F1 Drivers Guessing Game

Alfonso Díaz-Infante
A01350809

Table of Contents

Problem

Formula one is a multinational sport that is considered to be the pinnacle of motorsports. It is a sport that has drivers from many different nationalities and is followed by millions of people. Mexico is a country that has a long-standing story in Formula One races that started back in the 1950's.

Mexico has had a total of 6 Formula One drivers in the story of the sport, including Sergio Pérez, current driver and Ricardo Rodriguez, Scuderia Ferrari's youngest ever driver.

Having such a rich history, it is a shame that many people are not that involved in what is happening in a sport so competitive and with so much tradition.

Solution

This guessing game aims to do just that, create awareness of the current drivers and get people interested in this amazing sport by giving the user fun facts about the racing drivers and the chance to learn more about the current grid.

Why PROLOG?

Prolog is a logical programming language that is suited perfectly for the type of interactive experience I wanted to develop.
Prolog is very well suited for programs that involve non-numeric computation. Great examples of non-numeric computational problems would be puzzles like sudoku, where one has to check against a big number of possibilities which one is the correct answer. Or in my case, check which rules fit with which facts.
Backtracking is one of the reasons why PROLOG is so good at solving these kinds of problems, if it has a good database and you ask a question, PROLOG can run what it knows to see what it needs to know, making a decision and going back if the decision taken is not what it is looking for. The knowledge database is also a reason I chose PROLOG. PROLOG runs all the facts and cross checks them with the database and backtracks until it receives a definitive answer.

How to run the program on Windows

1.- install SWI PROLOG from the following page:

https://www.swi-prolog.org/download/stable

2.- Run SWI PROLOG and create a new file, in the new file paste the source code provided at the bottom of the page and go to the top of the screen, select compile -> make

3.- Once compiled, return to the SWI PROLOG application and select from the top of the screen file -> consult and select the newly created file.

4.- Once running, the program accepts y or yes as an affirmative answer, anything else will be taken as a no. Either affirmative or negative answers must end with a period '.'

5.- Once the program is finished, to start it again type:

`undo`. And press enter

This is done to undo all previous assertions that were made in the program and start again.

How to run the program on OSX

1.- install SWI PROLOG by opening a new terminal window and type:

```
brew install swi-prolog
```

2.- Open Swipl by typing:

```
Swipl.
```

3.- Type in the path where the program is stored, in my case it would look something like this:

```
['/Users/fonsdiaz/Desktop/School/Lenguajes de Progra/PROLOG/Proyecto PROLOG.pl'].
```

```
FonsPro:~ fonsdiaz$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.1.13)
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).

?- ['/Users/fonsdiaz/Desktop/School/Lenguajes_de_Progra/PROLOG/Proyecto_PROLOG.pl'].
true.

?-
```

Once you enter this command, you must get "*true*" in return. If you get true it means that SWI PROLOG has found your file.

4.- Start the program by typing:

```
start.
```

```
FonsPro:~ fonsdiaz$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.1.13)
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).

?- ['/Users/fonsdiaz/Desktop/School/Lenguajes_de_Progra/PROLOG/Proyecto_PROLOG.pl'].
true.

?- start.
Think of an F1 driver from the current 2019 season.
Please answer yes or no to the following questions so that I can tell you what racing driver you are thinking of:

Is the driver a_podium_winner?
```

5.- The program accepts y or yes as an affirmative answer, anything else will be taken as a no. Either affirmative or negative answers must end with a period '.'

```
FonsPro:~ fonsdiaz$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.1.13)
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).

?- ['/Users/fonsdiaz/Desktop/School/Lenguajes_de_Progra/PROLOG/Proyecto_PROLOG.pl'].
true.

?- start.
Think of an F1 driver from the current 2019 season.
Please answer yes or no to the following questions so that I can tell you what racing driver you are thinking of:

Is the driver a_podium_winner?y.

Is the driver a_race_winner?|: ye.

Is the driver a_world_champion?|: yes.

Is the driver a_multiple_world_champion?|:
```

6.- Once the program is finished, to start it again type:

`undo`. And press enter

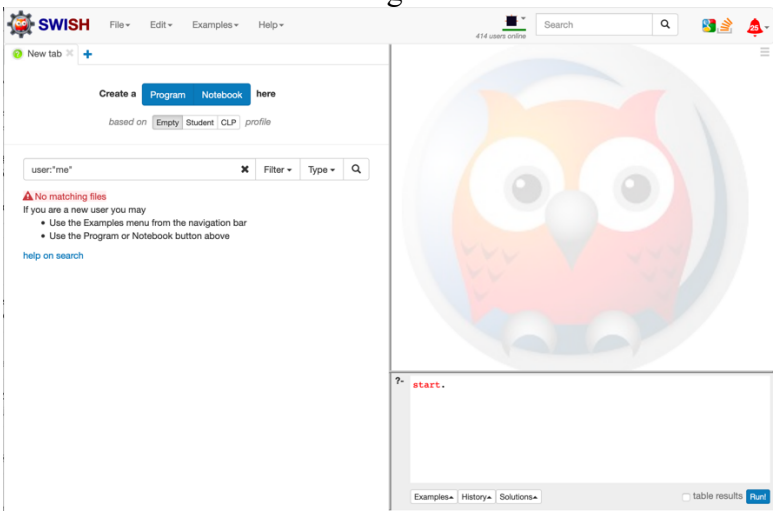
This is done to undo all previous assertions that were made in the program and start again.

How to run the program online

1.- Enter the following website:

https://swish.swi-prolog.org/

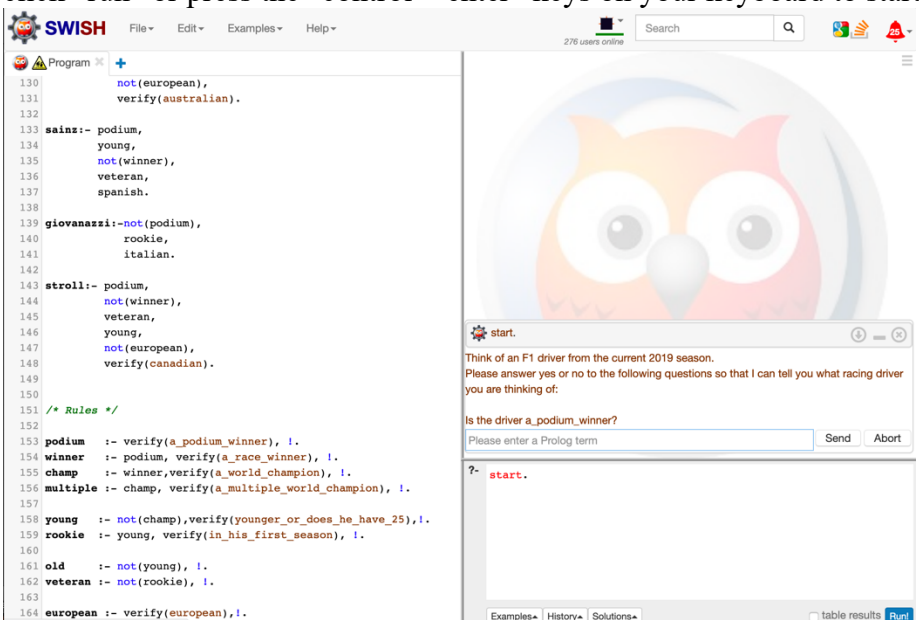
You should enter the following website:



Select the “Program” button to start writing.

2.- Copy and paste the downloaded code into the program text box.

3. Start the program by typing “start.” In the smaller text box on the bottom right of the page and click “run” or press the “control + enter” keys on your keyboard to start the program.



3. Type y or yes to introduce an affirmation statement into the small textbox with the placeholder text that reads “Please enter a Prolog term” and press enter.

How the program works

In this section I will do my best to explain the source code that is located at the end of this document.

The program starts by printing out some text lines to salute the user.
After printing some lines it reaches a fact with a Variable called Driver as an argument.

```
start :-
    write('Please answer yes or no to the following questions so that I can tell you what racing
driver you are thinking of:'),
    guess(Driver),
    write('The driver is '),
    write(Driver).
```

To continue, PROLOG must know what the variable Driver is so it checks all the possible things that Driver can be. Since driver can be one of many names, but can only be one of those, it must check which one it is.

```
guess(vettel)    :- vettel,!.
guess(lewis)     :- lewis, !.
guess(valtteri)  :- valtteri, !.
.
.
.
guess(sainz)     :- sainz, !.
guess(stroll)    :- stroll, !.
guess(giovanazzi):- giovanazzi, !.
```

To be one of those drivers, the driver must fulfill all of the rules that a driver has been set so that PROLOG can be 100% sure what Driver is.

```
grosjean:- podium,
          old,
          not(winner),
           veteran,
          french.

giovanazzi:-    not(podium),
                rookie,
                italian.

stroll:- podium,
          not(winner),
         veteran,
         young,
          not(european),
        verify(canadian).
```

Now PROLOG has a new problem, how can it know if Grosjean is young, a podium winner, a veteran and German? It must check with in the database for the rules that make podium, young, veteran and french all true.

```
podium   :- verify(a_podium_winner), !.
winner   :- podium, verify(a_race_winner), !.
champ    :- winner, verify(a_world_champion), !.

european :- verify(european),!.
french   :- european, verify(french), !.
spanish  :- european, verify(spanish),!.
```

These are some of the rules that must be fulfilled, and they are (hopefully) very simple to understand.
A driver can only European, if he is European then he can be Spanish, Finnish, French and so on, but not the other way around!
Following that same logic, a podium winner can be a race winner, and a race winner can also be a champion, a world champion must have achieved a race win and a podium. He cannot be a champion if he never even finished on the podium!
To change the arguments (french) to true, we will send it to more rules to be unified.

```
verify(Q) :-
        (yes(Q) -> true ;
        (no(Q) -> fail ;
        ask(Q))).
```

What verify does is receive in this case french as Q and change it to true or false, but how does it do all of that? More Rules! We send Q (french) to ask.

```
ask(Question) :-
            write('Is the driver '),
            write(Question),
            write('?'),
            read(Response),
            ( (Response == yes ; Response == y ; Response == ye) ->
          assert(yes(Question)) ;
             assert(no(Question)), fail).
```

Ask receives the question (french) as Question and prints on screen :

"Is the driver " + Question + "?"

so it will say:

"Is the driver French?"

Then, PROLOG will wait for a user input and take it in as Response.

If Response is equal to “yes”, “y” or in case of a small mistake “ye”, then PROLOG will introduce to the database (yes(Question)) or in our case (yes(french)). Anything other than ‘y’,’ye’ or ‘yes’ will be taken as a negative answer.

And this is where the fun begins

Backtracking!

We start going back and unifying the now true answer

```
verify(french) :-
        (yes(french) -> true ;
        (no(french) -> fail ;
        ask(french))).
```

It now turns true so…

```
french   :- european, verify(french). %is now true as well!
```

So….

```
grosjean:- podium,
        old,
        not(winner),
         veteran,
        french. %This is now true!
```

If all else is true, it is safe to assume that grosjean is our driver.

PROLOG does this same process with every one of the rules until one driver fulfills all of the rules. That driver name then gets replaced in the Driver variable and printed on screen

```
write('The driver is '),
    write(Driver).
```

In our case

```
write('The driver is '),

    write(Grosjean).
```

To make this program run the way I intended, I had to research some PROLOG functions: retractall and dynamic, both of which are used in the program.

```
undo :- retractall(yes(_)),fail. %retractall makes all facts or clauses in the database for
which the head unifies with Head are removed.
```

undo :- retractall(no(_)),fail. %fail makes the predicate to always fail and continue on to the next one.
undo.

What retractall[1] does is that if eliminates any assertion made from the Head, in this case, the head is yes(_), the underline means that we don't care what yes is unified with, we want to eliminate all of them assertions, and fail[2] means that we don't want it to stop when it finds the first assertion and returns true.
Find the referenced code link at the bottom of the document [4]

:- dynamic yes/1,no/1.

Dynamic[3] informs the interpreter that the definition of the predicates may change during execution, in our case, yes and no.

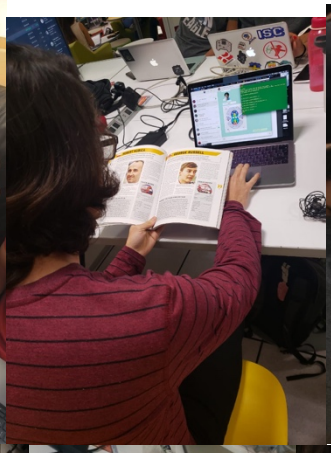
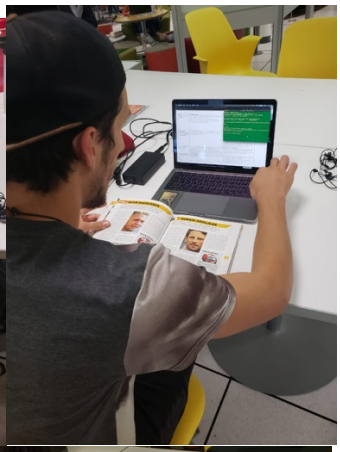
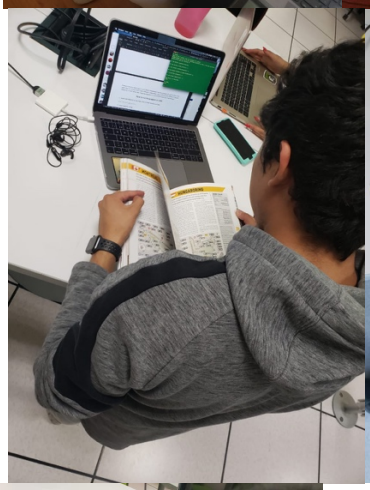
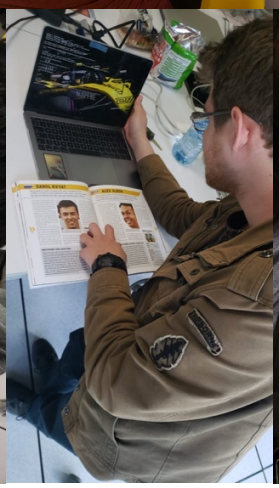
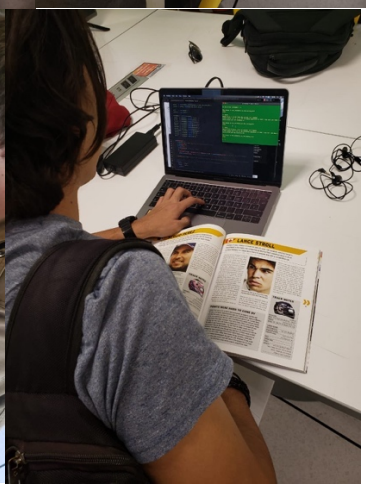
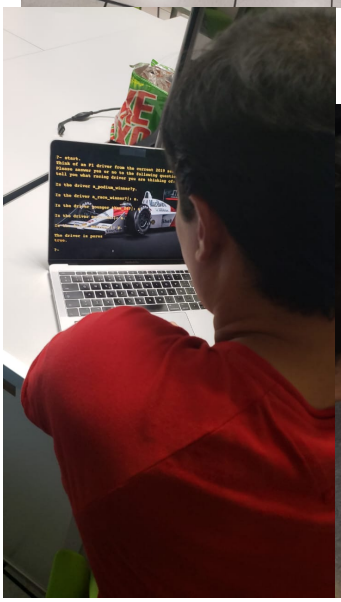
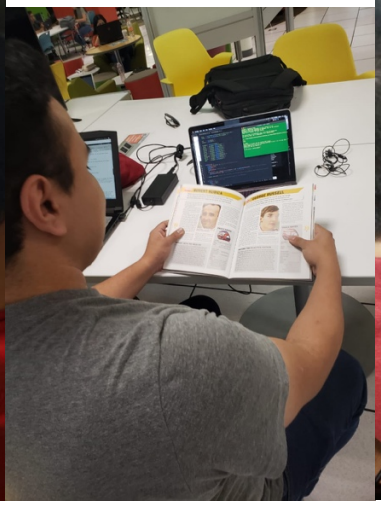
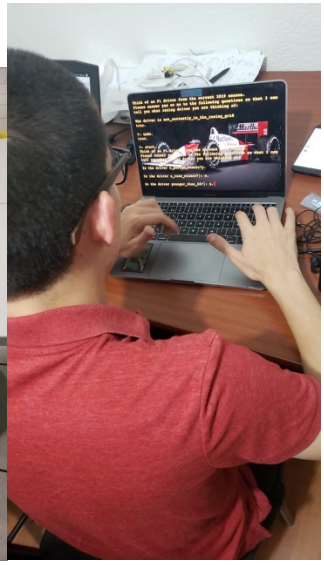
Program Logical Mind Map

Here is a mind map for all the possible paths the program can take depending on your decision.

The program will always start with the same question: Is the driver a podium winner, or in other words, has the driver ever finished top 3 in any of his races.

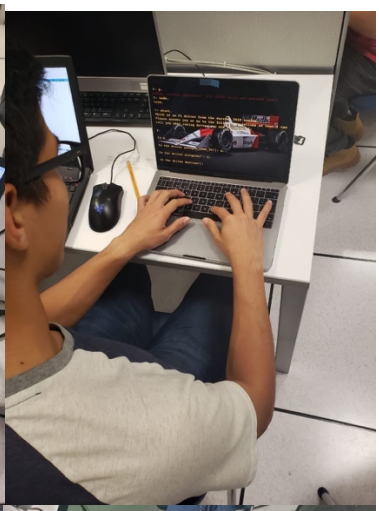
Tests

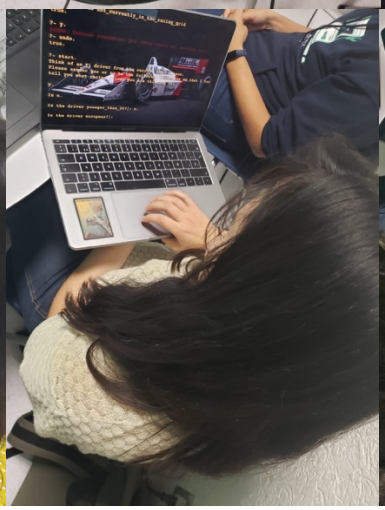
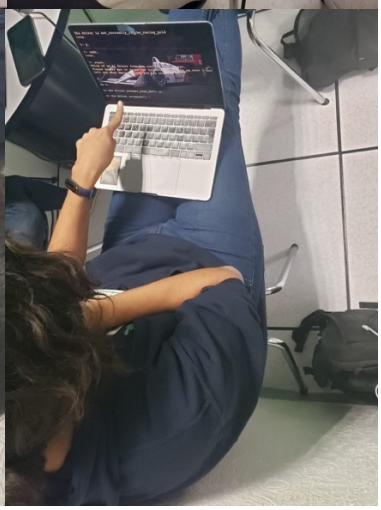
Different people tried the game out during the development process, here are the pictures:







GitHub Repository

https://github.com/Fonsdiaz/Prolog_guessing_game/upload/master

Source Code

```
/*
 * F1 drivers guessing game. Think of a driver from the current 2019 season and
 * Copyright (C) 2019 Alfonso Diaz Infante
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <https://www.gnu.org/licenses/>.
 *
```

```
 * Alfonso Diaz-Infante
 * A01350809
 * start the game with ?- start. statement
*/

start :- % This is where the game starts, these are a set of rules that must be completed
    write('Think of an F1 driver from the current 2019 season.'),
    nl, % New line
    write('Please answer yes or no to the following questions so that I can tell you what racing driver you are thinking of:'),
    nl, nl,
    guess(Driver), %Driver is the variable that will be replaced
    write('The driver is '),
    write(Driver),
    nl.

%guess will replace Driver with a name if the rules are met.
guess(sebastian_vettel)   :- vettel .
guess(lewis_hamilton)     :- lewis .
guess(valtteri_bottas)  :- valtteri .
guess(george_russell)   :- russell .
guess(kimi_raikkonnen):- raikkonnen .
guess(sergio_perez)     :- perez .
guess(pierre_gasly)     :- gasly .
guess(romain_grosjean)  :- grosjean .
guess(lando_norris)    :- norris .
guess(charles_leclerc)   :- leclerc .
guess(robert_kubica)    :- kubica .
guess(max_verstappen):- verstappen .
guess(alex_albon)     :- albon .
guess(daniil_kvyat)     :- kvyat .
guess(kevin_magnussen) :- magnussen .
guess(nico_hulkenberg):- hulkenberg .
guess(daniel_ricciardo) :- ricciardo .
guess(carlos_sainz_jr)     :- sainz .
guess(lance_stroll)    :- stroll .
guess(antonio_giovanazzi):- giovanazzi .

guess(not_currently_in_the_racing_grid).

/* Driver identification rules */
vettel :- multiple, % Driver can be vettel if he is a multiple world champion and he is German
                    german,
      write('Did you know: This driver holds the record for the youngest Pole Position.'),
      nl.

leclerc :-winner, % Driver can be vettel if he is a race winner,young, more than one season and he is monegasque
                    young,
                    veteran,
                    not(champ),
                    monegasque,
      write('Did you know: This driver holds the record for the second youngest Ferrarri driver, mexican driver Ricardo Rodriguez still holds that record ;).'),
      nl.

lewis :- multiple,
                                    british,
     write('Did you know: This driver is the second driver with the most championships ever, behind only Michael Schumacher.'),
     nl.

raikkonnen :- champ,
                                    veteran,
                                                old,
                                    finnish,
       write('Did you know: This driver is the oldest on the grid, he is 40 years old!.'),
       nl.

valtteri :-winner,
                                     not(champ),
                    old,
                    finnish,
                    veteran,
      write('Did you know: This driver holds the record for the unofficial Formula One record speed of 378 km/h.'),
      nl.

norris :- not(podium),
                   rookie,
       british,
                                    verify(mcLaren_driver),
     write('Did you know: This driver is the youngest driver on the grid? only 20 years old'),
     nl.

russell :-not(podium),
```

```
        rookie,
        british,
                verify(a_williams_driver).

perez:- podium,
      veteran,
                                not(winner),
       not(european),
                old,
                verify(mexican),
     write('Did you know: This driver holds the record for the mexican with most Podiums.'),
     nl.

gasly:- podium,
                                not(winner),
                young,
                veteran,
                french,
     write('Did you know: This driver started racing for Red Bull at the begginning of the season but got demoted to Toro Rosso.'),
     nl.

kvyat:- podium,
                                not(winner),
                young,
     veteran,
                russian,
     write('Did you know: This driver was fired from the Red Bull team two years ago, but his talent brought him back!.'),
     nl.

magnussen:-podium,
                                  not(winner),
                  old,
                  veteran,
                  danish,
       write('Did you know: This driver stood on the podium on his first race.'),
       nl.

grosjean:-podium,
                                  not(winner),
        veteran,
        french,
                old,
      write('Did you know: This driver starred in David Guettas music video for the song: "Dangerous".'),
      nl.

verstappen:- winner,
                                not(champ),
                  young,
                                dutch,
                  veteran,
         write('Did you know: This driver started racing in F1 when he was 17 years old!.'),
         nl.

kubica:- winner,
                old,
                                not(champ),
                polish,
     write('Did you know: This driver was going to race for Ferrari before he had a life threatening accident.'),
     nl,
     write('After 9 years of rehabilitation and dedication, he returned to F1.'),
     nl.

albon:- rookie,
                                not(podium),
                not(european),
                verify(thai),
      write('Did you know: This driver is the first Thai driver since the prince of Thailand last raced in 1954!.'),
      nl.

hulkenberg:- veteran,
          old,
          not(podium),
          german,
          write('Did you know: This driver has a pole position but has never been on the podium.'),
          nl.

ricciardo:-winner,
          old,
          not(champ),
          not(european),
          verify(australian).
```

```
sainz:- podium,
                young,
                                not(winner),
    veteran,
    spanish,
    write('Did you know: This driver is the son of two time rally world champion driver Carlos Sainz.'),
    nl.

giovanazzi:-not(podium),
        rookie,
        italian.

stroll:- podium,
                                not(winner),
                veteran,
    young,
    not(european),
                verify(canadian),
    write('Did you know: This drivers dad is also the team owner.'),
    nl.


/* Rules */
podium   :- verify(a_podium_winner) . % A driver can be a podium winner
winner   :- podium, verify(a_race_winner) . % A driver cannot be a race winner if he is not a podium winner
champ    :- winner,verify(a_world_champion) . % A driver cannot be a world champion if he is not a race winner
multiple :- champ, verify(a_multiple_world_champion) . % A driver cannot be a multiple wolrd champion if he has never been a world champion

young   :- not(champ),verify(younger_than_26) .
rookie  :- young,verify(in_his_first_season) .

old     :- not(young) .
veteran :- not(rookie) .

european :- verify(european) .
french   :- european, verify(french) .
german   :- european, verify(german) .
finnish  :- european, verify(finnish) .
british  :- european, verify(british) .
polish   :- european, verify(polish) .
russian  :- european, verify(russian) .
danish   :- european, verify(danish) .
monegasque:- european,verify(monegasque) .
spanish  :- european, verify(spanish) .
italian  :- european, verify(italian) .
dutch    :- european, verify(dutch) .

/* How to verify if the answer introduced by the user is true something */
verify(Q) :- %S is a variable that will initially be what we want to check, ex. french, we will pass it down to the ask function to ask the user and make him change the state to true or false
  (yes(Q) -> true ;
   (no(Q) -> fail ;
    ask(Q))).

ask(Question) :-
   write('Is the driver '), %This will be printed on screen
   write(Question),
   write('?'),
   read(Response), %reads user input
   nl, %new line
   ( (Response == yes ; Response == y ; Response == ye) -> assert(yes(Question)) ; %If user types y, ye or yes then verify will be yes and that will mean that the statement will be introduced to the database as true
     assert(no(Question)), fail). %anything not yes, ye or y will be a no

:- dynamic yes/1,no/1. %Informs the interpreter that the definition of the predicates may change during execution


undo :- retractall(yes(_)),fail. %retractall makes all facts or clauses in the database for which the head unifies with Head are removed.
undo :- retractall(no(_)),fail. %fail makes the predicate to always fail and continue on to the next one.
undo.
```

Online references

[1] Predicate retractall/1. (n.d.). Retrieved November 22, 2019, from https://www.swi-prolog.org/pldoc/doc_for?object=retractall/1.

[2] Prolog -- fail/0. (n.d.). Retrieved November 22, 2019, from https://www.swi-prolog.org/pldoc/man?predicate=fail/0.

[3] Predicate dynamic/1. (n.d.). Retrieved November 22, 2019, from https://www.swi-prolog.org/pldoc/doc_for?object=(dynamic)/1.

[4]Adventures, Objects, Animals, and Taxes. (n.d.). Retrieved November 21, 2019, from http://www.amzi.com/articles/prolog_fun.htm.