3A STI : TP Tests en Python

Consignes générales

— Vous aurez besoin de votre éditeur préféré en Python et de l'outil Coverage pour python
— Vous rendez vos rapports pour le contrôle continu sur Celene, avant 19h le jour du TP, sous forme d'archive contenant un rapport en pdf ainsi que tout code utile.
— Le TP a été préparé en utilisant pycharm 2020.3, mais la procédure générale doit continuer à être correcte quel que soit votre environnement python.
— Le contenu du TP est prévu sur 2 séances et sera trop long pour une seule, ne stressez pas parce que vous ne l'avez pas fini au bout d'une heure 20 !

1 Premier pas : partage de code et tests boîte noire

Un de vos collègues (John) a écrit une librairie de mathématique. Il est fourni sur Celene.

La librairie contient des implémentations de la recherche du plus petit commun multiple (ppcm en français, lcm en anglais, **voir la page Wikipédia**). Il y a trois versions : la correcte, l'optimisée et celle qui ne fonctionne pas ...

Vous allez devoir tester chacune des fonctions disponibles en utilisant Unittest. Unittest est un outil Python qui permet de mettre en place des tests unitaires.

1.1 Comment créer un test unitaire avec Unittest

C'est très simple (sous Pycharm 2020.3) :
— clic droit sur le nom de la fonction
— Go to ... choisir test
— Et hop un test par défaut est crée avec une assertion dedans.
— Une fois codé il ne reste plus qu'à lancer le fichier pour avoir les résultats des tests.

Pycharm vous a généré une fonction de test. Cette fonction contient une assertion par défaut. L'assertion, c'est ce qu'on vérifie pour savoir si un test c'est bien passé ou non, typiquement on va comparer un résultat avec un résultat attendu.

C'est dans chacune de ces fonctions que vous devez initialiser vos données et configurer les différents paramètres pour pouvoir tester ce que vous voulez. Il suffit donc de tester que les résultats sont ceux attendus. Cela se fait par exemple par la fonction assertTrue() :

```
self.assertTrue(0 == lcm_better(0, 0))
```

QUESTION
Implémentez pour chacune des fonctions pour différents cas à tester :
— lancer ppcm de (0,0) doit donner 0.
— lancer ppcm de (0,X) doit donner 0.
— lancer ppcm de (X,0) doit donner 0.
— lancer ppcm de (X,X) doit donner X.
— lancer ppcm de (168, 60) doit donner 840.

Sauvegardez votre code dans un coin pour le rendre à la fin des TP. Observez les résultats et écrivez dans votre rapport :
— Les différents codes fonctionnent-ils correctement ?
— D'après vous, en terme de couverture, vos tests sont-ils exhaustifs ?

2 Couverture de code

Couverture de tests et graphe de contrôle

Quel que soit le type de test, celui-ci repose toujours sur le même principe :
— On fixe les données d'entrées
— On exécute le composant
— On vérifie que le comportement est attendu (ce qui est souvent fait par quelque chose qu'on appelle *l'oracle*).

Un test va donc permettre de tester pour un jeu de données d'entrée le comportement du composant. Cela signifie donc que pour tester le composant totalement, il faudrait tester l'intégralité des jeux de données possibles ! Heureusement, quand on fait des tests "boîte blanche" on a accès au code et on peut vérifier si l'on parcours tous les chemins possibles d'exécution ou si l'on teste bien toutes les fonctionnalités d'un programme.

On appelle *couverture* la portion du programme qui a été testé. Il existe bien sûr plusieurs types de couverture suivant ce que l'on teste : la couverture structurelle qui s'assure que l'on passe bien par tous les chemins d'exécutions possible et la fonctionnelle, qui s'assure que l'on teste bien toutes les fonctionnalités d'un programme.

Un outil habituel pour voir cette couverture est le graphe de flot de contrôle qui permet de voir les différentes branches d'un programme, dont voici un exemple à la figure 1.

Chaque bloc d'instruction y est représenté comme un noeud, chaque branchement conditionnel comme un arc (on parle de branche). La couverture peut être calculée par instruction, par branche, par chemins avec itérations, par flux de données. Par défaut dans ce document, nous nous intéresserons à la couverture par branche.

Pour visualiser le parcours d'un programme, il existe aussi des outils, comme le plugin eclEmma sous Eclipse en java, ou l'outil Coverage en python.

Nota Bene : Python n'est pas un langage destiné principalement à de la production de logiciel de grande taille et à la vie longue, les outils de test sont plus basiques que ceux existant pour les autres langages comme C, C++, Java, etc.

2.1 Graphe de contrôle

John a donc des erreurs dans son code.

QUESTION : Pour chacune des 3 fonctions, dessinez les graphes de contrôle des méthodes et incluez le dans votre rapport.

2.2 Visualisation de la couverture des tests avec coverage

Coverage est un paquet python pour vous permettre de regarder un peu la couverture de code des vos tests. **Tout est décrit dans cet excellent petit tutoriel.**

Quand vous choisissez html, un text higlighting basé sur la couverture par les tests : quand un bloc d'instruction n'est pas couvert, il est coloré en rouge.

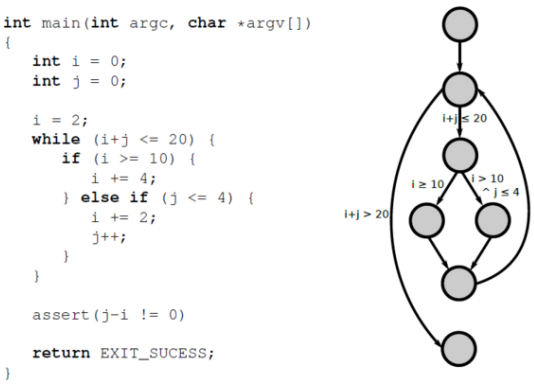

FIGURE 1 – Exemple de flot de contrôle (copyright Attribution-Share Alike 3.0 Unported ENSIMAG

QUESTION : L'intégralité de votre code est-il couvert par les tests ? Pour quelles méthodes des branches ne sont pas couvertes ? Lesquelles, pourquoi ?

QUESTION : Ecrire des tests qui permettent de couvrir tous les branchements de chacune des 3 versions.

2.3 Faisons évoluer le code

En fait, si le code remplit bien les spécifications, celles-ci ne suivent pas la définition du ppcm, qui ne doit pas être défini quand l'un ou l'autre des paramètres sont nuls. Il faut ajouter au code de John une exception, qui doit être levée quand cette condition est remplie. Pour cela, il faut utiliser la syntaxe suivante :

```
 def test1(self):
        self.assertRaises(SomeCoolException, mymod.myfunc)
}
```

QUESTION :

— Ajoutez l'exception, modifiez le code et les tests en conséquence pour que les méthodes passent correctement les tests.