Functions, ROT13 example

30 Sep 2009
CMPT140
Dr. Sean Ho
Trinity Western University

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

Predicates: pre-/post- conditions

```
def ASCII_to_char(code):
    """Convert from a numerical ASCII code
    to the corresponding character.
    """
    return chr(code)
```

- The parameter code needs to be <128: either
  - State preconditions clearly in docstring:
    - **"""Pre: code is an integer between 1 and 128.   Post: returns the corresponding character.”””**
  - Or code error-checking in the function:
    - **if code >= 128:**

Example: error-handling

```
def ASCII_to_char(code):
    """Convert from a numerical ASCII code
    to the corresponding character.

    pre: code is an integer
    post: returns the corresponding character
    """
    if (code <= 0) or (code >= 128):
        print "ASCII_to_char(): needs to be <128"
    else:
        return chr(code)
```

Call-by-value, call-by-reference

- In some languages functions can have side effects:(M2)

```
PROCEDURE DoubleThis(VAR x: INT);
BEGIN
    x := x * 2;
END DoubleThis;

numApples := 5;
DoubleThis(numApples);
```

- Call-by-value means that the value in the actual parameter is copied into the formal parameter
- Call-by-reference means that the formal parameter is a reference to the actual parameter, so it can modify the actual parameter (side effects)

Python is both CBV and CBR

- In M2, parameters are call-by-value
  - Unless the formal parameter is prefixed with “VAR”: then it's call-by-reference
- In C, parameters are call-by-value
  - But parameters can be “pointers”
- Python is a bit complicated: roughly speaking,
  - Immutable objects (7, -3.5, False) are call-by-value
  - Mutable objects (lists, user-defined objects) are call-by-reference

Example of CBV in Python

```
def double_this(x):
    ”””Double whatever is passed as a parameter.”””
    x *= 2

numApples = 5
double_this(5)                    # x == 10
double_this(numApples)            # x == 10
double_this(“Hello”)              # x == “HelloHello”
```

- The global variable numApples isn't modified, because the changes are only done to the local formal parameter x.

A fun example: ROT13

- Task: Translate characters into ROT13 one line at a time:
  - Treat each letter A-Z as a number 1-26,
  - Add 13, wrap-around if needed
  - 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 A-Z to 1-26

- How do we convert from a letter character to a numerical code?
  - Use ord(char): try this out in IDLE
  - Or write a testbed program:

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

- ASCII codes: 'A' = 65, 'B' = 66, ..., 'Z' = 90, 'a' = 97, 'z' = 122
- Convert back with chr(code)

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
  - Else if character is a lowercase letter,
    - Add 13 to code
    - If code is now beyond 'z', subtract 26
  - Else (any other kind of character),
    - Leave it alone
  - Convert back to character and print

More fun with strings

- Index into a string (more on array indexing later):
  - **name = "Golden Delicious"**
  - **name[0] is 'G'**
- Length of a string:
  - **len(name)   >>> 16**
  - **name[len(name) - 1] >>> 's' (last char)**
- Iterate over string:
  - **for idx in range(len(myString)):**
  - Or just: **for char in myString:**
- In Python, chars are just strings of length 1
  - In C, M2, etc., strings are arrays of characters

Test for 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
```

Case check, simplified

- Python can combine comparison operators:

```
if 5 < x < 12:
```

- So: uppercase/lowercase check, simplified:

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

Outputting just one character

- We want to process one character at a time
  - And output one character at a time
- But print always adds something to the output
  - Either a newline (print) or space (print ,)
- How to output exactly what we want?

```
import sys
sys.stdout.write( “Hello, World!” )
```

  - No newline unless it's in the string (“\n”)

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

Stub program: Python code

```
"""Convert to ASCII code and back."""
text = raw_input("Input text? ")
for char in text:              # iterate over string
    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 char in text:                   # iterate over string
    code = ord(char)
    if ord('a') <= code <= ord('z'): # lowercase
        code += 13
        if code > ord('z'):          # wraparound
            code –= 26
```

ROT13: Full program code, p.2

```
    elif ord('A') <= code <= ord('Z'):  #uppercase
        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?