AI ASSISTED CODING

SUMANTH POLAM **2303A51121**

BATCH – 03 **13 – 02 – 2026**

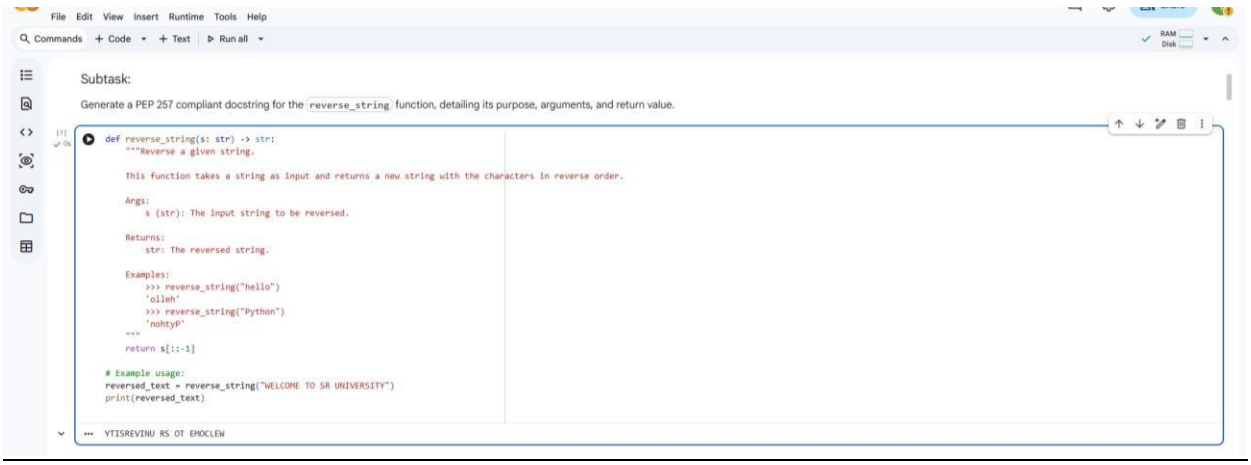
ASSIGNMENT – 9.5

Lab 9.5: Documentation Generation -Automatic documentation and code comments

Task1: String Utilities Function

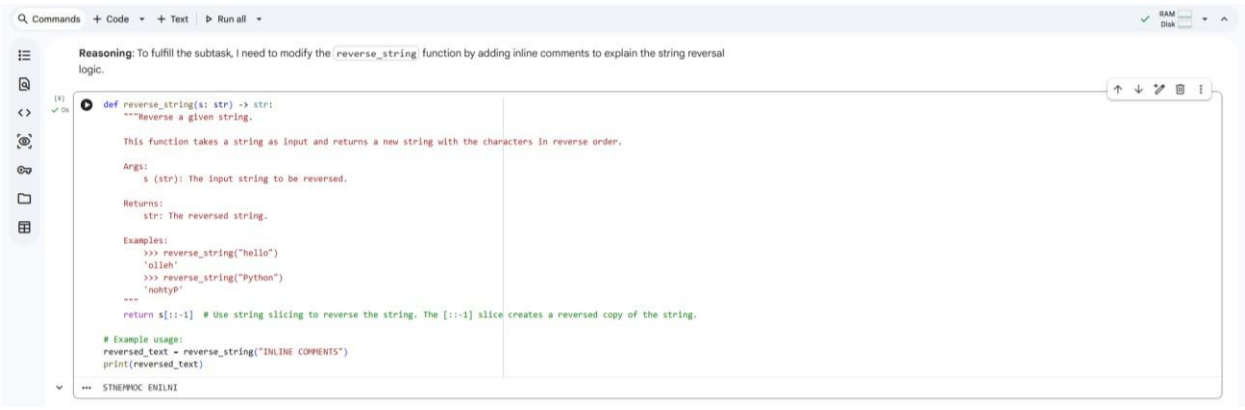
Prompt 1: Generate a PEP 257 compliant docstring for the reverse_string function, detailing its purpose, arguments, and return value.

CODE & OUTPUT:



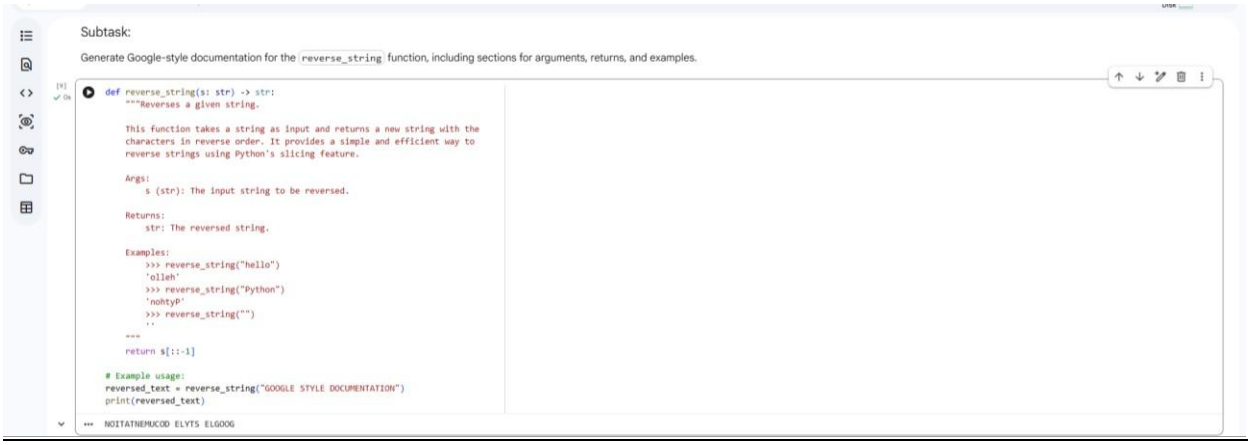
Prompt 2: Add appropriate inline comments to the reverse_string function to explain each line or logical block of code.

CODE & OUTPUT:



Prompt 3: Generate Google-style documentation for the reverse_string function, including sections for arguments, returns, and examples.

CODE & OUTPUT:



Comparison:

Documentation Style	Clarity	Structure	Suitability for Security Code	Limitations
Inline Comments	Basic explanation of logic.	No fixed structure	✗ Not suitable for critical security functions	Cannot clearly explain assumptions, limitations, or warnings.
Standard Docstring (PEP 257)	Clear description of parameters and return values.	Moderate structure	✓ Suitable for small security functions	May not separate security notes clearly.
Google-Style Documentation	Very clear and detailed	Highly structured (Args, Returns, Notes)	✓✓ Most suitable for security-related code	Slightly longer to write.

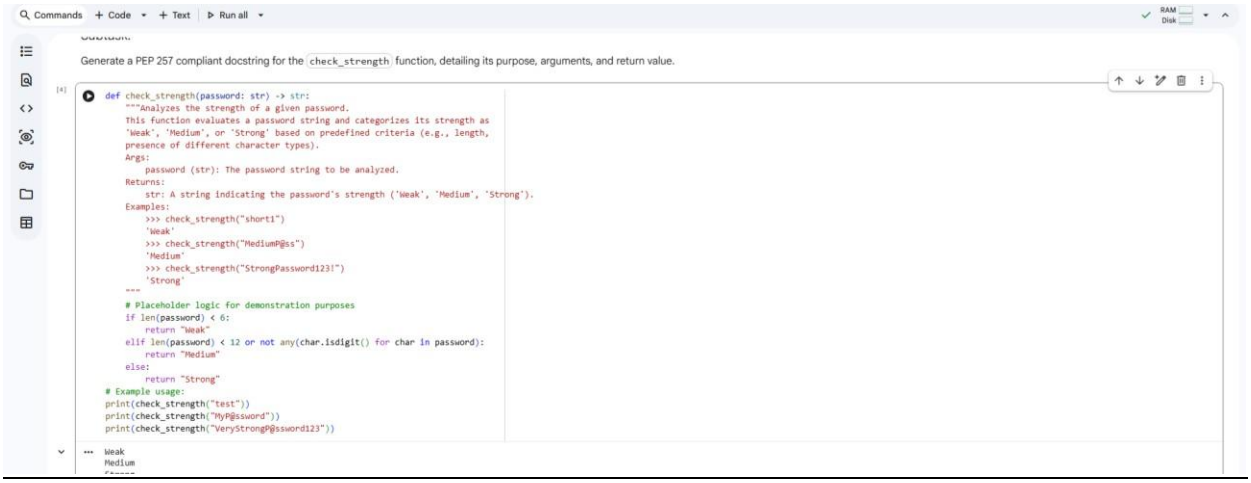
JUSTIFICATION FOR BEST DOCUMENT STYLE:

Google-style documentation is the most appropriate for security-related code because it provides a clear and structured format using sections like Args, Returns, and Notes.

TASK 2: Password Strength Checker

Prompt 1: Generate a PEP 257 compliant docstring for the check_strength function, detailing its purpose, arguments, and return value.

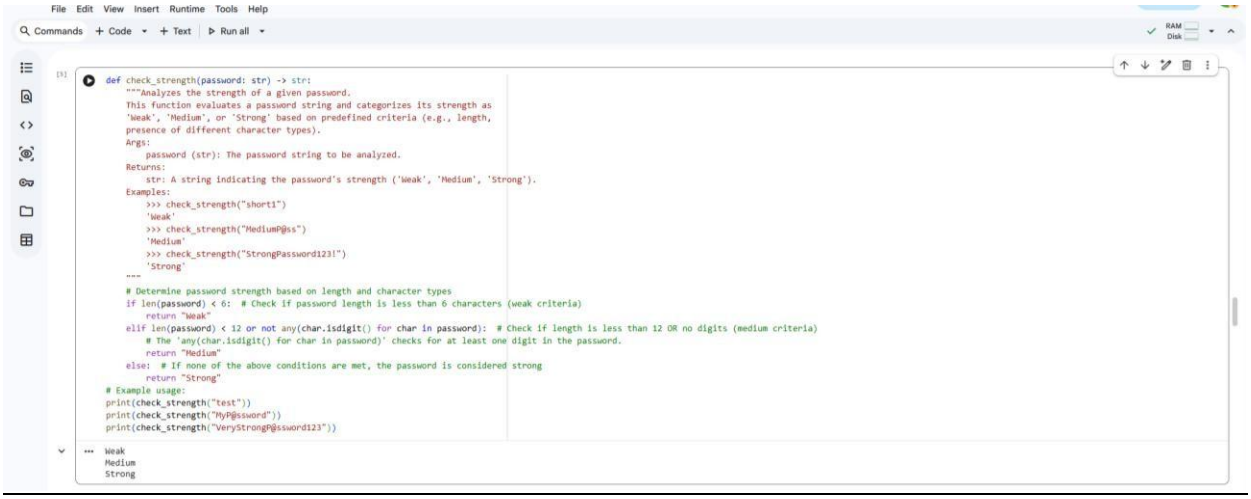
CODE & OUTPUT:



```
Generate a PEP 257 compliant docstring for the check_strength function, detailing its purpose, arguments, and return value.

def check_strength(password: str) -> str:
    """Analyzes the strength of a given password.
    This function evaluates a password string and categorizes its strength as
    'Weak', 'Medium', or 'Strong' based on predefined criteria (e.g., length,
    presence of different character types).
    Args:
        password (str): The password string to be analyzed.
    Returns:
        str: A string indicating the password's strength ('Weak', 'Medium', 'Strong').
    Examples:
        >>> check_strength("short1")
        'Weak'
        >>> check_strength("MediumP@ss")
        'Medium'
        >>> check_strength("StrongPassword123!")
        'Strong'
    """
    # Placeholder logic for demonstration purposes
    if len(password) < 6:
        return "Weak"
    elif len(password) < 12 or not any(char.isdigit() for char in password):
        return "Medium"
    else:
        return "Strong"
# Example usage:
print(check_strength("test"))
print(check_strength("MyP@ssword"))
print(check_strength("VeryStrongP@ssword123"))

Weak
Medium
```

Prompt 2: Add appropriate inline comments to the check_strength function to explain each line or logical block of code.

CODE & OUTPUT:

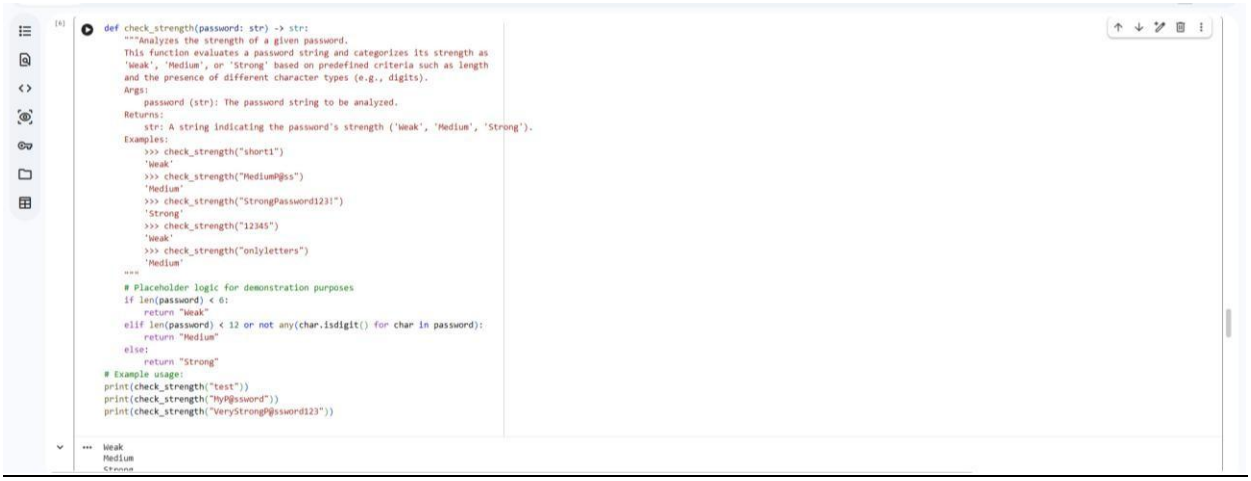


```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

def check_strength(password: str) -> str:
    """Analyzes the strength of a given password.
    This function evaluates a password string and categorizes its strength as
    'Weak', 'Medium', or 'Strong' based on predefined criteria (e.g., length,
    presence of different character types).
    Args:
        password (str): The password string to be analyzed.
    Returns:
        str: A string indicating the password's strength ('Weak', 'Medium', 'Strong').
    Examples:
        >>> check_strength("short1")
        'Weak'
        >>> check_strength("MediumP@ss")
        'Medium'
        >>> check_strength("StrongPassword123!")
        'Strong'
    """
    # Determine password strength based on length and character types
    if len(password) < 6:  # Check if password length is less than 6 characters (weak criteria)
        return "Weak"
    elif len(password) < 12 or not any(char.isdigit() for char in password):  # Check if length is less than 12 OR no digits (medium criteria)
        # The 'any(char.isdigit() for char in password)' checks for at least one digit in the password.
        return "Medium"
    else:  # If none of the above conditions are met, the password is considered strong
        return "Strong"
# Example usage:
print(check_strength("test"))
print(check_strength("MyP@ssword"))
print(check_strength("VeryStrongP@ssword123"))

Weak
Medium
Strong
```

Prompt 3: Generate Google-style documentation for

the check_strength function, including sections for arguments, returns, and examples.

CODE & OUTPUT:



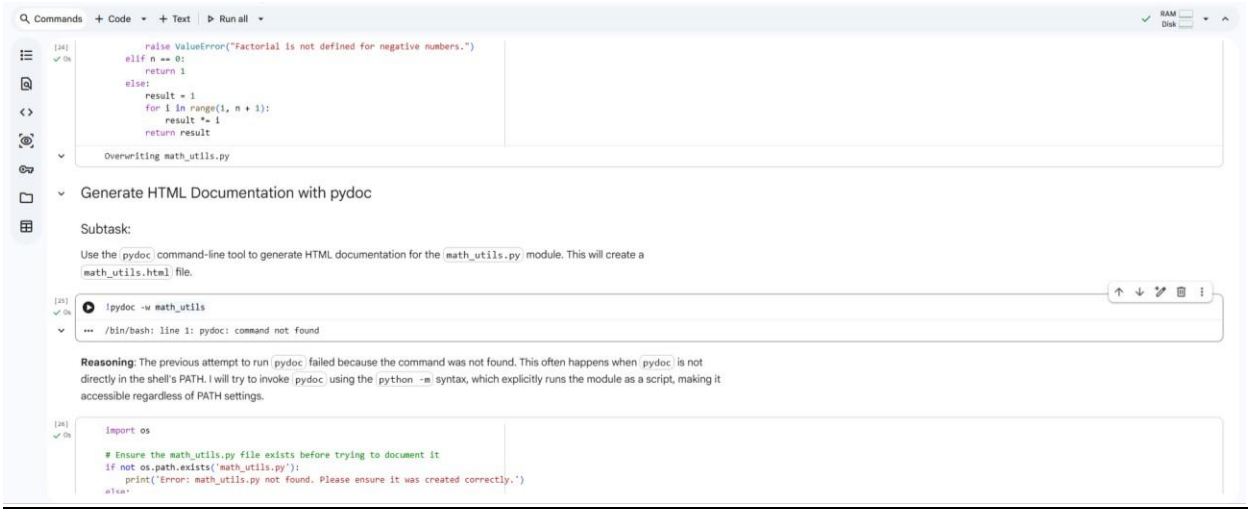
```
def check_strength(password: str) -> str:
    """Analyzes the strength of a given password.
    This function evaluates a password string and categorizes its strength as
    'Weak', 'Medium', or 'Strong' based on predefined criteria such as length
    and the presence of different character types (e.g., digits).
    Args:
        password (str): The password string to be analyzed.
    Returns:
        str: A string indicating the password's strength ('Weak', 'Medium', 'Strong').
    Examples:
        >>> check_strength("short1")
        'Weak'
        >>> check_strength("MediumP@ss")
        'Medium'
        >>> check_strength("StrongPassword123!")
        'Strong'
        >>> check_strength("12345")
        'Weak'
        >>> check_strength("onlyletters")
        'Medium'
    """
    # Placeholder logic for demonstration purposes
    if len(password) < 6:
        return "Weak"
    elif len(password) < 12 or not any(char.isdigit() for char in password):
        return "Medium"
    else:
        return "Strong"
# Example usage:
print(check_strength("test"))
print(check_strength("MyP@ssword"))
print(check_strength("VeryStrongP@ssword123"))
```

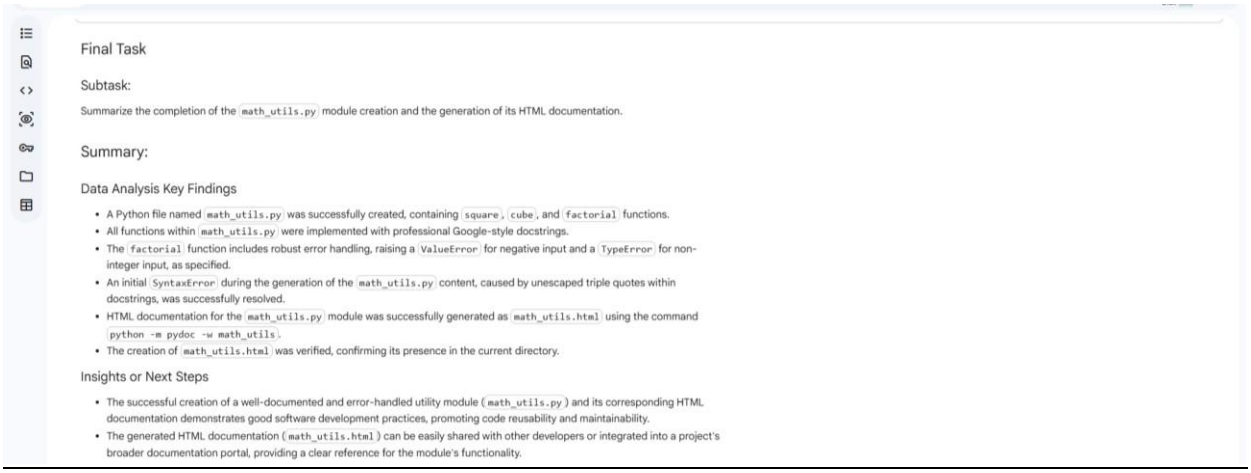
```
Weak
Medium
Strong
```

TASK 3: Math Utilities Module

Prompt : Generate a complete Google Colab workflow for creating a Python module called math_utils.py with square, cube, and factorial functions including proper docstrings. Also include the commands to save the file and generate HTML documentation using pydoc.

CODE & OUTPUT:

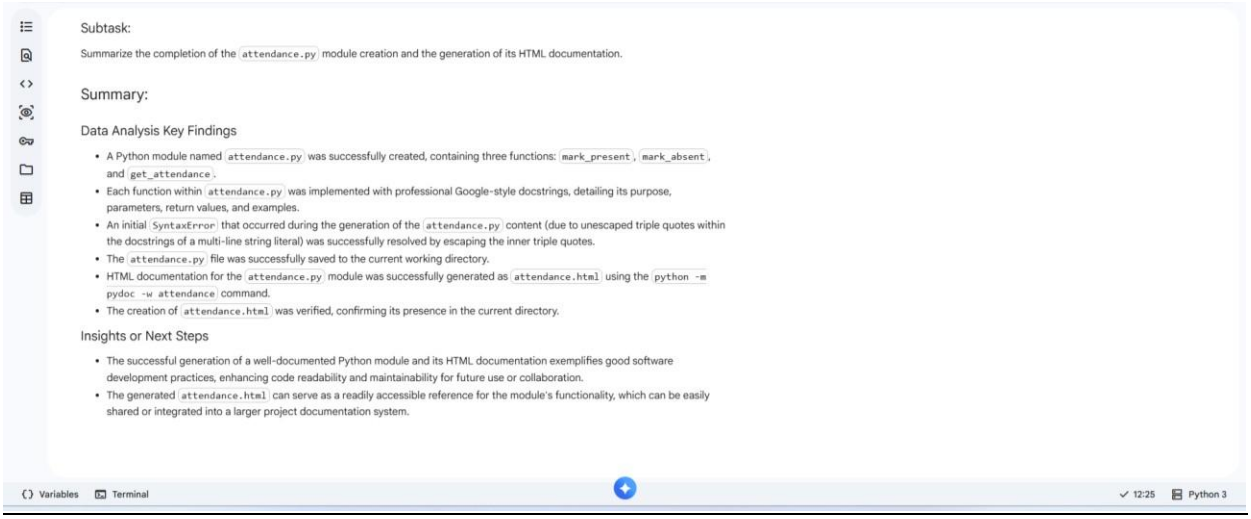




TASK 4: Attendance Management Module

Prompt : Generate a Python module named attendance.py with functions mark_present(student), mark_absent(student), and get_attendance(student) using a dictionary to store records, include professional docstrings explaining purpose, parameters and return values, and also provide Google Colab commands to save the file and generate HTML documentation using pydoc.

CODE & OUTPUT:



TASK 5: File Handling Function

Prompt 1: Generate a PEP 257 compliant docstring for the read_file function, detailing its purpose, arguments, return value, and clearly mentioning FileNotFoundError and IOError.

OUTPUT:

Prompt 2: Add appropriate inline comments to the read_file function to explain each line or logical block of code, including potential exception points.

CODE & OUTPUT:

```
# Test with a non-existent file path
print("\n--- Testing with non-existent file (inline comments) ---")
try:
    content = read_file('non_existent_path_inline.txt')
    print(f"Content of 'non_existent_path_inline.txt':\n{content}")
except (FileNotFoundError, IOError) as e:
    print(f"Error reading 'non_existent_path_inline.txt': {e}")

# Clean up the dummy file
if os.path.exists('example_file_inline.txt'):
    os.remove('example_file_inline.txt')
    print("\nCleaned up 'example_file_inline.txt'.")
```

```
--- Testing with valid file (inline comments) ---
Content of 'example_file_inline.txt':
This is a test file for inline comments.

--- Testing with non-existent file (inline comments) ---
Error reading 'non_existent_path_inline.txt': File not found: non_existent_path_inline.txt

Cleaned up 'example_file_inline.txt'.
```

Prompt 3: Generate Google-style documentation for the read_file function, including sections for arguments, returns, Raises (explicitly listing FileNotFoundError and IOError), and examples.

CODE & OUTPUT:

```
try:
    content = read_file('example_file_google.txt')
    print(f"Content of 'example_file_google.txt':\n{content}")
except (FileNotFoundError, IOError) as e:
    print(f"Error reading 'example_file_google.txt': {e}")

# Test with a non-existent file path
print("\n--- Testing with non-existent file (Google-style) ---")
try:
    content = read_file('non_existent_path_google.txt')
    print(f"Error reading 'non_existent_path_google.txt': {content}")
except (FileNotFoundError, IOError) as e:
    print(f"Error reading 'non_existent_path_google.txt': {e}")

# Clean up the dummy file
if os.path.exists('example_file_google.txt'):
    os.remove('example_file_google.txt')
    print("\nCleaned up 'example_file_google.txt'.")
```

```
--- Testing with valid file (Google-style) ---
Content of 'example_file_google.txt':
This is a test file for Google-style documentation.

--- Testing with non-existent file (Google-style) ---
Error reading 'non_existent_path_google.txt': File not found: non_existent_path_google.txt

Cleaned up 'example_file_google.txt'.
```

COMPARISON:

Documentation Style	Exception Explanation	Exception Handling Details		Structure
Inline Comments	Basic to moderate clarity	Errors mentioned briefly within code	🟡🟡🟡	Unstructured
Standard Docstring (PEP 257)	Moderate clarity with parameter sections	May mention common errors at the end	🟡🟡🟡	Moderately structured
Google-Style Documentation	High clarity with 'Raises' section	Clearly lists possible exceptions like FileNotFoundError, IOError	✅✅✅	Highly structured (Args, Returns, Raises)

RECOMMENDATION:

Google-style documentation is the most appropriate style for file handling functions because it clearly explains exception handling using a structured format. It provides separate sections such as Args, Returns, and Raises, which make it easy to understand possible errors like FileNotFoundError and IOError.

Since file operations are prone to runtime errors, clearly documenting exceptions improves code reliability, maintainability, and debugging. Therefore, Google-style documentation is recommended for explaining exception handling in file handling functions.