SI 106 – Winter 2015 **Name: ___________________________**

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 git and github? (Circle all that apply).

    a. You want to be able to allow some people but not others to make changes to the version of the source code that gets distributed.
    b. You want to be able to allow some people but not others to make private modifications of the source code.
    c. You want to prevent other people from taking your code and incorporating it into a commercial product that they charge money for.
    d. You want to hold yourself accountable as an open source project team leader by making it possible for anyone, at any time, to fork the code and start a competing project that excludes you from leadership.
    e. You would like to be able to see or revert to any past version of any of the files in your project.

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. Criticizing the quality of code in the released version of the project and proposing a "patch" to replace some of it with code that you have written.
    b. Offering to take "ownership" of a project that has had no new code released for more than a year and which needs new code in order to make it keep working with other software that has changed during that year.
    c. After a public disagreement with the project "owner" on an email list about a technical issue (which of several similar python modules to use), making a fork of the project and inviting other project members to join your version of the project and abandon the original.

3. (300) Your friends like your 106 final project so much that they encourage you to start a company to commercialize it. The code for your project includes a file that you found on the Internet. (Fortunately, you credited the original source, and were not accused of plagiarism when we graded the project). The file has a GPL license. Briefly justify your answers to the following questions.

    a. Does that prevent you from selling a license to your project code (including that GPL'ed file) with a requirement that your customers not share the code you wrote with other (non-paying) customers?

    b. Does that prevent you from offering a service where you run your code for them and share the program outputs with them, but never let them see the project code at all (including the GPL'ed file)?

4. (250) Fill in the missing code after the vals = on the first line, so that it generates the output shown in bold.

```
vals =
templ = "%s, %s are the %d items in the list"
print templ % (vals[0], vals[1], len(vals))
```

```
v1, v2 are the 2 items in the list
```

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

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

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(z=5)
```

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

```
def enum(L):
    return zip(range(len(L)), L)

print enum(['a', 'b', 'c'])
```

8. (500) Define a function flatten that takes a list of sub-lists and returns a list that contains all of the elements of the sub-lists.

```
def flatten(L):
```

```
flattened = flatten([['a', 'b', 'c'], ['d'], ['e', 'f']])
test.testEqual(flattened, ['a', 'b', 'c', 'd', 'e', 'f'])
```

9. (250) Write code to sort L in order based on the last two characters in each string, so that 253 is first and 356 is last.

```
L = ['154', '253', '356', '455']
```

10. (600) Define a function sorted_by_keys that takes a dictionary as input and returns of a list of its values, sorted in alphabetic order based on the keys that have those values.

```
def sorted_by_keys(d):
```

```
d2 = {'alpha': 10,
      'bravo': 30,
      'delta': 20,
      'charlie': 10}
```

```
test.testEqual(sorted_by_keys(d2), [10, 30, 10, 20])
```

11. (600) Define a function *sleepiest* that takes as input a list of strings and returns as output the string that has the most 'z's in it.

```
def sleepiest(L):
```

```
some_strings = ['zany', 'pizza', 'zzzzz', 'longest']
test.testEqual(sleepiest(some_strings), 'zzzzz')
```

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:
        if w in d:
            d[w] = d[w] + 1
        else:
            d[w] = 1



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['a'], correct_results['a'])
```

```
Traceback (most recent call last):
  File "bank.py", line 12, in <module>
    test.testEqual(results['a'], correct_results['a'])
TypeError: 'NoneType' object has no attribute
'__getitem__'
```

14. (600) Define a function f that takes a list of strings and returns a list containing the first letter of every word that contains the letter z. Make it pass the test below. Your function **must use** manual accumulation (i.e., you may not use map, filter, reduce, zip, or list comprehensions.)

```
test.testEqual(f(['whiz', 'success', 'in the zone',
'yep', 'not another pizza!']), ['w','i','n'])
```

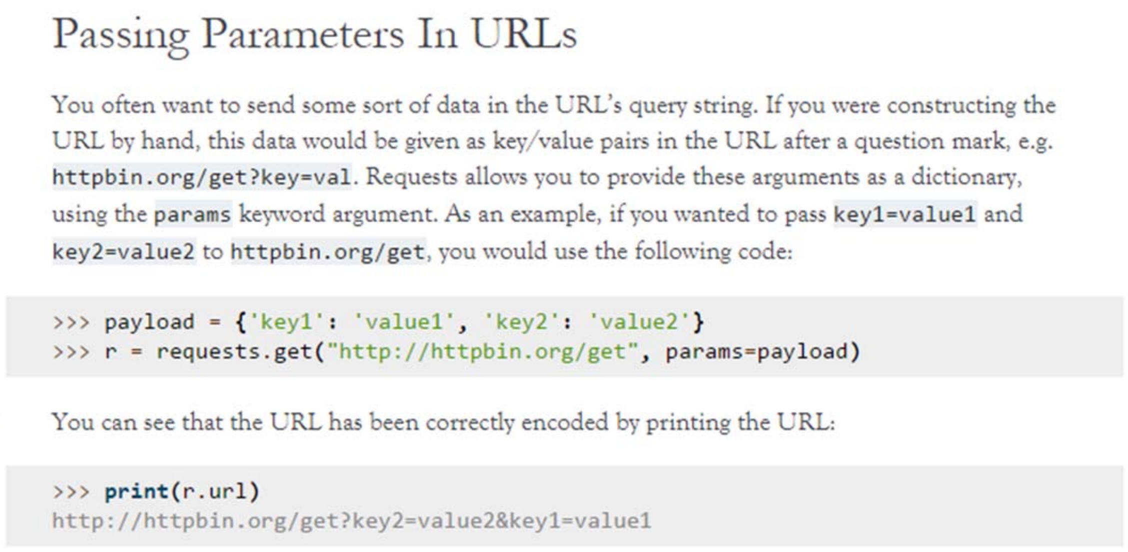
15. (600) Now define the same function **without** using manual accumulation. Instead it should use some combination of map, filter, reduce, zip, and list comprehensions.

```
test.testEqual(f(['whiz', 'success', 'in the zone',
'yep', 'not another pizza!']), ['w','i','n'])
```

16. (300) You are using a new API, offered by runestone.org. It follows the standard format of a base URL plus key-value parameters. It also provides a website that helps you construct URLs. Using that, you have assembled the following URL: http://runestone.org/API/get_score?assignment_id=24&student_id=374.

    For this question write python code, using the requests module, to retrieve that URL. You are **not permitted to use the characters = or &** in your code. In other words, you have to use the requests module's built-in feature for encoding data as part of urls.

    To help you, we have reproduced a section of the documentation for the requests module.



Passing Parameters In URLs

You often want to send some sort of data in the URL's query string. If you were constructing the URL by hand, this data would be given as key/value pairs in the URL after a question mark, e.g. `httpbin.org/get?key=val`. Requests allows you to provide these arguments as a dictionary, using the `params` keyword argument. As an example, if you wanted to pass `key1=value1` and `key2=value2` to `httpbin.org/get`, you would use the following code:

```
>>> payload = {'key1': 'value1', 'key2': 'value2'}
>>> r = requests.get("http://httpbin.org/get", params=payload)
```

You can see that the URL has been correctly encoded by printing the URL:

```
>>> print(r.url)
http://httpbin.org/get?key2=value2&key1=value1
```

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": "58763",
                "name": "JO"
            },
            {
                "id": "123948",
                "name": "???"
            },
            {
                "id": "848",
                "name": "KR"
            }
        ]
    },
    "comments": {
        "data": [
            {
                "from": {
                    "id": "8237",
                    "name": "MB"
                }
            },
            {
                "from": {
                    "id": "58763",
                    "name": "JO"
                }
            }
        ]
    }
}
```

In the ps11 solution set, you had the following class definition. We have added a couple of additional methods that you may find useful, likers and commenters. Feel free to carefully tear out this sheet from the exam so that you can refer to it more easily. We have shown a sample of what likers() and commenters produce.

```
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()

    def likers(self):
        return [d['id'] for d in self.likes]

    def commenters(self):
        return [d['from']['id'] for d in self.comments]
```

```
p1 = Post(sample_post)
print p1.likers()
print p1.commenters()
```

Output:

```
['6912', '58763', '123948', '848']
['8237', '58763']
```

17. (600) Define a method *enthusiast_count* for the Post class. It will return a count of the number of people who both liked **and** commented on a post. For example, for the sample post, it would return 1, since only id '58763' both liked and commented on it.

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

p1 = Post(sample_post)