Es realmente **común** el diseño de lenguajes que soporten múltiples paradigmas de programación. Estos lenguajes son aquellos que soportan **al menos dos paradigmas**. Dentro de esta categoría podemos encontrar nuevas caras y viejos conocidos:

* **Scala**: Imperativo, orientado a objetos, funcional, genérico y concurrente
* **Erlang**: Funcional, concurrente y distribuido
* **Perl**: Imperativo, orientado a objetos y funcional
* **PHP**: Imperativo, orientado a objetos, funcional y reflexivo
* **JavaScript**: Imperativo, orientado a objetos (prototipos) y funcional
* **Java**: Imperativo, orientado a objetos, reflexivo y genérico
* **Python y Ruby**: Imperativo, orientado a objetos, reflexivo y funcional
* **C++**: Imperativo, orientado a objetos, funcional y genérico
* **C#**: Imperativo, orientado a objetos, funcional (lambda), reflexivo y genérico

Estos son algunos ejemplos, existen lenguajes como [Oz](http://www.mozart-oz.org/) que soporta **nueve**

Programación a nivel de valores

La **programación a nivel de valores** es unos de los dos paradigmas contrastantes identificados por [John Backus](https://es.wikipedia.org/wiki/John_Backus) en su trabajo sobre los [Programas como objetos matemáticos](https://es.wikipedia.org/wiki/Programas_como_objetos_matem%C3%A1ticos), siendo el otro la [programación a nivel funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_a_nivel_funcional). El término inicialmente utilizado por Backus fue el de programación a nivel de objetos, pero en la actualidad ese término traería confusión con la [programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos).

Los programas a nivel de valores describen como combinar diferentes *valores* (por ejemplo, números, caracteres, etc.) para formar nuevos valores hasta obtener el *resultado final*. Los nuevos valores se obtienen como resultado de la aplicación de operaciones que transforman valores en otros valores, como por ejemplo, la suma, la concatenación, la inversión de matrices, etc.

Los lenguajes que siguen el estilo de [von Neumann](https://es.wikipedia.org/wiki/Von_Neumann) son de nivel de valores: las expresiones a la derecha de una asignación tienen por objeto la creación del nuevo valor a asignar.

Relación con los tipos de datos[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_a_nivel_de_valores&action=edit&section=1" \o "Editar sección: Relación con los tipos de datos)]

El enfoque de programación a nivel de valores o imperativa se presta para el estudio de los valores bajo las operaciones de formación de valores y de sus propiedades algebraicas.

Relación con los lenguajes basados en el cálculo lambda[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_a_nivel_de_valores&action=edit&section=2" \o "Editar sección: Relación con los lenguajes basados en el cálculo lambda)]

Según este punto de vista, los lenguajes basados en el [cálculo lambda](https://es.wikipedia.org/wiki/C%C3%A1lculo_lambda) (tales como [Lisp](https://es.wikipedia.org/wiki/Lisp" \o "Lisp), [ISWIM](https://es.wikipedia.org/wiki/ISWIM), y [Scheme](https://es.wikipedia.org/wiki/Scheme" \o "Scheme)) son lenguajes a nivel de valores, si bien su diseño no los restringe a ello.

Por ejemplo, en una definición típica en el cálculo lambda de la forma **f** = λ*x*.**E** la variable *x* así como la expresión **E** denotan valores. Típicamente, **E** es una expresión que aplica funciones de formación de valores a variables y constantes.

Programación a nivel funcional

La **programación a nivel funcional** es unos de los dos [paradigmas](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) contrastantes identificados por [John Backus](https://es.wikipedia.org/wiki/John_Backus) en su trabajo sobre los [Programas como objetos matemáticos](https://es.wikipedia.org/wiki/Programas_como_objetos_matem%C3%A1ticos), siendo el otro la [programación a nivel de valores](https://es.wikipedia.org/wiki/Programaci%C3%B3n_a_nivel_de_valores).

En su discurso de aceptación del [Premio Turing](https://es.wikipedia.org/wiki/Premio_Turing) en [1977](https://es.wikipedia.org/wiki/1977), Backus describió lo que considera como la necesidad de un cambio a una filosofía diferente en el diseño de [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n):

"*Pareciera existir un problema en el diseño de los lenguajes de programación. Cada nuevo lenguaje incorpora, luego de algo de limpieza, todas las características de los lenguajes anteriores más algunas otras. [...] Cada nuevo lenguaje presenta nuevas características de moda... pero el hecho es que pocos lenguajes hacen que la tarea de programar sea más económica, o más segura como para justificar el costo de producirlo y aprender a utilizarlo*."

El lenguaje de programación [FP](https://es.wikipedia.org/wiki/FP_(lenguaje_de_programaci%C3%B3n)) fue el primer [lenguaje](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) diseñado específicamente para dar soporte al estilo de programación a nivel funcional.

Un programa de nivel funcional no necesita la noción de **variable,** dado que las variables, que son elemento esencial en las definiciones a nivel de valores no hacen falta en el nivel funcional.

En el estilo de programación de nivel funcional los programas se escriben como combinación de otros programas con la ayuda de los **operaciones de construcción de programas** o funcionales.

Bajo este enfoque los programas, con los funcionales como operadores, forman un [espacio matemático](https://es.wikipedia.org/wiki/Programas_como_objetos_matem%C3%A1ticos).

Otra ventaja potencial de este enfoque es la posibilidad de restringirse únicamente a las [funciones estrictas](https://es.wikipedia.org/w/index.php?title=Funci%C3%B3n_estricta&action=edit&redlink=1) y asociarles un mecanismo de [evaluación por valor](https://es.wikipedia.org/w/index.php?title=Evaluaci%C3%B3n_por_valor&action=edit&redlink=1). que es el más sencillo de implementar. Otra ventaja es la existencia de definiciones de nivel funcional que no son simplemente el correspondiente de una definición de nivel de valores. Estas definiciones, a veces un poco crípticas por lo concisas representan un estilo de programación muy poderoso.

Si bien la propuesta de Backus data de los [años 70](https://es.wikipedia.org/wiki/A%C3%B1os_1970) ella ha sido poco adoptada por la comunidad de [programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional) que han preferido basar sus trabajos en el [cálculo Lambda](https://es.wikipedia.org/wiki/C%C3%A1lculo_Lambda).

La programación a nivel funcional en el estilo de FP tiene una fuerte relación con la [lógica combinatoria](https://es.wikipedia.org/wiki/L%C3%B3gica_combinatoria) de [Haskell Curry](https://es.wikipedia.org/wiki/Haskell_Curry" \o "Haskell Curry), con los lenguajes de combinadores, antecesores de Miranda y [Haskell](https://es.wikipedia.org/wiki/Haskell" \o "Haskell), así como con las [categorías cartesianas cerradas](https://es.wikipedia.org/w/index.php?title=Categor%C3%ADa_cartesiana_cerrada&action=edit&redlink=1), teoría que dio origen al lenguaje CAML (CategoricalAbstract Machine Languaje) antecesor del lenguaje [Ocaml](https://es.wikipedia.org/wiki/Ocaml" \o "Ocaml).

# Programación con restricciones

La **Programación por restricciones** es un paradigma de la [programación](https://es.wikipedia.org/wiki/Programaci%C3%B3n) en [informática](https://es.wikipedia.org/wiki/Inform%C3%A1tica), donde las relaciones entre las variables son expresadas en términos de restricciones ([ecuaciones](https://es.wikipedia.org/wiki/Ecuaci%C3%B3n)). Actualmente es usada como una tecnología de[software](https://es.wikipedia.org/wiki/Software) para la descripción y resolución de [problemas combinatorios](https://es.wikipedia.org/w/index.php?title=Problema_combinatorio&action=edit&redlink=1) particularmente difíciles, especialmente en las áreas de planificación y programación de tareas (calendarización).

Este paradigma representa uno de los desarrollos más fascinantes en los [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguajes_de_programaci%C3%B3n) desde 1990 y no es sorprendente que recientemente haya sido identificada por la [ACM](https://es.wikipedia.org/wiki/Association_for_Computing_Machinery) (Asociación de Maquinaria Computacional) como una dirección estratégica en la investigación en computación.

Se trata de un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) basado en la especificación de un conjunto de restricciones, las cuales deben ser satisfechas por cualquier solución del problema planteado, en lugar de especificar los pasos para obtener dicha solución.

La programación con restricciones se relaciona mucho con la [programación lógica](https://es.wikipedia.org/wiki/Programaci%C3%B3n_l%C3%B3gica) y con la [investigación operativa](https://es.wikipedia.org/wiki/Investigaci%C3%B3n_operativa). De hecho cualquier programa lógico puede ser traducido en un programa con restricciones y viceversa. Muchas veces los programas lógicos son traducidos a programas con restricciones debido a que la solución es más eficiente que su contraparte.

La diferencia entre ambos radica principalmente en sus estilos y enfoques en el modelado del mundo. Para ciertos problemas es más natural (y por ende más simple) escribirlos como programas lógicos, mientras que en otros es más natural escribirlos como programas con restricciones.

El enfoque de la programación con restricciones se basa principalmente en buscar un estado en el cual una gran cantidad de restricciones sean satisfechas simultáneamente. Un problema se define típicamente como un estado de la realidad en el cual existe un número de variables con valor desconocido. Un programa basado en restricciones busca dichos valores para todas las variables.

Algunos dominios de aplicación de este paradigma son:

* Dominios [booleanos](https://es.wikipedia.org/wiki/%C3%81lgebra_de_Boole), donde solo existen restricciones del tipo verdadero/falso.
* Dominios en variables [enteras](https://es.wikipedia.org/wiki/N%C3%BAmero_entero) y [racionales](https://es.wikipedia.org/wiki/N%C3%BAmero_racional).
* Dominios [lineales](https://es.wikipedia.org/wiki/%C3%81lgebra_lineal), donde solo se describen y analizan funciones lineales.
* Dominios finitos, donde las restricciones son definidas en conjuntos finitos.
* Dominios mixtos, los cuales involucran dos o más de los anteriores.

Los lenguajes de programación con restricciones son típicamente ampliaciones de otro lenguaje. El primer lenguaje utilizado a tal efecto fue [Prolog](https://es.wikipedia.org/wiki/Prolog" \o "Prolog). Por esta razón es que este campo fue llamado inicialmente **Programación Lógica con Restricciones**. Ambos paradigmas comparten características muy similares, tales como las variables lógicas (una vez que una variable es asignada a un valor, no puede ser cambiado), o el [backtracking](https://es.wikipedia.org/wiki/Backtracking" \o "Backtracking).

La programación con restricciones puede ser implementado como un lenguaje propio o como bibliotecas para ser usadas en algún lenguaje de programación imperativo. Algunos lenguajes populares de programación con restricciones son:

* [B-Prolog](http://www.probp.com/) (Basado en Prolog, propietario)
* [CHIP V5](http://www.cosytec.com/production_scheduling/chip/optimization_product_chip.htm) (Basado en Prolog, también existen bibliotecas en [C](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C) y [C++](https://es.wikipedia.org/wiki/C%2B%2B), propietario)
* [CiaoProlog](http://www.clip.dia.fi.upm.es/Software/Ciao/) (Basado en Prolog, [software libre](https://es.wikipedia.org/wiki/Software_libre): [GPL](https://es.wikipedia.org/wiki/Licencia_p%C3%BAblica_general_de_GNU)/[LGPL](https://es.wikipedia.org/wiki/GNU_LGPL))
* [ECLiPSe](http://www.eclipseclp.org/) (Basado en Prolog, software libre)
* [Mozart](http://www.mozart-oz.org/) ( Basado en [Oz](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Oz), software libre: [X11](https://es.wikipedia.org/wiki/X_Window_System))
* [SICStus](http://www.sics.se/sicstus/) (Basado en Prolog, propietario)
* [GNU Prolog](https://es.wikipedia.org/wiki/GNU_Prolog) (Basado en Prolog, software libre)
* [SWI-Prolog](https://es.wikipedia.org/wiki/SWI-Prolog) Un entorno Prolog que contiene varias librerías para soluciones con restricciones (LGPL)

Programación declarativa

La **Programación Declarativa**, en contraposición a la [programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa) es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que está basado en el desarrollo de [programas](https://es.wikipedia.org/wiki/Software) 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](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/Transparencia_referencial)

Diferencia entre imperativo y declarativo[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_declarativa&action=edit&section=1" \o "Editar sección: Diferencia entre imperativo y declarativo)]

En la [programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_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](https://es.wikipedia.org/wiki/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 pero a nivel de usuario, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de [inferencia](https://es.wikipedia.org/wiki/Inferencia) de información a partir de la descripción realizada.

# Programación dirigida por eventos

La **programación dirigida por eventos** es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.

Para entender la programación dirigida por eventos, podemos oponerla a lo que no es: mientras en la programación secuencial (o [estructurada](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada)) es el programador el que define cuál va a ser el flujo del programa, en la programación dirigida por eventos será el propio usuario —o lo que sea que esté accionando el programa— el que dirija el flujo del programa. Aunque en la programación secuencial puede haber intervención de un agente externo al programa, estas intervenciones ocurrirán cuando el programador lo haya determinado, y no en cualquier momento como puede ser en el caso de la programación dirigida por eventos.

El creador de un programa dirigido por eventos debe definir los eventos que manejarán su programa y las acciones que se realizarán al producirse cada uno de ellos, lo que se conoce como el [administrador de evento](https://es.wikipedia.org/w/index.php?title=Administrador_de_evento&action=edit&redlink=1). Los eventos soportados estarán determinados por el [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) utilizado, por el [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo) e incluso por eventos creados por el mismo programador.

En la programación dirigida por eventos, al comenzar la ejecución del programa se llevarán a cabo las inicializaciones y demás código inicial y a continuación el programa quedará bloqueado hasta que se produzca algún evento. Cuando alguno de los eventos esperados por el programa tenga lugar, el programa pasará a ejecutar el código del correspondiente[administrador de evento](https://es.wikipedia.org/w/index.php?title=Administrador_de_evento&action=edit&redlink=1). Por ejemplo, si el evento consiste en que el usuario ha hecho clic en el botón de play de un reproductor de películas, se ejecutará el código del [administrador de evento](https://es.wikipedia.org/w/index.php?title=Administrador_de_evento&action=edit&redlink=1), que será el que haga que la película se muestre por pantalla.

Un ejemplo claro lo tenemos en los sistemas de programación [Lexico](https://es.wikipedia.org/wiki/Lexico" \o "Lexico) y [Visual Basic](https://es.wikipedia.org/wiki/Visual_Basic), en los que a cada elemento del programa (objetos, controles, etcétera) se le asignan una serie de eventos que generará dicho elemento, como la pulsación de un botón del ratón sobre él o el redibujado del control.

La programación dirigida por eventos es la base de lo que llamamos [interfaz de usuario](https://es.wikipedia.org/wiki/Interfaz_de_usuario), aunque puede emplearse también para desarrollar interfaces entre componentes de Software o [módulos del núcleo](https://es.wikipedia.org/wiki/M%C3%B3dulo_de_n%C3%BAcleo).

En los primeros tiempos de la computación, los programas eran secuenciales, también llamados Batch. Un programa secuencial arranca, lee parámetros de entrada, procesa estos parámetros, y produce un resultado, todo de manera lineal y sin intervención del usuario mientras se ejecuta.

Con la aparición y popularización de los PC, el software empezó a ser demandado para usos alejados de los clásicos académicos y empresariales para los cuales era necesitado hasta entonces, y quedó patente que el paradigma clásico de programación no podía responder a las nuevas necesidades de interacción con el usuario que surgieron a raíz de este hecho.

Programación estructurada

La **programación estructurada** es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) orientado a mejorar la claridad, calidad y tiempo de desarrollo de un [programa de computadora](https://es.wikipedia.org/wiki/Programa_de_computadora), utilizando únicamente [subrutinas](https://es.wikipedia.org/wiki/Subrutina) y tres estructuras: [secuencia](https://es.wikipedia.org/wiki/Bloque_de_c%C3%B3digo), selección (*[if](https://es.wikipedia.org/wiki/If" \o "If)* y[*switch*](https://es.wikipedia.org/wiki/Switch)) e [iteración](https://es.wikipedia.org/wiki/Iteraci%C3%B3n) ([bucles](https://es.wikipedia.org/wiki/Bucle_(programaci%C3%B3n)) *[for](https://es.wikipedia.org/wiki/Bucle_for" \o "Bucle for)* y *[while](https://es.wikipedia.org/wiki/Bucle_while" \o "Bucle while)*), considerando innecesario y contraproducente el uso de la instrucción de[transferencia incondicional](https://es.wikipedia.org/wiki/Transferencia_incondicional) ([GOTO](https://es.wikipedia.org/wiki/GOTO)), que podría conducir a "[código espagueti](https://es.wikipedia.org/wiki/C%C3%B3digo_espagueti)", que es mucho más difícil de seguir y de mantener, y era la causa de muchos [errores de programación](https://es.wikipedia.org/wiki/Error_de_programaci%C3%B3n).

Surgió en la década de 1960, particularmente del trabajo de [Böhm](https://es.wikipedia.org/wiki/Corrado_B%C3%B6hm" \o "Corrado Böhm) y Jacopini,[1](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada#cite_note-1) y una famosa carta, «La sentencia goto considerada perjudicial», de [EdsgerDijkstra](https://es.wikipedia.org/wiki/Edsger_Dijkstra" \o "Edsger Dijkstra) en 1968[2](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada#cite_note-dijkstra1968-2) — y fue reforzado teóricamente por el [teorema del programa estructurado](https://es.wikipedia.org/wiki/Teorema_del_programa_estructurado), y prácticamente por la aparición de lenguajes como [ALGOL](https://es.wikipedia.org/wiki/ALGOL) con adecuadas y ricas [estructuras de control](https://es.wikipedia.org/wiki/Estructuras_de_control).

Orígenes de la programación estructurada[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_estructurada&action=edit&section=1" \o "Editar sección: 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.

# Programación funcional

En [ciencias de la computación](https://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n), la **programación funcional** es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) [declarativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa) basado en el uso de [funciones matemáticas](https://es.wikipedia.org/wiki/Funci%C3%B3n_matem%C3%A1tica), en contraste con la [programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa), que enfatiza los cambios de estado mediante la mutación de variables. La programación funcional tiene sus raíces en el [cálculo lambda](https://es.wikipedia.org/wiki/C%C3%A1lculo_lambda), un sistema formal desarrollado en los años 1930 para investigar la definición de función, la aplicación de las funciones y la recursión. Muchos lenguajes de programación funcionales pueden ser vistos como elaboraciones del cálculo lambda.

En la práctica, la diferencia entre una función matemática y la noción de una "función" utilizada en la programación imperativa, es que las funciones imperativas pueden tener efectos secundarios, como cambiar el valor de cálculos realizados previamente. Por esta razón carecen de transparencia referencial, es decir, la misma expresión sintáctica puede resultar en valores diferentes en varios momentos de la ejecución del programa. Con código funcional, en contraste, el valor generado por una función depende exclusivamente de los argumentos alimentados a la función. Al eliminar los efectos secundarios se puede entender y predecir el comportamiento de un programa mucho más fácilmente. Ésta es una de las principales motivaciones para utilizar la programación funcional.

Los lenguajes de programación funcional, especialmente los puramente funcionales, han sido enfatizados en el ambiente académico y no tanto en el desarrollo comercial o industrial. Sin embargo, lenguajes de programación funcional como[Scheme](https://es.wikipedia.org/wiki/Scheme), [Erlang](https://es.wikipedia.org/wiki/Erlang" \o "Erlang), [Rust](https://es.wikipedia.org/wiki/Rust_(lenguaje_de_programaci%C3%B3n)" \o "Rust (lenguaje de programación)), [ObjectiveCaml](https://es.wikipedia.org/wiki/Objective_CAML" \o "Objective CAML) , [Scala](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Scala" \o "Lenguaje de programación Scala), F# y [Haskell](https://es.wikipedia.org/wiki/Haskell" \o "Haskell), han sido utilizados en aplicaciones comerciales e industriales por muchas organizaciones. La programación funcional también es utilizada en la industria a través de lenguajes de dominio específico como R (estadística), Mathematica (matemáticas simbólicas), J y K (análisis financiero).

Los lenguajes de uso específico usados comúnmente como SQL y Lex/Yacc, utilizan algunos elementos de programación funcional, especialmente al procesar valores mutables. Las hojas de cálculo también pueden ser consideradas lenguajes de programación funcional.

La programación funcional también puede ser desarrollada en lenguajes que no están diseñados específicamente para la programación funcional. En el caso de [Perl](https://es.wikipedia.org/wiki/Perl), por ejemplo, que es un lenguaje de programación imperativo, existe un libro que describe como aplicar conceptos de programación funcional. [JavaScript](https://es.wikipedia.org/wiki/JavaScript), uno de los lenguajes más ampliamente utilizados en la actualidad, también incorpora capacidades de programación funcional. [Python](https://es.wikipedia.org/wiki/Python) también incorpora particularidades de los lenguajes funcionales como listas de comprensión y funciones de tratamiento de listas como matemática de conjuntos. Java en su versión 8, está incorporando la programación funcional, así como el uso de las expresiones lambda.

# Programación imperativa

La **programación imperativa**, en contraposición a la [programación declarativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa), es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al [computador](https://es.wikipedia.org/wiki/Computadora_electr%C3%B3nica) cómo realizar una tarea.

La implementación de [hardware](https://es.wikipedia.org/wiki/Hardware) de la mayoría de computadores es imperativa; prácticamente todo el hardware de los computadores está diseñado para ejecutar [código de máquina](https://es.wikipedia.org/wiki/C%C3%B3digo_de_m%C3%A1quina), que es nativo al computador, escrito en una forma imperativa. Esto se debe a que el hardware de los computadores implementa el paradigma de las [Máquinas de Turing](https://es.wikipedia.org/wiki/M%C3%A1quina_de_Turing). Desde esta perspectiva de bajo nivel, el estilo del programa está definido por los contenidos de la memoria, y las sentencias son instrucciones en el lenguaje de máquina nativo del computador (por ejemplo el [lenguaje ensamblador](https://es.wikipedia.org/wiki/Lenguaje_ensamblador)).

Los lenguajes imperativos de alto nivel usan [variables](https://es.wikipedia.org/wiki/Variable_(programaci%C3%B3n)) y sentencias más complejas, pero aún siguen el mismo paradigma. Las recetas y las listas de revisión de procesos, a pesar de no ser programas de computadora, son también conceptos familiares similares en estilo a la programación imperativa; donde cada paso es una instrucción.

Son aquellos cuyos valores son atómicos y están formados por elementos en un caso especial.

Los primeros lenguajes imperativos fueron los lenguajes de máquina de los computadores originales. En estos lenguajes, las instrucciones fueron muy simples, lo cual hizo la implementación de hardware fácil, pero obstruyendo la creación de programas complejos. [Fortran](https://es.wikipedia.org/wiki/Fortran), cuyo desarrollo fue iniciado en [1954](https://es.wikipedia.org/wiki/1954) por [John Backus](https://es.wikipedia.org/wiki/John_Backus) en [IBM](https://es.wikipedia.org/wiki/IBM), fue el primer gran lenguaje de programación en superar los obstáculos presentados por el código de máquina en la creación de programas complejos.

Programación lógica

La **programación lógica** es un tipo de [paradigmas de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) dentro del paradigma de [programación declarativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa). El resto de los subparadigmas de programación dentro de la programación declarativa son: [programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional),[programación con restricciones](https://es.wikipedia.org/wiki/Programaci%C3%B3n_con_restricciones), programas DSL (de dominio específico) e híbridos. La programación funcional se basa en el concepto de [función](https://es.wikipedia.org/wiki/Funci%C3%B3n_matem%C3%A1tica) (que no es más que una evolución de los predicados), de corte más matemático. La programación lógica gira en torno al concepto de [predicado](https://es.wikipedia.org/wiki/Predicado_(l%C3%B3gica)), o relación entre elementos.

Motivación[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_l%C3%B3gica&action=edit&section=1" \o "Editar sección: Motivación)]

Históricamente, los [ordenadores](https://es.wikipedia.org/wiki/Computadora_electr%C3%B3nica) se han programado utilizando [lenguajes](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) muy cercanos a las peculiaridades de la propia máquina: operaciones aritméticas simples, instrucciones de acceso a memoria, etc. Un programa escrito de esta manera puede ocultar totalmente su propósito a la comprensión de un ser humano, incluso uno entrenado. Hoy día, estos lenguajes pertenecientes al paradigma de la [Programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa) han evolucionado de manera que ya no son tan crípticos.

En cambio, la [lógica matemática](https://es.wikipedia.org/wiki/L%C3%B3gica_matem%C3%A1tica) es la manera más sencilla, para el intelecto humano, de expresar formalmente problemas complejos y de resolverlos mediante la aplicación de reglas, hipótesis y teoremas. De ahí que el concepto de "programación lógica" resulte atractivo en diversos campos donde la programación tradicional es un fracaso.

Campos de aplicación[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_l%C3%B3gica&action=edit&section=2" \o "Editar sección: Campos de aplicación)]

La programación lógica encuentra su *hábitat* natural en aplicaciones de inteligencia artificial o relacionadas:

* [Sistemas expertos](https://es.wikipedia.org/wiki/Sistema_experto), donde un sistema de información imita las recomendaciones de un experto sobre algún dominio de conocimiento.
* [Demostración automática](https://es.wikipedia.org/wiki/Demostraci%C3%B3n_autom%C3%A1tica) de teoremas, donde un programa genera nuevos teoremas sobre una teoría existente.
* Reconocimiento de [lenguaje natural](https://es.wikipedia.org/wiki/Lenguaje_natural), donde un programa es capaz de comprender (con limitaciones) la información contenida en una expresión lingüística humana.

La programación lógica también se utiliza en aplicaciones más "mundanas" pero de manera muy limitada, ya que la programación tradicional es más adecuada a tareas de propósito general.

# Programación modular

Diagrama del funcionamiento de un subprograma.

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_(programaci%C3%B3n)). 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 aspectos

La **Programación Orientada a Aspectos** o POA (en [inglés](https://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s): ***aspect-orientedprogramming***) es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de responsabilidades (Obligación o correspondencia de hacer algo).

Gracias a la POA se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables. Varias tecnologías con nombres diferentes se encaminan a la consecución de los mismos objetivos y así, el término [POA](https://es.wikipedia.org/wiki/POA) es usado para referirse a varias tecnologías relacionadas como los [métodos adaptativos](https://es.wikipedia.org/w/index.php?title=M%C3%A9todo_adaptativo&action=edit&redlink=1), los [filtros de composición](https://es.wikipedia.org/w/index.php?title=Filtro_de_composici%C3%B3n&action=edit&redlink=1), la[programación orientada a sujetos](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_orientada_a_sujetos&action=edit&redlink=1) o la [separación multidimensional de competencias](https://es.wikipedia.org/w/index.php?title=Separaci%C3%B3n_multidimensional_de_competencias&action=edit&redlink=1).

* [5Enlaces externos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_aspectos#Enlaces_externos)

Objetivo[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_orientada_a_aspectos&action=edit&section=1" \o "Editar sección: Objetivo)]

El principal objetivo de la POA es la separación de las funcionalidades dentro del sistema:

* Por un lado funcionalidades comunes utilizadas a lo largo de la aplicación.
* Por otro lado, las funcionalidades propias de cada módulo.

Cada funcionalidad común se encapsulará en una entidad.

Conceptos Básicos[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_orientada_a_aspectos&action=edit&section=2" \o "Editar sección: Conceptos Básicos)]

* **Aspecto** (en inglés *Aspect*) es una funcionalidad transversal (*cross-cutting*) que se va a implementar de forma modular y separada del resto del sistema. El ejemplo más común y simple de un aspecto es el logging (registro de sucesos) dentro del sistema, ya que necesariamente afecta a todas las partes del sistema que generan un suceso.
* **Punto de Cruce o de Unión** (en inglés *Joinpoint*) es un punto de ejecución dentro del sistema donde un aspecto puede ser conectado, como una llamada a un método, el lanzamiento de una excepción o la modificación de un campo. El código del aspecto será insertado en el flujo de ejecución de la aplicación para añadir su funcionalidad.
* **Consejo** (en inglés *Advice*) es la implementación del aspecto, es decir, contiene el código que implementa la nueva funcionalidad. Se insertan en la aplicación en los Puntos de Cruce.
* **Puntos de Corte** (en inglés *Pointcut*) define los Consejos que se aplicarán a cada Punto de Cruce. Se especifica mediante Expresiones Regulares o mediante patrones de nombres (de clases, métodos o campos), e incluso dinámicamente en tiempo de ejecución según el valor de ciertos parámetros.
* **Introducción** (en inglés *Introduction*) permite añadir métodos o atributos a clases ya existentes. Un ejemplo en el que resultaría útil es la creación de un Consejo de Auditoría que mantenga la fecha de la última modificación de un objeto, mediante una variable y un método setUltimaModificacion(fecha), que podrían ser introducidos en todas las clases (o sólo en algunas) para proporcionarles esta nueva funcionalidad.
* **Destinatario** (en inglés *Target*) es la clase aconsejada, la clase que es objeto de un consejo. Sin AOP, esta clase debería contener su lógica, además de la lógica del aspecto.
* **Resultante** (en inglés *Proxy*) es el objeto creado después de aplicar el Consejo al Objeto Destinatario. El resto de la aplicación únicamente tendrá que soportar al Objeto Destinatario (pre-AOP) y no al Objeto Resultante (post-AOP).
* **Tejido** (en inglés *Weaving*) es el proceso de aplicar Aspectos a los Objetos Destinatarios para crear los nuevos Objetos Resultantes en los especificados Puntos de Cruce. Este proceso puede ocurrir a lo largo del ciclo de vida del Objeto Destinatario:
  + Aspectos en Tiempo de Compilación, que necesita un compilador especial.
  + Aspectos en Tiempo de Carga, los Aspectos se implementan cuando el Objeto Destinatario es cargado. Requiere un ClassLoader especial.
  + Aspectos en Tiempo de Ejecución.

# Programación orientada a componentes

La **programación orientada a componentes** (que también es llamada *basada en componentes*) es una rama de la[ingeniería del software](https://es.wikipedia.org/wiki/Ingenier%C3%ADa_del_software), con énfasis en la descomposición de sistemas ya conformados en componentes funcionales o lógicos con [interfaces](https://es.wikipedia.org/wiki/Interfaces) bien definidas usadas para la comunicación entre componentes.

Se considera que el nivel de [abstracción](https://es.wikipedia.org/wiki/Capa_de_abstracci%C3%B3n) de los componentes es más alto que el de los [objetos](https://es.wikipedia.org/wiki/Objeto) y por lo tanto no comparten un estado y se comunican intercambiando mensajes que contienen datos.

## Componente de software[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_orientada_a_componentes&action=edit&section=1" \o "Editar sección: Componente de software)]

### Definición[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_orientada_a_componentes&action=edit&section=2" \o "Editar sección: Definición)]

Un [componente de software](https://es.wikipedia.org/wiki/Componente_de_software) es un elemento de un [sistema](https://es.wikipedia.org/wiki/Sistema) que ofrece un servicio predefinido, y es capaz de comunicarse con otros componentes.

Una definición más simple puede ser: **Un componente es un objeto escrito de acuerdo a unas especificaciones**. No importa que especificación sea esta, siempre y cuando el objeto se adhiera a la especificación. Solo cumpliendo correctamente con esa especificación es que el objeto se convierte en componente y adquiere características como reusabilidad.

Cuando se necesita el acceso a un componente o cuando este debe ser compartido entre distintas redes, se recurre a procesos como la [serialización](https://es.wikipedia.org/wiki/Serializaci%C3%B3n" \o "Serialización) para entregar el componente a su destino.

La capacidad de ser reutilizado (reusability), es una característica importante de los componentes de software de alta calidad. Un componente debe ser diseñado e implementado de tal forma que pueda ser reutilizado en muchos programas diferentes.

Requiere gran esfuerzo y atención escribir un componente que es realmente reutilizable. Para esto, el componente debe estar:

* Completamente documentado.
* Probado intensivamente:
  + Debe ser robusto, comprobando la validez de las entradas.
  + Debe ser capaz de pasar mensajes de error apropiados.
* Diseñado pensando en que será usado de maneras imprevistas.

Programación orientada a objetos

La **programación orientada a objetos** (**POO**, u **OOP** según sus siglas en inglés) es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que usa [objetos](https://es.wikipedia.org/wiki/Objetos_(programaci%C3%B3n_orientada_a_objetos)) en sus interacciones, para diseñar [aplicaciones](https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica) y [programas informáticos](https://es.wikipedia.org/wiki/Programas_inform%C3%A1ticos).

Está basada en varias técnicas, incluyendo [herencia](https://es.wikipedia.org/wiki/Herencia_(inform%C3%A1tica)), [cohesión](https://en.wikipedia.org/wiki/Cohesion_(computer_science)), [abstracción](https://es.wikipedia.org/wiki/Abstracci%C3%B3n_(inform%C3%A1tica)), [polimorfismo](https://es.wikipedia.org/wiki/Polimorfismo_(inform%C3%A1tica)), [acoplamiento](https://es.wikipedia.org/wiki/Acoplamiento_(inform%C3%A1tica)) y[encapsulamiento](https://es.wikipedia.org/wiki/Encapsulamiento_(inform%C3%A1tica)).

Su uso se popularizó a principios de la [década de 1990](https://es.wikipedia.org/wiki/D%C3%A9cada_de_1990). En la actualidad, existe una gran variedad de [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguajes_de_programaci%C3%B3n) que soportan la orientación a objetos.

**Índice**

  [[ocultar](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos)]

Introducción[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_orientada_a_objetos&action=edit&section=1" \o "Editar sección: 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](https://es.wikipedia.org/wiki/Variable_(programaci%C3%B3n)) o una [constante](https://es.wikipedia.org/wiki/Constante_(programaci%C3%B3n))).

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](https://es.wikipedia.org/wiki/M%C3%A9todo_(programaci%C3%B3n_orientada_a_objetos)), 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](https://es.wikipedia.org/wiki/Programador" \o "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](https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) camuflada" en un lenguaje de POO.

La programación orientada a objetos difiere de la [programación estructurada](https://es.wikipedia.org/wiki/Programaci%C3%B3n_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[[editar](https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_orientada_a_objetos&action=edit&section=2)]

Los conceptos de la POO tienen origen en [Simula 67](https://es.wikipedia.org/wiki/Simula), un lenguaje diseñado para hacer simulaciones, creado por [Ole-Johan Dahl](https://es.wikipedia.org/wiki/Ole-Johan_Dahl) y [Kristen Nygaard](https://es.wikipedia.org/wiki/Kristen_Nygaard" \o "Kristen Nygaard), del Centro de Cómputo Noruego en [Oslo](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/Smalltalk), desarrollado en Simula en [Xerox PARC](https://es.wikipedia.org/wiki/Xerox_PARC) (cuya primera versión fue escrita sobre [Basic](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/A%C3%B1os_1980), en gran parte debido a la influencia de [C++](https://es.wikipedia.org/wiki/C%2B%2B), una extensión del [lenguaje de programación C](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C). Su dominación fue consolidada gracias al auge de las[interfaces gráficas de usuario](https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario), para las cuales la POO está particularmente bien adaptada. En este caso, se habla también de [**programación dirigida por eventos**](https://es.wikipedia.org/wiki/Programaci%C3%B3n_dirigida_por_eventos).

Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo [Ada](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Ada), [BASIC](https://es.wikipedia.org/wiki/BASIC), [Lisp](https://es.wikipedia.org/wiki/Lisp" \o "Lisp) más [Pascal](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_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](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Eiffel) de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos, pero ahora ha sido esencialmente reemplazado por [Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)), en gran parte debido a la aparición de [Internet](https://es.wikipedia.org/wiki/Internet) y a la implementación de la [máquina virtual Java](https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java) en la mayoría de [navegadores web](https://es.wikipedia.org/wiki/Navegadores_web). [PHP](https://es.wikipedia.org/wiki/PHP) en su versión 5 se ha modificado; soporta una orientación completa a objetos, cumpliendo todas las características propias de la orientación a objetos.