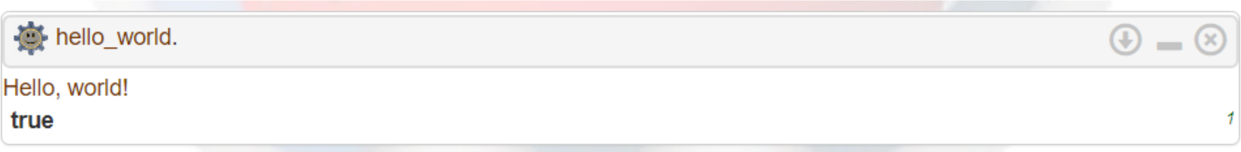
Program 1: Hello World

Code:

```
hello_world :-
    write('Hello, world!').
```

Output:



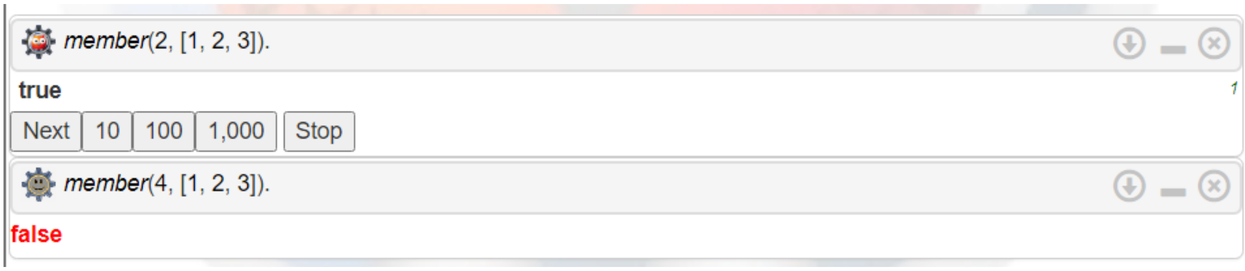
```
hello_world.
Hello, world!
true
```

Program 2: Check whether a number is a member of a list

Code:

```
member(X, [X|_]).
member(X, [_|T]) :- member(X, T).
```

Output:



```
member(2, [1, 2, 3]).
true
```

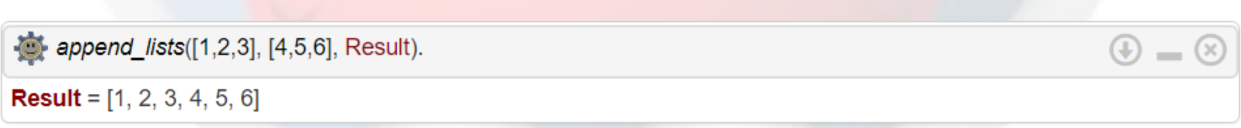
```
member(4, [1, 2, 3]).
false
```

Program 3: Program to append two lists

Code:

```
append_lists([], L, L).
append_lists([H|T], L, [H|Result]) :-
    append_lists(T, L, Result).
```

Output:



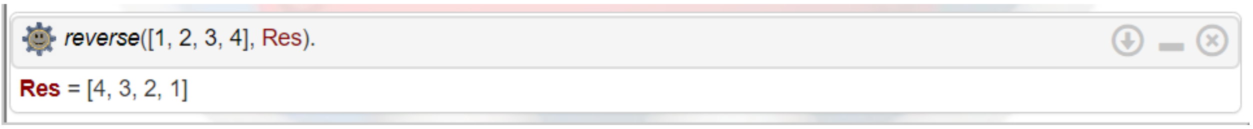
```
append_lists([1,2,3], [4,5,6], Result).
Result = [1, 2, 3, 4, 5, 6]
```

Program 4: Reverse a list

Code:

```
reverse([], []).
reverse([H|T], Res) :- reverse(T, TRes), append(TRes, [H], Res).
```

Output:



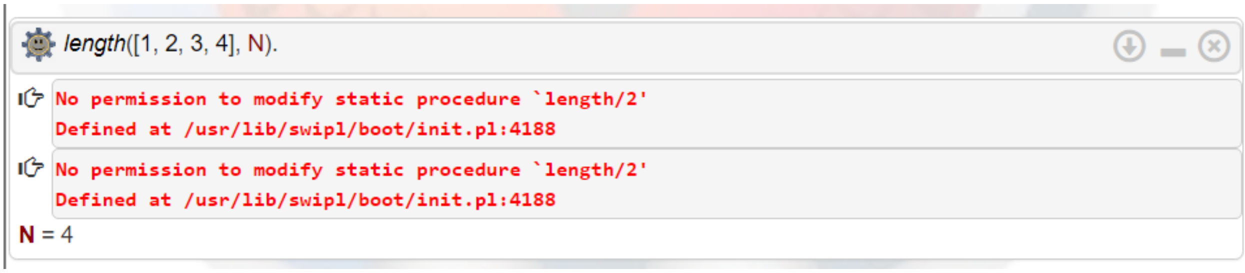
```
reverse([1, 2, 3, 4], Res).
Res = [4, 3, 2, 1]
```

Program 5: Find length of list

Code:

```
length([], 0).
length([_|T], N) :- length(T, N1), N is N1 + 1.
```

Output:



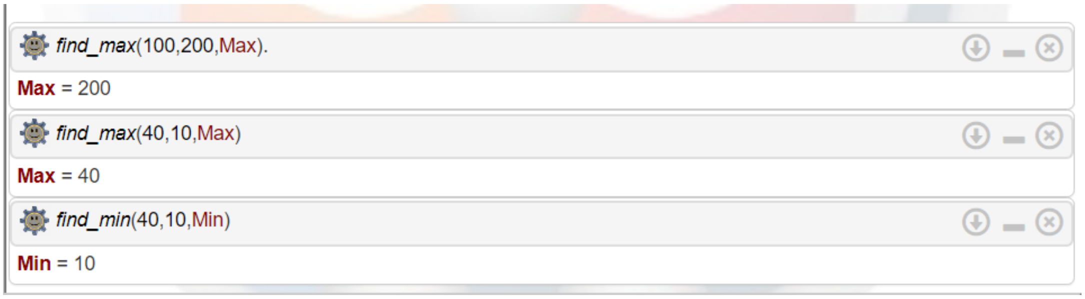
```
length([1, 2, 3, 4], N).
No permission to modify static procedure `length/2'
Defined at /usr/lib/swipl/boot/init.pl:4188
No permission to modify static procedure `length/2'
Defined at /usr/lib/swipl/boot/init.pl:4188
N = 4
```

Program 6: Minimum and Maximum

Code:

```
find_max(X, Y, X) :- X >= Y, !.
find_max(X, Y, Y) :- X < Y.

find_min(X, Y, X) :- X =< Y, !.
find_min(X, Y, Y) :- X > Y.
```

Output:



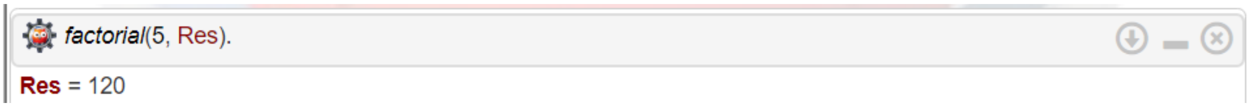
```
find_max(100,200,Max).
Max = 200
find_max(40,10,Max)
Max = 40
find_min(40,10,Min)
Min = 10
```

Program 7: Factorial

Code:

```
factorial(0, 1).
factorial(N, Res) :- N > 0, N1 is N - 1, factorial(N1, Res1), Res is N * Res1.
```

Output:



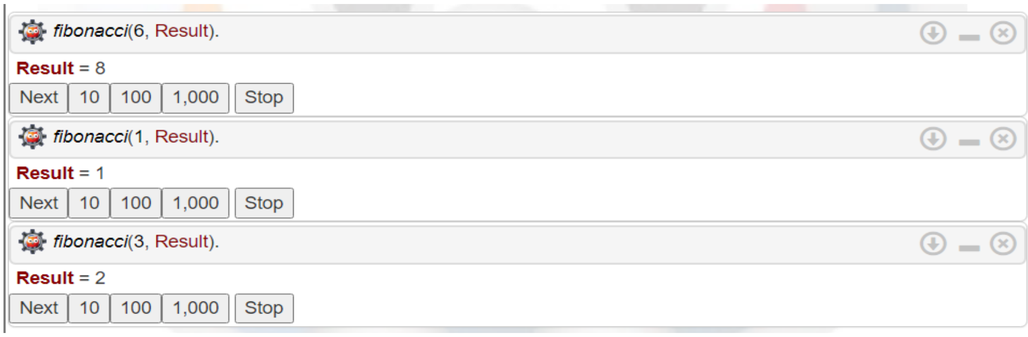
```
factorial(5, Res).
Res = 120
```

Program 8: Program to find nth number of fibonacci series

Code:

```
fibonacci(0, 0).
fibonacci(1, 1).
fibonacci(N, Result) :-
    N > 1,
    N1 is N - 1,
    N2 is N - 2,
    fibonacci(N1, Result1),
    fibonacci(N2, Result2),
    Result is Result1 + Result2.
```

Output:



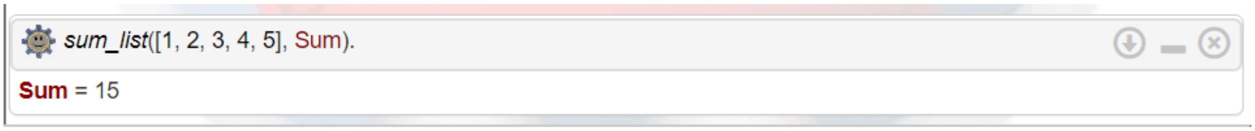
```
fibonacci(6, Result).
Result = 8
fibonacci(1, Result).
Result = 1
fibonacci(3, Result).
Result = 2
```

Program 9: Program to find sum of elements of a list

Code:

```
sum_list([], 0).
sum_list([H|T], Sum) :-
    sum_list(T, Rest),
    Sum is H + Rest.
```

Output:



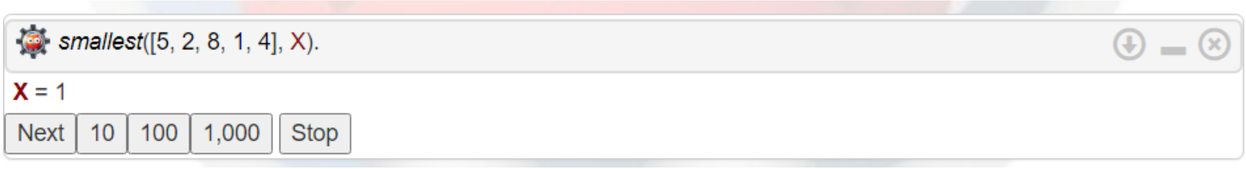
```
sum_list([1, 2, 3, 4, 5], Sum).
Sum = 15
```

Program 10: Program to find the smallest element of the list

Code:

```
smallest([X], X).
smallest([H|T], X) :-
    smallest(T, Y),
    (H < Y -> X = H ; X = Y).
```

Output:



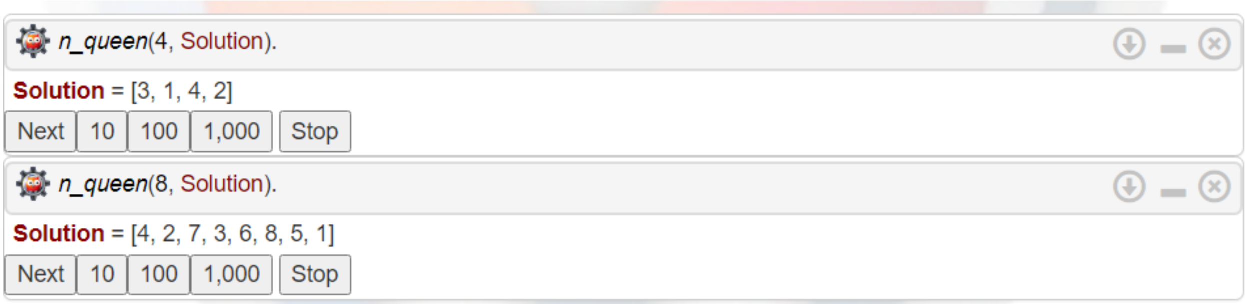
```
smallest([5, 2, 8, 1, 4], X).
X = 1
```

N queen

Code:

```
n_queen(N, Solution) :-
        %create a list of N dummy variabiles
        length(Solution, N),

        queen(Solution, N). %search for a configuration of N queens


%returns a list of integer from K to N included es up2N(1,3,X) X = [1,2,3]
up2N(N,N,[N]) :-!.
up2N(K,N,[K|Tail]) :- K < N, K1 is K+1, up2N(K1, N, Tail).


queen([],_). %No queens is a solution for any N queens problem. All queens are in a safe
position.

queen([Q|Qlist],N) :-

        queen(Qlist, N), %first we solve the subproblem

        %we then generate all possible positions for queen Q
        up2N(1,N,Candidate_positions_for_queenQ),

        %we pick one of such position
        member(Q, Candidate_positions_for_queenQ),

        %we check whether the queen Q is safe
        check_solution(Q,Qlist, 1).



check_solution(_,[], _).

check_solution(Q,[Q1|Qlist],Xdist) :-
        Q =\= Q1, %not on the same row
        Test is abs(Q1-Q),
        Test =\= Xdist, %diagonal distance
        Xdist1 is Xdist + 1,
        check_solution(Q,Qlist,Xdist1).
```

Output:



```
n_queen(4, Solution).
Solution = [3, 1, 4, 2]
```

```
n_queen(8, Solution).
Solution = [4, 2, 7, 3, 6, 8, 5, 1]
```