



# PROLOG - comando trace

- SWISH
- SWI-Prolog desde PC

# Uso de trace

The screenshot shows the SWISH Prolog IDE interface. The browser address bar displays `https://swish.swi-prolog.org`. The interface includes a menu bar with `File`, `Edit`, `Examples`, and `Help`. A search bar is located in the top right corner. The main workspace is divided into two panels. The left panel, titled "Program", contains the following Prolog code:

```
1 % hechos
2 perro('Canela').
3 perro('Melchor').
4 perro('Dulce').
5 perro('Sol').
6 gato('Michi').
7 gato('Kitty').
8 lince('lince1').
9 lince('lince2').
10 leon('leon1').
11 leon('leon2').
12 % reglas
13 /*uso de la negación en lugar del OR*/
14 animal_salvaje(A):-lince(A).
15 animal_salvaje(A):-leon(A).
16 animal_domestico(A):-perro(A),not(animal_salvaje(A)).
17 animal_domestico(A):-gato(A),not(animal_salvaje(A)).
18
19
```

The right panel, titled "?-", contains the query:

```
trace, (animal_domestico(X)).
```

At the bottom of the interface, there are tabs for `Examples`, `History`, and `Solutions`. On the far right, there is a checkbox for `table results` and a `Run!` button.

# Uso de trace

The screenshot shows the SWISH Prolog IDE interface. The left pane contains a Prolog program with the following code:

```
1 % hechos
2 perro('Canela').
3 perro('Melchor').
4 perro('Dulce').
5 perro('Sol').
6 gato('Michi').
7 gato('Kitty').
8 lince('lince1').
9 lince('lince2').
10 leon('leon1').
11 leon('leon2').
12 % reglas
13 /*uso de la negación en lugar del OR*/
14 animal_salvaje(A):-lince(A).
15 animal_salvaje(A):-leon(A).
16 animal_domestico(A):-perro(A),not(animal_salvaje(A)).
17 animal_domestico(A):-gato(A),not(animal_salvaje(A)).
18
19
```

The right pane shows the execution trace for the query `trace, (animal_domestico(X)).`. The trace window displays the call `Call: animal_domestico(_6732)` and the query `?- trace, (animal_domestico(X)).`. The bottom right corner has buttons for `Examples`, `History`, `Solutions`, `table results`, and a `Run!` button.

# Uso de trace

The screenshot shows the SWISH web interface at <https://swish.swi-prolog.org>. The interface is divided into two main panels: a code editor on the left and a trace window on the right.

**Code Editor (Left Panel):**

```
1 % hechos
2 perro('Canela').
3 perro('Melchor').
4 perro('Dulce').
5 perro('Sol').
6 gato('Michi').
7 gato('Kitty').
8 lince('lince1').
9 lince('lince2').
10 leon('leon1').
11 leon('leon2').
12 % reglas
13 /*uso de la negación en lugar del OR*/
14 animal_salvaje(A):-lince(A).
15 animal_salvaje(A):-leon(A).
16 animal_domestico(A):-perro(A),not(animal_salvaje(A)).
17 animal_domestico(A):-gato(A),not(animal_salvaje(A)).
18
19
```

**Trace Window (Right Panel):**

The trace window shows the execution of the query `trace, (animal_domestico(X)).`. It displays a sequence of calls and exits, along with the current value of `X`.

**Trace Log:**

- Call: `animal_domestico(_6732)`
- Call: `perro(_514)`
- Exit: `perro('Canela')`
- Call: `not(animal_salvaje('Canela'))`
- Exit: `not(82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Canela'))`
- Exit: `animal_domestico('Canela')`
- X = 'Canela'**
- Reset: `perro(_514)`
- Exit: `perro('Melchor')`
- Call: `not(animal_salvaje('Melchor'))`
- Exit: `not(82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Melchor'))`
- Exit: `animal_domestico('Melchor')`
- X = 'Melchor'**

Navigation buttons: Next, 10, 100, 1,000, Stop.

**Execution Command:**

```
?- trace, (animal_domestico(X)).
```

At the bottom of the interface, there are tabs for Examples, History, and Solutions, and a button to Run the code.

# Uso de trace

The screenshot shows the SWISH Prolog IDE interface. The left pane contains a Prolog program with the following code:

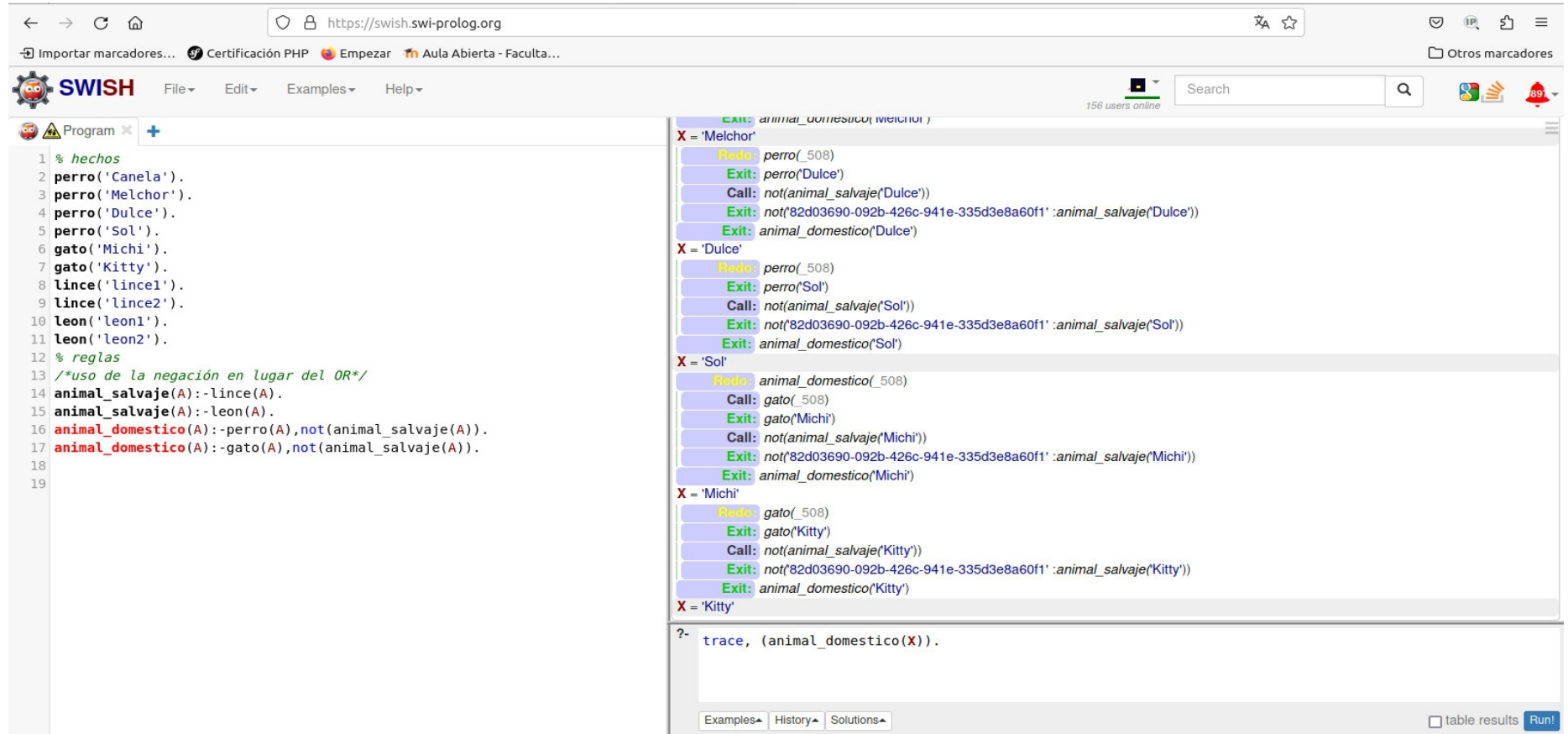
```
1 % hechos
2 perro('Canela').
3 perro('Melchor').
4 perro('Dulce').
5 perro('Sol').
6 gato('Michi').
7 gato('Kitty').
8 lince('lince1').
9 lince('lince2').
10 leon('leon1').
11 leon('leon2').
12 % reglas
13 /*uso de la negación en lugar del OR*/
14 animal_salvaje(A):-lince(A).
15 animal_salvaje(A):-leon(A).
16 animal_domestico(A):-perro(A),not(animal_salvaje(A)).
17 animal_domestico(A):-gato(A),not(animal_salvaje(A)).
18
19
```

The right pane displays the execution trace for the query `trace, (animal_domestico(X)).`. The trace shows the following steps:

- X = 'Dulce'**
  - Redo: `perro(_508)`
  - Exit: `perro('Sol')`
  - Call: `not(animal_salvaje('Sol'))`
  - Exit: `not('82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Sol'))`
  - Exit: `animal_domestico('Sol')`
- X = 'Sol'**
  - Redo: `animal_domestico(_508)`
  - Call: `gato(_508)`
  - Exit: `gato('Michi')`
  - Call: `not(animal_salvaje('Michi'))`
  - Exit: `not('82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Michi'))`
  - Exit: `animal_domestico('Michi')`
- X = 'Michi'**
  - Redo: `gato(_508)`
  - Exit: `gato('Kitty')`
  - Call: `not(animal_salvaje('Kitty'))`
  - Exit: `not('82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Kitty'))`

The bottom of the right pane shows the query `?- trace, (animal_domestico(X)).` and buttons for `Examples`, `History`, `Solutions`, `table results`, and `Run!`.

# Uso de trace



The screenshot shows the SWISH Prolog IDE interface. The left pane contains a Prolog program with the following code:

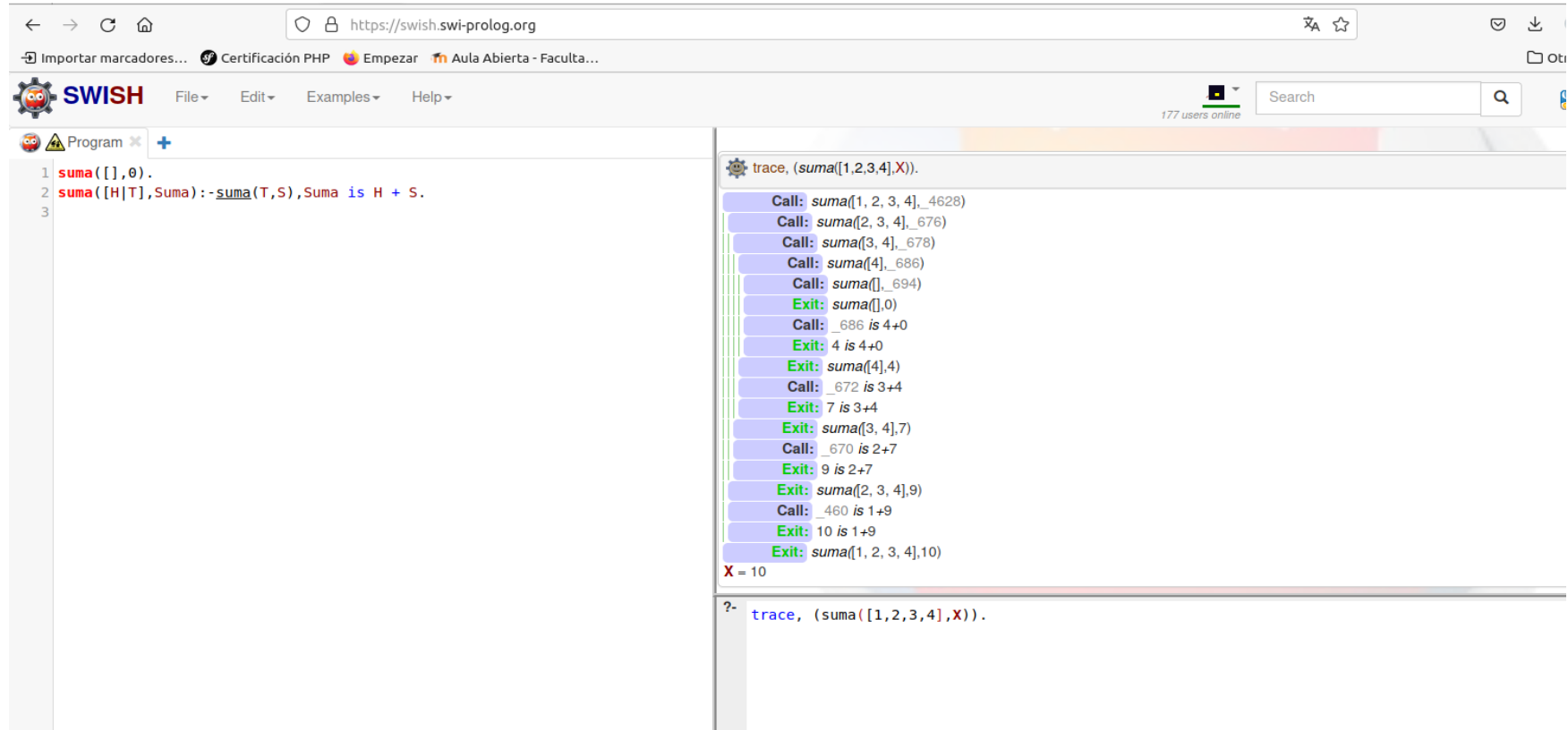
```
1 % hechos
2 perro('Canela').
3 perro('Melchor').
4 perro('Dulce').
5 perro('Sol').
6 gato('Michi').
7 gato('Kitty').
8 lince('lince1').
9 lince('lince2').
10 leon('leon1').
11 leon('leon2').
12 % reglas
13 /*uso de la negación en lugar del OR*/
14 animal_salvaje(A):-lince(A).
15 animal_salvaje(A):-leon(A).
16 animal_domestico(A):-perro(A),not(animal_salvaje(A)).
17 animal_domestico(A):-gato(A),not(animal_salvaje(A)).
18
19
```

The right pane displays the execution trace for the query `trace, (animal_domestico(X)).`. The trace shows the following steps:

- X = 'Melchor'**
  - Redo: `perro(_508)`
  - Exit: `perro('Dulce')`
  - Call: `not(animal_salvaje('Dulce'))`
  - Exit: `not('82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Dulce'))`
  - Exit: `animal_domestico('Dulce')`
- X = 'Dulce'**
  - Redo: `perro(_508)`
  - Exit: `perro('Sol')`
  - Call: `not(animal_salvaje('Sol'))`
  - Exit: `not('82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Sol'))`
  - Exit: `animal_domestico('Sol')`
- X = 'Sol'**
  - Redo: `animal_domestico(_508)`
  - Call: `gato(_508)`
  - Exit: `gato('Michi')`
  - Call: `not(animal_salvaje('Michi'))`
  - Exit: `not('82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Michi'))`
  - Exit: `animal_domestico('Michi')`
- X = 'Michi'**
  - Redo: `gato(_508)`
  - Exit: `gato('Kitty')`
  - Call: `not(animal_salvaje('Kitty'))`
  - Exit: `not('82d03690-092b-426c-941e-335d3e8a60f1':animal_salvaje('Kitty'))`
  - Exit: `animal_domestico('Kitty')`
- X = 'Kitty'**

The bottom of the right pane shows the query `?- trace, (animal_domestico(X)).` and buttons for `Examples`, `History`, `Solutions`, `table results`, and `Run!`.

# Uso de trace



The screenshot shows the SWISH Prolog IDE interface. The browser address bar displays `https://swish.swi-prolog.org`. The SWISH logo and navigation menu (File, Edit, Examples, Help) are at the top. A search bar and a status indicator "177 users online" are also present.

The main editor area contains a Prolog program:

```
1 suma([],0).  
2 suma([H|T],Suma):-suma(T,S),Suma is H + S.  
3
```

The right-hand pane shows the execution trace for the query `trace, (suma([1,2,3,4],X)).`. The trace consists of a series of "Call" and "Exit" events, each with a unique identifier and the current state of the stack:

- Call: `suma([1, 2, 3, 4],_4628)`
- Call: `suma([2, 3, 4],_676)`
- Call: `suma([3, 4],_678)`
- Call: `suma([4],_686)`
- Call: `suma([],_694)`
- Exit: `suma([],0)`
- Call: `_686 is 4+0`
- Exit: `4 is 4+0`
- Exit: `suma([4],4)`
- Call: `_672 is 3+4`
- Exit: `7 is 3+4`
- Exit: `suma([3, 4],7)`
- Call: `_670 is 2+7`
- Exit: `9 is 2+7`
- Exit: `suma([2, 3, 4],9)`
- Call: `_460 is 1+9`
- Exit: `10 is 1+9`
- Exit: `suma([1, 2, 3, 4],10)`

Below the trace, the variable `X` is shown with its value: `X = 10`.

The bottom of the right pane shows the query being executed: `?- trace, (suma([1,2,3,4],X)).`

# Uso de trace. swi-prolog desde PC

```
reydejuda@pegasus:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- consult('/home/reydejuda/prolog/animales.pl').
true.

?- trace (animal_domestico(X)).
ERROR: Syntax error: Operator expected
ERROR: trace
ERROR: ** here **
ERROR: (animal_domestico(X)) .
?- apropos(trace).
% SWI trace/0           Start the tracer.
% LIB trace/1          Equivalent to trace(Pred, +all).
% LIB trace/2          Put a trace point on all predicates satisfying the predicate specification Pred.
% LIB gtrace/0         Utility defined as guitracetrace.
% SWI notrace/0        Stop the tracer.
% SWI notrace/1        Call Goal, but suspend the debugger while Goal is executing.
% LIB chr_trace/0      Activate the CHR tracer.
% LIB guitracetrace/0  This predicate installs the above-mentioned hooks that redirect tracing to the window-based environment.
% LIB chr_notrace/0    Deactivate the CHR tracer.
% LIB noguitracetrace/0 Disable the hooks installed by guitracetrace/0, reverting to normal text console-based tracing.
% C 'PL_backtrace'()   Dump a Prolog backtrace to the user error stream.
% C 'PL_backtrace string'() As PL backtrace(), but returns the stack as a string.
% SWI catch_with_backtrace/3 As catch/3, but if library(library(prolog_stack)) is loaded and an exception of the shape error(Format, Context) is raised Context is extended with a bac ..
% LIB prolog_trace_interception/4 Dynamic predicate, normally not defined.
% LIB gdebug/0         Utility defined as guitracetrace,debug.
% SWI tracing/0        True if the tracer is currently switched on.
% LIB gspy/1           Utility defined as guitracetrace,spy(Predicate).
% LIB tdebug/1         Prepare Threadid for debugging using the graphical tracer.
% LIB chr_leash/1      Define the set of CHR ports on which the CHR tracer asks for user intervention (i.e.
% SEC tracehook        Intercepting the Tracer
true.

?- trace.
true.
```



## Uso de trace. swi-prolog desde PC

```
?- consult('/home/reydejuda/prolog/animales.pl').
true.

?- trace.
true.

[trace] ?- animal_domestico('Dulce').
  Call: (10) animal_domestico('Dulce') ? creep
  Call: (11) perro('Dulce') ? creep
  Exit: (11) perro('Dulce') ? creep
^ Call: (11) not(animal_salvaje('Dulce')) ? creep
  Call: (12) animal_salvaje('Dulce') ? creep
  Call: (13) lince('Dulce') ? creep
  Fail: (13) lince('Dulce') ? creep
  Redo: (12) animal_salvaje('Dulce') ? creep
  Call: (13) leon('Dulce') ? creep
  Fail: (13) leon('Dulce') ? creep
  Fail: (12) animal_salvaje('Dulce') ? creep
^ Exit: (11) not(user:animal_salvaje('Dulce')) ? creep
  Exit: (10) animal_domestico('Dulce') ? creep
true .

[trace] ?- ☐
```

## Uso de trace. swi-prolog desde PC

```
[trace] ?- animal_salvaje(S).  
  Call: (10) animal_salvaje(_16704) ? creep  
  Call: (11) lince(_16704) ? creep  
  Exit: (11) lince(lince1) ? creep  
  Exit: (10) animal_salvaje(lince1) ? creep  
S = lince1 ;  
  Redo: (11) lince(_16704) ? creep  
  Exit: (11) lince(lince2) ? creep  
  Exit: (10) animal_salvaje(lince2) ? creep  
S = lince2 ;  
  Redo: (10) animal_salvaje(_16704) ? creep  
  Call: (11) leon(_16704) ? creep  
  Exit: (11) leon(leon1) ? creep  
  Exit: (10) animal_salvaje(leon1) ? creep  
S = leon1 ;  
  Redo: (11) leon(_16704) ? creep  
  Exit: (11) leon(leon2) ? creep  
  Exit: (10) animal_salvaje(leon2) ? creep  
S = leon2.  
[trace] ?- █
```

## Uso de trace. swi-prolog desde PC

```
?- consult('/home/rejdejudas/prolog/sumaLista.pl').
true.

?- trace.
true.

[trace] ?- suma([1,2,3,4],SumaLista).
  Call: (10) suma([1, 2, 3, 4], _8816) ? creep
  Call: (11) suma([2, 3, 4], _9280) ? creep
  Call: (12) suma([3, 4], _9324) ? creep
  Call: (13) suma([4], _9368) ? creep
  Call: (14) suma([], _9412) ? creep
  Exit: (14) suma([], 0) ? creep
  Call: (14) _9504 is 4+0 ? creep
  Exit: (14) 4 is 4+0 ? creep
  Exit: (13) suma([4], 4) ? creep
  Call: (13) _9642 is 3+4 ? creep
  Exit: (13) 7 is 3+4 ? creep
  Exit: (12) suma([3, 4], 7) ? creep
  Call: (12) _9780 is 2+7 ? creep
  Exit: (12) 9 is 2+7 ? creep
  Exit: (11) suma([2, 3, 4], 9) ? creep
  Call: (11) _8816 is 1+9 ? creep
  Exit: (11) 10 is 1+9 ? creep
  Exit: (10) suma([1, 2, 3, 4], 10) ? creep
SumaLista = 10.

[trace] ?- □
```