



white-cheeked-turaco-9024880_1280(1).jpg



white-cheeked-turaco-9024880_1280(2).jpg

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C
00000000 FF D8 FF E0 00 10 4A 46 49 46 00 01 01
```

I opened up the image in a hex editor to check out its raw data. Since it's a JPEG, the first two bytes are FF D8, which is the standard signature for JPEGs. Right after that, there's FF E0, which starts the APP0 marker; this is where the image's metadata is stored. I also noticed the 4A 46 49 46 part, which spells out "JFIF," showing that this image follows the JFIF (JPEG File Interchange Format) standard. The next few bytes, 00 01 01, tell me it's version 1.01 of that standard. From here, the file's data would continue with more metadata and the actual image data.

```
0001D170 18 8A 2D F9 89 05 F0 7E A3 1B 70 FF 00 11 29 F7 .S-ù%.8~f.pÿ..)÷
0001D180 3F FF D9 20 48 61 70 70 79 54 75 65 73 64 61 79 ?ÿÛ HappyTuesday
0001D190 21 !
```



The two images uploaded were analyzed to determine any difference in size. Image 1 had a file size of 119,185 bytes, while Image 2 measured 119,171 bytes, resulting in a minimal difference of 14 bytes. This small variation is unlikely to have a significant impact on the visual content of the images. The difference may be attributed to minor compression or metadata variations rather than any substantial changes in the images themselves. Given the negligible difference in file size, it is expected that both images will appear identical when viewed, with no observable discrepancies.

```
C:\Users\Fmahm\Downloads> certutil -hashfile white-cheeked-turaco-9024880_1280.jpg
SHA1 hash of white-cheeked-turaco-9024880_1280.jpg:
f4290347cadd42732b565d752bbe72b642bd3243
CertUtil: -hashfile command completed successfully.
```

I ran the certutil command to get the SHA-1 hash of the file. The command I used was certutil -hashfile and it gave me the hash f4290347cadd42732b565d752bbe72b642bd3243. This hash is like a unique fingerprint for the file, meaning if even the smallest thing in the file changes, the hash would be completely different. The message Certutil hashfile command completed successfully just confirms everything worked as it should.

```
C:\Users\Fmahm\Downloads> certutil -hashfile white-cheeked-turaco-9024880_1280(1).jpg
SHA1 hash of white-cheeked-turaco-9024880_1280(1).jpg:
837a5d801a4c1f630e8b5bfc22ff08a542a41dfa
CertUtil: -hashfile command completed successfully.
```

I ran the certutil command to get the SHA-1 hash for the file. The command I used was certutil -hashfile white-cheeked-turaco-9024880_1280(1).jpg, and it gave me the hash 837a5d801a4c1f630e8b5bfc22ff08a542a41dfa. This hash is like a unique fingerprint for the file, and if anything in the file changes, the hash will be different. The message at the end says "CertUtil: -hashfile command completed successfully," meaning everything worked fine. The new hash is different from the previous one because I made changes to the file, adding "HappyTuesday." The first image was 119,185 bytes, and the second one was a bit smaller at 119,171 bytes, with just a 14-byte difference. This small difference probably doesn't mean there's any noticeable change in how the images look. It's likely just due to small things like compression or metadata that you wouldn't be able to see. So, overall, both images are basically the same in terms of what's in them and their size.

```
C:\Users\Fmahm\Downloads>FORFILES /M *.jpg /C "cmd /c echo @FILE"

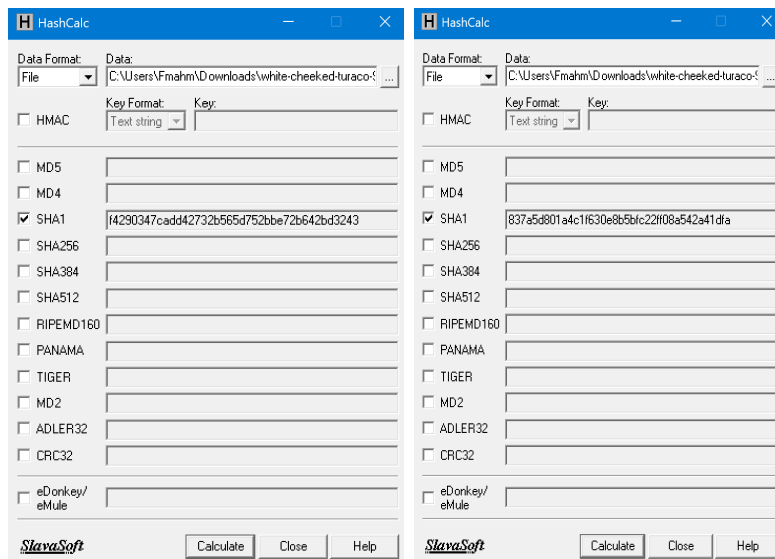
"rotterdam-7095262_1280.jpg"
"white-cheeked-turaco-9024880_1280(1).jpg"
"white-cheeked-turaco-9024880_1280.jpg"
```

I used the FORFILES command in the Command Prompt to list all the *.jpg files in my Downloads folder. The command I ran was FORFILES /M *.jpg /C "cmd /c echo @FILE", which searches for files with a .jpg extension and displays their names. The output shows that there are three .jpg files in the directory. This method is a quick way to identify specific file types without having to navigate through the folder manually.

```
C:\Users\Fmahm\Downloads> FORFILES /M *.jpg /C "cmd /c certutil -hashfile @FILE"

SHA1 hash of rotterdam-7095262_1280.jpg:
fc0fc6878ebf5ba94ac48e4351b9d6a8b4bb9a84
CertUtil: -hashfile command completed successfully.
SHA1 hash of white-cheeked-turaco-9024880_1280(1).jpg:
837a5d801a4c1f630e8b5bfc22ff08a542a41dfa
CertUtil: -hashfile command completed successfully.
SHA1 hash of white-cheeked-turaco-9024880_1280.jpg:
f4290347cadd42732b565d752bbe72b642bd3243
CertUtil: -hashfile command completed successfully.
```

I used the FORFILES command combined with certutil to calculate the SHA-1 hashes for all .jpg files in my Downloads folder. The command I used was FORFILES /M *.jpg /C "cmd /c certutil -hashfile @FILE". This command goes through each .jpg file in the directory and generates its SHA-1 hash. The output shows the SHA-1 hash for each file, followed by a confirmation that the certutil command ran successfully. The command calculated a unique hash for each image file, indicating their content. If any of these files were altered in any way, their hashes would change. This process helps in verifying file integrity by ensuring that the files haven't been modified.



In these screenshots, I used a program called HashCalc to compute the SHA-1 hashes for two different versions of a file. On the left side, I calculated the hash for one version of the file, which produced the SHA-1 hash. On the right side, I calculated the hash for a different version of the same file, and it generated a different SHA-1 hash. HashCalc lets me select different hashing algorithms, but I specifically used SHA-1 in this case. The fact that the hashes are different tells me

that some change was made to the file, as even a small modification results in a completely different hash. This comparison helps confirm if a file has been altered or remains unchanged.

```
PS C:\windows\system32> SET-LOCATION -PATH C:\Users\Fmahm\Downloads
PS C:\Users\Fmahm\Downloads> GET-FILEHASH -Algorithm SHA1 *.jpg

Algorithm      Hash
-----
SHA1           FC0FC6878EBF5BA94AC48E4351B9D6A8B48B9A84
SHA1           837A5D801A4C1F630E8B5BFC22FF08A542A41DFA
SHA1           F4290347CADD42732B565D752BBE72B642BD3243
```

Path
C:\Users\Fmahm\Downloads\rotterdam-7095262_1280.jpg
C:\Users\Fmahm\Downloads\white-cheeked-turaco-9024880_1280(1).jpg
C:\Users\Fmahm\Downloads\white-cheeked-turaco-9024880_1280.jpg

In this screenshot, I used PowerShell to generate SHA-1 hashes for all the .jpg files in my Downloads folder. The output displays the hash for each file along with its path, which serves as a unique identifier for the file's content. By comparing these hashes, I can easily tell if any of the files have been modified since even a small change would result in a completely different hash. This method provides a straightforward way to verify the integrity of the files in the folder.

```
(C:\Users\Fmahm\anaconda3) C:\Windows\System32>conda update anaconda
Retrieving notices: ...working... done
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

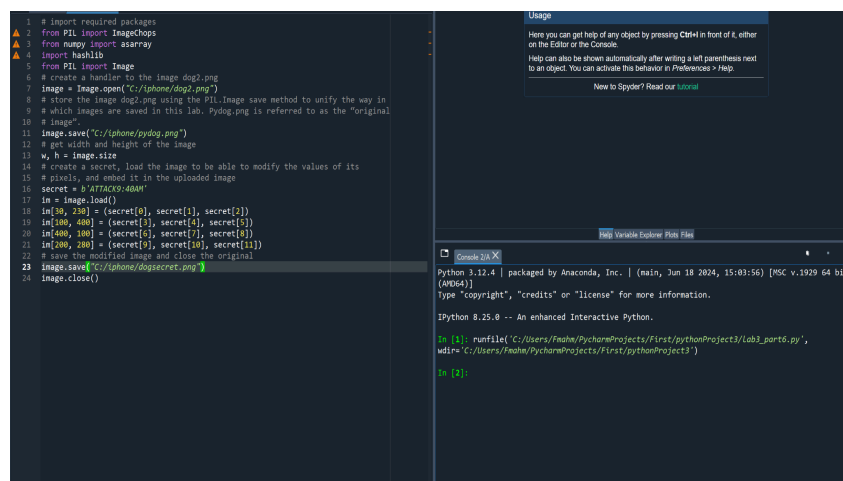
# All requested packages already installed.
```

I ran a command to update Anaconda using `conda update anaconda`. The system first retrieved necessary information and checked the available channels and platform. It then collected package metadata and solved the environment to figure out what needed to be updated. In the end, the message All requested packages already installed appeared.

```
1 import hashlib
2
3 f1 = open("C:/Users/Fmahm/Downloads/white-cheeked-turaco-9024880_1280.jpg", 'rb')
4 f2 = open("C:/Users/Fmahm/Downloads/white-cheeked-turaco-9024880_1280(1).jpg", 'rb')
5
6 data1 = f1.read()
7 data2 = f2.read()
8
9 print(hashlib.sha1(data1).hexdigest())
10 print(hashlib.sha1(data2).hexdigest())
```

C:\Users\Fmahm\PycharmProjects\First\pythonProject3\.venv\Scripts\python.exe C:\Users\Fmahm\PycharmProjects\First\pythonProject3\lab3_part3.py
f4290347cadd42732b565d752bbe72b642bd3243
837a5d801a4c1f630e8b5bfc22ff08a542a41dfa

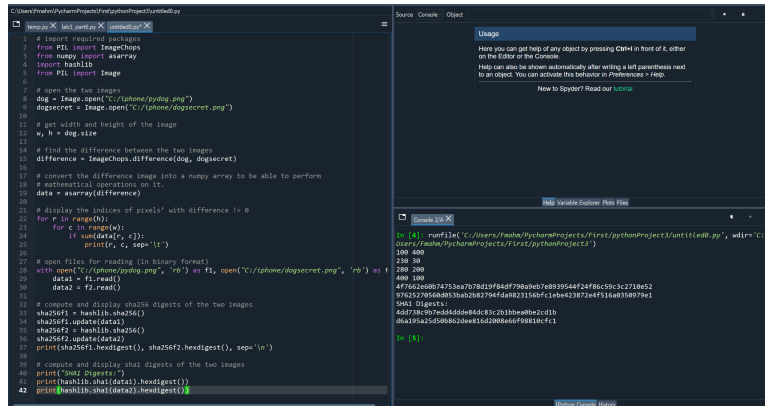
In this screenshot, I wrote a Python script using the hashlib library to calculate the SHA-1 hash of two image files. The script starts by importing hashlib and then opens two files in binary read mode. The first file is opened as f1, and the second one as f2. I read the contents of both files into data1 and data2 using the read method. Then, I use hashlib.sha1 to compute the SHA-1 hash for each file's data. The hexdigest method converts the hash to a readable hexadecimal string. Finally, the script prints the SHA-1 hashes for both files. In the output at the bottom, you can see the two different hash values, one for each file. Since the hashes are different, it confirms that the files are not identical, which is expected since one of them was modified, like adding HappyTuesday earlier.



A Python script was given to us that embeds a hidden message into an image. The script starts by importing necessary libraries, such as ImageChops and Image from the Python Imaging Library (PIL) and hashlib. I then open an image file and get its size, storing it for later use. The script defines a secret message as a byte string, which is then

embedded into specific pixels of the image. The coordinates [30, 230], [100, 400], [400, 100], and [200, 280] are modified to contain parts of the secret message. This alters the image in a

subtle way, hiding the secret within it. After embedding the secret data, I save the modified image as a new file named dogsecret.png to avoid overwriting the original. Finally, the script closes the image file. In the console on the right, you can see that the script was successfully run, confirming that the changes were applied to the image. This technique is a basic form of steganography, which involves hiding information within media files.

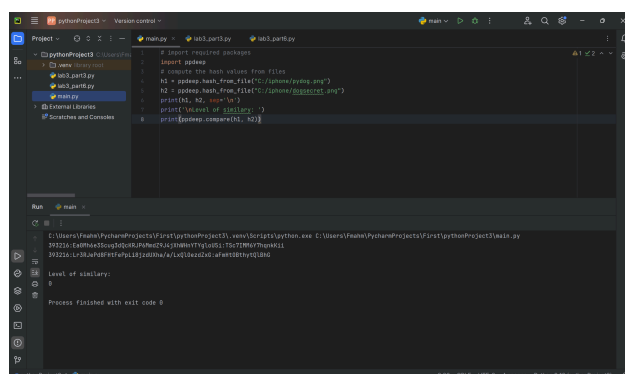


```
1 # Import required packages
2 from PIL import ImageChops
3 from numpy import asarray
4 import hashlib
5 from PIL import Image
6
7 # Open the two images
8 dog = Image.open('C:/phone/pydog.png')
9 dogsecret = Image.open('C:/phone/dogsecret.png')
10
11 # Get width and height of the image
12 w, h = dog.size
13
14 # Find the difference between the two images
15 difference = ImageChops.difference(dog, dogsecret)
16
17 # Convert the difference image into a numpy array to be able to perform
18 # mathematical operations on it
19 data = asarray(difference)
20
21 # Display the indices of pixels with difference != 0
22 for r in range(h):
23     for c in range(w):
24         if not data[r, c] == 0:
25             print(r, c, sep='|')
26
27 # Open files for reading (in binary format)
28 with open('C:/phone/pydog.png', 'rb') as f1, open('C:/phone/dogsecret.png', 'rb') as f2:
29     data1 = f1.read()
30     data2 = f2.read()
31
32 # Compute and display SHA256 digests of the two images
33 sha256f1 = hashlib.sha256(data1)
34 sha256f2 = hashlib.sha256(data2)
35 sha256f1.update(data1)
36 sha256f2.update(data2)
37 print(sha256f1.hexdigest(), sha256f2.hexdigest(), sep='\n')
38
39 # Compute and display SHA1 digests of the two images
40 print('SHA1 Digests:')
41 print(hashlib.sha1(data1).hexdigest())
42 print(hashlib.sha1(data2).hexdigest())
```

```
Usage
Here you can get help of any object by pressing Ctrl+H in front of it, either
in the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next
to an object. You can activate this behavior in Preferences > Help.
How to Spyder? Read our tutorial
New Variable Explorer: Plot File
[0] [4] runfile('C:/Users/Flame/PycharmProjects/First/pythonProject3/untitled6.py', wdir='C:/
Users/Flame/PycharmProjects/First/pythonProject3')
300 400
230 30
400 100
476226d074723a7b78d19f84df79a9b768920544f24f8659c3c2718a52
9762527050a0b93ba2b82794fda02319bfc10b423872e4f510a939079e1
5041 f0d685c5
d40f38c8b7ed4dddb8dc83c2b1b0e0a2c21b
d0a19a25d50b8b2d0e13d200e0d4f083bfc1
[0] [4]
```

A Python script to compare two images and identify any differences. The script begins by importing necessary libraries, such as ImageChops and Image from PIL, asarray from NumPy, and hashlib for hash computations. I opened two images: one original and one modified. The script then calculates the difference between the two images using

ImageChops.difference. This difference image is then converted into a NumPy array to analyze pixel values and identify where the changes occurred. The script uses nested loops to go through each pixel and prints the coordinates of pixels where there is a difference. In the console, you can see the coordinates [100, 400], [230, 30], [200, 200], and [400, 100], which correspond to the locations where the secret was embedded in the previous script. Next, the script opens both images in binary read mode and reads their contents into data1 and data2. I then used hashlib to compute both the SHA-256 and SHA-1 hashes of the two images. The output in the console shows two different sets of SHA-256 and SHA-1 hashes, confirming that the images are not identical due to the embedded modifications. This script demonstrates how to detect changes in an image and verify those changes using hash comparisons.



```
1 # Import required packages
2 import ppdeep
3
4 # Compute the hash values from files
5 h1 = ppdeep.hash_from_file('C:/phone/pydog.png')
6 h2 = ppdeep.hash_from_file('C:/phone/dogsecret.png')
7 print(h1, h2, sep='\n')
8 print('Secret of similarity: ')
9 print(ppdeep.compare(h1, h2))
```

```
0
Level of similarity:
0
Process finished with exit code 0
```

I used the ppdeep library in Python to compare the similarity between two image files. The script starts by importing ppdeep and then uses the hash_from_file method to generate fuzzy hashes for both images: the original and the modified one with the secret embedded. The hashes are stored in variables h1 and h2, and the script prints these hashes for reference. The next step is to calculate the level of similarity between the two files using ppdeep.compare(h1, h2). This

method is useful for detecting minor changes between files since fuzzy hashing is designed to find similarities rather than exact matches. In the output, the similarity level is shown as 0, indicating that the two files have no noticeable similarity according to the fuzzy hashing method. This makes sense given that the hidden data embedded in the modified image creates a significant difference from the original.