CMPT120: Introduction to Computing Science and Programming I

Instructor: Hassan Khosravi

Summer 2012

Assignment 2

Due: Wed July 4th

This assignment is to be done individually.

The university policy on academic dishonesty (cheating) will be taken very seriously in this course. You may not discuss the specific questions nor their solutions with any other student. You are encouraged to discuss the general concepts involved in the questions. Please take advantage of office hours offered by the instructor and the TAs if you are having difficulties

Part I: Written

Create a text file named part1.txt that contains the answers to these questions.

1. Have a look at the documentation on the standard Python modules. Find two functions in different modules and explain what they do (in your own words), explain what type of argument(s) they take (string, integer, etc.) and what kind of value (if any) they return. Give some examples of their use and the return values that would be provided. Some modules that might be fruitful are: random, calendar, math.

2. Consider this Python code:

Line #	Code
1	def func (a):
2	# line 2
3	a = 1
4	# line 4
5	a = func2 (a,a)
6	# line 6
7	def func2(a,b):
8	# line 8
9	b = a+1
10	return b
11	a = 6
12	# line 12
13	func (a)
14	# line 14

a. List the line numbers in the order they are executed. (You can ignore the blank lines and `def` lines.)
b. What is the value of `a` when line 6 is executed? (i.e. when line 6 actually runs, not when it is first seen by the interpreter)
c. What is the value of `b` when line 14 is executed?
d. Is the value of `a` on line 14 different from the value of `a` on line 12? Explain.

3. Give the running time of these algorithms. Assume the value of n has already been set; express the running time in terms of n. For each of them, the number of "steps" is the number of times the calculation in the loop is repeated.

a.
```
x = 0
for i in range(0, n, 2):
    for j in range(n)
        x = x + i
print x
```

b.
```
x = 0
for i in range(n):
    x = x + 4
print x
```

4. What does the second program in the previous question [3(b)] calculate? That is, what is in x at the end of the program? Your answer should be a formula/ description in terms of n.

Write pseudocode for an algorithm with a smaller running time that does the same thing.

Part II: Programming

For this assignment, your job is to create a "Hangman" game played by two players. For those who have never played Hangman, the rules are simple. One player thinks of a secret word or phrase, and the other tries to guess it, letter-by-letter.

Just in case you haven't played hangman before, we have provided a sample of what it looks like when played by hand.

Game play should (eventually) look like this:

```
Enter the secret word: LETTERS

------------------------------------------------------------

Word so far: _______
Misses: 0
What letter would you like to guess? E

Word so far: _E__E__
Misses: 0
What letter would you like to guess? R

Word so far: _E__ER_
Misses: 0
What letter would you like to guess? A

Word so far: _E__ER_
Misses: 1
What letter would you like to guess? T

Word so far: _ETTER_
Misses: 1
What letter would you like to guess? S

Word so far: _ETTERS
Misses: 1
What letter would you like to guess? B

Word so far: _ETTERS
Misses: 2
What letter would you like to guess? Q

Word so far: _ETTERS
Misses: 3
What letter would you like to guess? L

You guessed the secret correctly: LETTERS
```

There is another page of sample runs that you can look at as well.

There is no general way to clear the screen in Python, but you don't want to leave the secret word(s) on the screen when the guessing starts. The easiest way to do this is to scroll the secret off the screen. You can do this by printing a bunch of line breaks:

```
print "\n" * 60
```

In sample runs, this will be represented with a row of dashes. (You really don't want to scroll down through 60 blank lines, do you?)

Creating the Program

Here are the things that you need to keep track of in the main part of the program:

- The secret word/words that the first user enters (`secret`).
- The displayed version of the secret (`display`). This will be what the second user sees as they are guessing. It will start with all underscores (e.g.____), be filled in with letters as they are guessed (e.g. `_M_T`), and for a winning game, will end up the same as the original secret entered by the first user (e.g. `CMPT`)
- The number of incorrect guesses (`misses`) that the user has made.
- The number of letters that are still unguessed in the secret.
- Whatever else you need.

For this program, you must break the task up into functions. Here is a guide to creating the program for this problem:

1. Start by creating a function `get_secret` that asks the user for the secret word/phrase, clears the screen, and returns the secret string.

```
def get_secret():
    """
    Get the secret word/phrase.
    """
```

2. In the main part of the program, call the `get_secret` function and store the result in `secret`. Create a string `display` that contains `len(secret)` underscores.

   The string `display` will be what we show the user as they are guessing. The underscores will be replaced with letters as they are guessed.

   Create a variable to count the number of incorrect guesses the user has made ("misses") and initialize it appropriately. You will also need to keep track of the number of unguessed letters left in the secret: this (along with the count of the misses) will be used to decide when the game is over.

3. Create a function `do_turn` that takes two arguments: the `display` string, and the number of misses made so far. This function should do the part of the turn the user sees (display the part of the secret they have guessed, display the number of misses, and ask them to guess a letter).

```
def do_turn(display, misses):
    """
    Display the current status for the user and let them
make a guess.
    """
```

Don't worry about the error checking (exactly one letter). The function should return the letter the user enters.

4. Once we have both the secret word, and a guess, we need to be able to update the `display` string, and keep track of the number of letters discovered. Create a function `new_display` that takes three arguments: the `secret`, the `display` string, and the letter the user guessed.

   Once it has these values, the function can calculate the new value for the display string (i.e. replace all of the underscores where the letter the user guessed is the letter). While it does this, it can count the replacements.

   Both of those values (the new `display` string string, and the replacement count) should be returned. A Python function can return multiple values like this:

   ```
   return newdisp, count
   ```

   Then, you can call the function and capture both return values like this:

   ```
   display, count = new_display(…)
   ```

   Here are some examples of calling this function, and the values it should return in each case:

   ```
   new_display("SECRET", "_E__E_", "C") == "_EC_E_", 1
   new_display("SECRET", "_E__E_", "Q") == "_E__E_", 0
   new_display("ABBA", "A__A", "B") == "ABBA", 2
   ```

   Hint: Create a `for` loop that examines the characters in the old display string (and the secret string: the characters in each position should correspond). For each one, either copy it to the new display string, or replace it with the guessed letter. Here's a partial function (i.e. it's partially complete, but not totally finished after filling in the blanks):

```
def new_display(word, display, letter):
    newdisp = ""
    for i in range(len(word)):
        if [redacted] == letter:
            newdisp = newdisp + letter
        else:
            newdisp = newdisp + [redacted]
    return newdisp, [redacted]
```

5. In the main part of your program, create the main loop for the game. In it, you should:

a. Call `do_turn` to do the part of the visible to the user, and get the letter they want to guess.
b. Call `new_display` to update the `display` string and get the number of occurrences of the letter.
c. Update the variables holding the number of misses and the number of unguessed letters as appropriate.

If you do these things correctly, you should have everything you need to create a condition for this loop. The loop should continue when there are both letters remaining to be guessed and less than six misses have been made.

As you're working on your program, you may find that your loop doesn't exit properly. Press control-C to stop your program.

6. After the loop, you should print out an appropriate win/loss message as seen in the sample runs.

7. Add code where appropriate to make sure the user enters exactly one character when prompted for a guess. Display an error message if they don't.

Notes

You may find that the tasks described above can be further broken up to subtasks that you can place into other functions. You should do that whenever you think it's helpful. You can, of course, call any one of the above functions or any other function if you find it handy.

When you play the game, it should look like the example given above (i.e. all the messages, prompts, etc. should be the same).

Your code should be easy to read and understand. This includes (but isn't limited to) good variables names, code formatting, comments, and docstrings.

Submitting Your Assignment

You must create your assignment in electronic form and submit it online.

Put both files (part1.txt and your code for the second part) in a zip file. There are some instructions in the website if you have trouble with zipping your files. Call the zip file assignment1 (so it would be assignment1.rar or assignment1.zip).

Login to your account in https://courses.cs.sfu.ca/. You should be able to upload your file when you go into assignment2.