



# Prolog Programs

Get started using Prolog facts, rules, and variables

# Lesson Overview (Agenda)

In the following lesson, we will explore:

1. Prolog Basics
2. Example: reasoning about family relationships
3. Prolog Rules

# Prolog Basics

- Prolog name comes from PROgramming in LOGic
- DO NOT try to compare Prolog programming to the programming you know
- There are no loops, no if-statements, and no variables of the kind you're used to)
- DO NOT try to translate from Python to Prolog or vice versa
- Prolog = Declarative Programming:
  - Declare facts and rules (prolog program) |: prompt, or enter in file
  - Run prolog interpreter and load program
  - Issue query (run the program) ?- prompt
  - Prolog will find a set of variable bindings such that the query is true, or state "no" if it isn't true

# TWO contexts, code looks same, but different meaning

## 1. Writing a Prolog program

Typing Prolog code into a file  
or Prompt is |:

You are making **true statements**

```
parent(jim,todd).
```

```
parent(jim,X)
```

vs

## 2. Running a Prolog program

Typing Prolog queries into a  
**Prolog interpreter**

Prompt is ?-

You are asking **questions**

```
?- parent(jim,todd).
```

```
?- parent(jim,X).
```

Meaning is different from context 1

# Prolog programs

- A prolog program is a set of statements which are facts and rules, called clauses.
- When writing a prolog program, the programmer has a "world" in mind, and the programmer thinks of and writes down all of the true statements about that world, in prolog syntax.
- Usually we put the set of clauses into a text file with a name that ends with the .pl extension
- Beware the Perl programming language also uses the .pl extension
- Beware that Windows File Explorer (by default) doesn't show the extension of a filename, when you see myfile.pl it might actually have the name myfile.pl.txt

# Prolog Interpreter

- A prolog interpreter is necessary to run a prolog program.
- The user runs the prolog interpreter and **consults** the program file, causing the prolog interpreter to read the prolog program from the file and store the clauses in its memory
- The pseudo-file called **user** corresponds to the programmer typing directly into the interpreter, and those clauses disappear when the interpreter shuts down
- After the program has been consulted into the interpreter, it is ready to run

# Running a Prolog program

- In order to run a Prolog program, after the clauses have been loaded into the interpreter from a file, the user asks questions and receives answers
- The questions are issued in Prolog syntax, and they might involve variables.
- The output of a Prolog program is the answer to the question, which might be false or true.
- If the answer is true, the output also includes the **variable bindings** that make the question true
- A true answer can be thought of as "true if the variables have these values"

# Working with Prolog: what are the stages?

- Modeling: imagine the world you are representing, and what is true about that world
- **Programming:** Write down all the relevant true facts and rules about that world
- Launch the prolog interpreter and consult the program file
- **Running the Program:** issue queries (questions) to the Prolog interpreter and receive the answers
- To distribute your prolog program to your friends, family and other users, you distribute a binary of the interpreter with your program pre-consulted, set up to automatically issue a kickoff query when it runs



# Prolog Basics (Cont'd)

- Usually we put facts and rules in a file (or several files), using a ".pl" extension
- To load our program, we run the prolog interpreter (swipl):
  - Supply our files as arguments on command line
  - Or
  - Use the `consult` builtin predicate
    - `consult` adds the contents of the file to current prolog interpreter state

# Loading programs: consult

Examples of loading facts and rules from a file:

```
?- consult(myfile) .
```

```
?- consult('myfile.pl') .
```

```
?- consult('/path/to/the/file/myfile.pl') .
```

Notice:

- the first two examples above are equivalent if the file is named `myfile.pl`, need quotes if name is not a prolog constant (lower case letter followed by letters, underscores, digits)

# List notation for consult

We can use prolog's list notation to consult files:

```
?- [myfile].
```

```
?- ['myfile.pl'].
```

```
?- [file1, file2, 'file3.1'].
```

- If the filename is a single word without the .pl extension, if necessary, prolog will try adding the .pl extension
- If the filename has special characters like period/dot, or begins with a capital letter, it must be in quotes

# Special file name "user"

- We can type facts and rules directly into prolog with the file name `user` (EOF character to finish)
- Those facts and rules will be lost when we exit the prolog interpreter

```
?- [user].
```

```
| : abc.
```

```
| : like(zzz).
```

```
| : ^D% user://1 compiled 0.00 sec, 2 clauses
```

```
true.
```

```
?-
```

# Simple first program

Contents of the file called `first.pl`:

```
% a simple program stating that xyz is true  
% xyz means nothing in particular  
xyz.
```

# Run the first program

Assume % is the command line prompt

```
% swipl
```

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)
```

```
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).
```

```
?- [first].
```

```
true.
```

```
?- xyz.
```

```
true.
```

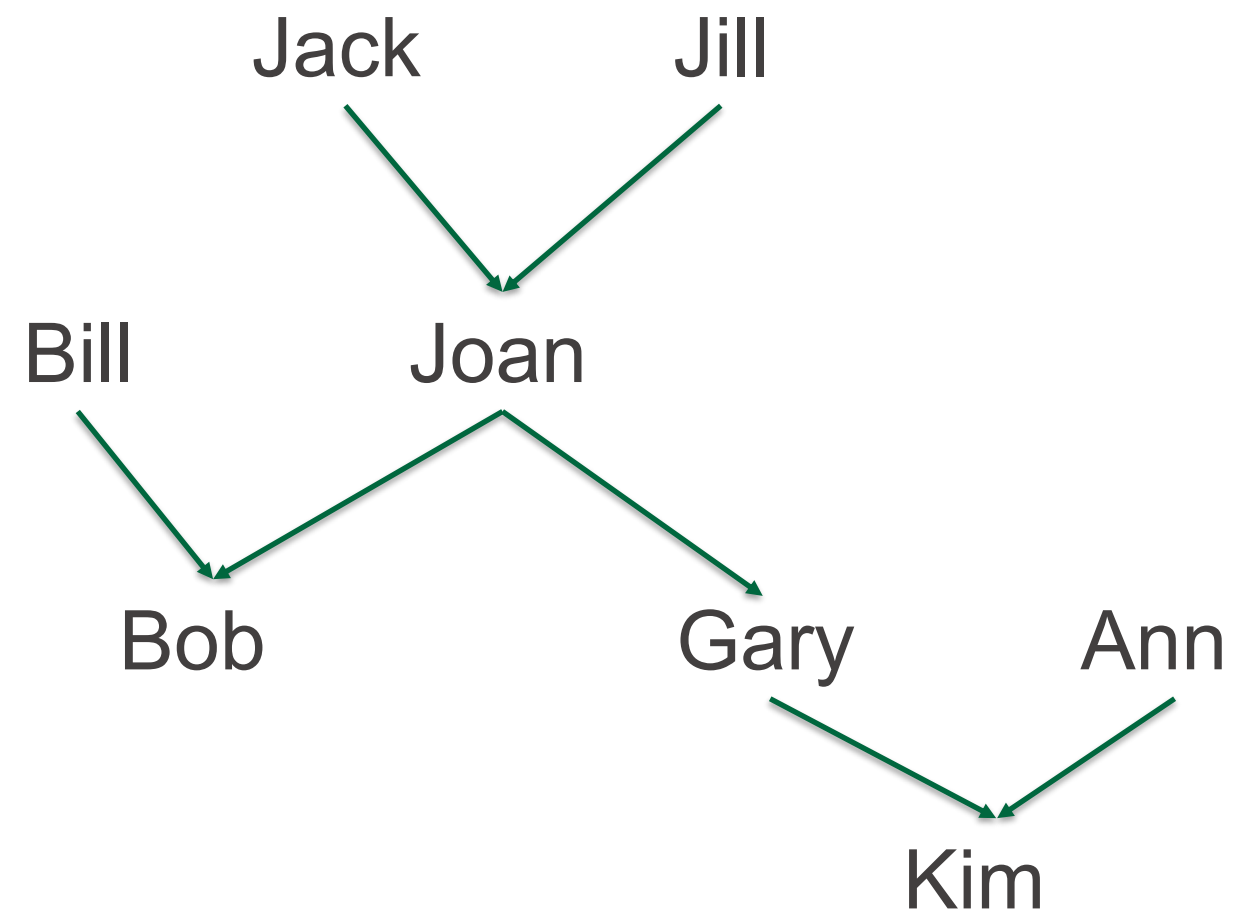
# Time to check your learning!

Let's see how many key concepts from prolog basics you recall by answering the following questions!

1. What built-in predicate do we use to load programs from files into the prolog interpreter?
2. When the user is consulted, to type facts/rules directly into a prolog interpreter, what happens to those facts/rules when the prolog interpreter is terminated?

# Example: Family Relationships

Consider the following family tree:





# Family Relationships (cont'd)

The arrow points from a parent to a child.

We can represent this as a prolog program (notice there are no capital letters yet)

```
% parent(X,Y) means that X is a parent of Y
```

```
parent(jack, joan).
```

```
parent(jill, joan).
```

```
parent(bill, bob).
```

```
parent(joan, bob).
```

```
parent(joan, gary).
```

```
parent(gary, kim).
```

```
parent(ann, kim).
```

# Declarative Comments

We need to state what a predicate means when we define it.

The comment character is %

```
% yellow(X) means that X is of yellow colour  
yellow(sun) .
```

Or

```
% yellow(X) is true when X is of yellow colour  
yellow(sun) .
```

# Variables (begin with capital, X on this slide)

Context 1 (making statements):

`% yellow(X) means that X is of yellow colour`

`yellow(sun) .      % means the sun is yellow`

Or

`% yellow(X) is true when X is of yellow colour`

`yellow(X) .      % means that everything is yellow`

Context 2 (asking questions)

`?- yellow(sun)      % is the sun yellow?`

`?- yellow(X) .      % is there something X that is yellow?`

# Family Relationships (cont'd)

We can type our family relationships program into a file called family.pl and load it into prolog :

```
% swipl
```

```
?- [family].
```

```
true.
```

```
?- parent(bill,bob).
```

```
true.
```

```
?- parent(jim,bob).
```

```
false.
```

# Predicates and constants

We have defined a `parent` predicate of arity 2 (2 arguments).

We use the notation `parent/2` to indicate a `parent` predicate of arity 2

The general form is `predicate(term1, term2, ..., termk)`

The number of terms is the arity of the predicate, and if the arity is 0, the parenthesis can be left out (remember `xyz` from `first.pl` a few slides back?)

# Closed world assumption

Prolog operates according to the closed-world assumption:

The only things that are true are the things we state, and the logical entailments of what we state.

In our program about family relationships, there are only 7 persons: no other persons exist in the closed world

How many of those persons have parents? (only 4 have parents in the closed world)

# Prolog Queries and Variable Bindings

A variable begins with a capital letter, or a single underscore is the anonymous variable.

Here's a query with a variable, typing return after P=jack:

```
?- parent (P, joan) .
```

```
P=jack .
```

```
?-
```

Here we type semi-colon ; after P=jack, meaning find another solution:

```
?- parent (P, joan) .
```

```
P=jack;
```

```
P=jill
```

```
?-
```

# Anonymous Variable

Anonymous variable `_` represents something but we don't care what it is.

```
?- parent(X,_) .      % query context, who has a child
```

```
X = jack
```

```
?- parent(X,Y) .      % who has a child, and which child?
```

```
X = jack
```

```
Y = joan
```



# Time to check your learning!

Let's see how many key concepts from the family program you recall by answering the following questions!

What is meant by the arity of a predicate?

What constitutes a prolog variable?

What is the closed world assumption?

# Prolog Rules

We have a parent relation defined in our family.pl program

Now let's think about an ancestor relation defined in English:

If person X is a parent of person Y, then person X is an ancestor of person Y

Also

If person X is a parent of person Y, and person Y is an ancestor of person Z, then person X is an ancestor of person Z

More compactly:

If `parent(X,Y)` then `ancestor(X,Y)`

If `parent(X,Y)` and `ancestor(Y,Z)` then `ancestor(X,Z)`

# Prolog Rules (cont'd)

A rule such as

if P then Q

is written the opposite way around in Prolog as

$Q :- P$

We read that as "Q if P" and it means that when we are trying to show that Q is true, we can succeed by showing that P is true

# ancestor rules

We can write our rules about ancestors in Prolog form:

```
ancestor(X,Y):-parent(X,Y).
```

```
ancestor(X,Z):-parent(X,Y), ancestor(Y,Z).
```

Note that comma "," means "and" in Prolog

The scope of variables is the single clause

We haven't used it yet, but ";" means "or" in Prolog

There would be problems if we said

```
ancestor(X,Z):-ancestor(X,Y),ancestor(Y,Z).
```

Even though it's true, a Prolog programmer would not say that (why?).

# ancestor query

Let's add the ancestor clauses to our family.pl program, then

```
?- [family].
```

```
true.
```

```
?- ancestor(X,Y).
```

We can keep typing ";" to see all of Prologs answers to this query.

Let's also try the same query with the problematic version of that ancestor rule.

# Time to check your learning!

Let's see how many key concepts from Prolog rules you recall by answering the following questions!

How would we write "if a then b" as a Prolog rule

How would we write "if X is a student, then X is a person" as a Prolog rule?

# Conclusion

In this lesson, you learned how to

- Write and run simple Prolog programs
- Use Prolog facts in programs
- Use Prolog variables in programs and queries
- Use Prolog rules in programs

In the next lesson, you will learn

- More details about how Prolog answers queries