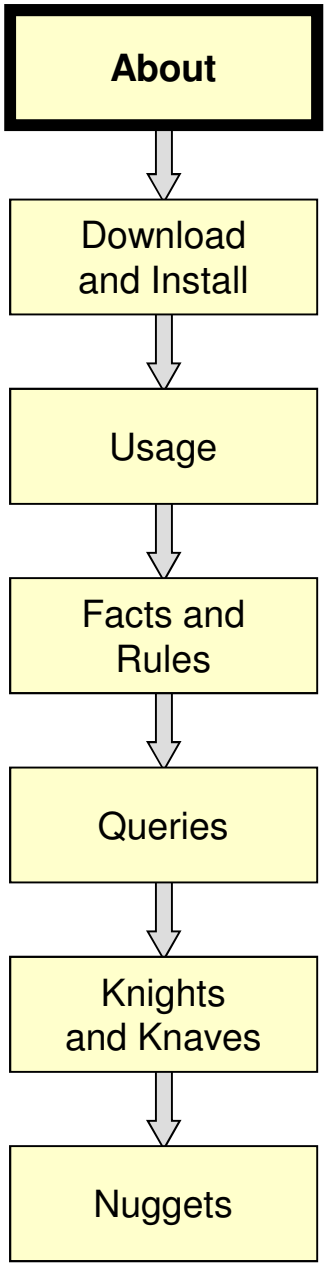
A Brief Introduction to
Prolog Programming

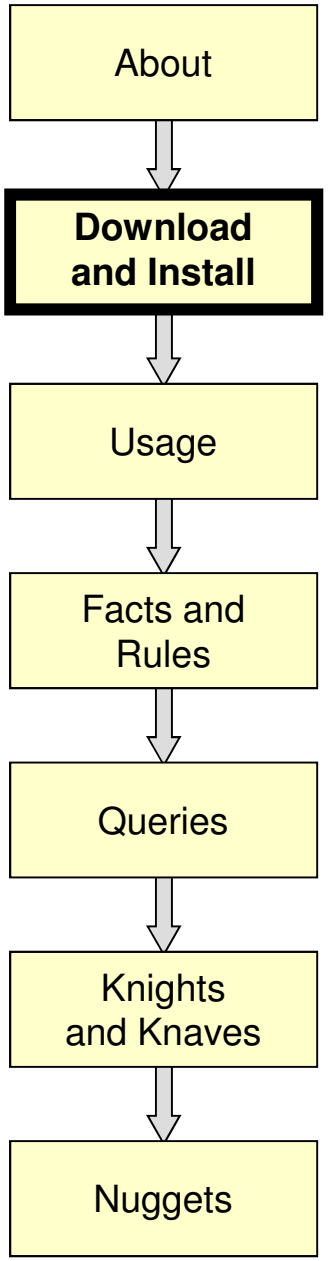
by
Bill Qualls

CSC480 – Artificial Intelligence I
DePaul University
Dr. Jonathan Gemmell, Instructor
Winter 2014



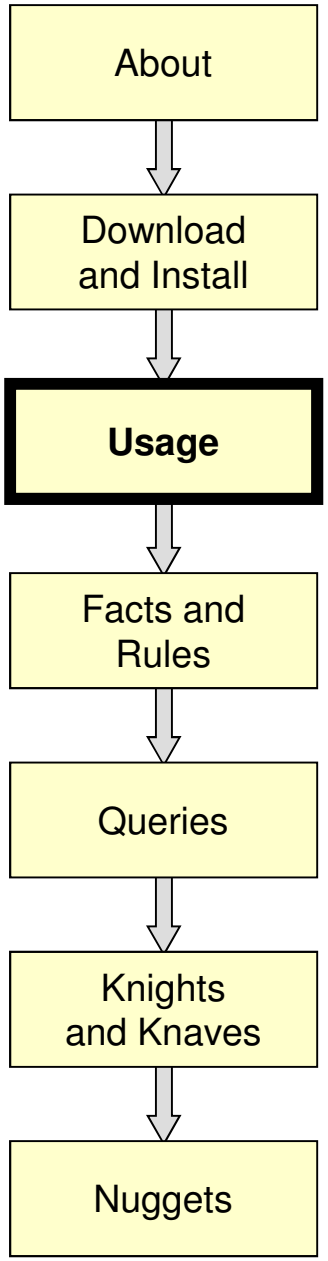
About

- Programming in Logic
- c. 1972
- Declarative (vs. procedural)
- KB = facts and rules stored in file
- User issues queries thru UI



Download and Install

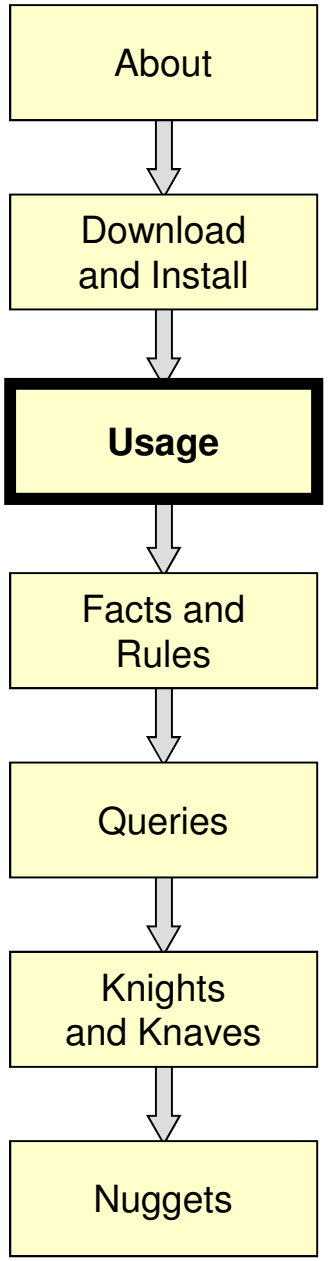
- http://www.swi-prolog.org/
- Version 6.6.1 used herein
- w64pl661.exe



Usage

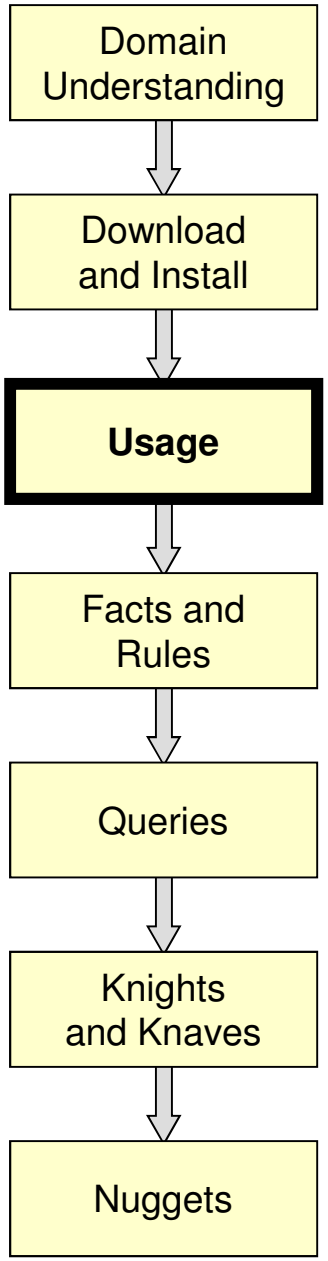
- Use any text editor to create a **knowledge base** with extension of **.pl**
- **%** for comment to end of line
- Each logical line ends in a **period**.
- kids.pl

```
% Bill's kids
son(william).
daughter(cora).
daughter(hannah).
daughter(emma).
```



Usage

- Option 1:
  - Start SWI-Prolog
  - Choose File → Consult
  - Pick file
- Option 2:
  - Double-click on filename
  - Opens SQI-Prolog, Consults file.
- Issue query



Usage

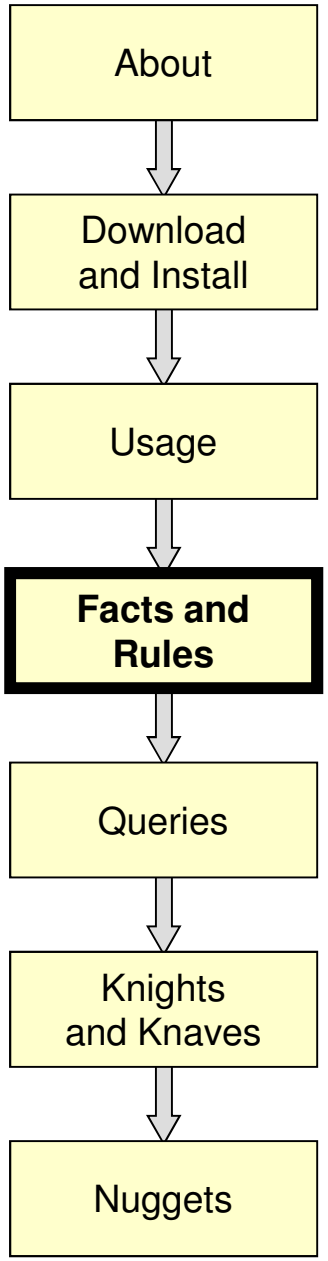
- Sample queries:

```
1 ?- son(william).
true.

2 ?- son(hannah).
false.

3 ?- daughter(X).
X = cora .

4 ?- daughter(X).
X = cora ;
X = hannah ;
X = emma.

5 ?- halt.
```

← uppercase = variable
← pressed Enter

← pressed Space bar
← pressed Space bar

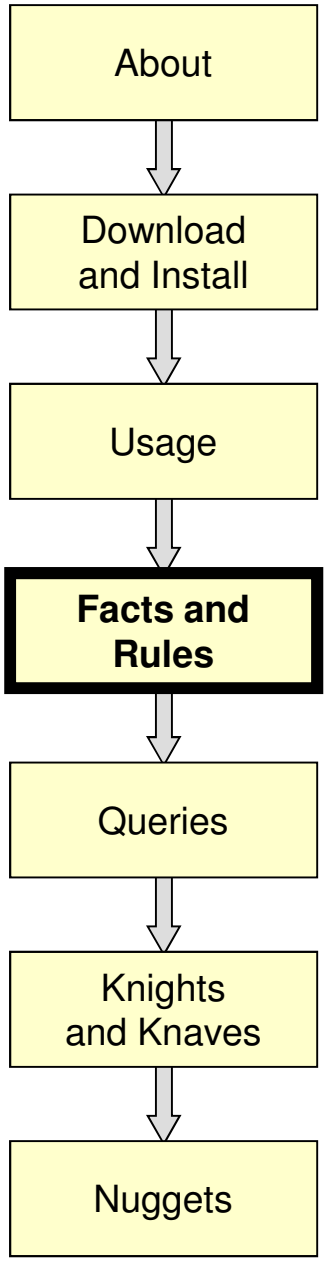
← closes SWI-Prolog



Facts and Rules

- A **fact** is a name (predicate) followed by one or more constants.

```
% Lions, dogs, and zebras are animals.
animal(lion).
animal(dog).
animal(zebra).

% Dogs are carnivores.
carnivore(dog).

% Lions eat zebras.
eats(lion, zebra).

% Zebras can outrun dogs
outruns(zebra, dog).
```

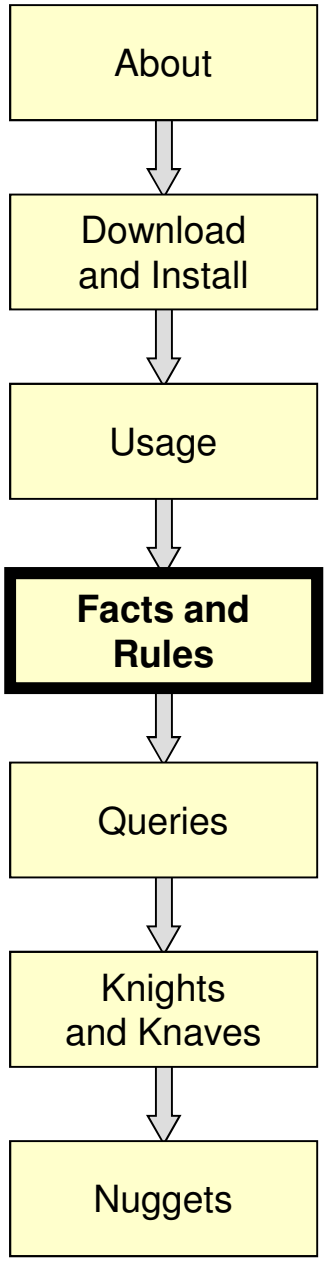


Facts and Rules

- A **rule** is the Prolog equivalent of an "if".

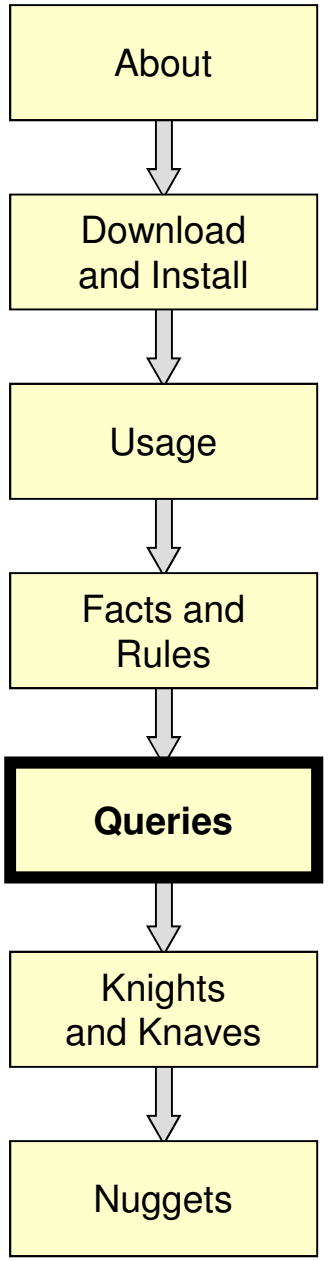
```
man(socrates).
mortal(X) :- man(X).
```

- Use uppercase letters as variables.
- Commas imply "and".
- Semi-colons imply "or".
- More commonly, multiple lines for "or".



Facts and Rules

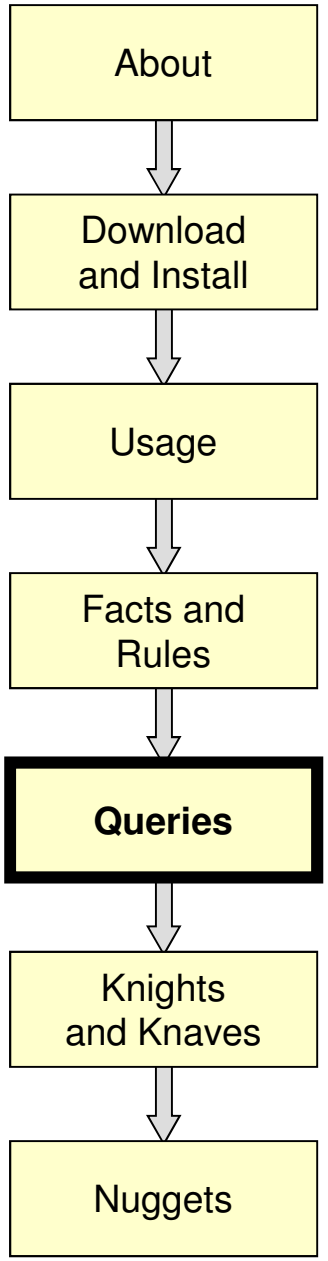
```
% Animals can outrun any animals they eat.
outruns(X, Y) :- animal(X), animal(Y), eats(X, Y).

% Carnivores eat (some) other animals.
carnivore(X) :- animal(X), animal(Y), eats(X, Y).

% Outrunning is transitive.
% This causes stack overflow. Will show fix.
outruns(X, Y) :- outruns(X, A), outruns(A, Y).
```



Queries

- Facts and rules reside in the knowledge base, but queries are entered thru the UI.

```
1 ?- man(socrates).
true.

2 ?- man(X).
X = socrates.

3 ?- mortal(X).
X = socrates.

4 ?- mortal(zebra).
false.
```



Queries

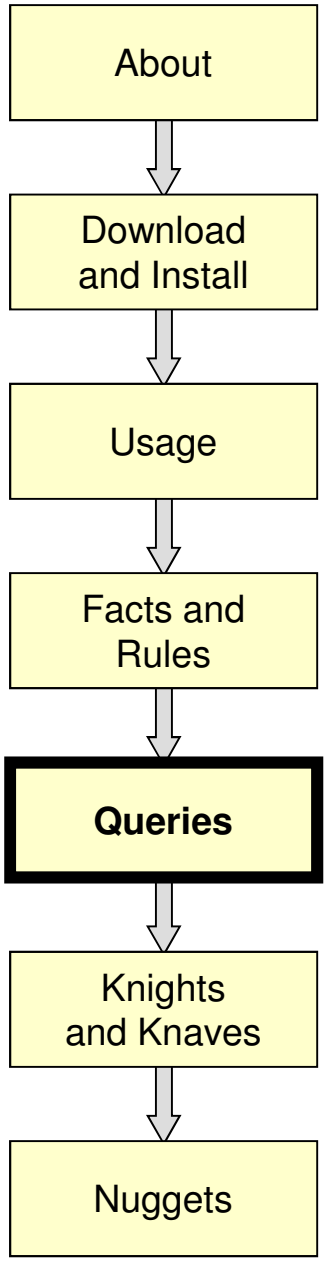
- Can a lion outrun a dog?

```
1 ?- outruns(lion, dog).
true ;
ERROR: Out of local stack
```

- What other animals can a lion outrun?

```
2 ?- outruns(lion, X).
X = zebra ;
X = dog ;
ERROR: Out of local stack
```

- Problem with recursion…let's fix it…



Queries

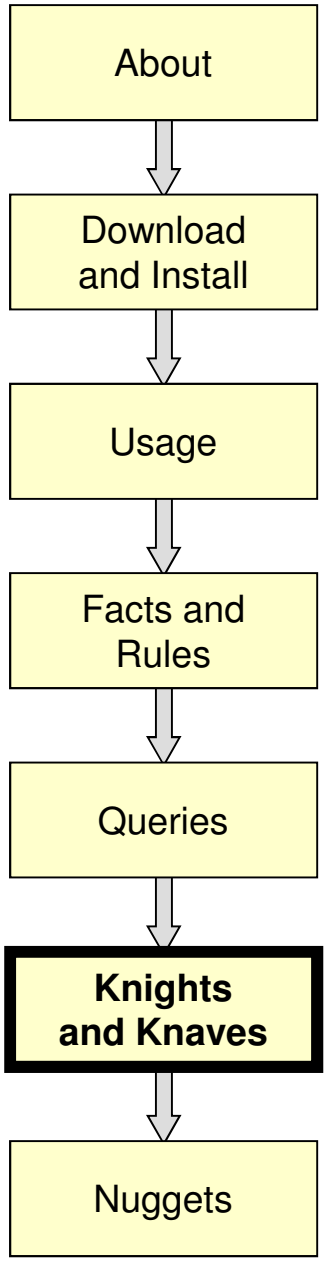
- Modify knowledge base:

```
% Example of tail recursive.
faster(X,Y) :- outruns(X,Y).
faster(X,Z) :- outruns(X,Y), faster(Y,Z).
```

- What other animals can a lion outrun?

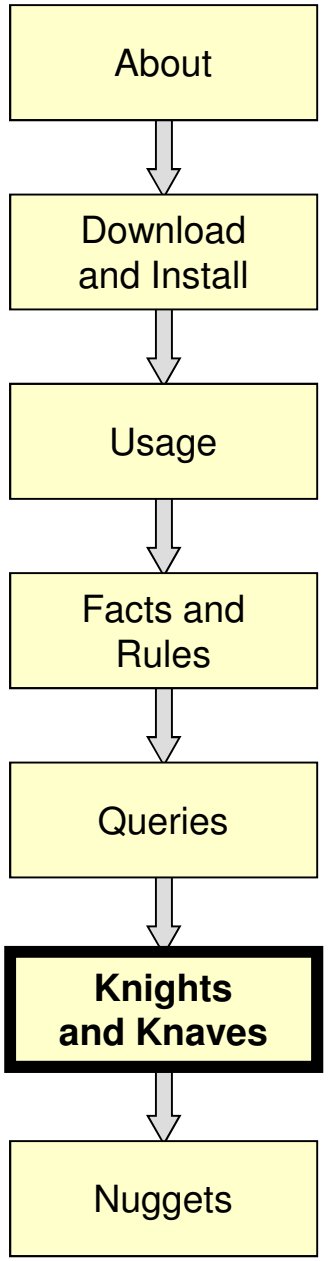
```
3 ?- faster(lion,X).
X = zebra ;
X = dog ;
false.
```

- About that **`false`** … see "Nuggets".



Knights and Knaves

- There are many solutions to the Knights and Knaves problem on the web but this one seemed concise and instructive: http://annisaihsani.wordpress.com/2011/12/11/knight-knave-puzzle/

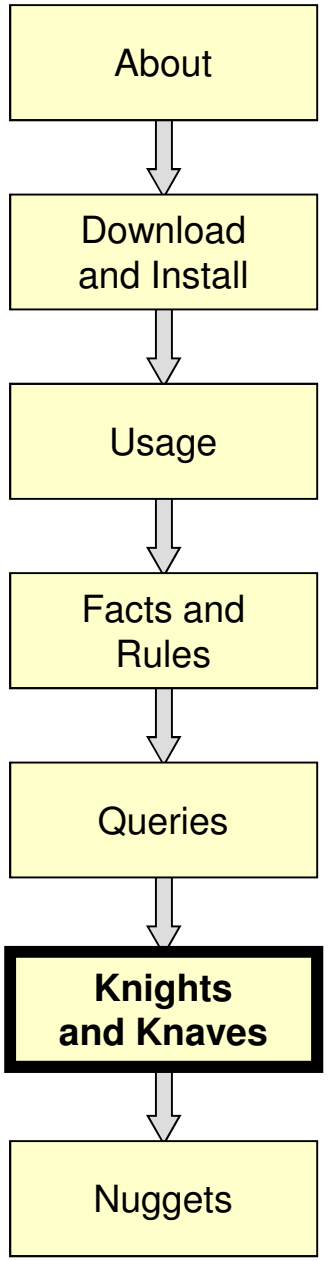


Knights and Knaves

```
truth(not(X),true) :- truth(X,false).

truth(and(X,Y),true) :- truth(X,true), truth(Y,true).

truth(or(X,Y),true) :- truth(X,true) ; truth(Y,true).

truth(imp(X,Y),true) :- truth(X,false) ; truth(Y,true).

truth(exor(X,Y),true) :- (truth(X,true),truth(Y,false))
   ; (truth(X,false),truth(Y,true)).

truth(biimp(X,Y),true) :- truth(exor(X,Y),false).
```

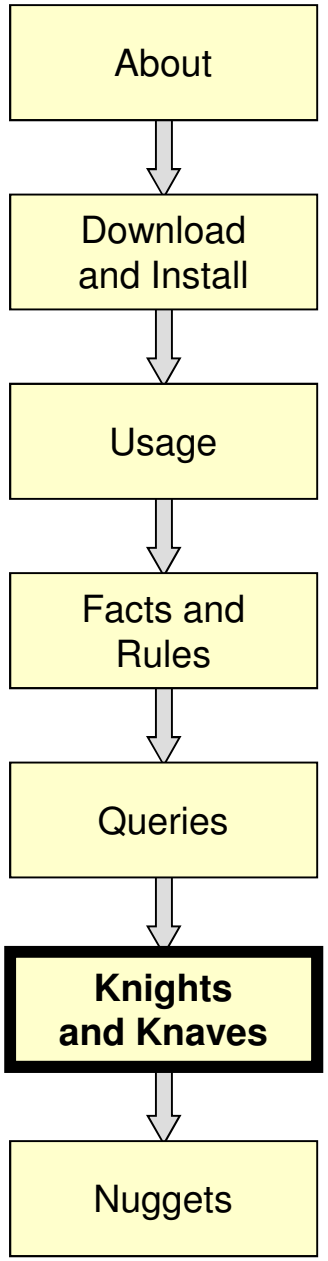
and

or



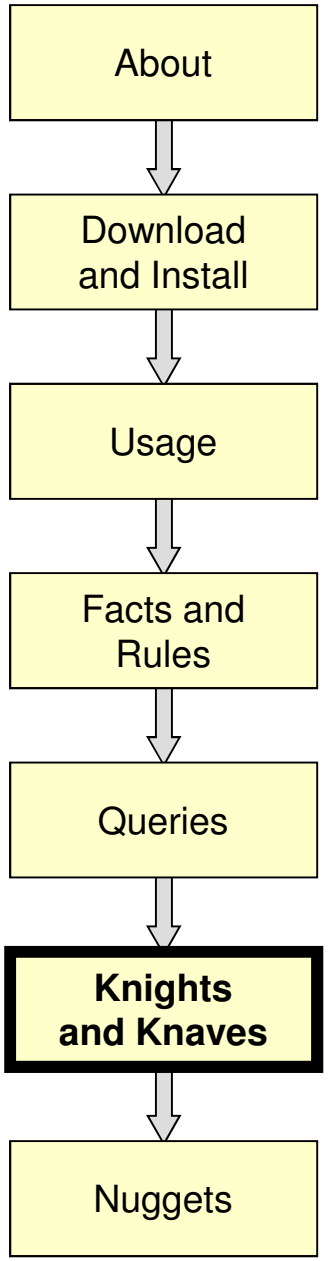
Knights and Knaves

```
truth(not(X),false) :- truth(X,true).

truth(and(X,Y),false) :- truth(X,false) ; truth(Y,false).

truth(or(X,Y),false) :- truth(X,false) , truth(Y,false).

truth(imp(X,Y),false) :- truth(X,true) , truth(Y,false).

truth(exor(X,Y),false) :- (truth(X,true),truth(Y,true))
  ; (truth(X,false),truth(Y,false)).

truth(biimp(X,Y),false) :- truth(exor(X,Y),true).
```



Knights and Knaves

```
truth(is_a(knight,knight),true).
truth(is_a(knave,knave),true).
truth(is_a(knight,knave),false).
truth(is_a(knave,knight),false).

truth(say(knight,Statement),true) :-
  truth(Statement,true).
truth(say(knave,Statement),true) :-
  truth(Statement,false).
truth(say(knight,Statement),false) :-
  truth(Statement,false).
truth(say(knave,Statement),false) :-
  truth(Statement,true).
```



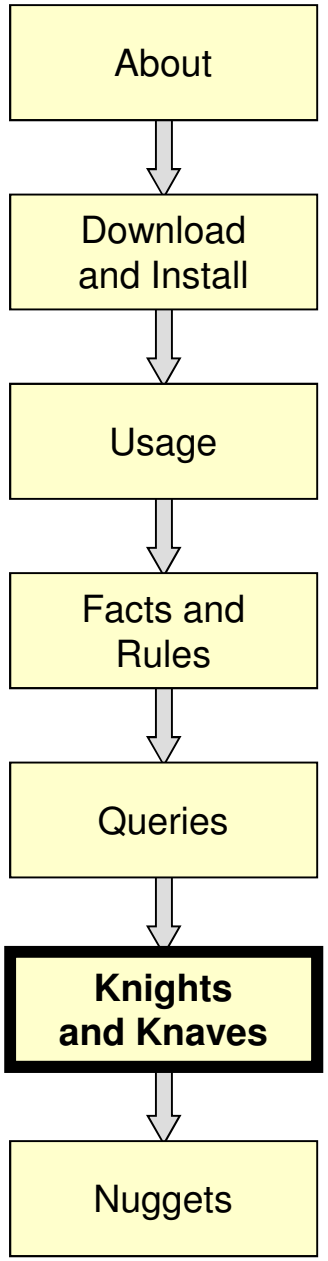
Knights and Knaves

Sample query: A says he is a knave.

```
1 ?- truth(say(A,is_a(knave,A)),TruthValue).
A = knight,
TruthValue = false ;
A = knave,
TruthValue = false ;
false.
```

TruthValue is a variable so it will show all combinations and true or false.

"If A is a knight, he wouldn't lie by saying he is a knave; if A is a knave, he wouldn't admit that he is."



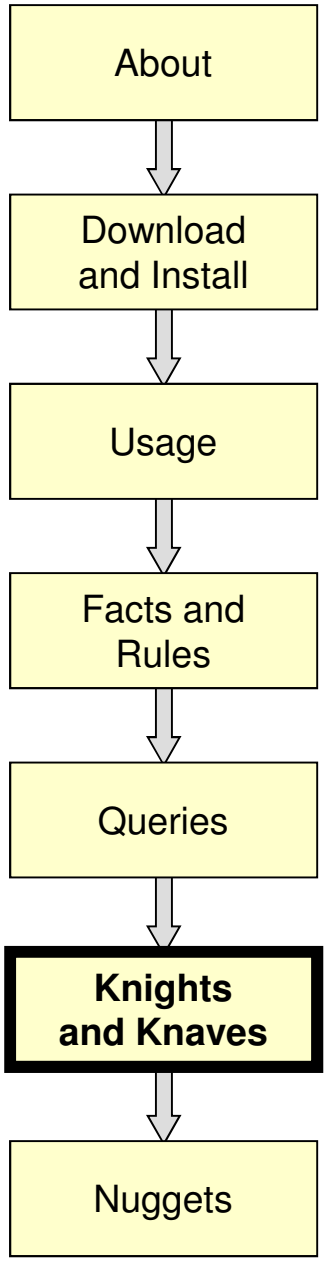
Knights and Knaves

Queries from Quiz #4:

A says, "I am a knave or B is a knight" and B says nothing.

```
?- truth(say(A,or(is_a(knave,A),
   is_a(knight,B))),true).
A = B, B = knight ;
false.
```

true is a constant so show only those combinations for which the statement is true.

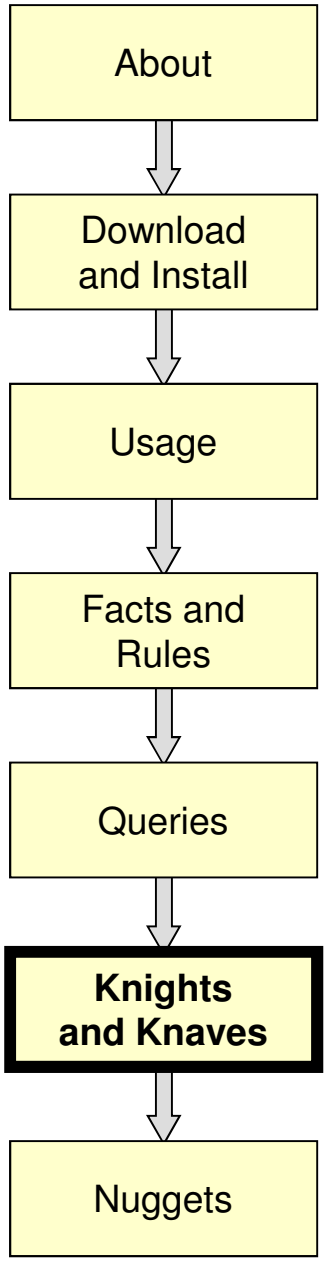


Knights and Knaves

Queries from Quiz #4:

A says, "We are both knaves" and B says nothing.

```
?- truth(say(A,and(is_a(knave,A),
   is_a(knave,B))),true).
A = knave,
B = knight ;
false.
```

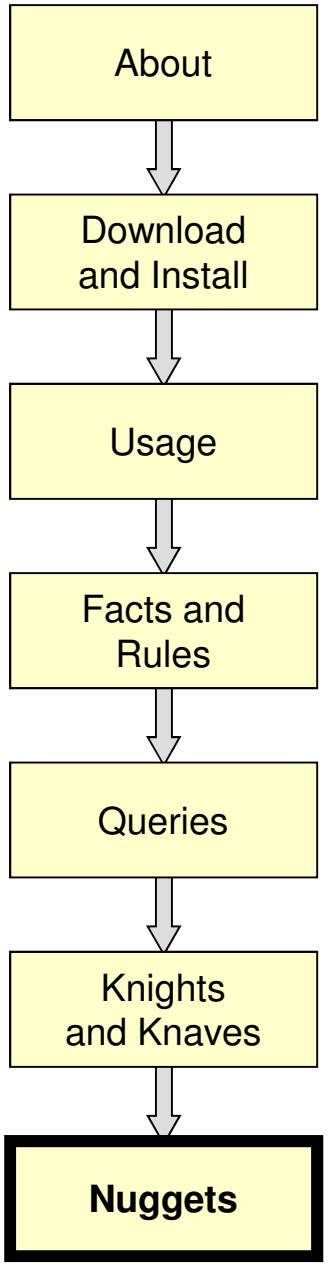


Knights and Knaves

Queries from Quiz #4:

Both A and B say, "I am a knight."

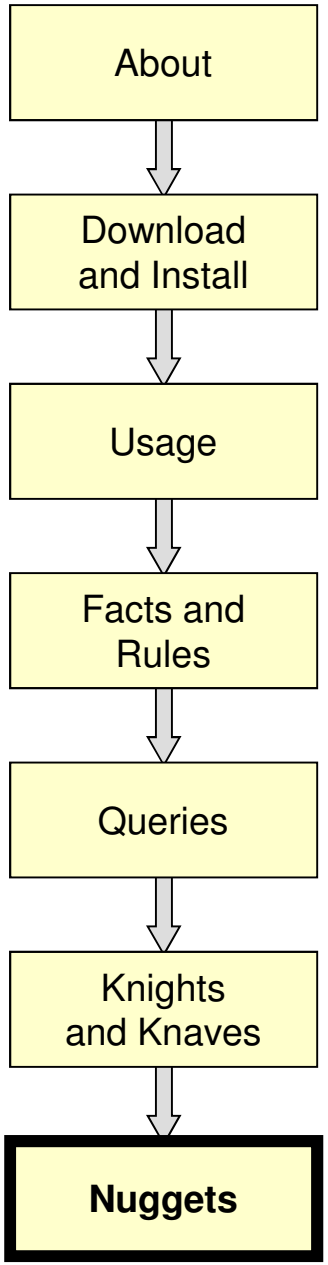
```
?- truth(and(say(A,is_a(knight,A))
   ,say(B,is_a(knight,B))),true).
A = B, B = knight ;
A = knight,
B = knave ;
A = knave,
B = knight ;
A = B, B = knave ;
false.
```



Nuggets

What is a nugget?

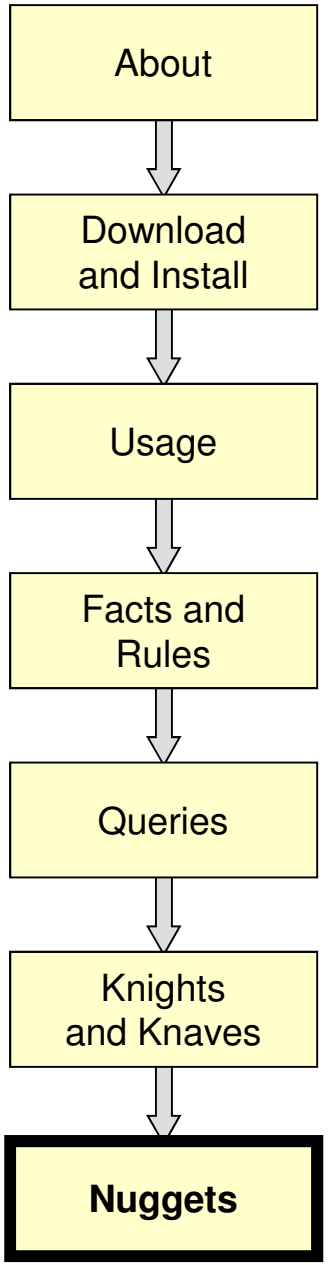
- Nuggets are little gems I have stumbled upon while completing this project.
- They aren't intended to be part of this presentation.
- They are things I don't want to forget; in some cases they represent battle scars.



Nuggets

- Careful! An if is followed by :- not :=
- Colon + hyphen, not colon + equal !!!
- That cost me one night…

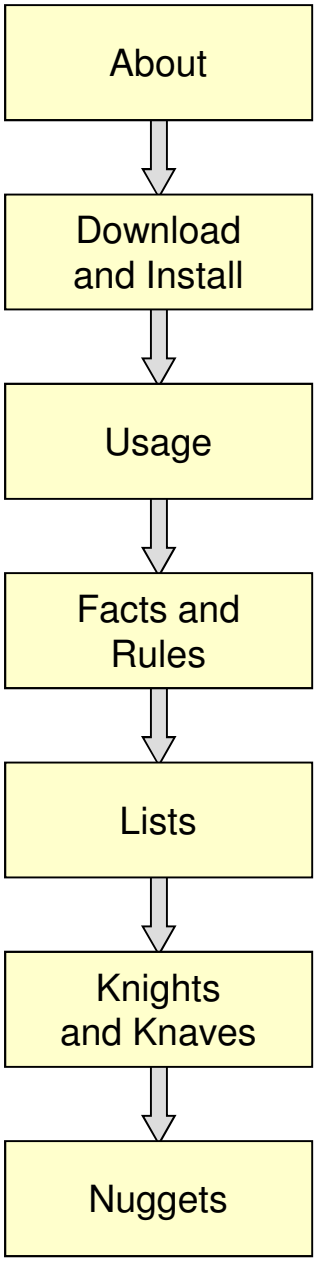
- Stop recursion with "cut" symbol: !
- Cut is to Prolog as GO TO is to COBOL.



Nuggets

Unexpected "false"

- "The user can type the semi-colon (;) or spacebar if (s)he wants another solution. Use the return key if you do not want to see the more answers. Prolog completes the output with a full stop (.) if the user uses the return key or Prolog knows there are no more answers. If Prolog cannot find (more) answers, it writes false. Finally, Prolog answers using an error message to indicate the query or program contains an error." (SWI website)
- "The false response can appear inconsistent to beginning Prolog programmers and "feel" like an error or warning, but it's actually a perfectly normal Prolog response. Prolog is quite consistent in its behavior of attempting to find solutions and, when no more choices exist, will return false. If Prolog has exhausted all the other choices before it finds the final solution, it displays the solution and doesn't return false (as in the case above in which the second clause is the only solution)." (StackOverflow response to a question I posted.)


About
Download and Install
Usage
Facts and Rules
Lists
Knights and Knaves
Nuggets

Thank you.