de JavaScript y su interacción con las páginas web a partir del modelo de objetos DOM. En el manual encontraremos algunas actividades para poner en práctica lo aprendido y en ciertos puntos se incluyen pruebas evaluativas.

Es deseo del autor que este texto sirva de material de estudio a quienes deseen acercarse por primera vez a la creación de páginas web y sirva de base, asimismo, a la formación de verdaderos profesionales en este ámbito, capaces de seguir aprendiendo e investigando autónomamente.

METODOLOGÍA DE LA PROGRAMACIÓN

1

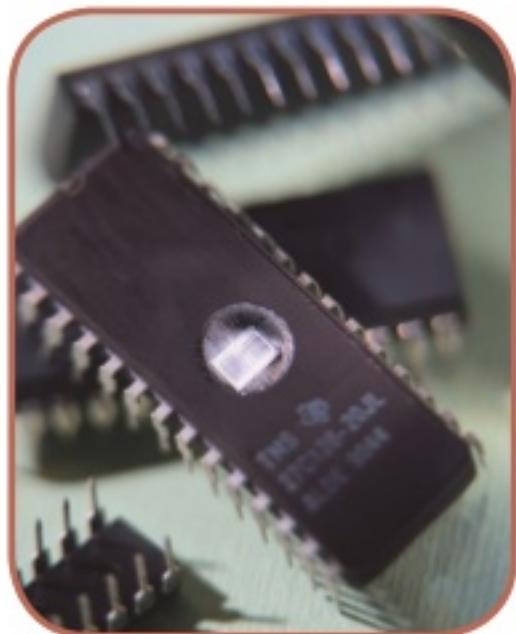
- **CONTENIDO**

- 1.1. Lógica de la programación
- 1.2. Ordinogramas
- 1.3. Pseudocódigos
- 1.4. Objetos
- 1.5. Ejemplos de códigos en diferentes lenguajes

- **RESUMEN**

- **ACTIVIDADES**

El conjunto de métodos necesarios para resolver problemas mediante programas informáticos se conoce como metodología de la programación. Un programa es la serie de instrucciones que le damos a un sistema informático para que alcance el resultado deseado. El programador le dice al sistema lo que debe hacer utilizando un idioma que la máquina entienda, o sea, un lenguaje de programación. Así por ejemplo, el ordenador, siguiendo las directrices codificadas por el programador, solicitará al usuario que introduzca a través del teclado los parámetros del problema, con estos datos realizará a continuación operaciones lógicas y aritméticas, y al final mostrará por pantalla la solución.



Chips informáticos

Un lenguaje de programación es una notación para escribir programas, y viene dado por una gramática o conjunto de reglas que se aplican a un alfabeto. El primer lenguaje de programación que se utilizó fue el *lenguaje máquina*, el único que entiende directamente el ordenador, cuyo alfabeto es el binario (formado por los símbolos 0 y 1). El lenguaje *ensamblador*, resultó de la evolución del lenguaje máquina, al sustituir las cadenas de símbolos binarios por palabras más fáciles de reconocer para nosotros. Se dice de estos lenguajes que son de *bajo nivel* porque son muy distintos del lenguaje natural humano y se parecen más al de las máquinas.

1

1.1. Lógica de la programación

Los ordenadores digitales manejan impulsos eléctricos. Un impulso eléctrico es un *bit*, la mínima cantidad de información o unidad básica que mide la información. Representamos con un 0 y un 1 la ausencia o presencia de un *bit*, lo cual nos permite describir las señales e interconexiones de los circuitos lógicos de un microprocesador por medio del álgebra *booleana*¹, que solo admite dos valores: verdadero y falso.

Obviamente hablar de información en términos de impulsos eléctricos es muy poco intuitivo para el ser humano. Los lenguajes de guión para páginas web pertenecen a un tipo de lenguajes denominados de *alto nivel*, más comprensibles para las personas. Como veremos, utilizan una gramática y un vocabulario en virtud del cual es mucho más fácil decirle a un ordenador lo que queremos que haga por nosotros.

1.1.1. Descripción y utilización de operaciones lógicas

Con todo, los operadores *booleanos* son imprescindibles en cualquier lenguaje de programación. Para empezar a familiarizarnos con el modo de pensar de un ordenador veremos a continuación cómo se utilizan los principales operadores lógicos incluidos en todos los lenguajes de programación.

AND (conjunción)			OR (disyunción)		
a	b	a AND b	a	b	a OR b
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

¹ Denominada así en honor a George Boole (1815-1864), quien la definió como parte de un sistema lógico para tratar las expresiones de la lógica proposicional por medio de técnicas algebraicas. Su obra más importante a este respecto se titula Las leyes del pensamiento de 1854. En http://es.wikipedia.org/wiki/George_Boole.

XOR (disyunción exclusiva)

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

NOT (negación)

a	NOT a
0	1
1	0

Como puedes observar en la tabla, estos operadores lógicos manejan uno o dos valores sobre los que realizan transformaciones como las indicadas. En las tablas *a* y *b* representan un estado igual a 1 o igual a 0, pero en un programa informático podrían representar cualquier expresión que resulte en verdadero o falso.

Operemos por ejemplo con valores numéricos con la ayuda de dos nuevos operadores, en este caso de comparación (serán tratados más adelante), que también producen valores *booleanos*:

- La expresión « $6 > 5$ », que se lee «seis es mayor que cinco», es verdadera.
- La expresión « $8 < 7$ », «ocho es menor que siete», es falsa.

El resultado de combinar estas expresiones de comparación con los operadores lógicos de la tabla anterior devolverá también dos posibles valores: verdadero o falso.

$6 > 5$	AND	$8 < 7$	→ Falso
$6 > 5$	OR	$8 < 7$	→ Verdadero
$6 > 5$	XOR	$8 < 7$	→ Verdadero
NOT($6 > 5$) → Falso		NOT($8 < 7$) → Verdadero	

Fíjate en esta tabla. En la primera fila se utiliza el operador AND, la conjunción, el cual solo resulta verdadero cuando los dos operandos son verdaderos. En nuestro ejemplo, aunque « $6 > 5$ » es verdadero, como « $8 < 7$ » es falso, la conjunción de ambos es falsa.

El operador OR, la disyunción, solo devuelve falso cuando los dos operandos

son falsos, luego el resultado aquí es verdadero, pues « $6 > 5$ » es verdadero y con eso es suficiente.

XOR es la disyunción exclusiva, y es verdadera cuando únicamente uno de los operandos es verdadero. Si los dos son falsos o los dos son verdaderos el resultado será falso. Siguiendo nuestro ejemplo, como la primera expresión es verdadera y la segunda falsa se cumple el criterio de este operador para arrojar un resultado verdadero.

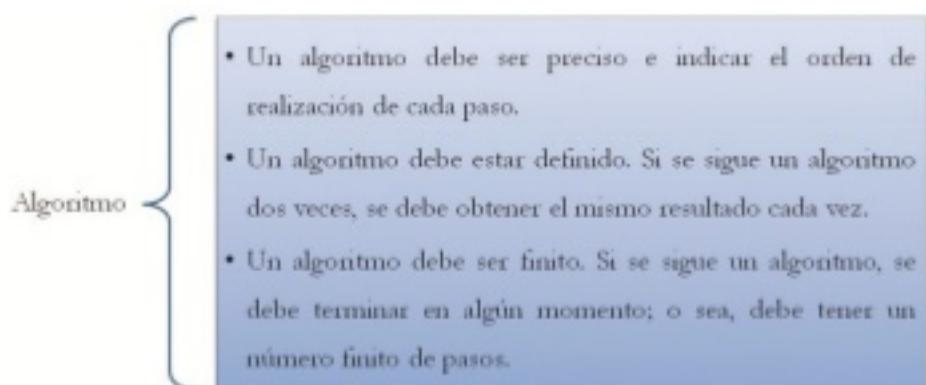
Por último, la negación, el operador NOT, es unario: solo se aplica a una expresión cambiándola de signo. Como « $6 > 5$ » es verdadero, su negación es falsa, y en el caso de la expresión « $8 < 7$ », que es falsa, su negación es verdadera.

1.1.2. Secuencias y partes de un programa

Un programa consiste en una secuencia o sucesión de instrucciones que serán ejecutadas una a una en el orden en el que aparezcan. A su vez, estas instrucciones pertenecerán a las distintas partes del programa, que como veremos son fundamentalmente tres:

- Entrada de datos
- Proceso
- Salida de datos

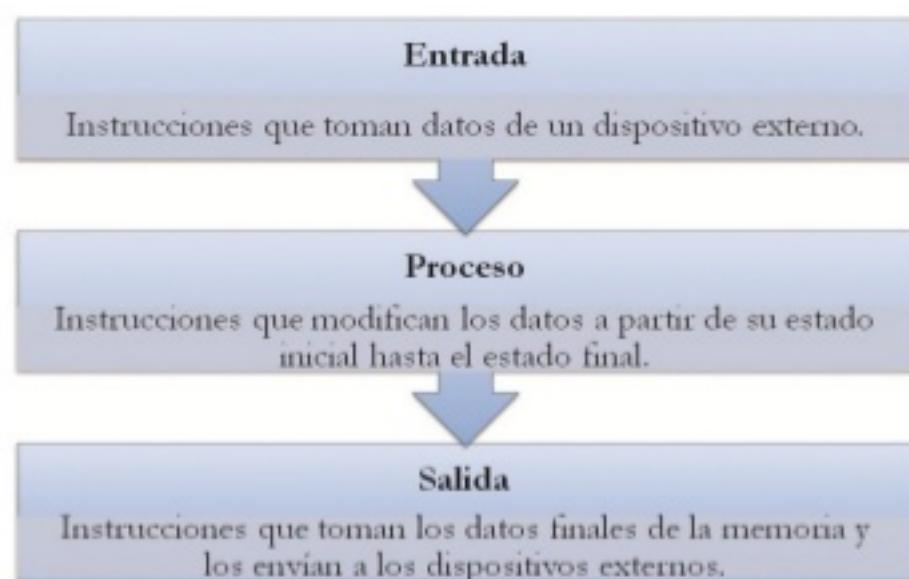
Antes de empezar a codificar en el lenguaje de programación seleccionado, el programador debe tener muy clara la secuencia y el orden de las instrucciones. Aquí se encuentra el núcleo central de la metodología de la programación, resumido en el concepto de algoritmo: una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para realizar una determinada tarea.



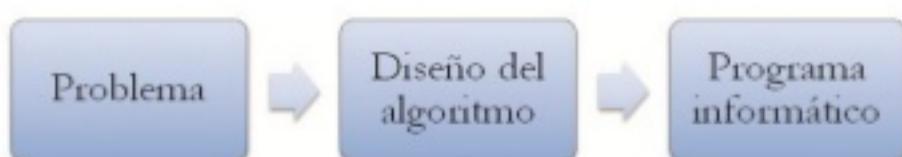
El algoritmo será siempre independiente del lenguaje de programación en el que se implemente y del sistema en la cual vaya a ser ejecutado. El mismo algoritmo se puede expresar en lenguajes de programación diferentes y ser implementado en distintos dispositivos.

También es cierto que es posible diseñar distintos algoritmos para la resolución de un mismo problema. En efecto, concebimos múltiples algoritmos para realizar la misma tarea y escogemos el mejor entre ellos. Para decidirnos por un algoritmo u otro tendremos en cuenta factores como el tiempo que tardará en hallar la solución o los recursos necesarios para implantarlo.

La definición de un algoritmo debe describir tres partes, y estas se corresponden con las partes de un programa informático:



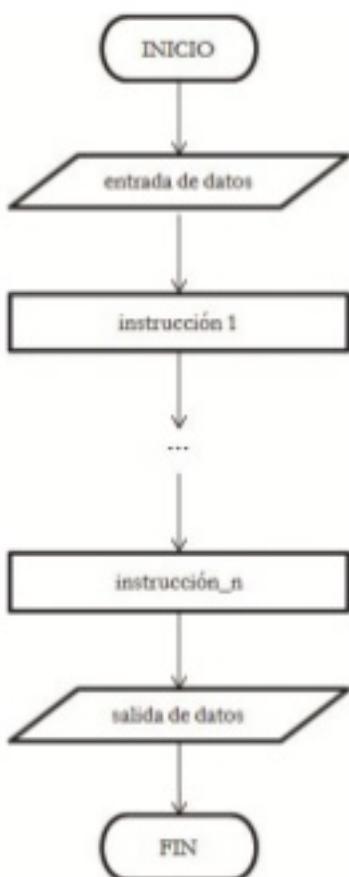
Un programador es en realidad una persona que resuelve problemas, y programar no es otra cosa que el proceso de diseñar, escribir, depurar y mantener el código fuente de programas informáticos para que, posteriormente, realicen por nosotros un trabajo concreto. Con el fin de cumplir adecuadamente con nuestro cometido procederemos de un modo sistemático y riguroso. Solo tras analizar el problema y diseñar el algoritmo que describe la secuencia de pasos que conducen a su solución, lo codificaremos, es decir, lo traduciremos a un lenguaje de programación concreto.



1.2. Ordinogramas

Un programa traduce a un determinado lenguaje de programación uno o más algoritmos. Un algoritmo puede expresarse de distintas maneras: en forma gráfica, como un ordinograma o diagrama de flujo, en forma de código como en pseudocódigo o en un lenguaje de programación, en forma explicativa, etc.

Hemos visto que antes de escribir un programa en un lenguaje de programación específico (C, Pascal, JavaScript, etc.) es conveniente diseñar un algoritmo para definir la secuencia de pasos o acciones que debe llevar a cabo el programa en cuestión. Dicho algoritmo se suele diseñar utilizando pseudocódigo o, también, un ordinograma. El pseudocódigo es un lenguaje de programación intermedio entre el lenguaje natural y un lenguaje de programación concreto. Sin embargo, mediante un ordinograma se puede representar el mismo algoritmo pero de manera gráfica, y este debe ser independiente del lenguaje de programación que utilicemos.



1.2.1. Descripción de un ordinograma

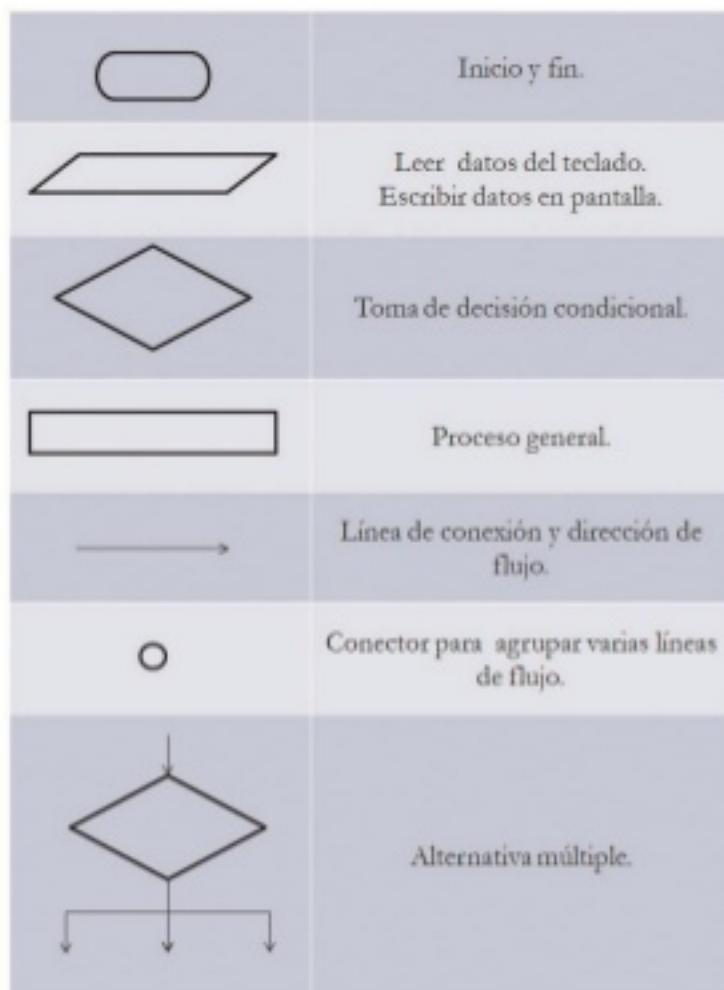
El ordinograma, diagrama de flujo o diagrama de actividades según el

estándar UML² (*Unified Modeling Language*) es la representación gráfica de un algoritmo o proceso. En realidad un ordinograma muestra la secuencia lógica y detallada de las operaciones que necesitamos para la realización cualquier tipo de tarea que pueda ser descrita en una serie de pasos definidos. En la imagen se muestra la estructura básica de un ordinograma.

Todo ordinograma debe estar compuesto, al menos, de un símbolo de inicio, representado por un óvalo, unas flechas que indican el orden de las acciones o instrucciones del algoritmo, simbolizadas por rectángulos, y otro óvalo para indicar el fin de programa.

1.2.2. Elementos de un ordinograma

Estos son algunos de los símbolos básicos para dibujar ordinogramas:



Principales símbolos en un ordinograma

² Visita el sitio web de la organización en su página oficial www.uml.org.

Las reglas básicas para utilizar estos símbolos son las siguientes:

1. Todos los símbolos han de estar conectados.
2. A un símbolo de proceso pueden llegarle varias líneas.
3. A un símbolo de decisión condicional pueden llegarle varias líneas, pero de él solo saldrán dos.
4. A un símbolo de inicio nunca le llegan líneas.
5. De un símbolo de fin no parte ninguna línea.

1.2.3. Operaciones en un programa

Veremos a continuación algunas de las instrucciones, operaciones o estructuras más utilizadas en los programas, así como su representación gráfica mediante los símbolos de ordinograma vistos en el apartado anterior:

- En programación una instrucción indica al sistema la operación u operaciones que este debe realizar con unos datos determinados. Existen muchos tipos de operaciones, de asignación, operaciones aritméticas, lógicas, de comparación, etc. En un diagrama de flujo son representadas por un rectángulo.

instrucción

- Una instrucción de asignación consiste en asociar a una variable el resultado de la evaluación de una expresión. Para representar una instrucción de asignación en un ordinograma utilizamos un rectángulo, y dentro escribimos la expresión y la variable donde se guardará el resultado de su evaluación.

variable \leftarrow expresión

- En un ordinograma, tanto las instrucciones de entrada como las de salida se escriben también en pseudocódigo pero dentro de un romboide. Una instrucción de entrada consiste en asignar a una o más variables uno o más valores recibidos desde el exterior. Normalmente, los datos son

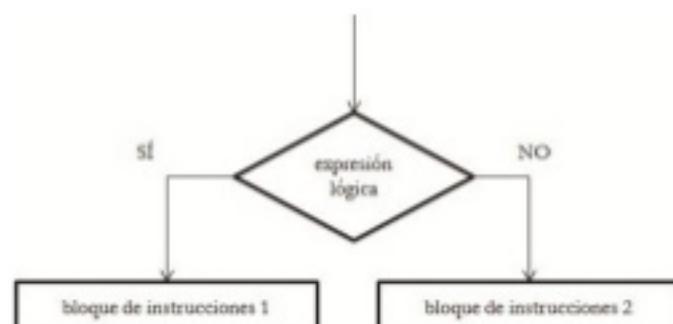
recogidos desde la entrada estándar, el teclado, pero también existen otros dispositivos de entrada: el ratón, el escáner, etc.

leer(variables)

- Una instrucción de salida consiste en llevar hacia el exterior los valores obtenidos como resultado de su procesamiento. Normalmente, los datos son enviados a la salida estándar, la pantalla, pero también existen otros dispositivos de salida: la impresora, el *plotter*, etc. Como la entrada, la salida de datos se representa con un romboide.

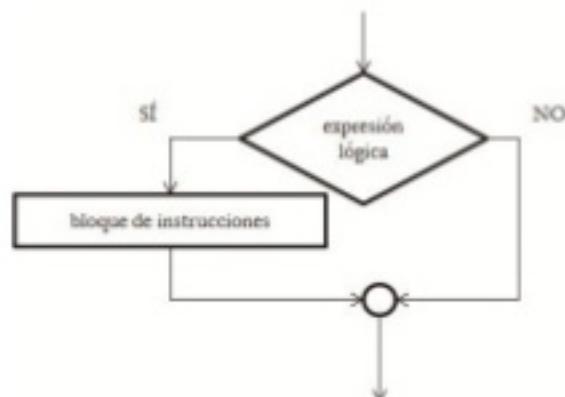
escribir(expresión)

- Una instrucción alternativa doble o condicional se dibuja de la siguiente forma:



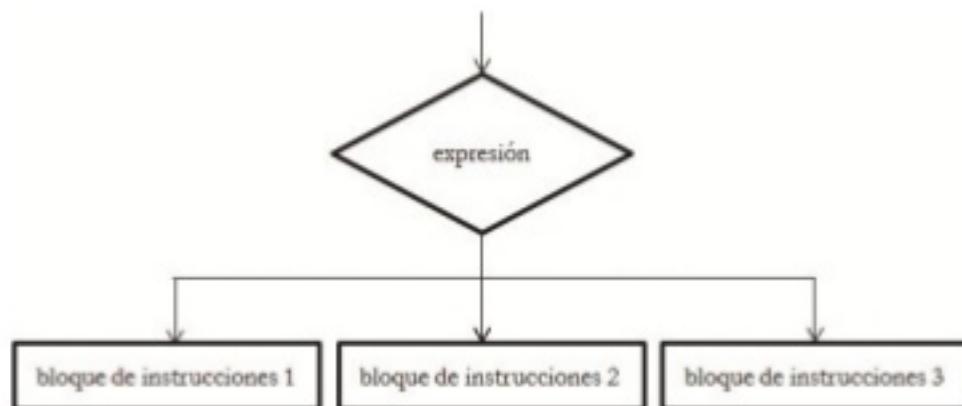
La condición puede ser cualquier expresión que devuelva un valor *booleano*. Para que se ejecute el «bloque de instrucciones 1», tiene que ser verdadera. Si, por el contrario, la condición es falsa se ejecutará el «bloque de instrucciones 2».

- Una instrucción alternativa simple es una variante de una instrucción alternativa doble.

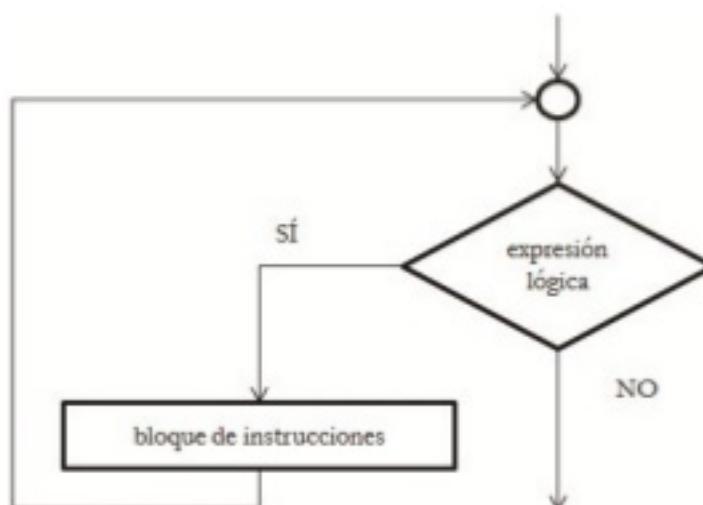


Si la evaluación de la expresión lógica devuelve un valor verdadero, entonces se ejecuta el bloque de instrucciones, si no, continúa el programa.

- Instrucción alternativa múltiple: Una instrucción alternativa múltiple permite seleccionar por medio de una expresión el siguiente bloque de instrucciones a ejecutar de entre varios posibles.



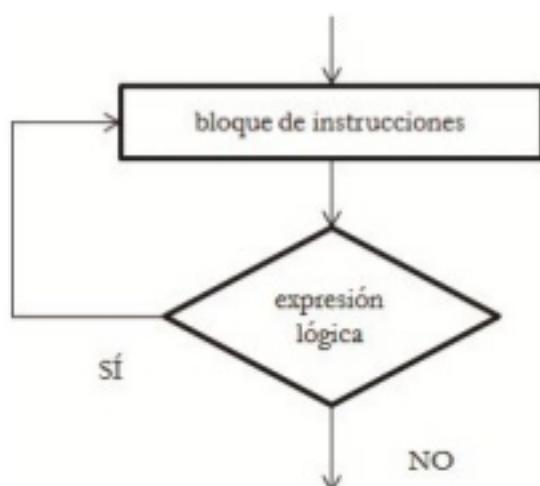
- Estructura iterativa o bucle «mientras».



En un bucle «mientras» se ejecutará el bloque de instrucciones mientras la expresión lógica sea verdadera. Cuando la condición sea falsa, el bloque de instrucciones ya no se ejecutará y el programa saldrá del bucle para continuar su camino. En este tipo de estructuras repetitivas se acostumbra a establecer un contador, esto es, una variable que se inicializa a cero antes del bucle. Dentro del bucle, en cada iteración el contador se irá incrementando o decreciendo hasta llegar a un límite, establecido como condición de salida.

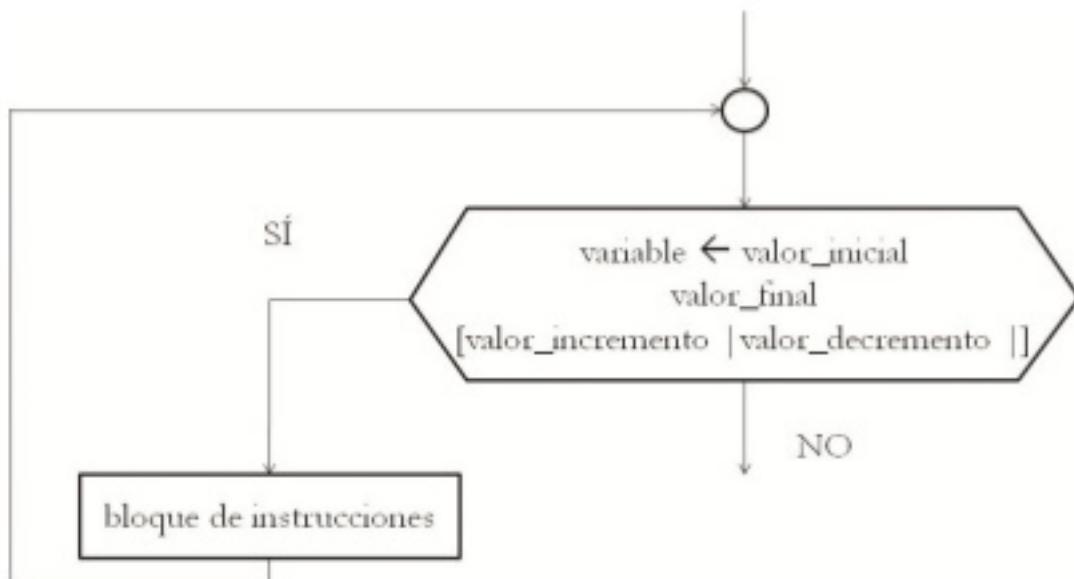
- Una instrucción repetitiva «hacer..mientras» se representa de la siguiente

forma:



En un bucle «hacer...mientras» primero se ejecuta el bloque de instrucciones y después se evalúa la condición. En el caso de que esta sea verdadera, se vuelve a ejecutar el bloque de instrucciones, y así sucesivamente hasta que la condición sea falsa.

- En un ordinograma, una instrucción repetitiva «para» se puede representar del siguiente modo:

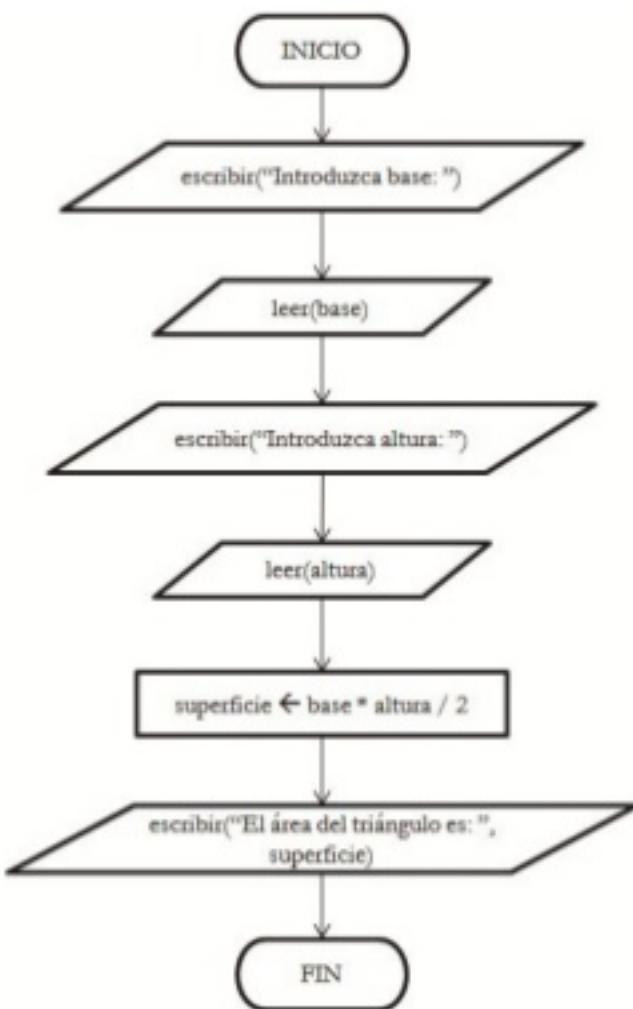


En una instrucción repetitiva «para», siempre se utiliza una variable a la que se debe asignar un valor inicial, un valor final o condición de salida, y el decremento o incremento que experimentará dicha variable por cada iteración. Suele emplearse en situaciones en las que el programador conoce de antemano el número de veces que ha de ejecutarse el bloque de instrucciones.

1.2.4. Implementación de elementos y operaciones en un ordinograma

Veamos algunos ejemplos de ordinogramas. Diseñemos el ordinograma de un programa que calcule el área de un triángulo siguiendo los siguientes pasos:

- 1) Pide por teclado la base.
- 2) Pide por teclado la altura.
- 3) Calcula el área del triángulo.
- 4) Muestra por pantalla el resultado.

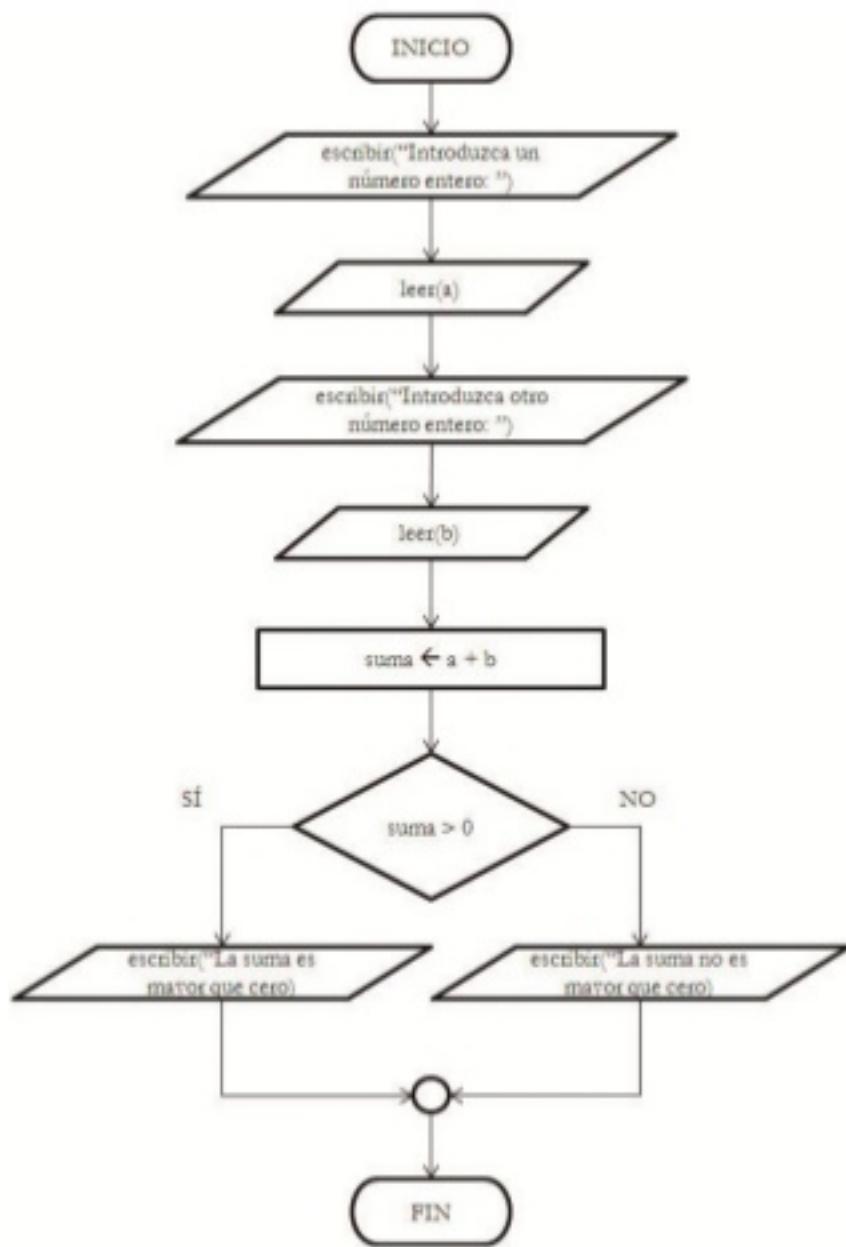


Dibujaremos ahora el ordinograma de un algoritmo un poco más complejo:

- 1) Pide por teclado dos números.
- 2) Calcula la suma de los números introducidos por el usuario.

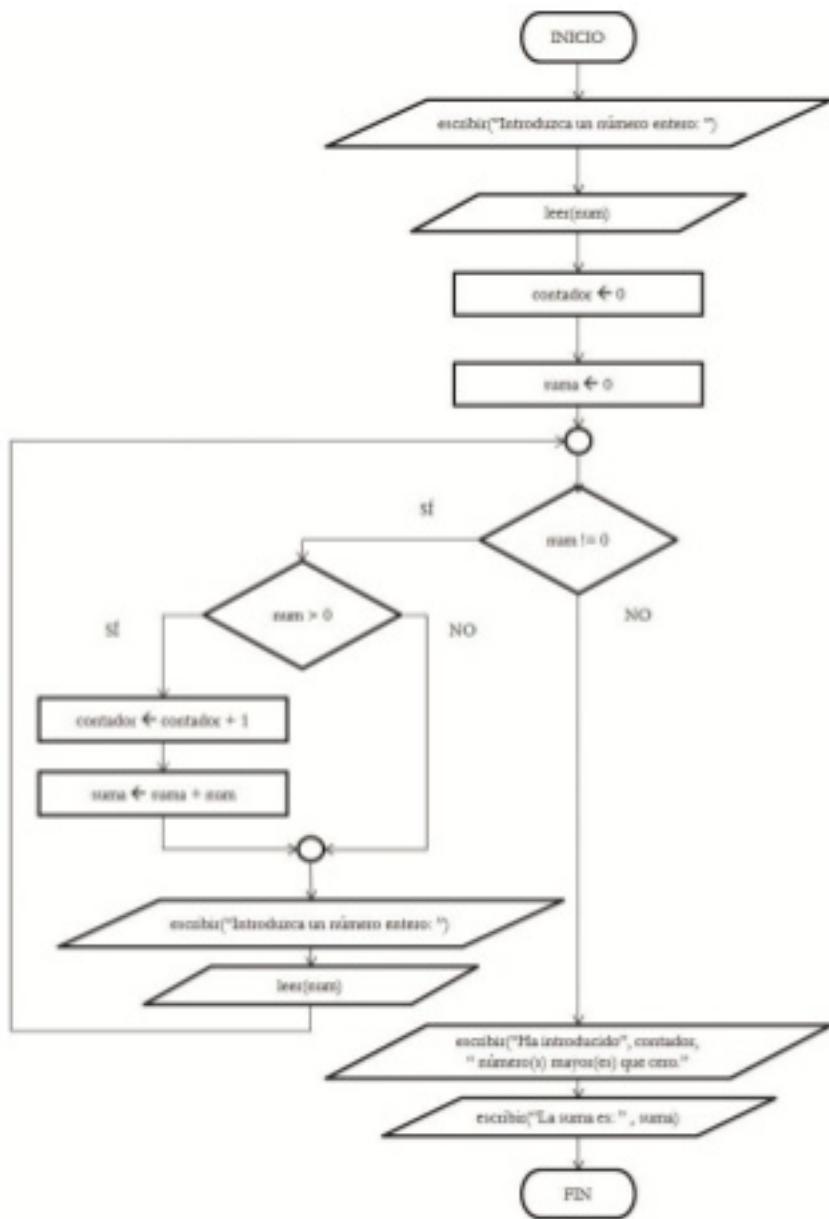
3) Muestra por pantalla estos mensajes:

- «La suma es mayor que cero», en el caso de que sí lo sea.
- «La suma no es mayor que cero», en el caso de que no lo sea.



Diseñemos el ordinograma de un programa que:

- 1) Pida por teclado un número.
- 2) Repita el paso 1 mientras que el número introducido sea distinto de cero.
- 3) Muestre cuántos números mayores que cero han sido introducidos por el usuario y también la suma de todos ellos.



1.3. Pseudocódigos

Utilizando los ordinogramas o diagramas de flujo vimos algunos ejemplos de estructuras de control: condicional simple, condicional de doble alternativa, de alternativa múltiple, estructuras iterativas como el bucle «mientras», el bucle «hacer mientras», el bucle «for», etc. En pseudocódigo, esto puede expresarse utilizando un lenguaje a caballo entre el lenguaje natural y el lenguaje de programación.

1.3.1. Descripción de pseudocódigo

El pseudocódigo es una descripción de un algoritmo informático que utiliza

las convenciones estructurales de un lenguaje de programación en concreto. Como su nombre indica es «casi-código», pero está diseñado para la lectura humana en lugar de la lectura mediante una máquina.

Así, los ejemplos presentados a continuación se asemejan en su sintaxis a la de JavaScript, el lenguaje de guión que estudiaremos en este libro.

- Una sencilla estructura secuencial se describiría en pseudocódigo como sigue:

instrucción_1;

instrucción_2;

.....

instrucción_n;

instrucción

- Para representar la entrada o salida de datos, lo hacemos textualmente de un modo similar a como se expresaba en el interior de los símbolos de los ordinogramas:

leer(variable);

escribir(expresión);

leer(variables)

escribir(expresión)

Te habrás fijado en que todas las sentencias o instrucciones terminan en un punto y coma. Esto se debe a que el lenguaje JavaScript presenta esta sintaxis, cada sentencia debe terminar en un punto y coma.

- Para representar una operación de asignación podemos hacer lo siguiente:

variable = valor;

O bien:

variable ← expresión

variable ← valor;

- Y las operaciones aritméticas, lógicas, etc. se representan, como es habitual, asignando el resultado a una variable donde este será almacenado.

suma ← a + b;

resta ← a - b;

multiplicación $\leftarrow a * b;$

división $\leftarrow a / b;$

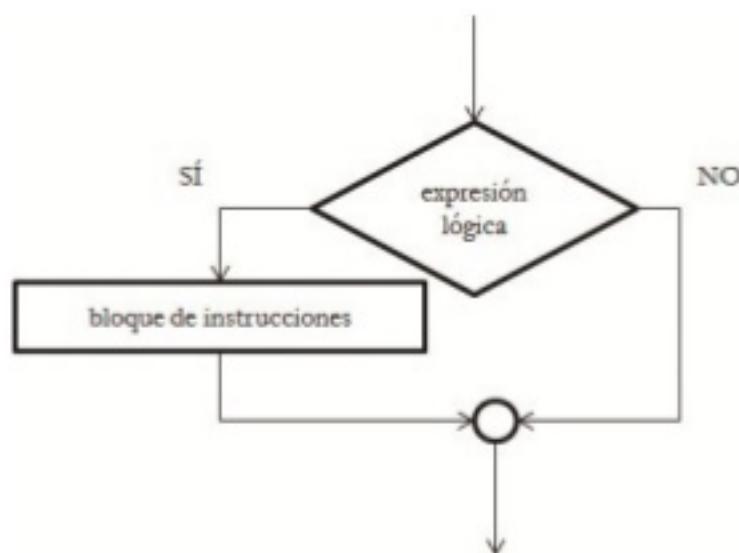
modulo $\leftarrow a \% b;$

- Una estructura de alternativa simple se describiría así:

Si (expresión lógica) Entonces

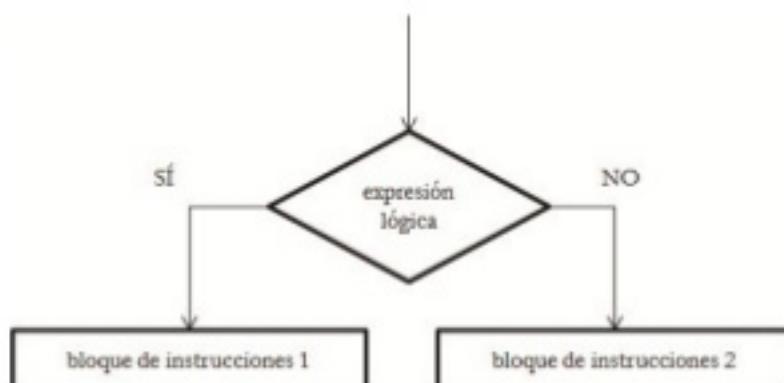
instrucciones;

Fin Si



La primera línea no termina en punto y coma porque todavía no ha finalizado la instrucción. Es una costumbre cuando esto sucede, y como verás cuando tengamos instrucciones anidadas dentro de otras estructuras, sangrar o identar por bloques el código para hacerlo más comprensible.

- Una estructura de alternativa doble se representaría así en pseudocódigo:



Si (expresión lógica) Entonces

instrucciones1;

Si no

instrucciones2;

Fin Si

- La selección múltiple se representa del modo siguiente:

Según (Variable) Hacer

Caso Valor_1

instrucciones_1;

Fin Según

Caso Valor_2

instrucciones_2;

Fin Según

...

Caso Valor_n

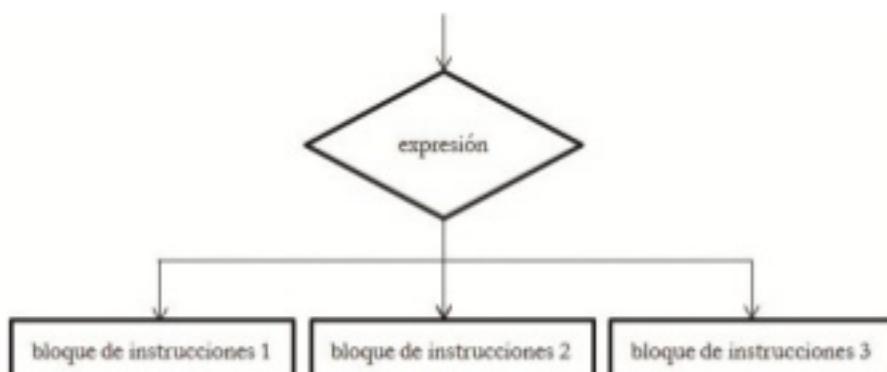
instrucciones_n

Fin Según

Por defecto

instrucciones_defecto;

Fin Según

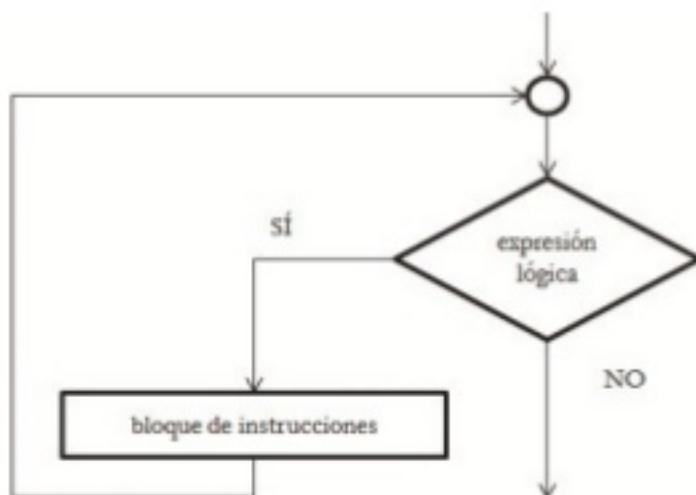


- Veamos las estructuras iterativas o de repetición.

Mientras (expresión lógica) hacer

instrucciones;

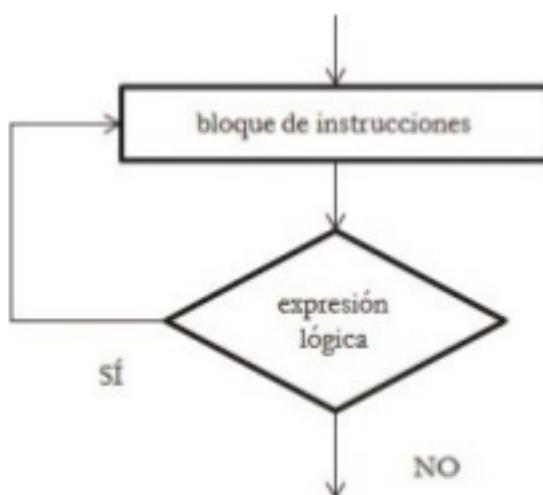
Fin Mientras



Hacer

instrucciones;

Mientras (expresión lógica)

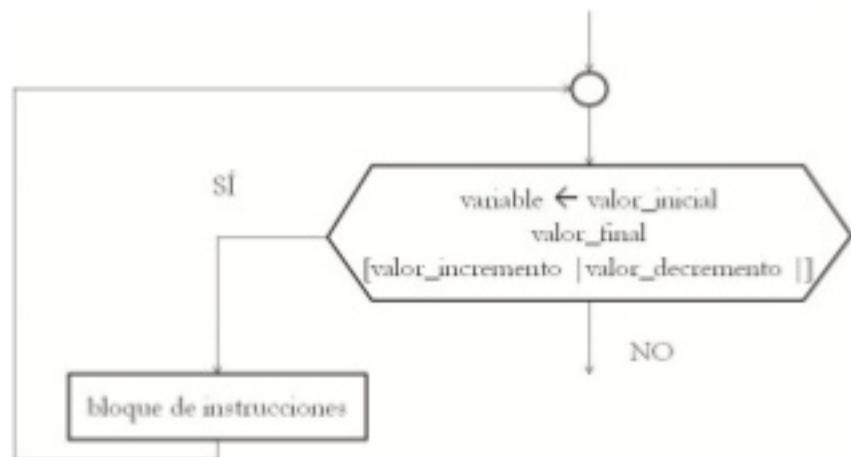


- Una estructura de control muy común es el ciclo «para», que se usa cuando se desea repetir el bloque de instrucciones un número conocido de veces empleando como índice una variable que se incrementa (o decrece) en cada vuelta del bucle:

Para $i \leftarrow x$ Hasta y con Paso z Hacer

instrucciones;

Fin Para



Con pseudocódigo también podemos anidar estructuras, por ejemplo una alternativa simple dentro de un bucle:

Mientras (expresión lógica) Hacer

Si (expresión lógica) Entonces

instrucciones;

Fin Si

Fin Mientras

1.3.2. Creación del pseudocódigo

Veamos cómo quedaría el pseudocódigo de un algoritmo ya representado mediante un ordinograma en el apartado anterior. ¿Serías capaz de identificarlo?

escribir('Introduzca un número entero: ');

leer(num);

contador ← 0;

suma ← 0;

Mientras (num != 0) Hacer

Si (num > 0) Entonces

contador ← contador + 1;

```

    suma ← suma + num;
    Si no
        escribir("Introduzca un número entero: ");
        leer(num);
    Fin Mientras
        escribir("Ha introducido", contador, " número(s) mayor(es) que cero");
        escribir("La suma es: ", suma);

```



Como puedes ver, dentro del bucle «mientras» tenemos un «contador» que únicamente se incrementa en 1 cuando se cumple la condición del condicional, es decir, solo cuenta cuando el número introducido por el usuario es distinto de 0. La variable «suma» es un acumulador, que también va actualizándose en cada vuelta solo cuando el número «num» es mayor que cero, sumando a su anterior valor el nuevo número introducido.

El pseudocódigo tiene más ventajas que los ordinogramas para representar un programa. Ocupa mucho menos espacio en el desarrollo del problema, permite representar de forma fácil operaciones repetitivas complejas y es más sencilla la tarea de pasar del pseudocódigo a un lenguaje de programación formal. Además, si se siguen las reglas de *identación* (la sangría del código que puedes observar en el ejemplo anterior), es posible distinguir claramente los niveles en la estructura del programa. En definitiva, el pseudocódigo está más cerca del paso siguiente que tenemos que dar en el proceso de aprendizaje: la codificación en un lenguaje determinado, en nuestro caso JavaScript.

En pseudocódigo, por ejemplo, es muy sencillo representar una *función* o procedimiento. Una *función* es un bloque de instrucciones identificada por medio de un nombre. Posteriormente, mediante dicho nombre, la función será

invocada durante la ejecución del programa para que incluya las instrucciones que contiene sin tener que volver a escribirlas cada vez que sean necesarias.

Veamos un ejemplo: la función «Potencia» calcula la potencia de un número « a » elevado a « b »:

Función Potencia (a, b)

Si $b == 0$ Entonces

$p \leftarrow 1;$

Si no

$p \leftarrow a;$

Para $i \leftarrow 2$ Hasta b con paso 1 Hacer

*$p \leftarrow p * a;$*

Fin Para

Fin Si

Devolver $p;$

Fin Función

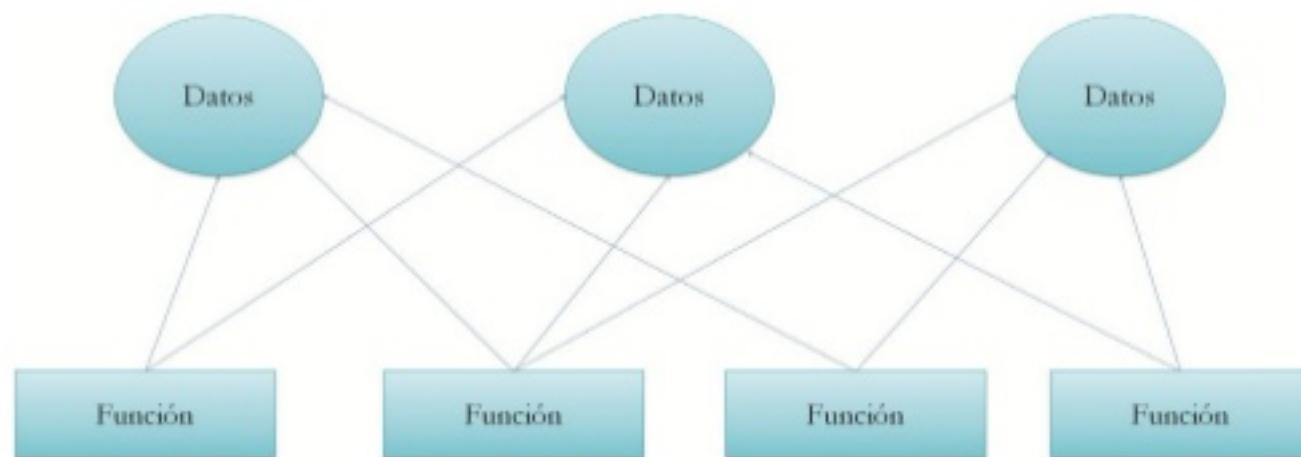


1.4. Objetos

La programación orientada a objetos (POO) es el nuevo paradigma (conjunto de principios metodológicos y de organización) en el que se basan todos los lenguajes modernos. Nace como respuesta a las dificultades a las que se enfrentaban los anteriores enfoques, como la programación procedural, empleada en los años ochenta, que utilizaba lenguajes como C, Pascal o Fortran.

para crear programas como secuencias de instrucciones o sentencias que la computadora *compilaba*, traducía al lenguaje máquina, y ejecutaba una a una.

El modelo procedimental se mostraba eficiente para programas pequeños pero no para programas complejos donde el desarrollador de *software* había de vérselas con un elevado número de líneas de código muy difíciles de controlar cuando los programas se hacían más grandes.



Para resolver este problema los programas grandes se dividieron en partes, subprogramas o funciones que se ocuparían de una tarea definida, a la vez que se comunicaban entre ellas. Las funciones se agruparon en módulos o ficheros cada vez mayores, siguiendo sin embargo el mismo principio, ejecutar listas de instrucciones.

A medida que los programas crecían, este paradigma, denominado estructural manifestaba sus limitaciones. Las conexiones entre datos y funciones se hicieron cada vez más intrincadas y el código del programa cada vez más difícil de mantener, es decir, de corregir, modificar o ampliar.

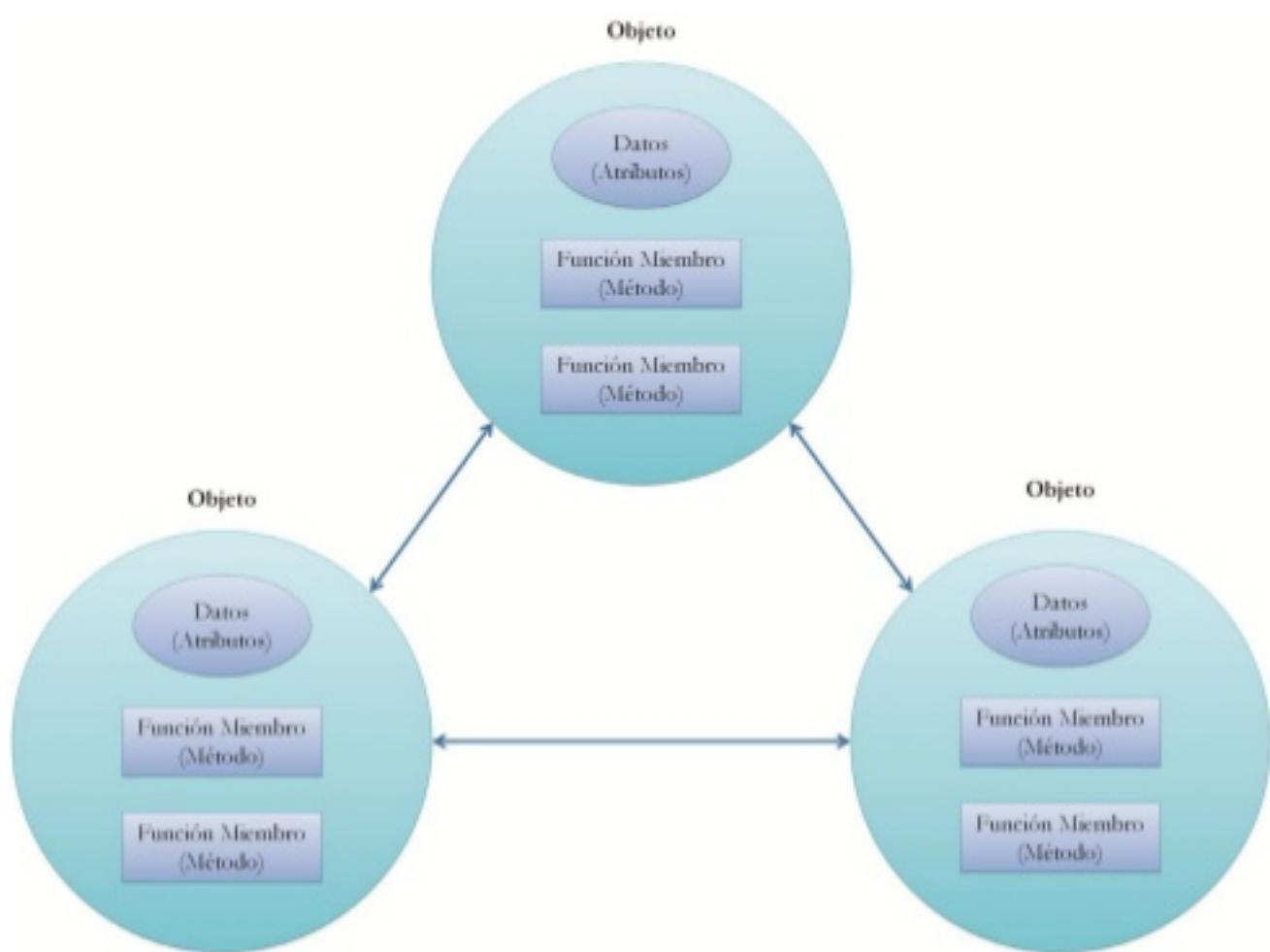
1.4.1. Descripción de objetos

El objeto es el núcleo de la programación orientada a objetos. Para la POO prácticamente todo son objetos. Por ejemplo, veremos que en JavaScript la ventana donde se carga la página web es un objeto, y es un objeto la propia página o documento. Los elementos de texto que contiene, las imágenes, formularios, capas, etc., son también objetos.

Los datos de un objeto son sus propiedades o atributos, y las funciones que los procesan, denominadas métodos, gestionan su comportamiento. Dependiendo

del problema diseñamos un objeto con unas propiedades y unos métodos determinados para que juegue un cierto papel en la resolución del mismo.

El paradigma de la programación orientada a objetos (POO), como ya se ha dicho, surge como respuesta a los retos que se le presentan a la programación procedural y estructurada cuando se enfrenta a problemas complejos. Es uno de los enfoques de programación más utilizados en la actualidad y constituye una nueva forma de entender la programación: ya no se trata de adaptar un problema a un lenguaje, sino de ajustar el lenguaje al problema, a la realidad.



Con la programación estructural se trataba de crear funciones para procesar datos sin tener en cuenta su estructura, sin establecer una conexión esencial entre ambos. En la POO se enfatizará la estructura de los datos, combinando datos y funciones en una unidad a la que se denomina *objeto*.

Cada objeto se diferencia de otros por su estructura o sencillamente por el valor de sus propiedades. Es a raíz de esto que se utiliza el concepto de *instancia* de un objeto. En lenguajes como Java o PHP se introduce el concepto de

clase, donde se diseña un patrón o plantilla que servirá de modelo a los objetos construidos a partir de la misma. Son como los planos donde se define qué tipo de propiedades van a tener todas las piezas construidas según sus directrices. La clase es un molde, y cada objeto un ejemplar o instancia realizado en la práctica, ocupando la memoria principal con una serie de datos concretos durante la ejecución del programa.

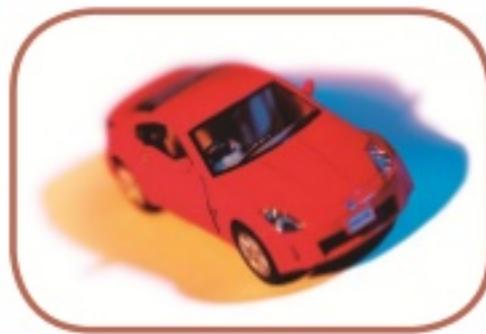
Como veremos, en JavaScript no se definen clases sino objetos, ejemplares o prototipos, y a partir de ellos, mediante una función constructora, se generan nuevas instancias con sus propios atributos y comportamientos³.

1.4.2. Funciones de los objetos

Decíamos que para el enfoque procedimental y estructurado programar era escribir funciones que procesan datos. Se centraba más en los algoritmos, mientras las estructuras de los datos pasaban a un segundo plano, y datos y funciones eran tratados por separado. Este planteamiento hacía imposible representar o modelar las cosas del mundo real, ya que estas poseen una identidad con sus atributos y su comportamiento.

Esta es la función más importante de los objetos: representar las cosas del mundo real. De este modo cualquier cosa es susceptible de ser modelada como un objeto, desde entidades concretas como automóviles o personas, hasta cosas abstractas como pueden ser un proceso o un evento.

Un coche, por ejemplo, posee unas propiedades que lo distinguen de otros (matrícula, color, potencia, posición), y un comportamiento (arrancar, acelerar, girar, frenar, etc.).



³Véase Mozilla Developer Network, Introducción a JavaScript orientado a objetos [Internet]. Disponible en: https://developer.mozilla.org/es/docs/JavaScript/Introducci%C3%B3n_a_JavaScript_orientado_a_objetos [Fecha de acceso: 1 abril 2014]