Les ppltp

Ou les pltp programmables.

Premierement compatibles avec les pltp.

Un ppltp est un fichier au format pl:

- des balises définie par les opérateurs @/=/==/etc cf la documentation.

avec une balise information, des déclaration de pl l'état actuel est des lignes de la forme

```
@ url
```

par défaut cela créer une variable plID ou ID est l'indice de la déclaration dans le fichier : pl0 pl1 etc.

Comme c'est pas commode la syntaxe suivante permet d'avoir un ordre quelconque pour les fichiers:

```
@ url [variablename]
```

enfin la syntaxe suivante permet de créer une liste de pl nommée **name** (ou une syntaxe équivalente FIXME @mcisse) `name@@ url url url ... @@`

Enfin la balise **runner** (je n'ai pas fait mon choix encore) contient le "code" de l'activité.

La balise runner est un script python qui est exécuté dans un environnement protégé (pyenv réduit ou sur la sandbox) et qui n'a pas accès directement au code django (éventuellement peut obtenir des informations en utilisant une api restfull avec les bon droits i.e. je ne peux accèder qu'aux information de l'utilisateur et des assets du cours.) FIXME un jail peut être ??

Le script est lancé avec comme dictionnaire global le dictionnaire produit par le fichier pltp, il est ainsi possible d'initialiser des variables.

Dans le script tous les pl qui ont été chargés (au moment de la publication de la ressource en asset), sont visible sous la forme d'une instance de la classe **Activity** .

Cette classe à des methodes suivante qui vont nous permettre de programmer des activités complexes basées sur des activités plus simples.

```
Class Activity:

  def score():
 """ Returns the score of the activity. If the activity is a pl this is direct.
    Else the activity must define an score base on the sub activities used in it.
    Default value : If there is a global variable score in the pltp use the value stored there, else re
    The global variable is of the responsability of **runner** code in the case of a ppltp.
  """

  def status():
  """ Returns the status (sucess, failed, not started, stopped/abandonné).
         If the activity is a pl this is direct.
         Else the activity must define an status base on the sub activities used in it.
         Default value : If there is a global variable status in the pltp use the value stored there,
           else return not started.
       The global variable is of the responsability of **runner** code in the case of a ppltp.
       """

  def exec(rerun=False):
    """ if rerun==False and if the status of the activity is success or failed.
        Return immedialty with True or False.
        ELSE The current activity yields to this smaller activity.
```

```
        The activity is loadded
        Bread crumps are updated.
        This is the new Bottom Activity.
    """
   def init():""" prepare the activity for execution see @pavell9000 """
   def start():""" idem """
```

The code of the **runner** is state less. Meaning you should write your code thinking it will be restarted each time the activity is activated.

```
# pl1 ... pl10 where declared in the ppltp

if not pl1.exec() and not pl8.exec() : # if pl 1 not valide do pl 8 !
  notification(current_teacher,currentstudent, " problème avec le concept bascule ")

while pl2.score() + pl3.score() + pl4.score() < 100: #
  pl2.exec();
  pl3.exec();
  pl4.exec();
```

Has you can see on this exemple as exec replace the current activity. We need to rerun the **runner** script to knows witch is the next sub activity to run (a plX in this case).