



**WIKIPEDIA**  
*The Free Encyclopedia*

[Main Page](#)  
[Recent changes](#)  
[Random page](#)  
[Current events](#)

[Edit this page](#)  
[Discuss this page](#)  
[Page history](#)  
[What links here](#)  
[Related changes](#)

[Special pages](#)  
[Bug reports](#)  
[Donations](#)

[Main Page](#) | [Recent changes](#) | [Edit this page](#) | [Page history](#)  
[Printable version](#)

Not logged in  
[Log in](#) | [Help](#)

[Go](#) [Search](#)

Other languages: [Deutsch](#) | [Esperanto](#) | [Español](#) | [Français](#) | [Italiano](#) | [? ? ?](#)  
[\(Nihongo\)](#) | [Nederlands](#) | [Polski](#)

## Prolog

From Wikipedia, the free encyclopedia.

**Prolog** is a leading [logical programming](#) language. It was created by [Alain Colmerauer](#) around [1970s](#). It was an attempt to make a programming language that enables to express logic instead of carefully specifying instructions on the computer.

Prolog is used in many [artificial intelligence](#) programs, but its syntax and semantics are very simple and clear (the original goal was to provide a tool to computer-illiterate linguists). The name prolog is an acronym for PROgramming in LOGic.

Prolog is based on [predicate calculus](#) (more precisely [first-order predicate calculus](#)); however it is restricted to allow only [Horn clauses](#). Execution of a Prolog program is effectively an application of theorem proving by [first order resolution](#). Fundamental concepts are [unification](#), [tail recursion](#) and [backtracking](#).

**Table of contents** [\[hide\]](#)

- [1 Data types](#)
  - [1.1 Atoms](#)
  - [1.2 Numbers](#)
  - [1.3 Variables](#)
  - [1.4 Terms](#)
  - [1.5 Lists](#)
  - [1.6 Strings](#)
- [2 Facts](#)
- [3 Rules](#)
- [4 Implementations](#)
- [5 References](#)

## Data types

Prolog does not employ [data types](#) in the way usual in the common programming languages. We may rather speak about Prolog lexical elements instead of data types.

### Atoms

The text constants are introduced by means of atoms. An atom is a sequence consisting of letters, numbers and underscore, which begins with a lower-case letter. Usually, if non-alphanumeric atom is needed, it is surrounded with

apostrophes (e.g. '+' is an atom, + is an operator).

## Numbers

Most Prolog implementations don't differ between integral and real numbers.

## Variables

Variables are denoted by a string consisting of letters, numbers and underscore characters, and beginning with an upper-case letter. In the Prolog environment, a variable is not a container, which can be assigned to (unlike procedural programming languages). Its behaviour is closer to a pattern, which is increasingly specified by [unification](#).

The so called *anonymous variable* is written as a single underscore (`_`).

## Terms

Terms are the only way Prolog can represent complex data. A term consists of a head, also called functor (which must be an atom) and parameters (unrestricted types). The number of parameters, so called [arity](#) of term, is significant. A term is identified by its head and arity, usually written as functor/arity.

## Lists

A list isn't a standalone data type, because it is defined by a recursive construction (using term `'./2`):

1. atom `[]` is an empty list
2. if `L` is a list and `X` is an element, then the term `'(X, L)` is a list. The first element is `X`, which is followed by the contents of `L`. Syntactic shortcut is `[X | L]`.

For programmer's convenience, the lists can be constructed and deconstructed in a variety of ways.

- Element enumeration: `[abc, 1, f(x), Y, g(A,rst)]`
- Prepending single element: `[abc | LI]`
- Prepending multiple elements: `[abc, 1, f(x) | L2]`
- Term expansion: `'(abc, '(1, '(f(x), '(Y, '(g(A,rst), [])))))`

## Strings

Strings are usually written as a sequence of characters surrounded by quotes. They are often internally represented as lists of ASCII codes.

## Facts

Programming in Prolog is very different from programming in a procedural language. In Prolog you supply a database of facts and rules; you can then perform queries on the database. The basic unit of Prolog is the predicate, which is defined to be true. A predicate consists of a head and a number of arguments. For example:

```
cat(tom).
```

Here 'cat' is the head, and 'tom' is the argument. Here are some sample queries you could ask a Prolog interpreter basing on this fact:

```
?- cat(tom).
   yes.
```

```
?- cat(X).
   X = tom;
   no.
```

Predicates are usually defined to express some fact the program knows about the world. In most of the cases, the usage of predicates requires a certain convention. Thus, which version of the two below would signify that Pat is the father of Sally?

```
father(sally,pat).
father(pat,sally).
```

In both cases 'father' is the head and 'sally' and 'pat' are arguments. However in the first case, Sally comes first in the argument list, and in the second, Pat comes first (the order in the argument list matters). The first case is an example of a definition in [Verb Subject Object](#) order, and the second of [Verb Object Subject](#) order. Since Prolog does not understand English, both versions are fine so far as it is concerned; however it is good programming style to stick to either convention during the writing of a single program, so that to avoid writing something like

```
father(pat,sally).
father(jessica,james).
```

Some predicates are built in into the language, and allow a Prolog program to perform routine activities (such as input/output, using graphics and otherwise communicating with the operating system). For example, the predicate `write` can be used for output to the screen. Thus,

```
write('Hello')
```

will display the word 'Hello' on the screen.

## Rules

The second type of statements in Prolog is rules. An example of a rule is

```
light(on) :- switch(on).
```

The ":-" means "if"; this rule means light(on) is true if switch(on) is true. Rules can also make use of [variables](#); variables begin with capital letters while constants begin with lower case letters. For example,

```
father(X,Y) :- parent(X,Y),male(Y).
```

This means "if someone is a parent of someone and he's male, he is a father". The antecedent and consequent are in reverse order to that normally found in logic. It is possible to place multiple predicates in a consequent, joined with conjunction, for example:

```
a,b,c :- d.
```

which is simply equivalent to three separate rules:

```
a :- d.
b :- d.
c :- d.
```

What is not allowed are rules like:

```
a;b :- c.
```

that is "if c then a or b". This is because of the restriction to Horn clauses.

## Implementations

- Open Prolog (<http://www.cs.tcd.ie/open-prolog/>)
- Ciao Prolog (<http://www.clip.dia.fi.upm.es/Software/Ciao>)
- GNU Prolog (<http://gnu-prolog.inria.fr>)
- YAP Prolog (<http://www.ncc.up.pt/~vsc/Yap>)
- SWI Prolog (<http://www.swi-prolog.org>)
- Visual Prolog (<http://www.visual-prolog.com>)
- SICStus Prolog (<http://www.sics.se/sicstus/>)
- Amzi! Prolog (<http://www.amzi.com/>)

## References

- [Short intro - learning unit with exercices](#)
- [More comprehensive course](#)
- [A prolog interpreter that runs in the browser](#)
- [Prolog: The ISO standard](#)

---

[Edit this page](#) | [Discuss this page](#) | [Page history](#) | [What links here](#) | [Related changes](#)

Other languages: [Deutsch](#) | [Esperanto](#) | [Español](#) | [Français](#) | [Italiano](#) | [? ? ?](#)  
([Nihongo](#)) | [Nederlands](#) | [Polski](#)

[Main Page](#) | [About Wikipedia](#) | [Recent changes](#) |

Go  Search

This page was last modified 17:19, 4 Dec 2003. All text is available under the terms of the [GNU Free Documentation License](#).