**Decoding has 2 parts:**

```
# Convert binary to message
def bin_to_msg(binary):
    chars = [chr(int(binary[i:i+8], 2)) for i in range(0, len(binary), 8)]
    text = ''.join(chars)
    if END_MARKER in text:
        return text[:text.index(END_MARKER)]
    else:
        return "[Error: No hidden message found or corrupted image]"
```

**1. Converting Binary to Message**

The function bin_to_msg() basically does the reverse of what we saw in msg_to_bin().

1. First, we take the long binary string and break it into chunks of 8 bits.
   – Because each character was originally encoded using 8 bits.
2. Then, each of these 8-bit chunks is converted back into a character using chr(int(...)).
   – This gives us the original characters from the hidden message.
3. All characters are then combined to form the complete message.
4. Now we check if the message contains an END_MARKER, If we find it, that means the message was successfully decoded, and we return only the part before the marker.
5. But if the marker isn't found, we return an error message. This could mean either there was no hidden message, or the image might be corrupted.

---

```
# Extract hidden message from image
def extract_data(img):
    binary = ""
    for pixel in img.getdata():
        for value in pixel[:3]:
            binary += str(value & 1)

    chunks = [binary[i:i+8] for i in range(0, len(binary), 8)]
    message = ""
    for byte in chunks:
        char = chr(int(byte, 2))
        message += char
        if END_MARKER in message:
            return message[:message.index(END_MARKER)]
    return "[Error: No hidden message found or corrupted image]"
```

**2. Extracting the Hidden Message**

1. We go through each pixel of the image, and for each pixel, we extract the last bit of the Red, Green and Blue values."
   As we hid our message in those last bits, so now we're collecting them back into one long binary string.
2. Next, we split the binary string into 8-bit chunks — because each character in our message was stored as 8 bits.
3. We convert each 8-bit chunk back into a character using chr().
   Then we keep adding the characters together to rebuild the original message.
   If we find our special keyword — the END_MARKER — that means we've reached the end of the hidden message.
   So we return only the message part and ignore anything beyond that marker.
4. But if the END_MARKER is never found, we assume something’s wrong, maybe the image didn’t have a hidden message, or it got corrupted. In that case, we return an error message to inform the user.