UNIVERSITE LIBRE DE BRUXELLES 30 août 2012
Ecole polytechnique de Bruxelles

INFO-H-100 – Informatique – Programmation – Prof. Th. Massart
1ère année du grade de Bachelier en Sciences de l'Ingénieur
Examen de seconde session

Remarques préliminaires

- On vous demande de répondre à **chaque question sur une feuille séparée**.
- N'oubliez pas d'inscrire votre nom, prénom et numéro de matricule sur chaque feuille, en haut à gauche.
- Vous disposez de 3 heures et vous ne pouvez pas utiliser de notes.
- La réponse à la question doit comprendre, si approprié, le code *Python* structuré et conforme aux règles de bonne pratique et conventions, avec sauf mention contraire, aucune fonction de librairies (pas d'`import`), une découpe en fonctions de manière pertinente et des structures de données appropriées.

Question 1 - Python (1 point)

Expliquez succinctement ce que fait et imprime le code suivant en justifiant votre réponse grâce à un ou des diagrammes d'état.

```
t=range(3)
t2=[t]*2
print t2
t3=t2[:]
t=[5,6,7]
print t2
print t3
```

Question 2 - Python et Complexité (2 points)

Soient les fonctions suivantes :

```
from math import sqrt
def f1(nmax):
    nums = range(2,nmax+1)
    for i in range(2, int(sqrt(nmax))+1):
        f = lambda x: x == i or x % i != 0
        nums = filter(f, nums)
    return nums
```

```
def f2(a):
    a = a.split()
    b = {}
    num = 0
    for i in a:
        b[i] = b.get(i,0) + 1
        if b[i] == 1:
            num += 1
    return num
```

Expliquez ce qu'elles font. Sachant que la création d'une fonction lambda se fait en temps constant, donnez la complexité maximale en $\mathcal{O}()$ du temps d'exécution de chacune de ces fonctions (précisez de quoi elle dépend). Veillez à détailler vos réponses en justifiant la complexité de chaque partie du code.

Question 3 - Compilation / interprétation (2 points)

Expliquez les différentes phases de la compilation d'un programme. Python est un langage interprété. Expliquez la différence avec un langage compilé. La gestion de la mémoire lors de l'exécution d'un programme est-elle différente dans les deux cas (justifiez) ?

Question 4 - Correcteur orthographique (3 points)

Pour cette question, vous ne pouvez pas utiliser de boucle. Veillez de plus à utiliser une structure de données qui rend la fonction efficace à la fois pour un grand dictionnaire et un long texte à vérifier.

Dans le cadre de la création d'un correcteur orthographique simple, on vous demande d'écrire une fonction qui reçoit en paramètres un dictionnaire et un texte, et qui renvoie les mots du texte qui ne sont pas dans le dictionnaire (et qui sont donc potentiellement mal orthographiés).

Le dictionnaire est reçu sous la forme d'une liste de chaînes de caractères, où chaque chaîne correspond à un mot et est tout en majuscules. Vous pouvez considérer que cette liste est déjà triée par ordre alphabétique.

Le texte est reçu sous la forme d'une liste de chaînes de caractères également, où chaque chaîne correspond à un mot du texte. Le texte a déjà été "nettoyé" : il ne reste pas de caractères de ponctuation. Les mots ont par contre encore la capitalisation qu'ils avaient dans le texte (les mots qui étaient en début de phrase ont encore leur majuscule, par exemple).

Vous pouvez considérer que ni les mots du dictionnaire ni les mots du texte ne contiennent d'accents.

Votre fonction doit renvoyer une nouvelle liste contenant les mots du texte qui ne sont pas dans le dictionnaire, en conservant l'ordre et la capitalisation de la liste passée en second argument.

Question 5 - Deep copy (5 points)

Ecrivez une fonction qui reçoit en paramètre une liste pouvant contenir des sous-listes et qui effectue une copie en profondeur de la liste de départ, c'est-à-dire qu'il n'y a au final aucune sous-liste partagée entre l'originale et la copie. Votre fonction doit pouvoir traiter des listes imbriquées de n'importe quelle profondeur.

Rappel : Pour tester le type de la valeur associée à une variable `a`, vous pouvez utiliser l'opérateur `type(a)`. Par exemple, `type(a) == int` renverra `true` si la valeur associée à `a` est un entier. Les types vus au cours sont `int`, `str`, `float`, `list`, `tuple`, `dict`, `bool` et `set`.

Question 6 - Tri de polynômes (7 points)

On vous demande d'écrire une fonction qui reçoit deux arguments, une liste de polynômes et un nombre, et qui renvoie une nouvelle liste contenant les polynômes triés par ordre croissant en fonction de leur valeur en ce nombre.

Les polynômes sont donnés dans la représentation vue au cours, c'est-à-dire sous la forme d'une liste `p` telle que `p[i]` contient le coefficient de x^i.

Faites particulièrement attention à l'efficacité de vos calculs et au découpage en fonctions. En particulier, votre solution doit rester raisonnablement efficace tant du point de vue du tri (si la liste de polynômes est grande) que du point de vue du calcul des polynômes (si les polynômes ont beaucoup de termes).

BON TRAVAIL !

UNIVERSITE LIBRE DE BRUXELLES 30 août 2012
Ecole polytechnique de Bruxelles

INFO-H-100 – Informatique – Programmation – Prof. Th. Massart
1ère année du grade de Bachelier en Sciences de l'Ingénieur
Correction de l'examen de seconde session

Question 1 - Python (1 point)

```
t=range(3)
t2=[t]*2
print t2
t3=t2[:]
t=[5,6,7]
print t2
print t3
```

Le code imprime

```
[[0, 1, 2], [0, 1, 2]]
[[0, 1, 2], [0, 1, 2]]
[[0, 1, 2], [0, 1, 2]]
```

A la ligne 1, on créé une liste désignée par `t`. A la ligne 2, on créé une nouvelle liste dont les deux éléments sont la liste désignée par `t`. A la ligne 4, on créé une copie de `t2`, c'est-à-dire qu'on créé une nouvelle liste dont les deux éléments désignent toujours la même liste que `t`. Enfin, à la ligne 5, on créé une nouvelle liste, puis on change la flèche de `t` pour pointer vers cette nouvelle liste. La liste originale n'est pas modifiée et reste désignée par les 4 éléments de `t2` et `t3`.

Question 2 - Python et Complexité (2 points)

Soit les fonctions suivantes :

```
def f1(nmax):
    nums = range(2,nmax+1)                  # O(nmax)
    for i in range(2, int(sqrt(nmax))+1):   # O(sqrt(nmax)) * O(len(nums))
        f = lambda x: x == i or x % i != 0  # # O(1)
        nums = filter(f, nums)              # # O(len(nums))
    return nums                             # O(1)
```

La fonction calcule l'ensemble des nombres premiers plus petits que `nmax`, qui doit être un entier plus grand ou égal à 2. L'analyse de complexité est indiquée dans le code ; la seule inconnue qui reste est la valeur de `len(nums)` à chaque itération. En comptant large, on obtient $\mathcal{O}$(`sqrt(nmax)*nmax`).

```
def f2(a):
    a = a.split()               # O(len(a))
    b = {}                      # O(1)
    num = 0                     # O(1)
    for i in a:                 # O(len(a)) * O(len(b))
```

```
        b[i] = b.get(i,0) + 1  # # O(len(b))
        if b[i] == 1:          # # O(len(b))
            num += 1           # # O(1)
    return num                 # O(1)
```

La fonction reçoit un texte sous la forme d'une chaîne de caractères et renvoie le nombre de mots différents dans le texte, en considérant qu'un mot est tout ensemble de lettres séparées par des espaces. Pour connaître la complexité finale, il faut connaître `len(b)` à chaque itération. Au pire cas, tous les mots sont différents et on obtient $\mathcal{O}((\texttt{nombre de mots différents dans a})^2)$.

Question 3 - Compilation / interprétation

Voir notes sur l'Université Virtuelle.
Il fallait parler
- des 6 phases (3 d'analyse (lexicale, syntaxique et sémantique) et 3 de synthèse (génération du code intermédiaire, optimisation et génération du code final)
- expliquer que l'interpréteur décode chaque instruction python une par une et exécute l'effet voulu. Contrairement à un langage compilé où on a 2 étapes bien séparées (compilation et exécution).
- que la définition du langage implique une gestion d'un stack (pile) et d'un heap (tas) run time que le langage soit compilé ou interprété.

Question 4 - Correcteur orthographique

Version la plus simple, si on pense à utiliser un `set` :

```
def bad_words(good_words, text):
    d = set(good_words)
    return filter(lambda w: w.upper() not in d, text)
```

Sinon, il faut construire un dictionnaire. Plusieurs méthodes sont possibles ; la plus simple est sans doute :

```
def bad_words(good_words, text):
  d = dict(zip(good_words, good_words))
  return filter(lambda w: w.upper() not in d, text)
```

Question 5 - Deep copy

```
def deep_copy(li):
    new = []
    for e in li:
        if type(e) == list:
            new.append(deep_copy(e))
        else:
            new.append(e)
    return new
```

Question 6 - Tri de polynômes

```
def eval_poly(poly, val):
```

```
    prev_x = 1
    tot = 0
    for coeff in poly:
        tot += coeff * prev_x
        prev_x *= val
    return tot
def selection_sort(ls, comp):
    for i in range(len(ls)-1):
        pos = pos_min_from(ls, i, comp)
        ls[i], ls[pos] = ls[pos], ls[i]
    return ls
def pos_min_from(ls, i, comp):
    res = i
    for n in range(i, len(ls)):
        if comp(ls[n], ls[res]):
            res = n
    return res
def sort_poly(polys, x):
    liste_val_poly = map(lambda p: {"valeur_en_x" : eval_poly(p, x),
                                    "polynome" : p}, polys)
    liste_triee = selection_sort(liste_val_poly,
                                 lambda p1, p2: p1["valeur_en_x"] < p2["valeur_en_x"])
    return map(lambda p: p["polynome"], liste_triee)
```