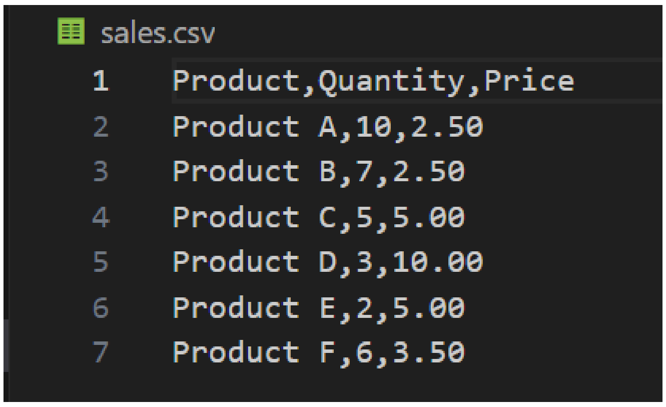
Company Problems Solved with Python

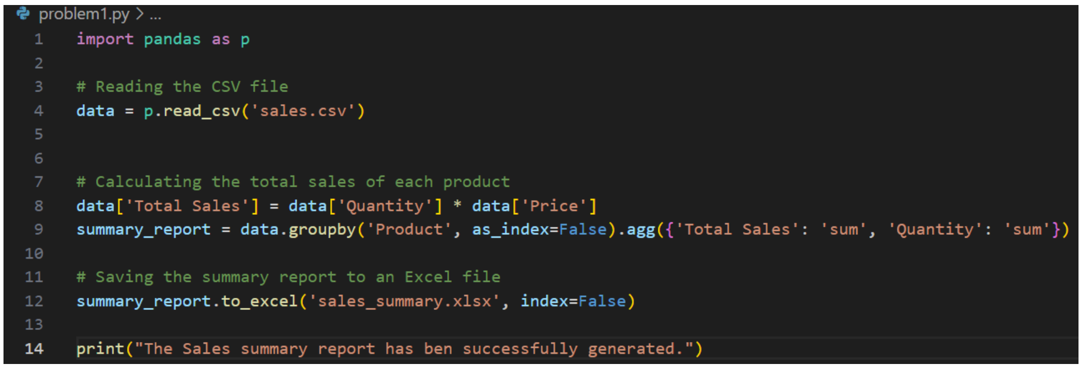
The IT office section in a company is facing 5 different problems that can be addressed and solved with the use of the Python programming language.

1. **Automating report generation**
   - **Problem:** Employees are spending a lot of time manually compiling reports from various data sources.
   - **Solution:** I used the pandas library for data manipulation and generated a report in an Excel format with the use of openpyxl.



sales.csv

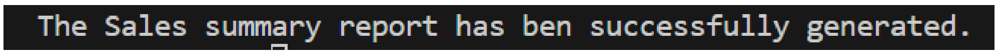
```
Product,Quantity,Price
Product A,10,2.50
Product B,7,2.50
Product C,5,5.00
Product D,3,10.00
Product E,2,5.00
Product F,6,3.50
```



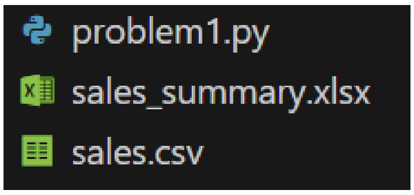
problem1.py > ...

```python
import pandas as p

# Reading the CSV file
data = p.read_csv('sales.csv')


# Calculating the total sales of each product
data['Total Sales'] = data['Quantity'] * data['Price']
summary_report = data.groupby('Product', as_index=False).agg({'Total Sales': 'sum', 'Quantity': 'sum'})

# Saving the summary report to an Excel file
summary_report.to_excel('sales_summary.xlsx', index=False)

print("The Sales summary report has ben successfully generated.")
```

By running the script python problem1.py in the terminal I see the following output:



```
The Sales summary report has ben successfully generated.
```

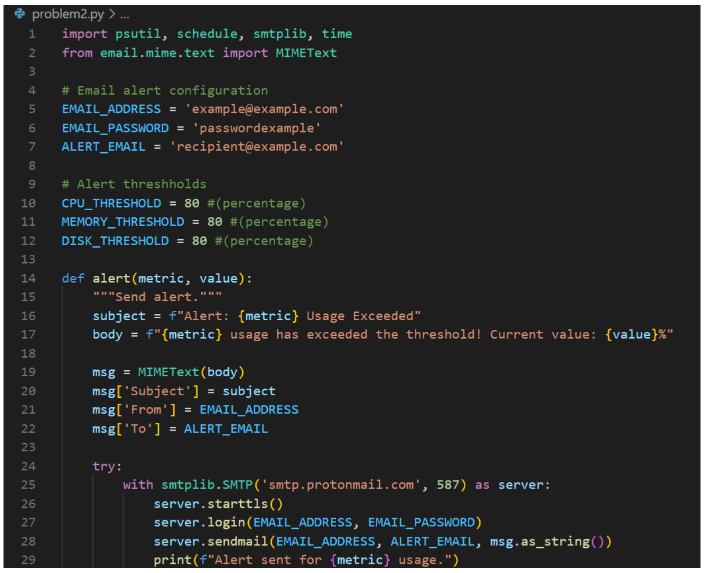


```
problem1.py
sales_summary.xlsx
sales.csv
```

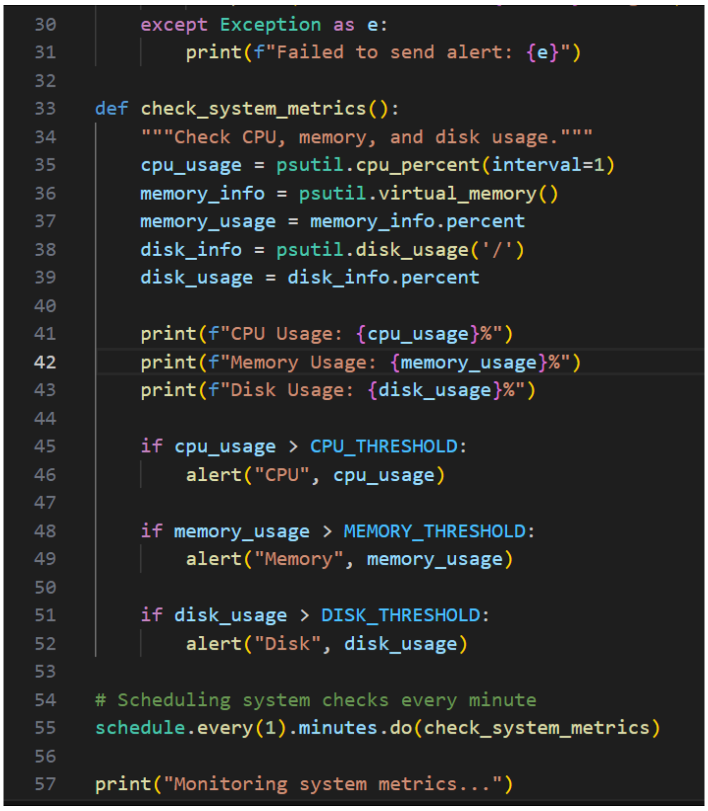
The Excel file has been successfully generated.

2. Monitoring system performance

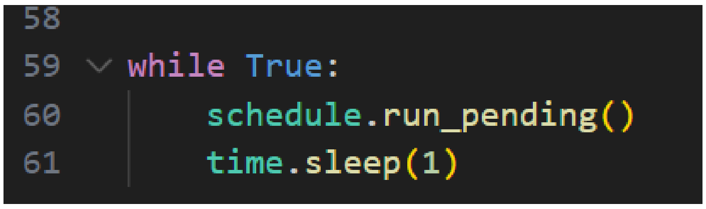
- **Problem:** IT staff need to monitor server performance and uptime manually.
- **Solution:** The metrics of CPU usage, memory usage and disk space need to be monitored. A threshold for these three metrics can be defined and if exceeded, an alert email is sent to the recipients responsible for monitoring the server. For this, the psutil, schedule and smtplib libraries can be used for gathering system metrics, scheduling regular checks and sending automated emails. The MIMEtext works as the body of the email and is used to aid the automated messages.



```python
import psutil, schedule, smtplib, time
from email.mime.text import MIMEText

# Email alert configuration
EMAIL_ADDRESS = 'example@example.com'
EMAIL_PASSWORD = 'passwordexample'
ALERT_EMAIL = 'recipient@example.com'

# Alert threshholds
CPU_THRESHOLD = 80 #(percentage)
MEMORY_THRESHOLD = 80 #(percentage)
DISK_THRESHOLD = 80 #(percentage)

def alert(metric, value):
    """Send alert."""
    subject = f"Alert: {metric} Usage Exceeded"
    body = f"{metric} usage has exceeded the threshold! Current value: {value}%"

    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = EMAIL_ADDRESS
    msg['To'] = ALERT_EMAIL

    try:
        with smtplib.SMTP('smtp.protonmail.com', 587) as server:
            server.starttls()
            server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
            server.sendmail(EMAIL_ADDRESS, ALERT_EMAIL, msg.as_string())
            print(f"Alert sent for {metric} usage.")
```



```python
    except Exception as e:
        print(f"Failed to send alert: {e}")

def check_system_metrics():
    """Check CPU, memory, and disk usage."""
    cpu_usage = psutil.cpu_percent(interval=1)
    memory_info = psutil.virtual_memory()
    memory_usage = memory_info.percent
    disk_info = psutil.disk_usage('/')
    disk_usage = disk_info.percent

    print(f"CPU Usage: {cpu_usage}%")
    print(f"Memory Usage: {memory_usage}%")
    print(f"Disk Usage: {disk_usage}%")

    if cpu_usage > CPU_THRESHOLD:
        alert("CPU", cpu_usage)

    if memory_usage > MEMORY_THRESHOLD:
        alert("Memory", memory_usage)

    if disk_usage > DISK_THRESHOLD:
        alert("Disk", disk_usage)

# Scheduling system checks every minute
schedule.every(1).minutes.do(check_system_metrics)

print("Monitoring system metrics...")
```

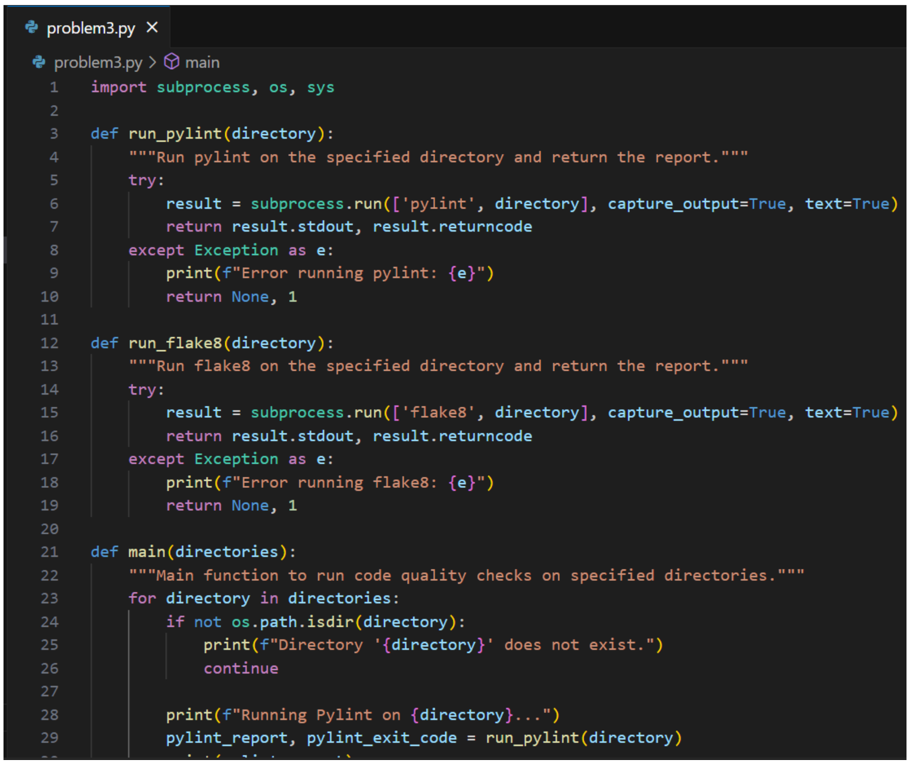


```python
while True:
    schedule.run_pending()
    time.sleep(1)
```

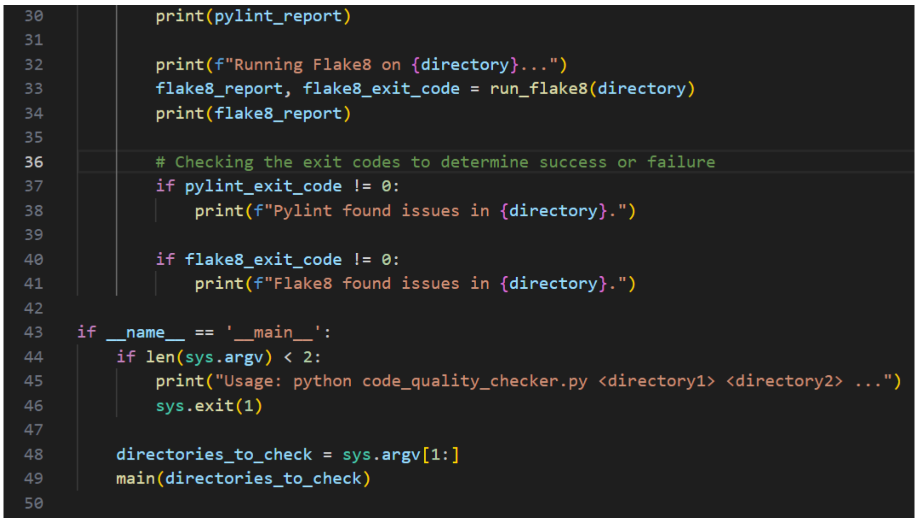
When the code is run on the terminal, the process will be executed with system metrics being monitored every minute, and an alert email will be sent in case the defined thresholds are exceeded.

3. Code quality checks

- **Problem:** In larger teams in the company, ensuring code quality and its adherence to standards can be challenging.
- **Solution:** The code quality check can be automated with the uses of Pylint and Flake8, the first being a static code analyzer and the second being a command-line tool which checks for style and syntax errors. The subprocess, os and sys libraries are imported, which aid in running shell commands, directory handling and command-line argument parsing.



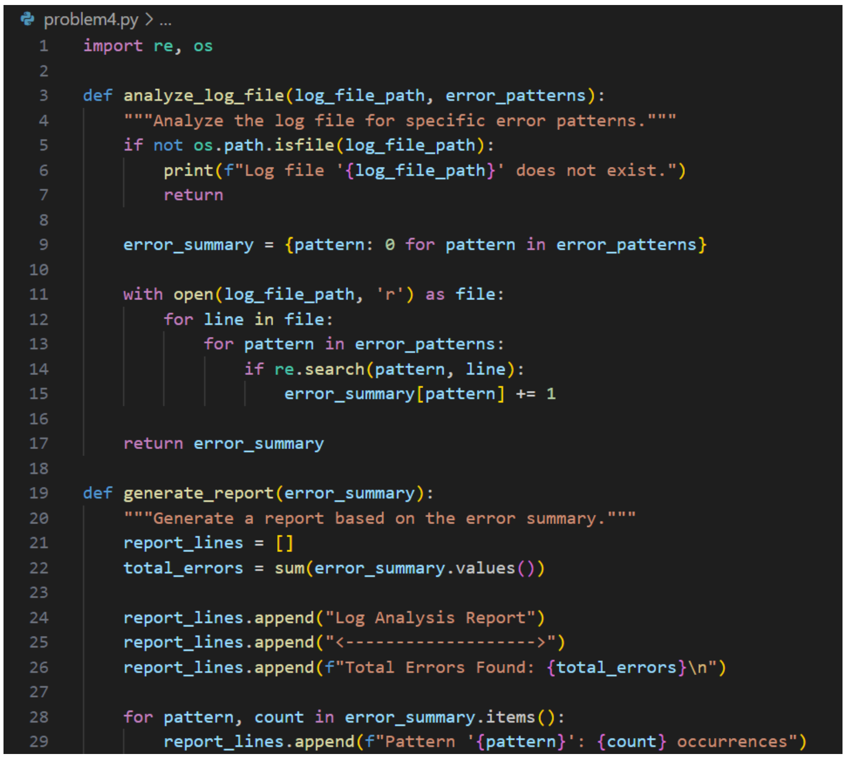
```python
import subprocess, os, sys

def run_pylint(directory):
    """Run pylint on the specified directory and return the report."""
    try:
        result = subprocess.run(['pylint', directory], capture_output=True, text=True)
        return result.stdout, result.returncode
    except Exception as e:
        print(f"Error running pylint: {e}")
        return None, 1

def run_flake8(directory):
    """Run flake8 on the specified directory and return the report."""
    try:
        result = subprocess.run(['flake8', directory], capture_output=True, text=True)
        return result.stdout, result.returncode
    except Exception as e:
        print(f"Error running flake8: {e}")
        return None, 1

def main(directories):
    """Main function to run code quality checks on specified directories."""
    for directory in directories:
        if not os.path.isdir(directory):
            print(f"Directory '{directory}' does not exist.")
            continue

        print(f"Running Pylint on {directory}...")
        pylint_report, pylint_exit_code = run_pylint(directory)
```



```
        print(pylint_report)

        print(f"Running Flake8 on {directory}...")
        flake8_report, flake8_exit_code = run_flake8(directory)
        print(flake8_report)

        # Checking the exit codes to determine success or failure
        if pylint_exit_code != 0:
            print(f"Pylint found issues in {directory}.")

        if flake8_exit_code != 0:
            print(f"Flake8 found issues in {directory}.")

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage: python code_quality_checker.py <directory1> <directory2> ...")
        sys.exit(1)

    directories_to_check = sys.argv[1:]
    main(directories_to_check)

```

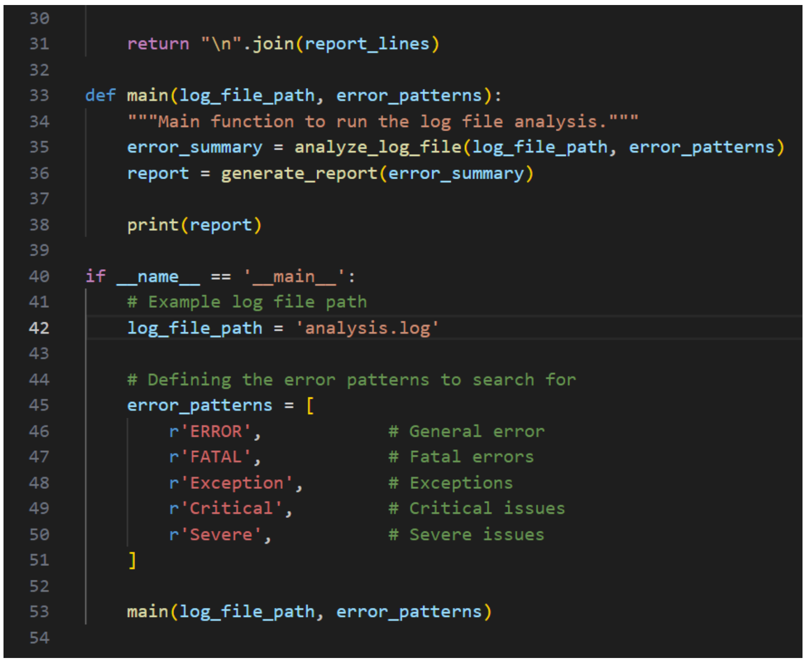
The code can then be integrated into a CI/CD pipeline.

4. **Log file analysis**
    - **Problem:** Analyzing server log files for errors and unusual patterns is a tedious task.
    - **Solution:** The task of analyzing log files can be analyzed by creating a script that reads log files, searches for specific error patterns with the use of regular expressions and then generates summaries/alerts based on the findings of every analysis. The re and os libraries are used for reading regular expressions and file handling.



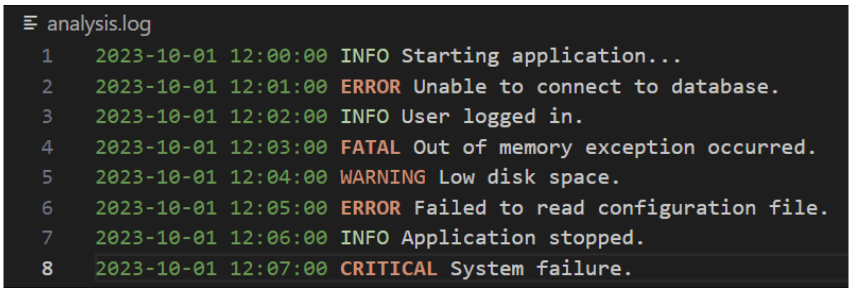
problem4.py > ...

```
import re, os

def analyze_log_file(log_file_path, error_patterns):
    """Analyze the log file for specific error patterns."""
    if not os.path.isfile(log_file_path):
        print(f"Log file '{log_file_path}' does not exist.")
        return

    error_summary = {pattern: 0 for pattern in error_patterns}

    with open(log_file_path, 'r') as file:
        for line in file:
            for pattern in error_patterns:
                if re.search(pattern, line):
                    error_summary[pattern] += 1

    return error_summary

def generate_report(error_summary):
    """Generate a report based on the error summary."""
    report_lines = []
    total_errors = sum(error_summary.values())

    report_lines.append("Log Analysis Report")
    report_lines.append("<------------------->")
    report_lines.append(f"Total Errors Found: {total_errors}\n")

    for pattern, count in error_summary.items():
        report_lines.append(f"Pattern '{pattern}': {count} occurrences")
```



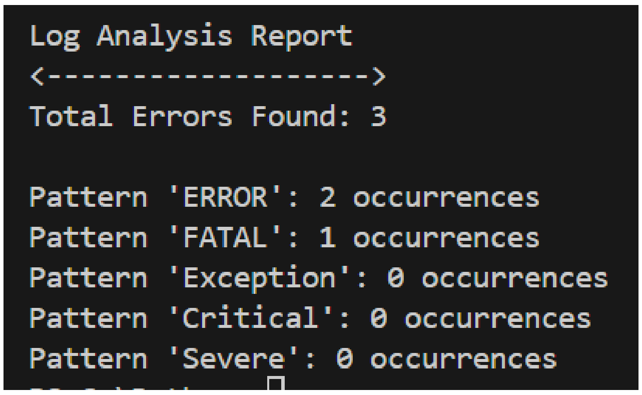
```python

    return "\n".join(report_lines)

def main(log_file_path, error_patterns):
    """Main function to run the log file analysis."""
    error_summary = analyze_log_file(log_file_path, error_patterns)
    report = generate_report(error_summary)

    print(report)

if __name__ == '__main__':
    # Example log file path
    log_file_path = 'analysis.log'

    # Defining the error patterns to search for
    error_patterns = [
        r'ERROR',          # General error
        r'FATAL',          # Fatal errors
        r'Exception',      # Exceptions
        r'Critical',       # Critical issues
        r'Severe',         # Severe issues
    ]

    main(log_file_path, error_patterns)

```

analysis.log file:



```
analysis.log
2023-10-01 12:00:00 INFO Starting application...
2023-10-01 12:01:00 ERROR Unable to connect to database.
2023-10-01 12:02:00 INFO User logged in.
2023-10-01 12:03:00 FATAL Out of memory exception occurred.
2023-10-01 12:04:00 WARNING Low disk space.
2023-10-01 12:05:00 ERROR Failed to read configuration file.
2023-10-01 12:06:00 INFO Application stopped.
2023-10-01 12:07:00 CRITICAL System failure.
```

Output:

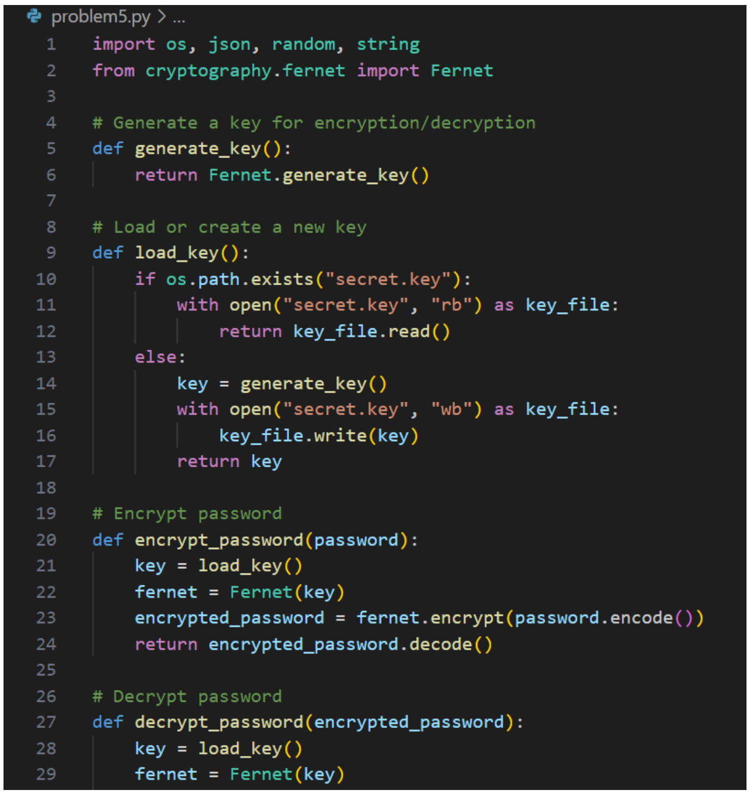


```
Log Analysis Report
<------------------->
Total Errors Found: 3

Pattern 'ERROR': 2 occurrences
Pattern 'FATAL': 1 occurrences
Pattern 'Exception': 0 occurrences
Pattern 'Critical': 0 occurrences
Pattern 'Severe': 0 occurrences
```

5. Password management

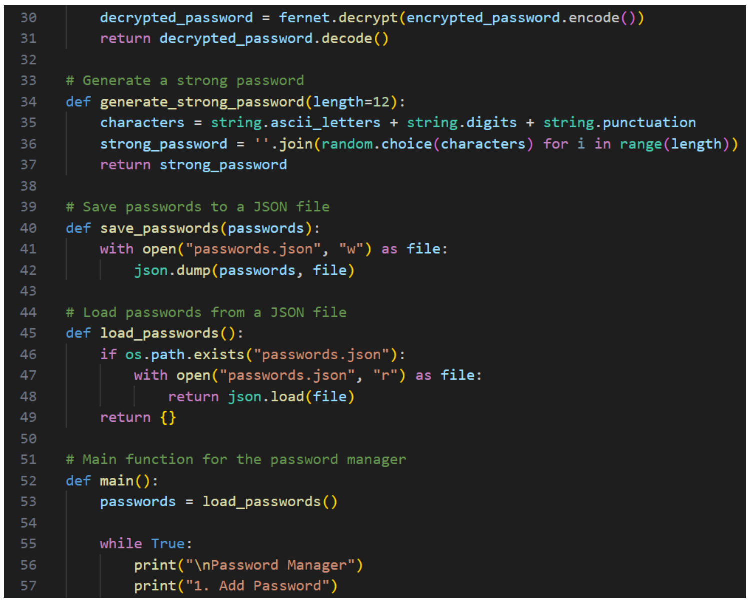
- **Problem:** Employees have a difficult time remembering multiple passwords securely.

- **Solution:** A password manager application can be created which can allow users to store and encrypt passwords securely, generate strong passwords and provide a user-friendly interface by importing and using the cryptography library for encryption. The encrypted passwords are saved in a JSON file. The command-line user interface that's created can then help users view stored passwords in their decrypted form and generate strong passwords of specified lengths. I didn't further enhance the application, but a GUI can even be added by using libraries such as Tkinter and PyQt.

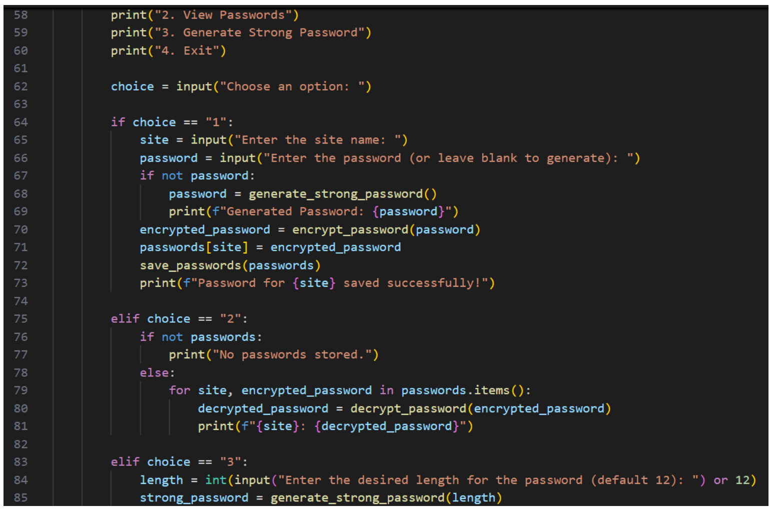


problem5.py > ...

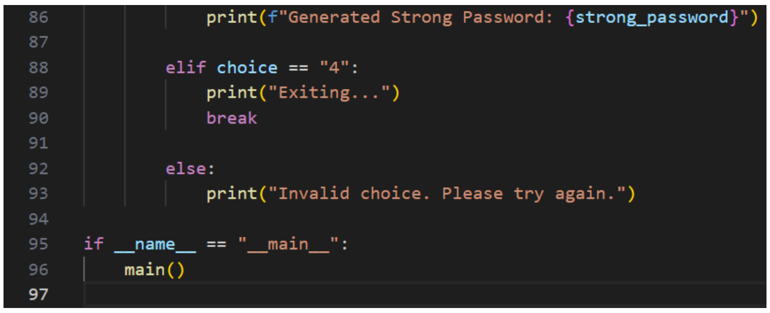
```python
import os, json, random, string
from cryptography.fernet import Fernet

# Generate a key for encryption/decryption
def generate_key():
    return Fernet.generate_key()

# Load or create a new key
def load_key():
    if os.path.exists("secret.key"):
        with open("secret.key", "rb") as key_file:
            return key_file.read()
    else:
        key = generate_key()
        with open("secret.key", "wb") as key_file:
            key_file.write(key)
        return key

# Encrypt password
def encrypt_password(password):
    key = load_key()
    fernet = Fernet(key)
    encrypted_password = fernet.encrypt(password.encode())
    return encrypted_password.decode()

# Decrypt password
def decrypt_password(encrypted_password):
    key = load_key()
    fernet = Fernet(key)
```



```python
    decrypted_password = fernet.decrypt(encrypted_password.encode())
    return decrypted_password.decode()

# Generate a strong password
def generate_strong_password(length=12):
    characters = string.ascii_letters + string.digits + string.punctuation
    strong_password = ''.join(random.choice(characters) for i in range(length))
    return strong_password

# Save passwords to a JSON file
def save_passwords(passwords):
    with open("passwords.json", "w") as file:
        json.dump(passwords, file)

# Load passwords from a JSON file
def load_passwords():
    if os.path.exists("passwords.json"):
        with open("passwords.json", "r") as file:
            return json.load(file)
    return {}

# Main function for the password manager
def main():
    passwords = load_passwords()

    while True:
        print("\nPassword Manager")
        print("1. Add Password")
```



```python
        print("2. View Passwords")
        print("3. Generate Strong Password")
        print("4. Exit")

        choice = input("Choose an option: ")

        if choice == "1":
            site = input("Enter the site name: ")
            password = input("Enter the password (or leave blank to generate): ")
            if not password:
                password = generate_strong_password()
                print(f"Generated Password: {password}")
            encrypted_password = encrypt_password(password)
            passwords[site] = encrypted_password
            save_passwords(passwords)
            print(f"Password for {site} saved successfully!")

        elif choice == "2":
            if not passwords:
                print("No passwords stored.")
            else:
                for site, encrypted_password in passwords.items():
                    decrypted_password = decrypt_password(encrypted_password)
                    print(f"{site}: {decrypted_password}")

        elif choice == "3":
            length = int(input("Enter the desired length for the password (default 12): ") or 12)
            strong_password = generate_strong_password(length)
```



```python
            print(f"Generated Strong Password: {strong_password}")

        elif choice == "4":
            print("Exiting...")
            break

        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

I found examples of different problems online and modified them in order to create this folder.