Functional programming

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Functional_programming#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Functional_programming#searchInput)

*For subroutine-oriented programming, see*[*Procedural programming*](https://en.wikipedia.org/wiki/Procedural_programming)*.*

|  |
| --- |
| [**Programming paradigms**](https://en.wikipedia.org/wiki/Programming_paradigm) |
| * [Action](https://en.wikipedia.org/wiki/Action_language) * [Agent-oriented](https://en.wikipedia.org/wiki/Agent-oriented_programming) * [Array-oriented](https://en.wikipedia.org/wiki/Array_programming) * [Automata-based](https://en.wikipedia.org/wiki/Automata-based_programming) * [Concurrent computing](https://en.wikipedia.org/wiki/Concurrent_computing)   + [Relativistic programming](https://en.wikipedia.org/wiki/Relativistic_programming) * [Data-driven](https://en.wikipedia.org/wiki/Data-driven_programming) * [Declarative](https://en.wikipedia.org/wiki/Declarative_programming) (contrast: [Imperative](https://en.wikipedia.org/wiki/Imperative_programming))   + Functional     - [Functional logic](https://en.wikipedia.org/wiki/Functional_logic_programming)     - [Purely functional](https://en.wikipedia.org/wiki/Purely_functional_programming)   + [Logic](https://en.wikipedia.org/wiki/Logic_programming)     - [Abductive logic](https://en.wikipedia.org/wiki/Abductive_logic_programming)     - [Answer set](https://en.wikipedia.org/wiki/Answer_set_programming)     - [Concurrent logic](https://en.wikipedia.org/wiki/Concurrent_logic_programming)     - [Functional logic](https://en.wikipedia.org/wiki/Functional_logic_programming)     - [Inductive logic](https://en.wikipedia.org/wiki/Inductive_logic_programming)   + [Constraint](https://en.wikipedia.org/wiki/Constraint_programming)     - [Constraint logic](https://en.wikipedia.org/wiki/Constraint_logic_programming)       * [Concurrent constraint logic](https://en.wikipedia.org/wiki/Concurrent_constraint_logic_programming)   + [Dataflow](https://en.wikipedia.org/wiki/Dataflow_programming)     - [Flow-based](https://en.wikipedia.org/wiki/Flow-based_programming)     - [Reactive](https://en.wikipedia.org/wiki/Reactive_programming)   + [Ontology](https://en.wikipedia.org/wiki/Ontology_language) * [Differentiable](https://en.wikipedia.org/wiki/Differentiable_programming) * [Dynamic/scripting](https://en.wikipedia.org/wiki/Dynamic_programming_language) * [Event-driven](https://en.wikipedia.org/wiki/Event-driven_programming) * [Function-level](https://en.wikipedia.org/wiki/Function-level_programming) (contrast: [Value-level](https://en.wikipedia.org/wiki/Value-level_programming))   + [Point-free style](https://en.wikipedia.org/wiki/Tacit_programming)     - [Concatenative](https://en.wikipedia.org/wiki/Concatenative_programming_language) * [Generic](https://en.wikipedia.org/wiki/Generic_programming) * [Imperative](https://en.wikipedia.org/wiki/Imperative_programming) (contrast: [Declarative](https://en.wikipedia.org/wiki/Declarative_programming))   + [Procedural](https://en.wikipedia.org/wiki/Procedural_programming)   + [Object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming)     - [Polymorphic](https://en.wikipedia.org/wiki/Polymorphic_code) * [Intentional](https://en.wikipedia.org/wiki/Intentional_programming) * [Language-oriented](https://en.wikipedia.org/wiki/Language-oriented_programming)   + [Domain-specific](https://en.wikipedia.org/wiki/Domain-specific_language) * [Literate](https://en.wikipedia.org/wiki/Literate_programming) * [Natural-language programming](https://en.wikipedia.org/wiki/Natural-language_programming) * [Metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming)   + [Automatic](https://en.wikipedia.org/wiki/Automatic_programming)     - [Inductive programming](https://en.wikipedia.org/wiki/Inductive_programming)   + [Reflective](https://en.wikipedia.org/wiki/Reflection_(computer_programming))     - [Attribute-oriented](https://en.wikipedia.org/wiki/Attribute-oriented_programming)   + [Macro](https://en.wikipedia.org/wiki/Macro_(computer_science))   + [Template](https://en.wikipedia.org/wiki/Template_metaprogramming) * [Non-structured](https://en.wikipedia.org/wiki/Non-structured_programming) (contrast: [Structured](https://en.wikipedia.org/wiki/Structured_programming))   + [Array](https://en.wikipedia.org/wiki/Array_programming) * [Nondeterministic](https://en.wikipedia.org/wiki/Nondeterministic_programming) * [Parallel computing](https://en.wikipedia.org/wiki/Parallel_computing)   + [Process-oriented](https://en.wikipedia.org/wiki/Process-oriented_programming) * [Probabilistic](https://en.wikipedia.org/wiki/Probabilistic_programming_language) * [Quantum](https://en.wikipedia.org/wiki/Quantum_programming) * [Set-theoretic](https://en.wikipedia.org/wiki/Set_theoretic_programming) * [Stack-based](https://en.wikipedia.org/wiki/Stack-oriented_programming) * [Structured](https://en.wikipedia.org/wiki/Structured_programming) (contrast: [Non-structured](https://en.wikipedia.org/wiki/Non-structured_programming))   + [Block-structured](https://en.wikipedia.org/wiki/Block_(programming))     - [Structured concurrency](https://en.wikipedia.org/wiki/Structured_concurrency)   + [Object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming)     - [Actor-based](https://en.wikipedia.org/wiki/Actor_model)     - [Class-based](https://en.wikipedia.org/wiki/Class-based_programming)     - [Concurrent](https://en.wikipedia.org/wiki/Concurrent_object-oriented_programming)     - [Prototype-based](https://en.wikipedia.org/wiki/Prototype-based_programming)     - By [separation of concerns](https://en.wikipedia.org/wiki/Separation_of_concerns):       * [Aspect-oriented](https://en.wikipedia.org/wiki/Aspect-oriented_programming)       * [Role-oriented](https://en.wikipedia.org/wiki/Role-oriented_programming)       * [Subject-oriented](https://en.wikipedia.org/wiki/Subject-oriented_programming)   + [Recursive](https://en.wikipedia.org/wiki/Recursion_(computer_science)) * [Symbolic](https://en.wikipedia.org/wiki/Symbolic_programming) * [Value-level](https://en.wikipedia.org/wiki/Value-level_programming) (contrast: [Function-level](https://en.wikipedia.org/wiki/Function-level_programming)) |
| * [v](https://en.wikipedia.org/wiki/Template:Programming_paradigms) * [t](https://en.wikipedia.org/wiki/Template_talk:Programming_paradigms) * [e](https://en.wikipedia.org/w/index.php?title=Template:Programming_paradigms&action=edit) |

In [computer science](https://en.wikipedia.org/wiki/Computer_science), **functional programming** is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) where programs are constructed by [applying](https://en.wikipedia.org/wiki/Function_application) and [composing](https://en.wikipedia.org/wiki/Function_composition_(computer_science)) [functions](https://en.wikipedia.org/wiki/Function_(computer_science)). It is a [declarative programming](https://en.wikipedia.org/wiki/Declarative_programming) paradigm in which function definitions are [trees](https://en.wikipedia.org/wiki/Tree_(data_structure)) of [expressions](https://en.wikipedia.org/wiki/Expression_(computer_science)) that each return a [value](https://en.wikipedia.org/wiki/Value_(computer_science)), rather than a sequence of [imperative](https://en.wikipedia.org/wiki/Imperative_programming) [statements](https://en.wikipedia.org/wiki/Statement_(computer_science)) which change the [state](https://en.wikipedia.org/wiki/State_(computer_science)) of the program.

In functional programming, functions are treated as [first-class citizens](https://en.wikipedia.org/wiki/First-class_citizen), meaning that they can be bound to names (including local [identifiers](https://en.wikipedia.org/wiki/Identifier_(computer_languages))), passed as [arguments](https://en.wikipedia.org/wiki/Parameter_(computer_programming)), and [returned](https://en.wikipedia.org/wiki/Return_value) from other functions, just as any other [data type](https://en.wikipedia.org/wiki/Data_type) can. This allows programs to be written in a [declarative](https://en.wikipedia.org/wiki/Declarative_programming) and [composable](https://en.wikipedia.org/wiki/Composability) style, where small functions are combined in a [modular](https://en.wikipedia.org/wiki/Modular_programming) manner.

Functional programming is sometimes treated as synonymous with [purely functional programming](https://en.wikipedia.org/wiki/Purely_functional_programming), a subset of functional programming which treats all functions as [deterministic](https://en.wikipedia.org/wiki/Deterministic_system) mathematical [functions](https://en.wikipedia.org/wiki/Function_(mathematics)), or [pure functions](https://en.wikipedia.org/wiki/Pure_function). When a pure function is called with some given arguments, it will always return the same result, and cannot be affected by any mutable [state](https://en.wikipedia.org/wiki/State_(computer_science)) or other [side effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)). This is in contrast with impure [procedures](https://en.wikipedia.org/wiki/Procedure_(computer_science)), common in [imperative programming](https://en.wikipedia.org/wiki/Imperative_programming), which can have side effects (such as modifying the program's state or taking input from a user). Proponents of purely functional programming claim that by restricting side effects, programs can have fewer [bugs](https://en.wikipedia.org/wiki/Software_bug), be easier to [debug](https://en.wikipedia.org/wiki/Debugging) and [test](https://en.wikipedia.org/wiki/Software_testing), and be more suited to [formal verification](https://en.wikipedia.org/wiki/Formal_verification).[[1]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-hudak1989-1)[[2]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-hughesWhyFPMatters-2)

Functional programming has its roots in [academia](https://en.wikipedia.org/wiki/Academia), evolving from the [lambda calculus](https://en.wikipedia.org/wiki/Lambda_calculus), a formal system of computation based only on functions. Functional programming has historically been less popular than imperative programming, but many functional languages are seeing use today in industry and education, including [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp), [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)),[[3]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-clinger1987-3)[[4]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-hartheimer1987-4)[[5]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-kidd2007-5)[[6]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-cleis2006-6) [Clojure](https://en.wikipedia.org/wiki/Clojure), [Wolfram Language](https://en.wikipedia.org/wiki/Wolfram_Language),[[7]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-reference.wolfram.com-7)[[8]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-Amath-CO-8) [Racket](https://en.wikipedia.org/wiki/Racket_(programming_language)),[[9]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-racket-video-games-9) [Erlang](https://en.wikipedia.org/wiki/Erlang_(programming_language)),[[10]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-erlang-faq-10)[[11]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-armstrong2007-11)[[12]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-larson2009-12) [OCaml](https://en.wikipedia.org/wiki/OCaml),[[13]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-minksy2008-13)[[14]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-leroy2007-14) [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)),[[15]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-haskell-industry-15)[[16]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-hudak2007-16) and [F#](https://en.wikipedia.org/wiki/F_Sharp_(programming_language)).[[17]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-quantFSharp-17)[[18]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-businessAppsFSharp-18) Functional programming is also key to some languages that have found success in specific domains, like [R](https://en.wikipedia.org/wiki/R_(programming_language)) in statistics,[[19]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-useR-19)[[20]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-Chambers-20) [J](https://en.wikipedia.org/wiki/J_(programming_language)), [K](https://en.wikipedia.org/wiki/K_(programming_language)) and [Q](https://en.wikipedia.org/wiki/Q_(programming_language_from_Kx_Systems)) in financial analysis, and [XQuery](https://en.wikipedia.org/wiki/XQuery)/[XSLT](https://en.wikipedia.org/wiki/XSLT) for [XML](https://en.wikipedia.org/wiki/XML).[[21]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-Novatchev-21)[[22]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-Mertz-22) Domain-specific declarative languages like [SQL](https://en.wikipedia.org/wiki/SQL) and [Lex](https://en.wikipedia.org/wiki/Lex_(software))/[Yacc](https://en.wikipedia.org/wiki/Yacc) use some elements of functional programming, such as not allowing [mutable values](https://en.wikipedia.org/wiki/Mutable_object).[[23]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-Chamberlin_Boyce-23) In addition, many other programming languages support programming in a functional style or have implemented features from functional programming, such as [C++11](https://en.wikipedia.org/wiki/C%2B%2B11), [Kotlin](https://en.wikipedia.org/wiki/Kotlin_(programming_language)),[[24]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-:0-24) [Perl](https://en.wikipedia.org/wiki/Perl),[[25]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-25) [PHP](https://en.wikipedia.org/wiki/PHP),[[26]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-26) [Python](https://en.wikipedia.org/wiki/Python_(programming_language)),[[27]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-AutoNT-13-27) [Raku](https://en.wikipedia.org/wiki/Raku_(programming_language)),[[28]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-28) and [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)).[[29]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-effective-scala-29)



**Contents**

* [1History](https://en.wikipedia.org/wiki/Functional_programming#History)
* [2Concepts](https://en.wikipedia.org/wiki/Functional_programming#Concepts)
  + [2.1First-class and higher-order functions](https://en.wikipedia.org/wiki/Functional_programming#First-class_and_higher-order_functions)
  + [2.2Pure functions](https://en.wikipedia.org/wiki/Functional_programming#Pure_functions)
  + [2.3Recursion](https://en.wikipedia.org/wiki/Functional_programming#Recursion)
  + [2.4Strict versus non-strict evaluation](https://en.wikipedia.org/wiki/Functional_programming#Strict_versus_non-strict_evaluation)
  + [2.5Type systems](https://en.wikipedia.org/wiki/Functional_programming#Type_systems)
  + [2.6Referential transparency](https://en.wikipedia.org/wiki/Functional_programming#Referential_transparency)
  + [2.7Data structures](https://en.wikipedia.org/wiki/Functional_programming#Data_structures)
* [3Comparison to imperative programming](https://en.wikipedia.org/wiki/Functional_programming#Comparison_to_imperative_programming)
  + [3.1Side-by-side comparison of Imperative vs. Functional programming](https://en.wikipedia.org/wiki/Functional_programming#Side-by-side_comparison_of_Imperative_vs._Functional_programming)
  + [3.2Simulating state](https://en.wikipedia.org/wiki/Functional_programming#Simulating_state)
  + [3.3Efficiency issues](https://en.wikipedia.org/wiki/Functional_programming#Efficiency_issues)
  + [3.4Functional programming in non-functional languages](https://en.wikipedia.org/wiki/Functional_programming#Functional_programming_in_non-functional_languages)
* [4Applications](https://en.wikipedia.org/wiki/Functional_programming#Applications)
  + [4.1Academia](https://en.wikipedia.org/wiki/Functional_programming#Academia)
  + [4.2Industry](https://en.wikipedia.org/wiki/Functional_programming#Industry)
  + [4.3Education](https://en.wikipedia.org/wiki/Functional_programming#Education)
* [5See also](https://en.wikipedia.org/wiki/Functional_programming#See_also)
* [6References](https://en.wikipedia.org/wiki/Functional_programming#References)
* [7Further reading](https://en.wikipedia.org/wiki/Functional_programming#Further_reading)
* [8External links](https://en.wikipedia.org/wiki/Functional_programming#External_links)

History[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=1)]

The [lambda calculus](https://en.wikipedia.org/wiki/Lambda_calculus), developed in the 1930s by [Alonzo Church](https://en.wikipedia.org/wiki/Alonzo_Church), is a [formal system](https://en.wikipedia.org/wiki/Formal_system) of [computation](https://en.wikipedia.org/wiki/Computation) built from [function application](https://en.wikipedia.org/wiki/Function_application). In 1937 [Alan Turing](https://en.wikipedia.org/wiki/Alan_Turing) proved that the lambda calculus and [Turing machines](https://en.wikipedia.org/wiki/Turing_machines) are equivalent models of computation,[[30]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-30) showing that the lambda calculus is [Turing complete](https://en.wikipedia.org/wiki/Turing_complete). Lambda calculus forms the basis of all functional programming languages. An equivalent theoretical formulation, [combinatory logic](https://en.wikipedia.org/wiki/Combinatory_logic), was developed by [Moses Schönfinkel](https://en.wikipedia.org/wiki/Moses_Sch%C3%B6nfinkel) and [Haskell Curry](https://en.wikipedia.org/wiki/Haskell_Curry) in the 1920s and 1930s.[[31]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-31)

Church later developed a weaker system, the [simply-typed lambda calculus](https://en.wikipedia.org/wiki/Simply-typed_lambda_calculus), which extended the lambda calculus by assigning a [type](https://en.wikipedia.org/wiki/Type_(computer_science)) to all terms.[[32]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-32) This forms the basis for statically-typed functional programming.

The first functional programming language, [LISP](https://en.wikipedia.org/wiki/Lisp_(programming_language)), was developed in the late 1950s for the [IBM 700/7000 series](https://en.wikipedia.org/wiki/IBM_700/7000_series#Scientific_Architecture) of scientific computers by [John McCarthy](https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)) while at [Massachusetts Institute of Technology](https://en.wikipedia.org/wiki/Massachusetts_Institute_of_Technology) (MIT).[[33]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-33) LISP functions were defined using Church's lambda notation, extended with a label construct to allow [recursive](https://en.wikipedia.org/wiki/Recursion_(computer_science)) functions.[[34]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-34) Lisp first introduced many paradigmatic features of functional programming, though early Lisps were [multi-paradigm languages](https://en.wikipedia.org/wiki/Programming_paradigm#Multi-paradigm), and incorporated support for numerous programming styles as new paradigms evolved. Later dialects, such as [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)) and [Clojure](https://en.wikipedia.org/wiki/Clojure), and offshoots such as [Dylan](https://en.wikipedia.org/wiki/Dylan_(programming_language)) and [Julia](https://en.wikipedia.org/wiki/Julia_(programming_language)), sought to simplify and rationalise Lisp around a cleanly functional core, while [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp) was designed to preserve and update the paradigmatic features of the numerous older dialects it replaced.[[35]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-35)

[Information Processing Language](https://en.wikipedia.org/wiki/Information_Processing_Language) (IPL), 1956, is sometimes cited as the first computer-based functional programming language.[[36]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-36) It is an [assembly-style language](https://en.wikipedia.org/wiki/Assembly_language) for manipulating lists of symbols. It does have a notion of *generator*, which amounts to a function that accepts a function as an argument, and, since it is an assembly-level language, code can be data, so IPL can be regarded as having higher-order functions. However, it relies heavily on mutating list structure and similar imperative features.

[Kenneth E. Iverson](https://en.wikipedia.org/wiki/Kenneth_E._Iverson) developed [APL](https://en.wikipedia.org/wiki/APL_(programming_language)) in the early 1960s, described in his 1962 book *A Programming Language* ([ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [9780471430148](https://en.wikipedia.org/wiki/Special:BookSources/9780471430148)). APL was the primary influence on [John Backus](https://en.wikipedia.org/wiki/John_Backus)'s [FP](https://en.wikipedia.org/wiki/FP_(programming_language)). In the early 1990s, Iverson and [Roger Hui](https://en.wikipedia.org/wiki/Roger_Hui) created [J](https://en.wikipedia.org/wiki/J_(programming_language)). In the mid-1990s, [Arthur Whitney](https://en.wikipedia.org/wiki/Arthur_Whitney_(computer_scientist)), who had previously worked with Iverson, created [K](https://en.wikipedia.org/wiki/K_(programming_language)), which is used commercially in financial industries along with its descendant [Q](https://en.wikipedia.org/wiki/Q_(programming_language_from_Kx_Systems)).

[John Backus](https://en.wikipedia.org/wiki/John_Backus) presented [FP](https://en.wikipedia.org/wiki/FP_(programming_language)) in his 1977 [Turing Award](https://en.wikipedia.org/wiki/Turing_Award) lecture "Can Programming Be Liberated From the [von Neumann](https://en.wikipedia.org/wiki/Von_Neumann_architecture) Style? A Functional Style and its Algebra of Programs".[[37]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-Backus_1977-37) He defines functional programs as being built up in a hierarchical way by means of "combining forms" that allow an "algebra of programs"; in modern language, this means that functional programs follow the [principle of compositionality](https://en.wikipedia.org/wiki/Principle_of_compositionality).[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] Backus's paper popularized research into functional programming, though it emphasized [function-level programming](https://en.wikipedia.org/wiki/Function-level_programming) rather than the lambda-calculus style now associated with functional programming.

The 1973 language [ML](https://en.wikipedia.org/wiki/ML_(programming_language)) was created by [Robin Milner](https://en.wikipedia.org/wiki/Robin_Milner) at the [University of Edinburgh](https://en.wikipedia.org/wiki/University_of_Edinburgh), and [David Turner](https://en.wikipedia.org/wiki/David_Turner_(computer_scientist)) developed the language [SASL](https://en.wikipedia.org/wiki/SASL_(programming_language)) at the [University of St Andrews](https://en.wikipedia.org/wiki/University_of_St_Andrews). Also in Edinburgh in the 1970s, Burstall and Darlington developed the functional language [NPL](https://en.wikipedia.org/wiki/NPL_(programming_language)).[[38]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-38) NPL was based on [Kleene Recursion Equations](https://en.wikipedia.org/wiki/Kleene%27s_recursion_theorem) and was first introduced in their work on program transformation.[[39]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-39) Burstall, MacQueen and Sannella then incorporated the polymorphic type checking from ML to produce the language [Hope](https://en.wikipedia.org/wiki/Hope_(programming_language)).[[40]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-40) ML eventually developed into several dialects, the most common of which are now [OCaml](https://en.wikipedia.org/wiki/OCaml) and [Standard ML](https://en.wikipedia.org/wiki/Standard_ML).

In the 1970s, [Guy L. Steele](https://en.wikipedia.org/wiki/Guy_L._Steele) and [Gerald Jay Sussman](https://en.wikipedia.org/wiki/Gerald_Jay_Sussman) developed [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)), as described in the [Lambda Papers](https://en.wikipedia.org/wiki/Lambda_Papers) and the 1985 textbook [*Structure and Interpretation of Computer Programs*](https://en.wikipedia.org/wiki/Structure_and_Interpretation_of_Computer_Programs). Scheme was the first dialect of lisp to use [lexical scoping](https://en.wikipedia.org/wiki/Lexical_scope) and to require [tail-call optimization](https://en.wikipedia.org/wiki/Tail-call_optimization), features that encourage functional programming.

In the 1980s, [Per Martin-Löf](https://en.wikipedia.org/wiki/Per_Martin-L%C3%B6f) developed [intuitionistic type theory](https://en.wikipedia.org/wiki/Intuitionistic_type_theory) (also called *constructive* type theory), which associated functional programs with [constructive proofs](https://en.wikipedia.org/wiki/Constructive_proof) expressed as [dependent types](https://en.wikipedia.org/wiki/Dependent_type). This led to new approaches to [interactive theorem proving](https://en.wikipedia.org/wiki/Interactive_theorem_proving) and has influenced the development of subsequent functional programming languages.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

The lazy functional language, [Miranda](https://en.wikipedia.org/wiki/Miranda_(programming_language)), developed by David Turner, initially appeared in 1985 and had a strong influence on [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)). With Miranda being proprietary, Haskell began with a consensus in 1987 to form an [open standard](https://en.wikipedia.org/wiki/Open_standard) for functional programming research; implementation releases have been ongoing since 1990.

More recently it has found use in niches such as parametric [CAD](https://en.wikipedia.org/wiki/Computer_Aided_Design) courtesy of the [OpenSCAD](https://en.wikipedia.org/wiki/OpenSCAD) language built on the CSG geometry framework, although its restriction on reassigning values (all values are treated as constants) has led to confusion among users who are unfamiliar with functional programming as a concept.[[41]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-41)

Functional programming continues to be used in commercial settings.[[42]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-42)[[43]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-43)[[44]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-44)

Concepts[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=2)]

A number of concepts and paradigms are specific to functional programming, and generally foreign to [imperative programming](https://en.wikipedia.org/wiki/Imperative_programming) (including [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming)). However, programming languages often cater to several programming paradigms, so programmers using "mostly imperative" languages may have utilized some of these concepts.[[45]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-45)

**First-class and higher-order functions**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=3)]

*Main articles:*[*First-class function*](https://en.wikipedia.org/wiki/First-class_function)*and*[*Higher-order function*](https://en.wikipedia.org/wiki/Higher-order_function)

[Higher-order functions](https://en.wikipedia.org/wiki/Higher-order_function) are functions that can either take other functions as arguments or return them as results. In calculus, an example of a higher-order function is the [differential operator](https://en.wikipedia.org/wiki/Differential_operator) {\displaystyle d/dx}, which returns the [derivative](https://en.wikipedia.org/wiki/Derivative) of a function {\displaystyle f}.

Higher-order functions are closely related to [first-class functions](https://en.wikipedia.org/wiki/First-class_function) in that higher-order functions and first-class functions both allow functions as arguments and results of other functions. The distinction between the two is subtle: "higher-order" describes a mathematical concept of functions that operate on other functions, while "first-class" is a computer science term for programming language entities that have no restriction on their use (thus first-class functions can appear anywhere in the program that other first-class entities like numbers can, including as arguments to other functions and as their return values).

Higher-order functions enable [partial application](https://en.wikipedia.org/wiki/Partial_application) or [currying](https://en.wikipedia.org/wiki/Currying), a technique that applies a function to its arguments one at a time, with each application returning a new function that accepts the next argument. This lets a programmer succinctly express, for example, the [successor function](https://en.wikipedia.org/wiki/Successor_function) as the addition operator partially applied to the [natural number](https://en.wikipedia.org/wiki/Natural_number) one.

**Pure functions**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=4)]

*Main article:*[*Pure function*](https://en.wikipedia.org/wiki/Pure_function)

[Pure functions](https://en.wikipedia.org/wiki/Pure_function) (or expressions) have no [side effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)) (memory or I/O). This means that pure functions have several useful properties, many of which can be used to optimize the code:

* If the result of a pure expression is not used, it can be removed without affecting other expressions.
* If a pure function is called with arguments that cause no side-effects, the result is constant with respect to that argument list (sometimes called [referential transparency](https://en.wikipedia.org/wiki/Referential_transparency)), i.e., calling the pure function again with the same arguments returns the same result. (This can enable caching optimizations such as [memoization](https://en.wikipedia.org/wiki/Memoization).)
* If there is no data dependency between two pure expressions, their order can be reversed, or they can be performed in [parallel](https://en.wikipedia.org/wiki/Parallelization) and they cannot interfere with one another (in other terms, the evaluation of any pure expression is [thread-safe](https://en.wikipedia.org/wiki/Thread-safe)).
* If the entire language does not allow side-effects, then any evaluation strategy can be used; this gives the compiler freedom to reorder or combine the evaluation of expressions in a program (for example, using [deforestation](https://en.wikipedia.org/wiki/Deforestation_(computer_science))).

While most compilers for imperative programming languages detect pure functions and perform common-subexpression elimination for pure function calls, they cannot always do this for pre-compiled libraries, which generally do not expose this information, thus preventing optimizations that involve those external functions. Some compilers, such as [gcc](https://en.wikipedia.org/wiki/GNU_Compiler_Collection), add extra keywords for a programmer to explicitly mark external functions as pure, to enable such optimizations. [Fortran 95](https://en.wikipedia.org/wiki/Fortran_95) also lets functions be designated *pure*.[[46]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-fortran95-46) C++11 added constexpr keyword with similar semantics.

**Recursion**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=5)]

*Main article:*[*Recursion (computer science)*](https://en.wikipedia.org/wiki/Recursion_(computer_science))

[Iteration](https://en.wikipedia.org/wiki/Iteration) (looping) in functional languages is usually accomplished via [recursion](https://en.wikipedia.org/wiki/Recursion). [Recursive functions](https://en.wikipedia.org/wiki/Recursion_(computer_science)) invoke themselves, letting an operation be repeated until it reaches the [base case](https://en.wikipedia.org/wiki/Recursion_(computer_science)). In general, recursion requires maintaining a [stack](https://en.wikipedia.org/wiki/Call_stack), which consumes space in a linear amount to the depth of recursion. This could make recursion prohibitively expensive to use instead of imperative loops. However, a special form of recursion known as [tail recursion](https://en.wikipedia.org/wiki/Tail_recursion) can be recognized and optimized by a compiler into the same code used to implement iteration in imperative languages. Tail recursion optimization can be implemented by transforming the program into [continuation passing style](https://en.wikipedia.org/wiki/Continuation_passing_style) during compiling, among other approaches.

The [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)) language standard requires implementations to support proper tail recursion, meaning they must allow an unbounded number of active tail calls.[[47]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-SchemeProperTailRec-47)[[48]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-48) Proper tail recursion is not simply an optimization; it is a language feature that assures users that they can use recursion to express a loop and doing so would be safe-for-space.[[49]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-49) Moreover, on contrary to its name, it accounts for all tail calls, not just tail recursion. While proper tail recursion is usually implemented by turning code into imperative loops, implementations might implement it in other ways. For example, [CHICKEN](https://en.wikipedia.org/wiki/Chicken_(Scheme_implementation)) intentionally maintains a stack and lets the [stack overflow](https://en.wikipedia.org/wiki/Stack_overflow). However, when this happens, its [garbage collector](https://en.wikipedia.org/wiki/Garbage_collector) will claim space back,[[50]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-50) allowing an unbounded number of active tail calls even though it does not turn tail recursion into a loop.

Common patterns of recursion can be abstracted away using higher-order functions, with [catamorphisms](https://en.wikipedia.org/wiki/Catamorphism) and [anamorphisms](https://en.wikipedia.org/wiki/Anamorphism) (or "folds" and "unfolds") being the most obvious examples. Such recursion schemes play a role analogous to built-in control structures such as [loops](https://en.wikipedia.org/wiki/Program_loops) in [imperative languages](https://en.wikipedia.org/wiki/Imperative_languages).

Most general purpose functional programming languages allow unrestricted recursion and are [Turing complete](https://en.wikipedia.org/wiki/Turing_complete), which makes the [halting problem](https://en.wikipedia.org/wiki/Halting_problem) [undecidable](https://en.wikipedia.org/wiki/Undecidable_problem), can cause unsoundness of [equational reasoning](https://en.wikipedia.org/wiki/Equational_reasoning), and generally requires the introduction of [inconsistency](https://en.wikipedia.org/wiki/Inconsistency) into the logic expressed by the language's [type system](https://en.wikipedia.org/wiki/Type_system). Some special purpose languages such as [Coq](https://en.wikipedia.org/wiki/Coq) allow only [well-founded](https://en.wikipedia.org/wiki/Well-founded) recursion and are [strongly normalizing](https://en.wikipedia.org/wiki/Strongly_normalizing) (nonterminating computations can be expressed only with infinite streams of values called [codata](https://en.wikipedia.org/wiki/Codata_(computer_science))). As a consequence, these languages fail to be Turing complete and expressing certain functions in them is impossible, but they can still express a wide class of interesting computations while avoiding the problems introduced by unrestricted recursion. Functional programming limited to well-founded recursion with a few other constraints is called [total functional programming](https://en.wikipedia.org/wiki/Total_functional_programming).[[51]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-totalfp-51)

**Strict versus non-strict evaluation**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=6)]

*Main article:*[*Evaluation strategy*](https://en.wikipedia.org/wiki/Evaluation_strategy)

Functional languages can be categorized by whether they use *strict (eager)* or *non-strict (lazy)* evaluation, concepts that refer to how function arguments are processed when an expression is being evaluated. The technical difference is in the [denotational semantics](https://en.wikipedia.org/wiki/Denotational_semantics) of expressions containing failing or divergent computations. Under strict evaluation, the evaluation of any term containing a failing subterm fails. For example, the expression:

print length([2+1, 3\*2, 1/0, 5-4])

fails under strict evaluation because of the division by zero in the third element of the list. Under lazy evaluation, the length function returns the value 4 (i.e., the number of items in the list), since evaluating it does not attempt to evaluate the terms making up the list. In brief, strict evaluation always fully evaluates function arguments before invoking the function. Lazy evaluation does not evaluate function arguments unless their values are required to evaluate the function call itself.

The usual implementation strategy for lazy evaluation in functional languages is [graph reduction](https://en.wikipedia.org/wiki/Graph_reduction).[[52]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-52) Lazy evaluation is used by default in several pure functional languages, including [Miranda](https://en.wikipedia.org/wiki/Miranda_(programming_language)), [Clean](https://en.wikipedia.org/wiki/Clean_(programming_language)), and [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)).

[Hughes 1984](https://en.wikipedia.org/wiki/Functional_programming#CITEREFHughes1984) argues for lazy evaluation as a mechanism for improving program modularity through [separation of concerns](https://en.wikipedia.org/wiki/Separation_of_concerns), by easing independent implementation of producers and consumers of data streams.[[2]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-hughesWhyFPMatters-2) Launchbury 1993 describes some difficulties that lazy evaluation introduces, particularly in analyzing a program's storage requirements, and proposes an [operational semantics](https://en.wikipedia.org/wiki/Operational_semantics) to aid in such analysis.[[53]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-launchbury1993-53) Harper 2009 proposes including both strict and lazy evaluation in the same language, using the language's type system to distinguish them.[[54]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-54)

**Type systems**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=7)]

*Main article:*[*Type system*](https://en.wikipedia.org/wiki/Type_system)

Especially since the development of [Hindley–Milner type inference](https://en.wikipedia.org/wiki/Hindley%E2%80%93Milner_type_inference) in the 1970s, functional programming languages have tended to use [typed lambda calculus](https://en.wikipedia.org/wiki/Typed_lambda_calculus), rejecting all invalid programs at compilation time and risking [false positive errors](https://en.wikipedia.org/wiki/False_positives_and_false_negatives#False_positive_error), as opposed to the [untyped lambda calculus](https://en.wikipedia.org/wiki/Untyped_lambda_calculus), that accepts all valid programs at compilation time and risks [false negative errors](https://en.wikipedia.org/wiki/False_positives_and_false_negatives#False_negative_error), used in Lisp and its variants (such as [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language))), though they reject all invalid programs at runtime, when the information is enough to not reject valid programs. The use of [algebraic datatypes](https://en.wikipedia.org/wiki/Algebraic_datatypes) makes manipulation of complex data structures convenient; the presence of strong compile-time type checking makes programs more reliable in absence of other reliability techniques like [test-driven development](https://en.wikipedia.org/wiki/Test-driven_development), while [type inference](https://en.wikipedia.org/wiki/Type_inference) frees the programmer from the need to manually declare types to the compiler in most cases.

Some research-oriented functional languages such as [Coq](https://en.wikipedia.org/wiki/Coq), [Agda](https://en.wikipedia.org/wiki/Agda_(theorem_prover)), [Cayenne](https://en.wikipedia.org/wiki/Cayenne_(programming_language)), and [Epigram](https://en.wikipedia.org/wiki/Epigram_(programming_language)) are based on [intuitionistic type theory](https://en.wikipedia.org/wiki/Intuitionistic_type_theory), which lets types depend on terms. Such types are called [dependent types](https://en.wikipedia.org/wiki/Dependent_type). These type systems do not have decidable type inference and are difficult to understand and program with.[[55]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-55)[[56]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-56)[[57]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-57)[[58]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-58) But dependent types can express arbitrary propositions in [predicate logic](https://en.wikipedia.org/wiki/Predicate_logic). Through the [Curry–Howard isomorphism](https://en.wikipedia.org/wiki/Curry%E2%80%93Howard_isomorphism), then, well-typed programs in these languages become a means of writing formal [mathematical proofs](https://en.wikipedia.org/wiki/Mathematical_proof) from which a compiler can generate [certified code](https://en.wikipedia.org/wiki/Formal_verification). While these languages are mainly of interest in academic research (including in [formalized mathematics](https://en.wikipedia.org/wiki/Formalized_mathematics)), they have begun to be used in engineering as well. [Compcert](https://en.wikipedia.org/wiki/Compcert) is a [compiler](https://en.wikipedia.org/wiki/Compiler) for a subset of the [C programming language](https://en.wikipedia.org/wiki/C_(programming_language)) that is written in Coq and formally verified.[[59]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-59)

A limited form of dependent types called [generalized algebraic data types](https://en.wikipedia.org/wiki/Generalized_algebraic_data_type) (GADT's) can be implemented in a way that provides some of the benefits of dependently typed programming while avoiding most of its inconvenience.[[60]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-60) GADT's are available in the [Glasgow Haskell Compiler](https://en.wikipedia.org/wiki/Glasgow_Haskell_Compiler), in [OCaml](https://en.wikipedia.org/wiki/OCaml) (since version 4.00) and in [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)) (as "case classes"), and have been proposed as additions to other languages including Java and C#.[[61]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-61)

**Referential transparency**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=8)]

*Main article:*[*Referential transparency*](https://en.wikipedia.org/wiki/Referential_transparency)

Functional programs do not have assignment statements, that is, the value of a variable in a functional program never changes once defined. This eliminates any chances of side effects because any variable can be replaced with its actual value at any point of execution. So, functional programs are referentially transparent.[[62]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-62)

Consider [C](https://en.wikipedia.org/wiki/C_(programming_language)) assignment statement x = x \* 10, this changes the value assigned to the variable x. Let us say that the initial value of x was 1, then two consecutive evaluations of the variable x yields 10 and 100 respectively. Clearly, replacing x = x \* 10 with either 10 or 100 gives a program with different meaning, and so the expression *is not* referentially transparent. In fact, assignment statements are never referentially transparent.

Now, consider another function such as int plusone(int x) {**return** x+1;} *is* transparent, as it does not implicitly change the input x and thus has no such [side effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)). Functional programs exclusively use this type of function and are therefore referentially transparent.

**Data structures**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=9)]

*Main article:*[*Purely functional data structure*](https://en.wikipedia.org/wiki/Purely_functional_data_structure)

Purely functional [data structures](https://en.wikipedia.org/wiki/Data_structure) are often represented in a different way than their [imperative](https://en.wikipedia.org/wiki/Imperative_programming) counterparts.[[63]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-63) For example, the [array](https://en.wikipedia.org/wiki/Array_data_structure) with constant access and update times is a basic component of most imperative languages, and many imperative data-structures, such as the [hash table](https://en.wikipedia.org/wiki/Hash_table) and [binary heap](https://en.wikipedia.org/wiki/Binary_heap), are based on arrays. Arrays can be replaced by [maps](https://en.wikipedia.org/wiki/Map_(computer_science)) or random access lists, which admit purely functional implementation, but have [logarithmic](https://en.wikipedia.org/wiki/Logarithm) access and update times. Purely functional data structures have [persistence](https://en.wikipedia.org/wiki/Persistent_data_structure), a property of keeping previous versions of the data structure unmodified. In Clojure, persistent data structures are used as functional alternatives to their imperative counterparts. Persistent vectors, for example, use trees for partial updating. Calling the insert method will result in some but not all nodes being created.[[64]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-64)

Comparison to imperative programming[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=10)]

Functional programming is very different from [imperative programming](https://en.wikipedia.org/wiki/Imperative_programming). The most significant differences stem from the fact that functional programming avoids [side effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)), which are used in imperative programming to implement state and I/O. Pure functional programming completely prevents side-effects and provides referential transparency.

Higher-order functions are rarely used in older imperative programming. A traditional imperative program might use a loop to traverse and modify a list. A functional program, on the other hand, would probably use a higher-order “map” function that takes a function and a list, generating and returning a new list by applying the function to each list item.

**Side-by-side comparison of Imperative vs. Functional programming**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=11)]

The following two examples (written in [JavaScript](https://en.wikipedia.org/wiki/JavaScript_(programming_language))) achieve the same effect - they multiply all even numbers in an array by 10 and add them all, storing the final sum in the variable "result".

Traditional Imperative Loop:

**const** numList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

**let** result = 0;

**for** (**let** i = 0; i < numList.length; i++) {

**if** (numList[i] % 2 === 0) {

result += numList[i] \* 10;

}

}

Functional Programming with higher-order functions:

**const** result = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

.filter(n => n % 2 === 0)

.map(a => a \* 10)

.reduce((a, b) => a + b);

**Simulating state**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=12)]

There are tasks (for example, maintaining a bank account balance) that often seem most naturally implemented with state. Pure functional programming performs these tasks, and I/O tasks such as accepting user input and printing to the screen, in a different way.

The pure functional programming language [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) implements them using [monads](https://en.wikipedia.org/wiki/Monad_(functional_programming)), derived from [category theory](https://en.wikipedia.org/wiki/Category_theory). Monads offer a way to abstract certain types of computational patterns, including (but not limited to) modeling of computations with mutable state (and other side effects such as I/O) in an imperative manner without losing purity. While existing monads may be easy to apply in a program, given appropriate templates and examples, many students find them difficult to understand conceptually, e.g., when asked to define new monads (which is sometimes needed for certain types of libraries).[[65]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-65)

Functional languages also simulate states by passing around immutable states. This can be done by making a function accept the state as one of its parameters, and return a new state together with the result, leaving the old state unchanged.[[66]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-66)

Impure functional languages usually include a more direct method of managing mutable state. [Clojure](https://en.wikipedia.org/wiki/Clojure), for example, uses managed references that can be updated by applying pure functions to the current state. This kind of approach enables mutability while still promoting the use of pure functions as the preferred way to express computations.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

Alternative methods such as [Hoare logic](https://en.wikipedia.org/wiki/Hoare_logic) and [uniqueness](https://en.wikipedia.org/wiki/Uniqueness_type) have been developed to track side effects in programs. Some modern research languages use [effect systems](https://en.wikipedia.org/wiki/Effect_system) to make the presence of side effects explicit.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

**Efficiency issues**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=13)]

Functional programming languages are typically less efficient in their use of [CPU](https://en.wikipedia.org/wiki/Central_processing_unit) and memory than imperative languages such as [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [Pascal](https://en.wikipedia.org/wiki/Pascal_(programming_language)).[[67]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-67) This is related to the fact that some mutable data structures like arrays have a very straightforward implementation using present hardware. Flat arrays may be accessed very efficiently with deeply pipelined CPUs, prefetched efficiently through caches (with no complex [pointer chasing](https://en.wikipedia.org/w/index.php?title=Pointer_chasing&action=edit&redlink=1)), or handled with SIMD instructions. It is also not easy to create their equally efficient general-purpose immutable counterparts. For purely functional languages, the worst-case slowdown is logarithmic in the number of memory cells used, because mutable memory can be represented by a purely functional data structure with logarithmic access time (such as a balanced tree).[[68]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-Spiewak-68) However, such slowdowns are not universal. For programs that perform intensive numerical computations, functional languages such as [OCaml](https://en.wikipedia.org/wiki/OCaml) and [Clean](https://en.wikipedia.org/wiki/Clean_(programming_language)) are only slightly slower than C according to [The Computer Language Benchmarks Game](https://en.wikipedia.org/wiki/The_Computer_Language_Benchmarks_Game).[[69]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-69) For programs that handle large [matrices](https://en.wikipedia.org/wiki/Matrix_(mathematics)) and multidimensional [databases](https://en.wikipedia.org/wiki/Database), [array](https://en.wikipedia.org/wiki/Array_programming) functional languages (such as [J](https://en.wikipedia.org/wiki/J_(programming_language)) and [K](https://en.wikipedia.org/wiki/K_(programming_language))) were designed with speed optimizations.

Immutability of data can in many cases lead to execution efficiency by allowing the compiler to make assumptions that are unsafe in an imperative language, thus increasing opportunities for [inline expansion](https://en.wikipedia.org/wiki/Inline_expansion).[[70]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-70)

[Lazy evaluation](https://en.wikipedia.org/wiki/Lazy_evaluation) may also speed up the program, even asymptotically, whereas it may slow it down at most by a constant factor (however, it may introduce [memory leaks](https://en.wikipedia.org/wiki/Memory_leak) if used improperly). Launchbury 1993[[53]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-launchbury1993-53) discusses theoretical issues related to memory leaks from lazy evaluation, and O'Sullivan *et al.* 2008[[71]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-71) give some practical advice for analyzing and fixing them. However, the most general implementations of lazy evaluation making extensive use of dereferenced code and data perform poorly on modern processors with deep pipelines and multi-level caches (where a cache miss may cost hundreds of cycles)[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)].

**Functional programming in non-functional languages**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=14)]

It is possible to use a functional style of programming in languages that are not traditionally considered functional languages.[[72]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-72) For example, both [D](https://en.wikipedia.org/wiki/D_(programming_language))[[73]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-73) and [Fortran 95](https://en.wikipedia.org/wiki/Fortran_95)[[46]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-fortran95-46) explicitly support pure functions.

[JavaScript](https://en.wikipedia.org/wiki/JavaScript), [Lua](https://en.wikipedia.org/wiki/Lua_(programming_language))[[74]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-74) and [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) had [first class functions](https://en.wikipedia.org/wiki/First-class_function) from their inception.[[75]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-75) Python had support for "[lambda](https://en.wikipedia.org/wiki/Anonymous_function)", "[map](https://en.wikipedia.org/wiki/Map_(higher-order_function))", "[reduce](https://en.wikipedia.org/wiki/Fold_(higher-order_function))", and "[filter](https://en.wikipedia.org/wiki/Filter_(higher-order_function))" in 1994, as well as closures in Python 2.2,[[76]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-76) though Python 3 relegated "reduce" to the functools standard library module.[[77]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-77) First-class functions have been introduced into other mainstream languages such as [PHP](https://en.wikipedia.org/wiki/PHP) 5.3, [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic) 9, [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) 3.0, [C++11](https://en.wikipedia.org/wiki/C%2B%2B11), and [Kotlin](https://en.wikipedia.org/wiki/Kotlin_(programming_language)).[[24]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-:0-24)[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

In [PHP](https://en.wikipedia.org/wiki/PHP), [anonymous classes](https://en.wikipedia.org/wiki/Anonymous_class), [closures](https://en.wikipedia.org/wiki/Closure_(computer_science)) and lambdas are fully supported. Libraries and language extensions for immutable data structures are being developed to aid programming in the functional style.

In [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [anonymous classes](https://en.wikipedia.org/wiki/Anonymous_class) can sometimes be used to simulate [closures](https://en.wikipedia.org/wiki/Closure_(computer_science));[[78]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-78) however, anonymous classes are not always proper replacements to [closures](https://en.wikipedia.org/wiki/Closure_(computer_science)) because they have more limited capabilities.[[79]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-79) Java 8 supports lambda expressions as a replacement for some anonymous classes.[[80]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-80)

In [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [anonymous classes](https://en.wikipedia.org/wiki/Anonymous_class) are not necessary, because [closures](https://en.wikipedia.org/wiki/Closure_(computer_science)) and lambdas are fully supported. Libraries and language extensions for immutable data structures are being developed to aid programming in the functional style in C#.

Many [object-oriented](https://en.wikipedia.org/wiki/Object-oriented) [design patterns](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)) are expressible in functional programming terms: for example, the [strategy pattern](https://en.wikipedia.org/wiki/Strategy_pattern) simply dictates use of a higher-order function, and the [visitor](https://en.wikipedia.org/wiki/Visitor_(design_pattern)) pattern roughly corresponds to a [catamorphism](https://en.wikipedia.org/wiki/Catamorphism), or [fold](https://en.wikipedia.org/wiki/Fold_(higher-order_function)).

Similarly, the idea of immutable data from functional programming is often included in imperative programming languages,[[81]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-81) for example the tuple in Python, which is an immutable array.

Applications[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=15)]

**Academia**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=16)]

Functional programming is an active area of research in the field of [programming language theory](https://en.wikipedia.org/wiki/Programming_language_theory). There are several [peer-reviewed](https://en.wikipedia.org/wiki/Peer-review) publication venues focusing on functional programming, including the [International Conference on Functional Programming](https://en.wikipedia.org/wiki/International_Conference_on_Functional_Programming), the [Journal of Functional Programming](https://en.wikipedia.org/wiki/Journal_of_Functional_Programming), and the [Symposium on Trends in Functional Programming](https://en.wikipedia.org/wiki/Symposium_on_Trends_in_Functional_Programming).

**Industry**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=17)]

Functional programming has seen use in a wide variety of industrial applications. For example, [Erlang](https://en.wikipedia.org/wiki/Erlang_(programming_language)), which was developed by the [Swedish](https://en.wikipedia.org/wiki/Sweden) company [Ericsson](https://en.wikipedia.org/wiki/Ericsson) in the late 1980s, was originally used to implement [fault-tolerant](https://en.wikipedia.org/wiki/Fault_tolerance) [telecommunications](https://en.wikipedia.org/wiki/Telecommunication) systems,[[11]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-armstrong2007-11) but has since become popular for building a range of applications at companies such as [Nortel](https://en.wikipedia.org/wiki/Nortel), [Facebook](https://en.wikipedia.org/wiki/Facebook), [Électricité de France](https://en.wikipedia.org/wiki/%C3%89lectricit%C3%A9_de_France) and [WhatsApp](https://en.wikipedia.org/wiki/WhatsApp).[[10]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-erlang-faq-10)[[12]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-larson2009-12)[[82]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-82)[[83]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-Sim-Diasca-83)[[84]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-whatsapp.blog.2012-84) [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)), a dialect of [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)), was used as the basis for several applications on early [Apple Macintosh](https://en.wikipedia.org/wiki/Apple_Macintosh) computers,[[3]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-clinger1987-3)[[4]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-hartheimer1987-4) and has been applied to problems such as training [simulation software](https://en.wikipedia.org/wiki/Computer_simulation)[[5]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-kidd2007-5) and [telescope](https://en.wikipedia.org/wiki/Telescope) control.[[6]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-cleis2006-6) [OCaml](https://en.wikipedia.org/wiki/OCaml), which was introduced in the mid-1990s, has seen commercial use in areas such as financial analysis,[[13]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-minksy2008-13) [driver](https://en.wikipedia.org/wiki/Software_driver) verification, industrial [robot](https://en.wikipedia.org/wiki/Robot) programming, and static analysis of [embedded software](https://en.wikipedia.org/wiki/Embedded_software).[[14]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-leroy2007-14) [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)), though initially intended as a research language,[[16]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-hudak2007-16) has also been applied by a range of companies, in areas such as aerospace systems, hardware design, and web programming.[[15]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-haskell-industry-15)[[16]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-hudak2007-16)

Other functional programming languages that have seen use in industry include [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)),[[85]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-85) [F#](https://en.wikipedia.org/wiki/F_Sharp_(programming_language)),[[17]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-quantFSharp-17)[[18]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-businessAppsFSharp-18) [Wolfram Language](https://en.wikipedia.org/wiki/Wolfram_Language),[[7]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-reference.wolfram.com-7) [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)),[[86]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-86) [Standard ML](https://en.wikipedia.org/wiki/Standard_ML),[[87]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-87)[[88]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-88) and [Clojure](https://en.wikipedia.org/wiki/Clojure).[[89]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-89)

Functional "platforms" have been popular in finance for risk analytics (particularly with the larger investment banks). Risk factors are coded as functions that form interdependent graphs (categories) to measure correlations in market shifts not unlike [Gröbner basis](https://en.wikipedia.org/wiki/Gr%C3%B6bner_basis) optimizations but also for regulatory compliance such as [Comprehensive Capital Analysis and Review](https://en.wikipedia.org/wiki/Comprehensive_Capital_Analysis_and_Review). Given the use of [OCAML](https://en.wikipedia.org/wiki/OCAML) or [CAML](https://en.wikipedia.org/wiki/Caml) variations in finance, these systems are sometimes considered related to a [categorical abstract machine](https://en.wikipedia.org/wiki/Categorical_abstract_machine) or CAM. Indeed, functional programming is heavily influenced by [category theory](https://en.wikipedia.org/wiki/Category_theory).

**Education**[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=18)]

Many [universities](https://en.wikipedia.org/wiki/University) teach or have taught functional programming as part of their [undergraduate](https://en.wikipedia.org/wiki/Undergraduate_degree) Computer Science degrees.[[90]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-oxfordFP-90)[[91]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-imperialFP-91)[[92]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-nottinghamFP-92)[[93]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-mitFP-93) Some use it as their introduction to programming,[[93]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-mitFP-93) while others teach it after teaching imperative programming.[[92]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-nottinghamFP-92)[[94]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-61A-94)

Outside of computer science, functional programming is being used as a method to teach problem solving, algebra and geometric concepts.[[95]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-bootstrapworld-95) It has also been used as a tool to teach classical mechanics in [*Structure and Interpretation of Classical Mechanics*](https://en.wikipedia.org/wiki/Structure_and_Interpretation_of_Classical_Mechanics).

See also[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=19)]

* ***[icon](https://en.wikipedia.org/wiki/File:Octicons-terminal.svg)***[***Computer programming portal***](https://en.wikipedia.org/wiki/Portal:Computer_programming)
* [Purely functional programming](https://en.wikipedia.org/wiki/Purely_functional_programming)
* [Comparison of programming paradigms](https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms)
* [Eager evaluation](https://en.wikipedia.org/wiki/Eager_evaluation)
* [List of functional programming topics](https://en.wikipedia.org/wiki/List_of_functional_programming_topics)
* [Nested function](https://en.wikipedia.org/wiki/Nested_function)
* [Inductive functional programming](https://en.wikipedia.org/wiki/Inductive_functional_programming)
* [Functional reactive programming](https://en.wikipedia.org/wiki/Functional_reactive_programming)

References[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=20)]

* 1. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-hudak1989_1-0) [*Hudak, Paul*](https://en.wikipedia.org/wiki/Paul_Hudak)*(September 1989).*[*"Conception, evolution, and application of functional programming languages"*](http://www.dbnet.ece.ntua.gr/~adamo/languages/books/p359-hudak.pdf)*(PDF). ACM Computing Surveys.****21****(3): 359–411.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/72551.72554*](https://doi.org/10.1145%2F72551.72554)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*207637854*](https://api.semanticscholar.org/CorpusID:207637854)*.*
  2. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-hughesWhyFPMatters_2-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-hughesWhyFPMatters_2-1) [*Hughes, John*](https://en.wikipedia.org/wiki/John_Hughes_(computer_scientist))*(1984).*[*"Why Functional Programming Matters"*](http://www.cse.chalmers.se/~rjmh/Papers/whyfp.html)*.*
  3. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-clinger1987_3-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-clinger1987_3-1) *Clinger, Will (1987).*[*"MultiTasking and MacScheme"*](http://www.mactech.com/articles/mactech/Vol.03/03.12/Multitasking/index.html)*. MacTech.****3****(12). Retrieved 2008-08-28.*
  4. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-hartheimer1987_4-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-hartheimer1987_4-1) *Hartheimer, Anne (1987).*[*"Programming a Text Editor in MacScheme+Toolsmith"*](https://web.archive.org/web/20110629183752/http:/www.mactech.com/articles/mactech/Vol.03/03.1/SchemeWindows/index.html)*. MacTech.****3****(1). Archived from*[*the original*](http://www.mactech.com/articles/mactech/Vol.03/03.1/SchemeWindows/index.html)*on 2011-06-29. Retrieved 2008-08-28.*
  5. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-kidd2007_5-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-kidd2007_5-1) *Kidd, Eric.*[*Terrorism Response Training in Scheme*](http://cufp.galois.com/2007/abstracts.html#EricKidd)*. CUFP 2007. Retrieved 2009-08-26.*
  6. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-cleis2006_6-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-cleis2006_6-1) *Cleis, Richard.*[*Scheme in Space*](http://cufp.galois.com/2006/abstracts.html#RichardCleis)*. CUFP 2006. Retrieved 2009-08-26.*
  7. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-reference.wolfram.com_7-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-reference.wolfram.com_7-1) [*"Wolfram Language Guide: Functional Programming"*](http://reference.wolfram.com/language/guide/FunctionalProgramming.html)*. 2015. Retrieved 2015-08-24.*
  8. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-Amath-CO_8-0) [*"Functional vs. Procedural Programming Language"*](https://web.archive.org/web/20071113175801/http:/amath.colorado.edu/computing/mmm/funcproc.html)*. Department of Applied Math. University of Colorado. Archived from*[*the original*](http://amath.colorado.edu/computing/mmm/funcproc.html)*on 2007-11-13.*
  9. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-racket-video-games_9-0) [*"State-Based Scripting in Uncharted 2"*](https://web.archive.org/web/20121215014637/http:/www.gameenginebook.com/gdc09-statescripting-uncharted2.pdf)*(PDF). Archived from*[*the original*](http://www.gameenginebook.com/gdc09-statescripting-uncharted2.pdf)*(PDF) on 2012-12-15. Retrieved 2011-08-08.*
  10. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-erlang-faq_10-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-erlang-faq_10-1) [*"Who uses Erlang for product development?"*](http://erlang.org/faq/introduction.html#idp32582608)*. Frequently asked questions about Erlang. Retrieved 2018-04-27.*
  11. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-armstrong2007_11-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-armstrong2007_11-1) *Armstrong, Joe (June 2007). A history of Erlang. Third ACM SIGPLAN Conference on History of Programming Languages. San Diego, California.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/1238844.1238850*](https://doi.org/10.1145%2F1238844.1238850)*.*
  12. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-larson2009_12-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-larson2009_12-1) *Larson, Jim (March 2009). "Erlang for concurrent programming". Communications of the ACM.****52****(3): 48.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/1467247.1467263*](https://doi.org/10.1145%2F1467247.1467263)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*524392*](https://api.semanticscholar.org/CorpusID:524392)*.*
  13. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-minksy2008_13-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-minksy2008_13-1) *Minsky, Yaron; Weeks, Stephen (July 2008).*[*"Caml Trading — experiences with functional programming on Wall Street"*](http://journals.cambridge.org/action/displayAbstract?aid=1899164)*. Journal of Functional Programming.****18****(4): 553–564.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1017/S095679680800676X*](https://doi.org/10.1017%2FS095679680800676X)*. Retrieved 2008-08-27.*
  14. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-leroy2007_14-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-leroy2007_14-1) *Leroy, Xavier.*[*Some uses of Caml in Industry*](http://cufp.galois.com/2007/slides/XavierLeroy.pdf)*(PDF). CUFP 2007. Retrieved 2009-08-26.*
  15. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-haskell-industry_15-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-haskell-industry_15-1) [*"Haskell in industry"*](http://www.haskell.org/haskellwiki/Haskell_in_industry)*. Haskell Wiki. Retrieved 2009-08-26. Haskell has a diverse range of use commercially, from aerospace and defense, to finance, to web startups, hardware design firms and lawnmower manufacturers.*
  16. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-hudak2007_16-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-hudak2007_16-1) [***c***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-hudak2007_16-2) [*Hudak, Paul*](https://en.wikipedia.org/wiki/Paul_Hudak)*; Hughes, J.; Jones, S. P.; Wadler, P. (June 2007).*[*A history of Haskell: being lazy with class*](http://dl.acm.org/citation.cfm?doid=1238844.1238856)*. Third ACM SIGPLAN Conference on History of Programming Languages. San Diego, California.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/1238844.1238856*](https://doi.org/10.1145%2F1238844.1238856)*. Retrieved 2013-09-26.*
  17. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-quantFSharp_17-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-quantFSharp_17-1) *Mansell, Howard (2008).*[*Quantitative Finance in F#*](http://cufp.galois.com/2008/abstracts.html#MansellHoward)*. CUFP 2008. Retrieved 2009-08-29.*
  18. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-businessAppsFSharp_18-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-businessAppsFSharp_18-1) *Peake, Alex (2009).*[*The First Substantial Line of Business Application in F#*](https://web.archive.org/web/20091017070140/http:/cufp.galois.com/2009/abstracts.html#AlexPeakeAdamGranicz)*. CUFP 2009. Archived from*[*the original*](http://cufp.galois.com/2009/abstracts.html#AlexPeakeAdamGranicz)*on 2009-10-17. Retrieved 2009-08-29.*
  19. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-useR_19-0) [*"The useR! 2006 conference schedule includes papers on the commercial use of R"*](http://www.r-project.org/useR-2006/program.html)*. R-project.org. 2006-06-08. Retrieved 2011-06-20.*
  20. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-Chambers_20-0) [*Chambers, John M.*](https://en.wikipedia.org/wiki/John_Chambers_(programmer))*(1998). Programming with Data: A Guide to the S Language. Springer Verlag. pp. 67–70.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-387-98503-9*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-387-98503-9)*.*
  21. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-Novatchev_21-0) *Novatchev, Dimitre.*[*"The Functional Programming Language XSLT — A proof through examples"*](http://fxsl.sourceforge.net/articles/FuncProg/Functional%20Programming.html)*. Retrieved May 27, 2006.*
  22. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-Mertz_22-0) *Mertz, David.*[*"XML Programming Paradigms (part four): Functional Programming approached to XML processing"*](http://gnosis.cx/publish/programming/xml_models_fp.html)*. IBM developerWorks. Retrieved May 27, 2006.*
  23. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-Chamberlin_Boyce_23-0) [*Chamberlin, Donald D.*](https://en.wikipedia.org/wiki/Donald_D._Chamberlin)*;*[*Boyce, Raymond F.*](https://en.wikipedia.org/wiki/Raymond_F._Boyce)*(1974). "SEQUEL: A structured English query language". Proceedings of the 1974 ACM SIGFIDET: 249–264.*
  24. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-:0_24-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-:0_24-1) [*"Functional programming - Kotlin Programming Language"*](https://kotlinlang.org/docs/tutorials/kotlin-for-py/functional-programming.html)*. Kotlin. Retrieved 2019-05-01.*
  25. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-25) [*Dominus, Mark J.*](https://en.wikipedia.org/wiki/Mark_Jason_Dominus)*(2005).*[*Higher-Order Perl*](https://en.wikipedia.org/wiki/Higher-Order_Perl)*.*[*Morgan Kaufmann*](https://en.wikipedia.org/wiki/Morgan_Kaufmann)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-55860-701-9*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-55860-701-9)*.*
  26. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-26) *Holywell, Simon (2014). Functional Programming in PHP. php[architect].*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*9781940111056*](https://en.wikipedia.org/wiki/Special:BookSources/9781940111056)*.*
  27. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-AutoNT-13_27-0) *The Cain Gang Ltd.*[*"Python Metaclasses: Who? Why? When?"*](https://web.archive.org/web/20090530030205/http:/www.python.org/community/pycon/dc2004/papers/24/metaclasses-pycon.pdf)*(PDF). Archived from*[*the original*](https://www.python.org/community/pycon/dc2004/papers/24/metaclasses-pycon.pdf)*(PDF) on 30 May 2009. Retrieved 27 June 2009.*
  28. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-28) *Vanderbauwhede, Wim.*[*"Cleaner code with functional programming"*](https://web.archive.org/web/20200728013926/https:/wimvanderbauwhede.github.io/articles/decluttering-with-functional-programming/)*. Archived from*[*the original*](https://wimvanderbauwhede.github.io/articles/decluttering-with-functional-programming/)*on 11 Sep 2020. Retrieved 6 October 2020.*
  29. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-effective-scala_29-0) [*"Effective Scala"*](https://twitter.github.com/effectivescala/?sd)*. Scala Wiki. Retrieved 2012-02-21. Effective Scala.*
  30. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-30) *Turing, A. M. (1937). "Computability and λ-definability". The Journal of Symbolic Logic. Cambridge University Press.****2****(4): 153–163.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.2307/2268280*](https://doi.org/10.2307%2F2268280)*.*[*JSTOR*](https://en.wikipedia.org/wiki/JSTOR_(identifier))[*2268280*](https://www.jstor.org/stable/2268280)*.*
  31. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-31) *Haskell Brooks Curry; Robert Feys (1958).*[*Combinatory Logic*](https://archive.org/details/combinatorylogic0002curr)*. North-Holland Publishing Company. Retrieved 10 February 2013.*
  32. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-32) [*Church, A.*](https://en.wikipedia.org/wiki/Alonzo_Church)*(1940). "A Formulation of the Simple Theory of Types". Journal of Symbolic Logic.****5****(2): 56–68.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.2307/2266170*](https://doi.org/10.2307%2F2266170)*.*[*JSTOR*](https://en.wikipedia.org/wiki/JSTOR_(identifier))[*2266170*](https://www.jstor.org/stable/2266170)*.*
  33. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-33) [*McCarthy, John*](https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist))*(June 1978).*[*History of Lisp*](http://jmc.stanford.edu/articles/lisp/lisp.pdf)*(PDF). History of Programming Languages. Los Angeles, CA. pp. 173–185.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/800025.808387*](https://doi.org/10.1145%2F800025.808387)*.*
  34. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-34) [*John McCarthy*](https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist))*(1960).*[*"Recursive functions of symbolic expressions and their computation by machine, Part I."*](http://jmc.stanford.edu/articles/recursive/recursive.pdf)*(PDF). Communications of the ACM. ACM New York, NY, USA.****3****(4): 184–195.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/367177.367199*](https://doi.org/10.1145%2F367177.367199)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*1489409*](https://api.semanticscholar.org/CorpusID:1489409)*.*
  35. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-35) *Guy L. Steele; Richard P. Gabriel (February 1996).*[*The Evolution of Lisp*](http://dreamsongs.com/Files/HOPL2-Uncut.pdf)*(PDF). In ACM/SIGPLAN Second History of Programming Languages. pp. 233–330.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/234286.1057818*](https://doi.org/10.1145%2F234286.1057818)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-201-89502-5*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-201-89502-5)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*47047140*](https://api.semanticscholar.org/CorpusID:47047140)*.*
  36. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-36) The memoir of [Herbert A. Simon](https://en.wikipedia.org/wiki/Herbert_A._Simon) (1991), *Models of My Life*pp.189-190 [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0-465-04640-1](https://en.wikipedia.org/wiki/Special:BookSources/0-465-04640-1) claims that he, Al Newell, and Cliff Shaw are "...commonly adjudged to be the parents of [the] artificial intelligence [field]," for writing [Logic Theorist](https://en.wikipedia.org/wiki/Logic_Theorist), a program that proved theorems from [*Principia Mathematica*](https://en.wikipedia.org/wiki/Principia_Mathematica) automatically. To accomplish this, they had to invent a language and a paradigm that, viewed retrospectively, embeds functional programming.
  37. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-Backus_1977_37-0) *Backus, J. (1978).*[*"Can programming be liberated from the von Neumann style?: A functional style and its algebra of programs"*](https://doi.org/10.1145%2F359576.359579)*. Communications of the ACM.****21****(8): 613–641.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/359576.359579*](https://doi.org/10.1145%2F359576.359579)*.*
  38. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-38) R.M. Burstall. Design considerations for a functional programming language. Invited paper, Proc. Infotech State of the Art Conf. "The Software Revolution", Copenhagen, 45–57 (1977)
  39. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-39) R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. Journal of the Association for Computing Machinery 24(1):44–67 (1977)
  40. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-40) R.M. Burstall, D.B. MacQueen and D.T. Sannella. HOPE: an experimental applicative language. Proc. 1980 LISP Conference, Stanford, 136–143 (1980).
  41. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-41) [*"Make discovering assign() easier!"*](http://forum.openscad.org/Make-discovering-assign-easier-td10964.html)*. OpenSCAD.*
  42. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-42) *Peter Bright (March 13, 2018).*[*"Developers love trendy new languages but earn more with functional programming"*](https://arstechnica.com/gadgets/2018/03/developers-love-trendy-new-languages-but-earn-more-with-functional-programming/)*.*[*Ars Technica*](https://en.wikipedia.org/wiki/Ars_Technica)*.*
  43. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-43) *John Leonard (January 24, 2017).*[*"The stealthy rise of functional programming"*](https://www.computing.co.uk/ctg/analysis/3003123/the-slow-but-steady-rise-of-functional-programming)*. Computing.*
  44. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-44) *Leo Cheung (May 9, 2017).*[*"Is functional programming better for your startup?"*](https://www.infoworld.com/article/3190185/software/is-functional-programming-better-for-your-startup.html)*.*[*InfoWorld*](https://en.wikipedia.org/wiki/InfoWorld)*.*
  45. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-45) *Pountain, Dick.*[*"Functional Programming Comes of Age"*](https://web.archive.org/web/20060827094123/http:/byte.com/art/9408/sec11/art1.htm)*. BYTE.com (August 1994). Archived from*[*the original*](http://byte.com/art/9408/sec11/art1.htm)*on 2006-08-27. Retrieved August 31, 2006.*
  46. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-fortran95_46-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-fortran95_46-1) [*"ISO/IEC JTC 1/SC 22/WG5/N2137"*](https://isotc.iso.org/livelink/livelink/19089292/N2137.pdf.flv?func=sspndocuments.Fetch&nodeid=19089292&viewType=1)*. International Organization for Standardization. July 6, 2017.*
  47. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-SchemeProperTailRec_47-0) [*"Revised^6 Report on the Algorithmic Language Scheme"*](http://www.r6rs.org/final/html/r6rs/r6rs-Z-H-8.html#node_sec_5.11)*. R6rs.org. Retrieved 2013-03-21.*
  48. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-48) [*"Revised^6 Report on the Algorithmic Language Scheme - Rationale"*](http://www.r6rs.org/final/html/r6rs-rationale/r6rs-rationale-Z-H-7.html#node_sec_5.3)*. R6rs.org. Retrieved 2013-03-21.*
  49. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-49) *Clinger, William (1998). "Proper tail recursion and space efficiency". Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation - PLDI '98. pp. 174–185.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/277650.277719*](https://doi.org/10.1145%2F277650.277719)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*0897919874*](https://en.wikipedia.org/wiki/Special:BookSources/0897919874)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*16812984*](https://api.semanticscholar.org/CorpusID:16812984)*.*
  50. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-50) *Baker, Henry (1994).*[*"CONS Should Not CONS Its Arguments, Part II: Cheney on the M.T.A."*](http://home.pipeline.com/~hbaker1/CheneyMTA.html)
  51. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-totalfp_51-0) [*Turner, D.A.*](https://en.wikipedia.org/wiki/David_Turner_(computer_scientist))*(2004-07-28).*[*"Total Functional Programming"*](http://www.jucs.org/jucs_10_7/total_functional_programming)*. Journal of Universal Computer Science.****10****(7): 751–768.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.3217/jucs-010-07-0751*](https://doi.org/10.3217%2Fjucs-010-07-0751)*.*
  52. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-52) [The Implementation of Functional Programming Languages](http://research.microsoft.com/~simonpj/papers/slpj-book-1987/index.htm). Simon Peyton Jones, published by Prentice Hall, 1987
  53. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-launchbury1993_53-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-launchbury1993_53-1) [*John Launchbury*](https://en.wikipedia.org/wiki/John_Launchbury)*(1993). "A Natural Semantics for Lazy Evaluation".*[*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.35.2016*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.2016)*.*
  54. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-54) [*Robert W. Harper*](https://en.wikipedia.org/wiki/Robert_Harper_(computer_scientist))*(2009).*[*Practical Foundations for Programming Languages*](https://web.archive.org/web/20160407095249/https:/www.cs.cmu.edu/~rwh/plbook/book.pdf)*(PDF). Archived from*[*the original*](https://www.cs.cmu.edu/~rwh/plbook/book.pdf)*(PDF) on 2016-04-07.*
  55. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-55) *Huet, Gérard P. (1973). "The Undecidability of Unification in Third Order Logic". Information and Control.****22****(3): 257–267.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1016/s0019-9958(73)90301-x*](https://doi.org/10.1016%2Fs0019-9958%2873%2990301-x)*.*
  56. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-56) *Huet, Gérard (Sep 1976). Resolution d'Equations dans des Langages d'Ordre 1,2,...ω (Ph.D.) (in French). Universite de Paris VII.*
  57. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-57) *Huet, Gérard (2002).*[*"Higher Order Unification 30 years later"*](http://pauillac.inria.fr/~huet/PUBLIC/Hampton.pdf)*(PDF). In Carreño, V.; Muñoz, C.; Tahar, S. (eds.). Proceedings, 15th International Conference TPHOL. LNCS.****2410****. Springer. pp. 3–12.*
  58. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-58) *Wells, J. B. (1993). "Typability and type checking in the second-order lambda-calculus are equivalent and undecidable". Tech. Rep. 93-011.*[*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.31.3590*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.3590)*.*
  59. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-59) *Leroy, Xavier (17 September 2018).*[*"The Compcert verified compiler"*](http://compcert.inria.fr/doc/index.html)*.*
  60. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-60) *Peyton Jones, Simon; Vytiniotis, Dimitrios;*[*Weirich, Stephanie*](https://en.wikipedia.org/wiki/Stephanie_Weirich)*; Geoffrey Washburn (April 2006).*[*"Simple unification-based type inference for GADTs"*](http://research.microsoft.com/en-us/um/people/simonpj/papers/gadt/)*. Icfp 2006: 50–61.*
  61. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-61) *Kennedy, Andrew; Russo, Claudio (October 2005).*[*Generalized Algebraic Data Types and Object-Oriented Programming*](https://web.archive.org/web/20061229164852/http:/research.microsoft.com/~akenn/generics/gadtoop.pdf)*(PDF). OOPSLA. San Diego, California.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*9781595930316*](https://en.wikipedia.org/wiki/Special:BookSources/9781595930316)*. Archived from*[*the original*](http://research.microsoft.com/~akenn/generics/gadtoop.pdf)*(PDF) on 2006-12-29.* [source of citation](http://lambda-the-ultimate.org/node/1134)
  62. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-62) *Hughes, John.*[*"Why Functional Programming Matters"*](http://www.cse.chalmers.se/~rjmh/Papers/whyfp.pdf)*(PDF).*[*Chalmers University of Technology*](https://en.wikipedia.org/wiki/Chalmers_University_of_Technology)*.*
  63. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-63) [*Purely functional data structures*](http://www.cambridge.org/us/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/purely-functional-data-structures) by [Chris Okasaki](https://en.wikipedia.org/wiki/Chris_Okasaki), [Cambridge University Press](https://en.wikipedia.org/wiki/Cambridge_University_Press), 1998, [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [0-521-66350-4](https://en.wikipedia.org/wiki/Special:BookSources/0-521-66350-4)
  64. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-64) *L’orange, Jean Niklas.*[*"polymatheia - Understanding Clojure's Persistent Vector, pt. 1"*](http://www.hypirion.com/musings/understanding-persistent-vector-pt-1)*. Polymatheia. Retrieved 2018-11-13.*
  65. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-65) *Newbern, J.*[*"All About Monads: A comprehensive guide to the theory and practice of monadic programming in Haskell"*](http://monads.haskell.cz/html/index.html/html/)*. Retrieved 2008-02-14.*
  66. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-66) [*"Thirteen ways of looking at a turtle"*](https://fsharpforfunandprofit.com/posts/13-ways-of-looking-at-a-turtle/#2-basic-fp---a-module-of-functions-with-immutable-state)*. fF# for fun and profit. Retrieved 2018-11-13.*
  67. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-67) [*Paulson, Larry C.*](https://en.wikipedia.org/wiki/Lawrence_Paulson)*(28 June 1996).*[*ML for the Working Programmer*](https://books.google.com/books?id=XppZdaDs7e0C)*. Cambridge University Press.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-521-56543-1*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-521-56543-1)*. Retrieved 10 February 2013.*
  68. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-Spiewak_68-0) *Spiewak, Daniel (26 August 2008).*[*"Implementing Persistent Vectors in Scala"*](http://www.codecommit.com/blog/scala/implementing-persistent-vectors-in-scala)*. Code Commit.*
  69. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-69) [*"Which programs are fastest? | Computer Language Benchmarks Game"*](https://web.archive.org/web/20130520162513/http:/benchmarksgame.alioth.debian.org/u32/which-programs-are-fastest.php?gcc=on&ghc=on&clean=on&ocaml=on&sbcl=on&fsharp=on&racket=on&clojure=on&hipe=on&calc=chart)*. benchmarksgame.alioth.debian.org. Archived from*[*the original*](http://benchmarksgame.alioth.debian.org/u32/which-programs-are-fastest.php?gcc=on&ghc=on&clean=on&ocaml=on&sbcl=on&fsharp=on&racket=on&clojure=on&hipe=on&calc=chart)*on 2013-05-20. Retrieved 2011-06-20.*
  70. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-70) *Igor Pechtchanski; Vivek Sarkar (2005). "Immutability specification and its applications". Concurrency and Computation: Practice and Experience.****17****(5–6): 639–662.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1002/cpe.853*](https://doi.org/10.1002%2Fcpe.853)*.*
  71. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-71) [*"Chapter 25. Profiling and optimization"*](http://book.realworldhaskell.org/read/profiling-and-optimization.html#x_eK1)*. Book.realworldhaskell.org. Retrieved 2011-06-20.*
  72. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-72) *Hartel, Pieter; Henk Muller; Hugh Glaser (March 2004).*[*"The Functional C experience"*](http://www.ub.utwente.nl/webdocs/ctit/1/00000084.pdf)*(PDF). Journal of Functional Programming.****14****(2): 129–135.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1017/S0956796803004817*](https://doi.org/10.1017%2FS0956796803004817)*.*; *David Mertz.*[*"Functional programming in Python, Part 3"*](https://web.archive.org/web/20071016124848/http:/www-128.ibm.com/developerworks/linux/library/l-prog3.html)*. IBM developerWorks. Archived from*[*the original*](http://www-128.ibm.com/developerworks/linux/library/l-prog3.html)*on 2007-10-16. Retrieved 2006-09-17.*([Part 1](https://web.archive.org/web/20071016124848/http:/www-128.ibm.com/developerworks/linux/library/l-prog.html), [Part 2](https://web.archive.org/web/20071016124848/http:/www-128.ibm.com/developerworks/linux/library/l-prog2.html))
  73. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-73) [*"Functions — D Programming Language 2.0"*](http://www.digitalmars.com/d/2.0/function.html#pure-functions)*. Digital Mars. 30 December 2012.*
  74. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-74) [*"Lua Unofficial FAQ (uFAQ)"*](http://www.luafaq.org/#T1.2)*.*
  75. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-75) *Eich, Brendan (3 April 2008).*[*"Popularity"*](https://brendaneich.com/2008/04/popularity/)*.*
  76. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-76) [*van Rossum, Guido*](https://en.wikipedia.org/wiki/Guido_van_Rossum)*(2009-04-21).*[*"Origins of Python's "Functional" Features"*](http://python-history.blogspot.de/2009/04/origins-of-pythons-functional-features.html)*. Retrieved 2012-09-27.*
  77. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-77) [*"functools — Higher order functions and operations on callable objects"*](https://docs.python.org/dev/library/functools.html#functools.reduce)*. Python Software Foundation. 2011-07-31. Retrieved 2011-07-31.*
  78. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-78) *Skarsaune, Martin (2008). The SICS Java Port Project Automatic Translation of a Large Object Oriented System from Smalltalk to Java.*
  79. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-79) *Gosling, James.*[*"Closures"*](https://web.archive.org/web/20130414180002/https:/blogs.oracle.com/jag/entry/closures)*. James Gosling: on the Java Road. Oracle. Archived from*[*the original*](http://blogs.oracle.com/jag/entry/closures)*on 2013-04-14. Retrieved 11 May 2013.*
  80. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-80) *Williams, Michael (8 April 2013).*[*"Java SE 8 Lambda Quick Start"*](https://blogs.oracle.com/javatraining/entry/java_se_8_lambda_quick)*.*
  81. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-81) *Bloch, Joshua (2008). "Item 15: Minimize Mutability".*[*Effective Java*](https://archive.org/details/effectivejava00bloc_0)*(Second ed.). Addison-Wesley.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0321356680*](https://en.wikipedia.org/wiki/Special:BookSources/978-0321356680)*.*
  82. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-82) *Piro, Christopher (2009).*[*Functional Programming at Facebook*](https://web.archive.org/web/20091017070140/http:/cufp.galois.com/2009/abstracts.html#ChristopherPiroEugeneLetuchy)*. CUFP 2009. Archived from*[*the original*](http://cufp.galois.com/2009/abstracts.html#ChristopherPiroEugeneLetuchy)*on 2009-10-17. Retrieved 2009-08-29.*
  83. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-Sim-Diasca_83-0) [*"Sim-Diasca: a large-scale discrete event concurrent simulation engine in Erlang"*](http://research.edf.com/research-and-the-scientific-community/software/sim-diasca-80704.html)*. November 2011.*
  84. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-whatsapp.blog.2012_84-0) [1 million is so 2011](https://blog.whatsapp.com/index.php/2012/01/1-million-is-so-2011/) // WhatsApp blog, 2012-01-06: "the last important piece of our infrastracture is Erlang"
  85. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-85) *Momtahan, Lee (2009).*[*Scala at EDF Trading: Implementing a Domain-Specific Language for Derivative Pricing with Scala*](https://web.archive.org/web/20091017070140/http:/cufp.galois.com/2009/abstracts.html#LeeMomtahan)*. CUFP 2009. Archived from*[*the original*](http://cufp.galois.com/2009/abstracts.html#LeeMomtahan)*on 2009-10-17. Retrieved 2009-08-29.*
  86. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-86) *Graham, Paul (2003).*[*"Beating the Averages"*](http://www.paulgraham.com/avg.html)*. Retrieved 2009-08-29.*
  87. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-87) *Sims, Steve (2006).*[*Building a Startup with Standard ML*](http://cufp.galois.com/2006/slides/SteveSims.pdf)*(PDF). CUFP 2006. Retrieved 2009-08-29.*
  88. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-88) *Laurikari, Ville (2007).*[*Functional Programming in Communications Security*](http://cufp.galois.com/2007/abstracts.html#VilleLaurikari)*. CUFP 2007. Retrieved 2009-08-29.*
  89. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-89) *Lorimer, R. J. (19 January 2009).*[*"Live Production Clojure Application Announced"*](http://www.infoq.com/news/2009/01/clojure_production)*. InfoQ.*
  90. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-oxfordFP_90-0) [*"Functional Programming: 2019-2020"*](https://www.cs.ox.ac.uk/teaching/courses/2019-2020/fp/)*. University of Oxford Department of Computer Science. Retrieved 28 April 2020.*
  91. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-imperialFP_91-0) [*"Programming I (Haskell)"*](https://www.imperial.ac.uk/computing/current-students/courses/120_1/)*. Imperial College London Department of Computing. Retrieved 28 April 2020.*
  92. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-nottinghamFP_92-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-nottinghamFP_92-1) [*"Computer Science BSc - Modules"*](https://www.nottingham.ac.uk/ugstudy/course/Computer-Science-BSc#yearsmodules)*. Retrieved 28 April 2020.*
  93. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-mitFP_93-0) [***b***](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-mitFP_93-1) [*Abelson, Hal*](https://en.wikipedia.org/wiki/Hal_Abelson)*;*[*Sussman, Gerald Jay*](https://en.wikipedia.org/wiki/Gerald_Jay_Sussman)*(1985).*[*"Preface to the Second Edition"*](https://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book-Z-H-6.html)*.*[*Structure and Interpretation of Computer Programs*](http://mitpress.mit.edu/sicp/)*(2 ed.). MIT Press.*
  94. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-61A_94-0) *John DeNero (Fall 2019).*[*"Computer Science 61A, Berkeley"*](https://cs61a.org/articles/about.html)*. Department of Electrical Engineering and Computer Sciences, Berkeley. Retrieved 2020-08-14.*
  95. [**^**](https://en.wikipedia.org/wiki/Functional_programming#cite_ref-bootstrapworld_95-0) [Emmanuel Schanzer of Bootstrap](https://twit.tv/shows/triangulation/episodes/196/) interviewed on the TV show Triangulation on the [TWiT.tv](https://en.wikipedia.org/wiki/TWiT.tv) network

Further reading[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=21)]

* [*Abelson, Hal*](https://en.wikipedia.org/wiki/Hal_Abelson)*;*[*Sussman, Gerald Jay*](https://en.wikipedia.org/wiki/Gerald_Jay_Sussman)*(1985).*[*Structure and Interpretation of Computer Programs*](http://mitpress.mit.edu/sicp/)*. MIT Press.*
* Cousineau, Guy and Michel Mauny. *The Functional Approach to Programming*. Cambridge, UK: [Cambridge University Press](https://en.wikipedia.org/wiki/Cambridge_University_Press), 1998.
* Curry, Haskell Brooks and Feys, Robert and Craig, William. *Combinatory Logic*. Volume I. North-Holland Publishing Company, Amsterdam, 1958.
* [*Curry, Haskell B.*](https://en.wikipedia.org/wiki/Haskell_Curry)*;*[*Hindley, J. Roger*](https://en.wikipedia.org/wiki/J._Roger_Hindley)*;*[*Seldin, Jonathan P.*](https://en.wikipedia.org/w/index.php?title=Jonathan_P._Seldin&action=edit&redlink=1)*(1972). Combinatory Logic. Vol. II. Amsterdam: North Holland.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-7204-2208-5*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-7204-2208-5)*.*
* Dominus, Mark Jason. [*Higher-Order Perl*](http://hop.perl.plover.com/book/pdf/HigherOrderPerl.pdf). [Morgan Kaufmann](https://en.wikipedia.org/wiki/Morgan_Kaufmann). 2005.
* *Felleisen, Matthias; Findler, Robert; Flatt, Matthew; Krishnamurthi, Shriram (2001).*[*How to Design Programs*](http://www.htdp.org/)*. MIT Press.*
* Graham, Paul. *ANSI Common LISP*. Englewood Cliffs, New Jersey: [Prentice Hall](https://en.wikipedia.org/wiki/Prentice_Hall), 1996.
* MacLennan, Bruce J. *Functional Programming: Practice and Theory*. Addison-Wesley, 1990.
* *O'Sullivan, Brian; Stewart, Don; Goerzen, John (2008).*[*Real World Haskell*](http://book.realworldhaskell.org/read/)*. O'Reilly.*
* Pratt, Terrence, W. and Marvin V. Zelkowitz. *Programming Languages: Design and Implementation*. 3rd ed. Englewood Cliffs, New Jersey: [Prentice Hall](https://en.wikipedia.org/wiki/Prentice_Hall), 1996.
* Salus, Peter H. *Functional and Logic Programming Languages*. Vol. 4 of Handbook of Programming Languages. Indianapolis, Indiana: [Macmillan Technical Publishing](https://en.wikipedia.org/w/index.php?title=Macmillan_Technical_Publishing&action=edit&redlink=1), 1998.
* Thompson, Simon. *Haskell: The Craft of Functional Programming*. Harlow, England: [Addison-Wesley Longman Limited](https://en.wikipedia.org/w/index.php?title=Addison-Wesley_Longman_Limited&action=edit&redlink=1), 1996.

External links[[edit](https://en.wikipedia.org/w/index.php?title=Functional_programming&action=edit&section=22)]

**Listen to this article**



MENU

0:00

Spoken Wikipedia icon

[This audio file](https://en.wikipedia.org/wiki/File:En-Functional_programming.ogg) was created from a revision of this article dated 2011-08-25, and does not reflect subsequent edits.

(

* [Audio help](https://en.wikipedia.org/wiki/Wikipedia:Media_help)
* [More spoken articles](https://en.wikipedia.org/wiki/Wikipedia:Spoken_articles)

)

* *Ford, Neal (2012-01-29).*[*"Functional thinking: Why functional programming is on the rise"*](http://www.ibm.com/developerworks/java/library/j-ft20/index.html)*. Retrieved 2013-02-24.*
* *Akhmechet, Slava (2006-06-19).*[*"defmacro – Functional Programming For The Rest of Us"*](http://www.defmacro.org/ramblings/fp.html)*. Retrieved 2013-02-24.* An introduction
* *Functional programming in Python* (by David Mertz): [part 1](http://gnosis.cx/publish/programming/charming_python_13.html), [part 2](http://gnosis.cx/publish/programming/charming_python_16.html), [part 3](http://gnosis.cx/publish/programming/charming_python_19.html)