CSE/ISE 337 – Scripting Languages
Spring, 2023
Assignment 01 – Python Scripting

Assigned: Tuesday, 01/30/2023
Due: **Tuesday, 02/20/2023, at 11:59 PM**
Total Problems: 5, Total Points: 70

Submission Instructions:

1. Submit all of your solutions in a single file named: `cseise337_a01.py`
2. Make sure you include your name, netid, and Student ID number as comments at the top of your submission file.
3. Please label each separate solution with the problem number in a comment.
4. Make sure you use the specified, file, function and class names (and any other names specified in the instructions). Grading may be partially automated (by importing your code to run against our test cases) and the use of incorrect names will interfere.
5. This is an individual programming assignment. Any collaboration on coding will be considered a violation of academic honesty.

Problem 1 – Chaotic Strings (10 points)
A string is called *chaotic* if all characters of the string appear a different number of times. Given a string **`s`**, determine if it is chaotic. If so, return the string: `'TOHRU'`; otherwise, return the string: `'ELMA'`.

As an example, consider the string `s = 'abbccc'`. The string `s` is chaotic since every character occurs a number of times distinct from every other character, `{'a' : 1, 'b' : 2, 'c' : 3}`. However, the string `'abcc'` is not chaotic because the characters `'a'` and `'b'` both appear the same number of times as each other, namely once.

Function Description: Write a function `is_chaotic()` that takes a parameter `s` and returns the string `'TOHRU'` if `s` is valid; the string `'ELMA'` otherwise.

Additional Constraints The string provided as parameter to the function `is_chaotic()` will only contain characters [a - z].

Sample Test Cases:

```
1. is_chaotic('aabbcd') = 'ELMA'
2. is_chaotic('aaaabbbccd') = 'TOHRU'
3. is_chaotic('abaacccdee') = 'ELMA'
```

Problem 2 – Balanced Brackets (10 points)

A bracket is considered to be any one of: `(, ), {, }, [, or ]`

Two brackets are considered matched if an opening bracket (i.e., `(, {, [`) is followed by a closing bracket (i.e., `), }, ]`) of the exact same type. There are 3 types of brackets – parenthesis, that is, `()`, braces, that is, `{}`, and square brackets, that is `[]`.

A matching pair of brackets is not balanced if the set of brackets it encloses is not balanced. For example, `{ [ ( ] ) }` is not balanced because the set of brackets between `{}` is not balanced. The pair of square brackets encloses a single unbalanced open parenthesis, `(`, and the pair of parenthesis encloses a single unbalanced closing square bracket, `]`.

Hence, a sequence of brackets is balanced if the following conditions are met:

1. It contains no unmatched brackets
2. The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Function Description: Write a function `is_balanced()` that takes a string, where each character in the string is a bracket, and returns the Boolean value True if the brackets are balanced; otherwise, returns the Boolean value False

Sample Test Case

```
1. is_balanced('{[()]}') = True
2. is_balanced('{[(])}') = False
3. is_balanced('{{[[(())]]}}') = True
```

Problem 3 – Functional Programming (10 points)

Function Description: Write a function, `winning_function()` that takes three parameters as input: a list of integers and two functions that each take an integer as input and return a Boolean value. `winning_function()` will then count how many times each function returns True when applied to the elements of the input list. The function that returns True the most is the winning function. Return the name of the winning function as a string. If both functions return True the same number of times, then return the string, `'TIE'`. For this problem, we will pass the `even()` and `odd()` functions to `winning_function()`:

```
def even(x):
    return x % 2 == 0

def odd(x):
    return x % 2 == 1
```

For example, if the input list is `a = [2,3,4,5,6,8]`, then calling

`winning_function(a, even, odd)` would return `'even'` because the `even()` function will return True 4 times when applied to the elements of the list `a`, while the `odd()` function will only return True 3 times.

Problem 4 – Representing Filesystems (20 points)

The Linux shell command: `ls -lR > lsoutput.txt`
prints the recursively explored contents of the filesystem starting at the current directory (using a long listing format) to the standard output. The output is redirected to a file called lsoutput.txt in the current directory.

System management requires operating on filesystem data from within our programs and scripts. Properly representing the entities in the file system within our programs is a good application of object orientation.

Please create three classes: (a) `FS_Item`, (b) `Folder`, (c) `File`

a. `FS_Item` class: This class will be the parent class for the other two. Every `FS_Item` has a single instance variable called `name`. The value of `name` is a string. The value of this instance variables should be set by the `__init__()` method of the `FS_Item` class.

b. `Folder` class: This class will be a subclass of the `FS_Item` class. Folders represent directories. In addition to the name attribute that is inherited from `FS_Item`, every instance of the `Folder` class contains an additional instance variable called `items`, which should be initialized as an empty list.

Define a method within the `Folder` class called `add_item()`, which takes an instance of `FS_Item` (either a `Folder` or a `File`) as argument passed to a parameter called `item`. The argument is appended to the current `Folder` objects `self.items` list. This method does not return anything.

c. `File` class: This class will be a subset of the `FS_Item` class. `Files` represent documents stored in the file system. In addition to the name attribute that is inherited from `FS_Item`, every instance of the `File` class contains an additional instance variable called `size`. The value of this instance variable should be set by the `__init__()` method for the `File` class and represent the size of the file in bytes.

d. Function Description: Write a function called `load_fs()`, which has a single parameter called `ls_output`. The argument passed to `ls_output` is the name of a file which contains the output of the system command `ls -lR`.

The function should read this file and use it to construct an internal representation of the part of the file system recorded in the file named by `ls_output`. For each directory, create a `Folder` object with the same name. Add each directory and document contained in that directory as a

`Folder` or `File` element of its items list. For each `File` element make sure to set its name and filesize when adding it to the `items` list of the `Folder` that contains it.

When done the function should return a reference to the top-level `Folder` item (the one corresponding to the top-level directory in `ls_output`.

I have posted samples of lsoutput.txt that you can use to test your solutions.

Problem 5 – Decoding (20 points)

Function Description: Write a function, `decode()`, which takes a string of cyphertext (which is some encrypted English text) to a formal parameter named `ct` and returns a string of plaintext (which is the original English text that we can understand).

The following encryption scheme is used:

- The n^{th} plaintext alphabetic letter, for `n > 1`, is encrypted to the letter whose lexical position is the sum modulo 26 of the ordinal value of the n^{th} letter with the sum of the ordinal values of all the letters that precede it in the plaintext.
- The first plaintext alphabetic letter of the message is encrypted to the letter whose lexical position in the alphabet is the sum modulo 26 of the ordinal value of the first letter plus the integer 59.
- Calculate the lexical position of the ciphertext letter as the letter at the position between 0 and 25, with `'a' at 0, 'b' at 1, …, 'z' at 25`.
- Calculate the ordinal value of a letter by passing the character to the `ord()` function. The integer returned is the ordinal value of the letter.
- To simplify matters the only alphabetic characters contained in the plaintext message will be lowercase. No uppercase characters will be used.
- Leave non-alphabetic characters, including whitespace, punctuation, numbers, etc. unchanged.

Here is a small example: `"secret"`

1. The ordinal value of 's' is 115. We compute 115 + 59 = 174 (mod 26) = 18. The letter whose lexical position is 18 is 's'. That's the first letter of the ciphertext.
2. The ordinal value of 'e' is 101. We compute 101 + 115 (the ordinal value of 's') = 216 (mod 26) = 8. The letter whose lexical position is 8 is 'i'. That's the second letter of the ciphertext.
3. The ordinal value of 'c' is 99. We compute 99 + (115 + 101) [the sum of the ordinal values of the preceding two plaintext letters] = 315 (mod 26) = 3. The letter whose lexical position is 3 is 'd'. That's the third letter of the ciphertext.
4. `ord('r')` = 114. 114 + 115 + 101 + 99 = 429 (mod 26) = 13 => 'n'
5. `ord('e')` = 101. 101 + 114 + 115 + 101 + 99 = 530 (mod 26) = 10 => 'k'
6. `ord('t')` = 116. 116 + 101 + 114 + 115 + 101 + 99 = 646 (mod 26) = 22 => 'w'

   So, the corresponding ciphertext is `"sidnkw"`

Your function will reverse this process in order to decode messages that have been "encrypted" in this way.

I have posted some examples with both the plaintext and the cyphertext that you can test your function on. You can hardcopy the cyphertext into your assignment file. You can use docstrings (""" """) for multiline cyphertexts.