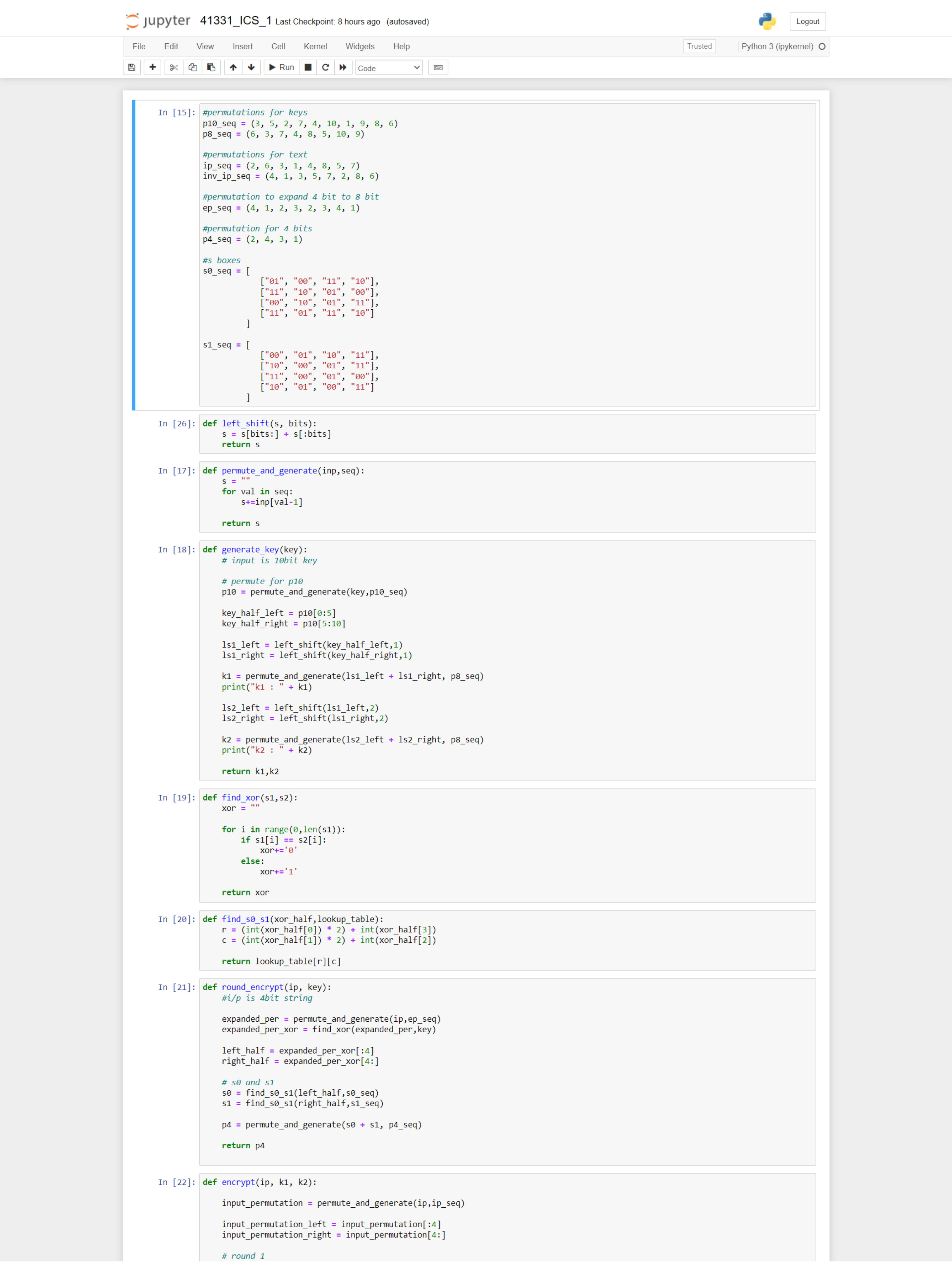

jupyter 41331_ICS_1 Last Checkpoint: 8 hours ago (autosaved)

In [15]:

```
#permutations for keys
p10_seq = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)
p8_seq = (6, 3, 7, 4, 8, 5, 10, 9)

#permutations for text
ip_seq = (2, 6, 3, 1, 4, 8, 5, 7)
inv_ip_seq = (4, 1, 3, 5, 7, 2, 8, 6)

#permutation to expand 4 bit to 8 bit
ep_seq = (4, 1, 2, 3, 2, 3, 4, 1)

#permutation for 4 bits
p4_seq = (2, 4, 3, 1)

#s boxes
s0_seq = [
            ["01", "00", "11", "10"],
            ["11", "10", "01", "00"],
            ["00", "10", "01", "11"],
            ["11", "01", "11", "10"]
         ]

s1_seq = [
            ["00", "01", "10", "11"],
            ["10", "00", "01", "11"],
            ["11", "00", "01", "00"],
            ["10", "01", "00", "11"]
         ]
```

In [26]:

```
def left_shift(s, bits):
    s = s[bits:] + s[:bits]
    return s
```

In [17]:

```
def permute_and_generate(inp,seq):
    s = ""
    for val in seq:
        s+=inp[val-1]

    return s
```

In [18]:

```
def generate_key(key):
    # input is 10bit key

    # permute for p10
    p10 = permute_and_generate(key,p10_seq)

    key_half_left = p10[0:5]
    key_half_right = p10[5:10]

    ls1_left = left_shift(key_half_left,1)
    ls1_right = left_shift(key_half_right,1)

    k1 = permute_and_generate(ls1_left + ls1_right, p8_seq)
    print("k1 : " + k1)

    ls2_left = left_shift(ls1_left,2)
    ls2_right = left_shift(ls1_right,2)

    k2 = permute_and_generate(ls2_left + ls2_right, p8_seq)
    print("k2 : " + k2)

    return k1,k2
```

In [19]:

```
def find_xor(s1,s2):
    xor = ""

    for i in range(0,len(s1)):
        if s1[i] == s2[i]:
            xor+='0'
        else:
            xor+='1'

    return xor
```

In [20]:

```
def find_s0_s1(xor_half,lookup_table):
    r = (int(xor_half[0]) * 2) + int(xor_half[3])
    c = (int(xor_half[1]) * 2) + int(xor_half[2])

    return lookup_table[r][c]
```

In [21]:

```
def round_encrypt(ip, key):
    #i/p is 4bit string

    expanded_per = permute_and_generate(ip,ep_seq)
    expanded_per_xor = find_xor(expanded_per,key)

    left_half = expanded_per_xor[:4]
    right_half = expanded_per_xor[4:]

    # s0 and s1
    s0 = find_s0_s1(left_half,s0_seq)
    s1 = find_s0_s1(right_half,s1_seq)

    p4 = permute_and_generate(s0 + s1, p4_seq)

    return p4
```

In [22]:

```
def encrypt(ip, k1, k2):

    input_permutation = permute_and_generate(ip,ip_seq)

    input_permutation_left = input_permutation[:4]
    input_permutation_right = input_permutation[4:]

    # round 1
    r1_output = round_encrypt(input_permutation_right,k1)
    r1_output = find_xor(r1_output, input_permutation_left)

    # round 2
    r2_output = round_encrypt(r1_output,k2)
    r2_output = find_xor(r2_output, input_permutation_right)

    inv_ip = permute_and_generate(r2_output + r1_output, inv_ip_seq)

    return inv_ip
```

In [23]:

```
k1, k2 = generate_key("1010000010")
```

```
k1 : 10100100
k2 : 01000011
```

In [24]:

```
plaintext = "01100011"
ciphertext = encrypt(plaintext,k1,k2)

print("ciphertext : ", ciphertext)
```

```
ciphertext :  11101000
```

In [25]:

```
deciphered_text = encrypt(ciphertext,k2,k1)
print('deciphered_text : ', deciphered_text)
```

```
deciphered_text :  01100011
```

In [ ]: