10 Vigenère Cipher Implementation in Python

Jacques Mock Schindler

23.09.2025

This notebook presents a possible implementation of the Vigenère cipher in Python.

```python
def vigenere(text: str, key: str, mode: str) -> str:
    """Encrypts or decrypts a given text using the Vigenère cipher.

    The function processes a string of uppercase alphabetic characters
    using a provided key for encryption or decryption.

    Args:
        text: The string to be encrypted or decrypted. It should only contain
              uppercase alphabetic characters (A-Z).
        key: The key for the cipher. It should also only contain
             uppercase alphabetic characters (A-Z).
        mode: The operation to perform, must be 'encrypt' or 'decrypt'.

    Returns:
        The resulting ciphertext or plaintext.

    Raises:
        ValueError: If the mode is not 'encrypt' or 'decrypt'.
    """

    # Ensure the mode is valid before proceeding.
    if mode not in ['encrypt', 'decrypt']:
        raise ValueError("Mode must be 'encrypt' or 'decrypt'")

    key_length = len(key)

    if mode == 'encrypt':
       cipher = ''
       # Iterate through each character of the input text.
       for i, char in enumerate(text):
           # Convert the current text character and the corresponding key
           # character to a number (0-25).
           # The key character is determined using modulo to cycle through
           # the key.
           char_num = ord(char) - ord('A')
           key_num = ord(key[i % key_length]) - ord('A')

           # Calculate the new character's number using the Vigenère encryption formula.
           # The modulo operator ensures the result stays within the range 0-25.
           cipher_num = (char_num + key_num) % 26

           # Convert the resulting number back to an uppercase character and append it.
           cipher += chr(cipher_num + ord('A'))
       return cipher
    else:  # mode == 'decrypt'
        plain = ''
        # Iterate through each character of the input text.
        for i, char in enumerate(text):
            # Convert the current text character and the corresponding key
            # character to a number (0-25).
            char_num = ord(char) - ord('A')
            key_num = ord(key[i % key_length]) - ord('A')

            # Calculate the new character's number using the Vigenère
            # decryption formula.
            # The modulo operator handles negative results, ensuring the
            # result is correct.
            plain_num = (char_num - key_num) % 26

            # Convert the resulting number back to an uppercase
            # character and append it.
            plain += chr(plain_num + ord('A'))
        return plain
```

The `vigenere()` function is explained in detail below.

The function signature

```
def vigenere(text: str, key: str, mode: str) -> str:
```

shows that the function expects three parameters: * `text`: The string to be encrypted or decrypted. * `key`: The key for encryption or decryption as a string. * `mode`: The mode that specifies whether the text should be encrypted or decrypted. Possible values are `'encrypt'` for encryption and `'decrypt'` for decryption (although this is not directly apparent from the signature).

Following the signature is a detailed docstring that describes the function and its parameters. Although written in English, it is self-explanatory.

The docstring is followed by a check to see if valid values were passed for the `mode` parameter. If not, a `ValueError` exception is raised.

```python
if mode not in ['encrypt', 'decrypt']:
    raise ValueError("Mode must be 'encrypt' or 'decrypt'")
```

To perform this check, the allowed values are provided in a list. It then checks if the value assigned to `mode` is contained in the list. If this is not the case, an error message is displayed and the function's execution is terminated.

If the `mode` parameter contains a valid value, the actual function logic comes into play.

First, the length of the key is assigned to the variable `key_length` in

```python
key_length = len(key)
```

This information will be needed later to iterate over the key's text in a special type of loop.

After this assignment, the function splits into the encryption and decryption branches.

First, the encryption - initiated with `if mode == 'encrypt':` - is considered.

Inside this block, an empty string `cipher` is first initialized, which will later hold the encrypted characters.

Then, a `for` loop is used to iterate over the text to be encrypted.

```python
for i, char in enumerate(text):
```

In this loop, the `enumerate()` function is used. This function returns a tuple consisting of the index and the respective element of the structure being iterated over. The values of the tuple are assigned to the variables `i` and `char`.

The variables `i` and `char` are used inside the loop to convert the individual characters of the text into a number.

```python
    char_num = ord(char) - ord('A')
    key_num = ord(key[i % key_length]) - ord('A')
```

The `ord()` function converts a letter into the corresponding number from the ASCII table. To ensure the numbers are in the range of 0 to 25, the ASCII value of the letter 'A' is subtracted. The letter 'A' is used because the characters to be encrypted are specified in uppercase. Since the key can be shorter than the text to be encrypted, `i % key_length` is used to iterate over the key. The modulo operator `%` ensures that the index value of the used index always stays between 0 and the length of the key `key_length`. This ensures that the key is started again from the beginning once the end of the key is reached. The individual letters of the key are then processed in the same way as the letters of the text.

After the letters of the text and the key have been converted into numbers, the actual encryption is performed according to the formula $C_i = (P_i + K_i)\ mod\ 26$.

```
    cipher_num = (char_num + key_num) % 26
```

The numerical values of the encrypted text are then converted back into letters and appended to the string `cipher`. For this, the `chr()` function is used, which converts a number into the corresponding letter of the ASCII table. Since the numerical values are in the range of 0 to 25, the ASCII value of the letter ‘A’ is added.

```
    cipher += chr(cipher_num + ord('A'))
```

The string stored under `cipher` is returned at the end of the block.

In the second block, the decryption - initiated with `else:` - is performed. The process is very similar to that of encryption, however, the inverse formula ($P_i = (C_i - K_i + 26)\ mod\ 26$) is used here.

```
    plain_num = (char_num - key_num + 26) % 26
    plain += chr(plain_num + ord('A'))
```

Everything else corresponds to the procedure for encryption.