Prolog2Table

The goal of the project is to create a table starting form a Prolog file. A Prolog program consists of a number of clauses. Each clause is either a fact or a rule. A fact must start with a predicate (which is an atom) and end with a fullstop. The predicate may be followed by one or more arguments which are enclosed by parentheses. The arguments can be atoms (in this case, these atoms are treated as constants), numbers or variables. Arguments are separated by commas.
See the following example:

```
sunny.
father(john, peter).
father(john, mary).
mother(susan, peter).
```

A rule can be viewed as an extension of a fact with added conditions that also have to be satisfied for it to be true. It consists of two parts. The first part is similar to a fact (a predicate with arguments). The second part consists of other clauses (facts which are separated by commas) which must all be true for the rule itself to be true. These two parts are separated by ":-". You may interpret this operator as "if" in English.
See the following example:

```
parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).
```

Thus, starting from a Prolog file formed by facts and rules, a table like the one described in the following example should be created.

PROLOG:

```
blue(A).
red(B).
red(C).

colored(X) :-  red(X).
colored(X) :- blue(X).
```

OUTPUT:

	red	blue	colored (function)
A	-	1	1
B	1	-	1
C	1	-	1

Table 1. Example of an output table

In a Prolog program, a presence of a fact indicates a statement that is true. Based on this sentence, a '1' should be inserted in the cell corresponding at one fact. For instance, in the cell 'A-red' there is a '1' because the fact B (red) is in the Prolog file given in input.
Concerning the function, we have a '1' if the rule is true. For instance, in the cell 'A-colored' there is a 1 because the fact blue(A) is in the Prolog file given in input and if something is blue, it is colored as well (colored (X) :- blue (X).).

We assume that in the input Prolog file, there are all facts with at least one argument, so as to create the table in the best possible way. For instance, facts like 'sunny.' are avoid since it has not arguments that can be placed in the object column.
Moreover, facts like 'john_is_cold.' are avoid as well. A fact like 'cold(john).' should be used.

The program scrolls all the lines in the input Prolog file and it recognizes if it is a fact or a rule just checking the presence or the absence of the ':-'.
First of all, the predicate and the arguments are identified for each fact in order to build the row and column of the output table.
The predicates are inserted in the table as column.
If the predicate is followed by one parameter, it is inserted in the table as string. Otherwise, if the predicate is followed by more parameter, they are inserted in the table as a list. The parameters of each fact are inserted in the column 'object' of the table.
Once the predicate and the parameters are inserted in the table, all the '1' are inserted as well in the right position. A '0' is inserted if the 'not' is present in the fact.

The predicate of the left-side of each rule is inserted in the table as well as a column (as the function).
A special column ('----------') is inserted to separate the predicates of the fact and the predicate of the rules.

It is also used a Python library called PySWIP [https://github.com/yuce/pwswip]. PySwip is a Python - SWI-Prolog bridge enabling to query SWI-Prolog in your Python programs. It features an SWI-Prolog foreign language interface, a utility class that makes it easy querying with Prolog.
The use of this library was necessary because all the facts and the rules have to be executed in order to implement the right decision. In fact, in order to put the '1' in the columns identified by the functions, a query corresponding to the right-side part of each rule is executed. The corresponding output will be used to check for which parameters the rule is true and thus, in order to put the '1' in the right positions.

A very simple example is shown below.

PROLOG:

```
car(vw_beatle).
car(ford_escort).
bike(harley_davidson).
red(vw_beatle).
red(ford_escort).
blue(harley_davidson).

fun(X) :- red(X), car(X).
fun(X) :- blue(X),bike(X).
```

OUTPUT:

	object	car	bike	red	blue	----------	fun
0	vw_beatle	1	-	1	-	-	1
1	ford_escort	1	-	1	-	-	1
2	harley_davidson	-	1	-	1	-	1
3	----------	-	-	-	-	-	-

Image 1. The first example of an output table

As you can see, in the fourth row the object '---------' is present. The reason is because after the execution of the query corresponding to the right-side part of each rule, a new object might be added. Moreover, a new column might be added as well, since a new fact has been created. This new fact will be executed with PySWIP as well.

The example that is shown below will clarify this fact.

PROLOG:

```
blue(sky).
red(fire).
father(jack, susan).
father(jack, ray).
father(david, liza).
father(david, john).
father(john, peter).
father(john, mary).
mother(karen, susan).
mother(karen, ray).
mother(amy, liza).
mother(amy, john).
mother(susan, peter).
mother(susan, mary).

likes(john, susie).
likes(X, susie).
likes(john, Y).
prefers(john, susie); likes(john, francy).
not(likes(john, pizza)).
not(prefers(pippo, lasagna)); not(likes(luca, pasta)).
not(prefers(francesco, pizza)); likes(andrea, matriciana).


hates(X,Y) :- not(likes(X,Y)).
enemies(X,Y) :- not(likes(X,Y)),not(likes(Y,X)).
parent(X, Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
grandfather(X,Y) :- father(X,Z), parent(Z,Y).
grandmother(X,Y) :- mother(X,Z), parent(Z,Y).
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
mama(X,Y) :- mother(X,Z), father(Z,Y).
colored(X) :- red(X).
colored(X) :- blue(X).
```

OUTPUT:

	object	blue	red	...	grandparent	mama	colored
0	sky	1	-	...	-	-	1
1	fire	-	1	...	-	-	1
2	['jack', 'susan']	-	-	...	-	-	-
3	['jack', 'ray']	-	-	...	-	-	-
4	['david', 'liza']	-	-	...	-	-	-
5	['david', 'john']	-	-	...	-	-	-
6	['john', 'peter']	-	-	...	-	-	-
7	['john', 'mary']	-	-	...	-	-	-
8	['karen', 'susan']	-	-	...	-	-	-
9	['karen', 'ray']	-	-	...	-	-	-
10	['amy', 'liza']	-	-	...	-	-	-
11	['amy', 'john']	-	-	...	-	-	-
12	['susan', 'peter']	-	-	...	-	-	-
13	['susan', 'mary']	-	-	...	-	-	-
14	['john', 'susie']	-	-	...	-	-	-
15	['X', 'susie']	-	-	...	-	-	-
16	['john', 'Y']	-	-	...	-	-	-
17	['john', 'francy']	-	-	...	-	-	-
18	['john', 'pizza']	-	-	...	-	-	-
19	['pippo', 'lasagna']	-	-	...	-	-	-
20	['luca', 'pasta']	-	-	...	-	-	-
21	['francesco', 'pizza']	-	-	...	-	-	-
22	['andrea', 'matriciana']	-	-	...	-	-	-
23	----------	-	-	...	-	-	-
24	['susie', 'susie']	-	-	...	-	-	-
25	['john', 'john']	-	-	...	-	-	-
26	['jack', 'peter']	-	-	...	1	-	-
27	['jack', 'mary']	-	-	...	1	-	-
28	['david', 'peter']	-	-	...	1	-	-
29	['david', 'mary']	-	-	...	1	-	-
30	['karen', 'peter']	-	-	...	1	-	-

Image 2. The second example of an output table

The output table is then converted to csv file and saved in the working directory.

Table2Prolog

The project also allows the user to generate a Prolog file starting from the table described in the section above. It is just a preliminary version, so some causes might not be executed in the best way.
The mechanism is the opposite of the one described above. First of all, the facts are written in the file. The program scrolls all the table's rows until the row with '----------' is found.
The predicate and the arguments are identified for each fact checking the cell where the '1' is placed. The corresponding fact is then written in the output file.

Concerning the construction of the rules, the algorithm can build the rules if and only if the arguments of the rules are all variables. Let's see some examples:

loves(X, Y) :- right_for(X, Y).
hates(X, Y) :- not(likes(X, Y)).
mama(X, Y) :- mother(X, Z), father(Z, Y)
colored(X) :- blue(X).
classmates(X, Y) :- takes(X, Z), takes(Y, Z).
fun(X) :- red(X), car(X).

As you can see from the example above, the algorithm is able to build rules that have a single fact in the right-side, but also more than one fact (connected with the 'and'). If the rule has one or more constants as argument in the left-side or in the right-side, the algorithm cannot properly predict and write the rule. For instance, rules like the following ones, will be not predicted.

maturity(X) :- age(X, Y), Y>18.
likes(dick, X) :- likes(X, wine).
right_for(X, joe) :- loves(X, computers).

Now we are going to see an example of the algorithm execution.

TABLE

	object	car	bike	red	blue	----------	fun
0	vw_beatle	1	-	1	-	-	1
1	ford_escort	1	-	1	-	-	1
2	harley_davidson	-	1	-	1	-	1
3	----------	-	-	-	-	-	-

Table 2. Example of an input table

PROLOG

```
car(vw_beatle).
red(vw_beatle).
car(ford_escort).
red(ford_escort).
bike(harley_davidson).
blue(harley_davidson).
fun(X) :- car(X),red(X).
fun(X) :- bike(X),blue(X).
```