Contents

Part 1: The overall goal

Part 2: The approach

Part 3: Designing

Part 4: Hints (Functions)

Part 5: Hints (special string functions)

Part 6: Finished? Part 7: A very big hint

The goal

In this lab you will write a Python program that plays a game of hangman. You may do this any way you wish as long as you make use of at least two function definitions.

In the game hangman one player selects a "secret word" and the other player has to guess what the word is. The game begins when the first player selects a word, tells the second player how many letters are in the word (by drawing some "blanks"), and draws a gallows.

The second player is then given a number of turns to guess letters that are contained in the word. Each time they guess a letter correctly they are rewarded by having the appropriate blanks filled in. Every time they guess incorrectly the punishment is that a portion of a "hanged man" is drawn on the gallows. When the man is complete (usually after six bad guesses) the player has lost.

Approach to the interaction

You can store a list of secret words and pick one at random; use input to read input from the user. For output you should use the print() function. Just draw the output for each round in the window. The output does not have to be fancy -- for the first iteration it would be enough to print something like: Guess A letter: \_ \_ \_ \_ \_ \_ (6 guesses left)

You guessed 'A'. No 'A's in the word! Guess the word: \_ \_ \_ \_ \_ \_ (5 guesses left)

You guessed 'E'. 2 'E's in the word Guess the word: \_ E \_ E \_ \_ (5 guesses left) You can spruce up the output later on (see below). It will be fairly easy as long as you design the program in a modular way. Designing the program

You should approach writing the program like this:

#希望你事先想好程式整個結構

First, sketch the design of the program on paper (this is much more efficient than trying to write the while thing straight into the editor). Decide what functions you will write (see below for some suggestions), and sketch a main program that will use them to implement the full game. You don't have to think about how the functions will work yet. There are many possible ways you could design the program; once you have tried this step, you can look at one possible solution, below. Do not do this until you have made an attempt yourself -- it will be much more valuable to look at the example after you have thought about the solution.

#注意函數所需要的參數和傳值

Next, look at each of the functions and decide what parameters each will need, and what they will return. You may find at this stage that you want to revisit your sketch of the main program now that you have thought more about the functions -- that's probably a good idea.

#希望你想好所需要的函數，不會讓你的主程式變得很複雜

Next, you want to think about writing the functions. It's probably OK to start programming at this point, by the way. It is a good idea to write just one function, and then write a few simple statements to test it before proceeding. Once you are confident that the function works you can move on to write the next one, and so on. The main program that makes use of the functions can be done last -- if you have chosen the functions well then it should not be complex.

You will need several variables in this program: A string to contain the secret word Another variable to contain the letters that have been correctly guessed. I suggest this starts out empty, and every time the player guesses a letter correctly the letter is added. When each letter in the secret word is contained in this word the secret word has been guessed correctly (even if they are not stored in the correct order in this variable). A number to contain the count of remaining guesses (or, if you prefer, the number of incorrect guesses made -- up to you!). Pay careful attention to the difference between local (within a function only) and global (throughout the program) variables. Try to avoid using too many global variables if you can (the ones mentioned above might be all you need!). Note also that even though you may be dealing with text a Python string isn't always the most useful tool for this; you cannot change individual letters in a string easily while storing the letters in a list does allow that.

Choosing your functions

One of the hardest parts of program design is choosing exactly how you are going to subdivide the problem into sub-parts like functions. Since this is your first program on this scale, here are some suggestions for suitable functions to write (and some suggested names). You don't have to follow these suggestions, of course, feel free to explore the design yourself.

#part1—輸入密語

A function to read the secret word from the first player (you could call it get\_secret). If you are working with a built in list of words then the function could just choose one instead of using input. Up to you.

#part2:uess A letter: \_ \_ \_ \_ \_ \_ (6 guesses left)

A function to read a guess from the second player (get\_guess). This function should read a string, and then check that there is only one letter in the string. It might be a good idea to loop if the player has entered too many (or too few) letters. If you feel ambitious you could even loop when they enter a letter they previously guessed, but that's not really necessary. Remember that if the variable s is a string then you can find out how many letters are in it by writing len(s).

#part3:check

A function to check a guess against the secret word (check\_guess).

#part4: You guessed 'E'. 2 'E's in the word Guess the word: \_ E \_ E \_ \_ (5 guesses left)

A function to print into the document (using document.write) the current status of the game (you could call this print\_move). This should just print out the current status so the player knows where they stand. The easiest way to do this is to write a loop that looks at each letter in the secret word and prints either an underscore or the letter, depending on whether the letter is also contained in the list of correctly guessed letters (which you can check using indexof).

#part5:檢查剩下幾個字，猜完則結束

A function to count the number of blanks that are left in the word. This is useful for working out whether the game is over! You can take a similar approach to the printing function.