 LICEO COMPU-MARKET

CATEDRA: Programación

CATEDRATICO: Erick Gómez

TEMARIO:

PROGRAMACIÓN

Nombre del Estudiante: Beverly Leticia Sandoval lemus

GRADO: 5to.Bachillerato en Computación con Orientación Científica

CLAVE: 29

JORNADA: matutina

SECCION: “B”

**INDICE**

1. [**caratula**](#A)**..........................................................................................................1**
2. [**índice**](#b)**……………………………………………………………………………………………………………2**
3. [**Introducción**](#c)**…………………………………………………………………………………………….....3**
4. [**Programación**](#d)**………………………………………………………………………………………………4-5**
5. [**Lenguaje de programación**](#f)**……………………………………………………………………………5-6**
6. [**Clasificación de los lenguajes de programación**](#g)**……………………………………………7-8**
7. [**Programas y algoritmos**](#h)**………………………………………………………………………………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)**……………………4-53**
20. [**Programación e ingeniería del software**](#t)**…………………………………………….54-55**
21. [**Ciclo de vida del software**](#u)**…………………………………………………………………56-57**
22. [**Programa informático**](#v)**………………………………………………………………………….58**
23. [**Paradigmas**](#w)**………………………………………………………………………………………59-60**
24. [**Ejecución y almacenamiento de los programas**](#x)**…………………………………61-62**
25. [**Algoritmos**](#y)**…………………………………………………………………………………………63-71**
26. [**Análisis de algoritmos**](#z)**………………………………………………………………………72-73**
27. [**Lenguaje de máquina**](#maquina)**………………………………………………………………………….74**
28. [**lenguajes de programación más utilizados**](#utilizados)**…………………………………………..75**
29. [**Conclusión**](#conclusion)**…………………………………………………………………………………………..76**
30. [**E-grafía**](#egrafia)**…………………………………………………………………………………………………77**

**INTRODUCCION**

El siguiente trabajo vemos lo que es la programación realmente ya que en cada tema nos damos cuenta que la programación a tenido un gran avance. La programación ahora abarca la mayor parte de la vida, la utilizamos, en las redes sociales, desarrollo web, aplicaciones móviles, programas, Para poder programar es necesario estudiar cada código y desarrollar más allá la lógica humana.

**PROGRAMACIÓN**

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

**Historia**

Para crear un programa, y que la computadora lo interprete y ejecute las instrucciones escritas en él, debe escribirse en un lenguaje de programación. En sus inicios las computadoras interpretaban solo instrucciones en un lenguaje específico, del más bajo nivel, conocido como código máquina, siendo éste excesivamente complicado para programar. De hecho solo consiste en cadenas de números 1 y 0 (sistema binario). Para facilitar el trabajo de programación, los primeros científicos, que trabajaban en el área, decidieron reemplazar las instrucciones, secuencias de unos y ceros, por palabras o abreviaturas provenientes del inglés; las codificaron y crearon así un lenguaje de mayor nivel, que se conoce como Assembly o lenguaje ensamblador. Por ejemplo, para sumar se podría usar la letra A de la palabra inglesa add (sumar). En realidad escribir en lenguaje ensamblador es básicamente lo mismo que hacerlo en lenguaje máquina, pero las letras y palabras son bastante más fáciles de recordar y entender que secuencias de números binarios. A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método sencillo para programar. Entonces, se crearon los lenguajes de alto nivel. Mientras que una tarea tan trivial como multiplicar dos números puede necesitar un conjunto de instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará con solo una. Una vez que se termina de escribir un programa, sea en ensamblador o en algunos lenguajes de alto nivel, es necesario compilarlo, es decir, traducirlo completo a lenguaje máquina.1 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.

**Lenguaje de programación**

**De Wikipedia, la enciclopedia libre**

Con la llegada de las [computadoras](http://es.wikipedia.org/wiki/Computadora) desaparecen las secuencias de posiciones de llaves mecánicas que debían desconectarse para obtener una acción determinada, una clave conectada era un 1 y una llave desconectada era un 0. Una sucesión de llaves en cualquiera de sus dos posiciones definía una secuencia de ceros y unos (por ejemplo: 0100011010011101...) que venía a representar una instrucción o un conjunto de instrucciones ([programa](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29)) para el ordenador (o computador) en el que se estaba trabajando. A esta primera forma de especificar programas para una computadora se la denomina [lenguaje máquina o código máquina](http://es.wikipedia.org/wiki/Lenguaje_m%C3%A1quina).

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](http://es.wikipedia.org/wiki/Lenguaje_ensamblador).

Posteriormente aparecieron diferentes lenguajes de programación, los cuales reciben su denominación porque tienen una estructura [sintáctica](http://es.wikipedia.org/wiki/Sint%C3%A1ctica) similar a los lenguajes escritos por los humanos.

Concepto

Un [lenguaje](http://es.wikipedia.org/wiki/Lenguaje) de [programación](http://es.wikipedia.org/wiki/Programaci%C3%B3n) es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una [computadora](http://es.wikipedia.org/wiki/Computadora). Consiste en un conjunto de símbolos y reglas [sintácticas](http://es.wikipedia.org/wiki/Sintaxis) y [semánticas](http://es.wikipedia.org/wiki/Sem%C3%A1ntica) que definen su estructura y el significado de sus elementos y expresiones.

Aunque muchas veces se usa lenguaje de programación y [lenguaje informático](http://es.wikipedia.org/wiki/Lenguaje_inform%C3%A1tico) como si fuesen sinónimos, no tiene por qué ser así, ya que los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como, por ejemplo, el [HTML](http://es.wikipedia.org/wiki/HTML) (lenguaje para el marcado de [páginas web](http://es.wikipedia.org/wiki/P%C3%A1gina_web)).

Un lenguaje de programación permite a uno o más [programadores](http://es.wikipedia.org/wiki/Programador) especificar de *manera precisa*: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados, transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un [lenguaje](http://es.wikipedia.org/wiki/Lenguaje) que intenta estar *relativamente* próximo al lenguaje humano o natural, tal como sucede con el lenguaje [Léxico](http://es.wikipedia.org/wiki/L%C3%A9xico). Una característica relevante de los lenguajes de programación es precisamente que más de un programador puedan tener un conjunto común de instrucciones que puedan ser comprendidas entre ellos para realizar la construcción del programa de forma colaborativa.

Los procesadores usados en las computadoras son capaces de entender y actuar según lo indican programas escritos en un lenguaje fijo llamado [lenguaje de máquina](http://es.wikipedia.org/wiki/Lenguaje_de_m%C3%A1quina). Todo programa escrito en otro lenguaje puede ser ejecutado de dos maneras:

* Mediante un programa que va adaptando las instrucciones conforme son encontradas. A este proceso se lo llama *interpretar* y a los programas que lo hacen se los conoce como [intérpretes](http://es.wikipedia.org/wiki/Int%C3%A9rprete_inform%C3%A1tico).
* Traduciendo este programa al programa equivalente escrito en lenguaje de máquina. A ese proceso se lo llama *compilar* y al traductor se lo conoce como [compilador](http://es.wikipedia.org/wiki/Compilador).

**Clasificación de los lenguajes de programación**

Los lenguajes de programación se determinan según el nivel de abstracción, Según la forma de ejecución y Según el paradigma de programación que poseen cada uno de ellos y esos pueden ser:

**Según su nivel de abstracción**

**Lenguajes de bajo nivel**

Los lenguajes de bajo nivel son lenguajes de programación que se acercan al funcionamiento de una computadora. El lenguaje de más bajo nivel es, por excelencia, el [código máquina](http://es.wikipedia.org/wiki/C%C3%B3digo_m%C3%A1quina). A éste le sigue el [lenguaje ensamblador](http://es.wikipedia.org/wiki/Lenguaje_ensamblador), ya que al programar en ensamblador se trabajan con los [registros](http://es.wikipedia.org/wiki/Registros) de [memoria](http://es.wikipedia.org/wiki/Memoria) de la computadora de forma directa.

**Lenguajes de medio nivel**

Hay lenguajes de programación que son considerados por algunos expertos como lenguajes de medio nivel (como es el caso del [lenguaje C](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C)) al tener ciertas características que los acercan a los lenguajes de bajo nivel pero teniendo, al mismo tiempo, ciertas cualidades que lo hacen un lenguaje más cercano al humano y, por tanto, de alto nivel.

**Lenguajes de alto nivel**

Los lenguajes de alto nivel son normalmente fáciles de aprender porque están formados por elementos de lenguajes naturales, como el inglés. En BASIC, el lenguaje de alto nivel más conocido, los comandos como "IF CONTADOR = 10 THEN STOP" pueden utilizarse para pedir a la computadora que pare si CONTADOR es igual a 10. Por desgracia para muchas personas esta forma de trabajar es un poco frustrante, dado que a pesar de que las computadoras parecen comprender un lenguaje natural, lo hacen en realidad de una forma rígida y sistemática.

**Según la forma de ejecución**

**Lenguajes compilados**

Naturalmente, un programa que se escribe en un lenguaje de alto nivel también tiene que traducirse a un código que pueda utilizar la máquina. 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. Se requiere una corrida de compilación antes de procesar los datos de un problema.

Los compiladores son aquellos cuya función es traducir un programa escrito en un determinado lenguaje a un idioma que la computadora entienda (lenguaje máquina con código binario).

Al usar un lenguaje compilado (como lo son los lenguajes del popular Visual Studio de Microsoft), el programa desarrollado nunca se ejecuta mientras haya errores, sino hasta que luego de haber compilado el programa, ya no aparecen errores en el código

**Lenguajes interpretados**

Se puede también utilizar una alternativa diferente de los compiladores para traducir lenguajes de alto nivel. En vez de traducir el programa fuente y grabar en forma permanente el código objeto que se produce durante la corrida de compilación para utilizarlo en una corrida de producción 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 proceso de los datos. No se graba el código objeto para utilizarlo posteriormente.

La siguiente vez que se utilice una instrucción, se le debe interpretar otra vez y traducir a lenguaje máquina. Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo, cada instrucción del ciclo tendrá que volver a ser interpretado cada vez que se ejecute el 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 corrida de 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 corrida de producción por ejemplo 2057894642631201584

**Según el paradigma de programación**

Un paradigma de programación representa un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro

Atendiendo al paradigma de programación, se pueden clasificar los lenguajes en:

**Lenguajes imperativos**  **(**[BASIC](http://es.wikipedia.org/wiki/BASIC), [C](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C), [C++](http://es.wikipedia.org/wiki/C%2B%2B), [Java](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java), etc.)

**Lenguajes Funcionales**

Puros: ([Haskell](http://es.wikipedia.org/wiki/Haskell" \o "Haskell), [Miranda](http://es.wikipedia.org/wiki/Miranda_%28lenguaje%29))

Híbridos: ([Lisp](http://es.wikipedia.org/wiki/Lisp" \o "Lisp), [Scheme](http://es.wikipedia.org/wiki/Scheme)n [Ocaml](http://es.wikipedia.org/wiki/Ocaml), [ML](http://es.wikipedia.org/wiki/ML), etc)

**Lenguajes Logicos (**[Prolog](http://es.wikipedia.org/wiki/Prolog" \o "Prolog))

**Lenguajes orientados a objetos (**[Ada](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Ada), [C++](http://es.wikipedia.org/wiki/C%2B%2B) **,** [Visual FoxPro](http://es.wikipedia.org/wiki/Visual_FoxPro), [Java](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java), etc.)

**Programación**

Se conoce como **programación de** [**computadores**](http://es.wikipedia.org/wiki/Computadores) a la implementación de un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) en un determinado [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n), conformando un [programa](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29). Mientras que un algoritmo se ejecuta en una máquina abstracta que no tiene limitaciones de memoria o tiempo, un programa se ejecuta en una máquina real, que sí tiene esas limitaciones. El lenguaje de programación puede ser de alto nivel, medio nivel o bajo nivel, en función del grado de abstracción.

**Programas y algoritmos**

Un [algoritmo](http://es.wikipedia.org/wiki/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) un algoritmo. Nótese que es la secuencia de instrucciones en sí la que debe ser finita, no el número de pasos realizados como la ejecución de ellas.

Los programas suelen subdividirse en partes menores (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.

Según [Niklaus Wirth](http://es.wikipedia.org/wiki/Niklaus_Wirth) un programa está formado por [algoritmos](http://es.wikipedia.org/wiki/Algoritmo) y [estructura de datos](http://es.wikipedia.org/wiki/Estructura_de_datos).

Se han propuesto diversas técnicas de programación, cuyo objetivo es mejorar tanto el proceso de creación de software como su mantenimiento. Entre ellas se pueden mencionar las programaciones [lineal](http://es.wikipedia.org/wiki/Programaci%C3%B3n_lineal), [estructurada](http://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada), [modular](http://es.wikipedia.org/wiki/Programaci%C3%B3n_modular) y [orientada a objetos](http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos).

Compilación

El programa escrito en un [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) (comprensible por el ser humano, aunque se suelen corresponder con lenguajes formales descritos por gramáticas independientes del contexto) no es inmediatamente ejecutado en una computadora. La opción más común es compilar el programa, aunque también puede ser ejecutado mediante un [intérprete informático](http://es.wikipedia.org/wiki/Int%C3%A9rprete_inform%C3%A1tico)

El código fuente del [programa](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29) se debe someter a un proceso de [transformación](http://es.wikipedia.org/wiki/Proceso_de_traducci%C3%B3n_de_programas) para convertirse en lenguaje máquina, interpretable por el [procesador](http://es.wikipedia.org/wiki/Microprocesador). A este proceso se le llama [compilación](http://es.wikipedia.org/wiki/Compilaci%C3%B3n).

Normalmente la creación de un programa [ejecutable](http://es.wikipedia.org/wiki/Ejecutable) (un típico.exe para Microsoft Windows) conlleva dos pasos. El primer paso se llama compilación (propiamente dicho) y traduce el código fuente escrito en un [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) almacenado en un archivo a código en bajo nivel, (normalmente en código objeto no directamente al lenguaje máquina). El segundo paso se llama [enlazado](http://es.wikipedia.org/wiki/Enlazador) (del inglés *link* o *linker*) se junta el código de bajo nivel generado de todos los ficheros 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 con el sistemas operativo y traduce el [código objeto](http://es.wikipedia.org/wiki/C%C3%B3digo_objeto) a [código máquina](http://es.wikipedia.org/wiki/C%C3%B3digo_m%C3%A1quina).

Estos dos pasos se pueden mandar hacer por separado, almacenando el resultado de la fase de compilación en archivos objetos (un típico.obj para Microsoft Windows,.o para Unix), para enlazarlos posteriormente, o crear directamente el ejecutable con lo que la fase de compilación se almacena sólo temporalmente. Un programa podría tener partes escritas en varios lenguajes (generalmente C, C++ y Asm), que se podrían compilar de forma independiente y enlazar juntas para formar un único [ejecutable](http://es.wikipedia.org/wiki/Ejecutable).

Programación e [ingeniería del software](http://es.wikipedia.org/wiki/Ingenier%C3%ADa_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 tiene los siguientes pasos:

1. Reconocer la necesidad de un programa para solucionar un problema ó 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 cómo debe realizar el programa las cosas que debe hacer. 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 debería resultar inmediata.
6. 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 únicamente en escribir el código.

Objetivos de la programación

La programación de ordenadores debe perseguir tres objetivos fundamentales:

* Corrección: un programa es correcto si hace lo que debe hacer. 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 mejorar el mantenimiento del software. Cuando se acaba de escribir el código del programa, se deben buscar errores y corregirlos. Más concretamente, cuando el programa está concluido, es necesario hacerle ampliaciones o modificaciones, según la demanda de los usuarios, esta labor puede ser llevada acabo por el mismo [programador](http://es.wikipedia.org/wiki/Programador) que implementó el programa o por otros.
* Eficiencia: debe consumir la menor cantidad de recursos posible. Normalmente al hablar de eficiencia se suele hacer referencia al consumo de tiempo y/o memoria.

La eficiencia y la claridad de un programa pueden ser objetivos contrapuestos: se puede conseguir mayor claridad sacrificando parte de la eficiencia o viceversa. Pero hay que tener en cuenta que el tiempo del programador es caro, y que hoy en día el precio de los ordenadores es razonable y cada vez son más baratos.

Programas: Algoritmos para ser ejecutados por un ordenador

Un ordenador o computadora está, desde que se enciende hasta que se apaga totalmente, ejecutando un algoritmo. Por lo general, estos algoritmos son vagos y confusos para que los entienda una máquina. Una máquina no puede entender "escribe Hola Mundo!" porque no sabe lo que es "escribe" ni lo que es una letra o un espacio, ni lo que es una pantalla. En cambio, puede entender "mov eax, 0x23afb31" (escribir en la dirección de memoria eax el número 0x23afb31), aunque nosotros no. Un ordenador es solo un circuito electrónico, no funciona a base de magia ni nada por el estilo.

Debido a lo dificil que es escribir en lenguaje máquina, e incluso en ensamblador, se crearon diferentes lenguajes de programación, más o menos parecidos al inglés actual y a cómo se redacta un algoritmo. Estos lenguajes proveen de cosas tan complejas para una máquina como los bucles for. Los compiladores se encargan de traducir esos ficheros al lenguaje ensamblador que corresponda, el ensamblador de traducirlos a lenguaje máquina y el enlazador de juntar todo ese código máquina en un solo archivo, el programa. Y el microprocesador, de ir encendiendo o apagando transistores según lo que le diga el código máquina.

¿Qué instrucciones ejecuta un ordenador?

**Lenguaje de máquina**

Cada tipo de microprocesador contiene un conjunto de instrucciones que realizan ciertas operaciones sobre una o más palabras de bits; las instrucciones van también codificadas en bits. No queremos hacer aquí una discusión sobre arquitectura de ordenadores, por lo que con esto debe valer por ahora.

Se entiende que escribir sólo con dos teclas, el 0 y el 1, es incómodo. Históricamente, a la hora de diseñar un algoritmo para que el ordenador ejecutara, se escribía mediante unas etiquetas mnemotécnicas; éste fue el origen del [lenguaje ensamblador](http://es.wikipedia.org/wiki/lenguaje_ensamblador). Por ejemplo quizás en una cierta arquitectura la instrucción de borrado de memoria (Memory Clear, en inglés) corresponda al código 010. Pronto surgieron programas que leían, siguiendo el ejemplo, MC, y lo sustituían por 010.

**Lenguaje ensamblador**

El código máquina tenía dos grandes inconvenientes para los programadores:

- El primero es que se trata de unas instrucciones difíciles de recordar ya que no guardan relación con la operación que se está realizando.

- El segundo inconveniente es que puede y de hecho hay diferencias entre las instrucciones de un procesador a otro.

Todo esto ha llevado a "poner nombre" a las instrucciones de código máquina de manera que a una secuencia concreta de bits que realiza una operación se le pone un nombre sencillo que identifique la operación. Esta traducción a un lenguaje más sencillo para las personas resulta en una mayor comodidad para el programador, además el proceso de traducción inverso de lenguaje ensamblador a código máquina puede ser realizado por un sencillo programa.

Programación para seres humanos

**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.

Programacion:



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.

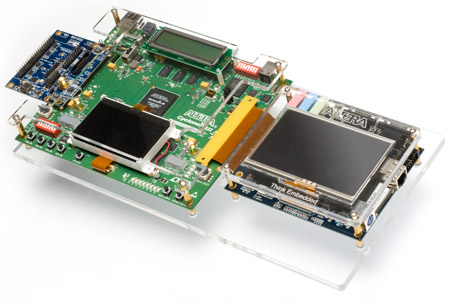
Un lenguaje de programación no es estricta y exclusivamente compilado o interpretado. La clasificación usualmente refleja el método más popular de la ejecución del lenguaje. Por ejemplo, BASIC se trata como un lenguaje interpretado y C como un lenguaje compilado, a pesar de la existencia de compiladores 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), para que luego un usuario de la [computadora](http://es.wikipedia.org/wiki/Computadora), directa o indirectamente, solicite su ejecución. Al momento de dicha solicitud, el programa se carga en la [memoria de acceso aleatorio](http://es.wikipedia.org/wiki/Memoria_de_acceso_aleatorio), por medio del software llamado [sistema operativo](http://es.wikipedia.org/wiki/Sistema_operativo), el cual puede acceder directamente al [procesador](http://es.wikipedia.org/wiki/Microprocesador). El procesador ejecuta (corre) el programa, instrucción por instrucción hasta que termina. A un programa en ejecución se le llama [proceso](http://es.wikipedia.org/wiki/Proceso_%28inform%C3%A1tica%29). Un programa puede terminar de forma normal o a causa de un error, dicho error puede ser de software o de hardware.

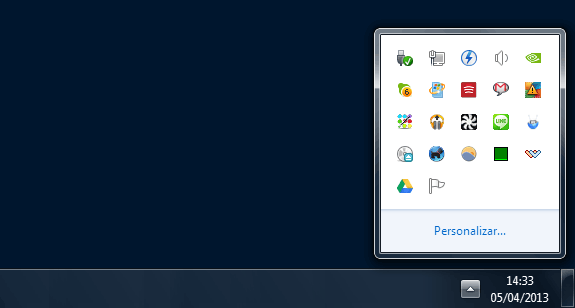
**Programas empotrados en hardware**



El [micro controlador](http://es.wikipedia.org/wiki/Microcontrolador) a la derecha de la [memoria USB](http://es.wikipedia.org/wiki/Memoria_USB) está controlada por un [firmware](http://es.wikipedia.org/wiki/Firmware) empotrado.

Algunos programas están empotrados en el [hardware](http://es.wikipedia.org/wiki/Hardware). Una [computadora](http://es.wikipedia.org/wiki/Computadora) con arquitectura de [programas almacenados](http://es.wikipedia.org/wiki/Arquitectura_Eckert-Mauchly) requiere un programa inicial almacenado en su [ROM](http://es.wikipedia.org/wiki/ROM) para [arrancar](http://es.wikipedia.org/wiki/Arranque). El proceso de arranque es para identificar e inicializar todos los aspectos del sistema, desde los [registros del procesador](http://es.wikipedia.org/wiki/Registro_%28hardware%29), [controladores de dispositivos](http://es.wikipedia.org/wiki/Controlador_de_dispositivo) hasta el contenido de la [memoria RAM](http://es.wikipedia.org/wiki/Memoria_RAM).[[6]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-osc-ch2-p30-5#cite_note-osc-ch2-p30-5) Seguido del proceso de inicialización, este programa inicial carga al [sistema operativo](http://es.wikipedia.org/wiki/Sistema_operativo) e inicializa al [contador de programa](http://es.wikipedia.org/wiki/Contador_de_programa) para empezar las operaciones normales. Independiente de la computadora, un [dispositivo de hardware](http://es.wikipedia.org/wiki/Perif%C3%A9rico) podría tener [firmware](http://es.wikipedia.org/wiki/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.[[7]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-sco-ch1-p11-6#cite_note-sco-ch1-p11-6)

**Programas cargados manualmente**



Interruptores para la carga manual en una [Data General Nova](http://es.wikipedia.org/w/index.php?title=Data_General_Nova&action=edit&redlink=1) 3

Los programas se cargaron manualmente al procesador central mediante interruptores. Una instrucción era representada por una configuración de estado abierto o cerrado de los interruptores. Después de establecer la configuración, se ejecutaba un botón de ejecución. Este proceso era repetitivo. También, historicamente los programas se cargaban manualmente mediante una [cinta de papel](http://es.wikipedia.org/w/index.php?title=Cinta_de_papel&action=edit&redlink=1) o [tarjetas perforadas](http://es.wikipedia.org/wiki/Tarjeta_perforada). Después de que el programa se cargaba, la dirección de inicio se establecía mediante interruptores y el botón de ejecución se presionaba.[[8]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-osc-ch1-p6-7#cite_note-osc-ch1-p6-7)

**Programas generados automáticamente**

La [programación automática](http://es.wikipedia.org/wiki/Programaci%C3%B3n_autom%C3%A1tica) es un estilo de [programación](http://es.wikipedia.org/wiki/Programaci%C3%B3n) que crea [código fuente](http://es.wikipedia.org/wiki/C%C3%B3digo_fuente) mediante [clases](http://es.wikipedia.org/wiki/Clase_%28inform%C3%A1tica%29) [genéricas](http://es.wikipedia.org/wiki/Programaci%C3%B3n_gen%C3%A9rica), [prototipos](http://es.wikipedia.org/wiki/Programaci%C3%B3n_basada_en_prototipos), [plantillas](http://es.wikipedia.org/w/index.php?title=Plantilla_%28programaci%C3%B3n%29&action=edit&redlink=1), [aspectos](http://es.wikipedia.org/w/index.php?title=Aspecto_%28inform%C3%A1tica%29&action=edit&redlink=1), y [generadores de código](http://es.wikipedia.org/wiki/Generaci%C3%B3n_de_c%C3%B3digo) para aumentar la productividad del [programador](http://es.wikipedia.org/wiki/Programador). El código fuente se genera con [herramientas de programación](http://es.wikipedia.org/w/index.php?title=Herramienta_de_programaci%C3%B3n&action=edit&redlink=1) tal como un [procesador de plantilla](http://es.wikipedia.org/w/index.php?title=Procesador_de_plantilla&action=edit&redlink=1) o un [IDE](http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado). La forma más simple de un generador de código fuente es un procesador [macro](http://es.wikipedia.org/wiki/Macro), tal como el [preprocesador de C](http://es.wikipedia.org/wiki/Preprocesador_de_C), que reemplaza patrones de código fuente de acuerdo a reglas relativamente simples.

Un [motor de software](http://es.wikipedia.org/w/index.php?title=Motor_de_software&action=edit&redlink=1) da de salida código fuente o [lenguaje de marcado](http://es.wikipedia.org/wiki/Lenguaje_de_marcado) que simultáneamente se vuelve la entrada de otro [proceso informático](http://es.wikipedia.org/wiki/Proceso_%28inform%C3%A1tica%29). Podemos pensar como analogía un proceso manejando a otro siendo el código máquina quemado como combustible. Los [servidores de aplicaciones](http://es.wikipedia.org/wiki/Servidor_de_aplicaciones) son motores de software que entregan aplicaciones a [computadoras cliente](http://es.wikipedia.org/wiki/Cliente_%28inform%C3%A1tica%29). Por ejemplo, un [software para wikis](http://es.wikipedia.org/wiki/Software_para_wikis) es un sevidor de aplicaciones que permite a los usuarios desarrollar [contenido dinámico](http://es.wikipedia.org/wiki/Contenido_din%C3%A1mico) ensamblado a partir de [artículos](http://es.wikipedia.org/wiki/Art%C3%ADculo). Las Wikis generan [HTML](http://es.wikipedia.org/wiki/HTML), [CSS](http://es.wikipedia.org/wiki/CSS), [Java](http://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29), y [Javascript](http://es.wikipedia.org/wiki/Javascript) los cuales son [interpretados](http://es.wikipedia.org/wiki/Int%C3%A9rprete_%28inform%C3%A1tica%29) por un [navegador web](http://es.wikipedia.org/wiki/Navegador_web).

**Ejecución simultánea**

Muchos programas pueden correr simultáneamente en la misma computadora, a lo cual se le conoce como [multitarea](http://es.wikipedia.org/wiki/Multitarea) y puede lograrse a través de mecanismos de software o de hardware. Los sistemas operativos modernos pueden correr varios programas a través del [planificador de procesos](http://es.wikipedia.org/wiki/Planificaci%C3%B3n) — un mecanismo de software para [conmutar](http://es.wikipedia.org/w/index.php?title=Conmutaci%C3%B3n_de_contexto&action=edit&redlink=1) con frecuencia la cantidad de procesos del procesador de modo que los usuarios puedan [interactuar](http://es.wikipedia.org/wiki/Tiempo_compartido_%28inform%C3%A1tica%29) con cada programa mientras estos están corriendo.[[9]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-osc-ch4-100-8#cite_note-osc-ch4-100-8) 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.[[10]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-mcore-9#cite_note-mcore-9)

Categorías funcionales

Los programas se pueden categorizar según líneas funcionales. Estas categorías funcionales son [software de sistema](http://es.wikipedia.org/wiki/Software_de_sistema) y [software de aplicación](http://es.wikipedia.org/wiki/Software_de_aplicaci%C3%B3n). El software de sistema incluye al [sistema operativo](http://es.wikipedia.org/wiki/Sistema_operativo) el cual acopla el [hardware](http://es.wikipedia.org/wiki/Hardware) con el software de aplicación.[[11]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-osc-ch1-1-10#cite_note-osc-ch1-1-10) El propósito del sistema operativo es proveer un ambiente en el cual el software de aplicación se ejecuta de una manera conveniente y eficiente.[[11]](http://es.wikipedia.org/wiki/Programa_%28computaci%C3%B3n%29#cite_note-osc-ch1-1-10#cite_note-osc-ch1-1-10) Además del sistema operativo, el software de sistema incluye [programas utilitarios](http://es.wikipedia.org/w/index.php?title=Programa_utilitario&action=edit&redlink=1) que ayudan a manejar y configurar la computadora. Si un programa no es software de sistema entonces es software de aplicación. El [middleware](http://es.wikipedia.org/wiki/Middleware) también es un software de aplicación que acopla el software de sistema con la [interfaz de usuario](http://es.wikipedia.org/wiki/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.

Lenguaje ensamblador

**De Wikipedia, la enciclopedia libre**

Saltar a [navegación](http://es.wikipedia.org/wiki/Lenguaje_ensamblador#column-one#column-one), [búsqueda](http://es.wikipedia.org/wiki/Lenguaje_ensamblador#searchInput#searchInput)

[Lenguaje de máquina](http://es.wikipedia.org/wiki/Lenguaje_de_m%C3%A1quina) del [Intel 8088](http://es.wikipedia.org/wiki/Intel_8088). El código de máquina se resalta en rojo, el equivalente en lenguaje assembler en magenta, y las direcciones de memoria donde se encuentra el código, en azul.

El lenguaje ensamblador es un tipo de lenguaje de bajo nivel utilizado para escribir [programas informáticos](http://es.wikipedia.org/wiki/Programas_inform%C3%A1ticos), y constituye la representación más directa del [código máquina](http://es.wikipedia.org/wiki/Lenguaje_de_m%C3%A1quina) específico para cada [arquitectura de computadoras](http://es.wikipedia.org/wiki/Arquitectura_de_computadoras) legible por un programador.

Fue usado ampliamente en el pasado para el desarrollo de [software](http://es.wikipedia.org/wiki/Software), pero actualmente sólo se utiliza en contadas ocasiones, especialmente cuando se requiere la manipulación directa del [hardware](http://es.wikipedia.org/wiki/Hardware) o se pretenden rendimientos inusuales de los equipos.

Características Programar en lenguaje ensamblador es difícil de aprender, entender, leer, escribir, depurar y mantener, por eso surgió la necesidad de los lenguajes compilados.

* A pesar de perder rendimiento en un proceso de compilación, en la actualidad la mayoría de las computadoras son suficientemente rápidas.
* El lenguaje ensamblador no es portable.
* Programar en lenguaje ensamblador lleva mucho tiempo.
* Los programas hechos en lenguaje ensamblador son generalmente más rápídos. Al programar cuidadosamente en lenguaje ensamblador se pueden crear programas de 5 a 10 veces más rápidos que con lenguajes de alto nivel.
* Los programas hechos en lenguaje ensamblador generalmente ocupan menos espacio. Un buen programa en lenguaje ensamblador puede ocupar casi la mitad de espacio que su contraparte en lenguaje de alto nivel.
* Con el lenguaje ensamblador se pueden crear segmentos de código imposibles de formar en un lenguaje de alto nivel.

Ensambladores

Un [ensamblador](http://es.wikipedia.org/wiki/Ensamblador) crea [código objeto](http://es.wikipedia.org/wiki/C%C3%B3digo_objeto) traduciendo instrucciones [mnemónicas](http://es.wikipedia.org/wiki/Mnem%C3%B3nico) a códigos operativos, e interpretando los nombres simbólicos para direcciones de memoria y otras entidades. El uso de referencias simbólicas es una característica básica de los ensambladores, evitando tediosos cálculos y direccionamiento manual después de cada modificación del programa. La mayoría de los ensambladores también incluyen facilidades para crear [macros](http://es.wikipedia.org/wiki/Macro), a fin de generar series de instrucciones cortas que se ejecutan en tiempo real, en lugar de utilizar subrutinas.[[1]](http://es.wikipedia.org/wiki/Lenguaje_ensamblador#cite_note-0#cite_note-0)

Los ensambladores son por lo general más fáciles de programar que los [compiladores](http://es.wikipedia.org/wiki/Compilador) de lenguajes de alto nivel, y han estado disponibles desde la década de [1950](http://es.wikipedia.org/wiki/1950). Los ensambladores modernos, especialmente para arquitecturas basadas en [RISC](http://es.wikipedia.org/wiki/RISC), como por ejemplo [MIPS](http://es.wikipedia.org/wiki/MIPS), [SPARC](http://es.wikipedia.org/wiki/SPARC) y [PA-RISC](http://es.wikipedia.org/wiki/PA-RISC) optimizan las instrucciones para explotar al máximo la eficiencia de [segmentación](http://es.wikipedia.org/wiki/Segmentaci%C3%B3n_%28inform%C3%A1tica%29)[[2]](http://es.wikipedia.org/wiki/Lenguaje_ensamblador#cite_note-1#cite_note-1) del [CPU](http://es.wikipedia.org/wiki/CPU).

Los ensambladores de alto nivel ofrecen posibilidades de abstracción que incluyen:

* Control avanzado de estructuras.
* Procedimientos de alto nivel, declaración de funciones.
* Tipos de datos que incluyen estructuras, registros, uniones, clases y conjuntos.
* Sofisticado procesamiento de macros.

**Lenguaje**

Un programa escrito en lenguaje ensamblador consiste en una serie de instrucciones que corresponden al flujo de órdenes ejecutables que pueden ser cargadas en la memoria de una computadora. Por ejemplo, un procesador [x86](http://es.wikipedia.org/wiki/X86) puede ejecutar la siguiente instrucción [binaria](http://es.wikipedia.org/wiki/Binario) como se expresa en código de máquina:

* Binario: 10110000 01100001 (Hexadecimal: 0xb061)

La representación equivalente en lenguaje ensamblador es más fácil de recordar:

* MOV al, 061h

Esta instrucción significa:

* Mueva el valor [hexadecimal](http://es.wikipedia.org/wiki/Hexadecimal) 61 (97 [decimal](http://es.wikipedia.org/wiki/Decimal)) al registro "al".

El mnemónico "mov" es un *código de operación* u "[opcode](http://es.wikipedia.org/wiki/Opcode" \o "Opcode)", elegido por los diseñadores de la colección de instrucciones para abreviar "move" (mover). El opcode es seguido por una lista de argumentos o *parámetros*, completando una instrucción de ensamblador típica.

La transformación del lenguaje ensamblador en código máquina la realiza un programa ensamblador, y la traducción inversa la puede efectuar un desensamblador. A diferencia de los lenguajes de alto nivel, aquí hay usualmente una correspondencia 1 a 1 entre las instrucciones simples del ensamblador y el lenguaje de máquina. Sin embargo, en algunos casos, un ensamblador puede proveer "pseudo instrucciones" que se expanden en un código de máquina más extenso a fin de proveer la funcionalidad necesaria. Por ejemplo, para un código máquina condicional como "si X mayor o igual que" , un ensamblador puede utilizar una pseudoinstrucción al grupo "haga si menor que" , y "si = 0" sobre el resultado de la condición anterior. Los ensambladores más completos también proveen un rico lenguaje de macros que se utiliza para generar código más complejo y secuencias de datos.

Cada [arquitectura de computadoras](http://es.wikipedia.org/wiki/Arquitectura_de_computadoras) tiene su propio lenguaje de máquina, y en consecuencia su propio lenguaje ensamblador. Los ordenadores difieren en el tipo y número de operaciones que soportan; también pueden tener diferente cantidad de registros, y distinta representación de los tipos de datos en memoria. Aunque la mayoría de las computadoras son capaces de cumplir esencialmente las mismas funciones, la forma en que lo hacen difiere, y los respectivos lenguajes ensambladores reflejan tal diferencia.

Pueden existir múltiples conjuntos de mnemónicos o [sintáxis](http://es.wikipedia.org/w/index.php?title=Sint%C3%A1xis&action=edit&redlink=1) de lenguaje ensamblador para un mismo conjunto de instrucciones, instanciados típicamente en diferentes programas ensamblador. En estos casos, la alternativa más popular es la provista por los fabricantes, y usada en los manuales del programa.

Código máquina (o [lenguaje de máquina](http://es.wikipedia.org/wiki/Lenguaje_de_m%C3%A1quina))

El lenguaje de máquina está formado por instrucciones sencillas, que -dependiendo de la estructura del procesador- pueden especificar:

* Registros específicos para operaciones aritméticas, direccionamiento o control de funciones.
* Posiciones de memoria específicas (*offset*).
* Modos de direccionamiento usados para interpretar operandos.

Las operaciones más complejas se realizan combinando estas instrucciones sencillas, que pueden ser ejecutadas secuencialmente o mediante instrucciones de control de flujo.

Las operaciones disponibles en la mayoría de los conjuntos de instrucciones incluye:

* mover
  + llenar un registro con un valor constante
  + mover datos de una posición de memoria a un registro o viceversa
  + escribir y leer datos de dispositivos
* computar
  + sumar, restar, multiplicar o dividir los valores de dos registros, colocando el resultado en uno de ellos o en otro registro
  + realizar operaciones binarias, incluyendo operaciones lógicas (AND/OR/XOR/NOT)
  + comparar valores entre registros (mayor, menor, igual)
* afectar el flujo del programa
  + saltar a otra posición en el programa y ejecutar instrucciones allí
  + saltar si se cumplen ciertas condiciones (IF)
  + saltar a otra posición, pero guardar el punto de salida para retornar (CALL, llamada a subrutinas)

Algunas computadoras incluyen instrucciones complejas dentro de sus capacidades. Una sola instrucción compleja hace lo mismo que en otras computadoras puede requerir una larga serie de instrucciones, por ejemplo:

* salvar varios registros en la pila de una sola vez
* mover grandes bloques de memoria
* operaciones aritméticas complejas o de punto flotante ([seno](http://es.wikipedia.org/wiki/Seno), [coseno](http://es.wikipedia.org/wiki/Coseno), [raíz cuadrada](http://es.wikipedia.org/wiki/Ra%C3%ADz_cuadrada) )

El nivel de lenguaje ensamblador tiene aspectos importantes de los niveles de microarquitectura, en los cuales se encuentra (ISA y sistema operativo) estos dos se utilizan para la traducción en lugar de la interpretación. Algunas características del lenguaje se describen a continuación Los programas que sirven para traducir algún programa para el usuario se llama traductores, el lenguaje en que esta escrito el programa original se llama lenguaje fuente, el lenguaje original que sea modificado se llama lenguaje objeto.

Se usa la traducción cuando se cuenta con un procesador (ya sea hardware o un intérprete) para el lenguaje objeto pero no para el lenguaje fuente, Si la traducción se realiza correctamente, la ejecución del programa traducido dará exactamente los mismos resultados que habría dado la ejecución del programa fuente. Hay dos diferencias entre traducción e interpretación, en la traducción no se ejecuta directamente el programa original, en el lenguaje fuente se convierte en un programa equivalente llamado programa objeto o programa binario ejecutable y este funciona solo cuando se ha acabado la traducción.

El código máquina, un simple patrón de [bits](http://es.wikipedia.org/wiki/Bit), es hecho legible reemplazando *valores crudos* por símbolos denominados [mnemónicos](http://es.wikipedia.org/wiki/Mnem%C3%B3nico). Se inventó para facilitar la tarea de los primeros programadores que hasta ese momento tenían que escribir directamente en código binario. Inicialmente el código de ceros y unos (el programa) debía introducirse en una tarjeta perforada. La posición ocupada por cada punto equivalía a un "1" o a un "0" según hubiera o no una perforación. Lo cual suponía una forma casi idéntica en la que hoy se escriben los datos binaros en soportes tales como los CDs y DVDs.

Mientras que una computadora reconoce la [instrucción de máquina](http://es.wikipedia.org/wiki/Instrucci%C3%B3n_de_m%C3%A1quina) IA-32

10110000 01100001

para los [programadores](http://es.wikipedia.org/wiki/Programador) de microprocesadores [x86](http://es.wikipedia.org/wiki/X86) es mucho más fácil reconocer dicha instrucción empleando lenguaje ensamblador :

movb 0x61,%al

(que significa mover el valor [hexadecimal](http://es.wikipedia.org/wiki/Hexadecimal) 61 (97 [decimal](http://es.wikipedia.org/wiki/Decimal)) al [registro](http://es.wikipedia.org/wiki/Registro) 'al'.)

Cada instrucción de la máquina se transforma en una única instrucción en código simbólico.

Pero además, para mejorar la legibilidad del programa, el código simbólico introduce instrucciones adicionales, que no corresponden a ninguna instrucción de la máquina y que proporcionan información. Se llaman "pseudoinstrucciones".

El código simbólico puede parecer de difícil acceso, pero es más fácil de recordar e interpretar que el [binario](http://es.wikipedia.org/wiki/Binario) o el [hexadecimal](http://es.wikipedia.org/wiki/Hexadecimal).

Los lenguajes simbólicos no resuelven definitivamente el problema de cómo programar un ordenador de la manera más sencilla posible. Para utilizarlos, hay que conocer a fondo el [microprocesador](http://es.wikipedia.org/wiki/Microprocesador), los registros de trabajo de que dispone, la estructura de la memoria, y muchas cosas más.

Además, el lenguaje ensamblador está muy ligado al [microprocesador](http://es.wikipedia.org/wiki/Microprocesador) para que sea posible escribir programas independientes de la máquina en que van a ejecutarse.

Este **código simbólico** no puede ser ejecutado directamente por un [ordenador](http://es.wikipedia.org/wiki/Ordenador), por lo que es preciso traducirlo previamente. Pero la traducción es un proceso mecánico y repetitivo, que se presta a su realización por un programa de ordenador.

Los programas que traducen código simbólico al [lenguaje de máquina](http://es.wikipedia.org/wiki/Lenguaje_de_m%C3%A1quina) se llaman [ensambladores](http://es.wikipedia.org/wiki/Ensamblador) ("assembler", en inglés), porque son capaces de ensamblar el programa traducido a partir de varias piezas, procedimientos o subrutinas a código binario ("1" y "0") que entiende el procesador.

**Programas y algoritmos**

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**

La programación declarativa, en contraposición a la programación imperativa, es un paradigma de programación que está basado en el desarrollo de programas especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica a la computadora qué es lo que se desea obtener o qué es lo que se está buscando). No existen asignaciones destructivas, y las variables son utilizadas con transparencia referencial.

**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..

**Curry (lenguaje de programación)**

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.

**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)

 [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)

 [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**

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**

(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 Action Script 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 Action Script 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**

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 , 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 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. 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 sub objetivo 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.

|  |
| --- |
| 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.

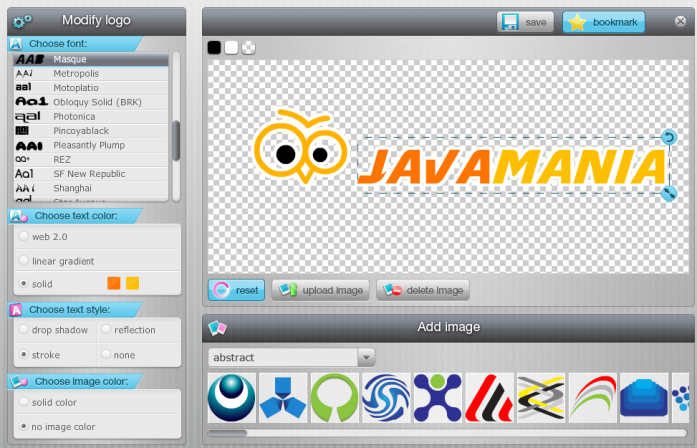
**Programas cargados manualmente**

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

**Programas generados automáticamente**

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

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

****

**Ejecución simultánea**

Muchos programas pueden ejecutarse simultáneamente en la misma computadora, hecho al cual se lo conoce como multitarea, pudiéndose lograr mediante mecanismos de software o de hardware. Los sistemas operativos modernos pueden ejecutar varios programas a través del planificador de procesos — un mecanismo de software para conmutar con frecuencia la cantidad de procesos del procesador de modo que los usuarios puedan interactuar con cada programa mientras estos están corriendo.11 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.

**Categorías funcionales**

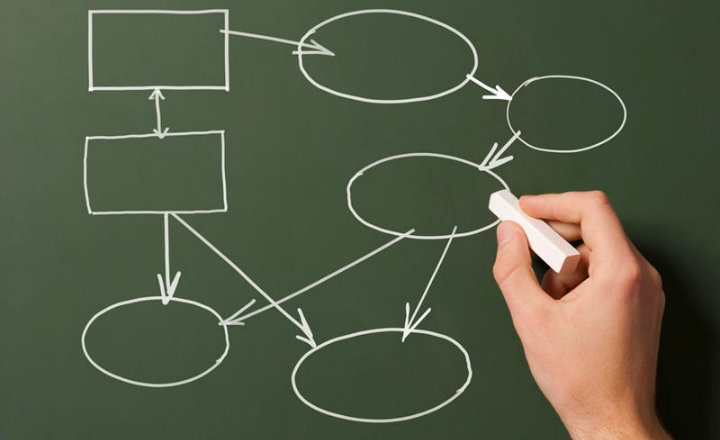
Los programas se pueden categorizar aplicando criterios funcionales. Estas categorías funcionales son software de sistema y software de aplicación. El software de sistema incluye al sistema operativo el cual acopla el hardware con el software de aplicación.13 El propósito del sistema operativo es proveer 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.

**ALGORITMOS**

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

Algunos ejemplos en matemática son el algoritmo de multiplicación, para calcular el producto, el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para obtener el máximo común divisor de dos enteros positivos, o el método de Gauss para resolver un sistema de ecuaciones lineales.

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



**Definición formal**

En general, no existe ningún consenso definitivo en cuanto a la definición formal de algoritmo. Muchos autores los señalan como listas de instrucciones para resolver un [cálculo](https://es.wikipedia.org/wiki/C%C3%A1lculo) o un [problema abstracto](https://es.wikipedia.org/wiki/Problema_abstracto), es decir, que un número finito de pasos convierten los datos de un problema (entrada) en una solución (salida).[1](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Brassard-1) [2](https://es.wikipedia.org/wiki/Algoritmo#cite_note-rae-2) [3](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Cormen-3) [4](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Grimaldi-4) [5](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Johnsonbaugh-5) [6](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Reynolds-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](https://es.wikipedia.org/wiki/Criba_de_Erat%C3%B3stenes) que nunca termine de calcular números primos no deja de ser un algoritmo.[7](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Gurevich-7)

A lo largo de la historia varios autores han tratado de definir formalmente a los algoritmos utilizando modelos matemáticos. Esto fue realizado por [Alonzo Church](https://es.wikipedia.org/wiki/Alonzo_Church) en 1936 con el concepto de "calculabilidad efectiva" basada en su [cálculo lambda](https://es.wikipedia.org/wiki/C%C3%A1lculo_lambda) y por [Alan Turing](https://es.wikipedia.org/wiki/Alan_Turing) basándose en la [máquina de Turing](https://es.wikipedia.org/wiki/M%C3%A1quina_de_Turing). Los dos enfoques son equivalentes, en el sentido en que se pueden resolver exactamente los mismos problemas con ambos enfoques.[8](https://es.wikipedia.org/wiki/Algoritmo#cite_note-savage-8) [9](https://es.wikipedia.org/wiki/Algoritmo#cite_note-sipser-9) Sin embargo, estos modelos están sujetos a un tipo particular de datos como son números, símbolos o [gráficas](https://es.wikipedia.org/wiki/Grafo) mientras que, en general, los algoritmos funcionan sobre una vasta cantidad de [estructuras de datos](https://es.wikipedia.org/wiki/Estructura_de_datos).[3](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Cormen-3) [1](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Brassard-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](https://es.wikipedia.org/wiki/Algoritmo_paralelo):

**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](https://es.wikipedia.org/wiki/L%C3%B3gica_matem%C3%A1tica) 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](https://es.wikipedia.org/wiki/M%C3%A9todo_de_Newton) y la [eliminación de Gauss-Jordan](https://es.wikipedia.org/wiki/Eliminaci%C3%B3n_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](https://es.wikipedia.org/wiki/Algoritmo#cite_note-Dershowitz-10) En particular es posible considerar una cuarta propiedad que puede ser usada para validar la [tesis de Church-Turing](https://es.wikipedia.org/wiki/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)

**Medios de expresión de un algoritmo**

Los algoritmos pueden ser expresados de muchas maneras, incluyendo al [lenguaje natural](https://es.wikipedia.org/wiki/Lenguaje_natural), [pseudocódigo](https://es.wikipedia.org/wiki/Pseudoc%C3%B3digo), [diagramas de flujo](https://es.wikipedia.org/wiki/Diagramas_de_flujo) y [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) 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:

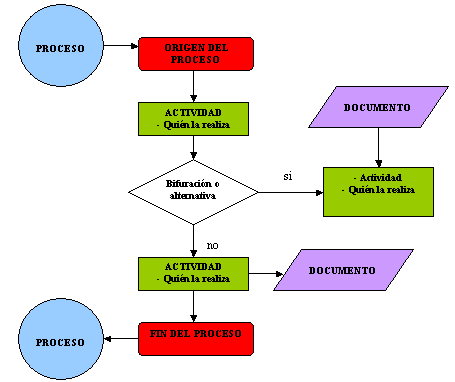
1. **Descripción de alto nivel**. Se establece el problema, se selecciona un modelo matemático y se explica el algoritmo de manera verbal, posiblemente con ilustraciones y omitiendo detalles.
2. **Descripción formal**. Se usa pseudocódigo para describir la secuencia de pasos que encuentran la solución.
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](https://es.wikipedia.org/wiki/Teorema) que demuestre que el algoritmo es correcto, un análisis de complejidad o ambos.

**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.

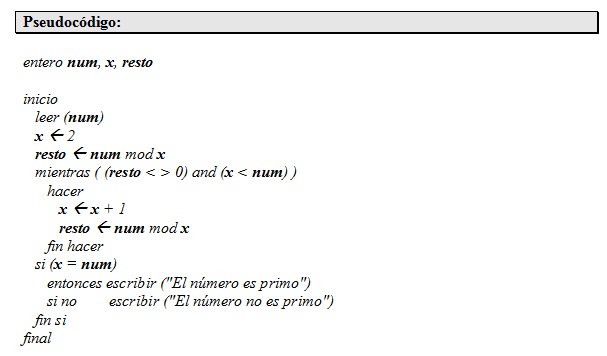


**Pseudocódigo**

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.

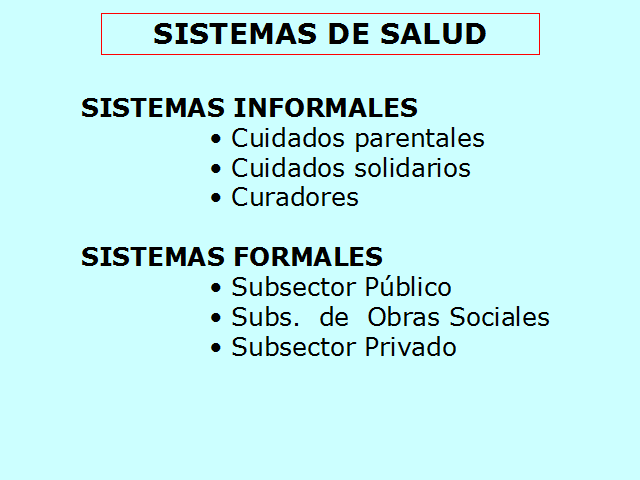
El pseudocódigo está pensado para facilitar a las personas el entendimiento de un algoritmo, y por lo tanto puede omitir detalles irrelevantes que son necesarios en una implementación. Programadores diferentes suelen utilizar convenciones distintas, que pueden estar basadas en la sintaxis de lenguajes 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:

Mediante una sentencia de asignación.

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).

**Algoritmos como funciones**

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} } {\displaystyle f:\mathbf {N} \to \mathbf {N} } donde cada número natural es la codificación de un problema o de una solución.

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

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

**Análisis de algoritmos**

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

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

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

**Ejemplo de algoritmo**

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

**Algoritmo de Euclides**

El algoritmo de Euclides es un método antiguo y eficaz para calcular el máximo común divisor (MCD). Fue originalmente descrito por Euclides en su obra Elementos. El algoritmo de Euclides extendido es una ligera modificación que permite además expresar al máximo común divisor como una combinación lineal. Este algoritmo tiene aplicaciones en diversas áreas como álgebra, teoría de números y ciencias de la computación, entre otras. Con unas ligeras modificaciones suele ser utilizado en computadoras electrónicas debido a su gran eficiencia.

**Algoritmo original de Euclides**

En la concepción griega de la matemática, los números se entendían como magnitudes geométricas. Un tema recurrente en la geometría griega es el de la conmensurabilidad de dos segmentos: dos segmentos (números) AB y CD son conmensurables cuando existe un tercer segmento PQ el cual cabe exactamente un número entero de veces en los primeros dos, es decir, PQ «mide» (mensura: medida) a los segmentos AB y CD.

No cualquier par de segmentos es conmensurable, como encontraron los pitagóricos cuando establecen que el lado y la diagonal de un cuadrado no son conmensurables, pero en el caso de dos segmentos conmensurables se desea hallar la mayor medida común posible.

Euclides describe en la proposición VI I.2 de sus Elementos un método que permite hallar la mayor medida común posible de dos números (segmentos) que no sean primos entre sí, aunque de acuerdo a la época tal método se explica en términos geométricos, lo que se ilustra en la siguiente transcripción.

Para encontrar la máxima medida común de dos números que no sean primos entre sí.

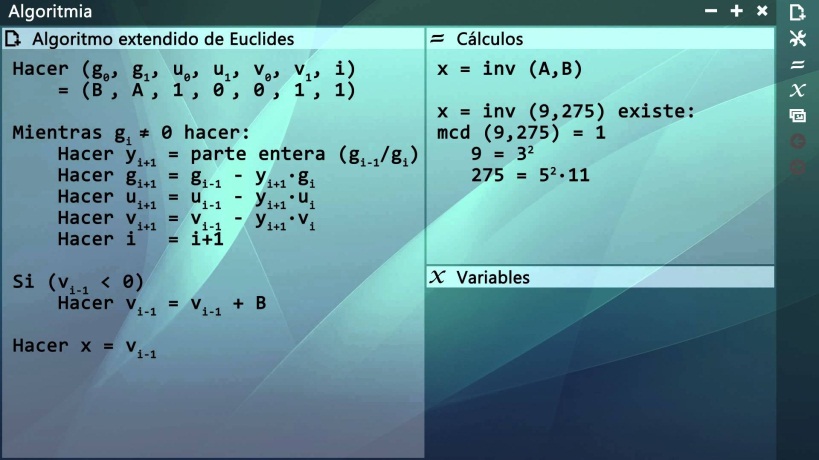
Sean AB y CD los dos números que no son primos uno al otro. Se necesita entonces encontrar la máxima medida común de AB y CD.

Si CD mide AB entonces es una medida común puesto que CD se mide a sí mismo. Y es manifiesto que también es la mayor medida pues nada mayor a CD puede medir a CD. Pero si CD no mide a AB entonces algún número quedará de AB y CD, el menor siendo continuamente restado del mayor y que medirá al número que le precede. Porque una unidad no quedará pues si no es así, AB y CD serán primos uno del otro [Prop. VII.1], lo cual es lo contrario de lo que se supuso.

Por tanto, algún número queda que medirá el número que le precede. Y sea CD midiendo BE dejando EA menor que sí mismo y sea EA midiendo DF dejando FC menor que sí mismo y sea FC medida de AE. Entonces, como FC mide AE y AE mide DF, FC será entonces medida de DF. Y también se mide a sí mismo. Por tanto también medirá todo CD. Y CD mide a BE. Entonces CF mide a BE y también mide a EA. Así mide a todo BA y también mide a CD. Esto es, CF mide tanto a AB y CD por lo que es una medida común de AB y CD.

Afirmo que también es la mayor medida común posible porque si no lo fuera, entonces un número mayor que CF mide a los números AB y CD, sea éste G. Dado que G mide a CD y CD mide a BE, G también mide a BE. Además, mide a todo BA por lo que mide también al residuo AE. Y AE mide a DF por lo que G también mide a DF. Mide también a todo DC por lo que mide también al residuo CF, es decir el mayor mide al menor, lo cual es imposible.

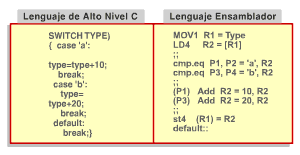
Por tanto, ningún número mayor a CF puede medir a los números AB y CD. Entonces CF es la mayor medida común de AB y CD, lo cual se quería demostrar.



Dados dos segmentos AB y CD (con AB>CD), restamos CD de AB tantas veces como sea posible. Si no hay residuo, entonces CD es la máxima medida común.

Si se obtiene un residuo EA, éste es menor que CD y podemos repetir el proceso: restamos EA tantas veces como sea posible de CD. Si al final no queda un residuo, EA es la medida común. En caso contrario obtenemos un nuevo residuo FC menor a EA.

El proceso se repite hasta que en algún momento no se obtiene residuo. Entonces el último residuo obtenido es la mayor medida común.

El hecho de que los segmentos son conmesurables es clave para asegurar que el proceso termina tarde o temprano.

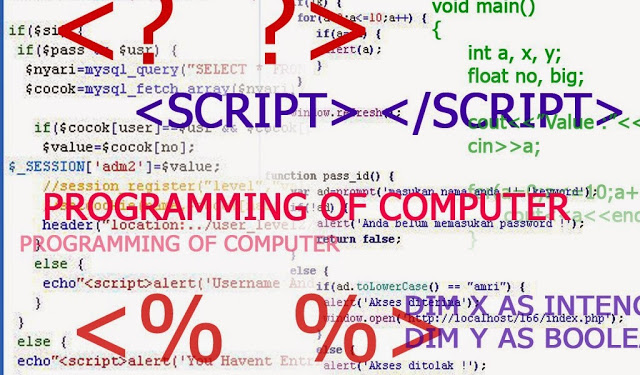
**Lenguaje de máquina**

El lenguaje de máquina o código máquina es el sistema de códigos directamente interpretable por un circuito micro programable, como el microprocesador de una computadora o el micro controlador de un autómata. Este lenguaje está compuesto por un conjunto de instrucciones que determinan acciones a ser tomadas por la máquina. Un programa consiste en una cadena de estas instrucciones más un conjunto de datos sobre el cual se trabaja. Estas instrucciones son normalmente ejecutadas en secuencia, con eventuales cambios de flujo causados por el propio programa o eventos externos. El lenguaje de máquina es específico de la arquitectura de la máquina, aunque el conjunto de instrucciones disponibles pueda ser similar entre arquitecturas distintas.

Los circuitos micro programables son digitales, lo que significa que trabajan con dos únicos niveles de tensión. Dichos niveles, por abstracción, se simbolizan con los números 0 y 1, por eso el lenguaje de máquina sólo utiliza dichos signos. Esto permite el empleo de las teorías del álgebra booleana y del sistema binario en el diseño de este tipo de circuitos y en su programación.

Este concepto es el núcleo de las puertas lógicas, las cuales son, por su parte, los ladrillos con que se construyen sistemas lógicos cada vez más complejos. Shannon utilizaba el relé como dispositivo físico de conmutación en sus redes, dado que el relé, a igual que una lámpara eléctrica, posee dos estados: activado (encendido) o desactivado (apagado).

**LENGUAJES DE PROGRAMACION MÁS UTILIZADOS**

****

**Conclusión**

**En el trabajo anterior pudimos apreciar muchas cosas las cuales algunas personas desconocemos ya que solo nos basamos en las partes principales de la programación sin darnos cuenta que existen muchas cosas más dándole seguimiento al tema.**

**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)