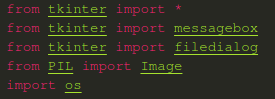
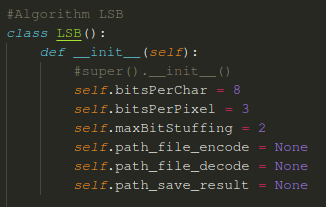
**LSB Algorithm Support**

1. LSB Algorithm

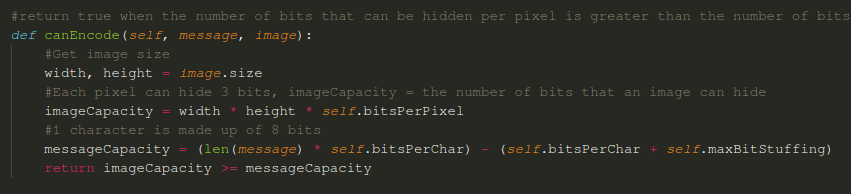
* Ý tưởng:
  + Mỗi 1 ảnh có nhiều pixel, mỗi pixel có 3 giá trị RGB (ảnh màu), mỗi giá trị R hoặc G hoặc B đều đổi được sang dạng nhị phân (ví dụ R = 1 => nhị phân 0000 0001
  + Đổi từng kí tự của message muốn ẩn sang dạng nhị phân (ví dụ kí tự "H" trong bảng ASCII là 72 đổi sang nhị phân là 01001000
  + Mỗi pixel ta tận dụng được 1 bit cuối của mỗi giá trị R, G, B. Vậy 3 pixel tận dụng được 9 bit, đủ để ẩn 1 kí tự (1 kí tự dùng 8 bit)
  + Kiểm tra độ dài của message, nếu số kí tự quá dài, vượt qua số lượng pixel có thể đáp ứng thì thông báo
  + Cách đổi cụ thể:
    - Kí tự ‘H' 01001000
    - Giả sử 3 pixel của 1 bức ảnh có giá trị là (27, 64, 164), (248, 244, 194), (174, 246, 250)
    - Đổi giá trị R G B của pixel thành lẻ cho 1 và chẵn cho 0. Ví dụ bit đầu là '0' của "H" gặp 27 của pixel, đổi 27 thành 26 cứ đổi như vậy đến khi hết 8 bit của 1 kí tự
    - Kết quả (26, 63, 164), (248, 243, 194), (174, 246, 250)
  + Ảnh kết quả sau khi ẩn nên để định dạng png, vì đây là kiểu nén ảnh không mất dữ liệu
* Import thư viện cần thiết cho chương trình, sử dụng tkinter trong python để tạo giao diện



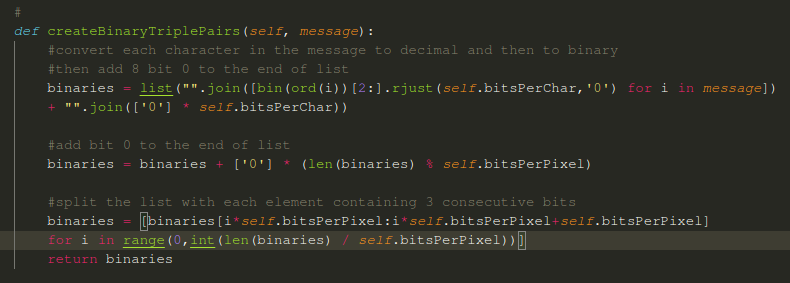
* Tạo class LSB() để xây dựng nội dung thuật toán



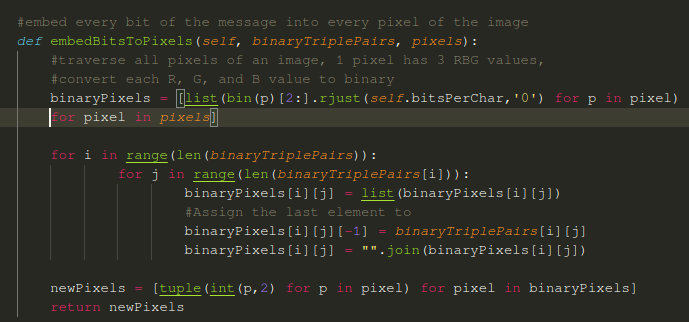
* Hàm **canEncode**:
  + Tính tổng pixel của ảnh đầu vào, mỗi pixel có thể hỗ trợ ẩn 3 bit (tham số bitsPerPixel được khởi tạo = 3 trong hàm init).
  + Sau đó tính độ dài message nhập vào, mỗi kí tự trong message cần 8 bit để biểu diễn, nếu tổng pixel đáp ứng đủ độ dài message thì trả về **true** để thực hiện bước tiếp theo



* Hàm **createBinaryTriplePairs**: Nhóm 3 bit liên tiếp của 1 kí tự thành 1 phần tử trong list
  + Duyệt từng kí tự trong message và chuyển chúng thành nhị phân bằng **bin(ord(i)) =>** ví dụ kí tự “H” sẽ có dạng **0b1001000**, sử dụng **"".join([bin(ord(i))[2:].rjust(*self*.bitsPerChar,'0')** để cắt phần 0b của mỗi kí tự và chèn ‘0’ vào để đủ 8 bit cho 1 kí tự, sau đó thêm 8 bit 0 vào trong list
  + 8 bit 0 đại diện cho kí tự NULL trong bảng ASCII, mục tiêu là để đánh dấu điểm kết thúc của đoạn message ẩn vào ảnh. Khi giải mã, tìm mật khẩu ẩn trong ảnh, nếu đọc được 8 bit 0 liên tiếp này, sẽ trả về đoạn message đã ẩn
  + Giả sử ta nhập vào 3 kí tự => 24 bit, sau đó + thêm 8 bit 0 => 32 bit => 1 list có 32 phần tử, nhóm 3 phần tử liên tiếp lại, ta sẽ thấy 2 bit cuối cùng không tạo thành 1 nhóm 3 phần tử được do thiếu
    - Phướng án: ta lấy độ dài của list hiện tại chia dư cho 3, rồi cộng thêm chính số dư lần bit 0 vào list hiện tại => khi đó ta sẽ có 1 list gồm 34 phần tử
  + Tạo 1 list với mỗi phần tử là 3 bit liên tiếp của list trước



* Hàm **embedBitsToPixels**: Nhúng phần tử chứa 3 bit liên tiếp của 1 kí tự vừa tạo ở bên trên vào từng pixel của ảnh
  + Duyệt toàn bộ pixel của 1 ảnh, 1 pixel có 3 giá trị RGB, đổi từng giá trị R, G , B thành nhị phân, tương tự như phía trên, sử dụng câu lệnh **[list(bin(p)[2:].rjust(*self*.bitsPerChar,'0') for p in pixel) for pixel in *pixels*]** => ta sẽ có 1 list mẹ chứa từng pixel của ảnh, và 1 list con chứa từng giá trị nhị phân R G B của từng pixel trong ảnh
  + Từ hàm **createBinaryTriplePairs**, ta sẽ có 1 list mẹ với mỗi phần tử là 1 list con chứa 3 bit liên tiếp của 1 kí tự. Ta có 2 vòng for như dưới đây: vòng for 1 - duyệt từng phần tử của list mẹ, vòng for thứ 2 duyệt từng phần tử của list con
  + Gán bit cuối cùng của từng giá trị R G B trong ảnh với từng phần tử trong list con trả về từ hàm **createBinaryTriplePairs** phía trên dựa vào câu lệnh **binaryPixels[i][j][-1] = *binaryTriplePairs*[i][j]**
  + Sau khi gán hoàn tất, ta lại đổi giá trị nhị phân của từng R G B trong từng pixel của ảnh thành thập phân dựa vào **[tuple(int(p,2) for p in pixel) for pixel in binaryPixels]** với int(p,2) đổi từ nhị phân sang thập phân



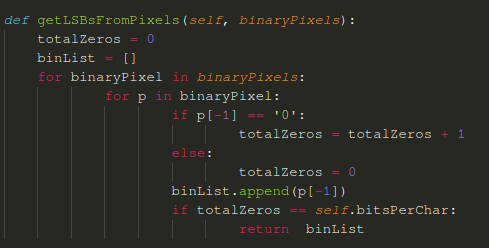
* Hàm **LSB\_encode:** Ẩn mật mã vào ảnh
  + Mở ảnh, sử dụng hàm **canEncode** phía trên để kiểm tra điều kiện giữa mật mã nhập vào và kích thước ảnh
  + Gọi hàm **createBinaryTriplePairs()** và truyền mật mã muốn ẩn làm đối số đầu vào
  + Lấy giá trị các pixel của ảnh với hàm getdata()
  + Nhúng các bit của mật mã vào ảnh sử dụng hàm **embedBitsToPixels()**
  + Tạo 1 ảnh RGB mới và sao chép các giá trị pixels trả về từ hàm **embedBitsToPixels()** vào ảnh mới đó
  + Lưu ảnh thu được vào đường dẫn đã chọn ở nút **Save Image**



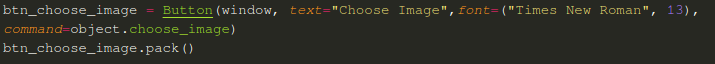
* Hàm **LSB\_decode**:
  + Mở ảnh, lấy giá trị các pixel của ảnh, duyệt toàn bộ pixel của ảnh, 1 pixel có 3 giá trị RGB, đổi từng giá trị R, G , B thành nhị phân như bước trên
  + Gọi hàm **getLSBsFromPixels** để lấy ra các bit của kí tự đã nhúng vào ảnh và lưu vào biến **binList**
  + Duyệt từ phần tử đầu tiên đến **len(binList)** trừ đi 8 bit 0 thêm vào phía sau lúc encode ở phía trên
  + Gép các bit vào nhau và chuyển lại thành giá trị thập phân (**int("".join(binList[i:i+*self*.bitsPerChar]),2)**), từ giá trị thập phân thành từng kí tự bằng cách đặt **chr()** để ép kiểu

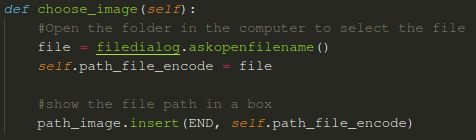


* Hàm **getLSBsFromPixels**:
  + Duyệt từng phần tử R G B đã chuyển sang nhị phân của mỗi pixel trong ảnh, nếu check được 8 bit 0 liên tiếp thì trả về chuỗi bit

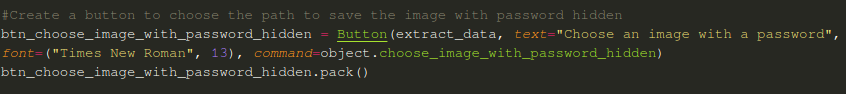


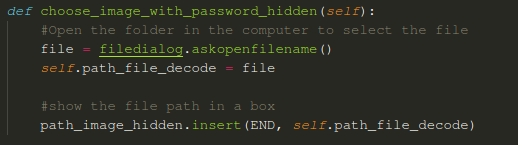
* Tạo nút “Choose Image” để chọn ảnh, khi click vào nút này, hàm choose\_image sẽ được gọi, sử dụng **filedialog.askopenfilename()** để chọn file trong máy tính của bạn, và sử dụng hàm **insert** để hiển thị đường dẫn vừa chọn vào box phía dưới



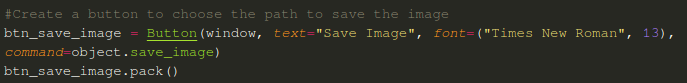


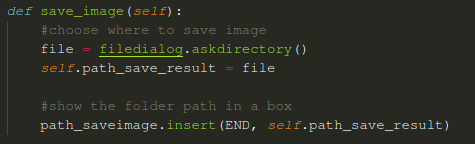
* Tương tự như phía trên, nhưng đây là nút chọn ảnh đã ẩn mật khẩu cho option extract



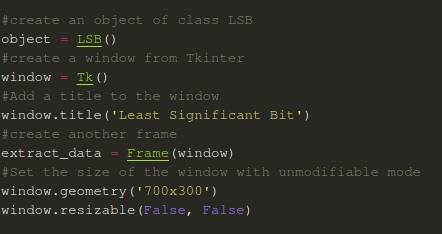


* Tương tự như phía trên, nhưng khác 1 chút, **filedialog.askdirectory()** được gọi để lấy đường dẫn folder mà bạn muốn lưu ảnh vào đó

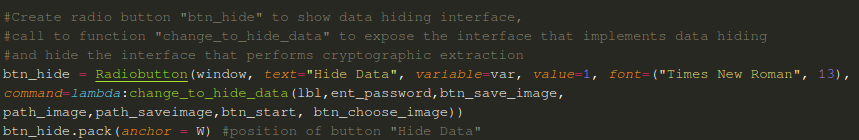


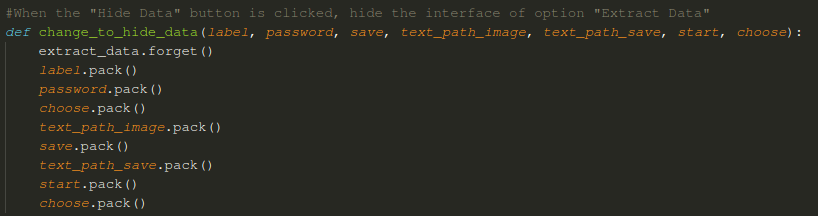


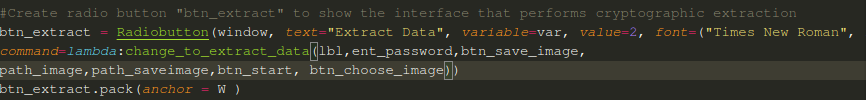
* Tạo 1 cửa sổ sử dụng tkinter, gắn title mà bạn muốn, cài đặt kích thước cửa số hiển thị ra màn hình, tạo 1 frame khác có tên là extract\_data, mục tiêu là khi sử dụng option Extract data, giao diện phần này sẽ hiển thị và ẩn giao diện phần Hide data đi

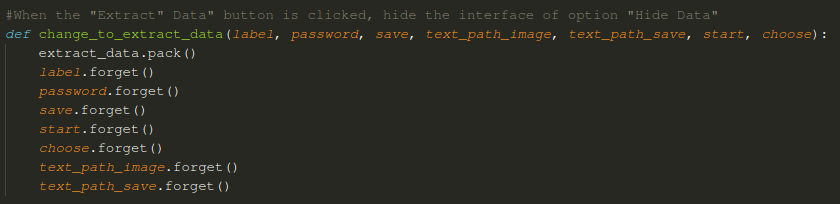


* Tạo 2 radiobutton Hide Data và Extract Data, đây là 2 option của ứng dụng, ẩn và hiện thị mật mã trong ảnh, click vào nút nào hàm tương ứng sẽ được gọi, và phần giao diện tương ứng sẽ hiển thị, hàm **pack()** để hiển thị và **forget()** để ẩn giao diện, những tham số truyền vào là những button, label hay box phục vụ chức năng trong từng option

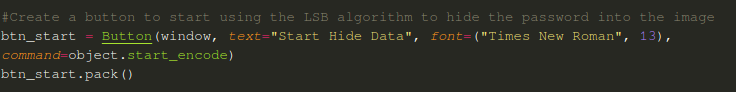


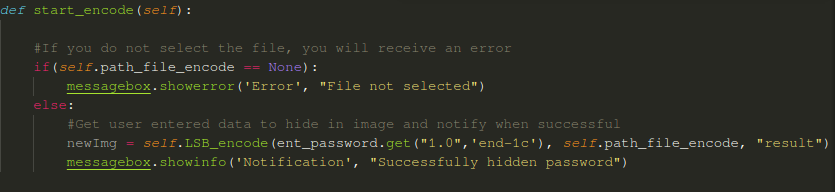






* Tạo nút Start Hide Data, khi click vào button này, hàm start\_encode sẽ được gọi, nếu chưa chọn ảnh đầu vào, sẽ báo lỗi, nếu chọn rồi, LSB\_encode sẽ lấy password mà bạn đã nhập, ảnh đầu vào để thực hiện thuật toán, và trả về là ảnh "result.png” trong folder mà bạn muốn lưu





* Tạo nút Show hidden password, khi click vào button này, hàm start\_decode sẽ được gọi, nếu chưa chọn ảnh đầu vào, sẽ báo lỗi, nếu chọn rồi, LSB\_decode sẽ lấy ảnh mà bạn chọn để thực hiện thuật toán, và trả về password đã ẩn trên ô thông báo

