Programmation Impérative en Python

Examen L1-Info

Université Nice Sophia-Antipolis, Janvier 2016

*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 [2 pts]

Montrez ce qui est affiché à l'écran après l'exécution du script ci-dessous.

```
1  def foo(L) :
2      L1 = L + [2016]
3      L1[0] = 1000
4      L = L1
5      return L
```

```
6  >>> L = [1,2,3,4]
7  >>> foo(L)
8  ...
9  >>> L
10 ...
```

2 Un Nombre Bien Etrange [5 pts]

a) Programmez la fonction `explose(n)` prenant un entier `n > 0` et retournant la liste des chiffres de `n` en base 10, de la gauche vers la droite. Exemple : `explose(12320)` $\rightsquigarrow$ `[1,2,3,2,0]`

b) Déduisez-en la fonction à valeurs booléennes `mêmes_chiffres(p,q)` prenant deux entiers `p` et `q > 0`, et retournant True si et seulement si `p` et `q` ne diffèrent que par une permutation de leurs chiffres. *Vous pouvez la programmer en une ligne en utilisant un tri.* Exemples :

`mêmes_chiffres(1304,3041)` $\rightsquigarrow$ `True`, `mêmes_chiffres(112,221)` $\rightsquigarrow$ `False`

c) Sur mon ticket de bus, je vois le nombre `N = 142857`. Il me semble que si je multiplie `N` par un nombre de 2 à 6, j'obtiens un nombre composé de six chiffres qui sont toujours les mêmes chiffres que ceux de `N` dans un ordre différent. En utilisant la fonction `mêmes_chiffres`, aidez-moi à écrire quelques lignes de code dont l'exécution affichera (une seule fois) `VRAI` si c'est vrai et `FAUX` sinon (auquel cas on stoppera bien entendu au premier contre-exemple trouvé !).

3 Le Tri de la Bulle [4 pts]

Un assez mauvais tri, mais rigolo à programmer. Il s'agit de trier **sur place** (donc sans construire d'autre liste, avec des **mutations**) une liste de nombres entiers `L` de la manière suivante. On va effectuer un certain nombre de **passages** sur cette liste `L`, chaque passage la triant un peu plus en faisant *remonter les bulles*. Ce jargon de programmeur signifie la chose suivante. Un passage consiste à parcourir la liste `L = [`x_1`,`x_2`,...,`x_n`]` du début à la fin et chaque fois que l'on voit une **inversion** x_i `>` x_{i+1}, on remet de l'ordre en échangeant x_i et x_{i+1}. On stoppe dès qu'un passage n'a trouvé aucune inversion, il faudra donc le détecter. Voici par exemple les divers passages (leur nombre n'est pas connu à l'avance) pour trier la liste `L` donnée.

```
L = [5, 3, 8, 1, 4, 2, 10, 7]
    [3, 5, 1, 4, 2, 8, 7, 10]
    [3, 1, 4, 2, 5, 7, 8, 10]
    [1, 3, 2, 4, 5, 7, 8, 10]
    [1, 2, 3, 4, 5, 7, 8, 10]
```

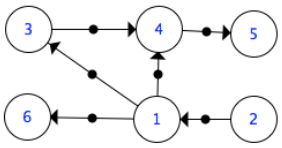
a) Programmez la fonction `un_passage(L)` qui va faire un seul passage sur `L` et opérer des échanges à l'intérieur de `L`. Elle retournera `True` si aucun échange n'a eu lieu, et `False` sinon.

```
>>> L = [3, 1, 4, 2, 5, 6, 7, 8]
>>> un_passage(L)
False               # donc après un passage, des échanges ont eu lieu et L a muté !
>>> L
[1, 3, 2, 4, 5, 6, 7, 8]
```

b) Déduisez-en la fonction `tri_bulle(L)`, sans résultat, modifiant la liste `L` en plusieurs passages jusqu'à ce que `L` soit triée. Exemple :

```
>>> L = [5, 3, 8, 1, 4, 2, 6, 7]
>>> tri_bulle(L)
>>> L
[1, 2, 3, 4, 5, 6, 7, 8]
```

4 Utilisation des Ensembles dans un Réseau Aérien [6 pts]



On s'intéresse ici à des vols d'avion un peu spéciaux reliant des villes qui seront notées par des numéros `1`, `2`, `3`, etc. Ces *vols directs* seront stockés dans une **liste** `VOLS = [(i,j),...]`, qui exprime que de la ville `i` on dispose d'un vol direct vers la ville `j`. L'exemple `VOLS` suivant n'est là que pour illustrer, on ne supposera bien entendu pas son contenu connu dans les algorithmes.

`VOLS = [(1,6),(3,4),(1,3),(2,1),(1,4),(4,5)]  # dans le désordre !`

a) Avec une ou plusieurs lignes de code Python, sans définir aucune fonction, montrez comment vous définissez la variable `VILLES` dont la valeur est l'**ensemble** de toutes les villes figurant dans `VOLS` (au départ comme à l'arrivée). Ici par exemple, la variable `VILLES` vaudrait {`1,2,3,4,5,6`}.

b) Une ville v_2 est un *successeur* d'une ville v_1 s'il existe un vol direct $v_1 \to v_2$. Définissez la fonction `successeurs(v)` retournant l'**ensemble** des villes successeurs de la ville `v`. Exemple :

`successeurs(3)` $\rightsquigarrow$ {`4`}, `successeurs(1)` $\rightsquigarrow$ {`3,4,6`}, `successeurs(5)` $\rightsquigarrow$ `set()`

c) On suppose que le réseau de vols ne contient *aucune boucle* : on ne peut pas partir d'une ville puis après un certain nombre de vols revenir à cette même ville, c'est le cas du réseau de notre exemple ci-dessus. Complétez la définition de la fonction `un_vol(v1,v2)` retournant une suite de vols consécutifs – peu importe lesquels – reliant `v1` à `v2`. Le résultat sera une liste de villes débutant par `v1`, terminant par `v2` et contenant les étapes intermédiaires, dans le bon ordre. Par exemple :

`un_vol(1,2)` $\rightsquigarrow$ `[]`, `un_vol(1,4)` $\rightsquigarrow$ `[(1,4)]`, `un_vol(2,5)` $\rightsquigarrow$ `[(2,1),(1,3),(3,4),(4,5)]`

Vous complèterez le code de la fonction ci-dessous :

```
def un_vol(v1,v2) :          # on ne demande pas de chercher le plus court !
    S = successeurs(v1)
    if S == [] : return ...
    if v2 in S : return ...
    for v in S :
        ...                  # plusieurs lignes. Raisonnez par RECURRENCE.
    ...
```

5 Tortue et Fichiers [5 pts]

Cet exercice ne comporte que des lignes de code, aucune fonction. Vous importerez uniquement ce qu'il faut.

a) Programmez la construction automatique d'un nouveau fichier `'prog.txt'`. Chaque ligne contiendra une instruction `forward d` suivie d'un point-virgule (entouré d'un espace), suivi d'une instruction `left a`. On choisira chaque fois la distance `d` au hasard dans `[10,20]` et l'angle `a` au hasard dans `[-70,70]`. Le nombre de lignes du fichier sera choisi au hasard entre 20 et 40. Exemple d'une telle ligne :

```
forward 17 ; left -32
```

b) Nous supposerons que les lignes du fichier `'prog.txt'` sont bien de la forme décrite en a). Programmez une lecture de ce fichier, en faisant exécuter chaque ligne par la tortue (qui va donc chaque fois avancer et tourner). Vous ferez afficher à la fin la distance totale parcourue par la tortue lors de son petit voyage. *Indication : utilisez la fonction* `split` *sur chaque ligne lue.*

FIN