Relatório do Desafio Syngenta Digital

Matheus de Miranda Novelli

A linguagem escolhida para o desafio foi a Python, devido a seu grande número de pacotes, que poderiam ser de grande ajuda na manipulação da imagem bitmap que foi fornecida.

Primeira parte

A primeira parte do desafio consistia na contagem dos pixel verdes do arquivo bitmap dado.

Esse bitmap era majoritariamente composto por pixels pretos e também possuía pixels brancos que formavam um texto no meio, além dos pixels verdes, que estavam espalhados de uma maneira que parecia randômica por toda a imagem, que precisavam ser contados.

Para realizar essa contagem, utilizei o pacote PIL (Python Imaging Library), para poder manipular a imagem com mais facilidade, e utilizei o método checkforpixels(), que foi criado dentro do arquivo main.py, para contar o número de pixels por cor, usando o código RGB.

Com isso, foi possível encontrar um resultado final para o número de pixels verdes, que foi de **298**.

Segunda parte

Já a segunda parte do desafio enuncia a existência de uma mensagem escondida no arquivo bitmap e pede para que ela seja encontrada, sem mais informações.

Visto essa ausência de informações e a falta de conhecimento quanto a criptografia de mensagem em imagens, tive que pesquisar mais sobre o assunto e a maioria esmagadora dos resultados falavam sobre uma técnica de criptografia chamada esteganografia. A técnica da esteganografia que tentei abordar foi a LSB (Least Significant Bit).

Uma imagem digital é um conjunto de valores, denominados pixels. Cada pixel guarda três valores, que vão de 0 a 255 (8 bits), o RGB, que representa sua luminosidade e determina sua coloração.

A técnica do LSB consiste em pegar o bit menos significativo de cada byte dos pixels e manipulá-lo para formar uma mensagem.

Nas minhas tentativas, criei um método imagedecoder1(), que percorre a lista (imagepixel) que contém os valores RGB de cada pixel e o converte para byte, usando o método converttobinary(), criado anteriormente. Após a conversão, o último bit de cada byte é extraído e armazenado em uma bitstream. Em seguida, cada byte desse bitstream é extraído e armazenado em uma lista, que é percorrida para que cada um de seus elementos seja convertido para binário e depois para um caractere, utilizando a tabela ASCII. O resultado atingido não foi o esperado e somente foram descobertos caracteres que aparentavam não ter significado algum.

O segundo método utilizado foi o imagedecoder2(), que seguiu o mesmo princípio do anterior até a criação da bitstream, mas que em seguida partiu para outra abordagem, tentando converter essa bitstream para caracteres utilizando funções built-in do Python. Essa abordagem não funcionou e a função não conseguiu passar pela compilação, pois, aparentemente o método de codificação que as funções built-in utilizam não foi o mesmo utilizado na codificação da imagem.