

## Solutions à : Message lointain

Le script à la source : challenge.py effectue un chiffrement simple, basé sur la fonction suivante pour chaque caractère du message :  $y = \text{pow}(2, x, n+1)$

où :

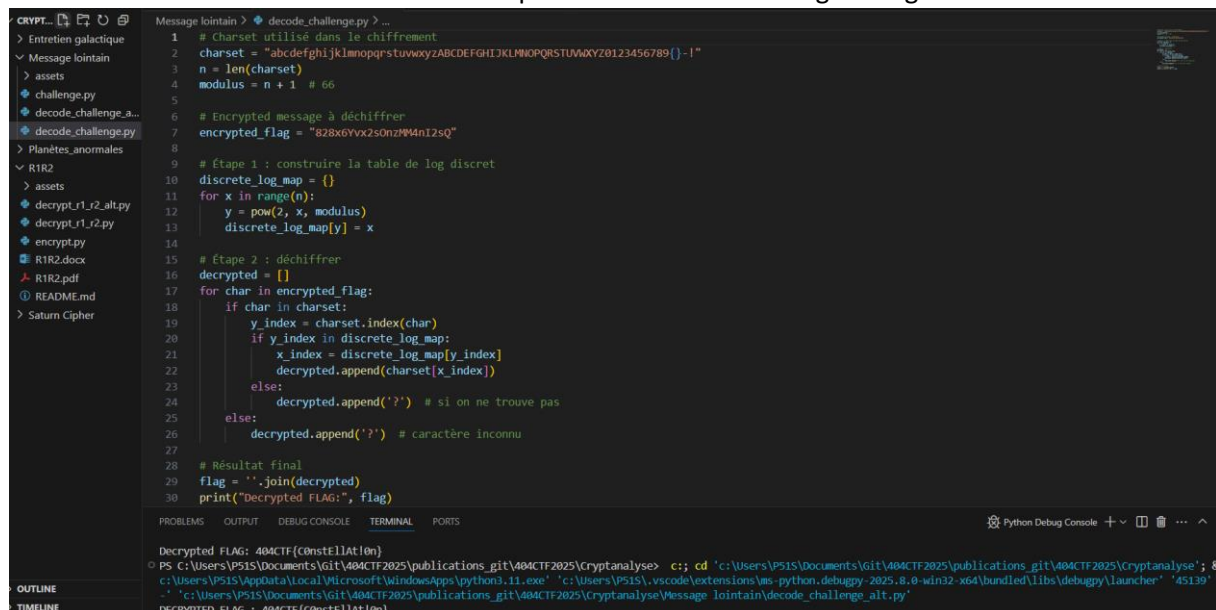
- $x$  est l'indice du caractère dans l'alphabet du jeu de caractères (*charset*)
- $n = \text{len}(\text{charset})$  (donc ici 65)
- $n+1 = 66$
- $y = 2^x \bmod 66$
- et le caractère chiffré est `charset[y]`

## Étapes pour le déchiffrement

Le chiffrement est basé sur une fonction bijective sur un sous-ensemble du *jeu de caractères*. On peut donc créer un **dictionnaire d'inversion** :

1. Pour chaque caractère  $c$  dans `charset`, calculer son indice  $x$
2. Calculer  $y = \text{pow}(2, x, 66)$
3. Si  $y < \text{len}(\text{charset})$ , on garde la correspondance `charset[y] → charset[x]`

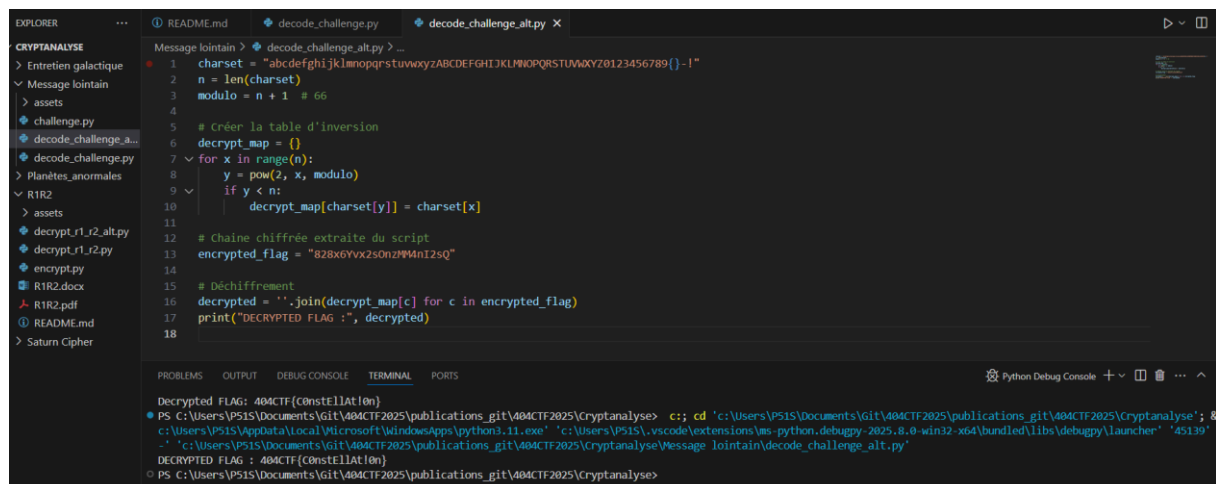
On inverse ainsi la fonction de chiffrement pour en obtenir le message d'origine.



```
1 # Charset utilisé dans le chiffrement
2 charset = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-!"
3 n = len(charset)
4 modulus = n + 1 # 66
5
6 # Encrypted message à déchiffrer
7 encrypted_flag = "828x6Vvx2s0nz9Mn12sQ"
8
9 # Étape 1 : construire la table de log discret
10 discrete_log_map = {}
11 for x in range(n):
12     y = pow(2, x, modulus)
13     discrete_log_map[y] = x
14
15 # Étape 2 : déchiffrer
16 decrypted = []
17 for char in encrypted_flag:
18     if char in charset:
19         y_index = charset.index(char)
20         if y_index in discrete_log_map:
21             x_index = discrete_log_map[y_index]
22             decrypted.append(charset[x_index])
23         else:
24             decrypted.append('?') # si on ne trouve pas
25     else:
26         decrypted.append('?') # caractère inconnu
27
28 # Résultat final
29 flag = ''.join(decrypted)
30 print("Decrypted FLAG:", flag)
```

Decrypted FLAG: 404CTF{C0nstE1lAt!0n}

## Une alternative de résolution :



```
1 charset = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-!"
2 n = len(charset)
3 modulo = n + 1 # 66
4
5 # Créer la table d'inversion
6 decrypt_map = {}
7 for x in range(n):
8     y = pow(2, x, modulo)
9     if y < n:
10         decrypt_map[charset[y]] = charset[x]
11
12 # Chaîne chiffrée extraite du script
13 encrypted_flag = "828x6Vvx2s0nz9Mn12sQ"
14
15 # Déchiffrement
16 decrypted = ''.join(decrypt_map[c] for c in encrypted_flag)
17 print("DECRYPTED FLAG :", decrypted)
```

DECRYPTED FLAG : 404CTF{C0nstE1lAt!0n}

Decrypted FLAG: 404CTF{C0nstE1lAt!0n}