A programming language is a system of notation for writing computer programs.

Programming languages are described in terms of their syntax (form) and semantics (meaning), usually defined by a formal language. Languages usually provide features such as a type system, variables and mechanisms for error handling. An implementation of a programming language is required in order to execute programs, namely a compiler or an interpreter. An interpreter directly executes the source code, while a compiler produces an executable program.

Computer architecture has strongly influenced the design of programming languages, with the most common type (imperative languages—which implement operations in a specified order) developed to perform well on the popular von Neumann architecture. While early programming languages were closely tied to the hardware, over time they have developed more abstraction to hide implementation details for greater simplicity.

Thousands of programming languages—often classified as imperative, functional, logic, or object-oriented—have been developed for a wide variety of uses. Many aspects of programming language design involve tradeoffs—for example, exception handling simplifies error handling, but at a performance cost. Programming language theory is the subfield of computer science that studies the design, implementation, analysis, characterization, and classification of programming languages.

Definitions

There are a variety of criteria that may be considered when defining what constitutes a programming language.

Computer languages vs programming languages

The term computer language is sometimes used interchangeably with programming language.[2] However, the usage of both terms varies among authors, including the exact scope of each. One usage describes programming languages as a subset of computer languages.[3] Similarly, languages used in computing that have a different goal than expressing computer programs are generically designated computer languages. For instance, markup languages are sometimes referred to as computer languages to emphasize that they are not meant to be used for programming.[4] One way of classifying computer languages is by the computations they are capable of expressing, as described by the theory of computation. The majority of practical programming languages are Turing complete,[5] and all Turing complete languages can implement the same set of algorithms. ANSI/ISO SQL-92 and Charity are examples of languages that are not Turing complete, yet are often called programming languages.[6][7] However, some authors restrict the term "programming language" to Turing complete languages.[1][8]

Another usage regards programming languages as theoretical constructs for programming abstract machines and computer languages as the subset thereof that runs on physical computers, which have finite hardware resources.[9] John C. Reynolds emphasizes that formal specification languages are just as much programming languages as are the languages intended for execution. He also argues that textual and even graphical input formats that affect the behavior of a computer are programming languages, despite the fact they are commonly not Turing-complete, and remarks that ignorance of programming language concepts is the reason for many flaws in input formats.