**TP2 - Python**

Vous pouvez utiliser une VM, un conteneur ou votre hôte pour ce TP. Seule condition : **Python3 up-to-date**.

Pour chacun des exercices vous rendrez un fichier .py exécutable.

N'hésitez pas à faire vos tests directement dans la console python, et à vous aider du [mémo pour la syntaxe de base](https://gist.github.com/It4lik/097def35b9abb53f0ce96334fba4d932).

Le nom des scripts est imposé encore une fois et est précisé pour chacun d'entre eux. Aussi, tous vos scripts devront avoir :

* Un en-tête
* Des déclarations de fonctions
* Des déclarations de variables
* Commentaire

**1. Manipulation de bases**

**1a-add.py**

Demander deux nombres à l’utilisateur et afficher le résultat de l’addition des deux nombres.  
Contrôler que l’utilisateur a bien saisi deux nombres et pas autre chose.

|  |
| --- |
| #!/usr/bin/python36  import re  numb1 = input('numéros 1 ?')  pattern1 = re.compile('[0-9]+')  numb2 = input('numéros 2 ?')  pattern2 = re.compile('[0-9]+')  while not pattern1.match(numb1):  numb1 = input('numéros 1 ?')  while not pattern2.match(numb2):  numb2 = input('numéros 2 ?')  inumb1 = int(numb1)  inumb2 = int(numb2)  def add(inumb1, inumb2):  return inumb1 + inumb2  print(add(inumb1, inumb2)) |

**1b-dic.py**

Demander une saisie utilisateur de plusieurs prénoms.  
L’utilisateur peut choisir d’arrêter la saisie en appuyant sur la touche ‘q’.  
Les stocker dans une liste. Afficher le résultat en ordonnant les prénoms par ordre alphabétique.

**1c-moy.py**

Demander une saisie utilisateur de plusieurs notes et prénoms :

* L’utilisateur saisit un prénom, et une note pour chacun d'entre eux
* L’utilisateur peut saisir 'q' pour stopper la saisie
* Affichez ensuite la moyenne et un top 5 des meilleures notes
* Hint : utiliser un dictionnaire python

**TODO**

Réécrire les trois précédents scripts, et contrôler la saisie utilisateur. Pour l'addition par exemple, vérifier que l'utilisateur a bien saisi des nombres.  
Vous le ferez ensuite pour **TOUTES LES ENTREES UTILISATEURS**.

**1d-mol.py**

Jeu du plus ou moins :

* Générer un nombre aléatoire (entre 0 et 100 par exemple).
* Demander une saisie utilisateur tant qu’il ne trouve pas le nombre que vous avez généré.
* Afficher à chaque nombre saisi s’il est plus grand ou plus petit que le nombre que vous avez généré.

**2. Next step**

**2a-mol.py**

Jeu du plus ou moins qui lit dans un fichier. Développer un jeu du plus ou moins qui tourne en démon. Pour ce faire :

* Créer une boucle pour que le script tourne en permanence
* Écrire dans un fichier un message de bienvenue
* L’utilisateur peut saisir un nombre dans ce fichier
* Votre programme doit lire le nombre et écrire 'plus' ou 'moins' dans le fichier

**2b-auto.py**

Ecrire un programme qui résout le programme précédent.

**3. Sauvegarde**

**3a-save.py**

Modules à regarder : shutil, gzip, os, sys) Ecrire un script qui permet d’effectuer une sauvegarde : Votre outil doit :

* Archiver et compresser (en .tar.gz) le répertoire choisi
* Déplacer la sauvegarde dans un répertoire data créer à cet effet.
* La sauvegarde ne doit être déplacée dans le répertoire data uniquement si elle est différente de la précédente
* Utiliser le module sys pour afficher des informations dans la sortie standard plutôt qu’avec print
* Utiliser aussi sys pour retourner un code d’erreur spécifique en cas d’échec du programme
* Utiliser le module signal pour intercepter le signal SIGINT et effectuer une action personnalisée.
* Gérer les exceptions et affichez dans la sortie d’erreur un texte personnalisé en cas d’échec d’une étape importante (pas les droits sur le fichier, ou sur la destination, ou encore, en cas de remplissage du disque par exemple).

Allez-y étape par étape et commencer par un archivage simple :)

**3b-opt.py**

Le même script que le précédent mais en plus il doit disposer d'options au lancement pour :

* -h : un help pour le reste
* Permettre de choisir un répertoire à surveiller en le passant en option de lancement
  + Plusieurs répertoires pourront être gérés
* Choisir le chemin du répertoire data

**3c-ssh.py**

On continue.  
Si l’utilisateur le souhaite, il peut choisir de copier l’archive résultante sur un serveur distant SSH.  
Les options permettront de préciser un utilisateur et un mot de passe pour la connexion SSH.  
Pour ce faire, vous pouvez utiliser le module paramiko et le module scp suivant : <https://github.com/jbardin/scp.py> (par exemple)

**3d-daemon.py**

Votre script évolue, encore :

* Il doit pouvoir tourner en démon pour surveiller la taille d'un répertoire
* Au-dessus d'un seuil critique, il doit déplacer le contenu après l'avoir archivé dans un répertoire data prévu à cet effet et effacer l'ancien contenu du répertoire
* Le script dispose d'un fichier de configuration (plus pratique que des options quand ça devient complexe). On peut y préciser :
  + X répertoires à surveiller
  + Une destination pour chacun des répertoires (avec les infos ssh si besoin)

**3e-backup.service ?**

Besoin d'un OS avec systemd pour cette partie. La VM CentOS7 des cours précédents peut faire l'affaire. Vous allez créer un fichier .service simple qui permet de démarrer et arrêter votre script. Autrement dit, on pourra :

* Configurer votre service de backup avec un fichier de configuration
* Démarrer ou arrêter le service avec systemctl start super-backup et systemctl stop super-backup

En livraison pour cette étape on a :

* L’application de backup : 3d-daemon.py
  + Avec un -h bien fait
  + On peut tester des backups directement en ligne de commande avec les options
* Un fichier de configuration d'exemple
  + Pour une configuration plus complexe et pérenne
* Une unité de service systemd
  + Pour gérer le service comme les autres services du système
* Bonus : un fichier de log
  + Le contenu est libre à vous, s'inspirer de fichiers de logs existants peut donner des idées

**4. HIDS**

**4a-hids.py**

La fonction de cet outil est bien différente. Mais si vous m'avez suivi jusque-là, dans la mise en place technique c'est assez similaire.  
Un **HIDS** (*HostBased Intrusion Detection System*) **est un outil de détection d'intrusion**. Une des fonctions de base est de surveiller des répertoires ou des fichiers.  
En cas de changement de l'un d'entre eux, il effectue un rapport.

Votre script doit :

* **Surveiller une liste de répertoire et de fichiers** (fichier de conf ?)
* Doit permettre plusieurs méthodes pour **vérifier l'état des fichiers**. Par exemple :
  + Nom
  + Taille
  + Date de modification
  + Hash
* Pour ce faire, vous devrez **conserver les données** au fil des exécutions du script :
  + Un fichier simple en json par exemple
  + Une base de données pour les motivés (redis, mysql, ou autres). Redis est très simple à utiliser (mais très puissant), c'est un simple système de clé/valeur. Je pourrai vous indiquer comment en mettre un en place rapidement si vous le souhaitez.
* En cas de détection d'un changement, il doit **effectuer une alerte** : écrire dans un **fichier de log**

**Bonus** :

* Créer une **unité de service systemd** pour gérer l'HIDS
* Si vous craquez : créer une **API HTTP** simpliste qui permet de récupérer les alertes simplement :
  + Une action pour lister toutes les alertes
  + Une action pour lister une alerte (avec son ID)
  + Une action pour récupérer la liste des dossiers surveillés
  + Une action pour récupérer les alertes d'un dossier (avec son ID)
  + Un fichier de log en json peut être utile pour être facilement parsable