

# SWI-Prolog

---

For working in the Prolog programming language we will use the SWI-Prolog editor. You can download SWI-Prolog from the following address: <http://www.swi-prolog.org/Download.html>

Install SWI-Prolog choosing the default settings during the installation process. Once installed, the following icon will appear (which can be used to start the program):



When started, the console will appear:

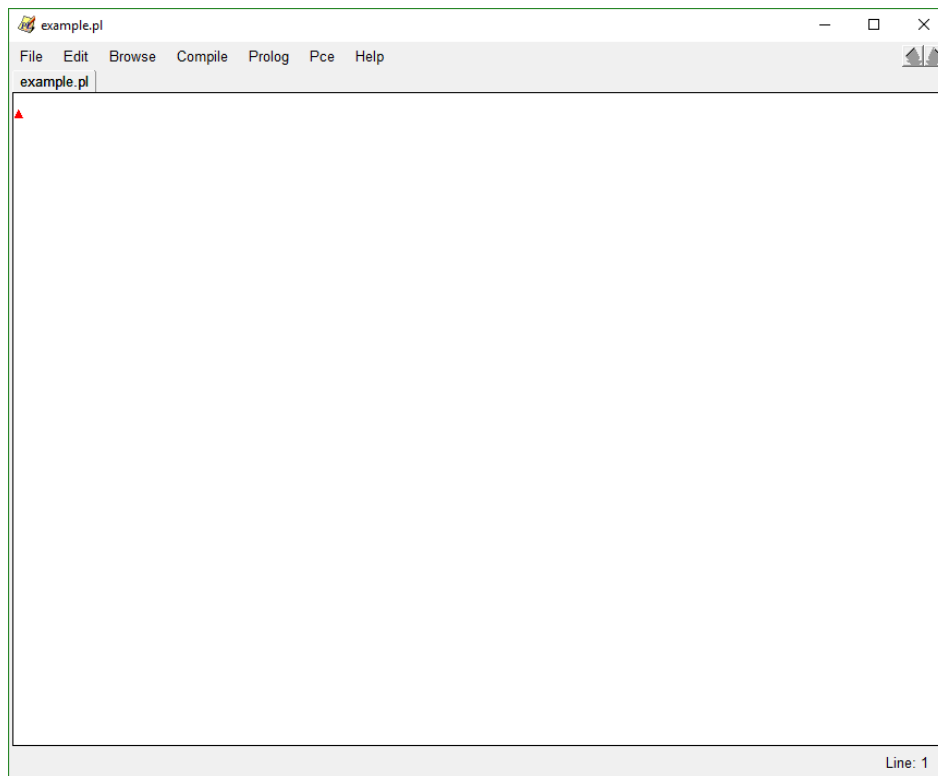
The screenshot shows a window titled 'SWI-Prolog (Multi-threaded, version 7.2.3)'. The window has a menu bar with 'File', 'Edit', 'Settings', 'Run', 'Debug', and 'Help'. The main area is a text console with the following text:

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

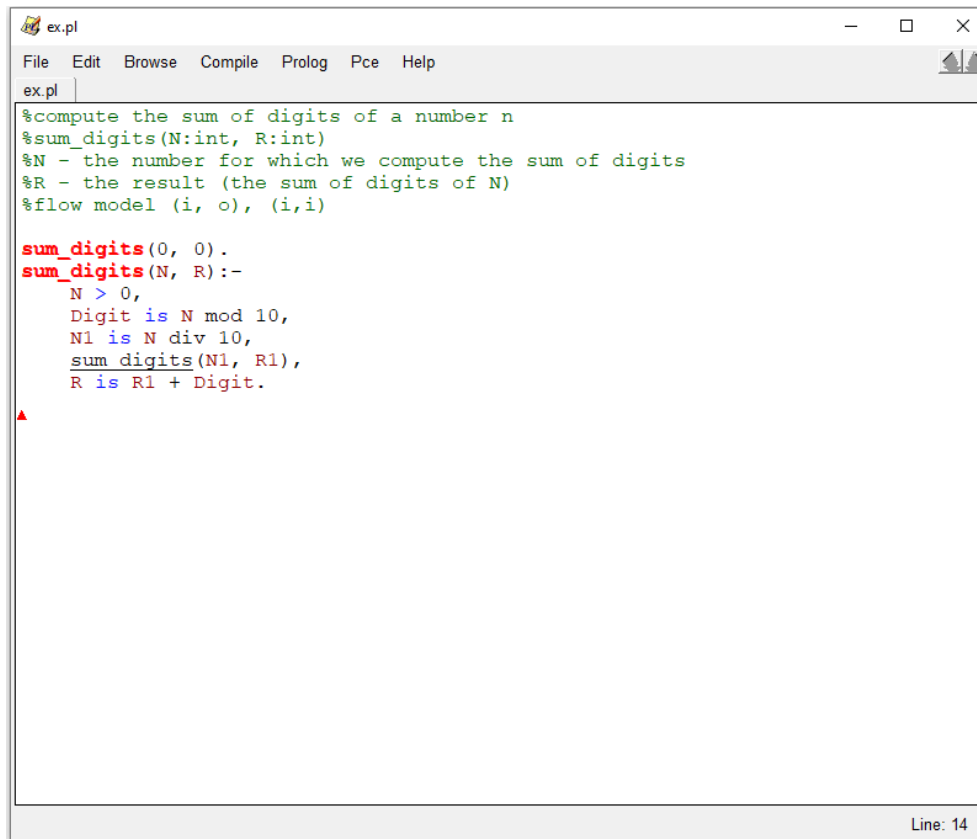
For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- █
```

For implementing the predicates you can use any text editor, or you can use SWI-Prolog's own editor, which can be opened with the File -> New ... menu item. You will have to provide a name for your file (with the .pl or .pro extension) and the editor will open:



In this window the predicate implementations can be written. While SWI-Prolog allows you to just write the implementation of the predicates (without any prior declarations), before implementing any predicate, you will have to write some specifications as comments. The specifications have to contain the name of the predicate, the parameter list and the type of the parameters, semantification of the parameters and the flow models. Comments start with %.



The screenshot shows a Prolog editor window titled 'ex.pl'. The menu bar includes File, Edit, Browse, Compile, Prolog, Pce, and Help. The code in the editor is as follows:

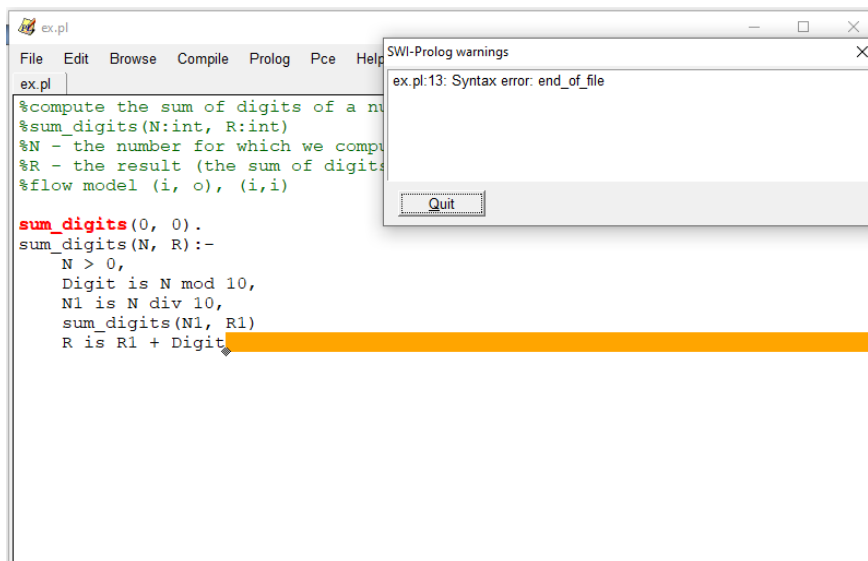
```
%compute the sum of digits of a number n
%sum_digits(N:int, R:int)
%N - the number for which we compute the sum of digits
%R - the result (the sum of digits of N)
%flow model (i, o), (i,i)

sum_digits(0, 0).
sum_digits(N, R):-
    N > 0,
    Digit is N mod 10,
    N1 is N div 10,
    sum_digits(N1, R1),
    R is R1 + Digit.
```

A red arrow points to the end of the last line of code. The status bar at the bottom right indicates 'Line: 14'.

When the implementation is done, the code can be compiled and loaded using the Compile -> Compile buffer option.

If there are syntax problems in the code, after the compilation a window will appear which contains these errors. For example, if we remove some commas from the previous code, after compiling we will get the following message (and the orange color in the editor marks the problem).



The screenshot shows the same Prolog editor window, but with a syntax error. A dialog box titled 'SWI-Prolog warnings' is open, displaying the message 'ex.pl:13: Syntax error: end\_of\_file'. The line 'sum\_digits(N1, R1)' in the code is highlighted in orange, indicating the location of the error. The 'Quit' button is visible in the dialog box.

If everything is OK, after compiling a green message appears in the console (the red message is from the previous compilation of the buggy version):

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
ERROR: [Thread pce] c:/users/zsu/documents/prolog/ex.pl:13:19: Syntax error: Unexpected end of file
% c:/users/zsu/documents/prolog/ex compiled 0.00 sec, -1 clauses
```

Now we can call the predicate from the command line. It is important to end the call with a period.

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- sum_digits(23145, X).
X = 15 ;
false.

?- sum_digits(111, X).
X = 3 ;
false.

?-
```

One file can contain several predicates.

```
ex.pl
File Edit Browse Compile Prolog Pce Help
ex.pl
%compute the sum of digits of a number n
%sum_digits(N:int, R:int)
%N - the number for which we compute the sum of digits
%R - the result (the sum of digits of N)
%flow model (i, o), (i,i)

sum_digits(0, 0).
sum_digits(N, R):-
    N > 0,
    Digit is N mod 10,
    N1 is N div 10,
    sum_digits(N1, R1),
    R is R1 + Digit.

%replace in a list all the elements with the sum of their digits
%replace(L: list, LR: list)
%L - initial list
%LR - resulting list
%flow model (i, o), (i, i)
replace([], []).
replace([H|T], LR):-
    sum_digits(H, SH),
    replace(T, LR1),
    LR = [SH|LR1].
```

```
SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- sum_digits(23145, X).
X = 15 ;
false.

?- sum_digits(111, X).
X = 3 ;
false.

?-
% c:/users/zsu/documents/prolog/ex compiled 0.00 sec, 1 clauses
?- replace([42, 88, 121, 31, 9], R).
R = [6, 16, 4, 4, 9] ;
false.

?- |
```

If a predicate is not working correctly, we can debug it, using the trace command (do not forget the period at the end of the call).

SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.  
Please run `?- license.` for legal details.

For online help and background, visit <http://www.swi-prolog.org>

For built-in help, use `?- help(Topic).` or `?- apropos(Word).`

`?- sum_digits(23145, X).`

`X = 15 ;`

**false.**

`?- sum_digits(111, X).`

`X = 3 ;`

**false.**

`?-`

`% c:/users/zsu/documents/prolog/ex compiled 0.00 sec, 1 clauses`

`?- replace([42, 88, 121, 31, 9], R).`

`R = [6, 16, 4, 4, 9] ;`

**false.**

`?- trace.`

**true.**

Now we are in trace mode. Any call performed now will be executed step-by-step, and you can see the instructions and the calls that are made during the execution.

`?- trace.`

**true.**

`[trace] ?- sum_digits(239, X).`

**Call:** (8) `sum_digits(239, _980) ? creep`

**Call:** (9) `239>0 ? creep`

**Exit:** (9) `239>0 ? creep`

**Call:** (9) `_1204 is 239 mod 10 ? creep`

**Exit:** (9) `9 is 239 mod 10 ? creep`

**Call:** (9) `_1210 is 239 div 10 ? creep`

**Exit:** (9) `23 is 239 div 10 ? creep`

**Call:** (9) `sum_digits(23, _1212) ? creep`

**Call:** (10) `23>0 ? creep`

**Exit:** (10) `23>0 ? creep`

**Call:** (10) `_1216 is 23 mod 10 ? creep`

**Exit:** (10) `3 is 23 mod 10 ? creep`

**Call:** (10) `_1222 is 23 div 10 ? creep`

**Exit:** (10) `2 is 23 div 10 ? creep`

**Call:** (10) `sum_digits(2, _1224) ? creep`

**Call:** (11) `2>0 ? creep`

**Exit:** (11) `2>0 ? creep`

**Call:** (11) `_1228 is 2 mod 10 ? creep`

**Exit:** (11) `2 is 2 mod 10 ? creep`

**Call:** (11) `_1234 is 2 div 10 ? creep`

**Exit:** (11) `0 is 2 div 10 ? creep`

**Call:** (11) `sum_digits(0, _1236) ? creep`

**Exit:** (11) `sum_digits(0, 0) ? creep`

**Call:** (11) `_1240 is 0+2 ? creep`

**Exit:** (11) `2 is 0+2 ? creep`

**Exit:** (10) `sum_digits(2, 2) ? creep`

**Call:** (10) `_1246 is 2+3 ? creep`

**Exit:** (10) `5 is 2+3 ? creep`

**Exit:** (9) `sum_digits(23, 5) ? creep`

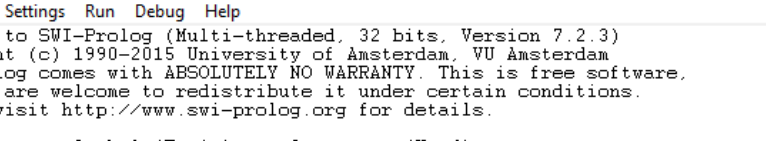
**Call:** (9) `_980 is 5+9 ? creep`

**Exit:** (9) `14 is 5+9 ? creep`

**Exit:** (8) `sum_digits(239, 14) ? creep`

`X = 14 |`

SWI-Prolog has a graphical debugger as well. For activating the graphical debugger, use the Debug -> Graphical debugger option.



SWI-Prolog (Multi-threaded, version 7.2.3)

File Edit Settings Run Debug Help

Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 7.2.3)  
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,  
and you are welcome to redistribute it under certain conditions.  
Please visit <http://www.swi-prolog.org> for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-

% The graphical front-end will be used for subsequent tracing

The screenshot shows a Prolog IDE with the following components:

- File Path:** c:/users/zsu/documents/prolog/ex.pl
- Tool Bar:** Contains icons for File, Edit, View, Compile, Help, and various Prolog-specific actions like backtracking, goal execution, and debugging.
- Bindings:**
  - N = 23
  - Digit = 3
  - N1 = 2
- Call Stack:**
  - 8: sum\_digits/2
  - 9: sum\_digits/2
  - 10: sum\_digits/2
- Main Window (Prolog Code):**

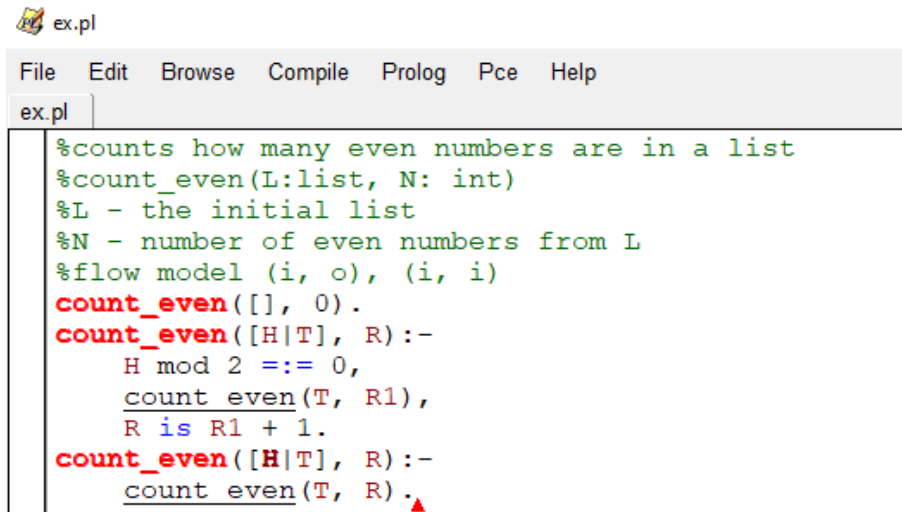
```

%compute the sum of digits of a number n
%sum_digits(N:int, R:int)
%N - the number for which we compute the sum of digits
%R - the result (the sum of digits of N)
%flow model (i, o), (i,i)

sum_digits(0, 0).
sum_digits(N, R):-
    N > 0,
    Digit is N mod 10,
    N1 is N div 10,
    sum_digits(N1, R1),
    R is R1 + Digit.

%replace in a list all the elements with the sum of their digits
%replace(L: list, LR: list)
%L - initial list
%LR - resulting list
%flow model (i, o), (i, i)
replace([], []).
replace([H|T], LR):-
    sum_digits(H, SH),
    replace(T, LR1),
    LR = [SH|LR1].
  
```

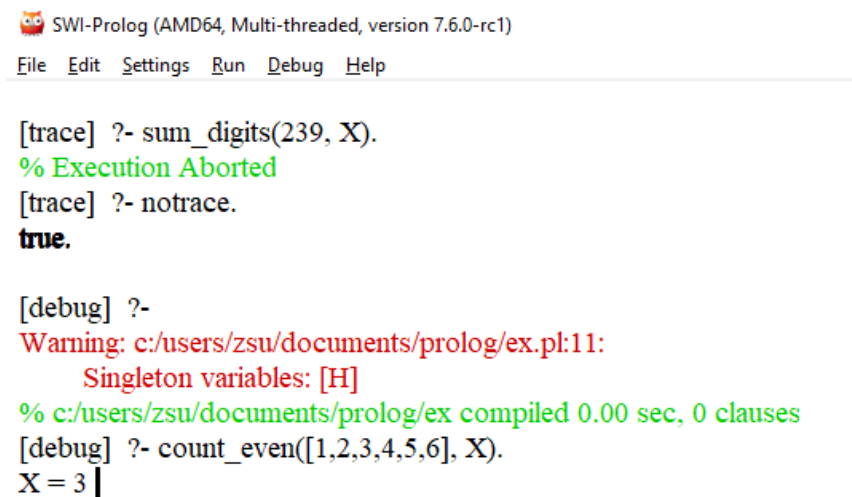
Prolog uses backtracking for solving problems and for finding all the solutions for a problem, not just one solution. Thus, if at a given point several clauses fit, Prolog will enter all of them and, probably, will return a solution for each of them. For example, let's look at the following implementation:



```

%counts how many even numbers are in a list
%count_even(L:list, N: int)
%L - the initial list
%N - number of even numbers from L
%flow model (i, o), (i, i)
count_even([], 0).
count_even([H|T], R):-
    H mod 2 == 0,
    count_even(T, R1),
    R is R1 + 1.
count_even([_H|T], R):-
    count_even(T, R).
  
```

If we run the above code for the list [1,2,3,4,5,6], we will have the following result:



```

SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help

[trace] ?- sum_digits(239, X).
% Execution Aborted
[trace] ?- notrace.
true.

[debug] ?-
Warning: c:/users/zsu/documents/prolog/ex.pl:11:
Singleton variables: [H]
% c:/users/zsu/documents/prolog/ex compiled 0.00 sec, 0 clauses
[debug] ?- count_even([1,2,3,4,5,6], X).
X = 3
  
```

The result is correct. However, the call is not over yet. If we press ; (semicolon) Prolog continues searching for solutions. Pressing semicolon every time a new solution is returned we will get other the following other solutions:



```

SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help
% c:/users/zsu/documents/prolog/ex compiled 0.00 sec, 0 clauses
[debug] ?- count_even([1,2,3,4,5,6], X).
X = 3 ;
X = 2 ;
X = 2 ;
X = 1 ;
X = 2 ;
X = 1 ;
X = 1 ;
X = 0.

[debug] ?-|

```

The reason for so many solutions is the lack of condition in the last clause. Thus, whenever the first element from the list is an even number, both the second and the third clauses fit. Prolog will enter both of them and will return us a lot of solution (two solution for every situation when the list starts with an even number) out of which all, except for the first one, are incorrect. For avoiding such a situation (having multiple solutions), make sure that you have a condition on every clause, and whenever you execute your code, use semicolon to check if the predicate returns multiple solutions or not...for deterministic problems (problems where there is one single solution) it is not correct to have multiple solutions (not even if it is the same correct result several times).

Adding a condition to the last clause, we will have:

```

ex.pl
File Edit Browse Compile Prolog Pce Help
ex.pl
%counts how many even numbers are in a list
%count_even(L:list, N: int)
%L - the initial list
%N - number of even numbers from L
%flow model (i, o), (i, i)
count_even([], 0).
count_even([H|T], R):-
    H mod 2 == 0,
    count_even(T, R1),
    R is R1 + 1.
count_even([H|T], R):-
    H mod 2 == 1,
    count_even(T, R).

```

And if we run the code again:

```

SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help
X = 2 ;
X = 1 ;
X = 1 ;
X = 0.

[debug] ?-
% c:/users/zsu/documents/prolog/ex compiled 0.00 sec, -1 clauses
[debug] ?- count_even([1,2,3,4,5,6], X).
X = 3 ;
false.

[debug] ?-

```

Now we have one single solution. The false means that Prolog kept searching for other solutions after returning  $X=3$ , but found no other solutions. It is not a problem if you have a false after the correct result was returned.

If we modify the predicate (introduce a bug) to have a situation which is not covered ( $H \bmod 3 == 2$  is missing) and run the code again we will have:

```

ex.pl [modified]
File Edit Browse Compile Prolog Pce Help
ex.pl [modified]
%counts how many even numbers are in a list
%count_even(L:list, N: int)
%L - the initial list
%N - number of even numbers from L
%flow model (i, o), (i, i)
count_even([], 0).
count_even([H|T], R):-
    H mod 3 == 0,
    count_even(T, R1),
    R is R1 + 1.
count_even([H|T], R):-
    H mod 3 == 1,
    count_even(T, R).

```

```


SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help
[debug] ?-
% c:/users/zsu/documents/prolog/ex compiled 0.00 sec, -1 clauses
[debug] ?- count_even([1,2,3,4,5,6], X).
X = 3 ;
false.

[debug] ?-
% c:/users/zsu/documents/prolog/ex compiled 0.00 sec, 0 clauses
[debug] ?- count_even([1,2,3,4,5,6], X).
false.

```

Now we have just the false as result. This is a problem (with one exception described below), it means that the execution arrived to a situation when no clause matched the current situation.

```

 SWI-Prolog (AMD64, Multi-threaded, version 7.6.0-rc1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

```

```

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

```

```
?- trace.
```

```
true.
```

```

[trace] ?- count_even([1,2,3,4,5,6], X).
Call: (8) count_even([1, 2, 3, 4, 5, 6], _1016) ? creep
Call: (9) 1 mod 3:=0 ? creep
Fail: (9) 1 mod 3:=0 ? creep
Redo: (8) count_even([1, 2, 3, 4, 5, 6], _1016) ? creep
Call: (9) 1 mod 3:=1 ? creep
Exit: (9) 1 mod 3:=1 ? creep
Call: (9) count_even([2, 3, 4, 5, 6], _1016) ? creep
Call: (10) 2 mod 3:=0 ? creep
Fail: (10) 2 mod 3:=0 ? creep
Redo: (9) count_even([2, 3, 4, 5, 6], _1016) ? creep
Call: (10) 2 mod 3:=1 ? creep
Fail: (10) 2 mod 3:=1 ? creep
Fail: (9) count_even([2, 3, 4, 5, 6], _1016) ? creep
Fail: (8) count_even([1, 2, 3, 4, 5, 6], _1016) ? creep
false.


```

With trace we can see that this happens when the list starts with the element 2 and  $2 \bmod 3$  is not 0 and it is not 1.

It is OK to have just a false as answer if we have a Boolean predicate. In this case, false means that the result of the predicate is false. Let's modify our previous predicate to check if there is an even number in the list. Note that, in case of Boolean predicates we no longer have output parameter and that not all the possible cases are implemented. Indeed, implementation of Boolean predicates is a little different:

- No output parameters
- We only write clauses for the case(s) which returns true and for the recursive case(s) (Since we already know that Prolog will automatically return false if it finds a situation when no clause matches).

```

 ex.pl
File Edit Browse Compile Prolog Pce Help
ex.pl
%checks if there is an even number in the list
%check_even(L:list)
%L - the initial list
%flow model (i)
check_even([H|T]) :-
    H mod 2 =:= 0.
check_even([H|T]) :-
    check_even(T).

```

Explanation of the clauses:

- The list contains an even number if the first number is even.
- The list contains an even number if the rest of the list contains an even number.

If we run our predicate:

```
[debug] ?- check_even([1,2,3]).  
true ;  
false.  
  
[debug] ?- check_even([1,3,5]).  
false.
```

The second case is clear. No even number in the [1,3,5] list. For the first case, Prolog returned true (it found the number 2) and when we pressed the semicolon, it continued searching. Since it found no other solutions, it returned false. And this is correct, even if it looks strange.

What happens if the list contains multiple even numbers?

```
[debug] ?- check_even([1,2, 3,4, 5, 6]).  
true ;  
true ;  
true ;  
false.
```

Prolog finds an even number (2) and returns true. If we continue searching for solutions, it will find another even number (4) and return true again. And then for 6 we will get another true. Finally a false for not finding any other even numbers after finding 6. This is not OK, but not because of the false, the problem is that true was returned three times.

The reason is that we do not have conditions on all clauses. The first one has a condition (H is even), but the second has no condition. So when H is an even number, execution will enter both branches and possibly return several solution.