**Paradigma de programación**

Un paradigma de programación es una propuesta tecnológica que es adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados. Es un estilo de programación empleado. La resolución de estos problemas debe suponer consecuentemente un avance significativo en al menos un parámetro que afecte a la ingeniería de software. Tiene una estrecha relación con la formalización de determinados lenguajes en su momento de definición. Un paradigma de programación está delimitado en el tiempo en cuanto a aceptación y uso ya que nuevos paradigmas aportan nuevas o mejores soluciones que la sustituyen parcial o totalmente.

**Ejemplo :**

Probablemente el paradigma de programación que actualmente es el más usado a todos los niveles es la orientación a objeto. El núcleo central de este paradigma es la unión de datos y procesamiento en una entidad llamada "objeto", relacionable a su vez con otras entidades "objeto".

Tradicionalmente datos y procesamiento se han separado en áreas diferente del diseño y la implementación de software. Esto provocó que grandes desarrollos tuvieran problemas de fiabilidad, mantenimiento, adaptación a los cambios y escalabilidad. Con la orientación a objetos y características como el encapsulado, polimorfismo o la herencia se permitió un avance significativo en el desarrollo de software a cualquier escala de producción.

La orientación a objeto parece estar ligada en sus orígenes con lenguajes como Lisp y Simula aunque el primero que acuñó el título de programación orientada a objetos fue Smalltalk

**Tipos de paradigmas de programación más comunes**

* Imperativo o por procedimientos: es considerado el más común y está representado, por ejemplo, por [C](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C), BASIC o Pascal
* [Funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional): está representado por [Scheme](https://es.wikipedia.org/wiki/Scheme) o [Haskell](https://es.wikipedia.org/wiki/Haskell%22%20%5Co%20%22Haskell). Este es un caso del paradigma declarativo.
* Lógico: está representado por [Prolog](https://es.wikipedia.org/wiki/Prolog%22%20%5Co%20%22Prolog). Este es otro caso del paradigma declarativo.
* [Declarativo](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa): por ejemplo la [programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional), la [programación lógica](https://es.wikipedia.org/wiki/Programaci%C3%B3n_l%C3%B3gica), o la combinación lógico-funcional.
* [Orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos): está representado por [Smalltalk](https://es.wikipedia.org/wiki/Smalltalk%22%20%5Co%20%22Smalltalk), un lenguaje completamente orientado a objetos.
* [Programación dinámica](https://es.wikipedia.org/wiki/Programaci%C3%B3n_din%C3%A1mica): está definida como el proceso de romper problemas en partes pequeñas para analizarlos.
* Si bien puede seleccionarse la forma pura de estos paradigmas al momento de programar, en la práctica es habitual que se mezclen, dando lugar a la [programación multiparadigma](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_multiparadigma).
* Actualmente el paradigma de programación más usado es el de la [programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos).

**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**

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, 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.

**Tipos**

* Existen varios tipos de lenguajes declarativos:
* Los lenguajes lógicos, como [Prolog](https://es.wikipedia.org/wiki/Prolog%22%20%5Co%20%22Prolog).
* Los lenguajes algebraicos, como [Maude](https://es.wikipedia.org/wiki/Maude%22%20%5Co%20%22Maude) y [SQL](https://es.wikipedia.org/wiki/SQL)
* Los lenguajes funcionales, como [Haskell](https://es.wikipedia.org/wiki/Haskell%22%20%5Co%20%22Haskell)

**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.[1](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa#cite_note-1)

Son fiables, elegantes y expresivos.

**Algunos lenguajes declarativos**

* [Haskell](https://es.wikipedia.org/wiki/Haskell) ([Programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional))
* [ML](https://es.wikipedia.org/wiki/ML) ([Programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional))
* [Lisp](https://es.wikipedia.org/wiki/Lisp) ([Programación funcional](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional))
* [Prolog](https://es.wikipedia.org/wiki/Prolog) ([Programación Lógica](https://es.wikipedia.org/wiki/Programaci%C3%B3n_L%C3%B3gica))
* F-Prolog (Programación Lógica Difusa)
* Curry (Programación Lógico-Funcional)
* [SQL](https://es.wikipedia.org/wiki/SQL)
* G

**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 basado en la utilización de [funciones aritméticas](https://es.wikipedia.org/wiki/Funci%C3%B3n_matem%C3%A1tica) que no maneja datos mutables o de estado. Enfatiza la aplicación de funciones, en contraste con el estilo de [programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa), que enfatiza los cambios de estado. 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 1930s 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, al 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 diferentes momentos dependiendo del estado del programa siendo ejecutado. 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, y esta es una de las principales motivaciones para utilizar la programación funcional.

Los lenguajes de programación funcional, especialmente los que son puramente funcionales, han sido enfatizados en el ambiente académico principalmente y no tanto en el desarrollo de software comercial. Sin embargo, lenguajes de programación importantes tales como Scheme, Erlang, Objective Caml y Haskel, 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), F# en Microsoft.NET y XSLT (XML). 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.

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 (que no es más que una evolución de los predicados), de corte más matemático. La programación lógica gira en torno al concepto de predicado, o relación entre elementos.

**En qué consiste (ejemplo)**

La programación lógica permite formalizar hechos del mundo real, por ejemplo:

 las aves vuelan

 los pingüinos no vuelan

 "pichurri" es un ave

 "sandokan" es un perro

 "alegría" es un ave

y también reglas o restricciones:

 una mascota vuela si es un ave y no es un pingüino

Ante dicho "programa" es posible establecer hipótesis que no son más que preguntas o incógnitas, por ejemplo:

 ¿ "pichurri" vuela ?

 ¿ qué mascotas vuelan ?....

Gracias a que la lógica de primer orden es computable, el [ordenador](https://es.wikipedia.org/wiki/Ordenador) será capaz de verificar la hipótesis, es decir, responder a las incógnitas:

 Es cierto que "pichurri" vuela.

 "pichurri" y "alegría" vuelan.

Obsérvese que el programa lógico no solamente es capaz de responder si una determinada hipótesis es verdadera o falsa. También es capaz de determinar que valores de la incógnita hacen cierta la hipótesis.

Este ejemplo es claramente académico. Sin embargo, consideremos el siguiente ejemplo: el sistema de control de semáforos de una ciudad.

El estado de cada uno de los semáforos (verde, rojo o ámbar) constituye los hechos del mundo real. El programa en sí consiste en unas pocas reglas de sentido común: determinados semáforos no pueden permanecer simultáneamente en verde, un semáforo solamente puede transitar de verde a ámbar y de ámbar a rojo, etc. La hipótesis es el estado en el que deberían estar cada uno de los semáforos en el siguiente instante de tiempo.

Éste es un ejemplo imposible de resolver mediante programación tradicional, ya que la lógica subyacente al comportamiento de los semáforos en su conjunto queda enmascarada por simples órdenes imperativas del tipo "cambiar color de tal o cual semáforo".

**Programación orientada a objetos**

La programación orientada a objetos o POO (OOP según sus siglas en inglés) es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que usa los [objetos](https://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) en sus interacciones, para diseñar aplicaciones y programas [informáticos](https://es.wikipedia.org/wiki/Inform%C3%A1tica). Está basado en varias técnicas, incluyendo [herencia](https://es.wikipedia.org/wiki/Herencia_%28inform%C3%A1tica%29), [cohesión](https://es.wikipedia.org/w/index.php?title=Cohesi%C3%B3n_(inform%C3%A1tica)&action=edit&redlink=1), [abstracción](https://es.wikipedia.org/wiki/Abstracci%C3%B3n_%28inform%C3%A1tica%29), [polimorfismo](https://es.wikipedia.org/wiki/Polimorfismo_%28inform%C3%A1tica%29), [acoplamiento](https://es.wikipedia.org/w/index.php?title=Acoplamiento_(inform%C3%A1tica)&action=edit&redlink=1) y [encapsulamiento](https://es.wikipedia.org/wiki/Encapsulamiento_%28inform%C3%A1tica%29). Su uso se popularizó a principios de la década de los años 1990. En la actualidad, existe 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:

* El estado está compuesto de datos o informaciones; serán uno o varios atributos a los que se habrán asignado unos valores concretos (datos).
* El comportamiento está definido por los [métodos](https://es.wikipedia.org/wiki/M%C3%A9todo_%28inform%C3%A1tica%29) o mensajes a los que sabe responder dicho objeto, es decir, qué operaciones se pueden realizar con él.
* 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_%28programaci%C3%B3n%29) o una [constante](https://es.wikipedia.org/wiki/Constante_%28programaci%C3%B3n%29)).

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_%28programaci%C3%B3n_orientada_a_objetos%29), 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) 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 programación orientado a objetos.

La POO 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**

Los conceptos de la programación orientada a objetos 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), 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%22%20%5Co%20%22Smalltalk), 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 programación orientada a objetos se fue convirtiendo en el estilo de programación dominante a mediados de los años ochenta, 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 programación orientada a objetos 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%22%20%5Co%20%22Lisp) y [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 [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_%28lenguaje_de_programaci%C3%B3n%29), en gran parte debido a la aparición de [Internet](https://es.wikipedia.org/wiki/Internet) y a la implementación de la máquina virtual de Java en la mayoría de [navegadores](https://es.wikipedia.org/wiki/Navegadores). [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.