Introduction à l'écriture d'exercices PL

Dominique Revuz

Bienvenu à Marne la Vallée

Pourquoi

Des étudiants

- Très hétérogènes (culture, pré-requis, méthodes)
- Très Nombreux (Les L1 dans chaque licences)
- Avec de mauvaises stratégies d'apprentissage
- Grands besoins de feedback (réguliers et personnalisés)

Des enseignants

- Pas assez nombreux
- Débordés
- Pas amateurs de correction de copies
- Besoin d'outils de synchronisation pour rationaliser les interventions des grosses équipes pédagogiques

Présentation de PL

- Permettre à des étudiants de s'entraîner avec des exercices calculatoires aléatoires.
- Proposer des exercices ou l'ordinateur calcule les réponses, donne des retours, conseil et valide le travail de l'étudiant.
- L'étudiant a besoin d'évaluations critiques régulières de son travail, pour ainsi pouvoir identifier ses erreurs et apprendre de celles-ci.
- Proposition de feedback (évaluations, critiques, remarques) plus riches qu'une simple évaluation correct/faux. Mise en relief et exploitation globale du travail de l'apprenant (dans le temps, sur plusieurs matières, etc).

Stratégie générale

Les exercices doivent être construits , c'est à dire qu'une partie de l'exercice doit être généré au moment ou il est proposé à l'élève. Nous utiliserons le terme de builder pour désigner le programme qui réalise cette construction.

Deuxièmement nous voulons pouvoir:

- 1. Vérifier la réponse.
- 2. Produire un feedback.
- 3. Modifier la question pour fournir plus d'aide. Pour de la programmation, permettre de corriger les erreurs syntaxiques.
- 4. Produire une note.

Nous utiliserons le terme grader pour désigner le programme qui réalise ces quatre points.

Les exercices

Les exercices sont des **dictionnaires python** :
ensemble (clef, valeur), qui contiennent toute l'information nécessaire à la plateforme pour faire jouer l'exercice à l'élève.

Une fois l'exercice chargé il n'est plus modifiable.
(sauf à être rechargé ce qui peut impliquer des effets de bords)

l'extension

Ces dictionnaires contiennent de nombreuses clefs qu'il est pénible d'apprendre pour l'utilisateur débutant et fastidieux à spécifier systématiquement. C'est pourquoi un système d'extension (héritage) est proposé.

```
# exercice A.pl
text=l'énoncé de l'exercice
form= ...
title=Haha
```

```
# exercice B.pl
extends=A.pl
title=Oh! le B devient un clone du A en 2 clefs.
```

Le dictionnaire résultant

Le dictionnaire de l'exercice B.pl

```
{
"text":"l'énoncé de l'exercice",
"form":"...",
"title":"Oh! le B devient un clone du A en 2 clefs.",
}
```

Info : remarquez que vous n'avez pas de déspécialisation de caractères sauf le passage à la ligne

Info : l'auteur n'a spécifié que deux clefs en écrivant l'énoncé de B mais il a pu possiblement hériter d'un très large contenu situé dans un grand nombre de clefs situées dans A.

Les élements multilignes

```
# ceci est un commentaire
# exercice avec un énoncé multi ligne

text==
l'énoncé de l'exercice
# cette ligne fait partie de l'énoncé
et sera affiché en tant que tel
Qui plus est comme l'énoncé est en markdown/latex
la ligne suivante est un titre avec la fraction a/b !!!
# $$\frac{a}{b}$$
----------------
==

# la ligne contenant uniquement '=='
permet de terminer la balise text
```

Un premier exercice : La somme de deux entiers.

- Connexion à la plateforme, passage par LTI.
- Connectons nous à un LMS client: http://elearning.u-pem.fr / ATTENTION POUR LE TP D'AUJOURD'HUI N'UTILISER PAS VOTRE COMPTE HABITUEL

Login: EssoxLucius1
Password: EssoxLucius1

Objectif demander un calcul a l'élève

```
title=Calcul d'une somme
author=Moi ou Vous
```

La balise **form** permet d'afficher à l'étudiant un formulaire html. Comme ici, nous souhaitons une simple valeur entière, ce formulaire fait partie des formulaires prédéfinis :

```
form=@ /form/simplenumeric.html
```

la notation avec @ permet de lire le fichier /lib/form/simplenumeric.html et d'affecter son contenu à la clef **form**.

D'autre type d'inputs il sont décrit dans la page suivante: https://www.w3schools.com/tags/tag_input.asp

L'énoncé la balise text

Nous voulons un énoncé qui dépend de valeurs numériques, l'exemple suivant définit deux variables a et b et utilise la syntaxe des template html en calculant le **text** résultant du remplacement de **{{a}}** par la valeur de **a**;

```
a=12
b=12
text="Que vaux la somme de {{a}} et {{b}}"
```

Des substitutions automatiques très pratiques pour les contenus dynamiques et aléatoires.

Évaluation

```
title=Calcul d'une somme
author=Moi et Vous
form=@ /form/simplenumeric.html
a=12
b=12
text="Que vaux la somme de {{a}} et {{b}}"
```

Il ne nous reste maintenant plus qu'a évaluer la réponse de l'étudiant. C'est le rôle du grader.

Grader

La ligne suivante permet de charger un **grader** préprogrammé qui est dans la librairie standard /lib/grader/evaluator.py
La syntaxe suivante permet d'ajouter le fichier à l'environnement sécurisé de l'exercice. Dans cet environnement, le fichier s'appellera "grader.py".

```
@ /grader/evaluator.py [grader.py]
```

Ce grader est un peu spécifique ; il utilise une balise **evaluator** dans le pl et c'est celle ci qui contient l'évaluation.

Cette balise **evaluator** doit affecter la variable **grade** avec un couple ([(-1) à 100], "feedback string") où le premier champs est -1 (pas d'évaluation) ou un entier de 0 (tout faux) à 100 (tout juste). Le deuxième champs est un texte qui est affiché à l'utilisateur une fois l'évaluation terminée.

Un exemple d'**evaluator**

Ici une balise mutltiligne contenant du code python.

```
evaluator==
import traceback
import sys

try:
    # evaluation de l'exercice
    if int(response['txt_answer']) == a + b:
        grade = (100, "Bonne réponse")
    else:
        grade = (0, "Mauvaise réponse Bonne réponse : "+str(a+b))
except:
    # si l'élève a fait une faute de frappe
    print(traceback.format_exc(), file=sys.stderr)
    grade = (-1, "Merci de rentrer un entier")
==
```

Tout **Python** est disponible dans un grader/evaluator. L'exercice est ici simple mais les possibilités de feedback sont infinies.

Solution finale

```
title= Calcul de la sommes de deux entiers
author= Moi et Vous
form=@  /form/simplenumeric.html
@ /grader/evaluator.py [grader.py]

a=12
b=12
text=" Que vaux la somme de {{a}} et {{b}} ?"

evaluator==
import traceback
try:
    # evalaution de l'exercice
    if int(response['txt_answer']) == a + b  :
        grade = (100, "Bonne réponse")
    else:
        grade = (0, "Mauvaise réponse Bonne réponse : "+str(a+b))
except:
    # si l'élève a fait une faute de frappe
    print(traceback.format_exc(), file=sys.stderr)
    grade = (-1, "Merci de rentrer un entier")
==
```

Comment rendre l'exercice aléatoire

Nous allons ajouter un **builder** qui utilise la balise **before**.

```
builder=@ /builder/before.py

before==
import random
a=random.randint(3,10)
b=random.randint(3,10)
==
```

Maintenant notre exercice est aléatoire !!

Les inputs

Pour simplifier le travail des développeurs de template nous avons proposer un certain nombre d'inputs préparés, ils sont accessibles dans le répertoire /form de la librairie standard.