General principles of good programming

- Correctness: a program should do what it is supposed to do.
- Efficiency: a program should not use excessive time or space
- Transparency: a program should be easy to read and understand
- Modifiability: a program should be easy to modify and extend
- Robustness: a program should respond gracefully to any user input
- Documentation: a program should at least be commented

Rules of good Prolog style

- ✔ Clauses should be short, containing no more than a few goals each
- ✔ Definitions of a relation should be short, unless it has a uniform structure (such as a collection of facts)
- ✔ Functors, variables and atoms should have informative names
- ✔ Spacing, blank lines, indentation, and other stylistic conventions should be used consistently
- ✔ The cut operator and the not predicate should be used with care
- ✔ `assert` and `retract` reduce transparency of a program and should be used sparingly
- ✔ ‘;’ can make definitions less readable and should be used sparingly

An example of bad Prolog style

```
merge(List1,List2,List3) :- List1 = [],!, List3 = List2;
   List2 = [],!,List3 = List1;
List1 = [X|Rest1],
      List2 = [Y|Rest2],
   (X<Y, !,
Z=X, merge(Rest1,List2,Rest3); Z=Y,
      merge(List1,Rest2,Rest3)), List3 = [Z|Rest3].
```

An example of good Prolog style

```
/*
** merge(L1,L2,L3) is true if L1,L2,L3 are lists of
** numbers, and L3 is the result of merging L1 and L2
** according to the < relation
*/

merge([],List,List) :- !. % prevent redundant solutions

merge(List,[],List). % if one list empty, nothing to do

merge([X|Rest1],[Y|Rest2],[X|Rest3]) :-
    X<Y, !, % if X < Y, put X first in merged list
    merge(Rest1,[Y|Rest2],Rest3).

merge(List1,[Y|Rest2],[Y|Rest3]) :-
    % else put Y first in merged list
    merge(List1,Rest2,Rest3).
```