CSC 106 Assignment 3: Guess Who's Programming in Scratch and Python!

Marks: 20 marks.

Learning Goals:
At the end of this assignment you will be able to:

- Implement pseudocode in two different programming languages
- Create, from "scratch" a Scratch program to solve a guessing game problem
- Create, from "scratch" a Python program to solve a guessing game problem
- Assess the pros and cons of two different programming environments: Scratch and Python
- Explore the concept of "pair programming".

Submission: This assignment may in pairs. You may also work individually if you prefer. The submitted assignment will have two components:
One component will be a **Scratch file (.sb) file called GuessIt.sb**
The other component will be a **Python file (.py) called GuessIt.py** –

Your .sb program MUST run with Scratch 1.4

Your .py program MUST run with Python 2.7 – do NOT use Python 3.X

All files should be attached to the CSC106 Assignment 2 submission page on Connex.

If you are working as a pair, only ONE member of the pair submits the assignment.

A few reminders:
---- Make sure that both partner's NAMES are in the Assignment Text box on Connex. As well, please make sure that the names of the partners are in the Project Notes in the Scratch program and in a comment at the top of the code in the Python program.

- MOST IMPORTANT: ONLY ONE ASSIGNMENT SUBMITTED PER TEAM! –

Resources:

- **You DO NOT need to install or use Python on your own computer -- every machine in the computer science building has Python on it.**
- The main Python resource: https://www.python.org/about/gettingstarted/

Special resources for Mac Users:

- Python 2.7 is already pre-installed on Mac OS X (Mavericks) machines. Although I haven't tested our assignment on an OS X machine, we don't think you'll need anything more than what comes pre-installed. Here is a quick video showing you how to access and use Python on a Mac (OS X) https://www.youtube.com/watch?v=BE1wDsLzOJA
- An excellent text editor for Mac Users – check out this great video on how to use TextWrangler for Mac with Python: https://www.youtube.com/watch?v=et2vjUAz9-k
- If you want more bells and whistles for Python on your Mac this web page give you some instruction for installing Python on your Mac – step by step (caution though – this can get VERY complex and we can't help you out if things go south): http://www.pyladies.com/blog/Get-Your-Mac-Ready-for-Python-Programming/

Special resources for Windows Users:

- Sadly, Windows doesn't come with Python installed. If you want to use Python on your own Windows 7 or Windows 8.x machine, you need to install it. It is relatively straightforward, but, again – if something goes wrong, we can't help you out.
  Install info for Windows 7 (you do not need Mechanize, Requests, Beautiful Soup, or CSV components – no need to download and install): http://www.anthonydebarros.com/2011/10/15/setting-up-python-in-windows-7/
  Install info for Windows 8.1(you do not need Mechanize, Requests, Beautiful Soup, or CSV components talked about towards the end of the instructions, so don't bother downloading and installing them.): http://www.anthonydebarros.com/2014/02/16/setting-up-python-in-windows-8-1/
- TextPad text editor for use with Windows: https://www.textpad.com/download/index.html
- NotePad++ http://notepad-plus-plus.org/

Special resources for running Python online (NO INSTALL NECESSARY):

- You've read all the install info for your computer and have decided it is just too time consuming / complex / whatever – what to do? If you don't want to install Python on your computer you can still write and practice your own Python code through the use of an online Python interpreter. There are a number of them, but check out Repl: http://repl.it/languages and click on Python. Remember to save your work!

- Scratch reference: http://scratch.mit.edu/help/
- Scratch lessons (video): http://learnscratch.org/video-courses
- Scratch ideas: http://scratch.mit.edu/
- Installing Scratch v. 1.4 on your own computer: http://scratch.mit.edu/scratch_1.4/

TASK 1: Implement the Guessing Game in Scratch (10 marks)

In class we came up with an algorithm to play a number guessing game. The number guessing game worked as follows:

- The program picks a number between 1 and 100, and the player tries to guess the number.
- Each time the player guesses, the program will tell the player if the guess is correct, too high, or too low. (A flow chart and pseudocode is provided on the last page of this document.)

1. (6 marks) Try your hand at implementing the Guessing Game in Scratch. You can implement it any way you want – top marks are given for a game that is visually interesting and engages the user in some way over and above having the user simply put in the required numbers.

- *3.5 to 4.5 given for a game that meets the basic specifications. The marker will play the game several times and examine the code itself.*
- *4.5 to 6 marks given for a game that is visually interesting and engages the user in some way over and above having the user simply put in the required numbers. Certainly, this last component is a bit subjective, but have some fun with it if you decide to go beyond the basic specifications.*
- *You will lose 3 marks for not providing program notes*

2. (4 marks) Whenever user input is requested, you can be sure errors will result! In your program notes, describe how your program would handle bad input at any stage of the game – e.g. say the user entered letters, or a decimal value, instead of an integer. You do not have to implement this error checking in your Scratch code but you must clearly describe how, where and what you would do to introduce error checking.

- *3-4 Marks for error checking – top marks for ideas that handle error checking at the beginning and throughout the game.*
- *1-2 Marks for minimal error checking.*
- *You will lose up to 3 marks for poor commenting*

TASK 1 DELIVERABLES: A .sb file called GuessIt.sb

TASK 2: Implement the Guessing Game in Python (10 marks)

In class we came up with an algorithm to play a number guessing game. The number guessing game worked as follows:

- The program picks a number between 1 and 100, and the player tries to guess the number.
- Each time the player guesses, the program will tell the player if the guess is correct, too high, or too low. (A flow chart and pseudocode is provided on the last page of this document.)

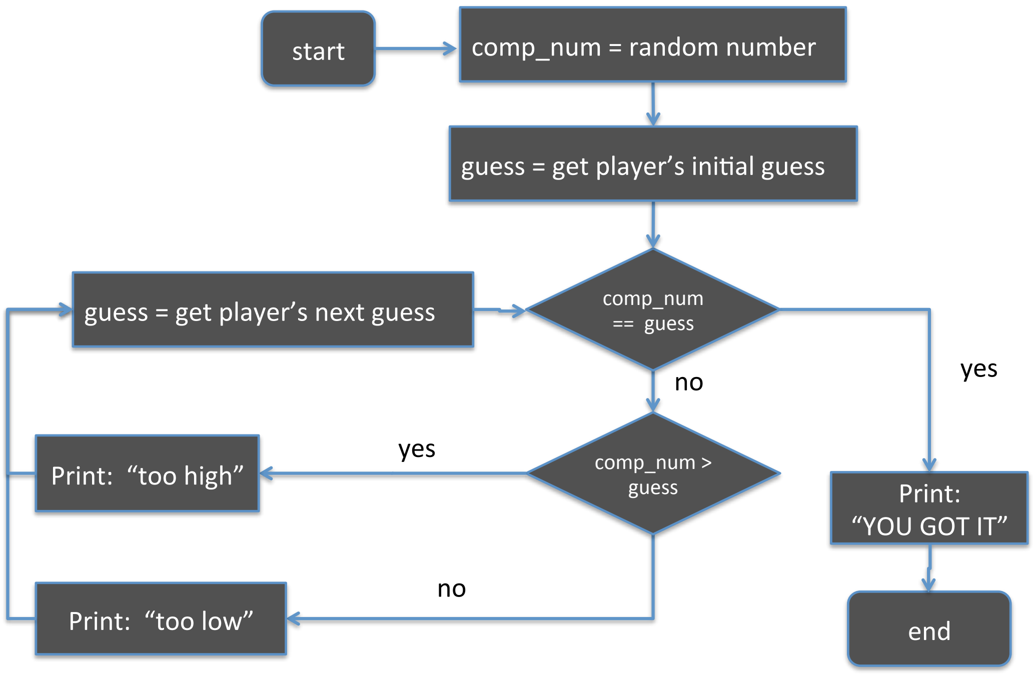
1. (6 marks) Try your hand at implementing the Guessing Game in Python. You can implement it any way you want. The game must compile, run, and allow the user to correctly play the game. Top marks are given for a game that is well commented throughout and that somehow engages the user a bit more than strictly asking for number entry – perhaps something like giving a hint after a certain number of unsuccessful tries, or suggestions about getting "warmer" or "colder".

2. (4 marks) Whenever user input is requested, you can be sure errors will result! Have your program handle bad input at any stage of the game – e.g. say the user entered letters, or a decimal value, instead of an integer. Implement error checking in your Python code. Our marking program will test on letters and decimal numbers and will test at various stages in the game.

- *3-4 Marks for error checking – top marks for ideas that handle error checking at the beginning and throughout the game.*
- *1-2 Marks for minimal error checking.*
- *Make sure to add comments where you are error checking to indicate what type of errors you are checking for.*
  The key to this question -- don't make the marker have to guess at what you have done.

TASK 2 DELIVERABLES: A .py file called GuessIt.py

Guessing game flowchart



guessing game pseudocode

Assumptions: **comp_num** will be an integer taken at random between 1 and 100
players_guess will be an integer between 1 and 100.

```
GuessingGame
    comp_num = an int selected at random between 1 and 100
    players_guess = an int entered by player between 1 and 100

    WHILE players_guess != comp_num

        Check IF players_guess > comp_num
            PRINT "Your number is too high. Guess again."
        ELSE
            PRINT "Your number is too low. Guess again."

        players_guess = an int entered by player between 1 and 100
        // player enters a new integer in players_guess and it is
        // checked for equality with comp_num
    END WHILE
    Print "You got it – finally!"
```