21. Write a python program for ECB, CBC, and CFB modes, the plaintext must be a sequence of one or more complete data blocks (or, for CFB mode, data segments). In other words, for these three modes, the total number of bits in the plaintext must be a positive multiple of the block (or segment) size. One common method of padding, if needed, consists of a 1 bit followed by as few zero bits, possibly none, as are necessary to complete the final block. It is considered good practice for the sender to pad every message, including messages in which the final message block is already complete. What is the motivation for including a padding block when padding is not needed?

---- Utility functions ----

Code:

```
def to_bytes(text):
    return bytearray(text, 'utf-8')
def pad(data, block_size):
    data = bytearray(data)
    # append 1 bit → add byte 0x80 (10000000)
    data.append(0x80)
    # add zeros until full block
    while len(data) % block_size != 0:
        data.append(0x00)
    return data
def xor_blocks(b1, b2):
    return bytearray([x ^ y for x, y in zip(b1, b2)])
# ---- Toy Block Cipher (XOR cipher) ----
def encrypt_block(block, key):
    return xor_blocks(block, key)

def decrypt_block(block, key):
    return xor_blocks(block, key)
```

```
# ---- ECB MODE ----
def ecb_encrypt(plaintext, key):
    block_size = len(key)
    data = pad(to_bytes(plaintext), block_size)
 ciphertext = bytearray()
    for i in range(0, len(data), block_size):
        block = data[i:i+block_size]
        ciphertext.extend(encrypt_block(block, key))
    return ciphertext
def ecb_decrypt(ciphertext, key):
    block_size = len(key)
    plaintext = bytearray()
    for i in range(0, len(ciphertext), block_size):
        block = ciphertext[i:i+block_size]
        plaintext.extend(decrypt_block(block, key))
    return plaintext
# ---- CBC MODE ----
def cbc_encrypt(plaintext, key, iv):
    block_size = len(key)
    data = pad(to_bytes(plaintext), block_size)
 ciphertext = bytearray()
    prev = iv
 for i in range(0, len(data), block_size):
        block = data[i:i+block_size]
        xored = xor_blocks(block, prev)
        enc = encrypt_block(xored, key)
        ciphertext.extend(enc)
```

```
        prev = enc
    return ciphertext
def cbc_decrypt(ciphertext, key, iv):
    block_size = len(key)
    plaintext = bytearray()
    prev = iv
    for i in range(0, len(ciphertext), block_size):
        block = ciphertext[i:i+block_size]
        dec = decrypt_block(block, key)
        plaintext.extend(xor_blocks(dec, prev))
        prev = block
return plaintext
# ---- CFB MODE ----
def cfb_encrypt(plaintext, key, iv):
    block_size = len(key)
    data = pad(to_bytes(plaintext), block_size)
   ciphertext = bytearray()
    shift = iv
  for i in range(0, len(data), block_size):
        enc = encrypt_block(shift, key)
        block = data[i:i+block_size]
        c = xor_blocks(block, enc)
        ciphertext.extend(c)
        shift = c  # feedback
 return ciphertext
def cfb_decrypt(ciphertext, key, iv):
    block_size = len(key)
    plaintext = bytearray()
```

```
    shift = iv
for i in range(0, len(ciphertext), block_size):
        enc = encrypt_block(shift, key)
        block = ciphertext[i:i+block_size]
        p = xor_blocks(block, enc)
        plaintext.extend(p)
        shift = block
  return plaintext
# ---- Example ----
key  = bytearray(b"12345678")  # 8-byte key
iv   = bytearray(b"ABCDEFGH")  # 8-byte IV
pt   = "HELLO WORLD"
print("ECB:", ecb_encrypt(pt, key))
print("CBC:", cbc_encrypt(pt, key, iv))
print("CFB:", cfb_encrypt(pt, key, iv))
```