Algorithmique

Quelques algorithmes sur les entiers

1. Écrire une fonction `d(n)` qui prend en argument un entier `n` et retourne la somme des diviseurs de `n` (en incluant `n`) :

```
>>> d(1)
1

>>> d(4)
7

>>> d(6)
12

>>> d(7)
8
```

2. Écrire une fonction `estCube(n)` qui prend en argument un entier `n`, et renvoie le booléen `True` si `n` est le cube d'un entier, et le booléen `False` sinon :

```
>>> estCube(0)
True

>>> estCube(1)
True

>>> estCube(2)
False

>>> estCube(8)
True

>>> estCube(27)
True
```

Comparer la complexité de deux programmes

On considère 2 versions du code Python pour une fonction `pow(a,n)` :

```python
# Version 1
def pow(a,n):
    if n == 0:
        return 1
    else:
        return a * pow(a, n-1)
```

```python
# Version 2
def pow(a,n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return pow(a,n//2) ** 2
    else :
        return pow(a,n//2) ** 2 * a
```

1. Pour chaque version du code, que vaut `pow(3,3)`, `pow(2,4)` ?
2. Pour chaque version du code, décrire en une phrase courte ce que calcule la fonction `pow(a,n)` dans le cas général ?
3. Pour chaque version du code, combien d'opérations sont nécessaires pour calculer `pow(2, 8)` ? Combien d'opérations dans chaque cas pour calculer `pow(2,64)` (on ne demande pas d'effectuer le calcul de ce que retourne la fonction) ?

Intersection de deux suites

Écrire une fonction `intersection(l1, l2)` qui prend en argument deux listes `l1` et `l2` qu'on suppose **triées**, et qui renvoie une liste contenant les éléments qui se trouvent à la fois dans `l1` et `l2` (répétés autant de fois qu'ils se trouvent dans les deux

listes). On veillera à limiter la complexité de l'algorithme (en particulier, il est possible de faire en sorte de ne parcourir chacune de listes qu'une seule fois).

```
>>> intersection([0, 1, 2, 4], [0, 2, 4])
[0, 2, 4]

>>> intersection([0, 0, 1, 1, 3], [0, 0, 0, 1, 2])
[0, 0, 1]

>>> intersection([0, 1, 3], [2, 4, 5, 10])
[]

>>> intersection([1, 2, 3, 4, 4, 5, 7, 10, 12, 13, 21], [0, 0, 2,
4, 4, 5, 6, 10, 13])
[2, 4, 4, 5, 10, 13]
```

Programmation Orientée Objet

Langage Python : Dans ce qui suit, chaque question est relative au code Python ci-dessous.

Partie 1

```
1°) Que représente l'dentificateur Ville dans le code Python ci-
dessous ?

class Ville:
    def __init__ (self, nomVille, nbHab):
        self.nom = nomVille
        self.nombreHabitant = nbHab

2°)  Modifier le constructeur de la classe Ville pour générer une erreur
dans le cas où la valeur du paramètre nbHab est négative

    def estMetropole(self):
```

```
        if self.nombreHabitant > 2000000:
            print (self.nom, "  est une grande métropole")
        else:
            if self.nombreHabitant < 20000:
                print (self.nom, "  est vraiment une ?")
            else:
                print (self.nom, "  est une ville moyenne")
```

3°) Quel est le lien entre les classes Ville et Capitale ?

```
class Capitale(Ville):
    def __init__ (self,nomVille, nbHab, pays):
        super().__init__(nomVille, nbHab)
        self.pays = pays
```

4°) Que font ces lignes de code ci-dessous ?

```
v1=Ville("Paris", 2600000)
v1.estMetropole()

v2=Ville ("Tarbes", 40000)
v2.estMetropole()

v3=Ville ("Igny", 10000)
v3.estMetropole()

v4=Capitale ("Londres", 10000, "Royaume-unis")
v4.estMetropole()

v5=Capitale ("Bruxelles", 10000, "Belgique")
v5.estMetropole()
```

5°) Que représentent les variables v1, v2, v3, V4, V5 (en termes de programmation orientée objet) ?

6°) Comment appelle-t-on en programmation orientée objet, la possibilité d'utiliser le même nom pour des méthodes qui réalisent des traitements différents ?

7°) À votre avis, que fait le code ci-dessous ?

```
tab = [v1, v2, v3, v4, v5]
for i in range(len(tab)):
    print ("Nombre habitants de " , tab[i].nom, " est : ",
tab[i].nombreHabitant)
```

Partie 2

```
class B:
    I = 0

8°) Quel est le statut de la variable I?

9°) Même question pour la variable x

10°) Que fait cette méthode __init__ ?

    def __init__(self, x):
        self.x = x
        B.I = B.I + 1

11°) Qu'est-ce qui sera affiché après le texte : Résultat 1 =, Résultat
2 =, Résultat 3 = etc. ci-après ?

print("Résultat 1 = ", B.I)
a = B(3)
print("Résultat 2 = ", B.I)
b = B(6)
print("Résultat 3 = ", B.I)
print("Résultat 4 = ", B.I)
a = B(3)
print("Résultat 5 =", B.I)

12°)  quel est le résultat d'affichage pour chacune des instructions ci-
dessous ?

print("a : x = ", a.x, " I = ", a.I)
print()
b = B(6)
print("Résultat 6 = ", B.I)
print("Résultat 7 : ")
print("a : x = ", a.x, " I = ", a.I)
print("b : x = ", b.x, " I = ", b.I)
c = B(8)
print("Résultat 8 : ")
print("B : I = ", B.I)
print("a : x = ", a.x, " I = ", a.I)
print("b : x = ", b.x, " I = ", b.I)
print("c : x = ", c.x, " I = ", c.I)
```