# Programming paradigm

From Wikipedia, the free encyclopedia

This article is about classification of programming languages. For definition of the term "programming model", see [Programming model](https://en.wikipedia.org/wiki/Programming_model). For the use of the term in project management methodology, see [Software development § Programming paradigm](https://en.wikipedia.org/wiki/Software_development#Programming_paradigm).

**Programming paradigms** are a way to classify [programming languages](https://en.wikipedia.org/wiki/Programming_language) based on their features. Languages can be classified into multiple paradigms.

Some paradigms are concerned mainly with implications for the [execution model](https://en.wikipedia.org/wiki/Execution_model) of the language, such as allowing [side effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)), or whether the sequence of operations is defined by the execution model. Other paradigms are concerned mainly with the way that code is organized, such as grouping a code into units along with the state that is modified by the code. Yet others are concerned mainly with the style of syntax and grammar.

Common programming paradigms include:[[1]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-2)[[3]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-symbolic-programming-lisp-3)

* [imperative](https://en.wikipedia.org/wiki/Imperative_programming) in which the programmer instructs the machine how to change its state,
* [procedural](https://en.wikipedia.org/wiki/Procedural_programming) which groups instructions into procedures,
* [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) which groups instructions together with the part of the state they operate on,
* [declarative](https://en.wikipedia.org/wiki/Declarative_programming) in which the programmer merely declares properties of the desired result, but not how to compute it
* [functional](https://en.wikipedia.org/wiki/Functional_programming) in which the desired result is declared as the value of a series of function applications,
* [logic](https://en.wikipedia.org/wiki/Logic_programming) in which the desired result is declared as the answer to a question about a system of facts and rules,
* [mathematical](https://en.wikipedia.org/wiki/Mathematical_programming) in which the desired result is declared as the solution of an optimization problem

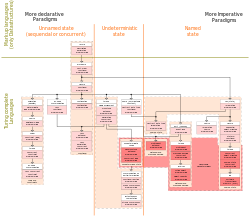
[Symbolic](https://en.wikipedia.org/wiki/Symbolic_programming) techniques such as [reflection](https://en.wikipedia.org/wiki/Reflection_(computer_programming)), which allow the program to refer to itself, might also be considered as a programming paradigm. However, this is compatible with the major paradigms and thus is not a real paradigm in its own right.

For example, languages that fall into the **imperative paradigm** have two main features: they state the order in which operations occur, with constructs that explicitly control that order, and they allow side effects, in which state can be modified at one point in time, within one unit of code, and then later read at a different point in time inside a different unit of code. The communication between the units of code is not explicit. Meanwhile, in **object-oriented** programming, code is organized into [objects](https://en.wikipedia.org/wiki/Object_(programming)) that contain state that is only modified by the code that is part of the object. Most object-oriented languages are also imperative languages. In contrast, languages that fit the **declarative paradigm** do not state the order in which to execute operations. Instead, they supply a number of operations that are available in the system, along with the conditions under which each is allowed to execute. The implementation of the language's execution model tracks which operations are free to execute and chooses the order on its own. More at [Comparison of multi-paradigm programming languages](https://en.wikipedia.org/wiki/Comparison_of_multi-paradigm_programming_languages).

## Contents

* [1 Overview](https://en.wikipedia.org/wiki/Programming_paradigm#Overview)
* [2 Criticism](https://en.wikipedia.org/wiki/Programming_paradigm#Criticism)
* [3 History](https://en.wikipedia.org/wiki/Programming_paradigm#History)
* [3.1 Machine code](https://en.wikipedia.org/wiki/Programming_paradigm#Machine_code)
* [3.2 Procedural languages](https://en.wikipedia.org/wiki/Programming_paradigm#Procedural_languages)
* [3.3 Object-oriented programming](https://en.wikipedia.org/wiki/Programming_paradigm#Object-oriented_programming)
* [3.4 Further paradigms](https://en.wikipedia.org/wiki/Programming_paradigm#Further_paradigms)
* [4 Support for multiple paradigms](https://en.wikipedia.org/wiki/Programming_paradigm#Support_for_multiple_paradigms)
* [5 See also](https://en.wikipedia.org/wiki/Programming_paradigm#See_also)
* [6 References](https://en.wikipedia.org/wiki/Programming_paradigm#References)
* [7 External links](https://en.wikipedia.org/wiki/Programming_paradigm#External_links)

## Overview

[](https://en.wikipedia.org/wiki/File:Programming_paradigms.svg)

Overview of the various programming paradigms according to Peter Van Roy[[4]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-4):5[[5]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-Van-RoyHaridi2004-5)

Just as [software engineering](https://en.wikipedia.org/wiki/Software_engineering) (as a process) is defined by differing *methodologies*, so the [programming languages](https://en.wikipedia.org/wiki/Programming_language) (as models of computation) are defined by differing *paradigms*. Some languages are designed to support one paradigm ([Smalltalk](https://en.wikipedia.org/wiki/Smalltalk) supports object-oriented programming, [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) supports functional programming), while other programming languages support multiple paradigms (such as [Object Pascal](https://en.wikipedia.org/wiki/Object_Pascal), [C++](https://en.wikipedia.org/wiki/C++), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)), [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic), [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp), [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)), [Perl](https://en.wikipedia.org/wiki/Perl), [PHP](https://en.wikipedia.org/wiki/PHP), [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [Oz](https://en.wikipedia.org/wiki/Oz_(programming_language)), and [F#](https://en.wikipedia.org/wiki/F_Sharp_(programming_language))). For example, programs written in C++, Object Pascal or PHP can be purely [procedural](https://en.wikipedia.org/wiki/Procedural_programming), purely [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), or can contain elements of both or other paradigms. Software designers and programmers decide how to use those paradigm elements.

In object-oriented programming, programs are treated as a set of interacting objects. In [functional programming](https://en.wikipedia.org/wiki/Functional_programming), programs are treated as a sequence of stateless function evaluations. When programming computers or systems with many processors, in [process-oriented programming](https://en.wikipedia.org/wiki/Process-oriented_programming), programs are treated as sets of concurrent processes acting on logically shared [data structures](https://en.wikipedia.org/wiki/Data_structure).

Many programming paradigms are as well known for the techniques they *forbid* as for those they *enable*. For instance, pure functional programming disallows use of [side-effects](https://en.wikipedia.org/wiki/Side-effect_(computer_science)), while [structured programming](https://en.wikipedia.org/wiki/Structured_programming) disallows use of the [goto](https://en.wikipedia.org/wiki/Goto) statement. Partly for this reason, new paradigms are often regarded as doctrinaire or overly rigid by those accustomed to earlier styles.[[6]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-rubin87goto-6) Yet, avoiding certain techniques can make it easier to understand program behavior, and to [prove theorems](https://en.wikipedia.org/wiki/Automated_theorem_proving) about program correctness.

Programming paradigms can also be compared with [*programming models*](https://en.wikipedia.org/wiki/Programming_model) which allow invoking an [execution model](https://en.wikipedia.org/wiki/Execution_model) by using only an API. Programming models can also be classified into paradigms, based on features of the execution model.

For [parallel computing](https://en.wikipedia.org/wiki/Parallel_computing), using a programming model instead of a language is common. The reason is that details of the parallel hardware leak into the abstractions used to program the hardware. This causes the programmer to have to map patterns in the algorithm onto patterns in the execution model (which have been inserted due to leakage of hardware into the abstraction). As a consequence, no one parallel programming language maps well to all computation problems. It is thus more convenient to use a base sequential language and insert API calls to parallel execution models, via a programming model. Such parallel programming models can be classified according to abstractions that reflect the hardware, such as shared memory, distributed memory with message passing, notions of *place* visible in the code, and so forth. These can be considered flavors of programming paradigm that apply to only parallel languages and programming models.

## Criticism

Some programming language researchers criticise the notion of paradigms as a classification of programming languages, e.g. Harper,[[7]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-7) and Krishnamurthi.[[8]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-8) They argue that many programming languages cannot be strictly classified into one paradigm, but rather include features from several paradigms. See [Comparison of multi-paradigm programming languages](https://en.wikipedia.org/wiki/Comparison_of_multi-paradigm_programming_languages).

## History

Different approaches to programming have developed over time, being identified as such either at the time or retrospectively. An early approach consciously identified as such is [structured programming](https://en.wikipedia.org/wiki/Structured_programming), advocated since the mid 1960s. The concept of a "programming paradigm" as such dates at least to 1978, in the [Turing Award](https://en.wikipedia.org/wiki/Turing_Award) lecture of [Robert W. Floyd](https://en.wikipedia.org/wiki/Robert_W._Floyd), entitled *The Paradigms of Programming*, which cites the notion of paradigm as used by [Thomas Kuhn](https://en.wikipedia.org/wiki/Thomas_Kuhn) in his [*The Structure of Scientific Revolutions*](https://en.wikipedia.org/wiki/The_Structure_of_Scientific_Revolutions) (1962).[[9]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-9)

### Machine code

The [lowest-level](https://en.wikipedia.org/wiki/Low-level_programming_language) programming paradigms are [machine code](https://en.wikipedia.org/wiki/Machine_code), which directly represents the [instructions](https://en.wikipedia.org/wiki/Instruction_set) (the contents of program memory) as a sequence of numbers, and [assembly language](https://en.wikipedia.org/wiki/Assembly_language) where the machine instructions are represented by mnemonics and [memory addresses](https://en.wikipedia.org/wiki/Memory_address) can be given symbolic labels. These are sometimes called [first-](https://en.wikipedia.org/wiki/First-generation_programming_language) and [second-generation](https://en.wikipedia.org/wiki/Second-generation_programming_language) languages.

In the 1960s, assembly languages were developed to support library COPY and quite sophisticated conditional macro generation and preprocessing abilities, CALL to ([subroutines](https://en.wikipedia.org/wiki/Subroutine)), external variables and common sections (globals), enabling significant code re-use and isolation from hardware specifics via use of logical operators such as READ/WRITE/GET/PUT. Assembly was, and still is, used for time critical systems and often in [embedded systems](https://en.wikipedia.org/wiki/Embedded_system) as it gives the most direct control of what the machine does.

### Procedural languages

The next advance was the development of [procedural languages](https://en.wikipedia.org/wiki/Procedural_language). These [third-generation](https://en.wikipedia.org/wiki/Third-generation_programming_language) languages (the first described as [high-level languages](https://en.wikipedia.org/wiki/High-level_programming_language)) use vocabulary related to the problem being solved. For example,

* COmmon Business Oriented Language ([COBOL](https://en.wikipedia.org/wiki/COBOL)) – uses terms like [file](https://en.wikipedia.org/wiki/Computer_file), [move](https://en.wikipedia.org/wiki/Move_(command)) and [copy](https://en.wikipedia.org/wiki/Copy_(command)).
* FORmula TRANslation ([FORTRAN](https://en.wikipedia.org/wiki/FORTRAN)) – using [mathematical](https://en.wikipedia.org/wiki/Mathematical) language terminology, it was developed mainly for scientific and engineering problems.
* ALGOrithmic Language ([ALGOL](https://en.wikipedia.org/wiki/ALGOL)) – focused on being an appropriate language to define [algorithms](https://en.wikipedia.org/wiki/Algorithm), while using mathematical language terminology and targeting scientific and engineering problems just like FORTRAN.
* Programming Language One ([PL/I](https://en.wikipedia.org/wiki/PL/I)) – a hybrid commercial-scientific general purpose language supporting [pointers](https://en.wikipedia.org/wiki/Pointer_(computer_programming)).
* Beginners All purpose Symbolic Instruction Code ([BASIC](https://en.wikipedia.org/wiki/BASIC)) – it was developed to enable more people to write programs.
* [C](https://en.wikipedia.org/wiki/C_(programming_language)) – a general-purpose programming language, initially developed by [Dennis Ritchie](https://en.wikipedia.org/wiki/Dennis_Ritchie) between 1969 and 1973 at [AT&T Bell Labs](https://en.wikipedia.org/wiki/AT&T_Bell_Labs).

All these languages follow the procedural paradigm. That is, they describe, step by step, exactly the procedure that should, according to the particular programmer at least, be followed to solve a specific problem. The [efficacy](https://en.wikipedia.org/wiki/Efficacy) and [efficiency](https://en.wikipedia.org/wiki/Algorithmic_efficiency) of any such solution are both therefore entirely subjective and highly dependent on that programmer's experience, inventiveness, and ability.

### Object-oriented programming

Main article: [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming)

Following the widespread use of procedural languages, [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) (OOP) languages were created, such as [Simula](https://en.wikipedia.org/wiki/Simula), [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk), [C++](https://en.wikipedia.org/wiki/C++), [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Eiffel](https://en.wikipedia.org/wiki/Eiffel_(programming_language)), [PHP](https://en.wikipedia.org/wiki/PHP), and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). In these languages, [data](https://en.wikipedia.org/wiki/Data) and methods to manipulate it are kept as one unit called an [object](https://en.wikipedia.org/wiki/Object_(computer_science)). With perfect [encapsulation](https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)), one of the distinguishing features of OOP, the only way that another object or user would be able to access the data is via the object's [*methods*](https://en.wikipedia.org/wiki/Method_(computer_programming)). Thus, the inner workings of an object may be changed without affecting any code that uses the object. There is still some [controversy](https://en.wikipedia.org/wiki/Object-oriented_programming#Criticism) raised by [Alexander Stepanov](https://en.wikipedia.org/wiki/Alexander_Stepanov), [Richard Stallman](https://en.wikipedia.org/wiki/Richard_Stallman)[[10]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-10) and other programmers, concerning the efficacy of the OOP paradigm versus the procedural paradigm. The need for every object to have associative methods leads some skeptics to associate OOP with [software bloat](https://en.wikipedia.org/wiki/Software_bloat); an attempt to resolve this dilemma came through [polymorphism](https://en.wikipedia.org/wiki/Polymorphism_(computer_science)).

Because object-oriented programming is considered a paradigm, not a language, it is possible to create even an object-oriented assembler language. [High Level Assembly](https://en.wikipedia.org/wiki/High_Level_Assembly) (HLA) is an example of this that fully supports advanced data types and object-oriented assembly language programming – despite its early origins. Thus, differing programming paradigms can be seen rather like *motivational* [*memes*](https://en.wikipedia.org/wiki/Meme) of their advocates, rather than necessarily representing progress from one level to the next[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]. Precise comparisons of the efficacy of competing paradigms are frequently made more difficult because of new and differing terminology applied to similar entities and processes together with numerous implementation distinctions across languages.

### Further paradigms

[Literate programming](https://en.wikipedia.org/wiki/Literate_programming), as a form of [imperative programming](https://en.wikipedia.org/wiki/Imperative_programming), structures programs as a human-centered web, as in a [hypertext](https://en.wikipedia.org/wiki/Hypertext) essay: documentation is integral to the program, and the program is structured following the logic of prose exposition, rather than compiler convenience.

Independent of the imperative branch, [declarative programming](https://en.wikipedia.org/wiki/Declarative_programming) paradigms were developed. In these languages, the computer is told what the problem is, not how to solve the problem – the program is structured as a set of properties to find in the expected result, not as a procedure to follow. Given a database or a set of rules, the computer tries to find a solution matching all the desired properties. An archetype of a declarative language is the [fourth generation language](https://en.wikipedia.org/wiki/Fourth-generation_programming_language) [SQL](https://en.wikipedia.org/wiki/SQL), and the family of functional languages and logic programming.

[Functional programming](https://en.wikipedia.org/wiki/Functional_programming) is a subset of declarative programming. Programs written using this paradigm use [functions](https://en.wikipedia.org/wiki/Subroutine), blocks of code intended to behave like [mathematical functions](https://en.wikipedia.org/wiki/Function_(mathematics)). Functional languages discourage changes in the value of variables through [assignment](https://en.wikipedia.org/wiki/Assignment_(computer_science)), making a great deal of use of [recursion](https://en.wikipedia.org/wiki/Recursion_(computer_science)) instead.

The [logic programming](https://en.wikipedia.org/wiki/Logic_programming) paradigm views computation as [automated reasoning](https://en.wikipedia.org/wiki/Automated_reasoning) over a body of knowledge. Facts about the [problem domain](https://en.wikipedia.org/wiki/Problem_domain) are expressed as logic formulas, and programs are executed by applying [inference rules](https://en.wikipedia.org/wiki/Inference_rule) over them until an answer to the problem is found, or the set of formulas is proved inconsistent.

[Symbolic programming](https://en.wikipedia.org/wiki/Symbolic_programming) is a paradigm that describes programs able to manipulate formulas and program components as data.[[3]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-symbolic-programming-lisp-3) Programs can thus effectively modify themselves, and appear to "learn", making them suited for applications such as [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence), [expert systems](https://en.wikipedia.org/wiki/Expert_system), [natural-language processing](https://en.wikipedia.org/wiki/Natural-language_processing) and computer games. Languages that support this paradigm include [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)) and [Prolog](https://en.wikipedia.org/wiki/Prolog).[[11]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-11)

## Support for multiple paradigms

See also: [Comparison of multi-paradigm programming languages](https://en.wikipedia.org/wiki/Comparison_of_multi-paradigm_programming_languages)

Most [programming languages](https://en.wikipedia.org/wiki/Programming_language) support more than one programming paradigm to allow programmers to use the most suitable programming style and associated language constructs for a given job.[[12]](https://en.wikipedia.org/wiki/Programming_paradigm#cite_note-12)

## See also

* [Architecture description language](https://en.wikipedia.org/wiki/Architecture_description_language)
* [Comparison of programming languages](https://en.wikipedia.org/wiki/Comparison_of_programming_languages)
* [Comparison of programming paradigms](https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms)
* [Domain-specific language](https://en.wikipedia.org/wiki/Domain-specific_language)
* [Mindset](https://en.wikipedia.org/wiki/Mindset)
* [Modeling language](https://en.wikipedia.org/wiki/Modeling_language)
* [Programming domain](https://en.wikipedia.org/wiki/Programming_domain)
* [Type system](https://en.wikipedia.org/wiki/Type_system)
* [Turing completeness](https://en.wikipedia.org/wiki/Turing_completeness)
* [Von Neumann programming languages](https://en.wikipedia.org/wiki/Von_Neumann_programming_languages)

## References

Nørmark, Kurt. [*Overview of the four main programming paradigms*](http://people.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigm-overview-section.html). Aalborg University, 9 May 2011. Retrieved 22 September 2012.

*Frans Coenen (1999-10-11).* [*"Characteristics of declarative programming languages"*](http://cgi.csc.liv.ac.uk/~frans/OldLectures/2CS24/declarative.html#detail)*. cgi.csc.liv.ac.uk. Retrieved 2014-02-20.*

*Michael A. Covington (2010-08-23).* [*"CSCI/ARTI 4540/6540: First Lecture on Symbolic Programming and LISP"*](https://web.archive.org/web/20120307124013/http://www.ai.uga.edu/mc/LispNotes/FirstLectureOnSymbolicProgramming.pdf) *(PDF). University of Georgia. Archived from* [*the original*](http://www.ai.uga.edu/mc/LispNotes/FirstLectureOnSymbolicProgramming.pdf) *(PDF) on 2012-03-07. Retrieved 2013-11-20.*

*Peter Van Roy (2009-05-12).* [*"Programming Paradigms for Dummies: What Every Programmer Should Know"*](http://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf) *(PDF). info.ucl.ac.be. Retrieved 2014-01-27.*

*Peter Van-Roy; Seif Haridi (2004).* [*Concepts, Techniques, and Models of Computer Programming*](https://books.google.com/books?id=_bmyEnUnfTsC)*. MIT Press.* [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number)[*978-0-262-22069-9*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-262-22069-9)*.*

*Frank Rubin (March 1987).* [*"'GOTO Considered Harmful' Considered Harmful"*](https://web.archive.org/web/20090320002214/http://www.ecn.purdue.edu/ParaMount/papers/rubin87goto.pdf) *(PDF). Communications of the ACM.* ***30*** *(3): 195–196.* [*doi*](https://en.wikipedia.org/wiki/Digital_object_identifier)*:*[*10.1145/214748.315722*](https://doi.org/10.1145/214748.315722)*. Archived from* [*the original*](http://www.ecn.purdue.edu/ParaMount/papers/rubin87goto.pdf) *(PDF) on March 20, 2009.*

*Harper, Robert (1 May 2017).* [*"What, if anything, is a programming-paradigm?"*](http://www.cambridgeblog.org/2017/05/what-if-anything-is-a-programming-paradigm/)*. FifteenEightyFour. Cambridge University Press.*

*Krishnamurthi, Shriram (November 2008).* [*"Teaching programming languages in a post-linnaean age"*](http://dl.acm.org/citation.cfm?id=1480846)*. SIGPLAN. ACM. pp. 81–83. Not. 43, 11.*.

*Floyd, R. W. (1979).* [*"The paradigms of programming"*](http://dl.acm.org/ft_gateway.cfm?id=359140&ftid=289772&dwn=1&CFID=285645736&CFTOKEN=55009136)*. Communications of the ACM.* ***22*** *(8): 455.* [*doi*](https://en.wikipedia.org/wiki/Digital_object_identifier)*:*[*10.1145/359138.359140*](https://doi.org/10.1145/359138.359140)*.*

[*"Mode inheritance, cloning, hooks & OOP (Google Groups Discussion)"*](http://groups.google.com/group/comp.emacs.xemacs/browse_thread/thread/d0af257a2837640c/37f251537fafbb03?lnk=st&q=)*.*

[*"Business glossary: Symbolic programming definition"*](http://www.allbusiness.com/glossaries/symbolic-programming/4950308-1.html)*. allbusiness.com. Retrieved 2014-07-30.*

1. [*"Multi-Paradigm Programming Language"*](https://web.archive.org/web/20130821052407/https://developer.mozilla.org/en-US/docs/multiparadigmlanguage.html)*. developer.mozilla.org.* [*Mozilla Foundation*](https://en.wikipedia.org/wiki/Mozilla_Foundation)*. Archived from* [*the original*](https://developer.mozilla.org/en-US/docs/multiparadigmlanguage.html) *on 21 August 2013.*

## External links

|  |  |
| --- | --- |
|  | Wikimedia Commons has media related to [***Programming paradigms***](https://commons.wikimedia.org/wiki/Category:Programming_paradigms). |

* [Classification of the principal programming paradigms](http://www.info.ucl.ac.be/~pvr/paradigms.html)
* [How programming paradigms evolve and get adopted?](http://www.janeve.me/articles/understanding-programming-paradigms)