Tests

Un programme dont la syntaxe est correcte peut toutefois comporter encore des erreurs, soit qu'elle proviennent de l'algorithme traduit, soit qu'elles aient été rajoutées lors de la phase de traduction. Il faut donc tester tout programme avant de l'utiliser au sein d'un ensemble plus complexe.
La phase de test :

- sert à détecter certaines erreurs, à vérifier l'adéquation au cahier des charges
- peut être manuelle ou automatique
- peut être statique ou dynamique

Un "bon" test est concis, permet de bien détecter les fautes. Un test peut être unitaire ou tester l'intégration ou la non-régression . Il peut tester le bon fonctionnement des fonctions (qu'on soumet à des données d'entrée valides) ; ou la robustesse (se fait avec des données d'entrée invalides).
On s'intéresse en particulier aux tests `fonctionnels`, dits "tests en boîte noire" : peu importe comment l'algorithme est conçu, on en observe les réponses attendues pour diverses entrées. Ces tests ne permettent pas de corriger les défauts de programmation (comme une boucle inutile, des tests redondants, ...). Il existe également des tests `fonctionnels`, dits "en boîte blanche", où l'architecture du programme est connue et représentée sous forme de graphe.

Exemple

Quel jeu de test proposer pour tester l'algorithme renvoyant le PGCD ?

On teste de manière à couvrir la **spécification**. On peut ajouter au partitionnement d'équivalence et au test aux limites des tests aléatoires ou combinatoires.

Partitionnement d'équivalence

Pour vérifier si un programme fait ce qu'on attend de lui, on ne le teste pas sur toutes les entrées possibles, mais on ordonne ces entrées en classe d'équivalence, et on propose un jeu de test pour chaque classe, le but étant de réduire le nombre de tests tout en gardant une exigence maximale.

Exemple

Soit une application de calcul du prix d'un forfait de ski.

- Une journée pour un adulte coûte 100 euros.
- Pour plus de 5 jours, une remise de 10% est accordée sur chaque jour supplémentaire ; à partir de 10 jours, il y a une remise de 15% par jour supplémentaire (et non plus de 10%)
- Un enfant paye moitié moins qu'un adulte
- Pour les familles, une remise de 15% est accordée sur le forfait de chaque enfant à partir du troisième

Quel(s) partitionnement(s) proposer ?

Tests à la limite

On cherche à éviter les erreurs dues à la proximité avec la limite des classes d'équivalence, en testant spécifiquement les entrées qui se trouvent aux limites, et celles qui sont de part et d'autre de ces limites.

Exemple

1. Reprendre l'exemple précédent.
2. Que tester dans le cas de l'algorithme du pgcd ?

Travaux pratiques

DM vacances d'hiver

Représenter un rubik's cube, à l'état ordonné, ou après une série précise de mouvements.

```
from matplotlib import patches
from matplotlib import pyplot

# taille de la fenêtre
largeur = 6
hauteur = 6
figure = pyplot.figure(figsize = (largeur, hauteur))

# création de l'objet sur lequel on dessine
axes = figure.add_subplot(111)

# Suppression des axes gradués et du cadre
axes.set_frame_on(False)
axes.xaxis.set_visible(False)
axes.yaxis.set_visible(False)

#dessin d'un rectangle
abs = 0.3
ord = 0.5
largRect = 0.5
hautRect = 0.2
axes.add_artist(
    patches.Rectangle((abs,ord), largRect, hautRect,
                      edgecolor = 'black', facecolor = 'blue',
                      fill = True,
                      linewidth = 1))


pyplot.show()
```

1. Représenter un Rubik's Cube 3x3 sous la forme de votre choix *Choix de l'encodage en Python, choix de la représentation visuelle.*
2. Écrire des fonctions de rotation : par exemple `gauche(rubik)` prend un rubik's cube tenu d'une manière fixe (on peut considérer que les centres des faces restent fixes), et lui fait subir une rotation de la tranche à gauche d'un quart de tour dans le sens trigonométrique.
3. Écrire une fonction prenant une liste de rotation, et permettant de visualiser le Rubik's cube après application de ces rotations.
4. *Écrire une fonction résolvant un Rubik's Cube mélangé.*