FLEMING

COMP 593
Scripting Applications

Lecture 6:

File System Management and Hashes

Objectives

- Use the **os.path** module to construct and test file and directory paths
- Use the **os** module to manipulate files and directories
- Explain the terms hash function and hash value
- Discuss some applications of cryptographic hash functions
- Use the **hashlib** module to calculate the hash value of any binary data item

File System Management

File System Management

- The os.path module provides various functions for working with file and directory paths, e.g.,
  - Building a path from a directory path and a file name
  - Determining whether a file or directory exists
  - Determining whether a path is an existing file or a directory
  - Determining whether a path is absolute or relative
- The pathlib module provides similar functionality
  - But uses an object-oriented approach, rather than treating paths as strings

Example Uses of the os.path Module

- Build a path from individual directory and file paths/names

```python
import os

parent_path = r'C:\Fleming'
child_dir = 'COMP593'
file_name = 'homework.txt'

# Builds full path of file: 'C:\Fleming\COMP593\homework.txt'
file_path = os.path.join(parent_path, child_dir, file_name)

# Builds full path of child directory: 'C:\Fleming\COMP593'
child_path = os.path.join(parent_path, child_dir)
```

Example Uses of the os.path Module

- Determine whether a directory or file exists

```
import os

file_path = r'C:\Fleming\COMP593\homework.txt'

if os.path.exists(file_path):
    print(file_path, "exists.")

if os.path.isdir(file_path):
    print(file_path, "is an existing directory")

if os.path.isfile(file_path):
    print(file_path, "is an existing file.")
```

Example Uses of the os.path Module

- Determine whether a path is relative or absolute

```python
import os

file_path = 'homework.txt'
if not os.path.isabs(file_path):
    # Build absolute path using CWD
    file_path = os.path.abspath(file_path)

print(file_path)
```

```
PS C:\users\bob\Desktop> python C:\temp\script.py
C:\users\bob\Desktop\homework.txt
```

Working with Files and Directories

- The os module provides various functions for working with files and directories, e.g.,
    - Creating, deleting, and renaming files and directories
    - Getting directory listings
- The pathlib module provides similar functionality
    - But uses an object-oriented approach, rather than treating paths as strings
    - i.e., Use a **Path** object and its class methods to manipulate file/directory

Example Uses of the os Module

- Create a directory

```
import os

dir_path = r'C:\Fleming\COMP593'

if not os.path.exists(dir_path):
    # Makes only the COMP593 directory
    os.mkdir(dir_path)

if not os.path.exists(dir_path):
    # Makes COMP593 directory and Fleming directory, if needed
    os.makedirs(dir_path)
```

Example Uses of the os Module

- Rename and delete

```python
import os

old_dir = r'C:\Fleming'
new_dir = r'C:\College'
file_name = 'homework.txt'

# Rename a directory
if os.path.exists(old_dir) and os.path.isdir(old_dir):
    os.rename(old_dir, new_dir)

# Delete a file
file_path = os.path.join(new_dir, file_name)
if os.path.exists(file_path) and os.path.isfile(file_path):
    os.remove(file_path)
```

Example Uses of the os Module

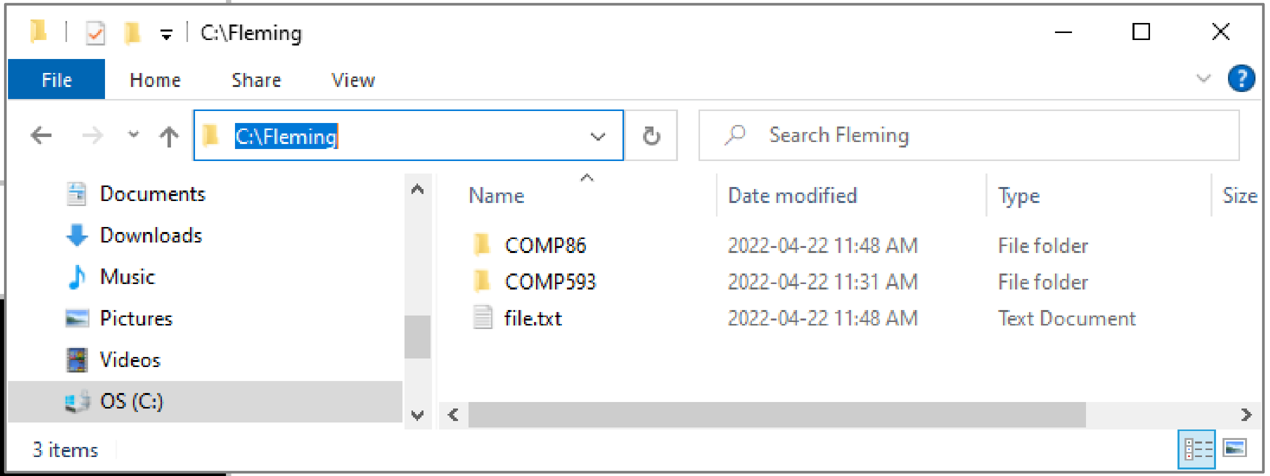
- Get directory listing

```
import os

dir_path = r'C:\Fleming'
dir_list = os.scandir(dir_path)

for item in dir_list:
    print(item.path)
```

```
C:\Fleming\COMP593
C:\Fleming\COMP86
C:\Fleming\file.txt
```



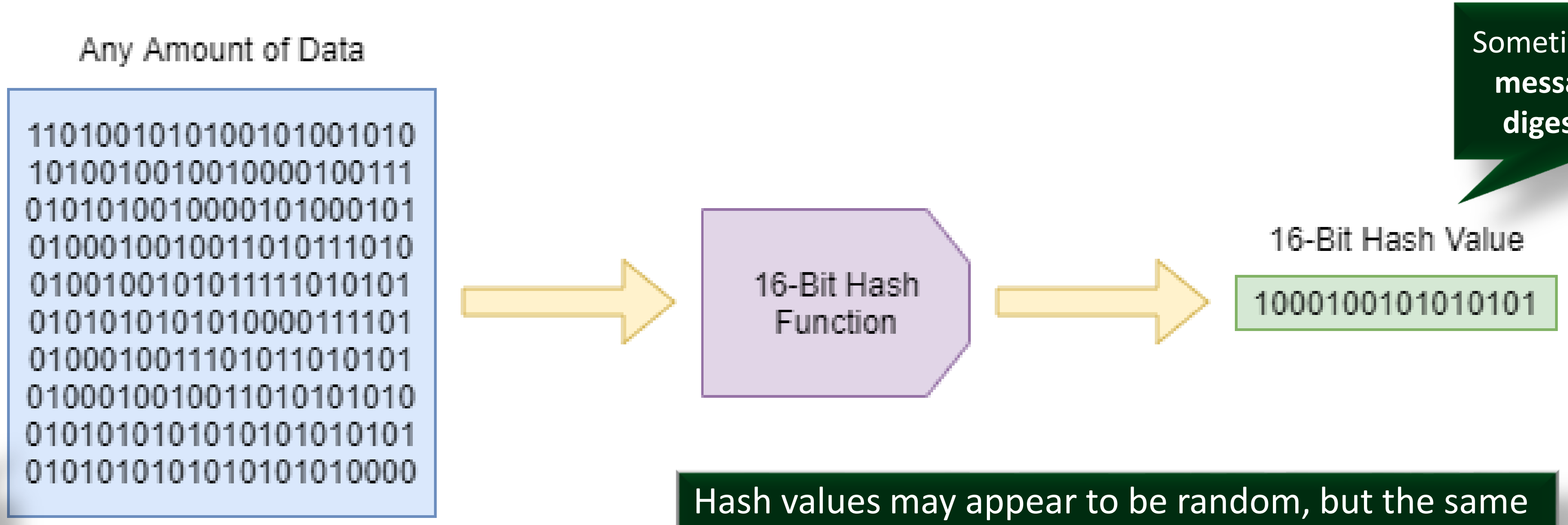
Name	Date modified	Type	Size
COMP86	2022-04-22 11:48 AM	File folder	
COMP593	2022-04-22 11:31 AM	File folder	
file.txt	2022-04-22 11:48 AM	Text Document	

3 items

Hashes

Hash Functions

- A hash function takes an input of any length and produces a hash value of some fixed length



```
1101001010100101001010
1010010010010000100111
0101010010000101000101
0100010010011010111010
0100100101011111010101
0101010101010000111101
0100010011101011010101
0100010010011010101010
0101010101010101010101
0101010101010101010000
```

Sometimes called a **message digest**, **digest**, or **hash**

Sometimes called a **message**

Hash values may appear to be random, but the same input data will always produce the same hash value

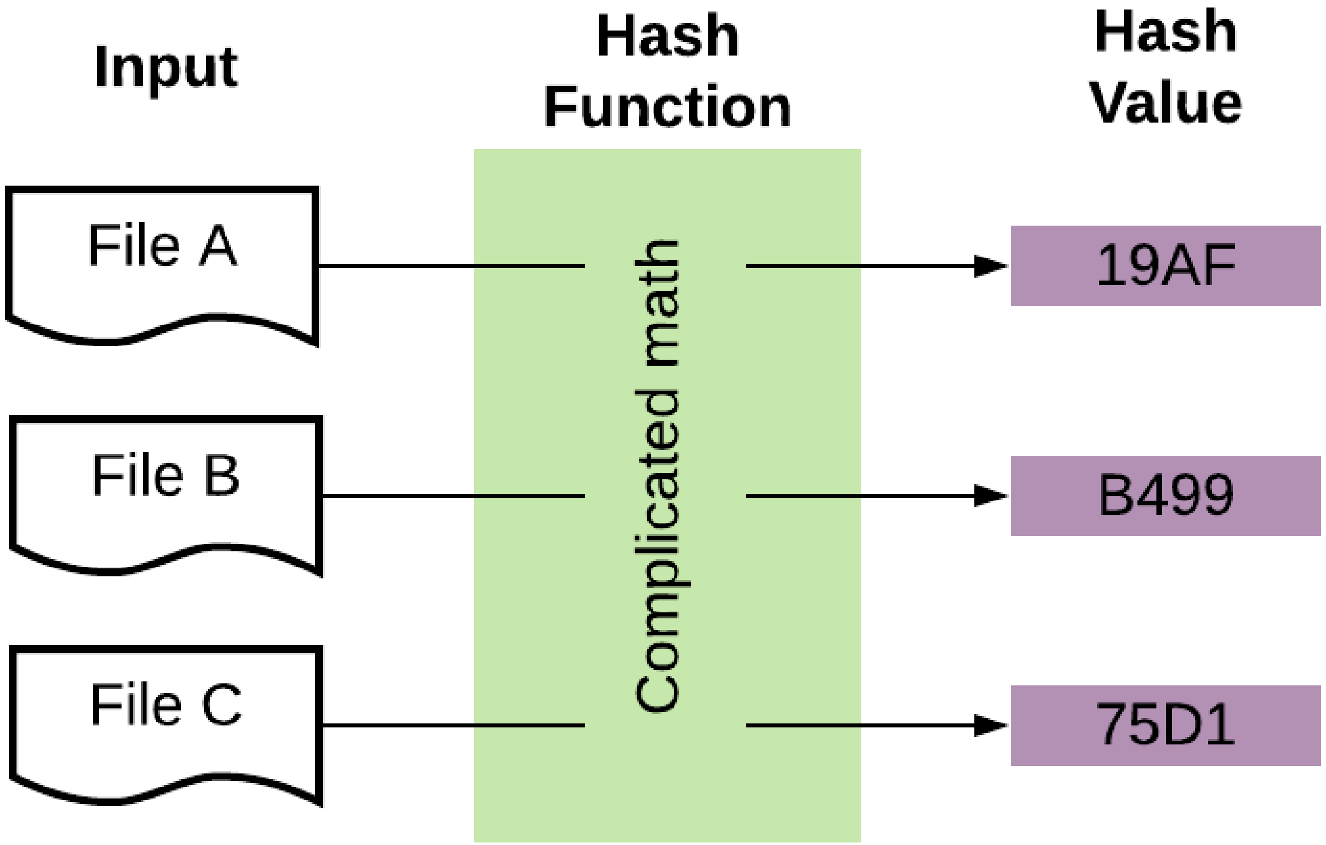
Hash Functions

- Various different hash functions have been devised for different purposes:
  - Checksum
  - Cyclical Redundancy Check
  - Message-Digest Algorithms (MD)
  - Secure Hash Algorithms (SHA)
- e.g., SHA-256 always produces a hash value that is 256-bits long and is good for verifying file integrity (among other things)

Hash Function

Example:

16-Bit Hash Function



```python
def main():
    s = 'Sparse is better than dense.'
    cs = calc_8bit_checksum(s)
    print(f'The checksum is 0x{cs:2X}')

def calc_8bit_checksum(string):
    """Calculates the 8-bit checksum of a string.
    Args:
        string (str): The string
    Returns:
        int: 8-bit checksum
    """
    # Sum ASCII values of each character
    checksum = 0
    for char in string:
        checksum += ord(char)
    # Remove all but the least-significant 8 bits
    return checksum % 0x100
```

Formats integer to be printed as an uppercase hexadecimal value

```
The checksum is 0x38
```

ord() is a built-in function that converts a character to its integer representation

```python
def main():
    s = 'Sparse is better than dense.'
    cs = calc_checksum(s, 16)
    print(f'The checksum is 0x{cs:0>4x}')

def calc_checksum(string, bits=8):
    """ Calculates the checksum of a string.
    Args:
        string (str): The string
        bits (int, optional): Bit size of checksum. Defaults to 8.
    Returns:
        int: Checksum
    """
    return sum(string.encode()) % pow(2, bits)
```

Formats integer to be printed as a 4-digit lowercase hexadecimal value with leading zeros

```
The checksum is 0x0a38
```

encode() is a str class method that returns an encoded version of the string as a bytes object, which is an immutable sequence of bytes.

sum() and pow() are built-in functions

Cryptographic Hash Functions

- Always compute the same hash value for the same message
  - i.e., Must be deterministic
- Should have the following properties:
  - Quick to compute the hash value for any given message
  - Infeasible to determine message from hash value
  - Infeasible that two different messages have the same hash value
  - Small change to a message extensively changes the hash value

https://en.wikipedia.org/wiki/Cryptographic_hash_function

Checksum

- Checksum is not a cryptographic hash function
  - ✓ Deterministic and quick to compute
  - ✗ *Somewhat* infeasible to determine message from hash
  - ✗ Likely to get same checksum for different messages, e.g.,
    - Order of values in message does not affect checksum
    - Checksum is unaffected if one value in the message is decreased by x and another value in the message is increased by x
  - ✗ Small change to message barely changes hash value, e.g.,
    - Increase value in message by 1 increases checksum by 1
- Useful for detecting errors in simple communication protocols

Cryptographic Hash Functions

- Algorithms for computing hash values are much more complicated than the checksum algorithm
  - Tend to incorporate lots of bit shifting, rotating, and logic operations
    - i.e., Operations that computers can do very quickly
  - Designed by academics > research papers written > mathematically proven
  - Algorithm details can be found online, if you're interested
- Usually no need to understand the algorithm details
  - Use pre-developed library functions to compute hash values
  - Trust that the experts have sorted out the details

Applications of Hashes

- Cryptographic hash functions are used extensively in computer science and computer security, e.g.,
  - Determine whether two files are identical
  - Determine whether a file has been modified
  - Detect malware files
  - Store user passwords
  - Verify whether a communication message is received correctly
  - Verify the integrity of a downloaded file

hashlib Module

- The hashlib module provides a common interface to many different cryptographic hashing algorithms
  - e.g., SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, MD5
  - Included in the Python standard library

```python
import hashlib
s = b'Sparse is better than dense.'
hash_value = hashlib.md5(s).hexdigest()
print(f'The MD-5 hash value is {hash_value}')
```

Prefixing a string literal with **b** makes it a bytes object, so it does not need to be encoded to compute its hash.

```
The MD-5 hash value is 55541308e54561a02dcf1a66a2f496c9
```

```python
# Example: Compute SHA-256 of a small file
import hashlib

def main():
    print(calc_file_sha256(r'C:\temp\gateway.log'))

def calc_file_sha256(file_path):
    # Open file for reading binary data
    with open(file_path, 'rb') as file:
        # Read entire file contents
        file_data = file.read()
        # Compute hash of file contents
        hash_obj = hashlib.sha256(file_data)
        # Return hash value as hexadecimal string
        return hash_obj.hexdigest()
```

Problem: For a large file, the **file_data** object would occupy a lot of memory

```
79879138622f2f9ebaed6df54f323874952bc279d43788d718fd225269d15802
```

```python
# Example: Compute SHA-256 of a file
import hashlib

def main():
    print(calc_file_sha256(r'C:\temp\gateway.log'))

def calc_file_sha256_ok(file_path):
    # Create hash object
    hash_obj = hashlib.sha256()
    with open(file_path, 'rb') as file:
        # Read file contents line by line
        for line in file:
            # Add the current line to the hash computation
            hash_obj.update(line)
    return hash_obj.hexdigest()
```

This function will work correctly, but efficiency can be improved by eliminating some buffering and reading the file in larger chunks.

(See next slide)

```
79879138622f2f9ebaed6df54f323874952bc279d43788d718fd225269d15802
```

```python
# Example: Compute SHA-256 of a file
import hashlib

def main():
    print(calc_file_sha256(r'C:\temp\gateway.log'))

def calc_file_sha256(file_path):
    hash_obj  = hashlib.sha256()
    read_buffer  = bytearray(128*1024)
    read_buffer_ref = memoryview(read_buffer)
    with open(file_path, 'rb', buffering=0) as f:
        while n := f.readinto(read_buffer_ref):
            hash_obj.update(read_buffer_ref[:n])
    return hash_obj.hexdigest()
```

If you want to understand how this function works, read about bytearray, memoryview, readinto() and assignment expressions.

```
79879138622f2f9ebaed6df54f323874952bc279d43788d718fd225269d15802
```

https://stackoverflow.com/questions/22058048/hashing-a-file-in-python

```python
# Example: Compute SHA-1 of HTTP response message content
import hashlib
image_url = 'https://gvanrossum.github.io/images/DO6GvRhi.gif'

# Send GET request for image
resp = requests.get(image_url)

# Check if GET request was successful
if resp.ok:

    # Extract response message content as bytes object
    image_data = resp.content

    # Calculate SHA-1 hash value
    image_hash = hashlib.sha1(image_data).hexdigest()
    print(image_hash)
```

```
d03eecfe8773cb2b00799ca8e68a7ebca57c7422
```

After downloading this image file, right click and choose **Set as desktop background**