Les environnements de tests unitaires existent pour la plupart des langages et plusieurs peuvent exister pour un même langage (CPPUnit, google test pour C et C++ par exemple). Dans ce TP vous allez mettre en oeuvre 3 environnements de tests pour 3 langages : Java, python et C.

Cette partie 2 du TP vous propose deux environnements de tests unitaires pour Python et C.

1 CUnit : environnement de test pour C

1.1 Présentation rapide

CUnit permet de faire des tests en C mais des environnements plus évolués existent pour traiter C++ (et C) comme CPPUnit et googletest.

Sous Ubuntu, le package installant CUnit fait partie de la distribution, mais on peut aussi l'installer sur son compte indépendamment et l'utiliser comme bibliothèque C.

1.2 Partie CUnit du TP

La bibliothèque CUnit n'est pas installée et vous devez donc l'installer sur votre compte. La version à installer est **CUnit-2.1-2** qui n'est pas la dernière dont l'archive n'est pas complète. Voir le fichier `CUnit-Louvain.pdf` (provenant de l'université de Louvain) pour une installation et son utilisation basique. Remarque : les packages sont dans la distribution Ubuntu et s'installent avec l'installateur standard si vous etes administrateur de votre machine (Un document sur l'utilisation de bibliothèques C est dans la partie biblio du cours).

Travail à faire : programmer les fonctions `typeTriangle` et `readData` dans un fichier `triangle.c` et une fonction main dans un fichier `mainTriangle.c` permettant de l'utiliser. La fonction `readData` renverra une structure C dont les 3 champs sont des floats correspondant aux valeurs des cotés. Dans le cas de fichiers non conforme ou inexistant, la fonction affectera -1 à chaque champ.

Ecrire les suites de tests `testTypeTriange.c` et testReadData.c et utiiser CUnit pour voir si vos fonctions passent les tests. Le site donne une documentation complète mais vous pouvez allez voir un exemple d'utilisation basique ici `http://wpollock.com/CPlus/CUnitNotes.htm`.

2 Unittest : environnement de test pour Python

2.1 Présentation rapide

Un équivalent de JUnit pour python est le package `unittest`. Pour vous rappeler les bases de python, vous pouvez aller ici : `https://docs.python.org/fr/3.5/tutorial/`.

2.2 Partie Unittest du TP

`unittest` est un package python similaire à JUnit. Pour écrire une suite de test on crée un classe de test qui contient une suite de tests. Par exemple :

```
class EssaitTest(unittest.TestCase):

    def testGetVal(self):    #un test
        e = Essai(1.0)
        val = e.getVal()
        expected = 1.0
        self.assertEquals(expected, val)

if __name__ == '__main__':  # ce qu'il faut ajouter pour que les
    unittest.main()  # tests soient effectu?s sous l'interpréteur
```

Les deux dernières lignes ne font pas partie de la classe mais sont nécessaire pour l'exécuter dans l'interpréteur (facultatives sous certains IDE).

Pour commencer reprogrammer la classe Essai en python et réécrire la suite de test JUnit avec unittest (classe `EssaiTest`) et l'exécuter.

Ecrire les deux classes de tests pour les méthodes `readData(filename)` qui renvoie un triplet `(a,b,c)` et `TypeTriangle`.

3 Rendu des TPs

3.1 Modalités

`A respecter absolument`

— Chaque groupe de TP rendra une archive au format zip appelée `TPTEST-PARTIEII-TPi.zip` (avec i le numéro du groupe) à déposer dans l'activité `Devoir Rendu TP TEST` du site AMETICE **à la fin du TP**.
— La décompression de l'archive créera un répertoire TPTEST-PARTIEII-TPi
— Chaque fichier source contiendra une en-tete qui est un commentaire avec les noms des membres du groupe et tout autre information jugée utile.

3.2 Travail demandé

L'archive zip contiendra un répertoire C avec les fichiers `triangle.c` (fonctions `typeTriangle` et `ReadData`, `triangleMain.c` (le programme principal), les .h correspondants et `testTypeTriangle.c`, `testReadData.c` et un répertoire PYTHON avec les classes correspondantes pour PYTHON.