CSE/ISE 337 – Scripting Languages
Spring, 2022
Assignment 01 – Python Scripting

Assigned: Tuesday, 02/01/2022
Due: **Tuesday, 02/22/2022, at 11:59 PM**
Total Problems: 6, 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 – Nice Strings (10 points)
A string is considered to be *nice* if all characters of the string appear the same number of times. Given a string s, determine if it is nice. If valid, return the string: HARD YES; otherwise, return the string: HARD NO.

As an example, consider the string s = abc. The string s is nice since every character occurs exactly once, {'a' : 1, 'b': 1, 'c': 1}. However, the string abcc is not nice because the characters do not occur an equal number of times in the string.

Function Description: Write a function `is_nice()` that takes a parameter s and returns the string HARD YES if s is valid; the string HARD NO otherwise.

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

Sample Test Cases

```
1. is_nice('aabbcd') = 'HARD NO'
2. is_nice('abcdefghhgfedcba') = 'HARD YES'
3. is_nice('abcdefghhgfedecba') = 'HARD NO'
```

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 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 are not matched. 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, `apply_fun()` that takes two parameters as input: a list of integers and a function. For this problem, we will pass the `even()` function to `apply_fun()`:

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

Using list comprehension, the function returns a list that contains the positions of the integers in the input list for which `even()` returns true.

For example, if the input list is `a = [2,3,4,5,6,8]`, then calling `apply_fun(a, even)` would return `[0,2,4,5]`.

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 all $n > 1$, is encrypted to the letter whose ordinal value is the ordinal value of the n^{th} letter plus the ordinal value of the n-1^{th} alphabetic letter in the message modulo 26.
- The first plaintext alphabetic letter of the message is encrypted to the letter whose position in the alphabet is the sum modulo 26 of the ordinal value of the first letter + the integer 17.
- For example, suppose the message is the string "I am not here right now".
    - The second plaintext alphabetic letter is "a". It will be encrypted as the alphabetic letter whose position in the alphabet is the sum modulo 26 of the ordinal value `ord("a")`, which is 97 and the ordinal value of the preceding plaintext alphabetic letter, `ord("I")`, which is 73. That sum is: [(((97 + 73) % 26)) = 14], which is the letter "o". So, the "a" would be replaced by "o" in the cyphertext (assuming 'a' is the 0^{th} letter of the alphabet).
    - The first plaintext alphabetic letter, "I", is encrypted as the alphabetic letter whose position in the alphabet is the sum modulo 26 of the ordinal value `ord("I")`, which is 73 and the integer 17. That sum is: [(((73 + 17) % 26)) = 12. Then, "I" will be encrypted as the 12th letter of the alphabet (again, starting with 'a' at 0), which is "M".
- Capitalization is preserved by encryption and decryption. Any letter capitalized in the plaintext should be capitalized in the ciphertext, and vice versa.

All non-alphabetic characters, including whitespace, are unchanged.

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

Hints: You should first derive the decryption scheme that corresponds to the encryption scheme. In your program, consider using the `isalpha()`, `isupper()`, `islower()`, `chr()`, and `ord()` functions.

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.