MGL7460 Énoncé du TP 3

Version 0.5

Description du projet

Le but de ce TP est de vous familiariser, par la pratique, avec *des* concepts de la *métaprogrammation* qui sont à la base de la plupart de *cadriciels* (*frameworks*) et *interstitiels* (*middleware*) en usage aujourd'hui : ce sont des programmes qui manipulent des (représentations) de programmes pour implanter des services qui seraient autrement trop complexes et pas très intéressants à développer, ou à invoquer, par les développeurs.ses, qui feraient mieux de se concentrer sur la *logique d'affaires*. En particulier, vous aurez l'occasion de vous familiariser, dans le cadre de l'écosystème python, avec les concepts suivants :

1. L'analyse de données sous format JSON en vue d'en *déduire*[1] la structure des données sous-jacentes, à représenter sous la forme de *classes*[2], avec des *attributs* (valeurs simples) ou des *relations* (vers d'autres objets composés). C'est ce que l'on appelle un *métamodèle*
    - Vous utiliserez le module « json » (et plus particulièrement, la méthode json.load(…)) pour lire le contenu d'un fichier json.
2. La génération de code de classes python, à partir de la représentation de leurs structures (par des *classes* et des *relations*)
3. Le chargement et la « compilation » des fichiers/modules python correspondants, de façon programmée (par l'exécution de code python), vous permettant d'avoir une représentation « run-time » de ces classes générées, et donc, éventuellement de les instancier et d'exécuter des opérations dessus
    - Vous utiliserez les modules : a) « importlib » (et plus particulièrement, la méthode importlib.import_module(…)) et b) inspect (et plus particulièrement, inspect.getmembers(…)), pour ce faire
4. La création d'objets des classes générées, initialisés avec les mêmes données (notre fichier json de l'étape 1), et leur impression avec des implémentations sur mesure de __str__() (à inclure dans l'étape 2).

---

[1] L'une des « différences majeures de philosophie » entre JSON et XML est que les données XML sont typiquement accompagnées de *schémas* permettant d'en décrire la structure. En principe, on ne peut « interpréter » un fichier de données XML que si on a son *schéma* correspondent. Avec JSON, on n'a pas la notion de *schéma*. Donc, dans notre cas, nous allons « deviner une *structure* (un *schéma*) plausible » pour les données que nous avons à l'entrée. C'est un peu dans l'esprit de l'approche « lightweight » de JSON et des bases de données NoSQL

[2] On aurait pu représenter la même structure avec *tables* et *colonnes*

- Pour ce faire, vous utiliserez la fonction python eval(<expression>,<dictionnaire de variables>) pour exécuter des instructions (expressions) générées à l'exécution[3]

Comme pour les deux premiers TPs, vous utiliserez git et gitlab pour :

1) La gestion de votre code source
2) La construction de « pipeline » de build
3) L'analyse statique de code en utilisant l'outil SonarQube

Le travail sera évalué sur la base de:

1. Une implantation des classes qui "passe" une batterie de tests développée par le professeur
2. La qualité de votre code et de sa documentation
3. Le succès du pipeline gitlab
4. Le résultat de l'exécution de SonarQube sur votre code
5. Un mini rapport écrit

Votre input

Le professeur vous fournira :

1) Le fichier de données (boutique.json)
2) Un gabarit du projet python, à utiliser comme guide
3) Un exemple de sortie (modules python générés automatiquement à partir du fichier de données)
4) Un ensemble de tests à réussir[4]
5) Des fragments de code, à fournir sur demande, pour vous faciliter certaines tâches

Barème

Critère	Poids
Du code qui passe (réussit) les tests	60%
Une pipeline gitlab fonctionnelle	10%
Qualité de codage et de la documentation, selon professeur	10%
Qualité de codage selon SonarQube	10%
Qualité du rapport écrit	10%

[3] Il y a d'autres façons, un peu plus robustes et un peu plus compliquées, pour ce faire.

[4] Je n'ai pas encore finalisé mon choix sur la forme de tests. Deux choix possibles : 1) une classe de tests unittest, 2) des descriptions abstraites de propriétés à vérifier que vous implanterez avec vos propres classes de test.