```
# M. Alfin Delvan Joeyantu  Program main1_120450024.py
# Author : M. Alfin Delvan Joeyantu
# NIM : 120450024
# Kelas : RB
# Affiliation : Sains Data ITERA
# Date : 30th march 2022
# Program Description : Program to solve simple encryption password problem (case

# a. Bantulah user tersebut dengan membuatkan sebuah program yang secara otomatis
def encrypt(plaintext):
    """encrypt encrypts given plaintext into a ciphertext."""
    if len(plaintext) > 100:
        raise "plaintext must be less than 100 characters"

    # cache computed token.
    computed = {}

    words = []
    for char in plaintext:
        if char not in computed:
            unicode_number = ord(char)
            token = tokenize(unicode_number)
            word = token_to_string(token)
            # store the word of token for later use.
            computed[char] = word

        # simple optimization.
        # reuse computed word of token.
        words.append(computed[char])

    return join_chars(words)


def encrypt_functional(plaintext):
    """encrypt_functional is a functional version of encrypt."""
    # notes: please read from innermost([1]) to outermost([3]).
    words = map(
        # [3] maps each token into a string representation.
        lambda token: token_to_string(token),
        map(
            # [2] maps each unicode into token.
            lambda unicode: tokenize(unicode),
            map(
                # [1] maps each character in plaintext into each unicode.
                lambda char: ord(char),
                list(plaintext),
            ),
        ),
    )

    return join_chars(words)


def decrypt(ciphertext):
```

```
def decrypt(ciphertext):
    """decrypt decrypts given ciphertext into a plaintext."""
    chars = []

    tokens = string_to_tokens(ciphertext)
    for token in tokens:
        unicode_number = detokenize(token)
        char = chr(unicode_number)
        chars.append(char)

    return join_chars(chars)


def decrypt_functional(ciphertext):
    """decrypt_functional is a functional version of decrypt."""
    # notes: please read from innermost([1]) to outermost([2]).
    chars = map(
        # [1] maps each unicode into character.
        lambda unicode: chr(unicode),
        map(
            # [2] converts each token into unicode.
            lambda token: detokenize(token),
            string_to_tokens(ciphertext),
        ),
    )

    return join_chars(chars)


def tokenize(n):
    """tokenize transforms an integer n into a token (3-column tuple of integers)
    >>> tokenize(52) == (82, 81, 45)
    True
    """

    if n <= 0:
        raise "n must be a positive number"

    a = n // 26 + 80
    b = n % 26 + 80
    c = ord('-') if a > b else ord('+')

    return a, b, c


def detokenize(token):
    """detokenize is an inverse of the tokenize function. Detokenize takes a toke
    >>> detokenize((82, 81, 45)) == 52
    True
    """

    if len(token) != 3:
        raise "token must be has exactly 3 items"

    # unpacks token tuple.
    (a, b, _) = token
    (a, b, _) = token

    if a < 0 or b < 0:
        raise "each item in token must be a positive number"

    rev_a = (a - 80) * 26
    rev_b = (b - 80)  # we can ignore the module inverse

    return rev_a + rev_b


def token_to_string(token):
    """token_to_string returns a string representation of given token"""
    if len(token) != 3:
        raise "token must be 3-column tuple"

    (a, b, c) = token

    return "{0}{1}{2}".format(chr(a), chr(b), chr(c))


def string_to_token(s):
    """string_to_token returns a token representation of a given s"""
    if len(s) < 3:
        raise "string must be at least 3 characters"

    a = ord(s[0])
    b = ord(s[1])
    c = ord(s[2])

    return a, b, c


def string_to_tokens(s):
    """string_to_tokens transform given s with length L into list of token with l

    if len(s) % 3 != 0:
        raise "length of s must be dividable by 3"

    # cache computed token.
    computed = {}

    tokens = []
    for i in range(0, len(s), 3):
        substring = s[i:i + 3]
        if substring not in computed:
            token = string_to_token(substring)
            computed[substring] = token
            tokens.append(token)
        else:
            tokens.append(computed[substring])

    return tokens


def join_chars(chars):
    """join_chars combines list of character into a word"""
    return "".join(chars)

# b. Apa output yang dihasilkan dari program tersebut jika input password adalah
plaintext = "anakanakcerdas2020"
ciphertext = encrypt(plaintext)
decrypted = decrypt(ciphertext)
print("ciphertext:", ciphertext)
print("plaintext :", decrypted)

# c. (Bonus) User tersebut lupa password asli yang dia inputkan ke dalam program
text = 'Sc+TV+Sc+TS-T[+Sc+TQ-TV+T[+Sf+Sc+T\+Sc+Qh+Qf+Qh+Qf+TS-Sg+Se+Sg+'
print('encrypted password :', text)
print('decrypted password :', decrypt(text))
```

```
ciphertext: Sc+TV+Sc+TS-Sc+TV+Sc+TS-Se+Sg+TZ+Sf+Sc+T[+Qh+Qf+Qh+Qf+
plaintext : anakanakcerdas2020
encrypted password : Sc+TV+Sc+TS-T[+Sc+TQ-TV+T[+Sf+Sc+T\+Sc+Qh+Qf+Qh+Qf+TS-Sg+Se+Sg+
decrypted password : anaksainsdata2020kece
```

✓ 0 d selesai pada 22.34 ● ×