SENG265, FALL 2020
ASSIGNMENT 4
UNIVERSITY OF VICTORIA

Due: Dec 7, 2020 by 10:00 am, by "git push".
(Late submissions **not** accepted)

1 Assignment Overview

This assignment involves writing an interactive program to decode a data set of encrypted ferry schedules from BC Ferries. The program will interactively prompt the user for input which will be used to decrypt and process the appropriate files.

The overall goal of this assignment is to familiarize yourself with `regex` and Python classes. The Section 3 `Implementation` describes the program and class you are to implement. Section 4 describes the `Constraints` you should consider in your implementation, and Section 5 describes the `Testing` component. `What you should Submit` is outlined in Section 6 and the `Evaluation` scheme is given in Section 7.

Your code is expected to run without warnings **in the virtual machine Senjhalla** using Python 3.6.9.

Read the assignment carefully to ensure you understand all the requirements of the assignment. The assignment might seem large and overwhelming at first (if you are new to Python); take a deep breath and read again, detailed explanations exist in this specification. The assignment is also very modular and individual components can be attempted separately

1.1 Terminology

- (Shift) Cipher: Encryption algorithm
- Encrypted or encoded: Text that has been transformed using the algorithm described in section 2;
- Decrypted or decoded: Original text recovered from the encrypted version, using a password or key;
- Password or key: The cipher key used to decrypt or encrypt a text;

2 Encrypted Data

There are 3 data files in ascii-text format included with the assignment using the naming convention `ferryX.out`. These files contain information from BC ferries regarding their sailings. The original CSV files have been encrypted using a basic shift cipher implementation (described below). You may assume that every original CSV is properly formatted and every row has the same number of fields. The first row will be the CSV file headers. These files can be decoded into CSV format with the following headers:

- departure_terminal
- arrival_terminal
- vessel_name
- scheduled_departure_year
- scheduled_departure_month
- scheduled_departure_day
- scheduled_departure_hour
- scheduled_departure_minute
- actual_departure_year
- actual_departure_month
- actual_departure_day
- actual_departure_hour
- actual_departure_minute
- arrival_year
- arrival_month
- arrival_day
- arrival_hour
- arrival_minute

2.0.1 Encryption Algorithm

You are provided three files that were encoded using a *shift cipher*.

As an encryption algorithm with a particular *key* and *alphabet*, the *shift cipher* works as follows to encode the given *input text*: For each character of the input text and the corresponding character of the password (key), add their value to obtain the *shift key*. The value is determined by the location or "rank" (starting at 0) in the encryption alphabet. If the value is too large is not long enough to map directly the alphabet, than take the modulo of the length of your alphabet (i.e. "wrap around"). The encoded text is the character from the encryption alphabet at index *value*.

Consider the following simplified example of the algorithm:

For the input text: `HELLO WORLD`
For the key: `ZIP`
For the encryption alphabet: `[A-Z]`, " " (27 characters)

First character: H (value=7 in the alphabet)
First key: Z (value=25 in the alphabet)
Shift key: $7 + 25 = 32$ modulo $27 = 5$
Encoded character = F ($alphabet[5]$)
Fifth character: O (value=14 in the alphabet)
Fifth key ($5 mod 3 = 2$): I (value=8 in the alphabet)
Shift key: $14 + 8 = 22$
Encoded character = W ($alphabet[22]$)

This process is repeated for each character in the input text until the entire input is encrypted. For this example, the full encrypted text is `FM JWOUWFJL`.

The files were encrypted using the following encryption alphabet (in this order: lowercase and uppercase letters, digits, punctuation, space and newline. **The alphabet is included for you in the `decoder.py` file provided to you.** Please note that the punctuation *is not preserved* (i.e. it is also encrypted).

The passwords (keys) are provided for you with the format `inX.txt`. The first line is the encrypted file (i.e. `ferry1.out`) and the second line is the encryption password or key used to encrypt the file. Below is an example of a `inX.txt` file:

```
cases/ferry1.out
A!00b12$
```

In order to decrypt the files, you need to perform the inverse (reverse) of the encoding process.

HINT: You can use the `inX.txt` files (and any you create yourself) to automate your testing using redirection.

3 Implementation

For this assignment you are expected to write an interactive Python program, contained in a source file called `decoder.py`, which calculates ferry delays for a CSV file containing ferry schedule data from BC ferries. The program must run, with no errors, using the following commands:

```
$ python3 decode.py
```

3.1 Program Flow

Your program will follow this general process:

1. Ask user for a encrypted filename;
2. Check if file exists, if not, prompt user again;
3. The password (key) to decode the file;
4. Check if the password is in a valid format, if not, prompt the user again;
5. Check if the provided valid password successfully decrypts the provided file, if not, prompt the user for the password again;
6. Iterate through the decrypted rows;
7. Calculate the average monthly delay between scheduled time and when the ferry actually leaves the port;

3.2 User Input

The program is required to prompt the user for the encrypted file to decode and the password (key) to use. You must also validate the data upon input. If the data is invalid notify the user of their error and prompt them to enter their selections again. Follow the steps below:

The user should also be able to enter **q** for quit at either prompt.

3.2.1 Check Password Validity

You will be using regular expressions to determine if the password is in a valid format. A valid password must have:

- At least 1 uppercase letter;
- At least 2 numerical digits;
- Exactly 2 special characters, `!@#$&*-_`
- Password length of 6-8 characters;

Use the python module `re` to determine if the password is a valid password. You may use multiple regular expressions (HINT: it is possible to use only one regular expression), but y**ou can only use a regular expression to validate the password (key) format**. For example, you may not use `string.count()` or `subtring in alphabet` (this list is not inclusive).

Even if the provided password (key) is a valid password, it may not be the password used to encrypt the provided file. Check if the provided valid password successfully decrypts the provided file, if not, prompt the user for the password again. See the next section (3.3) for details on the `FileDecoder` class you are to implement which will decrypt the encrypted file.

3.3 FileDecoder

You will create an **iterable class** called `FileDecoder` in a file called `cipher.py`. This class will (a) read the decrypted file, (b) decrypt the file, (c) check if the password successfully decrypted the file, and (d) return each decrypted CSV row iteratively. **For evaluation purposes, we will create and use your class separately** (i.e. outside of `decoder.py`).

3.3.1 Constructor

Your class will take in three **string** variables:

- `key`: password (key) provided by user;
- `filename`: file path to the encrypted file for your class to decode;
- `alphabet`: encryption alphabet used for encoding;

The following is a simplified example of the constructor call:

```
file_decoder = FileDecoder(key="ABC", filename="file.out", alphabet="ABC")
```

You may use inheritance to extend existing Python classes.

`file_decoder` will be used in the following examples to represent an FileDecoder object.

3.3.2 Decryption

Your class will load and decrypt the provided encrypted file using the provided key (password) and alphabet. See Section 2 for details on the encryption algorithm (cipher).

3.3.3 Exceptions

Your class should raise a custom `DecryptException` if the password provided does not successfully decode a *valid CSV file* (as described in Section 2). Your `decoder.py` solution should successfully handle the exception and continue the functionality described in Section 3.1.

3.3.4 Additional Required Functionality

Python provides built-in functions that you may override to provide certain behaviours to your classes. With this in mind, program your class to perform the following functionality.

print

By default, if you use `print(object)` on a class object, it will return a class description. For example:
`<class 'cipher.FileDecoder'>`

This is not very helpful. Instead, we can use `__repr__` to provide un-ambiguous description of our class, or `__str__` to provide human-readable information about our class. The `print` method will check for these two functions.

Your class should return a unambigious human-readable message string containing the `key` and `filename` used in the constructor when `print(your_class_object)` is used.

Example output for `print(file_decoder)` usage:
`FileDescriptor(key='A00!$a', file='ferry1.out')`

len

You have used the `len()` function in the past to get the length of a *list* or a *string*. Python provides the function `__len__` to allow you provide this functionality to your class.

Your class should return the number of entries (rows or lines) in the decoded CSV file (including the header). All entries (list or tuple) should be the same size.
Expected output for `len(file_decoder)` usage:
`498`

Iterable Class

Your class will be an **iterable class**. An example of an iterable class is `TextIOWrapper` (created when using `open("file.txt", "r")`). For assignment #2, you may have iterated over a file line by line using a for-loop. Your class will have the same functionality, only it will return a decoded row from the decrypted file that you must implement.

Your program will iterate over your class object, returning a list or tuple object containing the contents of a single row from the decoded CSV file. Example `iterator` usage with the `FileDecoder` object:

```
for decoded_csv_row in file_decoder:
    print(decoded_csv_row)
    break
```

Example of expected output:

```
['departure_terminal', 'arrival_terminal', 'vessel_name',
'scheduled_departure_year', 'scheduled_departure_month',
'scheduled_departure_day', 'scheduled_departure_hour',
'scheduled_departure_minute', 'actual_departure_year',
'actual_departure_month', 'actual_departure_day',
'actual_departure_hour', 'actual_departure_minute',
'arrival_year', 'arrival_month', 'arrival_day',
'arrival_hour', 'arrival_minute']
```

3.4 Program Output

You will process the output provided by your `FileDecoder` (see Section 3.3). The decrypted file will contain information on BC Ferries on their sailing schedules for some months. You will calculate the average departure delay for each month in the file. These files will have data for some, but not all, months. Ignore months (i.e. do not output) for which there is no data.

For each month in the file, output the line `Average departure delay for MON: AVG`, where MON is the 3 letter month abbreviation and AVG is the average delay in minutes rounded to 2 decimal places for each month.

- The departure `delay` is calculated by taking the difference between scheduled departure and actual departure. When calculating delay, you can assume that the no ships will be delayed by a day (i.e. delays are only within the same day). A negative departure delay indicates the ship left early, and a positive delay indicates the ship left late.

- Use the following three letter month abbreviations when printing month MON output: `Jan, Feb, Mar, Apr, Jun, Jul, Aug, Sep, Oct, Nov, Dec`
- To make sure our automated testing works properly, the printing of results should be framed between the keywords `RESULTS` and `END` (in all caps) – see example below.
- The example formatting must be followed exactly due to the automatic testing. Submissions that do not follow the output spec, will fail.

Here is an example of user input/output with the required formatting. **There is no standard for the messages to the user**.

When the user inputs the following (`in1.txt`):

```
Enter name of file: cases/ferry1.out
Enter password: A00!$a
```

The expected output (`out1.txt`): `RESULTS`

```
FileDecoder: FileDecoder(key='A00!$a', file='ferry1.out')
Entries: 498
```

```
    Average delay for Feb: 3.71
END
```

This particular file only contains information for February.

4 Constraints

You may only use python3 modules that are installed on Senjhalla. If an python3 module is not installed, you may not use that module in your code.

You may use multiple regular expressions, but you may only use regular expressions to validate the password (key) format.

Your `FileDecoder` the class **must** to be in the `cipher.py` file (not in the decoder file). Your class **must** be an iterable class.

Your must do all file handling inside your `FileDecoder`. For example, you can't read the file outside your class and pass the data to your class. Iterating through the ferry files must be done in the class using class iterator functions (not outside in `decoder.py`).

5 Test Inputs

You should test all of your programs with a variety of test inputs, covering as many different use cases as possible, not just the test input provided. You should ensure that your programs handle error cases (such as files which do not exist) appropriately and do not produce errors/exceptions.

You are expected to evaluate and test your code using the skills you have learned so far in the course.

You are provided three encrypted input files and their corresponding passwords and expected output. **The original CSV files are not provided.** During evaluation, three additional encrypted BC ferry CSV files will be used for testing. These files will be in the same CSV format as those files provided.

This assignment is modular by design to allow for individual testing of components. We will load and run your `FileDecoder` class independent of `decoder.py`.

Provided For You

For this assignment, you will be provided an `assign3.zip` containing `decoder.py` and an empty `cipher.py` file located in folder `src`. Three encrypted files (`ferryX.out`) and corresponding expected input and output (located in the folder `cases`) `inX.txt` and `outX.txt`.

(HINT: Use `diff` and redirection to compare your results with the provided expected output)

6 What you must submit

- Python source-code name `decoder.py` which contains your solution for assignment #4.
- Python source-code name `cipher.py` which contains your `FileDecoder` class.
- Ensure your work is **committed** to your local repository in the provided **a4** folder **and pushed** to the remote **before the due date/time**. (You may keep extra files used during development within the repository.)

7 Evaluation

The teaching staff will primarily mark solutions based on the input files provided for this assignment. Students must adhere to the software requirements (command execution and output formatting) outlined in this assignment.

For this assignment, there will be no student demos.

In addition to automated testing, your code will be evaluated based on:
- Proper error handling (i.e. exception handling);
- Good coding practices;
- Adherence to the assignment constraints

Good Coding Practices:

- **Your class is self-contained and can be run by any file;**
- Your code is properly structured into functions, without too much functionality in any one function;
- Your variables names are descriptive and easy to infer their functionality;
- Your function names are descriptive and easy to infer their functionality;
- Program properly checks for possible error conditions;
- Program properly checks for user error;
- Your code doesn't contain (nearly) duplicate code in different functions;
- Global variables are used sparingly, and appropriately;
- Your code is fairly efficient and executes in a reasonable time (i.e. roughly less than 1 minutes);

Assignment Requirements

1. print(FileDecoder) prints human-readable description;
2. FileDecoder raises DecoderException on incorrect password;
3. len(FileDecoder) properly returns number of entries in decoded CSV file;
4. FileDecoder properly iterates through decoded CSV file entries;
5. FileDecoder properly decodes provided files (1-3);
6. Program properly handles user input;
7. Program only uses regular expressions to validate the user password;
8. Program passes output tests (1-3);
9. Program passes output tests (4-6) (evaluation only);
10. Program uses robust regular expressions to validate the user password, passes tests for invalid password formats (1-6) (evaluation only);

Our grading scheme is relatively simple. See above for requirements. Not following good coding practices will lower your grade.

- "A/A+": A submission completing ALL requirements of the assignment.
- "A-": A submission that completes requirements 1-9. Passes some of the regular expression invalid password format tests.
- "B/B+": A submission that completes requirements 1-8. Passes some of the evaluation tests (4-6).
- "B-": A submission that completes requirements 1-7. Passes some of the output tests (1-6).
- "C+": A submission that completes requirements 1-6. Program does not use regular expressions or only partially uses regular expressions. Program does incorrectly states that valid passwords are invalid.
- "C": A submission that completes requirements 1-5. `FileDecoder` works as expected but `decoder.py` has issues.
- "D": A serious attempt is made to complete the requirements of the assignment. All class requirements are attempted but do not completely work successfully. Basic `decoder.py` which loads `FileDecoder` exists.
- "F": A submission that fails to complete the base requirements of the assignment. No submission given; submission represents very little work or understanding of the assignment.