TUTORIAL 6

CSCI3230 (2019-2020 First Term)

By Ran WANG (rwang@cse.cuhk.edu.hk)

Outline

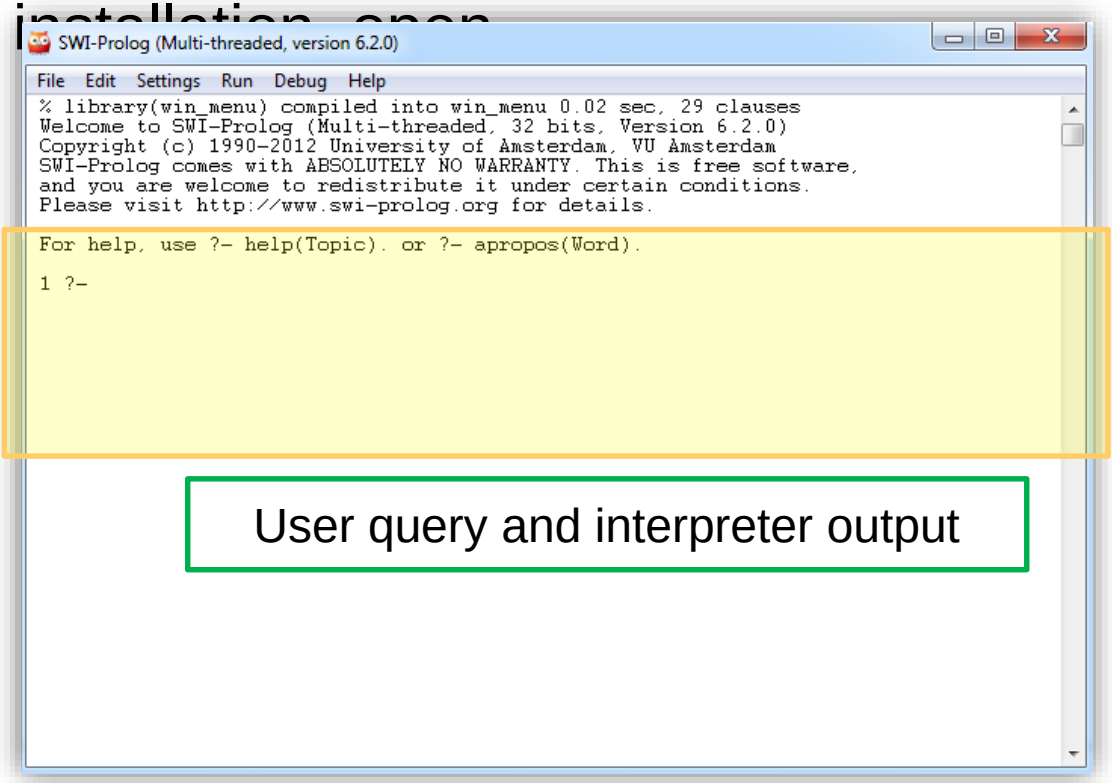
- SWI-Prolog
  - IDE
  - Your first Prolog program
  - Modify your Prolog file
  - Debug Prolog
  - For MaxOS
- Short Review with Guided Practice
- Programming Exercises

SWI-PROLOG

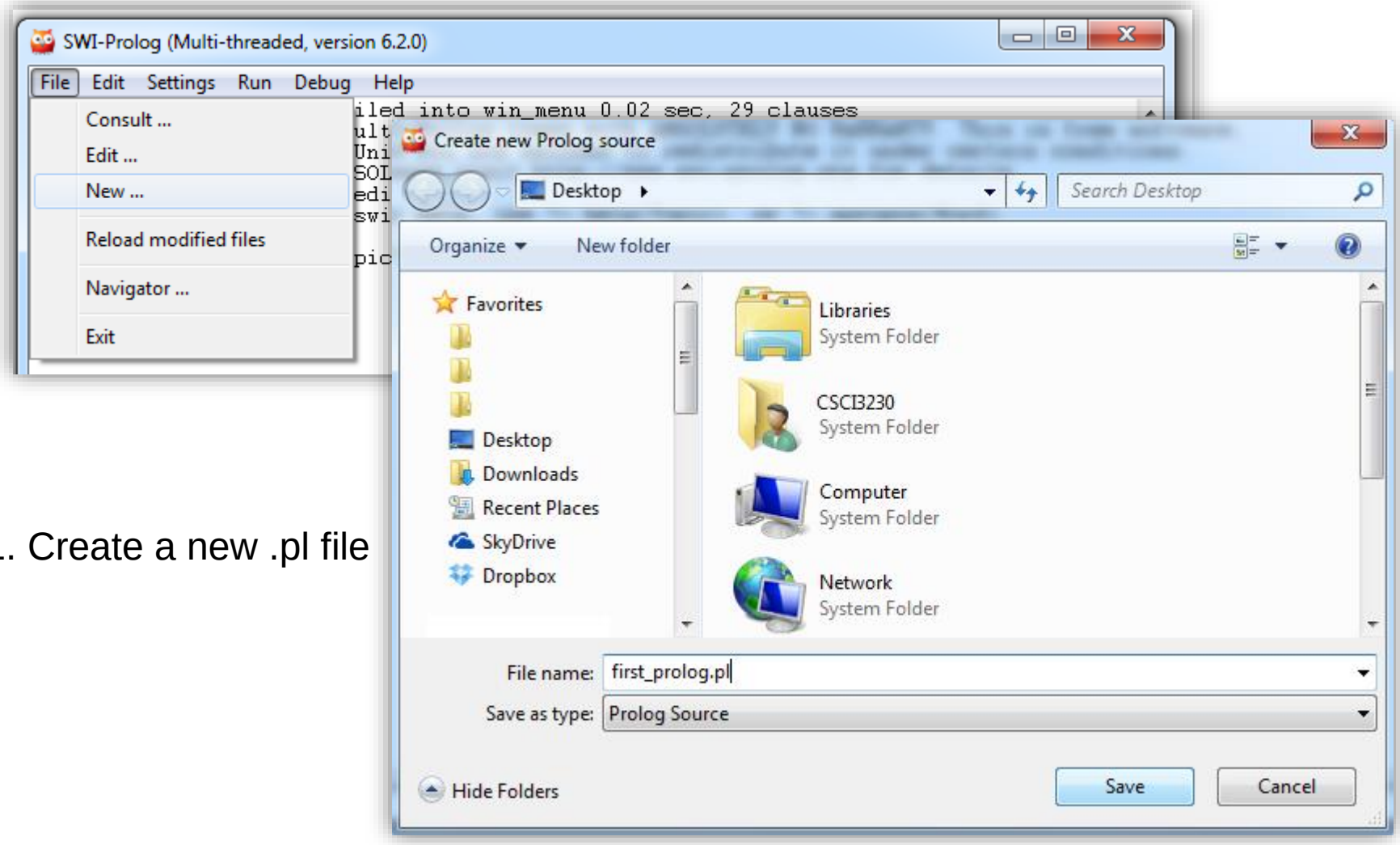
1. IDE
2. Your first Prolog program
3. Modify your Prolog file
4. Debug Prolog
5. For MaxOS

IDE

1. Download 
   - http://www.swi-prolog.org/download/stable (Official)
   - http://portableapps.com/apps/development/swi-prolog_portable (Portable) (For computers in Lab)
   - Version 5.10.4 http://www.swi-prolog.org/download/stable?show=all
2. After installation, open

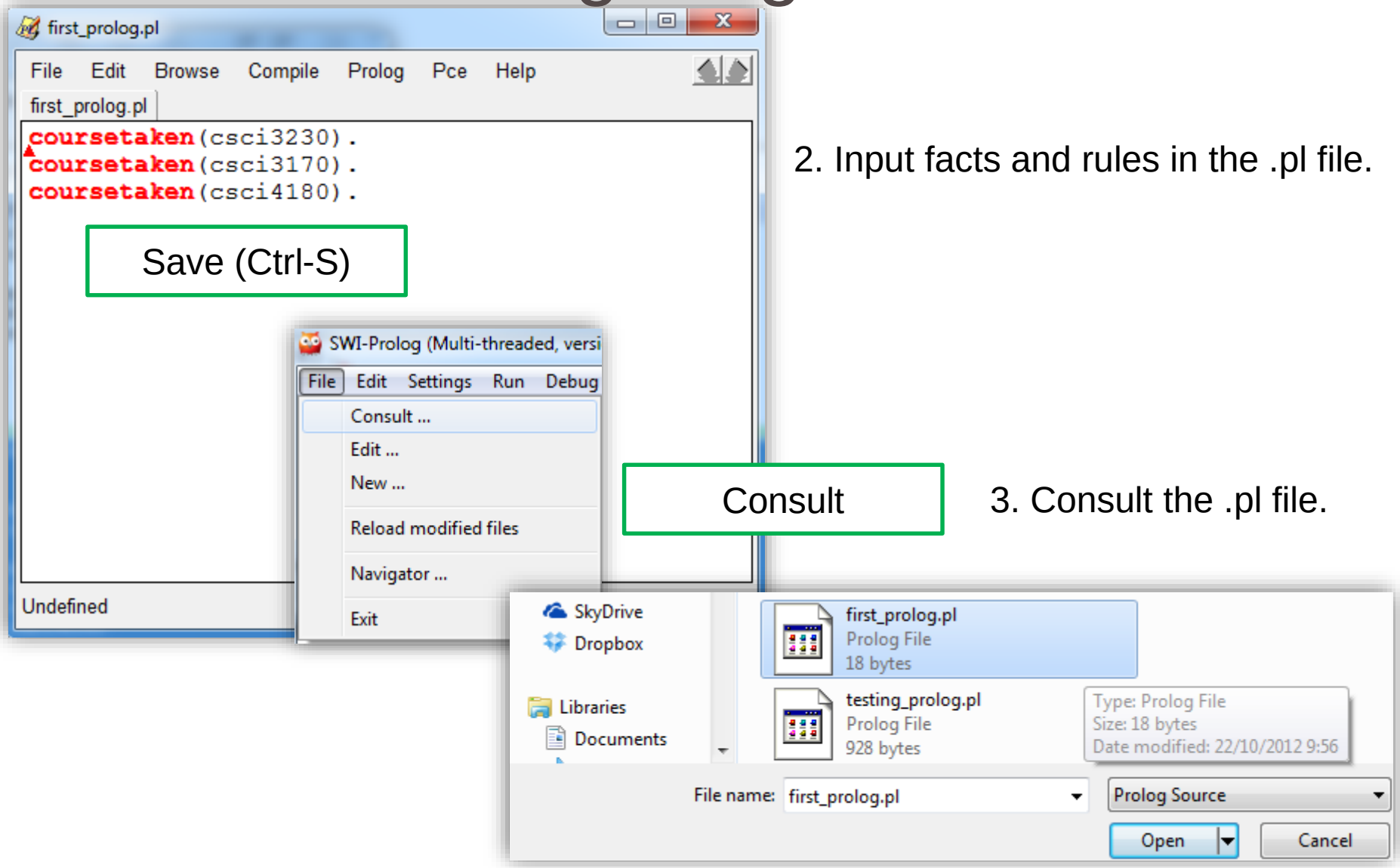


Your First Prolog Program



1. Create a new .pl file

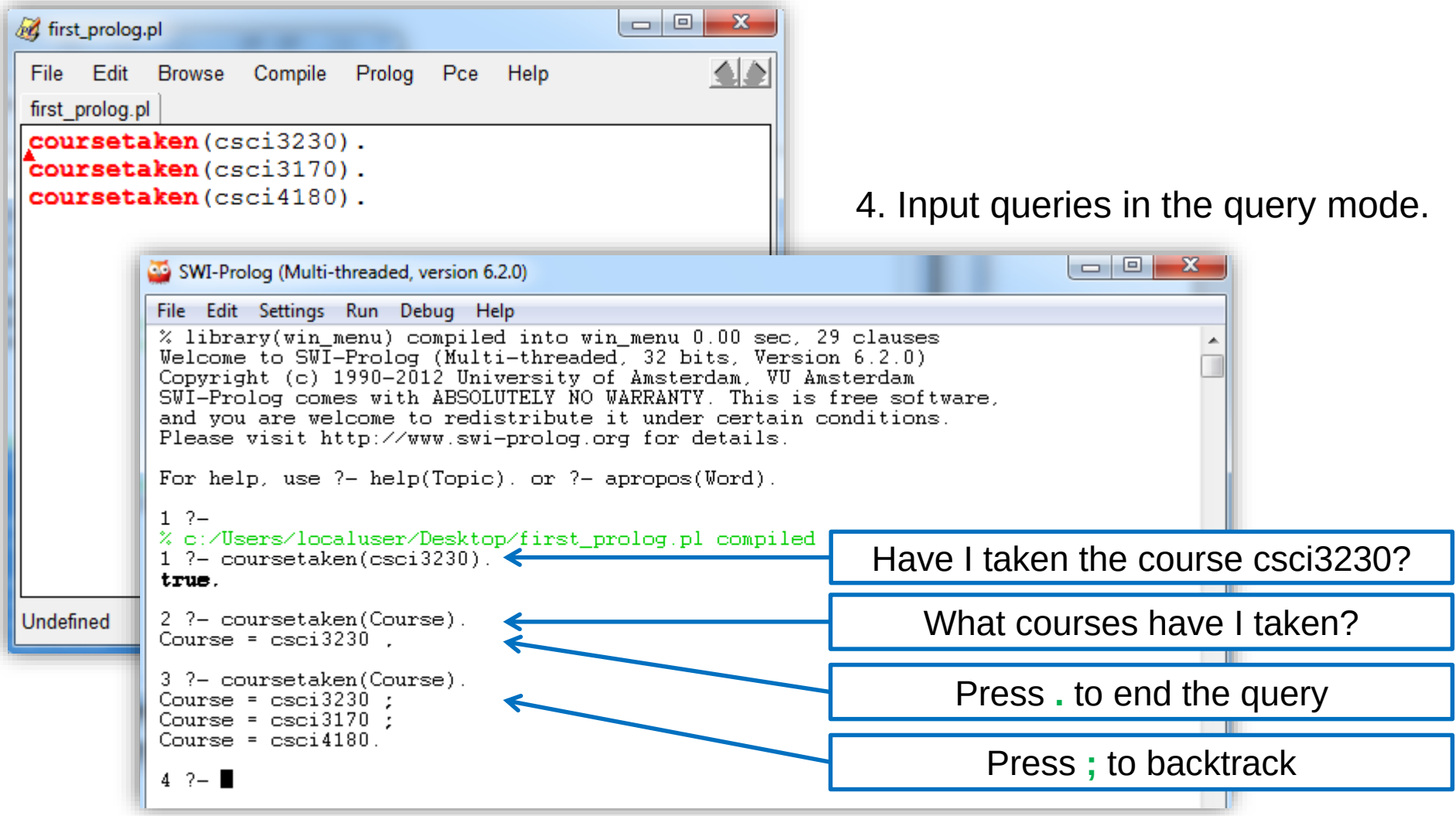
Your First Prolog Program



2. Input facts and rules in the .pl file.

3. Consult the .pl file.

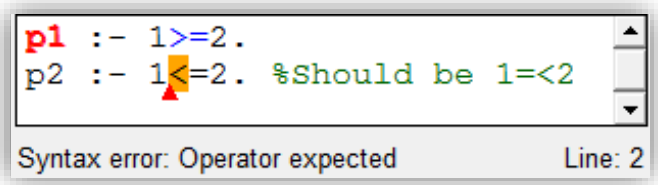
Your First Prolog Program



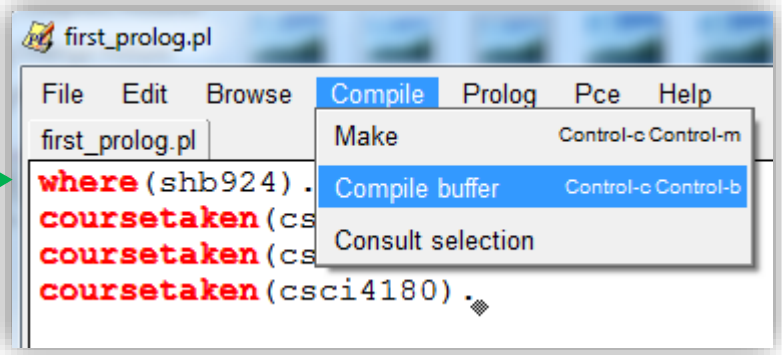
4. Input queries in the query mode.

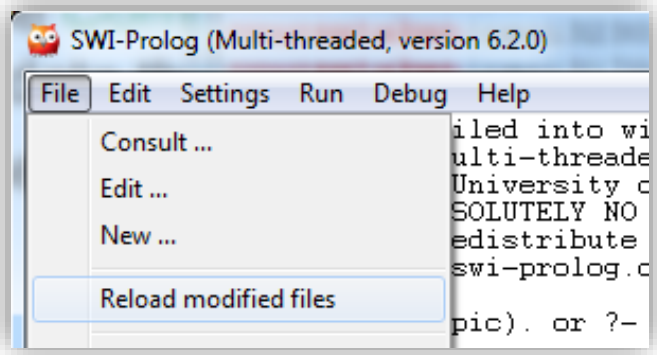
Modify Your Prolog File

- Editor
  - Use Compile buffer to consult the whole Prolog file again and update the Prolog database
  - Use Consult selection to consult the Prolog file partially
  - Highlight when there is syntax error
- Query Window
  - Use Reload modified files to reload the whole Prolog file (useful when you are using other editors)

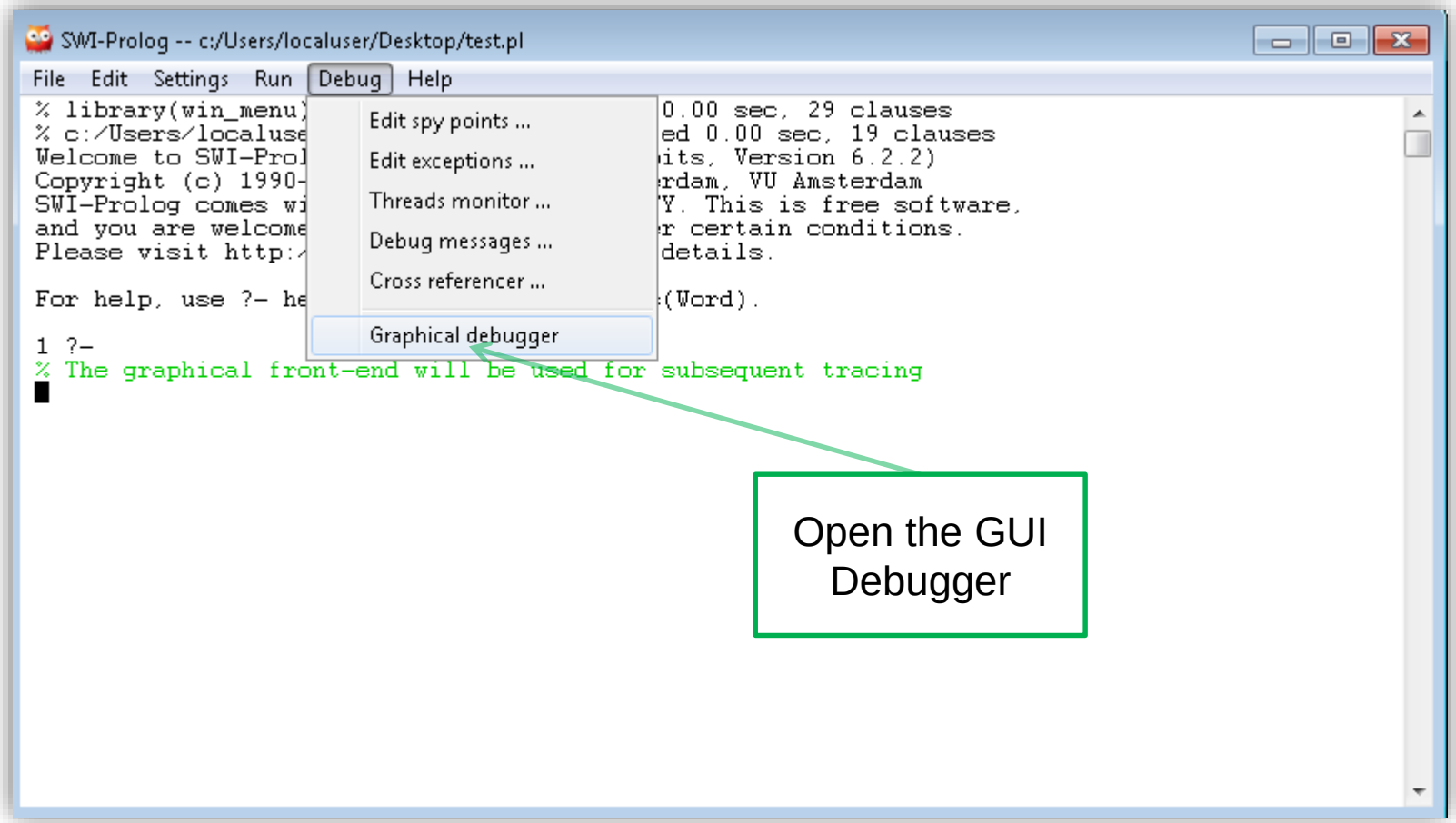


New fact





Debug Prolog



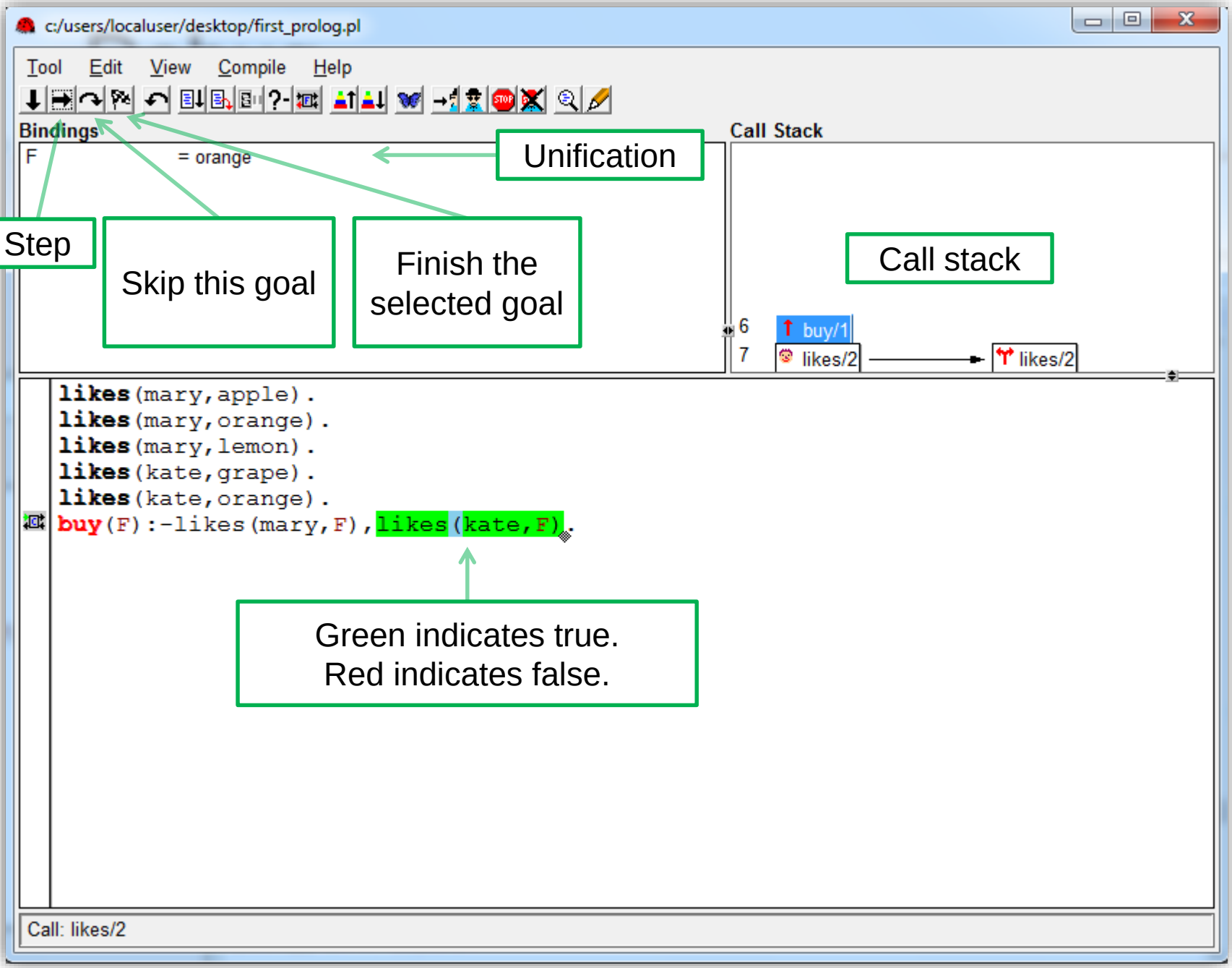
For more details:
http://www.swi-prolog.org/pldoc/man?section=debugoverview
http://www.swi-prolog.org/pldoc/man?section=debugger
http://www.cs.bham.ac.uk/~pjh/prolog_course/using_sicstus/debugging_sicstus.html

Debug Prolog

Example

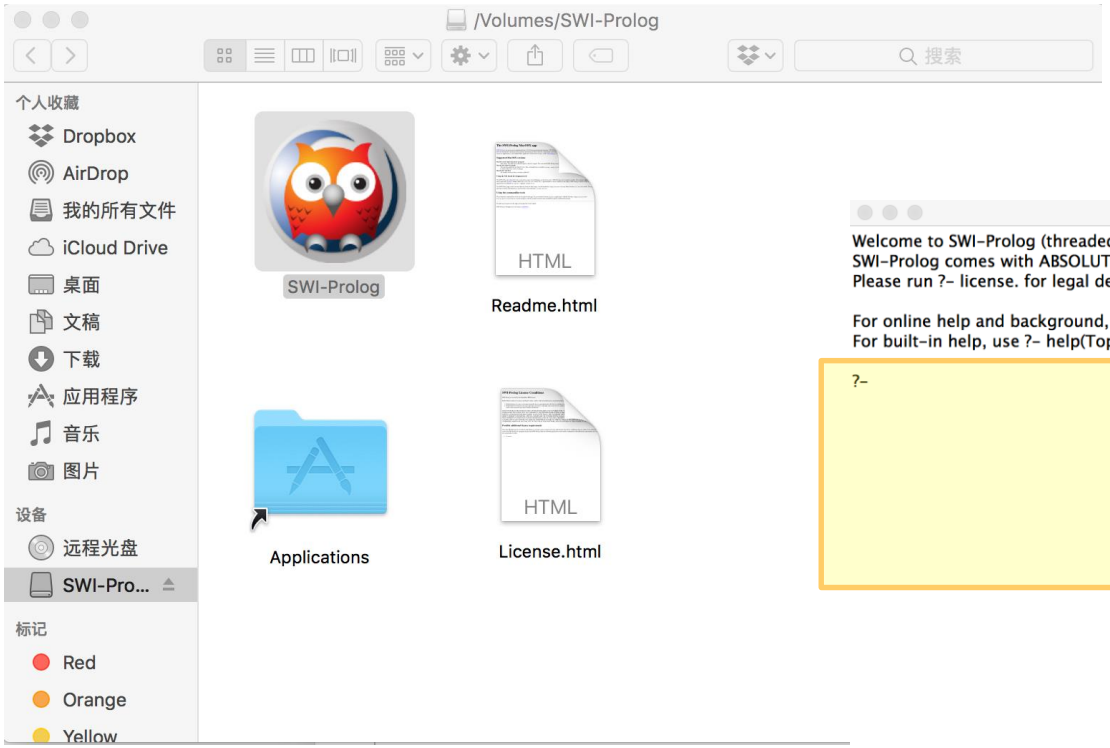
```
likes(mary,apple).
likes(mary,orange).
likes(mary,lemon).
likes(kate,grape).
likes(kate,orange).
buy(F):-likes(mary,F),likes(kate,F).
```

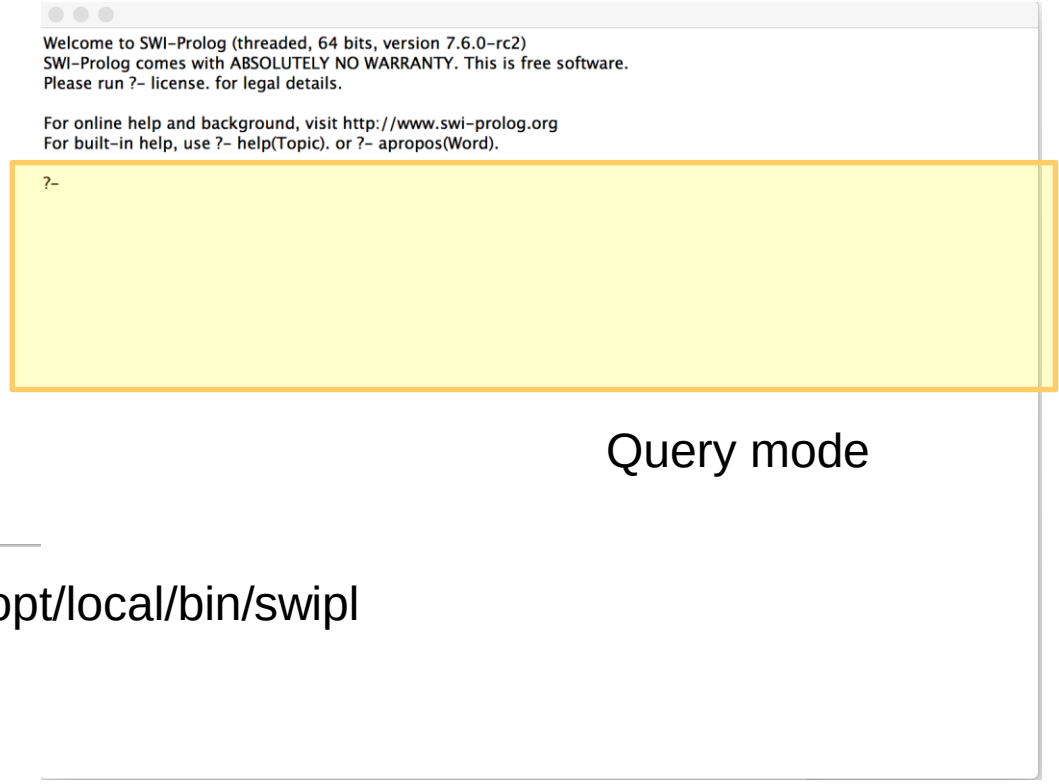
```
?- spy(likes/2). %Start the Graphical Debugger first
true.
[debug] ?- buy(F).
%See the Graphical Debugger
F = orange.
[trace] ?- notrace. %Exit the trace mode
true.
[debug] ?- nodebug. %Exit the debug mode
true.
```


c:/users/localuser/desktop/first_prolog.pl
Tool Edit View Compile Help
Bindings
F = orange
Call Stack
6 buy/1
7 likes/2
likes/2
likes(mary,apple).
likes(mary,orange).
likes(mary,lemon).
likes(kate,grape).
likes(kate,orange).
buy(F):-likes(mary,F),likes(kate,F).
Call: likes/2
Unification
Step
Skip this goal
Finish the selected goal
Call stack
Green indicates true.
Red indicates false.

For MacOS

- Installation



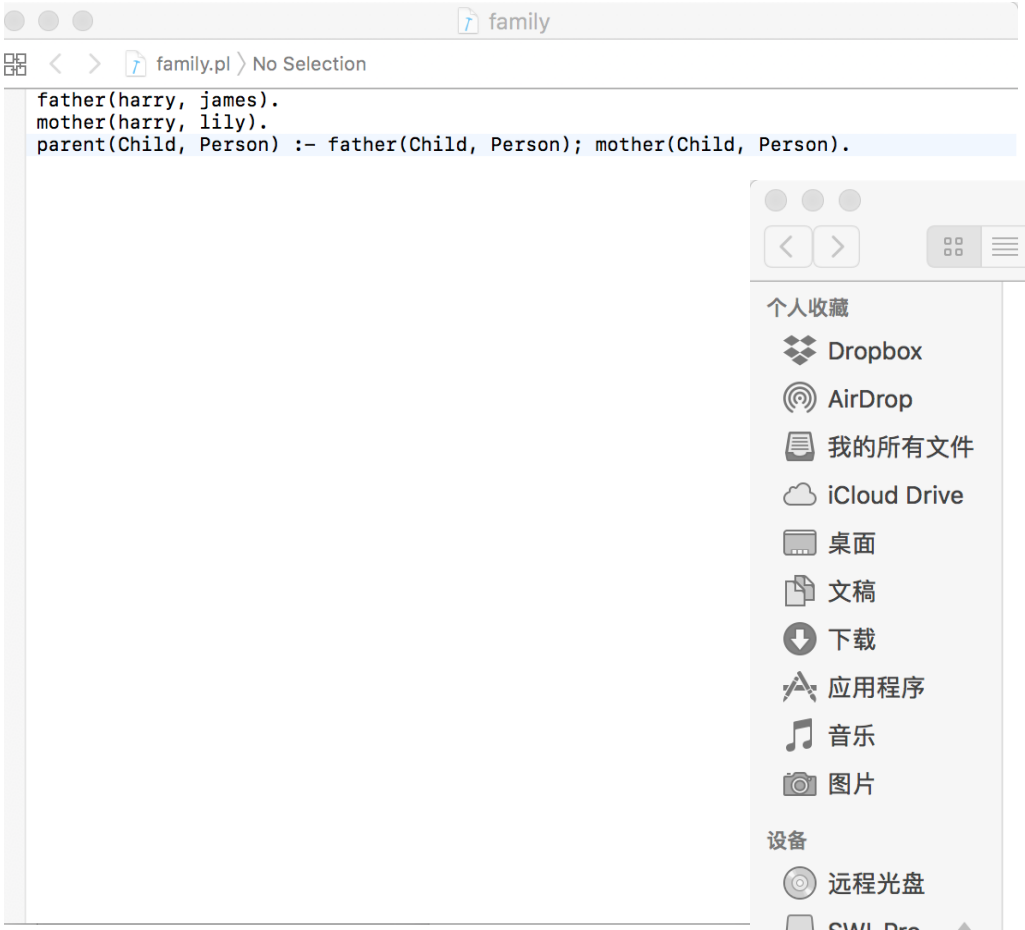


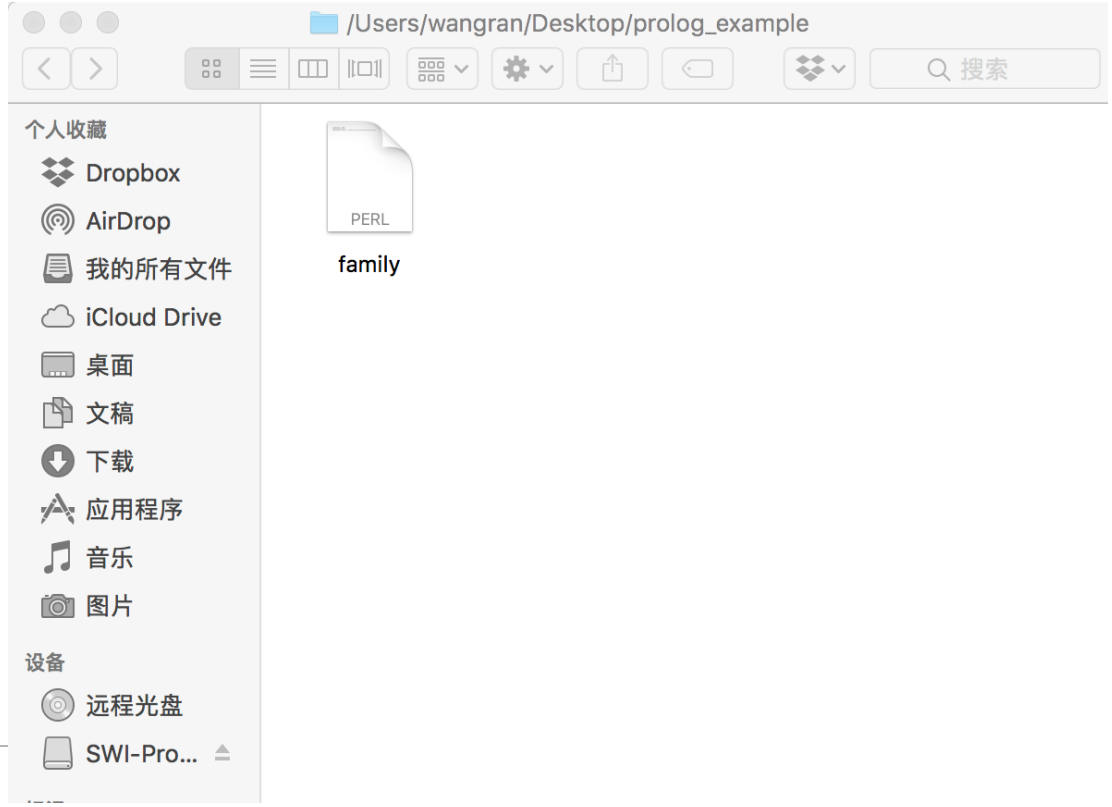
Query mode

Default location: /opt/local/bin/swipl

For MacOS

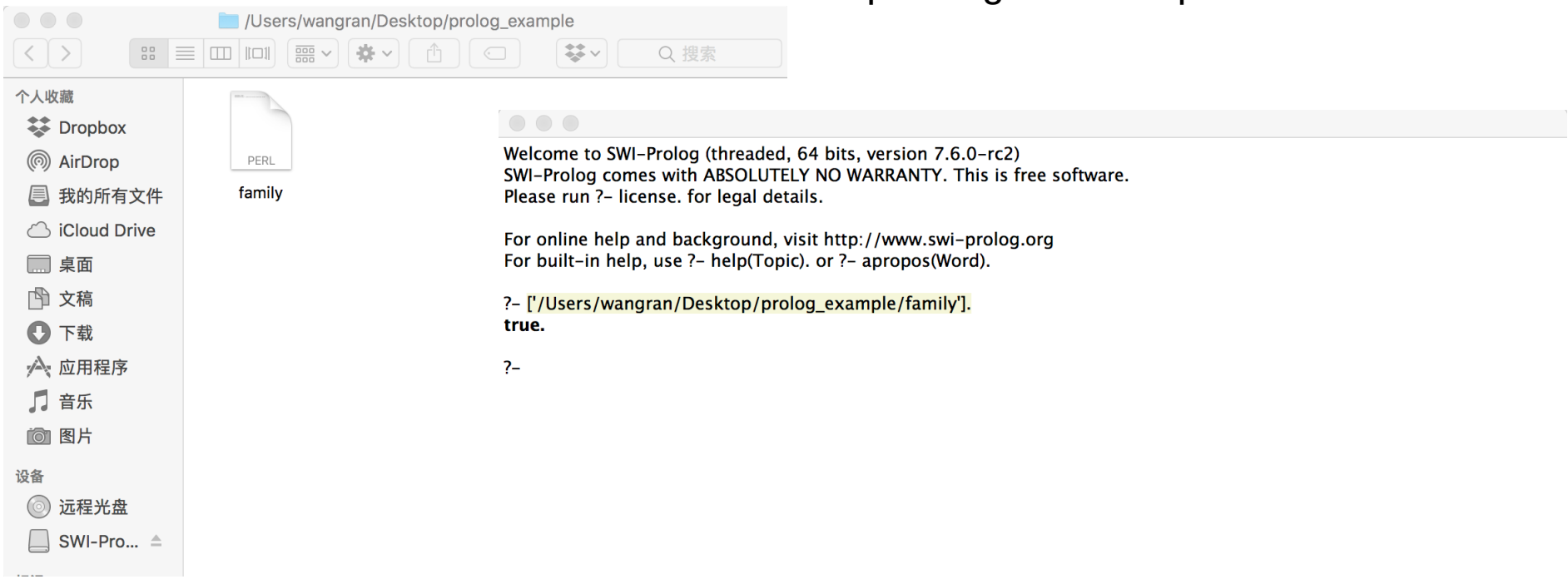
- Create a .pl file





For MacOS

- Consult
  - ?- consult(likes). %Load likes.pl from the current folder
  - ?- [‘/opt/local/lib/swipl-5.6.0/demo/likes’]
    %Load likes.pl using absolute path



For MacOS

- Query

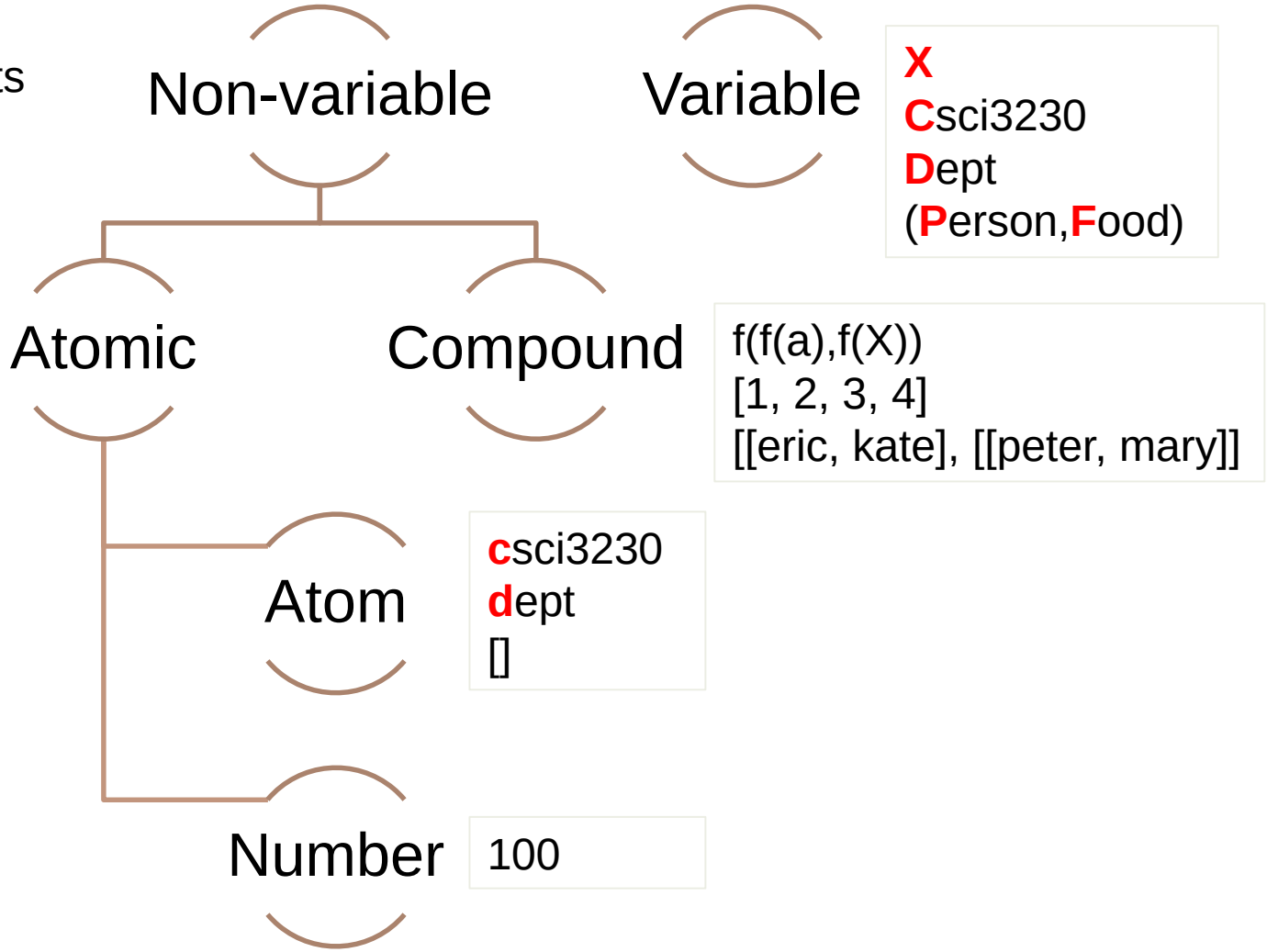
```
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.0-rc2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ['/Users/wangran/Desktop/prolog_example/family'].
true.

?- parent(harry, james).
true .

?- mother(harry, mary).
false.

?- mother(harry, lily).
true.

?-
```

GUIDED PRACTICE

1. Prolog basics
2. Membership function
3. Append two lists
4. Find the maximum number
5. Prime test

1. Prolog Basics

Terms: Data Objects



1. Prolog Basics: Fact

Example 1

```
%Fact
leo_is_handsome.
%Fact_with_Arguments
handsome(leo).
```

```
?- leo_is_handsome.
true.
?- leo_is_not_handsome.
ERROR: Undefined procedure: leo_is_not_handsome/0
?- handsome(leo).
true.
?- handsome(tom).
false.
?- handsome(X).
X = leo.
```

Statements

- **FACTS** states a predicate holds between terms.

Example 3

```
father(harry,james).     %Fact 1
mother(harry,lily).      %Fact 2
```

```
?- father(harry,james).
true.
```

1. Prolog Basics: Variable

Example 1

```
%Fact_with_Arguments_including_atom_and_number
age(leo, 26).
age(sun, 26).
age(peter, 27).
```

```
?- age(leo, 26).
true.
?- age(leo, 16).
false.
?- age(X, 26).
X = leo;
X = sun.
```

1. Prolog Basics: Variable

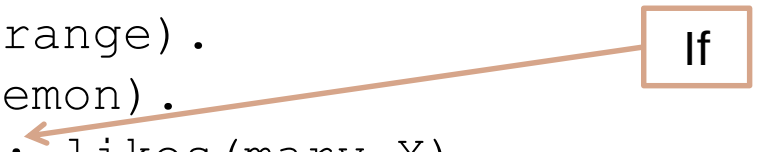
Example 1

```
%Fact_with_Arguments_including_atom_and_number
age(leo, 26).
age(sun, 26).
age(peter, 27).
```

```
?- age(X, Y).
X = leo,
Y = 26 ;
X = sun,
Y = 26 ;
X = peter,
Y = 27.
```

1. Prolog Basics: Rule

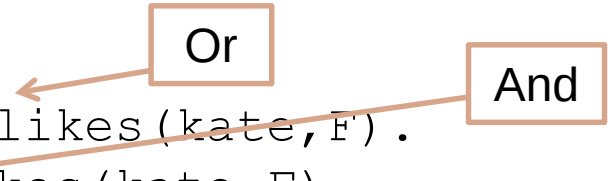
```
%Rules_Examples
likes(mary,apple).
likes(mary,orange).
likes(mary,lemon).
likes(tom,X):-likes(mary,X).
```




```
?- likes(mary,apple).
true.
?- likes(tom,apple).
true.
?- likes(tom,banana).
false.
```

1. Prolog Basics: Rule

```
%Rules_Examples
likes(mary,apple).
likes(mary,orange).
likes(mary,lemon).
likes(tom,X):-likes(mary,X).
likes(kate,grape).
likes(kate,orange).
taste(F):-likes(mary,F);likes(kate,F).
buy(F):-likes(mary,F),likes(kate,F).
```




```
?- taste(apple).
true.
?- buy(apple).
false.
?- buy(orange).
true.
```

1. Prolog Basics: Rule

- **RULES** defines the relationship

r(…) :- conditions for r(…) be true.

Meaning	Predicate Calculus	PROLOG
And	∧	,
Or	∨	;
if	←	:-
Not	¬	not

1. Prolog Basics: Arithmetic

- No arithmetic is carried out until commanded by *is* predicate
- Operators: +, -, *, /

Example

```
plus(X,Y,R):- R is X+Y.
```

```
?- plus(3,4,R).
R = 7.
```

Example

```
minus(X,Y,R):- R is X-Y.
```

```
?- plus(4,3,R).
R = 1.
```

Compound Term (a.k.a. Structure)

$$f(t_1,t_2,\ldots,t_n)$$

- f : functor
- T_i : terms
- Arity : number of sub-terms

Example 1

```
likes(fruit(lemon,who(tom,alex))).%Fact
likes(fruit(apple,who(ben,fred))).%Fact
```

```
?- likes(fruit(apple,who(ben,fred))).
true.
```

Compound Term: List

$$f(t_1,t_2,\ldots,t_n)$$

- f : functor
- T_i : terms
- Arity : number of sub-terms

Example 2

```
ls(.(a,.(b,.(c,[])))).  %Fact, this creates a list.
```

Some systems do not allow a list as a fact.

```
?- ls([a|[b,c]]).
true. %fact, different representation
?- ls([a,b,c]).
true. %fact, different representation
```

1. Prolog Basics: List

Example:a list of programming language

```
[c, cpp, csharp, java, php, python, ruby, lisp, prolog, sql]
```

```
%List_Examples
p([H|T],H,T).
o([A,B|T],A,B,T).
```

```
?- [X|Y] = [c,java,php,python,ruby]
X = c,
Y = [java, php, python, ruby]
?- [X,Y|Z] = [c,java,php,python,ruby]
X = c,
Y = java,
Z = [php, python, ruby]
?- p([c,java,php,python,ruby],H,T).
H = c,
T = [java, php, python, ruby].
```

1. Prolog Basics: List

Example: a list of programming language

```
[c, cpp, csharp, java, php, python, ruby, lisp, prolog, sql]
```

```
%List_Examples
p([H|T],H,T).
o([A,B|T],A,B,T).
```

```
?- p([c],H,T).
H = c,
T = [].
?- p([],H,T).
false.
?- o([c,java,php],A,B,C).
A = c,
B = java,
C = [php].
?- o([c],A,B,C).
false.
```

2. Membership function

- Given an item and a list, find if there exists an item in the list. E.g. Given ‘a’ and [a,b,c,d], does ‘a’ exist in the list?

Code

```
member(H,[H|T]). %Better version: member(H,[H|_]).
member(O,[H|T])  %Better version: member(O,[_|T]).
      :- member(O,T).
```

```
?- member(apple,[apple,orange,banana])
true.
?- member(pear,[apple,orange,banana])
false.
?- member(X,[apple,orange,banana])
Try this!
```

3. Append two lists

- Given two lists, append them and return the product
- E.g. [a,b] + [c,d] → [a,b,c,d]

Code

```
append([],Y,Y).
append([H|T],Y,[H|Z]):-
        append(T,Y,Z).
```

```
?- append([],[a,b,c],Z).
Z = [a,b,c].
?- append([a,b],[c,d],Z).
Z = [a,b,c,d].
?- append(X,Y,[a,b,c,d]).
Try this!
```

4. Find the maximum number

- Given a list of number, find the maximum one
- E.g. [-1.1, -0.7, 0, 0.5, 1.1, 2.2] → 2.2

Code

```
max(X,[X]).
max(H,[H|T]):- max(N,T), H >= N.
max(N,[H|T]):- max(N,T), N > H.
```

```
?- max(X, [-5, 2, -3, 1.1, 6.7]).
X = 6.7.
?- max(X,[2.2, 2.2]).
X = 2.2.
?- max(X,[]).
Try this!
```

4. Find the maximum number (cont.)

- Given a list of number, find the maximum one
- E.g. [-1.1, -0.7, 0, 0.5, 1.1, 2.2] → 2.2

Code

```
max(X,[X]).
max(H,[H|T]):- max(N,T), H >= N.
max(N,[H|T]):- max(N,T), N > H.
num_list([-1,2,3.3]).
num_list([-5,6,2,0.3]).
num_list([0.2,0.5,-0.1]).
max_list(X,L):- num_list(L),max(X,L).
```

```
?- findall(X,max_list(X,M),L).
L = [3.3, 6, 0.5].
```

4. Find the maximum number (cont.)

- Given a list of number, find the maximum one
- E.g. [-1.1, -0.7, 0, 0.5, 1.1, 2.2] → 2.2

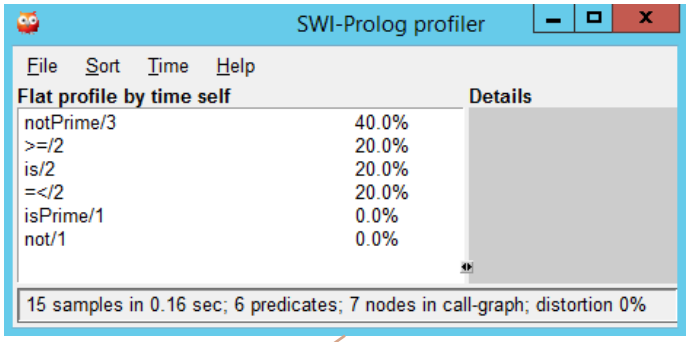
Code

```
max(X,[X]).
max(H,[H|T]):- max(N,T), H >= N.
max(N,[H|T]):- max(N,T), N > H.
num_list([-1,2,3.3]).
num_list([-5,6,2,0.3]):-!.
num_list([0.2,0.5,-0.1]).
max_list(X,L):- num_list(L),max(X,L).
```

```
?- findall(X,max_list(X,M),L).
Try this!
```

5. Prime Test

- Given an integer, test if it is prime
- E.g. 17 → Prime, 120 → Not Prime



Code

```
notPrime(S,E,N):- S >= 2, S =< E, 0 is N mod S.
notPrime(S,E,N):- S >= 2, S =< E, R is S+1, notPrime(R,E,N).
isPrime(N):- M is N-1, not(notPrime(2,M,N)).
```

```
?- isPrime(17).
true.
?- isPrime(120).
false.
?- profile(isPrime(123121)).
true.
?- time(isPrime(123121)).
% 738,721 inferences, 0.141 CPU in 0.141 seconds (100% CPU, 5253127 Lips)
true.
?- isPrime(1274893457).
Try this!
Press "Run" -> "Interrupt" -> Press "h" -> Press "b" to stop.
```

PROGRAMMING EX

1. Sum of numbers
2. Greatest common divisor
3. Length of list
4. Mean values for each column
5. Towers of Hanoi
6. Traverse a tree
7. Fill in a 3x3 puzzle
8. Propose an interesting question for yourself!

Programming Exercise 1

- Define `sum(N,R)`
- If `N` is a positive number,
  - `R` = 1+2+3+4+5+6+…+`N`
- Else if `N` is a list of number,
  - `R` is the sum of the list of number.

Example

```
?- sum(3,R).
R=6.
?- sum([1,5,7],R).
R=13.
```

Programming Exercise 2

- Define `gcd(X,Y,Z)` iff
  - `Z` (integer) is the Greatest Common Divisor of positive integers `X` and `Y`
- `X` and `Y` are input, `Z` is output

Example

```
?- gcd(2,6,Z).
Z=2.
?- gcd(18,24,Z).
Z=6.
```

Programming Exercise 3

- Define `listlen(L,N)` to find the length of the list `L`
- You have to use accumulator to store the intermediate result.

Example

```
?- listlen([1,2,3,4,5],N).
N=5.
?- listlen([3,6,1],N).
N=3.
```

Programming Exercise 4

- Given you have a m×n matrix, e.g. $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
- Define `mean(M,NRow,NCol,R)` to find the mean value of each column in the matrix, where `M`, `NRow` and `NCol` is a list of number, the number of row, and the number column respectively. The result is stored in `R`.

Example

```
?- mean([1,4,2,5,3,6],2,3,R).
R=[2.5,3.5,4.5].
```

Programming Exercise 5

- Tower of Hanoi: Move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk.
- Write `hanoi(N,X,Y,Z)`, where `N` is the number of disks on the left peg, to produce a series of instructions, e.g. “move a disk from left to middle”.

Example

```
?- Hanoi(3,left,right,middle).
Move top disk from left to right
Move top disk from left to middle
Move top disk from right to middle
Move top disk from left to right
Move top disk from middle to left
Move top disk from middle to right
Move top disk from left to right
```

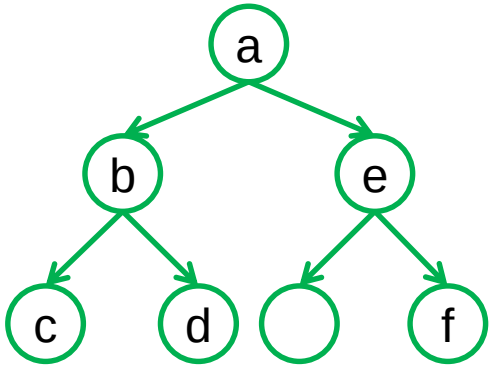
Programming Exercise 6

- Binary tree representation:
  - `tree(Q,L,R)`
  - `L` left child
  - `R` right child
  - `Q` is the term on the node
- Leaves:
  - `leaf(Q)`
- Empty subtree:
  - `nil`

- Now want pre-order traversal
  - root → left subtree→ right subtree
  - print out the terms on the leaves
- Define function `pr_tree(T,X)`

Example

```
?- pr_tree(tree(a, tree(b,leaf(c),leaf(d)),
tree(e,nil,leaf(f))),X).
X=[a,b,c,d,e,f].
```



Programming Exercise 7

- Fill in a 3x3 grid with number from 1-9 with each number appearing once only such that the sum of every row and every column are the same. Write a `puzzle(P)` to find the solution. The answer `P` is a list, e.g. [1 2 3 4 5 6 7 8 9].

Example

```
?- puzzle(P).
P = [1, 5, 9, 6, 7, 2, 8, 3, 4];
P = [1, 5, 9, 8, 3, 4, 6, 7, 2];
P = [1, 6, 8, 5, 7, 3, 9, 2, 4];
P = [1, 6, 8, 9, 2, 4, 5, 7, 3];
…
```

Hints

1. `sum([H,T],R) :- …`
   `sum(N,R) :- …`
2. gcd(a,0) = a
   gcd(a,b) = gcd(b,a mod b)
3. `listlen([],0).`
   `listlen([H|T],N) :- …`
4. May need to use accumulator
5. https://www.cpp.edu/~jrfisher/www/prolog_tutorial/2_3.html
6. `pr_tree(tree(Q,L,R),X) :-`
   `pr_tree(L,M),pr_tree(R,N), …`
7. Test and Generate Model: Generate the permutations one by one and test whether such permutation satisfies the requirement. If the test fails, Prolog will perform backtracking.

Reference

- Tutorials
  - http://www.cs.toronto.edu/~hojjat/384w10/PrologTutorial2.pdf
- Reference manual of SWI-Prolog
  - http://www.swi-prolog.org/pldoc/refman/
- More advanced Prolog
  - The Craft of Prolog by Richard A. O'Keefe