Test Introduction to Programming

The answer to each of the following questions goes into a separate file. The name of the file must be the name of the question. The directory sol-xxx, where xxx is your student ID, contains empty files for this purpose. In these files, lines beginning with # are considered comments and will be ignored for grading.

1.txt

For each of the following expressions, give the result:

```
3 * 4 + 16 / 2
5 ** 2 - 1
10 % 7
22 // 8
True and True
not (True or False)
False == False
int(5.8)
float(16)
str(24) * 2
```

2.txt

What is printed?

```
d = {'foo': 85, 'bar': 20, 'baz': 39, 'qux': 9}
print(d['baz'])
```

3.txt

Given

```
s = 'May 15, 2013'
```

What is printed?

```
s[2]
s[4:6]
s[-4]
s[:10]
len(s)
'A' in s
s.count(',')
s.find('15')
```

4.py

Rewrite the following statement as a function taking three parameters. Choose informative names for the function and the paramters. Include an explantory docstring.

```
volume = length * width * height
```

5.txt

What is the purpose of the following piece of code:

```
numbers = [64, 65, 79, -23, 2, -96, -89, 62, -12, 60, 1]
result = 0
for j in numbers:
    if j < 0:
        result = result + 1
```

Your answer is a single of the letters a, b, c, d, or e (and only that), meaning:

a. to find the smallest number in the list
b. to count the negative numbers in the list
c. to sum the negative numbers in the list
d. to add 1 to each of the negative numbers in the list
e. to find the index of the first negative number in the list

6.py

The function in the following piece code is meant to return the full name of a person (and nothing else), separated by space.

```
def full_name (first, last):
    full = first + last
    print(full)

first = 'Donald'
last = 'Trump'
print(full_name(first, last))
```

For instance, the expected outcome of the above code should have been "Donald Trump". It is not. Correct the function.

7

Consider the class `Cleaner` defined below:

```
class Cleaner:
""" Removes rude words from text. When text is a list of strings, the clean_line(text)
    method removes occurrences of rude words and replaces them by *beep!*.
"""

    def __init__(self, forbidden_word="frack"):
        """ Set the forbidden word """
        self.word = forbidden_word

    def clean_line(self, line):
        """Clean up a single string, replacing the forbidden word by *beep!*"""
        found = line.find(self.word)
        if found != -1:
            return line[:found] + "*beep!*" + line[found + len(self.word):]
        return line

    def clean(self, text):
        for i in range(len(text)):
            text[i] = self.clean_line(text[i])
```

7-1.py

Change `clean_line` so as to behave correctly on lines with more than one occurrence of the rude word.

7-2.py

Add a method `set_word` that sets the rude word to something else. The following has to work:

```
C = Cleaner()
C.clean_line(text)
C.set_word("bloody")
C.clean_line()
```

7-3.py

Improve the functionality of `Cleaner` so as to support more rude words than one. The following has to work:

```
C = Cleaner(['shit', 'frack', 'bloody'])
C.clean_line(text)
```

8

An *ingredients list* is a text file like the following:

```
1 cup of parsley
1 kg meat
2 l milk
2 tablespoons of sugar
```

Each line consists of an integer, followed by a unit of measurement, followed by a description that may or may not begin with the word "of". The directory "8" contains ingredients lists as text files with the extension `txt`. The allowed units are the following:

```
% 8-units.txt
l
pound(s)
cup(s)
kg
g
tablespoon(s)
teaspoon(s)
pinch(es)
```

To make this exercise easier, the ingredients are written in bad English because we ignore proper plurals. We say "2 orange" and "3 octopus" (instead of "2 oranges" and "3 octopi"). We *do* care about the plurals of measurements in `8-units.txt` (pounds, cups, tablespoons, teaspoons, and pinches). No ingredient contains a comma, so we never have "1 onion, finely chopped".

8-1.py

Write a program that converts an ingredient list into two servings by doubling all amounts in the ingredient list.

Partial credit: For partial credit, you can ignore grammar of the measurements, so you may output "2 cup of salt" instead of "2 cups of salt".

8-2.py

Write a program that prompts the user for a single ingredient name, such as "sugar" or "octopus". It then computes the *total* amount of that ingredient appearing in *all* the ingredient lists. The output is a single integer followed by a unit.

The necessary conversion between units (say, how many tablespoons make up a teaspoon) are given in the file `8-conversions.txt`.

9.

Given a list of strings, a *longest word* is a word such that all the other words in the list are equal or shorter in length.

Example: a longest word in the list `['a', 'ab', 'abc']` is `'abc'`, while a longest word in `['abc', 'abc', 'abc']` is `'abc'`.

Here is a piece of code that attempts to implement a Python function for finding the longest word:

```
def longest_word(my_list):
    for word in my_list:
        is_longest = True
        for other_word in my_list:
            if len(word) > len(other_word):
                is_longest = False
                break
        if is_longest:
            return word
```

9-1

The provided code has a bug. Fix it to return an actual longest word.

9-2

What is the worst-case runtime of `longest_word`?