1$^{\text{ère}}$ NSI — Exercices d'Entraînement

Algorithmique

Lycée Fustel de Coulanges, Massy

Marc Biver, février 2024, *v0.1*

Version incluant les réponses aux exercices.

Exercice 1: Docstring & Tests

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

```python
def Puissance(x, n):
    r = 1
    for i in range(n):
        r = r * x
    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
……	……	……

```python
def Puissance(x, n):
    '''
    Fonction qui prend en entrée deux entiers naturels et renvoie le premier
    élevé à la puissance du second.
    '''
    r = 1
    for i in range(n):
        r = r * x
    return r
```

Il n'y a pour le jeu de tests pas "une" bonne réponse, mais ce qu'on va chercher à faire c'est effectuer au moins deux tests de valeurs "normales" dont un avec une grande valeur (en plus, donc, du (2,2) de l'énoncé), ainsi que des tests avec les valeurs aux limites – en l'occurrence 0 et 1 – associées à des valeurs choisies au hasard. On pourra donc proposer quelque chose comme :

x	*n*	***Résultat attendu***
2	*2*	*4*
27	*7*	*10.460.353.203*
21	*0*	*1 (tout nombre élevé à la puissance 0 vaut 1)*
35	*1*	*35*
1	*12*	*1*
0	*99*	*0*
0	*0*	*1 (tout nombre élevé à la puissance 0 vaut 1 – même 0)*

Exercice 2: Variant & invariant de boucle

Soit la fonction suivante :

```python
def SommeEltLst(liste):
    '''
    Fonction qui prend en entrée une liste de nombres et qui renvoie la
    somme des nombres qui la constituent.
    '''
    res = 0
    for i in range(len(liste)):
        res += liste[i]

    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.

a. *Variant de boucle – on est dans le cas ultra-classique du parcours d'une liste par indice, le variant est donc la quantité* $len(liste) - i - 1$ *: il est en effet positif au départ (au pire nul si la liste est vide), et il décroît strictement à chaque itération de la boucle puisque* $len(liste)$ *reste constant et que* i *est incrémenté de 1 à chaque fois. Il va donc atteindre 0, ce qui terminera la boucle.*

b. *Décomposons la démarche :*

   (a) *Choix de l'invariant : le but de la boucle est le calcul de la somme des éléments de la liste, et c'est la variable* `res` *qui est construite pour atteindre ce résultat : à chaque étape elle contient la somme des éléments d'indice de 0 à* `i` *de la liste. On peut donc dire que l'invariant de boucle est : "la variable* `res` *contient la somme des éléments d'indice de 0 à* `i` *de la liste" ;*

   (b) *Avant la première itération de la boucle,* `i` *vaut 0 et* `res` *aussi – donc on peut dire que l'invariant est vérifié ;*

   (c) *Si res contient les i premiers éléments de la liste et que l'on passe dans la boucle une fois supplémentaire,* `i` *et* `res` *deviennent respectivement :* $i' = i + 1$ *et* $res' = res + liste[i'] = res + liste[i + 1]$ *– et donc l'invariant est bien vérifié à la fin de l'itération de la boucle.*

   (d) *A la fin de la boucle i vaut* $(n - 1)$ *donc l'invariant s'exprime "la variable* `res` *contient la somme des éléments d'indice de 0 à* `(n-1)` *de la liste" – soit tous les éléments de la liste, et donc la fonction est correcte.*

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

Soit la fonction suivante :

```python
def RechercheDiviseur(nb):
    '''
    Fonction qui prend en entrée un entier strictement supérieur à 1 nb
    et renvoie son plus petit diviseur autre que 1.
    Si elle n'en trouve pas (que le nombre est donc premier), elle
    renvoie 0.
    '''
    div = 0
    i = 2
    while i < nb:
        # On vérifie si nb est un multiple de i
        if nb % i == 0:
            # Dès qu'on en a trouvé un on le renvoie
            div = i
            return div
        i += 1
    # On atteindra ce point si on n'a pas trouvé de diviseur et on renverra donc la valeur initiale de div, soit 0.
    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 ?

a. Variant de boucle – situation assez simple : la quantité $nb - i$ débute à la valeur $nb - 2$ qui est nécessairement positive puisqu'il est spécifié que nb est strictement supérieur à 1 ; cette quantité décroît strictement à chaque itération de la boucle puisque nb reste constant et que i est incrémenté de 1 à chaque fois. Il va donc atteindre 0, ce qui terminera la boucle.

b. Décomposons la démarche :

(a) Choix de l'invariant : le but de la boucle est de trouver le diviseur le plus petit diviseur possible de nb, donc à chaque étape de la boucle, aucun nombre plus petit que celui que l'on est en train de considérer

n'est diviseur de nb. On peut donc exprimer l'invariant : "quel que soit l'entier k tel que $2 \leq k \leq$ (i-1), k n'est pas un diviseur de nb" ;

(b) *Avant la première itération de la boucle,* `i` *vaut 2, donc il n'existe aucun entier k tel que $2 \leq k \leq$ (i-1) – donc on peut dire que l'invariant est vérifié ;*

(c) *Si l'invariant est vérifié pour une valeur de i, alors pour la valeur suivante $i' = i + 1$, deux cas sont possibles :*

- *Soit i est un diviseur de nb et la fonction le renvoie et donc se termine (donc la question de l'invariant ne se pose plus) ;*
- *Soit i n'est pas un diviseur de nb – et la boucle se poursuit et l'invariant est vérifié : "quel que soit l'entier k tel que $2 \leq k \leq$ (i'-1), k n'est pas un diviseur de nb" (puisque $i' - 1 = i$).*

(d) *A la fin de la boucle, dans l'hypothèse où on n'est pas sorti de la fonction, i vaut nb, donc l'invariant s'exprime "quel que soit l'entier k tel que $2 \leq k \leq$ (nb-1), k n'est pas un diviseur de nb" – et donc nb est par définition un nombre premier.*

c. *On fait varier i de 2 à nb-1 donc la boucle correspondante s'écrirait "`for i in range(2, nb)`".*

d. *Il est évident que le plus grand diviseur d'un nombre (autre que lui-même) est sa moitié s'il est pair, ou son tiers s'il est impair mais un multiple de trois, etc. En tout état de cause, aucun nombre ne pourra avoir de diviseur strictement supérieur à sa moitié (ou la partie entière de sa moitié dans le cas d'un nombre impair). En écrivant la condition "`while i <= (nb // 2)`" on économiserait presque la moitié des opérations effectuées par la boucle tout en maintenant le résultat correct...*

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

Soit la fonction suivante :

```python
def RechercheMax(Lst):
    '''
    Fonction qui prend en entrée une liste de nombres positifs
    et renvoie la valeur du plus grand d'entre eux.
    '''
    res = 0
    for i in range(len(lst)):
        if lst[i] > res:
            res = lst[i]

    return res
```

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

1. *Choix de l'invariant : le but de la boucle est de trouver le maximum de la liste, donc avant chaque étape de la boucle, la variable* `res` *contient le maximum des éléments d'indice de 0 à* `i-1` *de la liste ;*
2. *Avant la première itération de la boucle,* `i` *vaut 0, donc la liste d'éléments dont l'indice est compris entre 0 et (i-1) est vide – donc on peut dire que l'invariant est vérifié ;*

3. *Si l'invariant est vérifié pour une valeur de i, alors pour la valeur suivante* $i' = i + 1$*, deux cas sont possibles :*
   - *Soit lst[i]* $\leq$ *res et donc, par définition, l'invariant reste vérifié (le maximum n'a pas changé) ;*
   - *Soit lst[i]* $>$ *res, res prend alors la valeur lst[i] qui est le nouveau maximum des éléments d'indice compris entre 0 et (i'-1) (donc i), et donc l'invariant est vérifié aussi.*
4. *A la fin de la dernière itération de la boucle, i vaut len(lst), donc res contient le maximum des éléments d'indice de 0 à* `len(lst)-1` *de la liste – donc de toute la liste, et donc la boucle est correcte.*