**Practical No. 1**

**Title -: Write a Java/C/C++/Python program that contains a string (char pointer) with a**

**value \Hello World’. The program should AND or and XOR each character in this**

**string with 127 and display the result.**

**Code -:**

**def process\_string(string, operation):**

**result = ""**

**for char in string:**

**if operation == 'AND':**

**result += chr(ord(char) & 127)**

**elif operation == 'XOR':**

**result += chr(ord(char) ^ 127)**

**return result**

**input\_string = "Hello, World!"**

**print("Input String:", input\_string)**

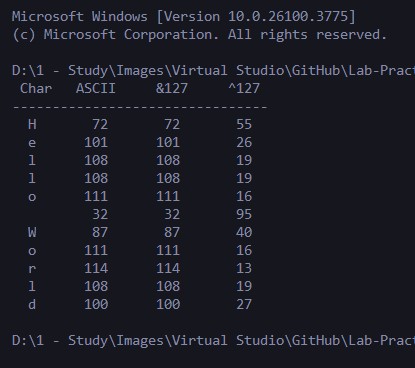
**result = process\_string(input\_string, 'AND')**

**print("Result (AND):", result)**

**result = process\_string(input\_string, 'XOR')**

**print("Result (XOR):", result)**

**Output -:**

****

**Practical No. 2**

**Title -: Write a Java/C/C++/Python program to perform encryption and decryption using the method of Transposition technique.**

**Code -:**

**import math**

**plainttext = input("Enter your plain text: ")**

**key = int(input("Enter key: "))**

**ciphertext = [''] \* key**

**for column in range(key):**

**pointer = column;**

**while pointer < len(plainttext):**

**ciphertext[column] += plainttext[pointer]**

**pointer+=key**

**print(''.join(ciphertext))**

**def main():**

**myMessage = input("Enter cipher text: ")**

**myKey = int(input("Enter key: "))**

**text = decryptMessage(myKey, myMessage)**

**print(text)**

**def decryptMessage(key, message):**

**numOfColumns = int(math.ceil(len(message) / key))**

**numOfRows = key**

**numOfShadedBoxes = (numOfColumns \* numOfRows) - len(message)**

**text = [''] \* numOfColumns**

**column = 0**

**row = 0**

**for symbol in message:**

**text[column] += symbol**

**column += 1**

**if (column == numOfColumns) or (column == numOfColumns - 1 and row >= numOfRows - numOfShadedBoxes):**

**column = 0**

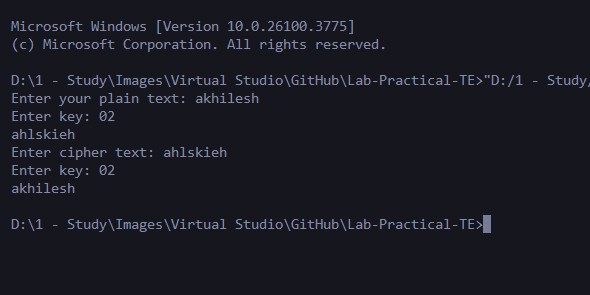
**row += 1**

**return ''.join(text)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

**Output -:**

****

**Practical No. 3**

**Title -: Write a Java/C/C++/Python program to implement DES algorithm.**

**Code -:**

**from Crypto.Cipher import DES**

**from Crypto.Util.Padding import pad, unpad**

**import base64**

**def get\_des\_key(key):**

**return key[:8].ljust(8, '0').encode()**

**def des\_encrypt(plaintext, key):**

**key = get\_des\_key(key)**

**cipher = DES.new(key, DES.MODE\_ECB)**

**padded\_text = pad(plaintext.encode(), DES.block\_size)**

**encrypted = cipher.encrypt(padded\_text)**

**return base64.b64encode(encrypted).decode()**

**def des\_decrypt(ciphertext, key):**

**key = get\_des\_key(key)**

**cipher = DES.new(key, DES.MODE\_ECB)**

**decrypted = cipher.decrypt(base64.b64decode(ciphertext))**

**return unpad(decrypted, DES.block\_size).decode()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**message = input("Enter your message: ")**

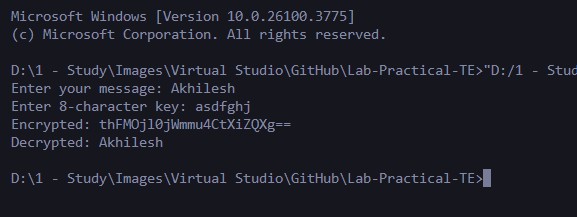
**key = input("Enter 8-character key: ")**

**encrypted = des\_encrypt(message, key)**

**print("Encrypted:", encrypted)**

**decrypted = des\_decrypt(encrypted, key)**

**print("Decrypted:", decrypted)**

**Output -:**

**Practical No. 4**

**Title -: Write a Java/C/C++/Python program to implement AES Algorithm.**

**Code -:**

**from Crypto.Cipher import AES**

**from Crypto.Util.Padding import pad, unpad**

**import base64**

**def get\_aes\_key(key):**

**return key[:16].ljust(16, '0').encode()**

**def aes\_encrypt(plaintext, key):**

**key = get\_aes\_key(key)**

**cipher = AES.new(key, AES.MODE\_CBC)**

**iv = cipher.iv**

**padded\_text = pad(plaintext.encode(), AES.block\_size)**

**encrypted = cipher.encrypt(padded\_text)**

**return base64.b64encode(iv + encrypted).decode()**

**def aes\_decrypt(ciphertext, key):**

**key = get\_aes\_key(key)**

**raw = base64.b64decode(ciphertext)**

**iv = raw[:AES.block\_size]**

**encrypted\_data = raw[AES.block\_size:]**

**cipher = AES.new(key, AES.MODE\_CBC, iv)**

**decrypted = cipher.decrypt(encrypted\_data)**

**return unpad(decrypted, AES.block\_size).decode()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**message = input("Enter your message: ")**

**key = input("Enter 16-character key: ")**

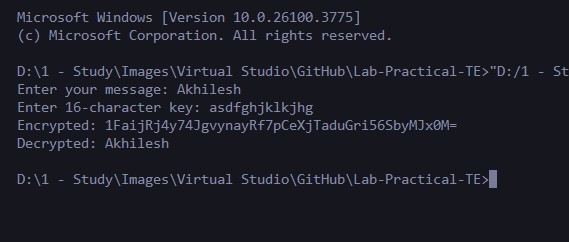
**encrypted = aes\_encrypt(message, key)**

**print("Encrypted:", encrypted)**

**decrypted = aes\_decrypt(encrypted, key)**

**print("Decrypted:", decrypted)**

**Output -:**

****

**Practical No. 5**

**Title -: Implement the Diffie-Hellman Key Exchange mechanism using HTML and**

**JavaScript. Consider the end user as one of the parties (Alice) and the JavaScript**

**application as other party (bob).**

**Code -:**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Diffie-Hellman Key Exchange</title>**

**<style>**

**body { font-family: Arial, sans-serif; padding: 20px; }**

**input, button { margin: 8px 0; padding: 6px; }**

**.output { margin-top: 20px; background: #f5f5f5; padding: 10px; border-radius: 8px; }**

**</style>**

**</head>**

**<body>**

**<h2>Diffie-Hellman Key Exchange</h2>**

**<p><strong>Public Prime (p):</strong> <span id="primeP">23</span></p>**

**<p><strong>Public Base (g):</strong> <span id="baseG">5</span></p>**

**<label for="alicePrivate">Enter your private key (Alice):</label><br>**

**<input type="number" id="alicePrivate" placeholder="e.g., 6"><br>**

**<button onclick="computeSharedKey()">Generate Shared Key</button>**

**<div class="output" id="outputArea" style="display: none;">**

**<p><strong>Alice's Public Key (A):</strong> <span id="alicePublic"></span></p>**

**<p><strong>Bob's Public Key (B):</strong> <span id="bobPublic"></span></p>**

**<p><strong>Shared Secret (Alice):</strong> <span id="sharedKeyAlice"></span></p>**

**<p><strong>Shared Secret (Bob):</strong> <span id="sharedKeyBob"></span></p>**

**</div>**

**<script>**

**function modPow(base, exponent, mod) {**

***let* result = 1;**

**base = base % mod;**

**while (exponent > 0) {**

**if (exponent % 2 === 1) {**

**result = (result \* base) % mod;**

**}**

**exponent = Math.floor(exponent / 2);**

**base = (base \* base) % mod;**

**}**

***return* result;**

**}**

**function computeSharedKey() {**

***const* p = 23;**

***const* g = 5;**

***const* alicePrivate = parseInt(document.getElementById("alicePrivate").value);**

**if (isNaN(alicePrivate) || alicePrivate <= 0) {**

**alert("Please enter a valid private key for Alice.");**

***return*;**

**}**

***const* bobPrivate = Math.floor(Math.random() \* 10) + 1;**

***const* A = modPow(g, alicePrivate, p);**

***const* B = modPow(g, bobPrivate, p);**

***const* sharedKeyAlice = modPow(B, alicePrivate, p);**

***const* sharedKeyBob = modPow(A, bobPrivate, p);**

**document.getElementById("alicePublic").textContent = A;**

**document.getElementById("bobPublic").textContent = B;**

**document.getElementById("sharedKeyAlice").textContent = sharedKeyAlice;**

**document.getElementById("sharedKeyBob").textContent = sharedKeyBob;**

**document.getElementById("outputArea").style.display = "block";**

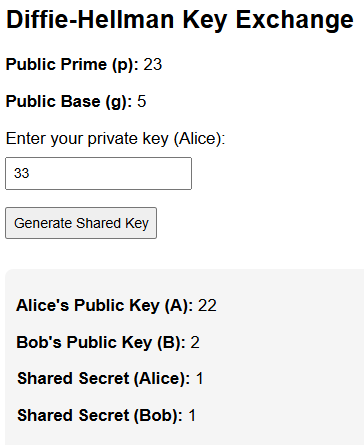
**}**

**</script>**

**</body>**

**</html>**

**Output -:**

****