SI 106 – Fall 2014 **Name: ___Solution Set_____**

Written Final Exam

This exam is closed book - you are allowed only one page of notes (double-sided). If a question seems unclear - please write down any assumptions you feel are needed. If you think that there is a just-plain mistake/typo - check with an instructor.

Anywhere we ask you what will be printed out, if you think an error will be generated, you may write "error". You do not need to write out what the whole error message would be.

1. (250) Which of the following are reasons to use a version control system like github? (Circle all that apply).

    a. You would like your code to be compressed so that it uses less space on your file system
    b. You would like to be able to see or revert to any past version of any of the files in your project
    c. You want to collaborate with others, working in parallel on a project and merging your changes together occasionally
    d. You would like your code to automatically be checked for syntax errors
    e. You would like to distribute your code in a public repository that others can easily fork or comment on

2. (300) Which of the following behaviors will be viewed **unfavorably** by most people who are active participants in open source projects? (Circle all that apply)

    a. Making a private fork of an existing project and modifying the code to meet the special needs of your application that are not shared by most other users of the project's code.
    b. Making a fork of an existing project, making major improvements to the code, and then answering questions about the project on StackOverflow by pointing people to your repository with improved code.
    c. Making an argument in an email on a project's public mailing list that a piece of existing code that was written by the project owner is really bad code and that it should be replaced by a piece of code that you have written.

3. (300) You have written a small UNIX utility program (analogous to cat or grep) for your own purposes. It's been very useful for you and you start to get requests from other people to share it. You don't mind sharing it freely, but you decide that it would bother you very much if somebody else started selling your code and keeping the money for themselves. What kind of license should you put on your code when you distribute it: GPL or BSD? Justify your answer in 2-3 sentences.

    You should distribute your code with a GPL license. That prevents others from distributing it or derivatives of it under anything other than GPL license. They wouldn't be able to charge for it, because anyone else who got it could redistribute it for free.

4. (250) What will the following code print out?

```
def interp(L, i):
    templ = "%s is at idx %d in a list with %d items"
    vals = (L[i], i, len(L))
    return templ % vals

print interp(['you', 'are', 'a', 'genius'], 3)
```

genius is at idx 3 in a list with 4 items

5. (250) What will the following code print out?

```
s = "<published>2009-01-23T20:04:53Z</published>"
print s.split(':')[1]
```

04

6. (250) What will the following code print out?

```
x = -1
y = -2
z = -3
def h(x, y = 2, z = 3):
   print x, y, z

h(1, z=4)
```

1 2 4

7. (300) What will the following code print out?

```
L1 = ['a', 'b', 'c']
L2 = [1, 2, 3]

zipped = zip(L1, L2)
print len(zipped)
print zipped[1]
print zipped[1][0]
```

3
('b', 2)
b

8. (500) Define a function join_strings that takes two inputs, a list of strings and a separator string. It returns a single string that contains all of the original strings concatenated together, separated by the separator string.

```
def join_strings(L, sep):
    res = ""
    for s in L:
        if res != "":
            res = res + sep
        res = res + s
    return res
```

```
joined = join_strings(['a', 'b', 'c'], "|")
test.testEqual(joined, "a|b|c")
```

9. (250) Write code to sort L in reverse alphabetic order, so that "Delightful 80" is first and "All 99" is last

```
L = ["Clear 40", "All 99", "Beautiful 20", "Delightful 80"]
```

sort(L, reverse = True)

10. (600) Define a function sorted_keys that takes a dictionary as input and returns a list of its keys, sorted based on the values associated with the keys, with the biggest one first and smallest last.

```
def sorted_keys(d):

    return sorted(d.keys(), key = lambda x: d[x],
reverse = True)

d1 = {'a': 10,
     'b': 30,
     'c': 20}
test.testEqual(sorted_keys(d1), ['b', 'c', 'a'])
```

11. (600) Define a function *longest_strings* that takes as input a list of strings and returns as output the three longest strings in the list, in order from longest to shortest.

```
def longest_strings(L):

    return sorted(L, key = len, reverse = True)[:3]

some_strings = ['a', 'abcd', 'ab', 'abc', 'abcde']
test.testEqual(longest_strings(some_strings),
['abcde', 'abcd', 'abc'])
```

12. (250) What will the following code print out?

```
x = 6
print (lambda x: x-2)(5)
```

a. 6
b. 5
c. 4
d. 3
e. (<function <lambda> at 0x00000000027CBD68>, 5)
f. There will be an error

13. (500) Something is not working correctly with the word_counts function defined below. It takes a string as input. It is supposed to produce a dictionary containing the words in the string as keys, and their counts as values. The output from running the code snippet is shown. Rewrite the definition of word_counts so that it will pass the test, changing as little of the code as you can.

```
def word_counts(s):
    d = {}
    words = s.split()
    for w in words:
        try:
            d[w] = d[w] + 1
        except:
            d[w] = 0
            d[w] = 1
 return d

results = word_counts("a man a plan a canal, Panama")
correct_results = {'a':3, 'man':1, 'plan':1,
"canal,":1, "Panama":1}
test.testEqual(results, correct_results)
```

```
-- Failed test 21:
        expected:       {'a': 3, 'Panama': 1, 'plan':
1, 'canal,': 1, 'man': 1}
        got:            {'a': 2, 'Panama': 0, 'plan':
0, 'canal,': 0, 'man': 0}
```

The remaining questions are based on ps11. Below is a simplified data structure representing a post, as it might be returned from Facebook. Feel free to carefully tear this sheet out of the exam so you can refer to it more easily.

```
sample_post =
{
    "from": {
        "id": "2220453",
        "name": "Paul Resnick"
    },
    "likes": {
        "data": [
            {
                "id": "6912",
                "name": "LB"
            },
            {
                "id": "730",
                "name": "JL"
            },
            {
                "id": "123948",
                "name": "???"
            },
            {
                "id": "848",
                "name": "KR"
            }
        ]
    },
    "comments": {
        "data": [
            {
                "from": {
                    "id": "8237",
                    "name": "MB"
                }
            },
            {
                "from": {
                    "id": "58763",
                    "name": "JO"
                }
            }
        ]
    }
}
```

14. (600) Write a function *likers* that takes as input a post of the kind in sample_post, and returns a list of the ids of all the people who liked the post. First, write the function **without** using a list comprehension, map, or filter.

```
def likers(post):
    res = []
    for d in post['likes']['data']:
        res.append(d['id'])
    return res
```

```
test.testEqual(likers(sample_post), ['6912', '730',
'123948', '848'])
```

15. (600) Now write the same function *likers* **with** a list comprehension, map, and/or filter.

Two alternatives, with map or list comprehension

```
def likers(post):
    return map(lambda d: d['id'], post['likes']['data'])

def likers(post):
    return [d['id'] for d in post['likes']['data']]
```

```
test.testEqual(likers(sample_post), ['6912', '730',
'123948', '848'])
```

In the ps11 solution set, you had the following class definition. Feel free to carefully tear out this sheet from the exam so that you can refer to it more easily.

```
class Post():
    """object representing status update"""
    def __init__(self, post_dict):
        if 'message' in post_dict:
            self.message = post_dict['message']
        else:
            self.message = ""
        if 'comments' in post_dict:
            self.comments = post_dict['comments']['data']
        else:
            self.comments = []
        if 'likes' in post_dict:
            self.likes = post_dict['likes']['data']
        else:
            self.likes = []

    def positive(self):
        return len([w for w in self.message.split() if w in
pos_ws])

    def negative(self):
        return len([w for w in self.message.split() if w in
neg_ws])

    def emo_score(self):
        return self.positive() - self.negative()
```

16. (300) Write code to create two instances of the Post class that have no likes or comments. The first instance should have as its message text the phrase "I love python." The second instance should have as its message text the phrase "Gadzooks, I am a programmer now!"

```
p1 = Post({'message': "I love python."})
p2 = Post({'message': "Gadzooks, I am a programmer
now!"})
```

Alternatively, you could set the message attribute after creating the instances.

```
p1 = Post({})
p2 = Post({})
p1.message = "I love python."
P2.message = "Gadzooks, I am a programmer now!"
```

17. (600) Define a method *entropy* for the Post class. It will provide a very poor estimate of the actual entropy. It will compute the percentage of letters in the Post's message text that are unusual letters (q, x, z, and j). For example, if the message has 100 letters of which 5 are q, x, j, or z, the entropy method would return .05.

```
    def entropy(self):
        ## the fraction of letters in the message that are
        ## unusual letters (q, x, j, and z). So, if the message
has 100 letters
        ## of which 5 are q, x, j, or z, the entropy would be
.05.
        cnt = 0
        for let in self.message:
            if let in ['q', 'x', 'j', 'z']:
                cnt = cnt + 1
        return cnt / float(len(self.message))
```

18. (300) Write a test (using test.testEqual) that checks whether the *entropy* method is defined correctly.

```
p = Post({'message': 'qxajj'})
test.testEqual(p.entropy(), 0.8)
```