Unit 5

Apply what you know

0. Study the programs in the *Learn something new* section until you can write them yourself from scratch without relying on this document or any other source of information. Here are the programs:

    0.1. Write a program that lists all items in the current directory. Modify the program so that it lists all items in the parent of the current directory.

    0.2. Write a program that lists all items in the parent of the current directory, flagging directories in the listing with stars.

    0.3. Modify the previous program so that the listing code is contained in a function called `lister` that takes a path and lists items in the location specified by the path, flagging directories with stars. Call `lister` with both the path to the current directory and the path to its parent.

    0.4. Modify `lister` so that it not only flags directories with stars but also lists their contents.

    0.5. Modify `lister` again, so that subdirectory listings are indented appropriately.

    0.6. Modify `lister` again so that only directories and Python program files are included in the listing.

1. Write a program that prints the name Fred 100 times, one time per line. Do not use `for` or `while`. Instead, create a function called `printFred` that takes in a number and prints Fred that number of times. The function operates by displaying the name Fred once and then calling itself with the next smaller number, if that number is at least one. The call `printFred(100)` should do what we want.

    A function that calls itself is said to be **recursive**. As this program demonstrates, recursion can always be used in place of loops.

2. Write a program that lists all files in the current directory that contain the string `'random'`.

3. Write a program that reports the total number of files in the current directory and any subdirectories, subsubdirectories and so on.

4. Write a program that *solves* the word-scramble puzzles produced by Program 2 in Unit 4. The user enters a scrambled string of letters and the program responds by

displaying all words in *Pride and Prejudice* that can be formed by rearranging these letters.

For this program, you should create a dictionary that has alphabetized strings of letters as keys and lists of words that can be formed with those letters as the associated values. For example, the key '`acer`'—with the letters in alphabetical order—would have the value ['`care`', '`race`']. These four letters can also be rearranged to form the word '`acre`', but this is not used in *Pride and Prejudice*.

To put a string of letters into alphabetical order, first use it to create a list. Next, use the list method `sort` to put the list in order—if your list is called `letters`, just write `letters.sort()`. Finally, use the `rejoin` function you wrote for Program 2 in Unit 4 to convert the list back into a string.

5. Write a program to find the five most common words in *Pride and Prejudice* ending with 'ing'. Start by creating a dictionary of counts for all words ending in 'ing'. Then use the dictionary to create a list like this:

```
[[5, 'walking'], [17, 'sing'], [2, 'balancing'], ...]
```

Each element is a two-item list containing a count and a word. Call the list method `sort` on this list of lists. The sorting will be done using the first item of the two-item lists—that is elements will be sorted from lowest counts at the beginning to highest counts at the end. The last five elements will then be the ones we want. Use a slice to select them.

To process all the items in a dictionary, use this pattern:

```
for key in dictionary:
```

The name `key` will be assigned the keys used in `dictionary`, one at a time, though not in any predictable order.

6. Write a program to choose a random five-card hand from a standard deck and then report if it is a one-pair hand (two cards of one face value and the other three all different face values), a two-pair hand (two cards of one face value, two cards of another face value and a fifth card of a third face value), a three-of-a-kind hand (three cards of the same face value and two others of different face values, a full house (three cards of one face value and two of another) or a four-of-a-kind hand (four cards of one face value and a fifth of a different face value).

Start by creating a dictionary of counts for the face values. Use this to form a list of just the counts. Sort the list. The result makes it easy to classify the hand. For example, if the sorted list of counts is `[2, 3]`, the hand is a full house.

You should, of course, reuse functions compiled in your `cards.py` module.