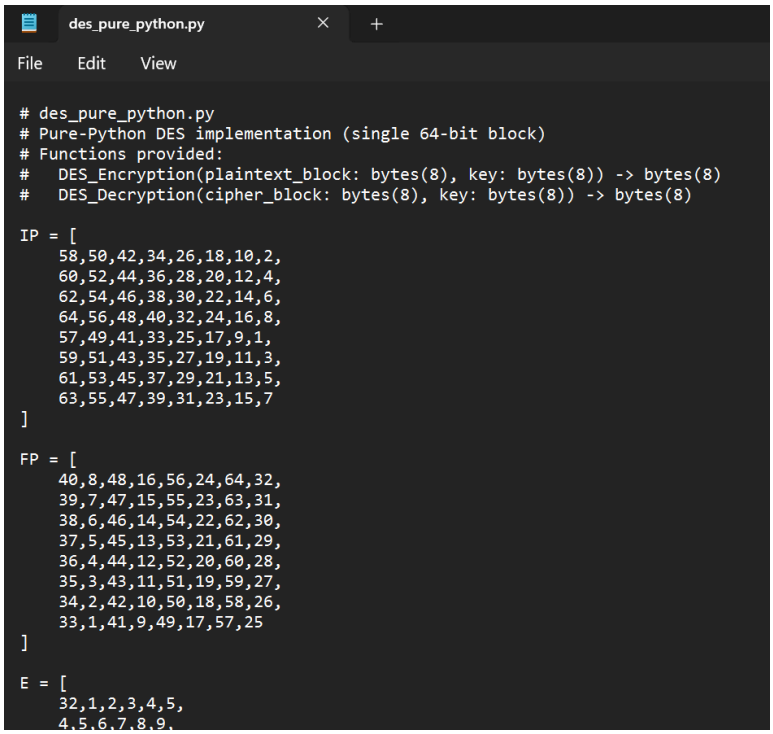
For this assignment, I only used notepad and the command line.

```
:\Users\Samsung\Downloads>des_pure_python.py
1. PT=4142434445464748 -> CT=D46EB4C9228013B7 -> DECRYPTED=4142434445464748 (ascii try: b'ABCDEFGH')
2. PT=0000000000000000 -> CT=3906333BE29DD787 -> DECRYPTED=0000000000000000 (ascii try: b'\x00\x00\x00\x00\x00\x00\x00\
00')
3. PT=FFFFFFFFFFFFFFFF -> CT=FB31361E72588617 -> DECRYPTED=FFFFFFFFFFFFFFFF (ascii try: b'\xff\xff\xff\xff\xff\xff\xff\
ff')
4. PT=3132333435363738 -> CT=19B710B50E00C28B -> DECRYPTED=3132333435363738 (ascii try: b'12345678')
5. PT=4142434445464721 -> CT=617EC771B2DB9195 -> DECRYPTED=4142434445464721 (ascii try: b'ABCDEFG!')
6. PT=5445535444415441 -> CT=52E759902A797653 -> DECRYPTED=5445535444415441 (ascii try: b'TESTDATA')
7. PT=4F70656E41492121 -> CT=478D679044A5AADF -> DECRYPTED=4F70656E41492121 (ascii try: b'OpenAI!!')
8. PT=706173737 76F7264 -> CT=B8459FCFC897B312 -> DECRYPTED=706173737 76F7264 (ascii try: b'password')
9. PT=0123456789ABCDEF -> CT=28B9D522081721B3 -> DECRYPTED=0123456789ABCDEF (ascii try: b'\x01#Eg\x89\xab\xcd\xef')
0. PT=48656C6C6F444553 -> CT=4671180FCB8F3D8A -> DECRYPTED=48656C6C6F444553 (ascii try: b'HelloDES')
```



des_pure_python.py

File Edit View

```
# des_pure_python.py
# Pure-Python DES implementation (single 64-bit block)
# Functions provided:
#   DES_Encryption(plaintext_block: bytes(8), key: bytes(8)) -> bytes(8)
#   DES_Decryption(cipher_block: bytes(8), key: bytes(8)) -> bytes(8)

IP = [
    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
]

FP = [
    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
]

E = [
    32,1,2,3,4,5,
    4,5,6,7,8,9,
```

```
ROTATIONS = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def bytes_to_bitlist(b):
    bits = []
    for byte in b:
        for i in range(8)[::-1]:
            bits.append((byte >> i) & 1)
    return bits

def bitlist_to_bytes(bits):
    out = bytearray()
    for i in range(0, len(bits), 8):
        byte = 0
        for j in range(8):
            byte = (byte << 1) | bits[i+j]
        out.append(byte)
    return bytes(out)

def permute(bits, table):
    return [bits[i-1] for i in table]

def left_rotate(lst, n):
    return lst[n:] + lst[:n]

def generate_subkeys(key8bytes):
    key_bits = bytes_to_bitlist(key8bytes)  # 64 bits (parity bits may be included in key input but ignored by PC1)
    key56 = permute(key_bits, PC1)  # 56 bits
    C = key56[:28]
    D = key56[28:]
```

```
    return des_decrypt_block(cipher_block, subkeys)

# ---------------------------
# Example usage
if __name__ == "__main__":
    key = b'ABCDEFGH'
    tests = [
        b'ABCDEFGH', b'\x00'*8, b'\xff'*8, b'12345678', b'ABCDEFG!',
        b'TESTDATA', b'OpenAI!!', b'password', bytes.fromhex('0123456789ABCDEF'), b'HelloDES'
    ]
    for i,pt in enumerate(tests,1):
        ct = DES_Encryption(pt, key)
        dt = DES_Decryption(ct, key)
        print(f"{i:2}. PT={pt.hex().upper()} -> CT={ct.hex().upper()} -> DECRYPTED={dt.hex().upper()} (ascii try: {dt})")
```

des_pure_python.py
Pure-Python DES implementation (single 64-bit block)
Functions provided:
DES_Encryption(plaintext_block: bytes(8), key: bytes(8)) -> bytes(8)
DES_Decryption(cipher_block: bytes(8), key: bytes(8)) -> bytes(8)

IP = [
   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
]

FP = [
   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
]

E = [
    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
]

P = [
    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
]

PC1 = [
    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
]

PC2 = [
    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
]

S_BOX = [
# 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]
]

ROTATIONS = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def bytes_to_bitlist(b):
    bits = []
    for byte in b:
        for i in range(8)[::-1]:
            bits.append((byte >> i) & 1)
    return bits

def bitlist_to_bytes(bits):
    out = bytearray()
    for i in range(0, len(bits), 8):
        byte = 0
        for j in range(8):
            byte = (byte << 1) | bits[i+j]
        out.append(byte)
    return bytes(out)

def permute(bits, table):
    return [bits[i-1] for i in table]

def left_rotate(lst, n):
    return lst[n:] + lst[:n]

def generate_subkeys(key8bytes):
    key_bits = bytes_to_bitlist(key8bytes)  # 64 bits (parity bits may be included in key input but
ignored by PC1)
    key56 = permute(key_bits, PC1)  # 56 bits
    C = key56[:28]
    D = key56[28:]
    subkeys = []
    for r in range(16):
        C = left_rotate(C, ROTATIONS[r])
        D = left_rotate(D, ROTATIONS[r])
        CD = C + D
        subkey = permute(CD, PC2)  # 48 bits
        subkeys.append(subkey)
    return subkeys

def sbox_substitution(bits48):
    output = []
```

```
    for i in range(8):
        block = bits48[i*6:(i+1)*6]
        row = (block[0] << 1) | block[5]
        col = (block[1] << 3) | (block[2] << 2) | (block[3] << 1) | block[4]
        val = S_BOX[i][row*16 + col]
        for j in range(4)[::-1]:
            output.append((val >> j) & 1)
    return output

def feistel(R, subkey):
    expanded = permute(R, E)
    xored = [a ^ b for a,b in zip(expanded, subkey)]
    sboxed = sbox_substitution(xored)
    permuted = permute(sboxed, P)
    return permuted

def des_encrypt_block(block8bytes, subkeys):
    bits = bytes_to_bitlist(block8bytes)
    bits = permute(bits, IP)
    L = bits[:32]
    R = bits[32:]
    for i in range(16):
        new_R = [l ^ r for l,r in zip(L, feistel(R, subkeys[i]))]
        L = R
        R = new_R
    preoutput = R + L
    final_bits = permute(preoutput, FP)
    return bitlist_to_bytes(final_bits)

def des_decrypt_block(block8bytes, subkeys):
    # decrypt by running the same encryption routine with subkeys in reverse order
    return des_encrypt_block(block8bytes, subkeys[::-1])

def DES_Encryption(plaintext_block: bytes, key: bytes) -> bytes:
    if len(plaintext_block) != 8 or len(key) != 8:
        raise ValueError("plaintext_block and key must each be 8 bytes long")
    subkeys = generate_subkeys(key)
    return des_encrypt_block(plaintext_block, subkeys)

def DES_Decryption(cipher_block: bytes, key: bytes) -> bytes:
    if len(cipher_block) != 8 or len(key) != 8:
        raise ValueError("cipher_block and key must each be 8 bytes long")
    subkeys = generate_subkeys(key)
    return des_decrypt_block(cipher_block, subkeys)
```

```
# ---------------------------
# Example usage
if __name__ == "__main__":
    key = b'ABCDEFGH'
    tests = [
        b'ABCDEFGH', b'\x00'*8, b'\xff'*8, b'12345678', b'ABCDEFG!',
        b'TESTDATA', b'OpenAI!!', b'password', bytes.fromhex('0123456789ABCDEF'),
b'HelloDES'
    ]
    for i,pt in enumerate(tests,1):
        ct = DES_Encryption(pt, key)
        dt = DES_Decryption(ct, key)
        print(f"{i:2}. PT={pt.hex().upper()} -> CT={ct.hex().upper()} ->
DECRYPTED={dt.hex().upper()} (ascii try: {dt})")
```