 Liceo Compu-Market

Catedrático: Erick González

Cátedra: Practica Supervisada

Temas:

* Programación

Nombre Del Estudiante: Selvin Gustavo

González Rivera

Grado: 5to Bachillerato En Computación Con Orientación Científica

Clave: 13 Sección: “B”

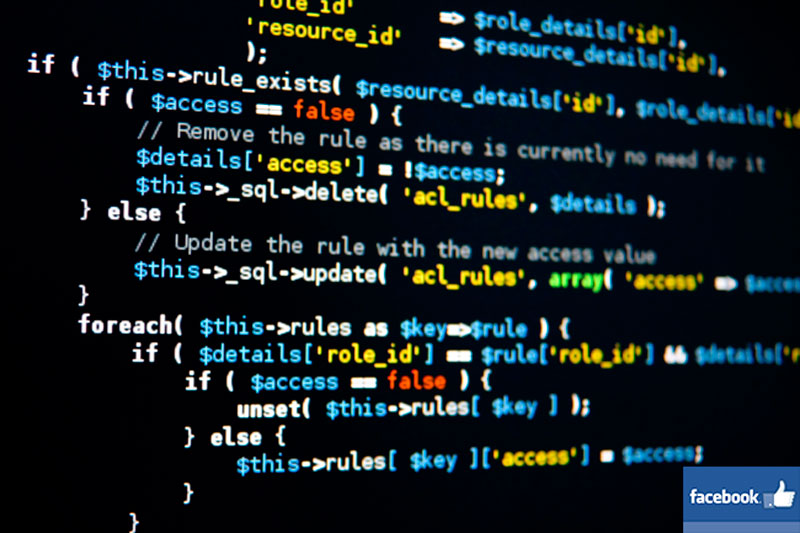
**ÍNDICE**

1. Carátula……………………………………………………………………………………………………………………………..[1](#cara)
2. Índice…………………………………………………………………………………………………………………………………[2](#indice)
3. Introducción………………………………………………………………………………………………………………………[3](#intro)
4. Comentarios personales…………………………………………………………………………………………..……….[4](#comen)
5. Programación ..………………………………………………………………………………………………………………....[5](#progra)
6. Historia de la programación……………….…………………………………………………………………………..[6-7](#hisdeprogra)
7. Programa informático…….…………………………………………….………………………………………………..[7-8](#prograinfo)
8. Codigo fuente……………….……………………………………………………………………………………………..[9-10](#codfuente)
9. Ejecutable ………….………………………………………………………………………………………….………….[10-11](#ejecutable)
10. Lenguaje de programación……………………………………………….………………………………………..[11-14](#lenguajesdeprogra)
11. Lenguaje formal………………………………………………………………………………………………………….[14-21](#lenguaformal)
12. Ecuaciones del Círculo………………………………………………………………………………………………[21-23](#depuprogramas)
13. Ecuaciones de la Recta……………………………………………………………………………………………..[23-33](#errorsoft)
14. ……………………………………………………………………………………………………………………….[33](#semaestatica)
15. Hipérbola………………………………………………………………………………………………………………[34-38](#_Sistema_de_tipos)
16. Funciones…………………………………………………………………………………………………………………[38-43](#prograestructurada)
17. Dominio y Rango de una Función……………………………………………………………………………..[43-50](#poo)
18. Función Creciente, Decreciente y Constante…………………………………………………………….[50-51](#prograimperativa)
19. Función Par e Impar…………………………………………………………………………………………………[51-52](#pih)
20. Función Lineal y Cuadrática………………………………………………………………………………………[52-53](#lenguajesdeprogra)
21. Función Polinómica y Racional…………………………………………………………………………………[53-54](#perl)
22. Función Exponencial y Logarítmica…………………………………………………………………………..[55-57](#java)
23. Función Seno y Coseno…………………………………………………………………………………………[57-66](#html)
24. Leyes de Senos……………………………………………………………………………………………………..[66](#html5)
25. Leyes de Coseno…………………………………………………………………………………………………..[67-69](#php)
26. Identidades Trigonométricas………………………………………………………………………………..[69-74](#javascript)
27. Conclusiones…………………………………………………………………………………………………………………[75-87](#asp)
28. Comentario Personal……………………………………………………………………………………………………..[87-93](#b4ad)
29. Egrafia…………………………………………………………………………..……………………………………..[93-95](#paradigmadeprogra)
30. Anexos…………………………………………………………………………………………………………………………..[95-97](#lenguajedeprograesoterico)
31. [98-100](#prograenhard)
32. [100-101](#ejecsimultanea)
33. [101-106](#_Algoritmo)
34. [106-114](#diagradeflujo)
35. [116-117](#_Programación_e_ingeniería)
36. [118-119](#ciclovidasoftware)
37. [119-120](#desarrolloagilsoft)
38. [121-127](#desarrolloagilsoft)
39. [**127-128**](#objetivosdeprogra)
40. **[129](#conclusion)**

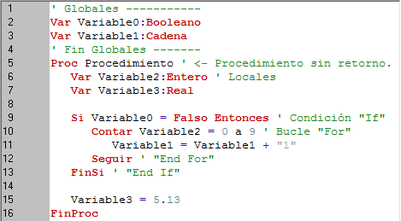
**Introduccion****comentarios personales:**

**Programación**

**La programación informática o programación algorítmica, acortada como programación, es el proceso de diseñar, codificar, depurar y mantener el código fuente de programas de computadora. El código fuente es escrito en un lenguaje de programación. El propósito de la programación es crear programas que exhiban un comportamiento deseado. El proceso de escribir código requiere frecuentemente conocimientos en varias áreas distintas, además del dominio del lenguaje a utilizar, algoritmos especializados y lógica formal. Programar no involucra necesariamente otras tareas tales como el análisis y diseño de la aplicación (pero sí el diseño del código), aunque sí suelen estar fusionadas en el desarrollo de pequeñas aplicaciones.**



## Historia

**[](https://commons.wikimedia.org/wiki/File:Pauscal_lenguaje_de_programaci%C3%B3n.png)**

**Para crear un programa, y que la computadora lo interprete y ejecute las instrucciones escritas en él, debe escribirse en un lenguaje de programación. En sus inicios las computadoras interpretaban solo instrucciones en un lenguaje específico, del más bajo nivel, conocido como código máquina, siendo éste excesivamente complicado para programar. De hecho solo consiste en cadenas de números 1 y 0 (sistema binario). Para facilitar el trabajo de programación, los primeros científicos, que trabajaban en el área, decidieron reemplazar las instrucciones, secuencias de unos y ceros, por palabras o abreviaturas provenientes del inglés; las codificaron y crearon así un lenguaje de mayor nivel, que se conoce como Assembly o lenguaje ensamblador. Por ejemplo, para sumar se podría usar la letra A de la palabra inglesa *add* (sumar). En realidad escribir en lenguaje ensamblador es básicamente lo mismo que hacerlo en lenguaje máquina, pero las letras y palabras son bastante más fáciles de recordar y entender que secuencias de números binarios. A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método sencillo para programar. Entonces, se crearon los lenguajes de alto nivel. Mientras que una tarea tan trivial como multiplicar dos números puede necesitar un conjunto de instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará con solo una. Una vez que se termina de escribir un programa, sea en ensamblador o en algunos lenguajes de alto nivel, es necesario compilarlo, es decir, traducirlo completo a lenguaje máquina.**[**1**](https://es.wikipedia.org/wiki/Programaci%C3%B3n#cite_note-1)**Eventualmente será necesaria otra fase denominada comúnmente *link* o enlace, durante la cual se anexan al código, generado durante la compilación, los recursos necesarios de alguna biblioteca. En algunos lenguajes de programación, puede no ser requerido el proceso de compilación y enlace, ya que pueden trabajar en modo intérprete. Esta modalidad de trabajo es equivalente pero se realiza instrucción por instrucción, a medida que es ejecutado el programa.**



# Programa informático

**Un programa informático o programa de computadora es una secuencia de instrucciones, escritas para realizar una tarea específica en una computadora. Este dispositivo requiere programas para funcionar, por lo general, ejecutando las instrucciones del programa en un procesador central.El programa tiene un formato ejecutable que la computadora puede utilizar directamente para ejecutar las instrucciones. El mismo programa en su formato de código fuente legible para humanos, del cual se derivan los programas ejecutables (por ejemplo, compilados), le permite a un programador estudiar y desarrollar sus algoritmos. Una colección de programas de computadora y datos relacionados se conoce como software.** 

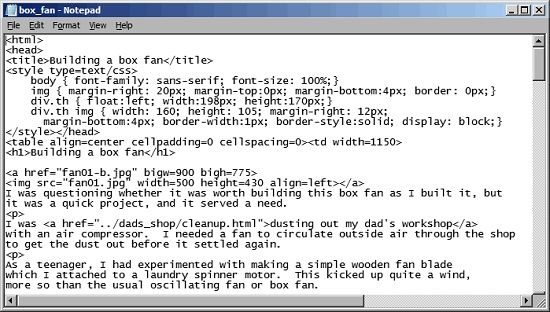
**Generalmente, el código fuente lo escriben profesionales conocidos como programadores de computadora.**[**3**](https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico#cite_note-3)**Este código se escribe en un lenguaje de programación que sigue uno de los siguientes dos paradigmas: imperativo o declarativo, y que posteriormente puede ser convertido en un archivo ejecutable (usualmente llamado un programa ejecutable o un binario) por un compilador y más tarde ejecutado por una unidad central de procesamiento. Por otra parte, los programas de computadora se pueden ejecutar con la ayuda de un intérprete, o pueden ser empotrados directamente en hardware.**

**De acuerdo a sus funciones, los programas informáticos se clasifican en software de sistema y software de aplicación. En las computadoras de 2015, al hecho de ejecutar varios programas de forma simultánea y eficiente, se lo conoce como multitarea.** 

# 

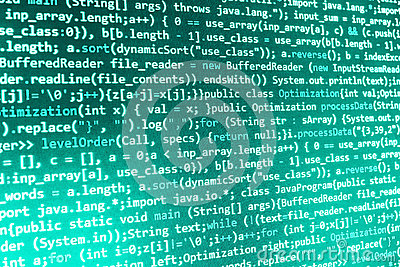
# Código fuente

**El código fuente de un programa informático (o software) es un conjunto de líneas de texto con los pasos que debe seguir la computadora para ejecutar dicho programa.**



El código fuen**te de un programa está escrito por un programador en algún lenguaje de programación, pero en este primer estado no es directamente ejecutable por la computadora, sino que debe ser traducido a otro lenguaje o código binario; así será más fácil para la máquina interpretarlo (lenguaje máquina o código objeto que sí pueda ser ejecutado por el hardware de la computadora). Para esta traducción se usan los llamados compiladores, ensambladores, intérpretes y otros sistemas de traducción.**

**El término código fuente también se usa para hacer referencia al código fuente de otros elementos del software, como por ejemplo el código fuente de una página web, que está escrito en lenguaje de marcadoHTML o en Javascript, u otros lenguajes de programación web, y que es posteriormente ejecutado por el navegador web para visualizar dicha página cuando es visitada.**

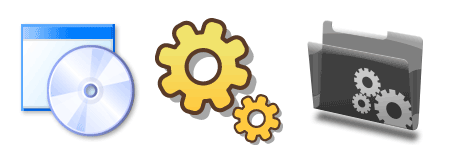


**El área de la informática que se dedica a la creación de programas, y por tanto a la creación de su código fuente, es la ingeniería de software.**

# Ejecutable

**En informática, un ejecutable o archivo ejecutable, es tradicionalmente un archivo binario, o como se le conoce habitualmente, cuyo contenido se interpreta por el ordenador como un programa.**

**Generalmente, contiene instrucciones en código máquina de un procesador en concreto, pero también puede contener bytecode que requiera un intérprete para ejecutarlo. Además, suele contener llamadas a funciones específicas de un sistema operativo (*llamadas al sistema*).**



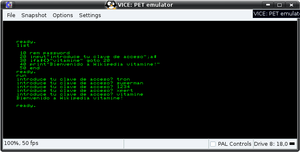
**Dependiendo del tipo de instrucciones de que se traten, hablaremos de ejecutables portables (se pueden ejecutar en varias plataformas) y no portables (destinado a una plataforma concreta). Por ejemplo, un ejecutable Java es portable ya que utiliza un *bytecode* no asociado a un procesador en concreto.**

**Sin embargo en un sentido más general, un programa ejecutable no tiene por qué necesariamente contener código de máquina, sino que puede tener instrucciones a interpretar por otro programa. Este tipo de ejecutables son conocidos con el nombre de *scripts*.**

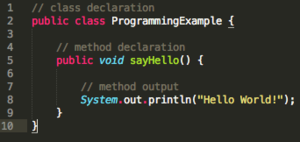
**Determinar si un archivo es ejecutable es sobre todo una cuestión de convención. Unos sistemas operativos se basan en la extensión de archivo (como la terminación *.exe*) y otros lo hacen leyendo los metadatos (como los bits de permiso de ejecución en Unix).**

**En la mayoría de los sistemas modernos, un archivo ejecutable contiene mucha información que no es parte del programa en sí: recursos como textos e imágenes, requisitos del entorno de ejecución, información simbólica y de depuración, u otra información que ayude al sistema operativo a ejecutar el programa.** 

# Lenguaje de programación

**[](https://commons.wikimedia.org/wiki/File:PET-basic.png)**

**Captura de la microcomputadora Commodore PET-32 mostrando un programa en el lenguaje de programación BASIC, bajo el emulador VICE en una distribución GNU/Linux.**

**[](https://commons.wikimedia.org/wiki/File:Classes_and_Methods.png)**

**Un ejemplo de código fuente escrito en el lenguaje de programación Java, que imprimirá el mensaje "Hello World!" a la salida estándar cuando es compilado y ejecutado**

**Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras.**

**Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.**

**Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación.**

**También la palabra programación se define como el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos, a través de los siguientes pasos:**

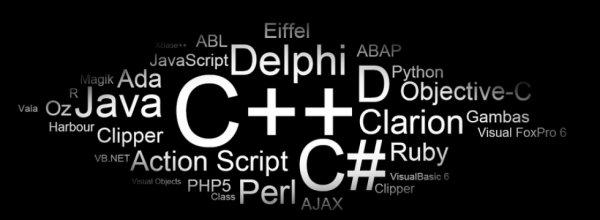
* **El desarrollo lógico del programa para resolver un problema en particular.**
* **Escritura de la lógica del programa empleando un lenguaje de programación específico (codificación del programa).**
* **Ensamblaje o compilación del programa hasta convertirlo en lenguaje de máquina.**
* **Prueba y depuración del programa.**
* **Desarrollo de la documentación.**
* 

**Existe un error común que trata por sinónimos los términos 'lenguaje de programación' y 'lenguaje informático'. Los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como por ejemplo HTML (lenguaje para el marcado de páginas web que no es propiamente un lenguaje de programación, sino un conjunto de instrucciones que permiten estructurar el contenido de los documentos).**

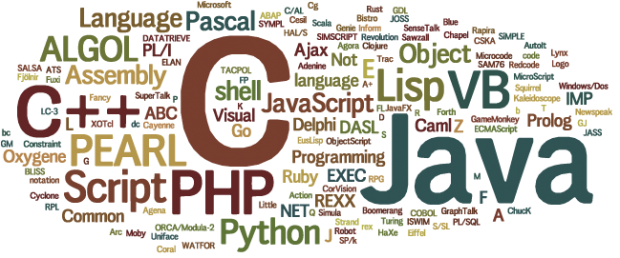
**Permite especificar de *manera precisa* sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar *relativamente* próximo al lenguaje humano o natural. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa.** 

# 

# Lenguaje formal

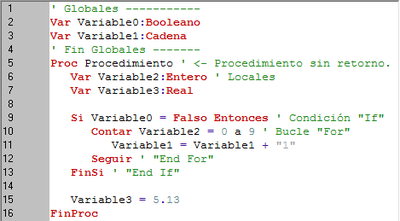
**En matemáticas, lógica y ciencias de la computación, un lenguaje formal es un lenguaje cuyos símbolos primitivos y reglas para unir esos símbolos están formalmente especificados.**[**1**](https://es.wikipedia.org/wiki/Lenguaje_formal#cite_note-1)[**2**](https://es.wikipedia.org/wiki/Lenguaje_formal#cite_note-2)**Al conjunto de los símbolos primitivos se le llama el alfabeto (o vocabulario) del lenguaje, y al conjunto de las reglas se lo llama la gramática formal (o sintaxis). A una cadena de símbolos formada de acuerdo a la gramática se la llama una fórmula bien formada (o palabra) del lenguaje. Estrictamente hablando, un lenguaje formal es idéntico al conjunto de todas sus fórmulas bien formadas. A diferencia de lo que ocurre con el alfabeto (que debe ser un conjunto finito) y con cada fórmula bien formada (que debe tener una longitud también finita), un lenguaje formal puede estar compuesto por un número infinito de fórmulas bien formadas.** 

**Por ejemplo, un alfabeto podría ser el conjunto {*a*,*b*}, y una gramática podría definir a las fórmulas bien formadas como aquellas que tienen el mismo número de símbolos *a* que *b*. Entonces, algunas fórmulas bien formadas del lenguaje serían: *ab*, *ba*, *abab*, *ababba*, etc.; y el lenguaje formal sería el conjunto de todas esas fórmulas bien formadas.**

**Para algunos lenguajes formales existe una semántica formal que puede interpretar y dar significado a las fórmulas bien formadas del lenguaje. Sin embargo, una semántica formal no es condición necesaria para definir un lenguaje formal, y eso es una diferencia esencial con los lenguajes naturales.** 

**En algunos lenguajes formales, la *palabra vacía* (esto es, la cadena de símbolos de longitud cero) está permitida, notándose frecuentemente mediante {\displaystyle \epsilon \,}**

## Ejemplo de lenguajes formales

* **Un conjunto de todas las palabras sobre {*a*,*b*}.**
* **La Numeración de Gödel {*an* : *a* es un número primo y *n* un número de Gödel}.**
* **El conjunto de todos los programas sintácticamente válidos en un determinado lenguaje de programación.**
* **El conjunto de todas las fórmulas bien formadas en la lógica de primer orden.**
* 

## Especificación de lenguajes formales

**Los lenguajes formales se pueden especificar de una amplia variedad de formas, como por ejemplo:**

**(Si el lenguaje es regular)**

* **Cadenas producidas por una gramática formal (véase la jerarquía de Chomsky).**
* **Cadenas descriptas por una expresión regular.**
* **Cadenas aceptadas por un autómata, tal como una máquina de Turing.**

**Las cadenas están formadas por un conjunto de símbolos que pertenecen a un mismo lenguaje, existen dos formas de componer una sentencia o función con los símbolos:**

* **Sintaxis**

### Sintaxis

**A la forma visible de un lenguaje de programación se le conoce como sintaxis. La mayoría de los lenguajes de programación son puramente textuales, es decir, utilizan secuencias de texto que incluyen palabras, números y puntuación, de manera similar a los lenguajes naturales escritos. Por otra parte, hay algunos lenguajes de programación que son más gráficos en su naturaleza, utilizando relaciones visuales entre símbolos para especificar un programa.**

**La sintaxis de un lenguaje de programación describe las combinaciones posibles de los símbolos que forman un programa sintácticamente correcto. El significado que se le da a una combinación de símbolos es manejado por su semántica (ya sea formal o como parte del código duro de la referencia de implementación). Dado que la mayoría de los lenguajes son textuales, este artículo trata de la sintaxis textual.**

**La sintaxis de los lenguajes de programación es definida generalmente utilizando una combinación de expresiones regulares (para la estructura léxica) y la Notación de Backus-Naur (para la estructura gramática). Este es un ejemplo de una gramática simple, tomada de Lisp:**

**expresión ::= átomo | lista**

**átomo ::= número | símbolo**

**número ::= [+-]? ['0'-'9']+**

**símbolo ::= ['A'-'Z'] ['a'-'z'].\***

**lista ::= '(' expresión\* ')'**

**Con esta gramática se especifica lo siguiente:**

* **una *expresión* puede ser un *átomo* o una *lista*;**
* **un *átomo* puede ser un *número* o un *símbolo*;**
* **un *número* es una secuencia continua de uno o más dígitos decimales, precedido opcionalmente por un signo más o un signo menos;**
* **un *símbolo* es una letra seguida de cero o más caracteres (excluyendo espacios); y**
* **una *lista* es un par de paréntesis que abren y cierran, con cero o más expresiones en medio.**

**Algunos ejemplos de secuencias bien formadas de acuerdo a esta gramática:**

**'12345', '()', '(a b c232 (1))'**

**No todos los programas sintácticamente correctos son semánticamente correctos. Muchos programas sintácticamente correctos tienen inconsistencias con las reglas del lenguaje; y pueden (dependiendo de la especificación del lenguaje y la solidez de la implementación) resultar en un error de traducción o ejecución. En algunos casos, tales programas pueden exhibir un comportamiento indefinido. Además, incluso cuando un programa está bien definido dentro de un lenguaje, todavía puede tener un significado que no es el que la persona que lo escribió estaba tratando de construir.**

**Usando el lenguaje natural, por ejemplo, puede no ser posible asignarle significado a una oración gramaticalmente válida o la oración puede ser falsa:**

* **"Las ideas verdes y descoloridas duermen furiosamente" es una oración bien formada gramaticalmente pero no tiene significado comúnmente aceptado.**

**"Juan es un soltero casado" también está bien formada gramaticalmente pero expresa un significado que no puede ser verdadero.**

**El siguiente fragmento en el lenguaje C es sintácticamente correcto, pero ejecuta una operación que no está definida semánticamente (dado que p es un apuntador nulo, las operaciones p->real y p->im no tienen ningún significado):**

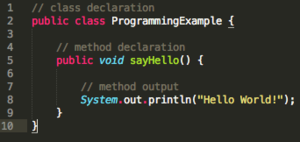
**complex \*p = NULL;**

**complex abs\_p = sqrt (p->real \* p->real + p->im \* p->im);**

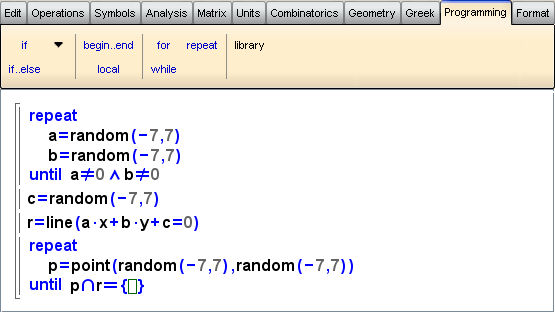
**Si la declaración de tipo de la primera línea fuera omitida, el programa dispararía un error de compilación, pues la variable "p" no estaría definida. Pero el programa sería sintácticamente correcto todavía, dado que las declaraciones de tipo proveen información semántica solamente.**

**La gramática necesaria para especificar un lenguaje de programación puede ser clasificada por su posición en la Jerarquía de Chomsky. La sintaxis de la mayoría de los lenguajes de programación puede ser especificada utilizando una gramática Tipo-2, es decir, son gramáticas libres de contexto. Algunos lenguajes, incluyendo a Perl y a Lisp, contienen construcciones que permiten la ejecución durante la fase de análisis. Los lenguajes que permiten construcciones que permiten al programador alterar el comportamiento de un analizador hacen del análisis de la sintaxis un problema sin decisión única, y generalmente oscurecen la separación entre análisis y ejecución. En contraste con el sistema de macros de Lisp y los bloques BEGIN de Perl, que pueden tener cálculos generales, las macros de C son meros reemplazos de cadenas, y no requieren ejecución de código.**

* **Semántica**



# Conjunto de instrucciones

* **Juan Perez architecture*, «arquitectura del conjunto de instrucciones») es una especificación que detalla las instrucciones que una unidad central de procesamiento puede entender y ejecutar, o el conjunto de todos los comandos implementados por un diseño particular de una CPU. El término describe los aspectos del procesador generalmente visibles para un programador, incluyendo los tipos de datos nativos, las instrucciones, los registros, la arquitectura de memoria y las interrupciones, entre otros aspectos.***
* **Existen principalmente tres tipos: CISC (*Complex Instruction Set Computer*), RISC (*Reduced Instruction Set Computer*) y SISC (*Simple Instruction Set Computing*).**
* 
* **La arquitectura del conjunto de instrucciones (ISA) se emplea a veces para distinguir este conjunto de características de la microarquitectura, que son los elementos y técnicas que se emplean para implementar el conjunto de instrucciones. Entre estos elementos se encuentran las microinstrucciones y los sistemas de caché.**
* **Procesadores con diferentes diseños internos pueden compartir un conjunto de instrucciones; por ejemplo, el Intel Pentium y AMD Athlon implementan versiones casi idénticas del conjunto de instrucciones x86, aunque tienen diseños diferentes.**

## Operaciones

**Se pueden utilizar varias operaciones para producir nuevos lenguajes a partir de otros dados. Supóngase que *L*1 y *L*2 son lenguajes sobre un alfabeto común. Entonces:**

* **La *concatenación* *L*1*L*2 consiste de todas aquellas palabras de la forma *vw* donde *v* es una palabra de *L*1 y *w* es una palabra de *L*2**
* **La *intersección* *L*1&*L*2 consiste en todas aquellas palabras que están contenidas tanto en *L*1 como en *L*2**
* **La *unión* *L*1|*L*2 consiste en todas aquellas palabras que están contenidas ya sea en *L*1 o en *L*2**
* **El *complemento* ~*L*1 consiste en todas aquellas palabras producibles sobre el alfabeto de *L*1 que no están ya contenidas en *L*1**
* **El *cociente* *L*1/*L*2 consiste de todas aquellas palabras *v* para las cuales existe una palabra *w* en *L*2 tales que *vw* se encuentra en *L*1**
* **La *estrella* *L*1\*(Llamada también clausura de Kleene del lenguaje *L*1) consiste de todas aquellas palabras que pueden ser escritas de la forma *W*1*W*2...*W*n donde todo *W*i se encuentra en *L*1 y n ≥ 0. (Nótese que esta definición incluye a λ en cualquier *L*\*)**
* **La Clausura positiva *L*1+ consiste de todas aquellas palabras que pueden ser escritas de la forma *W*1*W*2...*W*n donde todo *W*i se encuentra en *L*1 y n > 0. (Difiere de la clausura de Kleene ya que contiene a λ si y sólo si *L*1 contiene a λ)**
* **La *intercalación* *L*1\**L*2 consiste de todas aquellas palabras que pueden ser escritas de la forma *v*1*w*1*v*2*w*2...*v*n*w*n; son palabras tales que la concatenación *v*1...*v*n está en *L*1, y la concatenación *w*1...*w*n está en *L*2**

**Sean los {\displaystyle L\_{i}} lenguajes, de forma tal que para cada {\displaystyle i} entonces {\displaystyle L\_{i}} esté formado por todas las palabras que pueden surgir de concatenar {\displaystyle i} palabras del lenguaje {\displaystyle L}. Por ejemplo, si {\displaystyle L=\left\{v,w\right\}}, entonces {\displaystyle L\_{3}=\left\{v.v.v,v.v.w,v.w.v,v.w.w,w.v.v,w.v.w,w.w.v,w.w.w\right\}} En base al conceptor anterior pueden definirise la clausuras mencionadas anteriormente:**

* **La clausura de Kleene se denota formalmente: {\displaystyle L^{\*}=\bigcup \_{i\in \mathbb {N} \_{0}}L\_{i}=\left\{\lambda \right\}\cup L\_{1}\cup L\_{2}\cup L\_{3}\cup \ldots \cup L\_{i}}**
* **La clausura positiva se denota formalmente: {\displaystyle L^{+}=\bigcup \_{i\in \mathbb {N} }L\_{i}=L\_{1}\cup L\_{2}\cup L\_{3}\cup \ldots \cup L\_{i}}**

**Por lo tanto se deduce que {\displaystyle L^{\*}=\left\{\lambda \right\}\cup L^{+}}**

# Depuración de programas

**Depuración de programas es el proceso de identificar y corregir errores de programación. En inglés se le conoce como *debugging*, es que se asemeja a la eliminación de *bichos* (*bugs*), manera en que se conoce informalmente a los errores de programación. Se dice que el término *bug* proviene de la época de los ordenadores de válvula termoiónica, en los cuales los problemas se generaban por los insectos que eran atraídos por las luces y estropeaban el equipo. Si bien existen técnicas para la revisión sistemática del código fuente y se cuenta con medios computacionales para la detección de errores (depuradores) y facilidades integradas en los sistemas *lower CASE* y en los ambientes de desarrollo integrado, sigue siendo en buena medida una actividad manual, que desafía la paciencia, la imaginación y la intuición del programador. Muchas veces se requiere incluir en el código fuente instrucciones auxiliares que permitan el seguimiento de la ejecución del programa, presentando los valores de variables y direcciones de memoria y ralentizando la salida de datos (*modo de depuración*). Dentro de un proceso formal de aseguramiento de la calidad, puede ser asimilado al concepto de *prueba unitaria*.**

## Resultado de imagen para depuración de programas

## Origen

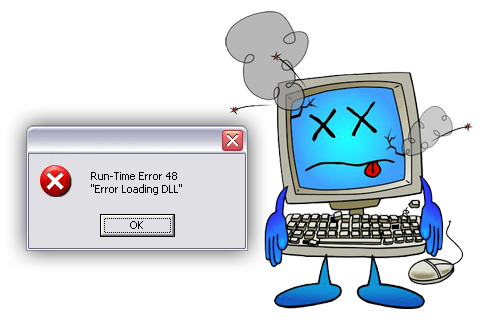
**Existe una controversia acerca del origen del término depuración o "debugging" en inglés. Los términos "bug" y "debugging" son atribuidos popularmente a la almirante Grace Murray Hopper por los años 1940. Mientras trabajaba con un [Mark II] en la Universidad de Harvard, ella encontró una polilla atrapada en un relé impidiendo las operaciones de dicha computadora, por lo cual ella comentó que cuando se sacó aquella polilla le habían hecho "debugging" al sistema. Sin embargo el término "bug" cómo significado de error técnico data cerca de 1878, y el término "debugging" o depuración ha sido usado en aeronáutica antes de entrar al mundo de las computadoras.**

## Aplicación

**Como el software y los sistemas electrónicos se vuelven generalmente más complejos, se han desarrollado varias técnicas comunes de depuración para detectar anomalías, corregir funcionalidades y optimizar código fuente. Existen algunos aficionados que consideran la depuración como una forma de arte.**

# Error de software

**Un error de *software*, error o simplemente fallo (también conocido por el inglés *bug*, «parásito, como las garrapatas y las pulgas»), problema en un programa de computador o sistema de *software* que desencadena un resultado indeseado. Los programas que ayudan a la detección y eliminación de errores de programación de *software* son denominados depuradores (*debuggers*).**

**Entre las numerosas incidencias notables causadas por este tipo de error se incluyen la destrucción, en 1962, de la sonda espacial Mariner 1, en 1996, del Ariane 5 501y, en 2015, el Airbus A400M.** 

## Orígenes del término

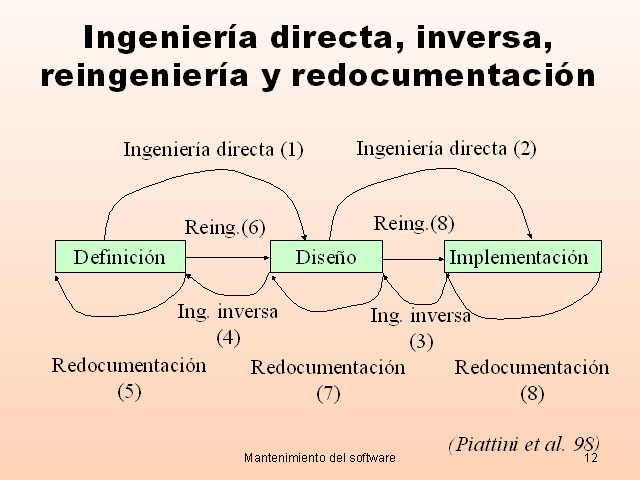
**En 1967, los creadores de Mark III informaron del primer caso de error en un ordenador causado por un *bug*. El Mark III, ordenador sucesor de ASCC Mark II, construido en 1944, sufrió un fallo en un relé electromagnético. Cuando se investigó ese relé, se encontró una polilla (*bug*) que provocó que el relé quedase abierto. Grace Murray Hopper, licenciada en física y destacada matemática que trabajó como programadora en el Mark II, pegó el insecto con cinta adhesiva en la bitácora.**

**Este incidente es erróneamente referido como el origen de la utilización del término inglés *bug* («bicho») para indicar un problema en un aparato o sistema. En realidad, el término *bug* ya formaba parte del idioma inglés, al menos desde que Thomas Alva Edison lo utilizó en 1889 refiriéndose a interferencias y mal funcionamiento. Es posible que Hopper lo haya asociado por primera vez a la informática, en este caso, relacionado a un insecto real. Por otra parte, aunque durante los años 50 del siglo XX, Hopper también empleó en inglés el término *debug* al hablar de la depuración de errores en los códigos de programación, el primer uso registrado del término se encuentra en la *Journal of the Royal Aeronautical Society* de 1945.**



## Defectos de diseño de programas

* **Diseños con colores inapropiados para las personas que padecen daltonismo**
* **Diseños que usan textos con tipografías de difícil lectura por su tamaño o diseño**
* **Diseños que fuerzan el uso del ratón sin dejar alternativas de teclado para personas con disfunciones motrices**
* **Diseños con implicaciones culturales, por ejemplo usando partes del cuerpo que en una determinada cultura sean objeto de vergüenza o burla o símbolos con características de identidad cultural o religiosa**
* **Estimar que el equipo donde se instalará tiene determinadas características (como la resolución de la pantalla, la velocidad del procesador, la cantidad de memoria o conectividad a Internet) propias de un equipo de gama alta, en vez de diseñar el *software* para su ejecución en equipos normales**



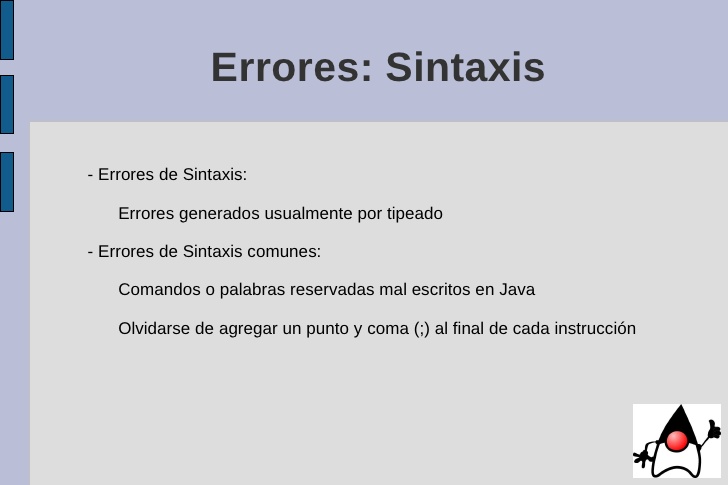
Errores de programación comunes:

**División por cero**

**En matemáticas, la división entre cero es aquella división en la que el divisor es igual a cero. En aritmética y álgebra, es considerada una «indefinición» o «indeterminación» que puede originar paradojas matemáticas.**

**En los números naturales, enteros y reales, la división entre cero no posee un valor definido, debido a que para todo número *n*, el producto *n* · 0 = 0, por lo que el 0 no tiene inverso multiplicativo. En otros cuerpos matemáticos, pueden existir *divisores de cero*, sin embargo, estos aparecen cuando el cero es el dividendo, no el divisor.**

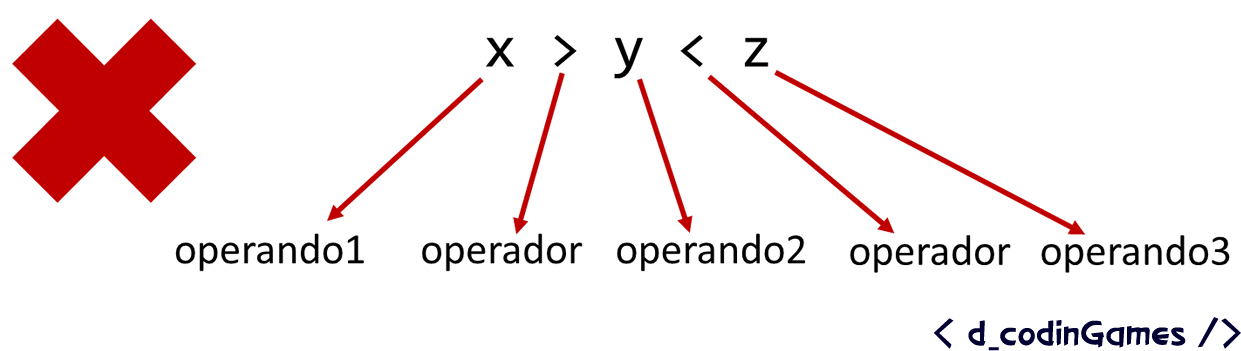
**El problema surgió en los años 650, cuando en India se comenzó a popularizar el uso del cero y los números negativos. El primero en aproximarse al planteamiento de este problema fue el matemático**



**Ciclo infinito**

**Bucle infinito en programación es un error que consiste en realizar un ciclo que se repite de forma indefinida ya que su condición para finalizar nunca se cumple.**[**1**](https://es.wikipedia.org/wiki/Bucle_infinito#cite_note-1)

**Por definición un bucle debe contener condiciones que establezcan cuándo empieza y cuándo acaba, de manera que, mientras las condiciones se cumplan, ejecute una secuencia de código de manera repetitiva. En el caso de ciclo infinito, como la condición de finalización no se alcanza, el bucle sigue ejecutando el segmento de código indefinidamente.**



# Problemas aritméticos

**Un problema aritmético (dentro de la computación), se denomina al desbordamiento de datos debido a la sobre ocupación (overflow) de la memoria disponible como tipos de datos.**

**Supongamos el siguiente trozo de código de un programa en C:**

**#include <stdio.h>**

**int main()**

**{**

**int x, y;**

**x = 32000;**

**y = 7;**

**printf("%d",x\*y);**

**}**

**Exceder el tamaño del array**

**En programación se denomina matriz, vector (de una sola dimensión) o formación (en inglés *array*)**[**1**](https://es.wikipedia.org/wiki/Vector_(inform%C3%A1tica)#cite_note-1)**a una zona de almacenamiento contiguo que contiene una serie de elementos del mismo tipo, los elementos de la matriz.**[**2**](https://es.wikipedia.org/wiki/Vector_(inform%C3%A1tica)#cite_note-2)**Desde el punto de vista lógico una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones).**

**En principio, se puede considerar que todas las matrices son de una dimensión, la dimensión principal, pero los elementos de dicha fila pueden ser a su vez matrices (un proceso que puede ser recursivo), lo que nos permite hablar de la existencia de matrices multidimensionales, aunque las más fáciles de imaginar son los de una, dos y tres dimensiones.**

**Estas estructuras de datos son adecuadas para situaciones en las que el acceso a los datos se realice de forma aleatoria e impredecible. Por el contrario, si los elementos pueden estar ordenados y se va a utilizar acceso secuencial sería más adecuado utilizar una lista, ya que esta estructura puede cambiar de tamaño fácilmente durante la ejecución de un programa.**

**Utilizar una variable no inicializada**

**Acceder a memoria no permitida (Violación de acceso)**

**Pérdida de memoria (*memory leak*)**

**Desbordamiento o subdesbordamiento de la pila (estructura de datos)**

# Desbordamiento de búfer

**En seguridad informática y programación, un desbordamiento de buffer (del inglés *buffer overflow* o *buffer overrun*) es un error de software que se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (buffer): Si dicha cantidad es superior a la capacidad preasignada, los bytes sobrantes se almacenan en zonas de memoria adyacentes, sobrescribiendo su contenido original, que probablemente pertenecían a datos o código almacenados en memoria. Esto constituye un fallo de programación.**

**En las arquitecturas comunes de computadoras no existe separación entre las zonas de memoria dedicadas a datos y las dedicadas a programa, por lo que los bytes que desbordan el buffer podrían grabarse donde antes había instrucciones, lo que implicaría la posibilidad de alterar el flujo del programa, llevándole a realizar operaciones imprevistas por el programador original. Esto es lo que se conoce como una vulnerabilidad.**

**Una vulnerabilidad puede ser aprovechada por un usuario malintencionado para influir en el funcionamiento del sistema. En algunos casos el resultado es la capacidad de conseguir cierto nivel de control saltándose las limitaciones de seguridad habituales. Si el programa con el error en cuestión tiene privilegios especiales constituye en un fallo grave de seguridad.**

**Se denomina shellcode al código ejecutable especialmente preparado que se copia al host objeto del ataque para obtener los privilegios del programa vulnerable.**

**La capacidad de los procesadores modernos para marcar zonas de memoria como protegidas puede usarse para aminorar el problema. Si se produce un intento de escritura en una zona de memoria protegida se genera una excepción del sistema de acceso a memoria, seguido de la terminación del programa. Por desgracia para que esta técnica sea efectiva los programadores han de indicar al sistema operativo las zonas que se necesita proteger, programa a programa y rutina a rutina, lo que supone un problema para todo el código heredado.**

# Bloqueo mutuo

**En sistemas operativos, el bloqueo mutuo (también conocido como interbloqueo, traba mortal, *deadlock*, abrazo mortal) es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos.**[**1**](https://es.wikipedia.org/wiki/Bloqueo_mutuo#cite_note-1)**A diferencia de otros problemas de concurrencia de procesos, no existe una solución general para los interbloqueos.**

**Todos los interbloqueos surgen de necesidades que no pueden ser satisfechas, por parte de dos o más procesos. En la vida real, un ejemplo puede ser el de dos niños que intentan jugar al arco y flecha, uno toma el arco, el otro la flecha. Ninguno puede jugar hasta que alguno libere lo que tomó.**

## Representación de *Bloqueos Mutuos* usando grafos

**El Bloqueo mutuo también puede ser representado usando grafos dirigidos, donde el proceso es representado por un cuadrado y el recurso, por un círculo. Cuando un proceso solicita un recurso, una flecha es dirigida desde el proceso al recurso. En cambio, cuando un recurso está asignado a un proceso, una flecha es dirigida del recurso al proceso.**[**2**](https://es.wikipedia.org/wiki/Bloqueo_mutuo#cite_note-2)

**En la figura del ejemplo, se pueden ver dos procesos diferentes (A y B), cada uno con un recurso diferente asignado (R1 y R2). En este ejemplo clásico de bloqueo mutuo, es fácilmente visible la condición de espera circular en la que los procesos se encuentran, donde cada uno solicita un recurso que está asignado a otro proceso.**

## Condiciones necesarias

**También conocidas como condiciones de Coffman por su primera descripción en 1971 en un artículo escrito por E. G. Coffman.**

**Estas condiciones deben cumplirse simultáneamente y no son totalmente independientes entre ellas.**

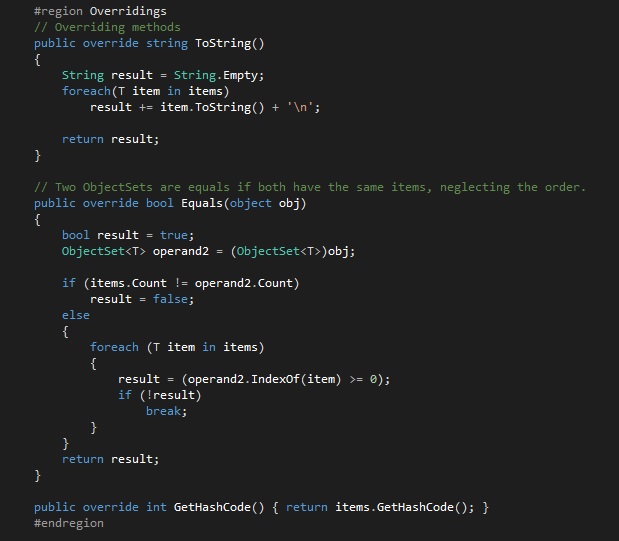
**Sean los procesos *P0, P1, ..., Pn* y los recursos *R0, R1, ..., Rm*:**

* **Condición de exclusión mutua: existencia de al menos de un recurso compartido por los procesos, al cual solo puede acceder uno simultáneamente.**
* **Condición de retención y espera: al menos un proceso *Pi* ha adquirido un recurso *Ri*, y lo retiene mientras espera al menos un recurso *Rj* que ya ha sido asignado a otro proceso.**
* **Condición de no expropiación: los recursos no pueden ser expropiados por los procesos, es decir, los recursos solo podrán ser liberados voluntariamente por sus propietarios.**
* **Condición de espera circular: dado el conjunto de procesos *P0...Pm*(subconjunto del total de procesos original),*P0* está esperando un recurso adquirido por *P1*, que está esperando un recurso adquirido por *P2*,... ,que está esperando un recurso adquirido por *Pm*, que está esperando un recurso adquirido por *P0*. Esta condición implica la condición de retención y espera.**

## Evitando bloqueos mutuos

**Los bloqueos mutuos pueden ser evitados si se sabe cierta información sobre los procesos antes de la asignación de recursos. Para cada petición de recursos, el sistema controla si satisfaciendo el pedido entra en un estado inseguro, donde puede producirse un bloqueo mutuo. De esta forma, el sistema satisface los pedidos de recursos solamente si se asegura que quedará en un estado seguro. Para que el sistema sea capaz de decidir si el siguiente estado será seguro o inseguro, debe saber por adelantado y en cualquier momento el número y tipo de todos los recursos en existencia, disponibles y requeridos. Existen varios algoritmos para evitar bloqueos mutuos:**

* **Algoritmo del banquero, introducido por Dijkstra.**
* **Algoritmo de grafo de asignación de recursos.**
* **Algoritmo de Seguridad.**
* **Algoritmo de solicitud de recursos.**



## El lenguaje máquina

## El lenguaje máquina

**El lenguaje máquina está construido a partir de los estados discretos o instrucciones. En la arquitectura de procesamiento, una instrucción dada puede especificar:**

* **Registros particulares para operaciones aritméticas, de direccionamiento o de control.**
* **Ubicaciones particulares de la memoria.**
* **Modos de direccionamiento para interpretar a los operandos.**

**Las operaciones más complejas se construyen a partir de estas, que (en una máquina Von Neumann) se ejecutan secuencialmente, o según el control de flujo.**

**Muchas o pocas de las operaciones disponibles incluidas en la mayoría de conjuntos son:**

* **Desplazamiento.**
  + **Establecer un registro a un valor constante.**
  + **Mover datos desde una posición de memoria a un registro y viceversa. Esto se realiza para obtener datos para operaciones matemáticas y su almacenamiento.**
  + **Leer y escribir datos desde dispositivos de hardware.**
* **Operaciones matemáticas.**
  + **Sumar, restar, multiplicar o dividir dos registros y colocar el resultado en alguno de ellos.**
  + **Realizar operaciones bit a bit, teniendo el AND y el OR de cada bit en un par de registros, o el NOT de cada bit en un registro.**
  + **Comparar dos valores que se encuentren en registros(por ejemplo, si son iguales o si uno es mayor que otro).**
* **Afectan al flujo de programa.**
  + **Saltar a otra posición del programa y ejecutar instrucciones allí.**
  + **Saltar a otra posición si se cumple cierta condición.**
  + **Saltar a otra posición, pero salvando la posición actual para poder volver (realizar una llamada, por ejemplo *call printf*).**

**Algunas computadoras incluyen instrucciones "complejas". Dichas instrucciones pueden tomar muchas instrucciones en otros equipos. Estas se caracterizan por instrucciones que necesitan varios pasos, como el control de múltiples unidades funcionales.**

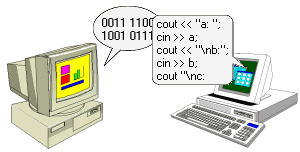


**Algunos ejemplos son:**

* **Ahorro de registros en la pila.**
* **Mover grandes bloques de memoria.**
* **Complejas operaciones and/or con aritmética de coma flotante, tales como el seno o la raíz cuadrada.**
* **Instrucciones que combinan ALU con un operando de la memoria más que de registros.**

**Un tipo complejo de la instrucción que ha llegado a ser particularmente popular recientemente es SIMD (*Single Instruction, Multiple Data*), una operación que realice la misma operación aritmética en pedazos múltiples de datos al mismo tiempo. SIMD tienen la capacidad de manipular vectores y matrices grandes en tiempo mínimo. Las instrucciones de SIMD permiten la paralelización fácil de los algoritmos implicados comúnmente en sonido, imagen, y el proceso video. Varias implementaciones de SIMD se han traído al mercado bajo nombres comerciales tales como MMX, 3DNow! y AltiVec.**

**El diseño de sistemas de instrucción es una edición compleja. Había dos etapas en la historia para el microprocesador. El primer era el CISC (*Complex Instruction Set Computer*) que tenía muchas instrucciones diferentes. En los años 1970, IBM hicieron la investigación y encontraron que muchas instrucciones en el sistema podrían ser eliminadas. El resultado era el RISC (*Reduced Instruction Set Computer*), una arquitectura que utiliza un sistema más pequeño de instrucciones. Un conjunto de instrucción más simple puede ofrecer el potencial para velocidades más altas, tamaño reducido del procesador, y consumo de energía reducido. Sin embargo, un conjunto más complejo puede optimizar operaciones comunes, mejorar memoria/eficiencia de caché, o simplificar la programación.**



### Semántica estática

**La semántica estática define las restricciones sobre la estructura de los textos válidos que resulta imposible o muy difícil expresar mediante formalismos sintácticos estándar. Para los lenguajes compilados, la semántica estática básicamente incluye las reglas semánticas que se pueden verificar en el momento de compilar. Por ejemplo el chequeo de que cada identificador sea declarado antes de ser usado (en lenguajes que requieren tales declaraciones) o que las etiquetas en cada brazo de una estructura *case* sean distintas. Muchas restricciones importantes de este tipo, como la validación de que los identificadores sean usados en los contextos apropiados (por ejemplo no sumar un entero al nombre de una función), o que las llamadas a subrutinas tengan el número y tipo de parámetros adecuado, puede ser implementadas definiéndolas como reglas en una lógica conocida como sistema de tipos. Otras formas de análisis estáticos, como los análisis de flujo de datos, también pueden ser parte de la semántica estática. Otros lenguajes de programación como java y c# tienen un análisis definido de asignaciones, una forma de análisis de flujo de datos, como parte de su semántica estática.**

### Sistema de tipos

**Un sistema de tipos define la manera en la cual un lenguaje de programación clasifica los valores y expresiones en *tipos*, cómo pueden ser manipulados dichos tipos y cómo interactúan. El objetivo de un sistema de tipos es verificar y normalmente poner en vigor un cierto nivel de exactitud en programas escritos en el lenguaje en cuestión, detectando ciertas operaciones inválidas. Cualquier sistema de tipos decidible tiene sus ventajas y desventajas: mientras por un lado rechaza muchos programas incorrectos, también prohíbe algunos programas correctos aunque poco comunes. Para poder minimizar esta desventaja, algunos lenguajes incluyen *lagunas de tipos*, conversiones explícitas no verificadas que pueden ser usadas por el programador para permitir explícitamente una operación normalmente no permitida entre diferentes tipos. En la mayoría de los lenguajes con tipos, el sistema de tipos es usado solamente para verificar los tipos de los programas, pero varios lenguajes, generalmente funcionales, llevan a cabo lo que se conoce como inferencia de tipos, que le quita al programador la tarea de especificar los tipos. Al diseño y estudio formal de los sistemas de tipos se le conoce como *teoría de tipos*.**

#### Tipos estáticos versus tipos dinámicos

**En lenguajes con *tipos estáticos* se determina el tipo de todas las expresiones antes de la ejecución del programa (típicamente al compilar). Por ejemplo, 1 y (2+2) son expresiones enteras; no pueden ser pasadas a una función que espera una cadena, ni pueden guardarse en una variable que está definida como fecha.**

**Los lenguajes con tipos estáticos pueden manejar tipos *explícitos* o tipos *inferidos*. En el primer caso, el programador debe escribir los tipos en determinadas posiciones textuales. En el segundo caso, el compilador *infiere* los tipos de las expresiones y las declaraciones de acuerdo al contexto. La mayoría de los lenguajes populares con tipos estáticos, tales como C++, c# y Java, manejan tipos explícitos. Inferencia total de los tipos suele asociarse con lenguajes menos populares, tales como Haskell y ML. Sin embargo, muchos lenguajes de tipos explícitos permiten inferencias parciales de tipo; tanto Java y C#, por ejemplo, infieren tipos en un número limitado de casos.**

**Los lenguajes con tipos dinámicos determinan la validez de los tipos involucrados en las operaciones durante la ejecución del programa. En otras palabras, los tipos están asociados con *valores en ejecución* en lugar de *expresiones textuales*. Como en el caso de lenguajes con tipos inferidos, los lenguajes con tipos dinámicos no requieren que el programador escriba los tipos de las expresiones. Entre otras cosas, esto permite que una misma variable se pueda asociar con valores de tipos distintos en diferentes momentos de la ejecución de un programa. Sin embargo, los errores de tipo no pueden ser detectados automáticamente hasta que se ejecuta el código, dificultando la depuración de los programas, no obstante, en lenguajes con tipos dinámicos se suele dejar de lado la depuración en favor de técnicas de desarrollo como por ejemplo BDD y TDD. Ruby, Lisp, JavaScript y Python son lenguajes con tipos dinámicos.**

#### Tipos débiles y tipos fuertes

**Los lenguajes *débilmente tipados* permiten que un valor de un tipo pueda ser tratado como de otro tipo, por ejemplo una cadena puede ser operada como un número. Esto puede ser útil a veces, pero también puede permitir ciertos tipos de fallas que no pueden ser detectadas durante la compilación o a veces ni siquiera durante la ejecución.**

**Los lenguajes *fuertemente tipados* evitan que pase lo anterior. Cualquier intento de llevar a cabo una operación sobre el tipo equivocado dispara un error. A los lenguajes con tipos fuertes se les suele llamar *de tipos seguros*.**

**Lenguajes con tipos débiles como Perl y JavaScript permiten un gran número de conversiones de tipo implícitas. Por ejemplo en JavaScript la expresión 2 \* x convierte implícitamente x a un número, y esta conversión es exitosa inclusive cuando x es null, undefined, un Array o una cadena de letras. Estas conversiones implícitas son útiles con frecuencia, pero también pueden ocultar errores de programación.**

**Las características de *estáticos* y *fuertes* son ahora generalmente consideradas conceptos ortogonales, pero su trato en diferentes textos varia. Algunos utilizan el término *de tipos fuertes* para referirse a *tipos fuertemente estáticos* o, para aumentar la confusión, simplemente como equivalencia de *tipos estáticos*. De tal manera que C ha sido llamado tanto lenguaje de tipos fuertes como lenguaje de tipos estáticos débiles.**



## Implementación

**La implementación de un lenguaje es la que provee una manera de que se ejecute un programa para una determinada combinación de software y hardware. Existen básicamente dos maneras de implementar un lenguaje: compilación e interpretación.**

* **Compilación: es el proceso que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz interpretar. Los programas traductores que pueden realizar esta operación se llaman compiladores. Éstos, como los programas ensambladores avanzados, pueden generar muchas líneas de código de máquina por cada proposición del programa fuente.**
* **Interpretación: es una asignación de significados a las fórmulas bien formadas de un lenguaje formal. Como los lenguajes formales pueden definirse en términos puramente sintácticos, sus fórmulas bien formadas pueden no ser más que cadenas de símbolos sin ningún significado. Una interpretación otorga significado a esas fórmulas.**

**Se puede también utilizar una alternativa para traducir lenguajes de alto nivel. En lugar de traducir el programa fuente y grabar en forma permanente el código objeto que se produce durante la compilación para utilizarlo en una ejecución futura, el programador sólo carga el programa fuente en la computadora junto con los datos que se van a procesar. A continuación, un programa intérprete, almacenado en el sistema operativo del disco, o incluido de manera permanente dentro de la máquina, convierte cada proposición del programa fuente en lenguaje de máquina conforme vaya siendo necesario durante el procesamiento de los datos. El código objeto no se graba para utilizarlo posteriormente.**

**La siguiente vez que se utilice una instrucción, se la deberá interpretar otra vez y traducir a lenguaje máquina. Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo o bucle, cada instrucción del bucle tendrá que volver a ser interpretada en cada ejecución repetida del ciclo, lo cual hace que el programa sea más lento en tiempo de ejecución (porque se va revisando el código en tiempo de ejecución) pero más rápido en tiempo de diseño (porque no se tiene que estar compilando a cada momento el código completo). El intérprete elimina la necesidad de realizar una compilación después de cada modificación del programa cuando se quiere agregar funciones o corregir errores; pero es obvio que un programa objeto compilado con antelación deberá ejecutarse con mucha mayor rapidez que uno que se debe interpretar a cada paso durante una ejecución del código.**

**La mayoría de lenguajes de alto nivel permiten la programación multipropósito, aunque muchos de ellos fueron diseñados para permitir programación dedicada, como lo fue el Pascal con las matemáticas en su comienzo. También se han implementado lenguajes educativos infantiles como Logo mediante una serie de simples instrucciones. En la actualidad son muy populares algunos lenguajes especialmente indicados para aplicaciones web, como Perl, PHP, Ruby, Python o JavaScript.**

## Técnica

**Para escribir programas que proporcionen los mejores resultados, cabe tener en cuenta una serie de detalles.**

* ***Corrección*. Un programa es correcto si hace lo que debe hacer tal y como se estableció en las fases previas a su desarrollo. Para determinar si un programa hace lo que debe, es muy importante especificar claramente qué debe hacer el programa antes de desarrollarlo y, una vez acabado, compararlo con lo que realmente hace.**
* ***Claridad*. Es muy importante que el programa sea lo más claro y legible posible, para facilitar así su desarrollo y posterior mantenimiento. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo en la edición; de esta forma se ve facilitado el trabajo del programador, tanto en la fase de creación como en las fases posteriores de corrección de errores, ampliaciones, modificaciones, etc. Fases que pueden ser realizadas incluso por otro programador, con lo cual la claridad es aún más necesaria para que otros programadores puedan continuar el trabajo fácilmente. Algunos programadores llegan incluso a utilizar Arte ASCII para delimitar secciones de código. Otros, por diversión o para impedir un análisis cómodo a otros programadores, recurren al uso de código ofuscado.**
* ***Eficiencia*. Se trata de que el programa, además de realizar aquello para lo que fue creado (es decir, que sea correcto), lo haga gestionando de la mejor forma posible los recursos que utiliza. Normalmente, al hablar de eficiencia de un programa, se suele hacer referencia al tiempo que tarda en realizar la tarea para la que ha sido creado y a la cantidad de memoria que necesita, pero hay otros recursos que también pueden ser de consideración al obtener la eficiencia de un programa, dependiendo de su naturaleza (espacio en disco que utiliza, tráfico de red que genera, etc.).**
* ***Portabilidad*. Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea hardware o software, diferente a aquella en la que se elaboró. La portabilidad es una característica muy deseable para un programa, ya que permite, por ejemplo, a un programa que se ha desarrollado para sistemas GNU/Linux ejecutarse también en la familia de sistemas operativos Windows. Esto permite que el programa pueda llegar a más usuarios más fácilmente.**

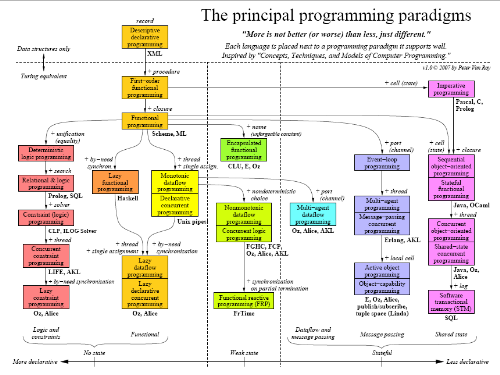
### Paradigmas

**Los programas se pueden clasificar por el paradigma del lenguaje que se use para producirlos. Los principales paradigmas son: imperativos, declarativos y orientación a objetos.**

**Los programas que usan un lenguaje imperativo especifican un algoritmo, usan declaraciones, expresiones y sentencias.**[**3**](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n#cite_note-cpl-ch4-75-3)**Una declaración asocia un nombre de variable con un tipo de dato, por ejemplo: var x: integer;. Una expresión contiene un valor, por ejemplo: 2 + 2 contiene el valor 4. Finalmente, una sentencia debe asignar una expresión a una variable o usar el valor de una variable para alterar el flujo de un programa, por ejemplo: x := 2 + 2; if x == 4 then haz\_algo();. Una crítica común en los lenguajes imperativos es el efecto de las sentencias de asignación sobre una clase de variables llamadas "no locales".**[**4**](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n#cite_note-cpl-ch9-213-4)

**Los programas que usan un lenguaje declarativo especifican las propiedades que la salida debe conocer y no especifican cualquier detalle de implementación. Dos amplias categorías de lenguajes declarativos son los lenguajes funcionales y los lenguajes lógicos. Los lenguajes funcionales no permiten asignaciones de variables no locales, así, se hacen más fácil, por ejemplo, programas como funciones matemáticas. El principio detrás de los lenguajes lógicos es definir el problema que se quiere resolver (el objetivo) y dejar los detalles de la solución al sistema. El objetivo es definido dando una lista de sub-objetivos. Cada sub-objetivo también se define dando una lista de sus sub-objetivos, etc. Si al tratar de buscar una solución, una ruta de sub-objetivos falla, entonces tal sub-objetivo se descarta y sistemáticamente se prueba otra ruta.**

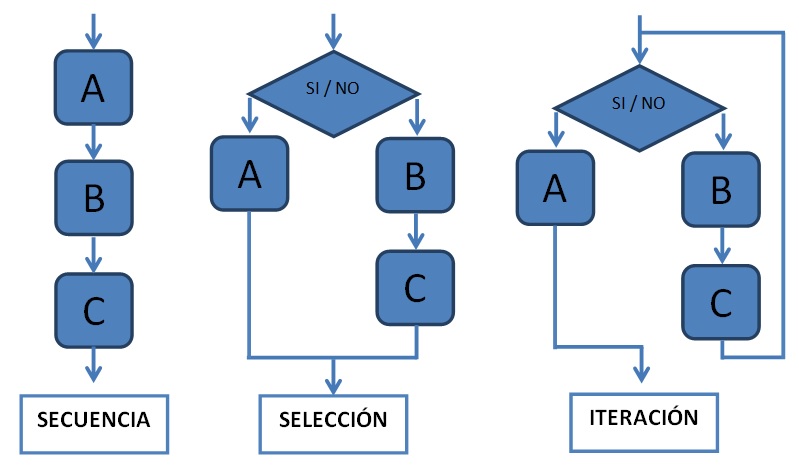
**La forma en la cual se programa puede ser por medio de texto o de forma visual. En la programación visual los elementos son manipulados gráficamente en vez de especificarse por medio de texto.**



# Programación estructurada

**La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora, utilizando únicamente subrutinas y tres estructuras: secuencia, selección (*if* y *switch*) e iteración (bucles *for* y *while*), considerando innecesario y contraproducente el uso de la instrucción de transferencia incondicional (GOTO), que podría conducir a "código espagueti", que es mucho más difícil de seguir y de mantener, y era la causa de muchos errores de programación.**

**Surgió en la década de 1960, particularmente del trabajo de Böhm y Jacopini,**[**1**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada#cite_note-1)**y una famosa carta, «La sentencia goto, considerada perjudicial», de Edsger Dijkstra en 1968**[**2**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada#cite_note-dijkstra1968-2)**— y fue reforzado teóricamente por el teorema del programa estructurado, y prácticamente por la aparición de lenguajes como ALGOL con adecuadas y ricas estructuras de control.**



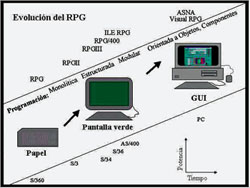
## Orígenes de la programación estructurada

**A finales de los años 1970 surgió una nueva forma de programar que no solamente daba lugar a programas fiables y eficientes, sino que además estaban escritos de manera que facilitaba su mejor comprensión, no sólo proveyendo ventajas durante la fase de desarrollo, sino también posibilitando una más sencilla modificación posterior.**

**El teorema del programa estructurado, propuesto por Böhm-Jacopini, demuestra que todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes:**

* **Secuencia.**
* **Instrucción condicional.**
* **Iteración (bucle de instrucciones) con condición al principio.**

**Solamente con estas tres estructuras se pueden escribir todos los programas y aplicaciones posibles. Si bien los lenguajes de programación tienen un mayor repertorio de estructuras de control, estas pueden ser construidas mediante las tres básicas citadas.**



## Historia

### Fundamentación teórica

**El teorema del programa estructurado proporciona la base teórica de la programación estructurada. Señala que la combinación de las tres estructuras básicas, secuencia, selección e iteración, son suficientes para expresar cualquier función computable. Esta observación no se originó con el movimiento de la programación estructurada. Estas estructuras son suficientes para describir el ciclo de instrucción de una unidad central de procesamiento, así como el funcionamiento de una máquina de Turing. Por lo tanto un procesador siempre está ejecutando un "programa estructurado" en este sentido, incluso si las instrucciones que lee de la memoria no son parte de un programa estructurado. Sin embargo, los autores usualmente acreditan el resultado a un documento escrito en 1966 por Böhm y Jacopini, posiblemente porque Dijkstra había citado este escrito. El teorema del programa estructurado no responde a cómo escribir y analizar un programa estructurado de manera útil. Estos temas fueron abordados durante la década de 1960 y principio de los años 1970, con importantes contribuciones de Dijkstra, Robert W. Floyd, Tony Hoarey y David Gries.**

### Debate

**P. J. Plauger, uno de los primeros en adoptar la programación estructurada, describió su reacción con el teorema del programa estructurado:**

**Nosotros los conversos ondeamos esta interesante pizca de noticias bajo las narices de los recalcitrantes programadores de lenguaje ensamblador que mantuvieron trotando adelante retorcidos bits de lógica y diciendo, 'Te apuesto que no puedes estructurar esto'. Ni la prueba por Böhm y Jacopini, ni nuestros repetidos éxitos en escribir código estructurado, los llevaron un día antes de lo que estaban listos para convencerse.**[**3**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada#cite_note-3)

**Donald Knuth aceptó el principio de que los programas deben escribirse con demostratividad en mente, pero no estaba de acuerdo (y aún está en desacuerdo)con la supresión de la sentencia GOTO. En su escrito de 1974 “Programación estructurada con sentencias Goto”, dio ejemplos donde creía que un salto directo conduce a código más claro y más eficiente sin sacrificar demostratividad. Knuth propuso una restricción estructural más flexible: debe ser posible establecer un diagrama de flujo del programa con todas las bifurcaciones hacia adelante a la izquierda, todas las bifurcaciones hacia atrás a la derecha, y sin bifurcaciones que se crucen entre sí. Muchos de los expertos en teoría de grafos y compiladores han abogado por permitir sólo grafos de flujo reducible.**

**Los teóricos de la programación estructurada ganaron a un aliado importante en la década de 1970 después de que el investigador de IBM Harlan Mills aplicara su interpretación de la teoría de la programación estructurada para el desarrollo de un sistema de indexación para el archivo de investigación del *New York Times*. El proyecto fue un gran éxito de la ingeniería, y los directivos de otras empresas lo citaron en apoyo de la adopción de la programación estructurada, aunque Dijkstra criticó las maneras en que la interpretación de Mills difería de la obra publicada.**

**Tan tarde como 1987 fue todavía posible elevar la cuestión de la programación estructurada en una revista de ciencia de la computación. Frank Rubin lo hizo en ese año, con una carta, “¿'La sentencia GOTO considerada dañina' considerada dañina?”. Numerosas objeciones siguieron, incluyendo una respuesta de Dijkstra, que criticaba duramente a Rubin y las concesiones que otros escritores hicieron cuando le respondieron.**

### Resultado

**A finales del siglo XX casi todos los científicos están convencidos de que es útil aprender y aplicar los conceptos de programación estructurada. Los lenguajes de programación de alto nivel que originalmente carecían de estructuras de programación, como FORTRAN, COBOL y BASIC, ahora las tienen.**

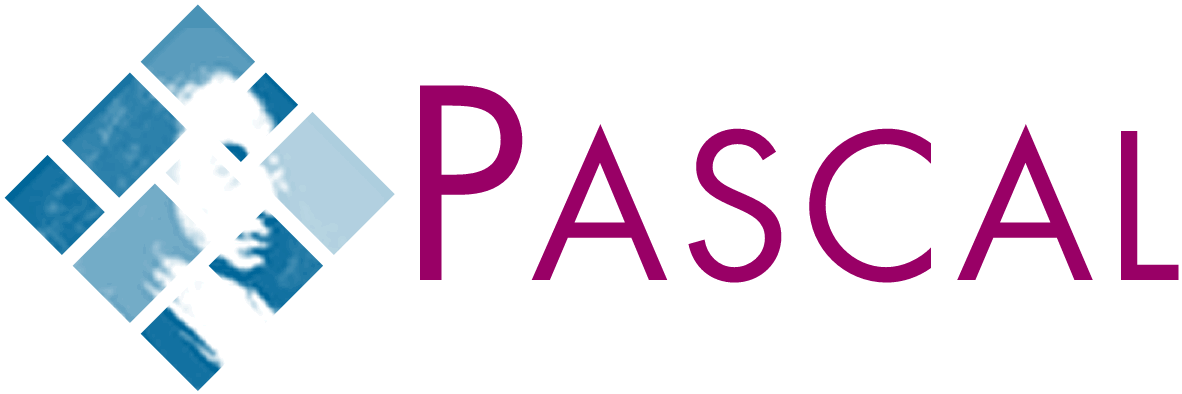
## Ventajas de la programación estructurada

**Ventajas de la programación estructurada comparada con el modelo anterior (hoy llamado despectivamente código espagueti).**

* **Los programas son más fáciles de entender, pueden ser leídos de forma secuencial y no hay necesidad de hacer engorrosos seguimientos en saltos de líneas (GOTO) dentro de los bloques de código para intentar entender la lógica.**
* **La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.**
* **Reducción del esfuerzo en las pruebas y depuración. El seguimiento de los fallos o errores del programa (*debugging*) se facilita debido a su estructura más sencilla y comprensible, por lo que los errores se pueden detectar y corregir más fácilmente.**
* **Reducción de los costos de mantenimiento. Análogamente a la depuración, durante la fase de mantenimiento, modificar o extender los programas resulta más fácil.**
* **Los programas son más sencillos y más rápidos de confeccionar.**
* **Se incrementa el rendimiento de los programadores.**

## Lenguajes de programación estructurada

**Es posible hacer la programación estructurada en cualquier lenguaje de programación, aunque es preferible usar algo como un lenguaje de programación procedimental. Algunos de los lenguajes utilizados inicialmente para programación estructurada incluyen: ALGOL, Pascal, PL/I y Ada —pero la mayoría de los nuevos lenguajes de programación procedimentales desde entonces han incluido características para fomentar la programación estructurada y a veces deliberadamente omiten características,**[**4**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada#cite_note-4)**en un esfuerzo para hacer más difícil la programación no estructurada.**



## Nuevos paradigmas

**Posteriormente a la programación estructurada se han creado nuevos paradigmas tales como la programación modular, la programación orientada a objetos, programación por capas, etc., y el desarrollo de entornos de programación que facilitan la programación de grandes aplicaciones y sistemas.**

# Programación orientada a objetos

**La programación orientada a objetos (POO, u OOP según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.**



**Muchos de los objetos pre-diseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas.**

**Está basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.**



**Su uso se popularizó a principios de la década de 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la orientación a objetos.**

## Introducción

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

* **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 o una constante).**

**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, 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.**

**Los métodos (comportamiento) y atributos (estado) están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta. El programador debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a alguno de ellos. Hacerlo podría producir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen a las primeras por el otro. De esta manera se estaría realizando una "programación estructurada camuflada" en un lenguaje de POO.**

**La programación orientada a objetos difiere de la programación 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 POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.**

## Origen

**Los conceptos de la POO tienen origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard, del Centro de Cómputo Noruego en Oslo. En este centro se trabajaba en simulaciones de naves, que fueron confundidas por la explosión combinatoria de cómo las diversas cualidades de diferentes naves podían afectar unas a las otras. La idea surgió al agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus "propios" datos y comportamientos. Fueron refinados más tarde en Smalltalk, desarrollado en Simula en Xerox PARC (cuya primera versión fue escrita sobre Basic) pero diseñado para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar "sobre la marcha" (en tiempo de ejecución) en lugar de tener un sistema basado en programas estáticos.**

**La POO se fue convirtiendo en el estilo de programación dominante a mediados de los años 1980, en gran parte debido a la influencia de C++, una extensión del lenguaje de programación C. Su dominación fue consolidada gracias al auge de las interfaces gráficas de usuario, para las cuales la POO está particularmente bien adaptada. En este caso, se habla también de programación dirigida por eventos.**

**Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo Ada, BASIC, Lisp más Pascal, entre otros. La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas condujo a menudo a problemas de compatibilidad y en la capacidad de mantenimiento del código. Los lenguajes orientados a objetos "puros", por su parte, carecían de las características de las cuales muchos programadores habían venido a depender. Para saltar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos, pero permitiendo algunas características imperativas de maneras "seguras". El lenguaje de programación Eiffel de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos, pero ahora ha sido esencialmente reemplazado por Java, en gran parte debido a la aparición de Internet y a la implementación de la máquina virtual Java en la mayoría de navegadores web. PHP en su versión 5 se ha modificado; soporta una orientación completa a objetos, cumpliendo todas las características propias de la orientación a objetos.**

## Conceptos fundamentales

**La POO es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:**

**Clase**

**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**

**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. En el caso de los componentes registrados como "protegidos" (*protected*) también se heredan, pero solo para esa clase, no para futuras clases heredadas. Esto es así para mantener hegemónico el ideal de POO.**

**Objeto**

**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).**

**Método**

**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.**

## Características de la POO

**Existe un acuerdo acerca de qué características contempla la "orientación a objetos". Las características siguientes son las más importantes:**

**Abstracción**

**Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar "cómo" se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos, y, cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción. El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.**

**Encapsulamiento**

**Significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión (diseño estructurado) de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.**

**Polimorfismo**

**Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O, dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.**

**Herencia**

**Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento, permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple; siendo de alta complejidad técnica por lo cual suele recurrirse a la herencia virtual para evitar la duplicación de datos.**

**Modularidad**

**Se denomina "modularidad" a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos. Al igual que la encapsulación, los lenguajes soportan la modularidad de diversas formas.**

**Principio de ocultación**

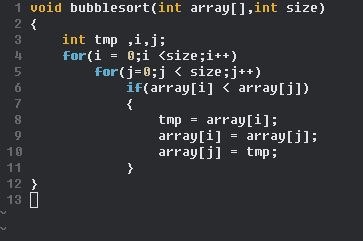
**Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una "interfaz" a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.**

**Recolección de basura**

**La recolección de basura (*garbage collection*) es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando. En la mayoría de los lenguajes híbridos que se extendieron para soportar el Paradigma de Programación Orientada a Objetos como C++ u Object Pascal, esta característica no existe y la memoria debe desasignarse expresamente.**

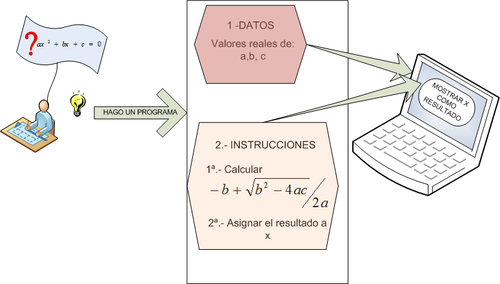
# Programación imperativa

**La programación imperativa es uno de los paradigmas de programación de computadoras más generales, que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador *cómo* realizar una tarea, de la misma manera que el modo imperativo en los lenguajes naturales humanos le dice qué hacer al interlocutor. Esto existe en contraposición al paradigma de programación declarativa, que se basa en *describir* el problema o solución sin hacer referencia a un algoritmo. Las recetas de cocina y las listas de revisión de procesos, a pesar de no ser programas de computadora, son también conceptos familiares similares en estilo a la programación imperativa; donde cada paso es una instrucción.**



## Programación imperativa y hardware

**La implementación de hardware de la mayoría de los computadores es imperativa; prácticamente todo el hardware de los computadores está diseñado para ejecutar código de máquina, que es nativo al computador, escrito en una forma imperativa. Esto se debe a que el hardware de los computadores puede modelarse como una máquina de Turing, o alguna otra forma de máquina abstracta que modifica su memoria o estado interno conforme sigue los pasos de un programa. Desde esta perspectiva de bajo nivel, la semántica del programa en cualquier momento está definida por los contenidos previos de la memoria, los datos de entrada y las instrucciones en el lenguaje de máquina nativo del computador.**



**Este paralelismo entre el paradigma imperativo y los objetos físicos que terminan realizando los cálculos significa que todo programa, así esté escrito en un lenguaje de alto nivel o incluso en uno completamente declarativo, eventualmente debe ser transformado a código máquina imperativo para tener efecto.**



## Lenguajes de programación

**Los primeros lenguajes imperativos fueron los lenguajes de máquina de los computadores originales. En estos lenguajes, las instrucciones fueron muy simples, lo cual hizo la implementación de hardware fácil, pero obstruyendo la creación de programas complejos. Los lenguajes imperativos de alto nivel usan variables y sentencias más complejas, pero aún siguen el mismo paradigma cuando modifican los contenidos de las variables. Fortran, cuyo desarrollo fue iniciado en 1954 por John Backus en IBM, fue el primer gran lenguaje de programación en superar los obstáculos presentados por el código de máquina en la creación de programas complejos.**

**Algunos ejemplos de lenguajes de programación imperativos (o que son usados idiomáticamente en estilo imperativo a pesar de ofrecer otros estilos) son:**

* **BASIC**
* **C**
* **D**
* **Fortran**
* **Pascal**
* **Pauscal en español**
* **Perl**
* **PHP**
* **Lua**
* **Java**
* **Python**
* **Go**
* **simple basic**
* **Javascript**

LENGUAJES DE PROGRAMACION:

**Perl:** es un lenguaje de programación diseñado por Larry Wall en mil novecientos ocenta y siete. Perl toma características del lenguaje C, del lenguaje interpretado bourne shell (SH), AWK, sed, Lisp y, en un grado inferior, de muchos otros lenguajes de programación.

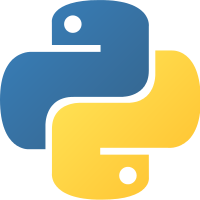
Estructuralmente, Perl está basado en un estilo de bloques como los del C o AWK, y fue ampliamente adoptado por su destreza en el procesado de texto y no tener ninguna de las limitaciones de los otros lenguajes de script.



**Python** es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programacin funcional. Es un lenguaje interpretado, usa tipado dinamico y es multiplataforma.

Es administrado por la Python software foundatin. Posee una licencia de código abierto, denominada [Python Software Foundation License](https://es.wikipedia.org/wiki/Python_Software_Foundation_License), que es compatible con la [Licencia pública general de GNU](https://es.wikipedia.org/wiki/GNU_General_Public_License) a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.



**Visual Basic** (VB) es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) dirigido por eventos, desarrollado por [Alan Cooper](https://es.wikipedia.org/wiki/Alan_Cooper) para [Microsoft](https://es.wikipedia.org/wiki/Microsoft). Este lenguaje de programación es un dialecto de [BASIC](https://es.wikipedia.org/wiki/BASIC), con importantes agregados. Su primera versión fue presentada en mil novecientos noventa y uno, con la intención de simplificar la programación utilizando un ambiente de desarrollo que facilitó en cierta medida la programación misma.

La última versión fue la 6, liberada en 1998, para la que Microsoft extendió el soporte hasta marzo de 2008.

En 2001 Microsoft propuso abandonar el desarrollo basado en la API win32 y pasar a un framework o marco común de librerías, independiente de la versión del sistema operativo, NET framework, a través de Visual basic.NET (y otros lenguajes como C sharp (C#) de fácil transición de código entre ellos); fue el sucesor de Visual Basic 6.

Aunque Visual Basic es de propósito general, también provee facilidades para el desarrollo de aplicaciones de [bases de datos](https://es.wikipedia.org/wiki/Bases_de_datos) usando [Data Access Objects](https://es.wikipedia.org/wiki/Data_Access_Objects), [Remote Data Objects](https://es.wikipedia.org/wiki/Remote_Data_Objects) o [ActiveX Data Objects](https://es.wikipedia.org/wiki/ActiveX_Data_Objects).

Visual Basic contiene un entorno de desarrollo integrado o [IDE](https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado) que integra [editor de textos](https://es.wikipedia.org/wiki/Editor_de_textos) para edición del código fuente, un [depurador](https://es.wikipedia.org/wiki/Depurador), un [compilador](https://es.wikipedia.org/wiki/Compilador) (y [enlazador](https://es.wikipedia.org/wiki/Enlazador)) y un editor de interfaces gráficas o [GUI](https://es.wikipedia.org/wiki/GUI).

Visual Basic dio un paso más en innovación y ahora es posible programar aplicaciones Nativas para Android o IPhone utilizando un software de la compañía Anywhere Software que transforma código Visual Basic (creado por dicha compañía) en APPs 100 % nativas en Java para ambos sistemas operativos de dispositivos móviles.[1](https://es.wikipedia.org/wiki/Visual_Basic#cite_note-1)

**Java** es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) de [propósito general](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_de_prop%C3%B3sito_general), [concurrente](https://es.wikipedia.org/wiki/Computaci%C3%B3n_concurrente), [orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos) que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los [desarrolladores](https://es.wikipedia.org/wiki/Desarrollador_de_software) de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como *WORA*, o "*write once, run anywhere*"), lo que quiere decir que el [código](https://es.wikipedia.org/wiki/C%C3%B3digo_fuente) que es ejecutado en una plataforma no tiene que ser [recompilado](https://es.wikipedia.org/wiki/Compilaci%C3%B3n_en_tiempo_de_ejecuci%C3%B3n) para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de [cliente-servidor](https://es.wikipedia.org/wiki/Cliente-servidor) de web, con unos 10 millones de usuarios reportados.[1](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)#cite_note-1) [2](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)#cite_note-2)

El lenguaje de programación Java fue originalmente desarrollado por [James Gosling](https://es.wikipedia.org/wiki/James_Gosling) de [Sun Microsystems](https://es.wikipedia.org/wiki/Sun_Microsystems) (la cual fue adquirida por la compañía [Oracle](https://es.wikipedia.org/wiki/Oracle_Corporation)) y publicado en 1995 como un componente fundamental de la [plataforma Java](https://es.wikipedia.org/wiki/Java_(plataforma_de_software)) de Sun Microsystems. Su sintaxis deriva en gran medida de [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) y [C++](https://es.wikipedia.org/wiki/C%2B%2B), pero tiene menos utilidades de [bajo nivel](https://es.wikipedia.org/wiki/Lenguaje_de_bajo_nivel) que cualquiera de ellos. Las aplicaciones de Java son generalmente [compiladas](https://es.wikipedia.org/wiki/Compilador) a [bytecode](https://es.wikipedia.org/wiki/Bytecode_Java) (clase Java) que puede ejecutarse en cualquier [máquina virtual Java](https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java) (JVM) sin importar la [arquitectura de la computadora](https://es.wikipedia.org/wiki/Arquitectura_de_computadoras) subyacente.

La compañía Sun desarrolló la [implementación de referencia](https://es.wikipedia.org/wiki/Implementaci%C3%B3n_de_referencia) original para los [compiladores](https://es.wikipedia.org/wiki/Compilador) de Java, máquinas virtuales, y [librerías](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)) de clases en 1991 y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento con las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la [Licencia Pública General de GNU](https://es.wikipedia.org/wiki/GNU_General_Public_License). Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, tales como el [Compilador de Java de GNU](https://es.wikipedia.org/wiki/GNU_Compiler_for_Java) y el [GNU Classpath](https://es.wikipedia.org/wiki/GNU_Classpath).

**C++** es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) diseñado a mediados de los años 1980 por [Bjarne Stroustrup](https://es.wikipedia.org/wiki/Bjarne_Stroustrup). La intención de su creación fue el extender al lenguaje de programación [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) mecanismos que permiten la manipulación de [objetos](https://es.wikipedia.org/wiki/Objeto_(programaci%C3%B3n)). En ese sentido, desde el punto de vista de los [lenguajes orientados a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de [programación genérica](https://es.wikipedia.org/wiki/Programaci%C3%B3n_gen%C3%A9rica), que se sumaron a los paradigmas de [programación estructurada](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) y [programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos). Por esto se suele decir que el C++ es un [lenguaje de programación multiparadigma](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_multiparadigma).

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT.

Una particularidad del C++ es la posibilidad de redefinir los [operadores](https://es.wikipedia.org/wiki/Operadores_de_C_y_C%2B%2B), y de poder crear nuevos [tipos](https://es.wikipedia.org/wiki/Tipo_de_datos) que se comporten como tipos fundamentales.

El nombre "C++" fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.



**Pascal** es un lenguaje creado por el profesor suizo [Niklaus Wirth](https://es.wikipedia.org/wiki/Niklaus_Wirth) entre los años 1968 y 1969 y publicado en 1970. Su objetivo era crear un lenguaje que facilitara el aprendizaje de programación a sus alumnos, utilizando la [programación estructurada](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) y estructuración de datos. Sin embargo con el tiempo su utilización excedió el ámbito académico para convertirse en una herramienta para la creación de aplicaciones de todo tipo.

Pascal se caracteriza por ser un lenguaje de programación estructurado [fuertemente tipado](https://es.wikipedia.org/wiki/Tipado_fuerte). Esto implica que:

1. El código está dividido en porciones fácilmente legibles llamadas *funciones o procedimientos*. De esta forma *Pascal* facilita la utilización de la *programación estructurada* en oposición al antiguo estilo de *programación monolítica*.

LENGUAJE DE PRACTICA:

## PSeInt es un software libre educativo multiplataforma dirigido a personas que se inician en la programación.

## Descripción

PSeInt es la abreviatura de **PSe**udo **Int**érprete, una herramienta educativa utilizada principalmente por estudiantes para aprender los fundamentos de la programación y el desarrollo de la lógica. Es un software muy popular de su tipo y es ampliamente utilizado en universidades de [Latinoamérica](https://es.wikipedia.org/wiki/Am%C3%A9rica_Latina) y [España](https://es.wikipedia.org/wiki/Espa%C3%B1a).

Utiliza [pseudocódigo](https://es.wikipedia.org/wiki/Pseudoc%C3%B3digo) para la solución de [algoritmos](https://es.wikipedia.org/wiki/Algoritmo).

## Propósito de PSeInt[

PSeInt está pensado para asistir a los estudiantes que se inician en la construcción de programas o algoritmos computacionales. El pseudocódigo se suele utilizar como primer contacto para introducir conceptos básicos como el uso de estructuras de control, expresiones, variables, etc, sin tener que lidiar con las particularidades de la sintaxis de un lenguaje real. Este software pretende facilitarle al principiante la tarea de escribir algoritmos en este pseudolenguaje presentando un conjunto de ayudas y asistencias, y brindarle además algunas herramientas adicionales que le ayuden a encontrar errores y comprender la lógica de los algoritmos.

## Características

* Lenguaje Autocompletado
* Ayudas Emergentes
* Plantillas de Comandos
* Soporta procedimientos y funciones
* Indentado Inteligente
* Exportación a otros lenguajes ([C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)), [C++](https://es.wikipedia.org/wiki/C%2B%2B), [C#](https://es.wikipedia.org/wiki/C_Sharp), [Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)), [PHP](https://es.wikipedia.org/wiki/PHP), [JavaScript](https://es.wikipedia.org/wiki/JavaScript), [Visual Basic .NET](https://es.wikipedia.org/wiki/Visual_Basic_.NET), [Python](https://es.wikipedia.org/wiki/Python), [Matlab](https://es.wikipedia.org/wiki/Matlab))
* Graficado, creación y edición de [diagramas de flujo](https://es.wikipedia.org/wiki/Diagrama_de_flujo)
* Editor con coloreado de sintaxis
* [Foro oficial de PSeInt](http://sourceforge.net/p/pseint/discussion/?source=navbar)
* Software multiplataforma[2](https://es.wikipedia.org/wiki/PSeInt#cite_note-2) sobre [Microsoft Windows](https://es.wikipedia.org/wiki/Microsoft_Windows), [GNU/Linux](https://es.wikipedia.org/wiki/Linux) y [Mac OS X](https://es.wikipedia.org/wiki/Mac_OS_X), en diciembre de 2016 empezó un desarrollo independiente para [Android](https://es.wikipedia.org/wiki/Android)[3](https://es.wikipedia.org/wiki/PSeInt#cite_note-3)

LENGUAJES DE PROGRAMACION PAGINAS WEB:

**HTML**, sigla en inglés de ***HyperText Markup Language*** (lenguaje de marcas de hipertexto), hace referencia al [lenguaje de marcado](https://es.wikipedia.org/wiki/Lenguaje_de_marcado) para la elaboración de [páginas web](https://es.wikipedia.org/wiki/P%C3%A1gina_web). Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del [*World Wide Web Consortium*](https://es.wikipedia.org/wiki/World_Wide_Web_Consortium) ([W3C](https://es.wikipedia.org/wiki/W3C)) o Consorcio WWW, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la [World Wide Web](https://es.wikipedia.org/wiki/World_Wide_Web) (WWW). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.[1](https://es.wikipedia.org/wiki/HTML#cite_note-1)

El lenguaje HTML basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página (imagen, vídeo, [*script*](https://es.wikipedia.org/wiki/Script), entre otros.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

Sin embargo, a lo largo de sus diferentes versiones, se han incorporado y suprimido diversas características, con el fin de hacerlo más eficiente y facilitar el desarrollo de páginas web compatibles con distintos navegadores y plataformas (PC de escritorio, portátiles, [teléfonos inteligentes](https://es.wikipedia.org/wiki/Tel%C3%A9fonos_inteligentes), [tabletas](https://es.wikipedia.org/wiki/Tableta_(computadora)), vipers etc.) No obstante, para interpretar correctamente una nueva versión de HTML, los desarrolladores de navegadores web deben incorporar estos cambios y el usuario debe ser capaz de usar la nueva versión del navegador con los cambios incorporados. Normalmente los cambios son aplicados mediante parches de actualización automática ([Firefox](https://es.wikipedia.org/wiki/Firefox), [Chrome](https://es.wikipedia.org/wiki/Google_Chrome)) u ofreciendo una nueva versión del navegador con todos los cambios incorporados, en un sitio web de descarga oficial ([Internet Explorer](https://es.wikipedia.org/wiki/Internet_Explorer)). Por lo que un navegador desactualizado no será capaz de interpretar correctamente una página web escrita en una versión de HTML superior a la que pueda interpretar, lo que obliga muchas veces a los desarrolladores a aplicar técnicas y cambios que permitan corregir problemas de visualización e incluso de interpretación de código HTML. Así mismo, las páginas escritas en una versión anterior de HTML deberían ser actualizadas o reescritas, lo que no siempre se cumple. Es por ello que ciertos navegadores todavía mantienen la capacidad de interpretar páginas web de versiones HTML anteriores. Por estas razones, todavía existen diferencias entre distintos navegadores y versiones al interpretar una misma página web.



## Primeras especificaciones de HTML

[Tim Berners-Lee](https://es.wikipedia.org/wiki/Tim_Berners-Lee) (TBL) en 1991[2](https://es.wikipedia.org/wiki/HTML#cite_note-2) [3](https://es.wikipedia.org/wiki/HTML#cite_note-3) describe 18 elementos que incluyen el diseño inicial y relativamente simple de HTML. Trece de estos elementos todavía existen en HTML 4.[4](https://es.wikipedia.org/wiki/HTML#cite_note-4)

Berners-Lee consideraba el HTML una ampliación de [SGML](https://es.wikipedia.org/wiki/SGML), pero no fue formalmente reconocida como tal hasta la publicación a mediados de 1993, por la [IETF](https://es.wikipedia.org/wiki/IETF) (en español: Grupo de Trabajo de Ingeniería de Internet), de una primera proposición para una especificación del HTML: el borrador del *Hypertext Markup Language* de Berners-Lee y [Dan Connolly](https://es.wikipedia.org/w/index.php?title=Dan_Connolly&action=edit&redlink=1), que incluía una [Definición de Tipo de Documento](https://es.wikipedia.org/wiki/DTD) SGML para definir la gramática.[5](https://es.wikipedia.org/wiki/HTML#cite_note-5) El borrador expiró a los seis meses, pero fue notable por su reconocimiento de la etiqueta propia del navegador [Mosaic](https://es.wikipedia.org/wiki/Mosaic) usada para insertar imágenes sin cambio de línea, que reflejaba la filosofía del IETF de basar estándares en prototipos con éxito. [6](https://es.wikipedia.org/wiki/HTML#cite_note-raymond-6) De la misma manera, el borrador competidor de Dave Raggett *HTML+ (Hypertext Markup Format)* (Formato de Marcaje de Hipertexto), de finales de 1993, sugería estandarizar características ya implementadas, como las tablas.[7](https://es.wikipedia.org/wiki/HTML#cite_note-raggett-7)

## Marcador HTML

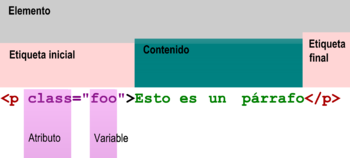
El HTML se escribe en forma de «etiquetas», rodeadas por [corchetes angulares](https://es.wikipedia.org/wiki/Par%C3%A9ntesis) (<,>,/). El HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir o hacer referencia a un tipo de programa llamado [*script*](https://es.wikipedia.org/wiki/Script), el cual puede afectar el comportamiento de [navegadores web](https://es.wikipedia.org/wiki/Navegador_web) y otros procesadores de HTML.[8](https://es.wikipedia.org/wiki/HTML#cite_note-8)

HTML también sirve para referirse al contenido del [tipo de MIME](https://es.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions) text/html o todavía más ampliamente como un término genérico para el HTML, ya sea en forma descendida del [XML](https://es.wikipedia.org/wiki/XML) (como [XHTML](https://es.wikipedia.org/wiki/XHTML) 1.0 y posteriores) o en forma descendida directamente de [SGML](https://es.wikipedia.org/wiki/SGML) (como HTML 4.01 y anteriores).

HTML consta de varios componentes vitales, entre ellos los *elementos* y sus *atributos*, *tipos de data* y la *declaración de tipo de documento*.

### Elementos

Los elementos son la estructura básica de HTML. Los elementos tienen dos propiedades básicas: atributos y contenido. Cada atributo y contenido tiene ciertas restricciones para que se considere válido al documento HTML. Un elemento generalmente tiene una etiqueta de inicio (por ejemplo, <nombre-de-elemento>) y una etiqueta de cierre (por ejemplo, </nombre-de-elemento>). Los atributos del elemento están contenidos en la etiqueta de inicio y el contenido está ubicado entre las dos etiquetas (por ejemplo, <nombre-de-elemento atributo="valor">Contenido</nombre-de-elemento>). Algunos elementos, tales como <br>, no tienen contenido ni llevan una etiqueta de cierre. Debajo se listan varios tipos de elementos de marcado usados en HTML.

[](https://commons.wikimedia.org/wiki/File:Etiquetas_en_HTML.png)

Estructura general de una línea de código en el lenguaje de etiquetas HTML.

El marcado *estructural* describe el propósito del texto. Por ejemplo, <h2>Golf</h2> establece «Golf» como un encabezamiento de segundo nivel, el cual se mostraría en un navegador de una manera similar al título «Marcado HTML» al principio de esta sección. El marcado estructural no define cómo se verá el elemento, pero la mayoría de los navegadores web han estandarizado el formato de los elementos. Puede aplicarse un formato específico al texto por medio de [hojas de estilo en cascada](https://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada).

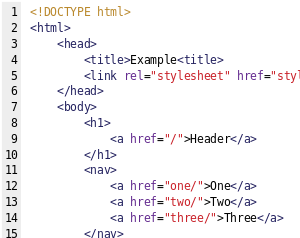
El marcado *presentacional* describe la apariencia del texto, sin importar su función. Por ejemplo, <b>negrita</b> indica que los navegadores web visuales deben mostrar el texto en **negrita**, pero no indica qué deben hacer los navegadores web que muestran el contenido de otra manera (por ejemplo, los que leen el texto en voz alta). En el caso de <b>negrita</b> e <i>itálica</i>, existen elementos que se ven de la misma manera pero tienen una naturaleza más semántica: <strong>énfasis fuerte</strong> y <em>énfasis</em>. Es fácil ver cómo un [lector de pantalla](https://es.wikipedia.org/wiki/Lector_de_pantalla) debería interpretar estos dos elementos. Sin embargo, son equivalentes a sus correspondientes elementos presentacionales: un lector de pantalla no debería decir más fuerte el nombre de un libro, aunque el nombre resalte en *itálicas* en una pantalla. La mayoría del marcado presentacional ha sido desechada con HTML 4.0, en favor de hojas de estilo en cascada.

El marcado *hipertextual* se utiliza para enlazar partes del documento con otros documentos o con otras partes del mismo documento. Para crear un enlace es necesario utilizar la etiqueta de ancla <a>junto con el atributo href, que establecerá la dirección URL a la que apunta el enlace. Por ejemplo, un enlace que muestre el texto de la dirección y vaya hacia nuestra Wikipedia podría ser de la forma <a href=”http://www.wikipedia.org”>http://www.wikipedia.org</a>. También se pueden crear enlaces sobre otros objetos, tales como imágenes <a href=”enlace”><img src=”imagen” /></a>.

### Atributos

La mayoría de los atributos de un elemento son pares nombre-valor, separados por un signo de igual «=» y escritos en la etiqueta de comienzo de un elemento, después del nombre del elemento. El valor puede estar rodeado por comillas dobles o simples, aunque ciertos tipos de valores pueden estar sin comillas en HTML (pero no en XHTML).[9](https://es.wikipedia.org/wiki/HTML#cite_note-9) [10](https://es.wikipedia.org/wiki/HTML#cite_note-10) De todas maneras, dejar los valores sin comillas es considerado poco seguro.[11](https://es.wikipedia.org/wiki/HTML#cite_note-11) En contraste con los pares nombre-elemento, hay algunos atributos que afectan al elemento simplemente por su presencia[12](https://es.wikipedia.org/wiki/HTML#cite_note-12) (tal como el atributo ismap para el elemento img).[13](https://es.wikipedia.org/wiki/HTML#cite_note-13)

## Códigos HTML básicos

[](https://commons.wikimedia.org/wiki/File:HTML_source_code_example.svg)

Un ejemplo de código HTML con [coloreado de sintaxis](https://es.wikipedia.org/wiki/Coloreado_de_sintaxis).

* <html>: define el inicio del documento HTML, le indica al [navegador](https://es.wikipedia.org/wiki/Navegador_web) que lo que viene a continuación debe ser interpretado como código HTML. Esto es así [*de facto*](https://es.wikipedia.org/wiki/De_facto), ya que en teoría lo que define el tipo de documento es el [DOCTYPE](https://es.wikipedia.org/wiki/DOCTYPE), que significa la palabra justo tras DOCTYPE el tag de raíz.
* <script>: incrusta un [script](https://es.wikipedia.org/wiki/Script) en una web, o llama a uno mediante src="url del script". Se recomienda incluir el [tipo MIME](https://es.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions) en el atributo type, en el caso de [JavaScript](https://es.wikipedia.org/wiki/JavaScript) text/javascript.
* <head>: define la [cabecera](https://es.wikipedia.org/wiki/Cabecera_(inform%C3%A1tica)) del documento HTML; esta cabecera suele contener información sobre el documento que no se muestra directamente al [usuario](https://es.wikipedia.org/wiki/Usuario) como, por ejemplo, el título de la ventana del navegador. Dentro de la cabecera <head> es posible encontrar:
  + <title>: define el título de la página. Por lo general, el título aparece en la barra de título encima de la ventana.
  + <link>: para vincular el sitio a [hojas de estilo](https://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada) o iconos. Por ejemplo:<link rel="stylesheet" href="/style.css" type="text/css">.
  + <style>: para colocar el estilo interno de la página; ya sea usando [CSS](https://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada) u otros lenguajes similares. No es necesario colocarlo si se va a vincular a un archivo externo usando la etiqueta <link>.
  + <meta>: para metadatos como la autoría o la licencia, incluso para indicar parámetros http (mediante http-equiv="") cuando no se pueden modificar por no estar disponible la configuración o por dificultades con server-side scripting.
* <body>: define el contenido principal o cuerpo del documento. Esta es la parte del documento html que se muestra en el navegador; dentro de esta etiqueta pueden definirse propiedades comunes a toda la página, como [color](https://es.wikipedia.org/wiki/HTML_(colores)) de fondo y márgenes. Dentro del cuerpo <body> es posible encontrar numerosas etiquetas. A continuación se indican algunas a modo de ejemplo:
  + <article>: Representa una composición auto-contenida en un documento, página, una aplicación o en el sitio, que se destina a distribuir de forma independiente o re-utilizable.
  + <h1> a <h6>: encabezados o títulos del documento con diferente relevancia.
  + <table>: define una tabla.
    - <tr>: fila de una tabla.
    - <td>: celda de una tabla (debe estar dentro de una fila).
    - <footer> : representa el pie de un documento o sección. La información que se suele añadir en este bloque es el autor del documento, enlaces a contenido relacionado, información de copyright, avisos legales, etc.
  + <a>: [hipervínculo](https://es.wikipedia.org/wiki/Hiperv%C3%ADnculo) o enlace, dentro o fuera del [sitio web](https://es.wikipedia.org/wiki/Sitio_web). Debe definirse el parámetro de pasada por medio del atributo *href*. Por ejemplo: <a href="http://www.example.com" title="Ejemplo" target="\_blank" tabindex="1">Ejemplo</a> se representa como ejemplo.[14](https://es.wikipedia.org/wiki/HTML#cite_note-14)
  + <div>: división de la página. Se recomienda, junto con css, en vez de <table> cuando se desea alinear contenido.
  + <img>: imagen. Requiere del atributo *src*, que indica la ruta en la que se encuentra la imagen. Por ejemplo: <img src="./imágenes/mifoto.jpg" />. Es conveniente, por accesibilidad, poner un atributo alt="texto alternativo".
  + <li><ol><ul>: etiquetas para listas.
  + <b>: texto en negrita (*etiqueta desaprobada. Se recomienda usar la etiqueta <strong>*).
  + <i>: texto en cursiva (*etiqueta desaprobada. Se recomienda usar la etiqueta <em>*).
  + <s>: texto tachado (*etiqueta desaprobada. Se recomienda usar la etiqueta <del>*).
  + <u>: Antes texto subrayado. A partir de HTML 5 define porciones de texto diferenciadas o destacadas del resto, para indicar correcciones por ejemplo (etiqueta desaprobada en HTML 4.01 y redefinida en HTML 5).[15](https://es.wikipedia.org/wiki/HTML#cite_note-15) [16](https://es.wikipedia.org/wiki/HTML#cite_note-16)
  + <main>: división estructural de la página que engloba el contenido principal de la misma. Dentro de esta etiqueta, por ejemplo, encontramos los <article>
  + <span> : Sirve para diferenciar un texto de otro.

La mayoría de etiquetas deben cerrarse como se abren, pero con una barra («/») tal como se muestra en los siguientes ejemplos:

* <table><tr><td>Contenido de una celda</td></tr></table>.
* <script>Código de un script integrado en la página</script>
* <header> División estructural en la parte del contenido.

## Colores para HTML[

Hay muchas formas de ponerle color a una página web con el lenguaje HTML, una de ellas es poner el siguiente atributo a un elemento: style="color: green" y el fondo se hará de color verde, aunque no se recomienda. La mejor solución es usar hojas de estilo . Otro caso sería añadir estilos a la letra, veamos un ejemplo: Texto de prueba y así la letra se pondrá de color verde.

## Nociones básicas de HTML

El lenguaje HTML puede ser creado y editado con cualquier [editor de textos](https://es.wikipedia.org/wiki/Editor_de_texto) básico, como puede ser [Gedit](https://es.wikipedia.org/wiki/Gedit) en [GNU/Linux](https://es.wikipedia.org/wiki/GNU/Linux), el [Bloc de notas](https://es.wikipedia.org/wiki/Bloc_de_notas) de Windows, o cualquier otro editor que admita texto sin formato como [GNU Emacs](https://es.wikipedia.org/wiki/GNU_Emacs), [Microsoft Wordpad](https://es.wikipedia.org/wiki/Wordpad), [TextPad](https://es.wikipedia.org/wiki/TextPad), [Vim](https://es.wikipedia.org/wiki/Vim), [Notepad++](https://es.wikipedia.org/wiki/Notepad%2B%2B), entre otros.

Existen, además, otros editores para la realización de sitios web con características [WYSIWYG](https://es.wikipedia.org/wiki/WYSIWYG) (*What You See Is What You Get*, o en español: «lo que ves es lo que obtienes»). Estos editores permiten ver el resultado de lo que se está editando en tiempo real, a medida que se va desarrollando el documento. Ahora bien, esto no significa una manera distinta de realizar sitios web, sino que una forma un tanto más simple, ya que estos programas, además de tener la opción de trabajar con la vista preliminar, tiene su propia sección HTML, la cual va generando todo el código a medida que se va trabajando. Algunos ejemplos de editores WYSIWYG son [KompoZer](https://es.wikipedia.org/wiki/KompoZer), [Microsoft FrontPage](https://es.wikipedia.org/wiki/Microsoft_FrontPage) o [Adobe Dreamweaver](https://es.wikipedia.org/wiki/Adobe_Dreamweaver).

Combinar estos dos métodos resulta muy interesante, ya que de alguna manera se ayudan entre sí. Por ejemplo, si se edita todo en HTML y el desarrollador olvida algún código o etiqueta, basta con dirigirse al editor visual o WYSIWYG y se continúa ahí la edición o viceversa, ya que hay casos en que resulta más rápido y fácil escribir directamente el código de alguna característica que el usuario desea adherir al sitio que buscar la opción en el programa mismo.

Existe otro tipo de editores HTML llamados [WYSIWYM](https://es.wikipedia.org/wiki/WYSIWYM) que dan más importancia al contenido y al significado que a la apariencia visual. Entre los objetivos que tienen estos editores es la separación del contenido y la presentación, fundamental en el diseño web.

HTML utiliza etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determina la forma en la que debe aparecer en su [navegador](https://es.wikipedia.org/wiki/Navegador_web) el texto, así como también las imágenes y los demás elementos, en la pantalla del ordenador.

Toda etiqueta se identifica porque está encerrada entre los signos menor que y mayor que (<>), y algunas tienen atributos que pueden tomar algún valor. En general las etiquetas se aplicarán de dos formas especiales:

* Se abren y se cierran, como por ejemplo: <b>negrita</b>, que se vería en su [navegador web](https://es.wikipedia.org/wiki/Navegador_web) como **negrita**.
* No pueden abrirse y cerrarse, como <hr />, que se vería en su [navegador web](https://es.wikipedia.org/wiki/Navegador_web) como una línea horizontal.
* Otras que pueden abrirse y cerrarse, como por ejemplo <p>.
* Las etiquetas básicas o mínimas son:

<!DOCTYPE HTML>

<**html**>

<**head**>

<**title**>Ejemplo1</**title**>

</**head**>

<**body**>

<**p**>Párrafo de ejemplo</**p**>

</**body**>

</**html**>

## Aprender HTML analizando páginas reales

Seleccionando la opción «ver código fuente» en el navegador, se puede ver realmente la información que está recibiendo el [navegador web](https://es.wikipedia.org/wiki/Navegador_web) y cómo la está interpretando.

[](https://commons.wikimedia.org/wiki/File:Captura_inspector_de_p%C3%A1gina_en_Firefox.png)

Captura del inspector de página de Firefox.

Por ejemplo: en [Internet Explorer](https://es.wikipedia.org/wiki/Internet_Explorer) o en [Firefox](https://es.wikipedia.org/wiki/Firefox), simplemente hay que desplegar el menú «ver» y luego elegir «código fuente», mientras que en [Chrome](https://es.wikipedia.org/wiki/Google_Chrome) presionar Ctrl+U. De esta forma, se abrirá el editor de texto configurado como predeterminado en el sistema con el código fuente de la página que se esté viendo en ese momento en el explorador. Otra forma más rápida consiste en hacer clic con el botón derecho del ratón en cualquier punto del área donde el navegador muestra la página web y elegir «ver código fuente».

Aparte de poder ver el código fuente HTML de una página web con las opciones antes descritas, Internet Explorer, Firefox y Google Chrome incorporan también unas herramientas conocidas como inspectores de página que se puede activar con F12.

Con estas herramientas es posible visualizar una página web y seleccionar dentro de ella un elemento concreto del cuál queremos conocer cuál es el código HTML con el que está hecho señalando el elemento en cuestión simplemente con el ratón. Al hacer esto, el código se mostrará en un área especial dentro del navegador en el que el usuario podrá ver el código HTML en cuestión (ver imagen), además, de las reglas [CSS](https://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada) que aplican a ese código HTML en concreto. Este tipo de análisis resulta sumamente instructivo para aprender a desarrollar en HTML.

Para el navegador Firefox, además, existe como alternativa a la herramienta nativa el plugin [Firebug](https://es.wikipedia.org/wiki/Firebug), muy similar a la herramienta que Firefox incorpora por defecto.

## Historia del estándar

En 1989 existían dos técnicas que permitían vincular documentos electrónicos, por un lado los hipervínculos o enlaces (*hiperlinks* o *links*) y por otro lado un poderoso lenguaje de etiquetas denominado [SGML](https://es.wikipedia.org/wiki/SGML). Por entonces, [Tim Berners-Lee](https://es.wikipedia.org/wiki/Tim_Berners-Lee), quien trabajaba en el Centro Europeo de Investigaciones Nucleares [CERN](https://es.wikipedia.org/wiki/CERN) da a conocer a la prensa que estaba trabajando en un sistema que va a permitir acceder a ficheros en línea que funcionaba sobre redes de computadoras o máquinas electrónicas basadas en el protocolo [TCP/IP](https://es.wikipedia.org/wiki/TCP/IP). Inicialmente fue desarrollado para que se pudiera compartir fácilmente información entre científicos de distintas universidades e institutos de investigación de todo el mundo.[17](https://es.wikipedia.org/wiki/HTML#cite_note-17)

A principios de [1990](https://es.wikipedia.org/wiki/1990), define por fin el HTML como un subconjunto del conocido [SGML](https://es.wikipedia.org/wiki/SGML) y crea algo más valioso incluso, el [World Wide Web](https://es.wikipedia.org/wiki/World_Wide_Web).

[Tim Berners-Lee](https://es.wikipedia.org/wiki/Tim_Berners-Lee) creó el proyecto [World Wide Web](https://es.wikipedia.org/wiki/World_Wide_Web) (Tejido o Telaraña Mundial), así como un sistema que facilitaba la lectura de información, mediante un programa de navegación. Sería el primer navegador web, llamado WorldWideWeb, y desarrollado durante la segunda mitad del año 1990; siendo tiempo después rebautizado como Nexus, para evitar confusiones por su nombre que era igual al de la tecnología que representaba. Le siguieron otros dos navegadores: el Line Mode Browser y el [ViolaWWW](https://es.wikipedia.org/wiki/ViolaWWW). Este último, desarrollado en 1992, fue el primer navegador en popularizarse entre los primeros usuarios de la World Wide Web.[18](https://es.wikipedia.org/wiki/HTML#cite_note-18)

[Pei-Yuan Wei](https://es.wikipedia.org/w/index.php?title=Pei-Yuan_Wei&action=edit&redlink=1) presentó el [ViolaWWW](https://es.wikipedia.org/wiki/ViolaWWW),[19](https://es.wikipedia.org/wiki/HTML#cite_note-19) que funcionaría en modo texto y sobre un sistema operativo [UNIX](https://es.wikipedia.org/wiki/UNIX).

Los trabajos para crear un sucesor del HTML, denominado [HTML +](https://es.wikipedia.org/wiki/HTML_%2B), comenzaron a finales de [1993](https://es.wikipedia.org/wiki/1993). HTML+ se diseñó originalmente para ser un superconjunto del HTML que permitiera evolucionar gradualmente desde el formato HTML anterior. A la primera especificación formal de HTML+ se le dio, por lo tanto, el número de versión 2 para distinguirla de las propuestas no oficiales previas. Los trabajos sobre HTML+ continuaron, pero nunca se convirtió en un estándar, a pesar de ser la base formalmente más parecida al aspecto compositivo de las especificaciones actuales.

El borrador del [estándar](https://es.wikipedia.org/wiki/Normalizaci%C3%B3n) HTML 3.0 fue propuesto por el recién formado [W3C](https://es.wikipedia.org/wiki/W3C) en marzo de [1995](https://es.wikipedia.org/wiki/1995). Con él se introdujeron muchas nuevas capacidades; por ejemplo, facilidades para crear tablas, hacer que el texto fluyese alrededor de las figuras y mostrar elementos matemáticos complejos. Aunque se diseñó para ser compatible con [HTML 2.0](https://es.wikipedia.org/wiki/HTML_2.0), era demasiado complejo para ser implementado con la tecnología de la época, y cuando el borrador del estándar expiró en septiembre de [1995](https://es.wikipedia.org/wiki/1995), se abandonó debido a la carencia de apoyos de los fabricantes de [navegadores web](https://es.wikipedia.org/wiki/Navegador_web). El [HTML 3.1](https://es.wikipedia.org/w/index.php?title=HTML_3.1&action=edit&redlink=1) nunca llegó a ser propuesto oficialmente, y el estándar siguiente fue el [HTML 3.2](https://es.wikipedia.org/w/index.php?title=HTML_3.2&action=edit&redlink=1), que abandonaba la mayoría de las nuevas características del [HTML 3.0](https://es.wikipedia.org/wiki/HTML_3.0) y, a cambio, adoptaba muchos elementos desarrollados inicialmente por los navegadores web [Netscape](https://es.wikipedia.org/wiki/Netscape_Navigator) y [Mosaic](https://es.wikipedia.org/wiki/Mosaic). La posibilidad de trabajar con fórmulas matemáticas que se había propuesto en el HTML 3.0 pasó a quedar integrada en un estándar distinto llamado [MathML](https://es.wikipedia.org/wiki/MathML).

En 1997, [HTML 4.0](https://es.wikipedia.org/wiki/HTML_4.0) se publicó como una recomendación del [W3C](https://es.wikipedia.org/wiki/W3C). [HTML 4.0](https://es.wikipedia.org/wiki/HTML_4.0) adoptó muchos elementos específicos desarrollados inicialmente para un navegador web concreto, pero al mismo tiempo comenzó a limpiar el HTML señalando algunos de ellos como «desaprobados» (*deprecated*, en inglés).

[HTML 4.0](https://es.wikipedia.org/wiki/HTML_4.0) implementa características como [XForms](https://es.wikipedia.org/wiki/XForms) 1.0 que no necesitan implementar motores de navegación que eran incompatibles con algunas páginas web HTML. En 2004 la [W3C](https://es.wikipedia.org/wiki/W3C) reabrió el debate de la evolución del HTML, y se dieron a conocer las bases para la versión [HTML5](https://es.wikipedia.org/wiki/HTML5). No obstante, este trabajo fue rechazado por los miembros del [W3C](https://es.wikipedia.org/wiki/W3C) y se daría preferencia al desarrollo del XML.

[Apple](https://es.wikipedia.org/wiki/Apple_Inc.), [Mozilla](https://es.wikipedia.org/wiki/Fundaci%C3%B3n_Mozilla) y [Opera](https://es.wikipedia.org/wiki/Opera_Software) anunciaron su interés en seguir trabajando en el proyecto bajo el nombre de [WHATWG](https://es.wikipedia.org/wiki/WHATWG),[20](https://es.wikipedia.org/wiki/HTML#cite_note-20) que se basa en la compatibilidad con tecnologías anteriores.

En 2006, el [W3C](https://es.wikipedia.org/wiki/W3C) se interesó en el desarrollo de [HTML5](https://es.wikipedia.org/wiki/HTML5), y en 2007 se unió al grupo de trabajo del [WHATWG](https://es.wikipedia.org/wiki/WHATWG) para unificar el proyecto.

**HTML5** (*HyperText Markup Language*, versión 5) es la quinta revisión importante del lenguaje básico de la [World Wide Web](https://es.wikipedia.org/wiki/World_Wide_Web), [HTML](https://es.wikipedia.org/wiki/HTML). HTML5 especifica dos variantes de sintaxis para HTML: una «clásica», HTML (text/html), conocida como *HTML5*, y una variante [XHTML](https://es.wikipedia.org/wiki/XHTML) conocida como sintaxis *XHTML5* que deberá servirse con sintaxis XML (application/xhtml+xml).[1](https://es.wikipedia.org/wiki/HTML5#cite_note-w3cSpec-1) [2](https://es.wikipedia.org/wiki/HTML5#cite_note-franganilloHTML5-2) Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo. La versión definitiva de la quinta revisión del estándar se publicó en octubre de 2014.[3](https://es.wikipedia.org/wiki/HTML5#cite_note-finalosnews-3)

Al no ser reconocido en viejas versiones de navegadores por sus nuevas etiquetas, se recomienda al usuario común actualizar su navegador a la versión más nueva, para poder disfrutar de todo el potencial que provee HTML5.

El desarrollo de este [lenguaje de marcado](https://es.wikipedia.org/wiki/Lenguaje_de_marcado) es regulado por el Consorcio [W3C](https://es.wikipedia.org/wiki/W3C).

# PHP

|  |
| --- |
| **PHP** |
| [PHP-logo.svg](https://commons.wikimedia.org/wiki/File:PHP-logo.svg) |

**PHP** es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página web resultante. PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

Fue creado originalmente por Rasmus Lerdorf en el año 1995. Actualmente el lenguaje sigue siendo desarrollado con nuevas funciones por el grupo PHP. Este lenguaje forma parte del software libre publicado bajo la licencia PHP, que es incompatible con la Licencia Pública General de GNU debido a las restricciones del uso del término *PHP*.[3](https://es.wikipedia.org/wiki/PHP#cite_note-licencia-3)

## Etimología

PHP es un [acrónimo recursivo](https://es.wikipedia.org/wiki/Acr%C3%B3nimo_recursivo) que significa *PHP Hypertext Preprocessor* (inicialmente *PHP Tools*, o, *Personal Home Page Tools*).[4](https://es.wikipedia.org/wiki/PHP#cite_note-4) Fue creado originalmente por [Rasmus Lerdorf](https://es.wikipedia.org/wiki/Rasmus_Lerdorf); sin embargo, la implementación principal de PHP es producida ahora por The PHP Group y sirve como el estándar de facto para PHP, al no haber una especificación formal. Publicado con la PHP License, la [Free Software Foundation](https://es.wikipedia.org/wiki/Free_Software_Foundation) considera esta licencia como [software libre](https://es.wikipedia.org/wiki/Software_libre).[3](https://es.wikipedia.org/wiki/PHP#cite_note-licencia-3)

## Visión general

PHP puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno. El lenguaje PHP se encuentra instalado en más de 20 millones de sitios web y en un millón de servidores. El número de sitios basados en PHP se ha visto reducido progresivamente en los últimos años, con la aparición de nuevas tecnologías como Node.JS, Golang, ASP.NET, etc. El sitio web de Wikipedia está desarrollado en PHP.[5](https://es.wikipedia.org/wiki/PHP#cite_note-porcentaje-5) Es también el módulo Apache más popular entre las computadoras que utilizan Apache como servidor web.

El gran parecido que posee PHP con los lenguajes más comunes de [programación estructurada](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada), como [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) y [Perl](https://es.wikipedia.org/wiki/Perl), permiten a la mayoría de los [programadores](https://es.wikipedia.org/wiki/Programador) crear aplicaciones complejas con una curva de aprendizaje muy corta. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones.

Aunque todo en su diseño está orientado a facilitar la creación de sitios webs, es posible crear aplicaciones con una [interfaz gráfica para el usuario](https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario), utilizando alguna extensión como puede ser [PHP-Qt](https://es.wikipedia.org/wiki/PHP-Qt), [PHP-GTK](https://es.wikipedia.org/wiki/PHP-GTK),[6](https://es.wikipedia.org/wiki/PHP#cite_note-php-gtk-6) WxPHP, WinBinder, Roadsend PHP, Phalanger, Phc o HiP Hop VM. También puede ser usado desde la [línea de comandos](https://es.wikipedia.org/wiki/L%C3%ADnea_de_comandos), de la misma manera como [Perl](https://es.wikipedia.org/wiki/Perl) o [Python](https://es.wikipedia.org/wiki/Python) pueden hacerlo; a esta versión de PHP se la llama PHP-CLI (*Command Line Interface*).[7](https://es.wikipedia.org/wiki/PHP#cite_note-php-cli-7)

Cuando el cliente hace una petición al servidor para que le envíe una [página web](https://es.wikipedia.org/wiki/P%C3%A1gina_web), el servidor ejecuta el [intérprete](https://es.wikipedia.org/wiki/Int%C3%A9rprete_(inform%C3%A1tica)) de PHP. Éste procesa el [script](https://es.wikipedia.org/wiki/Script) solicitado que generará el contenido de manera dinámica (por ejemplo obteniendo información de una base de datos). El resultado es enviado por el intérprete al servidor, quien a su vez se lo envía al cliente.

Mediante extensiones es también posible la generación de archivos [PDF](https://es.wikipedia.org/wiki/PDF),[8](https://es.wikipedia.org/wiki/PHP#cite_note-pdf-8) [Flash](https://es.wikipedia.org/wiki/Adobe_Flash), así como imágenes en diferentes formatos.

Permite la conexión a diferentes tipos de servidores de bases de datos tanto [SQL](https://es.wikipedia.org/wiki/SQL) como [NoSQL](https://es.wikipedia.org/wiki/NoSQL) tales como [MySQL](https://es.wikipedia.org/wiki/MySQL), [PostgreSQL](https://es.wikipedia.org/wiki/PostgreSQL), [Oracle](https://es.wikipedia.org/wiki/Oracle_Database), [ODBC](https://es.wikipedia.org/wiki/Open_Database_Connectivity), [DB2](https://es.wikipedia.org/wiki/DB2), [Microsoft SQL Server](https://es.wikipedia.org/wiki/Microsoft_SQL_Server), [Firebird](https://es.wikipedia.org/wiki/Firebird), [SQLite](https://es.wikipedia.org/wiki/SQLite) o [MongoDB](https://es.wikipedia.org/wiki/MongoDB).[9](https://es.wikipedia.org/wiki/PHP#cite_note-9)

PHP también tiene la capacidad de ser ejecutado en la mayoría de los [sistemas operativos](https://es.wikipedia.org/wiki/Sistema_operativo), tales como [Unix](https://es.wikipedia.org/wiki/Unix) (y de ese tipo, como [Linux](https://es.wikipedia.org/wiki/N%C3%BAcleo_Linux) o [Mac OS X](https://es.wikipedia.org/wiki/OS_X)) y [Microsoft Windows](https://es.wikipedia.org/wiki/Microsoft_Windows), y puede interactuar con los [servidores de web](https://es.wikipedia.org/wiki/Servidor_web) más populares ya que existe en versión [CGI](https://es.wikipedia.org/wiki/Interfaz_de_entrada_com%C3%BAn), módulo para [Apache](https://es.wikipedia.org/wiki/Servidor_HTTP_Apache), e [ISAPI](https://es.wikipedia.org/wiki/ISAPI).

PHP es una alternativa a las tecnologías de [Microsoft](https://es.wikipedia.org/wiki/Microsoft) [ASP](https://es.wikipedia.org/wiki/Active_Server_Pages) y [ASP.NET](https://es.wikipedia.org/wiki/ASP.NET) (que utiliza [C#](https://es.wikipedia.org/wiki/C_Sharp) y [Visual Basic .NET](https://es.wikipedia.org/wiki/Visual_Basic_.NET) como lenguajes), a [ColdFusion](https://es.wikipedia.org/wiki/ColdFusion) de la empresa [Adobe](https://es.wikipedia.org/wiki/Adobe), a [JSP](https://es.wikipedia.org/wiki/JavaServer_Pages)/[Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)), [CGI](https://es.wikipedia.org/wiki/Interfaz_de_entrada_com%C3%BAn)/[Perl](https://es.wikipedia.org/wiki/Perl) y a [Node.js](https://es.wikipedia.org/wiki/Node.js)/[Javascript](https://es.wikipedia.org/wiki/Javascript). Aunque su creación y desarrollo se da en el ámbito de los sistemas libres, bajo la licencia [GNU](https://es.wikipedia.org/wiki/GNU), existe además un entorno de desarrollo integrado comercial llamado [Zend Studio](https://es.wikipedia.org/wiki/Zend_Studio). [CodeGear](https://es.wikipedia.org/wiki/CodeGear) (la división de lenguajes de programación de [Borland](https://es.wikipedia.org/wiki/Borland)) ha sacado al mercado un entorno de desarrollo integrado para PHP, denominado 'Delphi for PHP. También existen al menos un par de módulos para [Eclipse](https://es.wikipedia.org/wiki/Eclipse_(software)), uno de los entornos más populares.[10](https://es.wikipedia.org/wiki/PHP#cite_note-plugins_eclipse-10)

## Historia

Fue originalmente diseñado en [Perl](https://es.wikipedia.org/wiki/Perl), con base en la escritura de un grupo de [CGI](https://es.wikipedia.org/wiki/Interfaz_de_entrada_com%C3%BAn) binarios escritos en el [lenguaje C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) por el [programador](https://es.wikipedia.org/wiki/Programador) danés-canadiense [Rasmus Lerdorf](https://es.wikipedia.org/wiki/Rasmus_Lerdorf) en el año 1994 para mostrar su [currículum vítae](https://es.wikipedia.org/wiki/Curr%C3%ADculum_v%C3%ADtae) y guardar ciertos datos, como la cantidad de tráfico que su página web recibía. El 8 de junio de 1995 fue publicado "Personal Home Page Tools" después de que Lerdorf lo combinara con su propio *Form Interpreter* para crear PHP/FI.

Dos programadores [israelíes](https://es.wikipedia.org/wiki/Israel) del [Technion](https://es.wikipedia.org/wiki/Technion), [Zeev Suraski](https://es.wikipedia.org/wiki/Zeev_Suraski) y [Andi Gutmans](https://es.wikipedia.org/wiki/Andi_Gutmans), reescribieron el [analizador sintáctico](https://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico) (*parser*, en [inglés](https://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s)) en 1997 y crearon la base del PHP3, y cambiaron el nombre del lenguaje por *PHP: Hypertext Preprocessor*.[2](https://es.wikipedia.org/wiki/PHP#cite_note-historia-2) Inmediatamente comenzaron experimentaciones públicas de PHP3, y se publicó oficialmente en junio de 1998. Para 1999, Suraski y Gutmans reescribieron el código de PHP, y produjeron lo que hoy se conoce como [motor Zend](https://es.wikipedia.org/wiki/Motor_Zend). También fundaron [Zend Technologies](https://es.wikipedia.org/wiki/Zend_Technologies) en [Ramat Gan](https://es.wikipedia.org/wiki/Ramat_Gan), [Israel](https://es.wikipedia.org/wiki/Israel).[2](https://es.wikipedia.org/wiki/PHP#cite_note-historia-2)

En mayo del 2000, PHP 4 se lanzó bajo el poder del [motor Zend](https://es.wikipedia.org/wiki/Motor_Zend) 1.0. El 13 de julio del 2007 se anunció la suspensión del soporte y desarrollo de la versión 4 de PHP,[11](https://es.wikipedia.org/wiki/PHP#cite_note-11) y, a pesar de lo anunciado, se ha liberado una nueva versión con mejoras de seguridad, la 4.4.8, publicada el 13 de enero del 2008, y posteriormente la versión 4.4.9, publicada el 7 de agosto del 2008.[12](https://es.wikipedia.org/wiki/PHP#cite_note-12) Según esta noticia,[13](https://es.wikipedia.org/wiki/PHP#cite_note-13) se le dio soporte a fallos críticos hasta el 9 de agosto del 2008.

El 13 de julio del 2004, se lanzó PHP 5, utilizando el motor Zend Engine 2.0 (o Zend Engine 2).[2](https://es.wikipedia.org/wiki/PHP#cite_note-historia-2) Incluye todas las ventajas que provee el nuevo Zend Engine 2, como:

* Mejor soporte para la [programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), que en versiones anteriores era extremadamente rudimentario.
* Mejoras de rendimiento
* Mejor soporte para [MySQL](https://es.wikipedia.org/wiki/MySQL) con extensión completamente reescrita
* Mejor soporte a [XML](https://es.wikipedia.org/wiki/Extensible_Markup_Language) (XPath, DOM, etc.)
* Soporte nativo para [SQLite](https://es.wikipedia.org/wiki/SQLite)
* Soporte integrado para [SOAP](https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol)
* [Iteradores](https://es.wikipedia.org/wiki/Iterador_(patr%C3%B3n_de_dise%C3%B1o)) de datos
* [Manejo de excepciones](https://es.wikipedia.org/wiki/Manejo_de_excepciones)
* Mejoras con la implementación con [Oracle](https://es.wikipedia.org/wiki/Oracle_Database)

# JavaScript

|  |  |
| --- | --- |
| **JavaScript** | |
| **Información general** | |
| **Paradigma** | Multiparadigma, programación funcional,[1](https://es.wikipedia.org/wiki/JavaScript#cite_note-jsfunc-1)programación basada en prototipos, imperativo, interpretado (*scripting*) |
| **Apareció en** | 1995 |
| **Diseñado por** | Netscape Communications Corp, Mozilla Fundation |
| **Última versión estable** | ECMAScript 2016; (17 de junio de 2016 (9 meses y 20 días)) |
| **Sistema de tipos** | Débil, dinámico, *duck* |
| **Implementaciones** | SpiderMonkey, Rhino, KJS, [JavaScriptCore](https://es.wikipedia.org/wiki/JavaScriptCore), V8, Chakra. |
| **Dialectos** | ECMAScript |
| **Influido por** | Java, Perl, Self, Python, C, Scheme |
| **Ha influido a** | ObjectiveJ, JScript, JScript .NET, TIScript |
| [[editar datos en Wikidata](https://www.wikidata.org/wiki/Q2005)] | |

|  |  |
| --- | --- |
| **JavaScript** | |
| **Información general** | |
| **Extensión de archivo** | .js |
| **Tipo de MIME** | application/javascript, text/javascript[2](https://es.wikipedia.org/wiki/JavaScript#cite_note-2) |
| **Tipo de formato** | *Scripting* |
| **Formato abierto** | ***?*** |
|  | |

**JavaScript** (abreviado comúnmente **JS**) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,[3](https://es.wikipedia.org/wiki/JavaScript#cite_note-3) basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Desde el 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1, una versión de javascript. Los navegadores más antiguos soportan por lo menos ECMAScript 3. La sexta edición se liberó en julio del 2015.

JavaScript se diseñó con una sintaxis similar a C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como AJAX. JavaScript se interpreta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

Desde el lanzamiento en junio de 1997 del estándar ECMAScript 1, han existido las versiones 2, 3 y 5, que es la más usada actualmente (la 4 se abandonó). En junio de 2015 se cerró y publicó la versión ECMAScript 6[7](https://es.wikipedia.org/wiki/JavaScript#cite_note-7) .

## Historia

### Nacimiento de JavaScript

JavaScript fue desarrollado originalmente por [Brendan Eich](https://es.wikipedia.org/wiki/Brendan_Eich) de [Netscape](https://es.wikipedia.org/wiki/Netscape_Communications_Corporation) con el nombre de *Mocha*, el cual fue renombrado posteriormente a *LiveScript*, para finalmente quedar como JavaScript. El cambio de nombre coincidió aproximadamente con el momento en que Netscape agregó compatibilidad con la tecnología Java en su [navegador web Netscape Navigator](https://es.wikipedia.org/wiki/Netscape_Navigator) en la versión 2.002 en diciembre de 1995. La denominación produjo confusión, dando la impresión de que el lenguaje es una prolongación de Java, y se ha caracterizado por muchos como una estrategia de mercadotecnia de Netscape para obtener prestigio e innovar en el ámbito de los nuevos lenguajes de programación web.[8](https://es.wikipedia.org/wiki/JavaScript#cite_note-8) [9](https://es.wikipedia.org/wiki/JavaScript#cite_note-9)

«JAVASCRIPT» es una [marca registrada](https://es.wikipedia.org/wiki/Marca_registrada) de [Oracle Corporation](https://es.wikipedia.org/wiki/Oracle_Corporation).[10](https://es.wikipedia.org/wiki/JavaScript#cite_note-10) Es usada con licencia por los productos creados por [Netscape Communications](https://es.wikipedia.org/wiki/Netscape_Communications) y entidades actuales como la [Fundación Mozilla](https://es.wikipedia.org/wiki/Fundaci%C3%B3n_Mozilla).

[Microsoft](https://es.wikipedia.org/wiki/Microsoft) dio como nombre a su dialecto de JavaScript «[JScript](https://es.wikipedia.org/wiki/JScript)», para evitar problemas relacionadas con la marca. JScript fue adoptado en la versión 3.0 de Internet Explorer, liberado en agosto de 1996, e incluyó compatibilidad con el [Efecto 2000](https://es.wikipedia.org/wiki/Efecto_2000) con las funciones de fecha, una diferencia de los que se basaban en ese momento. Los dialectos pueden parecer tan similares que los términos «JavaScript» y «JScript» a menudo se utilizan indistintamente, pero la especificación de JScript es incompatible con la de ECMA en muchos aspectos.

Para evitar estas incompatibilidades, el [World Wide Web Consortium](https://es.wikipedia.org/wiki/World_Wide_Web_Consortium) diseñó el estándar [Document Object Model](https://es.wikipedia.org/wiki/Document_Object_Model) (DOM, o Modelo de Objetos del Documento en español), que incorporan [Konqueror](https://es.wikipedia.org/wiki/Konqueror), las versiones 6 de [Internet Explorer](https://es.wikipedia.org/wiki/Internet_Explorer) y [Netscape Navigator](https://es.wikipedia.org/wiki/Netscape_Navigator), [Opera](https://es.wikipedia.org/wiki/Opera_(navegador)) la versión 7, [Mozilla Application Suite](https://es.wikipedia.org/wiki/Mozilla_Application_Suite) y [Mozilla Firefox](https://es.wikipedia.org/wiki/Mozilla_Firefox) desde su primera versión.

En 1997 los autores propusieron[13](https://es.wikipedia.org/wiki/JavaScript#cite_note-13) JavaScript para que fuera adoptado como estándar de la European Computer Manufacturers 'Association [ECMA](https://es.wikipedia.org/wiki/ECMA), que a pesar de su nombre no es europeo sino internacional, con sede en Ginebra. En junio de 1997 fue adoptado como un estándar ECMA, con el nombre de [ECMAScript](https://es.wikipedia.org/wiki/ECMAScript). Poco después también como un estándar [ISO](https://es.wikipedia.org/wiki/Organizaci%C3%B3n_Internacional_para_la_Estandarizaci%C3%B3n).

### JavaScript en el lado servidor

Netscape introdujo una implementación de [script del lado del servidor](https://es.wikipedia.org/wiki/Script_del_lado_del_servidor) con [Netscape Enterprise Server](https://es.wikipedia.org/w/index.php?title=Netscape_Enterprise_Server&action=edit&redlink=1), lanzada en diciembre de 1994 (poco después del lanzamiento de JavaScript para navegadores web).[14](https://es.wikipedia.org/wiki/JavaScript#cite_note-Newscape_JavaScript_Guide-1998-14) [15](https://es.wikipedia.org/wiki/JavaScript#cite_note-Morgan-Netcape_LiveWire-15) A partir de mediados de la década de los 2000, ha habido una proliferación de implementaciones de JavaScript para el lado servidor. [Node.js](https://es.wikipedia.org/wiki/Node.js) es uno de los notables ejemplos de JavaScript en el lado del servidor, siendo usado en proyectos importantes.[16](https://es.wikipedia.org/wiki/JavaScript#cite_note-RWW-Server-Side-2009-12-17-16) [17](https://es.wikipedia.org/wiki/JavaScript#cite_note-node.js-about-17)

### Desarrollos posteriores

JavaScript se ha convertido en uno de los lenguajes de programación más populares en internet. Al principio, sin embargo, muchos desarrolladores renegaban del lenguaje porque el público al que va dirigido lo formaban publicadores de artículos y demás aficionados, entre otras razones.[18](https://es.wikipedia.org/wiki/JavaScript#cite_note-18) La llegada de [Ajax](https://es.wikipedia.org/wiki/AJAX) devolvió JavaScript a la fama y atrajo la atención de muchos otros programadores. Como resultado de esto hubo una proliferación de un conjunto de [frameworks](https://es.wikipedia.org/wiki/Framework_para_aplicaciones_web) y [librerías](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)) de ámbito general, mejorando las prácticas de programación con JavaScript, y aumentado el uso de JavaScript fuera de los [navegadores web](https://es.wikipedia.org/wiki/Navegador_web), como se ha visto con la proliferación de entornos [JavaScript del lado del servidor](https://es.wikipedia.org/w/index.php?title=Server-side_JavaScript&action=edit&redlink=1). En enero de 2009, el proyecto [CommonJS](https://es.wikipedia.org/w/index.php?title=CommonJS&action=edit&redlink=1) fue inaugurado con el objetivo de especificar una librería para uso de tareas comunes principalmente para el desarrollo fuera del navegador web.[19](https://es.wikipedia.org/wiki/JavaScript#cite_note-19)

En junio de 2015 se cerró y publicó el estándar [ECMAScript](https://es.wikipedia.org/wiki/ECMAScript) 6[20](https://es.wikipedia.org/wiki/JavaScript#cite_note-20) [21](https://es.wikipedia.org/wiki/JavaScript#cite_note-21) con un soporte irregular entre navegadores[22](https://es.wikipedia.org/wiki/JavaScript#cite_note-22) y que dota a JavaScript de características avanzadas que se echaban de menos y que son de uso habitual en otros lenguajes como, por ejemplo, módulos para organización del código, verdaderas [clases](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)) para [programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), expresiones de flecha, iteradores, generadores o promesas para programación asíncrona.

La versión 7 de ECMAScript se conoce como ECMAScript 2016[23](https://es.wikipedia.org/wiki/JavaScript#cite_note-23) , y es la última versión disponible, publicada en junio de 2016. Se trata de la primera versión para la que se usa un nuevo procedimiento de publicación anual y un proceso de desarrollo abierto[24](https://es.wikipedia.org/wiki/JavaScript#cite_note-24) .

## Características

Las siguientes características son comunes a todas las implementaciones que se ajustan al estándar ECMAScript, a menos que especifique explícitamente en caso contrario.

### Imperativo y estructurado

JavaScript es compatible con gran parte de la [estructura de programación](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) de [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) (por ejemplo, sentencias if, bucles for, sentencias switch, etc.). Con una salvedad, en parte: en C, el [ámbito](https://es.wikipedia.org/wiki/%C3%81mbito_(programaci%C3%B3n)) de las [variables](https://es.wikipedia.org/wiki/Variable_(programaci%C3%B3n)) alcanza al bloque en el cual fueron definidas; sin embargo JavaScript no es compatible con esto, puesto que el ámbito de las variables es el de la función en la cual fueron declaradas. Esto cambia con la versión de ECMAScript 2015, ya que añade compatibilidad con block scoping por medio de la palabra clave let. Como en C, JavaScript hace distinción entre [expresiones](https://es.wikipedia.org/wiki/Expresi%C3%B3n_(inform%C3%A1tica)) y sentencias. Una diferencia sintáctica con respecto a C es la inserción automática de punto y coma, es decir, en JavaScript los puntos y coma que finalizan una sentencia pueden ser omitidos.[25](https://es.wikipedia.org/wiki/JavaScript#cite_note-FOOTNOTEFlanagan2006.7B.7B.7Bc.7D.7D.7D16-25)

### Dinámico

**Tipado dinámico**

Como en la mayoría de [lenguajes de scripting](https://es.wikipedia.org/wiki/Script), el [tipo](https://es.wikipedia.org/wiki/Sistema_de_tipos) está asociado al valor, no a la variable. Por ejemplo, una variable x en un momento dado puede estar ligada a un número y más adelante, religada a una [cadena](https://es.wikipedia.org/wiki/String). JavaScript es compatible con varias formas de comprobar el tipo de un objeto, incluyendo [duck typing](https://es.wikipedia.org/wiki/Duck_typing).[26](https://es.wikipedia.org/wiki/JavaScript#cite_note-FOOTNOTEFlanagan2006.7B.7B.7Bc.7D.7D.7D176.E2.80.93178-26) Una forma de saberlo es por medio de la palabra clave typeof.

**Objetual**

JavaScript está formado casi en su totalidad por [objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos). Los objetos en JavaScript son [arrays asociativos](https://es.wikipedia.org/wiki/Vector_asociativo), mejorados con la inclusión de prototipos (ver más adelante). Los nombres de las propiedades de los objetos son claves de tipo cadena: obj.x = 10 y obj['x'] = 10 son equivalentes, siendo la notación con punto [azúcar sintáctico](https://es.wikipedia.org/wiki/Az%C3%BAcar_sint%C3%A1ctico). Las propiedades y sus valores pueden ser creados, cambiados o eliminados en tiempo de ejecución. La mayoría de propiedades de un objeto (y aquellas que son incluidas por la cadena de la herencia prototípica) pueden ser enumeradas a por medio de la instrucción de bucle for... in. JavaScript tiene un pequeño número de objetos predefinidos como son Function y Date.

**Evaluación en tiempo de ejecución**

JavaScript incluye la función eval que permite evaluar expresiones expresadas como cadenas en tiempo de ejecución. Por ello se

recomienda que eval sea utilizado con precaución y que se opte por utilizar la función JSON.parse() en la medida de lo posible, pues resulta mucho más segura.

### Funcional

**Funciones de primera clase**

A las [funciones](https://es.wikipedia.org/wiki/Subrutinas) se les suele llamar ciudadanos de primera clase; son objetos en sí mismos. Como tal, poseen propiedades y métodos, como .call() y .bind(). Una función anidada es una función definida dentro de otra. Esta es creada cada vez que la función externa es invocada. Además, cada función creada forma una [clausura](https://es.wikipedia.org/wiki/Clausura_(inform%C3%A1tica)); es el resultado de evaluar un ámbito conteniendo en una o más variables dependientes de otro ámbito externo, incluyendo constantes, variables locales y argumentos de la función externa llamante. El resultado de la evaluación de dicha clausura forma parte del estado interno de cada objeto función, incluso después de que la función exterior concluya su evaluación.

### Lenguaje ASP

Es una tecnología del lado de servidor desarrollada por Microsoft para el desarrollo de sitio web dinámicos. ASP significa en inglés (Active Server Pages), fue liberado por Microsoft en 1996. Las páginas web desarrolladas bajo este lenguaje es necesario tener instalado Internet Information Server (IIS).

ASP no necesita ser compilado para ejecutarse. Existen varios lenguajes que se pueden utilizar para crear páginas ASP. El más utilizado es VBScript, nativo de Microsoft. ASP se puede hacer también en Perl and Jscript (no JavaScript). El código ASP puede ser insertado junto con el código HTML. Los archivos cuentan con la extensión (asp).

### Sintaxis:

<% %>

### Ventajas:

* Usa Visual Basic Script, siendo fácil para los usuarios.
* Comunicación óptima con SQL Server.
* Soporta el lenguaje JScript (Javascript de Microsoft).

### Desventajas:

* Código desorganizado.
* Se necesita escribir mucho código para realizar funciones sencillas.
* Tecnología propietaria.
* Hospedaje de sitios web costosos.

### Lenguaje ASP.NET

Este es un lenguaje comercializado por Microsoft, y usado por programadores para desarrollar entre otras funciones, sitios web. ASP.NET es el sucesor de la tecnología ASP, fue lanzada al mercado mediante una estrategia de mercado denominada .NET.

El ASP.NET fue desarrollado para resolver las limitantes que brindaba tu antecesor ASP. Creado para desarrollar web sencillas o grandes aplicaciones. Para el desarrollo de ASP.NET se puede utilizar C#, VB.NET o J#. Los archivos cuentan con la extensión (aspx). Para su funcionamiento de las páginas se necesita tener instalado IIS con el Framework .Net. Microsft Windows 2003 incluye este framework, solo se necesitará instalarlo en versiones anteriores.

### Ventajas:

* Completamente orientado a objetos.
* Controles de usuario y personalizados.
* División entre la capa de aplicación o diseño y el código.
* Facilita el mantenimiento de grandes aplicaciones.
* Incremento de velocidad de respuesta del servidor.
* Mayor velocidad.
* Mayor seguridad.

### Desventajas:

* Mayor consumo de recursos.

### Lenguaje JSP

Es un lenguaje para la creación de sitios web dinámicos, acrónimo de Java Server Pages. Está orientado a desarrollar páginas web en Java. JSP es un lenguaje multiplataforma. Creado para ejecutarse del lado del servidor.

JSP fue desarrollado por Sun Microsystems. Comparte ventajas similares a las de ASP.NET, desarrollado para la creación de aplicaciones web potentes. Posee un motor de páginas basado en los servlets de Java. Para su funcionamiento se necesita tener instalado un servidor Tomcat.

### Sintaxis:

<%= new java.util.Date() %>

### Características:

* Código separado de la lógica del programa.
* Las páginas son compiladas en la primera petición.
* Permite separar la parte dinámica de la estática en las páginas web.
* Los archivos se encuentran con la extensión (jsp).
* El código JSP puede ser incrustado en código HTML.

### Elementos de JSP

Los elementos que pueden ser insertados en las páginas JSP son los siguientes:

* **Código:** se puede incrustar código “Java”.
* **Directivas:** permite controlar parámetros del servlet.
* **Acciones:** permite alterar el flujo normal de ejecución de una página.

### Ventajas:

* Ejecución rápida del servlets.
* Crear páginas del lado del servidor.
* Multiplataforma.
* Código bien estructurado.
* Integridad con los módulos de Java.
* La parte dinámica está escrita en Java.
* Permite la utilización se servlets.

### Desventajas:

* Complejidad de aprendizaje.

### Lenguaje Python

Es un lenguaje de programación creado en el año 1990 por Guido van Rossum, es el sucesor del lenguaje de programación ABC. Python es comparado habitualmente con Perl. Los usuarios lo consideran como un lenguaje más limpio para programar. Permite la creación de todo tipo de programas incluyendo los sitios web.

Su código no necesita ser compilado, por lo que se llama que el código es interpretado. Es un lenguaje de programación multiparadigma, lo cual fuerza a que los programadores adopten por un estilo de programación particular:

* Programación orientada a objetos.
* Programación estructurada.
* Programación funcional.
* Programación orientada a aspectos.

### Sintaxis:

Ejemplo de una clase en Phyton:

def dibujar\_muneco(opcion):

if opcion == 1:

C.create\_line(580, 150, 580, 320, width=4, fill="blue")

C.create\_oval(510, 150, 560, 200, width=2, fill='PeachPuff')

### Ventajas:

* Libre y fuente abierta.
* Lenguaje de propósito general.
* Gran cantidad de funciones y librerías.
* Sencillo y rápido de programar.
* Multiplataforma.
* Licencia de código abierto (Opensource).
* Orientado a Objetos.
* Portable.

### Desventajas:

* Lentitud por ser un lenguaje interpretado.

### Lenguaje Ruby

Es un lenguaje interpretado de muy alto nivel y orientado a objetos. Desarrollado en el 1993 por el programador japonés Yukihiro “Matz” Matsumoto. Su sintaxis está inspirada en Phyton, Perl. Es distribuido bajo licencia de software libre (Opensource).

Ruby es un lenguaje dinámico para una programación orientada a objetos rápida y sencilla. Para los que deseen iniciarse en este lenguaje pueden encontrar un tutorial interactivo de ruby. Se encuentra también a disposición de estos usuarios un sitio con informaciones y cursos en español.

### Sintaxis:

puts "hola"

### Características:

* Existe diferencia entre mayúsculas y minúsculas.
* Múltiples expresiones por líneas, separadas por punto y coma “;”.
* Dispone de manejo de excepciones.
* Ruby puede cargar librerías de extensiones dinámicamente si el (Sistema Operativo) lo permite.
* Portátil.

### Ventajas:

* Permite desarrollar soluciones a bajo Costo.
* Software libre.
* Multiplataforma.
* **1. jQuery**
* jQuery es una librería JavaScript que permite la manipulación de documentos HTML en el DOM y los estilos CSS, la gestión de eventos, efectos y animaciones, AJAX y una API que facilita mucho la labor de los desarrolladores, compatible con todos los navegadores de última generación. El diseñador diseña, jQuery se encarga de que ese diseño se sirva sin problemas en todos los escenarios. Es una de las librerías más empleadas dentro del mundo del diseño para **la creación de proyectos digitales responsive** (con una variedad específica de jQuery llamada jQuery Mobile), un único diseño para todos los entornos (escritorio, móvil y tableta). Su versatilidad y [**capacidad para aumentar los recursos con plugins**](http://plugins.jquery.com/) la convierten en una referencia real. Grandes compañías como Google, Microsoft e IBM la usan en sus productos.
* jQuery tiene algunas otras ventajas que la hacen tan popular:
* **- La programación de los estilos es muy similar al tradicional CSS**. Si el diseñador tiene conocimientos en CSS, su adaptación es sencilla.
* **- Encadenamiento de enunciados**. Se pueden manipular distintos elementos a la vez con una única línea de código. Simplicidad.
* **- Es realmente sencillo añadirla a cualquier página web**. Para incluir dentro del código debemos colocar el <script> dentro del <head> así:
* <head>
* <script src = "jquery-1.11.3.min.js"> </ script>
* </ head>
* Es recomendable colocar el archivo JavaScript descargado en el mismo directorio utilizado para alojar las páginas web donde usemos jQuery.
* Dentro de esta librería de front-end existen dos versiones distintas, dependiendo del tipo de uso que se haga de la librería.
* **- Versión de producción**: una versión de jQuery disponible para páginas web con el código ya compactado y comprimido.
* **- Versión de desarrollo**: para pruebas (código sin comprimir).

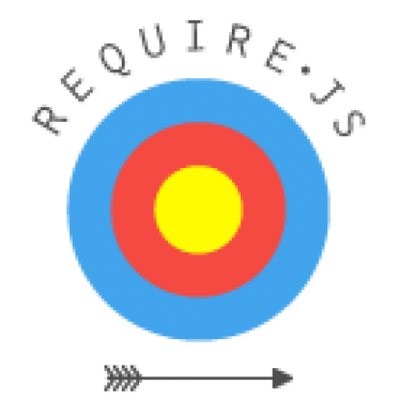


* **2. html5sql.js**
* html5sql es una librería JavaScript pensada especialmente para trabajar con la base de datos Web SQL para HTML5. Proporciona una estructura donde es posible el procesamiento de sentencias SQL en una única transacción (operaciones tratadas como una sola unidad). Todas las características de html5sql están pensadas para agilizar el desarrollo front-end:
* **- Capacidad de proceso de secuencias SQL** de varias formas distintas:
* - Como una sola serie de sentencias SQL.
* - Como una matriz de cadenas de sentencias SQL.
* - Como un conjunto de objetos de sentencias SQL.
* - Como una cadena que contiene varias sentencias SQL, cada una de las cuales finaliza en un punto y coma.
* - A partir de un archivo separado con instrucciones SQL.
* **- Dar un marco para el control de versiones de una base de datos**.



**3. RequireJS**

* RequireJS es una biblioteca JavaScript basada en la especificación AMD (Asynchronous Module Definition), que nos permite definir módulos y declarar explícitamente las dependencias necesarias para cada uno de los módulos de una aplicación hecha en JavaScript. RequireJS es una librería muy útil porque esta sintaxis no trae por defecto una manera de declarar módulos explícitamente sin añadir un número enorme de etiquetas script al código.
* Un ejemplo del código mediante el que se declaran módulos con la especificación AMD en JavaScript:
* // miModulo.js
* define(['dep1', 'dep2'], function (dep1, dep2) {
* var miModulo = function(){
* //Haz algo
* //Usa cualquier atributo de las dependencias
* dep1.atributo
* }
* return miModulo;
* });
* Algunas de las características fundamentales de RequireJS:
* **- Carga y descarga dinámica de dependencias**.
* **- Declaración explícita de dependencias**.
* **- Capacidad para crear módulos extensibles**.
* **- Posibilidad de definir el nombre de nuestros módulos**.
* **- Asignación de dependencias a una variable dentro de un módulo**.
* **- Gestión de timeouts**.
* **- Compatible con la mayoría de navegadores del mercado**: Internet Explorer, Mozilla Firefox, Google Chrome, Safari y Opera.
* Al final el objetivo de RequireJS es establecer un enfoque distintos en la carga de la escritura de las tradicionales etiquetas script de JavaScript. RequireJS lo que hace es apostar por el código modular, es decir, una sintaxis basada en identificadores de módulos en vez de direcciones URL. RequireJS lo que hace es cargar el código a partir de una baseURL, que normalmente se establece dentro del mismo directorio que el script. La librería hace una llamada al código y carga la página cuando comprueba ese atributo principal de datos.
* <!--This sets the baseUrl to the "scripts" directory, and    loads a script that will have a module ID of 'main'--><script data-main="scripts/main.js" src="scripts/require.js"></script>



**4. BonsaiJS**

* BonsaiJS es una librería JavaScript para la creación de gráficos SVG, animaciones e incorporación de archivos de audio, vídeo e imágenes sin problemas. Lo interesante del uso de JavaScript para la creación de visualizaciones de datos es que **la carga de los datos siempre se hace en el navegador, por el lado del cliente**, y nunca por el lado del servidor. Es la forma ideal de tener gráficos y animaciones de gran calidad sin coste añadido.
* No es la única ni tampoco la más conocida. De hecho, en BBVAOpen4U ya hemos hablado de otras librerías JavaScript para visualización de datos como D3.js, posiblemente la biblioteca en JS más utilizada para gráficos dinámicos.



**5. Prism.js**

PrismJS es una librería JavaScript específica para el resaltado de código en el texto de una página web. Se encarga de aplicar estilos a la sintaxis en función del tipo de lenguaje que estemos empleando. Los lenguajes que vienen por defecto son el trío habitual del diseño (HTML, JS y CSS), pero en la descarga de la librería se puede personalizar para resaltar otro tipo de lenguajes.

* PrismJS dispone de cinco temas distintos, **seis plugins para ampliar sus funcionalidades** y **la posibilidad de resaltar hasta 15 sintaxis diferentes** (no sólo HTML o JavaScript, también Java, C, PHP, Python, CoffeeScript…).
* Una vez descargada la librería, se coloca una llamada en el código HTML de la web, concretamente en el <head> y en el <body>, y a partir de ahí usando las etiquetas <pre> y <code> se puede empezar a resaltar el código que se quiera:
* **- Código insertado en el HTML**:
* <!Doctype html> <html> <head>     <link href="prism.css" rel="stylesheet" /> </head>  <body>     <script src="prism.js"></script> </body> </html>
* **- Uso de las etiquetas <pre> y <code> para resaltar código**:
* <pre class="language-xxx"><code class="language-xxx">Código</code></pre>
* 

**6. Infinity.js**

* Infinity.js es una librería que **permite cambiar el estilo de la paginación habitual en una web**. Un diseñador o un desarrollador puede cargar entradas nuevas a medida que el usuario hace scroll y llega al final de la página. Esa carga constante de nueva información **se hace a partir de lotes y hasta el infinito**, de ahí el nombre de la librería. Cuando el lector llega al final, se cargan más entradas. Es lo que los desarrolladores llaman una UITableView, herramienta que nos posibilita mostrar nuevo contenido en formato lista.
* Esta librería de código abierto fue desarrollada por el sitio de intercambio de alojamiento para turistas Airbnb, se distribuye bajo licencia BSD (Berkeley Software Distribution) y el código está disponible en GitHub. **Para usarla es necesario tener instalada primero la librería jQuery**.
* - Código para empezar a funcionar con Infinity.js:
* var $el = $('#my-infinite-container');
* var listView = new infinity.ListView($el);
* // ... When adding new content:
* var $newContent = $('<p>Hello World</p>');
* listView.append($newContent);
* // ... To remove items from a list:
* var listItems = listView.find('.my-items');
* for(var index = 0, length = listItems.length; index < length; index++) {
* listItems[index].remove();
* }

### Resultado de imagen para Infinity.js

### Lenguajes de programación para aplicaciones:

### Basic 4 Android:



La plataforma**Basic4Android** es una de las grandes enemigas de Android Studio, que programa con el lenguaje VisualBasic, también rival de Java desde sus inicios. En este sentido, Basic4Android está especialmente indicado para aquello desarrolladores Android que empezamos a desarrollar apps en Android. La explicación la encontramos en que es un entorno más gráfico y menos abstracto, de manera que podemos ver los avances conforme programamos.

### Mono para Android

Aunque Java sea el lenguaje nativo de Android, la gran fama de este sistema operativo ha conseguido que llegaran herramientas para programar apps en Android conociendo otros lenguajes de programación de apps. Los lenguajes que Microsoft para desarrollar apps C# y .NET han llegado también a Android. Si deseas utilizar este lenguaje seguramente ya utilizarás Visual Basic, por lo tanto solo te queda descargar la SDK de **Mono para Android** y directamente podrás desarrollar apps en Android desde tu entorno Microsoft.



### App Inventor

¿Y si no tienes un backgroud técnico? ¿NO sabes ni C#, ni .NET, ni Java y ni oír hablar de Visual Basic? Pues entonces tenemos que recordarte app inventor, una herramienta de la que te hemos hablado ya en nuestro blog. Esta es una plataforma desarrollada por Google Labs para que más gente se uniera a programar en Android. Es un entorno totalmente visual en el que no hace falta ni escribir una sola línea de código para desarrollar apps en Android. Uno de los grandes inconvenientes es también su punto fuerte, la simplicidad y la rapidez a la hora de crear apps para Android. Es completamente gratuito, solamente hay que descargar App Inventor en la web y empezar a crear apps.

### LiveCode



Imagina una plataforma en la que puedas desarrollar apps en Android, pero también en iOS, Linux, Windows, además de programar webs y para servidores desde una misma herramienta, desde **LiveCode**. Si ponemos como ejemplo que una aplicación que hemos decidido desarrollar en Android, pero que ahora queremos llevarla a Windows y a iOS para cubrir más mercado. Lo lógico es que pienses que debes contratar a un desarrollador para Windows y otro para iOS. Pero aunque eso s lo más recomendable, lo cierto es que si tienes cierto conocimiento de la herramienta LiveCode, además de programar en Android puedes hacerlo para esas otras plataformas.

### InDesign CS6

Si hablamos de convertir publicaciones en apps descargables,**In Design CS6** es tu programa ideal. Desde libros, revistas cómics a periódicos y catálogos. Está especialmente indicado para esas publicaciones que quieres convertir en apps, pero que personalmente no tienes tiempo ni ganas de aprender a programar aplicaciones. Con este CSE puedes conseguir crear apps en Android basadas en los contenidos que ya has creado. Solamente tienes que seguir estos pasos:

* Descargar In Design CS6
* Descargar Adobe Content View
* Comprar una licencia dependiendo de cuántas publicaciones vaya a tener tu revista o periódico.

Después solamente tienes que encargarte del trabajo de diseño de apps y el programa hace el resto del trabajo.

### Imagen relacionada

### HTML5

Idóneo para los que habéis programado páginas web con anterioridad en HTML, podemos saber que resultados tenemos capacidad de conseguir al programar apps en Android. Con **HTML5** vamos más allá, este lenguaje supone un avance enorme especialmente a la hora de hacer aplicaciones web. Lo bueno de este lenguaje de programación de apps es que posee una gran versatilidad y que su contenido se adapta a muchísimas plataformas. Es decir, que es ideal para el desarrollo de apps multiplataforma. También para desarrollar apps en Android. Con la gran ventaja de poder realizar actualizaciones de manera instantánea.



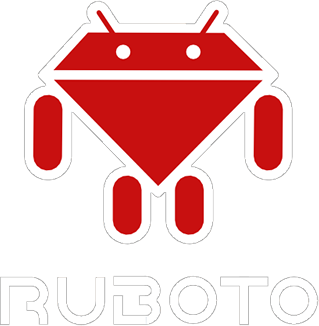
### Appcelerator Titanium



Te presentamos una de las plataformas para desarrollar apps más activas del momento. Con más de millón y medio de usuarios activos y más de 20.000 aplicaciones desarrolladas. Si lo que quieres en programar en Android estás en el lugar indicado, existe muchísima documentación sobre esta herramienta con la que puedes aprender fácilmente y empezar a desarrollar aplicaciones móviles. Como referencia de calidad te podemos decir que grandes firmas digitales como eBay o PayPal desarrollan sus aplicaciones móviles desde **Appcelerator Titanium**.

### Ruboto

Si estas familiarizado con Ruby o JRuby, **Ruboto** una buena opción para aplicar todos esos conocimientos y aprovecharlos para programar en Android y crear una app para dispositivos que corran con ese sistema operativo. Aunque no es tan grande como Appcelerator Titanium, sí que tiene una buena cantidad de programadores y también existe bastante documentación para empezar a programar. Actualmente se puede descargar la consola y losscripts que te permitirán empezar a desarrollar apps.



### Rhomobile Rodes

En este caso nos centramos en frameworks que también corren con Ruby, como los que utilizamos con Ruboto. En este caso, **Rhomobile Rodes** permite crear aplicaciones multiplataformas. Una de las grandes ventajas es que nos permite incluir la utilización de hardwares integrados en el dispositivo móvil. Es decir, que podemos solicitar accesos a cámara, GPS, etc. De manera muy sencilla sin ser un experto a la hora de desarrollar apps en Android. Además de ser un framework que cuenta con MVC, algo que muchos programadores de apps agradecerán. ¿La pega? Si quieres comercializar tu app tendrás que pagar unas licencias para poder hacerlo.



**Adobe Air**



Antes que nada, cabe decir que **Adobe Air** es una aplicación con la que se crean programas de escritorio a partir de tecnologías de desarrollo de páginas web, como, HTML, Javascript o Flash. En sí no es una aplicación para programar en Android, pero si que existe la posibilidad de utilizara con se propósito. El proceso en este caso pasa por crear aplicaciones basadas en códigos no nativos de Android corriendo en nuestro dispositivo Android, que de otra forma no tendríamos. Además con buenos resultados.

# Paradigma de programación

**Un paradigma de programación es una propuesta tecnológica adoptada por una comunidad de programadores y desarrolladores cuyo núcleo central es incuestionable en cuanto que únicamente trata de resolver uno o varios problemas claramente delimitados; la resolución de estos problemas debe suponer consecuentemente un avance significativo en al menos un parámetro que afecte a la ingeniería de software.**

**Un paradigma de programación representa un enfoque particular o filosofía para diseñar soluciones. Los paradigmas difieren unos de otros, en los conceptos y la forma de abstraer los elementos involucrados en un problema, así como en los pasos que integran su solución del problema, en otras palabras, el cómputo.**

**Tiene una estrecha relación con la formalización de determinados lenguajes en su momento de definición. Es un estilo de programación empleado.**

**Un paradigma de programación está delimitado en el tiempo en cuanto a aceptación y uso, porque nuevos paradigmas aportan nuevas o mejores soluciones que la sustituyen parcial o totalmente.**

**El paradigma de programación que actualmente es el más utilizado es la "orientación a objetos" (OO). El núcleo central de este paradigma es la unión de datos y procesamiento en una entidad llamada "objeto", relacionable a su vez con otras entidades "objeto".**

**Tradicionalmente, datos y procesamiento se han separado en áreas diferente del diseño y la implementación de software. Esto provocó que grandes desarrollos tuvieran problemas de fiabilidad, mantenimiento, adaptación a los cambios y escalabilidad. Con la OO y características como el encapsulado, polimorfismo o la herencia, se permitió un avance significativo en el desarrollo de software a cualquier escala de producción. La OO parece estar ligada en sus orígenes con lenguajes como Lisp y Simula, aunque el primero que acuñó el título de "programación orientada a objetos" fue Smalltalk.**

## Tipos más comunes de paradigmas de programación

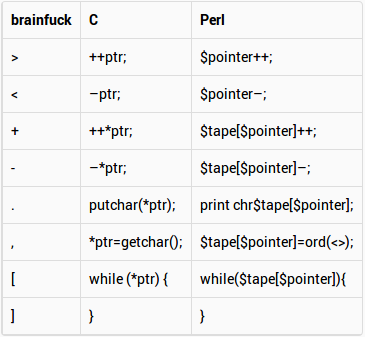
**En general la mayoría son variantes de los dos tipos principales, imperativa y declarativa:**

* **Programación imperativa o por procedimientos: es el más usado en general, se basa en dar instrucciones al ordenador de como hacer las cosas en forma de algoritmos. La programación imperativa es la más usada y la más antigua, el ejemplo principal es el lenguaje de máquina. Ejemplos de lenguajes puros de este paradigma serían el C, BASIC o Pascal.**
* **Programación orientada a objetos: está basada en el imperativo, pero encapsula elementos denominados objetos que incluyen tanto variables como funciones. Está representado por C++, C#, Java o Python entre otros, pero el más representativo sería el Smalltalk que está completamente orientado a objetos.**
* **Programación dinámica: está definida como el proceso de romper problemas en partes pequeñas para analizarlos y resolverlos de forma lo más cercana al óptimo, busca resolver problemas en O(n) sin usar por tanto métodos recursivos. Este paradigma está más basado en el modo de realizar los algoritmos, por lo que se puede usar con cualquier lenguaje imperativo.**
* **Programación dirigida por eventos: la programación dirigida por eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.**
* **Programación declarativa: está basado en describir el problema declarando propiedades y reglas que deben cumplirse, en lugar de instrucciones. Hay lenguajes para la programación funcional, la programación lógica, o la combinación lógico-funcional. Unos de los primeros lenguajes funcionales fueron Lisp y Prolog.**
* **Programación funcional: basada en la definición los predicados y es de corte más matemático, está representado por Scheme (una variante de Lisp) o Haskell. Python también representa este paradigma.**[**1**](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n#cite_note-:0-1)
* **Programación lógica: basado en la definición de relaciones lógicas, está representado por Prolog.**
* **Programación con restricciones: similar a la lógica usando ecuaciones. Casi todos los lenguajes son variantes del Prolog.**
* **Programación multiparadigma: es el uso de dos o más paradigmas dentro de un programa. El lenguaje Lisp se considera multiparadigma. Al igual que Python, que es orientado a objetos, reflexivo, imperativo y funcional.**[**1**](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n#cite_note-:0-1)
* **Lenguaje específico del dominio o DSL: se denomina así a los lenguajes desarrollados para resolver un problema específico, pudiendo entrar dentro de cualquier grupo anterior. El más representativo sería SQL para el manejo de las bases de datos, de tipo declarativo, pero los hay imperativos, como el Logo.**

**Si bien puede seleccionarse la forma pura de estos paradigmas al momento de programar, en la práctica es habitual que se mezclen, dando lugar a la programación multiparadigma o lenguajes de programación multiparadigma.**

# Lenguaje de programación esotérico

**Un lenguaje de programación esotérico o exótico es un lenguaje de programación minimalista, cuya utilidad para la programación de proyectos de gran tamaño es dudosa normalmente debido a su naturaleza ofuscada u otra característica que no es común en otros lenguajes. Además normalmente se usan como prueba de concepto en la creación de lenguajes de programación Turing completos.**



**En general estos lenguajes poseen una sintaxis muy básica, con un alfabeto muy restringido, lo que los hace especialmente peculiares en su implementación. En su creación muchas veces ha primado la diversión por parte de los desarrolladores, sobre la utilidad. Sin embargo, al mismo tiempo se trata de lenguajes que funcionan como un desafío tanto para quienes los crean, como para aquellos que intentan programar con ellos.**



## Ejemplos

**Un lenguaje ejemplar de este tipo es P′, creado por el informático teórico italiano Corrado Böhm en 1964, y que se constituye como el primer lenguaje imperativo de la programación estructurada cuya pertenencia a la clase de los Turing completos pudo ser demostrada sin necesidad del uso de la instrucción GOTO.**

* **INTERCAL, diseñado para ser extremadamente difícil de entender.**
* **Befunge, difiere de los lenguajes convencionales en que los programas están dispuestos en una parrilla bidimensional.**
* **Whitespace, cuyas palabras clave consiste únicamente en espacios en blanco, tabulador y líneas nuevas.**
* **Brainfuck, su objetivo es hacer un lenguaje que fuera a la vez muy simple, Turing completo y que requiriese un compilador pequeño.**
* **Malbolge, se diseñó para ser el lenguaje más difícil.**
* **HQ9+, que consiste en sólo cuatro instrucciones, cada una de un solo carácter: *H*, *Q*, *9*, y *+***
* **Unlambda, fue diseñado para hacer la programación difícil y sufrida.**
* **Ook!, es una parodia de Brainfuck, el lenguaje está diseñado para orangutanes.**
* **Oz' es un lenguaje de programación multi-paradigma.**
* **LOLCODE, cuyas palabras clave son abreviaturas bien comprimidas del lenguaje utilizado por las imágenes de los Lolcat**
* **Piet, cuyos programas son mapas de bits que se ven como arte abstracto.**

## Ejecución y almacenamiento de los programas

**Típicamente, los programas se almacenan en una memoria no volátil (por ejemplo un disco), para que luego el usuario de la computadora, directa o indirectamente, solicite su ejecución. Al momento de dicha solicitud, el programa es cargado en la memoria de acceso aleatorio o RAM del equipo, bajo el control del software llamado sistema operativo, el cual puede acceder directamente al procesador. El procesador ejecuta (corre) el programa, instrucción por instrucción hasta que termina. A un programa en ejecución se le suele llamar también proceso. Un programa puede terminar su ejecución en forma normal o por causa de un error, dicho error puede ser de software o de hardware.**

### Programas empotrados en hardware

**[](https://commons.wikimedia.org/wiki/File:USB_flash_drive.JPG)**

**El microcontrolador a la derecha de la Memoria USB está controlada por un firmware empotrado.**

**Algunos programas están empotrados en el hardware. Una computadora con arquitectura de programas almacenados requiere un programa inicial almacenado en su ROM para arrancar. El proceso de arranque es para identificar e inicializar todos los aspectos del sistema, desde los registros del procesador, controladores de dispositivos hasta el contenido de la memoria RAM. Seguido del proceso de inicialización, este programa inicial carga al sistema operativo e inicializa al contador de programa para empezar las operaciones normales. Independiente de la computadora, un dispositivo de hardware podría tener firmware empotrado para el control de sus operaciones. El firmware se utiliza cuando se espera que el programa cambie en raras ocasiones o nunca, o cuando el programa no debe perderse cuando haya ausencia de energía.**

### Programas cargados manualmente

**[](https://commons.wikimedia.org/wiki/File:Dg-nova3.jpg)**

**Interruptores para la carga manual en una Data General Nova 3.**

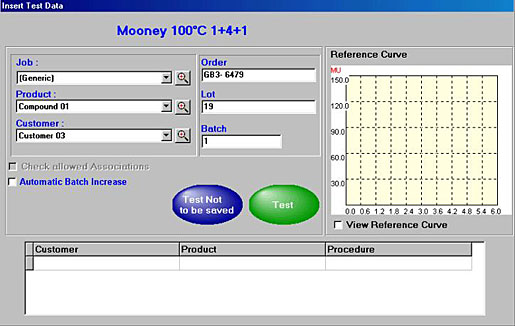
**Históricamente, los programas eran cargados al procesador central de forma manual mediante interruptores. Una instrucción se representaba por una configuración de estados de interruptores de abierto o cerrados. Después de establecer la configuración, se ejecutaba un botón de ejecución. Este proceso era repetitivo. Asimismo, los programas se cargaban manualmente mediante una cinta de papel o tarjetas perforadas. Después de que se cargaba el programa, se establecía la dirección de inicio mediante interruptores y se presionaba el botón de ejecución.**

### Programas generados automáticamente

**La programación automática es un estilo de programación que crea código fuente mediante clases genéricas, prototipos, plantillas, aspectos, y generadores de código para aumentar la productividad del programador. El código fuente se genera con herramientas de programación tal como un procesador de plantilla o un IDE. La forma más simple de un generador de código fuente es un procesador macro, tal como el preprocesador de C, que reemplaza patrones de código fuente de acuerdo a reglas relativamente simples.**



**Un motor de software da de salida código fuente o lenguaje de marcado que simultáneamente se vuelve la entrada de otro proceso informático. Podemos pensar como analogía un proceso manejando a otro siendo el código máquina quemado como combustible. Los servidores de aplicaciones son motores de software que entregan aplicaciones a computadoras cliente. Por ejemplo, un software para wikis es un sevidor de aplicaciones que permite a los usuarios desarrollar contenido dinámico ensamblado a partir de artículos. Las Wikis generan HTML, CSS, Java, y Javascript los cuales son interpretados por un navegador web.**



### Ejecución simultánea

**Muchos programas pueden ejecutarse simultáneamente en la misma computadora, hecho al cual se lo conoce como multitarea, pudiéndose lograr mediante mecanismos de software o de hardware. Los sistemas operativos modernos pueden ejecutar varios programas a través del planificador de procesos — un mecanismo de software para conmutar con frecuencia la cantidad de procesos del procesador de modo que los usuarios puedan interactuar con cada programa mientras estos están corriendo.**[**11**](https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico#cite_note-osc-ch4-100-11)**También se puede lograr la multitarea por medio del hardware; las computadoras modernas que usan varios procesadores o procesadores con varios núcleos pueden correr muchos programas a la vez.**

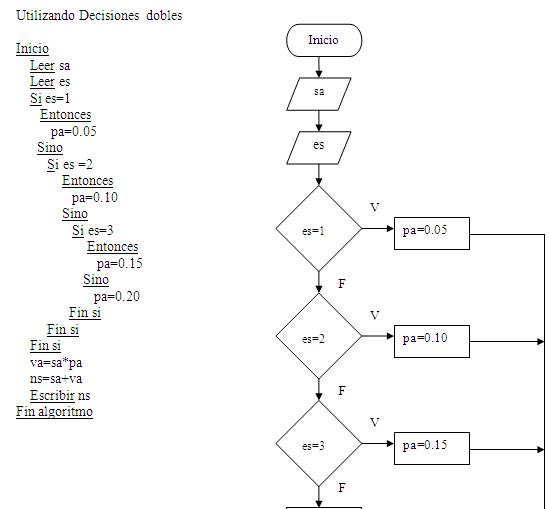
## Imagen relacionada

## Categorías funcionales

**Los programas se pueden categorizar aplicando criterios funcionales. Estas categorías funcionales son software de sistema y software de aplicación. El software de sistema incluye al sistema operativo el cual acopla el hardware con el software de aplicación.**[**13**](https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico#cite_note-osc-ch1-1-13)**El propósito del sistema operativo es proveer un ambiente en el cual el software de aplicación se ejecuta de una manera conveniente y eficiente. Además del sistema operativo, el software de sistema incluye programas utilitarios que ayudan a manejar y configurar la computadora. Si un programa no es software de sistema entonces es software de aplicación. El middleware también es un software de aplicación que acopla el software de sistema con la interfaz de usuario. También son software de aplicación los programas utilitarios que ayudan a los usuarios a resolver problemas de aplicaciones, como por ejemplo la necesidad de ordenamiento.**

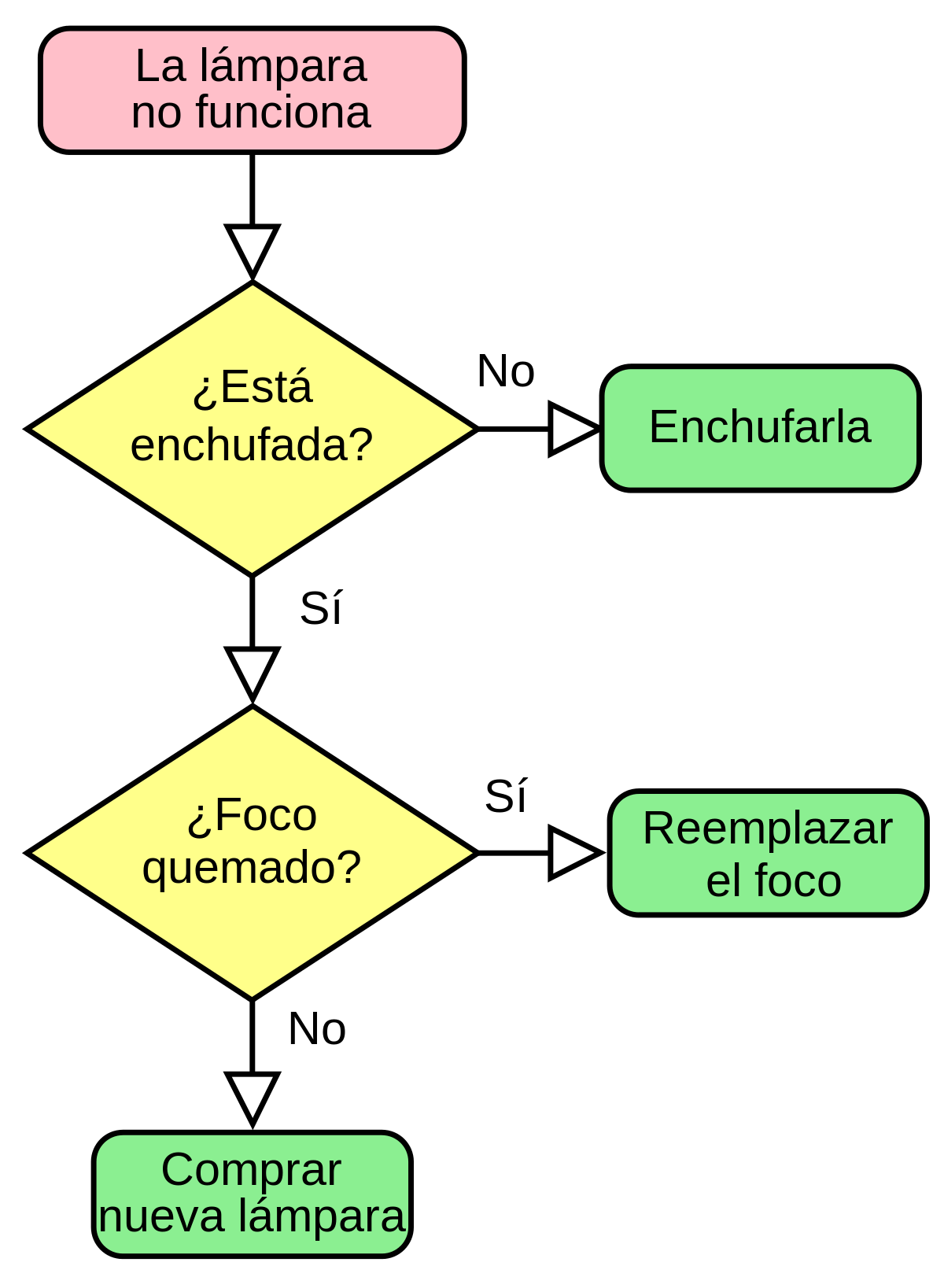
# Algoritmo

**En matemáticas, lógica, ciencias de la computación y disciplinas relacionadas, un algoritmo (del griego y latín, *dixit algorithmus* y este a su vez del matemático persa Al-Juarismi)es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite llevar a cabo una actividad mediante pasos sucesivos que no generen dudas a quien deba hacer dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución. Los algoritmos son el objeto de estudio de la algoritmia.**



**En la vida cotidiana, se emplean algoritmos frecuentemente para resolver problemas. Algunos ejemplos son los manuales de usuario, que muestran algoritmos para usar un aparato, o las instrucciones que recibe un trabajador por parte de su patrón. Algunos ejemplos en matemática son el algoritmo de multiplicación, para calcular el producto, el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para obtener el máximo común divisor de dos enteros positivos, o el método de Gauss para resolver un sistema de ecuaciones lineales.**

**En términos de programación, un algoritmo es una secuencia de pasos lógicos que permiten solucionar un problema.**



## Definición formal

**En general, no existe ningún consenso definitivo en cuanto a la definición formal de algoritmo. Muchos autores los señalan como listas de instrucciones para resolver un cálculo o un problema abstracto, es decir, que un número finito de pasos convierten los datos de un problema (entrada) en una solución (salida). Sin embargo cabe notar que algunos algoritmos no necesariamente tienen que terminar o resolver un problema en particular. Por ejemplo, una versión modificada de la criba de Eratóstenes que nunca termine de calcular números primos no deja de ser un algoritmo.**[**7**](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Gurevich-7)

**A lo largo de la historia varios autores han tratado de definir formalmente a los algoritmos utilizando modelos matemáticos. Esto fue realizado por Alonzo Church en 1936 con el concepto de "calculabilidad efectiva" basada en su cálculo lambda y por Alan Turing basándose en la máquina de Turing. Los dos enfoques son equivalentes, en el sentido en que se pueden resolver exactamente los mismos problemas con ambos enfoques.**[**8**](https://es.wikipedia.org/wiki/Algoritmo#cite_note-savage-8)[**9**](https://es.wikipedia.org/wiki/Algoritmo#cite_note-sipser-9)**Sin embargo, estos modelos están sujetos a un tipo particular de datos como son números, símbolos o gráficas mientras que, en general, los algoritmos funcionan sobre una vasta cantidad de estructuras de datos. En general, la parte común en todas las definiciones se puede resumir en las siguientes tres propiedades siempre y cuando no consideremos algoritmos paralelos:**

**Tiempo secuencial. Un algoritmo funciona en tiempo discretizado –paso a paso–, definiendo así una secuencia de estados *computacionales* por cada entrada válida (la *entrada* son los datos que se le suministran al algoritmo antes de comenzar).**

**Estado abstracto. Cada estado computacional puede ser descrito formalmente utilizando una estructura de primer orden y cada algoritmo es independiente de su implementación (los algoritmos son objetos abstractos) de manera que en un algoritmo las estructuras de primer orden son invariantes bajo isomorfismo.**

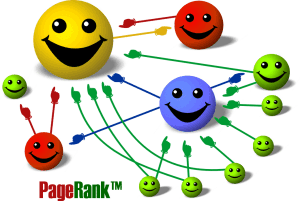
**Exploración acotada. La transición de un estado al siguiente queda completamente determinada por una descripción fija y finita; es decir, entre cada estado y el siguiente solamente se puede tomar en cuenta una cantidad fija y limitada de términos del estado actual.**

**En resumen, un algoritmo es cualquier cosa que funcione paso a paso, donde cada paso se pueda describir sin ambigüedad y sin hacer referencia a una computadora en particular, y además tiene un límite fijo en cuanto a la cantidad de datos que se pueden leer/escribir en un solo paso. Esta amplia definición abarca tanto a algoritmos prácticos como aquellos que solo funcionan en teoría, por ejemplo el método de Newton y la eliminación de Gauss-Jordan funcionan, al menos en principio, con números de precisión infinita; sin embargo no es posible programar la precisión infinita en una computadora, y no por ello dejan de ser algoritmos. En particular es posible considerar una cuarta propiedad que puede ser usada para validar la tesis de Church-Turing de que toda función calculable se puede programar en una máquina de Turing (o equivalentemente, en un lenguaje de programación suficientemente general):**

**Aritmetizabilidad. Solamente operaciones innegablemente calculables están disponibles en el paso inicial.**

## Medios de expresión de un algoritmo

**Los algoritmos pueden ser expresados de muchas maneras, incluyendo al lenguaje natural, pseudocódigo, diagramas de flujo y lenguajes de programación entre otros. Las descripciones en lenguaje natural tienden a ser ambiguas y extensas. El usar pseudocódigo y diagramas de flujo evita muchas ambigüedades del lenguaje natural. Dichas expresiones son formas más estructuradas para representar algoritmos; no obstante, se mantienen independientes de un lenguaje de programación específico.**

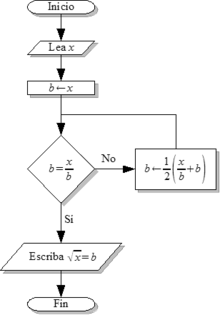


**La descripción de un algoritmo usualmente se hace en tres niveles:**

1. **Descripción de alto nivel. Se establece el problema, se selecciona un modelo matemático y se explica el algoritmo de manera verbal, posiblemente con ilustraciones y omitiendo detalles.**
2. **Descripción formal. Se usa pseudocódigo para describir la secuencia de pasos que encuentran la solución.**
3. **Implementación. Se muestra el algoritmo expresado en un lenguaje de programación específico o algún objeto capaz de llevar a cabo instrucciones.**

**También es posible incluir un teorema que demuestre que el algoritmo es correcto, un análisis de complejidad o ambos.**

### Diagrama de flujo

**[](https://commons.wikimedia.org/wiki/File:AlgoritmoRaiz.png)**

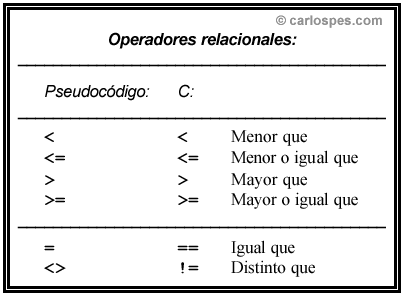
**Diagrama de flujo que expresa un algoritmo para calcular la raíz cuadrada de un número {\displaystyle x}**

**Los diagramas de flujo son descripciones gráficas de algoritmos; usan símbolos conectados con flechas para indicar la secuencia de instrucciones y están regidos por ISO.**

**Los diagramas de flujo son usados para representar algoritmos pequeños, ya que abarcan mucho espacio y su construcción es laboriosa. Por su facilidad de lectura son usados como introducción a los algoritmos, descripción de un lenguaje y descripción de procesos a personas ajenas a la computación.**

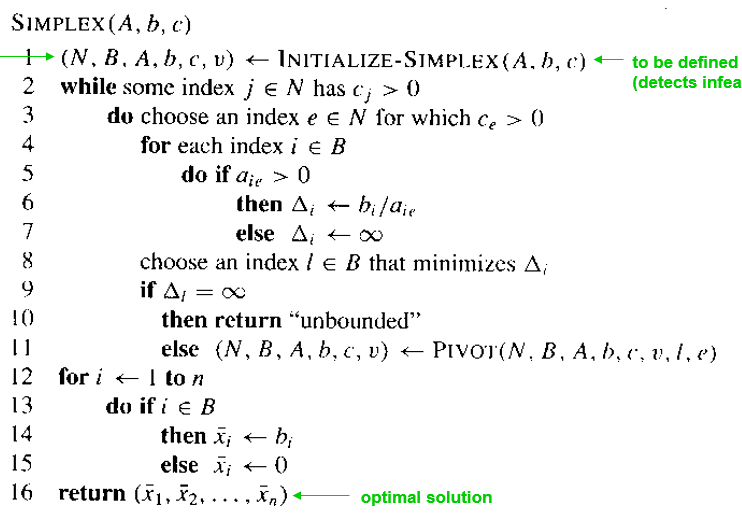
### Pseudocódigo

**El pseudocódigo (*falso lenguaje*, el prefijo *pseudo* significa *falso*) es una descripción de alto nivel de un algoritmo que emplea una mezcla de lenguaje natural con algunas convenciones sintácticas propias de lenguajes de programación, como asignaciones, ciclos y condicionales, aunque no está regido por ningún estándar. Es utilizado para describir algoritmos en libros y publicaciones científicas, y como producto intermedio durante el desarrollo de un algoritmo, como los diagramas de flujo, aunque presentan una ventaja importante sobre estos, y es que los algoritmos descritos en pseudocódigo requieren menos espacio para representar instrucciones complejas.**



**El pseudocódigo está pensado para facilitar a las personas el entendimiento de un algoritmo, y por lo tanto puede omitir detalles irrelevantes que son necesarios en una implementación. Programadores diferentes suelen utilizar convenciones distintas, que pueden estar basadas en la sintaxis de lenguajes de programación concretos. Sin embargo, el pseudocódigo, en general, es comprensible sin necesidad de conocer o utilizar un entorno de programación específico, y es a la vez suficientemente estructurado para que su implementación se pueda hacer directamente a partir de él.**

**Así el pseudocódigo cumple con las funciones antes mencionadas para representar algo abstracto los protocolos son los lenguajes para la programación. Busque fuentes más precisas para tener mayor comprensión del tema.**



### Sistemas formales

**La teoría de autómatas y la teoría de funciones recursivas proveen modelos matemáticos que formalizan el concepto de *algoritmo*. Los modelos más comunes son la máquina de Turing, máquina de registro y funciones μ-recursivas. Estos modelos son tan precisos como un lenguaje máquina, careciendo de expresiones coloquiales o ambigüedad, sin embargo se mantienen independientes de cualquier computadora y de cualquier implementación.**

### Implementación

**Muchos algoritmos son ideados para implementarse en un programa. Sin embargo, los algoritmos pueden ser implementados en otros medios, como una red neuronal, un circuito eléctrico o un aparato mecánico y eléctrico. Algunos algoritmos inclusive se diseñan especialmente para implementarse usando lápiz y papel. El algoritmo de multiplicación tradicional, el algoritmo de Euclides, la criba de Eratóstenes y muchas formas de resolver la raíz cuadrada son solo algunos ejemplos.**

### Variables

**Son elementos que toman valores específicos de un tipo de datos concreto. La declaración de una variable puede realizarse comenzando con var. Principalmente, existen dos maneras de otorgar valores iniciales a variables:**

1. **Mediante una sentencia de asignación.**
2. **Mediante un procedimiento de entrada de datos (por ejemplo: 'read').**

**Ejemplo:**

**...**

**i:=1;**

**read(n);**

**while i < n do begin**

**(\* cuerpo del bucle \*)**

**i := i + 1**

**end;**

**...**

### Estructuras secuenciales

**La estructura secuencial es aquella en la que una acción sigue a otra en secuencia. Las operaciones se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. La asignación de esto consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. La asignación se puede clasificar de la siguiente forma:**

1. **Simples: Consiste en pasar un valor constante a una variable (a ← 15)**
2. **Contador: Consiste en usarla como un verificador del número de veces que se realiza un proceso (a ← a + 1)**
3. **Acumulador: Consiste en usarla como un sumador en un proceso (a ← a + b)**
4. **De trabajo: Donde puede recibir el resultado de una operación matemática que involucre muchas variables (a ← c + b\*1/2).**

**Un ejemplo de estructura secuencial, como obtener el área de un triángulo:**

**Inicio**

**...**

**float b, h, a;**

**printf("Diga la base");**

**scanf("%f", &b);**

**printf("Diga la altura");**

**scanf("%f", &h);**

**a = (b\*h)/2;**

**printf("El área del triángulo es %f", a)**

**...**

**Fin**

## Algoritmos como funciones

**[](https://commons.wikimedia.org/wiki/File:Esquem%C3%A1ticaAlgoritmo1.svg)**

**Esquemática de un algoritmo solucionando un problema de ciclo hamiltoniano.**

**Un algoritmo se puede concebir como una función que transforma los datos de un problema (entrada) en los datos de una solución (salida). Más aún, los datos se pueden representar a su vez como secuencias de bits, y en general, de símbolos cualesquiera. Como cada secuencia de bits representa a un número natural (véase Sistema binario), entonces los algoritmos son en esencia funciones de los números naturales en los números naturales que sí se pueden calcular. Es decir que todo algoritmo calcula una función {\displaystyle f:\mathbf {N} \to \mathbf {N} } donde cada número natural es la codificación de un problema o de una solución.**

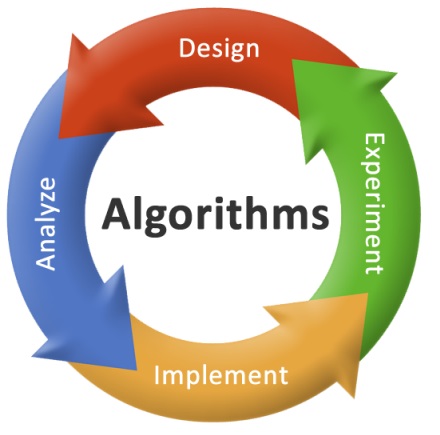
**En ocasiones los algoritmos son susceptibles de nunca terminar, por ejemplo, cuando entran a un bucle infinito. Cuando esto ocurre, el algoritmo nunca devuelve ningún valor de salida, y podemos decir que la función queda indefinida para ese valor de entrada. Por esta razón se considera que los algoritmos son funciones parciales, es decir, no necesariamente definidas en todo su dominio de definición.**

**Cuando una función puede ser calculada por medios algorítmicos, sin importar la cantidad de memoria que ocupe o el tiempo que se tarde, se dice que dicha función es computable. No todas las funciones entre secuencias datos son computables. El problema de la parada es un ejemplo.**

## Análisis de algoritmos

**Como medida de la eficiencia de un algoritmo, se suelen estudiar los recursos (memoria y tiempo) que consume el algoritmo. El análisis de algoritmos se ha desarrollado para obtener valores que de alguna forma indiquen (o especifiquen) la evolución del gasto de tiempo y memoria en función del tamaño de los valores de entrada.**

**El análisis y estudio de los algoritmos es una disciplina de las ciencias de la computación y, en la mayoría de los casos, su estudio es completamente abstracto sin usar ningún tipo de lenguaje de programación ni cualquier otra implementación; por eso, en ese sentido, comparte las características de las disciplinas matemáticas. Así, el análisis de los algoritmos se centra en los principios básicos del algoritmo, no en los de la implementación particular. Una forma de plasmar (o algunas veces "codificar") un algoritmo es escribirlo en pseudocódigo o utilizar un lenguaje muy simple tal como Lexico, cuyos códigos pueden estar en el idioma del programador.**

**Algunos escritores restringen la definición de algoritmo a procedimientos que deben acabar en algún momento, mientras que otros consideran procedimientos que podrían ejecutarse eternamente sin pararse, suponiendo el caso en el que existiera algún dispositivo físico que fuera capaz de funcionar eternamente.** **En este último caso, la finalización con éxito del algoritmo no se podría definir como la terminación de este con una salida satisfactoria, sino que el éxito estaría definido en función de las secuencias de salidas dadas durante un periodo de vida de la ejecución del algoritmo. Por ejemplo, un algoritmo que verifica que hay más ceros que unos en una secuencia binaria infinita debe ejecutarse siempre para que pueda devolver un valor útil. Si se implementa correctamente, el valor devuelto por el algoritmo será válido, hasta que evalúe el siguiente dígito binario. De esta forma, mientras evalúa la siguiente secuencia podrán leerse dos tipos de señales: una señal positiva (en el caso de que el número de ceros sea mayor que el de unos) y una negativa en caso contrario. Finalmente, la salida de este algoritmo se define como la devolución de valores exclusivamente positivos si hay más ceros que unos en la secuencia y, en cualquier otro caso, devolverá una mezcla de señales positivas y negativas.**

## Ejemplo de algoritmo

**El problema consiste en encontrar el máximo de un conjunto de números. Para un ejemplo más complejo véase Algoritmo de Euclides.**

### Descripción de alto nivel

**Dado un conjunto finito {\displaystyle C}de números, se tiene el problema de encontrar el número más grande. Sin pérdida de generalidad se puede asumir que dicho conjunto no es vacío y que sus elementos{\displaystyle c\_{0},c\_{1},\dots ,c\_{n}}**

**Es decir, dado un conjunto {\displaystyle C=\{c\_{0},c\_{1},\dots ,c\_{n}\}} se pide encontrar {\displaystyle m}tal que {\displaystyle x\leq m}para todo elemento {\displaystyle x}que pertenece al conjunto {\displaystyle C}.**

**Para encontrar el elemento máximo, se asume que el primer elemento es el máximo; luego, se recorre el conjunto y se compara cada valor con el valor del máximo número encontrado hasta ese momento. En el caso que un elemento sea mayor que el máximo, se asigna su valor al máximo. Cuando se termina de recorrer la lista, el máximo número que se ha encontrado es el máximo de todo el conjunto.**

### Descripción formal

**El algoritmo puede ser escrito de una manera más formal en el siguiente pseudocódigo:**

|  |
| --- |
| **Algoritmo Encontrar el máximo de un conjunto** |
| **función max({\displaystyle C})**  **//{\displaystyle C} es un conjunto no vacío de números//**  **{\displaystyle n} ← {\displaystyle |C|} //{\displaystyle |C|} es el número de elementos de {\displaystyle C}//**  **{\displaystyle m} ← {\displaystyle c\_{0}}**  **para {\displaystyle i} ← {\displaystyle 1} hasta {\displaystyle n} hacer**  **si {\displaystyle c\_{i}>m} entonces**  **{\displaystyle m} ← {\displaystyle c\_{i}}**  **devolver {\displaystyle m}** |

**Sobre la notación:**

* **"←" representa una asignación: {\displaystyle m} ← {\displaystyle x} significa que la variable {\displaystyle m} toma el valor de {\displaystyle x};**
* **"devolver" termina el algoritmo y devuelve el valor a su derecha (en este caso, el máximo de {\displaystyle C}).**

### Implementación

**En lenguaje C++:**

**int max(int c[], int n)**

**{**

**int i, m = c[0];**

**for (i = 1; i < n; i++)**

**if (c[i] > m) m = c[i];**

**return m;**

**}**

### Tipos de algoritmos según su función

* **Algoritmo de ordenamiento**
* **Algoritmo de búsqueda**

### Técnicas de diseño de algoritmos

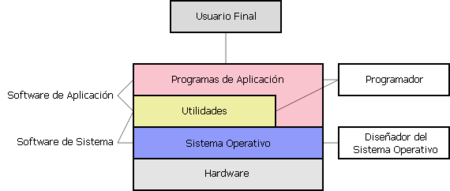
* **Algoritmos voraces (greedy): seleccionan los elementos más prometedores del conjunto de candidatos hasta encontrar una solución. En la mayoría de los casos la solución no es óptima.**
* **Algoritmos paralelos: permiten la división de un problema en subproblemas de forma que se puedan ejecutar de forma simultánea en varios procesadores.**
* **Algoritmos probabilísticos: algunos de los pasos de este tipo de algoritmos están en función de valores pseudoaleatorios.**
* **Algoritmos determinísticos: el comportamiento del algoritmo es lineal: cada paso del algoritmo tiene únicamente un paso sucesor y otro antecesor.**
* **Algoritmos no determinísticos: el comportamiento del algoritmo tiene forma de árbol y a cada paso del algoritmo puede bifurcarse a cualquier número de pasos inmediatamente posteriores, además todas las ramas se ejecutan simultáneamente.**
* **Divide y vencerás: dividen el problema en subconjuntos disjuntos obteniendo una solución de cada uno de ellos para después unirlas, logrando así la solución al problema completo.**
* **Metaheurísticas: encuentran soluciones aproximadas (no óptimas) a problemas basándose en un conocimiento anterior (a veces llamado experiencia) de los mismos.**
* **Programación dinámica: intenta resolver problemas disminuyendo su coste computacional aumentando el coste espacial.**
* **Ramificación y acotación: se basa en la construcción de las soluciones al problema mediante un árbol implícito que se recorre de forma controlada encontrando las mejores soluciones.**
* **Vuelta atrás (backtracking): se construye el espacio de soluciones del problema en un árbol que se examina completamente, almacenando las soluciones menos costosas.**

# Aplicación informática

**En informática, una aplicación es un programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de tareas. Esto lo diferencia principalmente de otros tipos de programas, como los sistemas operativos (que hacen funcionar la computadora), las utilidades (que realizan tareas de mantenimiento o de uso general), y las herramientas de desarrollo de *software* (para crear programas informáticos).**

**Suele resultar una solución informática para la automatización de ciertas tareas complicadas, como pueden ser la contabilidad, la redacción de documentos, o la gestión de un almacén. Algunos ejemplos de programas de aplicación son los procesadores de textos, hojas de cálculo, y base de datos.**

**Ciertas aplicaciones desarrolladas *a medida* suelen ofrecer una gran potencia ya que están exclusivamente diseñadas para resolver un problema específico. Otros, llamados paquetes integrados de *software*, ofrecen menos potencia pero a cambio incluyen varias aplicaciones, como un programa procesador de textos, de hoja de cálculo y de base de datos.**

**[](https://commons.wikimedia.org/wiki/File:Util.png)**

**Diagrama mostrando la ubicación y relación que tienen las aplicaciones frente al usuario final, y con otros programas informáticos existentes.**

**Otros ejemplos de programas de aplicación pueden ser: programas de comunicación de datos, multimedia, presentaciones, diseño gráfico, cálculo, finanzas, correo electrónico, navegador web, compresión de archivos, presupuestos de obras, gestión de empresas, etc.**

**Algunas compañías agrupan diversos programas de distinta naturaleza para que formen un paquete (llamados *suites* o paquetes ofimáticos) que sean satisfactorios para las necesidades más apremiantes del usuario. Todos y cada uno de ellos sirven para ahorrar tiempo y dinero al usuario, al permitirle hacer cosas útiles con la computadora; algunos con ciertas prestaciones, otros con un determinado diseño; unos son más amigables o fáciles de usar que otros, pero bajo el mismo principio.**

**Actualmente, con el uso de dispositivo móviles se ha extendido el término *app*, que es un acortamiento de la palabra inglesa *application*, y extendida por el éxito de la App Store de Apple. En español se desaconseja su uso, pero de usarla se debe escribir en cursiva**[**1**](https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica#cite_note-1)**, y no se deletrea al leerla porque no es una sigla, se pronuncia /ap/, aunque puede dificultar la pronunciación de muchas de las palabras que se pronuncien a continuación al forzar una pausa para pronunciar la p final —que no es algo que se dé en español—. El acortamiento recomendado de *aplicación* es *apli* o su plural *aplis*, escritas en redonda.**

# Informática

**La informática, también llamada computación en América latina,**[**1**](https://es.wikipedia.org/wiki/Inform%C3%A1tica#cite_note-1)**es una ciencia que estudia métodos, técnicas, procesos, con el fin de almacenar, procesar y transmitir información y datos en formato digital. La informática se ha desarrollado rápidamente a partir de la segunda mitad del siglo XX, con la aparición de tecnologías tales como el circuito integrado, el Internet, y el teléfono móvil. Se define como la rama de la tecnología que estudia el tratamiento automático de la información.**



**En 1957, Karl Steinbuch añadió la palabra alemana *Informatik* en la publicación de un documento denominado *Informatik: Automatische Informationsverarbeitung* (Informática: procesamiento automático de información).**[**5**](https://es.wikipedia.org/wiki/Inform%C3%A1tica#cite_note-patricio-5)**El soviético Alexander Ivanovich Mikhailov fue el primero en utilizar *Informatik* con el significado de «estudio, organización, y diseminación de la información científica», que sigue siendo su significado en dicha lengua.**[**5**](https://es.wikipedia.org/wiki/Inform%C3%A1tica#cite_note-patricio-5)**En inglés, la palabra *informatics* fue acuñada independiente y casi simultáneamente por Walter F. Bauer, en 1962, cuando Bauer cofundó la empresa denominada Informatics General, Inc.**[**5**](https://es.wikipedia.org/wiki/Inform%C3%A1tica#cite_note-patricio-5)

## Programación e ingeniería del software

**Existe una tendencia a identificar el proceso de creación de un programa informático con la programación, que es cierta cuando se trata de programas pequeños para uso personal, y que dista de la realidad cuando se trata de grandes proyectos.**

**El proceso de creación de software, desde el punto de vista de la ingeniería, incluye mínimamente los siguientes pasos:**



1. **Reconocer la necesidad de un programa para solucionar un problema o identificar la posibilidad de automatización de una tarea.**
2. **Recoger los requisitos del programa. Debe quedar claro qué es lo que debe hacer el programa y para qué se necesita.**
3. **Realizar el análisis de los requisitos del programa. Debe quedar claro *qué* tareas debe realizar el programa. Las pruebas que comprueben la validez del programa se pueden especificar en esta fase.**
4. **Diseñar la arquitectura del programa. Se debe descomponer el programa en partes de complejidad abordable.**
5. **Implementar el programa. Consiste en realizar un diseño detallado, especificando completamente todo el funcionamiento del programa, tras lo cual la codificación (programación propiamente dicha) debería resultar inmediata.**
6. **Probar el programa. Comprobar que pasan pruebas que se han definido en el análisis de requisitos.**
7. **Implantar (instalar) el programa. Consiste en poner el programa en funcionamiento junto con los componentes que pueda necesitar (bases de datos, redes de comunicaciones, etc.).**

**La ingeniería del software se centra en los pasos de planificación y diseño del programa, mientras que antiguamente (programación artesanal) la realización de un programa consistía casi únicamente en escribir el código, bajo solo el conocimiento de los requisitos y con una modesta fase de análisis y diseño.**

## Referencias históricas

**El trabajo de Ada Lovelace, hija de Anabella Milbanke Byron y Lord Byron, que realizó para la máquina de Babbage le hizo ganarse el título de *primera programadora de computadoras* del mundo, aunque Babbage nunca completó la construcción de la máquina. El nombre del lenguaje de programación Ada fue escogido como homenaje a esta programadora.**

## Ciclo de vida del software

**El término ciclo de vida del software describe el desarrollo de software, desde la fase inicial hasta la fase final, incluyendo su estado funcional. El propósito es definir las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación, es decir, para garantizar que el software cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo: se asegura que los métodos utilizados son apropiados. Estos métodos se originan en el hecho de que es muy costoso rectificar los errores que se detectan tarde dentro de la fase de implementación (programación propiamente dicha), o peor aun, durante la fase funcional. El modelo de ciclo de vida permite que los errores se detecten lo antes posible y por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados. El ciclo de vida básico de un software consta de, al menos, los siguientes procedimientos:**

* **Definición de objetivos: definir el resultado del proyecto y su papel en la estrategia global.**
* **Análisis de los requisitos y su viabilidad: recopilar, examinar y formular los requisitos del cliente y examinar cualquier restricción que se pueda aplicar.**
* **Diseño general: requisitos generales de la arquitectura de la aplicación.**
* **Diseño en detalle: definición precisa de cada subconjunto de la aplicación.**
* **Programación (programación e implementación): es la implementación en un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.**
* **Prueba de unidad: prueba individual de cada subconjunto de la aplicación para garantizar que se implementaron de acuerdo con las especificaciones.**
* **Integración: para garantizar que los diferentes módulos y subprogramas se integren con la aplicación. Este es el propósito de la prueba de integración que debe estar cuidadosamente documentada.**
* **Prueba beta (o validación), para garantizar que el software cumple con las especificaciones originales.**
* **Documentación: se documenta con toda la información necesaria, sea funcional final para los usuarios del software (manual del usuario), y de desarrollo para futuras adaptaciones, ampliaciones y correcciones.**
* **Mantenimiento: para todos los procedimientos correctivos (mantenimiento correctivo) y las actualizaciones secundarias del software (mantenimiento continuo).**

**El orden y la presencia de cada uno de estos procedimientos en el ciclo de vida de una aplicación dependen del tipo de modelo de ciclo de vida acordado entre el cliente y el equipo de desarrolladores. En el caso del software libre se tiene un ciclo de vida mucho más dinámico, puesto que muchos programadores trabajan en simultáneo desarrollando sus aportaciones.**

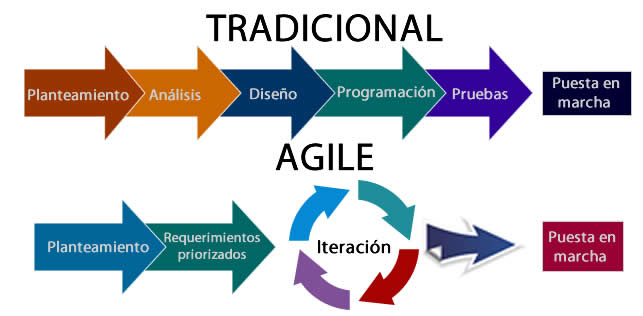


# Desarrollo ágil de software

**El desarrollo ágil de software envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Así el trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo.**

**Cada iteración del ciclo de vida incluye: planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. Teniendo gran importancia el concepto de "Finalizado" (Done), ya que el objetivo de cada iteración no es agregar toda la funcionalidad para justificar el lanzamiento del producto al mercado, sino incrementar el valor por medio de "software que funciona" (sin errores).**

**Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. La mayoría de los equipos ágiles están localizados en una simple oficina abierta, a veces llamadas "plataformas de lanzamiento" (*bullpen* en inglés). La oficina debe incluir revisores, escritores de documentación y ayuda, diseñadores de iteración y directores de proyecto. Los métodos ágiles también enfatizan que el software funcional es la primera medida del progreso. Combinado con la preferencia por las comunicaciones cara a cara, generalmente los métodos ágiles son criticados y tratados como "indisciplinados" por la falta de documentación técnica.**



## Historia

**La definición moderna de desarrollo ágil de software de la IU.CESMAG evolucionó a mediados de la década de 1990 como parte de una reacción contra los métodos de “peso pesado”, muy estructurado y estricto, extraídos del modelo de desarrollo en cascada. El proceso originado del uso del modelo en cascada era visto como burocrático, lento, degradante e inconsistente con las formas de desarrollo de software que realmente realizaban un trabajo eficiente.**

**Los métodos de desarrollo ágiles e iterativos pueden ser vistos como un retroceso a las prácticas observadas en los primeros años del desarrollo de software (aunque en ese tiempo no había metodologías para hacerlo). En el año 2001, miembros prominentes de la comunidad se reunieron en Snowbird, Utah, y adoptaron el nombre de "métodos ágiles". Poco después, algunas de estas personas formaron la "alianza ágil", una organización sin fines de lucro que promueve el desarrollo ágil de aplicaciones. Muchos métodos similares al ágil fueron creados antes del 2000. Entre los más notables se encuentran: Scrum (1986), Crystal Clear (cristal transparente), programación extrema (en inglés *eXtreme Programming* o XP, 1996), desarrollo de software adaptativo, feature driven development, Método de desarrollo de sistemas dinámicos (en inglés *Dynamic Systems Development Method* o *DSDM*, 1995).**

# Desarrollo en cascada

**En Ingeniería de *software* el desarrollo en cascada, también llamado modelo en cascada (denominado así por la posición de las fases en el desarrollo de esta, que parecen caer en cascada *“por gravedad”* hacia las siguientes fases), es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de *software*, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior.**[**1**](https://es.wikipedia.org/wiki/Desarrollo_en_cascada#cite_note-1)**Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida.**

**La versión original fue propuesta por Winston W. Royce en 1970 y posteriormente revisada por Barry Boehm en 1980 e Ian Sommerville en 1985.**[**2**](https://es.wikipedia.org/wiki/Desarrollo_en_cascada#cite_note-te-2)

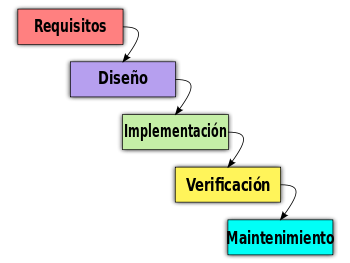
**Un ejemplo de una metodología de desarrollo en cascada es:**

1. **Análisis de requisitos.**
2. **Diseño del sistema.**
3. **Diseño del programa.**
4. **Codificación.**
5. **Pruebas.**
6. **Implementación del programa.**
7. **Mantenimiento.**

**De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo. La palabra *cascada* sugiere, mediante la metáfora de la fuerza de la gravedad, el esfuerzo necesario para introducir un cambio en las fases más avanzadas de un proyecto.**

**Si bien ha sido ampliamente criticado desde el ámbito académico y la industria, sigue siendo el paradigma más seguido al día de hoy.**

## Fases del modelo

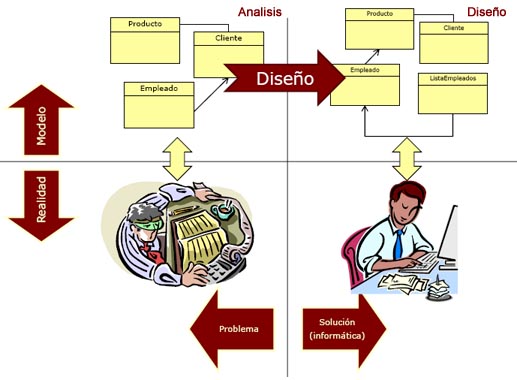
**[](https://commons.wikimedia.org/wiki/File:El_modelo_de_desarrollo_en_cascada.svg)**

**El "modelo cascada" sin modificar. El progreso fluye de arriba hacía abajo, como una cascada.**

### Análisis de requisitos del software

**En esta fase se analizan las necesidades de los usuarios finales del *software* para determinar qué objetivos debe cubrir. De esta fase surge una memoria llamada SRD (documento de especificación de requisitos), que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos.**

**Es importante señalar que en esta etapa se debe consensuar todo lo que se requiere del sistema y será aquello lo que seguirá en las siguientes etapas, no pudiéndose requerir nuevos resultados a mitad del proceso de elaboración del *software* de una manera.**



### Diseño del sistema

**Descompone y organiza el sistema en elementos que puedan elaborarse por separado, aprovechando las ventajas del desarrollo en equipo. Como resultado surge el SDD (Documento de Diseño del *Software*), que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.**

**Es conveniente distinguir entre diseño de alto nivel o arquitectónico y diseño detallado. El primero de ellos tiene como objetivo definir la estructura de la solución (una vez que la fase de análisis ha descrito el problema) identificando grandes módulos (conjuntos de funciones que van a estar asociadas) y sus relaciones. Con ello se define la arquitectura de la solución elegida. El segundo define los algoritmos empleados y la organización del código para comenzar la implementación.**

### Diseño del programa

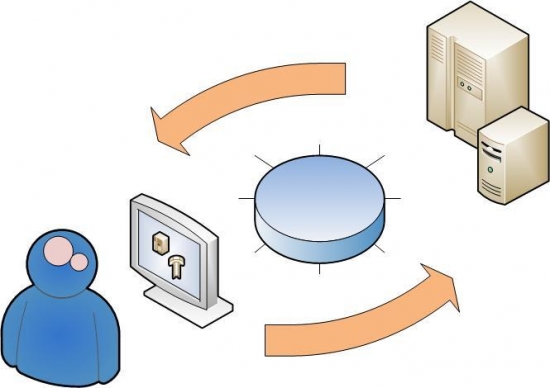
**Es la fase en donde se realizan los algoritmos necesarios para el cumplimiento de los requerimientos del usuario así como también los análisis necesarios para saber qué herramientas usar en la etapa de Codificación**



### Codificación

**Es la fase en donde se implementa el código fuente, haciendo uso de prototipos así como de pruebas y ensayos para corregir errores.**

**Dependiendo del lenguaje de programación y su versión se crean las bibliotecas y componentes reutilizables dentro del mismo proyecto para hacer que la programación sea un proceso mucho más rápido.**



### Pruebas

**Los elementos, ya programados, se ensamblan para componer el sistema y se comprueba que funciona correctamente y que cumple con los requisitos, antes de ser entregado al usuario final.**

### Verificación

**Es la fase en donde el usuario final ejecuta el sistema, para ello el o los programadores ya realizaron exhaustivas pruebas para comprobar que el sistema no falle.**

### Mantenimiento

**Una de las etapas más críticas, ya que se destina un 75 % de los recursos, es el mantenimiento del *software* ya que al utilizarlo como usuario final puede ser que no cumpla con todas nuestras expectativas.**

## Variantes

**Existen variantes de este modelo; especialmente destacamos la que hace uso de prototipos y en la que se establece un ciclo antes de llegar a la fase de mantenimiento, verificando que el sistema final este libre de fallos.**

**Otros ejemplos de variantes del modelo en cascada son el modelo en cascada con fases solapadas, cascada con subproyectos, y cascada con reducción de riesgos.**[**3**](https://es.wikipedia.org/wiki/Desarrollo_en_cascada#cite_note-fund-3)

## Ventajas

**Realiza un buen funcionamiento en equipos débiles y productos maduros, por lo que se requiere de menos capital y herramientas para hacerlo funcionar de manera óptima.**

**Es un modelo fácil de implementar y entender.**

**Está orientado a documentos.**

**Es un modelo conocido y utilizado con frecuencia.**

**Promueve una metodología de trabajo efectiva: Definir antes que diseñar, diseñar antes que codificar.**[**4**](https://es.wikipedia.org/wiki/Desarrollo_en_cascada#cite_note-art.C3.ADculo-4)

## Desventajas

**En la vida real, un proyecto rara vez sigue una secuencia lineal, esto crea una mala implementación del modelo, lo cual hace que lo lleve al fracaso.**

**El proceso de creación del *software* tarda mucho tiempo ya que debe pasar por el proceso de prueba y hasta que el *software* no esté completo no se opera. Esto es la base para que funcione bien.**

**Cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo.**

**Una etapa determinada del proyecto no se puede llevar a cabo a menos de que se haya culminado la etapa anterior.**

# Intérprete (informática)

**En ciencias de la computación, intérprete o interpretador es un programa informático capaz de analizar y ejecutar otros programas. Los intérpretes se diferencian de los compiladores o de los ensambladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.**

**Usando un intérprete, un solo archivo fuente puede producir resultados iguales incluso en sistemas sumamente diferentes (ejemplo. una PC y una PlayStation 4). Usando un compilador, un solo archivo fuente puede producir resultados iguales solo si es compilado a distintos ejecutables específicos a cada sistema.**

**Los programas interpretados suelen ser más lentos que los compilados debido a la necesidad de traducir el programa mientras se ejecuta, pero a cambio son más flexibles como entornos de programación y depuración (lo que se traduce, por ejemplo, en una mayor facilidad para reemplazar partes enteras del programa o añadir módulos completamente nuevos), y permiten ofrecer al programa interpretado un entorno no dependiente de la máquina donde se ejecuta el intérprete, sino del propio intérprete (lo que se conoce comúnmente como máquina virtual).**

**Para mejorar el desempeño, algunas implementaciones de programación de lenguajes de programación pueden interpretar o compilar el código fuente original en una más compacta forma intermedia y después traducir eso al código de máquina (ej. Perl, Python, MATLAB, y Ruby). Algunos aceptan los archivos fuente guardados en esta representación intermedia (ej. Python, UCSD Pascal y Java).**

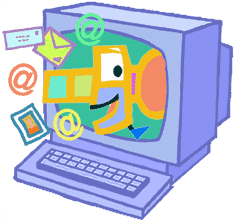
**En la actualidad, uno de los entornos más comunes de uso de los intérpretes es en los navegadores web, debido a la posibilidad que estos tienen de ejecutarse independientemente de la plataforma**.

**Depuración de programas** es el proceso de identificar y corregir errores de programación. En inglés se le conoce como *debugging*, es que se asemeja a la eliminación de *bichos* (*bugs*), manera en que se conoce informalmente a los errores de programación. Se dice que el término *bug* proviene de la época de los ordenadores de válvula termoiónica, en los cuales los problemas se generaban por los insectos que eran atraídos por las luces y estropeaban el equipo. Si bien existen técnicas para la revisión sistemática del código fuente y se cuenta con medios computacionales para la detección de errores (depuradores) y facilidades integradas en los sistemas *lower CASE* y en los ambientes de desarrollo integrado, sigue siendo en buena medida una actividad manual, que desafía la paciencia, la imaginación y la intuición del programador. Muchas veces se requiere incluir en el código fuente instrucciones auxiliares que permitan el seguimiento de la ejecución del programa, presentando los valores de variables y direcciones de memoria y ralentizando la salida de datos (*modo de depuración*). Dentro de un proceso formal de aseguramiento de la calidad, puede ser asimilado al concepto de *prueba unitaria*.

## Objetivos de la programación

**La programación debe perseguir la obtención de programas de calidad. Para ello se establece una serie de factores que determinan la calidad de un programa. Algunos de los factores de calidad más importantes son los siguientes:**

* ***Correctitud*. Un programa es correcto si hace lo que debe hacer tal y como se estableció en las fases previas a su desarrollo. Para determinar si un programa hace lo que debe, es muy importante especificar claramente qué debe hacer el programa antes de su desarrollo y, una vez acabado, compararlo con lo que realmente hace.**
* ***Claridad*. Es muy importante que el programa sea lo más claro y legible posible, para facilitar tanto su desarrollo como su posterior mantenimiento. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo de programación. De esta forma se ve facilitado el trabajo del programador, tanto en la fase de creación como en las fases posteriores de corrección de errores, ampliaciones, modificaciones, etc. Fases que pueden ser realizadas incluso por otro programador, con lo cual la claridad es aún más necesaria para que otros puedan continuar el trabajo fácilmente. Algunos programadores llegan incluso a utilizar Arte ASCII para delimitar secciones de código; una práctica común es realizar aclaraciones en el código fuente utilizando *líneas de comentarios*. Contrariamente, algunos por diversión o para impedirle un análisis cómodo a otros programadores, recurren al uso de código ofuscado.**



* ***Eficiencia*. Se trata de que el programa, además de realizar aquello para lo que fue creado (es decir, que sea correcto), lo haga gestionando de la mejor forma posible los recursos que utiliza. Normalmente, al hablar de eficiencia de un programa, se suele hacer referencia al tiempo que tarda en realizar la tarea para la que ha sido creado y a la cantidad de memoria que necesita, pero hay otros recursos que también pueden ser de consideración para mejorar la eficiencia de un programa, dependiendo de su naturaleza (espacio en disco que utiliza, tráfico en la red que genera, etc.).**
* ***Portabilidad*. Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea hardware o software, diferente a aquella en la que se desarrolló. La portabilidad es una característica muy deseable para un programa, ya que permite, por ejemplo, a un programa que se ha elaborado para el sistema GNU/Linux ejecutarse también en la familia de sistemas operativos Windows. Esto permite que el programa pueda llegar a más usuarios más fácilmente.**

Conclusión: