§2.3-2.4: Problem Solving, Documentation, Style

- *devo*

14 Sep 2006
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review of §2.2, 2.5, 2.11

- Components of a baby Python program
- Modules
- Library tools (what are some we know already?)
- Literals, identifiers and reserved words (examples?)
- Strings, quoting, newlines
- Statically-typed vs. dynamically-typed
- Declaring and initializing variables
  - (what is needed in C? In Python?)
- Keyboard input

Follow-up notes from yesterday

- Static typing = strong typing
  Dynamic typing = weak typing
- print “hello”,
  - No newline, but still a space after “hello”
  - For more control, import sys and use sys.stdout.write()
- Note the updated list of allowable Python constructs on the course website:

  http://cmpt14x.seanho.com/python_constructs.html

What's on for today (§2.3-2.4)

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

Steps to solving a problem

- 11 steps expanding upon ***WADES***:

- Analyze the problem
- Plan a solution
- Write down your data tables and I/O
- Refine your solution (several times)
- Execute your plan (code) and evaluate the results

Analyze the problem

- Step 1: Write the problem out
  - "Write a program that prints out a user-specifi number of hash marks (#)."
- Step 2: Ask whether a computer is appropriate
  - Other ways to solve the problem?
- Step 3: Rewrite the problem in your own words
  - Given: number of hash marks to print
  - To do: print hash marks
  - Result: a string of hash marks, e.g., ########
  - Formula: none needed

Plan and refine a solution

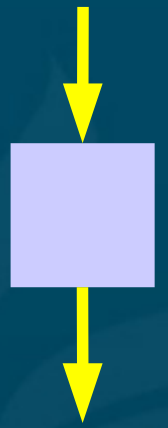
- Step 4: Re-use previous work where possible
  - Our program has input and output; in some languages (not Python), we need I/O libraries.
- Step 5: Break the problem into smaller steps
  - Input: read in desired number of hash marks
  - Computation: none
  - Output: print out hash marks

Further refinements

- Second refinement:
  - Input:
    - Ask user for desired number of hash marks
    - Input response and assign to a cardinal variable
  - Computation:
    - Initialize a cardinal counter to zero
  - Output:
    - While the counter is less than the desired number of hash marks:
      - Print a hash mark
      - Increment the counter

Data tables and I/O

- Step 6: List all variables and imports (data table)
  - Variables: numHashes, counter (int)
  - Imports: none
- Step 7: List required input (precondition) and expected output (postcondition)
  - Input: An int ≥ 0, e.g. 6
  - Output: A string of hashes, e.g. "#######"



Refining the solution

- Step 8: Pseudocode
  - Print "How many hashes do you want printed?"
  - Read user input into numHashes
  - counter <---- 0
  - While (counter < numHashes)
    - Print "#"
    - counter <----- counter + 1

Write the Python code

- Step 9: Python code (syntax matters here)
  - ” ” ” Print a bunch of hashes.
  -
  - Nellie Hacker, CMPT140
  - ” ” ”
  - numHashes = input(“How many hashes? ”)
  - counter = 0
  - while (counter < numHashes):
    - print “#”, # no newline
    - counter = counter + 1
  - print

Execution and evaluation

- Step 10: Compile, link, run
  - First run:
    - How many hashes do you want? 4
    - ####
  - Second run:
    - How many hashes do you want? 7
    - ########
- Step 11: Check against specifications
  - Does program print the right number of hashes? No one-off errors?
  - What about weird input: 0, -1, 120, 5.3, abc?

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

Examples of 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/

Review of today (§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)

TODO items

- Homework due tomorrow (Fri):
  - §1.11 # 35
- Reading: through §2.10 for Fri
- Quiz ch2 next Mon
- Lab 1 due next MTW in lab section
  - Short writeup ok