Projet : Gestion de Code-Source en Python

Philippe ROUSSILLE

Bachelor CSI 4

Rodez

Cadre général

Contexte

- Vous devez réaliser le projet ci-dessous pour le **7 avril 2025**, **11h59**.
- Un rendu de suivi sur Git sera effectué le **15 janvier 2025** à **11h59**.

Le projet sera automatiquement récupéré sur Git aux heures affichées.

Objectifs principaux

1. Réaliser un dépôt Git/GitHub respectant les bonnes pratiques.
2. Documenter parfaitement le code et le projet.
3. Ajouter des tests unitaires pour garantir la qualité du code.
4. Mettre en place une intégration continue avec GitHub Actions.

Choix de la cible (projets individuels)

Option A : Application CLI (avec argparse)

Description

Développez une application en ligne de commande pour gérer une liste de tâches personnelles.

Exigences

1. **Fonctionnalités obligatoires :**
   - Ajouter une tâche avec un titre, une description, une priorité et une date limite.
   - Afficher les tâches triées par priorité ou par date.
   - Modifier ou supprimer une tâche.
   - Sauvegarder les tâches dans un fichier JSON pour persistance.
2. **Fonctionnalité avancée :**
   - Implémenter un système de rappel automatique : afficher les tâches en retard ou proches de la date limite.

Exemple d'utilisation

```
# Ajouter une tâche
$ python task_manager.py add --title "Préparer le rapport" --desc
        "Rédiger la section sur les tests" --priority 1 --due
        2025-01-20

# Afficher les tâches triées par priorité
$ python task_manager.py list --sort priority

# Supprimer une tâche
$ python task_manager.py delete --id 3
```

Option B : Suivi de réseau (Détection d'IP actives/ inactives)

Description

Créez un outil capable de scanner une plage d'adresses IP pour détecter les IP actives et inactives.

Exigences

1. **Fonctionnalités obligatoires :**
   - Scanner une plage d'adresses spécifiée par l'utilisateur (ex. : `192.168.1.1/24`).
   - Afficher les IP actives avec leur temps de réponse.
   - Sauvegarder les résultats dans un fichier CSV.
2. **Fonctionnalité avancée :**
   - Permettre un scan asynchrone (via `asyncio`) pour améliorer la vitesse.
   - Ajouter une option pour détecter les ports ouverts sur les IP actives.

Exemples de configuration à pinger

- **Plage d'adresses IP à scanner**

  ```
  $ python net_scanner.py scan --range 192.168.1.0/24
  ```

- **Fichier de configuration personnalisée** Fichier `ip_list.txt` :

  ```
  192.168.1.1
  192.168.1.10
  192.168.1.50
  10.0.0.1
  ```

  Utilisation :

```
    $ python net_scanner.py scan --file ip_list.txt
```

Exemple de sortie générée

```
$ python net_scanner.py scan --range 192.168.1.0/24

192.168.1.1    Active   (Ping: 5ms)
192.168.1.10   Active   (Ping: 2ms)
192.168.1.20   Inactive
192.168.1.30   Active   (Ping: 7ms)

Résultats sauvegardés dans results.csv
```

Fichier `results.csv` :

```
IP,Status,Ping (ms)
192.168.1.1,Active,5
192.168.1.10,Active,2
192.168.1.20,Inactive,
192.168.1.30,Active,7
```

Option C : Analyseur de fichiers logs

Description

Créez un analyseur capable de traiter un fichier de journalisation (ex. : Apache).

Exigences

1. **Fonctionnalités obligatoires :**
    - Extraire et afficher :
        - Le nombre total de requêtes.
        - Les IP les plus actives.
        - Les erreurs fréquentes (404, 500, etc.).
    - Générer un rapport sous format JSON ou texte.
2. **Fonctionnalité avancée :**
    - Ajouter des filtres dynamiques (par exemple, afficher les requêtes entre deux dates ou d'une IP spécifique).
    - Créer des graphiques simples (ex. : histogramme des requêtes par heure).

Exemple de fichier log Apache

```
192.168.1.1 - - [12/Jan/2025:10:15:32 +0000] "GET /index.html
HTTP/1.1" 200 532
192.168.1.2 - - [12/Jan/2025:10:16:15 +0000] "POST /login HTTP/
```

```
1.1" 302 0
192.168.1.1 - - [12/Jan/2025:10:16:45 +0000] "GET /dashboard
HTTP/1.1" 404 199
192.168.1.3 - - [12/Jan/2025:10:17:10 +0000] "GET /api/data HTTP/
1.1" 500 123
192.168.1.4 - - [12/Jan/2025:10:18:20 +0000] "GET /index.html
HTTP/1.1" 200 532
192.168.1.1 - - [12/Jan/2025:10:18:45 +0000] "GET /profile HTTP/
1.1" 200 345
```

Exemple de sortie générée

Commande :

```
$ python log_analyzer.py analyze --file access.log
```

Sortie :

```
Total de requêtes : 6
Requêtes par statut HTTP :
  - 200 : 3
  - 302 : 1
  - 404 : 1
  - 500 : 1
Top 3 IPs :
  - 192.168.1.1 : 3 requêtes
  - 192.168.1.4 : 1 requête
  - 192.168.1.2 : 1 requête
```

Rapport généré en JSON :

```
{
  "total_requests": 6,
  "status_codes": {
    "200": 3,
    "302": 1,
    "404": 1,
    "500": 1
  },
  "top_ips": [
    {"ip": "192.168.1.1", "requests": 3},
    {"ip": "192.168.1.4", "requests": 1},
    {"ip": "192.168.1.2", "requests": 1}
  ]
}
```

Travail demandé

1. Dépôt Git/GitHub

- **Structure :**
  - Un fichier `README.md` clair avec :
    - Une description du projet.
    - Les commandes d'utilisation (avec exemples).
    - Les dépendances nécessaires (si utilisées).
  - Fichiers et dossiers clairement structurés (par exemple : `src/`, `tests/`, etc.).
- **Commits :**
  - Messages clairs et respectant un format standard.
  - Utilisation d'au moins une branche secondaire pour le développement.

2. Documentation

- Docstrings conformes à PEP 257 pour toutes les classes, fonctions et modules.
- Génération d'une documentation avec Sphinx, hébergée sur GitHub Pages.

3. Tests unitaires

- Couverture de 80% minimum avec `unittest`.
- Utilisation de `coverage.py` pour générer un rapport de couverture.

4. Intégration Continue (CI/CD)

- Workflow GitHub Actions qui :
  - Lance les tests unitaires à chaque push ou pull request.
  - Vérifie la qualité du code avec **pylint** (score minimum : 9.0/10).
  - Génère et déploie la documentation automatiquement.

5. Rapport

Introduction

Dans le cadre de ce projet, vous devez également rédiger un rapport technique pour présenter et analyser vos choix de conception ainsi que les difficultés rencontrées. Ce document doit refléter votre réflexion tout au long du projet et votre capacité à résoudre des problèmes de manière autonome.

Format attendu

Le rapport doit être rédigé en texte structuré, soit au format **Markdown** (`.md`), soit au format **reStructuredText (RST)** (`.rst`). Vous devez inclure ce

fichier directement dans votre dépôt GitHub, à la racine du projet, sous le nom suivant : `rapport.md` ou `rapport.rst`.

Structure du rapport

1. Introduction

- **Objectif du projet :** Expliquez brièvement la finalité de votre projet.
- **Cible choisie :** Indiquez laquelle des trois options vous avez sélectionnée (Application CLI, Suivi de réseau, Analyseur de logs) et pourquoi.

2. Solutions techniques

- Décrivez les outils et bibliothèques que vous avez utilisés (par exemple : argparse, asyncio, unittest, Sphinx, etc.).
- Justifiez vos choix en expliquant pourquoi ces solutions étaient adaptées à votre projet.

3. Difficultés rencontrées

- Listez les principaux problèmes techniques ou organisationnels rencontrés durant le projet.
- Expliquez comment vous avez surmonté ces difficultés :
  - Problèmes liés à la gestion de Git/GitHub.
  - Problèmes de code ou de logique.
  - Problèmes de configuration des workflows CI/CD.

4. Améliorations et optimisations

- Identifiez les points faibles de votre projet et proposez des solutions pour les améliorer.
- Listez les fonctionnalités avancées ou optimisations que vous auriez souhaité intégrer avec plus de temps.

5. Bilan personnel

- Résumez les compétences que vous avez développées au cours de ce projet (techniques et organisationnelles).
- Ajoutez une réflexion personnelle sur ce que vous auriez fait différemment et ce que vous retenez de cette expérience.

Barème

Critères	**Points**
Structure du dépôt Git (branches, commits)	20

Critères	Points
Documentation (docstrings, Sphinx)	20
Tests unitaires	20
Intégration continue avec GitHub Actions	20
Qualité du code (pylint ≥ 9.0)	20
Réalisation des fonctionnalités avancées (bonus)	20
Rapport	30
Total	**130**

- **Les fonctionnalités de base ne sont pas notées.** Leur absence entraînera un malus dans le barème final.

Modalités de rendu

- Le projet doit être déposé sur un dépôt GitHub public ou privé (avec accès partagé si privé).
- La date limite de rendu est fixée **au 7 avril 2025, 11h59, sans faute**.