JPEG Image Compression

With Python

Import

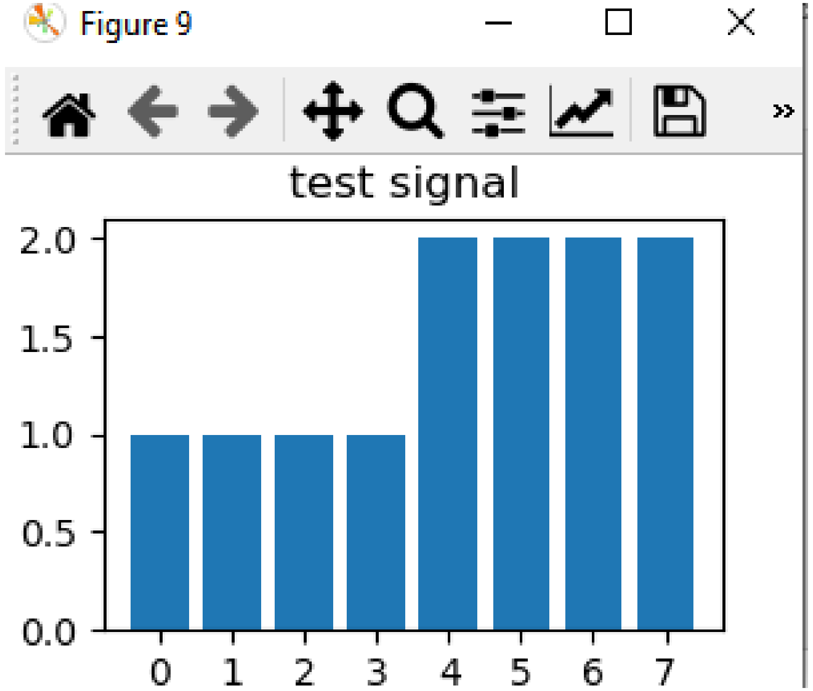
lec_code0_jpeg.py

```
import numpy as np
from numpy import pi
from numpy import cos
from numpy import zeros
from numpy import r_
import matplotlib.pylab as pylab
from scipy.fftpack import dct, idct
```

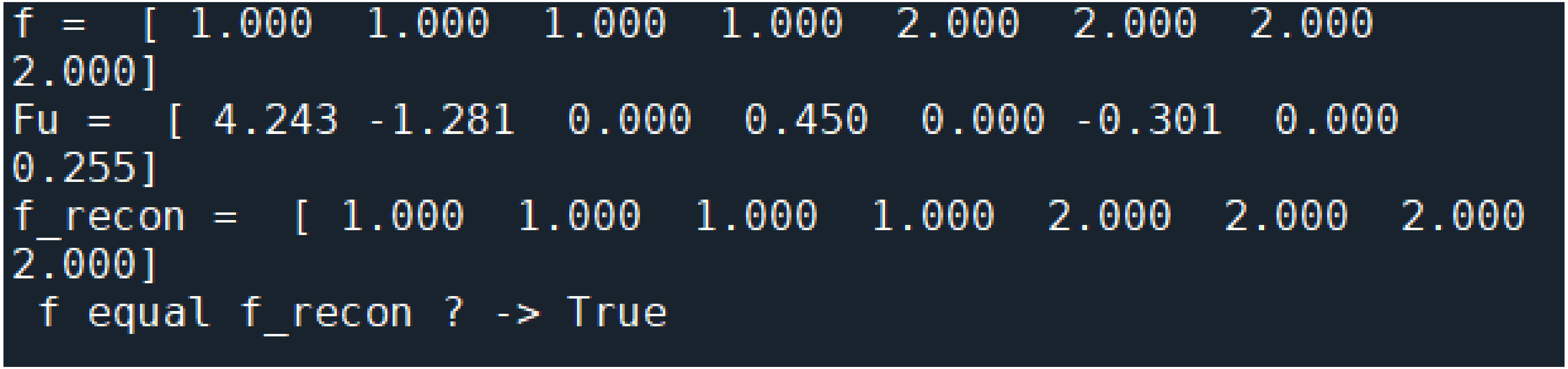
Using Scipy library

Lossless transformation

```
#%% Use DCT transform from the scipy library
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
# numpy array
f = np.array([1,1,1,1,2,2,2,2], dtype='float32')
print("x = ",f)

# apply dct function on array
F = dct(f, norm = 'ortho')
print("Fu = ",F)
f_recon = idct(F, norm = 'ortho')
print("f_recon = ",f_recon)
```

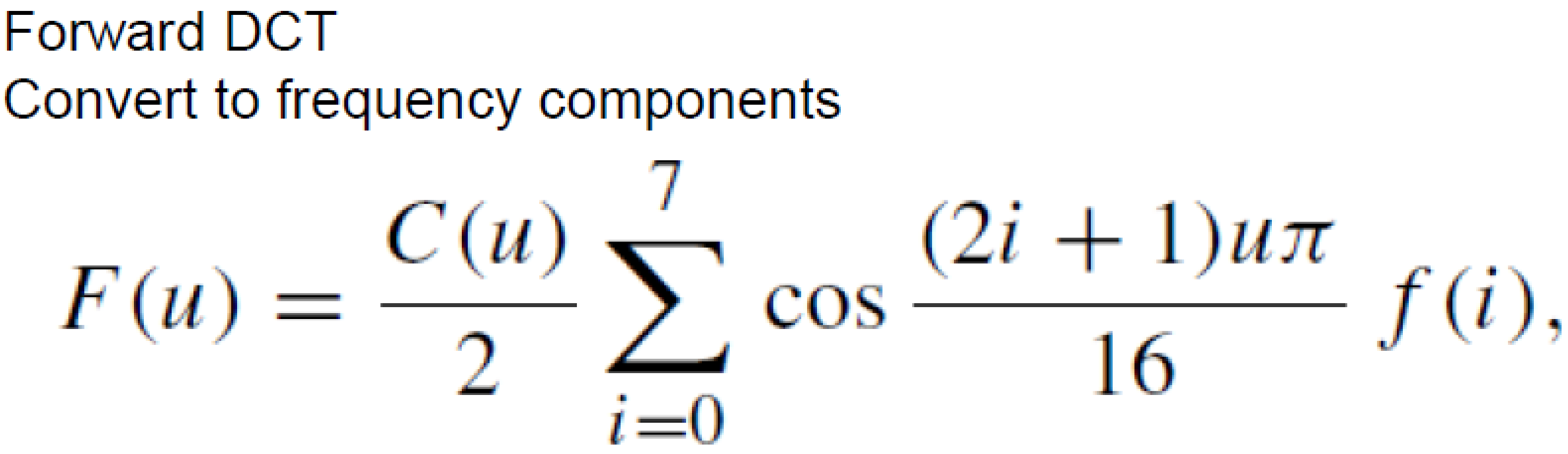


Lossless transformation



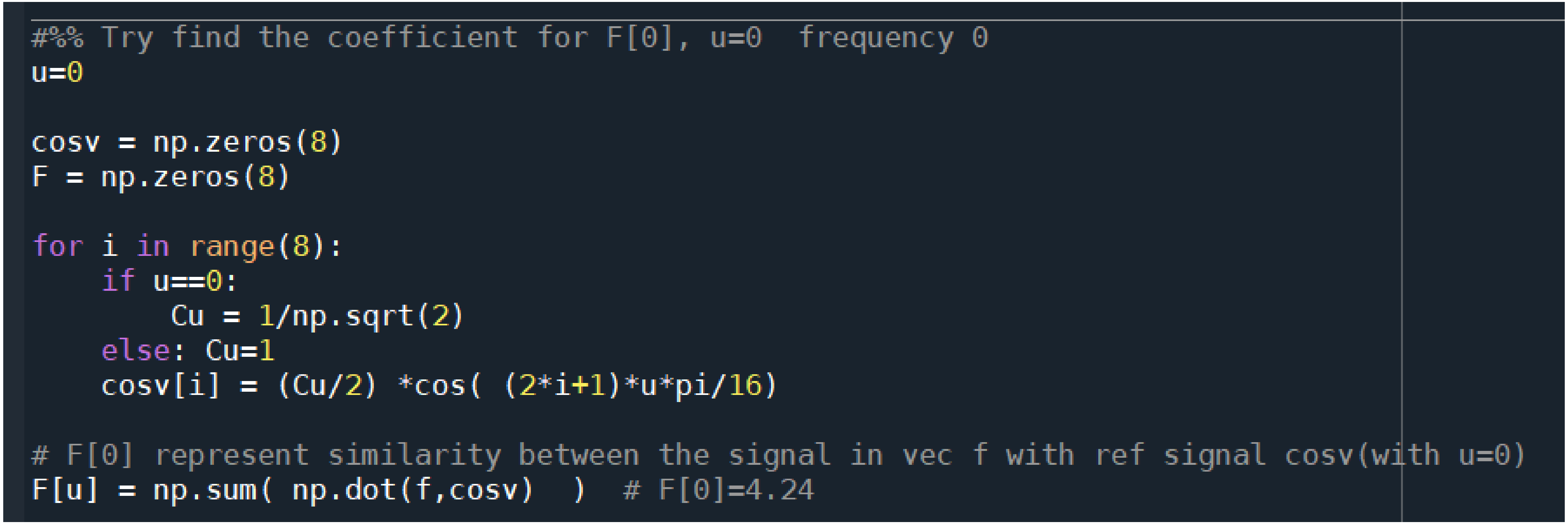
```
f =  [ 1.000  1.000  1.000  1.000  2.000  2.000  2.000
2.000]
Fu =  [ 4.243 -1.281  0.000  0.450  0.000 -0.301  0.000
0.255]
f_recon =  [ 1.000  1.000  1.000  1.000  2.000  2.000  2.000
2.000]
 f equal f_recon ? -> True
```

Try implement this in Python



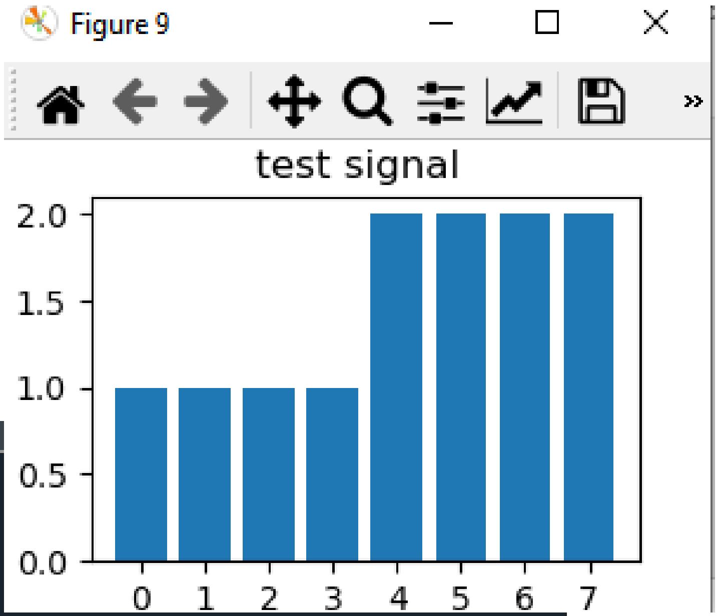
Forward DCT
Convert to frequency components

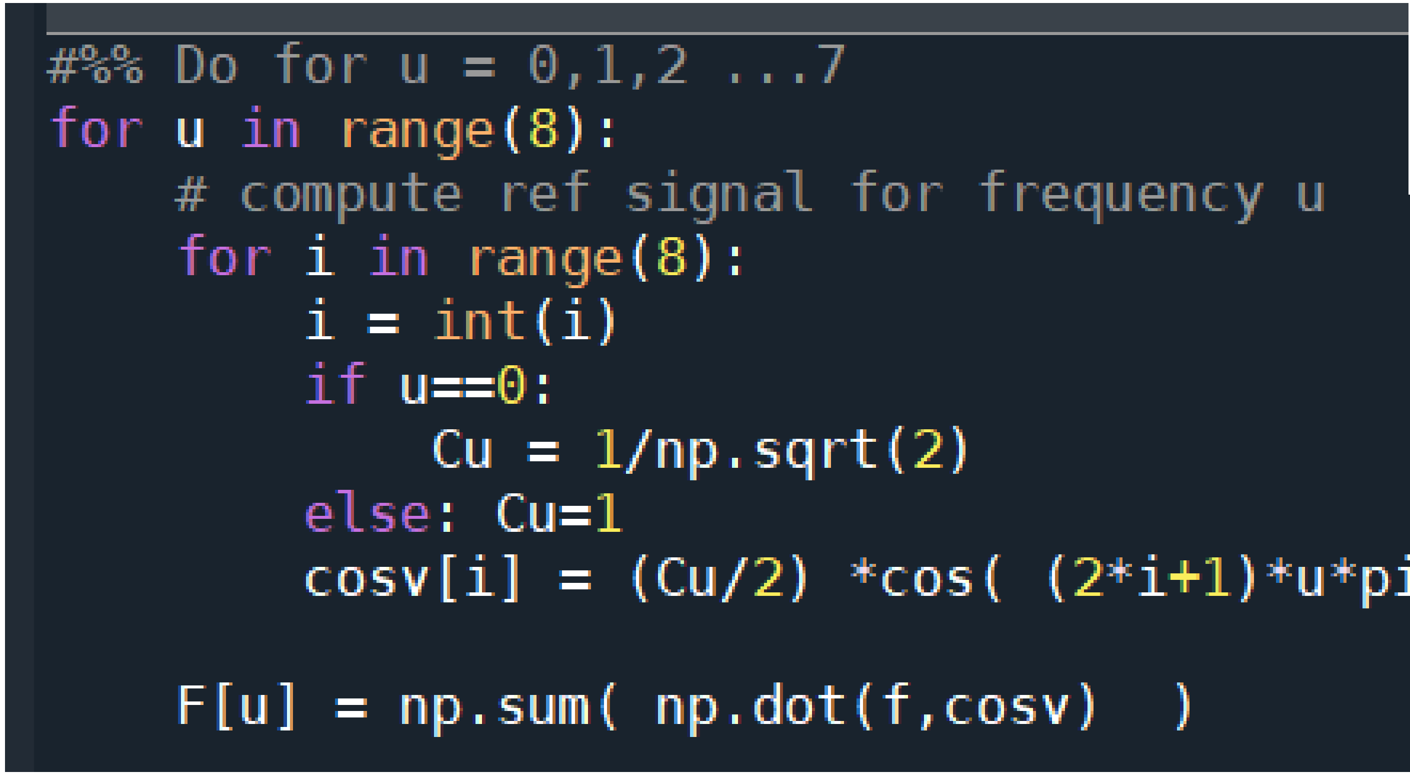
$$F(u) = \frac{C(u)}{2} \sum_{i=0}^{7} \cos \frac{(2i+1)u\pi}{16} f(i),$$



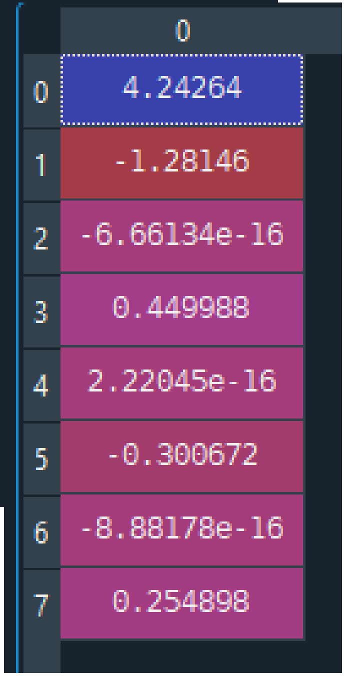
```
#%% Try find the coefficient for F[0], u=0  frequency 0
u=0

cosv = np.zeros(8)
F = np.zeros(8)

for i in range(8):
    if u==0:
        Cu = 1/np.sqrt(2)
    else: Cu=1
    cosv[i] = (Cu/2) *cos( (2*i+1)*u*pi/16)

# F[0] represent similarity between the signal in vec f with ref signal cosv(with u=0)
F[u] = np.sum( np.dot(f,cosv)  )  # F[0]=4.24
```

1D DCT Function





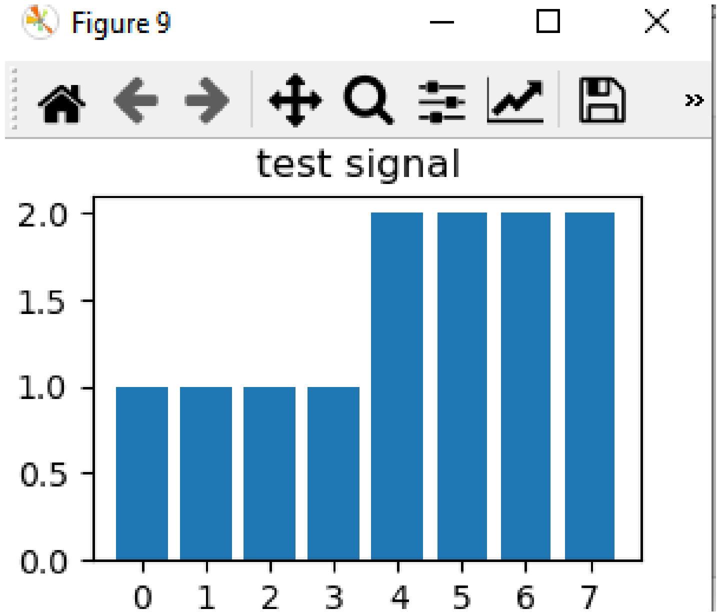
```
#%% Do for u = 0,1,2 ...7
for u in range(8):
    # compute ref signal for frequency u
    for i in range(8):
        i = int(i)
        if u==0:
            Cu = 1/np.sqrt(2)
        else: Cu=1
        cosv[i] = (Cu/2) *cos( (2*i+1)*u*pi/16)

    F[u] = np.sum( np.dot(f,cosv)  )
```

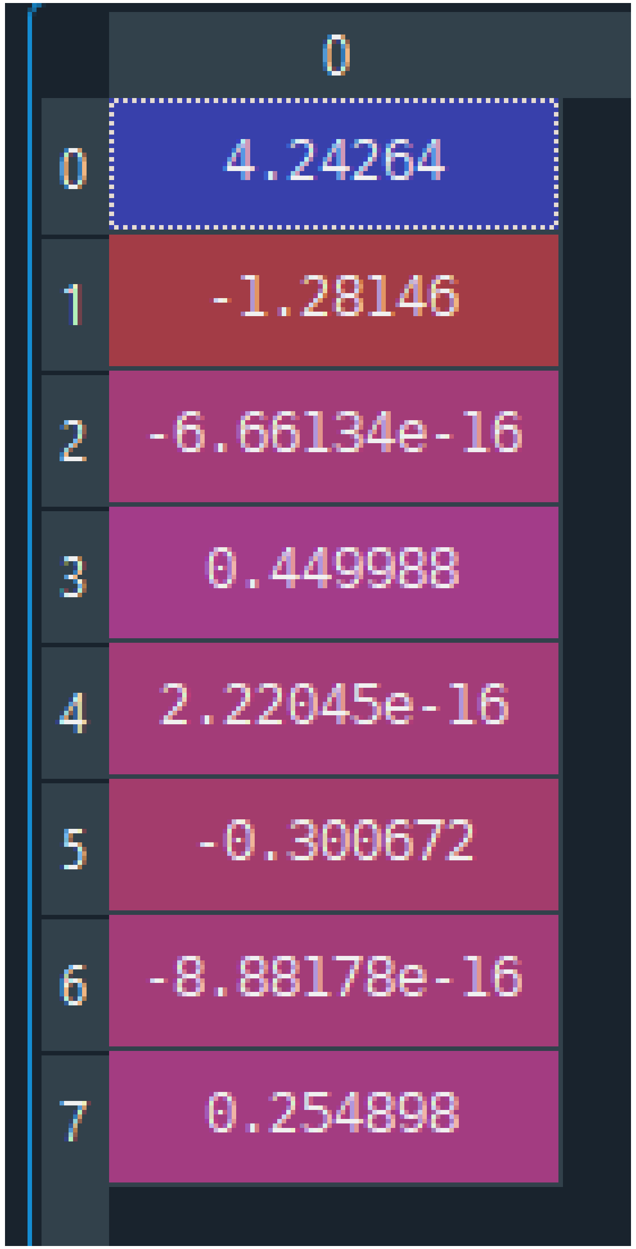


	0
0	4.24264
1	-1.28146
2	-6.66134e-16
3	0.449988
4	2.22045e-16
5	-0.300672
6	-8.88178e-16
7	0.254898

1D DCT Function

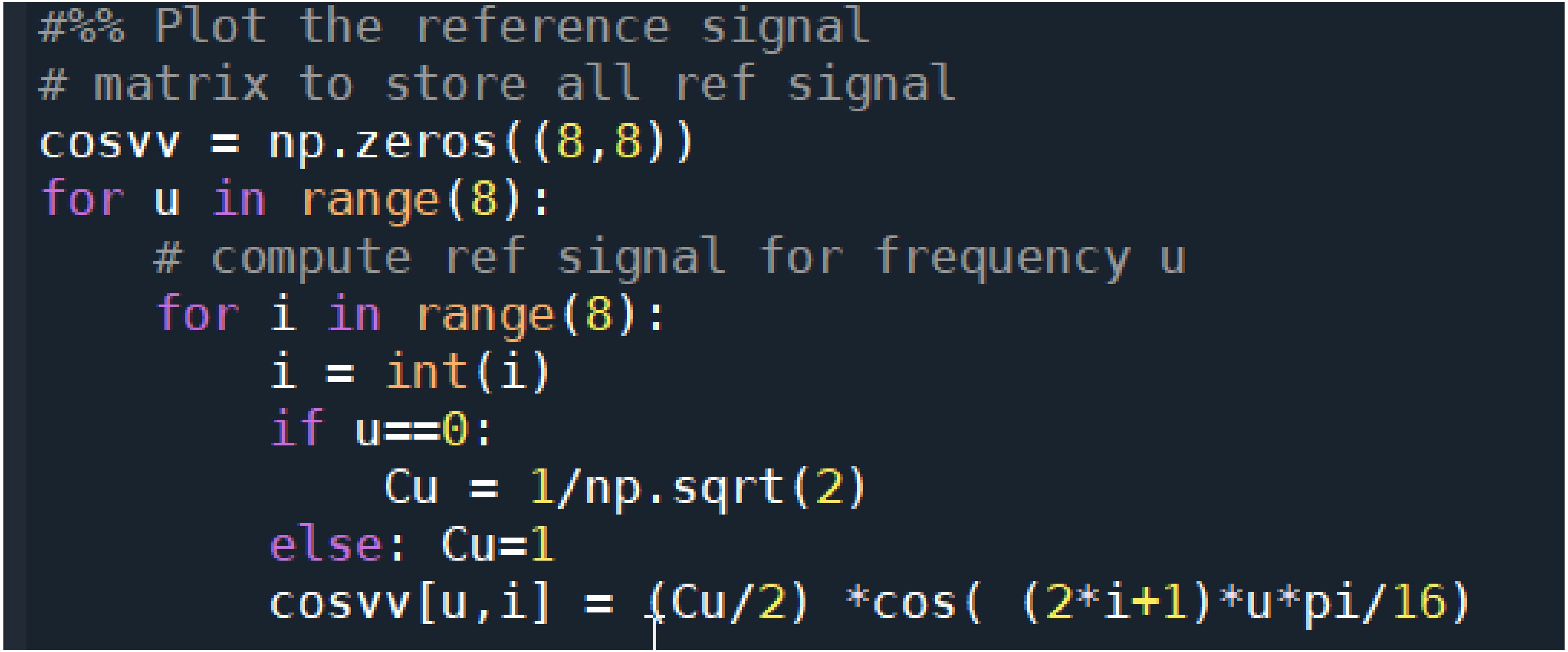
- What is the dominant frequency ?
- Hint: Which coefficient has strongest value ?



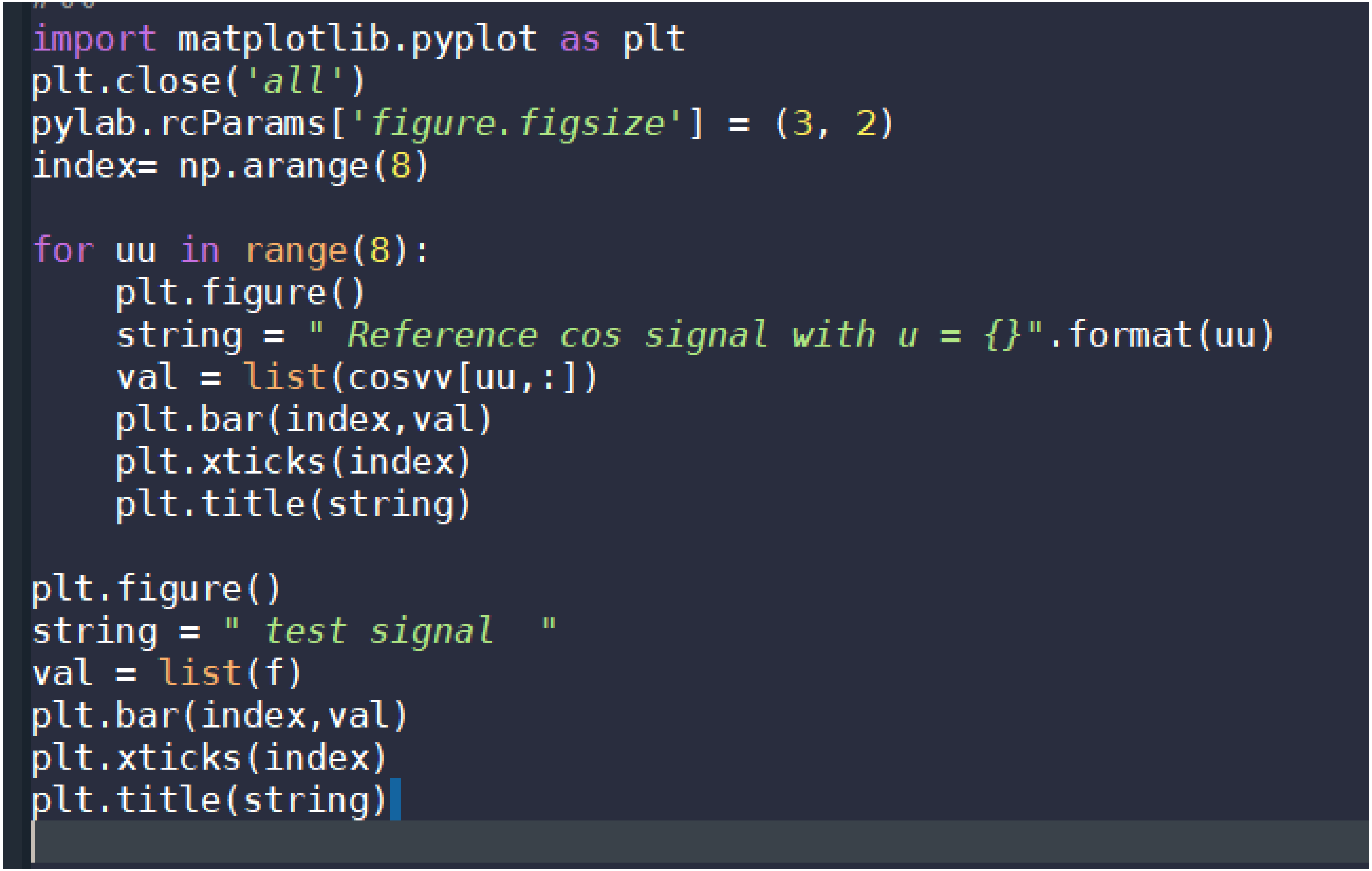


	0
0	4.24264
1	-1.28146
2	-6.66134e-16
3	0.449988
4	2.22045e-16
5	-0.300672
6	-8.88178e-16
7	0.254898

Visualize Reference Signal for 8 Point DCT



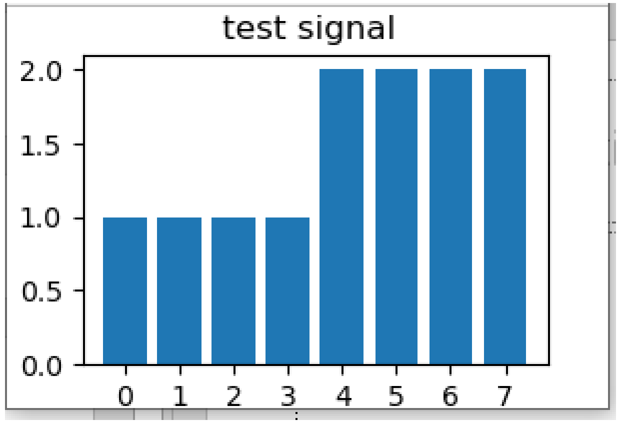
```
#%% Plot the reference signal
# matrix to store all ref signal
cosvv = np.zeros((8,8))
for u in range(8):
    # compute ref signal for frequency u
    for i in range(8):
        i = int(i)
        if u==0:
            Cu = 1/np.sqrt(2)
        else: Cu=1
        cosvv[u,i] = (Cu/2) *cos( (2*i+1)*u*pi/16)
```



```
import matplotlib.pyplot as plt
plt.close('all')
pylab.rcParams['figure.figsize'] = (3, 2)
index= np.arange(8)

for uu in range(8):
    plt.figure()
    string = " Reference cos signal with u = {}".format(uu)
    val = list(cosvv[uu,:])
    plt.bar(index,val)
    plt.xticks(index)
    plt.title(string)

plt.figure()
string = " test signal  "
val = list(f)
plt.bar(index,val)
plt.xticks(index)
plt.title(string)
```

1D DCT Coeffcients Represent the Similarity Between Test Signal f(i) and Reference Cos signal with frequency u

- F[7] shows similarity between test signal and cos signal with frequency u=7



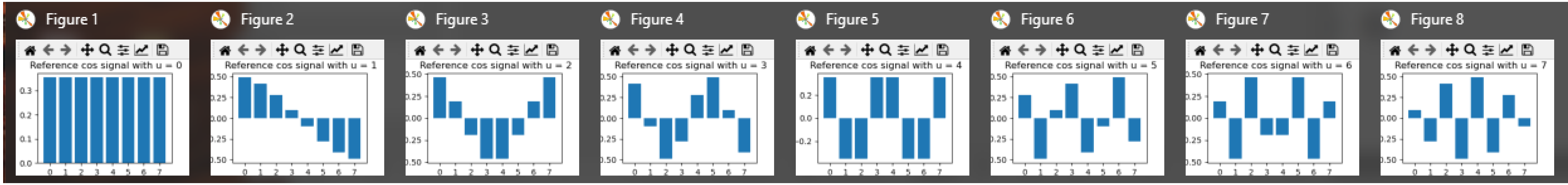


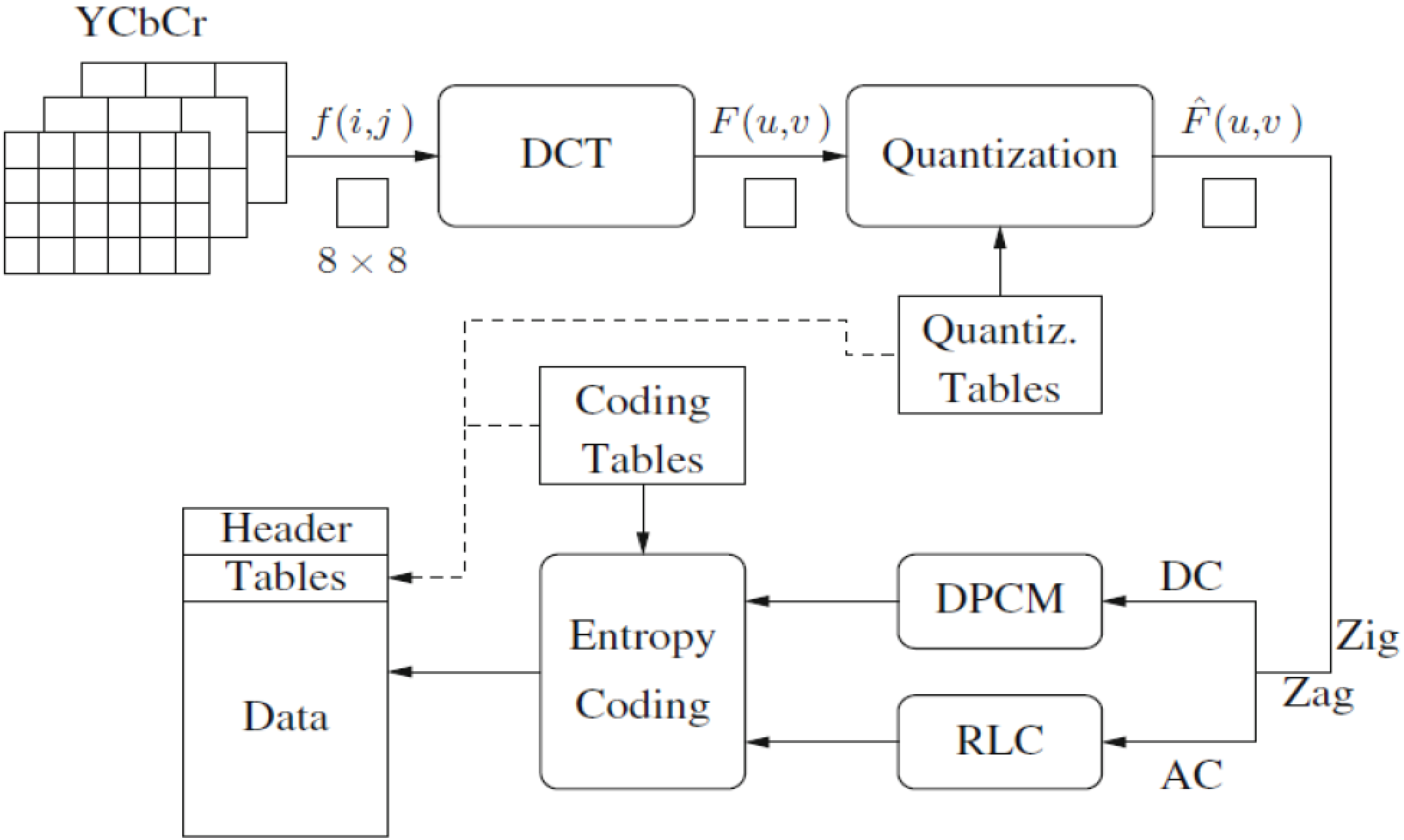
Figure 1
Reference cos signal with u = 0
Figure 2
Reference cos signal with u = 1
Figure 3
Reference cos signal with u = 2
Figure 4
Reference cos signal with u = 3
Figure 5
Reference cos signal with u = 4
Figure 6
Reference cos signal with u = 5
Figure 7
Reference cos signal with u = 6
Figure 8
Reference cos signal with u = 7
0.3
0.2
0.1
0.0
0.50
0.25
0.00
-0.25
-0.50
0.2
0.0
-0.2
0
1
2
3
4
5
6
7

F[0]

F[1]

F[7]

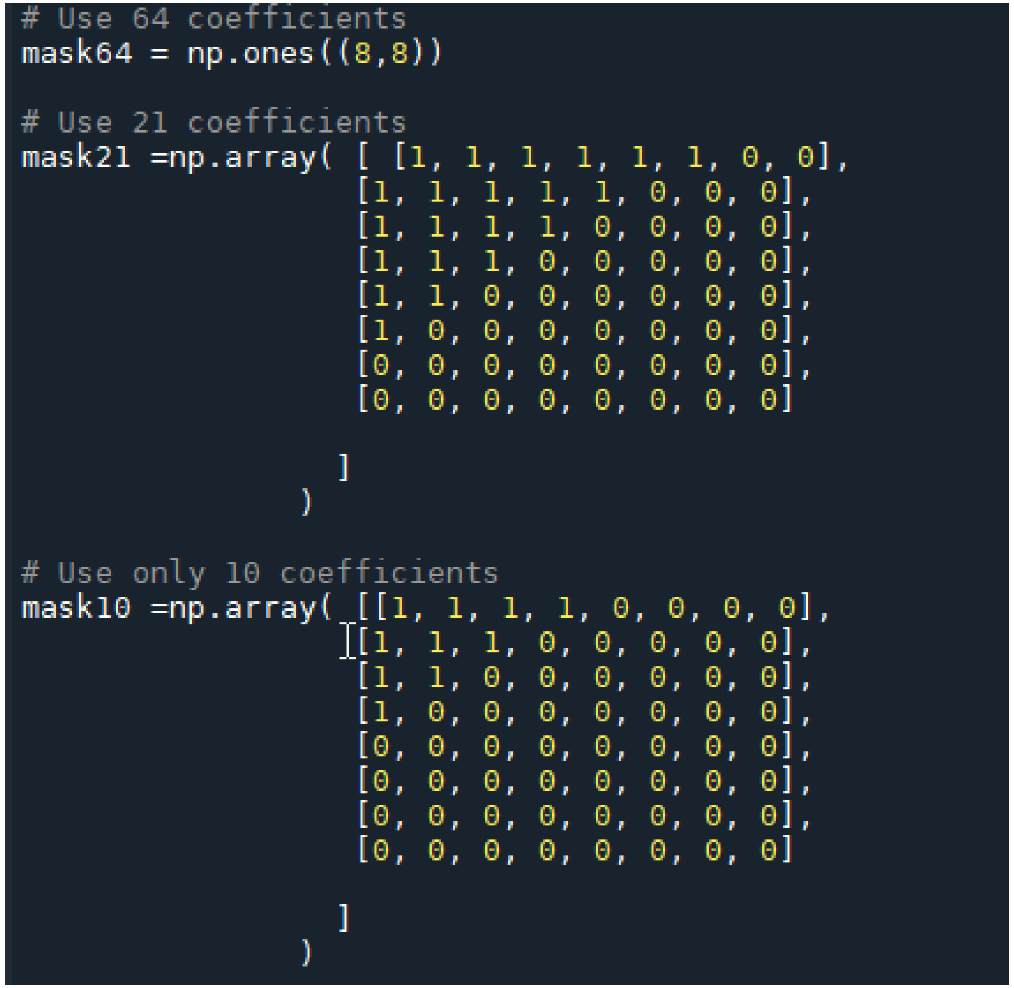
JPEG Encoder (Perform Compression)



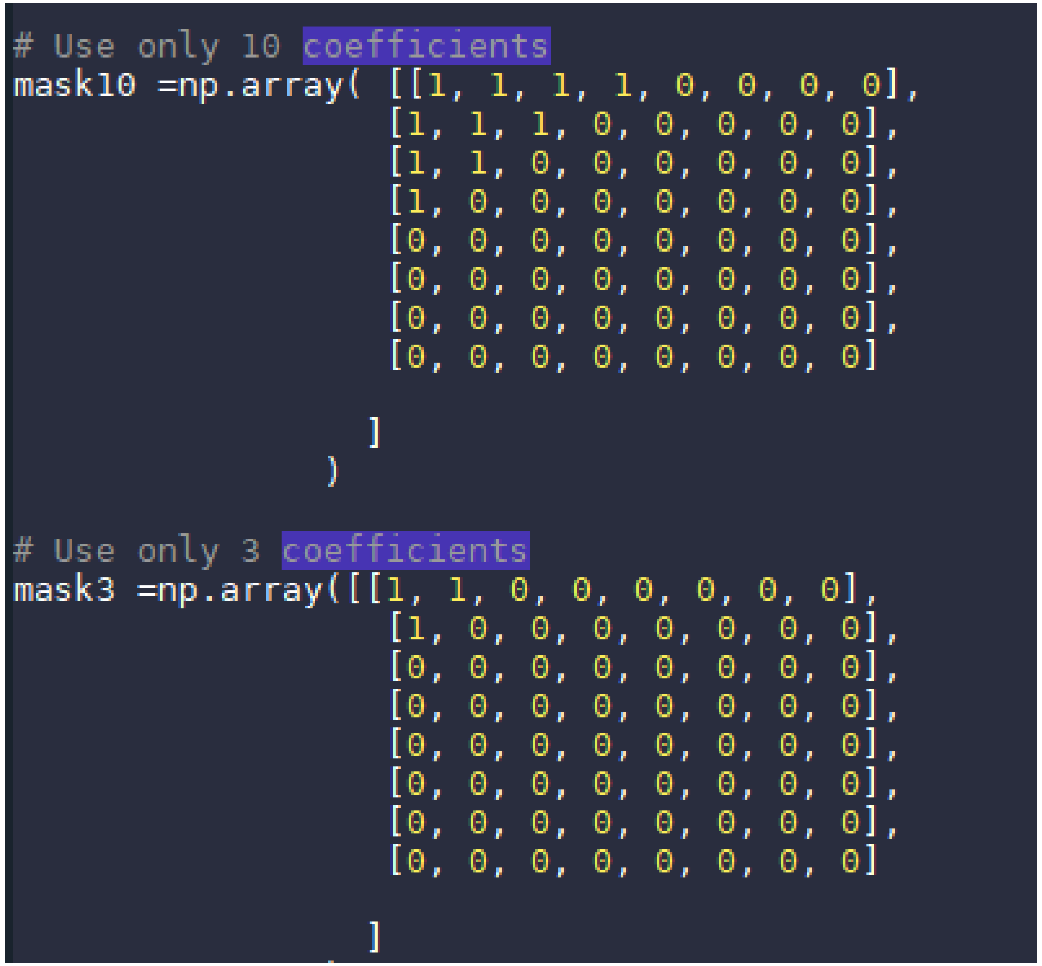
Examine Impact of Quantization of DCT Coefficients

- lec_code1_jpeg.py

Which mask give better quality?

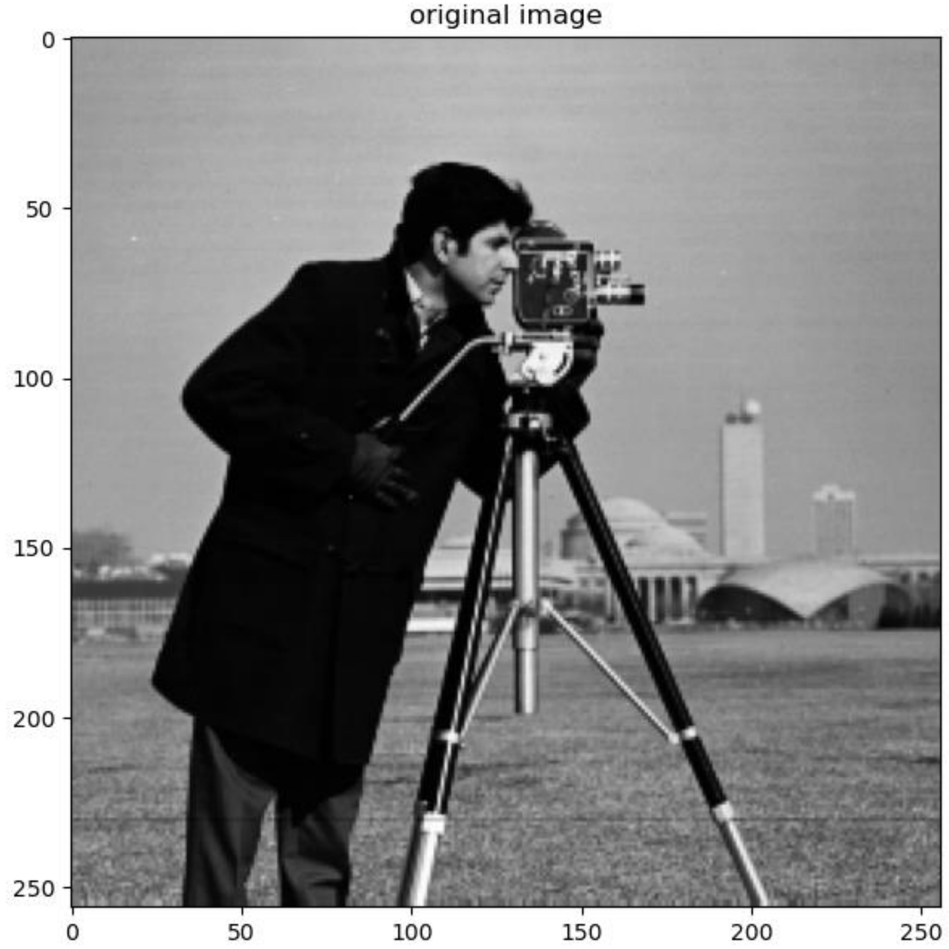


```
# Use 64 coefficients
mask64 = np.ones((8,8))

# Use 21 coefficients
mask21 =np.array( [ [1, 1, 1, 1, 1, 1, 0, 0],
                  [1, 1, 1, 1, 1, 0, 0, 0],
                  [1, 1, 1, 1, 0, 0, 0, 0],
                  [1, 1, 1, 0, 0, 0, 0, 0],
                  [1, 1, 0, 0, 0, 0, 0, 0],
                  [1, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0]

                 ]
               )

# Use only 10 coefficients
mask10 =np.array( [[1, 1, 1, 1, 0, 0, 0, 0],
                  [1, 1, 1, 0, 0, 0, 0, 0],
                  [1, 1, 0, 0, 0, 0, 0, 0],
                  [1, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0]

                 ]
               )
```

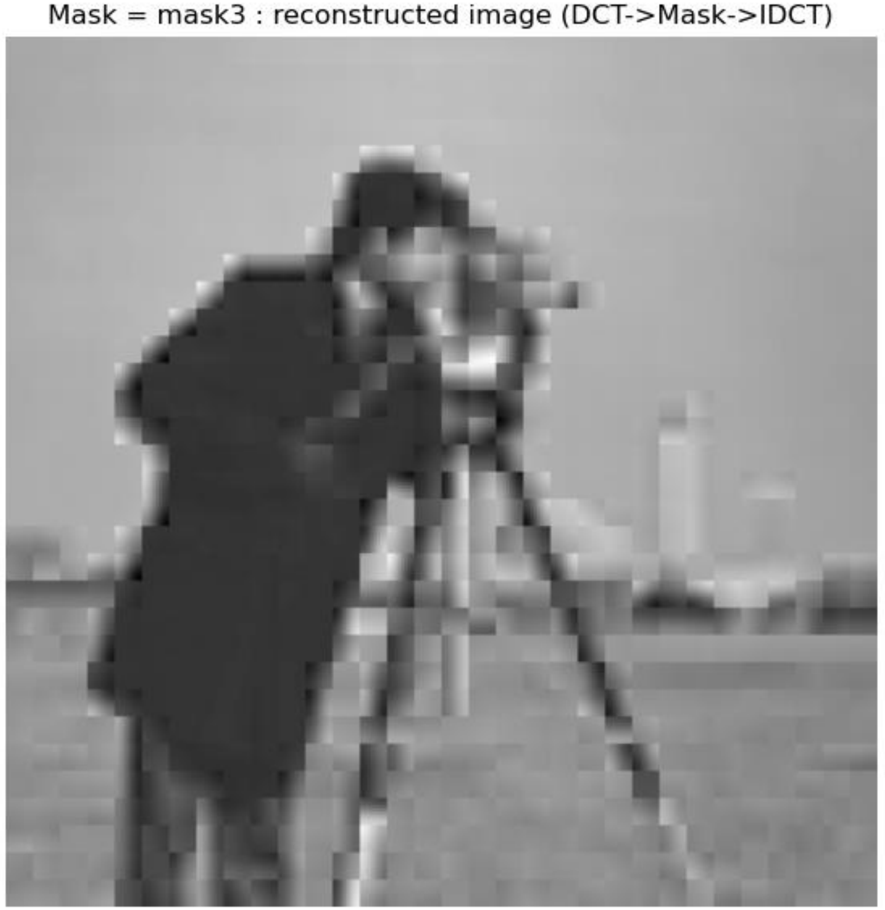


```
# Use only 10 coefficients
mask10 =np.array( [[1, 1, 1, 1, 0, 0, 0, 0],
                  [1, 1, 1, 0, 0, 0, 0, 0],
                  [1, 1, 0, 0, 0, 0, 0, 0],
                  [1, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0]

                 ]
               )

# Use only 3 coefficients
mask3 =np.array([[1, 1, 0, 0, 0, 0, 0, 0],
                  [1, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0]

                 ]
```

Using Mask3 (Preserve 3 DCT coefficients per Block)



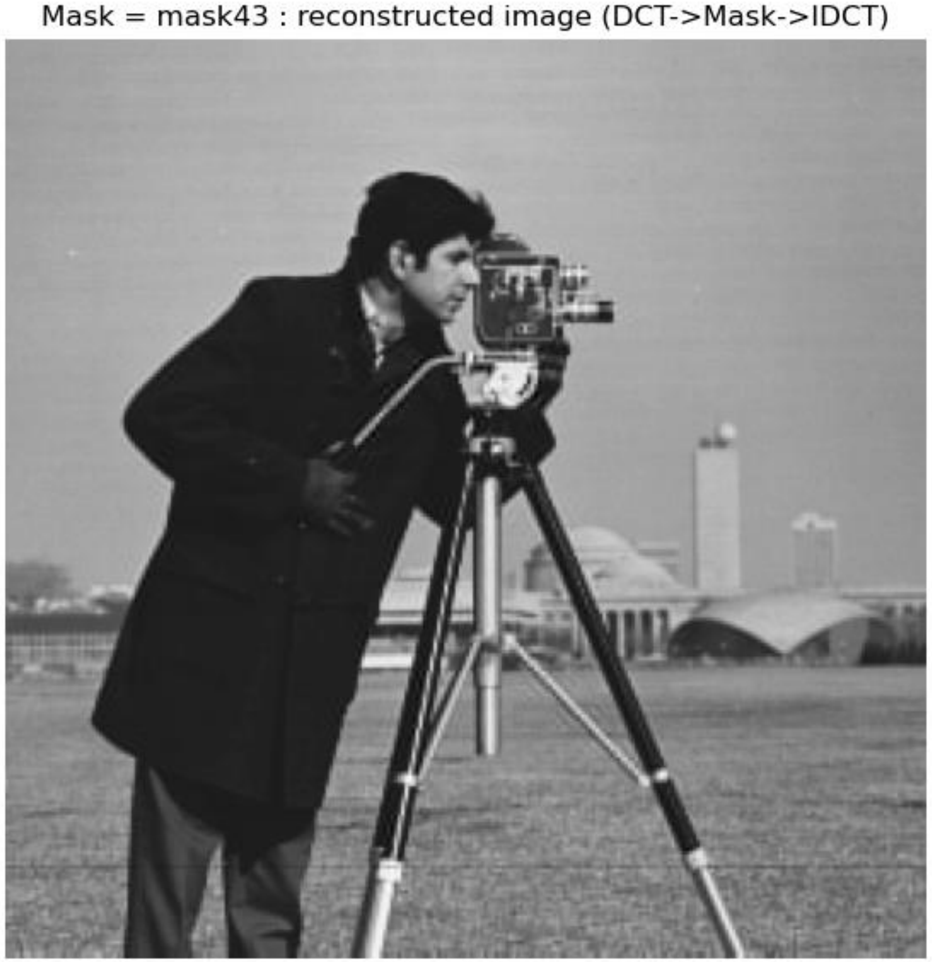
original image



Mask = mask3 : reconstructed image (DCT->Mask->IDCT)

Using 43 Coefficients

original image



Mask = mask43 : reconstructed image (DCT->Mask->IDCT)

How JPEG achieve compression

- By preserving only the important DCT coefficient
  - Data size is reduced and preserve perceptual quality as well