Worcester Polytechnic Institute
Department of Computer Science
Computer Forensics

Hands on 8: Develop your own steganography tool

Objectives

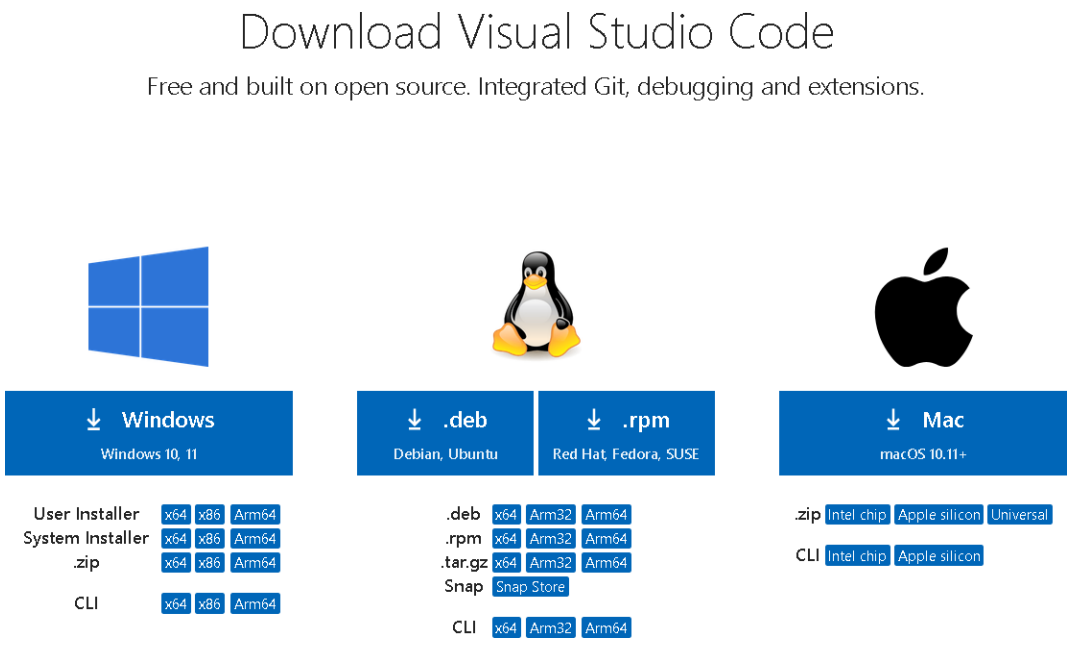
- Hide a message using the least-significant-bits of pixels in an image
- Extract the message from the steganography file

Tasks

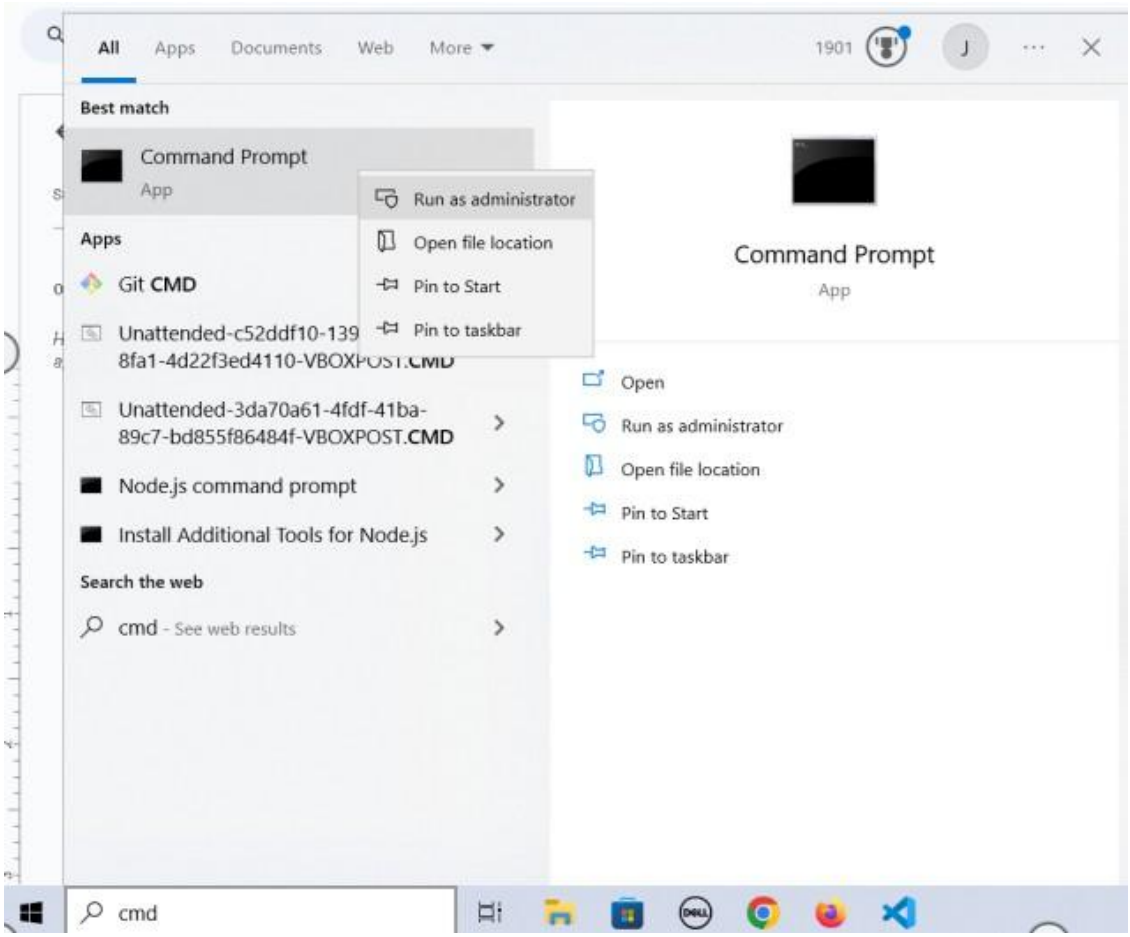
Task 0. Software Preparation

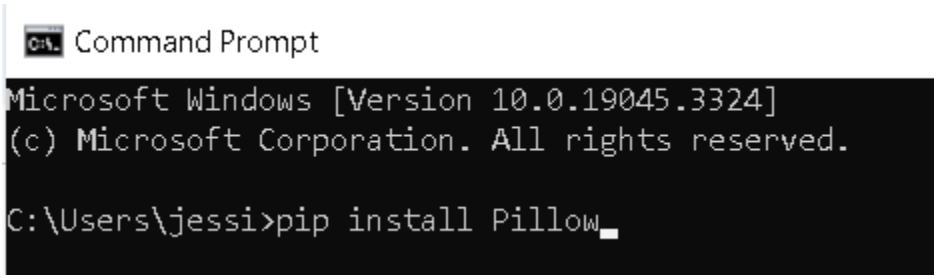
1. Download the python code templates for myStegTool.py and messageExtract.py

2. Install a IDE that supports python, such as Visual Studio Code:
   https://code.visualstudio.com/download

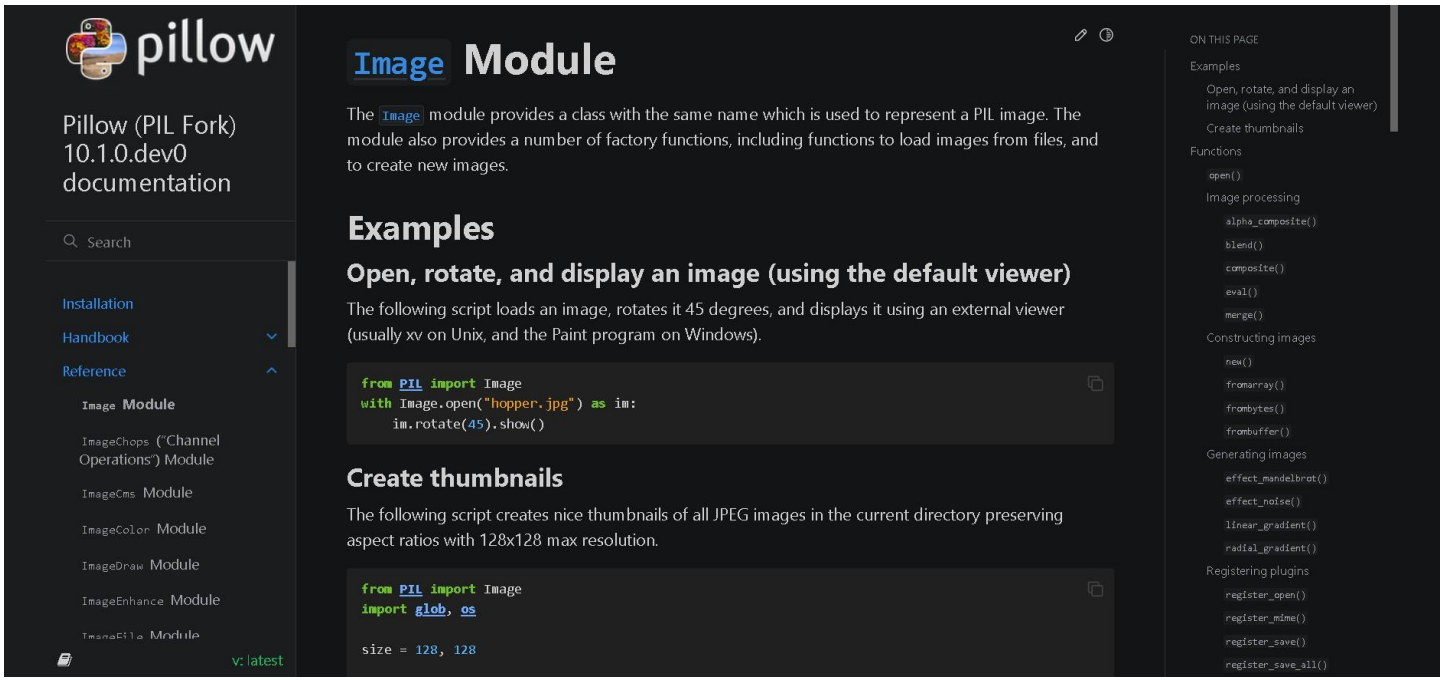


3. If you do not have the PIL library installed, run the following command to install it:
   pip install Pillow



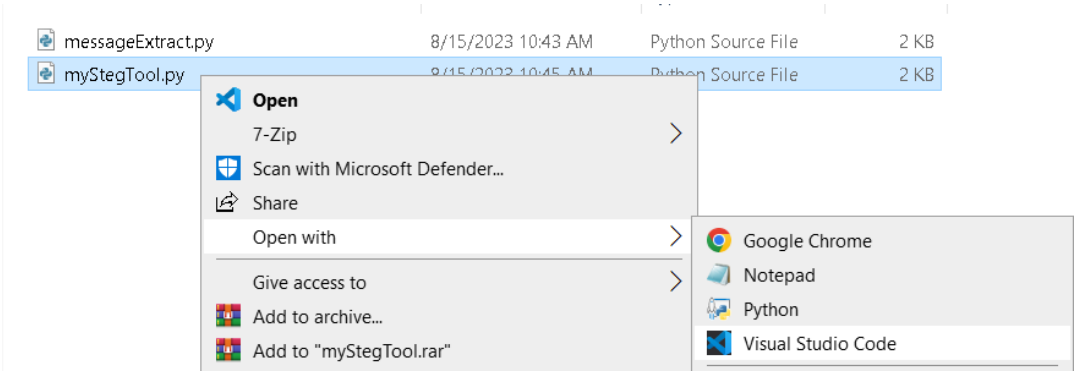


4. You may also want to have the Pillow documentation ready to refer to https://pillow.readthedocs.io/en/latest/reference/Image.html#



Task 1. Finish coding myStegTool.py

1. Open myStegTool.py in Visual Studio Code. You can right click it to open it with Visual Studio Code



2. The packages are already imported for you and the main() function has already been written. This program will open using the command line where the user can pass in the name of the file they would like to use. If an image name is not passed, the program will close. If an image name is passed, the program will ask for a message and run the hide_message() function to insert the message in the image's pixels using the least significant bits. We will be finishing the hide_message() function. You will need to edit the lines that say TODO and uncomment them.
3. Use the Image package to open the image passed into the program

Functions

`PIL.Image.open(fp, mode='r', formats=None)` [source]

Opens and identifies the given image file.

This is a lazy operation; this function identifies the file, but the file remains open and the actual image data is not read from the file until you try to process the data (or call the `load()` method). See `new()`. See File Handling in Pillow.

PARAMETERS:

- **fp** – A filename (string), pathlib.Path object or a file object. The file object must implement `file.read`, `file.seek`, and `file.tell` methods, and be opened in binary mode. The file object will also seek to zero before reading.

```
def hide_message(image_path, message):
    # Open the image using the .open() function from Image and image_path
    img = Image.open(image_path)
```

4. Convert the message parameter to binary. We will be using join(), format(), and ord().
For char in message - Loops through the characters in the message that was passed into the function, each character will be referred to as char
ord(char) - Gets the unicode value of char
format(ord(char), '08b') - used to specify we want to format this char in binary with 8 characters
''.join() - '' is an empty string and characters will be joined to the end of the string as the for loop iterates through the characters in message

```
# Convert the message to binary
binary_message = ''.join(format(ord(char), '08b') for char in message)
```

5. Load the pixels in img using the PIL library.

`Image.load()` [source]

Allocates storage for the image and loads the pixel data. In normal cases, you don't need to call this method, since the Image class automatically loads an opened image when it is accessed for the first time.

If the file associated with the image was opened by Pillow, then this method will close it. The exception to this is if the image has multiple frames, in which case the file will be left open for seek operations. See File Handling in Pillow for more information.

RETURNS:

An image access object.

```
# Embed the binary message into the image pixels
pixels = img.load()
```

6. Access the height of img using the PIL library

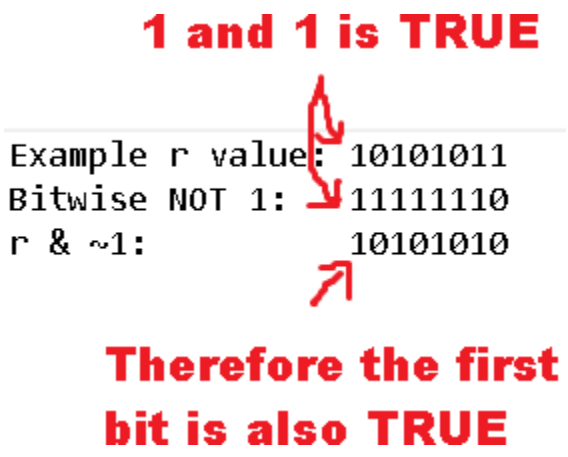
```
Image.width: int
    Image width, in pixels.
Image.height: int
    Image height, in pixels.
```

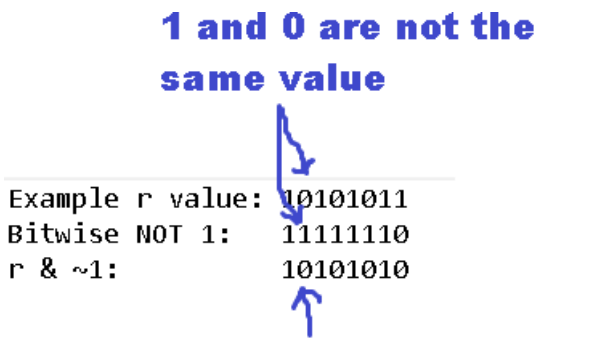
```
for x in range(img.width):
    for y in range(img.height):
        r, g, b = pixels[x, y]
```

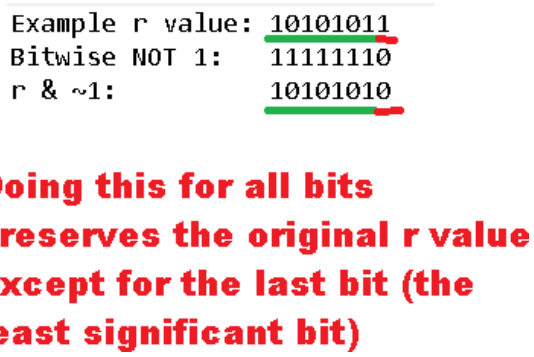
7. Clear only the last bit of the red(r) value. If binary_message[index] has a value then replace pixels[x,y] with that value.

```
if index < len(binary_message):
    pixels[x, y] = r & ~1 | int(binary_message[index])
```

if index < len(binary_message) - lets us know when the entire binary message has been added to the image
pixels[x, y] - the current pixel of the image we are on
~1 - Bitwise NOT 1. So this byte would be 11111110.
r & ~1 - R is the byte that represents the red value in this pixel. By using bitwise AND(&) 11111110, all of the bits except for the 0 at the end will be preserved. Here is an example:







r & ~1 | int(binary_message[index]) - If there is a value at the current index of binary_message, then it will replace the last bit. So the message will be spread out in order in the red values of the first pixels of the image.

8. Move to the next index by adding 1 to index

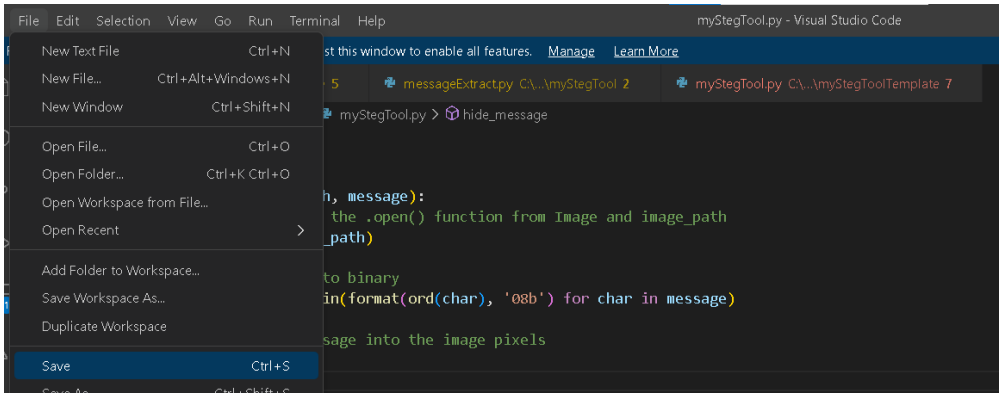
```
index += 1
```

We have passed a pixel now.

9. Save img by calling the .save() function from the PIL library on it. Pass in the file name "output.png" so you do not overwrite the original image.

```
# Save the steganographic image
img.save("output.png")
```

10. Save myStegTool.py by using CTRL+S or clicking File > save.



Task 2. Finish coding messageExtract.py

1. The packages are already imported for you and the main() function has already been written. This program will open using the command line where the user can pass in the name of the file they would like to use. If an image name is not passed, the program will close. If an image name is passed, the program will ask for a message and run the extract_message() function to extract the message in the image's pixels using the least significant bits. We will be finishing the extract_message() function. You will need to edit the lines that say TODO and uncomment them.
2. Open the steganographic image the same way we opened an image in myStegTool.py

```
def extract_message(image_path):
    # Open the steganographic image
    img = Image.open(image_path)
```

3. Load the pixels in img using the PIL library the same way we loaded the pixels of an image in myStegTool.py

```
# Extract the binary message from the image pixels
binary_message = ""
pixels = img.load()
```

4. Access the height of img using the PIL library the same way we did in Task 1 step 6

```
for x in range(img.width):
    for y in range(img.height):
        r, g, b = pixels[x, y]
```

5. Extract the least significant bit from the red channel

```
# Extract the least significant bit from the red channel
binary_message += str(r & 1)
```

binary_message - Will contain a binary value composed of the least significant bit from the red channel

r & 1 - r is the red value of each pixel, by using AND 00000001, we are only keeping the last bit. Here is an example of how it works:

```
Example r value: 10101011
1              : 00000001
Bitwise AND      00000001
```

We can see that none of the bits will result in TRUE except for potentially the last bit.

str() - converts the byte into a string

+= Adds this string onto the end of binary_message

6. How many bits are in a byte? Replace the following 2 TODOS with that number
   There are 8 bits in a byte.

```
# Convert the binary message back to text
message = ""
for i in range(0, len(binary_message), 8):
    byte = binary_message[i:i + 8]
```

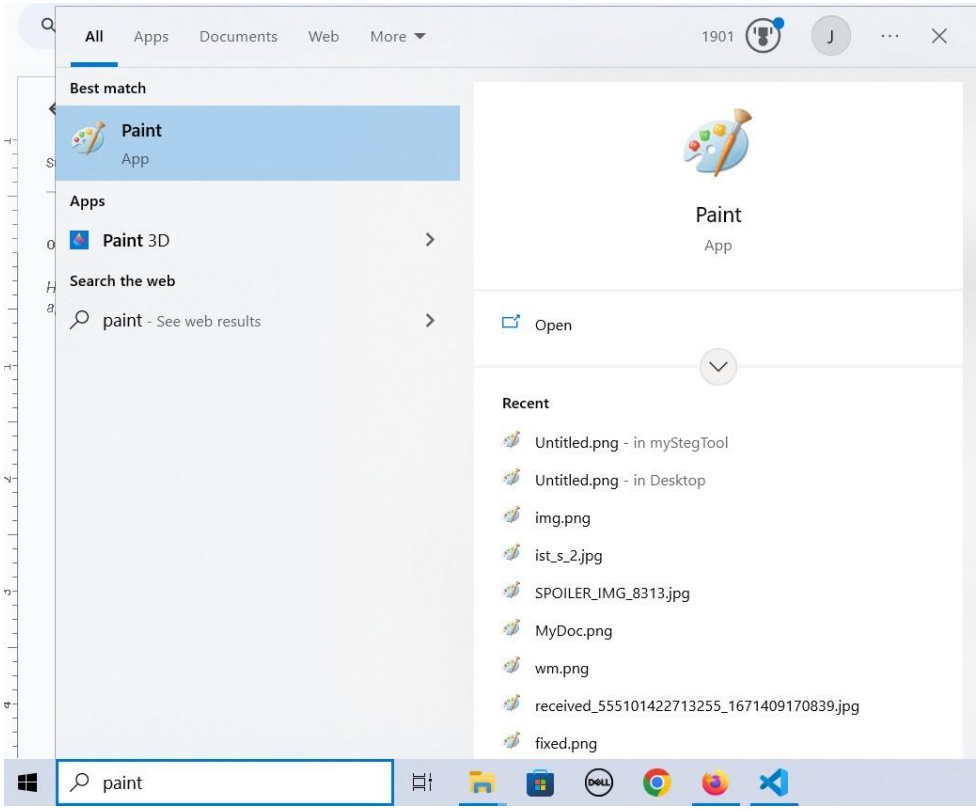
7. What is the base number in binary? Replace the following T0D0 with that number

```
message += chr(int(byte, 2))
```

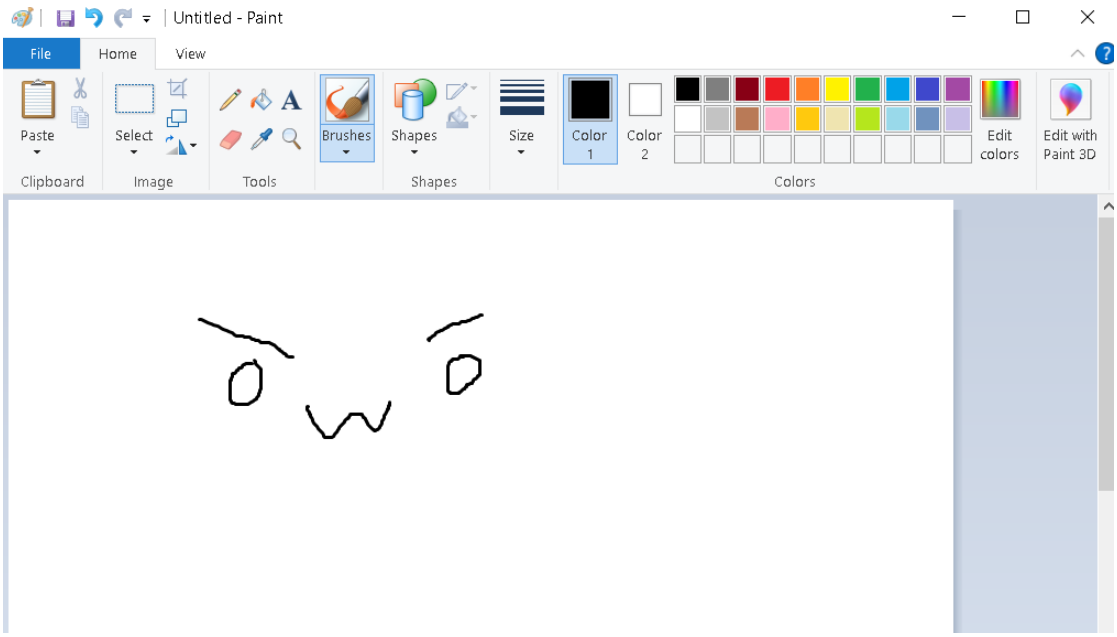
8. The code in the previous 2 steps lumps the binary value into groups of 8, which is how many bits are needed to make a byte. That 8 bit binary value is then converted to an int, which is the ascii value needed to find the corresponding char. That char is then added to the message value. Once the for loop is complete, the message will have all of the characters from the message **AS WELL AS** the least significant bits from pixels that did not have a hidden message hidden into them. This is because messageExtract.py has no way of determining if a bit contains a hidden message or not until after all the pixels are searched.
9. Save messageExtract.py by using CTRL+S or clicking File > save. You can now close VS Code.

Task 3. Create an image

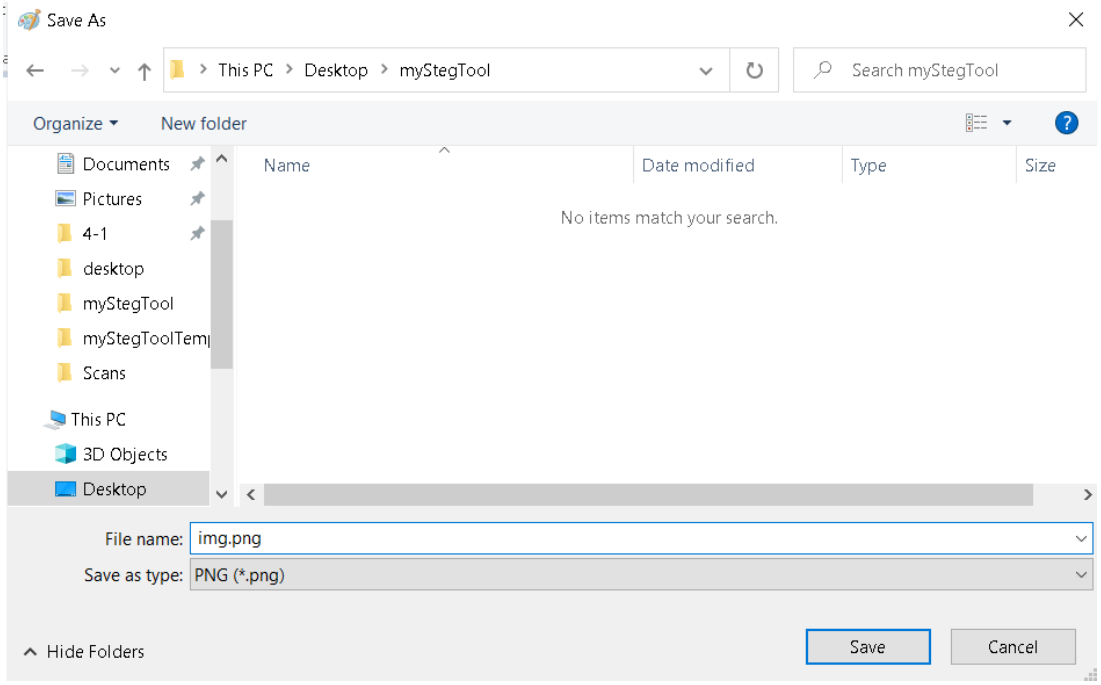
1. Open paint or any image software



2. Draw an image

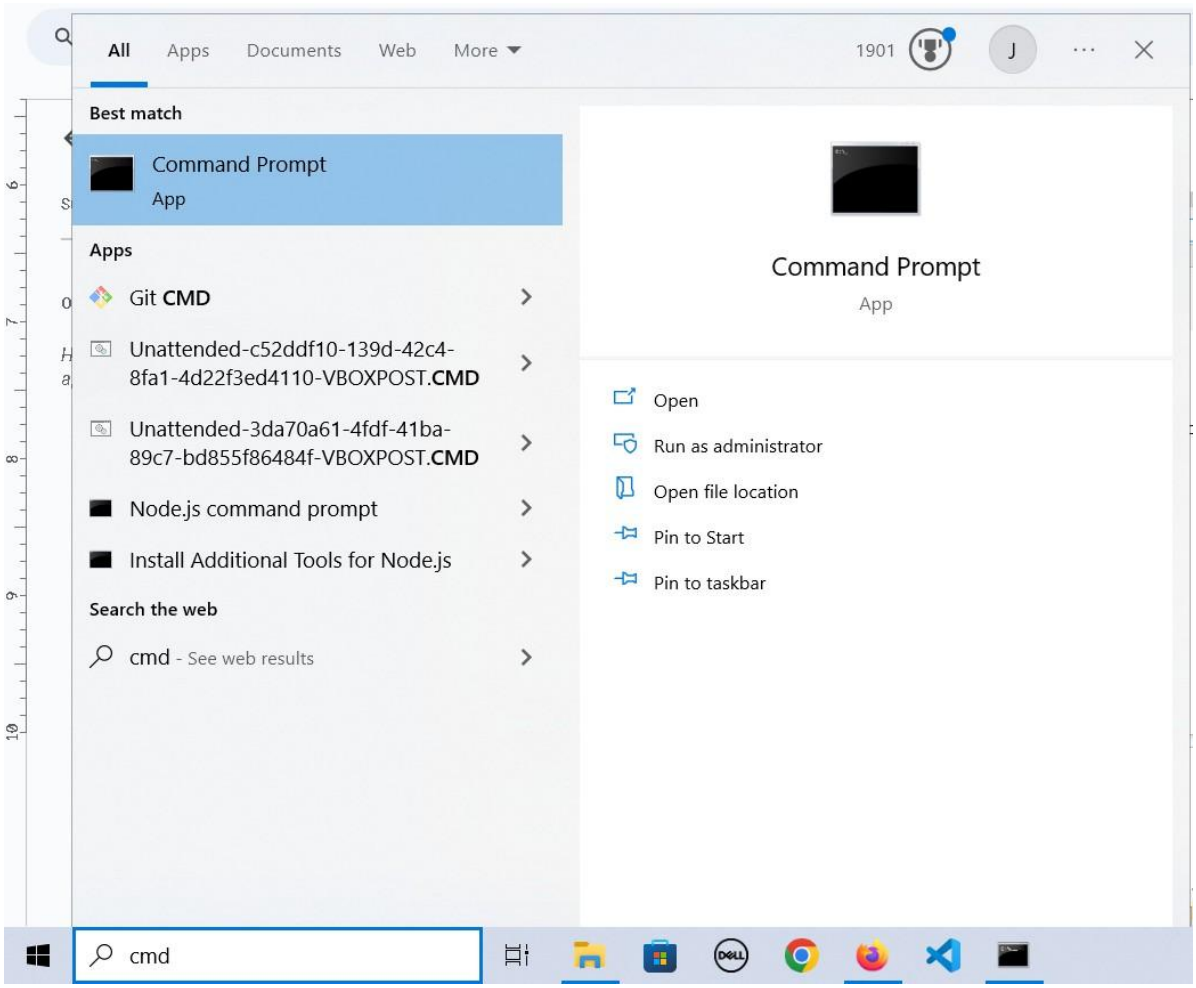


3. Save the image and make sure it's where myStegTool.py and messageExtract.py are

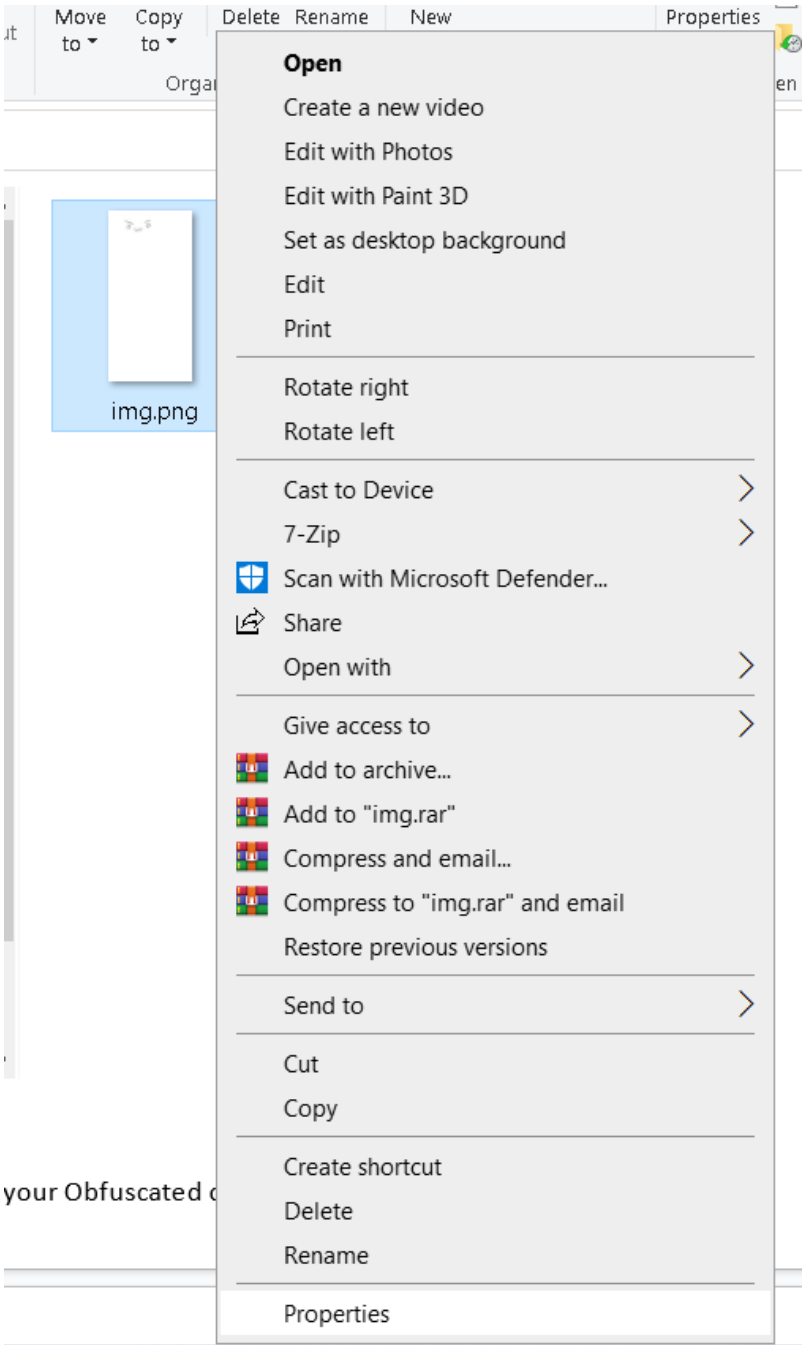


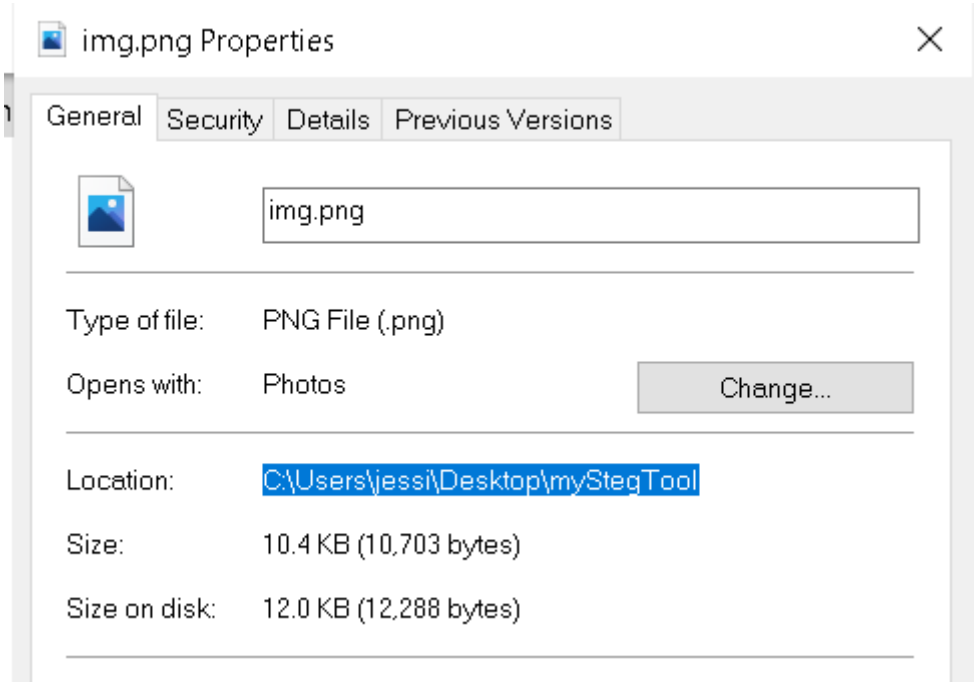
Task 4. Hide a message in the image using myStegTool.py

1. Open the command line



2. Use the cd command to move to the directory where your python and image files are. You can right click your files and find the location in “Properties”


Move to
Copy to
Delete
Rename
New
Properties
Open
Create a new video
Edit with Photos
Edit with Paint 3D
Set as desktop background
Edit
Print
Rotate right
Rotate left
Cast to Device
7-Zip
Scan with Microsoft Defender...
Share
Open with
Give access to
Add to archive...
Add to "img.rar"
Compress and email...
Compress to "img.rar" and email
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties
img.png
your Obfuscated




```
C:\Users\jessi>cd C:\Users\jessi\Desktop\myStegTool
```

3. Open myStegTool.py and pass in the name of your image
   Command line: myStegTool.py <image name>
   If this doesn't work, include python in the command: python myStegTool.py <image name>

```
C:\Users\jessi\Desktop\myStegTool>myStegTool.py img.png
Enter the message to hide:
```

4. Enter your message

```
C:\Users\jessi\Desktop\myStegTool>myStegTool.py img.png
Enter the message to hide: hello! this is a s3cret message...
Message hidden in the image. Output image: output.png

C:\Users\jessi\Desktop\myStegTool>
```

5. The output file will be in the same directory where myStegTool.py is located.

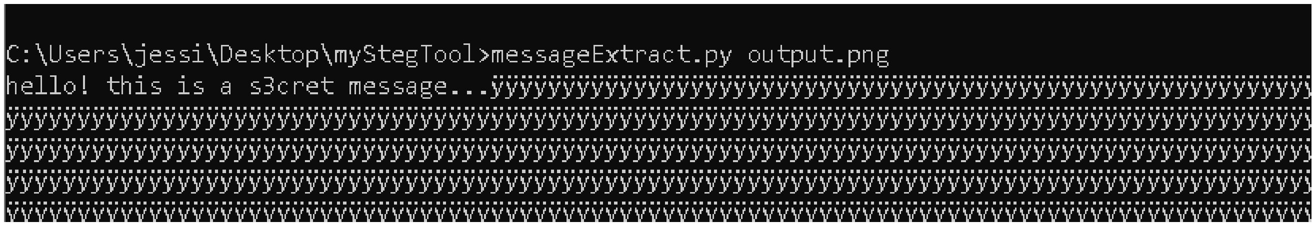
```
 Directory of C:\Users\jessi\Desktop\myStegTool

08/15/2023  01:23 PM    <DIR>          .
08/15/2023  01:23 PM    <DIR>          ..
08/15/2023  01:00 PM            10,703 img.png
08/15/2023  09:31 AM               935 messageExtract.py
08/15/2023  11:55 AM             1,111 myStegTool.py
08/15/2023  01:38 PM            11,604 output.png
               4 File(s)         24,353 bytes
               2 Dir(s)  225,194,733,568 bytes free

C:\Users\jessi\Desktop\myStegTool>_
```

Task 5. Extract the message in the image using extractMessage.py

1. Open messageExtract.py and pass in the name of your image
   Command line: messageExtract.py <image name>
   If this doesn't work, include python in the command: python messageExtract.py <image name>

2. The program will output the bytes of each least significant bit. The message will be the first characters, and there will be some nonsense after it. If your image was really big, the output will be really long.

QUESTIONS:

1. Show a screenshot of the message you hid using myStegTool.py and the same message being extracted with messageExtract.py
2. Show a screenshot of your original image (with no message) and your output image (with a message). Do the images look the same?
3. Compute 00110011 AND 11101010
4. How would you change the code in myStegTool.py and messageExtract.py to hide the message in the green channel of the pixels instead?
5. What is the least significant bit in the byte 11101010?