1ère NSI — Exercices d'Entraînement

Algorithmique

Lycée Fustel de Coulanges, Massy

Marc Biver, février 2024, *v0.2*

Exercice 1: Docstring & Tests

Soit la fonction de calcul de puissance suivante (dont on a étudié l'algorithme en cours) :

```
1  def Puissance(x, n):
2      r = 1
3      for i in range(n):
4          r = r * x
5      return r
```

a. Rédiger la docstring d'une telle fonction ;

b. Déterminer un jeu de tests pour cette fonction – en d'autres termes, compléter le tableau suivant qui liste les couples (x, n) qu'il conviendra d'utiliser pour convenablement tester cette fonction.

x	n	Résultat attendu
2	2	4
.....	.....	.....

Exercice 2: Variant & invariant de boucle

Soit la fonction suivante :

```
1  def SommeEltLst(liste):
2      '''
3      Fonction qui prend en entrée une liste de nombres et qui renvoie la
4      somme des nombres qui la constituent.
5      '''
6      res = 0
7      for i in range(len(liste)):
8          res += liste[i]
9
10     return res
```

a. Donner un variant de boucle pour cette fonction et montrer qu'elle se termine systématiquement ;

> **Rappel:**
>
> On rappelle qu'un variant de boucle est une quantité entière positive qui décroît strictement à mesure qu'on passe dans la boucle, ce qui permet de justifier que la boucle, tôt ou tard, se terminera.

b. Donner un invariant de boucle pour cette fonction et montrer qu'elle est correcte.

Rappel:

On rappelle qu'un invariant de boucle est une quantité ou une propriété qui est vraie avant et après chaque itération de la boucle – et en particulier *avant* que l'on rentre dans la boucle, et *après* sa dernière itération. Il permet de justifier que le résultat voulu sera atteint. On procède pour cette technique en quatre étapes :

(a) On choisit l'invariant :
- Comprendre clairement le but de la boucle – qu'est-elle censée accomplir ? Quel est le résultat attendu ?
- Partir "de la fin", c'est-à-dire du résultat attendu et identifier quelle quantité est "construite" au fur et à mesure des itérations de la boucle pour constituer ce résultat.
- Ceci devrait vous mettre sur la voie de votre invariant – une propriété (somme d'éléments déjà traités, ordre d'éléments dans une liste...) qui ne change pas malgré les itérations de la boucle.

(b) On montre que l'invariant est vérifié avant la boucle (initialisation) ;

(c) On montre que si l'invariant est vérifié *avant* un passage dans la boucle, alors il est préservé *après* le passage dans la boucle ;

(d) On peut conclure sur la valeur finale à la sortie de la boucle.

Exercice 3: Variant & invariant de boucle — suite...

Soit la fonction suivante :

```
1  def RechercheDiviseur(nb):
2      '''
3      Fonction qui prend en entrée un entier strictement supérieur à 1 nb
4      et renvoie son plus petit diviseur autre que 1.
5      Si elle n'en trouve pas (que le nombre est donc premier), elle
6      renvoie 0.
7      '''
8      div = 0
9      i = 2
10     while i < nb:
11         # On vérifie si nb est un multiple de i
12         if nb % i == 0:
13             # Dès qu'on en a trouvé un on le renvoie
14             div = i
15             return div
16         i += 1
17     # On atteindra ce point si on n'a pas trouvé de diviseur et on renverra
       ↪  donc la valeur initiale de div, soit 0.
18     return div
```

a. Donner un variant de boucle pour cette fonction et montrer qu'elle se termine systématiquement ;

b. Donner un invariant de boucle pour cette fonction et montrer qu'elle est correcte.

Indice:

On rappelle qu'un invariant de boucle n'est pas nécessairement une formule mathématique – il peut aussi être une propriété que l'on pourrait exprimer par une phrase du genre (où bien entendu il faut remplir les trous) : "Quel que soit l'entier k tel que $2 \leq k \leq$ ___, k n'est pas___".

c. (question bonus 1) Cette boucle "`while`" semble assez artificielle – on dirait presque qu'elle n'est là que pour vous forcer à travailler sur la notion de variant de boucle... Quelle est la boucle "`for`" équivalente qu'on utiliserait plus logiquement dans un tel cas ?

d. (question bonus 2) Cette fonction n'est pas franchement optimale... Comment remplacer la condition "`while i < nb:`" pour lui faire faire nettement moins de tests tout en gardant le bon résultat ?

Exercice 4: Variant & invariant de boucle — encore un !

Soit la fonction suivante :

```
1  def RechercheMax(Lst):
2      '''
3      Fonction qui prend en entrée une liste de nombres positifs
4      et renvoie la valeur du plus grand d'entre eux.
5      '''
6      res = 0
7      for i in range(len(lst)):
8          if lst[i] > res:
9              res = lst[i]
10
11     return res
```

Donner un invariant de boucle pour cette fonction et montrer qu'elle est correcte.

Exercice 5: Complexité – inversion de liste

Soit la fonction suivante :

```
1  def InverseListe(lst):
2      '''
3      Fonction qui prend en entrée une liste quelconque et qui renvoie la
       ↪  liste inversée.
4      '''
5      # On initialise la liste résultat - liste vide
6      res = []
7      # On la remplit en partant de la fin de la liste en entrée
8      for i in range(len(lst)):
9          res.append(lst[len(lst) - i - 1])
10
11     return res
```

a. Montrer, en utilisant un invariant de boucle, qu'elle fait bien ce que sa docstring spécifie.

b. Compter son nombre d'opérations élémentaires, et en déduire sa complexité (préciser sa dénomination et sa notation en "$\mathcal{O}$").

Exercice 6: Complexité – note pondérée

Soit la fonction suivante :

```
def EltPondere(lst_notes, indice):
    '''
    Fonction qui prend en entrée une liste de notes et un indice dans cette
    ↪  liste et qui renvoie la valeur pondérée de cette note (donc sa
    ↪  valeur divisée par le nombre de notes).
    '''
    res = lst_notes[indice] / len(lst_notes)
    return res
```

Déterminer sa complexité.

Exercice 7: Complexité – somme de produits ou produit de sommes ?

Soit les deux fonctions suivantes :

```
def ProdSomme(n1, n2):
    '''
    Fonction qui prend en entrée deux entiers n1 et n2 et renvoie
    le produit des sommes de tous les i et j pour i < n1 et j < n2.
    '''
    res1 = 0
    res2 = 0
    for i in range(n1):
        res1 += i
    for j in range(n2):
        res2 += j

    res = res1 * res2

    return res
```

```
def SommeProd(n1, n2):
    '''
    Fonction qui prend en entrée deux entiers n1 et n2 et renvoie
    la somme de tous les produits i.j pour i < n1 et j < n2.
    '''
    res = 0
    for i in range(n1):
        for j in range(n2):
            res += i * j
            #On rappelle que "a += b" est équivalent à "a = a + b"

    return res
```

a. Démontrer (simplement) que si j'applique ces deux fonctions aux mêmes paramètres n1 et n2 leur retour sera identique – qu'en d'autres termes elles font le même calcul.

b. Quelles sont les nombres d'opérations élémentaires qu'effectuent chacune de ces deux fonctions (exprimés en fonction de n1 et n2) ? Laquelle des deux, du coup, vous semble la meilleure pour atteindre le résultat ?

Exercice 8: Tri à bulles - bidouillons-le un peu...

On rappelle le code Python du tri à bulles dans sa version la plus basique :

```
1  def TriBulles(liste):
2      '''
3      Fonction qui effectue le tri par permutations de la liste passée en
       ↪ entrée.
4      '''
5      n = len(liste)
6      for i in range(n):
7          for j in range(n-1):
8              if liste[j] > liste[j+1]:
9                  # Cette syntaxe permet d'affecter deux variables
                   ↪ simultanément et donc de ne pas passer par une variable
                   ↪ intermédiaire
10                 liste[j], liste[j+1] = liste[j+1], liste[j]
11     return liste
```

a. A quoi sert l'indice "i" dans cette fonction ?

b. Que se passe-t-il si on remplace la ligne 7 par `for j in range(n-1, -1, -1)` ? Cette syntaxe – qui n'est rien de plus qu'une utilisation de la syntaxe `range` habituelle mais dans le contexte d'une suite décroissante) permet de réaliser une boucle allant de n-2 (le premier paramètre) à 0 (arrêt juste avant d'atteindre le deuxième paramètre) par pas de -1 (le troisième paramètre). L'indice va donc décrire la suite (n-2), (n-3), ..., 1, 0 au lieu de 0, 1, ..., (n-3), (n-2).

c. Que se passe-t-il si on remplace la ligne 8 par `if liste[j] < liste[j+1]` : ?

d. La boucle interne ne fait-elle pas des opérations inutiles ? Comment la modifier pour la rendre plus efficace ?