PC – Lycée René CASSIN **Informatique – Devoir maison autour des nombres premiers**

Ce devoir vous permettra de vous remémorer les connaissances acquises en première année, de les remettre en pratique et de vous familiariser avec de nouveaux exercices, notamment la preuve en informatique dont un exemple suit.

Un exemple de preuve en informatique

Exercice résolu

```
def fact(s) :
    t = 1
    i = 1
    while t < s:
        i = i + 1
        t = t * i
    print(i)
```

Soit s un entier supérieur ou égal à 2.
Justifier que l'appel de fact(s) affiche le plus petit entier n tel que $n! \geq s$.

Correction
Soit t_n et i_n les valeurs respectives de t et de i après la (n–1)-ième exécution de la boucle. Montrons par récurrence pour $n \geq 2$ la proposition suivante :
$\mathcal{P}(n)$: « $i_n = n$ et $t_n = n!$ »
Si $n = 2$, alors la boucle s'exécute une seule fois, i prend la valeur 2, t prend la valeur $1 \times 2 = 2$, donc $\mathcal{P}(2)$ est vraie.
Supposons que $\mathcal{P}(n)$ est vraie pour un entier $n \geq 2$.
Alors $i_n = n$ et $t_n = n!$
Lors de la n-ième exécution de la boucle, i est augmenté de 1, donc $i_{n+1} = n+1$ puis t est multiplié par i, donc $t_{n+1} = t_n \times (n+1) = (n+1)!$
$\mathcal{P}(n+1)$ est vraie.
La suite $(t_n)_{n\geq 2}$ diverge donc vers $+\infty$ et par définition de la limite, t_n, qui est égal à $n!$, est supérieure ou égale à s au bout d'un certain rang. Dès que cela se produit, la boucle s'arrête et affiche i_n, qui est égal à n, le plus petit entier tel que $n! \geq s$.

Partie A – Une première construction d'une liste de nombres premiers

On rappelle que sous Python, le passage d'une liste comme argument d'une fonction est un passage par adresse : la liste locale et la liste globale pointent vers le même objet. La fonction modifie donc la liste comme s'il s'agissait d'une variable globale.

Python permet d'appliquer les fonctions booléennes aux nombres entiers : l'idée générale est qu'un entier non nul est traité comme True et un entier nul comme False. Tester not(54) dans un shell Python.

On considère la fonction test(), qui prend en argument un entier m, et une liste d'entiers L.

```
def test(m,L) :
    for i in L :
        if not(m%i) :
            return
    L.append(m)
```

1) Dans quel(s) cas l'entier m est-il ajouté à la liste L ?

2) Prouver par récurrence que la fonction premier() définie ci-après pour $N \in \mathbb{N}\backslash\{0;1\}$ retourne la liste des nombres premiers inférieurs ou égaux à N.
On pourra nommer L_n la liste L après l'appel de test(n,L), pour $n \geq 3$.

```
def premier(N) :
    L = [2]
    for n in range(3,N+1) :
        test(n, L)
    return L
```

3) Ecrire une version optimisée de la fonction test(), que pourra appeler la fonction premier(), qui teste la divisibilité de *m* par les nombres de la liste L inférieurs ou égaux à $\sqrt{m}$.

4) Que retourne l'appel de premier2(N), où N ∈ ℕ* ? Justifier que la boucle **while** s'arrête bien.

```
def premier2(N) :
    L = [2]
    n = 3
    while len(L) < N :
        test(n, L)
        n = n+1
    return(L)
```

Partie B – Comparaison avec le crible d'Eratosthène

Activité préliminaire : tester le code suivant dans un shell Python.

```
L = list(range(20))
print(L)
L[5:15:2] = [3]*(5)
print(L)
L[8::2] = [7]*(6)
print(L)
```

Voici une version du crible d'Eratosthène, qui retourne une liste des nombres premiers inférieurs ou égaux à l'entier N.

```
def eratosthene(N) :
    P = []
    L = [True]*(N+1)
    for i in range(2,int(N**0.5)+1) :
        if L[i] :
            L[2*i::i] = [False]*(N//i-1)
    for i in range(2,N+1) :
        if L[i] :
            P.append(i)
    return P
```

Soit $\mathcal{P}$ l'ensemble des nombres premiers et pour tout entier *n*,

$\pi(n) = \text{Card}\ \{p \in \mathcal{P} / p \leq n\}$

Nous admettons deux résultats utiles dans cette partie :

- $\pi(n) \sim \frac{n}{\ln n}$ (théorème des nombres premiers)
- $\lim\limits_{n\to+\infty} \sum_{\substack{p=2 \\ p\in P}}^{n} \frac{1}{p} - \ln\ln n = M$, où $M \approx 0{,}261$ (constante de Mertens)

  En particulier, $\sum_{\substack{p=2 \\ p\in P}}^{n} \frac{1}{p} \sim \ln\ln n$ (résultat établi par Euler)

Nous souhaitons comparer l'algorithme A, codé par la fonction eratosthene() et l'algorithme B, codé par la fonction premier() couplée à la fonction test() optimisée.

1) Comparer la complexité en mémoire des deux algorithmes, en considérant le nombre de données stockées dans les listes sans différencier leurs types (entiers, booléens).

D'un point de vue algorithmique, un booléen peut être codé sur un bit alors qu'un entier est souvent codé sur plusieurs octets, mais en pratique Python 3 utilise en général 4 octets par variable d'une liste. Cela est dû au caractère dynamique des variables Python. Les langages utilisant des variables statiques, devant être déclarées, réservent un espace mémoire correspondant davantage à la nature de la variable et sont plus économes.

2) Tester expérimentalement la rapidité des deux algorithmes avec des valeurs de N allant de 10 000 à 10 000 000. Expliquer votre démarche et donner vos conclusions.

3) Montrer que le nombre d'affectations réalisées par l'algorithme A est en $\Theta(N \ln\ln N)$.

Remarque : pour évaluer la complexité en opérations élémentaires de l'algorithme B, on peut utiliser un compteur du nombre de calculs de reste de division euclidienne (résultats similaires à ceux des tests de rapidité).

Partie C – Amélioration du crible d'Eratosthène

1) Dans la fonction erathostene(), on modifie la ligne

```
L[2*i::i] = [False]*(N//i-1)
```

en remplaçant 2*i par i*i.

a) Quelle modification doit-on apporter à droite du signe = ?

b) Justifier que la fonction ainsi obtenue fournit toujours le même résultat.

Puisqu'à l'exception de 2, tous les nombres premiers sont impairs, on décide de ne traiter que les nombres impairs.

```
def eratosthene2(N) :
    n = (N + 1)//2
    P = [2]
    L = [True]*n
    for i in range(1,(int(N**0.5)-1)//2+1) :
        if L[i] :
            p = ……
            q = ……
            L[q::p] = [False]*((n-1-q)//p+1)
    for i in range(1,n) :
        if L[i] :
            P.append(2*i+1)
    return P
```

2) Déterminer les expressions de p et de q dans le code précédent afin que, suite à l'exécution de la première boucle **for**, L[i] = **True** si et seulement si $2i + 1$ est premier.

Dans le même esprit, on souhaite écarter de nos tests tous les multiples de 3.

3) Justifier que les nombres premiers différents de 2 et 3 sont à chercher parmi les nombres de la forme $6i - 1$ ou $6i + 1$, avec $i \in \mathbb{N}^*$.

Dans le code suivant, la liste L sert à coder la primalité des entiers sous la forme $6i - 1$, et la liste M celle des entiers sous la forme $6i + 1$.

```
def eratosthene3(N) :
    n = (N + 1)//6
    P = [2,3]
    L = [True]*(n+1)
    M = [True]*(n+1)
    for i in range(1,(int(N**0.5)+1)//6+1) :
        if L[i] :
            p = 6*i-1
            q = i*(p-1)
            M[q::p] = [False]*((n-q)//p+1)
            q = q + 2*i
            L[q::p] = [False]*((n-q)//p+1)
        if M[i] :
            ⋮
            ⋮
    for i in range(1,n+1) :
        if L[i] :
            P.append(6*i-1)
        if M[i] :
            P.append(6*i+1)
    return P
```

4) Ecrire la portion de code manquante.

Compléments

1 - Bytearray

L'utilisation de bytearray (tableau d'octets) à la place des listes permet de stocker des booléens (ou des entiers inférieurs à 256) pour un octet chacun.
Ainsi, remplacer les listes L et M par des bytearrays, en remplaçant [**True**] par bytearray([**True**]) et [**False**] par bytearray([**False**]) dans la fonction eratosthene3() divise par 4 l'espace mémoire utilisé.

2 – Générateur et fonction yield

Le plus souvent, il n'est pas utile de stocker la liste des nombres premiers.
Si nous voulons juste les afficher, il suffit de remplacer les ajouts à la liste P par des instructions **print**.

Une autre possibilité consiste à transformer notre fonction en générateur, un objet itérable (que l'on parcourt avec une boucle **for**) permettant de retourner des nombres sans interrompre le traitement de la fonction.
Il suffit pour cela de remplacer les ajouts à la liste P par des instructions **yield** (et bien sûr supprimer **return** P) :

yield 2; **yield** 3 au lieu de P = [2,3]

yield(6*i±1) au lieu de P.append(6*i±1)

On affiche alors la liste des nombres premiers avec les instructions :

```
for i in eratosthene3(int(input("Entrer N : "))) :
    print(i)
```

Partie D – Applications

Dans cette partie, vous pouvez utiliser et modifier librement les fonctions définies précédemment.

1 – Constante de Mertens

Vérifier l'approximation donnée dans la partie B de la constante de Mertens.

2 – Conjecture de Goldbach (formulée en 1742 qui reste à démontrer)

« Tout entier pair supérieur à 3 peut s'écrire comme la somme de deux nombres premiers »

Ecrire un programme permettant de vérifier la conjecture de Goldbach jusqu'à une valeur N entrée par l'utilisateur.