Source Code

```
class DES:

    def final_permutation(self, plain_text):
        """
        inputs: str
        output: str
        """
        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]
        return self.permutation(plain_text, table, 64)

    def key_permutation_2(self, key):
        """
        inputs: str
        outputs: str
        """
        table =  [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
    ]
        return self.permutation(key, table, 56)

    def rotate_key_bits_1(self, key):
        """
        inputs: str
        output: str
        """
        binary_value = self.hex_to_binary(key, 56)
        left = binary_value[:28]
        right = binary_value[28:]

        left_rotated = self.rotate_left(left)
```

```
        right_rotated = self.rotate_left(right)

        rotated_binary_value = left_rotated + right_rotated

        hex_str = self.binary_to_hex(rotated_binary_value,14)

        return hex_str

    def rotate_key_bits_2(self, key):
        """
        inputs: str
        output: str
        """
        binary_value = self.hex_to_binary(key, 56)
        left = binary_value[:28]
        right = binary_value[28:]

        left_rotated_1 = self.rotate_left(left)
        left_rotated_2 = self.rotate_left(left_rotated_1)
        right_rotated_1 = self.rotate_left(right)
        right_rotated_2 = self.rotate_left(right_rotated_1)

        rotated_binary_value = left_rotated_2 + right_rotated_2

        hex_str = self.binary_to_hex(rotated_binary_value, 14)

        return hex_str

    def rounds(self, plain_text, key, i=1):
        """
        inputs: str, str, int
        output: str
        """
        left1 = plain_text[:8]
        right1 = plain_text[8:]

        left2 = left1
        right2 = right1
        rotated_key = key

        for i in range(1,17):
            if i in (1,2,9,16):
                rotated_key = self.rotate_key_bits_1(rotated_key)
                k = self.key_permutation_2(rotated_key)
                f = self.f_function(right2,k)
```

```
                right1 = self.xor_hex(left2, f)
                left1 = right2

                right2 = right1
                left2= left1

            else:
                rotated_key = self.rotate_key_bits_2(rotated_key)
                k = self.key_permutation_2(rotated_key)
                f = self.f_function(right2,k)
                right1 = self.xor_hex(left2, f)
                left1 = right2

                right2 = right1
                left2= left1
        return right2 + left2

    def xor_hex(self, hex_value_1, hex_value_2):
        """
        inputs: str, str
        output: str
        """
        int_value_1 = int(hex_value_1, 16)
        int_value_2 = int(hex_value_2, 16)

        result_int = int_value_1 ^ int_value_2

        hex_value = hex(result_int)[2:]
        return hex_value

    def p_permutation(self, plain_text):
        """
        inputs: str
        output: str
        """

        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]
        return self.permutation(plain_text,table,32)

    def f_function(self, plain_text,key):
        """
        inputs: str, str
        output: str
```

```
        """
        e = self.expansion_32_48(plain_text)
        new_key = self.xor_hex(e, key)
        binary_value = self.hex_to_binary(new_key, 48)
        s_box_substitution = self.s1(binary_value[:6]) +
self.s2(binary_value[6:12]) + self.s3(binary_value[12:18]) +
self.s4(binary_value[18:24]) + self.s5(binary_value[24:30]) +
self.s6(binary_value[30:36]) + self.s7(binary_value[36:42]) +
self.s8(binary_value[42:48])
        return self.p_permutation(s_box_substitution)

    def s_box(self, binary_value, table):
        """
        inputs: str, int list list
        output: str
        """



        row = int(binary_value[0] + binary_value[5],2)
        column = int(binary_value[1:5],2)

        # Convert int to hex
        int_value = table[row][column]
        hex_value = hex(int_value)[2:]

        return hex_value

    def s1(self, plain_text):
        """
        inputs: str
        output: str
        """
        table = [
        [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]
    ]
        return self.s_box(plain_text, table)

    def s2(self, plain_text):
        """
        inputs: str
        output: str
        """
```

```
        table = [
        [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]
    ]
        return self.s_box(plain_text, table)

    def s3(self, plain_text):
        """
        inputs: str
        output: str
        """
        table = [
        [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]
    ]
        return self.s_box(plain_text, table)

    def s4(self, plain_text):
        """
        inputs: str
        output: str
        """
        table = [
        [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],
    ]
        return self.s_box(plain_text, table)

    def s5(self, plain_text):
        """
        inputs: str
        output: str
        """
        table = [
        [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],
    ]
```

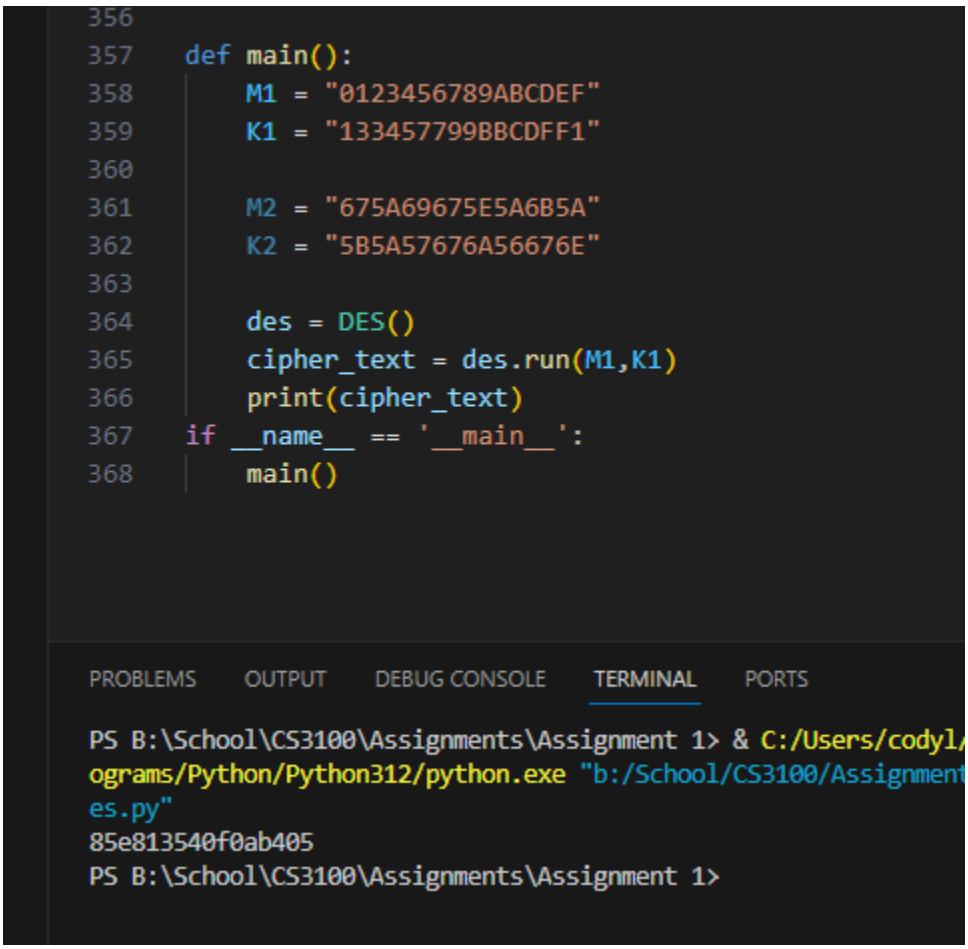
```
        return self.s_box(plain_text, table)

    def s6(self, plain_text):
        """
        inputs: str
        output: str
        """
        table = [
        [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],
    ]
        return self.s_box(plain_text, table)

    def s7(self, plain_text):
        """
        inputs: str
        output: str
        """
        table = [
        [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],
    ]
        return self.s_box(plain_text, table)

    def s8(self, plain_text):
        """
        inputs: str
        output: str
        """
        table = [
        [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],
    ]
        return self.s_box(plain_text, table)

    def expansion_32_48(self, plain_text):
        """
        inputs: str
        output: str
```

```
        """
        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]
        return self.permutation(plain_text, table, 32)

    def key_permutation_1(self, key):
        """
        inputs: str
        output: str
        """

        table = [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]
        return self.permutation(key,table,64)

    def rotate_left(self, binary_str):
        """
        inputs: str
        output: str
        """
        binary_list = list(binary_str)

        first_bit = binary_list.pop(0)
        binary_list.append(first_bit)

        rotated_binary_str = ''.join(binary_list)

        return rotated_binary_str

    def hex_to_binary(self, hex_str, bit):
        """
        input type: str
        output type: binary
        """
        int_value = int(hex_str, 16)  # Convert hex to integer
        binary_value = bin(int_value)[2:].zfill(bit) # Convert integer to binary
        return binary_value
```

```
    def binary_to_hex(self, binary_string, bit):
        """
        input type: int list
        output type: str
        """

        int_value = int(binary_string, 2)
        hex_value = hex(int_value)[2:].zfill(bit)
        return hex_value

    def permutation(self, plain_text, table, bit_size, ):
        """
        inputs: str, int list, int
        output: str
        """
        cipher_text = []

        binary_value = self.hex_to_binary(plain_text, bit_size) # make it 64bit
        for i in range(len(table)):
            cipher_text.append(binary_value[table[i]-1])

        binary_string = ''.join(str(bit) for bit in cipher_text)

        hex_value = self.binary_to_hex(binary_string, int(len(table)/4))
        return hex_value

    def initial_permutation(self, plain_text):
        """
        inputs: str
        output: str
        """
        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
,32,24,16,8,57,49,41,33,25,17,9,1,59,51,43,35,27,19,11,3,61,53,45,37,29,21,13,5,6
3,55,47,39,31,23,15,7]
        return self.permutation(plain_text, table, 64)

    def run(self, plain_text, key):
        """
        inputs: str, str
        output: str
        """
        ip = self.initial_permutation(plain_text)

        k = self.key_permutation_1(key).zfill(14)
```

```
        rnds = self.rounds(ip,k)

        fp = self.final_permutation(rnds)

        return fp

def main():
    M1 = "0123456789ABCDEF"
    K1 = "133457799BBCDFF1"

    M2 = "675A69675E5A6B5A"
    K2 = "5B5A57676A56676E"

    des = DES()
    cipher_text = des.run(M1,K1)
    print(cipher_text)
if __name__ == '__main__':
    main()
```

OutPuts



```
356
357 def main():
358     M1 = "0123456789ABCDEF"
359     K1 = "133457799BBCDFF1"
360
361     M2 = "675A69675E5A6B5A"
362     K2 = "5B5A57676A56676E"
363
364     des = DES()
365     cipher_text = des.run(M1,K1)
366     print(cipher_text)
367 if __name__ == '__main__':
368     main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS B:\School\CS3100\Assignments\Assignment 1> & C:/Users/codyl/
ograms/Python/Python312/python.exe "b:/School/CS3100/Assignment
es.py"
85e813540f0ab405
PS B:\School\CS3100\Assignments\Assignment 1>
```

```python
def main():
    M1 = "0123456789ABCDEF"
    K1 = "133457799BBCDFF1"

    M2 = "675A69675E5A6B5A"
    K2 = "5B5A57676A56676E"

    des = DES()
    cipher_text = des.run(M2,K2)
    print(cipher_text)
if __name__ == '__main__':
    main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
PS B:\School\CS3100\Assignments\Assignment 1> & C:/Users/codyl/AppData/Local/Programs/Python/Python312/python.exe "b:/School/CS3100/Assignments/Assignment 1/des.py"
974affbf86022d1f
PS B:\School\CS3100\Assignments\Assignment 1>
```