TP tests & bugs

@crédits du TP
Antoine MAROT (antoine.marot@ensemblescolaire-niort.com)
David SALLÉ (david.salle@ensemblescolaire-niort.com)
Julien SIMONNEAU (julien.simonneau@ensemblescolaire-niort.com)

Version du document v0.1
Date 15/10/2020

Table des matières

1 - Introduction

Les activités pratiques qui suivent sont en lien avec le document : **cours_tests_bugs.pdf**.
Elles ont pour objectif de vous faire appliquer les notions présentées dans le document de cours :

- les exceptions
- les bogues
- les tests unitaires
- les performances

2 - PizzaError

L'entreprise Pizza des conches souhaiterait créer une sorte de "drive" automatique pour ses clients afin qu'ils puissent commander en ligne une pizza selon les stocks disponibles.
Un stagiaire a déjà commencé l'écriture d'un programme prototype. Ce dernier est disponible sur Moodle : **pizza_drive.py**
Cependant comme vous allez pouvoir le constater, le programme est largement perfectible :

- mauvaise gestion des saisies (caractères VS entier)
- commande de pizza malgré les stocks vides
- plantage lorsque l'on commande la pizza 3

A l'aide des exceptions, rendez ce programme plus robuste et plus ergonomique pour le client, c'est à dire sans plantage inopiné.

3 - Chasse aux bogues

Téléchargez et décompressez l'archive **bogues.zip**.
Pour chacun des scripts :

- trouvez l'origine du bogue
- proposez un correctif pour le rendre fonctionnel

4 - Tests unitaires

A votre tour de mettre en œuvre les tests unitaires :

- sur une classe Rectangle à réaliser
- sur les classes Carte, JeuCarte Pile, File réalisées précédemment

Vous utiliserez la méthode simple avec le mot clef assert, puis la librairie unittest dans un second temps.

4.1 - Classe Rectangle

L'entreprise GEOFORME développe des logiciels et outils de dessin industriel.
Le développeur avant vous dans le service a commencé à développer une classe Rectangle disponible sur Moodle dans le fichier **rectangle.py**. Il a principalement défini l'API de la classe et positionné des TODO comme ci-dessous dans chaque méthode :

```
# TODO : compléter le code de la méthode
pass
```

Si vous détectez des problèmes potentiels, vous pourrez les signaler en levant une Exception comme ci-dessous :

```
# Un problème potentiel...
if probleme == True:
    raise Exception("Un problème est survenu")
```

Le responsable du service vous demande de **finir de coder** cette classe et d'**écrire les tests unitaires** qui permettront de la valider et de l'intégrer dans le reste du logiciel de dessin industriel.
Concernant les tests unitaires, vous créerez 2 versions et donc 2 fichiers :

- **tests_rectangle_assert.py** : version avec le mot-clef assert
- **tests_rectangle_unittest.py** : version avec la librairie unittest

4.2 - Classes du projet solitaire

Appliquer les tests unitaires sur les classes du projet solitaire Pile, File, Carte et JeuCarte.

5 - Optimisation et performances

5.1 - Chasse aux performances

Johnny BEGOOD participe en ce moment à un challenge algorithmique sur une plate-forme en ligne. Pour le problème n°10, il doit écrire un programme qui affiche le 2000ième nombre premier. Il a écrit un code qui fonctionne parfaitement bien (le résultat est 17389), mais la plate-forme a fixé une limite de temps d'exécution (<100ms) pour valider le challenge et son programme est trop lent.

5.1.1 - Chronométrage

Téléchargez son code source sur Moodle : **programme_v00.py**
A l'aide de la fonction **perf_counter()** du module **time**, mesurez et affichez le temps d'exécution de son programme et plus particulièrement de sa fonction **rechercher_nieme_nombre_premier()**.
Notez le résultat moyen obtenu sur 3 exécutions et confirmez le refus par la plate-forme.

5.1.2 - Première optimisation idiomatique

Harry a entendu parler qu'en Python les boucles **for** étaient plus performantes que les boucles **while**.
Créer un nouveau script **programme_v01.py,** copiez/collez le code de la version v00 et modifiez le code de sa fonction **etre_premier()** pour remplacer la boucle **while** par une boucle **for**.
Chronométrez le temps d'exécution obtenu et notez le résultat moyen sur 3 lancement. Comparez le temps entre les versions v00 et v01. Est-ce suffisant ?

5.1.3 - Seconde optimisation algorithmique

Après avoir lu la page wikipédia consacrée aux nombres premiers, il découvre que le plus grand facteur possible pour un nombre non premier N est
Créez un nouveau script **programme_v02.py** à partir de la précédente version et modifiéz la limite de la boucle for de n-1 à
Chronométrez cette nouvelle version comme précédemment et conclure.

5.1.4 - Profilage

Effectuez un profilage du script de départ programme_v00.py et notez les résultats :

- pour la fonction **rechercher_nieme_nombre_premier()** :
  - nombre d'appels
  - temps moyen d'exécution
  - temps total d'exécution
- pour la fonction **etre_premier()** :
  - nombre d'appels
  - temps moyen d'exécution
  - temps total d'exécution