Projet de compilation (Étape 1)

Samuel Tardieu
SE202 - Année scolaire 2016/2017

Mise en place d'un environnement de développement

Python et virtualenv

Le projet nécessite d'utiliser Python 3.4 ainsi que la bibliothèque Ply.

Pour cela, il est conseillé de mettre en place puis d'utiliser `virtualenvwrapper` qui permet de mettre en place des environnements de Python isolés, avec une version particulière de l'interpréteur et ses propres bibliothèques.

Liens :

- Python
- virtualenvwrapper
- Ply

virtualenvwrapper à l'école

Sur les machines de l'école, on peut utiliser `virtualenvwrapper` en rajoutant simplement dans le fichier d'initialisation de son shell (par exemple `.bashrc` ou `.zshrc` dans le répertoire utilisateur) la ligne :

```
. /usr/share/virtualenvwrapper/virtualenvwrapper.sh
```

(ne pas oublier le `.` suivi d'un espace en début de ligne)

et en ré-ouvrant le shell.

Création de l'environnement virtuel

Une fois `virtualenvwrapper` installé et mis en place dans les fichiers de configuration du shell, il faut créer l'environnement de travail (appelons le `se202`).

```
% mkvirtualenv -p /usr/bin/python3.4 se202
(se202)% pip install ply
```

Pour recommencer à travailler dans cet environnement, il suffira de faire:

```
% workon se202
```

Création ou utilisation de la clé SSH

Pour pouvoir accéder en SSH aux dépôts Gitlab, il vous faut utiliser une clé SSH. Si vous n'en avez pas, vous pouvez créer une paire clé publique / clé privée avec la commande

```
% ssh-keygen
```

Vous pouvez ensuite, dans les paramètres de Gitlab, configurer votre clé **publique** (qui se trouve dans `~/.ssh/id-rsa.pub`).

Au début d'une session, vous pouvez avoir besoin d'indiquer à l'agent SSH local votre mot de passe pour qu'il puisse utiliser votre clé, en tapant

```
% ssh-add
```

Dépôt git (accès en SSH)

Vous allez utiliser votre propre dépôt git, dans lequel vous ajouterez le code venant de l'équipe pédagogique. Pour cela, il vous faut sortir votre dépôt (où *prenom* et *nom* sont sans accents), et y ajouter comme nouveau dépôt distant le dépôt contenant le code de référence.

```
% git clone git@gitlab.enst.fr:SE202_2017/prenom-nom tiger
% cd tiger
% git remote add template \
    git@gitlab.enst.fr:SE202_2017/template
```

- `origin` représente votre dépôt, vers lequel vous ferez des `git push` ;
- `template` représente le dépôt de l'équipe pédagogique, d'où vous prendrez le nouveau code au fur et à mesure de l'avancée du projet.

Première étape

Avant de commencer

Avant de commencer à travailler, il vous faut intégrer le code venant de l'équipe pédagogique. Depuis votre dépôt :

```
% git remote update
% git merge template/step1
```

À ce stade, votre dépôt a été enrichi de l'étape `step1` du dépôt de référence. Vérifiez que les tests de base fonctionnent :

```
% workon se202
(se202)% python -m unittest
```

La première commande vous place dans votre environnement de développement virtuel Python `se202`, si ce n'était pas déjà fait. La seconde cherche tous les tests unitaires du dépôt et les exécute.

Travail préliminaire

- Explorez les fichiers qui sont disponibles. Assurez vous de comprendre à quoi ils servent.
- `dumper` est un visiteur qui parcourt l'arbre et affiche son contenu sous forme de chaîne de caractères. Des tests dans `test_dumper` vérifient que son comportement est correct.
- `evaluator` est un autre visiteur qui évalue l'expression (mode interprétation). Des tests dans `test_evaluator` vérifient que son comportement est correct.

Vous pouvez également tester vos visiteurs grâce au programme `tiger.py` qui se trouve à la racine de votre dépôt. Pour avoir l'aide, faîtes `./tiger.py --help`.

Travail à faire

- Ajoutez les opérateurs binaires `-`, `/`, `&`, `|`, ainsi que les opérateurs de comparaison (cf. cours). On rappelle qu'une comparaison vérifiée renvoie 1, qu'une comparaison non vérifiée renvoie 0.
- Écrivez des tests pour ces expressions.
- Attention à la précédence et à l'associativité de ces opérateurs. Notamment, `10 - 1 - 2` doit renvoyer 7 et s'afficher comme `((10 - 1) - 2)`.
- Implémentez la construction `if`/`then`/`else`, et bien entendu ajoutez les tests correspondants. On rappelle que toute valeur différente de 0 est considérée comme vraie.
- N'oubliez pas de *pusher* le résultat sur votre dépôt git personnel (branche `master`).

Note sur `if/then/else`

Les sous-expressions doivent être composées telles qu'elles soient les plus longues possibles. Par exemple, le code suivant

```
if 1 then 100 else 200 + 300
```

doit s'évaluer en 100, pas en 400, car il doit être compris comme

```
if 1 then 100 else (200 + 300)
```

et s'afficher ainsi dans le `dumper`.

En cas de problème

En cas de problème, si vous ne trouvez pas par vous-même, envoyez un message décrivant **précisément** :

- votre problème ;
- ce que vous avez essayé ;
- ce que ça a donné et ce que vous attendiez à la place ;

à la liste se202-2017@googlegroups.com.