

Lab 7 Feedback

- Missing function comments
 - Others need to know *how to use* the function
 - Good comments in lab description

Mar 13, 2012

Sprenkle - CSCI111

1

Comment Example

```
# Encodes a single character.  
# PRE: Input parameters are a single, lowercase  
# character string (char) and an integer key  
# (between -25 and 25, inclusive)  
# POST: returns the encoded character  
def translateLetter(char, key):
```

- Does not say *who* called function, where parameters came from, or where returned to
 - Any code can call the function and pass in input from anywhere (e.g., hardcoded, from user input, ...)
- Does not say variable name returned

Mar 13, 2012

Sprenkle - CSCI111

2

Commenting Exercise

- Write a comment for this function:

```
def encode(toEncode, key):
```

Mar 13, 2012

Sprenkle - CSCI111

3

Commenting Exercise

- Write a comment for this function:

```
# Encodes a lowercase string using a key,  
# preserving spaces in the string.  
# PRE: a lowercase string to be encoded  
# (toEncode) and the integer key (between -25  
# and 25, inclusive)  
# POST: returns the encoded string  
def encode(toEncode, key):
```

Mar 13, 2012

Sprenkle - CSCI111

4

Commenting Notes

- Well-named parameters make documentation easier
- I'm not strict on the pre/post format.
- Just need to be clear on
 - what the function does (at high level)
 - types of parameters
 - type of the output
- ➔ The caller knows what to pass to the function and if they should assign the output to a variable

Mar 13, 2012

Sprenkle - CSCI111

5

Caesar Cipher w/Functions

```
def main():  
    text = raw_input("Enter some text: ")  
    key = input("Enter an integer key (between -25 and 25): ")  
    # make sure it's a valid key  
    if key < -KEY_BOUND or key > KEY_BOUND:  
        print "Invalid key!"  
        sys.exit(1)  
    message = encoder(text, key)  
    print "The encoded message is", message  
  
# encoder takes in some text and integer key and returns  
# encoded message  
# PRE: Key must be between -25 and 25 inclusive  
def encoder(text, key):  
    message = ""  
    for ch in text:  
        if ch == " ":  
            encode = " "  
            message += encode  
        else:  
            message += translateLetter(ch, key)  
    return messagekey
```

More efficient: constants
not defined in function

Note: no "side effects"
e.g., no printing

Fines

Define a function that takes as parameters the speed limit and the clocked speed and returns the computed fine.

```
def computeFine(clocked_speed, speed_limit):
    if clocked_speed <= speed_limit:
        fine=0
    else:
        if clocked_speed > 90:
            fine = 50+5*(clocked_speed - speed_limit) + 200
        else:
            fine = 50+5*(clocked_speed - speed_limit)
    return fine
```

Missing comments

Mar 13, 2012

Sprenkle - CSCI111

7

Fines

The code that calls the function will print an appropriate message based on the returned fine.

Then, put the driver part of the program (i.e., the part that gets input from the user, calls the function, and displays the output) into a main function.

```
def main():
    speed_limit = eval(input("What is the speed limit? "))
    clocked_speed = eval(input("What was the speed? "))
    fine = computeFine(clocked_speed, speed_limit)
    if fine > 0:
        print("You were speeding. Your fine is", fine)
    else:
        print("The clocked speed is under the speed limit.")
        print("No fine. Continue safe driving.")
```

Missing comments

Mar 13, 2012

Sprenkle - CSCI111

8

Printing Cases Left

```
print("Cases Left to Choose from:")
# number of columns printed
colCount = 0
for casePos in range(len(cases)):
    # only print if the case hasn't been chosen yet
    if cases[casePos] != CHOSEN:
        print("%3d" % casePos, end='')
        colCount += 1
    # print a newline after every fourth printed case
    if colCount % 4 == 0:
        print()
print()
```

Mar 13, 2012

Sprenkle - CSCI111

9

Printing Board

```
border = "*****6"
print()
print(border)
print(" The Board: ")

for count in range(len(amounts)//2):
    if amounts[count] != CHOSEN:
        print("%10.2f" % amounts[count], end='')
    else:
        print("%11s" % "----", end='')

    second_col = len(amounts)//2 + count
    if amounts[second_col] != CHOSEN:
        print("%10.2f" % amounts[second_col])
    else:
        print("%11s" % "----")
print(border)
```

Mar 13, 2012

Sprenkle - CSCI111

10

Checking Valid Case

```
if choice < 0 or choice > len(cases)-1:
    return False
if cases[choice] == CHOSEN:
    return False
return True
```

Note that something is returned in **all** cases

```
if choice >= 0 and choice <= len(cases)-1 and
cases[choice] == CHOSEN:
    return False
```

Risky to not return True at the end of function so that something is returned in all cases

Mar 13, 2012

Sprenkle - CSCI111

11

Review

- How do you create a module?
- How do you use functions defined in a module?
- How do you make something repeat until a certain condition is true?
- How do you write code to do exception handling?
 - What are examples of exceptions?

Mar 13, 2012

Sprenkle - CSCI111

12

Lab 8 Overview

- Creating and using a module
- Indefinite Loops
- Exception Handling