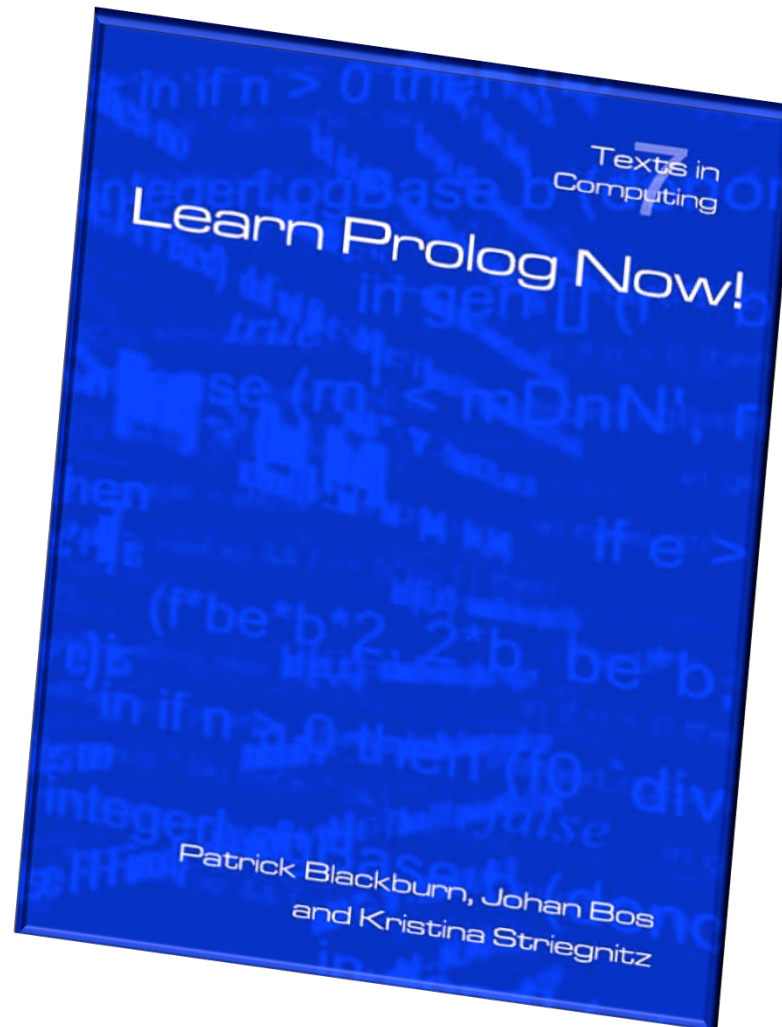


# Learn Prolog Now!



# SWI Prolog

---

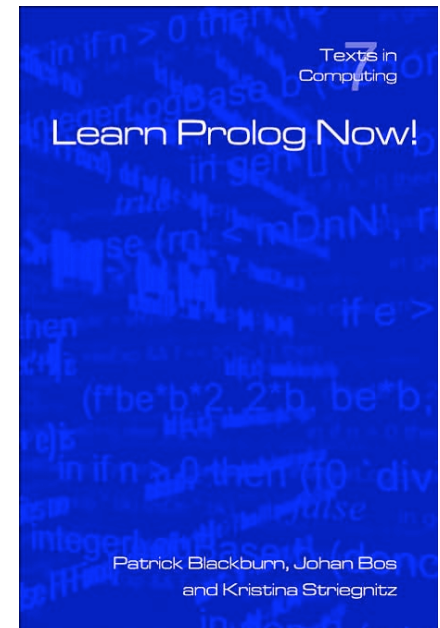
- Freely available Prolog interpreter
- Works with
  - Linux,
  - Windows, or
  - Mac OS
- There are many more Prolog interpreters
- Not all are ISO compliant/free



**SWI Prolog**

# Lecture 1

- Theory
  - Introduction to Prolog
  - Facts, Rules and Queries
  - Prolog Syntax
- Exercises
  - Exercises of LPN chapter 1
  - Practical work



# Aim of this lecture (1/2)

---

- Give some simple examples of Prolog programs
- Discuss the three basic constructs in Prolog:
  - Facts
  - Rules
  - Queries

# Aim of this lecture (2/2)

---

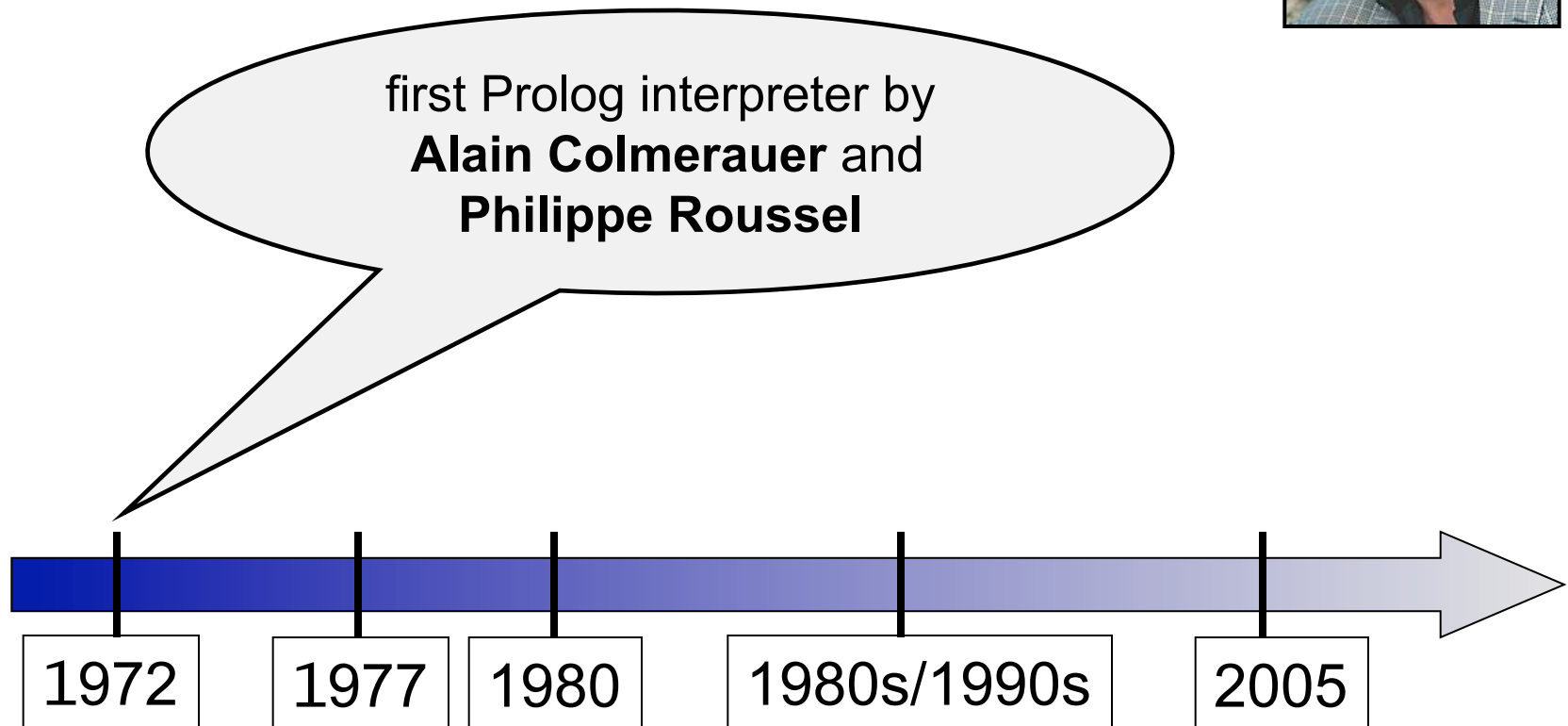
- Introduce other concepts, such as
  - the role of logic
  - unification with the help of variables
- Begin the systematic study of Prolog by defining
  - terms
  - atoms, and
  - variables

# Prolog

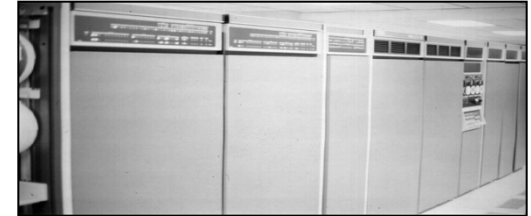
---

- "Programming with Logic"
- Very different from other programming languages
  - Declarative (not procedural)
  - Recursion (no "for" or "while" loops)
  - Relations (no functions)
  - Unification

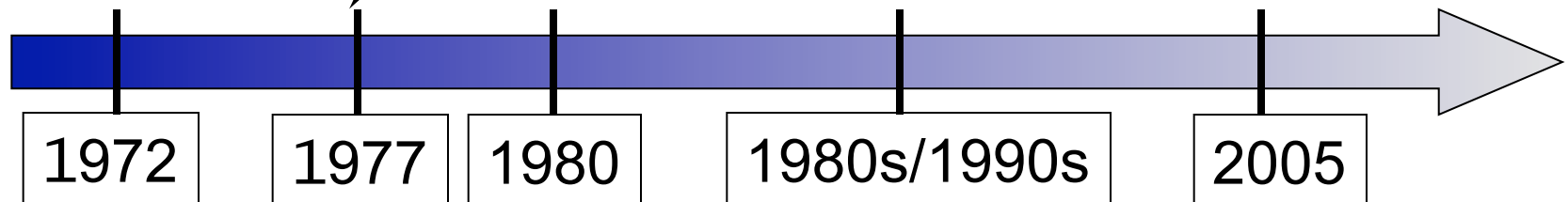
# History of Prolog



# History of Prolog



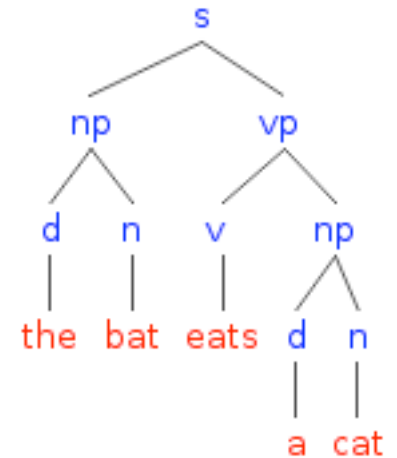
implementation of DEC10  
compiler by **David H.D. Warren**





# History of Prolog

Definite Clause Grammars  
implementation by  
**Pereira and Warren**



1972

1977

1980

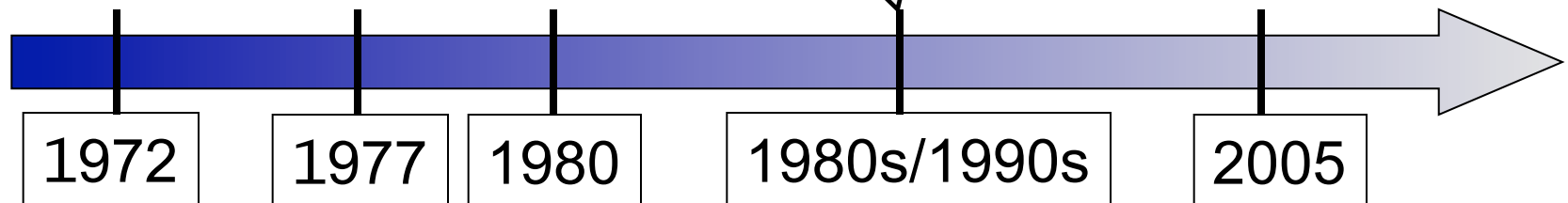
1980s/1990s

2005

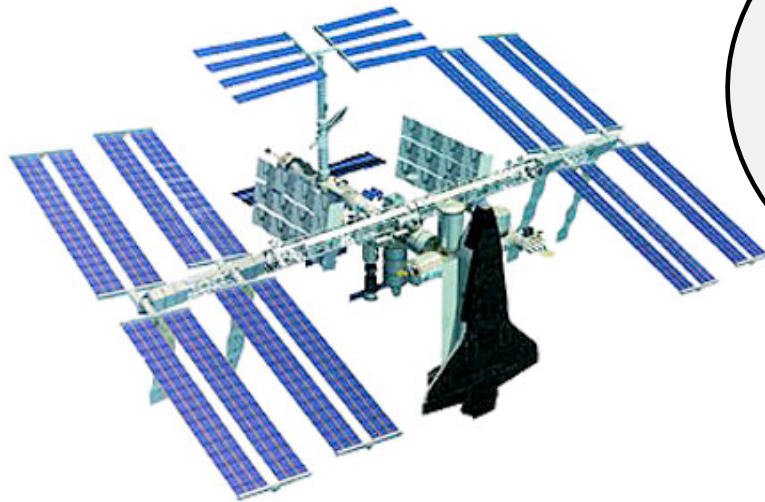
# History of Prolog



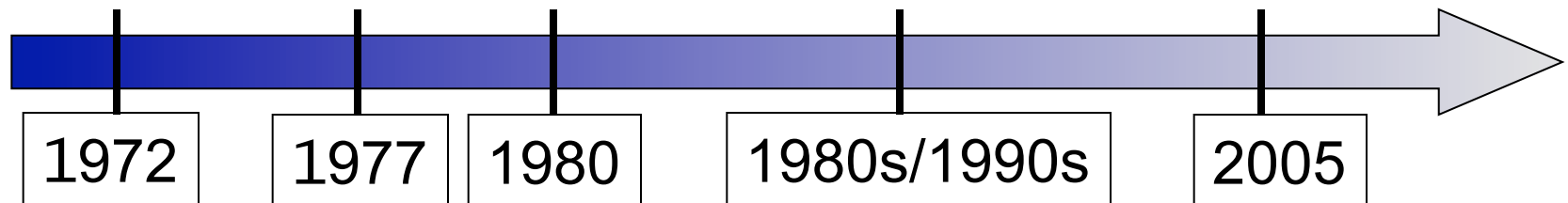
Prolog grows in popularity especially in Japan and Europe



# History of Prolog



Prolog used to program natural language interface in International Space Station by NASA



1972

1977

1980

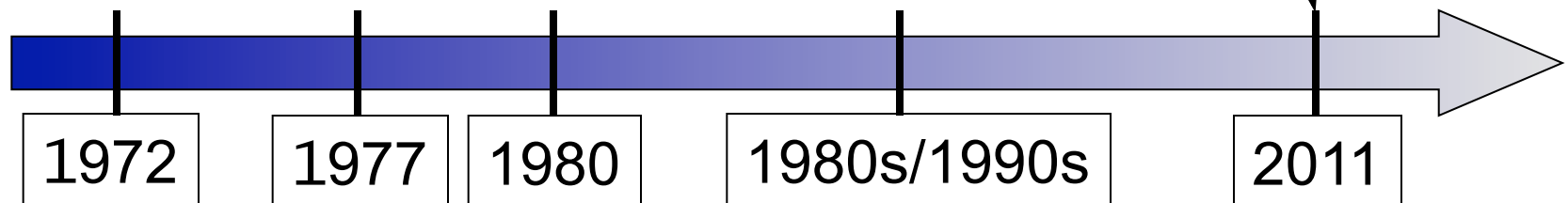
1980s/1990s

2005

# History of Prolog



Parts of IBM's  
Watson QA  
supercomputer  
were coded in  
Prolog



# Prolog and Web Applications

- prolog programs are often smaller
- smallness encourages well written code
- hence, easier to maintain



Source:

<http://www.pathways1ms.com/swipltuts/>

# Basic idea of Prolog

---

- Describe the situation of interest
- Ask a question
- Prolog:
  - logically deduces new facts about the situation we described
  - gives us its deductions back as answers

# Consequences

---

- Think declaratively, not procedurally
  - Challenging
  - Requires a different mindset
- High-level language
  - Not as efficient as, say, C
  - Good for rapid prototyping
  - Useful in many AI applications  
(knowledge representation, inference)

# Knowledge Base 1

woman(mia).

woman(jody).

woman(yolanda).

playsAirGuitar(jody).

party.





# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?-

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- woman(mia).

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- woman(mia).  
yes  
?-

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- woman(mia).  
yes  
?- playsAirGuitar(jody).

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- woman(mia).  
yes  
?- playsAirGuitar(jody).  
yes  
?-

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- woman(mia).  
yes  
?- playsAirGuitar(jody).  
yes  
?- playsAirGuitar(mia).

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- woman(mia).  
yes  
?- playsAirGuitar(jody).  
yes  
?- playsAirGuitar(mia).  
no

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- tattooed(jody).



# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- tattooed(jody).  
no  
?-

# Knowledge Base 1

```
woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.
```

```
?- tattooed(jody).  
ERROR: predicate tattooed/1 not defined.  
?-
```

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- party.

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- party.  
yes  
?-

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- rockConcert.

# Knowledge Base 1

woman(mia).  
woman(jody).  
woman(yolanda).  
playsAirGuitar(jody).  
party.

?- rockConcert.  
no  
?-

# Knowledge Base 2

happy(yolanda).

listens2music(mia).

listens2music(yolanda):- happy(yolanda).

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).



# Knowledge Base 2

happy(yolanda).

fact

listens2music(mia).

listens2music(yolanda):- happy(yolanda).

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).



# Knowledge Base 2

happy(yolanda).

fact

listens2music(mia).

fact

listens2music(yolanda):- happy(yolanda).

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).

# Knowledge Base 2

happy(yolanda).

fact

listens2music(mia).

fact

listens2music(yolanda):- happy(yolanda).

rule

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).

# Knowledge Base 2

happy(yolanda).

fact

listens2music(mia).

fact

listens2music(yolanda):- happy(yolanda).

rule

playsAirGuitar(mia):- listens2music(mia).

rule

playsAirGuitar(yolanda):- listens2music(yolanda).

# Knowledge Base 2

happy(yolanda).

fact

listens2music(mia).

fact

listens2music(yolanda):- happy(yolanda).

rule

playsAirGuitar(mia):- listens2music(mia).

rule

playsAirGuitar(yolanda):- listens2music(yolanda).

rule

# Knowledge Base 2

```
happy(yolanda).  
listens2music(mia).  
listens2music(yolanda):- happy(yolanda).  
playsAirGuitar(mia):- listens2music(mia).  
playsAirGuitar(yolanda):- listens2music(yolanda).
```



head

body

# Knowledge Base 2

happy(yolanda).

listens2music(mia).

listens2music(yolanda):- happy(yolanda).

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).

?-

# Knowledge Base 2

```
happy(yolanda).  
listens2music(mia).  
listens2music(yolanda):- happy(yolanda).  
playsAirGuitar(mia):- listens2music(mia).  
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?- playsAirGuitar(mia).  
yes  
?-
```

# Knowledge Base 2

```
happy(yolanda).  
listens2music(mia).  
listens2music(yolanda):- happy(yolanda).  
playsAirGuitar(mia):- listens2music(mia).  
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?- playsAirGuitar(mia).  
yes  
?- playsAirGuitar(yolanda).  
yes
```



# Clauses

```
happy(yolanda).  
listens2music(mia).  
listens2music(yolanda):- happy(yolanda).  
playsAirGuitar(mia):- listens2music(mia).  
playsAirGuitar(yolanda):- listens2music(yolanda).
```

*There are five clauses in this knowledge base:  
two facts, and three rules.*

*The end of a clause is marked with a full stop.*

# Predicates

```
happy(yolanda).  
listens2music(mia).  
listens2music(yolanda):- happy(yolanda).  
playsAirGuitar(mia):- listens2music(mia).  
playsAirGuitar(yolanda):- listens2music(yolanda).
```

*There are three **predicates** in  
this knowledge base:*

*happy, listens2music, and playsAirGuitar*

# Knowledge Base 3

happy(vincent).

listens2music(butch).

playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).

playsAirGuitar(butch):- happy(butch).

playsAirGuitar(butch):- listens2music(butch).



# Expressing Conjunction

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch).  
playsAirGuitar(butch):- listens2music(butch).
```

*The comma "," expresses conjunction in Prolog*

# Knowledge Base 3

happy(vincent).

listens2music(butch).

playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).

playsAirGuitar(butch):- happy(butch).

playsAirGuitar(butch):- listens2music(butch).

?- playsAirGuitar(vincent).

# Knowledge Base 3

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch).  
playsAirGuitar(butch):- listens2music(butch).
```

```
?- playsAirGuitar(vincent).  
no  
?-
```

# Knowledge Base 3

happy(vincent).

listens2music(butch).

playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).

playsAirGuitar(butch):- happy(butch).

playsAirGuitar(butch):- listens2music(butch).

?- playsAirGuitar(butch).

# Knowledge Base 3

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch).  
playsAirGuitar(butch):- listens2music(butch).
```

```
?- playsAirGuitar(butch).  
yes  
?-
```



# Expressing Disjunction

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch).  
playsAirGuitar(butch):- listens2music(butch).
```

```
happy(vincent).  
listens2music(butch).  
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).  
playsAirGuitar(butch):- happy(butch); listens2music(butch).
```

# Prolog and Logic

- Clearly, Prolog has something to do with logic...

	Prolog	Logic
Implication	$A :- B$	$B \rightarrow A$
Conjunction	$A, B$	$A \wedge B$
Disjunction	$A; B$	$A \vee B$

- Use of inference (modus ponens)
- Negation (?)

# Knowledge Base 4

woman(mia).

woman(jody).

woman(yolanda).

loves(vincent, mia).

loves(marsellus, mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).



# Prolog Variables

```
woman(mia).
```

```
woman(jody).
```

```
woman(yolanda).
```

```
loves(vincent, mia).
```

```
loves(marsellus, mia).
```

```
loves(pumpkin, honey_bunny).
```

```
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
```

# Variable Instantiation

woman(mia).

woman(jody).

woman(yolanda).

loves(vincent, mia).

loves(marsellus, mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).

?- woman(X).

X=mia

# Asking Alternatives

woman(mia).

woman(jody).

woman(yolanda).

loves(vincent, mia).

loves(marsellus, mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).

?- woman(X).

X=mia;

# Asking Alternatives

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

```
loves(vincent, mia).  
loves(marsellus, mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).
```

```
?- woman(X).  
X=mia;  
X=jody
```

# Asking Alternatives

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

```
loves(vincent, mia).  
loves(marsellus, mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).
```

```
?- woman(X).  
X=mia;  
X=jody;  
X=yolanda
```



# Asking Alternatives

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

```
loves(vincent, mia).  
loves(marsellus, mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).
```

```
?- woman(X).  
X=mia;  
X=jody;  
X=yolanda;  
no
```

# Knowledge Base 4

woman(mia).

woman(jody).

woman(yolanda).

loves(vincent, mia).

loves(marsellus, mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).

?- loves(marsellus,X), woman(X).

# Knowledge Base 4

woman(mia).

woman(jody).

woman(yolanda).

loves(vincent, mia).

loves(marsellus, mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).

?- loves(marsellus,X), woman(X).

X=mia

yes

?-

# Knowledge Base 4

woman(mia).

woman(jody).

woman(yolanda).

loves(vincent, mia).

loves(marsellus, mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).

?- loves(pumpkin,X), woman(X).

# Knowledge Base 4

woman(mia).

woman(jody).

woman(yolanda).

loves(vincent, mia).

loves(marsellus, mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).

?- loves(pumpkin,X), woman(X).

no

?-

# Knowledge Base 5

loves(vincent,mia).

loves(marsellus,mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).



# Knowledge Base 5

loves(vincent,mia).

loves(marsellus,mia).

loves(pumpkin, honey\_bunny).

loves(honey\_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).

?- jealous(marsellus,W).

# Knowledge Base 5

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

```
?- jealous(marsellus,W).  
W=vincent  
?-
```

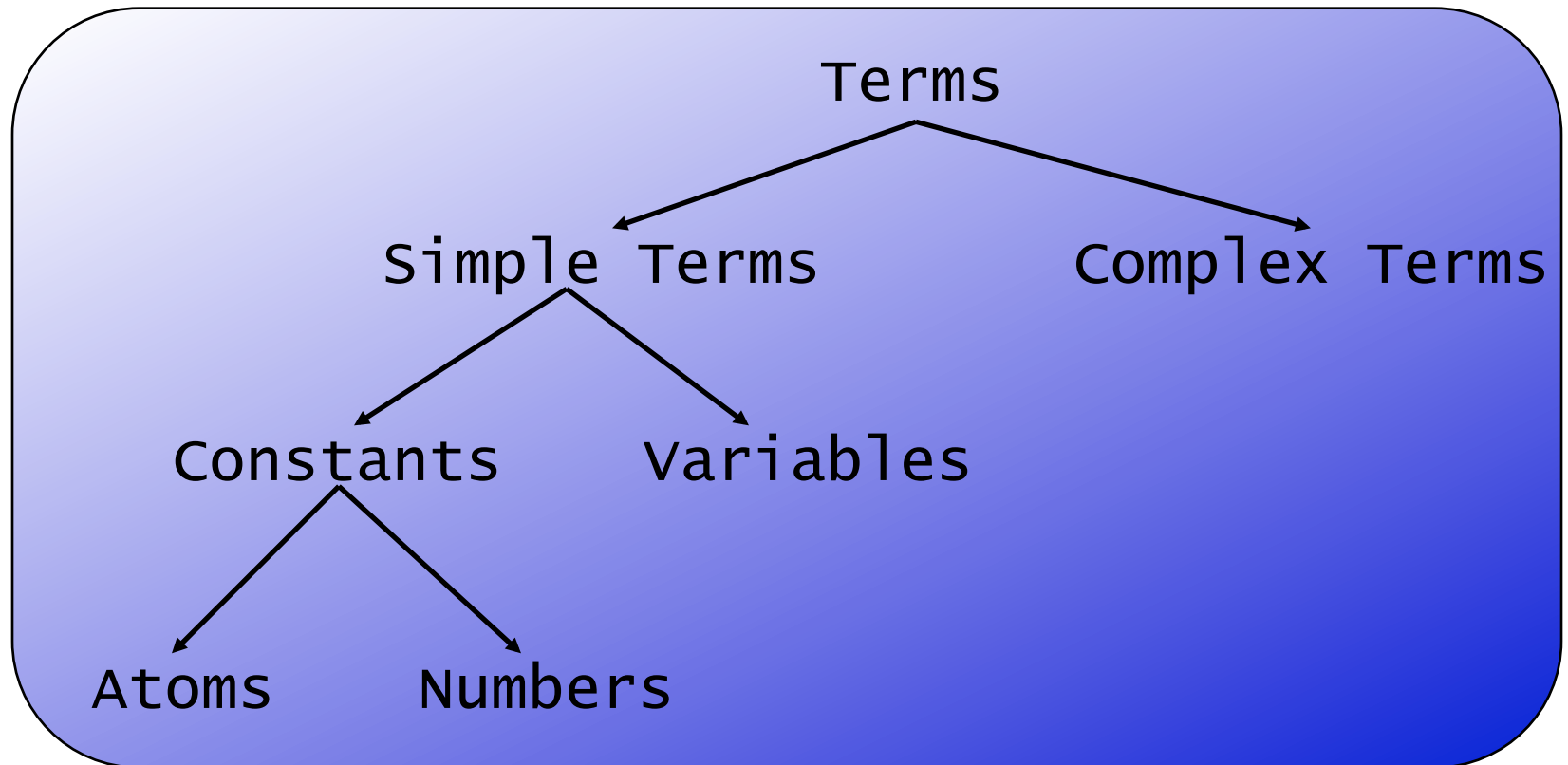


# Syntax of Prolog

---

- Q: What exactly are facts, rules and queries built out of?
- A: Prolog terms

# Prolog terms



# Atoms

---

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with a lowercase letter
  - *Examples:* **butch**, **big\_kahuna\_burger**, **playGuitar**

# Atoms

---

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with a lowercase letter
  - *Examples:* **butch**, **big\_kahuna\_burger**, **playGuitar**
- An arbitrary sequence of characters enclosed in single quotes
  - *Examples:* **'Vincent'**, **'Five dollar shake'**, **'@\$%'**

# Atoms

---

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with a lowercase letter
  - *Examples:* **butch**, **big\_kahuna\_burger**, **playGuitar**
- An arbitrary sequence of characters enclosed in single quotes
  - *Examples:* **'Vincent'**, **'Five dollar shake'**, **'@\$%'**
- A sequence of special characters
  - *Examples:* **:**, **,**, **;**, **.**, **:-**

# Numbers

---

- Integers:

12, -34, 22342

- Floats:

34573.3234, 0.3435

# Variables

---

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with either an uppercase letter or an underscore

- Examples:

**X, Y, Variable, Vincent, \_tag**

# Complex Terms

---

- Atoms, numbers and variables are building blocks for **complex terms**
- Complex terms are built out of a **functor** directly followed by a sequence of **arguments**
  - Arguments are put in round brackets, separated by commas
  - The functor must be an atom



# Examples of complex terms

---

- Examples we have seen before:
  - playsAirGuitar(jody)
  - loves(vincent, mia)
  - jealous(marsellus, W)
- Complex terms inside complex terms:
  - hide(X, father(father(father(butch))))

# Arity

---

- The number of arguments a complex term has is called its arity

- Examples:

**woman(mia)** is a term with arity 1  
**loves(vincent,mia)** has arity 2  
**father(father(butch))** arity 1

# Arity is important

---

- You can define two predicates with the same functor but with different arity
- Prolog would treat this as two different predicates!
- In Prolog documentation, arity of a predicate is usually indicated with the suffix "/" followed by a number to indicate the arity

# Example of Arity

```
happy(yolanda).  
listens2music(mia).  
listens2music(yolanda):- happy(yolanda).  
playsAirGuitar(mia):- listens2music(mia).  
playsAirGuitar(yolanda):- listens2music(yolanda).
```

- This knowledge base defines
  - happy/1
  - listens2music/1
  - playsAirGuitar/1