Applications

Exercice 1 – Somme des éléments d'une liste

Soit la fonction suivante.

```
def somme(L,x):
    """ La fonction prend en entrée une liste L de flottants ou d'entiers,
    et retourne la somme de ses éléments. """
    s=0
    for i in range(len(L)):
        #Inv(i): s est la somme des i premiers éléments de la liste.
        s+=L[i]
        #Inv(i+1): s est la somme des i+1 premiers éléments de la liste.
    return s
```

Question 1 *Montrer la terminaison et la correction de la fonction* `somme`

Correction

```
#Inv(i): s est la somme des i premiers éléments de la liste.
#Inv(i+1): s est la somme des i+1 premiers éléments de la liste.
```

Exercice 2 – Maximum d'une liste

Soit la fonction suivante.

```
def maximum(L):
    """ La fonction prend en entrée une liste L non vide de flottants ou d'entiers,
    et retourne le maximum de ses éléments. """
    m=L[0]
    for i in range(1,len(L)):
        #Inv(i): m est le plus grand élément de L[0:i].
        if L[i]>m:
            m=L[i]
        #Inv(i+1): m est le plus grand élément de L[0:i+1].
    return m
```

Question 1 *Montrer la terminaison et la correction de la fonction* `maximum`

Correction

```
#Inv(i): m est le plus grand élément de L[0:i].
#Inv(i+1): m est le plus grand élément de L[0:i+1].
```

Exercice 3 – Recherche d'un élément dans une liste

Soit la fonction suivante.

```
def recherche(L,x):
    """ La fonction prend en entrée une liste L et un élément x,
```

```
    et retourne True si x est dans L, False sinon."""
    for i in range(len(L)):
        if L[i]==x:
            return True
    return False
```

Question 1 *Montrer la correction de la fonction* `recherche`.

Correction
```
#Inv(i): x ne se trouve pas dans L[0:i].
#Inv(i+1)
```

Exercice 4 – Recherche dichotomique dans une liste triée Soit la fonction suivante.

```
def recherche_dicho(L,x):
    """ La fonction prend en entrée une liste L triée dans l'ordre croissant et un élément x,
    et retourne True si x est dans L, False sinon."""
    g=0
    d=len(L)
    while g<d:
        #Inv: x ne se trouve ni dans L[0:g] ni dans L[d:len(L)].
        m=(g+d)//2
        if L[m]==x:
            return True
        elif L[m]<x:
            g=m+1
        else:
            d=m
        #Inv: x ne se trouve ni dans L[0:g] ni dans L[d:len(L)].
    return False
```

Question 1 *Montrer la correction de la fonction* `recherche_dicho`.

Correction
- La terminaison de l'algorithme repose sur celle de la boucle `while` : la quantité $d-g$ est à valeurs dans $\mathbb{N}$ et décroît strictement à chaque itération de la boucle : l'algorithme termine.
- La correction repose elle aussi sur celle de la boucle `while`. On se rend compte facilement qu'elle admet l'invariant indiqué : si `L[m]` est égal à `x`, on renvoie simplement `True` et la fonction est correcte. Sinon, si `L[m]<x`, comme la liste est triée cela signifie que `x` ne peut se trouver qu'à un indice strictement supérieur à `m` et strictement inférieur si `L[m]>x`.
- Après la boucle, comme $g \geq d$ (en fait, $g = d$), l'invariant assure que `x` ne se trouve ni dans `L[0:g]` ni dans `L[d:len(L)]` donc en fait pas dans `L`. On renvoie `False` et la fonction est correcte.

Exercice 5 – Factorielle $n!$ On donne l'algorithme suivant.

```
for i in range(1,n+1):
    # en entrant dans le ième tour de boucle, p = (i-1)!
    p=p*i
    # en sortant du ième tour de boucle, p = i!
```

Question 1 *Montrer que l'algorithme précédent permet de calculer* $n!$.

Correction Ici, l'invariant de boucle est « p contient $(i-1)!$ » :
1. c'est bien une propriété qui est vraie pour $i = 1$;
2. supposons qu'au rang i, $p = (i-1)!$ à l'entrée de la boucle. Au cours de la boucle, p va prendre la valeur $p = (i-1)! \times i = i! = ((i+1)-1)!$ donc la propriété est vérifiée en sortie de boucle ;
3. enfin, au dernier tour de boucle, i vaut n donc $p = n!$ ce qui répond à la question.

Exercice 6 – L'algorithme d'Euclide `https://lgarcin.github.io/CoursPythonCPGE/preuve.html`

```
def pgcd(a, b):
  while b!= 0:
      a, b = b, a % b
```

```
    return a
```

Question 1 *Monter la terminaison de la fonction* `pgcd(a,b)`.

Correction On suppose que l'argument b est un entier naturel. En notant b_k la valeur de b à la fin de la $k^{\text{ème}}$ itération (b_0 désigne la valeur de b avant d'entrer dans la boucle), on a $0 \leq b_{k+1} < b_k$ si $b_k > 0$. La suite (b_k) est donc une suite strictement décroissante d'entiers naturels : elle est finie et la boucle se termine.

Question 2 *Monter la correction de la fonction* `pgcd(a,b)`.

Correction On note a_k et b_k les valeurs de a et b à la fin de la $k^{\text{ème}}$ itération (a_0 et b_0 désignent les valeurs de a et b avant d'entrer dans la boucle). Or, si $a = bq + r$, il est clair que tout diviseur commun de a et b est un diviseur commun de b et r et réciproquement. Notamment, $a \wedge b = b \wedge r$ ($\wedge =$ et ?). Ceci prouve que $a_k \wedge b_k = a_{k+1} \wedge b_{k+1}$. La quantité $a_k \wedge b_k$ est donc bien un invariant de boucle. En particulier, à la fin de la dernière itération (numérotée N), $b_N = 0$ de sorte que $a_0 \wedge b_0 = a_N \wedge b_N = a_N \wedge 0 = a_N$. La fonction `pgcd` renvoie donc bien le `pgcd` de a et b.

On donne une seconde version de l'algorithme d'Euclide. Pour cela on effectue la division euclidienne de a par b où $a$et b sont deux entiers strictement positifs. Il s'agit donc de déterminer deux entiers q et r tels que $a = bq + r$ avec $0 \leq r < b$. Voici un algorithme déterminant q et r :

```
q = 0
r = a
while r >= b :
    q = q + 1
    r = r -b
```

On choisit comme invariant de boucle la propriété $a = bq + r$.

Question 3 *Montrer la correction de l'algorithme.*

Correction

- Initialisation : q est initialisé à 0 et r à a, donc la propriété $a = bq + r = b \cdot 0 + a$ est vérifiée avant le premier passage dans la boucle.
- Hérédité : avant une itération arbitraire, supposons que l'on ait $a = bq + r$ et montrons que cette propriété est encore vraie après cette itération. Soient q' la valeur de q à la fin de l'itération et r' la valeur de r à la fin de l'itération. Nous devons montrer que $a = bq' + r'$. On a $q' = q + 1$ et $r' = r - b$, alors $bq' + r' = b(q+1) + (r-b) = bq + r = a$. La propriété est bien conservée.

Terminaison Nous reprenons l'exemple précédent.

- Commençons par montrer que le programme s'arrête : la suite formée par les valeurs de r au cours des itérations est une suite d'entiers strictement décroissante : r étant initialisé à a, si $a \geq b$ alors la valeur de r sera strictement inférieure à celle de b en un maximum de $a - b$ étapes.
- Ensuite, si le programme s'arrête, c'est que la condition du "tant que" n'est plus satisfaite, donc que $r < b$. Il reste à montrer que $r \geq 0$. Comme r est diminué de b à chaque itération, si $r < 0$, alors à l'itération précédente la valeur de r était $r' = r + b$; or $r' < b$ puisque $r < 0$. Et donc la boucle se serait arrêtée à l'itération précédente, ce qui est absurde ; on on déduit que $r \geq 0$.

En conclusion, le programme se termine avec $0 \leq r < b$ et la propriété $a = bq + r$ est vérifiée à chaque itération ; ceci prouve que l'algorithme effectue bien la division euclidienne de a par b.

Exercice 7 – Multiplication

L'objectif est de calculer le produit de deux nombres entiers positifs a et b sans utiliser de multiplication.

```
p = 0
m = 0
while m < a :
    m = m + 1
    p = p + b
```

Question 1 *Montrer la terminaison de l'algorithme.*

Correction Le programme se termine car la suite des valeurs de m est une suite d'entiers consécutifs strictement croissante, et atteint la valeur a en a étapes.

Question 2 *Proposer une propriété d'invariance.*

Correction Un invariant de boucle est ici : $p = m.b$.

Question 3 *Montrer la correction de l'algorithme.*

Correction Un invariant de boucle est ici : $p = m.b$.
- Initialisation : avant le premier passage dans la boucle, $p = 0$ et $m = 0$, donc $p = mb$.
- Hérédité : supposons que $p = mb$ avant une itération ; les valeurs de p et m après l'itération sont $p' = p + b$ et $m' = m + 1$. Or $p' = (p + b) = m.b + b = (m + 1)b = m'b$. Donc la propriété reste vraie.
- Conclusion : à la sortie de la boucle $p = m.b$.

Puisqu'à la sortie de la boucle $m = a$, on a bien $p = ab$.

Exercice 8 – Exponentiation rapide

```
def expo_rapide(x,n):
    y = x
    z = 1
    m = n
    # Invariant : x*y^m = x^n
    while m>0:
        q,r=m//2,m%2
        if r==1:
            z=z*y
        y=y*y
        m=q
    return z
```

Question 1 *Montrer que la fonction permet de calculer x^n.*

Correction **A vérifier**

Juste avant la boucle, on a $y = x$; $z = 1$ et $m = n$. Remarquez qu'ainsi, $z \times y^m = x^n$. Vérifions que si cette propriété est vraie en haut du corps de boucle, alors elle est vérifiée en bas du corps de boucle (là où se trouve le second). Si on se trouve en haut du corps de boucle, ceci signifie que la variable m contient un entier strictement positif. On a deux cas à examiner, suivant la parité de m. Notons x', y' et m' les valeurs des variables z ; y ; m en bas de la boucle.

- si m est pair, alors $q = \dfrac{m}{2}$, $r = 0$, $y' = y^2$, et $m' = q = \dfrac{m}{2}$ $z' = z$ est inchangé. On a alors $z' \times y'^{m'} = z \times \left(y^2\right)^{\frac{m}{2}} = z \times y^m$. Puisque $x \times y^m$ valait x^n en haut de la boucle, c'est toujours le cas en bas du corps de boucle.
- si m est impair, alors $q = \dfrac{m-1}{2}$, $r = 1$. Dans ce cas, z' prend la valeur $z \times y$, $y' = y^2$ et $m' = \dfrac{m-1}{2}$. On a alors $z' \times z'^{m'} = z \times y \times \left(y^2\right)^{\frac{m-1}{2}} = z \times y^m$. De même, puisque $z \times y^m$ valait x^n en haut de la boucle, c'est toujours le cas en bas du corps de boucle.

Ainsi, la propriété $x \times y^m = x^n$ est maintenue à chaque passage dans la boucle, c'est donc un invariant de boucle. On en déduit en particulier que cette propriété est vérifiée également après la sortie de la boucle. Rappelons que l'on a montré qu'en sortie de boucle, m était nul. Or comme l'invariant est vérifié, cela signifie que $z = x^n$. Comme on renvoie z, l'algorithme renvoie bien x^n, et est donc correct.

Exercice 9 – Recherche dichotomique dans un tableau trié – Formulation récursive

```
def dicho_rec(L,x):
    """L liste triée dans l'ordre croissant, x un élément. On renvoie True si x est dans L, ↙
        False sinon"""
    n=len(L)
    if n==0:
        return False
    m=n//2
    if L[m]==x:
        return(True)
    elif L[m]<x:
        return dicho_rec(L[m+1:],x) #la partie àdroite de L[m].
```

```
    else:
        return dicho_rec(L[:m],x) #la partie àgauche.
```

Question 1 *Montrer que la fonction termine.*

Correction La taille de la liste `L` est un variant de boucle.

Soit la proposition suivante : *« Si* `L` *est une liste triée dans l'ordre croissant et* `x` *un élément comparable à ceux de* `L`*, alors* `dicho_rec(L,x)` *retourne* `True` *si et seulement si* `x` *est dans* `L`*,* `False` *sinon.»*.

Question 2 *Montrer que la fonction termine.*

Correction
- Initialisation : si la longueur du tableau est 1, alors la sortie de la fonction est correcte.
- Hérédité : si `L` est de longueur au moins 2, un et un seul des cas suivants se produit :
  - `L[m]` est égal à `x`, auquel cas la sortie de la fonction est correcte.
  - `L[m]` est inférieur strictement à `x`, auquel cas `x` ne peut se trouver qu'après `m` puisque `L` est trié dans l'ordre croissant. Le tableau `L[m:]` correspond aux éléments placés après m (inclus), triés également dans l'ordre croissant. Par hypothèse de récurrence, la sortie de la fonction `dicho_rec` sur l'instance `(L[m:],x)` est correcte, donc également sur `(L,x)`.
  - On procède de même,si `L[m]` est strictement supérieur à x avec le tableau `L[:m]`.
- Par principe de récurrence, la fonction `dicho_rec` est correcte.