

Pair Exercise #3

David Kallemeyn

Last Revised February 2024

Due: Github repository link. Include the name of your partner.

Assignment type: Pair

Points: 50

INSTRUCTIONS

Code the functions and classes as outlined below in a file named **pe3.py** and push it to a qihub repository named 303_Spring_24.

The file **test_pe3.py** has a number of tests that will be used to evaluate your code. Your code should pass 25 tests, with each passed test worth 2 points. Your code is also expected to fail 2 tests, these are not scored and included only to illustrate XFAIL (expected to fail) tests in pytest.

To test your code, do the following:

- download the **test_pe3.py** and **pytest.ini** files and place them in the same directory as your *pe3.py* file. (the .ini file supresses some warnings, it does not contain any tests). The files *must be in the same directory* or the test file as written will fail to test your file.
- make sure you have *pytest* installed (globally, or if you are working in a virtual environment make sure it is active)
- run the test file with pytest using the following command from the terminal (in the directory where your files are located):

```
pytest -v test_pe3.py
```

Functions

Caesar's cipher is a cryptography scheme attributed to the Roman Emperor Julian Caesar who used it to communicate with his associates while aiming to prevent eavesdropping. In its simplest form, the cipher uses a substitution mechanism i.e., each letter in the communicated text is replaced by another, unique letter. The substitution is specified by a shift parameter, D, which denotes how many positions down the alphabet the replacing letter resides. More details on how the cipher works can be found in this Wikipedia article:

https://en.wikipedia.org/wiki/Caesar_cipher

In this part of the exercise, you will create two functions: **encode** and **decode**.

A. Create an encoding function called **encode** that takes two arguments:

- **input_text**, which is the text to be encrypted, and
- **shift**, which is the number of places to shift along the alphabet to where the replacing letter resides.

The function should return a tuple of two items:

- a list of letters of the English alphabet in lowercase, and
- the encoded text

Example function calls and expected output are shown below:

```
encode ("a", 3) # should return ([" a" , "b" , ... "z"] , "d")
encode (" abc", 4 ) # should return ([" a" , "b" , ... "z"] , " efg ")
encode (" xyz", 3 ) # should return ([" a" , "b" , ... "z"] , " abc ")
encode ("j!K,2?", 3) # should return ([" a" , "b" , ... "z"] , "m!n,2 ?")
```

B. Create a decoding function called **decode** that takes two arguments:

- **input_text**, which is the text to be decrypted, and
- **shift**, which is the number of places to shift along the alphabet to where the replacing letter resides.

The function should return only the decoded text.

Examples of function calls for the decode function:

```
decode ("d", 3) # should return "a"
decode (" efg", 4 ) # should return " abc "
decode (" abc", 3 ) # should return " xyz "
decode ("m!n,2?", 3) # should return "j!K,2 ?"
```

Classes

A. BankAccount

Create a class called BankAccount.

The class is initialized with the following attributes:

- owner's name (name)
- alphanumeric id (ID)
- date of account creation (creation_date). this must be of type datetime.date.
- account balance (balance)

The BankAccount class should have the following default values set:

- name: "Rainy"
- ID: "1234"
- creation_date: datetime.date.today()
- balance: 0

The BankAccount class must have the following (instance) methods that take the listed arguments:

- deposit(amount)
- withdraw(amount)
- view_balance()

The BankAccount class must obey the following rules:

- the date of creation may be a past date or today, but cannot be a future date.
- A date supplied with a future creation date must raise an exception of class Exception.
- negative deposit amounts are not allowed
- withdrawal and deposit actions should display the resulting account balance.

B. SavingsAccount

Create a subclass of BankAccount called **SavingsAccount**. SavingsAccount must follow the BankAccount rules plus the following:

- withdrawals are only permitted after the account has been in existence for 180 days.
- overdrafts (negative account balance) are not permitted.

C. CheckingAccount

Create a subclass of BankAccount called **CheckingAccount**. CheckingAccount must follow the BankAccount rules plus the following:

- overdrafts are permitted, but incur a \$30 fee each time a withdrawal results in a negative balance.
-

Additional instructions

Be sure that your names are precisely consistent with the following, or the tests will not work:

Class names: BankAccount, CheckingAccount, SavingsAccount

Method Names: deposit(self, amount), withdraw(self, amount), view_balance(self)