Devoir Surveillé 2 : Correction

1 Expression ou instruction

On considère le programme suivant, pour lequel on considérera que les erreurs ne provoquent pas l'arrêt du programme :

```
a=3
b="points"
c==4.
a=a+b
a=str(a)
a+b
```

Question 1. Indiquer pour chacune des 6 lignes de ce programme si ce sont des expressions ou des instructions.

`ligne 1` : Instruction ; on affecte 3 à `a` ; l'état de la mémoire est changé.
`ligne 2` : Instruction pour les mêmes raisons.
`ligne 3` : Expression ; elle sera évaluée comme un booléen.
`ligne 4` : Il s'agit d'une instruction (d'affectation).
`ligne 5` : Encore une instruction ; `a` est transtypé.
`ligne 6` : Il s'agit d'une expression ; qui sera évaluée comme une chaîne de caractères.

Question 2. Quelle(s) ligne(s) provoque(nt) une erreur et pourquoi ?

La ligne 3 provoquera une erreur ; en effet la variable `c` n'est pas encore déclarée, et l'évaluation de l'expression est impossible. La ligne 4 provoquera également une erreur ; l'addition entre un entier et une chaîne de caractères est impossible.

Question 3. Indiquer l'état de la mémoire après chacune des 6 lignes de ce programme. La mémoire est vide avant l'exécution de ce programme.

`ligne 1` : a contient 3
`ligne 2` : a contient 3 et b contient "points"
`ligne 3` : (erreur)
`ligne 4` : (erreur)
`ligne 5` : a contient "3" et b contient "points"
`ligne 6` : l'état de la mémoire est inchangé.

2 Théorèmes de Morgan

Question 4. Énoncer les deux théorèmes de Morgan

not(A or B) = not A and not B
not(A and B) = not A or not B

Question 5. Démontrer un des deux théorèmes de Morgan, après avoir précisé celui que vous choisissez de démontrer.

voir cours ; dresser une table de vérité.

3 Algorithme d'Euclide

Question 6. Écrire une fonction `Euclide(a,b)` qui prend en arguments deux entiers positifs `a` et `b` et qui détermine leur PGCD calculé à l'aide de l'algorithme d'Euclide par division successives.

```
def Euclide(a,b):
    while b>0 : # On suppose que a et b sont >0
        r=a%b
        a=b
        b=r
    return a
```

Question 7. Écrire la ligne de commande, saisie dans la console, qui affiche le PGCD de $10^{10}+2$ et de $10^{20}+2$.

`print(Euclide(10**20+2,10**10+2))`.

Il est à noter que puisque la commande est saisie dans la console, `print` n'est pas obligatoire, l'interpréteur Python affichant un résultat dès lors qu'il le peut pour les instructions saisies dans la console.

4 Factorielle

Question 8. Écrire une fonction `factorielle`, prenant en argument un entier positif `n` et renvoyant le résultat de $n!$. On rappelle que $0! = 1$ par convention.

Une boucle `for` est ici la solution la plus simple ; on prendra garde à l'initialisation de la variable temporaire.

```
def factorielle(n):
    p=1
    for i in range(n):
        p=p*i
    return p
```

5 Parcours de listes

Question 9. Écrire une fonction `moyenne(L)`, prenant en argument une liste de flottants `L` et qui renvoie sa moyenne.

Il s'agit de sommer la liste `L`, en effectuant son parcours, puis de diviser la somme obtenue par la longueur de la liste. Un parcours par éléments est adapté ici :

```
def moyenne(L):
    s=0
    for x in L:
        s=s+x
    return s/len(L)
```

Un parcours par indices reste possible :

```
def moyenne(L):
    s=0
    for i in range(len(L)):
        s=s+L[i]
    return s/len(L)
```

Enfin, l'énoncé n'interdisait pas explicitement l'usage de la fonction `sum` de Python, qui somme une liste :

```
def moyenne(L):
    return sum(L)/len(L)
```

Question 10. Écrire une fonction `test(L)` qui teste si une liste n'est composée que de nombres pairs

On va se servir du fait que `return` interrompt l'exécution d'une fonction, pour ne pas avoir à utiliser un booléen temporaire. On teste la parité d'un nombre en comparant à zéro le reste de sa division euclidienne par 2 :

```
def test(L):
    for i in L: # Un parcours par élément est efficace ici
        if i%2!=0 : # i%2==1 est acceptable si L n'est composée que d'entiers
            return False
    return True # placé en fin de boucle
```

6 Un problème de calendrier

Question 11. Dans la république de Linussie, certaines années sont des années de fête, en hommage à Torvald, fondateur de la république. Une année est festive si son nombre est divisible par 3, sauf si il est divisible par 12 (auquel cas l'année est destinée à préparer la guerre contre le pays voisin et non à festoyer ; sauf si il est divisible par 48, auquel cas l'année est festive.

Écrire une fonction `festive(n)` qui prend en argument le nombre d'une année et renvoie un booléen qui indique si cette année est festive ou non.

Une structure d'instructions conditionnelles avec alternatives est le plus simple (et le plus lisible ici), les tests commençant par la divisibilité par le plus grand nombre :

```
def festive(n):
    if n%48==0 :
        return True
    elif n%12==0:
        return False
    elif n%3==0:
        return True
    else:
        return False
```

Il est possible de factoriser ce code, une rédaction un peu extrême étant :

```
def festive(n):
    return n%4==0 and n%12!=0 or n%48==0
```

Question 12. Les habitants de Linussie veulent savoir rapidement quand sera la prochaine année festive. Écrire une fonction `prochaine(an)` qui prend en argument le numéro d'une année et renvoie la prochaine année festive.

Une boucle conditionnelle est ici adaptée, tant que l'année envisagée n'est pas festive, on passe à l'année suivante :

```
def prochaine(an):
    while not festive(an) :
        an=an+1
    return an
```

La deuxième ligne aurait pu être écrite `while festive(an)==False` mais c'est évidemment plus pataud.

7 Intégration numérique

On rappelle la formule de calcul approché d'une intégrale par la méthode des rectangles à gauche :

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\sum_{i=0}^{i=n-1} f\left(a+i\frac{b-a}{n}\right)$$

Question 13. Écrire une fonction `rectangles(f,a,b,n)` qui prend en arguments une fonction f, des flottants a et b et un entier n et qui renvoie le résultat du calcul approché de l'intégrale de a à b de la fonction f en n pas telle que donnée par la formule écrit plus haut.

```
def rectangles(f,a,b,n):
    s=0
    pas=(b-a)/n # Pas obligatoire mais simplifie le programme tant en lecture qu'en nombre d'opérations
    for i in range(n): # On s'arrête à n-1 à l'aide de range(n)
        s=s+f(a+i*pas)
    return s*pas
```

Question 14. Écrire une suite de commandes (on pourra éventuellement définir une fonction) qui calcule en 10^5 pas par la méthode des rectangles à gauche l'intégrale suivante :

$$\int_0^1 \frac{x}{1+x^3}dx$$

Une première solution consiste à définir une fonction et à la passer à la fonction `rectangles`. Ce qui donne :

```
def f(x):
    return x/(1+x**3)
```

```
print(rectangles(f,0,1,100000))
```

Il est également possible d'utiliser une fonction anonyme, à l'aide du mot clé `lambda` et d'écrire :

```
print(rectangles(lambda x:x/(1+x**3),0,1,100000))
```

8 Contrôle de rendu de monnaie

On s'intéresse dans cet exercice au fonctionnement d'une machine qui rend de la monnaie, selon un algorithme pour nous inconnu.

Cette machine travaille avec deux listes de nombres : une liste `denomination` et une liste `rendu`.

La liste `denomination` contient une liste d'entiers ou de flottants, rangée par ordre strictement décroissant, et correspondant aux billets et/ou pièces dont dispose la machine. Par exemple, dans le système monétaire européen, si la machine peut rendre des billets de 5, 10 et 20 euros, et toutes les pièces possibles, la liste serait :

```
denomination=[20,10,5,2,1,0.5,0.2,0.1,0.05,0.02,0.01]
```

La liste `rendu` est de la même longueur que la liste `denomination` et contient des entiers positifs. Ces entiers correspondent aux nombres de pièces et/ou billets associés à la liste `denomination` que la machine doit rendre. Par exemple, avec la liste `denomination` précédente, la liste :

```
rendu=[0,1,0,0,1,1,0,0,0,0,0]
```

correspondrait à un rendu d'un billet de 10 euros, un pièce de 1 euro et un pièce de 50 centimes, pour un montant à rendre de 11,50 euros.

Question 15. Écrire une fonction `testrendu(L,D)` qui prend en argument une liste `L` et qui vérifie si cette liste est susceptible d'être une liste de rendu de monnaie (si c'est une liste d'entiers positifs) associée à la liste `D` qui est une liste de dénomination supposée valide. Cette fonction renverra un booléen.

De manière générale dans cet exercice, nous allons utiliser le fait que `return` interrompt l'exécution d'une fonction.

Dans cette question spécifiquement, après avoir testé que `L` et `D` ont la même longueur, nous testerons les éléments de `L`, en utilisant un parcours par éléments :

```
def testrendu(L,D):
    if len(L)!=len(D): # différent de s'écrit != et non =!
        return False
    for x in D:
        if type(x)!=int or x<=0:
            return False
    return True
```

Question 16. Écrire une fonction `testdenomination(D)` qui prend en argument une liste `D` et qui vérifie si cette liste est susceptible d'être une liste de dénominations. Une liste de dénomination valide est une liste d'entiers ou flottants strictement décroissante et dont tous les termes sont strictement positifs.. Cette fonction renverra un booléen.

Un parcours par indices, jusqu'à l'avant dernier élément de `D`, donc en utilisant `range(len(D)-1)` est ici le plus simple. Le premier test aurait pu être incorporé dans la boucle, au prix de davantage de tests nécessaires :

```
def testdenomination(D):
    if type(D[0])!=int and type(D[0])!=float:
        return False
    for i in range(len(D)-1):
        if type(D[i+1])!=int and type(D[i+1])!=float or D[i]<=D[i+1]: #and est prioritaire sur or
            return False
    return True
```

Question 17. Écrire une fonction `monnaie(L,D)` qui prend en arguments deux listes `L` et `D`, qui sont ici supposées être des listes valides de rendu et de dénomination, et qui renvoie le montant total rendu par la machine.

Il s'agit de faire la somme des `L[i]*D[i]` ici. Un parcours par indices est pratiquement indispensable ici :

```
def monnaie(L,D):
    s=0
    for i in range(len(D)): # dans cette question on a par hypothèse len(L)=len(D)
        s=s+L[i]*D[i]
    return s
```

Question 18. Écrire une fonction `testretour(L,D,x)` qui prend en argument deux listes `L` et `D`, représentant `a priori` des listes de rendu et de dénomination, et un nombre `x` et qui renvoie le booléen :
- `False` si les listes `L` et `D` ne sont pas des listes de rendu et de dénomination valides ou si le montant à rendre associé aux listes `L` et `D` n'est pas égal à `x` ;
- `True` si les listes `L` et `D` sont valides et que le montant associé à ces listes et est égal à `x`

On se servira des fonctions précédemment codées dans cet exercice, même si on n'a pas su les coder.

Le cours sur le codage des flottants ne doit pas être enterré au fond du jardin des connaissances informatiques. On rappelle que pour tester l'égalité de deux flottants, il est hors de question d'écrire un simple $==$; la comparaison de flottants se fait toujours à un ε près, les erreurs d'arrondis ne permettant pas de garantir que deux flottants en principe égaux seront effectivement égaux au bit près, particulièrement après des opérations. Il suffit de se rappeler que $0.1+0.1+0.1\neq 0.3$ ...

Il est raisonnable ici de se dire que les systèmes de dénominations employées à travers le monde utilisent des centièmes d'unités. Un ε de l'ordre de 10^{-6} paraît justifié (mais on pourrait utiliser une marge d'erreur plus faible).

```
def testretour(L,D,x):
    if testdenomination(D) and testrendu(L,D): #Ces deux fonctions renvoient des booléens !
        if abs(x-monnaie(L,D))<10**-6:
            return True
    return False
```

Question 19. Avec la liste dénomination donnée en exemple au début de l'exercice, l'algorithme, pour le montant de 9 centimes crée la liste de rendu suivante :

`[0,0,0,0,0,0,0,0,1,1,1]`

Expliquer comment cette erreur a pu se produire, et ce que vous pourriez faire pour que cette erreur ne se produise pas (8 lignes maximum)

Les calculs effectués avec des flottants sont par nature approximatifs. Seuls les décimaux qui sont des sommes de puissances de 2 négatives peuvent donner lieu à des calculs exacts. Ce n'est pas le cas de nombres comme 0,1 qui si ils s'écrivent de manière exacte en base 10 ne peuvent s'écrire de manière exacte en base 2, quelque soit le nombre de bits employés. Rappelons que lors de calculs flottants, $0.1+0.1=0.2$ mais $0.1+0.1+0.1\neq 0.3$!

Ici, nous pouvons conjecturer que l'algorithme (dont nous ne disposons pas) est victime de ces erreurs d'approximations. Pour être certain que les calculs effectués donnent des résultats exacts, ce qui est absolument indispensable dans le cadre d'une machine à rendre la monnaie industrielle, nous travaillerons avec des entiers. Dans le cadre d'un système de dénominations où les unités seraient divisées en centièmes, il s'agirait de multiplier toutes les valeurs par 100 et de convertir l'ensemble des dénominations (et le prix à rendre) en entiers.

9 Boucles imbriquées

Question 20. Écrire une fonction `somme1` d'argument `n` qui retourne la valeur de la somme suivante :

$$\sum_{i=0}^{i=n}\sum_{j=0}^{j=n}\frac{1}{i^2+j^2+1}$$

Le titre de l'exercice est une indication en soi ... On prendra garde à bien effectuer les sommations pour des indices allant jusqu'à n inclus, en utilisant `range(n+1)` :

```
def somme1(n):
    s=0
    for i in range(n+1):
        for j in range(n+1):
            s=s+1/(i**2+j**2+1)
    return s
```

Question 21. Écrire une fonction `somme2` d'argument `n` qui retourne la valeur de la somme suivante :

$$\sum_{1\leqslant i\leqslant j\leqslant n}(i^2-\sqrt{j})$$

Deux implémentations sont ici possibles, suivant la réécriture choisie pour la double somme.

Celle-ci peut être vue comme :

$$\sum_{j=1}^{n}\sum_{i=1}^{j}(i^2-\sqrt{j})$$

ce qui donne l'implémentation suivante :

```
def somme2(n):
    s=0
    for j in range(n+1):
        for i in range(j+1):
            s=s+i**2-j**0.5
    return s
```

Il est également possible de voir la somme comme :

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(i^2-\sqrt{j})$$

et donne l'implémentation suivante :

```
def somme2(n):
    s=0
    for i in range(n+1):
        for j in range(i,n+1):
            s=s+i**2-j**0.5
    return s
```

C'est à chacun de se faire son opinion sur ce qui paraît plus naturel. À titre personnel, je trouve la première implémentation plus intuitive. Comme une majorité de monde. Je suppose que c'est une question de latéralité ; comme je confonds toujours la droite et la gauche en voiture, mon avis personnel est peut-être à prendre avec la plus grande circonspection.