1.1-1.7: Data Representation and Expressions

8 Sep 2006
CMPT14x
Dr. Sean Ho
Trinity Western University

- *Welcome 141/143 students*
- *Pick up syllabus*
- *Sign up for lab section*

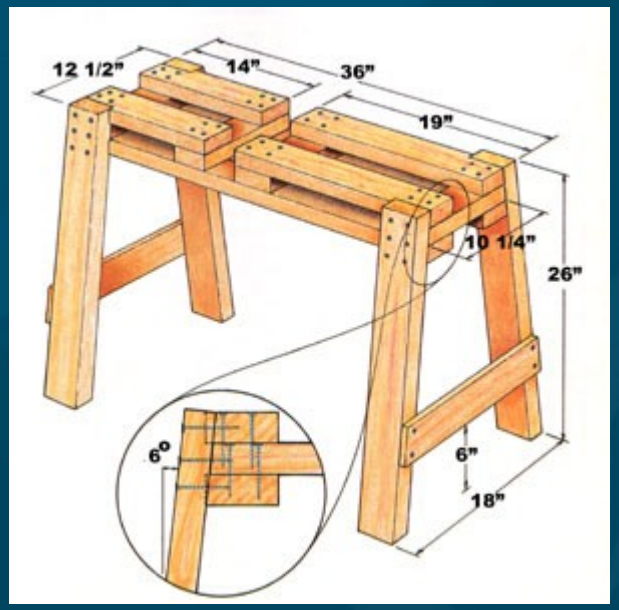
TRINITY WESTERN UNIVERSITY

http://cmpt14x.seanho.com/

Toolchains

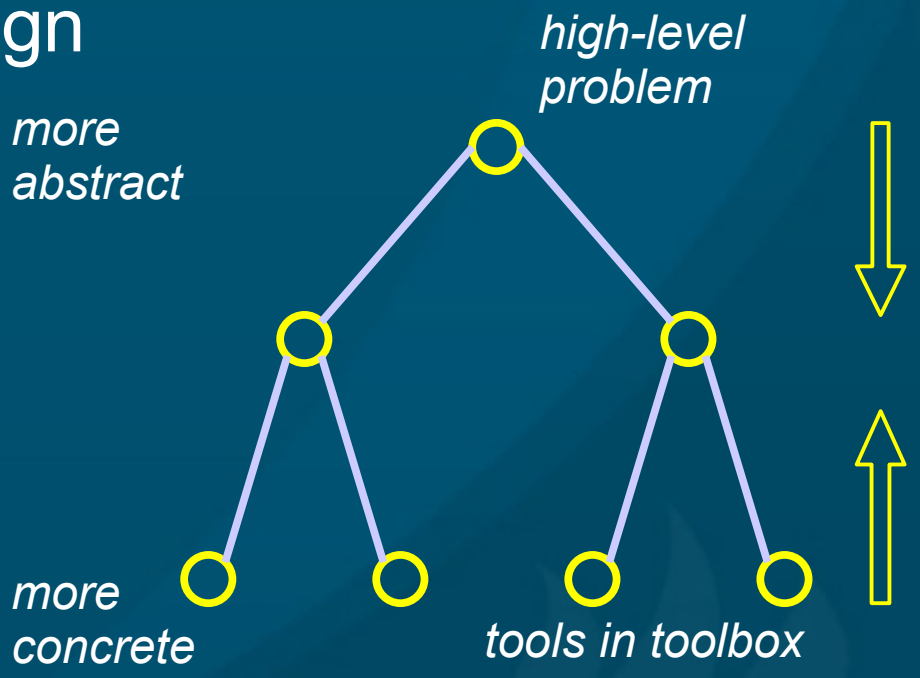
- Complex problems need sophisticated tools
- Complex tools are built up from simpler tools
- Always know what's in your toolbox:
  the tools you have to tackle problems
  - In software: libraries
  - In math: axioms
  - In philosophy: worldview, context

- In 14x: Python + libraries



Problem solving

- Top-down vs. bottom-up design



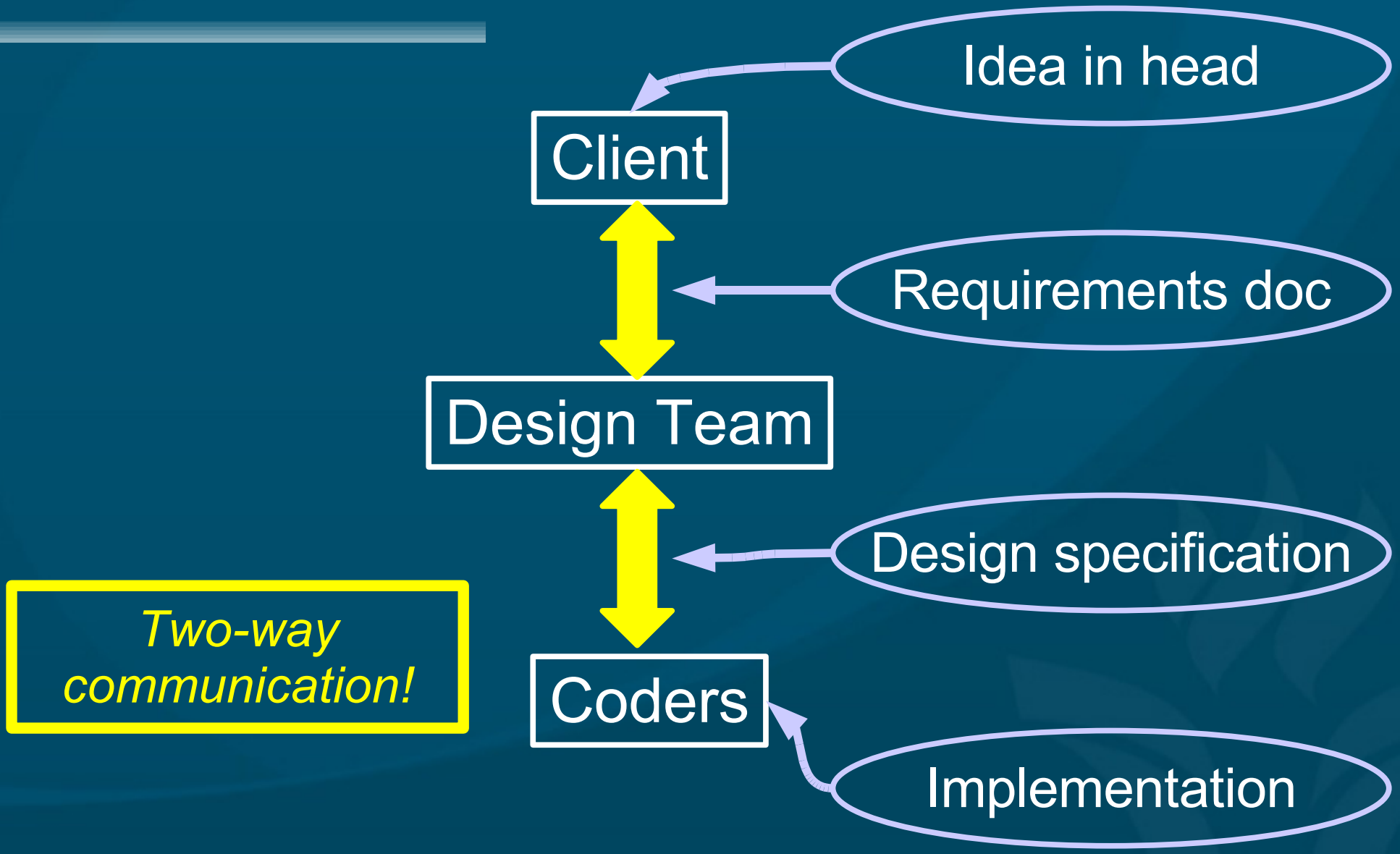
- ***W***rite everything down
- ***A***pprehend the problem
- ***D***esign a solution
- ***E***xecute the plan
- ***S***crutinize the results

Designing software vs. "hacking code"

- Look before you leap; think before you speak; ***design*** before you code!
- Programmer's optimistic schedule:
  - 4/5th coding
  - 1/5th testing/debugging
- Real-life schedule:
  - 1/3rd planning (***W***rite, ***A***pprehend, ***D***esign)
  - 1/6th coding (***E***xecute)
  - 1/2 testing/debugging (***S***crutinize)

Write
Apprehend
Design
Execute
Scrutinize

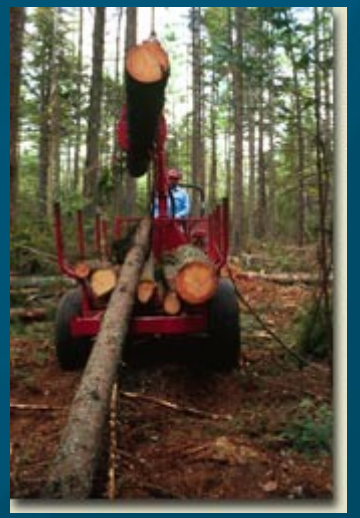
Interfaces in software development



Woodcutting example

- (see overheads / text pp.4-5)
- What are the library functions used in each version?

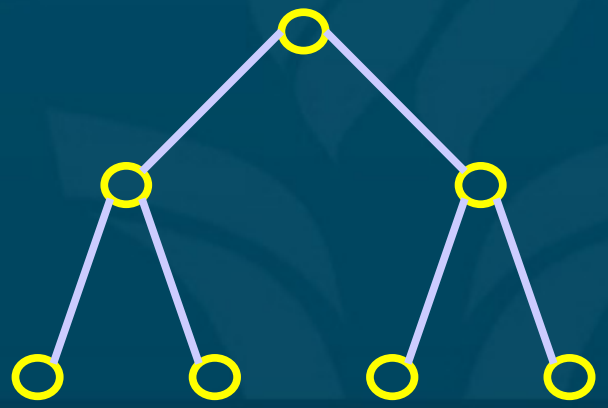
Woodcutting example



- We write out the solution in different levels of detail depending on
  - Who/what is executing the solution
  - What tools are available
- The solution is different for
  - An experienced lumberjack with good tools
  - A rookie who's never used a chainsaw
  - A software-controlled robot
  - A busy construction foreman
- *(which are more abstract / more concrete?)*

Review (1.1-1.4)

- Toolsmiths must know their toolboxes
  - (what does it mean for a computing scientist to be a toolsmith?)
- Top-down vs. bottom-up
- First step in problem-solving? (don't code yet!)
- ***WADES*** *(Write, Apprehend, Design, Execute, Scrutinize)*
- Levels of abstraction / levels of detail



Python/IDLE demo

- (demo of the Python programming environment)

Why Python?

- Why not M2, Java, C++, C#, PHP, Ruby, etc.?
- Syntax vs. semantics (more in a later section)
- At the CMPT14x level, the semantics of procedural programming in all these languages are pretty much the same
  - The only difference is syntax:

```
for (i=0; i<10; i++) {          (C++)
for i in range(10)              (Python)
```

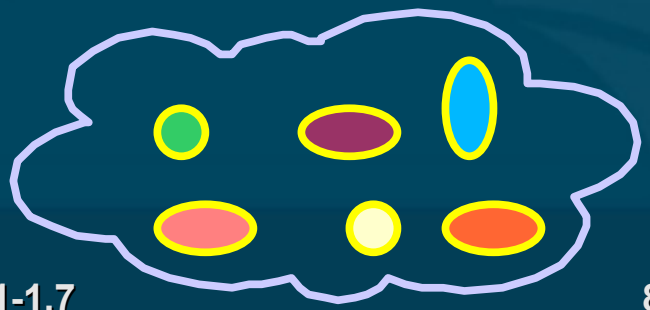
- After this class, you'll be able to pick up any procedural language pretty quickly

Data representation

- Data vs. information, knowledge vs. wisdom
- Raw data (factoids, memorized mantras) are useless unless you know what they mean!
- "There are 10 kinds of people in the world: those who know binary, and those who don't."
  - (what does "10" mean?)

Atomic vs. compound data

- Atomic: represents a single entity
  - e.g., 8, π, 6.022×10^{23}, z
- Compound: entity that also is a collection of components: e.g.,
  - Set: {43, 5, -29.3}
  - Ordered tuple: (3,9) *(what's the diff from set?)*
  - Complex number: 4.63+2i *(set or tuple?)*
  - Aggregate: (name, age, address, phone#)
- Singleton: {43}



Data types

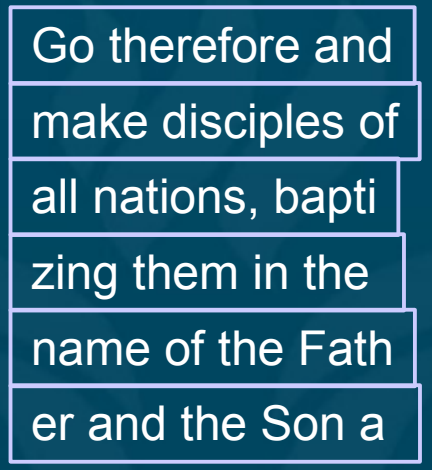
- Certainly atomic vs. compound data are different types
- But even among atomic data there are types: e.g.
  - Cardinals (unsigned whole numbers; naturals): 0, 1, 2, 3, 4, 5, ...
  - Integers (signed whole): -27, 0, 5, 247
  - Reals / Floats: 5.0, -23.0, $3x10^8$
  - Booleans: True, False
  - Characters: 'a', 'H', '5', '='
  - Strings: “Hello World!”, “5”

Types in Python

- Python has many built-in types; here are some:
  - int: e.g., 2, -5, 0
  - float: e.g., 2.3, -42e6, 0.
  - str: e.g., 'hello', “world”, '!', ''
  - bool: True, False
  - tuple: e.g, (2, -1, 'hi'), ()
- You can find the type of an expression with:
  - type(2.3)
- A complete list of types is at http://docs.python.org/ref/types.html

Different operations for different types (some examples)

- Operators work on operands:
  - e.g. 3+4: operator is "+", operands are 3, 4
- Cardinal type: e.g., +, -, *, /, *print*, etc.
- Character type: e.g., *capitalize*, *print*, etc.
  - 'b' / '4' doesn't make sense
- String type: e.g., *reverse*, *print*, etc.
  - *reverse*(1.3) doesn't make sense
- Array-of-strings type: e.g.,
  - Reverse each string in the array
  - Reverse the order of the array



(different?)

Abstract Data Types

- We define an Abstract Data Type (ADT) as a set of items w/ common properties and operations
  - e.g., Real ADT: reals w/ +, -, *, /, etc.
- Implementation of an ADT:
  - Real-world implementations of ADTs on actual computers have limitations
  - e.g. Can't represent integers bigger than 2147483647 (on a 32-bit machine)
  - e.g. Real (floating-point) numbers can be represented only up to a certain number of significant figures: 1.99999999999 ≠ 2

Variables and constants

- A constant's value remains fixed: e.g., π, e, 2
- A variable's value may change: e.g., x, numberOfApples
- We can assign new values to variables
  - numberOfApples = 12
  - numberOfApples = numberOfApples – 1
- But not to constants
  - π = 3.0 (don't want to do this!)
- In Python, there is no way to force a name to be constant
  - Convention: use ALLCAPS for names that are intended to be constant

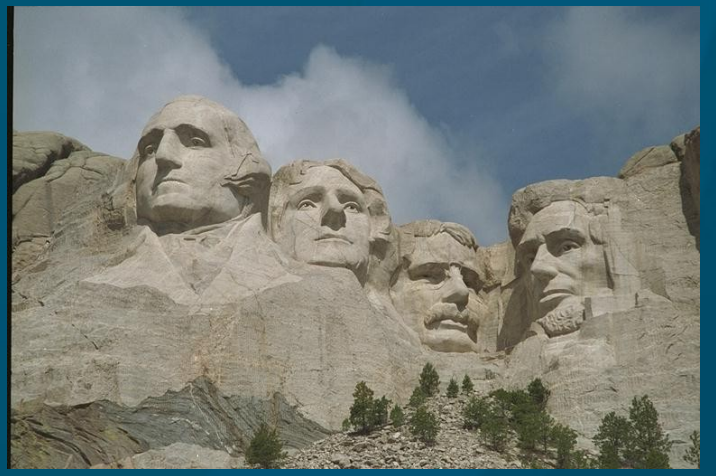
Expressions

- A combination of data items with appropriate operators is called an expression
- Expressions are evaluated to obtain a single numeric result
  - 15 + 9 + 11 + 2 ------evaluation--->>> 37
- Operators may evaluate to a different type than their operands:
  - 22.1 > 15.0:
    What is the type of the operands?
    What is the type of the result?

Logical operators

- Logical operators are operators on the bool type:
  - GodLovesMe = True
  - ILoveGod = False
- not: flips True to False and vice-versa
  - not GodLovesMe >>> False
- and: evaluates to True if both operands are True
  - GodLovesMe and ILoveGod >>> False
- or: evaluates to True if at least one operand is True
  - GodLovesMe or ILoveGod >>> True

Operator Precedence

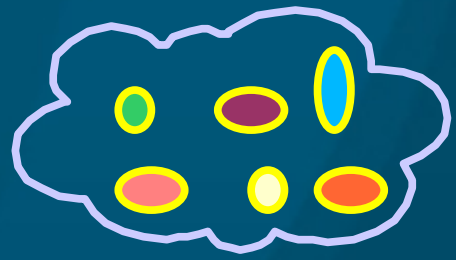


- How would you evaluate this?
  - 5 + 4 * 2
  - (5 + 4) * 2 >>> 18: Addition first
  - 5 + (4 * 2) >>> 13: Multiplication first
- Precedence is a convention for which operators get evaluated first (higher precedence)
  - Usually multiplication has higher precedence than addition
- When in doubt, use parentheses!

Expression compatibility

- 5 + True doesn't make sense: incompatible types
- What about 5(int) + 2.3(float)?
  - Works because the two types are expression compatible
- The "+" operator is overloaded:
  - It works for multiple types: both int and float
- It turns out that in Python, 5+True does evaluate:
  - 5+True >>> 6
    (interprets True as 1 and False as 0)

Review (1.5-1.7)



- Atomic vs. compound data (examples?)
- Data types (examples?)
  - What's the difference: 5, 5.0, '5', “5”, (5), {5}
- Operators, operands, ADTs, implementations
- Variables vs. constants
- Logical operators: not, and, or
- Operator precedence
- Expression compatibility (what types?)

=	NOT	*
AND	/	<
		-
+	OR	

TODO items

- For Monday
  - Read M2 text through §2.1
  - Read Python text ch1-2
- Go to Neu9 computer lab:
  - Make sure you can login
  - Python/IDLE intro on course www (due Wed)
- Ch1 quiz next Monday start of class
- Ch1 homework due next Wed:
  - M2 text, §1.11, #25, 31, 40