Outils standards en Python

Certains outils sont considérés comme étant standards dans le développement Python. Ce document a pour but de les décrire brièvement. Il suppose l'utilisation de la version 3 de Python (invoquée avec `python3`), sous l'environnement Linux (Ubuntu) fourni par l'école.

PIP

`pip` est un gestionnaire de paquets pour Python. Il permet d'installer des programmes et des librairies écrites en Python. Pour installer une librairie (par exemple `numpy`), il suffit de faire :

```
$ pip install numpy
```

Sous Linux, il vous faudra probablement les droits administrateurs (par défaut, les paquets sont installés pour le système entier). Il faut alors préfixer la commande par sudo.

Il est aussi possible de créer, pour son projet, un fichier de dépendances (communément nommé `requirements.txt`) dans lequel vous précisez les paquets requis, un par ligne. Pour installer toutes les dépendances d'un coup, il suffit d'invoquer la commande suivante :

```
$ pip3 install -r requirements.txt
```

Pour plus d'informations, vous pouvez consulter la documentation complète de l'outil, disponible ici. Tous les outils mentionnés ci-après peuvent être installés grâce à `pip`.

Gestion d'environnements virtuels avec Python

virtuelenv permet de créer un environnement "virtuel" dans lequel il est possible d'installer des modules, d'utiliser des version spécifiques de Python, sans "corrompre" l'installation de Python sur votre système. Un nouvel environnement est créé à l'intérieur de chaque projet.
virtualenvwrapper permet de simplifier la gestion des environnements virtuels. Les environnements créés sont centralisés dans un même dossier (`$PROJECT_HOME`). Ils sont donc mutualisables par plusieurs projets nécéssitant le même contexte (version de Python, modules, etc.).

Installation

- Installer les modules `virtualenv` et `virtualenvwrapper` :

```
# module de base
$ sudo pip install virtualenv

$ sudo pip install virtualenvwrapper
```

Configuration

- Dans votre **.bashrc** ou votre **.profile**, ajoutez les lignes suivantes
  (en supposant que la variable HOME est positionnée, sinon remplacer par un chemin valide de votre arborescence) :

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/<dossier à créer>
source /usr/local/bin/virtualenvwrapper.sh
```

Gérer un environnement virtuel

- Création :

```
$ mkvirtualenv ENVNAME
```

- Liste des environnements créés :

```
$ workon

$ lsvirtualenv
```

- Se placer dans l'environnement virtuel :

```
$ workon ENVNAME
```

- Installer les modules dans l'environnement virtuel :

```
(ENVNAME)$ pip install MODULE
```

- Quitter l'environnement virtuel :

```
(ENVNAME)$ deactivate
```

- Supprimer l'environnement virtuel :

```
$ rmvirtualenv ENVNAME
```

Quelques commandes utiles

- Spécifier une version de python :

```
$ mkvirtualenv [-p|--python /chemin/vers/pythonVErSiOn] ENVNAME
```

- Exporter la liste des paquets installés (depuis un environnement) :

```
(ENVNAME)$ pip freeze > requirement.txt
```

- Lancement avec fichier de configuration :

```
$ mkvirtualenv [-i package] [-r requirements_file] ENVNAME
```

- `wipeenv` supprime les packages installés dans l'environnement courant.
- Guide de démarrage rapide;
- Commandes de référence

Vérification de code

Python est un langage interprété, ce qui signifie que les erreurs de syntaxe ne sont pas apparentes avant l'exécution du code.

Des outils existent pour inspecter le code de façon statique (sans l'exécuter), nommés outils d'analyse statique de code.

Un tel outil en Python est pyflakes. Pour analyser un fichier, on appelle `pyflakes` de la façon suivante :

```
$ pyflakes <nomfichier>.py
```

`pyflakes` va ensuite afficher dans la console tous les problèmes de syntaxe du code. D'autres outils existent pour également vérifier le style du code, comme flake8, qui impose le guide de style pep8.