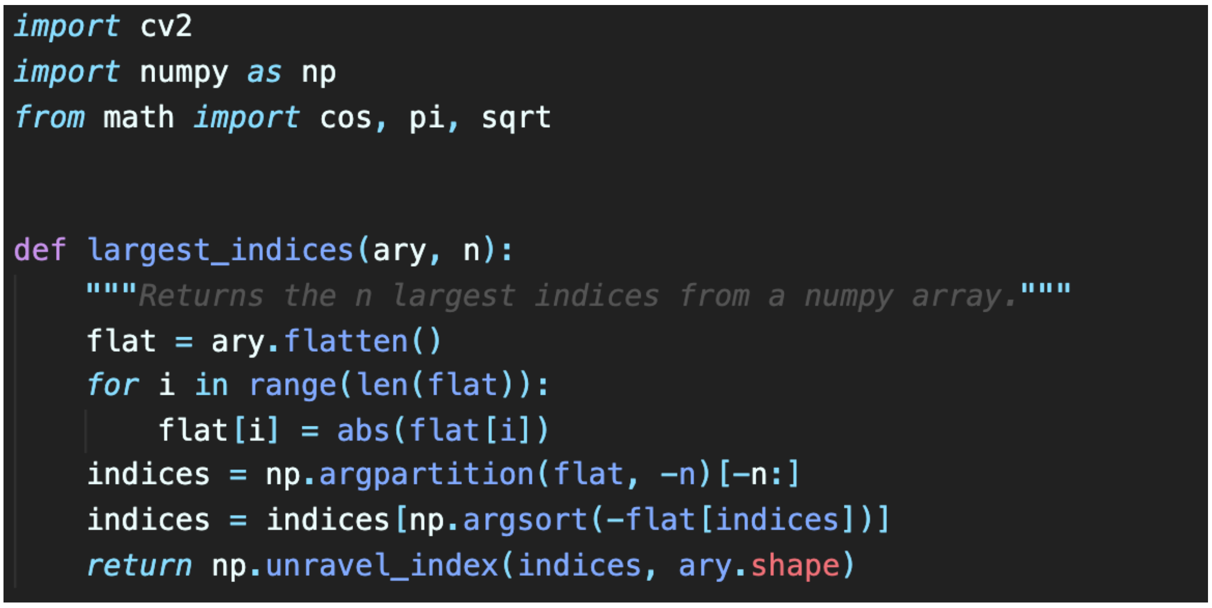
HW9 : DCT

2017029970 우원진

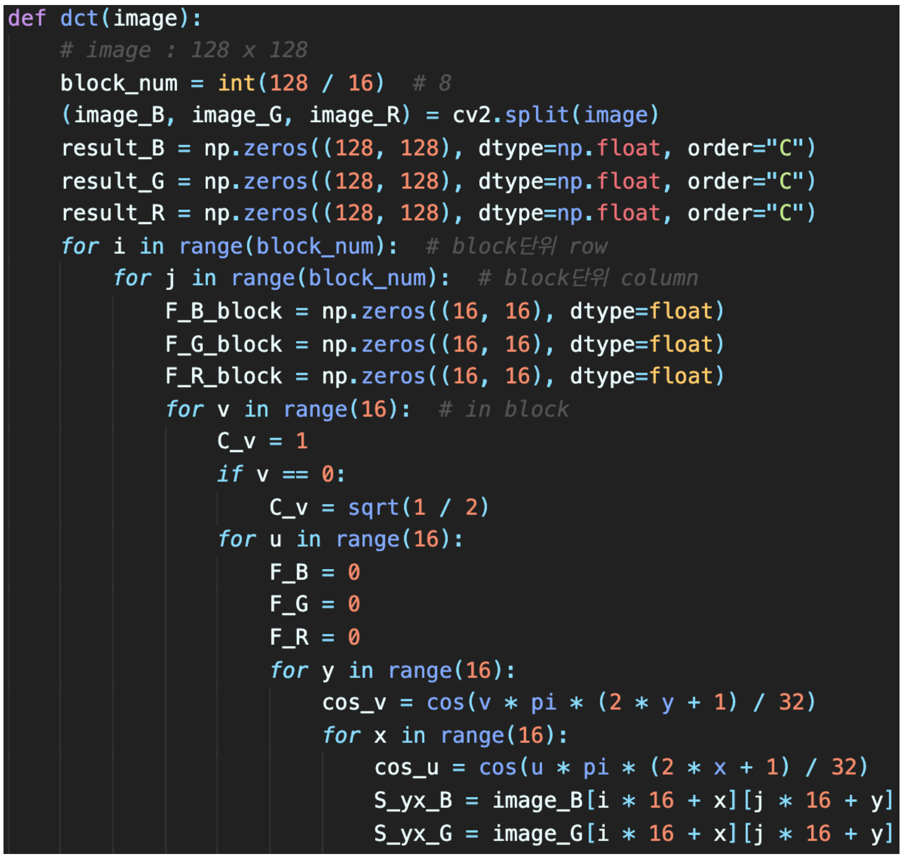
코드



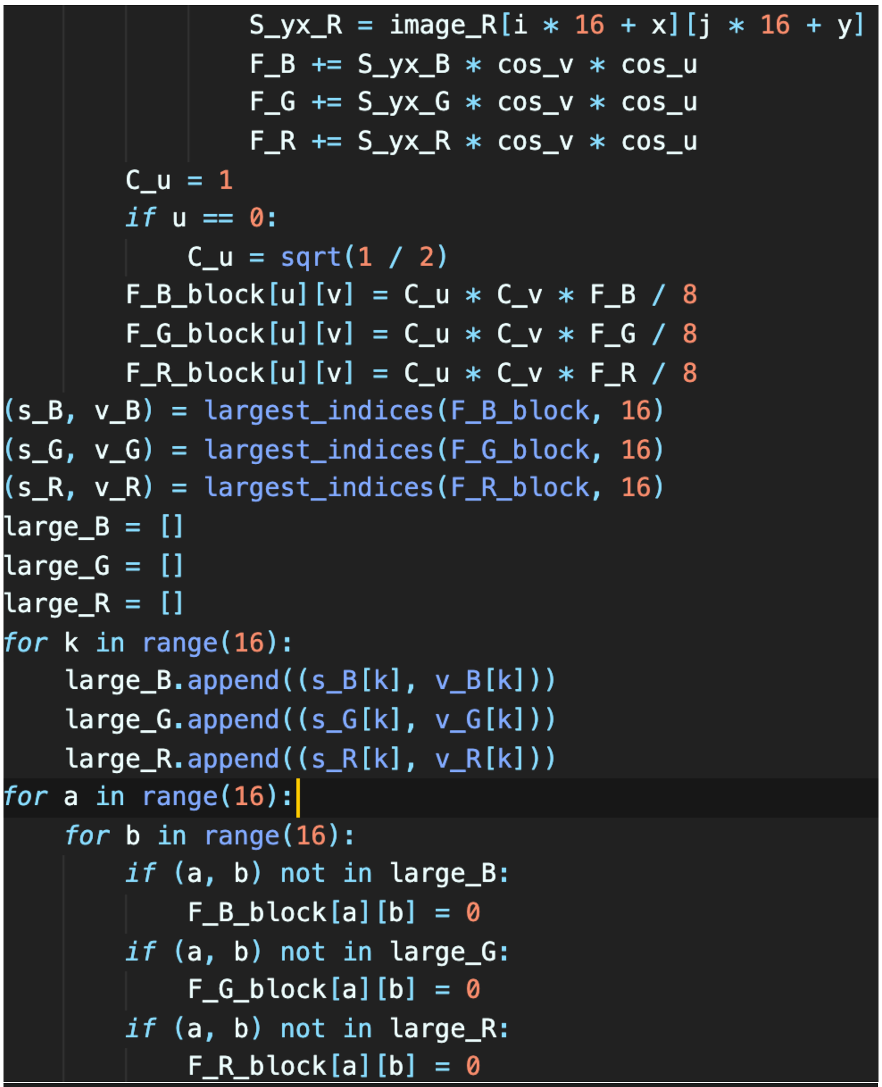
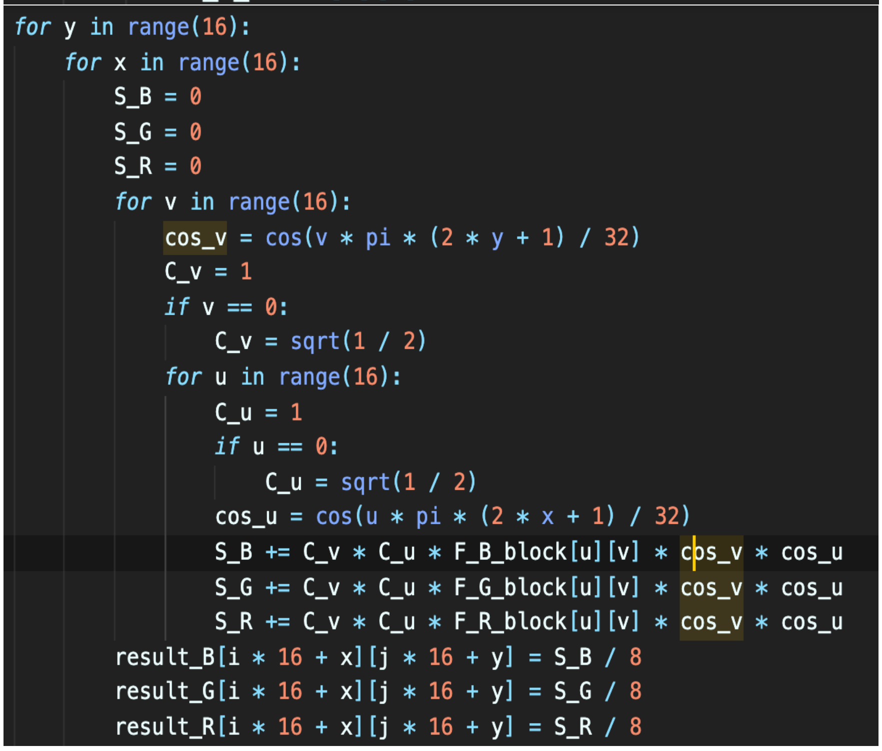
```python
import cv2
import numpy as np
from math import cos, pi, sqrt


def largest_indices(ary, n):
    """Returns the n largest indices from a numpy array."""
    flat = ary.flatten()
    for i in range(len(flat)):
        flat[i] = abs(flat[i])
    indices = np.argpartition(flat, -n)[-n:]
    indices = indices[np.argsort(-flat[indices])]
    return np.unravel_index(indices, ary.shape)
```

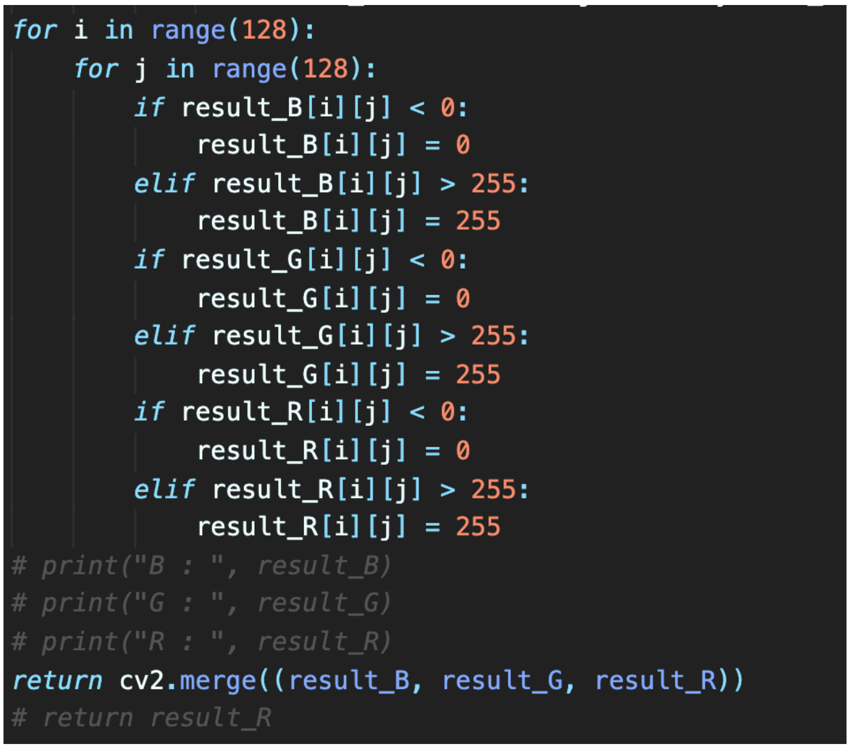
Fu,v를 구하고 그중 coefficient값이 큰 16개를 찾고 나머지는 0으로 만들기 위해 큰값 16개에 대한 index를 return 해주는 함수이다.

DCT 함수



```python
def dct(image):
    # image : 128 x 128
    block_num = int(128 / 16)  # 8
    (image_B, image_G, image_R) = cv2.split(image)
    result_B = np.zeros((128, 128), dtype=np.float, order="C")
    result_G = np.zeros((128, 128), dtype=np.float, order="C")
    result_R = np.zeros((128, 128), dtype=np.float, order="C")
    for i in range(block_num):  # block단위 row
        for j in range(block_num):  # block단위 column
            F_B_block = np.zeros((16, 16), dtype=float)
            F_G_block = np.zeros((16, 16), dtype=float)
            F_R_block = np.zeros((16, 16), dtype=float)
            for v in range(16):  # in block
                C_v = 1
                if v == 0:
                    C_v = sqrt(1 / 2)
                for u in range(16):
                    F_B = 0
                    F_G = 0
                    F_R = 0
                    for y in range(16):
                        cos_v = cos(v * pi * (2 * y + 1) / 32)
                        for x in range(16):
                            cos_u = cos(u * pi * (2 * x + 1) / 32)
                            S_yx_B = image_B[i * 16 + x][j * 16 + y]
                            S_yx_G = image_G[i * 16 + x][j * 16 + y]
```

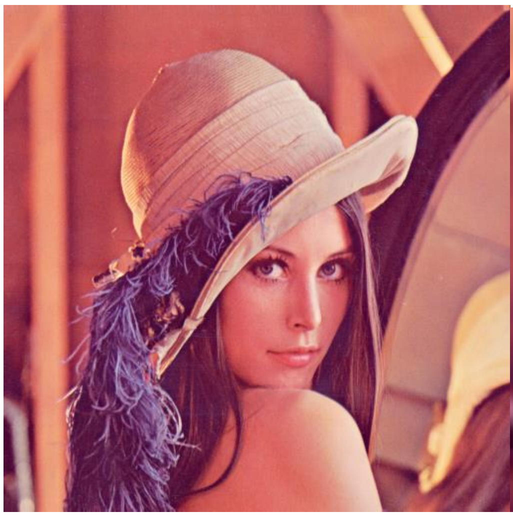

```
                S_yx_R = image_R[i * 16 + x][j * 16 + y]
                F_B += S_yx_B * cos_v * cos_u
                F_G += S_yx_G * cos_v * cos_u
                F_R += S_yx_R * cos_v * cos_u
        C_u = 1
        if u == 0:
            C_u = sqrt(1 / 2)
        F_B_block[u][v] = C_u * C_v * F_B / 8
        F_G_block[u][v] = C_u * C_v * F_G / 8
        F_R_block[u][v] = C_u * C_v * F_R / 8
(s_B, v_B) = largest_indices(F_B_block, 16)
(s_G, v_G) = largest_indices(F_G_block, 16)
(s_R, v_R) = largest_indices(F_R_block, 16)
large_B = []
large_G = []
large_R = []
for k in range(16):
    large_B.append((s_B[k], v_B[k]))
    large_G.append((s_G[k], v_G[k]))
    large_R.append((s_R[k], v_R[k]))
for a in range(16):
    for b in range(16):
        if (a, b) not in large_B:
            F_B_block[a][b] = 0
        if (a, b) not in large_G:
            F_G_block[a][b] = 0
        if (a, b) not in large_R:
            F_R_block[a][b] = 0
```

```
for y in range(16):
    for x in range(16):
        S_B = 0
        S_G = 0
        S_R = 0
        for v in range(16):
            cos_v = cos(v * pi * (2 * y + 1) / 32)
            C_v = 1
            if v == 0:
                C_v = sqrt(1 / 2)
            for u in range(16):
                C_u = 1
                if u == 0:
                    C_u = sqrt(1 / 2)
                cos_u = cos(u * pi * (2 * x + 1) / 32)
                S_B += C_v * C_u * F_B_block[u][v] * cos_v * cos_u
                S_G += C_v * C_u * F_G_block[u][v] * cos_v * cos_u
                S_R += C_v * C_u * F_R_block[u][v] * cos_v * cos_u
        result_B[i * 16 + x][j * 16 + y] = S_B / 8
        result_G[i * 16 + x][j * 16 + y] = S_G / 8
        result_R[i * 16 + x][j * 16 + y] = S_R / 8
```

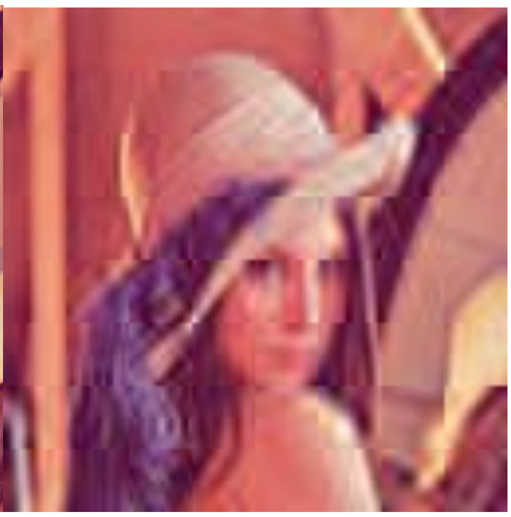


```
for i in range(128):
    for j in range(128):
        if result_B[i][j] < 0:
            result_B[i][j] = 0
        elif result_B[i][j] > 255:
            result_B[i][j] = 255
        if result_G[i][j] < 0:
            result_G[i][j] = 0
        elif result_G[i][j] > 255:
            result_G[i][j] = 255
        if result_R[i][j] < 0:
            result_R[i][j] = 0
        elif result_R[i][j] > 255:
            result_R[i][j] = 255
# print("B : ", result_B)
# print("G : ", result_G)
# print("R : ", result_R)
return cv2.merge((result_B, result_G, result_R))
# return result_R
```

결과

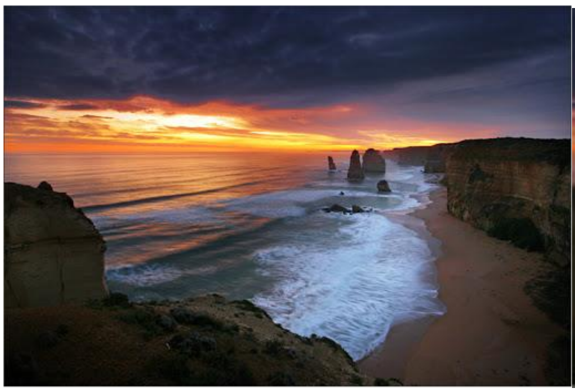
원　　본　　　　　　　　DCT/IDCT

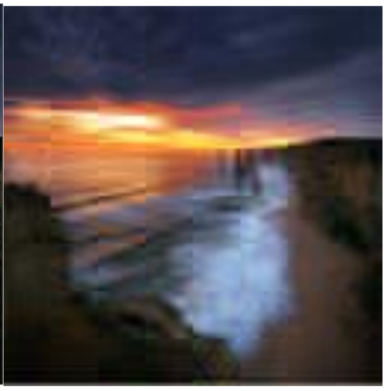




원 본

DCT/IDCT





원　　본

DCT/IDCT