Algorithmique et Programmation en Python

Rattrapage L1-I/MI

Université Nice Sophia-Antipolis, Juin 2017

*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 Chaînes de Caractères [3 pts]

a) Programmez une fonction `diminuer(n)` prenant un entier $n \geq 100$ (ceci est garanti) et retournant l'entier obtenu en supprimant les chiffres situés aux extrémités de l'écriture décimale de n :

`diminuer(465231)` $\rightsquigarrow$ `6523` `diminuer(100000006)` $\rightsquigarrow$ `0`

b) *Question de cours.* Programmez une fonction `binaire(n)` prenant un entier $n > 0$ et retournant une chaîne de caractères contenant l'écriture binaire de n. Il est **interdit** d'utiliser la primitive `bin` de Python !

`binaire(13)` $\rightsquigarrow$ `'1101'`

2 Décomposition Naïve en Facteurs Premiers [8 pts]

RAPPEL : tout entier $n \geq 2$ se décompose de manière unique en produit de puissances de nombres premiers p_i, soit : $n = p_1^{\alpha_1} \times p_2^{\alpha_2} \times \ldots \times p_k^{\alpha_k}$. Par exemple $4840 = 2^3 \times 5^1 \times 11^2$. Nous cherchons ici une manière simple de calculer une telle décomposition.

a) Expliquez ce que calcule précisément la fonction `ppd`, pour un argument `n` $\geq$ `2`, avec la définition ci-dessous :

```
1 def ppd(n) :                                          # on suppose n ≥ 2
2     L = [k for k in range(2,n+1) if n % k == 0]
3     return L[0]
```

b) Expliquez pourquoi le calcul de la liste `L` en ligne 2 rend la fonction `ppd` très inefficace.

c) Reprogrammez la fonction `ppd` pour éliminer cette inefficacité (on ne demande pas un algorithme optimal).

d) Programmez une fonction `facteurs_premiers(n)` prenant un argument entier $n \geq 2$, et retournant la **liste croissante des facteurs premiers distincts** de n. *Indication : dans une boucle* `while`*, utilisez la fonction* `ppd` *pour les extraire un par un, en évitant les répétitions.*

`facteurs_premiers(4840)` $\rightsquigarrow$ `[2,5,11]` `#` $4840 = 2^3 \times 5^1 \times 11^2$

e) Programmez une fonction `exposant(p,n)` retournant le nombre de fois que l'entier n est divisible par l'entier p : `exposant(2,40)` $\rightsquigarrow$ `3` `exposant(3,40)` $\rightsquigarrow$ `0`

f) Utilisez les fonctions `facteurs_premiers` et `exposant` pour programmer une fonction `factorise` retournant la liste des couples `[`p_i`,`α_i`]` de la décomposition de n :

`factorise(4840)` $\rightsquigarrow$ `[[2,3],[5,1],[11,2]]`

3 Tri par Insertion en Style Fonctionnel [4 pts]

a) *Question de cours.* Programmez une fonction `insertion(x,LT)` prenant une liste `LT` de nombres entiers triés en ordre croissant, et retournant une copie de cette liste - toujours croissante - dans laquelle l'élément `x` aura été inséré à sa juste place :

```
insertion(10,[1,6,8,15,19,22]) ⇝ [1,6,8,10,15,19,22]
insertion(6,[1,6,8,15,19,22])  ⇝ [1,6,6,8,15,19,22]
insertion(30,[1,6,8,15,19,22]) ⇝ [1,6,8,15,19,22,30]
insertion(0,[1,6,8,15,19,22])  ⇝ [0,1,6,8,15,19,22]
```

b) Déduisez-en la programmation *par récurrence* d'une fonction `tri_ins(L)` retournant une **copie** triée en croissant de la liste de nombres `L`. Vous ne procèderez à aucune mutation sur `L`.

```
tri_ins([15,1,8,22,6,19]) ⇝ [1,6,8,15,19,22]
```

4 Graphisme de la Tortue [4 pts]

a) Vous allez programmer avec la tortue, et nous aurons besoin : du graphisme de la tortue, de tirer des entiers au hasard, et de la fonction $x \mapsto \sqrt{x}$. Ecrivez sur votre copie les trois directives `import` nécessaires.

b) Programmez une fonction `trajectoire(n)` prenant un argument entier $n > 0$. Cette fonction va dessiner à l'écran mais aussi retourner un résultat. La tortue est à la position $(0, 0)$ au début, crayon baissé (ceci est garanti). Elle va effectuer n fois les actions suivantes : tourner d'un angle entier aléatoire entre $-80°$ et $80°$, puis avancer de 8 pixels. A chaque tour de boucle, vous rajouterez à une liste `L` vide au départ le point $(\texttt{x}_i, \texttt{y}_i)$ qu'elle visite. A la fin, cette fonction retourne comme résultat la liste `L` des n points visités.

c) Posons `T = trajectoire(300)`. Ecrivez quelques lignes de code Python :

1. calculant la plus grande ordonnée `ymax` atteinte lors de la trajectoire `T`, donc l'ordonnée du point le plus haut.
2. calculant la liste `H` des points d'ordonnée `ymax` (il peut exceptionnellement y en avoir plusieurs).
3. affichant avec `dot` un disque rouge de diamètre 10 sur chacun des points de `H`.

5 Programmation par Objets [4 pts]

Programmez une classe `Meteo`. Un objet de type `Meteo` aura un seul attribut `L` qui sera une liste de températures (des nombres approchés). L'utilisateur pourra construire un objet de type `Meteo` en passant *optionnellement* une liste de températures au constructeur de la classe. S'il n'en passe pas, la liste de températures sera initialisée à la liste vide. Cette classe comprendra – outre le constructeur – trois méthodes d'instance :

- une méthode `ajouter` prenant en argument un nombre `t` et rajoutant la température `t` à la fin de la liste `L`. Aucun résultat.
- une méthode `bilan` sans argument, affichant sur une même ligne la température minimale, la température maximale et la moyenne des températures. Aucun résultat.
- une méthode `extraction` prenant en argument un nombre `tmax`, et retournant la liste des températures de `L` qui sont strictement inférieures à `tmax`.

Exemple de session au toplevel Python :

```
>>> m = Meteo([-2,5,0])                                    # type(m) == Meteo
>>> m.ajouter(-6)
>>> m.L
[-2, 5, 0, -6]
>>> m.bilan()
mini = -6, maxi = 5, moy = -0.75
>>> m.extraction(0)
[-2, -6]
```