Aix*Marseille université
Initiative d'excellence
Année 2018-19

Diplôme Inter-Universitaire
Enseigner l'informatique au lycée

Bloc 1 : TP 3

Trouvez l'erreur !

Ce TP a pour objectif de vous faire chercher des erreurs dans des algorithmes, écrits en Python.

1 Jouer avec les variables

Question 1. Chacun des programmes suivants admet une erreur : trouvez-la de tête avant de vérifier votre hypothèse sous Python.

```
def fonction1(n : int) -> int:
  a = 1
  while a < n :
    b = 2*b + 1
    a = a+1
  return b

def fonction2(n : int) -> int:
  b = 1
  while n != 0 :
    b = 2*b
    n = n-2
  return b

def fonction3(n : int) -> int:
  a = 1
  b = 1
  while a < n :
    b = b + a
  return b
```

2 Factorielle

Un des premiers programmes que l'on peut écrire consiste au calcul de la factorielle d'un entier $n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$.

Question 2. Parmi les algorithmes suivants, lesquels sont corrects ? Après avoir donné une réponse de tête, évaluer les programmes sous Python à l'aide d'une batterie de tests bien choisis : un programme sera alors déclaré *incorrect* dès lors qu'au moins un des tests est falsifié, et *probablement correct* dès lors que tous les tests sont validés.

```
def factorielle1(n : int) -> int:
  i = 0
  f = 1
  while i <= n :
    i = i+1
    f = f*i
  return f
```

```python
def factorielle2(n : int) -> int:
  i = 0
  f = 1
  while i < n :
    i = i+1
    f = f*i
  return f

def factorielle3(n : int) -> int:
  f = 1
  for i in range(1,n+1):
    f = f*i
  return f

def factorielle4(n : int) -> int:
  f = 1
  for i in range(n):
    f = f*i
  return f
```

Question 3. Pour le ou les algorithmes que vous pensez être correct, pouvez-vous convaincre un interlocuteur en lui fournissant un argument, par exemple sous la forme d'un *invariant de boucle* ? Nous reverrons en détail les preuves d'algorithmes cet après-midi.

3 Que fait ce programme ?

Question 4. Un élève a produit un drôle de programme... Saurez-vous trouver ce qu'il renvoie ? Utiliser des tests en Python pour se donner une idée, puis essayez de trouver des arguments pour valider votre conjecture.

```python
def mystere1(n : int, m : int) -> int:
  i = m
  while i < n :
    i = i+1
  return i
```

Question 5. Même question avec le programme suivant du stagiaire qui vient d'arriver au laboratoire...

```python
def mystere2(n : int) -> int:
  a = 0
  s = 1
  t = 1
  while s <= n :
    a += 1
    s += t+2
    t += 2
  return a
```

4 Bug de Zune

Après ces petites mises en bouche, passons au plat principal. Il s'agit d'un bug rencontré par les lecteurs MP3 Microsoft Zune, le 31 décembre 2008 : le matin de ce jour-là, les utilisateurs voulant allumer leur appareil

FIGURE 1 – Page de démarrage gelée du lecteur Zune le 31 décembre 2008

ont eu la désagréable surprise de découvrir l'écran de démarrage restant gelé comme dans l'image ci-dessus. Impossible pour eux d'utiliser leur lecteur. Le plus *drôle* dans l'histoire est que rien ne pouvait être fait pour régler ce problème, mais que le lendemain matin, le 1er janvier 2009, tout s'est remis à fonctionner comme si rien ne s'était jamais passé. Une mise à jour du logiciel a évidemment ensuite été mise en ligne pour corriger le bug.

Mais quel est donc ce bug étrange ? Comme on peut le deviner du fait du redémarrage sans souci le lendemain, le bug se situe dans la partie du code en charge des fonctions calendaires du lecteur. Microsoft a rapidement trouvé l'origine du bug qu'ils ont depuis publié : la faute provient de 10 lignes de code écrites originellement en C, traduite en Python pour l'occasion :

```
year = 1980
while days > 365:
    if is_leap_year(year):
        if days > 366:
            days -= 366
            year += 1
    else:
        days -= 365
        year += 1
```

Ces quelques lignes utilisent une variable globale `days` qui est initialisée, lors du démarrage du lecteur, au nombre de jours écoulés depuis le 1er janvier 1980 (le début des temps pour Microsoft, contrairement aux systèmes Unix qui utilisent la date du 1er janvier 1970 comme *epoch*, c'est-à-dire comme date initiale à partir de laquelle est mesuré le temps par un système d'exploitation). Le principe du code est assez simple, et utilise un prédicat `is_leap_year` qui renvoie `True` si l'année est bissextile et `Faux` sinon.

Question 6. Commencer par écrire ce prédicat `is_leap_year` en se rappelant qu'un numéro d'année est bissextile si elle est divisible par 4 sans être divisible par 100, ou si elle est divisible par 400.

On peut donc maintenant tester le code du Zune, en attribuant préalablement une valeur à la variable `days`.

Question 7. Trouver ainsi une valeur de `days` où le programme *bugge*. Décrire le comportement du programme dans ce cas et expliquer le comportement observé du lecteur Zune le matin du 31 décembre 2008.

Question 8. Proposer finalement une correction du bug de Zune. Tester à nouveau votre programme pour s'assurer qu'il ne présente a priori plus d'erreur.

Morale de l'histoire : tester (à défaut de prouver la correction de) son code est crucial, même pour des petits programmes aussi indolores que le calcul de la date !

5 Un algorithme de tri ?

Nous allons reparler sous peu de tris dans cette formation. Avant de revoir les tris classiques, étudions un *tri* spécialisé pour les permutations de $\{0, 1, , \ldots, n-1\}$ avec n un entier naturel. On souhaite donc trier une liste de taille n contenant une permutation des éléments $\{0, 1, , \ldots, n-1\}$. Évidemment, cette tâche semble n'avoir aucun intérêt, puisqu'on connait d'avance le résultat de ce tri... mais que voulez-vous, comme le dit la devise Shadok : "Pourquoi faire simple quand on peut faire compliqué ?"

Notre stagiaire préféré a donc imaginé un algorithme des plus étranges... Pour chaque valeur i de $\{0, 1, , \ldots, n-1\}$, dans l'ordre croissant, il cherche l'élément i dans le tableau, et le déplace de $i+1$ cases vers la droite, en échangeant de manière répétée le contenu d'une case avec celui de sa voisine de droite : les échanges s'arrêtent dès lors que i atteint la dernière position de la liste.

Question 9. Implémenter cet algorithme.

Question 10. Tester le programme sur les exemples suivants :

— une liste vide ;
— une liste de taille 1 ;
— une liste contenant la permutation identité, pour plusieurs valeurs de n ;
— une liste contenant la permutation miroir $i \mapsto n-1-i$, pour plusieurs valeurs de n
— une liste contenant la permutation identité sur la première moitié de la liste et la permutation miroir sur la seconde moitié, pour plusieurs valeurs de n.

Question 11. Continuer l'écriture de tests pour réussir à trouver des listes que l'algorithme ne parvient pas à trier.

6 Recherche dichotomique

Voici un algorithme de recherche dichotomique dans un tableau trié écrit de manière récursive.

```
def indice_par_dichotomie(liste, element, gauche=0, droite=inf):
    if droite >= len(liste):
        droite = len(liste)
    if gauche >= droite:
        return False
    milieu = (gauche+droite)//2
    if liste[milieu]==element:
        return True
    if liste[milieu]>element:
        return indice_par_dichotomie(liste,element, gauche, milieu-1)
    return indice_par_dichotomie(liste,element, milieu+1, droite)
```

Notez l'utilisation d'arguments optionnels `gauche` et `droite`, permettant à la fois

— d'appeler la fonction à l'aide de 4 arguments pour rechercher `element` dans la sous-liste de `liste` entre `gauche` et `droite` ;
— et d'appeler la fonction à l'aide de 2 arguments seulement pour rechercher `element` dans `liste` (dans ce cas, `gauche` prend la valeur par défaut 0 et `droite` la valeur $+\infty$ qui existe dans les versions récentes de la librairie `math` de Python).

Question 12. Trouver le ou les éventuels bug(s) dans cet algorithme de recherche dichotomique à l'aide d'une batterie de tests.

Question 13. Proposer une correction du ou des bug(s).