Challenge #1 - Brute-force

a. Write a program that recover the cleartext password.

b. Recommend a more secure algorithm for password hashing.

b -> More secure algorithms for password hashing: PBKDF2, bcrypt, scrypt, Argon2.

We have the following small script. The embedded password is only 6 characters long.

```
AuthClass ELTE {
    self.username = administrator_elte
    rx = Crypt_SHA2(passwd.input)
    if b64.encode(hex.decode(rx))=="nyuPTeLboduudJ95DKbuMd7iHjozD8vFZ3Gjen9qDvA="
    {
    "Access granted!"
    }
}
```

```
import string
import base64
import hashlib

possibleCharacters = string.printable

correctHash = "nyuPTeLboduudJ95DKbuMd7iHjozD8vFZ3Gjen9qDvA="

#REALLY SLOW
def bruteForcePrintableSixCharactersPassword(correctHash, possibleCharacters):
    """It iterates through all the possible printable character combinations for
       a six character long password which match with the correct password's
       sha256 hash in base64 form."""

    for s1 in possibleCharacters:
        for s2 in possibleCharacters:
            for s3 in possibleCharacters:
                for s4 in possibleCharacters:
                    for s5 in possibleCharacters:
                        for s6 in possibleCharacters:
                            s = f"{s1}{s2}{s3}{s4}{s5}{s6}"
                            if (correctHash == base64.b64encode(
                                hashlib.sha256(s.encode()).digest()).decode()):
                                return s

def bruteForceAaaa11FormatSixCharactersPassword(correctHash):
    """It iterates through all the possible combinations for 'Abcd12' form
```

```
        passwords to find one which match with the correct password's
        sha256 hash in base64 form."""

    for s1 in string.ascii_uppercase:
        for s2 in string.ascii_lowercase:
            for s3 in string.ascii_lowercase:
                for s4 in string.ascii_lowercase:
                    for s5 in string.digits:
                        for s6 in string.digits:
                            s = f"{s1}{s2}{s3}{s4}{s5}{s6}"
                            if (correctHash == base64.b64encode(
                                hashlib.sha256(s.encode()).digest()).decode()):
                                return s

correctPassword = bruteForceAaaa11FormatSixCharactersPassword(correctHash)
isPasswordCorrect = (base64.b64encode(hashlib.sha256(
                     correctPassword.encode()).digest()).decode() == correctHash)

print(f"A correct password (for the given hash) is '{correctPassword}'.")
print(f"correctHash == correctPassword's hash : {isPasswordCorrect}")
```

```
A correct password (for the given hash) is 'Elte22'.
correctHash == correctPassword's hash : True
```

Challenge #2 - Finite Field

AES is using the following reducing polynomial for multiplication: x^8+ x^4 + x^3 + x + 1.

Implement the general AES GF(2^8) multiplication in any chosen programming language

and calculate the following four multiplication!

```
#Source: Wikipedia - Finite field arithmetic

#Multiply two numbers in the GF(2^8) finite field defined
#by the modulo polynomial relation x^8 + x^4 + x^3 + x + 1 = 0
def gmul(a, b):
    """a and b can be hexadecimals or integer numbers"""
    p = 0; #accumulator for the product of the multiplication
    while (a != 0 and b != 0):

        #if the polynomial for b has a constant term, add the corresponding a to p
        if (b & 1):
            p = p^a #addition in GF(2^m) is an XOR of the polynomial coefficients
```

```
        #GF modulo: if a has a nonzero term x^7, then must be reduced when it becomes x^8
        if (a & 0x80):

            # subtract (XOR) the primitive polynomial x^8 + x^4 + x^3 + x + 1
            a = (a << 1) ^ 0x11b
        else:
            a <<= 1 # equivalent to a*x
        b >>= 1

    return p

#I double checked the results with a github code:
#https://github.com/AndrewIjano/galois-field-calculator/blob/master/gf_calculator.py#L68
print(f"GF(2^8): 0xca * 0x53 = {hex(gmul(0xca, 0x53))[2:]} base16 = {gmul(0xca, 0x53)}")
print(f"GF(2^8): 0x11 * 0xda = {hex(gmul(0x11, 0xda))[2:]} base16 = {gmul(0x11, 0xda)}")
print(f"GF(2^8): 0x99 * 0xff = {hex(gmul(0x99, 0xff))[2:]} base16 = {gmul(0x99, 0xff)}")
print(f"GF(2^8): 0xcc * 0x39 = {hex(gmul(0xcc, 0x39))[2:]} base16 = {gmul(0xcc, 0x39)}")
```

```
GF(2^8): 0xca * 0x53 = 1 base16 = 1
GF(2^8): 0x11 * 0xda = d5 base16 = 213
GF(2^8): 0x99 * 0xff = 4a base16 = 74
GF(2^8): 0xcc * 0x39 = 6a base16 = 106
```

Challenge #3 - Collision

a. Provide two different strings s1 and s2 where encrypt(s1)=encrypt(s2)

b. Implement the solution in any chosen programming language.

```
import string, random, hashlib

password = "abcd1234"


#password encryption from Challenge #3
def encrypt(password):
        h = hashlib.md5(password.encode()).hexdigest()
        l = list(h)
        l.sort()
        return hashlib.md5((''.join(l)[:13]+"ABCD").encode()).hexdigest()

password_hash = encrypt(password)
collision = ""

#Generates a random string (here length = 8) from uppercase characters and digits
```

```python
def id_generator(size=8, chars=string.ascii_uppercase + string.digits):
    return ''.join(random.choice(chars) for x in range(size))

#Generates a lot of random strings and looks for a collision
for i in range(1,1000000):
    text = id_generator()
    if encrypt(text)==password_hash:
        collision = text
        break

print(f"password  = {password} -> {password_hash}")
print(f"collision = {collision} -> {encrypt(collision)}")
print(f"password_hash == collision_hash : {password_hash == encrypt(collision)}")
```

```
password  = abcd1234 -> 9e1ce92f65ca99ebadec82317a2f4a98
collision = XL3X8D0F -> 9e1ce92f65ca99ebadec82317a2f4a98
password_hash == collision_hash : True
```

Challenge #4 - AES or RSA

Describe the most important differences between AES and RSA (10-12 sentences).

RSA and AES are both encryption algorithms.

RSA is an asymmetric key algorithm. It uses two different keys (public and private key) for encryption and decryption. The public key is available to anyone but the private key is kept in secret (the private key is possessed by the owner). Public key encryption is used for exchanging data and private key encryption is used for authentication (digital signatures). It is a stream cipher algorithm. It is safe because of the hardness of factoring large integers.

AES is a symmetric key algorithm, the same key is used for both encryption and decryption. It is a 128-bit block cipher algorithm. It's strength is in the possible key permutations using Rijndael finite field method internally.

Main differences:

RSA: asymmetric, stream, slow algorithm only for encrypting short messages (like the symmetric key), it is safe because of the hardness of factoring large integers.

AES: symmetric, block cipher, fast algorithm for encrypting large data, its strength is in the possible permutations using Rijndael finite field method.