UNIVERSITE LIBRE DE BRUXELLES 9 juin 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 juin

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.
- Vous disposez de 2 heures 45 minutes et vous ne pouvez pas utiliser de notes.
- Si du code vous est demandé,
  - la réponse à la question doit comprendre le code *Python* structuré et conforme aux règles de bonne pratique et conventions,
  - sauf mention contraire, vous ne pouvez utiliser aucune fonction de librairies (pas d'`import`)
  - veillez à découper votre réponse en fonctions de manière pertinente
  - veillez à utiliser des structures de données appropriées.

Question 1 - Complexité (3 points)

Les 2 fonctions `equality_list` et `equality_dic` renvoient toutes les deux la valeur vraie si et seulement si la liste `b` est une permutation de la liste `a` (tous les éléments de `a` (resp. `b`) sont dans `b` (resp. `a`) et en même quantité).

```
def equality_dic(a,b):
    res = False
    if len(a)==len(b):
       dic_a = {}
       dic_b = {}
       for i in a:
           dic_a[i]=dic_a.get(i,0)+1
       for i in b:
           dic_b[i]=dic_b.get(i,0)+1
       res = True
       for j in dic_a:
           res = res and (dic_a[j] == dic_b.get(j,0))
       for j in dic_b:
           res = res and (dic_b[j] == dic_a.get(j,0))
    return res
```

```
def equality_list(a,b):
    res = False
    if len(a)==len(b):
       res = True
       for i in a:
           if i in b:
               a.remove(i)
               b.remove(i)
           else:
               res = False
    return res
```

En supposant que les éléments soient entiers, donner la complexité maximale en O du temps d'exécution de chacune des deux fonctions. Veillez à détailler vos réponses en justifiant la complexité de chaque partie du code.

Question 2 - 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 3 - Hachage (4 points)

ATTENTION : cette question doit être résolue à l'aide de `map`, `reduce` et `filter`. Vous n'avez pas le droit d'utiliser de boucle (`for`, `while`).

En programmation, il est souvent utile de pouvoir réduire une chaîne de caractères à un nombre (pour des comparaisons grossières et rapides). On vous demande d'écrire une fonction qui reçoit en paramètre une chaîne de caractères et qui renvoie comme résultat un entier calculé de la manière suivante : après avoir retiré tous les caractères qui ne sont pas des lettres, remplacer chaque lettre par sa valeur ASCII et calculer la somme de toutes ces valeurs ASCII. Notez que, par exemple, les mots `ab` et `ba` correspondront au même entier.

Rappel : la fonction `ord(c)` prend en argument un caractère (chaîne de caractères de longueur 1) et donne comme résultat sa valeur ASCII (entier).

UNIVERSITE LIBRE DE BRUXELLES
Ecole polytechnique de Bruxelles

9 juin 2012

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

Question 1 - Complexité (3 points)

```
def equality_dic(a,b):                                  O(n^2)
    res = False                                         O(1)
    if len(a)==len(b):                                  O(n^2)
       dic_a = {}                                       O(1)
       dic_b = {}                                       O(1)
       for i in a:                                      O(n^2)
           dic_a[i]=dic_a.get(i,0)+1                    O(n)
       for i in b:                                      O(n^2)
           dic_b[i]=dic_b.get(i,0)+1                    O(n)
       res = True                                       O(1)
       for j in dic_a:                                  O(n^2)
           res = res and (dic_a[j] == dic_b.get(j,0))   O(n)
       for j in dic_b:                                  O(n^2)
           res = res and (dic_b[j] == dic_a.get(j,0))   O(n)
    return res                                          O(1)
```

Explications :
La complexité au pire des cas est celle où `len(a) = len(b) = n`. L'algorithme est un $\mathcal{O}(n^2)$ car

- une séquence d'instructions simples en $\mathcal{O}(1)$ et d'une `if` qui lui même contient une séquence avec des instructions en $\mathcal{O}(1)$ et des `for` qui sont tous au pire en $\mathcal{O}(n^2)$.
- Chaque `for` tourne de l'ordre de `n` fois et chaque itération fait un ou 2 accès à un élément du dictionnaire en lecture ou écriture (au pire en $\mathcal{O}(n)$).
- Les règles du `for` (multiplie les complexités du corps et du nombre d'itérations) du `if` et des séquences (max des complexités) nous donnent le résultat.

```
def equality_list(a,b):                 O(n^2)
    res = False                         O(1)
    if len(a)==len(b):                  O(n^2)
       res = True                       O(1)
       for i in a:                      O(n^2)
           if i in b:                   O(n)
               a.remove(i)              O(n)
               b.remove(i)              O(n)
           else:
               res = False              O(1)
    return res
```

Explications : La complexité au pire des cas est celle où `len(a) = len(b) = n`
L'algorithme est un $\mathcal{O}(n^2)$ car

- le test `if i in b` ainsi que les `remove` sont en $\mathcal{O}(n)$ (recherche ou enlève dans une liste à n éléments
- le `if i in a` complet est donc formé d'une séquence qui au pire des cas est 3 fois en $\mathcal{O}(n)$ donc en $\mathcal{O}(n)$
- le `for` boucle de l'ordre de $\mathcal{O}(n)$ fois donc au total on obtient une comlexité pour l'instruction for en $\mathcal{O}(n^2)$
- le `if len(a)==len(b)` est donc aussi en $\mathcal{O}(n^2)$ ainsi que l'algorithme complet.

Question 2 - Compilation / interprétation (2 points)

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 3 - Hachage (4 points)

```
def isAlpha(c):
    """True ssi c est une lettre alphabetique"""
    return ('A' <= c <= 'Z') or ('a' <= c <= 'z')

def hachage(texte):
    texte = filter(isAlpha, texte)          # Ne garde que les alphabetiques
    ls    = map(ord, texte)                 # Liste des codes ASCII de chaque lettre
    return reduce(lambda a,b: a+b, ls, 0)   # ou: return sum(ls)
```

Question 4 - Flatten (5 points)

```
def flatten(l):
    r = []
    for el in l:
        if type(el) == list:
            r.extend(flatten(el))
        else:
            r.append(el)
    return r
```

Question 5 - Cryptage de fichier (6 points)

```
def openTrans(transfile):
        f = open(transfile)
        data = f.read().split()
        trans = {}
        for i in range(0,len(data),2):
                trans[data[i]] = data[i+1]
        f.close()

        return trans

def removeExtension(filename):
        i = len(filename)-1
        while i >= 0 and filename[i] != '.':
                i -= 1

        if i == -1:
                result = filename
        else:
                result = filename[:i]

        return result

def openSource(sourcefile):
        f = open(sourcefile)
        data = f.read()
        f.close()
        return data

def writeCrypt(crypt, cryptFile):
        f = open(cryptFile,'w')
        f.write(crypt)
        f.close()

def createCrypt(trans, source):
        source = list(source)    # travailler avec une liste et pas un string permet d'etre en temps lineaire
        for i in range(len(source)):
                source[i] = trans.get(source[i], source[i])

        return ''.join(source)

def translate(transfile, sourcefile):
        source = openSource(sourcefile)
        trans = openTrans(transfile)
        crypt = createCrypt(trans, source)
        # crypt = ''.join(map(lambda x: trans.get(x,x), source))        # ou avec un map
        cryptFile = removeExtension(sourcefile) + ".crypt"
        # cryptFile = '.'.join(sourcefile.split('.')[:-1] + ["crypt"])  # ou en travaillant sur le string
        writeCrypt(crypt, cryptFile)

translate("trans.txt", "bonjour.txt")
```

Question 4 - Flatten (5 points)

Ecrire une fonction qui "aplatit" une liste à plusieurs niveaux (une liste dont chaque élément est soit une valeur de n'importe quel type sauf une liste, soit une liste à plusieurs niveaux) en une liste à un seul niveau contenant les mêmes éléments, dans le même ordre. Ici, "dans le même ordre" signifie que si on devait écrire les deux listes littéralement, les valeurs apparaitraient dans le même ordre.

Exemples :

```
>>> flatten([[[[0]], 5, [3]], 4])
[0, 5, 3, 4]
>>> flatten([5, [2, [3, 7], [[4], 2], [[], 8]]])
[5, 2, 3, 7, 4, 2, 8]
>>>
```

Rappel : on peut demander à Python quel est le type de la valeur associée à une variable à l'aide de la fonction `type`. Par exemple, le code suivant :

```
x = 10
if type(x) == int:
    print "C'est un entier !"
else:
    print "Ce n'est pas un entier."
```

imprimera à l'écran "C'est un entier !". Les types vus au cours sont `int`, `float`, `bool`, `list`, `str`, `dict` et `tuple`.

Question 5 - Cryptage de fichier (6 points)

On vous demande d'écrire une fonction translate qui reçoit en paramètres deux noms de fichiers. Le premier fichier doit contenir, sur chaque ligne, deux caractères séparés par un espace. Le second fichier est un fichier texte que l'on souhaite crypter à l'aide du premier fichier. Votre programme doit créer un nouveau fichier, dont le nom est similaire au second fichier mais avec une extension différente (la nouvelle extension doit être `crypt`), et dont le contenu est celui du second fichier dans lequel tous les caractères qui apparaissent dans la première colonne du premier fichier ont été remplacés par le caractère correspondant dans la seconde colonne. Les caractères qui n'apparaissent pas dans le premier fichier doivent être laissés intacts.

Par exemple, si le premier fichier, `trans.txt`, contient les deux lignes

```
a b
z d
```

et que le fichier `bonjour.txt` contient le texte

```
Bonjour, comment allez-vous ?
```

l'appel de fonction

```
translate("trans.txt", "bonjour.txt")
```

devrait créer sur le disque le fichier `bonjour.crypt` contenant le texte

```
Bonjour, comment blled-vous ?
```

Rappel : L'extension d'un fichier est le groupe de lettres qui se trouve après le dernier point dans le nom du fichier (`txt` dans `dico.txt` par exemple).

Vous pouvez considérer que les fichiers se trouvent dans le même dossier que le programme.

BON TRAVAIL !