**Write-up for Solved picoCTF Challenges**

**Challenge 1: interencdec.py**

**Description:**

This challenge involved decoding a base64-encoded string and then decrypting the result using a Caesar cipher.

**Steps to Solve:**

1. **Base64 Decoding:** The encoded flag was first base64-decoded twice.
2. **Caesar Cipher Decryption:** The output was an encrypted string that had been shifted using a Caesar cipher. A brute-force approach was applied by shifting the text from 0 to 25 positions.
3. **Correct Flag Identification:** By iterating through all possible shifts, the correct plaintext was found at shift 7.

**Code:**  
import base64

# Step 1: Base64 decode

encoded\_flag = "YidkM0JxZGtwQlRYdHFhR3g2YUhsZmF6TnFlVGwzWVROclh6ZzJhMnd6TW1zeWZRPT0nCg=="

# Decode the base64 twice

decoded\_once = base64.b64decode(encoded\_flag).decode('utf-8')

# Split by single quotes and extract the second part

extracted\_part = decoded\_once.split("'")[1]

# Decode the extracted part again with base64

decoded\_twice = base64.b64decode(extracted\_part).decode('utf-8')

# Step 2: Caesar cipher decryption function

def caesar\_decrypt(ciphertext, shift):

    decrypted = []

    for char in ciphertext:

        if char.isalpha():

            shift\_val = ord('a') if char.islower() else ord('A')

            decrypted.append(chr((ord(char) - shift\_val - shift) % 26 + shift\_val))

        else:

            decrypted.append(char)

    return ''.join(decrypted)

# Try all possible Caesar cipher shifts (0-25)

for shift in range(26):

    print(f"Shift {shift}: {caesar\_decrypt(decoded\_twice, shift)}")

**Flag:**

picoCTF{caesar\_d3cr9pt3d\_86de32d2}

**Challenge 2: MOD26.py**

**Description:**

This challenge involved decrypting a message encoded using ROT13.

**Steps to Solve:**

1. **ROT13 Decryption:** The message was encrypted using ROT13, a simple letter substitution cipher that shifts characters by 13 places.
2. **Decoding with Python:** The codecs.decode function was used to decrypt the string.

**Code:**

import codecs

# Encrypted string

encrypted\_string = "cvpbPGS{arkg\_gvzr\_V'yy\_gel\_2\_ebhaqf\_bs\_ebg13\_GYpXOHqX}"

# ROT13 decryption using codecs

decrypted\_string = codecs.decode(encrypted\_string, 'rot\_13')

# Print the result

print(f"Decrypted string: {decrypted\_string}")

**Flag:**

picoCTF{next\_time\_I'll\_try\_2\_rounds\_of\_rot13\_TLcKBUdK}

**Challenge 3: TheNumbers.py**

**Description:**

This challenge involved decrypting a sequence of numbers mapped to letters.

**Steps to Solve:**

1. **Mapping Numbers to Letters:** The numbers were mapped to their corresponding letters in the alphabet (A=1, B=2, ..., Z=26).
2. **Reconstructing the Flag:** The outside and inside parts of the flag were reconstructed separately and then combined.

**Code:**

# Function to map numbers to letters

def number\_to\_letter(number\_sequence):

    return ''.join([chr(num + 64) for num in number\_sequence])

# Encrypted sequence (outside curly braces)

outside\_braces = [16, 9, 3, 15, 3, 20, 6]

# Encrypted sequence (inside curly braces)

inside\_braces = [20, 8, 5, 14, 21, 13, 2, 5, 18, 19, 13, 1, 19, 15, 14]

# Convert numbers to letters

outside\_decrypted = number\_to\_letter(outside\_braces)

inside\_decrypted = number\_to\_letter(inside\_braces)

# Construct the flag

flag = f"{outside\_decrypted}{{{inside\_decrypted}}}"

# Output the decrypted flag

print(f"Decrypted flag: {flag}")

**Flag:**

picoCTF{THENUMBERSMASON}

**Challenge 4: 13.py**

**Description:**

This challenge was another ROT13 decryption task.

**Steps to Solve:**

1. **ROT13 Decryption:** The encrypted message was decoded using the codecs.decode function.
2. **Retrieving the Flag:** The output was the correctly decrypted flag.

**Code:**

import codecs

# Encrypted string

encrypted\_string = "cvpbPGS{abg\_gbb\_onq\_bs\_n\_ceboyrz}"

# ROT13 decryption using codecs

decrypted\_string = codecs.decode(encrypted\_string, 'rot\_13')

# Print the result

print(f"Decrypted string: {decrypted\_string}")

**Flag:**

picoCTF{not\_too\_bad\_of\_a\_problem}

**Challenge 5: Basic-mod1.py**

**Description:**

This challenge involved decrypting a modular arithmetic-based encoding scheme.

**Steps to Solve:**

1. **Modulo Operation:** Each number in the encrypted list was reduced modulo 37.
2. **Character Mapping:** The resulting values were mapped to a custom alphabet consisting of lowercase letters, digits, and an underscore.
3. **Flag Construction:** The final string was built by iterating through the list and applying the decoding logic.

**Code:**

import string

alphabet = string.ascii\_lowercase

alphabet += "0123456789\_"

flag\_enc = [350, 63, 353, 198, 114, 369, 346, 184, 202, 322, 94, 235, 114, 110, 185, 188, 225, 212, 366, 374, 261, 213]

flag = ""

for c in flag\_enc:

    pos = c % 37

    flag += alphabet[pos]

print(flag)

**Flag:**

r0und\_n\_r0und\_add17ec2

**Conclusion:**

These challenges covered a variety of cryptographic techniques, including Base64 encoding/decoding, Caesar cipher decryption, ROT13 encryption, number-to-letter mapping, and modular arithmetic decryption. Each challenge required logical thinking and a fundamental understanding of encryption techniques to retrieve the correct flag.  
  
