Daño, Mark Anthony Asynch Lab

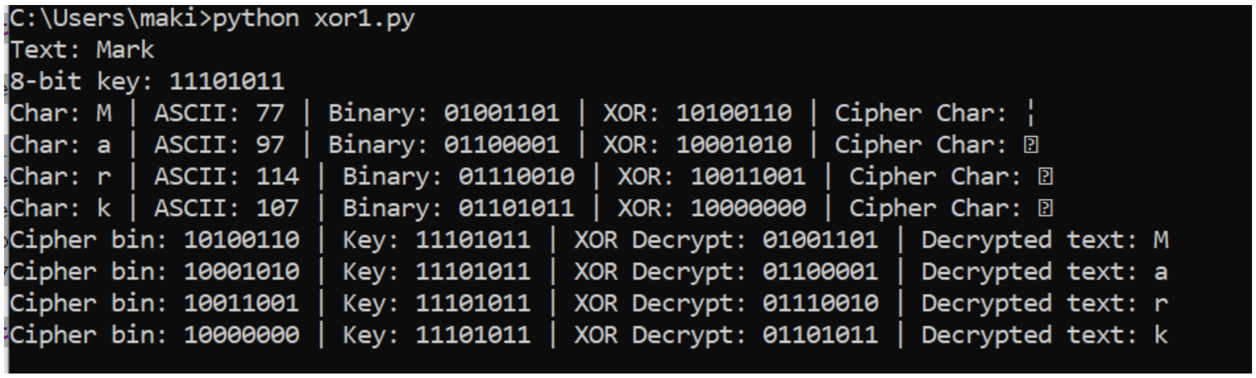
BST 3-3

XOR Cypher

Source Code:

```
text = input("Text: ")
key = input("8-bit key: ").zfill(8)

ciphertext = ""

#Encryption
for c in text:
    ascii_val = ord(c)
    bin_val = format(ascii_val, '08b')
    xor = ''.join('01'[bin_val[i] != key[i]] for i in range(8))
    cipher_char = chr(int(xor, 2))

    print(f"Char: {c} | ASCII: {ascii_val} | Binary: {bin_val} | XOR: {xor} | Cipher Char: {cipher_char}")

    ciphertext += cipher_char

#Decryption
for cipher_char in ciphertext:
    cipher_bin = format(ord(cipher_char), '08b')
    xor_back = ''.join('01'[cipher_bin[i] != key[i]] for i in range(8))
    decrypted = chr(int(xor_back, 2))

    print(f"Cipher bin: {cipher_bin} | Key: {key} | XOR Decrypt: {xor_back} | Decrypted text: {decrypted}")
```

Output:



```
C:\Users\maki>python xor1.py
Text: Mark
8-bit key: 11101011
Char: M | ASCII: 77 | Binary: 01001101 | XOR: 10100110 | Cipher Char: ¦
Char: a | ASCII: 97 | Binary: 01100001 | XOR: 10001010 | Cipher Char: ?
Char: r | ASCII: 114 | Binary: 01110010 | XOR: 10011001 | Cipher Char: ?
Char: k | ASCII: 107 | Binary: 01101011 | XOR: 10000000 | Cipher Char: ?
Cipher bin: 10100110 | Key: 11101011 | XOR Decrypt: 01001101 | Decrypted text: M
Cipher bin: 10001010 | Key: 11101011 | XOR Decrypt: 01100001 | Decrypted text: a
Cipher bin: 10011001 | Key: 11101011 | XOR Decrypt: 01110010 | Decrypted text: r
Cipher bin: 10000000 | Key: 11101011 | XOR Decrypt: 01101011 | Decrypted text: k
```

Explanation:

Encryption Process

1. The program starts by asking the user to input a message and an 8-bit key using the lines:

text = input("Text: ")

key = input("8-bit key: ").zfill(8)

- This means the user will type any text they want to encrypt and an 8-bit binary key. The .zfill(8) ensures that if the key is shorter than 8 bits, it will automatically add zeros in front to make it 8 bits long.

2. Next, an empty string called ciphertext is created to store the encrypted message: ciphertext = ""

3. The program then goes through every character in the message one by one using a loop: **for c in text:**

- This means each letter or symbol in the text will be processed separately.

4. Inside the loop, the character is first converted into its ASCII number using: **ascii_val = ord(c)**

- For example, if the character is 'A', its ASCII value is 65.

5. The ASCII value is then turned into binary form using: **bin_val = format(ascii_val, '08b')**

- This makes the ASCII value an 8-bit binary number. For example, 65 becomes 01000001.

6. After that, the program performs an XOR operation between the character's binary and the key: **xor = ''.join('01'[bin_val[i] != key[i]] for i in range(8))**

- This means that for each bit, if the bit from the binary value and the bit from the key are the same, the result is 0. If they are different, the result is 1. The XOR result becomes a new binary code that represents the encrypted form of the character.

7. The XOR result is then converted back into a normal character using: **cipher_char = chr(int(xor, 2))**

- The int(xor, 2) changes the binary into a number, and chr() converts that number back into a character. This new character is now the encrypted version of the original one.

8. The program also shows each step for better understanding by printing the details:

print(f"Char: {c} | ASCII: {ascii_val} | Binary: {bin_val} | XOR: {xor} | Cipher Char: {cipher_char}")

9. Lastly, the encrypted character is added to the ciphertext string: **ciphertext += cipher_char**

- The loop continues until all characters in the message are encrypted, forming the complete ciphertext.

Decryption Process

1. Once encryption is done, the program begins the decryption process by looping through each encrypted character: **for cipher_char in ciphertext:**

- This means it will decrypt each character one at a time.

2. Each encrypted character is converted to binary form again using:

cipher bin = format(ord(cipher_char), '08b')

- For example, if the encrypted character is 'e', its binary might be 11101011.

3. The program then performs XOR again with the same key:

xor_back = ''.join('01'[cipher_bin[i] != key[i]] for i in range(8))

- Since XOR is reversible, using it again with the same key gives back the original binary of the plaintext character.

4. The recovered binary is then changed back to a normal letter using: **decrypted = chr(int(xor_back, 2))**

- The int(xor_back, 2) turns the binary back into a number, and chr() converts it to a readable character.

5. The program prints the process for each decrypted character to show how the original message is restored:

print(f"Cipher bin: {cipher_bin} | Key: {key} | XOR Decrypt: {xor_back} | Decrypted text: {decrypted}")