Practical 6\AESRound.py

```python
import copy
S = (
    ["63","7c","77","7b","f2","6b","6f","c5","30","01","67","2b","fe","d7","ab","76"],
    ["ca","82","c9","7d","fa","59","47","f0","ad","d4","a2","af","9c","a4","72","c0"],
    ["b7","fd","93","26","36","3f","f7","cc","34","a5","e5","f1","71","d8","31","15"],
    ["04","c7","23","c3","18","96","05","9a","07","12","80","e2","eb","27","b2","75"],
    ["09","83","2c","1a","1b","6e","5a","a0","52","3b","d6","b3","29","e3","2f","84"],
    ["53","d1","00","ed","20","fc","b1","5b","6a","cb","be","39","4a","4c","58","cf"],
    ["d0","ef","aa","fb","43","4d","33","85","45","f9","02","7f","50","3c","9f","a8"],
    ["51","a3","40","8f","92","9d","38","f5","bc","b6","da","21","10","ff","f3","d2"],
    ["cd","0c","13","ec","5f","97","44","17","c4","a7","7e","3d","64","5d","19","73"],
    ["60","81","4f","dc","22","2a","90","88","46","ee","b8","14","de","5e","0b","db"],
    ["e0","32","3a","0a","49","06","24","5c","c2","d3","ac","62","91","95","e4","79"],
    ["e7","c8","37","6d","8d","d5","4e","a9","6c","56","f4","ea","65","7a","ae","08"],
    ["ba","78","25","2e","1c","a6","b4","c6","e8","dd","74","1f","4b","bd","8b","8a"],
    ["70","3e","b5","66","48","03","f6","0e","61","35","57","b9","86","c1","1d","9e"],
    ["e1","f8","98","11","69","d9","8e","94","9b","1e","87","e9","ce","55","28","df"],
    ["8c","a1","89","0d","bf","e6","42","68","41","99","2d","0f","b0","54","bb","16"]
)

Rconst = [
    "00000000", "01000000", "02000000", "04000000", "08000000",
    "10000000", "20000000", "40000000", "80000000", "1B000000", "36000000"
]

def xor(text: list, keywords: list) -> list:
    for ind, (txt, key) in enumerate(zip(text, keywords)):
        text[ind] = hex(int(txt, 16) ^ int(key, 16))[2:].zfill(8)
    return text


def SubBytes(text: list) -> list:
    idx: str = "0123456789abcdef"
    for ind, word in enumerate(text):
        newWord = ""
        for i in range(0,8,2):
            x = idx.index(word[i])
            y = idx.index(word[i+1])
            newWord += S[x][y]
        text[ind] = newWord
    return text

def shiftRows(text: list) -> list:
    text:str = "".join(text)

    w1: str = text[0:2] + text[10:12] + text[20:22] + text[30:32]
    w2: str = text[8:10] + text[18:20] + text[28:30] + text[6:8]
    w3: str = text[16:18] + text[26:28] + text[4:6] + text[14:16]
    w4: str = text[24:26] + text[2:4] + text[12:14] + text[22:24]

    return [w1, w2, w3, w4]


def gmul(a, b):
    if b == 1:
        return a
    tmp = (a << 1) & 0xff
    if b == 2:
        return tmp if a < 128 else tmp ^ 0x1b
    if b == 3:
        return gmul(a, 2) ^ a
```

```python

def MixColumn(text: list) -> list:
    for i, word in enumerate(text):
        a = int(word[0:2], 16)
        b = int(word[2:4], 16)
        c = int(word[4:6], 16)
        d = int(word[6:8], 16)

        w1 = hex(gmul(a, 2) ^ gmul(b, 3) ^ gmul(c, 1) ^ gmul(d, 1))[2:]
        w2 = hex(gmul(a, 1) ^ gmul(b, 2) ^ gmul(c, 3) ^ gmul(d, 1))[2:]
        w3 = hex(gmul(a, 1) ^ gmul(b, 1) ^ gmul(c, 2) ^ gmul(d, 3))[2:]
        w4 = hex(gmul(a, 3) ^ gmul(b, 1) ^ gmul(c, 1) ^ gmul(d, 2))[2:]

        text[i] = w1+w2+w3+w4

    return text

def generateKey(keywords: list, roundCount: int) -> list:
    Rcnst = Rconst[roundCount]

    four = keywords[-1]
    four = four[2:] + four[:2] # Rotate Word
    four = SubBytes([four])[0]  # subWord
    four = xor([four] , [Rcnst])[0]  # xor with Rconst
    newKeywords = list()
    newKeywords.append(four)
    for i, key in enumerate(keywords):
        newKeywords.append(xor([newKeywords[i]], [key])[0])
    return newKeywords[1:]

def printAES(round, text, state, keyword) -> None:
    print(f'{'-'*60}')
    for i in range(4):
        if i == 2:
            print(f'|{round:^10}', end="|")
        else:
            print(f'|{"":^10}', end="|")
        print(" ", text[0][i*2:i*2+2].zfill(2), text[1][i*2:i*2+2].zfill(2), text[2][i*2:i*2+2].zfill(2), text[3][i*2:i*2+2].zfill(2), " ", end="|")
        print(" ", state[0][i*2:i*2+2].zfill(2), state[1][i*2:i*2+2].zfill(2), state[2][i*2:i*2+2].zfill(2), state[3][i*2:i*2+2].zfill(2), " ", end="|")
        print(" ", keyword[0][i*2:i*2+2].zfill(2), keyword[1][i*2:i*2+2].zfill(2), keyword[2][i*2:i*2+2].zfill(2), keyword[3][i*2:i*2+2].zfill(2), " |")
    print(f'{'-'*60}')



def advanceEncryptionStandard(plaintext: list, keywords: list) -> None:

    # preRoundTransformation
    pt = copy.deepcopy(plaintext)
    stateMatrix = xor(plaintext, keywords)
    printAES('PreRound',pt, stateMatrix, keywords)

    for roundCount in range(1, 11):
        pt = copy.deepcopy(stateMatrix)
        #Step 1: Substitution Bytes
        stateMatrix = SubBytes(stateMatrix)
        #Step 2: Shift rows
        stateMatrix = shiftRows(stateMatrix)
        #step 3: Mix Column Multiplication
        if roundCount != 10:
            stateMatrix = MixColumn(stateMatrix)
        #step4: Add RoundKey
        keywords = generateKey(keywords, roundCount)
        stateMatrix = xor(stateMatrix, keywords)
        printAES(roundCount, pt, stateMatrix, keywords)

    return stateMatrix


def main() -> None:
    plaintext = list(input("Enter PlainText (in HEX format XXXX XXXX XXXX XXXX): ").lower().split(" "))
    #00041214 12041200 0c001311 08231919
    keywords = list(input("Enter Your Key (in HEX format XXXX XXXX XXXX XXXX): ").lower().split(" "))
    #2475a2b3 34755688 31e21200 13aa5487

    if len(keywords) != 4 or len(plaintext) != 4:
        print("There must be 4 input keys")
        return

    for key, text in zip(keywords, plaintext):
        if len(key) != 8 or len(text) != 8:
            print("Each input key must be 8 hex characters long!")
            return

    print("-"*60)
    print(f'|{'Round':^10}|{'Input State':^15}|{'Output State':^15}|{'Round Key':^15}|')
    print("-"*60)
    advanceEncryptionStandard(plaintext, keywords)

if __name__ == "__main__":
    main()
```