Coding Style and Basic Operators

21 Sep 2009
CMPT140
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Outline for today

- Static vs. dynamic typing
- Documentation: comments, docstrings, etc.
  - Style conventions for identifiers
- Keyboard input: input() and raw_input()
- Basic operators, type conversion
- Formatted output

Short lab-writeup for Labs 1, 2

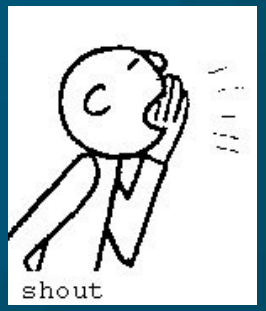
- Design (“WAD” in “WADES”) (10pts)
  - IPO: input – process – output
  - Variables needed?
  - Math formulas used?
- Code (“E” in “WADES”) (30pts)
  - Choose good identifiers
  - Docstrings, comments
- Output (“S” in “WADES”) (10pts)
  - A couple runs with different inputs

Static vs. dynamic typing

- All variables have a type: int, float, str, bool, ...
- Some languages (C, Java, M2): statically typed:
  - Must declare the variable type ahead of time
    - **x, y: REAL;**
    - **int numApples;**
  - Can't change the type or assign a value of a different type:
    - **x := “Hello, World”; /* won't work! */**
- But Python is dynamically typed:
    - **x = 5.0**
    - **x = True # works in Python**

Declaring vs. initializing

- This is only necessary for statically-typed languages:
  - Declare a variable to tell the compiler the type of the variable:
    - **VAR numApples : CARDINAL; (* M2 *)**
  - Its value is undefined until it is initialized:
    - **BEGIN**
      - **numApples := 5; (* M2 *)**
- In a dynamically-typed language like Python, just initialize the variable:
  - **numApples = 5 # okay in Python**



Documentation

- Document your thinking at every step,
  even the ideas that didn't work!
  - Programmer's diary: log of everything
- External documentation: outside the program
  - User manual:
    - What user input is required
    - What the user should expect the program to output
    - No details about program internals
- Internal documentation: within the program
  - Descriptive variable/module names
  - Comments in the code
  - Online help for the user

Internal documentation

- Good variable names: numHashes
  - Bad variable names: x, num, i
- Comments: # in Python (to end of line)
  - **# loop numHashes times**
  - **while (counter < numHashes):**
    - **print "#", # no newline**
    - **counter = counter + 1**
- Online help:
  - "Enter 'h' for online help."

Comments

- Explain the "why", not the "what":
  - Bad: `x = x + 1  # increment x`
  - Good: `x = x + 1  # do next hashmark`
- Keep comments up-to-date!
  - Incorrect comments are worse than no comments
- Comments are no substitute for external documentation
  - Still need a separate design doc, pseudocode, user manual, etc.

Docstrings

- Python convention is to create a docstring at the top of every module, function, class, etc.:
  - **” ” ” Print a bunch of hashes.**

    Nellie Hacker, CMPT140
    ” ” ”
    numHashes = input(“How many hashes? ”)
    . . .
- Triple-quotes: this is a string, not a comment
- First line is a short summary
- Second line is blank, then detailed description
- Automated Python tools read docstrings to help you organize your code
  - More info: http://www.python.org/dev/peps/pep-0257/

Style conventions

- Not hard-and-fast rules, but flexible conventions that make code easier to read and understand
- Variable names: numHashes
  - Flexible, but I prefer no underscores, and capitalize each word ("CamelCase")
  - First letter is lowercase
- File/module names: helloworld.py
  - Short, all lowercase, no underscores
- Function names: print_hashes()
  - lowercase, command predicate, underscores
- More details: http://www.python.org/dev/peps/pep-0008/

Keyboard input

- Output using print()
- Use input() to get a value from the user:
  - balance = input("Opening balance? ")
  - The argument is the prompt string
    - Note trailing space in the prompt
  - Python interprets the user's response as a Python expression and finds its type
    - Input can be any valid Python expression
  - Just pressing Enter w/o input gives an error

raw_input() vs. input()

- How to input a string from the user?
  - input() tries to interpret the user's input:
  - Too fancy; just want the straight text
- Use raw_input() instead
- Return type is always str
- You can use raw_input() at the end of your program to wait for the user to press Enter before the program finishes
  - raw_input("Press Enter to quit.")

Expressions

- An expression is a combination of
  - Literals, constants, and variables,
  - Using appropriate operations (by type)

```
12 – 7
numApples * 4
```

- A few operators we'll look at:
  - Binary: + - * / % // **
  - Comparison: == < > <= => !=
  - Boolean: and or not (shortcut)

Binary arithmetic operators

- + , –, *: addition, subtraction, multiplication
- **: power: 2**4 == 16
- /: division: 7.0 / 2 == 3.5
  - On two ints, returns an int (floor): 7 / 2 == 3
  - A note about float arithmetic: 7.2 / 2 ≠ 3.6
- //: floor division
  - Same as / for ints: 7 // 2 == 3
  - On floats, returns floor of quotient: 7.0 // 2 == 3.0
- %: modulo (remainder): 8 % 3 == 2
  - 8 % 0 => ZeroDivisionError

Comparison operators

- Test for quantitative equality: 2 + 3 == 5
- Test for inequality: 2 + 3 != 4
  - Can also use <>
- Comparison: <, >, <= , >=
- Test for identity: is, is not
  - (2, 3) == ((2, 3)), but
  - (2, 3) is not ((2, 3))

Boolean operators: shortcut

- Boolean operators: and or not
  - In C/C++/Java: && || !
- Python's boolean operators have shortcut semantics:
  - Second operand is only evaluated if necessary
    - (7 / 0) and False => ZeroDivisionError
    - False and (7 / 0) == False
      - Doesn't raise ZeroDivisionError
    - True or (7 / 0) == True
      - Same thing

Type conversions

- Python is dynamically typed, so operators can do implicit type conversions to their operands:
  - 2 (int) + 3.5 (float) == 5.5 (float)
    - Plus (+) operator converts 2 (int) to 2.0 (float)
- You can manually convert types:
  - int(2.7) == 2
  - int(True) == 1
  - Better alternative to input():
    - ageString = raw_input("Age? ")
    - age = int(ageString)

Precedence

- Of the operators we've learned, the precedence order from highest (evaluated first) to lowest (evaluated last) is
  - **
  - **Unary +, -**
  - *, /, %, //
  - **Binary +, -**
  - ==, !=, <>, <, >, <=, >=
  - **Is, is not**
  - **Not**
  - **And**
  - **or**
- Complete precedence rules at http://docs.python.org/ref/summary.html

TODO

- Lab1 due Wed/Thu!
  - 10pm upload to myCourses
  - #40: don't need looping; just run for 3 purchases
- Read ch3 for Wed
- Lab2 posted, due next week Wed/Thu
  - Uses selection(if) and/or looping
  - We will cover this on Wed
  - Short writeup ok