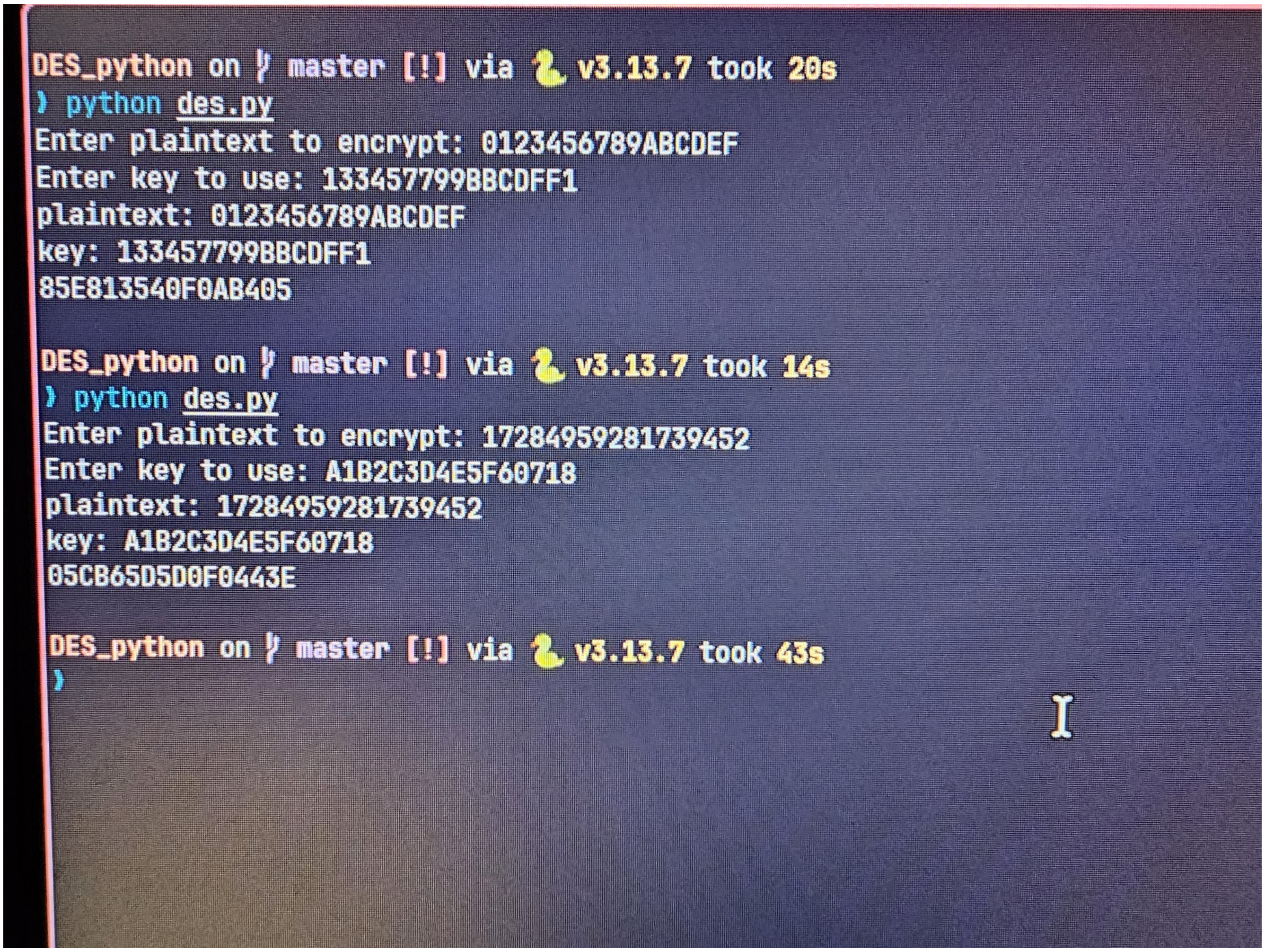
```
# Here is the source code for my des algorithim

pc_1 = [ #key permutation table one
    57, 49, 41, 33, 25, 17, 9,
    1, 58, 50, 42, 34, 26, 18,
    10, 2, 59, 51, 43, 35, 27,
    19, 11, 3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14, 6, 61, 53, 45, 37, 29,
    21, 13, 5, 28, 20, 12, 4
]
pc_2 = [ #key permutation table 2
    14, 17, 11, 24, 1, 5,
    3, 28, 15, 6, 21, 10,
    23, 19, 12, 4, 26, 8,
    16, 7, 27, 20, 13, 2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
]

ip_table = [
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
]

e_table = [
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1
]

s_boxes = [
    # S1
```

```
[
    [14, 4, 13, 1,  2, 15, 11, 8,  3, 10, 6, 12, 5, 9,  0, 7],
    [0, 15, 7, 4, 14, 2,  13, 1, 10, 6, 12, 11, 9, 5,  3, 8],
    [4, 1, 14, 8, 13, 6,   2, 11, 15, 12, 9, 7,  3, 10, 5, 0],
    [15, 12, 8, 2,  4, 9,  1, 7,  5, 11, 3, 14, 10, 0,  6, 13]
],

# S2
[
    [15, 1, 8, 14, 6, 11, 3, 4,  9, 7,  2, 13, 12, 0, 5, 10],
    [3, 13, 4, 7, 15, 2,  8, 14, 12, 0, 1, 10,  6, 9, 11, 5],
    [0, 14, 7, 11, 10, 4, 13, 1,  5, 8, 12, 6,  9, 3,  2, 15],
    [13, 8, 10, 1,  3, 15, 4, 2, 11, 6,  7, 12, 0, 5, 14, 9]
],

# S3
[
    [10, 0, 9, 14, 6, 3, 15, 5,  1, 13, 12, 7, 11, 4,  2, 8],
    [13, 7, 0, 9,  3, 4,  6, 10, 2, 8,  5, 14, 12, 11, 15, 1],
    [13, 6, 4, 9,  8, 15, 3, 0, 11, 1,  2, 12, 5, 10, 14, 7],
    [1, 10, 13, 0, 6, 9,  8, 7,  4, 15, 14, 3, 11, 5,  2, 12]
],

# S4
[
    [7, 13, 14, 3,  0, 6,  9, 10, 1, 2,  8, 5, 11, 12, 4, 15],
    [13, 8, 11, 5,  6, 15, 0, 3,  4, 7,  2, 12, 1, 10, 14, 9],
    [10, 6, 9, 0,  12, 11, 7, 13, 15, 1, 3, 14, 5, 2,  8, 4],
    [3, 15, 0, 6,  10, 1, 13, 8,  9, 4,  5, 11, 12, 7, 2, 14]
],

# S5
[
    [2, 12, 4, 1,  7, 10, 11, 6,  8, 5, 3, 15, 13, 0, 14, 9],
    [14, 11, 2, 12, 4, 7,  13, 1,  5, 0, 15, 10, 3, 9,  8, 6],
    [4, 2, 1, 11, 10, 13, 7, 8,  15, 9, 12, 5, 6, 3,  0, 14],
    [11, 8, 12, 7,  1, 14, 2, 13, 6, 15, 0, 9,  10, 4, 5, 3]
],

# S6
[
    [12, 1, 10, 15, 9, 2,  6, 8,  0, 13, 3, 4, 14, 7, 5, 11],
    [10, 15, 4, 2,  7, 12, 9, 5,  6, 1,  13, 14, 0, 11, 3, 8],
    [9, 14, 15, 5,  2, 8, 12, 3,  7, 0,  4, 10, 1, 13, 11, 6],
    [4, 3, 2, 12,  9, 5, 15, 10, 11, 14, 1, 7,  6, 0, 8, 13]
],
    # S7
    [
        [4, 11, 2, 14, 15, 0, 8, 13,  3, 12, 9, 7, 5, 10, 6, 1],
        [13, 0, 11, 7,  4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
        [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
        [6, 11, 13, 8,  1, 4, 10, 7,  9, 5, 0, 15, 14, 2, 3, 12]
    ],

    # S8
    [
        [13, 2, 8, 4,  6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
        [1, 15, 13, 8, 10, 3,  7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
        [7, 11, 4, 1,  9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
        [2, 1, 14, 7,  4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]
    ]
]

p_table = [
    16, 7, 20, 21,
    29, 12, 28, 17,
    1, 15, 23, 26,
    5, 18, 31, 10,
    2, 8, 24, 14,
    32, 27, 3, 9,
    19, 13, 30, 6,
    22, 11, 4, 25
]

final_table = [
    40, 8, 48, 16, 56, 24, 64, 32,
    39, 7, 47, 15, 55, 23, 63, 31,
    38, 6, 46, 14, 54, 22, 62, 30,
    37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28,
    35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26,
    33, 1, 41, 9, 49, 17, 57, 25
]


def xor(a, b):
    return ''.join('0' if i == j else '1' for i, j in zip(a, b))



def hex_to_bin(hex_text, bit_size=64):
    """Convert hex to a zero-padded binary string of given size."""
    return format(int(hex_text, 16), f'0{bit_size}b')
```

```
def permute(bit_string, table):
    """Rearrange bits in a string according to the table."""
    return ''.join(bit_string[i-1] for i in table)

def make_keys(hex_key):
    bin_key = hex_to_bin(hex_key, 64)
    # apply PC-1
    permuted = permute(bin_key, pc_1)
    c, d = permuted[:28], permuted[28:]
    shifts = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]
    keys = []
    for s in shifts:
        c = c[s:] + c[:s]
        d = d[s:] + d[:s]
        combined = c + d
        round_key = permute(combined, pc_2)
        keys.append(round_key)
    return keys

def sbox_substitution(bits48):
    result = ''
    for i in range(8):
        block = bits48[i*6:(i+1)*6]
        row = int(block[0] + block[5], 2)
        col = int(block[1:5], 2)
        val = s_boxes[i][row][col]
        result += format(val, '04b')
    return result

def f_func(right, cur_key):
    right = permute(right, e_table)
    right = xor(right, cur_key)
    substituted = sbox_substitution(right)
    result = permute(substituted, p_table)
    return result


def des_rounds(init_text, keys_list):
    left = init_text[:32]
    right = init_text[32:]

    for i in range(16):
        new_right = xor(left, f_func(right, keys_list[i]))
        left = right
        right = new_right
    return right + left


def main():
    plaintext = input("Enter plaintext to encrypt: ")
    hex_key = input("Enter key to use: ")

    #plaintext = "0123456789ABCDEF"
    #hex_key = "133457799BBCDFF1"
    print(f"plaintext: {plaintext}\nkey: {hex_key}")

    keys_list = make_keys(hex_key)
    #print(f"Final keys list: {keys_list}")

    bin_plaintext = hex_to_bin(plaintext)
    i_permutation = permute(bin_plaintext, ip_table)
    rounds_result = des_rounds(i_permutation, keys_list)
    cipher_bin = permute(rounds_result, final_table)
    cipher_hex = format(int(cipher_bin, 2), '016X')
    print(cipher_hex)

if __name__ == "__main__":
    main()




#both the given parameters and my own are in the screenshot:
```



```
DES_python on  master [!] via 🐍 v3.13.7 took 20s
❯ python des.py
Enter plaintext to encrypt: 0123456789ABCDEF
Enter key to use: 133457799BBCDFF1
plaintext: 0123456789ABCDEF
key: 133457799BBCDFF1
85E813540F0AB405

DES_python on  master [!] via 🐍 v3.13.7 took 14s
❯ python des.py
Enter plaintext to encrypt: 17284959281739452
Enter key to use: A1B2C3D4E5F60718
plaintext: 17284959281739452
key: A1B2C3D4E5F60718
05CB65D5D0F0443E

DES_python on  master [!] via 🐍 v3.13.7 took 43s
❯
```