TUTORIAL 6

CSCI3230 (2013-2014 First Term)

Presented by Antonio SZE-TO (hyszeto@cse.cuhk.edu.hk)
Q&A by Paco WONG (pkwong@cse.cuhk.edu.hk)
Qin CAO (qcao@cse.cuhk.edu.hk)

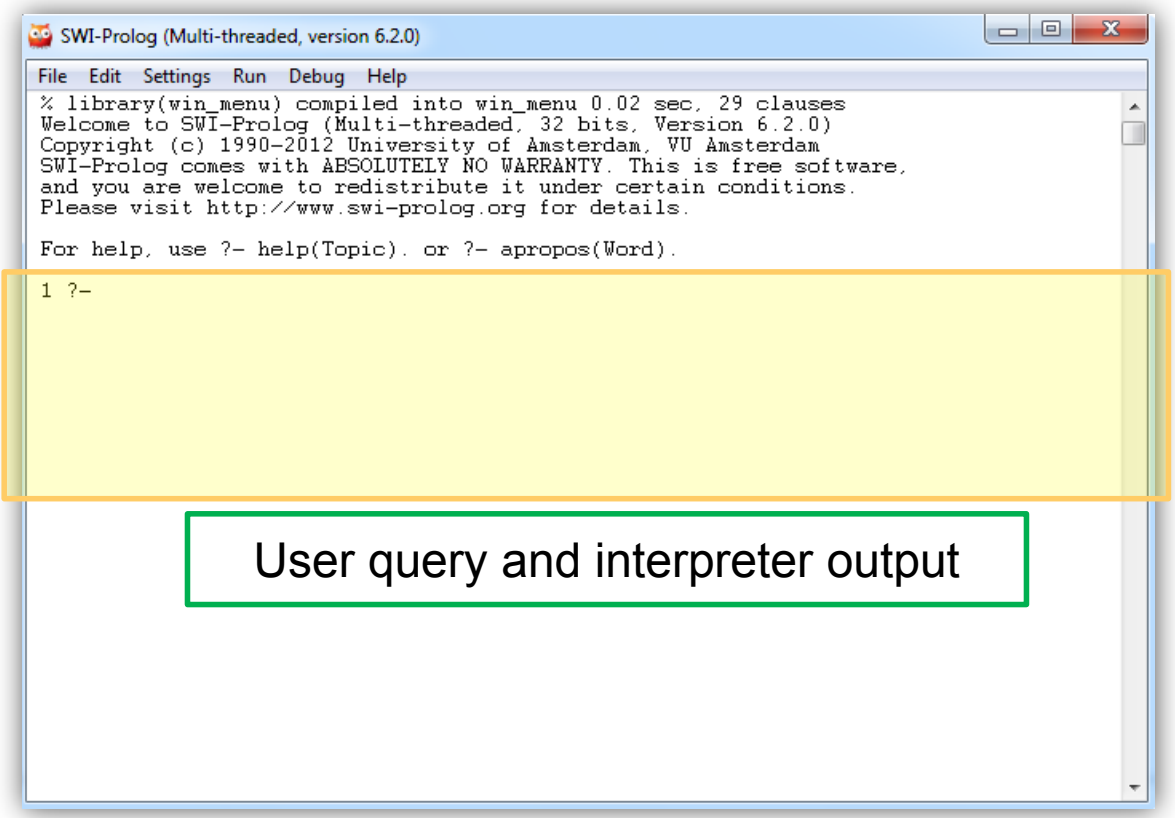
Hands on Lab
@SHB924

Outline

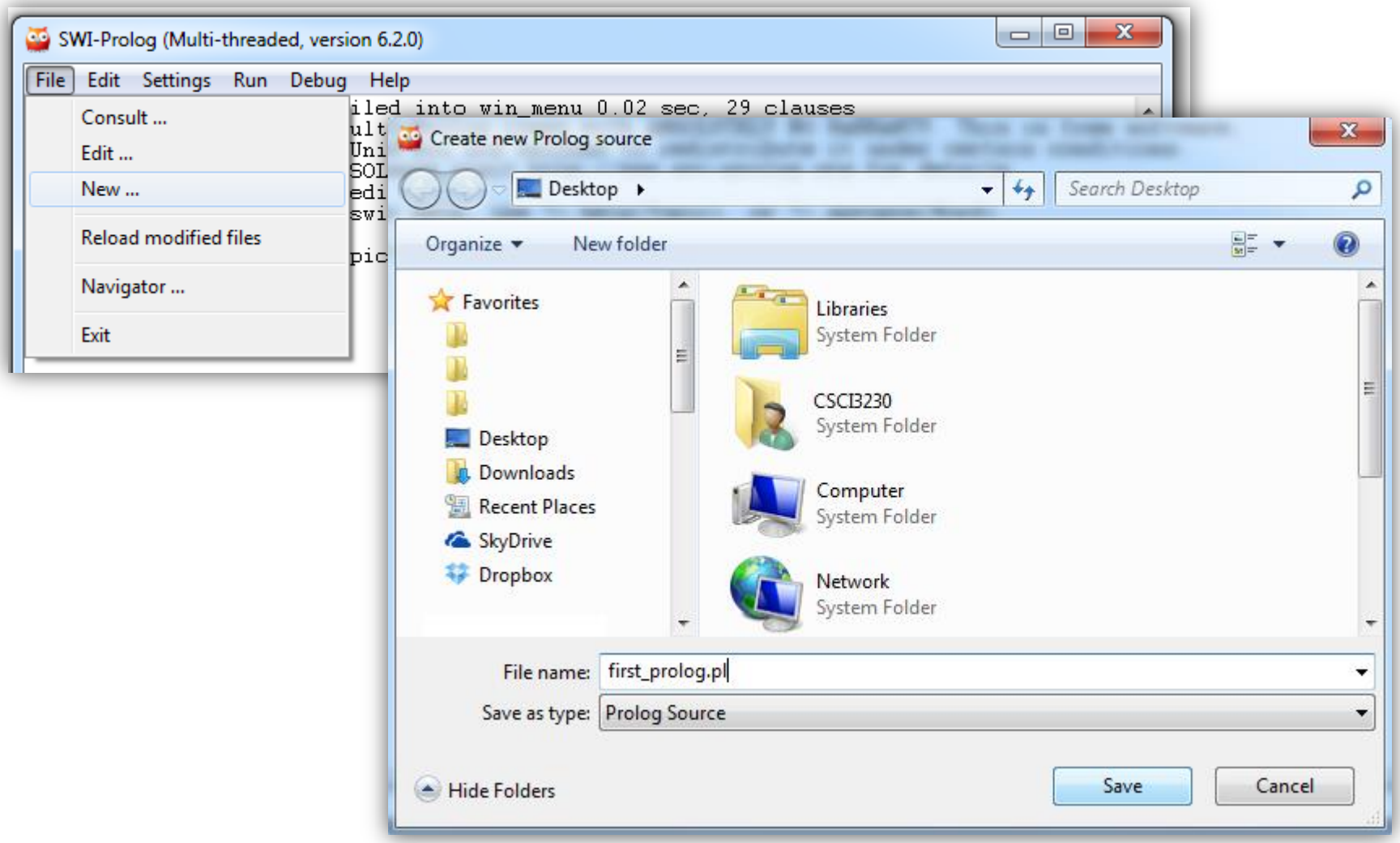
- SWI-Prolog
  - Setup
  - IDE
- Short Review with Guided Practice
  - Debug
- Programming Exercises

SWI-Prolog

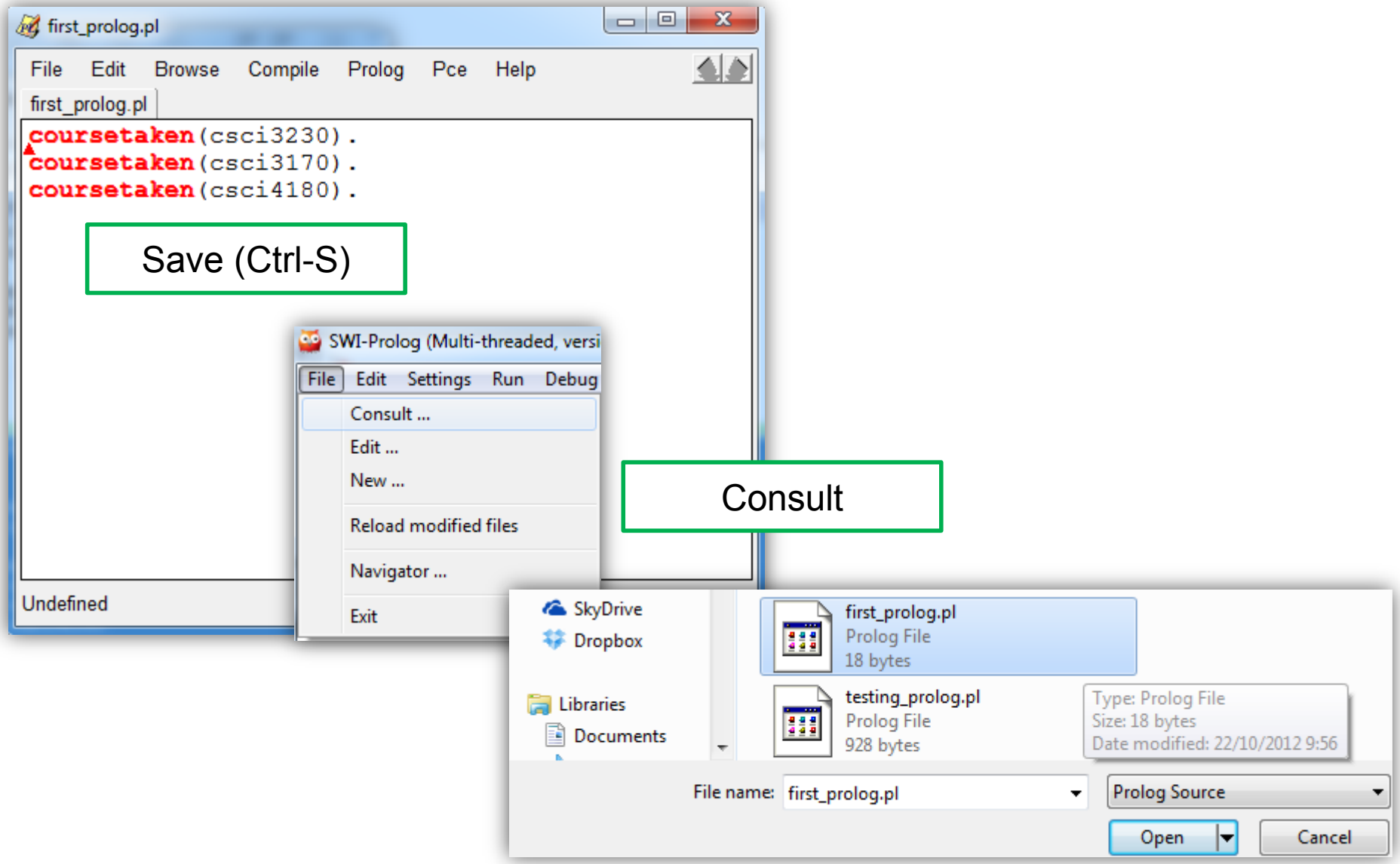
1. Download
   - http://www.swi-prolog.org/download/stable (Official)
   - http://portableapps.com/apps/development/swi-prolog_portable (Portable)
2. After installation, open



Your First Prolog Program



Create and Consult



Query

first_prolog.pl

File Edit Browse Compile Prolog Pce Help

first_prolog.pl

```
coursetaken(csci3230).
coursetaken(csci3170).
coursetaken(csci4180).
```

Undefined

SWI-Prolog (Multi-threaded, version 6.2.0)

File Edit Settings Run Debug Help

```
% library(win_menu) compiled into win_menu 0.00 sec, 29 clauses
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.2.0)
Copyright (c) 1990-2012 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
% c:/Users/localuser/Desktop/first_prolog.pl compiled
1 ?- coursetaken(csci3230).
true.

2 ?- coursetaken(Course).
Course = csci3230 .

3 ?- coursetaken(Course).
Course = csci3230 ;
Course = csci3170 ;
Course = csci4180.

4 ?-
```

Have I taken the course csci3230?

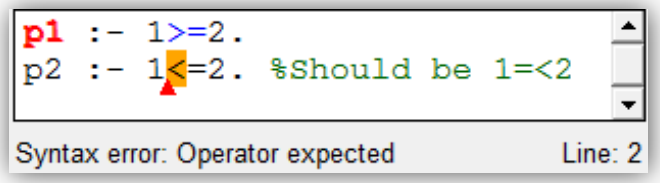
What courses have I taken?

Press . to end the query

Press ; to backtrack

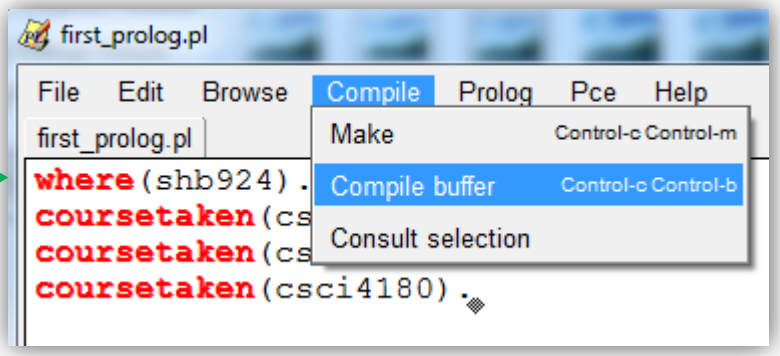
Modify Prolog File

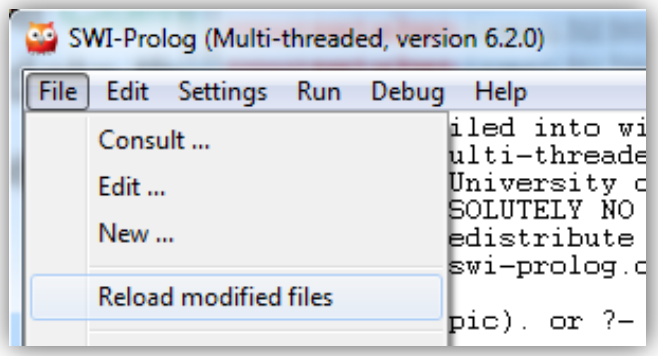
- Editor
  - Use Compile buffer to consult the whole Prolog file again and update the Prolog database
  - Use Consult selection to consult the Prolog file partially
  - Highlight when there is syntax error



- Query Window
  - Use Reload modified files to reload the whole Prolog file (useful when you are using other editors)

New fact



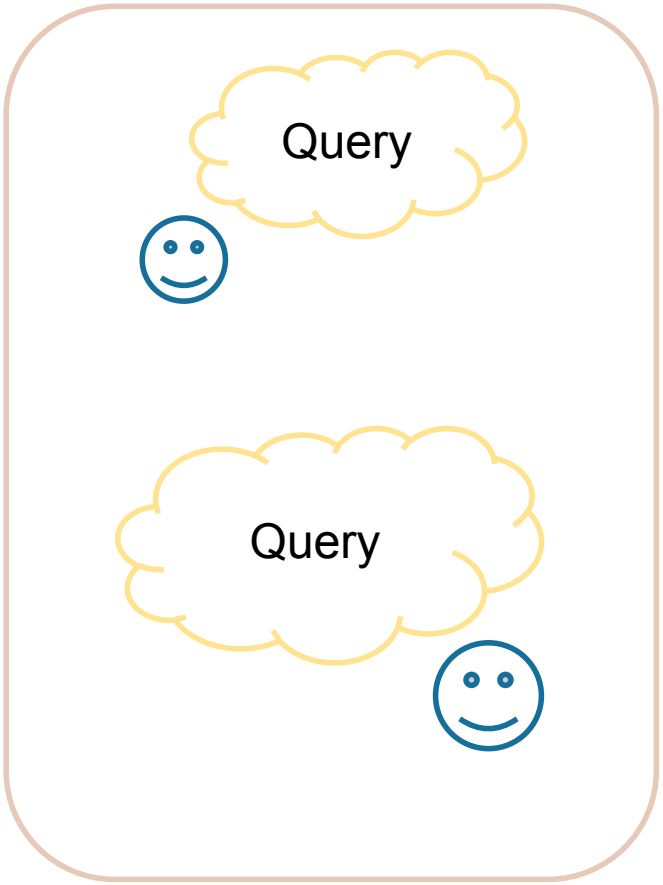


GUIDED PRACTICE

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

1. Prolog Basics

A world governed by facts and rules



Facts and rules are stored in a database (.pl file).

Example 1

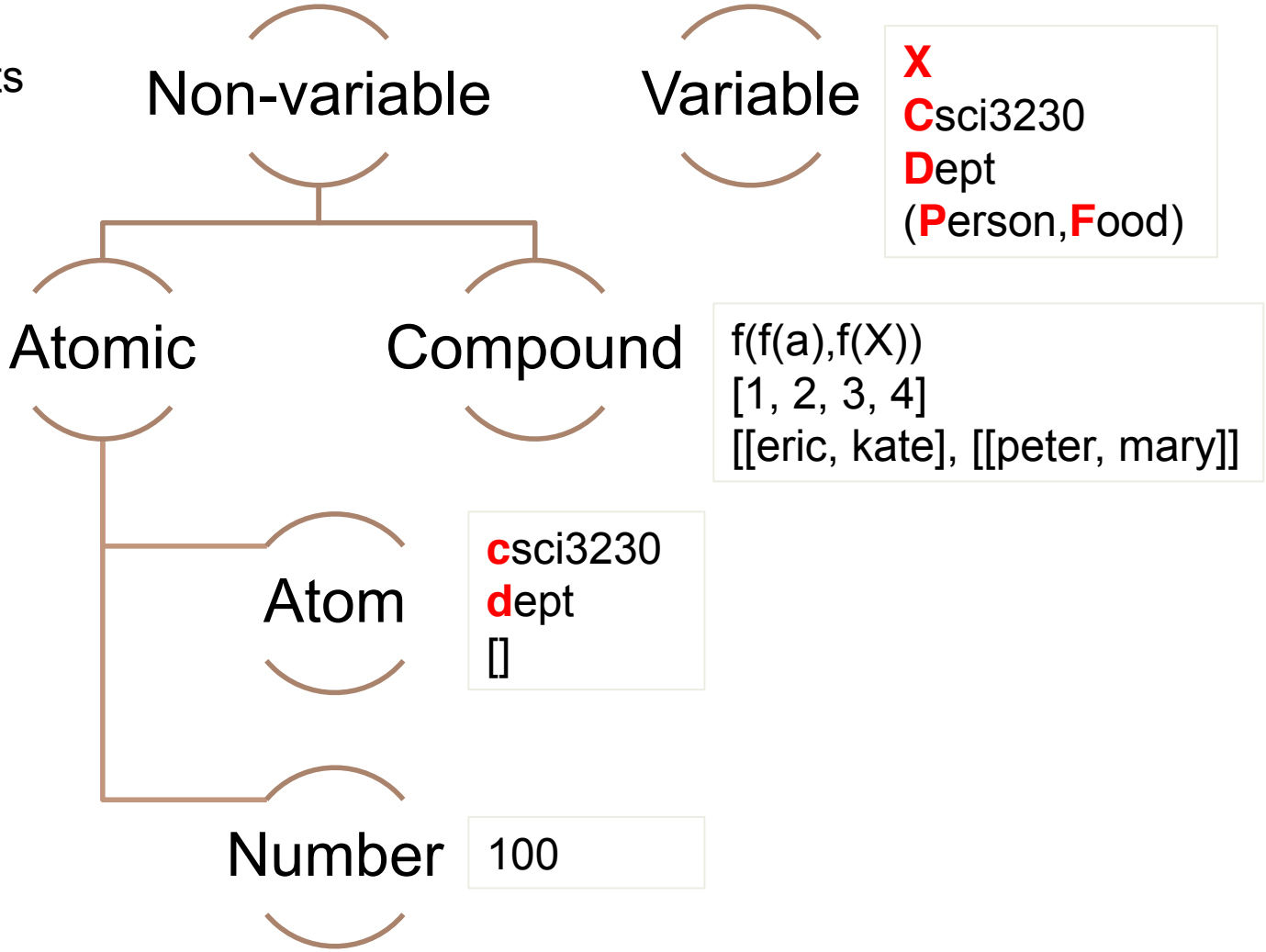
```
thinking(i).                %Fact
alive(X):-thinking(X).      %Rule
```

```
?- alive(i). %query
true. %fact
```

Ask your question in query mode

1. Prolog Basics

Terms: Data Objects



1. Prolog Basics: Fact

Example 1

```
%Fact
antonio_is_handsome.
%Fact_with_Arguments
handsome(antonio).
```

```
?- antonio_is_handsome.
true.
?- antonio_is_not_handsome.
ERROR: Undefined procedure: antonio_is_not_handsome/0
?- handsome(antonio).
true.
?- handsome(tom).
false.
?- handsome(X).
X = antonio.
```

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(antonio, 24).
age(sun, 24).
age(peter, 27).
```

```
?- age(antonio, 24).
true.
?- age(antonio, 16).
false.
?- age(X, 24).
X = antonio;
X = sun.
```

1. Prolog Basics: Variable

Example 1

```
%Fact_with_Arguments_including_atom_and_number
age(antonio, 24).
age(sun, 24).
age(peter, 27).
```

```
?- age(X, Y).
X = antonio,
Y = 24 ;
X = sun,
Y = 24 ;
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).
```

If

```
?- 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).
```

Or

And

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

1. Prolog Basics: Rule

- **<u>RULES</u>** 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.
```

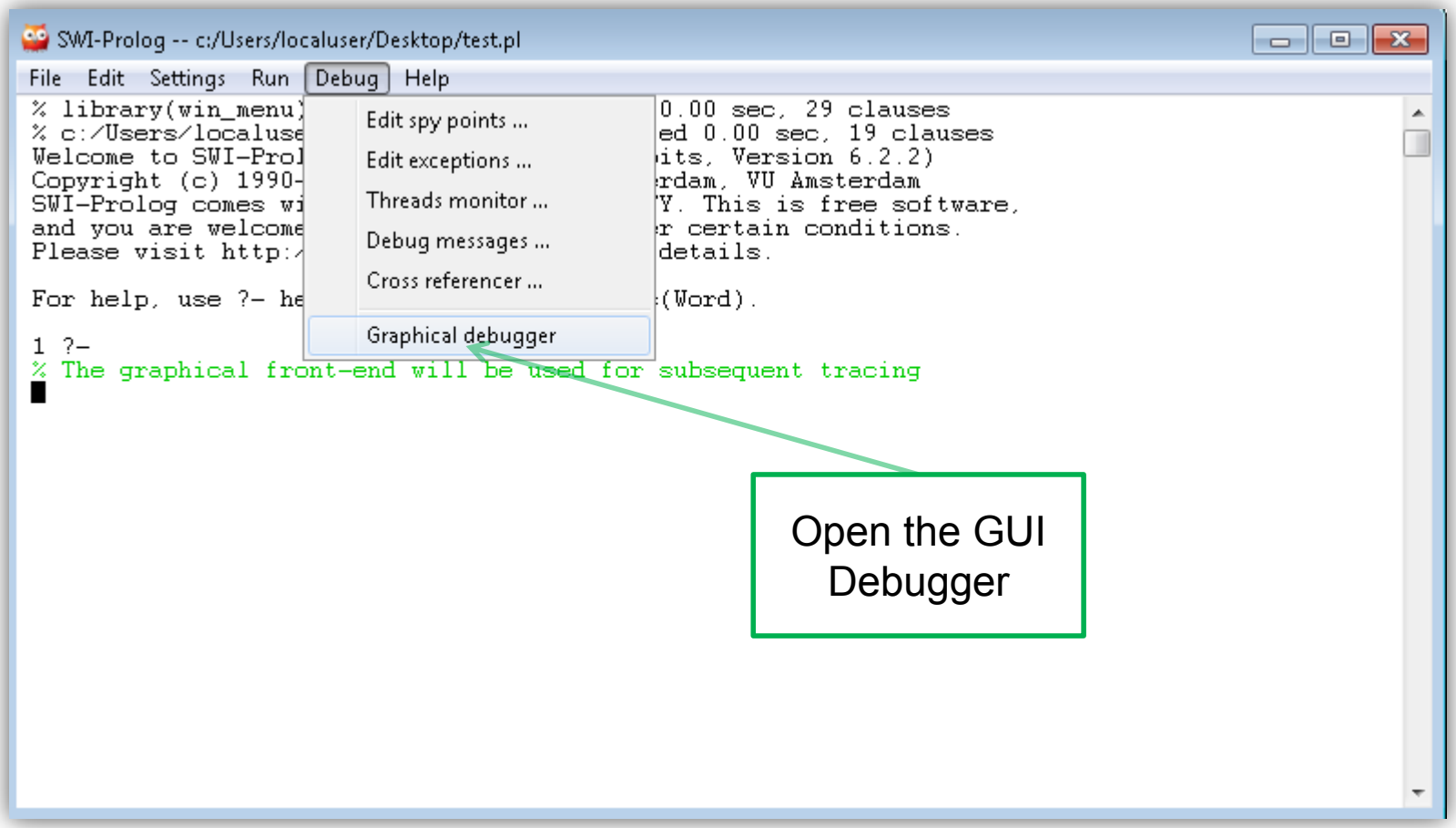
```
?- plus(3,4,R).
R = 7.
```

Example

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

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

Debug Prolog



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

Unification

Step

Skip this goal

Finish the selected goal

Call Stack

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).
```

Green indicates true.
Red indicates false.

Call: likes/2

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

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

```
?- [a|[b,c]].
true. %fact, different representation
?- [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 a 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,organge,banana])
true.
?- member(pear,[apple,organge,banana])
false.
?- member(X,[apple,organge,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.
?- 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

- You may review the gcd algorithm in the Hint section

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.

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)`, 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”.

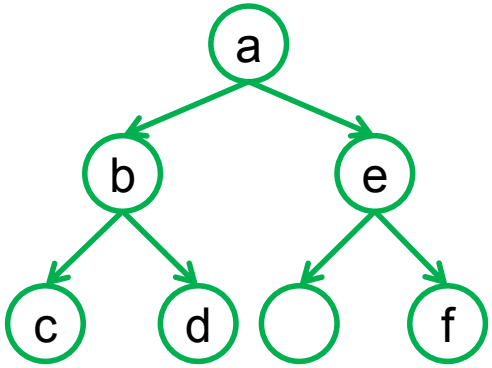
Programming Exercise 6

- 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 `puzzle3x3(Ans)` to find the solution. The answer `Ans` is a list, e.g. [1, 2, 3, 4, 5, 6, 7, 8, 9].

Hints

1. `sum([H,T],R) :- …`
   `sum(N,R) :- …`
2. gcd(a,0) = a
   gcd(a,b) = gcd(b,a mod b)
3. `listlen(L,N) :- lenacc(L,0,N).`
   `lenacc([H|T],A,N) :- …`
4. May need to use accumulator
5. http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/2_3.html
6. `ptree(ltree(L),rtree(R)) :- …`
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