 LICEO COMPU-MARKET

NOMBRE: Cristian Daniel Boror Santizo

GRADO: 5to.Bachillerato en computación

CATEDRA: Programación

CATEDRATICO: Erick

TEMARIO:

PROGRAMACIÓN

CLAVE: 3

JORNADA: matutina

SECCION: “b”

**INDICE**

1. [**caratula**](#A)**..........................................................................................................1**
2. [**índice**](#b)**……………………………………………………………………………………………………………2**
3. [**Introducción**](#c)**…………………………………………………………………………………………….....3**
4. [**Programación**](#d)**………………………………………………………………………………………………4-5**
5. **Léxico y programación ..……………………………………………………………………………5**
6. [**programación**](#g) **y algoritmos …………………………-.……………………………………………6-8**
7. **Programació y ingeniería ………………………………………………………………………………9-11**
8. [**Lenguaje ensamblador**](#i)**………………………………………………………………………………11-15**
9. [**Estructura de datos y de control**](#j)**…………………………………………………………………15-20**
10. [**Compilación o interpretación de lenguajes de programación**](#k)**……………20-24**
11. [**Lenguaje**](#l)**……………………………………………………………………………………………24-27**
12. [**Programas y algoritmos**](#m)**………………………………………………………………………27**
13. [**Programación declarativa**](#n)**…………………………………………………………………27-32**
14. [**Programación estructurada**](#o)**………………………………………………………………33-36**
15. [**Programación modular**](#p)**………………………………………………………………………36**
16. [**Programación orientada a objetos**](#q)**……………………………………………………37-39**
17. [**Conceptos fundamentales**](#r)**…………………………………………………………………39-41**
18. [**Características de la POO**](#poo)**………………………………………………………………….41-44**
19. [**tipos de lenguajes de programación orientada a objetos**](#s)**……………………45-55**
20. [**Programación e ingeniería del software**](#t)**…………………………………………….56-57**
21. [**Ciclo de vida del software**](#u)**…………………………………………………………………57-58**
22. [**Programa informático**](#v)**………………………………………………………………………….59**
23. [**Paradigmas**](#w)**………………………………………………………………………………………60-63**
24. [**Ejecución y almacenamiento de los programas**](#x)**…………………………………63-65**
25. [**Algoritmos**](#y)**…………………………………………………………………………………………66-72**
26. [**Análisis de algoritmos**](#z)**………………………………………………………………………73-75**
27. [**Lenguaje de máquina**](#maquina)**………………………………………………………………………….76**
28. [**lenguajes de programación más utilizados**](#utilizados)**…………………………………………..77**
29. [**Conclusión**](#conclusion)**…………………………………………………………………………………………..78**
30. [**Egrafía**](#egrafia)**…………………………………………………………………………………………………79**

**INTRODUCCION**

La programación ha servido de base para la creación de nueva tecnología durante décadas atrás desde la invención de la computadora, para utilizarla se debía darle instrucciones a la máquina, atreves de códigos en una sintaxis o logaritmo. Desde ese momento se le denomino programar instrucciones a la máquina, durante el transcurso se fueron implementando nuevos lenguajes de programación creados para cumplir con las necesidades del usuario. Hoy en día la programación ha abarcado diferentes aspectos tecnológicos desde la programación básica hasta la programación avanzada, la programación siempre está presente en los códigos de aplicaciones o programas. La programación se utiliza en todo como: redes sociales, aplicaciones, programas, juegos hasta e incluso las computadoras. Para programar se debe conocer los conceptos básicos, utilizar la lógica humana y la interpretación de los códigos, saber utilizarlos de forma correcta.



Programación

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 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, es decir, traducirlo completo a lenguaje máquina.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.

## **Léxico y programación**

La programación se rige por reglas y un conjunto más o menos reducido de órdenes, expresiones, instrucciones y comandos que tienden a asemejarse a una lengua natural acotada (en inglés); y que además tienen la particularidad de una reducida ambigüedad. Cuanto menos ambiguo es un lenguaje de programación, se dice, es más potente. Bajo esta premisa, y en el extremo, el lenguaje más potente existente es el binario, con ambigüedad nula (lo cual lleva a pensar así del lenguaje ensamblador)

En los lenguajes de programación de alto nivel se distinguen diversos elementos entre los que se incluyen el léxico propio del lenguaje y las reglas semánticas y sintácticas.

## **Programas y algoritmo**

Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema. Un programa normalmente implementa (traduce a un lenguaje de programación concreto) uno o más algoritmos. Un algoritmo puede expresarse de distintas maneras: en forma gráfica, como un diagrama de flujo, en forma de código como en pseudocódigo o un lenguaje de programación, en forma explicativa.

Los programas suelen subdividirse en partes menores, llamadas módulos, de modo que la complejidad algorítmica de cada una de las partes sea menor que la del programa completo, lo cual ayuda al desarrollo del programa. Esta es una práctica muy utilizada y se conoce como "refino progresivo".

Según Niklaus Wirth, un programa está formado por los algoritmos y la estructura de datos.La programación puede seguir muchos enfoques, o paradigmas, es decir, diversas maneras de formular la resolución de un problema dado. Algunos de los principales paradigmas de la programación son:

* Programación declarativa
* Programación estructurada
* Programación modular
* Programación orientada a objetos

El programa escrito en un lenguaje de programación de alto nivel (fácilmente comprensible por el programador) es llamado *programa fuente* y no se puede ejecutar directamente en una computadora. La opción más común es compilar el programa obteniendo un módulo objeto, aunque también puede ejecutarse en forma más directa a través de un intérprete informático.

El código fuente del programa se debe someter a un proceso de traducción para convertirlo a lenguaje máquina o bien a un código intermedio, generando así un módulo denominado "objeto". A este proceso se le llama *compilación*.

Habitualmente la creación de un programa ejecutable (un típico.exe para Microsoft Windows o DOS) conlleva dos pasos. El primer paso se llama compilación (propiamente dicho) y traduce el código fuente escrito en un lenguaje de programación almacenado en un archivo de texto a código en bajo nivel (normalmente en código objeto, no directamente a lenguaje máquina). El segundo paso se llama *enlazado* en el cual se enlaza el código de bajo nivel generado de todos los ficheros y subprogramas que se han mandado compilar y se añade el código de las funciones que hay en las bibliotecas del compilador para que el ejecutable pueda comunicarse directamente con el sistema operativo, traduciendo así finalmente el código objeto a código máquina, y generando un módulo ejecutable.

Estos dos pasos se pueden hacer por separado, almacenando el resultado de la fase de compilación en archivos objetos (un típico .o para Unix, .obj para MS-Windows, DOS); para enlazarlos en fases posteriores, o crear directamente el ejecutable; con lo que la fase de compilación puede almacenarse solo de forma temporal. Un programa podría tener partes escritas en varios lenguajes, por ejemplo, Java, C, C++ y ensamblador, que se podrían compilar de forma independiente y luego enlazar juntas para formar un único módulo ejecutable.

## **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**

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

## **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.

## **Ciclo de vida del software**

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 El 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

Programación es la acción y efecto de programar. Este verbo tiene varios usos: se refiere a idear y ordenar las acciones que se realizarán en el marco de un proyecto; al anuncio de las partes que componen un acto o espectáculo; a la preparación de máquinas para que cumplan con una cierta tarea en un momento determinado; a la elaboración de programas para la resolución de problemas mediante ordenadores; y a la preparación de los datos necesarios para obtener una solución

Los programadores tienen en sus manos el poder de dar vida a un sistema, a una aplicación, a un videojuego. Es importante notar que todos estos **productos** pueden ser definidos en absoluto detalle en un documento, incluyendo imágenes y gráficos que expliquen cómo funciona cada milímetro de los mismos; sin embargo, hasta que un desarrollador de software no entra en acción, no es posible verlos en movimiento, probarlos, pasar de la teoría a la práctica.

En un plano más técnico, la programación se realiza mediante el uso de **algoritmos**, que son secuencias finitas, ordenadas y no ambiguas de instrucciones que deben seguirse para resolver un **problema**. Algunas de ellas pueden agruparse y recibir un nombre para poder ser invocadas con facilidad tantas veces como sea necesario. Del mismo modo que los seres humanos necesitamos respirar constantemente, una aplicación informática necesita conocer la posición del puntero del ratón a cada momento, así como su actividad (si se ha hecho clic y con qué botón, si se ha soltado o si se mantiene presionado, etcétera).

Por último, cabe destacar que se conoce como programación al **conjunto de los programas de televisión o radio**. Por ejemplo: “Tengo ganas de ver una película; veamos qué hay en la programación”.

Programas para programació

Homesite

En el mercado existen muchos programas para editar páginas web, pero HomeSite es uno de los más completos y sencillos de usar.

Está pensado para editar tus páginas programando directamente el HTLM, aunque también tiene la posibilidad de diseñar WYSIWYG (What You See Is What You Get, que significa que lo que ves es lo que obtienes), lo que simplifica y ofrece más velocidad en el diseño de webs simples.   
  
Entre sus características más útiles podemos contar con un editor de estilos CSS, que nos permite definir estilos para toda una página o un sitio web. Posee tratamiento muy bueno de los archivos del sitio, con un panel a la izquierda que nos permite seleccionar cualquier archivo de nuestro sistema para editarlo. El mismo panel de archivos se puede convertir en un inspector de etiquetas, que permite modificar cualquier atributo casi instantáneamente y casi sin conocimientos de HTML, en una ayuda con temas diversos, en un mapa del sitio web, etc.   
  
Posee una barra de herramientas muy completa para colocar desde párrafos o imágenes hasta controles ASP o Cold Fusion, pasando por tablas, formularios, etc. También da soporte a los caracteres especiales del HTML y tiene herramientas como validadores de código, enlaces, etc.   
  
Existe un **tutorial en castellano sobre el Home Site** a la disposición de aquel que desee informarse mejor sobre el programa y aprender su funcionamiento. También tenemos disponible algún tutorial en Inglés: Macromedia Homesite 5 Tutorial.   
  
Somos muchos usuarios satisfechos con este programa: **Oscar**, con dirección de correo oscarab@telepolis.com, nos comenta: En mi opinión el mejor editor de HTML del mercado, no te mete ningún tipo de código basura, tiene una completa ayuda para gente que se está iniciando y gente que lleva programando mucho tiempo.   
  
Nosotros posiblemente no seamos tan entusiastas. El método WYSIWYG sí que mete código basura, aunque lo normal en este programa es realizar nosotros mismos la codificación, con lo cual, si alguien mete código basura, seremos nosotros. En fin, encantados del programa y no llegamos a aprovechar toda su potencia.

# Portal:Programación

En informática la programación es un proceso por el cual se escribe (en un lenguaje de programación), se prueba, se depura y se mantiene el código fuente de un programa informático. Dentro de la informática, los programas son los elementos que forman el software, que es el conjunto de las instrucciones que ejecuta el hardware de una computadora para realizar una tarea determinada. Por lo tanto, la programación es una de las principales áreas dentro de la informática.

Para el desarrollo de programas de cierta envergadura o complejos, con ciertas garantías de calidad, es conveniente seguir alguno de los modelos de desarrollo de software existentes, en donde la programación es sólo una de las etapas del proceso de desarrollo de software. Los modelos de desarrollo de software son tratados específicamente en la disciplina ingeniería del software dentro del campo de la informática.

El código **ASCII** (acrónimo inglés de **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange — *Código Estadounidense Estándar para el Intercambio de Información*), pronunciado generalmente [áski], es un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno y en otras lenguas occidentales. Fue creado en 1963 por el Comité Estadounidense de Estándares (ASA, conocido desde 1969 como el Instituto Estadounidense de Estándares Nacionales, o ANSI) como unarefundición o evolución de los conjuntos de códigos utilizados entonces en telegrafía. Más tarde, en 1967, se incluyeron las minúsculas, y se redefinieron algunos códigos de control para formar el código conocido como **US-ASCII**... Leer más

La primera programadora de la historia fue Ada Lovelace que nació en ¡1815!.

La mayoría de los núcleos de código abierto y sus aplicaciones están programados en C/C++.

La plataforma .NET compila el código tanto de C# VB u F# a un lenguaje de nivel intermedio llamado Common Intermediate Language.

El Código inflado (o inflamado) conocido como Code bloat en Inglés consiste en generar código innecesariamente largo, lento o que hace un mal uso de recursos.

Los hackers son personas aficionadas a la programación y no delincuentes informáticos a diferencia de los crackers

# Informática Programación

Un **programa informático** o **programa de computadora** es una secuencia de instrucciones, escritas para realizar una tarea específica en una computadora.1 Este dispositivo requiere programas para funcionar, por lo general, ejecutando las instrucciones del programa en un procesador central.2 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 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.

La programación de computadoras es el proceso iterativo de escribir o editar código fuente. Dicha edición implica probar, analizar y perfeccionar, y, a veces, coordinar con otros programadores, en el caso de un programa desarrollado en conjunto. Una persona que practica esta técnica se le conoce como programador de computadoras, desarrollador de software, o codificador. El proceso, a veces a largo plazo, de programación de computadoras normalmente se lo conoce como desarrollo de software. El término ingeniería de software se está convirtiendo en muy popular, ya que esta actividad es vista como una disciplina de ingeniería.

### Paradigmas

Los programas de ordenador se pueden clasificar según el paradigma del lenguaje de programación utilizado para producirlos. Dos de los principales paradigmas son imperativos y declarativos.

Los programas escritos con un lenguaje imperativo especifican un algoritmo utilizando declaraciones, expresiones e informes.4 Una declaración asocia un nombre de variable a un tipo de datos. Por ejemplo: var x: integer; . Una expresión produce un valor. Por ejemplo: 2 + 2 produce 4. Por último, una declaración puede asignar una expresión a una variable o usar el valor de una variable para alterar las estructuras de control del programa. Por ejemplo: x := 2 + 2; if x = 4 then hacer\_algo(); Una crítica de los lenguajes imperativos es el efecto secundario de una sentencia de asignación en una clase de variables llamadas variables no locales.5

Los programas escritos en un lenguaje declarativo especifican las propiedades que tienen o que deben cumplirse para la salida. No especifican detalles expresados ​​en términos de flujo de control de la máquina de ejecución pero sí de las relaciones matemáticas entre los objetos declarados y sus propiedades. Los lenguajes funcionales y lógicos son dos amplias categorías de lenguajes declarativos. El principio detrás de los lenguajes funcionales (como Haskell) es el de no permitir efectos secundarios, lo que hace que sea más fácil para razonar sobre los programas como si se tratasen de funciones matemáticas.5 El principio detrás de los lenguajes lógicos (como Prolog) es definir el problema a ser resuelto - la meta - y dejar la solución detallada al propio sistema Prolog.6 El objetivo se define proporcionando la lista de sub-objetivos. Luego, cada subobjetivo se define más arriba, proporcionando la lista de sus sub-objetivos, etc. Si la ruta de sub-objetivos no encuentra una solución, entonces ese subobjetivo se retrocede y otra vía se intenta sistemáticamente.

La forma en que se crea el programa puede ser textual o visual. En un programa de lenguaje visual, los elementos en vez de ser textualmente especificados son manipulados gráficamente.

### Compilado o interpretando

Un *programa de computadora* bajo la forma de lenguaje de programación de computadoras legible por un humano, se lo llama código fuente. Dicho código fuente se puede convertir en una imagen ejecutable por un compilador o ejecutarse inmediatamente con la ayuda de un intérprete.

Cualquiera de los programas compilados o interpretados pueden ser ejecutados en un proceso por lotes sin intervención humana, pero los programas interpretados le permiten al usuario escribir comandos en una sesión interactiva. En este caso, los programas son los comandos separados, cuya ejecución se produce secuencialmente, y por lo tanto simultáneamente. Cuando se utiliza un lenguaje para dar órdenes a una aplicación de software (como un shell de Unix u otra interfaz de línea de comandos), se le llama un lenguaje de scripts.

Los compiladores se utilizan para traducir el código fuente de un lenguaje de programación, ya sea en código objeto o código máquina.7 El código objeto de objeto necesita procesamiento adicional para convertirse en código máquina, y el código máquina es el código nativo de la unidad central de procesamiento, listo para su ejecución. Los programas de computadora compilados se conocen comúnmente como ejecutables, imágenes binarias, o simplemente como binarios — una referencia al formato de archivo binario utilizado para almacenar el código ejecutable.

Los programas de computadora — interpretados en un lote o una sesión interactiva — o bien se descodifican y luego ejecutados inmediatamente o se decodifican en alguna representación intermedia eficiente para la ejecución futura. BASIC, Perl y Python son ejemplos de programas de computadora ejecutados inmediatamente. Por otra parte, los programas de computadora de Java se compilan antes de tiempo y se almacena como un código independiente de la máquina llamado bytecode. Entonces, dicho bytecode es ejecutado a petición de un intérprete llamado máquina virtual.

La principal desventaja de los intérpretes es que los programas de computadora corren más lento que cuando son compilados. La interpretación de código resulta más lenta que la ejecución de la versión compilada porque el intérprete debe decodificar cada declaración cada vez que se carga y luego realizar la acción deseada. Sin embargo, el desarrollo de software puede ser más rápido usando un intérprete porque la prueba es inmediata cuando se omite el paso de la compilación. Otra desventaja de los intérpretes es que debe estar presente al menos uno en la computadora durante la ejecución del programa de computadora. Por el contrario, los programas de computadora compilados no necesitan compilador presente durante la ejecución.

No se requieren propiedades de un lenguaje de programación si se está compilado exclusivamente o interpretándose exclusivamente. Por lo general, la clasificación refleja el método más popular de ejecución del lenguaje. Por ejemplo, BASIC se considera un lenguaje interpretado y C un lenguaje compilado, a pesar de la existencia de compiladores de BASIC e intérpretes de C. Algunos sistemas utilizan compilación en tiempo de ejecución (JIT) mediante la cual las secciones de la fuente se compilan 'sobre la marcha' y se almacenan para ejecuciones posteriores.

### Programas que se auto-modifican

Un programa informático en ejecución normalmente es tratado como algo diferente de los datos con los cuales opera. Sin embargo, en algunos casos ésta distinción es ambigua, especialmente cuando un programa se modifica a sí mismo. El programa modificado es ejecutado secuencialmente como parte del mismo programa. En el caso de programas escritos en código máquina, lenguaje ensamblador, Lisp, C, COBOL, PL/1 y Prolog y JavaScript (la función eval), entre otros, es posible tener código que se auto-modifica.

## 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.8 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.9

### 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.10

### 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

Véanse también: *Proceso (informática)* y *Multiprocesamiento*.

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 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.12

## 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 El propósito del sistema operativo es proveer un ambiente en el cual el software de aplicación se ejecuta de una manera conveniente y eficiente.13 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)1 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.2 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**.1

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.

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).1 2 3 4 5 6 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

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 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.3 1 En general, la parte común en todas las definiciones se puede resumir en las siguientes tres propiedades siempre y cuando no consideremos algoritmos paralelos:7

**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.10 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):10

**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

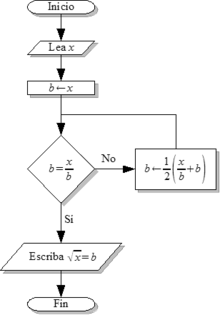
[](https://commons.wikimedia.org/wiki/File:AlgoritmoRaiz.png)

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

Artículo principal: *Diagrama de flujo*

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

Artículo principal: *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

Artículo principal: *Teoría de la computabilidad*

[](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.1 9 11 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 f : N → N {\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

Artículo principal: *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.

Dado un conjunto finito C {\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 están numerados como c 0 , c 1 , … , c n {\displaystyle c\_{0},c\_{1},\dots ,c\_{n}} .

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

Para encontrar el elemento máximo, se asume que el primer elemento ( c 0 {\displaystyle c\_{0}} ) 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.

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

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

Sobre la notación:

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

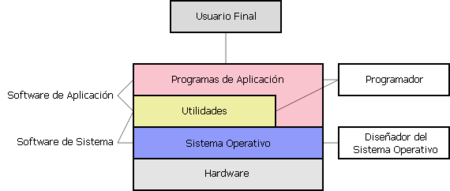
[](https://commons.wikimedia.org/wiki/File:Util.png)

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 cursiva1 , 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.

# Virus informático

Un **virus** es un *software* que tiene por objetivo alterar el funcionamiento normal del ordenador, sin el permiso o el conocimiento del usuario. Los virus, habitualmente, reemplazan archivos ejecutables por otros infectados con el código de este. Los virus pueden destruir, de manera intencionada, los datos almacenados en una computadora, aunque también existen otros más inofensivos, que solo producen molestias.

Los virus informáticos tienen, básicamente, la función de propagarse a través de un *software*, son muy nocivos y algunos contienen además una carga dañina (*payload*) con distintos objetivos, desde una simple broma hasta realizar daños importantes en los sistemas, o bloquear las redes informáticas generando tráfico inútil de información.

El funcionamiento de un virus informático es conceptualmente simple. Se ejecuta un programa que está infectado, en la mayoría de las ocasiones, por desconocimiento del usuario. El código del virus queda residente (alojado) en la memoria RAM de la computadora, incluso cuando el programa que lo contenía haya terminado de ejecutar. El virus toma entonces el control de los servicios básicos del sistema operativo, infectando, de manera posterior, archivos ejecutables que sean llamados para su ejecución. Finalmente se añade el código del virus al programa infectado y se graba en el disco, con lo cual el proceso de replicado se completa.

El primer virus atacó a una máquina IBM Serie 360 (y reconocido como tal). Fue llamado Creeper, (ENMS) creado en 1972. Este programa emitía periódicamente en la pantalla el mensaje: «I'm a creeper... catch me if you can!» («¡Soy una enredadera... agárrame si puedes!»). Para eliminar este problema se creó el primer programa antivirus denominado *Reaper* (cortadora).

Sin embargo, el término virus no se adoptaría hasta 1984, pero éstos ya existían desde antes. Victor Vyssotsky, Robert Morris Sr. y Doug McIlroy, investigadores de Bell Labs (se cita erróneamente a Dennis Ritchie o Ken Thompson como cuarto coautor) desarrollaron un juego de ordenador llamado *Darwin* (del que derivará *Core Wars*) que consiste en eliminar al programa adversario ocupando toda la RAM de la zona de juego (*arena*).1

Después de 1984, los virus han tenido una gran expansión, desde los que atacan los sectores de arranque de disquetes hasta los que se adjuntan en un correo electrónico.

## Virus informáticos y sistemas operativos

os virus informáticos afectan en mayor o menor medida a casi todos los sistemas más conocidos y usados en la actualidad.

Cabe aclarar que un virus informático mayoritariamente atacará sólo el sistema operativo para el que fue desarrollado, aunque ha habido algunos casos de virus multiplataforma.

MS-Windows y Android

Las mayores incidencias se dan en el sistema operativo Windows y Android debido, entre otras causas:

Su gran popularidad, como sistemas operativos, entre los computadores personales y dispositivos móviles. Se estima que, en 2007, un 90 % de ellos usaba Windows.[cita requerida] Mientras que Android tiene una cuota de mercado de 80 % en 2015. Esta popularidad basada en la facilidad de uso sin conocimiento previo alguno, motiva a los creadores de software malicioso a desarrollar nuevos virus; y así, al atacar sus puntos débiles, aumentar el impacto que generan.

Falta de seguridad en Windows plataforma (situación a la que Microsoft está dando en los últimos años mayor prioridad e importancia que en el pasado). Al ser un sistema tradicionalmente muy permisivo con la instalación de programas ajenos a éste, sin requerir ninguna autentificación por parte del usuario o pedirle algún permiso especial para ello en los sistemas más antiguos. A partir de la inclusión del Control de Cuentas de Usuario en Windows Vista y en adelante (y siempre y cuando no se desactive) se ha solucionado este problema, ya que se puede usar la configuración clásica de Linux de tener un usuario administrador protegido, pero a diario usar un Usuario estándar sin permisos se ve desprotegido ante una amenaza de virus.

Software como Internet Explorer y Outlook Express, desarrollados por Microsoft e incluidos de forma predeterminada en las versiones anteriores de Windows, son conocidos por ser vulnerables a los virus ya que éstos aprovechan la ventaja de que dichos programas están fuertemente integrados en el sistema operativo dando acceso completo, y prácticamente sin restricciones, a los archivos del sistema. Un ejemplo famoso de este tipo es el virus ILOVEYOU, creado en el año 2000 y propagado a través de Outlook. Hoy en día Internet Explorer ha sido separado de Windows y Outlook Express fue descontinuado.

La escasa formación de un número importante de usuarios de estos sistemas, lo que provoca que no se tomen medidas preventivas por parte de estos, ya que estos sistemas están dirigidos de manera mayoritaria a los usuarios no expertos en informática. Esta situación es aprovechada constantemente por los programadores de virus.

Unix y derivados

Commons-emblem-issue.svg

En este artículo sobre informática se detectaron varios problemas, por favor, edítalo para mejorarlo:

No tiene una redacción neutral.

Por favor, discute este problema en la discusión.

Carece de fuentes o referencias que aparezcan en una fuente acreditada.

Puedes avisar al redactor principal pegando lo siguiente en su página de discusión:

{{subst:Aviso PA|Virus informático|noneutral|referencias}} ~~~~

En otros sistemas operativos como las distribuciones GNU/Linux, BSD, Solaris, Mac OS X iOS y otros basados en Unix las incidencias y ataques son raros. Esto se debe principalmente a:

Los usuarios de este tipo de Sistemas Operativos suelen poseer conocimientos mucho mayores a los de los usuarios comunes de sistemas o cuenten con recursos para contratar mantenimiento y protección mayores que en Windows.

Tradicionalmente los programadores y usuarios de sistemas basados en Unix han considerado la seguridad como una prioridad por lo que hay mayores medidas frente a virus, tales como la necesidad de autenticación por parte del usuario como administrador o root para poder instalar cualquier programa adicional al sistema. En Windows esta prestación existe desde Windows Vista.

Los directorios o carpetas que contienen los archivos vitales del sistema operativo cuentan con permisos especiales de acceso, por lo que no cualquier usuario o programa puede acceder fácilmente a ellos para modificarlos o borrarlos. Existe una jerarquía de permisos y accesos para los usuarios.

Relacionado al punto anterior, a diferencia de los usuarios de Windows XP y versiones anteriores de Windows, la mayoría de los usuarios de sistemas basados en Unix no pueden normalmente iniciar sesiones como usuarios "administradores' o por el superusuario root, excepto para instalar o configurar software, dando como resultado que, incluso si un usuario no administrador ejecuta un virus o algún software malicioso, éste no dañaría completamente el sistema operativo ya que Unix limita el entorno de ejecución a un espacio o directorio reservado llamado comúnmente home. Aunque a partir de Windows Vista, se pueden configurar las cuentas de usuario de forma similar.

Estos sistemas, a diferencia de Windows, son usados para tareas más complejas como servidores que por lo general están fuertemente protegidos, razón que los hace menos atractivos para un desarrollo de virus o software malicioso.

En el caso particular de las distribuciones basadas en GNU/Linux y gracias al modelo colaborativo, las licencias libres y debido a que son más populares que otros sistemas Unix, la comunidad aporta constantemente y en un lapso de tiempo muy corto actualizaciones que resuelven bugs y/o agujeros de seguridad que pudieran ser aprovechados por algún malware.

Otros sistemas operativos

La mayoría de equipos con sistema operativo de disco de la década de 1990 (equipos de 8, 16 y 32 bits) han sufrido de las diferentes variantes de virus, principalmente de sector de arranque y de ficheros infectados.2 La única excepción parecen haber sido las versiones de CP/M, CP/M-86 y DOS Plus, pero no así su descendiente DR-DOS. En los directorios de BBS y la incipiente Internet, siempre está presente un apartado de antivirus. Sin embargo las versiones más actualizadas de estos sistemas operativos solo lo contemplan como algo histórico, al no haber desarrollos específicos para el OS (lo que no elimina, por ej., los ataques a través de navegado web). Esta pujanza se basa sobre todo en videojuegos que necesitan tener el disquete desprotegido de escritura para almacenar puntuaciones o estados del juego, o en determinadas protecciones. Varios están situados en ROM, por lo que no es posible infectar al sistema en sí, pero al necesitar cargar parte desde el disquete, no se realiza comprobación.

Commodore Amiga / Amiga OS: Son bastante numerosos, hasta el punto de que lo primero que haces cuando recibes un disco de terceros es escanearlo por si acaso. Se conocen al menos 548 virus.3

Atari ST / Atari TOS: Tiene el primer caso de virus de plataforma cruzada: los virus Aladinn y Frankie se escriben para el emulador de Apple Macintosh Aladinn.4 Su compatibilidad con el formato de disco de MS-DOS provoca que se den casos de discos ST infectados por virus de sector de DOS (sin efecto para el equipo), por lo que sus antivirus los contemplan, para dar protección a los emuladores de PC por soft y hard de la plataforma.

Acorn Archimedes / RISC OS: Menos conocidos por estar casi restringido al mercado británico, existen al menos 10 antivirus : VProtect, VZap, KillVirus, Hunter, Interferon, IVSearch, Killer, Scanner, VirusKill, VKiller5

MS-DOS / DR-DOS: el paraíso del virus en aquellos tiempos, con algunos de los primeros en su clase. De los proveedores de antivirus de entonces sobreviven hoy McAfee y Symantec, el resto entraron en el mercado con Microsoft Windows

Commodore 64: BHP VIRUS, Bula

Apple II: ostenta uno de los primeros virus el Elk Cloner de 1982

Apple Macintosh / Mac OS Classic : las versiones para procesadores 680x0 y PowerPC son infectados por virus específicos (la emulación del 680x0 en los PowerPC los hace vulnerables a algunos de los viejos virus, pero no a todos) como por virus de macro para MS Office. Los cambios de plataforma actúan de barrera para, por ej., los virus de sector de arranque. La aparición de Mac OS X marca un punto y aparte en los virus para Mac; aunque no supone su desaparición, los reduce notablemente.

Características

Dado que una característica de los virus es el consumo de recursos, los virus ocasionan problemas tales como: pérdida de productividad, cortes en los sistemas de información o daños a nivel de datos.

Una de las características es la posibilidad que tienen de diseminarse por medio de réplicas y copias. Las redes en la actualidad ayudan a dicha propagación cuando éstas no tienen la seguridad adecuada.

Otros daños que los virus producen a los sistemas informáticos son la pérdida de información, horas de parada productiva, tiempo de reinstalación, etc.

Hay que tener en cuenta que cada virus plantea una situación diferente.

Métodos de propagación

Existen dos grandes clases de contagio. En la primera, el usuario, en un momento dado, ejecuta o acepta de forma inadvertida la instalación del virus. En la segunda, el programa malicioso actúa replicándose a través de las redes. En este caso se habla de gusanos.

En cualquiera de los dos casos, el sistema operativo infectado comienza a sufrir una serie de comportamientos anómalos o imprevistos. Dichos comportamientos pueden dar una pista del problema y permitir la recuperación del mismo.

Dentro de las contaminaciones más frecuentes por interacción del usuario están las siguientes:

Mensajes que ejecutan automáticamente programas (como el programa de correo que abre directamente un archivo adjunto).

Ingeniería social, mensajes como ejecute este programa y gane un premio, o, más comúnmente: Haz 2 clics y gana 2 tonos para móvil gratis.

Entrada de información en discos de otros usuarios infectados.

Instalación de software modificado o de dudosa procedencia.

En el sistema Windows puede darse el caso de que la computadora pueda infectarse sin ningún tipo de intervención del usuario (versiones Windows 2000, XP y Server 2003) por virus como Blaster, Sasser y sus variantes por el simple hecho de estar la máquina conectada a una red o a Internet. Este tipo de virus aprovechan una vulnerabilidad de desbordamiento de buffer y puertos de red para infiltrarse y contagiar el equipo, causar inestabilidad en el sistema, mostrar mensajes de error, reenviarse a otras máquinas mediante la red local o Internet y hasta reiniciar el sistema, entre otros daños. En las últimas versiones de Windows 2000, XP y Server 2003 se ha corregido este problema en su mayoría.

Métodos de protección

Los métodos para disminuir o reducir los riesgos asociados a los virus pueden ser los denominados activos o pasivos.

Activos

Antivirus: son programas que tratan de descubrir las trazas que ha dejado un software malicioso, para detectarlo y eliminarlo, y en algunos casos contener o parar la contaminación. Tratan de tener controlado el sistema mientras funciona parando las vías conocidas de infección y notificando al usuario de posibles incidencias de seguridad. Por ejemplo, al verse que se crea un archivo llamado Win32.EXE.vbs en la carpeta C:\Windows\%System32%\ en segundo plano, ve que es comportamiento sospechoso, salta y avisa al usuario.

Filtros de ficheros: consiste en generar filtros de ficheros dañinos si el computador está conectado a una red. Estos filtros pueden usarse, por ejemplo, en el sistema de correos o usando técnicas de firewall. En general, este sistema proporciona una seguridad donde no se requiere la intervención del usuario, puede ser muy eficaz, y permitir emplear únicamente recursos de forma más selectiva.

Actualización automática: Consiste en descargar e instalar las actualizaciones que el fabricante del sistema operativo lanza para corregir fallos de seguridad y mejorar el desempeño. Dependiendo de la configuración el proceso puede ser completamente automático o dejar que el usuario decida cuándo instalar las actualizaciones.

Pasivos

Para no infectar un dispositivo, hay que:

No instalar software de dudosa procedencia.

No abrir correos electrónicos de desconocidos ni adjuntos que no se reconozcan.

Usar un bloqueador de elementos emergentes en el navegador.

Usar la configuración de privacidad del navegador.

Activar el Control de cuentas de usuario.

Borrar la memoria caché de Internet y el historial del navegador.

No abrir documentos sin asegurarnos del tipo de archivo. Puede ser un ejecutable o incorporar macros en su interior.

Tipos de virus

Existen diversos tipos de virus, varían según su función o la manera en que este se ejecuta en nuestra computadora alterando la actividad de la misma, entre los más comunes están:

Recicler: Consiste en crear un acceso directo de un programa y eliminar su aplicación original, además al infectar un pendrive convierte a toda la información en acceso directo y oculta el original de modo que los archivos no puedan ser vistos, pero con la creación de un archivo "batch" que modifique los atributos de los archivos contenidos en el pendrive, estos podrían ser recuperados.

Troyano: Consiste en robar información o alterar el sistema del hardware o en un caso extremo permite que un usuario externo pueda controlar el equipo.

Bombas lógicas o de tiempo: Son programas que se activan al producirse un acontecimiento determinado. La condición suele ser una fecha (Bombas de Tiempo), una combinación de teclas, o ciertas condiciones técnicas (Bombas Lógicas). Si no se produce la condición permanece oculto al usuario.

Gusano: Tiene la propiedad de duplicarse asi mismo.

Hoax: Los hoax no son virus ni tienen capacidad de reproducirse por sí solos. Son mensajes de contenido falso que incitan al usuario a hacer copias y enviarla a sus contactos. Suelen apelar a los sentimientos morales ("Ayuda a un niño enfermo de cáncer") o al espíritu de solidaridad ("Aviso de un nuevo virus peligrosísimo") y, en cualquier caso, tratan de aprovecharse de la falta de experiencia de los internautas novatos.

Joke: Al igual que los hoax, no son virus, pero son molestos, un ejemplo: una página pornográfica que se mueve de un lado a otro, y si se le llega a dar a cerrar es posible que salga una ventana que diga error.

Otros tipos por distintas características son los que se relacionan a continuación:

Virus residentes:

La característica principal de estos virus es que se ocultan en la memoria RAM de forma permanente o residente. De este modo, pueden controlar e interceptar todas las operaciones llevadas a cabo por el sistema operativo, infectando todos aquellos ficheros y/o programas que sean ejecutados, abiertos, cerrados, renombrados, copiados. Algunos ejemplos de este tipo de virus son: Randex, CMJ, Meve, MrKlunky.

Virus de acción directa:

Al contrario que los residentes, estos virus no permanecen en memoria. Por tanto, su objetivo prioritario es reproducirse y actuar en el mismo momento de ser ejecutados. Al cumplirse una determinada condición, se activan y buscan los ficheros ubicados dentro de su mismo directorio para contagiarlos.

Virus de sobreescritura:

Estos virus se caracterizan por destruir la información contenida en los ficheros que infectan. Cuando infectan un fichero, escriben dentro de su contenido, haciendo que queden total o parcialmente inservibles.

Virus de boot (bot\_kill) o de arranque:

Los términos boot o sector de arranque hacen referencia a una sección muy importante de un disco o unidad de almacenamiento CD, DVD, memorias USB, etc. En ella se guarda la información esencial sobre las características del disco y se encuentra un programa que permite arrancar el ordenador. Este tipo de virus no infecta ficheros, sino los discos que los contienen. Actúan infectando en primer lugar el sector de arranque de los dispositivos de almacenamiento. Cuando un ordenador se pone en marcha con un dispositivo de almacenamiento, el virus de boot infectará a su vez el disco duro.

Los virus de boot no pueden afectar al ordenador mientras no se intente poner en marcha a este último con un disco infectado. Por tanto, el mejor modo de defenderse contra ellos es proteger los dispositivos de almacenamiento contra escritura y no arrancar nunca el ordenador con uno de estos dispositivos desconocido en el ordenador.

Algunos ejemplos de este tipo de virus son: Polyboot.B, AntiEXE.

Virus de enlace o directorio:

Los ficheros se ubican en determinadas direcciones (compuestas básicamente por unidad de disco y directorio), que el sistema operativo conoce para poder localizarlos y trabajar con ellos.

Los virus de enlace o directorio alteran las direcciones que indican donde se almacenan los ficheros. De este modo, al intentar ejecutar un programa (fichero con extensión EXE o COM) infectado por un virus de enlace, lo que se hace en realidad es ejecutar el virus, ya que éste habrá modificado la dirección donde se encontraba originalmente el programa, colocándose en su lugar.

Una vez producida la infección, resulta imposible localizar y trabajar con los ficheros originales.

Virus cifrados:

Más que un tipo de virus, se trata de una técnica utilizada por algunos de ellos, que a su vez pueden pertenecer a otras clasificaciones. Estos virus se cifran a sí mismos para no ser detectados por los programas antivirus. Para realizar sus actividades, el virus se descifra a sí mismo y, cuando ha finalizado, se vuelve a cifrar.

Virus polimórficos:

Son virus que en cada infección que realizan se cifran de una forma distinta (utilizando diferentes algoritmos y claves de cifrado). De esta forma, generan una elevada cantidad de copias de sí mismos e impiden que los antivirus los localicen a través de la búsqueda de cadenas o firmas, por lo que suelen ser los virus más costosos de detectar.

Virus multipartitos

Virus muy avanzados, que pueden realizar múltiples infecciones, combinando diferentes técnicas para ello. Su objetivo es cualquier elemento que pueda ser infectado: archivos, programas, macros, discos, etc.

Virus del fichero

Infectan programas o ficheros ejecutables (ficheros con extensiones EXE y COM). Al ejecutarse el programa infectado, el virus se activa, produciendo diferentes efectos.

Virus de FAT:

La tabla de asignación de ficheros o FAT (del inglés File Allocation Table) es la sección de un disco utilizada para enlazar la información contenida en éste. Se trata de un elemento fundamental en el sistema. Los virus que atacan a este elemento son especialmente peligrosos, ya que impedirán el acceso a ciertas partes del disco, donde se almacenan los ficheros críticos para el normal funcionamiento del ordenador.

Virus hijackers:

Son programas que secuestran navegadores de internet principalmente el explorer. Los hijackers alteran las páginas iniciales del navegador e impide que el usuario pueda cambiarla, muestra publicidad en pops ups. Instala nuevas herramientas en la barra del navegador y a veces impiden al usuario acceder a ciertas páginas web. Un ejemplo puede ser no poder acceder a una página de antivirus.

Virus Zombie:

Son programas que secuestran computadoras de forma que es controlada por terceros. Se utiliza para diseminar virus, keyloggers y procedimientos invasivos en general. Esto puede ocurrir cuando la computadora tiene el firewall y su sistema operativo desactualizado.

Virus Keylogger:

Este virus se encarga de registrar cada tecla que sea pulsada, en algunos casos también registran los clics. Son virus que quedan escondidos en el sistema operativo de manera que la víctima no tiene como saber que está siendo monitorizada. Los keyloggers se utilizan usualmente para robar contraseñas de cuentas bancarias, obtener contraseñas personales como las del E-mail, Facebook, etc.

Acciones de los virus

Algunas de las acciones de algunos virus son:

Unirse a cualquier programa permitiendo su propagación y siendo más costoso liberarse de él.

Ralentizar el dispositivo.

Reduciendo el espacio en el disco.

Mostrando ventanas de forma constante.

Corrompiendo archivos del dispositivo, en algunos casos archivos vitales para el funcionamiento del dispositivo.

Descargando archivos o programas basura

Apagando o reiniciando su dispositivo

Haciendo llamadas a tus contactos u otros números con mayor costo

Eliminar todos los datos guardados en el disco duro

Orígenes teóricos: hasta 1985

El primer trabajo académico en la teoría de los programas de ordenador auto-replicantes6 fue publicado por John von Neumann en 1949 quien dio conferencias en la Universidad de Illinois sobre la Teoría y Organización de Autómatas Complicados (Theory and Organization of Complicated Automata). El trabajo de von Neumann fue publicado más tarde como la Teoría de los autómatas autorreproductivos. En su ensayo von Neumann describió cómo un programa de ordenador puede ser diseñado para reproducirse a sí mismo.7 El diseño de Von Neumann de un programa informático capaz de copiarse a sí mismo se considera el primer virus de computadoras del mundo, y es considerado como el padre teórico de la virología informática.8

En 1960 Victor Vyssotsky, Robert Morris Sr. y Doug McIlroy, investigadores de Bell Labs, implementaron un juego de ordenador llamado Darwin en un mainframe IBM 7090.1 En él, dos programas jugadores compiten en la arena por controlar el sistema, eliminando a su enemigo, intentado sobreescribir o inutilizar todas sus copias. Una versión mejorada del mismo se conocerá como Core Wars. Muchos de los conceptos de este se basan en un artículo de Alexander Dewdney en la columna Computer Recreations de la revista Scientific American.

En 1972 Veith Risak publica el artículo "Selbstreproduzierende Automaten mit minimaler Informationsübertragung" (autómata auto reproducible con mínimo intercambio de información).9 El artículo describe un virus por escrito con fines de investigación. Este contenía todos los componentes esenciales. Fue programado en Lenguaje ensamblador para el equipo SIEMENS 4004/35 y corrió sin problemas.

En 1975 el autor Inglés John Brunner publica la novela El jinete de la onda de shock, en la que anticipa el riesgo de virus de Internet. Thomas Joseph Ryan describió 1979 en The Adolescence of P-1 (la adolescencia de P-1), como una Inteligencia Artificial se propaga de forma similar a un virus en la red informática nacional.

En 1980, Jürgen Kraus escribió una tesis en la Universidad técnica de Dortmund, en la que compara a algunos programas con los virus biológicos.10

En 1982 Rich Skrenta, un estudiante de instituto de 15 años, programa el Elk Cloner para los Apple II, el primer virus informático conocido que tuvo una expansión real y no como un concepto de laboratorio. Puede ser descrito como el primer virus de sector de arranque.11

En 1984 el Profesor Leonard M. Adleman utilizó en una conversación con Fred Cohen por primera vez el término "virus informático".

# Estructura de datos

En ciencias de la computación, una **estructura de datos** es una forma particular de organizar datos en una computadora para que pueda ser utilizado de manera eficiente.

Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.

Las estructuras de datos son un medio para manejar grandes cantidades de datos de manera eficiente para usos tales como grandes bases de datos y servicios de indización de Internet. Por lo general, las estructuras de datos eficientes son clave para diseñar algoritmos eficientes. Algunos métodos formales de diseño y lenguajes de programación destacan las estructuras de datos, en lugar de los algoritmos, como el factor clave de organización en el diseño de software.

**Ejemplos**

Existen numerosos tipos de estructuras de datos, generalmente construidas sobre otras más simples:

* Un vector es una serie de elementos en un orden específico, por lo general todos del mismo tipo (si bien los elementos pueden ser de casi cualquier tipo). Se accede a los elementos utilizando un entero como índice para especificar el elemento que se requiere. Las implementaciones típicas asignan palabras de memoria contiguas a los elementos de los arreglos (aunque no siempre es el caso). Los arreglos pueden cambiar de tamaño o tener una longitud fija.
* Un vector asociativo (también llamado *diccionario* o *mapa* ) es una variante más flexible que una matriz, en la que se puede añadir y eliminar libremente pares nombre-valor. Una tabla de hash es una implementación usual de un arreglo asociativo.
* Un registro (también llamado *tupla* o *estructura*) es una estructura de datos agregados. Un registro es un valor que contiene otros valores, típicamente en un número fijo y la secuencia y por lo general un índice por nombres. Los elementos de los registros generalmente son llamados *campos*.
* Una unión es una estructura de datos que especifica cuál de una serie de tipos de datos permitidos podrá ser almacenada en sus instancias, por ejemplo *flotante* o *entero largo*. En contraste con un registro, que se podría definir para contener un *flotante* y un *entero largo*, en una unión, sólo hay un valor a la vez. Se asigna suficiente espacio para contener el tipo de datos de cualquiera de los miembros.
* Un tipo variante (también llamado *registro variante* o *unión discriminada*) contiene un campo adicional que indica su tipo actual.
* Un conjunto es un tipo de datos abstracto que puede almacenar valores específicos, sin orden particular y sin valores duplicados.
* Un Multiconjunto es un tipo de datos abstracto que puede almacenar valores específicos, sin orden particular. A diferencia de los conjuntos, los multicunjuntos admiten repeticiones.
* Un grafo es una estructura de datos conectada compuesta por nodos. Cada nodo contiene un valor y una o más referencias a otros nodos. Los grafos pueden utilizarse para representar redes, dado que los nodos pueden referenciarse entre ellos. Las conexiones entre nodos pueden tener dirección, es decir un nodo de partida y uno de llegada.
* Un árbol es un caso particular de grafo dirigido en el que no se admiten ciclos y existe un camino desde un nodo llamado raíz hasta cada uno de los otros nodos. Una colección de árboles es llamada un bosque.
* Una clase es una plantilla para la creación de objetos de datos según un modelo predefinido. Las clases se utilizan como representación abstracta de conceptos, incluyen campos como los registros y operaciones que pueden consultar el valor de los campos o cambiar sus valores.

Soporte en los lenguajes

La mayoría de los lenguajes ensambladores y algunos lenguajes de bajo nivel, tales como BCPL, carecen de soporte de estructuras de datos. En cambio, muchos lenguajes de alto nivel y algunos lenguajes ensambladores de alto nivel, tales como MASM, tienen algún tipo de soporte incorporado para ciertas estructuras de datos, tales como los registros y arreglos. Por ejemplo, los lenguajes C y Pascal soportan estructuras y registros, respectivamente, además de arreglos y matrices multidimensionales.1 2 La mayoría de los lenguajes de programación disponen de algún tipo de biblioteca o mecanismo que el uso de estructuras de en los programas. Los lenguajes modernos por lo general vienen con bibliotecas estándar que implementan las estructuras de datos más comunes. Ejemplos de ello son la biblioteca Standard Template Library de C++, las colecciones de Java3 y las librerías .NET de Microsoft.

Estructuras de datos en programación

En programación, una estructura de datos puede ser declarada inicialmente escribiendo una palabra reservada, luego un identificador para la estructura y un nombre para cada uno de sus miembros, sin olvidar los tipos de datos que estos representan. Generalmente, cada miembro se separa con algún tipo de operador, carácter o palabra reservada.

En el lenguaje de programación Pauscal, es posible crear una estructura de datos de la forma mencionada. La sintaxis básica es:

Estruct Identificador, \_

Miembro1:TipoDeDato, \_

Miembro2:TipoDeDato, \_

...

Miembro9:TipoDeDato

Para acceder a los miembros de una estructura, primero se debe crear una referencia a esta, generalmente con una variable de tipo; luego se pueden editar y obtener los datos de los miembros libremente.

Estruc Estructura,Miembro1:Entero,Miembro2:Cadena,Miembro3:Byte

Var Variable:Estructura

Variable.Miembro1 = 40000

Variable.Miembro2 = "Hola Mundo"

Variable.Miembro3 = 255

Mensaje(Variable.Miembro2) ' Muestra "Hola Mundo"

# Lenguaje de programación

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.1

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.

Para que la computadora entienda nuestras instrucciones debe usarse un lenguaje específico conocido como código máquina, el cual la máquina comprende fácilmente, pero que lo hace excesivamente complicado para las personas. De hecho sólo consiste en cadenas extensas de números 0 y 1.

Para facilitar el trabajo, los primeros operadores de computadoras decidieron hacer un traductor para reemplazar los 0 y 1 por palabras o abstracción de palabras y letras provenientes del inglés; éste se conoce como lenguaje ensamblador. Por ejemplo, para sumar se usa la letra A de la palabra inglesa *add* (sumar). El lenguaje ensamblador sigue la misma estructura del lenguaje máquina, pero las letras y palabras son más fáciles de recordar y entender que los números.

La necesidad de recordar secuencias de programación para las acciones usuales llevó a denominarlas con nombres fáciles de memorizar y asociar: ADD (sumar), SUB (restar), MUL (multiplicar), CALL (ejecutar subrutina), etc. A esta secuencia de posiciones se le denominó "instrucciones", y a este conjunto de instrucciones se le llamó lenguaje ensamblador. Posteriormente aparecieron diferentes lenguajes de programación, los cuales reciben su denominación porque tienen una estructura sintáctica semejante a la de los lenguajes escritos por los humanos, denominados también lenguajes de alto nivel.

El primer programador de computadora que se haya conocido fue una mujer: Ada Lovelace, hija de Anabella Milbanke Byron y Lord Byron. Anabella inició en las matemáticas a Ada quien, después de conocer a Charles Babbage, tradujo y amplió una descripción de su máquina analítica. Incluso aunque Babbage nunca completó la construcción de cualquiera de sus máquinas, el trabajo que Ada realizó con éstas le hizo ganarse el título de primera programadora de computadoras del mundo. El nombre del lenguaje de programación Ada fue escogido como homenaje a esta programadora.

A finales de 1953, John Backus sometió una propuesta a sus superiores en IBM para desarrollar una alternativa más práctica al lenguaje ensamblador para programar la computadora central IBM 704. El histórico equipo Fortran de Backus consistió en los programadores Richard Goldberg, Sheldon F. Best, Harlan Herrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Lois Haibt y David Sayre.2

El primer manual para el lenguaje Fortran apareció en octubre de 1956, con el primer compilador Fortran entregado en abril de 1957. Esto era un compilador optimizado, porque los clientes eran reacios a usar un lenguaje de alto nivel a menos que su compilador pudiera generar código cuyo desempeño fuera comparable al de un código hecho a mano en lenguaje ensamblador.

En 1960, se creó COBOL, uno de los lenguajes usados aún en la actualidad, en informática de gestión.

A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método más eficiente para programarlas. Entonces, se crearon los lenguajes de alto nivel, como lo fue BASIC en las versiones introducidas en los microordenadores de la década de 1980. Mientras que una tarea tan sencilla como sumar dos números puede necesitar varias instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará una sola sentencia.

### Variables y vectores

Las variables son títulos asignados a espacios en memoria para almacenar datos específicos. Son contenedores de datos y por ello se diferencian según el tipo de dato que son capaces de almacenar. En la mayoría de lenguajes de programación se requiere especificar un tipo de variable concreto para guardar un dato específico. Por ejemplo, en Java, si deseamos guardar una cadena de texto debemos especificar que la variable es del tipo *String*. Por otra parte, en lenguajes como PHP este tipo de especificación de variables no es necesario. Además, existen variables compuestas llamadas vectores. Un vector no es más que un conjunto de bytes consecutivas en memoria y del mismo tipo guardadas dentro de una variable contenedor. A continuación, un listado con los tipos de variables y vectores más comunes:

|  |  |
| --- | --- |
| **Tipo de dato** | **Breve descripción** |
| Char | Estas variables contienen un único carácter, es decir, una letra, un signo o un número. |
| Int | Contienen un número entero. |
| Float | Contienen un número decimal. |
| String | Contienen cadenas de texto, o lo que es lo mismo, es un vector con varias variables del tipo Char. |
| Boolean | Solo pueden contener un cero o un uno. |

En el caso de variables booleanas, el cero es considerado para muchos lenguajes como el literal falso ("**False**"), mientras que el uno se considera verdadero ("**True**").

### Condicionales

### Las sentencias condicionales son estructuras de código que indican que, para que cierta parte del programa se ejecute, deben cumplirse ciertas premisas; por ejemplo: que dos valores sean iguales, que un valor exista, que un valor sea mayor que otro... Estos condicionantes por lo general solo se ejecutan una vez a lo largo del programa. Los condicionantes más conocidos y empleados en programación son:

* **If**: Indica una condición para que se ejecute una parte del programa.
* **Else if**: Siempre va precedido de un "If" e indica una condición para que se ejecute una parte del programa siempre que no cumpla la condición del if previo y si se cumpla con la que el "else if" especifique.
* **Else**: Siempre precedido de "If" y en ocasiones de "Else If". Indica que debe ejecutarse cuando no se cumplan las condiciones previas.

### Bucles

Los bucles son parientes cercanos de los condicionantes, pero ejecutan constantemente un código mientras se cumpla una determinada condición. Los más frecuentes son:

* **For**: Ejecuta un código mientras una variable se encuentre entre 2 determinados parámetros.
* **While**: Ejecuta un código mientras que se cumpla la condición que solicita.

Hay que decir que a pesar de que existan distintos tipos de bucles, ambos son capaces de realizar exactamente las mismas funciones. El empleo de uno u otro depende, por lo general, del gusto del programador.

### Funciones

Las funciones se crearon para evitar tener que repetir constantemente fragmentos de código. Una función podría considerarse como una variable que encierra código dentro de si. Por lo tanto cuando accedemos a dicha variable (la función) en realidad lo que estamos haciendo es ordenar al programa que ejecute un determinado código predefinido anteriormente.

Todos los lenguajes de programación tienen algunos elementos de formación primitivos para la descripción de los datos y de los procesos o transformaciones aplicadas a estos datos (tal como la suma de dos números o la selección de un elemento que forma parte de una colección). Estos elementos primitivos son definidos por reglas sintácticas y semánticas que describen su estructura y significado respectivamente.

### 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 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

Artículo principal: *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*.

#### Lenguajes tipados versus lenguajes no tipados

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/2/2a/Translation_arrow.svg/45px-Translation_arrow.svg.png | **El texto que sigue es una traducción defectuosa.** Si quieres colaborar con Wikipedia, busca el artículo original y mejora esta traducción.  Copia y pega el siguiente código en la página de discusión del autor: {{subst:Aviso mal traducido|Lenguaje de programación}} ~~~~ |

Se dice que un lenguaje es tipado si la especificación de cada operación debe definir los tipos de datos para los cuales es aplicable, con la implicación de que no es aplicable a otros tipos. Por ejemplo, "este texto entre comillas" es una cadena de caracteres. En la mayoría de los lenguajes de programación, dividir un número por una cadena de caracteres no tiene ningún significado. Por tanto, la mayoría de los lenguajes de programación modernos rechazarían cualquier intento de ejecutar dicha operación por parte de algún programa. En algunos lenguajes, estas operaciones sin significado son detectadas cuando el programa es compilado (validación de tipos "estática") y son rechazadas por el compilador, mientras en otros son detectadas cuando el programa es ejecutado (validación de tipos "dinámica") y se genera una excepción en tiempo de ejecución.

Un caso especial de lenguajes de tipo son los lenguajes de *tipo sencillo*. Estos son con frecuencia lenguajes de marcado o de scripts, como REXX o SGML, y solamente cuentan con un tipo de datos; comúnmente cadenas de caracteres que luego son usadas tanto para datos numéricos como simbólicos.

En contraste, un lenguaje *sin tipos*, como la mayoría de los lenguajes ensambladores, permiten que cualquier operación se aplique a cualquier dato, que por lo general se consideran secuencias de bits de varias longitudes. Lenguajes de alto nivel *sin datos* incluyen BCPL y algunas variedades de Forth.

En la práctica, aunque pocos lenguajes son considerados con tipo desde el punto de vista de la teoría de tipos (es decir, que verifican o rechazan *todas* las operaciones), la mayoría de los lenguajes modernos ofrecen algún grado de manejo de tipos. Si bien muchos lenguajes de producción proveen medios para evitar o rodear el sistema de tipado.

#### Tipos estáticos versus tipos dinámicos

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/2/2a/Translation_arrow.svg/45px-Translation_arrow.svg.png | **El texto que sigue es una traducción defectuosa.** Si quieres colaborar con Wikipedia, busca el artículo original y mejora esta traducción.  Copia y pega el siguiente código en la página de discusión del autor: {{subst:Aviso mal traducido|Lenguaje de programación}} ~~~~ |

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

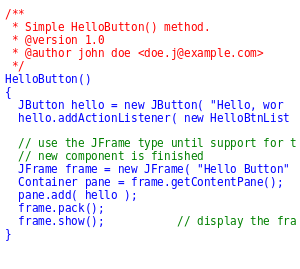
Los lenguajes *débilmente tipados* permiten que un valor de un tipo pueda ser tratado como de otro tipo, por ejemplo una cadena puede ser operada como un número. Esto puede ser útil a veces, pero también puede permitir ciertos tipos de fallas que no pueden ser detectadas durante la compilación o a veces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

[](https://commons.wikimedia.org/wiki/File:CodeCmmt002.svg)

Código fuente de un programa escrito en el lenguaje de programación Java.

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

[](https://commons.wikimedia.org/wiki/File:Programming_language_textbooks.jpg)

Libros sobre diversos lenguajes de programación.

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 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

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.4 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.5 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 dinámica

En informática, la **programación dinámica** es un método para reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas, como se describe a continuación.

El matemático Richard Bellman inventó la **programación dinámica** en 1953 que se utiliza para optimizar problemas complejos que pueden ser discretizados y secuencializados.

Una subestructura óptima significa que se pueden usar soluciones óptimas de subproblemas para encontrar la solución óptima del problema en su conjunto. Por ejemplo, el camino más corto entre dos vértices de un grafo se puede encontrar calculando primero el camino más corto al objetivo desde todos los vértices adyacentes al de partida, y después usando estas soluciones para elegir el mejor camino de todos ellos. En general, se pueden resolver problemas con subestructuras óptimas siguiendo estos tres pasos:

Dividir el problema en subproblemas más pequeños.

Resolver estos problemas de manera óptima usando este proceso de tres pasos recursivamente.

Usar estas soluciones óptimas para construir una solución óptima al problema original.

Los subproblemas se resuelven a su vez dividiéndolos en subproblemas más pequeños hasta que se alcance el caso fácil, donde la solución al problema es trivial.

Decir que un problema tiene subproblemas superpuestos es decir que se usa un mismo subproblema para resolver diferentes problemas mayores. Por ejemplo, en la sucesión de Fibonacci (F3 = F1 + F2 y F4 = F2 + F3) calcular cada término supone calcular F2. Como para calcular F5 hacen falta tanto F3 como F4, una mala implementación para calcular F5 acabará calculando F2 dos o más veces. Esto sucede siempre que haya subproblemas superpuestos: una mala implementación puede acabar desperdiciando tiempo recalculando las soluciones óptimas a problemas que ya han sido resueltos anteriormente.

Esto se puede evitar guardando las soluciones que ya hemos calculado. Entonces, si necesitamos resolver el mismo problema más tarde, podemos obtener la solución de la lista de soluciones calculadas y reutilizarla. Este acercamiento al problema se llama memoización (no confundir con memorización; en inglés es llamado memoization, véase en). Si estamos seguros de que no volveremos a necesitar una solución en concreto, la podemos descartar para ahorrar espacio. En algunos casos, podemos calcular las soluciones a problemas que de antemano sabemos que vamos a necesitar.

En resumen, la programación hace uso de:

Subproblemas superpuestos

Subestructuras óptimas

Memoización

La programación toma normalmente uno de los dos siguientes enfoques:

Top-down: El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursión.

Bottom-up: Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado.

Originalmente, el término de programación dinámica se refería a la resolución de ciertos problemas y operaciones fuera del ámbito de la Ingeniería Informática, al igual que hacía la programación lineal. Aquel contexto no tiene relación con la programación en absoluto; el nombre es una coincidencia. El término también lo usó en los años 40 Richard Bellman, un matemático norteamericano, para describir el proceso de resolución de problemas donde hace falta calcular la mejor solución consecutivamente.

Algunos lenguajes de programación funcionales, sobre todo Haskell, pueden usar la memorización automáticamente sobre funciones con un conjunto concreto de argumentos, para acelerar su proceso de evaluación. Esto sólo es posible en funciones que no tengan efectos secundarios, algo que ocurre en Haskell pero no tanto en otros lenguajes.

Principio de optimalidad

Cuando hablamos de optimizar nos referimos a buscar alguna de las mejores soluciones de entre muchas alternativas posibles. Dicho proceso de optimización puede ser visto como una secuencia de decisiones que nos proporcionan la solución correcta. Si, dada una subsecuencia de decisiones, siempre se conoce cuál es la decisión que debe tomarse a continuación para obtener la secuencia óptima, el problema es elemental y se resuelve trivialmente tomando una decisión detrás de otra, lo que se conoce como estrategia voraz. En otros casos, aunque no sea posible aplicar la estrategia voraz, se cumple el principio de optimalidad de Bellman que dicta que «dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima». En este caso sigue siendo posible el ir tomando decisiones elementales, en la confianza de que la combinación de ellas seguirá siendo óptima, pero será entonces necesario explorar muchas secuencias de decisiones para dar con la correcta, siendo aquí donde interviene la programación dinámica.

Contemplar un problema como una secuencia de decisiones equivale a dividirlo en problemas más pequeños y por lo tanto más fáciles de resolver como hacemos en Divide y Vencerás, técnica similar a la de programación dinámica. La programación dinámica se aplica cuando la subdivisión de un problema conduce a:

Una enorme cantidad de problemas.

Problemas cuyas soluciones parciales se solapan.

Grupos de problemas de muy distinta complejidad.

Ejemplos

Sucesión de Fibonacci

Esta sucesión puede expresarse mediante la siguiente recurrencia:

F i b ( n ) = { 0 si n = 0 1 si n = 1 F i b ( n − 1 ) + F i b ( n − 2 ) si n > 1 {\displaystyle Fib(n)={\begin{cases}0&{\textrm {si}}\;n=0\\1&{\textrm {si}}\;n=1\\Fib(n-1)+Fib(n-2)&{\textrm {si}}\;n>1\\\end{cases}}} {\displaystyle Fib(n)={\begin{cases}0&{\textrm {si}}\;n=0\\1&{\textrm {si}}\;n=1\\Fib(n-1)+Fib(n-2)&{\textrm {si}}\;n>1\\\end{cases}}}

Una implementación de una función que encuentre el n-ésimo término de la sucesión de Fibonacci basada directamente en la definición matemática de la sucesión realizando llamadas recursivas hace mucho trabajo redundante, obteniéndose una complejidad exponencial:

FUNC fib(↓n: NATURAL): NATURAL

INICIO

SI n = 0 ENTONCES

DEVOLVER 0

SI NO, SI n = 1 ENTONCES

DEVOLVER 1

SI NO

devolver fib(n-1) + fib(n-2)

FIN SI

FIN

Si llamamos, por ejemplo, a fib(5), produciremos un árbol de llamadas que contendrá funciones con los mismos parámetros varias veces:

fib(5)

fib(4) + fib(3)

(fib(3) + fib(2)) + (fib(2) + fib(1))

((fib(2) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))

(((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))

En particular, fib(2) se ha calculado tres veces desde cero. En ejemplos mayores, se recalculan muchos otros valores de fib, o subproblemas.

Para evitar este inconveniente, podemos resolver el problema mediante programación dinámica, y en particular, utilizando el enfoque de memorización (guardar los valores que ya han sido calculados para utilizarlos posteriormente). Así, rellenaríamos una tabla con los resultados de los distintos subproblemas, para reutilizarlos cuando haga falta en lugar de volver a calcularlos. La tabla resultante sería una tabla unidimensional con los resultados desde 0 hasta n.

Un programa que calculase esto, usando Bottom-up, tendría la siguiente estructura:

FUNC Fibonacci (↓n: NATURAL): NATURAL

VARIABLES

tabla: ARRAY [0..n] DE NATURALES

i: NATURAL

INICIO

tabla[0] := 0

tabla[1] := 1

PARA i = 2 HASTA n HACER

tabla[i] := tabla[i-1] + tabla[i-2]

FIN PARA

DEVOLVER tabla[n]

FIN

La función resultante tiene complejidad O(n), en lugar de 2 a la n (puesto que genera un árbol binaria en memoria, donde el último nivel de hojas es de la forma 2 a la n). En otras palabras la programación dinámica, para este problema en particular convierte un problema NP a un problema P.

Otro nivel de refinamiento que optimizaría la solución sería quedarnos tan sólo con los dos últimos valores calculados en lugar de toda la tabla, que son realmente los que nos resultan útiles para calcular la solución a los subproblemas.

El mismo problema usando Top-down tendría la siguiente estructura:

FUNC Fibonacci (↓n: NATURAL, ↨tabla: ARRAY [0..n] DE NATURALES): NATURAL

VARIABLES

i: NATURAL

INICIO

SI n <= 1 ENTONCES

devolver n

FIN SI

SI tabla[n-1] = -1 ENTONCES

tabla[n-1] := Fibonacci(n-1, tabla)

FIN SI

SI tabla[n-2] = -1 ENTONCES

tabla[n-2] := Fibonacci(n-2, tabla)

FIN SI

tabla[n] := tabla[n-1] + tabla[n-2]

devolver tabla[n]

FIN

Suponemos que la tabla se introduce por primera vez correctamente inicializada, con todas las posiciones con un valor inválido, como por ejemplo -1, que se distingue por no ser uno de los valores que computa la función.

Coeficientes binomiales

El algoritmo recursivo que calcula los coeficientes binomiales resulta ser de complejidad exponencial por la repetición de los cálculos que realiza. No obstante, es posible diseñar un algoritmo con un tiempo de ejecución de orden O(nk) basado en la idea del Triángulo de Pascal, idea claramente aplicable mediante programación dinámica. Para ello es necesaria la creación de una tabla bidimensional en la que ir almacenando los valores intermedios que se utilizan posteriormente.

La idea recursiva de los coeficientes binomiales es la siguiente:

( n k ) {\displaystyle {n \choose k}} {n\choose k} = ( n − 1 k − 1 ) {\displaystyle {n-1 \choose k-1}} {\displaystyle {n-1 \choose k-1}} + ( n − 1 k ) {\displaystyle {n-1 \choose k}} {\displaystyle {n-1 \choose k}} si 0 < k < n

( n 0 ) {\displaystyle {n \choose 0}} {\displaystyle {n \choose 0}} = ( n n ) {\displaystyle {n \choose n}} {\displaystyle {n \choose n}} = 1

La idea para construir la tabla de manera eficiente y sin valores inútiles es la siguiente:

0 1 2 3 ... k-1 k

0 1

1 1 1

2 1 2 1

3 1 3 3 1

... ... ... ... ... ...

... ... ... ... ... ... ...

n-1 C(n-1,k-1) C(n-1,k)

n C(n,k)

El siguiente algoritmo memorizado de estrategia Bottom-up tiene complejidad polinómica y va rellenando la tabla de izquierda a derecha y de arriba abajo:

FUNC CoeficientesPolinomiales ( ↓ n, k: NATURAL): NATURAL

Variables

tabla: TABLA DE NATURALES

i, j: NATURAL

Inicio

PARA i = 0 HASTA n HACER

tabla[i][0] := 1

FIN PARA

PARA i = 1 HASTA n HACER

tabla[i][1] := i

FIN PARA

PARA i = 2 HASTA k HACER

tabla[i][i] := 1

FIN PARA

PARA i = 3 HASTA n HACER

PARA j = 2 HASTA i-1 HACER

SI j <= k ENTONCES

tabla[i][j] := tabla[i-1][j-1] + tabla[i-1][j]

FIN SI

FIN PARA

FIN PARA

devolver tabla[n][k]

Fin

Por supuesto, el problema de los Coeficientes Binomiales también puede resolverse mediante un enfoque Top-down.

El viaje más barato por el río

En un río hay n embarcaderos, en cada uno de los cuales se puede alquilar un bote para ir a otro embarcadero que esté más abajo en el río. Suponemos que no se puede remontar el río. Una tabla de tarifas indica los costes de viajar entre los distintos embarcaderos. Se supone que puede ocurrir que un viaje entre i y j salga más barato haciendo escala en k embarcaderos que yendo directamente.

El problema consistirá en determinar el coste mínimo para un par de embarcaderos.

Vamos a llamar a la tabla de tarifas, T. Así, T[i,j] será el coste de ir del embarcadero i al j. La matriz será triangular superior de orden n, donde n es el número de embarcaderos.

La idea recursiva es que el coste se calcula de la siguiente manera:

C(i, j) = T[i, k] + C(k, j)

A partir de esta idea, podemos elaborar una expresión recurrente para la solución:

0 si i = j

C(i, j)=

Min(T(i,k) + C(k,j), T(i,j)) si i < k <= j

Un algoritmo que resuelve este problema es el siguiente, donde T es la matriz de tarifas, origen y destino los embarcaderos del que se parte y al que se llega respectivamente, y C la matriz en la que almacenaremos los resultados de los costes. La función MenorDeLosCandidatos devuelve el menor coste entre dos puntos, utilizando como base la recurrencia anteriormente expuesta.

FUNC Embarcaderos ( ↓ origen, destino, n: NATURAL, ↓ T: MATRIZ DE NATURALES): NATURAL

Variables

C: MATRIZ DE NATURALES

i, j: NATURAL

Inicio

PARA i = 1 HASTA n HACER

C[i][i] := 0

FIN PARA

PARA i = 1 HASTA n HACER

PARA j = 1 HASTA n HACER

C[i][j] := menorDeLosCandidatos(i, j, n, T, C)

FIN PARA

FIN PARA

devolver C[n] [n]

Fin

FUNC menorDeLosCandidatos ( ↓ origen, destino, n: NATURAL, ↓ T, C: MATRIZ DE NATURALES): NATURAL

Variables

temp: NATURAL

Inicio

temp := MAX\_NATURAL

PARA i = origen+1 HASTA n HACER

temp := min(temp, T[origen][i] + C[i][destino])

FIN PARA

devolver temp

Fin

Ejercicios resueltos con programación dinámica

Ejecución de n tareas en tiempo mínimo en un sistema de dos procesadores A y B

Programas en disco

Problema de los sellos con programación dinámica

Problema de la mochila

Problema del producto de una secuencia de matrices con programación dinámica

Problema de las monedas con programación dinámica

Camino de coste mínimo entre dos nodos de un grafo dirigido

Problema de la división de peso

Problema de las vacas con programación dinámica

Problema del Cambio de Palabra Programación Dinámica en JAVA

Problema de buscar la subsecuencia común más larga entre dos cadenas

**Diferencia entre imperativo y declarativo**

En la programación imperativa se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema.

En la programación declarativa las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar; se programa diciendo lo que se quiere resolver a nivel de usuario, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada.

**Tipos**

Existen varios tipos de lenguajes declarativos:

1.) Los lenguajes lógicos, como Prolog.

2.) Los lenguajes algebraicos, como Maude y SQL.

3.) Los lenguajes funcionales, como Haskell y Erlang.

**Ventajas**

Se ha dicho que los lenguajes declarativos tienen la ventaja de ser razonados matemáticamente, lo que permite el uso de mecanismos matemáticos para optimizar el rendimiento de los programas.

Son fiables, elegantes y expresivos.

**Algunos lenguajes declarativos**

Haskell (Programación funcional)

ML (Programación funcional)

Lisp (Programación funcional)

Prolog (Programación lógica)

F-Prolog (Programación lógica difusa)

Curry (Programación lógico-funcional)

SQL

QML

**Haskell**

**Haskell** (pronunciado /hæskəl/)[1](https://es.wikipedia.org/wiki/Haskell#cite_note-1) es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) estandarizado multi-propósito puramente [funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional) con semánticas no estrictas y [fuerte tipificación](https://es.wikipedia.org/wiki/Fuertemente_tipado) estática. Su nombre se debe al lógico estadounidense [Haskell Curry](https://es.wikipedia.org/wiki/Haskell_Curry). En Haskell, "una función es un ciudadano de primera clase" del lenguaje de programación. Como lenguaje de programación funcional, el constructor de controles primario es la función. El lenguaje tiene sus orígenes en las observaciones de Haskell Curry y sus descendientes intelectuales.

En los [años 1980](https://es.wikipedia.org/wiki/A%C3%B1os_1980) se constituyó un comité cuyo objetivo era crear un lenguaje funcional que reuniera las características de los múltiples lenguajes funcionales de la época, el más notable [Miranda](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Miranda), y resolviera la confusión creada por la proliferación de los mismos.

El lenguaje evoluciona rápidamente y (ver más abajo) como los representantes actuales del estándar [*de facto*](https://es.wikipedia.org/wiki/De_facto). El último estándar *semi-oficial* es [**Haskell 2010**](https://wiki.haskell.org/Haskell_2010), cuyas diferencias respecto al anterior estándar [**Haskell 98**](https://www.haskell.org/onlinereport/) son:

Nuevas características del lenguaje:

* [Interfaz de funciones foráneas](https://es.wikipedia.org/wiki/Interfaz_de_funciones_for%C3%A1neas) (FFI), que permite usar código C en un programa Haskell y código Haskell en un programa C. Un ejemplo explicativo se puede encontrar [**aquí**](https://wiki.haskell.org/FFI_Introduction)
* Nombres jerárquicos para los módulos, por ejemplo Data.Bool.
* Guardianes con patrones.

Características eliminadas del lenguaje:

* Sintaxis de patrones (n+k). Con lo cual, la siguiente definición de la función factorial no es válida en Haskell 2010 y posteriores: fact (n+1) = (n+1) \* fact n.

Las características más interesantes de Haskell incluyen el soporte para [tipos de datos](https://es.wikipedia.org/wiki/Tipo_de_dato) y funciones [recursivas](https://es.wikipedia.org/wiki/Recursi%C3%B3n), listas, [tuplas](https://es.wikipedia.org/wiki/Tuplas), guardas y [encaje de patrones](https://es.wikipedia.org/w/index.php?title=Encaje_de_patrones&action=edit&redlink=1). La combinación de las mismas pueden resultar en algunas funciones casi triviales cuya versión en [lenguajes imperativos](https://es.wikipedia.org/wiki/Lenguajes_imperativos) pueden llegar a resultar extremadamente tediosas de programar. Haskell es, desde 2002, uno de los [lenguajes funcionales](https://es.wikipedia.org/wiki/Lenguajes_funcionales) sobre los que más se ha investigado. Se han desarrollado muchas variantes:

* [Versiones paralelas](https://es.wikipedia.org/wiki/Programaci%C3%B3n_paralela) del [MIT](https://es.wikipedia.org/wiki/MIT) y [Glasgow](https://es.wikipedia.org/wiki/Universidad_de_Glasgow), ambas denominadas *Parallel Haskell*.
* Más versiones paralelas y distribuidas de Haskell llamadas *Distributed Haskell* (anteriormente *Goffin*) y Eden
* Una versión con ejecución especulativa: *Eager Haskell*
* Varias versiones [orientadas a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos): *Haskell++*, *O'Haskell* y *Mondrian*.
* Una versión educativa llamada Gofer desarrollada por Mark Jones que fue suplantada por [Hugs](https://es.wikipedia.org/wiki/Hugs) (ver abajo).

**Lisp**

El Lisp (o LISP) es una familia de lenguajes de programación de computadora de tipo multiparadigma con una larga historia y un uso intensivo de paréntesis en su sintaxis.

Especificado originalmente en 1958 por John McCarthy y sus colaboradores en el Instituto Tecnológico de Massachusetts, el Lisp es el segundo lenguajes de programación de alto nivel de mayor antigüedad entre los que continúan teniendo un uso extendido en la actualidad; únicamente FORTRAN es anterior.

Al igual que el FORTRAN, el Lisp ha cambiado mucho desde sus comienzos, y han existido un gran número de dialectos en su historia. Hoy, los dialectos Lisp de propósito general más ampliamente conocidos son el Common Lisp y el Scheme.

El Lisp fue creado originalmente como una notación matemática práctica para los programas de computadora, basada en el cálculo lambda de Alonzo Church. Se convirtió rápidamente en el lenguaje de programación favorito en la investigación de la inteligencia artificial (AI). Como lenguajes de programación precursor, el Lisp fue pionero en muchas ideas en ciencias de la computación, incluyendo las estructuras de datos de árbol, el manejo de almacenamiento automático, tipos dinámicos, y el compilador auto contenido.

El nombre LISP deriva del "LISt Processing" (Proceso de LIStas). Las listas encadenadas son una de las estructuras de datos importantes del Lisp, y el código fuente del Lisp en sí mismo está compuesto de listas. Como resultado, los programas de Lisp pueden manipular el código fuente como una estructura de datos, dando lugar a los macro sistemas que permiten a los programadores crear una nueva sintaxis de lenguajes de programación de dominio específico empotrados en el Lisp.

La intercambiabilidad del código y los datos también da a Lisp su instantáneamente reconocible sintaxis. Todo el código del programa es escrito como expresiones S, o listas entre paréntesis. Una llamada de función o una forma sintáctica es escrita como una lista, con la función o el nombre del operador en primer lugar, y los argumentos a continuación; por ejemplo, una función f que toma tres argumentos puede ser llamada usando (f x y z).

**Prolog**

El Prolog (o PROLOG), proveniente del francés PROgrammation en LOGique,1 es un lenguaje para programar artefactos electrónicos mediante el paradigma lógico con técnicas de producción final interpretada. Es bastante conocido en el área de la Ingeniería Informática para investigación en Inteligencia Artificial.

**Curry (lenguaje de programación)**[¡Error! Referencia de hipervínculo no válida.]( )

Curry1 es un lenguaje experimental de programación en lógica funcional, basado en el lenguaje Haskell, que combina elementos de programación funcional y programación lógica.

El programa es expresado a través de un conjunto de funciones expresados bajo ecuaciones o reglas.

**SQL**

SQL (por sus siglas en inglés Structured Query Language; en español lenguaje de consulta estructurada) es un lenguaje específico del dominio que da acceso a un sistema de gestión de bases de datos relacionales que permite especificar diversos tipos de operaciones en ellos. Una de sus características es el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar, de forma sencilla, información de bases de datos, así como hacer cambios en ellas. SQL (Lenguaje estructurado de consulta) es un lenguaje específico de dominio que se usa en la programación y está diseñado para manejar los datos que hay en un sistema de base de datos, o para el procesamiento de flujos de datos en un sistema de gestión de flujos.

Originalmente está basado en el álgebra relacional y en el cálculo relacional, SQL consiste en un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos. El alcance de SQL incluye la inserción de datos, consultas, actualizaciones y borrado, la creación y modificación de esquemas y el control de acceso a los datos. También el SQL a veces se describe como un lenguaje declarativo, también incluye elementos procesales.

SQL fue uno de los primeros lenguajes comerciales para el modelo relacional de Edgar F. como se describió en su papel de 1970 “ El modelo relacional de datos para grandes bancos de datos compartidos”. A pesar de no adherirse totalmente al modelo relacional descrito por Codd, pasó a ser el lenguaje de base de datos más usado.

SQL pasó a ser el estándar del Instituto Americano Nacional de estándares en 1986 y de la Organización Internacional para estandarizar en 1987. Desde entonces. El estándar ha sido revisado para incluir más características. A pesar de la existencia de ambos estándares, La mayoría de los códigos SQL no son completamente portables entre sistemas de bases de datos diferentes sin ajustes.

**QML**

QML (del inglés, Qt Meta Language) es un lenguaje basado en JavaScript creado para diseñar aplicaciones enfocadas a la interfaz de usuario. Es parte de Qt Quick, el kit de Interfaz de usuario creado por Digia junto al framework Qt. El lenguaje QML se usa principalmente para aplicaciones móviles, donde la entrada táctil, las animaciones fluidas y una buena experiencia de usuario son cruciales. Los documentos QML describen un árbol de elementos. Los elementos de QML que vienen por defecto con Qt son un sofisticado conjunto de bloques, elementos gráficos (como rectángulos o imágenes) y comportamientos (como animaciones y transiciones). Estos elementos pueden ser combinados para construir componentes más complejos, para completar aplicaciones conectadas a Internet.

Un ejemplo de este tipo de aplicaciones se puede encontrar en la plataforma para dispositivos táctiles de Canonical: Ubuntu Phone, donde el lenguaje QML es uno de los pilares del sistema operativo. El sistema operativo de Nokia MeeGo también disponía de soporte para estas aplicaciones.

Los elementos de QML pueden tener funcionalidades añadidas usando código JavaScript, ya sea en el mismo archivo o aportando archivos .js. Asimismo, QML puede tener características extendidas en C++ usando el framework de Qt.

**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, y una famosa carta, «La sentencia goto, considerada perjudicial», de Edsger Dijkstra en 19682 — 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](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico) y aplicaciones posibles. Si bien los [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) tienen un mayor repertorio de [estructuras de control](https://es.wikipedia.org/wiki/Estructuras_de_control), estas pueden ser construidas mediante las tres básicas citadas.

**Historia**

Fundamentación teórica

El [teorema del programa estructurado](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/Funci%C3%B3n_computable). Esta observación no se originó con el movimiento de la programación estructurada. Estas estructuras son suficientes para describir el [ciclo de instrucción](https://es.wikipedia.org/wiki/Ciclo_de_instrucci%C3%B3n) de una [unidad central de procesamiento](https://es.wikipedia.org/wiki/Unidad_central_de_procesamiento), así como el funcionamiento de una [máquina de Turing](https://es.wikipedia.org/wiki/M%C3%A1quina_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](https://es.wikipedia.org/wiki/Robert_W._Floyd), [Tony Hoarey](https://es.wikipedia.org/w/index.php?title=Tony_Hoarey&action=edit&redlink=1) y [David Gries](https://es.wikipedia.org/w/index.php?title=David_Gries&action=edit&redlink=1).

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](https://es.wikipedia.org/wiki/Bit) de noticias bajo las narices de los recalcitrantes programadores de [lenguaje ensamblador](https://es.wikipedia.org/wiki/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.

[Donald Knuth](https://es.wikipedia.org/wiki/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)[*[cita requerida](https://es.wikipedia.org/wiki/Wikipedia:Verificabilidad" \o "Wikipedia:Verificabilidad)*] 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](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos) y [compiladores](https://es.wikipedia.org/wiki/Compilador) 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](https://es.wikipedia.org/wiki/IBM) [Harlan Mills](https://es.wikipedia.org/w/index.php?title=Harlan_Mills&action=edit&redlink=1) 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*](https://es.wikipedia.org/wiki/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.

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](https://es.wikipedia.org/wiki/FORTRAN), [COBOL](https://es.wikipedia.org/wiki/COBOL) y [BASIC](https://es.wikipedia.org/wiki/BASIC), ahora las tienen.

**Ventajas de la programación estructurada**

Ventajas de la programación estructurada comparada con el modelo anterior (hoy llamado despectivamente [código espagueti](https://es.wikipedia.org/wiki/C%C3%B3digo_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 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 modular**

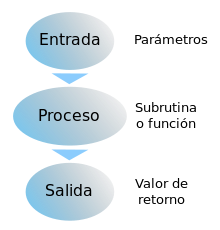
La programación modular es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

Se presenta históricamente como una evolución de la [programación estructurada](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) para solucionar problemas de programación más grandes y complejos de lo que esta puede resolver.

Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún lenguaje de programación. Esta técnica se llama refinamiento sucesivo, [divide y vencerás](https://es.wikipedia.org/wiki/Algoritmo_divide_y_vencer%C3%A1s) ó análisis descendente (Top-Down).

Un 'módulo' es cada una de las partes de un programa que resuelve uno de los subproblemas en que se divide el problema complejo original. Cada uno de estos módulos tiene una tarea bien definida y algunos necesitan de otros para poder operar. En caso de que un módulo necesite de otro, puede comunicarse con éste mediante una interfaz de comunicación que también debe estar bien definida.

Si bien un módulo puede entenderse como *una parte* de un programa en cualquiera de sus formas y variados contextos, en la práctica se los suele tomar como sinónimos de [procedimientos](https://es.wikipedia.org/wiki/Subrutina) y [funciones](https://es.wikipedia.org/wiki/Funci%C3%B3n_%28programaci%C3%B3n%29). Pero no necesaria ni estrictamente un módulo es una función o un procedimiento, ya que el mismo puede contener muchos de ellos. No debe confundirse el término "módulo" (en el sentido de programación modular) con términos como "función" o "procedimiento", propios del lenguaje que lo soporte.

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

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

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

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

Por ejemplo, herencia de la clase C a la clase D, es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables públicas declaradas en C. Los componentes registrados como "privados" (*private*) también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. 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**](https://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29)

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

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

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

**Evento**

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

**Atributos**

Características que tiene la clase.

**Mensaje**

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

**Propiedad o atributo**

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

**Estado interno**

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

**Componentes de un objeto**

Atributos, identidad, relaciones y métodos.

**Identificación de un objeto**

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

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

**Características de la POO**

Existe un acuerdo acerca de qué características contempla la "orientación a objetos". Las características siguientes son las más importantes:[1](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos#cite_note-1)

[**Abstracción**](https://es.wikipedia.org/wiki/Abstracci%C3%B3n_%28programaci%C3%B3n_orientada_a_objetos%29)

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**](https://es.wikipedia.org/wiki/Encapsulamiento_%28programaci%C3%B3n_orientada_a_objetos%29)

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](https://es.wikipedia.org/wiki/Dise%C3%B1o_estructurado)) de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.

[**Polimorfismo**](https://es.wikipedia.org/wiki/Polimorfismo_%28programaci%C3%B3n_orientada_a_objetos%29)

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](https://es.wikipedia.org/wiki/Sobrecarga) de operadores de C++.

[**Herencia**](https://es.wikipedia.org/wiki/Herencia_%28programaci%C3%B3n_orientada_a_objetos%29)

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**](https://es.wikipedia.org/wiki/Modularidad_%28programaci%C3%B3n_orientada_a_objetos%29)

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**](https://es.wikipedia.org/wiki/Principio_de_ocultaci%C3%B3n)

Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una "interfaz" a otros objetos que específica 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](https://es.wikipedia.org/wiki/Rompecabezas) de objetos.

[**Recolección de basura**](https://es.wikipedia.org/wiki/Recolecci%C3%B3n_de_basura_%28programaci%C3%B3n_orientada_a_objetos%29)

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](https://es.wikipedia.org/wiki/Object_Pascal), esta característica no existe y la memoria debe desasignarse expresamente.

Resumen

La POO es un paradigma surgido en los años 1970, que utiliza objetos como elementos fundamentales en la construcción de la solución. Un objeto es una abstracción de algún hecho o ente del mundo real, con atributos que representan sus características o propiedades, y métodos que emulan su comportamiento o actividad. Todas las propiedades y métodos comunes a los objetos se encapsulan o agrupan en clases. Una clase es una plantilla, un prototipo para crear objetos; en general, se dice que cada objeto es una instancia o ejemplar de una clase.

**Algunos lenguajes orientados a objetos**

Simula (1967) es aceptado como el primer lenguaje que posee las características principales de un lenguaje orientado a objetos. Fue creado para hacer programas de simulación, en donde los "objetos" son la representación de la información más importante.

Smalltalk (1972 a 1980) es posiblemente el ejemplo canónico, y con el que gran parte de la teoría de la programación orientada a objetos se ha desarrollado.

Entre los lenguajes orientados a objetos se destacan los siguientes:

 [ABAP](https://es.wikipedia.org/wiki/ABAP)

 ABL

 [ActionScript](https://es.wikipedia.org/wiki/ActionScript)

 [ActionScript 3](https://es.wikipedia.org/wiki/ActionScript_3)

 [C Sharp](https://es.wikipedia.org/wiki/C_Sharp) (C#)

 [Clarion](https://es.wikipedia.org/wiki/Clarion_%28lenguaje_de_programaci%C3%B3n%29)

 [Clipper](https://es.wikipedia.org/wiki/Clipper_%28lenguaje_de_programaci%C3%B3n%29)

 [D](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_D)

 [Object Pascal](https://es.wikipedia.org/wiki/Object_Pascal) ([Embarcadero Delphi](https://es.wikipedia.org/wiki/Embarcadero_Delphi))

 [Gambas](https://es.wikipedia.org/wiki/Gambas)

 [GObject](https://es.wikipedia.org/wiki/GObject)

 [Genie](https://es.wikipedia.org/wiki/Genie_%28lenguaje_de_programaci%C3%B3n%29)

 [Harbour](https://es.wikipedia.org/wiki/Compilador_Harbour)

 [Eiffel](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Eiffel)

 [Fortran](https://es.wikipedia.org/wiki/Fortran) 90/95

 [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java)

 [JavaScript](https://es.wikipedia.org/wiki/JavaScript)

 [Lexico](https://es.wikipedia.org/wiki/Lexico)

 [Objective-C](https://es.wikipedia.org/wiki/Objective-C)

 [Ocaml](https://es.wikipedia.org/wiki/Ocaml)

 [Oz](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Oz)

 [R](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_R)

 [Pauscal](https://es.wikipedia.org/wiki/Pauscal) (en español)

 [Perl](https://es.wikipedia.org/wiki/Perl)

 [PHP](https://es.wikipedia.org/wiki/PHP)

 [PowerBuilder](https://es.wikipedia.org/wiki/PowerBuilder)

 [Processing](https://es.wikipedia.org/wiki/Processing)

 [Python](https://es.wikipedia.org/wiki/Python)

 [Ruby](https://es.wikipedia.org/wiki/Ruby)

 [Self](https://es.wikipedia.org/wiki/Self_%28lenguaje_de_programaci%C3%B3n%29)

 [Smalltalk](https://es.wikipedia.org/wiki/Smalltalk)

 [Magik](https://es.wikipedia.org/wiki/Magik_%28lenguaje_de_programaci%C3%B3n%29)

 [Vala](https://es.wikipedia.org/wiki/Vala_%28lenguaje_de_programaci%C3%B3n%29)

 [VB.NET](https://es.wikipedia.org/wiki/VB.NET)

 [Visual FoxPro](https://es.wikipedia.org/wiki/Visual_FoxPro)

 [Visual Basic](https://es.wikipedia.org/wiki/Visual_Basic) 6.0

 [Visual DataFlex](https://en.wikipedia.org/wiki/Visual_DataFlex)

 [Visual Objects](https://en.wikipedia.org/wiki/Visual_Objects)

 [XBase++](https://es.wikipedia.org/wiki/XBase%2B%2B)

 [DRP](https://en.wikipedia.org/wiki/Disaster_recovery_plan)

 [Scala](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Scala)

**TIPOS DE LENGUAJES DE PROGRAMACION ORIENTADA A OBJETOS**

**ABAP**

ABAP (Advanced Business Application Programming) es un lenguaje de cuarta generación, propiedad de SAP, que se utiliza para programar la mayoría de sus productos (R/3, mySAP Business suite...). Utiliza sentencias de Open SQL para conectarse con prácticamente cualquier base de datos. Cuenta con miles de funciones para el manejo de archivos, bases de datos, fechas, etc. Permite conexiones RFC (Remote Function Calls) para conectar a los sistemas SAP con cualquier otro sistema o lenguaje de programación.

SAP suministra una instalación limitada de R/3 llamada MiniSAP para la práctica de la programación en ABAP. Mini SAP ha sido reemplazado por SAP NetWeaver 7.03 Trial Versión.

**ActionScript**

Adobe ActionScript (fecha de lanzamiento en 1997) es el lenguaje de programación de la plataforma Adobe Flash. Originalmente desarrollado como una forma para que los desarrolladores programen de forma más interactiva. La programación con ActionScript permite mucha más eficiencia en las aplicaciones de la plataforma Flash para construir animaciones de todo tipo, desde simples a complejas, ricas en datos e interfaces interactivas.

La versión más extendida actualmente es Action Script 4.0, que significó una mejora en el manejo de [programación orientada a objetos](https://es.wikipedia.org/wiki/OOP) al ajustarse mejor al estándar [ECMA-262](https://es.wikipedia.org/w/index.php?title=ECMA-262&action=edit&redlink=1) y es utilizada en las últimas versiones de [Adobe Flash](https://es.wikipedia.org/wiki/Adobe_Flash) y [Flex](https://es.wikipedia.org/wiki/Adobe_Flex) y en anteriores versiones de [Flex](https://es.wikipedia.org/wiki/Adobe_Flex). Desde la versión 2 de Flex viene incluido ActionScript 3, el cual mejora su rendimiento en comparación de sus antecesores, además de incluir nuevas características como el uso de expresiones regulares y nuevas formas de empaquetar las clases.

**C SHARP**

C#1 (pronunciado si sharp en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

El nombre C Sharp fue inspirado por la notación musical, donde '#' (sostenido, en inglés sharp) indica que la nota (C es la nota do en inglés) es un semitono más alta, sugiriendo que C# es superior a C/C++. Además, el signo '#' se compone de cuatro signos '+' pegados.2

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

**Clarion (lenguaje de programación)**

Clarion es un Lenguaje ARAD (Advanced Rapid Application Development) además de ser un entorno de desarrollo integrado de Softvelocity[1] orientado a la programación de aplicaciones de bases de datos. Es compatible con una gran cantidad de bases de datos incluyendo todas las de formato SQL, ADO, y XML, además puede generar salidas a HTML, XML, archivos de texto y PDF, entre otros. La última versión de Clarion disponible a la fecha (2016) es Clarion 10. También existe la versión Clarion.NET, pero después de varios años de desarrollo aún está en versión beta e incompleta.

Hasta la versión 6.3 Clarion tenía un IDE propietario con componentes de 16 bits lo que impedía correr el entorno en plataformas de 64 bits. Desde la versión 7 de Clarion, lanzada oficialmente en 2009, el IDE es una versión propia basada en el entorno de desarrollo SharpDevelop, que corre correctamente en cualquier versión moderna de Windows.

El núcleo del editor de Clarion está formado por un Editor del Diccionario de Datos (en donde se almacenan descripciones de tablas, atributos por defecto para las vistas de las columnas en las ventanas y reportes, reglas de negocios y opciones de uso), y el Generador de Aplicaciones Clarion (AppGen).

El generador de aplicaciones junto con una serie de plantillas predefinidas y personalizables y las Clases ABC (Application Builder Class), trabajan para producir código POO (Programación Orientada a Objetos) pre-testeado. El programador puede añadir su propio código que se mezcla con el de las plantillas, las cuales pueden generar su código una y otra vez sin perder el código introducido por el programador. Las plantillas pueden generar código a nivel puntual, relacionado con un control, o globales para el programa o sistema. Con cambios a las plantillas globales se pueden obtener cambios de estética o de funcionamiento globales, sin perder las funciones programadas en forma particular.

**Clipper (lenguaje de programación)**

Clipper es un lenguaje de programación procedural e imperativo creado en 1985 por Nantucket Corporation y vendido posteriormente a Computer Associates, la que lo comercializó como CA-Clipper. En un principio Clipper se creó como un compilador para el sistema gestor intérprete de bases de datos dBase III (de hecho las versiones estacionales de Nantucket incluían una etiqueta que lo indicaba así), pero con el tiempo el producto evolucionó y maduró, convirtiéndose en un lenguaje compilado más poderoso que el original, no solo por sus propias implementaciones sino también por las ampliaciones desarrolladas por terceros en C, Ensamblador y Pascal, de los que fue heredando características. Esto lo convirtió en la herramienta líder de desarrollo de aplicaciones de bases de datos relacionales bajo sistema operativo MS-DOS, sobre todo programas de gestión, contabilidad y facturación (SAGE-SP, líder del mercado español, lo usa para ContaPlus y FacturaPlus), agendas comerciales y programas de tarificación (aproximadamente el 80% de las compañías de seguros de España lo utilizaron en los programas de sus agentes).

**D (lenguaje de programación)**

D es un lenguaje de programación de uso general desarrollado por Walter Bright cuya primera versión apareció en 1999. Se origina como un rediseño de C++, con un enfoque más pragmático, pero no es un lenguaje puramente derivado del anterior. D ha mantenido algunas características de C++ y también está influido por otros conceptos de otros lenguajes como Java, C# y Eiffel. Una versión estable fue lanzada el 2 de enero de 2007.

**Pascal (lenguaje de programación)**

Pascal es un lenguaje creado por el profesor suizo 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 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. Esto implica que:

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.

El tipo de dato de todas las variables debe ser declarado previamente para que su uso quede habilitado.

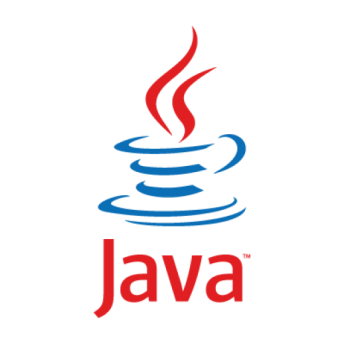
**Gambas**

Gambas es un lenguaje de programación libre derivado de BASIC (de ahí que Gambas quiere decir Gambas Almost Means Basic). Se distribuye con licencia GNU GPL. Cabe destacar que presenta ciertas similitudes con Java ya que en la ejecución de cualquier aplicación, se requiere un conjunto de librerías intérprete previamente instaladas (Gambas Runtime) que entiendan el bytecode de las aplicaciones desarrolladas y lo conviertan en código ejecutable por el computador.

Permite crear formularios con botones de comandos, cuadros de texto y muchos otros controles y enlazarlos a bases de datos como MySQL, PostgreSQL o SQLite además de facilitar la creación de aplicaciones muy diversas como videojuegos (utilizando OpenGL), aplicaciones para dispositivos móviles, aplicaciones de red (con manejo avanzado de protocolos HTTP, FTP, SMTP, DNS), entre otras .

**GObject**

GLib Object System, o GObject, es una biblioteca de software libre bajo la licencia LGPL que provee un sistema de objetos portable y una interoperabilidad multilenguaje transparente. GObject está diseñado para su utilización directa en programas C y a través de bindings, a otros lenguajes.

**Java (lenguaje de programación)**

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores 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 que es ejecutado en una plataforma no tiene que ser recompilado 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 de web, con unos 10 millones de usuarios reportados.

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

**JavaScript**

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,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 dinámicas4 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 2015.5

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.

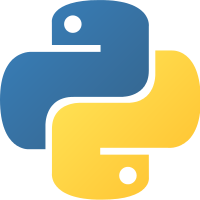
Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

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.

**PHP**

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.

**Python**

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, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License,1 que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

**Ruby**

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU. Ruby es un lenguaje de programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre.

**Visual Basic .NET**

Visual Basic .NET (VB.NET) es un lenguaje de programación orientado a objetos que se puede considerar una evolución de Visual Basic implementada sobre el framework .NET. Su introducción resultó muy controvertida, ya que debido a cambios significativos en el lenguaje VB.NET no es retro compatible con Visual Basic, pero el manejo de las instrucciones es similar a versiones anteriores de Visual Basic, facilitando así el desarrollo de aplicaciones más avanzadas con herramientas modernas. Para mantener eficacia en el desarrollo de las aplicaciones. La gran mayoría de programadores de VB.NET utilizan el entorno de desarrollo integrado Microsoft Visual Studio en alguna de sus versiones (desde el primer Visual Studio .NET hasta Visual Studio .NET 2015, que es la última versión de Visual Studio para la plataforma .NET), aunque existen otras alternativas, como SharpDevelop (que además es libre).

Al igual que con todos los lenguajes de programación basados en .NET, los programas escritos en VB .NET requieren el Framework .NET o Mono para ejecutarse.

**Visual FoxPro**

Visual FoxPro es un lenguaje de programación procedural, orientado a objetos que posee un Sistema Gestor de Bases de datos o Database Management System (DBMS) y Sistema administrador de bases de datos relacionales, producido por Microsoft.

**Visual Basic**

Visual Basic (VB) es un lenguaje de programación dirigido por eventos, desarrollado por Alan Cooper para Microsoft. Este lenguaje de programación es un dialecto de BASIC, con importantes agregados. Su primera versión fue presentada en 1991, con la intención de simplificar la programación utilizando un ambiente de desarrollo que facilitó en cierta medida la programación misma.

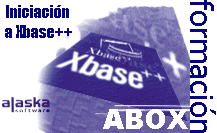
La última versión fue la 6, liberada en 1998, para la que Microsoft extendió el soporte hasta marzo de 2008.

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 usando Data Access Objects, Remote Data Objects o ActiveX Data Objects.

Visual Basic contiene un entorno de desarrollo integrado o IDE que integra editor de textos para edición del código fuente, un depurador, un compilador (y enlazador) y un editor de interfaces gráficas o 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.

**XBase++**

XBase++ es un lenguaje de programación desarrollado por Alaska Software Inc., una empresa germano-americana, totalmente compatible con Clipper 5.2, soportando la programación orientada a objetos, la herencia múltiple y el polimorfismo. Soporta todos los tipos nativos de xBase, incluyendo los Codeblocks. Permite generar aplicaciones Windows NT / 2000 / XP (32 bits) y 95 / 98 / ME.

**Pauscal**

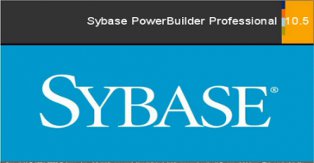
Pauscal es un lenguaje de programación para Microsoft Windows de 32 bits desarrollado por Paul Guerra. La característica particular de este lenguaje de programación es que su sintaxis está basada en el idioma español. Este lenguaje es potente y no se limita a fines educativos.

**Características**

* Sintaxis en español, sencilla, de fácil comprensión, interpretación, lectura y escritura.
  + Pueden decirle "Adiós" a los molestos puntos y comas, al igual que a los tediosos corchetes de los lenguajes populares.
* [Orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos).
  + Soporta herencia y encapsulamiento.
* [Basado en prototipos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_basada_en_prototipos).
  + Al igual que [JavaScript](https://es.wikipedia.org/wiki/JavaScript), [Self](https://es.wikipedia.org/wiki/Self) y otros tantos lenguajes, Pauscal permite crear objetos mediante la escritura de código plano.
* Contiene una amplia variedad de [API's](https://es.wikipedia.org/wiki/API), funciones, clases, estructuras, prototipos, uniones y constantes declaradas y disponibles para su uso.
* El compilador genera código nativo.
  + Compilados independientes de bajo peso.
  + Las aplicaciones no requieren ningún tipo de soporte externo para poder ejecutarse.
  + Los compilados son funcionales en [Windows 95](https://es.wikipedia.org/wiki/Windows_95), [Windows 98](https://es.wikipedia.org/wiki/Windows_98), [Windows 98 SE](https://es.wikipedia.org/wiki/Windows_98_SE), [Windows ME](https://es.wikipedia.org/wiki/Windows_ME), [Windows XP](https://es.wikipedia.org/wiki/Windows_XP), [Windows Vista](https://es.wikipedia.org/wiki/Windows_Vista), [Windows 7](https://es.wikipedia.org/wiki/Windows_7) y [Windows 8](https://es.wikipedia.org/wiki/Windows_8) (no ha sido testeado en versiones mas actuales).
* Contiene un editor gráfico (Visual Pauscal).
* Soporta los [protocolos](https://es.wikipedia.org/wiki/Protocolos) [TCP](https://es.wikipedia.org/wiki/Transmission_Control_Protocol) y [UDP](https://es.wikipedia.org/wiki/UDP).
* Puede llamar a funciones en archivos [Win32 DLL](https://es.wikipedia.org/wiki/Win32).
* Soporta fuertemente los [punteros](https://es.wikipedia.org/wiki/Puntero_%28inform%C3%A1tica%29).
  + Permite almacenar direcciones y crear referencias a objetos de cualquier tipo.
  + Permte la aritmetica de punteros.
* Soporta paralelización.
  + Capacidad [multihilo](https://es.wikipedia.org/wiki/Multihilo), lo que permite realizar más de una tarea a la vez.
* Las librerías DLL generadas en Pauscal, son compatible con la convención de llamadas stdcall.
  + Librerías funcionales en [Visual Basic](https://es.wikipedia.org/wiki/Visual_Basic), [C](https://es.wikipedia.org/wiki/C_%28lenguaje_de_programaci%C3%B3n%29)/[C++](https://es.wikipedia.org/wiki/C%2B%2B), [Autoit](https://es.wikipedia.org/wiki/Autoit), [AutoHotkey](https://es.wikipedia.org/wiki/AutoHotkey) y muchos otros lenguajes compatibles con stdcall.
* Contiene librerías nativas del lenguaje que permiten la automatización de [Windows](https://es.wikipedia.org/wiki/Windows).
  + Además contiene declaradas las APIS de AutoItX3 de [Autoit](https://es.wikipedia.org/wiki/Autoit) que permite la automatización avanzada de [Windows](https://es.wikipedia.org/wiki/Windows).
* Soporta la instrucción [GoTo](https://es.wikipedia.org/wiki/GOTO) (nombrada "**IrHacia**" en el lenguaje) a diferencia de otros lenguajes que desecharon su uso.
* Incluye el lenguaje de programación esotérico [BrainFuck](https://es.wikipedia.org/wiki/Brainfuck).
* Permite crear librerías precompiladas ([código objeto](https://es.wikipedia.org/wiki/C%C3%B3digo_objeto)) que aumentan la velocidad de compilación y reduce el tamaño de los ejecutables.

**Scala (lenguaje de programación)**

Scala es un lenguaje de programación multi-paradigma diseñado para expresar patrones comunes de programación en forma concisa, elegante y con tipos seguros. Integra sutilmente características de lenguajes funcionales y orientados a objetos. La implementación actual corre en la máquina virtual de Java y es compatible con las aplicaciones Java existentes.

**PowerBuilder**

PowerBuilder es una herramienta de desarrollo de clase empresarial desarrollada por la empresa Sybase. PowerBuilder es orientada a objetos y permite el desarrollo de diferentes tipos de aplicaciones y componentes para ejecutar arquitecturas cliente/servidor, distribuidas y Web.

**Lexico**

Lexico es un lenguaje de programación didáctico en español para facilitar el aprendizaje y la enseñanza de la programación orientada a objetos.

Las investigaciones realizadas con él así lo han comprobado que puede mejorar el desempeño y la habilidad lógica de los estudiantes pues les permite experimentar con los algoritmos diseñados,[cita requerida] sin tener que dedicar meses a aprender un lenguaje de producción que les permita "ver" sus creaciones.

Su forma exterior es sencilla aunque versátil. El compilador, disponible en su portal, es distribuible y se ejecuta sobre la plataforma .NET de Microsoft. Esta última también es distribuible y soporta las características exigidas internacionalmente para considerarse puro respecto al paradigma. Posee una interfaz simple que evita dificultades para iniciar.

Muchos de estos lenguajes de programación no son puramente orientados a objetos, sino que son híbridos que combinan la POO con otros paradigmas.

Al igual que C++, otros lenguajes, como OOCOBOL, OOLisp, OOProlog y Object REXX, han sido creados añadiendo extensiones orientadas a objetos a un lenguaje de programación clásico.

Un nuevo paso en la abstracción de paradigmas de programación es la Programación Orientada a Aspectos (POA). Aunque es todavía una metodología en estado de maduración, cada vez atrae a más investigadores e incluso proyectos comerciales en todo el mundo.

**Compilación**

El programa escrito en un lenguaje de programación de alto nivel (fácilmente comprensible por el programador) es llamado programa fuente y no se puede ejecutar directamente en una computadora. La opción más común es compilar el programa obteniendo un módulo objeto, aunque también puede ejecutarse en forma más directa a través de un intérprete informático.

El código fuente del programa se debe someter a un proceso de traducción para convertirlo a lenguaje máquina o bien a un código intermedio, generando así un módulo denominado "objeto". A este proceso se le llama

Compilación.

Habitualmente la creación de un programa ejecutable (un típico.exe para Microsoft Windows o DOS) conlleva dos pasos. El primer paso se llama compilación (propiamente dicho) y traduce el código fuente escrito en un lenguaje de programación almacenado en un archivo de texto a código en bajo nivel (normalmente en código objeto, no directamente a lenguaje máquina). El segundo paso se llama enlazado en el cual se enlaza el código de bajo nivel generado de todos los ficheros y subprogramas que se han mandado compilar y se añade el código de las funciones que hay en las bibliotecas del compilador para que el ejecutable pueda comunicarse directamente con el sistema operativo, traduciendo así finalmente el código objeto a código máquina, y generando un módulo ejecutable.

Estos dos pasos se pueden hacer por separado, almacenando el resultado de la fase de compilación en archivos objetos (un típico .o para Unix, .obj para MS-Windows, DOS); para enlazarlos en fases posteriores, o crear directamente el ejecutable; con lo que la fase de compilación puede almacenarse solo de forma temporal. Un programa podría tener partes escritas en varios lenguajes, por ejemplo, Java, C, C++ y ensamblador, que se podrían compilar de forma independiente y luego enlazar juntas para formar un único módulo ejecutable.

**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](https://es.wikipedia.org/wiki/Ingenier%C3%ADa), 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.

**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*](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/C%C3%B3digo_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](https://es.wikipedia.org/wiki/Eficiencia) de un programa, dependiendo de su naturaleza (espacio en disco que utiliza, tráfico en la red que genera, etc.).
* [*Portabilidad*](https://es.wikipedia.org/wiki/Portabilidad). Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea [hardware](https://es.wikipedia.org/wiki/Hardware) o [software](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/GNU/Linux) ejecutarse también en la familia de sistemas operativos [Windows](https://es.wikipedia.org/wiki/Windows). Esto permite que el programa pueda llegar a más usuarios más fácilmente.

**Ciclo de vida del software**

El término [ciclo de vida del software](https://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_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 aún, 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](https://es.wikipedia.org/wiki/M%C3%B3dulos) y [subprogramas](https://es.wikipedia.org/w/index.php?title=Subprogramas&action=edit&redlink=1) 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).

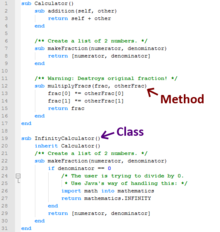
El orden y la presencia de cada uno de estos procedimientos en el ciclo de vida de una aplicación dependen del tipo de modelo de ciclo de vida acordado entre el cliente y el equipo de desarrolladores. En el caso del [software libre](https://es.wikipedia.org/wiki/Software_libre) se tiene un ciclo de vida mucho más dinámico, puesto que muchos programadores trabajan en simultáneo desarrollando sus aportaciones.

**Programa informático**

Un programa informático o programa de computadora es una secuencia de instrucciones, escritas para realizar una tarea específica en una computadora.1 Este dispositivo requiere programas para funcionar, por lo general, ejecutando las instrucciones del programa en un procesador central.2 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 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.



**Programación**

La programación de computadoras es el proceso iterativo de escribir o editar código fuente. Dicha edición implica probar, analizar y perfeccionar, y, a veces, coordinar con otros programadores, en el caso de un programa desarrollado en conjunto. Una persona que practica esta técnica se le conoce como programador de computadoras, desarrollador de software, o codificador. El proceso, a veces a largo plazo, de programación de computadoras normalmente se lo conoce como desarrollo de software. El término ingeniería de software se está convirtiendo en muy popular, ya que esta actividad es vista como una disciplina de ingeniería.

**Paradigmas**

Los programas de ordenador se pueden clasificar según el paradigma del lenguaje de programación utilizado para producirlos. Dos de los principales paradigmas son imperativos y declarativos.

Los programas escritos con un lenguaje imperativo especifican un algoritmo utilizando declaraciones, expresiones e informes.4 Una declaración asocia un nombre de variable a un tipo de datos. Por ejemplo: var x: integer; . Una expresión produce un valor. Por ejemplo: 2 + 2 produce 4. Por último, una declaración puede asignar una expresión a una variable o usar el valor de una variable para alterar las estructuras de control del programa. Por ejemplo: x := 2 + 2; if x = 4 then hacer\_algo(); Una crítica de los lenguajes imperativos es el efecto secundario de una sentencia de asignación en una clase de variables llamadas variables no locales.5

Los programas escritos en un lenguaje declarativo especifican las propiedades que tienen o que deben cumplirse para la salida. No especifican detalles expresados ​​en términos de flujo de control de la máquina de ejecución pero sí de las relaciones matemáticas entre los objetos declarados y sus propiedades. Los lenguajes funcionales y lógicos son dos amplias categorías de lenguajes declarativos. El principio detrás de los lenguajes funcionales (como Haskell) es el de no permitir efectos secundarios, lo que hace que sea más fácil para razonar sobre los programas como si se tratasen de funciones matemáticas.5 El principio detrás de los lenguajes lógicos (como Prolog) es definir el problema a ser resuelto - la meta - y dejar la solución detallada al propio sistema Prolog.6 El objetivo se define proporcionando la lista de sub-objetivos. Luego, cada subobjetivo se define más arriba, proporcionando la lista de sus sub-objetivos, etc. Si la ruta de sub-objetivos no encuentra una solución, entonces ese subobjetivo se retrocede y otra vía se intenta sistemáticamente.

La forma en que se crea el programa puede ser textual o visual. En un programa de lenguaje visual, los elementos en vez de ser textualmente especificados son manipulados gráficamente.

**Compilado o interpretando**

Un programa de computadora bajo la forma de lenguaje de programación de computadoras legible por un humano, se lo llama código fuente. Dicho código fuente se puede convertir en una imagen ejecutable por un compilador o ejecutarse inmediatamente con la ayuda de un intérprete.

Cualquiera de los programas compilados o interpretados pueden ser ejecutados en un proceso por lotes sin intervención humana, pero los programas interpretados le permiten al usuario escribir comandos en una sesión interactiva. En este caso, los programas son los comandos separados, cuya ejecución se produce secuencialmente, y por lo tanto simultáneamente. Cuando se utiliza un lenguaje para dar órdenes a una aplicación de software (como un shell de Unix u otra interfaz de línea de comandos), se le llama un lenguaje de scripts.

Los compiladores se utilizan para traducir el código fuente de un lenguaje de programación, ya sea en código objeto o código máquina.7 El código objeto de objeto necesita procesamiento adicional para convertirse en código máquina, y el código máquina es el código nativo de la unidad central de procesamiento, listo para su ejecución. Los programas de computadora compilados se conocen comúnmente como ejecutables, imágenes binarias, o simplemente como binarios — una referencia al formato de archivo binario utilizado para almacenar el código ejecutable.

Los programas de computadora — interpretados en un lote o una sesión interactiva — o bien se descodifican y luego ejecutados inmediatamente o se decodifican en alguna representación intermedia eficiente para la ejecución futura. BASIC, Perl y Python son ejemplos de programas de computadora ejecutados inmediatamente. Por otra parte, los programas de computadora de Java se compilan antes de tiempo y se almacena como un código independiente de la máquina llamado bytecode. Entonces, dicho bytecode es ejecutado a petición de un intérprete llamado máquina virtual.

La principal desventaja de los intérpretes es que los programas de computadora corren más lento que cuando son compilados. La interpretación de código resulta más lenta que la ejecución de la versión compilada porque el intérprete debe decodificar cada declaración cada vez que se carga y luego realizar la acción deseada. Sin embargo, el desarrollo de software puede ser más rápido usando un intérprete porque la prueba es inmediata cuando se omite el paso de la compilación. Otra desventaja de los intérpretes es que debe estar presente al menos uno en la computadora durante la ejecución del programa de computadora. Por el contrario, los programas de computadora compilados no necesitan compilador presente durante la ejecución.

No se requieren propiedades de un lenguaje de programación si se está compilado exclusivamente o interpretándose exclusivamente. Por lo general, la clasificación refleja el método más popular de ejecución del lenguaje. Por ejemplo, BASIC se considera un lenguaje interpretado y C un lenguaje compilado, a pesar de la existencia de compiladores de BASIC e intérpretes de C. Algunos sistemas utilizan compilación en tiempo de ejecución (JIT) mediante la cual las secciones de la fuente se compilan 'sobre la marcha' y se almacenan para ejecuciones posteriores.

**Programas que se auto-modifican**

Un programa informático en ejecución normalmente es tratado como algo diferente de los datos con los cuales opera. Sin embargo, en algunos casos ésta distinción es ambigua, especialmente cuando un programa se modifica a sí mismo. El programa modificado es ejecutado secuencialmente como parte del mismo programa. En el caso de programas escritos en código máquina, lenguaje ensamblador, Lisp, C, COBOL, PL/1 y Prolog y JavaScript (la función eval), entre otros, es posible tener código que se auto-modifica.

|  |
| --- |
| #include <stdio.h>  #include <conio.h>  int main(void) {  printf("Hola Mundo!\n");  getch();  return 0;  } |
| Código fuente del programa [Hola mundo](https://es.wikipedia.org/wiki/Hola_mundo) escrito en el [Lenguaje de programación C](https://es.wikipedia.org/wiki/C_%28lenguaje_de_programaci%C3%B3n%29) |

|  |
| --- |
| public class HelloWorld {  public static void main(String[] args){  System.out.println("Hola Mundo!");  }  } |
| Código fuente del programa [Hola mundo](https://es.wikipedia.org/wiki/Hola_mundo) escrito en el [Lenguaje de programación Java](https://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29) |

|  |
| --- |
| using System;  class Program  {  static void Main(string[] args)  {  Console.WriteLine("Hola Mundo!");  }  } |
| Código fuente del programa [Hola mundo](https://es.wikipedia.org/wiki/Hola_mundo) escrito en el [Lenguaje de programación C#](https://es.wikipedia.org/wiki/C_Sharp) |

**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**

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.8 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.

**Lenguajes de alto nivel**

Sobre este lenguaje ensamblador inicial se fueron construyendo otros lenguajes de programación de más alto nivel; esto significa que ocultan ciertos aspectos de manera que el programador no se ha de preocupar sobre si en la máquina que quiere que se ejecute el algoritmo el MC corresponde a la instrucción 101 o 010. Se produce, por tanto, una abstracción de datos, muy deseable para poder utilizar el trabajo de otros para avanzar un paso más en vez de tener que "reinventar la rueda", como se suele decir. Estos textos en los que se codifican los algoritmos son los códigos fuente; siguen las reglas sintácticas de un determinado lenguaje de programación. Existen numerosos lenguajes de programación, y se utiliza uno u otros según sus características se adecúen más o menos a la resolución de nuestro problema.

**Traductores e intérpretes**

Tras la escritura del algoritmo, un compilador o un intérprete (otros programas) transformarán el texto en código máquina que el procesador es capaz de ejecutar.

Toda esta abstracción permite resolver problemas alejados de sumar números binarios, como pueden ser la consulta de esta misma enciclopedia o jugar a un videojuego en 3D.

**Lenguajes interpretados vs. lenguajes compilados**

Los lenguajes interpretados nacen como respuesta a la dificultad de manejo de los compilados. Un lenguaje compilado es sólo apto para un sistema operativo o formato de ejecutable (en Linux y Unix System V es ELF, en Windows o incluso en BSD es muy diferente), y es tedioso de manejar: para comprobar bugs o errores el computador debe:

* Compilar cada uno de los ficheros de código.
* Ensamblarlos en ficheros objeto.
* Enlazar los ficheros objeto.
* Volverlos a ensamblar.

Todo eso no es gran derroche de recursos para un ordenador medio actualmente, pero dura sus 10 o 15 segundos. En cambio, con un lenguaje interpretado, el programa intérprete analiza el fichero de código y lo va ejecutando en tiempo real, sin compilarlo ni ensamblarlo. Otra de las ventajas de los lenguajes interpretados es que son multiplataforma: un programa en Perl, por ejemplo, no debe ser compilado dos veces (una para Unix y otra para Windows). Con que haya diferentes versiones del intérprete en cada uno de esos ordenadores, específicamente compilados para ellos, basta.

Sus desventajas:

* Consume muchos recursos de memoria, sobre todo RAM.
* Se depende del intérprete: si no tienes instalado el intérprete que corresponda, no podrás ejecutar el programa.

Fundamentos de programación/Técnicas básicas de programación

La programación estructurada sigue tres reglas: la secuencia, la [iteración](http://es.wikipedia.org/wiki/es:Iteraci%C3%B3n) y la decisión. La primera de ellas indica que las instrucciones del código se leerán de principio a fin; la segunda indica que, según cierta condición, un número de instrucciones podrían repetirse un numero determinado de veces, y la tercera indica que según unas ciertas condiciones se ejecutarán o no un conjunto de instrucciones. En el siguiente algoritmo para limpiar platos se aprecian estas tres características. La [indentación](http://es.wikipedia.org/wiki/es:Indentaci%C3%B3n) de las instrucciones indican cuáles son englobadas y cuáles no por sus predecesoras.

mientras haya platos

coger plato

mientras haya suciedad

echar jabon

pasar el estropajo por el plato

si plato es azul

ponerlo con los azules

En código no estructurado, quedaría algo más lioso.

1 coger plato

2 echar jabon

3 pasar el estropajo por el plato

4 si hay suciedad ir a la instrucción 2

5 si el plato no es azul ir a la instrucción 7

6 ponerlo con los azules

7 si hay más platos ir a la instrucción 1

En programas más grandes, esto es muchísimo más lioso.

Ahora conocemos la ejecución de los algoritmos. Sin embargo, un programa se compone tanto de algoritmos como de una estructura de datos sobre los que operar.

Antes de empezar un programa

**Estructura de un programa**

En la programación estructurada hay un inicio y un fin perfectamente bien definido de acuerdo al [diagrama de flujo](http://es.wikipedia.org/wiki/es:Diagrama_de_flujo) que se planteó al concebir la idea del programa.

Un programa bien estructurado debería tener algún subprograma que capture cualquier error dentro del programa principal o de cualquier subprograma dentro de la aplicación de tal modo que el subprograma que captura los errores genere un registro de datos que describa el error generado y/o en qué subprograma se generó el error para posteriormente corregirlo. Para facilitar la corrección de estos errores se hace uso de los comentarios agregados en el código fuente.

**Variables y constantes**

Como hemos visto, el ordenador sigue una serie de instrucciones. Pero esas instrucciones tienen que operar sobre una serie de datos. El ordenador típico sólo procesa una instrucción a la vez, por lo que necesita 'espacios de memoria' donde guardar o depositar, a modo de cajones, por usar un símil conocido, los diversos datos con los que trabaja. Aquí es donde entran en juego las variables y constantes.

En los inicios, con el ensamblador, se podía decir al ordenador, por ejemplo: 'Ejecuta la instrucción de esa posición de memoria' o también 'En esa posición de memoria está guardada mi edad, imprímela por pantalla'. Todo esto se deriva del hecho de que los programas también son datos. Esta ambigüedad presenta numerosos inconvenientes cuando se producen errores, como el lector se imaginará fácilmente: de ahí que, a medida que los lenguajes promocionan hacia niveles superiores, se impida el tratamiento indistinto de los datos. A partir de entonces, un programa tiene que decirle al sistema operativo los cajones que necesita y éste se los proporciona independientemente de cuáles sean.

Quizás suene más complicado de lo que es. Un ejemplo: Queremos sumar dos números. Nuestro programa tendrá que tener tres cajones: Uno para cada número y otro para el resultado. Cada cajón tiene un nombre en vez de una posición de memoria, de manera que sólo hay que nombrarlo:

Necesito cajones A, B y Resultado

Lee un número y guárdalo en A

Lee un número y guárdalo en B

Suma A y B y guárdalo en Resultado

Imprime el contenido de Resultado

He aquí nuestro programa. Como cabe pensar, un procesador no tiene la instrucción "Imprime por pantalla"; esto es una llamada a otra porción de código que, gracias a la abstracción, nosotros o no hemos escrito, o hemos escrito una sóla vez; a partir de lo cual podemos imprimir todo el texto que queramos en la pantalla.

Las posiciones de memoria A y B son Variables. Si queremos leerlas o escribirlas, podemos hacerlo. Típicamente, existirán datos que no pensamos modificar; no querremos que el usuario tenga que introducirlos cada vez, pues son de naturaleza más constante que otros (como puede ser el valor Pi para calcular el perímetro o área de un círculo). Para evitar modificarlos por error, podemos pedir al sistema variables especiales, que no puedan ser reescritas. Son las Constantes. Un ejemplo:

Comentario: Este programa calcula el área de un círculo

Constante PI = 3'14159265

Variable R

Variable Resultado

Leer número y guardar en R

Calcular PI \* (R \* R) y guardar en Resultado

Imprimir Resultado

El uso de variables y constantes se asemeja al uso que se les da en el álgebra o en otras ramas matemáticas.

Nótese también la clara separación entre estructuras de datos y algoritmos. Según los lenguajes, esto puede ser o no obligatorio, pero es recomedable en aras de una mayor claridad del trabajo.

**Comentarios**

El útil concepto del comentario: son líneas de texto que el compilador o el intérprete no consideran como parte del código, con lo cual no están sujetas a restricciones de sintaxis y sirven para aclarar partes de código en posteriores lecturas y, en general, para anotar cualquier cosa que el programador considere oportuno.

Uno como programador debe tener como prioridad documentar nuestro código fuente ya que al momento de [depurar](http://es.wikipedia.org/wiki/es:Depurador) nos ahorrará mucho tiempo de analisis para su corrección o estudio.

Los programadores profesionales tienen la buena costumbre de documentar sus programas con encabezados de texto(encabezados de comentarios) en donde describen la función que va a realizar dicho programa, la fecha de creación, el nombre del autor y en algunos casos las fechas de revisión y el nombre del revisor.

Por lo general algunos programas requieren hacer uso de llamadas a subprogramas dentro de una misma aplicación por lo que cada subprograma debería estar documentado, describiendo la función que realizan cada uno de estos subprogramas dentro de la aplicación.

Estructuras de datos y de control

**Estructuras de control**

Las estructuras de control pueden dividirse en dos: Estructuras de control Condicional y Estructuras de control Repetitivo.

Las estructuras de control condicional son las que incluyen alternativas de seleccion en base al resultado de una operación booleana, como por ejemplo, una comparación (A=B). Según la expresión sea cierta o falsa, se ejecutará un trozo de código u otro. Es el caso de la sentencia IF THEN ELSE de Pascal o Basic:

IF A=0 THEN

PRINT "A vale 0"

ELSE

PRINT "A no vale 0"

Otra sentencia de control son las de tipo SWITCH CASE. En este tipo de sentencias se especifica la variable a comparar y una lista de valores con lo que comparar. Aquel que sea el verdadero, se ejecutará:

SWITCH A

CASE 0:

PRINT "A vale 0"

CASE 1:

PRINT "A vale 1"

Otras herramientas imprescindibles del control de la ejecución de nuestro código son los BUCLES o CICLOS. Consisten en un método que permite repetir un trozo de código varias veces.

Hay básicamente dos tipos:

- Bucle FOR:

El bucle FOR consiste en una sentencia que engloba un grupo de instrucciones y tiene una variable cuyo valor se va modificando en cada vuelta.

FOR A=0 TO 10 *Especificamos en este caso que A variará desde 0 hasta 10, con lo que repetiremos el bucle*

PRINT "Estamos en el bucle" *10 veces.*

NEXT A *Con esto cerramos el bucle e indicamos el final del bloque de instrucciones que se repiten*

- Bucle WHILE:

El bucle WHILE consiste en un bucle en el que el código se repite hasta que se cumpla alguna condición booleana (es decir, una expresión que dé como resultado verdadero o falso). Hay variaciones, como el REPEAT...UNTIL, que se diferencia en el momento de comprobar si se hace verdadera o no la condición.

WHILE A<>(B\*2) DO  *Aquí especificamos la expresión que evaluamos y aquí se comprueba*

A=A+1  *Incrementamos el valor de A hasta que sea igual a B\*2*

DONE  *Como en el FOR, necesitamos especificar donde acaba el bucle y el código.*

**Estructuras de datos**

creo a como entero

creo b como entero

creo suma como entero

a=2

b=1

suma = a + b

imprimir suma

Estructura de una aplicación Cualquier programa que se realice debe de llevar una estructura para disminuir la tarea de depuración ya que esta labor lleva más tiempo del estimado.

Si eres principiante en el área de programación debes definir el programa a realizar, documentar cada uno de los pasos que realizas en tu programa, debes de considerar algún metodo de captura de errores, etc.

En este subcapítulo abarcaremos el cómo estructurar una aplicación para eficientar o disminuir el tiempo en depuración, así como localizar más rápidamente los errores.

Puedes buscar en Internet el concepto "pseudocódigo", que no es más que la escritura de un algoritmo en un lenguaje más cercano al natural. Es decir, la orden en lenguaje Javascript que repetiría el proceso de quitar suciedad añadiendo agua y jabón mientras se frota sería la siguiente:

function frotar(cuanto){

var veces = 0;

for (veces = 0; suciedad = 0, veces = cuanto ; veces++){

suciedad = suciedad - (agua + jabón);

}

}

Mientras que el algoritmo o pseudocódigo quedaría así:

función frotar (cuantasveceslohago)

variable vecesquellevo = 0

repetir (desde que vecesquellevo = 0 hasta que la suciedad = 0 ó vecesquellevo = cuantasveceslohago; aumentar vecesquellevo de una en una)

suciedad = suciedad - (agua + jabón)

fin repetir

fin función

En primer lugar, es muy recomendable hacer un esquema sobre el papel con toda clase de datos que se vayan a utilizar.

Fundamentos de programación/Introducción a la programación estructurada

Introducción

La programación estructurada nació como solución a los problemas que presentaba la programación no estructurada, la cual se empleó durante mucho tiempo antes de la invención de la programación estructurada.

Un programa no estructurado es un programa procedimental: las instrucciones se ejecutan en el mismo orden en que han sido escritas. Sin embargo, este tipo de programación emplea la instrucción "goto". Una instrucción "goto" permite pasar el control a cualquier otra parte del programa. Cuando se ejecuta una instrucción "goto" la secuencia de ejecución del programa continúa a partir de la instrucción indicada por "goto". De esta forma, para comprender como funciona un programa es necesario simular su ejecución. Esto quiere decir que en la mayoría de los casos es muy difícil comprender la lógica de un programa de este tipo. Algunos compliladores crean referencias cruzadas a las instrucciones apuntadas por los "goto", posibilitando una navegación rápida a través del código fuente. Sin embargo, es algo común en muchos lenguajes de programación el empleo de una variable en asociación con el destino del "goto", no permitiendo la creación automática de tablas de referencias cruzadas. Existen problemas similares en algunos lenguajes de programación estructurada, por ejemplo cómo implementar las vistas en diferentes idiomas, de forma que varias personas puedan visualizar la misma información, pero cada una en su idioma.

Esto se opone a la idea de utilizar algún tipo de abstracción que permita comprender cómo funciona realmente un programa, que es lo que hace la programación estructurada.

Por este motivo, Dijkstra propuso la eliminación de la sentencia "goto".

Obtenido de "<http://es.wikibooks.org/wiki/Fundamentos_de_programaci%C3%B3n/Introducci%C3%B3n_a_la_programaci%C3%B3n_estructurada>"

Programa (computación)

De Wikipedia, la enciclopedia libre

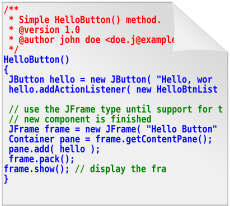
Saltar a [navegación](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#column-one#column-one), [búsqueda](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#searchInput#searchInput)

Un programa, o también llamado programa informático, programa de computación o programa de ordenador, es simplemente un conjunto de instrucciones para una [computadora](http://es.wikipedia.org/wiki/Computadora).[[1]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-pis-ch4-p132-0#cite_note-pis-ch4-p132-0) Las computadoras necesitan de los programas para funcionar, y un programa no hace nada a menos que sus instrucciones sean ejecutadas por el [procesador](http://es.wikipedia.org/wiki/Microprocesador).[[2]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-osc-ch3-p58-1#cite_note-osc-ch3-p58-1) Un programa se puede referir tanto a un programa ejecutable como a su [código fuente](http://es.wikipedia.org/wiki/C%C3%B3digo_fuente), el cual es [transformado](http://es.wikipedia.org/wiki/Proceso_de_traducci%C3%B3n_de_programas) en un ejecutable cuando es [compilado](http://es.wikipedia.org/wiki/Compilador).

Generalmente el código fuente de los programas es escrito por profesionales conocidos como [programadores](http://es.wikipedia.org/wiki/Programador). El código fuente es escrito en un lenguaje de programación que sigue uno de los siguientes dos paradigmas: imperativo o declarativo. El código fuente puede ser convertido en una imagen ejecutable por un compilador. Cuando se pide que el programa sea ejecutado, el procesador ejecuta el programa instrucción por instrucción, hasta que el programa termina.

De acuerdo a sus funciones, los programas pueden ser clasificados en [software de sistema](http://es.wikipedia.org/wiki/Software_de_sistema) y [software de aplicación](http://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica). Pueden ejecutarse muchos programas de forma simultánea en un mismo ordenador, a lo cual se le llama multitarea.

Programación

[](http://es.wikipedia.org/wiki/Imagen:CodeCmmt002.svg)

Código fuente de un programa escrito en el [lenguaje de programación Java](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java)

La [programación](http://es.wikipedia.org/wiki/Programaci%C3%B3n) es un proceso el cual consiste en escribir o editar el código fuente. Editar el código fuente significa poner a prueba, analizar y redefinir. La persona que tiene la habilidad para programar se le llama [programador](http://es.wikipedia.org/wiki/Programador) o desarrollador de software. Usualmente, al gran proceso de programación se le llama desarrollo de software. El término ingeniería de software se está volviendo popular, refiriéndose a la ingeniería como disciplina.

Paradigmas en los lenguajes de programación

Los programas se pueden clasificar por el [paradigma](http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) del [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) que se usa para producirlos. Los principales paradigmas son imperativos y declarativos.

Los programas que usan un lenguaje imperativo especifican un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo), usan declaraciones, expresiones y sentencias.[[3]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-cpl-ch4-75-2#cite_note-cpl-ch4-75-2) 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]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-cpl-ch9-213-3#cite_note-cpl-ch9-213-3)

Los programas que usan un lenguaje declarativo especifican las propiedades que la salida debe conocer y no especifica cualquier detalle de implementación. Dos amplias categorías de lenguajes declarativos son los lenguajes funcionales y los lenguajes lógicos. Los lenguajes funcionales (como [Haskell](http://es.wikipedia.org/wiki/Haskell)) no permiten asignaciones de variables no locales, así, se hacen más fácil, por ejemplo, programas como funciones matemáticas.[[4]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-cpl-ch9-213-3#cite_note-cpl-ch9-213-3) El principio detrás de los lenguajes lógicos (como [Prolog](http://es.wikipedia.org/wiki/Prolog)) es definir el problema que se quiere resolver (el objetivo) y dejar los detalles de la solución a el sistema de Prolog.[[5]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-cpl-ch10-244-4#cite_note-cpl-ch10-244-4) El objetivo es definido dando una lista de sub-objetivos. Cada sub-objetivo también se define dando una lista de sus sub-objetivos, etcétera. 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 es programa se crea puede ser por medio de texto o de forma visual. En un [lenguaje de programación visual](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_visual), los elementos son manipulados gráficamente en vez de especificarse por medio de texto.

**Compilación o interpretación de lenguajes de programación**

Si un programa está escrito en un lenguaje de programación comprensible para un humano, se le llama [código fuente](http://es.wikipedia.org/wiki/C%C3%B3digo_fuente). El código fuente se puede convertir en un [archivo ejecutable](http://es.wikipedia.org/wiki/Archivo_ejecutable) con la ayuda de un [compilador](http://es.wikipedia.org/wiki/Compilador) o también puede ser ejecutado de inmediato por medio de un [intérprete](http://es.wikipedia.org/wiki/Int%C3%A9rprete_%28inform%C3%A1tica%29).

Los programas que son compilados comúnmente son llamados ejecutables, imágenes binarias, o simplemente como [binarios](http://es.wikipedia.org/wiki/Archivo_binario), ya que la forma en que se almacena el código de los ejecutables es en [binario](http://es.wikipedia.org/wiki/Sistema_de_numeraci%C3%B3n_binario). Los compiladores se utilizan para traducir el código fuente de un lenguaje de programación, ya sea a [código objeto](http://es.wikipedia.org/wiki/C%C3%B3digo_objeto) o a [código de máquina](http://es.wikipedia.org/wiki/C%C3%B3digo_de_m%C3%A1quina). El código objeto necesita una transformación más para convertirse en código de máquina, y el código de máquina es el [código](http://es.wikipedia.org/wiki/Microc%C3%B3digo) nativo del [procesador](http://es.wikipedia.org/wiki/Microprocesador), listo para su ejecución. Un lenguaje de programación utilizado comúnmente para compilar es el [lenguaje C](http://es.wikipedia.org/wiki/C_%28lenguaje_de_programaci%C3%B3n%29).

Los programas interpretados podrían primeramente ser decodificados e inmediatamente después ejecutarse, o también puede darse el caso que se transforme a una eficiente representación intermedia para su futura ejecución. [BASIC](http://es.wikipedia.org/wiki/BASIC), [Perl](http://es.wikipedia.org/wiki/Perl), y [Python](http://es.wikipedia.org/wiki/Python) son ejemplos de lenguajes en los cuales los programas se ejecutan inmediatamente. De forma alternativa, los programas escritos en [Java](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) primeramente son compilados y almacenados en un código independiente de la máquina al cual se le llama [bytecode](http://es.wikipedia.org/wiki/Bytecode). Un intérprete llamado **máquina virtual** ejecuta dicho bytecode cuando se le solicita.

La principal desventaja de los intérpretes es que los programas se ejecutan más lentamente que si fueran compilados. El código interpretado es más lento que el código compilado porque el intérprete debe de decodificar cada sentencia cada vez que se carga y luego ejecutar dicha acción. Sin embargo, el desarrollo del software puede ser más rápido usando un intérprete porque las pruebas sobre el código fuente se llevan de forma inmediata cuando se omite la fase de compilación. Otra desventaja de los intérpretes es que el intérprete debe de estar presente en la computadora para poder ejecutar los programas, en cambio los programas hechos con compiladores no necesitan tener el compilador presente en tiempo de ejecución.

Un lenguaje de programación no es estricta y exclusivamente compilado o interpretado. La clasificación usualmente refleja el método más popular de la ejecución del lenguaje. Por ejemplo, BASIC se trata como un lenguaje interpretado y C como un lenguaje compilado, a pesar de la existencia de compiladores para BASIC e intérpretes para C.

.

Ejecución y almacenamiento de los programas

Típicamente, los programas se almacenan en la [memoria no volátil](http://es.wikipedia.org/wiki/Memoria_no_vol%C3%A1til).

**CONCLUSIÓN**

**Después de realizar este trabajo, entenderé algo más sobre la programación de computadoras, que el mismo no es una tarea troll, sino, que requiere un análisis y un reconocimiento claro y profundo de un lenguaje de programación que permita transmitir las instrucciones a la computadora.**

**La programación orientada a objetos permite la optimización del código generado gracias a que mediante técnicas de herencia, atributos estáticos entre otros permiten, que el código sea genérico de manera que sea reutilizable.**

**Mediante las técnicas aprendidas podemos establecer una solución primitiva de un problema real, tan solo con relacionarlo con objetos lógicos que serán usados para el desarrollo del software.**

**Podemos dar a conocer de una forma sencilla los mecanismos que se usan en este nivel de programación, a personas que deseen una explicación rápida y sencilla de lo que es la programación orientada a objetos.**

**Tenemos los conocimientos necesarios como para enfrentar un problema real y desarrollo en otro lenguaje de programación, pues creo la idea de que el lenguaje C es la base de la programación.**

**EGRAFÍA**

[**https://es.wikipedia.org/wiki/Lenguaje\_de\_m%C3%A1quina**](https://es.wikipedia.org/wiki/Lenguaje_de_m%C3%A1quina)

[**https://es.wikipedia.org/wiki/Programaci%C3%B3n**](https://es.wikipedia.org/wiki/Programaci%C3%B3n)

[**https://es.wikipedia.org/wiki/Lenguaje\_de\_programaci%C3%B3n**](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n)

[**https://es.wikipedia.org/wiki/Programaci%C3%B3n\_declarativa**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa)

[**https://es.wikipedia.org/wiki/Programaci%C3%B3n\_estructurada**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada)

[**https://es.wikipedia.org/wiki/Programaci%C3%B3n\_modular**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_modular)

[**https://es.wikipedia.org/wiki/Programaci%C3%B3n\_orientada\_a\_objetos**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos)

[**http://definicion.de/programacion/**](http://definicion.de/programacion/)

[**http://www.vix.com/es/btg/tech/2007/03/18/%C2%BFque-es-la-programacion-de-software**](http://www.vix.com/es/btg/tech/2007/03/18/%C2%BFque-es-la-programacion-de-software)

[**http://wiki.elhacker.net/programacion/introduccion/que-es-la-programacion**](http://wiki.elhacker.net/programacion/introduccion/que-es-la-programacion)

[**http://www.alegsa.com.ar/Dic/programacion.php**](http://www.alegsa.com.ar/Dic/programacion.php)