List of programming languages by type

This is a list of notable programming languages, grouped by type.

The groupings are overlapping; not mutually exclusive. A language can be listed in multiple groupings.

Agent-oriented programming languages

Agent-oriented programming allows the developer to build, extend and use software agents, which are abstractions of objects that can message other agents.

- Clojure
- F#
- GOAL
- SARL

Array languages

Array programming (also termed *vector* or *multidimensional*) languages generalize operations on scalars to apply transparently to vectors, matrices, and higher-dimensional arrays.

- A+
- Ada
- Analytica
- APL
- Chapel
- Dartmouth BASIC
- Fortran (As of Fortran 90)
- FreeMat
- GAUSS
- Interactive Data Language (IDL)
- J
- Julia
- K
- MATLAB
- Octave
- Q
- R
- Raku[1]
- S
- Scilab
- S-Lang
- SequenceL
- Speakeasy
- Wolfram Mathematica (Wolfram language)
- X10
- ZPL

Aspect-oriented programming languages

Aspect-oriented programming enables developers to add new functionality to code, known as "advice", without modifying that code itself; rather, it uses a pointcut to implement the advice into code blocks.

- Ada
- AspectJ
- Groovy
- Nemerle

- Raku[2][3]

Assembly languages

Assembly languages directly correspond to a machine language (see below), so machine code instructions appear in a form understandable by humans, although there may not be a one-to-one mapping between an individual statement and an individual instruction. Assembly languages let programmers use symbolic addresses, which the assembler converts to absolute or relocatable addresses. Most assemblers also support macros and symbolic constants.

Authoring languages

An authoring language is a programming language designed for use by a non-computer expert to easily create tutorials, websites, and other interactive computer programs.

- Darwin Information Typing Architecture (DITA)
- Lasso
- PILOT
- TUTOR
- Authorware

Command-line interface languages

Command-line interface (CLI) languages are also called batch languages or job control languages. Examples:

- 4DOS (shell for IBM PCs)
- 4OS2 (shell for IBM PCs)
- bash (the Bourne-Again shell from GNU, Free Software Foundation)
- CLIST (MVS Command List)
- CMS EXEC
- csh and tcsh (by Bill Joy UC Berkeley)
- DIGITAL Command Language CLI for VMS (DEC, Compaq, HP)
- DOS batch language (for IBM PC DOS, pre-Windows)
- EXEC 2
- Expect (a Unix automation and test tool)
- fish (a Unix shell)
- Hamilton C shell (a C shell for Windows)
- ksh (a standard Unix shell, written by David Korn)
- PowerShell (.NET-based CLI)
- rc (shell for Plan 9)
- Rexx
- sh (standard Unix shell, by Stephen R. Bourne)
- TACL (Tandem Advanced Command Language)
- Windows batch language (input for COMMAND.COM or CMD.EXE)
- zsh (a Unix shell)

Compiled languages

These are languages typically processed by compilers, though theoretically any language can be compiled or interpreted.

- ArkTS
- ActionScript
- Ada (multi-purpose language)
- ALGOL 58
  - JOVIAL
  - NELIAC
- ALGOL 60 (influential design)
  - SMALL a Machine ALGOL
- ALGOL 68
- Ballerina → bytecode runtime
- BASIC (including the first version of Dartmouth BASIC)
- BCPL
- C (widely used procedural language)
- C++ (multiparadigm language derived from C)
- C# (into CIL runtime)
- Ceylon (into JVM bytecode)
- CHILL
- CLIPPER 5.3 (DOS-based)
- CLEO for Leo computers
- Clojure (into JVM bytecode)
- COBOL
- Cobra
- Common Lisp
- Crystal
- Curl
- D (from a reengineering of C++)
- DASL → Java, JS, JSP, Flex.war
- Delphi (Borland's Object Pascal development system)
- DIBOL (a Digital COBOL)
- Dylan
- Eiffel (developed by Bertrand Meyer)
  - Sather
  - Ubercode
- Elm
- Emacs Lisp
- Emerald
- Erlang
- Factor
- Fortran (first compiled by IBM's John Backus)
- GAUSS
- Go (Golang)
- Gosu (into JVM bytecode)
- Groovy (into JVM bytecode)
- Haskell
- Harbour
- HolyC
- Inform (usually story files for Glulx or Z-code)
- Java (usually JVM bytecode; to machine code)
- JOVIAL
- Julia (on the fly to machine code)
- Kotlin (Kotlin/Native uses LLVM to produce binaries)
- LabVIEW
- Mercury
- Mesa
- Nemerle (into intermediate language bytecode)
- Nim
- Objective-C
- P
- Pascal (most implementations)
- PL/I (originally for IBM mainframes)
- Plus
- Python (to intermediate VM bytecode)
- RPG (Report Program Generator)
- Red
- Rust
- Scala (into JVM bytecode)
- Scheme (e.g. Gambit)
- SequenceL – purely functional, parallelizing and race-free
- Simula (object-oriented superset of ALGOL 60)
- Smalltalk platform independent VM bytecode
- Swift
- ML
  - Standard ML (SML)
    - Alice
  - OCaml
  - F# (into CIL, generates runtime)
- Turing
- V (Vlang)
- Vala (GObject type system)
- Visual Basic (CIL JIT runtime)
- Visual FoxPro

- Visual Prolog
- Xojo
- Zig

Concatenative programming languages

A concatenative programming language is a point-free computer programming language in which all expressions denote functions, and the juxtaposition of expressions denotes function composition.[4] Concatenative programming replaces function application, which is common in other programming styles, with function composition as the default way to build subroutines.

- Factor
- Forth
- jq (function application is also supported)
- Joy
- PostScript
- Raku[5]

Concurrent languages

Message passing languages provide language constructs for concurrency. The predominant paradigm for concurrency in mainstream languages such as Java is shared memory concurrency. Concurrent languages that make use of message passing have generally been inspired by process calculi such as communicating sequential processes (CSP) or the π-calculus.

- Ada – multi-purpose language
- Alef – concurrent language with threads and message passing, used for systems programming in early versions of Plan 9 from Bell Labs
- Ateji PX – an extension of the Java language for parallelism
- Ballerina – a language designed for implementing and orchestrating micro-services. Provides a message based parallel-first concurrency model.
- ChucK – domain specific programming language for audio, precise control over concurrency and timing
- Cilk – a concurrent C
- Cω – C Omega, a research language extending C#, uses asynchronous communication
- Clojure – a dialect of Lisp for the Java virtual machine
- Chapel
- Co-array Fortran
- Concurrent Pascal (by Brinch-Hansen)
- Curry
- E – uses promises, ensures deadlocks cannot occur
- Eiffel (through the SCOOP mechanism, Simple Concurrent Object-Oriented Computation)
- Elixir (runs on the Erlang VM)
- Emerald – uses threads and monitors
- Erlang – uses asynchronous message passing with nothing shared
- Gambit Scheme – using the Termite library
- Gleam (runs on the Erlang VM)
- Go (Golang)
- Haskell – supports concurrent, distributed, and parallel programming across multiple machines
- Java
  - Join Java – concurrent language based on Java
  - X10
- Julia
- Joule – dataflow language, communicates by message passing
- LabVIEW

- Limbo – relative of Alef, used for systems programming in Inferno (operating system)
- MultiLisp – Scheme variant extended to support parallelism
- OCaml
- occam – influenced heavily by Communicating Sequential Processes (CSP)
  - occam-π – a modern variant of occam, which incorporates ideas from Milner's π-calculus
- Orc
- Oz – multiparadigm language, supports shared-state and message-passing concurrency, and futures, and Mozart Programming System cross-platform Oz
- P
- Pict – essentially an executable implementation of Milner's π-calculus
- Python – uses thread-based parallelism and process-based parallelism[6]
- Raku[7]
- Rust
- Scala – implements Erlang-style actors on the JVM
- SequenceL – purely functional, automatically parallelizing and race-free
- SR – research language
- V (Vlang)
- Unified Parallel C
- XProc – XML processing language, enabling concurrency

Constraint programming languages

A constraint programming language is a declarative programming language where relationships between variables are expressed as constraints. Execution proceeds by attempting to find values for the variables which satisfy all declared constraints.

- Claire
- Constraint Handling Rules
- CHIP
- ECLiPSe
- Kaleidoscope
- Raku[8]

Curly bracket languages

A **curly bracket** or **curly brace** language has syntax that defines a block as the statements between curly brackets, a.k.a. braces, `{}`. This syntax originated with BCPL (1966), and was popularized by C. Many curly bracket languages descend from or are strongly influenced by C. Examples:

- ABCL/c+
- Alef
- AWK
- ArkTS
- B
- bc
- BCPL
- Ballerina
- C – developed circa 1970 at Bell Labs
- C++
- C#
- Ceylon
- ChucK – audio programming language
- Cilk – concurrent C for multithreaded parallel programming
- Cyclone – a safer C variant
- D
- Dart
- DASL – based on Java
- E

- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript
  - JScript
  - TypeScript
- GLSL
- Go (Golang)
- HLSL
- Java
  - Processing
  - Groovy
  - Join Java
  - Kotlin
  - Tea
  - X10
- Limbo
- LPC
- MEL
- Nemerle (curly braces optional)[9]
- Objective-C
- PCASTL
- Perl
- PHP
- Pico
- Pike
- PowerShell
- R
- Raku
- Rust
- S-Lang
- Scala (curly-braces optional)
- sed
- Solidity[10]
- SuperCollider
- Swift
- UnrealScript
- V (Vlang)
- Yorick
- YASS

Dataflow languages

Dataflow programming languages rely on a (usually visual) representation of the flow of data to specify the program. Frequently used for reacting to discrete events or for processing streams of data. Examples of dataflow languages include:

- Analytica
- Ballerina
- BMDFM
- Hartmann pipelines
- G (used in LabVIEW)
- Lucid
- Max
- Oz
- Prograph
- Pure Data
- Reaktor
- StreamBase StreamSQL EventFlow
- Swift (parallel scripting language)
- VEE
- VHDL
- VisSim
- Vvvv
- WebMethods Flow

Data-oriented languages

Data-oriented languages provide powerful ways of searching and manipulating the relations that have been described as entity relationship tables which map one set of things into other sets. Examples of data-oriented languages include:

- Clarion
- Clipper
- dBase a relational database access language
- Gremlin
- MUMPS (an ANSI standard general-purpose language with specializations for database work)
- Caché ObjectScript (a proprietary superset of MUMPS)
- RETRIEVE
- RDQL
- SPARQL
- SQL
- Visual FoxPro – a native RDBMS engine, object-oriented, RAD
- WebDNA
- Wolfram Mathematica (Wolfram language)

Decision table languages

Decision tables can be used as an aid to clarifying the logic before writing a program in any language, but in the 1960s a number of languages were developed where the main logic is expressed directly in the form of a decision table, including:

- Filetab

Declarative languages

Declarative languages express the logic of a computation without describing its control flow in detail. Declarative programming stands in contrast to imperative programming via imperative programming languages, where control flow is specified by serial orders (imperatives). (Pure) functional and logic-based programming languages are also declarative, and constitute the major subcategories of the declarative category. This section lists additional examples not in those subcategories.

- Analytica
- Ant (combine declarative programming and imperative programming)
- Curry
- Cypher
- Datalog
- Distributed Application Specification Language (DASL) (combine declarative programming and imperative programming)
- ECL
- Gremlin
- Inform (combine declarative programming and imperative programming)
- Lustre
- Mercury
- Metafont
- MetaPost
- Modelica
- Nix
- Prolog
- QML
- Oz
- RDQL
- SequenceL – purely functional, automatically parallelizing and race-free
- SPARQL
- SQL (Only DQL, not DDL, DCL, and DML)
- Soufflé
- Wolfram Mathematica (Wolfram language)
- xBase
- XSL Transformations

Embeddable languages

In source code

Source embeddable languages embed small pieces of executable code inside a piece of free-form text, often a web page.

Client-side embedded languages are limited by the abilities of the browser or intended client. They aim to provide dynamism to web pages without the need to recontact the server.

Server-side embedded languages are much more flexible, since almost any language can be built into a server. The aim of having fragments of server-side code embedded in a web page is to generate additional markup dynamically; the code itself disappears when the page is served, to be replaced by its output.

Server side

- PHP
- VBScript
- Tcl – server-side in NaviServer and an essential component in electronics industry systems
- WebDNA – dedicated to database-driven websites

The above examples are particularly dedicated to this purpose. A large number of other languages, such as Erlang, Scala, Perl, Ring and Ruby can be adapted (for instance, by being made into Apache modules).

Client side

- ActionScript
- JavaScript (aka ECMAScript or JScript)
- VBScript (Windows only)

In object code

A wide variety of dynamic or scripting languages can be embedded in compiled executable code. Basically, object code for the language's interpreter needs to be linked into the executable. Source code fragments for the embedded language can then be passed to an evaluation function as strings. Application control languages can be implemented this way, if the source code is input by the user. Languages with small interpreters are preferred.

- AngelScript
- Ch
- EEL
- Io
- jq (C and Go)
- Julia
- Lua
- Python
- Ring
- Ruby (via mruby)
- Squirrel
- Tcl

Educational programming languages

Languages developed primarily for the purpose of teaching and learning of programming.

- Alice
- Blockly
- Catrobat
- COMAL
- Elan
- Emerald
- Ezhil
- Logo
- Modula-2
- Pascal
- Racket
- Scheme
- Scratch
- Snap!
- Turing
- Wolfram Mathematica (Wolfram language)

Esoteric languages

An esoteric programming language is a programming language designed as a test of the boundaries of computer programming language design, as a proof of concept, or as a joke.

- Beatnik
- Befunge
- Brainfuck
- Chef
- INTERCAL
- LOLCODE
- Malbolge
- Piet
- Shakespeare
- Whitespace

Extension languages

Extension programming languages are languages embedded into another program and used to harness its features in extension scripts.

- AutoLISP (specific to AutoCAD)
- BeanShell
- CAL
- C/AL (C/SIDE)
- Guile
- Emacs Lisp
- JavaScript and some dialects, e.g., JScript
- Lua (embedded in many games)

- OpenCL (extension of C and C++ to use the GPU and parallel extensions of the CPU)
- OptimJ (extension of Java with language support for writing optimization models and powerful abstractions for bulk data processing)
- Perl
- Pike
- PowerShell
- Python (embedded in Maya, Blender, and other 3-D animation packages)
- Rexx
- Ring
- Ruby (Google SketchUp)
- S-Lang
- SQL
- Squirrel
- Tcl
- Vim script (vim)
- Visual Basic for Applications (VBA)

Fourth-generation languages

Fourth-generation programming languages are high-level languages built around database systems. They are generally used in commercial environments.

- 1C:Enterprise programming language
- ABAP
- CorVision
- CSC's GraphTalk
- CA-IDEAL (Interactive Development Environment for an Application Life) for use with CA-DATACOM/DB
- Easytrieve report generator (now CA-Easytrieve Plus)
- FOCUS
- IBM Informix-4GL
- LINC 4GL
- LiveCode (Not based on a database; still, the goal is to work at a higher level of abstraction than 3GLs.)
- MAPPER (Unisys/Sperry) – now part of BIS
- MARK-IV (Sterling/Informatics) now VISION:BUILDER of CA
- NATURAL
- Progress 4GL
- PV-Wave
- RETRIEVE
- SAS
- SQL
- Ubercode (VHLL, or Very-High-Level Language)
- Uniface
- Visual DataFlex
- Visual FoxPro

- xBase

Functional languages

Functional programming languages define programs and subroutines as mathematical functions and treat them as first-class. Many so-called functional languages are "impure", containing imperative features. Many functional languages are tied to mathematical calculation tools. Functional languages include:

Pure

- Agda
- Clean
- Coq (Gallina)
- Cuneiform
- Curry
- Elm
- Futhark
- Haskell
- Hope
- Idris
- Joy
- jq (but functions are 2nd class)
- Lean
- Mercury
- Miranda
- PureScript
- Ur
- KRC
- SAC
- SASL
- SequenceL

Impure

- APL
- ATS
- CAL
- C++ (since C++11)
- C#
- VB.NET
- Ceylon
- Curl
- D
- Dart
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript
  - JScript
  - Source
  - ArkTS
- Erlang
  - Elixir
  - Gleam
  - LFE
- Fexl
- Flix
- G (used in LabVIEW)
- Groovy
- Hop
- J
- Java (since version 8)
- Julia
- Kotlin
- Lisp
  - Clojure
  - Common Lisp
  - Dylan
  - Emacs Lisp
  - LFE
  - Little b
  - Logo
  - Racket
  - Scheme
    - Guile
  - Tea
- ML
  - Standard ML (SML)
    - Alice
  - OCaml
  - F#
- Nemerle
- Nim
- Opal

- OPS5
- Perl
- PHP
- PL/pgSQL
- Python
- Q (equational programming language)
- Q (programming language from Kx Systems)
- R
- Raku
- Rebol
- Red
- Ring
- Ruby
- REFAL
- Rust
- Scala
- Swift
- Spreadsheets
- V (Vlang)
- Tcl
- Wolfram Mathematica (Wolfram language)

Hardware description languages

In electronics, a hardware description language (HDL) is a specialized computer language used to describe the structure, design, and operation of electronic circuits, and most commonly, digital logic circuits. The two most widely used and well-supported HDL varieties used in industry are Verilog and VHDL. Hardware description languages include:

HDLs for analog circuit design

- Verilog-AMS (Verilog for Analog and Mixed-Signal)
- VHDL-AMS (VHDL with Analog/Mixed-Signal extension)

HDLs for digital circuit design

- Advanced Boolean Expression Language
- Altera Hardware Description Language
- Bluespec
- Confluence
- ELLA
- Handel-C
- Impulse C
- Lola
- MyHDL
- PALASM
- Ruby (hardware description language)
- SystemC
- SystemVerilog
- Verilog
- VHDL (VHSIC HDL)

Imperative languages

Imperative programming languages may be multi-paradigm and appear in other classifications. Here is a list of programming languages that follow the imperative paradigm:

- Ada
- ALGOL 58
  - JOVIAL
  - NELIAC
- ALGOL 60 (very influential language design)
- ALGOL 68
- BASIC
- C
- C++
- C#
- Ceylon
- CHILL
- COBOL
- D
- Dart
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript
  - JScript
  - Source
- FORTRAN
- GAUSS
- Go
- Groovy
- Icon
- Java
- Julia
- Lua
- MATLAB
- Machine languages
- Modula-2, Modula-3
- MUMPS
- Nim
- OCaml
- Oberon
- Object Pascal
- Open Object Rexx (ooRexx)
- Open Programming Language (OPL)
- OpenEdge Advanced Business Language (ABL)
- Pascal
- Perl
- PHP
- PL/I
- PL/S
- PowerShell
- PROSE
- Python
- Raku
- Rexx
- Ring
- Ruby
- Rust
- SETL
- Speakeasy
- Swift
- Tcl
- V (Vlang)
- Wolfram Mathematica (Wolfram language)

Interactive mode languages

Interactive mode languages act as a kind of shell: expressions or statements can be entered one at a time, and the result of their evaluation is seen immediately. The interactive mode is also termed a read–eval–print loop (REPL).

- APL
- BASIC (some dialects)
- Clojure
- Common Lisp
- Dart (with Observatory or Dartium's developer tools)
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript
  - JScript
  - Source
  - ArkTS
- Erlang
- Elixir (with iex)
- F#
- Fril
- GAUSS
- Groovy
- Guile
- Haskell (with the GHCi or Hugs interpreter)
- IDL
- J
- Java (since version 9)
- Julia
- Lua
- MUMPS (an ANSI standard general-purpose language)

- Maple
- MATLAB
- ML
- OCaml
- Perl
- PHP
- Pike
- PostScript
- PowerShell (.NET-based CLI)
- Prolog
- Python
- PROSE
- R
- Raku
- Rebol
- Red
- Rexx
- Ring
- Ruby (with IRB)
- Scala
- Scheme
- Smalltalk (anywhere in a Smalltalk environment)
- S-Lang (with the S-Lang shell, slsh)
- Speakeasy
- Swift
- Tcl (with the Tcl shell, tclsh)
- Unix shell
- Visual FoxPro
- Wolfram Mathematica (Wolfram language)

Interpreted languages

Interpreted languages are programming languages in which programs may be executed from source code form, by an interpreter. Theoretically, any language can be compiled or interpreted, so the term *interpreted language* generally refers to languages that are usually interpreted rather than compiled.

- Ant
- APL
- AutoHotkey scripting language
- AutoIt scripting language
- BASIC (some dialects)
- Programming Language for Business (PL/B, formerly DATABUS, later versions added optional compiling)
- Eiffel (via *Melting Ice Technology* in EiffelStudio)
- Emacs Lisp
- FOCAL
- GameMaker Language
- Groovy
- J
- jq
- Julia (compiled on the fly to machine code, by default, interpreting also available)
- JavaScript
- Lisp (early versions, pre-1962, and some experimental ones; production Lisp systems are compilers, but many of them still provide an interpreter if needed)
- LPC
- Lua
- MUMPS (an ANSI standard general-purpose language)
- Maple
- MATLAB
- OCaml
- Pascal (early implementations)
- PCASTL
- Perl
- PHP
- PostScript
- PowerShell
- PROSE
- Python
- Rexx
- R
- Raku
- Rebol
- Red
- Ring
- Ruby
- S-Lang
- Seed7
- Speakeasy
- Standard ML (SML)

- Spin
- Tcl
- Tea
- TorqueScript
- thinBasic scripting language
- VBScript
- Windows PowerShell – .NET-based CLI
- Some scripting languages – below
- Wolfram Mathematica (Wolfram language)

Iterative languages

Iterative languages are built around or offering generators.

- Aldor
- Alphard
- C++
- C#
- CLU
- Cobra
- ECMAScript (ES6+)
- Eiffel, through "agents"
- Icon
- IPL-v
- jq
- Julia
- Lua
- Nim
- PHP
- Python
- Raku[11]
- Sather

Languages by memory management type

Garbage collected languages

Garbage Collection (GC) is a form of automatic memory management. The garbage collector attempts to reclaim memory that was allocated by the program but is no longer used.

- APL
- C#
- Clean
- Crystal
- Dart
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript
  - JScript
  - Source
- Emerald
- Erlang
- Go
- Groovy
- Haskell
- Java
- Julia
- Kotlin
- LabVIEW
- Lisp (originator)
  - Arc
  - Clojure
  - Common Lisp
  - Dylan
  - Emacs Lisp
  - Guile
  - Racket
  - Scheme
  - Logo
- Lua
- ML
  - Standard ML (SML)
    - Alice
  - OCaml
- Modula-3

- Perl
- PHP
- PowerShell
- Python
- Ring
- Ruby
- Smalltalk
- Speakeasy

Languages with manual memory management

- C
- C++
- Component Pascal
- Forth
- Fortran
- FreeBASIC
- Modula-2
- Oberon
- Pascal
- PL/I
- Zig

Languages with optional manual memory management

- Ada implementations are not required to offer garbage collection, but the language semantics support it, and many implementations include it.
- Blitz BASIC (also known as BlitzMax) is usually reference-counted,[12] and also supports a garbage collector. However, it also ships with optional utilities for using pointers[13] and for directly allocating and freeing memory.[14]
- COBOL supports pointers[15] and heap allocation[16] as of COBOL 2002, along with a garbage collector.[17]
- Cython provides optional manual memory management by letting the user import `malloc`, `realloc`, and `free` from C, which they can then use in Python code.[18]
- D provides programmers with full control over its own garbage collector, including the ability to disable it outright.[19]
- Nim is usually garbage-collected or reference-counted by default, depending on its configuration, but the programmer may use the switch `--mm:none` to deallocate memory manually.[20]
- Objective-C and Objective-C++ support optional reference counting and garbage collection as alternatives to manual memory management (Apple deprecated the garbage collector).
- PostScript originally required developers to manually reclaim memory using the `save` and `restore` operators. PostScript Level 2 introduced a garbage collector, but its usage is optional.[21]
- Rust supports optional reference counting, but manual memory management is preferred.
- Scala normally manages the memory automatically in its JVM and JavaScript targets. However, the LLVM-based Scala Native compiler supports the use of pointers, as well as C-style heap allocation (e.g. `malloc`, `realloc`, `free`) and stack allocation (`stackalloc`).[22]

- Swift normally uses reference counting, but also allows the user to manually manage the memory using `malloc` and `free`. On Apple platforms, these functions are imported from the C standard library (which is imported from `Foundation`, `AppKit` or `UIKit`); on Linux, the developer needs to import `Glibc`, and `ucrt` on Windows.
- V (Vlang) uses GC by default, for user convenience, which can be turned off (-gc none). Users are free to manage memory manually. Can also use autofree (-autofree) or arena allocation (-prealloc).
- Vala uses reference counting by default, but the user is free to manage the memory manually if they wish.[23]

Languages with deterministic memory management

- Ada
- C
- C++
- Fortran
- Pascal
- Rust[24][25]
- Objective-C
- Zig

Languages with automatic reference counting (ARC)

- Objective-C
- Perl
- Swift
- Visual Basic
- Xojo

List-based languages – LISPs

List-based languages are a type of data-structured language that are based on the list data structure.

- Lisp
  - Arc
  - Clojure
  - Common Lisp
  - Dylan
  - Emacs Lisp
  - Guile
  - Racket
  - Scheme
  - Logo
- Joy
- R
- Source
- Tcl
  - Tea
- TRAC

Little languages

Little languages[26] serve a specialized problem domain.

- awk – used for text file manipulation.
- sed – parses and transforms text
- SQL – has only a few keywords and not all the constructs needed for a full programming language[a] – many database management systems extend SQL with additional constructs as a stored procedure language

Logic-based languages

Logic-based languages specify a set of attributes that a solution must-have, rather than a set of steps to obtain a solution.

Notable languages following this programming paradigm include:

- ALF
- Alma-0
- Curry
- Datalog
- Fril
- Flix (a functional programming language with first-class Datalog constraints)
- Janus
- λProlog (a logic programming language featuring polymorphic typing, modular programming, and higher-order programming)
- Oz, and Mozart Programming System cross-platform Oz
- Prolog (formulates data and the program evaluation mechanism as a special form of mathematical logic called Horn logic and a general proving mechanism called logical resolution)
  - Mercury (based on Prolog)
  - Visual Prolog (object-oriented Prolog extension)
- ROOP
- Soufflé

Machine languages

Machine languages are directly executable by a computer's CPU. They are typically formulated as bit patterns, usually represented in octal or hexadecimal. Each bit pattern causes the circuits in the CPU to execute one of the fundamental operations of the hardware. The activation of specific electrical inputs (e.g., CPU package pins for microprocessors), and logical settings for CPU state values, control the processor's computation. Individual machine languages are specific to a family of processors; machine-language code for one family of processors cannot run directly on processors in another family unless the processors in question have additional hardware to support it (for example, DEC VAX processors included a PDP-11

compatibility mode). They are (essentially) always defined by the CPU developer, not by 3rd parties.[b] The symbolic version, the processor's assembly language, is also defined by the developer, in most cases. Some commonly used machine code instruction sets are:

- RISC-V
- ARM
  - Original 32-bit
  - 16-bit Thumb instructions (subset of registers used)
  - 64-bit (major architecture change)
- DEC:
  - 18-bit: PDP-1, PDP-4, PDP-7, PDP-9, PDP-15
  - 12-bit: PDP-5, PDP-8, LINC-8, PDP-12
  - 36-bit: PDP-6, PDP-10, DECSYSTEM-20
  - 16-bit: PDP-11 (influenced VAX and M68000)
  - 32-bit: VAX
  - 64-bit: Alpha
- Intel 8008, 8080 and 8085
  - Zilog Z80
- x86:
  - 16-bit x86, first used in the Intel 8086
    - Intel 8086 and 8088 (the latter was used in the first and early IBM PC)
    - Intel 80186
    - Intel 80286 (the first x86 processor with protected mode, used in the IBM PC AT)
  - IA-32, introduced in the 80386
  - x86-64 – The original specification was created by AMD. There are vendor variants, but they're essentially the same:
    - AMD's AMD64
    - Intel's Intel 64
- IBM[c]
  - 305
  - 650
  - 701
  - 702, 705 and 7080
  - 704, 709, 7040, 7044, 7090, 7094
  - 1400 series, 7010
  - 7030
  - 7070
  - System/360 and successors, including z/Architecture
- MIPS
- Motorola 6800 (8-bit)
- Motorola 68000 series (CPUs used in early Macintosh and early Sun computers)
- MOS Technology 65xx (8-bit)
  - 6502 (CPU for VIC-20, BBC Micro, Apple II, and Atari 8-bit computers)
  - 6510 (CPU for Commodore 64)
  - Western Design Center 65816/65802 (CPU for Apple IIGS and (variant) Super Nintendo Entertainment System)

- National Semiconductor NS320xx
- POWER, first used in the IBM RS/6000
  - PowerPC – used in Power Macintosh and in many game consoles, particularly of the seventh generation.
  - Power ISA – an evolution of PowerPC.
- Sun Microsystems (now Oracle) SPARC
- UNIVAC[c]
  - 30-bit computers: 490, 492, 494, 1230
  - 36-bit computers
    - 1101, 1103, 1105
    - 1100/2200 series
- MCST Elbrus 2000

Macro languages

Textual substitution macro languages

Macro languages transform one source code file into another. A "macro" is essentially a short piece of text that expands into a longer one (not to be confused with hygienic macros), possibly with parameter substitution. They are often used to preprocess source code. Preprocessors can also supply facilities like file inclusion.

Macro languages may be restricted to acting on specially labeled code regions (pre-fixed with a # in the case of the C preprocessor). Alternatively, they may not, but in this case it is still often undesirable to (for instance) expand a macro embedded in a string literal, so they still need a rudimentary awareness of syntax. That being the case, they are often still applicable to more than one language. Contrast with source-embeddable languages like PHP, which are fully featured.

- cpp (the C preprocessor)
- m4 (originally from AT&T, bundled with Unix)
- ML/I (general-purpose macro processor)
- TTM (developed at the California Institute of Technology)

Application macro languages

Scripting languages such as Tcl and ECMAScript (ActionScript, ECMAScript for XML, JavaScript, JScript) have been embedded into applications. These are sometimes called "macro languages", although in a somewhat different sense to textual-substitution macros like m4.

Metaprogramming languages

Metaprogramming is the writing of programs that write or manipulate other programs, including themselves, as their data or that do part of the work that is otherwise done at run time during compile time. In many cases, this allows programmers to get more done in the same amount of time as they would take to write all the code manually.

- C++
- CWIC
- Curl
- D
- Emacs Lisp
- Elixir
- F#
- Groovy
- Haskell
- Julia
- Lisp
- Lua
- Maude system
- META II (and META I, a subset)
- MetaOCaml
- Nemerle
- Nim
- Perl
- Python
- Raku[27]
- Red
- Ring
- Ruby
- Rust[28]
- Scheme
- SequenceL
- Smalltalk
- Source
- TREE-META
- Wolfram Mathematica (Wolfram language)

Multiparadigm languages

Multiparadigm languages support more than one programming paradigm. They allow a program to use more than one programming style. The goal is to allow programmers to use the best tool for a job, admitting that no one paradigm solves all problems in the easiest or most efficient way.

- 1C:Enterprise programming language (generic, imperative, object-oriented, prototype-based, functional)
- Ada (concurrent, distributed, generic (template metaprogramming), imperative, object-oriented (class-based))
- ALF (functional, logic)
- Alma-0 (constraint, imperative, logic)
- APL (functional, imperative, object-oriented (class-based))
- BETA (functional, imperative, object-oriented (class-based))
- C++ (generic, imperative, object-oriented (class-based), functional, metaprogramming)
- C# (generic, imperative, object-oriented (class-based), functional, declarative)
- Ceylon (generic, imperative, object-oriented (class-based), functional, declarative)
- ChucK (imperative, object-oriented, time-based, concurrent, on-the-fly)
- Cobra (generic, imperative, object-oriented (class-based), functional, contractual)
- Common Lisp (functional, imperative, object-oriented (class-based), aspect-oriented (user may add further paradigms, e.g., logic))
- Curl (functional, imperative, object-oriented (class-based), metaprogramming)
- Curry (concurrent, functional, logic)
- D (generic, imperative, functional, object-oriented (class-based), metaprogramming)
- Dart (generic, imperative, functional, object-oriented (class-based))
- Delphi Object Pascal (generic, imperative, object-oriented (class-based), metaprogramming)
- Dylan (functional, object-oriented (class-based))
- ECMAScript (functional, imperative, object-oriented (prototype-based))
  - ActionScript
  - ECMAScript for XML
  - JavaScript
  - JScript
- Eiffel (imperative, object-oriented (class-based), generic, functional (agents), concurrent (SCOOP))

- F# (functional, generic, object-oriented (class-based), language-oriented)
- Fantom (functional, object-oriented (class-based))
- Go, Golang (imperative, procedural),
- Groovy (functional, object-oriented (class-based), imperative, procedural)
- Harbour
- Hop
- J (functional, imperative, object-oriented (class-based))
- Julia (imperative, multiple dispatch ("object-oriented"), functional, metaprogramming)
- LabVIEW (visual, dataflow, concurrent, modular, functional, object-oriented, scripting)
- Lua (functional, imperative, object-oriented (prototype-based))
- Mercury (functional, logical, object-oriented)
- Metaobject protocols (object-oriented (class-based, prototype-based))
- Nemerle (functional, object-oriented (class-based), imperative, metaprogramming)
- Objective-C (imperative, object-oriented (class-based), reflective)
- OCaml (functional, imperative, object-oriented (class-based), modular)
- Oz (functional (evaluation: eager, lazy), logic, constraint, imperative, object-oriented (class-based), concurrent, distributed), and Mozart Programming System cross-platform Oz
- Object Pascal (imperative, object-oriented (class-based))
- Perl (imperative, functional (can't be purely functional), object-oriented, class-oriented, aspect-oriented (through modules))
- PHP (imperative, object-oriented, functional (can't be purely functional))
- Pike (interpreted, general-purpose, high-level, cross-platform, dynamic programming language)
- Prograph (dataflow, object-oriented (class-based), visual)
- Python (functional, compiled, interpreted, object-oriented (class-based), imperative, metaprogramming, extension, impure, interactive mode, iterative, reflective, scripting)
- R (array, interpreted, impure, interactive mode, list-based, object-oriented prototype-based, scripting)
- Racket (functional, imperative, object-oriented (class-based) and can be extended by the user)
- Raku (concurrent, concatenative, functional, metaprogramming generic, imperative, reflection object-oriented, pipelines, reactive, and via libraries constraints, distributed)
- Rebol (functional, imperative, object-oriented (prototype-based), metaprogramming (dialected))
- Red (functional, imperative, object-oriented (prototype-based), metaprogramming (dialected))
- ROOP (imperative, logic, object-oriented (class-based), rule-based)
- Ring (imperative, functional, object-oriented (class-based), metaprogramming, declarative, natural)
- Ruby (imperative, functional, object-oriented (class-based), metaprogramming)
- Rust (concurrent, functional, imperative, object-oriented, generic, metaprogramming, compiled)
- Scala (functional, object-oriented)
- Seed7 (imperative, object-oriented, generic)
- SISAL (concurrent, dataflow, functional)
- Spreadsheets (functional, visual)
- Swift (protocol-oriented, object-oriented, functional, imperative, block-structured)
- Tcl (functional, imperative, object-oriented (class-based))
  - Tea (functional, imperative, object-oriented (class-based))
- V (Vlang) (functional, imperative, procedural, structured, concurrent)
- Windows PowerShell (functional, imperative, pipeline, object-oriented (class-based))
- Wolfram Mathematica (Wolfram language)

Numerical analysis

Several general-purpose programming languages, such as C and Python, are also used for technical computing, this list focuses on languages almost exclusively used for technical computing.

- AIMMS
- AMPL
- Analytica
- Fortran
- FreeMat
- GAUSS
- GAMS
- GNU Octave
- Julia
- Klerer-May System
- MATLAB
- PROSE
- R
- Seneca – an Oberon variant
- Scilab
- Speakeasy
- Wolfram Mathematica (Wolfram language)

Non-English-based languages

- Chinese BASIC (Chinese)
- Fjölnir (Icelandic)
- Language Symbolique d'Enseignement (French)
- Rapira (Russian)
- ezhil (Tamil)

Object-oriented class-based languages

Class-based object-oriented programming languages support objects defined by their class. Class definitions include member data. Message passing is a key concept, if not the main concept, in object-oriented languages.

Polymorphic functions parameterized by the class of some of their arguments are typically called methods. In languages with single dispatch, classes typically also include method definitions. In languages with multiple dispatch, methods are defined by generic functions. There are exceptions where single dispatch methods are generic functions (e.g. Bigloo's object system).

Multiple dispatch

- Common Lisp
- Cecil
- Dylan

- Julia[d]
- Raku[29]

Single dispatch

- ActionScript 3.0
- Actor
- Ada 95 and Ada 2005 (multi-purpose language)
- APL
- BETA
- C++
- C#
- Ceylon
- Dart
- Oxygene (formerly named Chrome)
- ChucK
- Cobra
- ColdFusion
- Curl
- D
- Distributed Application Specification Language (DASL)
- Delphi Object Pascal
- E
- GNU E
- Eiffel
  - Sather
  - Ubercode
- Fortran 2003
- Fortress
- Gambas
- Game Maker Language
- Harbour
- J
- Java
  - Processing
  - Groovy
  - Join Java
  - Tea
  - X10
- LabVIEW
- Lua
- Modula-2 (data abstraction, information hiding, strong typing, full modularity)
  - Modula-3 (added more object-oriented features to Modula-2)
- Nemerle
- NetRexx
- Oberon-2 (full object-orientation equivalence in an original, strongly typed, Wirthian manner)
- Object Pascal
- Object REXX
- Objective-C (a superset of C adding a Smalltalk derived object model and message passing syntax)
- OCaml
- OpenEdge Advanced Business Language (ABL)
- Oz, Mozart Programming System
- Perl 5
- PHP
- Pike
- Prograph
- Python (interpretive language, optionally object-oriented)
- Revolution (programmer does not get to pick the objects)
- Ring
- Ruby
- Scala
- Speakeasy
- Simula (first object-oriented language, developed by Ole-Johan Dahl and Kristen Nygaard)
- Smalltalk (pure object-orientation, developed at Xerox PARC)
  - Little Smalltalk
  - Pharo
  - Squeak
    - Scratch
  - IBM VisualAge
  - VisualWorks
- SPIN
- SuperCollider
- VBScript (Microsoft Office 'macro scripting' language)
- Visual DataFlex
- Visual FoxPro

- Visual Prolog
- X++
- Xojo
- XOTcl

Object-oriented prototype-based languages

Prototype-based languages are object-oriented languages where the distinction between classes and instances has been removed:

- 1C:Enterprise programming language
- Actor-Based Concurrent Language (ABCL, ABCL/1, ABCL/R, ABCL/R2, ABCL/c+)
- Agora
- Cecil
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript (first named Mocha, then LiveScript)
  - JScript
- Etoys in Squeak
- Io
- Lua
- MOO
- NewtonScript
- Obliq
- R
- Rebol
- Red
- Self (first prototype-based language, derived from Smalltalk)
- TADS

Off-side rule languages

Off-side rule languages denote blocks of code by their indentation.

- ISWIM, the abstract language that introduced the rule
- ABC, Python's parent
  - Python
    - Cobra
    - Boo
- Miranda, Haskell's parent
  - Orwell
  - Haskell
    - Curry
- Elixir (, do: blocks)
- F#
- Nemerle (off-side optional)[9]
- Nim
- Occam
- SPIN
- Scala (off-side optional)

Procedural languages

Procedural programming languages are based on the concept of the unit and scope (the data viewing range) of an executable code statement. A procedural program is composed of one or more units or modules, either user coded or provided in a code library; each module is composed of one or more procedures, also called a function, routine, subroutine, or method, depending on the language. Examples of procedural languages include:

- Ada (multi-purpose language)

- ALGOL 58
  - JOVIAL
  - NELIAC
- ALGOL 60 (very influential language design)
  - SMALL Machine ALGOL Like Language
- ALGOL 68
- Alma-0
- BASIC (these lack most modularity in (especially) versions before about 1990)
- BCPL
- BLISS
- C
- C++
- C# (similar to Java/C++)
- Ceylon
- CHILL
- ChucK (C/Java-like syntax, with new syntax elements for time and parallelism)
- COBOL
- Cobra
- ColdFusion
- CPL (Combined Programming Language)
- Curl
- D
- Distributed Application Specification Language (DASL) (combine declarative programming and imperative programming)
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript (first named Mocha, then LiveScript)
  - JScript
  - Source
- Eiffel
- Forth
- Fortran (better modularity in later Standards)
  - F
- GAUSS
- Go
- Harbour
- HyperTalk
- Java
  - Groovy
  - Join Java
  - Tea
- JOVIAL
- Julia
- Language H
- Lasso
- Modula-2 (fundamentally based on modules)
- MATLAB
- Mesa
- MUMPS (first release was more modular than other languages of the time; the standard has become even more modular since then)
- Nemerle
- Nim
- Oberon, Oberon-2 (improved, smaller, faster, safer follow-ons for Modula-2)
  - Component Pascal
  - Seneca
- OCaml
- Occam
- Oriel
- Pascal (successor to ALGOL 60, predecessor of Modula-2)
  - Free Pascal (FPC)
  - Object Pascal, Delphi
- PCASTL
- Perl
- Pike
- PL/C
- PL/I (large general-purpose language, originally for IBM mainframes)
- Plus
- PowerShell
- PROSE
- Python
- R
- Raku
- Rapira
- RPG
- Rust
- S-Lang
- VBScript

- Visual Basic
- Visual FoxPro
- Wolfram Mathematica (Wolfram language)
- Microsoft Dynamics AX (X++)

Query languages

Reflective languages

Reflective programming languages let programs examine and possibly modify their high-level structure at runtime or compile-time. This is most common in high-level virtual machine programming languages like Smalltalk, and less common in lower-level programming languages like C. Languages and platforms supporting reflection:

- Befunge
- Ceylon
- Charm
- ChucK
- CLI
  - C#
- Cobra
- Component Pascal BlackBox Component Builder
- Curl
- Cypher
- Delphi Object Pascal
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript
  - JScript
- Emacs Lisp
- Eiffel
- Harbour
- Julia
- JVM
  - Java
  - Groovy
  - Join Java
  - X10
- Lisp
  - Clojure
  - Common Lisp
  - Dylan
  - Logo
  - Scheme
- Lua
- Maude system
- Oberon-2 – ETH Oberon System
- Objective-C
- PCASTL
- Perl
- PHP
- Pico
- Poplog
  - POP-11
- PowerShell
- Prolog
- Python
- Raku[30]
- Rebol
- Red
- Ring
- Ruby
- Smalltalk (pure object-orientation, originally from Xerox PARC)
  - Little Smalltalk
  - Self
  - Squeak
  - IBM VisualAge
  - VisualWorks
- SNOBOL
- Tcl
- Wolfram Mathematica (Wolfram language)
- XOTcl
- X++
- Xojo

Rule-based languages

Rule-based languages instantiate rules when activated by conditions in a set of data. Of all possible activations, some set is selected and the statements belonging to those rules execute. Rule-based languages include:

- awk
- CLIPS
- Claire
- Constraint Handling Rules
- Drools
- GOAL agent programming language
- Jess
- OPS5
- Prolog
- ToonTalk – robots are rules
- Wolfram Mathematica (Wolfram language)
- XSLT

Scripting languages

- AngelScript
- AppleScript
- AutoHotKey
- AutoIt
- AWK
- bc
- BeanShell
- Bash
- Ch (Embeddable C/C++ interpreter)
- CLI
  - C# (compiled to bytecode, and running JIT inside VM)
- CLIST
- ColdFusion
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript (first named Mocha, then LiveScript)
  - JScript
  - Source
- Emacs Lisp
- CMS EXEC
- EXEC 2
- Game Maker Language (GML)
- GDScript
- Io
- JASS
- Julia (compiled on the fly to machine code, by default, interpreting also available)
- JVM
  - Groovy
  - Join Java
- Ksh
- Lasso
- Lua
- MAXScript
- MEL
- Object REXX (OREXX, OOREXX)
- Oriel
- Pascal Script
- Perl
- PHP (intended for Web servers)
- PowerShell
- Python
- R
- Raku
- Rebol
- Red
- Rexx
- Revolution
- Ring
- Ruby
- S-Lang
- sed
- Sh
- Smalltalk
- Squirrel
- Tea
- Tcl
- TorqueScript
- VBScript
- WebDNA, dedicated to database-driven websites
- Windows PowerShell (.NET-based CLI)
- Many shell command languages such as Unix shell or DIGITAL Command Language (DCL) on VMS have powerful scripting abilities.

Stack-based languages

Stack-based languages are a type of data-structured language that are based on the stack data structure.

- Beatnik
- Befunge
- Factor
- Forth
- Joy (all functions work on parameter stacks instead of named parameters)
- Piet
- Poplog via its implementation language POP-11
- PostScript
- RPL
- S-Lang

Synchronous languages

Synchronous programming languages are optimized for programming reactive systems, systems that are often interrupted and must respond quickly. Many such systems are also called realtime systems, and are used often in embedded systems.

Examples:

- Argus
- Averest
- Esterel
- Lustre
- Signal
- Céu (programming language)

Shading languages

A shading language is a graphics programming language adapted to programming shader effects. Such language forms usually consist of special data types, like "color" and "normal". Due to the variety of target markets for 3D computer graphics.

Real-time rendering

They provide both higher hardware abstraction and a more flexible programming model than previous paradigms which hardcoded transformation and shading equations. This gives the programmer greater control over the rendering process and delivers richer content at lower overhead.

- Adobe Graphics Assembly Language (AGAL)[31]
- ARB assembly language (ARB assembly)
- OpenGL Shading Language (GLSL or glslang)
- High-Level Shading Language (HLSL) or DirectX Shader Assembly Language
- PlayStation Shader Language (PSSL)
- Metal Shading Language (MSL)
- Cg

Offline rendering

Shading languages used in offline rendering produce maximum image quality. Processing such shaders is time-consuming. The computational power required can be expensive because of their ability to produce photorealistic results.

- RenderMan Shading Language (RSL)
- Open Shading Language (OSL)

Syntax-handling languages

These languages assist with generating lexical analyzers and parsers for context-free grammars.

- ANTLR
- Coco/R (EBNF with semantics)
- GNU bison (FSF's version of Yacc)
- GNU Flex (FSF version of Lex)
- lex (Lexical Analysis, from Bell Labs)
- M4
- Parsing expression grammar (PEG)
- Prolog
- Emacs Lisp
- Lisp
- Raku[32]
- SableCC
- Scheme
- yacc (yet another compiler-compiler, from Bell Labs)
- JavaCC

System languages

The **system programming languages** are for low-level tasks like memory management or task management. A system programming language usually refers to a programming language used for system programming; such languages are designed for writing system software, which usually requires different development approaches when compared with application software.

System software is computer software designed to operate and control the computer hardware, and to provide a platform for running application software. System software includes software categories such as operating systems, utility software, device drivers, compilers, and linkers. Examples of system languages include:

Language	Originator	First appeared	Influenced by	Used for
ESPOL	Burroughs Corporation	1961	ALGOL 60	MCP
PL/I	IBM, SHARE	1964	ALGOL 60, FORTRAN, some COBOL	Multics
PL360	Niklaus Wirth	1968	ALGOL 60	ALGOL W
C	Dennis Ritchie	1969	BCPL	Most operating system kernels, including Windows NT and most Unix-like systems
PL/S	IBM	196x	PL/I	OS/360
BLISS	Carnegie Mellon University	1970	ALGOL-PL/I[33]	VMS (portions)
PL/8	IBM	197x	PL/I	AIX
PL/MP and PL/MI	IBM	197x	PL/I	CPF, OS/400
PL-6	Honeywell, Inc.	197x	PL/I	CP-6
SYMPL	CDC	197x	JOVIAL	NOS subsystems, most compilers, FSE editor
C++	Bjarne Stroustrup	1979	C, Simula	See C++ Applications[34]
Ada	Jean Ichbiah, S. Tucker Taft	1983	ALGOL 68, Pascal, C++, Java, Eiffel	Embedded systems, OS kernels, compilers, games, simulations, CubeSat, air traffic control, and avionics
D	Digital Mars	2001	C++	Multiple domains[35]
Nim	Andreas Rumpf	2008	Ada, Modula-3, Lisp, C++, Object Pascal, Python, Oberon	OS kernels, compilers, games
Rust	Mozilla Research[36]	2010	C++, Haskell, Erlang, Ruby	Servo layout engine, Redox OS
Swift	Apple Inc.	2014	C, Objective-C, Rust	macOS, iOS app development[e]
Zig	Andrew Kelley	2016	C, C++, LLVM IR, Go, Rust, JavaScript	As a replacement for C
V (Vlang)	Alexander Medvednikov	2019	C, Go, Oberon-2, Rust, Swift, Kotlin	Vinix OS, OS kernels, compilers, games

Transformation languages

Transformation languages serve the purpose of transforming (translating) source code specified in a certain formal language into a defined destination format code. It is most commonly used in intermediate components of more complex super-systems in order to adopt internal results for input into a succeeding processing routine.

- ATL
- AWK
- MOFM2T
- QVT
- Raku
- XSLT is the best known XML transformation language

Visual languages

Visual programming languages let users specify programs in a two-(or more)-dimensional way, instead of as one-dimensional text strings, via graphic layouts of various types. Some dataflow programming languages are also visual languages.

- Analytica
- Blockly
- Clickteam Fusion
- DRAKON
- Fabrik
- Grasshopper
- Max
- NXT-G
- Pict
- Prograph
- Pure Data
- Quartz Composer
- Scratch (written in and based on Squeak, a version of Smalltalk)
- Snap!
- Simulink
- Spreadsheets
- Stateflow
- Subtext
- ToonTalk
- VEE
- VisSim
- Vvvv
- XOD

Wirth languages

Computer scientist Niklaus Wirth designed and implemented several influential languages.

- ALGOL W
- Euler
- Modula
  - Modula-2, Modula-3, variants
    - Obliq Modula 3 variant
- Oberon (Oberon, Oberon-07, Oberon-2)
  - Component Pascal
  - Oberon-2
- Pascal
  - Object Pascal (umbrella name for Delphi, Free Pascal, Oxygene, others)

XML-based languages

These are languages based on or that operate on XML.

- Ant
- Cω
- ECMAScript for XML
- MXML
- LZX
- XAML
- XPath
- XQuery
- XProc
- eXtensible Stylesheet Language Transformations (XSLT)

See also

- Programming paradigm
- IEC 61131-3 – a standard for programmable logic controller (PLC) languages
- List of educational programming languages
- List of markup languages
- Esoteric programming language

Notes

a. The objects of SQL are collections of database records, called tables. A full programming language can specify algorithms, irrespective of runtime. Thus an algorithm can be considered to generate usable results. In contrast, SQL can only select records that are limited to the current collection, the data at hand in the system, rather than produce a statement of the correctness of the result.

b. A notable exception would be the Soviet/Russian 1801 series CPU, which originally used their own domestic ISA, but were later redesigned to be PDP-11 compatible as a policy decision.

c. Submodels are not listed, only base models.

d. The concept of *object* with the traditional single-dispatch OO semantics is not present in Julia, instead with the more general multiple dispatch on different types at runtime.
e. Swift uses automatic reference counting.

References

1. "Operators" (https://docs.raku.org/language/operators#Operators). Retrieved 2024-05-13.
2. "wrap" (https://docs.raku.org/routine/wrap).
3. " "Aspects in Raku" " (https://conf.raku.org/talk/142).
4. "Christopher Diggins: What is a concatenative language" (http://drdobbs.com/blogs/architecture-and-design/228701299). Drdobbs.com. 2008-12-31. Retrieved 2013-07-01.
5. "Feed operator" (https://docs.perl6.org/routine/==%3E).
6. Documentation » The Python Standard Library » Concurrent Execution (https://docs.python.org/3/library/concurrency.html)
7. "Channels and other mechanisms" (https://docs.perl6.org/language/concurrency).
8. "ProblemSolver" (https://raku.land/github:FCO/ProblemSolver).
9. "Indentation based syntax · rsdn/nemerle Wiki" (https://github.com/rsdn/nemerle/wiki/Indentation-based-syntax). *GitHub*. Retrieved 2022-03-18.
10. "Solidity: Solidity 0.8.11 documentation" (https://docs.soliditylang.org/en/v0.8.11/).
11. "Iterator" (https://docs.raku.org/type/Iterator).
12. "Memory Management · BlitzMax" (https://blitzmax.org/docs/en/language/memory_management/). Retrieved 2023-07-14.
13. "Pointers · BlitzMax" (https://blitzmax.org/docs/en/language/pointers/). Retrieved 2023-07-14.
14. "BRL.Blitz · BlitzMax" (https://blitzmax.org/docs/en/api/brl/brl.blitz/). Retrieved 2023-07-14.
15. "Using Pointers in an ILE COBOL Program - IBM Documentation" (https://www.ibm.com/docs/en/i/7.3?topic=considerations-using-pointers-in-ile-cobol-program). *IBM*. Retrieved 2023-07-14.
16. "HEAP - IBM Documentation" (https://www.ibm.com/docs/en/zos/2.3.0?topic=options-heap). *IBM*. Retrieved 2023-07-14.
17. "SOM-based OO COBOL language elements that are changed - IBM Documentation" (https://www.ibm.com/docs/en/cobol-zos/6.1?topic=usbooocp-som-based-oo-cobol-language-elements-that-are-changed). *IBM*. Retrieved 2023-07-14.
18. "Memory Allocation — Cython 3.0.0.dev0 documentation" (https://cython.readthedocs.io/en/latest/src/tutorial/memory_allocation.html). Retrieved 2023-07-14.
19. "Garbage Collection" (https://dlang.org/spec/garbage.html). *D Programming Language*. Retrieved 2022-03-18.
20. "Nim's Memory Management" (https://nim-lang.github.io/Nim/mm.html). Retrieved 2022-03-18.
21. Adobe (February 1999). *PostScript Language Reference, third edition* (https://www.adobe.com/jp/print/postscript/pdfs/PLRM.pdf) (PDF). Addison-Wesley Publishing Company. pp. 56–65.
22. "Native code interoperability – Scala Native 0.4.14 documentation" (https://scala-native.org/en/stable/user/interop.html). Retrieved 2023-07-05.
23. "Projects/Vala/ReferenceHandling - GNOME Wiki!" (https://web.archive.org/web/20240121085048/https://wiki.gnome.org/Projects/Vala/ReferenceHandling#Manual_memory_management_with_pointer_syntax). Archived from the original (https://wiki.gnome.org/Projects/Vala/ReferenceHandling#Manual_memory_management_with_pointer_syntax) on 2024-01-21. Retrieved 2022-03-21.

24. "Understanding Ownership - The Rust Programming Language" (https://doc.rust-lang.org/nightly/book/ch04-00-understanding-ownership.html). *doc.rust-lang.org*.
25. "Smart Pointers - The Rust Programming Language" (https://doc.rust-lang.org/nightly/book/second-edition/ch15-00-smart-pointers.html). *doc.rust-lang.org*.
26. Jon Bentley (AT&T) August 1986 *CACM* **29** (8) "Little Languages", pp 711-721 from his Programming Pearls column (http://www.cs.toronto.edu/~chechik/courses18/csc2125/paper13.pdf)
27. "Meta-programming: What, why and how" (https://perl6advent.wordpress.com/2011/12/14/meta-programming-what-why-and-how/). 2011-12-14.
28. "Procedural Macros for Generating Code from Attributes" (https://doc.rust-lang.org/nightly/book/ch19-06-macros.html#procedural-macros-for-generating-code-from-attributes). *doc.rust-lang.org*.
29. "Classes and Roles" (https://docs.perl6.org/language/classtut).
30. "Meta-object protocol (MOP)" (https://docs.perl6.org/language/mop).
31. Scabia, Marco. "What is AGAL" (https://www.adobe.com/devnet/flashplayer/articles/what-is-agal.html). *Adobe Developer Connection*. Adobe. Retrieved 8 May 2018.
32. "Grammars" (https://docs.raku.org/language/grammars).
33. Wulf, W.A.; Russell, D.B.; Haberman, A.N. (December 1971). "BLISS: A Language for Systems Programming". *Communications of the ACM*. **14** (12): 780–790. CiteSeerX 10.1.1.691.9765 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.691.9765). doi:10.1145/362919.362936 (https://doi.org/10.1145%2F362919.362936). S2CID 9564255 (https://api.semanticscholar.org/CorpusID:9564255).
34. "C++ Applications" (http://www.stroustrup.com/applications.html).
35. "Organizations using the D Language" (https://dlang.org/orgs-using-d.html). *D Programming Language*.
36. "Mozilla Research" (https://www.mozilla.org/en-US/research/). 1 January 2014.
