QCMs, eforms, Python, pdftk et PyPDF2

I. Pré-requis

Cadre

Afin de profiter au mieux des capacités de python, eforms, pypdf2 et pdftk, il faudra commencer par vérifier que les éléments suivants sont bien installés :

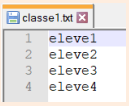
- LaTeX bien évidemment avec les packages pythontex et eforms ;
- python avec le module pypdf2 (via conda ou pip) ;
- pdftk (sa version gratuite suffit).

II. Préparation des fichiers de configuration

II.1. Les fichiers des « classes »

Intro

Pour commencer, il est conseillé de travailler dans un répertoire temporaire (beaucoup de fichiers temporaires et utiles sont à générer...).
Dans ce répertoire, il y aura, pour commencer les fichiers `txt` pour chacune des classes à considérer (chaque fichier contiendra les élèves sous la forme d'un pseudo par exemple).



II.2. Les fichiers de configuration « fdf »

Tuto

L'idée est de préparer le terrain pour créer un fichier QCM par élève, en complétant *par défaut* le champ nomprenom.

Pour cela, il faut déjà configurer les fichiers `fdf` pour chaque élève, et on va utiliser un script python.

Code Python

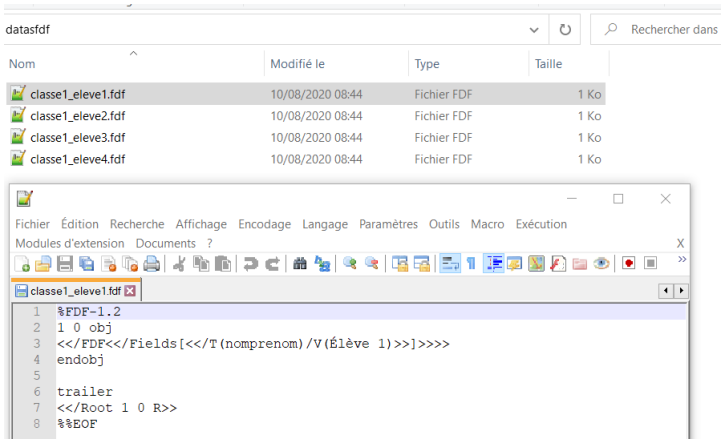
```python
import os
#Données des classes
classe1pseudos = ['eleve1','eleve2','eleve3','eleve4'] #à compléter
classe1noms = ['Élève 1','Élève 2','Élève 3','Élève 4'] #à compléter
#puis classe2... classe3...
#script de génération des fichiers fdf
def genfdf(classe):
    if not os.path.exists('datasfdf'):
        os.mkdir('datasfdf')
    if classe == 'classe1' :
        listepseudos,listenoms = classe1pseudos,classe1noms
    #elif ...
    n = len(listepseudos)
    for i in range(n):
        file = classe + '_' + listepseudos[i] + '.fdf'
        f = open('datasfdf/'+file, "w")
        f.write('%FDF-1.2\n')
        f.write('1 0 obj\n')
        f.write('<</FDF<</Fields[<</T(nomprenom)/V('+ listenoms[i] + ')>>]>>>>\n')
        f.write('endobj\n')
        f.write('\n')
        f.write('trailer\n')
        f.write('<</Root 1 0 R>>\n')
        f.write('%%EOF')
    f.close()
    return
```

Illustration

Le script est à charger (l'exemple est donné pour la classe1) :

- directement dans l'IDE en lançant ensuite la commande `genfdf('classe1')` ;
- en utilisant l'invite de commandes et en lançant la commande
  `python -c "import fdfcreate ; fdfcreate.genfdf('classe1')"`.

Les fichiers fdf créés (une fois pour toute) ont la structure suivante, qui permettra d'insérer dans le champ <*nomprenom*> la valeur indiquée :



II.3. Les codes python et LaTeX pour les réponses placées aléatoirement

Tuto

On peut commencer par déclarer les commandes pour créer les boutons via eforms :

Code LaTeX

```
\usepackage{eforms}
%1 choix parmi 4, on ne peut pas décocher (plus le bouton n°5 pour la non réponse !)
\newcommand\choixqcmu[2]{\radioButton[\V{E}\DV{E}]{#1}{1em}{1em}{#2}}
\newcommand\choixqcmunochoice[1]{\radioButton[\V{E}\DV{E}]{#1}{1em}{1em}{E}}
```

On continue par le code générique python relatif au placement aléatoire :

Code LaTeX

```
\usepackage{pythontex}
%Code Python pour le mélange des réponses
\begin{pycode}
def qcmalea(a,b,c,d):
    from random import shuffle
    reponses = [a,b,c,d]
    shuffle(reponses)
    return reponses

#éventuellement une version à 3 réponse def qcmalea3()...

#Question1
Q1A = r"\choixqcmu{Q1}{A}~~blablareponseA"
Q1B = r"\choixqcmu{Q1}{B}~~blablareponseB"
Q1C = r"\choixqcmu{Q1}{C}~~blablareponseC"
Q1D = r"\choixqcmu{Q1}{D}~~blablareponseD"
reponsesQ1 = qcmalea(Q1A,Q1B,Q1C,Q1D)

#Question2
Q2A = r"\choixqcmu{Q2}{A}~~blablareponseA"
Q2B = r"\choixqcmu{Q2}{B}~~blablareponseB"
Q2C = r"\choixqcmu{Q2}{C}~~blablareponseC"
Q2D = r"\choixqcmu{Q2}{D}~~blablareponseD"
reponsesQ2 = qcmalea(Q2A,Q2B,Q2C,Q2D)

#etc...
\end{pycode}
```

Et de ce fait le code LaTeXpour générer le champ <*nomprenom*> et (par exemple) le tableau des réponses avec cases à

cocher :

Code LaTeX

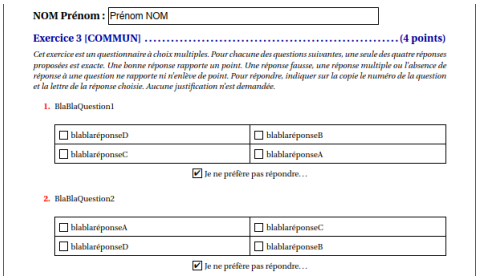
```
%La gestion du champ <nomprenom>
{\Large NOM Prénom : }~{\textField[\V{À remplir}\BC{0 0 0}\textSize{12}]{nomprenom}{12cm}{20pt}}

%Un exmeple de question avec les réponses mélangées :
BlaBlaQuestion1

\begin{center}
    \renewcommand{\arraystretch}{1.75}
    \begin{tabularx}{\linewidth}{|X|X|}
        \hline
        \py{reponsesQ1[0]} & \py{reponsesQ1[1]} \\ \hline
        \py{reponsesQ1[2]} & \py{reponsesQ1[3]} \\ \hline
        \multicolumn{2}{c}{\choixqcmunochoice{Q1}~~Je ne préfère pas répondre\dots} \\
    \end{tabularx}
\end{center}
```

Illustration

Avec le logiciel Adobe Acrobat Reader (pour les autres il faut vérifier que les cases se cochent bien et mettent à jour les champs !) on obtient le rendu suivant :



III. Génération des fichiers élèves et analyse des résultats

III.1. Fichiers élèves

Tuto

On peut maintenant générer les fichiers élèves à l'aide du script generateqcmeleves.bat qui est placé dans le répertoire contenant le fichier tex à « dupliquer » :

Code Shell

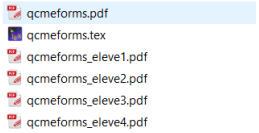
```
set fichier=%1
set classe=%2
for /F %%x in (%classe%.txt) DO (
    copy %fichier%.tex %fichier%_%%x.tex
    pdflatex.exe --shell-escape -synctex=0 -interaction=batchmode  %fichier%_%%x.tex
    pythontex --rerun=always %fichier%_%%x.tex
    pdflatex.exe --shell-escape -synctex=0 -interaction=batchmode  %fichier%_%%x.tex
    del %fichier%_%%x.tex
    )
```

On lance un terminal (ou une invite de commandes) et on lance la commande :

SHELL `generateqcmeleves.bat <nomfichiersansextension> <nomclasse>`.

Illustration

On obtient de ce fait un fichier par élève :



Tuto

On peut maintenant insérer les données <nomprenom> dans chaque fichier élève à l'aide d'un fichier createqcms.bat qui va chercher les données fdf et les insérer dans les fichiers pdf :

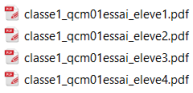
Code Shell

```
set /p fichier="Nom du fichier pdf : "
set file=%fichier:.pdf=%
set /p niv="Choix de la classe : "
set /p dest="Masque du fichier de sortie : "
mkdir out
for /F %%e in (%niv%.txt) DO (
    pdftk %file%_%%e.pdf fill_form datasfdf\%niv%_%%e.fdf^
    output out\%niv%_%dest%_%%e.pdf need_appearances)
```

On lance une invite de commandes et on lance la commande `createqcms.bat` et on répond aux questions.

Illustration

On obtient désormais un fichier personnalisé par élève :



III.2. Analyse des résultats

Tuto

Une fois les fichiers remplis par les élèves et récupérés, on peut lancer l'analyse avec un script python.

Code Python

```python
import PyPDF2, glob, sys, os

def listereponses(nb,fichier):
    freponse = PyPDF2.PdfFileReader(fichier)
    reponses = freponse.getFields()
    liste_reponses = ''
    nom_copie = reponses['nomprenom']['/V']
    for i in range(1,nb+1):
        num = 'Q'+str(i)
        rep = reponses[num]['/V'][1:]
        if rep == 'E':
            liste_reponses = liste_reponses + '-'
        else :
            liste_reponses = liste_reponses + rep
    return nom_copie,liste_reponses

def note(reponses,bonnesreponses,ptok=1,ptnok=0):
    note = 0
    for i in range(len(bonnesreponses)):
        if reponses[1][i] == '-':
            note += 0
        elif reponses[1][i] == bonnesreponses[i]:
            note += ptok
        else :
            note += ptnok
    return reponses[0],reponses[1],note

def bilan(bonnesreponses,ptok=1,ptnok=0):
    nbquestions = len(bonnesreponses)
    classe = [f for f in glob.glob("*.pdf")]
    f = open("bilan.txt", "w")
    f.write('fichier;nom;réponses;note\n')
    for fichier in classe :
        res = note(listereponses(nbquestions,fichier),bonnesreponses,ptok,ptnok)
        ligne = fichier + ";" + res[0] + ";" + res[1] + ";" + str(res[2])
        f.write(ligne + '\n')
        print(f"fichier={fichier} ; nom={res[0]} ; réponses={res[1]} ; note={res[2]}")
    f.close()
```

On peut maintenant exécuter le script python :

- directement dans l'IDE avec la commande `bilan('<bonnesréponses>',ptok,ptnok)` ;
- via l'une invite de commandes dans le répertoire contenant les fichiers récupérée, grâce à la commande `python -c "import recupdatapdf ; recupdatapdf.bilan('<bonnesréponses>',ptok,ptnok)"`.

Le fichier réponse est ainsi créé et affiché.

Illustration

Ci-dessous un test avec :

- des copies de 4 élèves remplies,
- une analyse avec 1 point par bonne réponse et 0 sinon ;
- une autre avec 1 point par bonne réponse et -0.5 sinon.