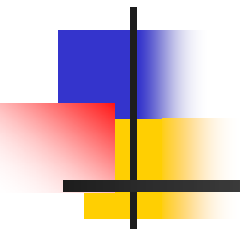
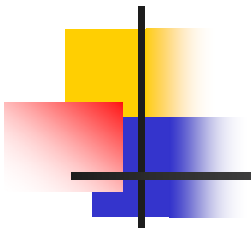


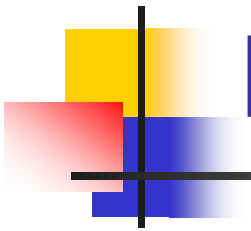
# Prolog

A Gentle Introduction



# An Example

- “Prolog” is all about programming in logic.
  - Socrates is a man.
  - All men are mortal.
  - Therefore, Socrates is mortal.



# Facts, rules, and queries

- Fact: Socrates is a man.
  - man(socrates).
- Rule: All men are mortal.
  - mortal(X) :- man(X).
- Query: Is Socrates mortal?
  - mortal(socrates).

# Running Prolog I

- Create your "database" (program) in any editor.
- Save it as *text only,* with a **.P** extension
- Here's the complete "program":

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

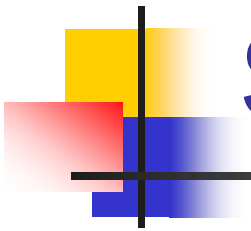
# Running Prolog II

- Prolog is *completely interactive.*
- Begin by invoking the Prolog interpreter.
- Then load your program.
- Then, ask your question at the prompt:
  - `? mortal(socrates).`
- Prolog responds:
  - `Yes`

# Syntax 1: Structures or Terms

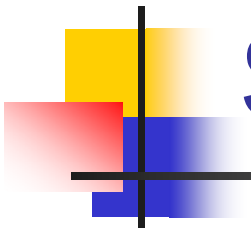
- Example structures:
  - sunshine
  - man(socrates)
  - path(garden, south, sundial)
- <structure> ::= <name>| <name> ( <arguments> )
- <arguments> ::= <argument> | <argument> , <arguments>

# Syntax II: Base Clauses



- Base clauses are like simple facts.
- Example base clauses:
  - loves(john, mary).
  - loves(mary, bill).
- <base clause> ::= <structure> .

# Syntax III: Nonbase Clauses

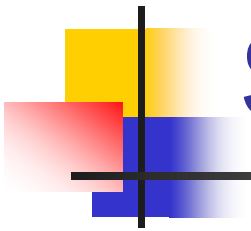


- Non-base clauses are like rules.
- Example non-base clauses:
  - mortal(X) :- man(X).
  - mortal(X) :- woman(X)
  - happy(X) :- healthy(X), wealthy(X), wise(X).
- <nonbase clause> ::= <structure> :- <structures>.
- <structures> ::= <structure> | <structures> , <structure>

# Syntax IV: Predicates

- A predicate is a collection of clauses with the same *functor* and *arity*.
  - loves(john, mary).
  - loves(mary, bill).
  - loves(chuck, X) :- female(X), rich(X).
- <predicate> ::= <clause> | <predicate> <clause>
- <clause> ::= <base clause> | <nonbase clause>

# Syntax V: Programs



- A program is a collection of predicates.
- Predicates can be in any order.
- Predicates are used in the order in which they occur.

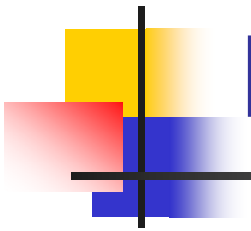
# Syntax VI: Assorted details

- Variables begin with a capital letter or an underscore:
  - X, Socrates, _result
- Atomic symbols do not begin with a capital letter:
  - x, socrates
- Other atomic symbols must be enclosed in single quotes:
  - 'Socrates'
  - 'C:/My Documents/examples.P'

# Syntax VII: Assorted details

- In a quoted atom, a single quote must be quoted or backslashed: 'Can''t, or won\'t?'
- /* Comments are like this */
- Prolog allows some infix operators, such as **:-** (turnstile) and **,** (comma). These are syntactic sugar for the functors ':-' and ','.
- These are equivalent:
  - ':-'(mortal(X), man(X)).
  - mortal(X) :- man(X).

# Backtracking



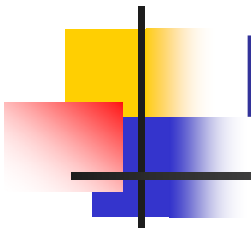
- loves(chuck, X) :- female(X), rich(X).
- female(jane).
- female(mary).
- rich(mary).

*Now, suppose we ask: ?* loves(chuck, X).

- female(X) = female(jane), X = jane.
- rich(jane) fails.
- female(X) = female(mary), X = mary.
- rich(mary) succeeds.

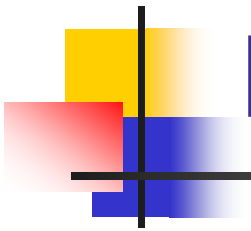
# Additional answers

```
female(jane).
female(mary).
female(susan).

 ?- female(X).
 X = jane ;
 X = mary
Yes
```



# Readings

- loves(chuck, X) :- female(X), rich(X).
- *Declarative reading*: Chuck loves X if X is female and rich.
- Approximate *procedural reading*: To find an X that Chuck loves, first find a female X, then check that X is rich.
- Declarative readings are almost always preferred.

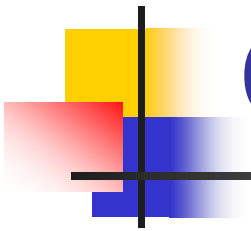


# Non-monotonicity

Prolog's facts and rules can be changed at anytime.

```
assert(man(plato)).
assert((loves(chuck,X) :- female(X), rich(X))).

retract(man(plato)).
retract((loves(chuck,X) :- female(X), rich(X))).
```

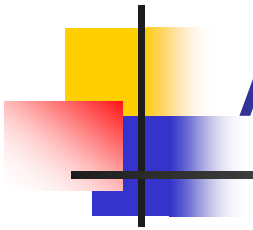
# Common problems



- Capitalization is *extremely* important!
- No space between a functor and its argument list:

  man(socrates), *not* man (socrates).
- Don't forget the period! (But you can put it on the next line.)

# A Simple Prolog Model



- Imagine prolog as a system which has a database composed of two components:

- FACTS - statements about true relations which hold between particular objects in the world. For example:
  - parent(adam,able): adam is a parent of able
  - parent(eve,able): eve is a parent of able
  - male(adam): adam is male.

- RULES - statements about true relations which hold between objects in the world which contain generalizations, expressed through the use of variables. For example, the rule
  - father(X,Y) :- parent(X,Y), male(X). might express: for any X and any Y, X is the father of Y if X is a parent of Y and X is male.

# Nomenclature and Syntax

- A prolog rule is called a **clause**.
- A clause has a head, a neck and a body:

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

*head* *neck* *body*

- The head is a rule's conclusion.
- The body is a rule's premise or condition.

Note:

- read :- as IF
- read , as AND
- a . marks the end of input

# Prolog Database

```
parent(adam,able)
parent(adam,cain)
male(adam)
...
```

**Facts** comprising the "extensional database"

```
father(X,Y) :- parent(X,Y),
male(X).
sibling(X,Y) :- ...
```

**Rules** comprising the "intensional database"

# Extensional vs. Intensional

- The terms *extensional* and *intensional* are borrowed from the language philosophers use for *epistemology*.
  - *Extension* refers to whatever *extends*, i.e., "is quantifiable in space as well as in time".
  - *Intension* is an antonym of extension, referring to "that class of existence which may be quantifiable in time but not in space."

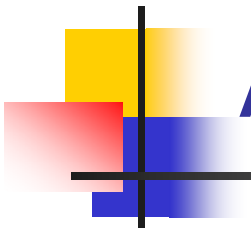
- NOT *intentional* with a "t", which has to do with "will, volition, desire, plan, ..." For KBs and DBs we use
  - • *extensional* to refer to that which is explicitly represented (e.g., a fact), and
  - • *intensional* to refer to that which is represented abstractly, e.g., by a rule of inference.

```
parent(adam,able)
parent(adam,cain)
male(adam)
...
```

```
father(X,Y) :- parent(X,Y),
male(X).
sibling(X,Y) :- ...
```

**Facts** comprising the "extensional database"

**Rules** comprising the "intensional database"



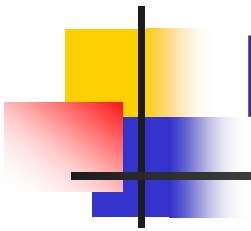
# A Simple Prolog Session

```
| ?- assert(parent(adam,able)).
Yes

| ?- assert(parent(eve,able)).
Yes

| ?- assert(male(adam)).
Yes

| ?- parent(adam,able).
Yes
```

# How to Satisfy a Goal

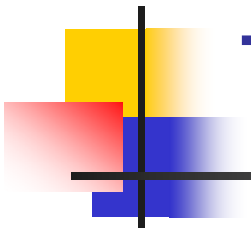
Here is an informal description of how Prolog satisfies a goal (like father(adam,X)). Suppose the goal is G:

- **Conjunction:** if G = P,Q then first satisfy P, carry any variable bindings forward to Q, and then satisfy Q.
- **Disjunction:** if G = P;Q then satisfy P. If that fails, then try to satisfy Q.
- **Negation:** if G = not(P) then try to satisfy P. If this succeeds, then fail and if it fails, then succeed.
- **Simple goal:** if G is a simple goal, then:
  - look for a fact in the DB that unifies with G.
  - look for a rule whose conclusion unifies with G and try to satisfy its body.



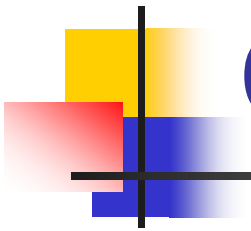
# Note

- Two basic conditions are **true**, which always succeeds, and **fail**, which always fails.
- A comma (,) represents conjunction (and) and a semi-colon represents disjunction (or), as in:
  - grandParent(X,Y) :- grandFather(X,Y); grandMother(X,Y).
- There's no real distinction between rules and facts, which are just rules whose bodies are the trivial condition true. These are equivalent:
  - parent(adam,cain)
  - parent(adam,cain) :- true.
- Goals can be posed with any combination of variables and constants:
  - *parent(cain,able)* - Is Cain Able's parent?
  - *parent(cain,X)* - Who is a child of Cain?
  - *parent(X,cain)* - Who is Cain a child of?
  - *parent(X,Y)* - What two people have a parent/child relationship?



# Terms

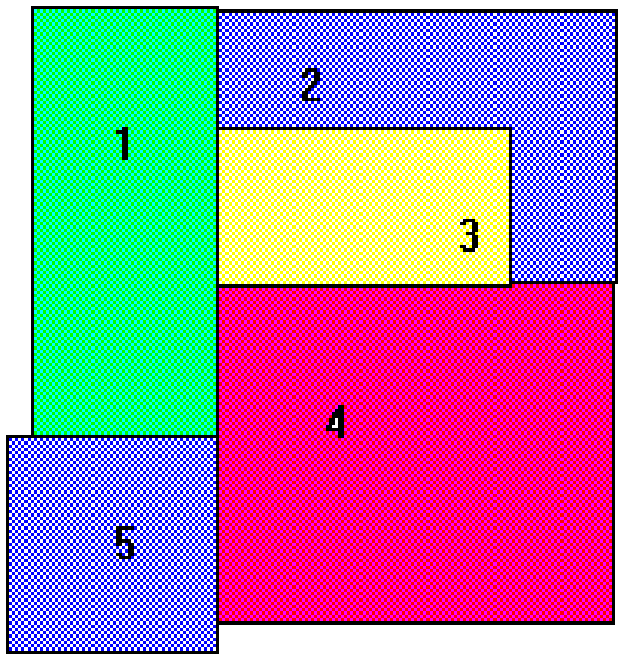
- The term is the basic data structure in Prolog.
- The term is to Prolog what the s-expression is to Lisp.
- A term is either:
  - a constant - e.g.
    - john , 13, 3.1415, +, 'a constant'
  - – a variable - e.g.
    - X, Var, _, _foo
  - – a compound term - e.g.
    - part(arm,body)
    - part(arm(john),body(john))

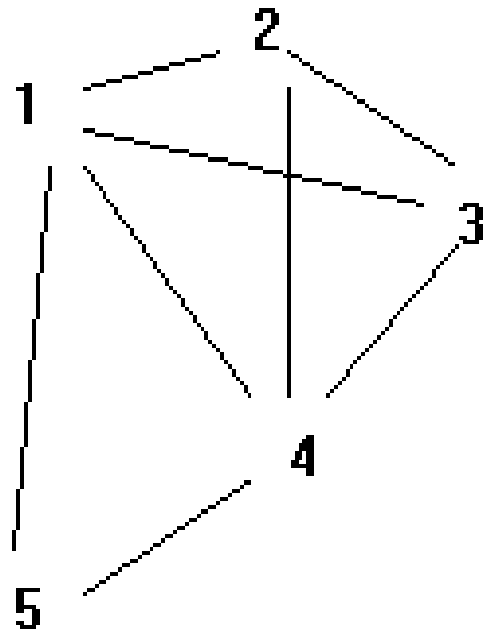


# Compound Terms

- A compound term can be thought of as a relation between one or more terms: *part_of(finger,hand)* and is written as:
  - The relation name (called the principle functor) which must be a constant.
  - An open parenthesis
  - The arguments - one or more terms separated by commas.
  - A closing parenthesis.
- The number of arguments of a compound term is called its **arity**.

# Map Coloring Example




**XSB Demo of the Example**