There are 2 projects for Lab 5, Chapter 5, the Hangman Game and one additional project.

1. **REQUIRED:** Complete the “Hangman Game” for Chapter 5. Use ‘HangmanGame-username.py’ as the file for the filename.
2. Write a program in which you manually sort a list of integers from 0 to 9. You can use the following code at the top of your program to generate and randomize the list unordered:

import random

unordered = list(range(10))

ordered = []

random.shuffle(unordered)

You can use an established method to sort the list called the Selection Sort. Here’s the pseudocode for an implementation of the Selection Sort algorithm that removes elements from an original, unordered list and appends them to a new, ordered list:

***Create an empty list to hold the ordered elements***

***While there are still elements in the unordered list***

***Set a variable, lowest, to the first element in the unordered list***

***For each element in the unordered list***

***If the element is lower than lowest***

***Assign the value of that element to lowest***

***Append lowest to the ordered list***

***Remove lowest from the unordered list***

***Print out the ordered list***

An important point to remember is that Python’s implementation of list sorting is far more efficient than this implementation of the Selection Sort. So, when you want to sort a list (other than for this project), you should use the built-in sort() list method.

1. Write a program that creates a dictionary of son-father pairs–you can use celebrities, fictional characters, or even historical figures for fun. Your program should present the user a menu with three options. The following is an example:

**Father Finder**

**0 - Quit**

**1 - Find a Father**

**2 - Find a Grandfather**

**3 - List all the sons/keys**

* Option 0 should end the program.
* Option 1 should prompt the user for the name of a son. If the dictionary contains the son-father pair, the program should display the father. Otherwise, the program should tell the user it doesn’t know who the father is.
* Option 2 should prompt the user for the name of a grandson. If the dictionary contains enough information, the program should display the grandson’s grandfather. Otherwise, the program should tell the user it doesn't know who the grandfather is.
* Option 3 should provide all the names of the sons/keys.
* The user should be informed if one of the three are true, the son does not exist or the father does not exist or the grandfather does not exist in the dictionary.

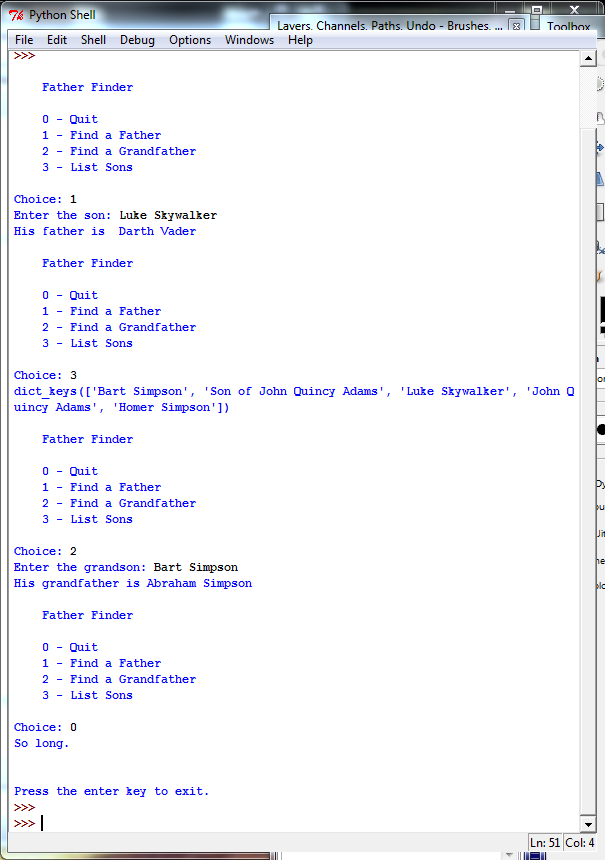
The ONE dictionary you create should include several generations so that a grandfather can be found given the name of a grandson. Let’s start with

Table 1-Sons&Fathers

|  |  |
| --- | --- |
| John Quincy Adams | John Adams |
| Bart Simpson | Homer Simpson |
| Homer Simpson | Grandpa Simpson |
| John Adams | John Adams Sr. |

Put the entries above into a dictionary, and add your own son/father pairs.

Script name should be ‘SonToFather-username.py’. Here is a simulation:



Each file should be a python program, a text file, extended by ".py". The header of each file should be comments which tell the name of the program, your name, and the date.