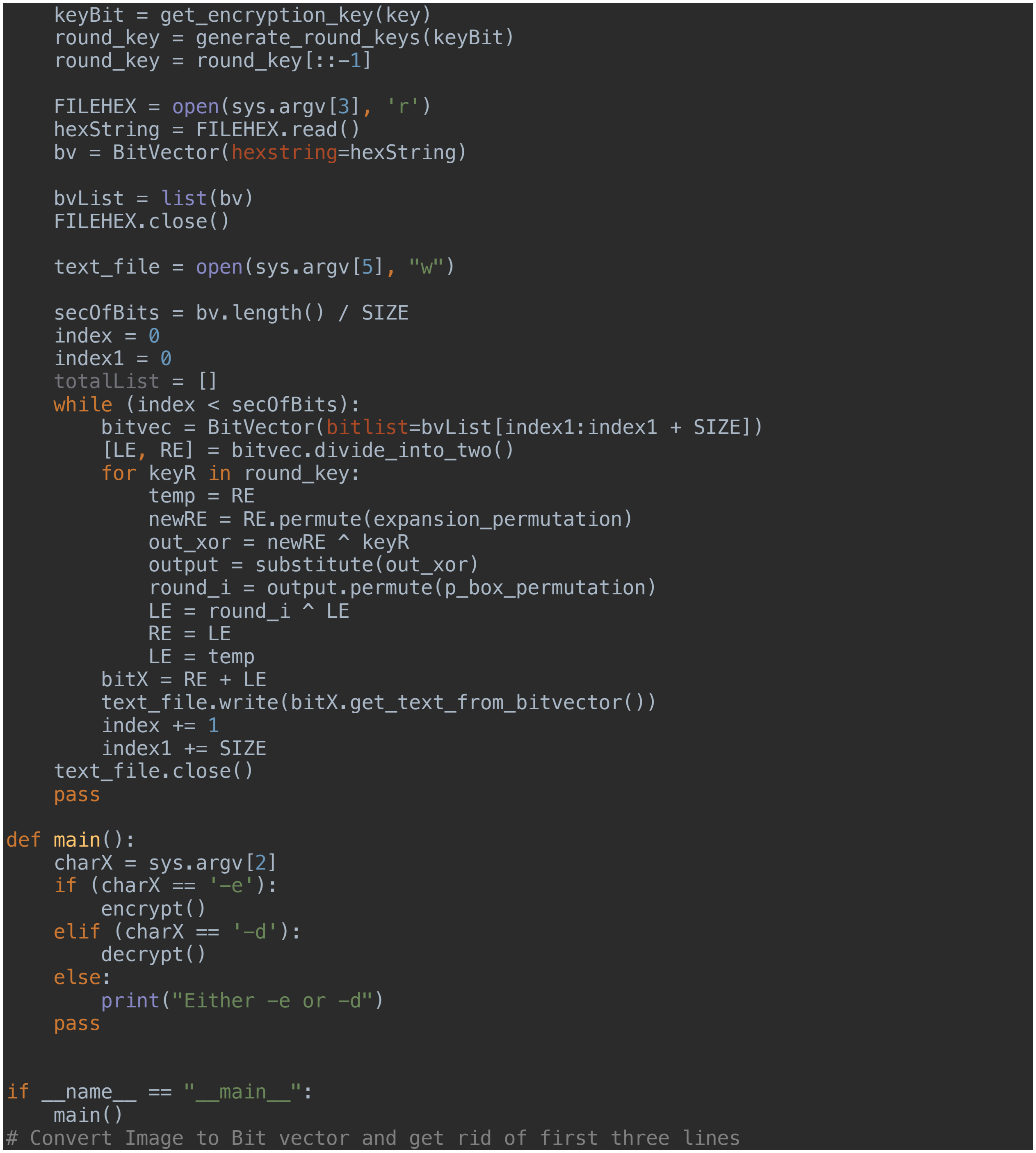
PROBLEM 1
Script

```
"""
Homework Number: 2
Name: Sneha Mahapatra
ECN Login: mahapat0
Due Date: 01/30/2020
"""
# !/usr/bin/env python
# !/usr/bin/env python -W ignore:tostring:DeprecationWarning
### DES_text.py
import codecs
import sys
import BitVector
# from get_encryption_key import *
from generate_round_keys import *
from illustrate_des_substitution import *
import warnings
warnings.simplefilter("ignore", DeprecationWarning)
expansion_permutation = [31, 0, 1, 2, 3, 4, 3, 4, 5, 6, 7, 8, 7, 8, 9, 10, 11, 12, 11,
12, 13, 14, 15, 16, 15, 16, 17,18, 19, 20, 19, 20, 21, 22, 23, 24, 23, 24, 25, 26, 27,
28, 27, 28, 29, 30, 31, 0]
p_box_permutation = [15, 6, 19, 20, 28, 11, 27, 16, 0, 14, 22, 25, 4, 17, 30, 9, 1, 7,
23, 13, 31, 26, 2, 8, 18, 12, 29, 5, 21, 10, 3, 24]
SIZE = 64

def encrypt():
    FILEREAD = open(sys.argv[4], 'r')
    key = FILEREAD.read()
    FILEREAD.close()
    keyBit = get_encryption_key(key)
    round_key = generate_round_keys(keyBit)
    bv = BitVector(filename=sys.argv[3])
    text_file = open(sys.argv[5], "w")
    while (bv.more_to_read):
        bitvec = bv.read_bits_from_file(SIZE)
        if (len(str(bitvec)) % SIZE != 0):
            x = bitvec.length() % SIZE
            bitvec.pad_from_left(SIZE-x)
        if (len(str(bitvec)) > 0):
            [LE, RE] = bitvec.divide_into_two()
            for keyR in round_key:
                temp = RE
                newRE = RE.permute(expansion_permutation)
                out_xor = newRE ^ keyR
                output = substitute(out_xor)
                round_i = output.permute(p_box_permutation)
                LE = LE ^ round_i
                RE = LE
                LE = temp
        bitX = RE + LE
        myhexstring = bitX.get_bitvector_in_hex()
        text_file.write(myhexstring)
    text_file.close()
    pass


def decrypt():
    FILEREAD = open(sys.argv[4], 'r')
    key = FILEREAD.read()
    FILEREAD.close()
```



```
    keyBit = get_encryption_key(key)
    round_key = generate_round_keys(keyBit)
    round_key = round_key[::-1]

    FILEHEX = open(sys.argv[3], 'r')
    hexString = FILEHEX.read()
    bv = BitVector(hexstring=hexString)

    bvList = list(bv)
    FILEHEX.close()

    text_file = open(sys.argv[5], "w")

    secOfBits = bv.length() / SIZE
    index = 0
    index1 = 0
    totalList = []
    while (index < secOfBits):
        bitvec = BitVector(bitlist=bvList[index1:index1 + SIZE])
        [LE, RE] = bitvec.divide_into_two()
        for keyR in round_key:
            temp = RE
            newRE = RE.permute(expansion_permutation)
            out_xor = newRE ^ keyR
            output = substitute(out_xor)
            round_i = output.permute(p_box_permutation)
            LE = round_i ^ LE
            RE = LE
            LE = temp
        bitX = RE + LE
        text_file.write(bitX.get_text_from_bitvector())
        index += 1
        index1 += SIZE
    text_file.close()
    pass

def main():
    charX = sys.argv[2]
    if (charX == '-e'):
        encrypt()
    elif (charX == '-d'):
        decrypt()
    else:
        print("Either -e or -d")
    pass


if __name__ == "__main__":
    main()
# Convert Image to Bit vector and get rid of first three lines
```

Explanation

Des uses the Fiestal Structure to encrypted their data. What's great about the Fiestal Structure is that every 64 bit section in the encrypt is the same as the reverse of decrypt. That's how I treated it when writing the code. So encrypt and decrypt are very similar I only had to change the way the file is read as one is a string and the other is hex.

Encrypted

```
605c6f3e13083a378764e40a8f2254f45b6ca29b034a7780ca45d40d67cc02bd44db1d8e453ceda55d9e54
65152afef9caeb8ec0f02d82bc7ffabfe89b887e4d60e21e9c9eecc280b91b4f7005743f09ca25bad6b3d5
208d5f20dea2715d10dec7d59e19e835d9edc78cb6086a7de91ca60d5fa49e79e8550b519e0b275243c311
346f917df2aff2c18680db97de8f64781405c1c6d57594ced10cec5c5f25533f96066cf395136779ad02c4
6a68de8866dc905616d46729cf82d3e402b7daf98adaa11e2bfa27785b774487e0d51205b74361d8330187
dd32a0d498c4743d023cbb6c8d18eae20dd1dc3ceb1c7477855d439e250bc8dc6fbb0c5af42ce813e47b8e
0daf5cbafa003e6609bf6ae29030381a819ee5dec49be9bb0ca9dafde688038f76f9ce344abbc269281db6
417db0c423e86aded601870c60fb93e1624b5b8d94df99ab41f61ca6e846f836a3e1261fcaa2febe41fc17
c459b83582f182ff9a65126a7a0dc7e2776aa23c20792057edbb681ed4e0e56c9e91cf9fc1b9b1266d66bc
30f968978041822c9b9c8ab919429881422f3c1556b2a16facdabb84677c9aadae181d83aeb66688ffb35d
d0dd14dbcabff8b9990375fea81b347d7318808a2f7231bcf90363a94c2e4a0a9133477336634c7a44e213
b2c6e86258509d6770ea58895bcf57f9e5ad412374897bd67d467d34fd077db85489bd996efcf0352af076
45b4614c7a41126731e4e8357f6c1a9f19d164683c84c9154bb6271438cf1ac748653d1c5f77bc336e2831
084b453ff68ac7c5870ea1f2994058b81e10f5699586c9718df402a1c6cc710a9a0591043525df23249aaa
3e9d599cb9a055ef7bfc360bc19a4baa9ec5f6c2117916669fab00c240e64ce100345f92618cf1b6f16f7b
76614dc26a70de7dea0f37426211591095b172cd327446424512353fb1960c67bfa5fe5cb543d7440cdfdf
1c92ebd7a6e4a14c7c9aadae181d83ae367c2d09fa57949ad3b80db0426c72a576239f51083965ca6770ea
58895bcf571ab1c366b6e2904911554eb2b508534f41b423a02c41c30f100fe12f65fcf3401fdf2946d24e
651446bfab2c48e5bf20b7e8431f1740031213b3aebc2cc8059f4dc37efaa9b16d065f653c52ef06ee72fb
61a8375b77209ac2236d345892a4aac212665ea415844122f22d777e02aa52e18d4416d7c33df7eaf44d6d
3cc43388b7bb16d8dd25bcfc78b5a5d82ec5657c5d6ff4025aef08b0285aa47b24eeb850f5dd6e02f1d3fe
73a960cd5afdcee7ac882ec8590551c3c016c3c51e9c9a6e7e045fddd184aa60ba16343fab24601f9b882e
c8590551c3c016c3c51e9c9a6e7ef8afba5eb270d8493b506939fb6f39170b22bf3dbe7f7e55297dc61b9e
c15b07abba294bbd23834802469307f609c9232529622488901efd608835825777cf05527faafff91f6550
ea299e9c005501361600c17b99e8d5134523fee0dd15b65cc157b0b48c6e166023ff42df2446af74f8d28d
235d94ba8fd25d09d33972eee1714a2e4a8e54310f9f14c918f60c717536a64cca35c181e82dff5a431d60
ad981b5f587b7b321527a5014fd5e8de2f04d713549f570efa46
```

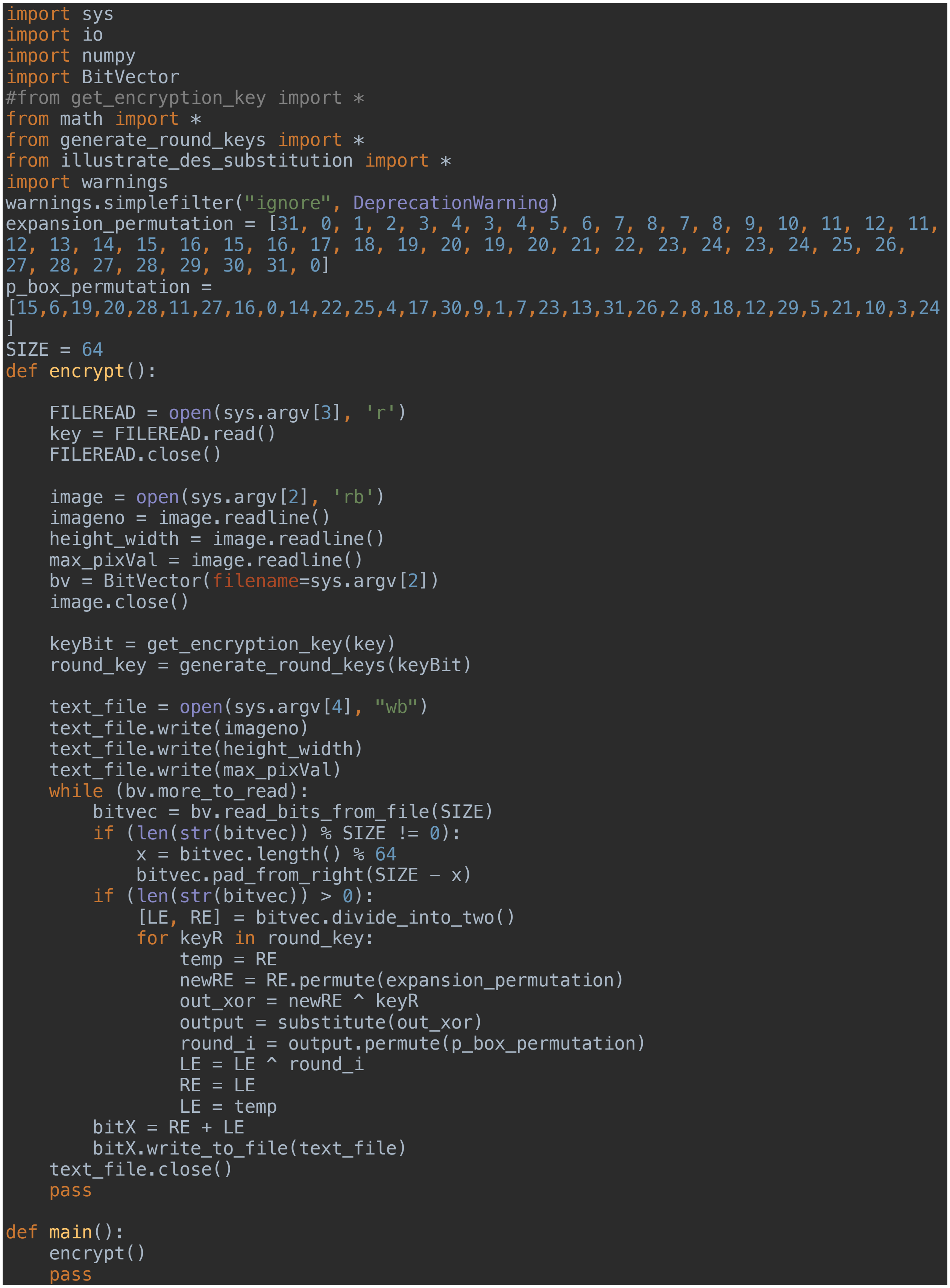
Decrypted

```
Earlier this week, security researchers took note of a series of changes Linux and
Windows developers began rolling out in beta updates to address a critical security
flaw: A bug in Intel chips allows low-privilege processes to access memory in the
computer's kernel, the machine's most privileged inner sanctum. Theoretical attacks
that exploit that bug, based on quirks in features Intel has implemented for faster
processing, could allow malicious software to spy deeply into other processes and data
on the target computer or smartphone. And on multi-user machines, like the servers run
by Google Cloud Services or Amazon Web Services, they could even allow hackers to
break out of one user's process, and instead snoop on other processes running on the
same shared server. On Wednesday evening, a large team of researchers at Google's
Project Zero, universities including the Graz University of Technology, the University
of Pennsylvania, the University of Adelaide in Australia, and security companies
including Cyberus and Rambus together released the full details of two attacks based
on that flaw, which they call Meltdown and Spectre.
```

PROBLEM 2

Script

```
"""
Homework Number: 2
Name: Sneha Mahapatra
ECN Login: mahapat0
Due Date: 01/30/2020
"""
#!/usr/bin/env python
#!/usr/bin/env python -W ignore::DeprecationWarning
### DES_text.py
import codecs
```



```
import sys
import io
import numpy
import BitVector
#from get_encryption_key import *
from math import *
from generate_round_keys import *
from illustrate_des_substitution import *
import warnings
warnings.simplefilter("ignore", DeprecationWarning)
expansion_permutation = [31, 0, 1, 2, 3, 4, 3, 4, 5, 6, 7, 8, 7, 8, 9, 10, 11, 12, 11,
12, 13, 14, 15, 16, 15, 16, 17, 18, 19, 20, 19, 20, 21, 22, 23, 24, 23, 24, 25, 26,
27, 28, 27, 28, 29, 30, 31, 0]
p_box_permutation =
[15,6,19,20,28,11,27,16,0,14,22,25,4,17,30,9,1,7,23,13,31,26,2,8,18,12,29,5,21,10,3,24
]
SIZE = 64
def encrypt():

    FILEREAD = open(sys.argv[3], 'r')
    key = FILEREAD.read()
    FILEREAD.close()

    image = open(sys.argv[2], 'rb')
    imageno = image.readline()
    height_width = image.readline()
    max_pixVal = image.readline()
    bv = BitVector(filename=sys.argv[2])
    image.close()

    keyBit = get_encryption_key(key)
    round_key = generate_round_keys(keyBit)

    text_file = open(sys.argv[4], "wb")
    text_file.write(imageno)
    text_file.write(height_width)
    text_file.write(max_pixVal)
    while (bv.more_to_read):
        bitvec = bv.read_bits_from_file(SIZE)
        if (len(str(bitvec)) % SIZE != 0):
            x = bitvec.length() % 64
            bitvec.pad_from_right(SIZE - x)
        if (len(str(bitvec)) > 0):
            [LE, RE] = bitvec.divide_into_two()
            for keyR in round_key:
                temp = RE
                newRE = RE.permute(expansion_permutation)
                out_xor = newRE ^ keyR
                output = substitute(out_xor)
                round_i = output.permute(p_box_permutation)
                LE = LE ^ round_i
                RE = LE
                LE = temp
        bitX = RE + LE
        bitX.write_to_file(text_file)
    text_file.close()
    pass

def main():
    encrypt()
    pass
```

```
if __name__ == "__main__":
    main()
#Convert Image to Bit vector and get rid of first three lines
```

Explanation

I used the same enycrpted function, only this time I treated the file as a binary. The file given was a pbm format and I had to make sure to deal with the three line header and than the body.

Image