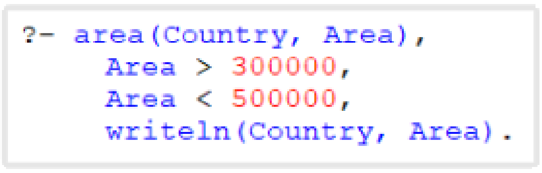
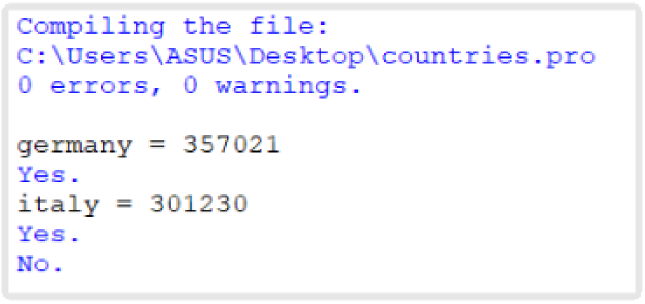
Nakrit Kreechaichana 6510110212

1.1. Find a country whose area is

- Greater than 300,000 km^2 and
- Less than 500,000 km^2 .



```
?- area(Country, Area),
     Area > 300000,
     Area < 500000,
     writeln(Country, Area).
```

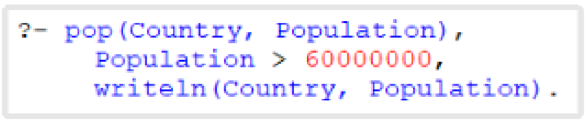


```
Compiling the file:
C:\Users\ASUS\Desktop\countries.pro
0 errors, 0 warnings.

germany = 357021
Yes.
italy = 301230
Yes.
No.
```

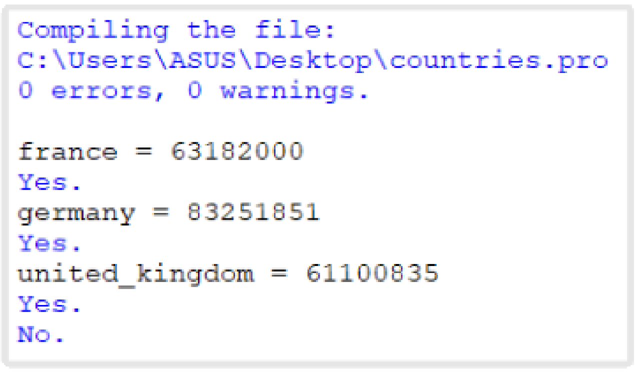
EXPLAIN: This query retrieves each country that has **300,000 km² < Areas < 500,000 km²**, and then prints that country along with area.

1.2. Find all countries with

- more than 60 million people.



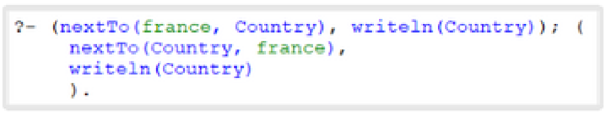
```
?- pop(Country, Population),
    Population > 60000000,
    writeln(Country, Population).
```



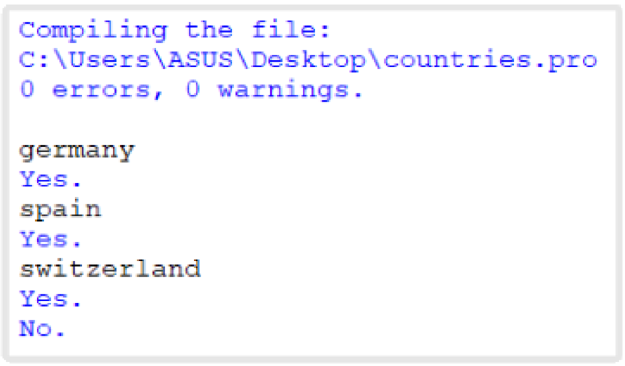
```
Compiling the file:
C:\Users\ASUS\Desktop\countries.pro
0 errors, 0 warnings.

france = 63182000
Yes.
germany = 83251851
Yes.
united_kingdom = 61100835
Yes.
No.
```

EXPLAIN: This query retrieves each country that has populations greater than 60 million, and then prints country along with population.

1.3. Find all the countries **next to** France. Hint: you should use ";" (or).



```
?- (nextTo(france, Country), writeln(Country)); (
     nextTo(Country, france),
     writeln(Country)
     ).
```

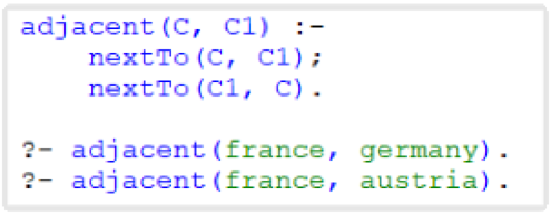


```
Compiling the file:
C:\Users\ASUS\Desktop\countries.pro
0 errors, 0 warnings.

germany
Yes.
spain
Yes.
switzerland
Yes.
No.
```

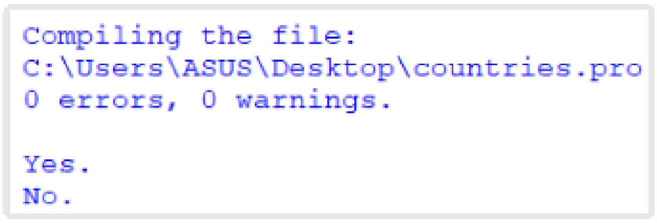
EXPLAIN: This query retrieves countries adjacent to France or countries that France is adjacent to, and then prints each country.

Nakrit Kreechaichana 6510110212

2.1. adjacent(C,C1): country C is adjacent to C1.
Note: this is NOT the same as the nextTo/2 predicate



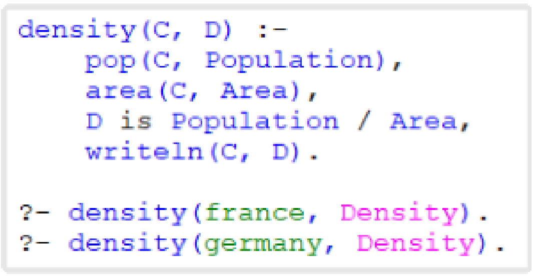
```
adjacent(C, C1) :-
    nextTo(C, C1);
    nextTo(C1, C).

?- adjacent(france, germany).
?- adjacent(france, austria).
```



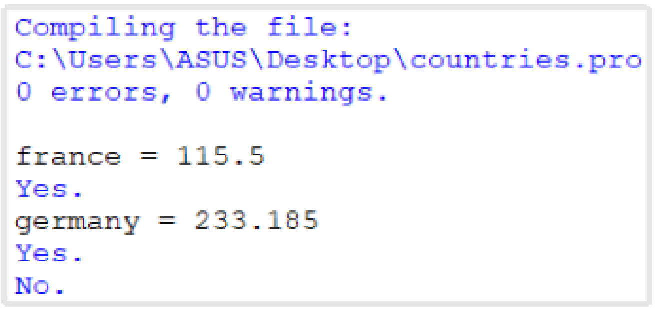
```
Compiling the file:
C:\Users\ASUS\Desktop\countries.pro
0 errors, 0 warnings.

Yes.
No.
```

<u>EXPLAIN:</u> **"adjacent/2"** predicate checks if two countries are adjacent based on the **nextTo** predicate,
returning "Yes" if they are adjacent and "No" if they are not.

2.2. density(C, D): country C has a density D.
Density is population / country area.



```
density(C, D) :-
    pop(C, Population),
    area(C, Area),
    D is Population / Area,
    writeln(C, D).

?- density(france, Density).
?- density(germany, Density).
```



```
Compiling the file:
C:\Users\ASUS\Desktop\countries.pro
0 errors, 0 warnings.

france = 115.5
Yes.
germany = 233.185
Yes.
No.
```

<u>EXPLAIN:</u> **"density/2"** calculates the **D** (density) of a **C** (country) by dividing
its **population** by its **area.**

Nakrit Kreechaichana 6510110212

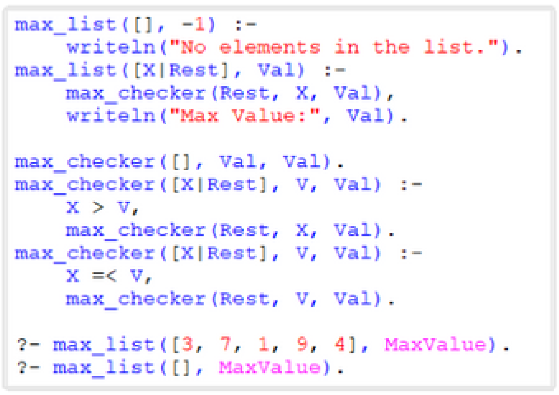
3.

```
foo([], -1).

foo([X|Rest], Val) :-
    foo1(Rest, X, Val).


foo1([], Val, Val).
foo1([X|Rest], V, Val) :-
    X > V,
    foo1(Rest, X, Val).
foo1([X|Rest], V, Val) :-
    X =< V,
    foo1(Rest, V, Val).
```

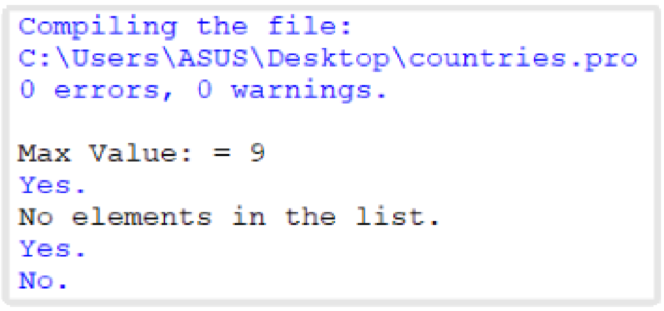
```
max_list([], -1) :-
    writeln("No elements in the list.").
max_list([X|Rest], Val) :-
    max_checker(Rest, X, Val),
    writeln("Max Value:", Val).

max_checker([], Val, Val).
max_checker([X|Rest], V, Val) :-
    X > V,
    max_checker(Rest, X, Val).
max_checker([X|Rest], V, Val) :-
    X =< V,
    max_checker(Rest, V, Val).
```

queries

```
?- max_list([3, 7, 1, 9, 4], MaxValue).      # Integer list
?- max_list([], MaxValue).                   #Empty list
```



```
max_list([], -1) :-
    writeln("No elements in the list.").
max_list([X|Rest], Val) :-
    max_checker(Rest, X, Val),
    writeln("Max Value:", Val).

max_checker([], Val, Val).
max_checker([X|Rest], V, Val) :-
    X > V,
    max_checker(Rest, X, Val).
max_checker([X|Rest], V, Val) :-
    X =< V,
    max_checker(Rest, V, Val).

?- max_list([3, 7, 1, 9, 4], MaxValue).
?- max_list([], MaxValue).
```



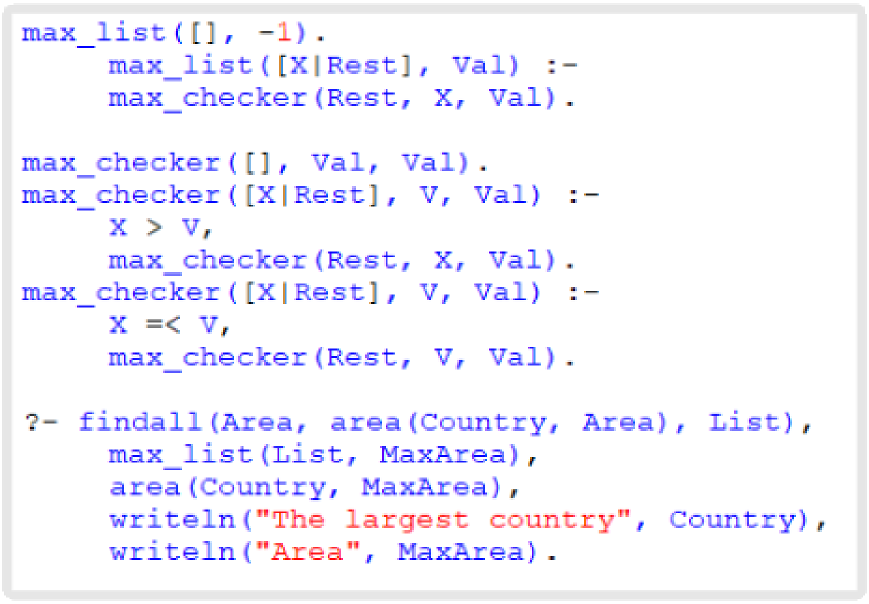
```
Compiling the file:
C:\Users\ASUS\Desktop\countries.pro
0 errors, 0 warnings.

Max Value: = 9
Yes.
No elements in the list.
Yes.
No.
```

EXPLAIN: **"max_list/2"** finds the maximum value in a list and binds it to Val.

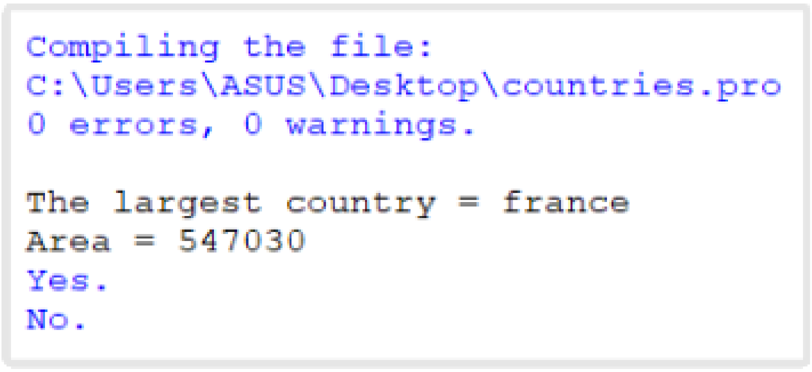
Nakrit Kreechaichana 6510110212

4.

Write a query that finds the country with the largest area.
Hint: use **countries.pro, findall/3**, and the **foo** predicates from **Ex. 3**



```
max_list([], -1).
     max_list([X|Rest], Val) :-
     max_checker(Rest, X, Val).

max_checker([], Val, Val).
max_checker([X|Rest], V, Val) :-
     X > V,
     max_checker(Rest, X, Val).
max_checker([X|Rest], V, Val) :-
     X =< V,
     max_checker(Rest, V, Val).

?- findall(Area, area(Country, Area), List),
     max_list(List, MaxArea),
     area(Country, MaxArea),
     writeln("The largest country", Country),
     writeln("Area", MaxArea).
```



```
Compiling the file:
C:\Users\ASUS\Desktop\countries.pro
0 errors, 0 warnings.

The largest country = france
Area = 547030
Yes.
No.
```

EXPLAIN: This query finds the maximum area among all countries,
using "**findall/3**" and "**max_list**" to retrieves the country with largest area.