CptS355 - Assignment 1 (Python)

January 13, 2017

Overview

Weight: This assignment will count for approximately 5% of your final grade.
Due Date: Wednesday, Jan 25 by 11:59PM

In this assignment you will explore use of some of the features of Python.

Turning in your assignment

Note the following directions carefully.
All the problem solutions should be placed in a single file named **HW1.py**.When you are done and certain that everything is working correctly, turn in your file by uploading on the Assignment-1(Python) DROPBOX on Blackboard (under AssignmentSubmisions menu). The file that you upload must be named **HW1.py**. Be sure to include your name as a comment at the top of the file. Also in a comment, indicating whether your code is intended for Unix/Linux or Windows – programs' behavior may differ slightly between the two systems; your code only has to work on one or the other. You may turn in your assignment up to 5 times. Only the last one submitted will be graded. Please let the instructor know if you need to resubmit it a 6th time. Implement your code for **Python 3**.

The work you turn in is to be **your own personal work**. You may NOT copy another student's code or work together on writing code. You may not copy code from the web, or anything else that lets you avoid solving the problems for yourself.

Grading

The assignment will be marked for good programming style (appropriate algorithms, good indentation and appropriate comments – refer to the http://www.python.org/peps/pep-0008.htmlPython style guide) – as well as thoroughness of testing and clean and correct execution. **For each problem below, around 20% of the points will be reserved for the test functions and the programming style.** We will talk about the "testing requirements" in class.

- Good python style favors for loops rather than while loops (when possible) for reasons discussed in class.
- Also, code to do the same thing (or something easily parameterizable) at different points in a program should not be duplicated but extracted into a callable function.
- Turning in "final" code that produces debugging output is bad form, and points will be deducted. Here's a trick. Near the top of your program write a debug function that can be turned on and off by changing a single variable.

```
debugging = True
def debug(*s): if debugging: print(*s)
```

Where you want to produce debugging output use debug(”This is my debugging output”) instead of print. (How it works: Using * in front of the parameter of a function means that a variable number of arguments can be passed to that parameter. Then using *s as print’s argument passes along those arguments to print.)

1. Warmup – `cryptDict()` and `decrypt()` – 35%

A text of a message can be decrypted by applying translation function to each character.

Define a function `cryptDict(s1,s2)` that returns a dictionary such that each character in `s1` is mapped to the character at the corresponding position in `s2`. You may assume that the characters in `s1` are unique and that the two strings are the same length. (“You may assume X” means that your code does not have to check whether X holds or not).
So for example, `cryptDict('abcdefghijk', 'kjihgfedcba')` returns:

```
{'a':'k', 'b':'j', 'c':'i', 'd':'h', 'e':'g', 'f':'f', 'g':'e', 'h':'d',
'i':'c', 'j':'b', 'k':'a'}
```

Important note: “A function returns a value” do not mean that it prints the value. Please pay attention to the difference.

Now, define a function `decrypt(cdict,S)`. decrypt translates its string argument, S, according to its decryption dictionary argument, `cdict`. Argument `cdict` is, of course, a dictionary returned by the `cryptDict` function, and the result of `decrypt` is obtained by replacing each character, c, of S by `cdict[c]`, if c is amongst the keys of the `cdict`. If however c is not in `cdict.keys()` then it is unchanged in the output. This is all harder to describe than to do! To easily look up k in dictionary, D, specifying that the same character is to be used if it is not present as a key in the dictionary you can use:

```
    trans = D.get(k,k)  # use help({}.get) to see more about get
```

The above trick with get to get a default value when a key is not present in a dictionary is important to learn and use when appropriate. **Points will be taken off for not using it when appropriate!**

Also define a function `testDecrypt()` that minimally does the following (more testing code would be good.):

```
# function to test translation code
# return True if successful, False if any test fails
def testDecrypt():
    cdict = cryptDict('abc', 'xyz')
    revcdict = cryptDict('xyz', 'abc')
    tests = "Now␣I␣know␣my␣abc's"
    answer = "Now␣I␣know␣my␣xyz's"
    if decrypt(cdict, tests) != answer:
          return False
    if decrypt(revcdict, decrypt(cdict, tests)) != tests:
          return False
    if decrypt(cdict,'') != '':
          return False
    if decrypt(cryptDict('',''), 'abc') != 'abc':
          return False
    return True
```

You can start with the following code:

```
def cryptDict(str1,str2):
    #write your code here
    pass
def decrypt(cdict,s):
    #write your code here
    pass
def testDecrypt():
    #write your code here -- see above
    pass
testDecrypt()
```

2. charCount() - 30%

Define a function, `charCount(S)` counting the number of times that each character appears in a given string. `charCount(S)` should return a list of characters in the input string S each paired with its frequency (as a 2-tuple). The "space character" should be excluded in the result. Characters must appear in the list ordered from least frequent to most frequent. For example, `charCount('Cpts355 --- Assign1')` is:

```
[('1', 1), ('3', 1), ('A', 1), ('C', 1), ('g', 1), ('i', 1), ('n', 1),
('p', 1), ('t', 1), ('5', 2), ('-', 3), ('s', 3) ].
```

Characters with the same frequency must appear in increasing alphabetical order. To implement the sorting you must use the Python built-in function sorted. Do not write your own sorting code. (help(sorted) and the Python documentation are your friends.)

(***Hints***: Use a dictionary to maintain the counts. But remember that your function should return a list of tuples. You should be able to build the dictionary in a single pass over the input string. Make sure to use the `get` function.)

Define a function `testCount()` that tests your `charCount(S)` function, returning `True` if the code passes your tests, and `False` if the tests fail.

You can start with the following code:

```
def charCount(S):
    #write your code here
    pass
def testCount():
    #write your code here
    pass
testCount()
```

3. dictAddup() - 35%

Assume you keep track of the number of hours you study every day for each of the courses you are enrolled in. You maintain the weekly log of your hours in a Python dictionary as follows:

```
{Monday:{'355':2,'451':1,'360':2},Tuesday:{'451':2,'360':3},
 Thursday:{'355':3,'451':2,'360':3}, Friday:{'355':2},
 Sunday:{'355':1,'451':3,'360':1}}
```

The keys of the dictionary are the days you studied and the values are the dictionaries which include the number of hours for each of the courses you studied. Please note that you may not study for some courses on some days OR you may not study at all on some days of the week.

Define a function, `dictAddup(d)` which adds up the number of hours you studied for each of the courses during the week and returns the summed values as a dictionary. Note that the keys in the resulting dictionary should be the course names and the values should be the total number of hours you have studied for the corresponding courses. `dictAddup` would return the following for the above dictionary:

```
{'355':8,'451':8,'360':9}
```

Define a function `testAddup()` that tests your `dictAddup(d)` function, returning `True` if the code passes your tests, and `False` if the tests fail.

You can start with the following code:

```
    def dictAddup(d):
        #write your code here
        pass
    def testAddup():
        #write your code here
        pass
    testAddup()
```

4. Main program

So far we've just defined a bunch of functions in our `HW1.py` program. To actually execute the code, we need to call those functions. Unlike in C or Java, this is not done by writing a function with a special name. Instead the following idiom is used. The "`__main__`" module represents the (otherwise anonymous) scope in which the interpreter's main program executes.

You can start with the following code:

```
    if __name__ == '__main__':
        #...code to do whatever you want done...
```

This code is to be written at the left margin of your input file (or at the same level as the def lines if you've indented those).

For this assignment, we want to run all the tests, so your main should look like:

```
if __name__ == '__main__':
    passedMsg = "%s␣passed"
    failedMsg = "%s␣failed"
    if testDecrypt():
       print ( passedMsg % 'testDecrypt' )
    else:
       print ( failedMsg % 'testDecrypt' )
    # etc. for the other tests.
    # notice how you are repeating a lot of code here
    # think about how you could avoid that
```

Note that having your code pass your tests is not proof of the code's correctness – but it is evidence. You can't prove a program is correct by testing, though you can prove it is incorrect!