Programmation Impérative en Python

Examen L1-E/M/MI/P

Université Nice Sophia-Antipolis, Avril 2015

*LE FORMULAIRE EN DERNIERE PAGE EST LE SEUL DOCUMENT AUTORISE. Tâchez de soigner l'**indentation**, la limpidité et l'exactitude du code. Répondez aux questions sur la copie d'examen qui vous est fournie, ne joignez aucune autre feuille ni intercalaire. N'hésitez pas à utiliser le résultat d'une question précédente même si vous ne l'avez pas faite. Et ne perdez pas de temps...*

1 Travail au Toplevel [7%]

Montrez ce qui est affiché à l'écran après l'exécution de chacun des deux scripts ci-dessous.

```
1  L = [1,2]
2  R = [L, L, L+L]
3  print('R original =',R)
4  R[1][1] = 1789
5  R[2][1] = 2015
6  print('R ensuite =',R)
```

```
7  # attention au piège !
8  s = "C'est la forme ?"
9  s = s[9:]
10 print('slice =',s)
11 s[0] = s[0].upper()
12 print('s =',s)
```

2 La Suite de Syracuse [33%]

On s'intéresse [1] à la **transformation de Syracuse** $S : \mathbb{N}^* \to \mathbb{N}^*$ définie de la manière suivante. Etant donné un entier $n > 0$: si n est pair alors $S(n) = n/2$, et si n est impair alors $S(n) = 3n + 1$. Par exemple, on obtient $S(10) = 5, S(5) = 16, S(16) = 8$.

a) Programmez la fonction `S(n)` pour $n > 0$ entier.

b) La **suite de Syracuse** $(u_{n,a})_{n\geq 0}$ est définie en se donnant un premier terme $a > 0$ entier, puis de proche en proche en posant $u_{n+1,a} = S(u_{n,a})$. Les premiers termes de cette suite sont donc $u_{0,a} = a$, $u_{1,a} = S(a)$, $u_{2,a} = S(S(a))$, etc. Ecrivez sur votre copie les 16 premiers termes de la suite $(u_{n,3})$. Qu'observez-vous ?

c) Programmez la fonction $u : \mathbb{N} \times \mathbb{N}^* \to \mathbb{N}^*$ telle que $u(n, a) = u_{n,a}$, par **récurrence** sur $n \geq 0$, et en supposant $a = u_0$ donné. Donc $u(n, a)$ est le terme d'indice n de la suite de Syracuse débutant par a.

d) Reprogrammez la fonction $u(n, a)$ mais cette fois avec une **boucle**.

e) Comment se comporte la suite $(u_{n,a})_{n\geq 0}$ si l'un quelconque de ses termes est une puissance de 2 ?

f) Jusqu'à nos jours, tous les choix faits pour le premier terme a aboutissent toujours sur 1 au bout d'un certain temps, mais aucun mathématicien au monde ne sait le démontrer ! La fonction ci-dessous essaye de le mettre empiriquement en évidence. On appellera *vol* d'un entier a (qui monte et qui descend) la suite des entiers obtenus de a jusqu'au premier 1 rencontré :

```
def print_vol(a) :
    while a != 1 :
        print(a,end=' ')
        a = S(a)
    print(a)
```

1. Voir http://fr.wikipedia.org/wiki/Conjecture_de_Syracuse

Par exemple `print_vol(13)` affiche : `13 40 20 10 5 16 8 4 2 1`. La fonction `print_vol` ci-dessus n'a aucun résultat. Eliminez les `print` et modifiez *soigneusement* son texte pour qu'au lieu d'afficher, la nouvelle fonction que l'on nommera `liste_vol(a)` retourne la **liste** de tous les entiers rencontrés depuis `a` jusqu'au premier `1`. Par exemple, la nouvelle version donnerait :

`liste_vol(13)` $\rightarrow$ `[13, 40, 20, 10, 5, 16, 8, 4, 2, 1]`

On dit que la longueur de cette liste est le *temps de vol* de l'entier a.

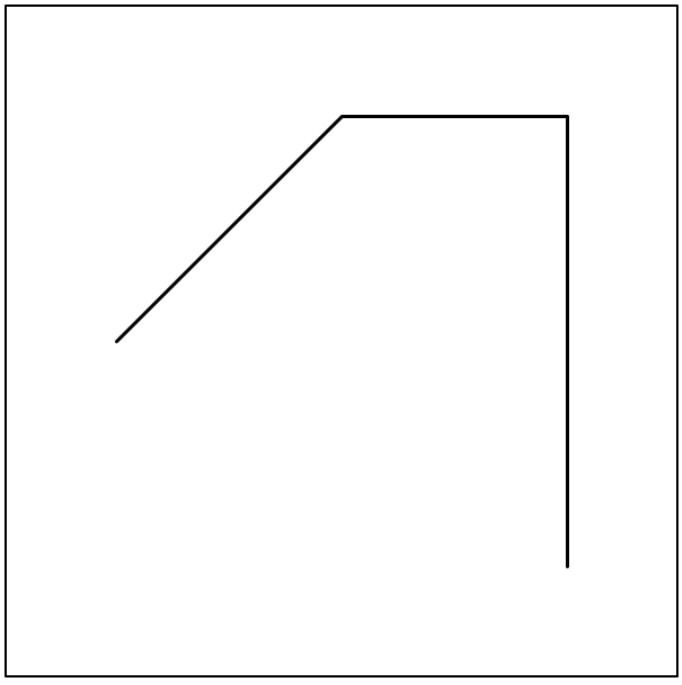
g) Le *temps de vol en altitude* d'un entier $a \geq 2$ est l'indice k du premier entier $\leq a$ rencontré dans `liste_vol(n)`. Le temps de vol en altitude de $a = 13$ est égal à 3. Quel est le temps de vol en altitude de l'entier $a = 7$?

h) Programmez la fonction `tv_alt(a)` retournant le temps de vol en altitude de l'entier $a \geq 2$.

3 Classes et Objets [27%]

Nous souhaitons modéliser un objet graphique *polygone ayant un nombre quelconque de sommets*. Un polygone sera un objet de la classe `Polygone`. Il a un seul attribut `L` qui est une liste `[(x1,y1),(x2,y2),...]` de points du plan représentés en Python par des tuples `(x,y)` avec `x` et `y` entre -300 et 300. Dans la classe, le constructeur initialisera l'attribut. On pourra demander d'afficher au toplevel un objet de type `Polygone` (suivant les lignes 2-3 ci-dessous). La méthode `longueur` permettra de demander à un polygone de calculer sa longueur. La méthode `dessine_toi` permettra de lui demander de bien vouloir se dessiner avec une tortue et dans une certaine couleur (on ne dessinera que les segments reliant les sommets consécutifs du polygone). Programmez la classe `Polygone` sans oublier d'importer tout ce qu'il faut.

```
1 >>> poly = Polygone([(-200, 0),(0, 200),(200, 200),(200, -200)])
2 >>> poly
3 Polygone[(-200, 0),(0, 200),(200, 200),(200, -200)]        # n'oubliez pas l'affichage !
4 >>> poly.longueur()
5 882.842712474619
6 >>> poly.dessine_toi('black')                              # avec la tortue
7 >>>
```



4 Nombres Premiers et Décomposition d'un Entier [33%]

a) J'ai trouvé sur le Web la page `http://nombrespremiersliste.free.fr/listes/1-1000000.txt` contenant tous les nombres premiers de 2 à 1000000, un seul nombre par ligne. Je l'ai téléchargée sur mon disque dur sous le nom de fichier `1-1000000.txt`. Montrez comment vous lisez ce fichier avec un script Python pour construire en mémoire centrale la liste `LP` de tous ces nombres premiers. Un petit test :

```
>>> LP[-1]      # le dernier élément
999983
```

b) Cette question est indépendante de a). Quelle est la valeur de `calc(750,5)` avec la définition ci-dessous ?

```
1 def calc(n,p) :     # n et p entiers >= 2
2     e = 0           # e va compter combien de fois... quoi ?
3     while n % p == 0 :
4         n = n // p
5         e = e + 1
6     return e
```

c) Utilisez la liste `LP` de a) et la fonction `calc` de b) pour produire la décomposition d'un entier n de l'intervalle $[2, 1000000]$ en produit de facteurs premiers p_i distincts, chacun affecté d'un certain exposant $e_i > 0$. Programmez à cet effet la fonction `decomp(n)` retournant la liste des $[p_i, e_i]$. Par exemple $n = 1176$ se décompose sous la forme $n = 2^3 3^1 7^2$:

`decomp(1176)` $\rightarrow$ `[[2,3],[3,1],[7,2]]`

d) Soit `DD = {}` un dictionnaire vide déclaré en variable globale. Utilisez la fonction `decomp(n)` sans la modifier, pour programmer une nouvelle fonction à mémoire `mdecomp(n)` qui stockera chacun de ses résultats de calcul `mdecomp(n)` $\equiv$ `L` dans le dictionnaire `DD` sous la forme d'un couple `n:L`. Bien entendu, avant d'effectuer le calcul, `mdecomp` vérifiera dans `DD` s'il n'a pas déjà été fait !

```
1 >>> DD = {}
2 >>> mdecomp(11)
3 [[11,1]]
4 >>> mdecomp(1176)
5 [[2, 3], [3, 1], [7, 2]]
```

```
6 >>> mdecomp(1176)              # déjà calculé,
7 [[2, 3], [3, 1], [7, 2]]      # time = 0 ms
8 >>> DD
9 {1176: [[2, 3], [3, 1], [7, 2]], 11: [[11, 1]]}
10 # notez que l'utilisateur n'utilise plus decomp
```

e) Ecrivez quelques lignes de code permettant de calculer le nombre `NS` d'entiers n entre 2 et 10000 n'ayant que deux diviseurs premiers, donc de la forme $n = p^a \times q^b$ avec p et q premiers distincts et $a, b \geq 1$. *Indication : il suffit de considérer la forme de leur décomposition.*

f) BONUS. Comment vérifieriez-vous en une ligne au toplevel Python que `NS` est une puissance de 2 en utilisant tout ce qui précède ?

ANNEXE au dos !