10.009 The Digital World

Term 3. 2020

Problem Set 6 (for Week 6)

Last update: January 9, 2020

Due dates:

- **Problems: Cohort sessions**: Following week: Tuesday 11:59pm.
- **Problems: Homework**: Same as for the cohort session problems.
- **Problems: Exercises**: These are practice problems and will not be graded. You are encouraged to solve these to enhance your programming skills. Being able to solve these problems will likely help you prepare for the midterm examination.

Objectives:

1. Learn to manipulate strings.
2. Learn to read and write files.

Note: Solve the programming problems listed using your favorite text editor. Make sure you save your programs in files with suitably chosen names, **and try as much as possible to write your code with good style (see the style guide for python code)**. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

Problems: Cohort sessions

1. *Strings:* Write a function named `reverse` that takes a string argument and returns its reverse. For example, `reverse('I am testing')` should return the string `'gnitset ma I'`. Use iteration for this exercise and answers using slicing will not be accepted.

2. *Strings: Check password:* Some web sites impose certain rules for passwords. Write a function named `check_password` that checks whether a string is a valid password. Return True if it is a valid password and False otherwise. The password rules are as follows:

   - A password must have at least 8 characters.
   - A password must consist of only letters and digits.
   - A password must contain at least two digits.

   To test:

```
print("check_password('test')")
ans=check_password('test')
print(ans)

print("check_password('testtest')")
ans=check_password('testtest')
print(ans)

print("check_password('testt22')")
ans=check_password('testt22')
print(ans)

print("check_password('testte22')")
ans=check_password('testte22')
print(ans)
```

   The output should be:

```
check_password('test')
False
check_password('testtest')
False
check_password('testt22')
False
check_password('testte22')
True
```

3. *Strings: Longest Common Prefix* Write a function named `longest_common_prefix` that takes two strings and returns the longest common prefix of the two strings. For example, the longest common prefix of `distance` and `disinfection` is `dis`. If the two strings have no common longest common prefix, the method returns an empty string.

   To test:

```
print("longest_common_prefix('distance','disinfection')")
ans=longest_common_prefix('distance','disinfection')
print(ans)

print("longest_common_prefix('testing','technical')")
ans=longest_common_prefix('testing','technical')
print(ans)

print("longest_common_prefix('drinking','drinker')")
ans=longest_common_prefix('drinking','drinker')
print(ans)

print("longest_common_prefix('rosses','crosses')")
ans=longest_common_prefix('rosses','crosses')
print(ans)

print("longest_common_prefix('distancetion','distance')")
ans=longest_common_prefix('distancetion','distance')
print(ans)
```

The output should be:

```
longest_common_prefix('distance','disinfection')
dis
longest_common_prefix('testing','technical')
te
longest_common_prefix('drinking','drinker')
drink
longest_common_prefix('rosses','crosses')

longest_common_prefix('distancetion','distance')
distance
```

4. *Files and I/O:* A file named `xy.dat` contains two columns of numbers, corresponding to the `x` and `y` coordinates on a curve. The start of the file is given below, with x-coordinates in the left column and y-coordinates in the right column. The values on each line are separated by the tab character (`\t`).

```
-5.0            -283.0
-4.89898989899  -269.1728292
-4.79797979798  -255.76800244
-4.69696969697  -242.77933606
```

Write a function named `get_maxmin_mag(f)` that takes a file object `f` as an argument (ie., `f=open ('xy.dat','r')`). **You must not call `f=open('xy.dat','r')` in your function**. The function should read the first column as an `x` coordinate and the second column as a `y` coordinate. The function then returns two instances of the Coordinate custom data type. The first instance contains the coordinates that gives the maximum magnitude, and the second instance contains the coordinate that gives the minimum magnitude. The magnitude is defined as:

$$|p| = \sqrt{(p.x)^2 + (p.y)^2} \tag{1}$$

where p is of the type Coordinate defined below.

```
class Coordinate:
    x=0
    y=0
```

The following is an example how a function would return an instance of the Coordinate custom data type:

```
def foo():
    a=Coordinate()
    a.x=1.0
    a.y=2.0
    return a
```

The following test script runs your function and displays its return values.

```
f=open('xy.dat','r')
pmax,pmin=get_maxmin_mag(f)
print('max: ({:f}, {:f})'.format(pmax.x,pmax.y))
print('min: ({:f}, {:f})'.format(pmin.x,pmin.y))
```

After running the test script, the output you should get is as follows:

```
max: (-5.000000, -283.000000)
min: (-1.060606, -0.184796)
```

Optional: If you are familiar with object-oriented programming, think about how you could improve the structure of your program by defining an instance method on the `Coordinate` class. In particular, which calculation(s) in your current implementation of `get_maxmin_mag` should actually be the responsibility of a `Coordinate` object instead?

Problems: Homework

1. *Strings: Binary to decimals:* Write a function named `binary_to_decimal` that parses a binary number as a string into a decimal form, and returns it as an integer. You can assume that the string is never empty and only contains characters 0 or 1.

```
>>> print(binary_to_decimal('100'))
4
>>> print(binary_to_decimal('101'))
5
>>> print(binary_to_decimal('10001'))
17
>>> print(binary_to_decimal('10101'))
21
```

2. *Strings:* Write a function named `uncompressed` that takes a compressed string as an input and returns an uncompressed string, where each alphabetic character is preceded by a single digit, indicating the number of times that the character should be entered in the uncompressed version of the string. For example:

   - The uncompressed version of `2a5b1c` is `aabbbbbc`
   - The uncompressed version of `1a1b2c` is to `abcc`
   - The uncompressed version of `1a9b3b1c` is `abbbbbbbbbbbbc`

3. *Strings: DNA:* Write a function named `get_base_counts2` by modifying `get_base_counts` that you wrote in homework problem set 4. `get_base_counts2` takes a string as an input. The input string may contain letters other than A, C, G, and T. The function should return the counts of only A, C, G, and T in the form of a dictionary (even if the input string does not have any of those letters!) The input string is only invalid if it contains non-letters and lower case letters, in which case the function should return `'The input DNA string is invalid'`. If there are any uppercase letters other than A, C, G, and T, the string is not invalid, but the counts of those letters should not be added into the dictionary (note that this is different from the invalidity condition for `get_base_counts`).

4. *Files and I/O:* A file named `constants.txt` contains a table of the values and the dimensions of some fundamental constants from physics. The file's format is as follows:

name of constant	value	dimension
speedoflight	299792458.0	m/s
gravitationalconstant	6.67259e-11	m**3/kg/s**2
Planckconstant	6.6260755e-34	J*s
elementarycharge	1.60217733e-19	C
Avogadronumber	6.0221367e23	1/mol
Boltzmannconstant	1.380658e-23	J/K
electronmass	9.1093897e-31	kg
protonmass	1.6726231e-27	kg

Load this table into a dictionary named `constants`, where the keys are the names of the constants. For example, `constants['gravitational constant']` holds the value of the gravitational constant ($6.67259 * 10^{-11}$). Write a function named `get_fundamental_constants` that takes a file object as an argument, reads and interprets the text in the file, and returns the dictionary.

Test Cases:

For the following script:

```
f = open('constants.txt','r')
print(get_fundamental_constants(f))
```

The expected output is (Note that the order may be different from what your code produces):

```
{'speedoflight': 299792458.0, 'gravitationalconstant': 6.67259e
    -11, 'Planckconstant': 6.6260755e-34, 'elementarycharge':
    1.60217733e-19, 'Avogadronumber': 6.0221367e+23, '
    Boltzmannconstant': 1.380658e-23, 'electronmass': 9.1093897e
    -31, 'protonmass': 1.6726231e-27}
```

5. *Files: Process scores:* Suppose that a text file contains an unspecified number of scores. Write a function named `process_scores` that takes in a file object `f` as an argument, reads the scores from `f` and returns their total and average. Scores are separated by whitespace on a single line. Use `scores.txt` for testing.

   To test:

   ```
   f=open('scores.txt','r')
   ans=process_scores(f)
   print('Output: ',ans)
   ```

   The output should be:

   ```
   Output:  (385.19999999999993, 35.01818181818181)
   ```

Problems: Exercises

1. An anagram is a type of word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once; e.g., orchestra = carthorse. Using the word list from `http://www.puzzlers.org/pub/wordlists/`

`unixdict.txt`, write a function `find_anagram` that takes a file object as an input, and returns a list of anagrams. Each anagram set is a list of words that share the same characters. Return the list of anagram with the maximum number of letters in `unixdict.txt`.

Test Cases:

Input: file= unixdict.txt
Output: [[‘abel’, ‘able’, ‘bale’, ‘bela’, ‘elba’], [‘alger’, ‘glare’, ‘lager’, ‘large’, ‘regal’], [‘angel’, ‘angle’, ‘galen’, ‘glean’, ‘lange’], [‘evil’, ‘levi’, ‘live’, ‘veil’, ‘vile’], [‘caret’, ‘carte’, ‘cater’, ‘crate’, ‘trace’], [‘elan’, ‘lane’, ‘lean’, ‘lena’, ‘neal’]]

2. A sentence splitter is a program capable of splitting a text into sentences. The standard set of heuristics for sentence splitting includes (but is not limited to) the following rules: Sentence boundaries occur at one of “.” (periods), “?” or “!”, exceptions to these are:

   (a) Periods followed by whitespace followed by a lower case letter are not sentence boundaries.

   (b) Periods followed by a digit with no intervening whitespace are not sentence boundaries.

   (c) Periods followed by whitespace and then an upper case letter, but preceded by any titles are not sentence boundaries. Sample titles include Mr., Mrs., Dr., and so on.

   (d) Periods internal to a sequence of letters with no adjacent whitespace are not sentence boundaries (for example, www.aptex.com, or e.g).

   (e) Periods followed by certain kinds of punctuation (notably comma and more periods) are not sentence boundaries.

   Your task here is to write a function `split_sentences(f)` that takes in a file object as an argument and writes out its content in a list with each sentence as a string item in a list. Test your function with the following short text: *Mr. Smith bought cheapsite.com for 1.5 million dollars, i.e. he paid a lot for it. Did he mind? Adam Jones Jr. thinks he didn‘t. In any case, this isn’t true... Well, with a probability of .9 it isn’t.* The result should be:

```
[‘Mr.  Smith bought cheapsite.com for 1.5 million dollars, i.e.  he paid a lot
for it.’,
‘Did he mind?’,
‘Adam Jones Jr.  thinks he didn’t.’,
‘In any case, this isn’t true...’,
‘Well, with a probability of .9 it isn’t.’]
```

   Test Cases:

Test case 1

Input: filename= fileEx2_1.txt

Output: outp = [‘Mr. Smith bought cheapsite.com for 1.5 million dollars, i.e. he paid a lot for it.’, ‘Did he mind?’, ‘Adam Jones Jr. thinks he didn’t’, ‘In any case, this isn’t true...’, ‘Well, with a probability of .9 it isn’t.’]

Test case 2

Input: filename= fileEx2_2.txt

Output: [‘Lorem ipsum dolor sit amet, consectetur adipiscing elit.’, ’Donec sed nisl eu nulla scelerisque fermentum.’, ‘Nulla facilisi.’, ‘Suspendisse hendrerit quam in dui sollicitudin egestas.’, ‘Nullam euismod massa at nisi luctus dictum tristique nibh hendrerit.‘, ’Fusce eleifend sollicitudin sapien, nec imperdiet erat vestibulum et.‘, ’Phasellus vitae leo neque, ut bibendum orci.’, ‘Nullam a velit sit amet erat auctor adipiscing.’, ‘Maecenas sollicitudin dui sagittis nisi dapibus a rhoncus nisi dictum.’, ‘Suspendisse potenti., Phasellus nec risus eget nisl sodales lobortis.’, ‘Lorem ipsum dolor sit amet, consectetur adipiscing elit.’, ‘Aliquam sodales, justo vitae blandit viverra, est est lacinia tellus, eget euismod sem dui non felis.’, ‘In est lorem, eleifend eget blandit ac, lobortis et neque.’, ‘Etiam porttitor, justo quis cursus semper, elit odio semper augue, a suscipit lacus mauris et ipsum.’, ‘Nulla nec quam ante.’, ‘Phasellus diam quam, porta id eleifend sit amet, vestibulum eu ligula.’]

Test case 3

Input: filename= fileEx2_3.txt

Output: outp = [‘[32] But I must explain to you how all this mistaken idea of denouncing of a pleasure and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master- builder of human happiness.’, ‘No one rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because those who do not know how to pursue pleasure rationally encounter consequences that are extremely painful.’, ‘Nor again is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain, but occasionally circumstances occur in which toil and pain can procure him some great pleasure.’, ‘To take a trivial example, which of us ever undertakes laborious physical exercise, except to obtain some advantage from it?’, ‘But who has any right to find fault with a man who chooses to enjoy a pleasure that has no annoying consequences, or one who avoids a pain that produces no resultant pleasure?’]

Test case 4

Input: filename= fileEx2_4.txt

Output: outp = [‘[33] On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain.’, ‘These cases are perfectly simple and easy to distinguish.’, ‘In a free hour, when our power of choice is untrammeled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided.’, ‘But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted.’, ‘The wise man therefore always holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or else he endures pains to avoid worse pains.’]

Test case 5

Input: filename= fileEx2_5.txt

Output: outp = [‘In publishing and graphic design, lorem ipsum[1] is placeholder text (filler text) commonly used to demonstrate the graphic elements of a document or visual presentation, such as font, typography, and layout, by removing the distraction of meaningful content.’, ‘The lorem ipsum text is typically a section of a Latin text by Cicero with words altered, added and removed that make it nonsensical in meaning and not proper Latin.’]

End of Problem Set 6.