§4.8-4.10: ROT13 example, Recursion

- *Quiz ch2-3 today*

24 Sep 2007
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Quiz 02: 10 minutes, 20 points

- Which are legal Python variable names? Why?

num-apples	BEGIN	for
05LabScore	_3	Paul's_age

- Contrast strong-typing vs. weak-typing. Pros/Cons?
- Describe at least four kinds of documentation (internal or external) you should have in your Python programs.
- Write a complete Python program to evaluate and print the value of the sum 1 + 2 + 3 + ... + 999 + 1000.
  - Docstring / comments not necessary but useful for partial credit.

Quiz 02: answers #1

- Which are legal Python variable names? Why?
  - **num-apples**: not okay (punctuation)
  - **BEGIN**: okay
  - **for**: not okay (reserved word)
  - **05LabScore**: not okay (starts with number)
  - **_3**: okay (albeit poor style!)
  - **Paul's_age**: not okay (punctuation)

Quiz 02: answers #2-3

- Contrast strong-typing vs. weak-typing.
  - Strong typing: variables cannot change type
    - Must declare type of variable and initialize value
    - Compiler enforces correct type
  - Weak typing: can change type
    - More flexible, but usually we don't want variables to change type
- Describe at least four kinds of documentation (internal or external) you should have in your Python programs.
  - Comments, docstrings, online help/prompts, user manual, programmers' diary, variable names

Quiz 02: answers #4

- Write a complete Python program to evaluate and print the value of the sum 1 + 2 + 3 + ... + 999 + 1000.
  - While loop or for loop:

```
sum = 0
for counter in range(1, 1001):
    sum += counter
print "The sum is", sum
```

  - Or:

```
print "The sum is 500500"
```

What's on for today (§4.8-4.10)

- Some debugging tips
- A fun example: ROT13
  - ord(), chr(), string indexing, len()
  - Stub program
- Recursion

Some debugging tips

- Do hand-simulation on your code
- Use print statements liberally
- Double-check for off-by-one errors
  - Especially in counting loops: for, range()
- Try a stub program first
  - General structure of full program
  - Skip over computation/processing
    - Use dummy values for output
- Check out the debugger in IDLE

A fun example: ROT13

- Task: Translate characters into ROT13 one line at a time
  - ROT13:
    - Treat each letter A-Z as a number between 1-26,
    - Add 13 to the number and wrap-around if necessary
    - Convert back to a letter
    - Preserve case
    - Leave all non-letter characters alone
  - e.g., ROT13 ('a') == 'n', ROT13 ('P') == 'C', ROT13 ('#') == '#'

ROT13: Problem restatement

- Input:
  - A sequence of letters, ending with a newline
- Computation:
  - Convert letter to numerical form
  - Add 13 and wrap-around if necessary
  - Convert back to letter form
- Output:
  - Print ROT13'd character to screen

ROT13: convert letters to numbers

- How do we convert from a letter character to a numerical code?
  - Use ord(char): testbed program

```
char = raw_input("Type one character: ")
print "The ASCII code for %s is %d." % \
    (char, ord(char))
```

- ASCII codes: 'A' = 65, 'Z' = 90, 'a' = 97, 'z' = 122
- Convert back with chr(code)

More fun with strings

- How do we read one character from a string?
  - In Python, characters are just strings of length 1
  - In C, M2, etc., strings are arrays of characters
- Index into a string (more on array indexing later):
  - **name = "Golden Delicious"**
  - **name[0] is 'G'**
- Length of a string:
  - **len(name) is 16**
  - **name[len(name)-1] is 's'** **# (the last character)**
- Iterate over string:
  - **for idx in range(len(string)):**

ROT13: Pseudocode

- Print intro to the user
- For each character in the string:
  - Convert to ASCII numerical code
  - If character is an uppercase letter,
    - Add 13 to code
    - If code is now beyond 'Z', subtract 26 (wrap-around)
  - Else if character is a lowercase letter,
    - Add 13 to code
    - If code is now beyond 'z', subtract 26 (wrap-around)
  - Else (any other kind of character),
    - Leave it alone
  - Convert numerical code back to character and print

How to test if upper/lower case?

- Our pseudocode involves a test if the character is an uppercase letter or lowercase letter
- How to do that?

```
if (code >= ord('a')) and (code <= ord('z')):
    # lowercase
elif (code >= ord('A')) and (code <= ord('Z')):
    # uppercase
else:
    # non-letter
```

ROT13: Stub program pseudocode

- For each character in the string:
  - Convert to ASCII numerical code
  - Convert back to character
  - Print ASCII code and converted character

- This stub program allows us to test the char<->ASCII conversion process and the string indexing
- Tackle the ROT13 processing later

ROT13: Stub program code

```
"""Convert to ASCII code and back."""
text = raw_input("Input text? ")
for idx in range(len(text)):
    char = text[idx]
    code = ord(char)
    char = chr(code)
    print char, code,
```

- Sample input: hiya
- Sample output: h 104 i 105 y 121 a 97

ROT13: Full program code

```
"""Apply ROT13 encoding."""
import sys                              # sys.stdout.write()

text = raw_input("Input text? ")
for idx in range(len(text)):            # iterate over string
    char = text[idx]
    code = ord(char)
    if (code >= ord('a')) and (code <= ord('z')):   # lower
        code += 13
        if code > ord('z'):             # wraparound
            code –= 26
```

ROT13: Full program code, p.2

```
    elif (code >= ord('A')) and (code <= ord('Z')):    # upper
        code += 13
        if code > ord('Z'):          # wraparound
            code –= 26
    char = chr(code)
    sys.stdout.write(char)
print
```

http://twu.seanho.com/python/rot13.py

ROT13: Results and analysis

- Input: hiya
  - Output: uvln
- Input: uvln
  - Output: hiya
- Input: Hello World! This is a longer example.
  - Output: Uryyb Jbeyq! Guvf vf n ybatre rknzcyr.
- Generalizations/extensions?
  - Handle multiple lines one line at a time?
    - How to quit?

Recursion

- Recursion is when a function invokes itself
- Classic example: factorial (!)
  - n! = n(n-1)(n-2)(n-3) ... (3)(2)(1)
  - 0! = 1
- Compute recursively:
  - Inductive step: n! = n*(n-1)!
  - Base case: 0! = 1
- Inductive step: assume (n-1)! is calculated correctly; then we can find n!
- Base case is needed to tell us where to start

factorial() in Python

```
def factorial(n):
    """Calculate n!.  n should be a positive integer."""
    if n == 0:                          # base case
        return 1
    else:                               # inductive step
        return n * factorial(n-1)
```

- Progress is made each time: factorial(n-1)
- Base case prevents infinite recursion
- What about factorial(-1)? Or factorial(2.5)?

Review of today (§4.8-4.10)

- Some debugging tips
- A fun example: ROT13
  - ord(), chr(), string indexing, len()
  - Stub program
- Recursion

TODO

- Lab 02 due Wed:
  - 3.14 # 16 / 17 / 23a / 23b / 23c
- Quiz ch4 on Fri