Test d'entraînement (correction)

Question 1. Parmi les objets Python définis ci-dessous, lesquels sont mutables ?

- ☒ `[1,1,0]` une liste (`list`) est mutable
- ☐ `(3,4)` un `tuple` est immutable
- ☒ `{1,2}` un ensemble (`set`) est mutable
- ☐ `frozenset([0,1])` un `frozenset` est immutable
- ☐ `"bonjour"` une chaîne de caractères est immutable

Question 2. On considère la fonction Python suivante, qui prend en entrée une liste de nombres et renvoie un nombre :

```
def max_prod(L):
    n = len(L)
    m = 0
    for i in range(n):
        for j in range(i):
            if L[i]*L[j]>m:
                m = L[i]*L[j]
    return m
```

Si n désigne le nombre d'éléments de la liste L, quelle est la complexité de cet algorithme ?

- ☐ $O(1)$
- ☐ $O(\log n)$
- ☐ $O(n)$
- ☐ $O(n \log n)$
- ☒ $O(n^2)$
- ☐ $O(2^n)$
- ☐ $O(e^n)$

Que l'on compte le nombre d'accès à un élément de L, le nombre de comparaisons ou le nombre de multiplications, dans tous les cas on arrive à un nombre d'opérations $O(n^2)$.

Le nombre de comparaisons, par exemple, est égal à

$$c(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=0}^{n-1} i = 1 + 2 + \ldots + (n-1) = \frac{n(n-1)}{2} = O(n^2).$$

Question 3. On considère une fonction Python f définie avec l'en-tête ci-dessous :

```
def f(x,y=1,z="hello"):
    ...
```

Quels sont, parmi les appels ci-dessous, ceux qui produiront nécessairement une erreur ?

- ☐ `f(3)`
- ☐ `f("hello",2)`
- ☐ `f(y=8,x=2)`
- ☒ `f(3,5,y=2)` y est spécifié 2 fois, par position et par nom
- ☐ `f(0,y="hello",z=1)`
- ☒ `f(y=1)` x n'est pas spécifié

Question 4. Qu'affiche le programme ci-dessous ?

```
I = lambda f,x:f(x+1)-f(x)
I(lambda x:x**2,5)
```

La première ligne définit une fonction `I(f,x)` qui renvoie la valeur `f(x+1)-f(x)`. La deuxième ligne est un appel à la fonction `I`, avec comme premier argument (`f`) la fonction $x \mapsto x^2$, et comme deuxième argument $x = 5$. Le résultat est donc $(5+1)^2 - 5^2 = 36 - 25 = \mathbf{11}$.

Question 5. Si A est une matrice carrée, combien de multiplications matricielles sont-elles nécessaires pour calculer A^{29} par exponentiation rapide ?

On écrit

$$\begin{aligned}
A^{29} &= A^{28} \times A \\
A^{28} &= (A^{14})^2 \\
A^{14} &= (A^7)^2 \\
A^7 &= A^6 \times A \\
A^6 &= (A^3)^2 \\
A^3 &= A^2 \times A \\
A^2 &= A \times A
\end{aligned}$$

Il y a donc **7** multiplications matricielles (un carré est une multiplication d'une matrice par elle-même).

Question 6. Déterminer le type de l'objet Python produit par l'instruction considérée :

a) `[(x**2,) for x in range(10)]` liste (composée de 10 tuples)
b) `(x**2 for x in range(10))` générateur
c) `(sum([x**2 for x in range(10)]))` nombre (la parenthèse extérieure ne sert à rien)
d) `[sum(x**2 for x in range(10))]` liste (composée d'un seul nombre)
e) `(sum([x**2]) for x in range(10))` générateur
f) `(sum(x**2 for x in range(10)),)` tuple (composé d'un seul élément)

choix possibles: nombre, tuple, liste, générateur, ensemble, dictionnaire

---

Question 7. Que vaut la file F après exécution de l'algorithme suivant ?

```
P <- pile_vide()
F <- file_vide()
pour n=1,2,3,4
    empiler n dans P
tant que P n'est pas vide
    dépiler x de P
    ajouter x à F
    extraire y de F
    ajouter y à F
```

Donner les éléments de la file F du dernier au premier, sous la forme de nombres séparés par des virgules, sans espace.

Les étapes successives sont les suivantes :

```
--  pile P --|-- file F --
haut     bas | fin  début
 4 3 2 1     |
 3 2 1       |  4
 2 1         |  4 3
 1           |  3 2 4
             |  4 1 3 2
```

Le résultat est donc **4,1,3,2**.

Question 8. Donner un pseudo-code pour une fonction f qui prend en argument un entier n et retourne la liste de tous les entiers entre 1 et n (inclus) qui sont des carrés parfaits. Par exemple, f(20) doit retourner la liste [1,4,9,16]

Correction :

```
fonction f(n)
    // retourne la liste des carrés parfaits compris entre 1 et n
    L <- liste_vide()
    k <- 1
    tant que k^2 <= n
        ajouter k^2 à L
        k <- k+1
    retourner L
```

Question 9. Donner un pseudo-code pour une fonction récursive f qui prend en argument la racine d'un arbre dont les étiquettes sont des nombres, et retourne la valeur maximale parmi les étiquettes de tous les nœuds de l'arbre. Ce pseudo-code pourra utiliser les fonctions de base `étiquette()` et `enfants()`, dont les signatures sont :

```
étiquette() : noeud -> nombre
enfants() : noeud -> ensemble de nœuds
```

```
fonction f(N)
    // retourne la valeur maximale de toutes les étiquettes d'un arbre
    m <- étiquette(N)
    pour e dans enfants(N)
        x <- f(e)
        si x>m
            m <- x
    retourner m
```

Question 10. Donner un pseudo-code pour une fonction f qui prend en entrée un ensemble de nombres A et retourne l'ensemble de tous les couples (x, y) avec $x \in A$, $y \in A$ et $x < y$.

Correction :

```
fonction f(A)
    "retourne l'ensemble de tous les couples (x,y) avec x,y dans A et x<y"
    E = ensemble_vide()
    pour tout x dans A
        pour tout y dans A
            si x<y
                ajouter (x,y) à E
    retourner E
```

Question 11. Traduire le pseudo-code ci-dessous en une fonction Python (on supposera qu'une fonction `pgcd()` a été préalablement écrite en Python):

```
fonction f(n)
    // retourne un graphe simple non orienté dont les sommets sont les entiers de 1 à n
    // et deux entiers a et b sont voisins si pgcd(a,b)>1
    // le graphe est codé par un dictionnaire
    D <- dictionnaire vide
    pour i allant de 1 à n
        E <- ensemble vide
        pour j allant de 1 à n
            si j est différent de i et pgcd(i,j)>1
                ajouter j à E
            ajouter à D la clé i, avec comme valeur correspondante l'ensemble E
    retourner D
```

Correction :

```python
def f(n):
    """
    retourne un graphe simple non orienté dont les sommets
    sont les entiers de 1 à n
    et deux entiers a et b sont voisins si pgcd(a,b)>1
    le graphe est codé par un dictionnaire
    """
    D = dict()
    for i in range(1,n+1):
        E = set()
        for j in range(1,n+1):
            if i!=j and pgcd(i,j)>1:
                E.add(j)
        D[i] = E
    return D
```

Question 12. Traduire le pseudo-code ci-dessous en une fonction génératrice Python :

```
fonction génératrice f(s)
    // retourne un générateur de tous les sous-ensembles de N*
    // dont la somme des éléments vaut s
    vérifier que s est de type entier et que s>0
    délivrer l'ensemble {s}
    pour k=1,2,...,s-1
        pour tout E délivré par f(s-k)
            si la valeur maximale de E est strictement inférieure à k
                ajouter k à E
                délivrer E
```

Correction :

```python
def f(s):
    """
    retourne un générateur de tous les ensembles de N*
    dont la somme des éléments vaut s
    """
    assert s>0 and type(s) is int
    yield {s}
    for k in range(1,s):
        for E in f(s-k):
            if max(E)<k:
                E.add(k)
                yield E

# vérification
>>> list(f(7))
[{7}, {3, 4}, {1, 2, 4}, {2, 5}, {1, 6}]
```

Question 13. Traduire le pseudo-code ci-dessous en une fonction récursive Python :

```
fonction inversions(L)
    // retourne le nombre d'indices i,j tels que i<j et L(i)>L(j)
    si L est vide
        retourner 0
    x <- L(0)
    s <- inversions( L privé de son premier élément L(0) )
    s <- s + nombre d'éléments y de L tels que y<x
    retourner s
```

Correction :

```python
def inversions(L):
    "retourne le nombre d'indices i,j tels que i<j et L[i]>L[j]"
    if len(L)==0:
        return 0
    x = L[0]
    s = inversions(L[1:])
    s = s + sum(y<x for y in L)
    return s

# vérification
>>> inversions([3,8,11,17,9,5])
6
```

En effet les 6 inversions de la liste `[3,8,11,17,9,5]` sont `(8,5) (11,9) (11,5) (17,9) (17,5) (9,5)`

Question 14. Écrire un code Python qui calcule et affiche le plus petit nombre entier $n \geqslant 0$ tel que

$$n^3 - 19n^2 \geqslant 1351.$$

Correction :

```
n = 0
while n**3-19*n**2<1351:
    n = n+1

>>> print(n)
22

# vérification
>>> n=21;n**3-19*n**2
882
>>> n=22;n**3-19*n**2
1452
```

Question 15. Écrire une fonction Python `occurrences()` qui prend en entrée une liste de nombres `L` et renvoie un dictionnaire `D` tel que pour toute valeur `x` présente dans `L`, `D[x]` contient l'indice de la dernière occurrence de `x` dans `L`. Par exemple, l'appel `occurrences([1,3,2,1,5,2,1,5])` renvoie le dictionnaire

`{1:6, 2:5, 3:1, 5:7}`

Correction :

```
def occurrences(L):
    D = dict()
    for i in range(len(L)):
        D[L[i]] = i
    return D

# test
>>> occurrences([1,3,2,1,5,2,1,5])
{1: 6, 2: 5, 3: 1, 5: 7}
```

Question 16. Écrire en Python une fonction `limite()` qui prend en entrée un argument `x` de type `float` et renvoie la limite quand $n \to +\infty$ de x^n (et `None` si la limite n'existe pas).

On sait que si $x \leqslant -1$, la quantité x^n n'a pas de limite quand $n \to +\infty$. En revanche, la limite vaut 0 si $-1 < x < 1$, vaut 1 si $x = 1$, et vaut $+\infty$ si $x > 1$. Ceci se traduit par le code suivant :

```
def limite(x):
    if x<=-1:
        return None
    if x<1:
        return 0
    if x==1:
        return 1
    return float('inf')

# vérification
>>> for x in [-1.2,-1,-0.1,0,0.9,1,1.3]:
...     print(x,limite(x))
...
-1.2 None
-1 None
-0.1 0
0 0
0.9 0
1 1
1.3 inf
```

Question 17. Écrire en Python une fonction génératrice `f()` qui prend en entrée une chaîne de caractères `s` et délivre (dans un ordre quelconque) toutes les sous-chaînes formées de 2 caractères consécutifs de `s`.

Par exemple, l'appel `f("Bonjour")` doit renvoyer successivement les valeurs `'Bo'`, `'on'`, `'nj'`, `'jo'`, `'ou'`, `'ur'`

Correction :

```
def f(s):
    n = len(s)
    for i in range(n-1):
        yield s[i:i+2]

# vérification
>>> list(f("Bonjour"))
['Bo', 'on', 'nj', 'jo', 'ou', 'ur']
```

Question 18. Écrire une fonction **récursive** `f(a,n)` qui prend en entrée deux entiers naturels a et n, et renvoie le terme u_n de la suite définie par $u_0 = a$ et la récurrence

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} u_n - 1 & \text{si} \quad u_n \text{ est impair,} \\ \dfrac{u_n}{2} & \text{si} \quad u_n \text{ est pair.} \end{cases}$$

Par exemple, `f(13,1)` doit renvoyer `12`, `f(13,2)` doit renvoyer `6` et `f(13,5)` doit renvoyer 1

Correction :

```
def f(a,n):
    if n==0:
        return a
    x = f(a,n-1)
    if x%2==1:
        return x-1
    else:
        return x//2

# vérification
>>> [(n,f(13,n)) for n in [1,2,5]]
[(1, 12), (2, 6), (5, 1)]
```

Question 19. Écrire une fonction non récursive `f(n)` qui prend en entrée un entier naturel $n \geqslant 3$ et renvoie la liste $[u_1, u_2, \ldots, u_n]$, où $(u_n)_{n\geqslant 1}$ est la suite définie par

$$\forall n \in \mathbb{N}^*, \quad u_n = \begin{cases} 1 & \text{si} \quad n \leqslant 3, \\ u_{n-2} + u_{n-3} & \text{si} \quad n \geqslant 4. \end{cases}$$

Par exemple, `f(10)` doit renvoyer la liste `[1,1,1,2,2,3,4,5,7,9]`

Correction :

```
def f(n):
    u = [1,1,1]
    for k in range(4,n+1):
        u.append(u[-2]+u[-3])
    return u

# vérification
>>> f(10)
[1, 1, 1, 2, 2, 3, 4, 5, 7, 9]
```
