

# 1.1-1.7: Data Representation and Expressions

---

10 Sep 2008

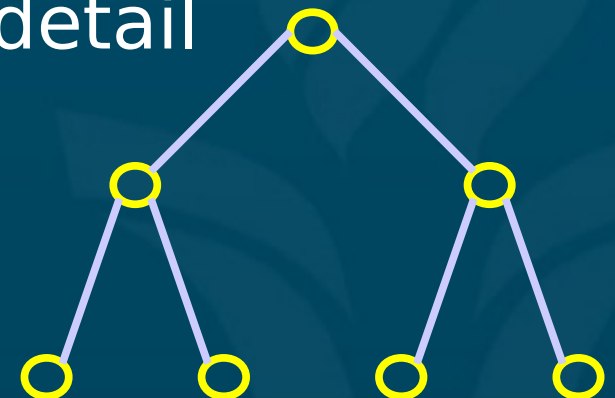
CMPT14x

Dr. Sean Ho

Trinity Western University

# 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



# Data representation

```
01000100100101110111010001001000
10100100110011110100100100001110
1011101110101010111110001001001
10111101011000001110001001111000
10001000100100101000111001000100
10010010100010101011101110101101
01010101001001010100010001001110
1111111110101010111110001000001
10101010000001011000010011111001
10000010001011110011010101000010
111110101010101010101010111001
00101000111000001111100010001011
```

- 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,  $\pi$ ,  $6.022 \times 10^{23}$ , 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,  $3 \times 10^8$
  - **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 (different?)
  - Reverse the order of the array

Go therefore and  
make disciples of  
all nations, bapti  
zing them in the  
name of the Fath  
er and the Son a

# 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**:  $1.99999999999999 \neq$



# Variables and constants

- A **constant**'s value remains fixed: e.g.,  $\pi$ , 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
  - $\pi = 3.0$  (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



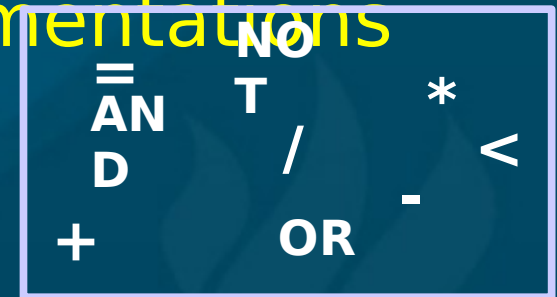
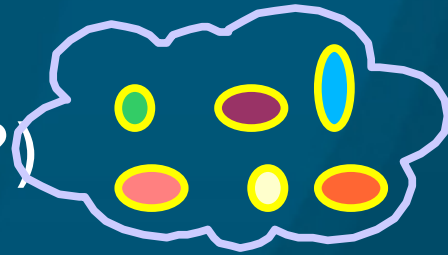
- How would you **evaluate** this?
  - $5 + 4 * 2$
  - $(5 + 4) * 2 \gg \gg 18$ : Addition first
  - $5 + (4 * 2) \gg \gg 13$ : Multiplication first
- **Precedence** is a convention for which operators get evaluated first (higher precedence)
  - Usually multiplication has higher precedence than addition
- 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)

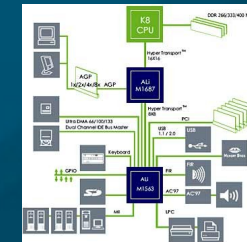
- Atomic vs. compound data (examples?)
- Data types (examples?)
  - What's the difference: 5, 5.0, '5', "5", (5), {5}
- Operators, operands, ADTs, implementations
- Variables vs. constants
- Logical operators: not, and, or
- Operator precedence
- Expression compatibility (what types?)



# Hardware abstractions





- Generally, most computers have these basic hardware components:

- Input
- Memory
- Processing
- Control
- Output

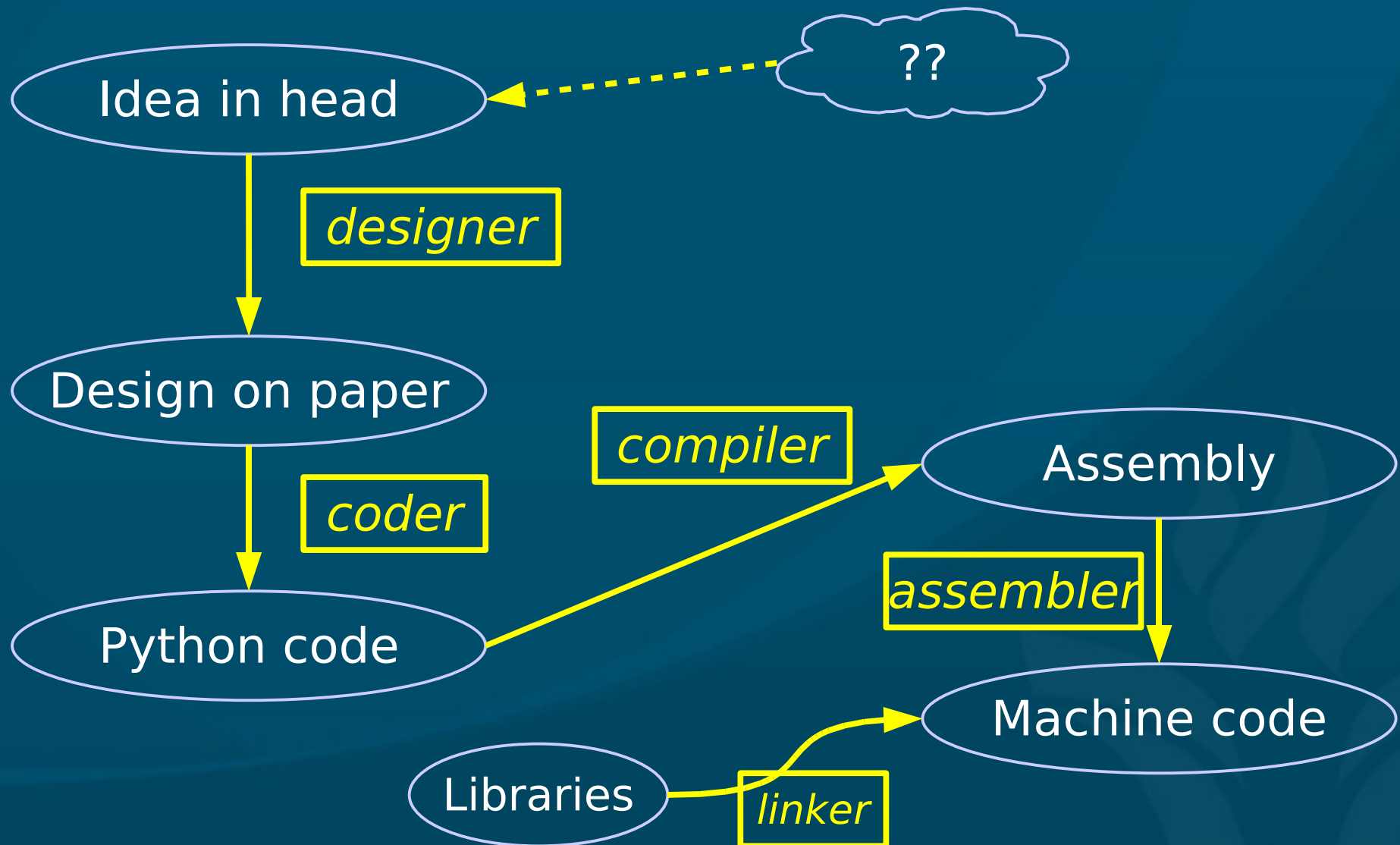


- Together with the software, the environment presented to the computer user by these is the **virtual machine**

# Software abstractions

- **Instructions:** basic commands to computer
  - e.g., ADD x and y and STORE the result in z
- **Programming language:** set of all available instructions
  - e.g., Python, C++, machine language 
- **Program:** sequence of instructions
  - e.g., your “Hello World” program
- **Software:** package of one or more programs
  - e.g. Microsoft Word, Microsoft Office 
- **Operating system:** software running the computer: provides environment for programmer
  - e.g., Windows XP, Mac OSX, Linux, etc.  

# Programming is translation



# Control abstractions

---

- **Sequence**: first do this; then do that
- **Selection (branch)**: IF ... THEN ... ELSE ...
- **Repetition (loop)**: WHILE ... DO ....
- **Composition (subroutine)**: call a function
- **Parallelism**: do all these at the same time
  
- These are the basic building blocks of program control and structure

# TODO items

---

- Go to **Neu9** computer lab:
  - Make sure you can **login**
  - **Python/IDLE** intro on course www (due Fri)
- Ch1 **homework** due Fri
- **Reading** for Fri:
  - M2 text through **§2.1**
  - Python text **ch1-2**