Activity 2-6

March 15, 2012

Task 1: Python Dictionary Review

Download and save `ACT2-6.py`. Open it in `IDLE` and press `F5`.

1. Look at the dictionary called `passwordDictionary`. This is a list of the 25 easiest passwords to break in 2011. First, make sure your password *doesn't* appear on this list: if it does, you should change it after class!

2. Run the function `printDictionary` with `passwordDictionary` as input. Make sure you understand how this function works (it is similar to a function from last class).

3. Modify `printDictionary` so it prints the keys in alphabetical order. Use the `sort()` function, which works on a list. When you call `myList.sort()`, remember that it sorts `myList`! Try sorting a short list of strings.

4. Add the key `'george'` (lowercase 'g'!) with the password of your choice as the value. Run `printDictionary` again. How is the list sorted?

5. Change the key `'george'` to `'George'`. You actually need to do *two* things to `printDictionary` to achieve this.

Task 2: Write an `addPassword` Function

The `addPassword` function, as it's currently written, is a bit dangerous. It's fine if I add a *new* name and password, but I want to be sure that I don't accidentally overwrite an *existing* name and password.

1. Modify the `addPassword` function to print a statement that says `'Warning, overwriting an existing password!'` if you overwrite a password.
2. If I accidentally overwrote a password, then I've lost the original password. Modify the function to ask the user `'Do you really want to overwrite?'`. If the user enters `'y'` or `'yes'`, then overwrite the password. If the user enters `'no'` (or anything else), print `'Returning original dictionary'` and do not modify the dictionary.
3. I'm worried about someone else resetting passwords maliciously. Modify the function to prompt the user for the original password (the password that is about to be replaced). If that password is correct, then overwrite the password. Otherwise, print a statement and do not modify the dictionary.

Task 3: String Matching & Regular Expressions

In `File...Open`, select `Tools/Scripts/redemo.py` and open it. If you don't have this file, download it from the class website. Press `F5`: a box pops up! This is a demo that Python provides.

1. Copy and paste the output of the `print` statement into the middle box (under 'Enter a string to search:'). Also, select 'Highlight all matches'.
2. Enter some text in the first box (under 'Enter a Perl-style regular expression'). What happens to the text in the middle box? Enter a string to highlight all the suffixes of your name.
3. Enter the following lines into the top box. What do you think brackets do?

   ```
   n
   fn
   [fn]
   [aeiou]
   ```

4. Get all the *words* that that start with `b`,`f`, or `m` and end in your name (or the suffix of your name).
5. Enter the following lines into the top box. What do you think `\w` does? What do you think the `+` sign does?

   ```
   f
   f\w
   f\w\w
   f\w+
   \s
   \sm
   ```

6. Get all the words that start with `b`,`f`, or `m`.
7. Add some numbers to the middle box. Test what `\d` does.

Task 4: More Examples of Regular Expressions

Close the demo and return to `ACT2-6.py`. Download and save `poem.txt` in the same directory as `ACT2-6.py`. Press `F5`. Run the following statements and try to figure out what each line does.

```
1  myStr = readShel()
2
3  printRegEx('g\w+',myStr)
4  printRegEx('\sg\w+',myStr)
5  printRegEx('\s[gG]\w+',myStr)
6
7  printRegEx('out',myStr)
8  printRegEx('\sout',myStr)
9  printRegEx('\wout',myStr)
10 printRegEx('\w+out',myStr)
11 printRegEx('[\s\w]out',myStr)
```

Take a look at the code for `printRegEx`. First, note that there is an `import re` statement - this is *required* to use regular expressions.

Then, try to determine what this new thing called an *iterator* does. It's very similar to a list, but has some extra functionality.