Université de Caen **Année 2013/2014**
UFR de Sciences **session de décembre**

Licence L2,

Examen de Programmation Objet / Python

Durée: 2,0 H. Les documents ne sont pas autorisés.

Chaque candidat doit, au début de l'épreuve, porter son nom dans le coin de la copie qu'il cachera par collage après avoir été pointé. Il devra en outre porter son numéro de place sur chacune des copies, intercalaires, ou pièces annexées.

1. Écrire un paquetage opérant sur les "nombres" étant des séquences de chiffres, par ex. `N(1,0,6,3)` représentant le nombre décimal 1063. Construire la classe **N** de tels nombres, ***par la surcharge des listes***. Le constructeur acceptera un nombre arbitraire de paramètres – chiffres comme ci-dessus. La représentation de ces objets doit avoir la forme un peu différente de la forme d'entrée : `N[1,0,6,3]`, avec les crochets. `N()` s'affiche comme `N[]`, et `N(9)` comme `N[9]`.

   – Si `a=N(1,0,6,3)`, une "tranche" (slice), par ex. `a[1:-1]` peut retourner une liste, ici `[0,6]`, si on surcharge les listes, mais pas leurs méthodes. Comment la convertir en objet de classe **N**?
   – Surcharger l'opérateur d'addition '+' et l'affichage. ***Bonus*** : surchargez aussi la multiplication.

   Une stratégie *possible* pour `(+)` est la suivante. Commencez par la procédure qui ajoute à une liste N(...) un seul chiffre. On ajoute ce chiffre au dernier élément, on vérifie si la somme dépasse 10, et on propage la retenue à gauche. On peut exploiter `divmod(n,m)` $\rightarrow$ (quotient, reste) de la division n/m (pour $m = 10$). Ensuite pour addition des listes utilisez la récursivité. La multiplication peut aussi utiliser la récursivité, si on sait comment multiplier la liste par un seul chiffre.

2. Compacteur des flots. **A.** Écrire une procédure – générateur, qui prend comme entrée un flot (produit par un autre générateur) avec les items qui peuvent se répéter, par ex. : 1, 0, 0, 1, 2, 7, 3, 3, 3, 3, 7, 0, 2, 2, 5, ...
   Votre générateur ramasse les données voisines qui se répètent, par ex. ...,x,x, ..., et fournit un tuple (m,x), où m est le facteur de répétition. Le flot ci-dessus doit se transformer en 1, (2,0), 1, 2, 7, (4,3), 7, 0, (2,2), 5, ... Le générateur doit gracieusement terminer son travail quand le flot d'entrée se termine (ne pas oublier d'émettre l'élément déjà dans la mémoire). **B.** Écrire le générateur qui effectue la transformation inverse.

3. Écrire une « calculatrice postfixe » un objet, dont la fonctionnalité se réduit à deux méthodes.
   – Le lecteur, qui lit une chaîne composé de chiffres, symboles d'opérateurs et symboles fonctionnels : mots préfixés par le caractère '!', par exemple `"876 12 + 34 61 +!sin 23 /!exp *"`, où les éléments sont séparés par les espaces, et qui la transforme en liste-code : `[876, 12, '+', 34, 61, '+', '!sin', 23, '/', '!exp', '*']`. Vous pouvez utiliser `eval` pour transformer les chaînes numériques en nombres. Les chaînes symboliques seront les clés dans un dictionnaire qui associe ces symboles à la description des objets fonctionnels correspondants.
     Attention, les opérateurs (4 standard) seront des fonctions à deux arguments (écrites par l'utilisateur), les fonctions prefixées par '!' – des fonctions à un argument, récupérées du module math, ou écrites par l'utilisateur, sans le point d'exclamation. Il faut pouvoir reconnaître cette différence (par ex., ajouter l'"arité" de la fonction dans ce dictionnaire ou dans un autre).
   – La calculatrice, la méthode `calc` qui interprète le code : si l'élément est un nombre, il est placé sur une pile (utilisez une structure arbitraire, par ex. une liste). Si c'est une fonction décodée depuis sa clé, le programme récupère le nombre adéquat d'arguments, 1 ou 2, les supprime de la pile, exécute la fonction et dépose le résultat sur la pile. Quand le code se termine, la méthode retourne la pile. Par ex., le code `[8, 5, '-']` retourne `[3]`, et `[876, 12, '+', 34, 61, '+', '!sin', 23, '/', '!exp', '*']` donne `[914.77]` (la valeur de $(876 + 12) \times \exp(\sin(34 + 61)/23)$.

Écrivez de manière lisible, un texte difficile à déchiffrer risque d'être rejeté. Ne jamais copiez des morceaux de code inutile, tout code hors sujet peut être pénalisé. Évitez toute discussion qualitative « philosophique » inutile. Par contre, *commentez votre code*, sinon il risque d'être peu compréhensible.