Project 3

Python Caesar Cipher Project

## Objective

The purpose of this project is to design and implement a class for different ciphers, each with methods to encode and decode messages. Additionally, you will incorporate exception handling to manage invalid inputs and errors gracefully.

This project will help you practice:

* Object-Oriented Programming (OOP) concepts such as classes, methods, and encapsulation.
* Exception handling using try/except blocks, custom exceptions, or assertions.
* Applying concepts from **Chapter 9: Classes and Objects**and**Chapter 10: Exceptions.**

## Problem Description

Ciphers are algorithms used to encode and decode messages. In this project, you will implement a Caesar Cipher class, with methods to encode and decode messages. You will also handle invalid inputs and errors using exception handling.

### Requirements

You will create a class for each cipher. Each class must:

1. **Initialize Parameters**: Accept parameters (e.g., shift value for Caesar cipher) during initialization.
2. **Encode Method**: Convert a plaintext message into its ciphered version.
3. **Decode Method**: Convert a ciphered message back into plaintext.
4. **Validation**: Validate inputs using exception handling (e.g., raise a ValueError for invalid characters).
5. **String Representation**: Include a \_\_str\_\_ method to provide a string representation of the cipher object.

**Hint**: Use the ord() function to convert characters to their ASCII values and chr() to convert back.

## Caesar Cipher Class

You will implement the following cipher class:

* **Class Name**: CaesarCipher(shift=None)
* **Description**: Shifts each letter in the message by a fixed number of places in the alphabet.
* **Methods**:
  + \_\_init\_\_(self, shift): Initializes the cipher with a shift value. Raise an ValueError if shift is not an integer.
  + encode(self, message): Encodes the message by shifting each letter by the specified shift value. Raise an InvalidCipherTextError if the message contains non-alphabetic characters or is not a string.
  + decode(self, cipher\_text): Decodes the cipher text by reversing the shift. Raise an InvalidCipherTextError if the message contains non-alphabetic characters or is not a string.
  + \_\_str\_\_(self): Returns a string representation of the cipher, e.g., “CaesarCipher(shift=3)”.

## Exception Handling

Incorporate exception handling to manage errors such that your program does not crash on invalid input. Consider the following scenarios:

* Invalid characters in the input string (e.g., non-alphabetic characters for the Caesar cipher).
* Missing or invalid parameters (e.g., empty shift for Caesar cipher).
* Custom exceptions for specific errors (e.g., InvalidCipherTextError).

Use the following techniques:

* try/except blocks to catch and handle errors.
* Custom exception classes for specific error types.

## Program Structure

Create a Python program that demonstrates the functionality of your cipher class. The program script should:

1. Create instances of each cipher class with appropriate parameters.
2. Encode and decode sample messages.
3. Handle errors gracefully and display meaningful error messages.

**Copy and paste** the following main code structure into your program to test your implementation. This code assumes that your cipher class are implemented correctly and will raise exceptions if errors occur.

def main():

*# Create instances of each cipher class*

caesar\_cipher = CaesarCipher(3) *# Initialize Caesar Cipher with a shift of 3*

*# Sample messages*

messages = [

"HELLO", *# Valid message*

"WORLD", *# Valid message*

"INVALID123", *# Invalid: contains numbers*

"PYTHON", *# Valid message*

"$P3C!AL", *# Invalid: contains special characters*

41232, *# Invalid: not a string*

]

*# Test Caesar Cipher*

print("Testing Caesar Cipher:", end='')

for message in messages:

print(f"\nOriginal: {message}", end=' ')

encoded = caesar\_cipher.encode(message) *# Encode the message*

if encoded is not None:

decoded = caesar\_cipher.decode(encoded) *# Decode the encoded message*

print(f"Encoded: {encoded}, Decoded: {decoded}", end='')

*# Test Caesar Cipher Object print*

print(f"\n{caesar\_cipher}")

*# Test Invalid Caesar Cipher Initialization*

caesar\_cipher = CaesarCipher() *# Initialize Caesar Cipher with an invalid shift*

caesar\_cipher = CaesarCipher('three') *# Initialize Caesar Cipher with an invalid shift*

if \_\_name\_\_ == "\_\_main\_\_":

main()

### Handling Exceptions

When implementing exception handling, consider the following:

* **Unhandled Exception**: Causes the program to crash and display a Traceback.
* **Handled Exception**: Allows the program to recover and continue running.

#### **Unhandled Exception Example**

When attempting to write your code without proper exception handling in your cipher class, you might encounter an unhandled exception like this:

Testing Caesar Cipher:

Original: HELLO Encoded: KHOOR, Decoded: HELLO

Original: WORLD Encoded: ZRUOG, Decoded: WORLD

Original: INVALID123

Original: PYTHON Encoded: SBWKRQ, Decoded: PYTHON

Original: $P3C!AL

Original: 41232

Traceback (most recent call last):

File "/code/projects/project3.py", line 66, in <module>

main()

~~~~^^

File "/code/projects/project3.py", line 53, in main

encoded = caesar\_cipher.encode(message) # Encode the message

File "/code/project3.py", line 12, in encode

for char in message:

^^^^^^^

TypeError: 'int' object is not iterable

As seen above, invalid characters in the message could lead to exceptions, however, here this current example invalid characters are part of the input string which are not handled and skipped silently. When the program obtains an input that is not a string then the program crashes when it encounters the invalid input (e.g., a number instead of a string).

Therefore, ensure that your cipher class handle such exceptions gracefully such that the program continues to run and provides meaningful error messages. Then continue testing the remaining messages.

#### **Handled Exception Example**

With proper exception handling, the output should look like this:

Testing Caesar Cipher:

Original: HELLO Encoded: KHOOR, Decoded: HELLO

Original: WORLD Encoded: ZRUOG, Decoded: WORLD

Original: INVALID123 Error encoding message: Message must contain only alphabetic characters.

Original: PYTHON Encoded: SBWKRQ, Decoded: PYTHON

Original: $P3C!AL Error encoding message: Message must contain only alphabetic characters.

Original: 41232 Error encoding message: Message must be a string.

CaesarCipher(shift=3)

Error initializing CaesarCipher: Shift must be an integer.

Error initializing CaesarCipher: Shift must be an integer.

## Deliverables

* **Source Code**: The Python source code file (project3.py) containing the implementation of the cipher class and the main test script.
* **Documentation and Code Structure**: (Optional Bonus 20 points) Documentation (doc-strings and in-line comments) explaining the code and cipher rules. Clean and well-organized code will be considered for bonus points.
* **Note**: This project is an individual effort; ensure all submitted work is entirely your own. Your submission will be scrutinized for originality to maintain academic integrity; this includes your understanding of your code mentioned in the documentation.

## Unicode Reference

In Python, the ord() function returns the Unicode code point (integer representation) of a character, and the chr()chr() function converts a Unicode code point back to a character.

### Uppercase Characters

The Unicode numbers for the uppercase English alphabet (A-Z) have code points ranging from 65 to 90.

* A → 65
* B → 66
* ...
* Y → 89
* Z → 90

### Lowercase Characters

The Unicode numbers for the lowercase English alphabet (a-z) have code points ranging from 97 to 122.

* a → 97
* b → 98
* ...
* y → 121
* z → 122