

Functions: ROT13 example

12 Oct 2010

CMPT140

Dr. Sean Ho

Trinity Western University

Outline for today

- Call-by-**value** vs. call-by-**reference**
- **ROT13** “secret decoder ring” example
 - Problem restatement, **pseudocode**
 - **String** operations, **comparing** strings
 - Using **stub** functions
 - Writing functions to be **reusable**
 - Writing **testbeds**

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**

● *(Technically, it's “call-by-object” (ref))*

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 single string (**cleartext**)
- Computation:
 - For each letter in the string:
 - ◆ Convert letter to **numerical** form
 - ◆ Add **13** and wrap-around if necessary
 - ◆ Convert back to **letter** form
- Output:
 - The string converted to ROT13 encoding (**ciphertext**)

ROT13: convert A-Z to 1-26

- How do we convert a single letter from a character to a **numerical** code?

- Use `ord(char)`: try this out in **IDLE**

- Or write a **testbed** program:

```
char = 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 later with lists):

```
name = "Golden Delicious"
```

```
name[0]          # returns 'G'
```

- **Length** of a string:

```
len( name )     # returns 16
```

- **Iterate** over a string:

```
for char in name:
```

- In Python, **chars** are just strings of length 1
- In C++/Java/M2, strings are **arrays** of chars

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
```

Lexicographic sorting

- Python can directly compare strings, using **lexicographic** sorting:
 - Uses ordering of **ASCII** codes: e.g., digits < uppercase < lowercase
 - Compares one **letter** at a time (l-to-r)

'999' < 'APP' < 'App' < 'Apple'

- Simplifies checking case:

```
if 'a' <= char <= 'z':           # lowercase  
elif 'A' <= char <= 'Z':        # uppercase  
else:                            # non-letter
```

Using Python string methods

- Python strings also have **methods** (functions) that we can use: if `myStr` is any string var,
 - `myStr.upper()` returns an **upper-cased** version of the string `myStr`
 - `myStr.isupper()` returns `True` if **uppercase**
 - Similarly for `.lower()/.islower()`
- So the **Python** way to check case of var `'char'`:
if char.islower():
elif char.isupper():
else: # non-letter

Stub function: pseudocode

- For each character in the string:
 - Convert to **ASCII** numerical code
 - Convert back to **character**
 - ◆ *(skip doing the ROT13 for now!)*
 - **Print** ASCII code and converted character
- This **stub** function allows us to test the char<->ASCII **conversion** process and the **string indexing**
- Tackle the **ROT13** processing later

Stub function: Python code

```
"""Convert to ASCII code and back."""
```

```
text = input("Input text? ")
```

```
for char in text:           # iterate over string
```

```
    code = ord(char)
```

```
    char = chr(code)
```

```
    print(char, code, end=' ')
```

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

Writing reusable functions

- A function is a block of code which can be **reused** (invoked) in many different ways
 - Input is via **parameters** (not `input()`!)
 - Output is via **return** value (not `print()`!)

```
def rot13( cleartext ):
    ...
    return ciphertext
```

- Put the **user interface** (`input()/print()`) in a **testbed** which calls the core function:

```
def main():
    print( rot13( input( "Text? " ) ) )
```

ROT13: Full Python code

```
def rot13( cleartext ):
    """Apply ROT13 encoding."""
    ciphertext = ""           # init empty string

    for char in cleartext:   # iterate over string
        code = ord(char)

        if char.isupper():   # uppercase letter
            code += 13

            if code > ord('Z'): # wraparound
                code -= 26
```

ROT13: Full Python code, p.2

```
elif char.islower():      # lowercase letter
    code += 13

    if code > ord('z'):    # wraparound
        code -= 26

    char = chr(code)      # convert to char
    ciphertext += char    # append

return ciphertext
```

Full source: <http://twu.seanho.com/python/rot13.py>

Testbed program

- The **core** functionality is in the **rot13()** function
 - Can provide this function for **others** to use
- Put **user interface** in separate **testbed** function:

```
def main():
```

```
    """Testbed for ROT13."""
```

```
    text = input("Clear text? ")
```

```
    encoded = rot13( text )
```

```
    print("ROT13:", encoded)
```

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 then to quit?