§2.7-2.10: Basic Arithmetic Operators

- *HW 1.35 due today*

15 Sep 2006
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Announcements

- Electronic turn-in (optional):
  - Go to the eCourse for 140/141/143/145
  - Workspace -> (MTW) Lab
  - Create a folder with your name ("John Doe")
    - Location: Root/Lab 01 (or Root/Homework)
    - Uncheck "Students can add documents"
  - Navigate to your new turn-in folder
    - e.g., (MTW) Lab -> Lab 01 -> John Doe
  - Upload all your files for this lab
    - Lab writeup, Python .py code, output

Review of (§2.3-2.4)

- Steps to problem solving: WADES in more detail
- Documentation
  - External documentation: design, manuals
  - Internal documentation:
    - Comments
    - Docstrings
- Style guidelines

- (see bankinterest.py example)

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/

What's on for today (§2.7-2.10)

- Expressions, operators, operands
  - Binary arithmetic: + - * / % // **
  - Comparison: == < > <= => != <> is, is not
  - Boolean: and or not (shortcut semantics)
- Type conversions
- Precedence rules
- Formatted output
  - %d, %f, %s

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: == < > <= => != <> is
  - 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

Formatted output: print with %

- The built-in function print can accept a format string:
  - print “You have %d apples.” % 7
    - Output: “You have 7 apples.”
  - It can take multiple arguments:
    - print “%d apples and %d oranges.” % 7, 10
      - Output: “7 apples and 10 oranges.”
  - Format codes:
    - %d: integer
    - %f: float
    - %s: string

Formatting: %d, %f

- You can specify the field width:
  - print "%3d apples" % 5
    - Output: " 5 apples" (note two leading spaces)
  - print "%-3d apples" % 5
    - Output: "5 apples" (left-aligned: two trailing spaces)
  - print "%03d apples" % 5
    - Output: "005 apples" (padded with zeros)
  - print "%4.1f apples" % 5.273
    - Output: " 5.3 apples"
    - 4 is the total field width, including the decimal
    - 1 is the number of digits after the decimal

Review of today (§2.7-2.10)

- Expressions, operators, operands
  - Binary arithmetic: + - * / % // **
  - Comparison: == < > <= => != <> is, is not
  - Boolean: and or not (shortcut semantics)
- Type conversions
- Precedence rules
- Formatted output
  - %d, %f, %s

TODO

- Lab 01 due MTW 10pm to your TA
  - Ch2: (35 or 36) and 40
  - eCourses turn-in okay
- Quiz ch2 on Mon
- Reading for Mon: through M2 §3.3 and Py ch4