UNIVERSITÉ LIBRE DE BRUXELLES
École polytechnique de Bruxelles

Janvier 2015

INFO-H-100 – Informatique – Prof. Th. Massart
1ère année du grade de Bachelier en Sciences de l'Ingénieur
Interrogation de janvier

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 trois heures 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 - Théorie (7 points)

Pour chacun des codes suivants,
- expliquez ce qu'il fait et ce qu'il imprime
- donnez l'état du programme lors de chaque `print` grâce à des diagrammes d'état (comme fait au cours).
- donnez la complexité moyenne et maximale ($\mathcal{O}()$) de la **fonction** `foo_`i, $i = 1..5$) en temps d'exécution. Précisez bien les paramètres utilisés pour exprimer la complexité et justifiez bien vos réponses (le résultat seul ne suffit pas pour obtenir des points).

NB : dans la fonction `foo_2`, l'opérateur `|` dénote l'union ensembliste.

1.
```
def foo_1(liste, max):
    length = len(liste)
    new_list = [0]*length
    count = [liste.count(i) for i in range(max)]
    print(count)
    count = [sum(count[:i]) for i in range(1,max+1)]
    print(count)
    for j in range(length):
        count[liste[j]] -= 1
        new_list[count[liste[j]]] = liste[j]
    liste[:] = new_list

>>> m = 5
>>> t = [1,3,3,1,0,4]
>>> foo_1(t,m)
>>> print(t)
```

2.
```
def foo_2(liste):
    dico = {}
    for elem in liste:
        dico[liste.count(elem)] = dico.setdefault(liste.count(elem), set({})) | {elem}
    print(dico)
    return dico

>>> u = ['a','b','c','a','c','d','a','d']
>>> foo_2(u)
```

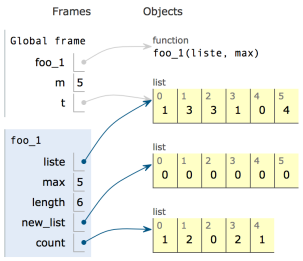
3.
```
def foo_3(matrix):
    for i in range(len(matrix)):
        print(matrix[-i-1][i])

>>> m =[[ 1, 2, 3, 4],
...     [ 5, 6, 7, 8],
...     [ 9,10,11,12],
...     [13,14,15,16]]
>>> foo_3(m)
```

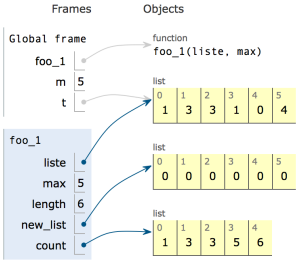
4.
```
def foo_4(res, remain):
    print(res)
    for i in range(len(remain)):
        foo_4(res + remain[i], remain[:i] + remain[i+1:])

>>> foo_4("", "ab")
```

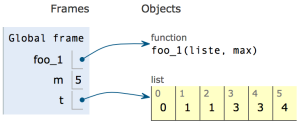
5.
```
class English_length (object):
    def __init__(self, m, y):
        self.miles = m
        self.yards = y
    def __str__(self):
        return "{0:02d} : {1:4.2f}".format(self.miles, self.yards)

def foo_5():
    long = English_length(2, 3.0)
    print(long)
    long2 = English_length(2)
    print(long2)

>>> foo_5()
```

Solution

1. Tri par énumération de liste (un seul champs avec la clé dont la valeur est dans [0..m[
   Imprime :
   [1, 2, 0, 2, 1] : `count[i]` contient le nombre de valeurs `i` dans liste
   [1, 3, 3, 5, 6] : `count[i]` contient le nombre de valeurs 0 à `i` dans liste
   [0, 1, 1, 3, 3, 4] : liste après le tri (par énumération)







Complexités

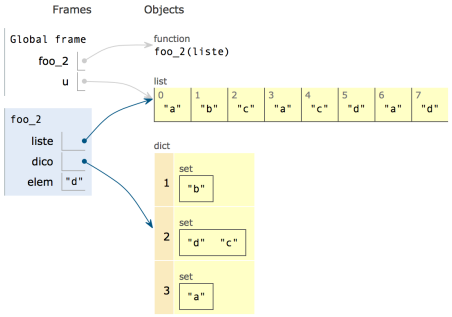
```
                          complexités                    moyenne          max
def foo_1(liste, max):
    length = len(liste)                                  O(1)             O(1)
    new_list = [0]*length                                O(length)        O(length)
    count = [liste.count(i) for i in range(max)]         O(max.length)    O(max^2.length)
    print(count)                                         O(max)           O(max)
    count = [sum(count[:i]) for i in range(1,max+1)]     O(max^2)         O(max^3)
    print(count)                                         O(max)           O(max)
    for j in range(length): au total de la boucle for    O(length)     O(length)
        count[liste[j]] -= 1                                O(1)             O(1)
        new_list[count[liste[j]]] = liste[j]                O(1)             O(1)
    liste[:] = new_list                                  O(length)        O(length)
```

Détails :

- `new_list` est créée avec `length` éléments
- un comptage prend un temps proportionnel au nombre d'éléments dans la liste (`length`)
- la liste count est créée avec `max` éléments (la création peut prendre au maximum `O(max^2.length)` si recopie de la liste à chaque rajout d'un élément et `O(max^3)` pour la partie cumul des valeurs : d'où les complexités moyennes et maximales.
- imprimer une liste prend un temps proportionnel au nombre d'éléments

Complexité moyenne : `O(max.maximum(max, length))`
Complexité maximale : `O(max^2.maximum(max, length))`
Si on suppose `max << length` : complexité moyenne et maximale : `O(length)`

2. Créée un dictionnaire `dico` dont les clés sont le nombre d'occurences des différents élements de la `liste`, et les champs sont les éléments de la liste.
   Imprime : `{1: {'b'}, 2: {'d', 'c'}, 3: {'a'}}`

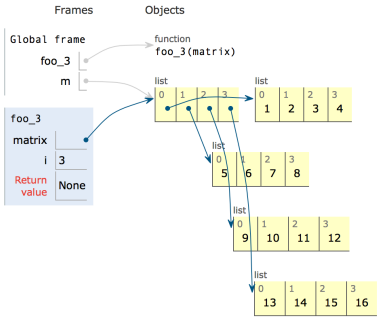


Complexités
Hypothèses : peut de valeurs semblales et n = len(liste)

```
                                        complexités                    moyenne        max
def foo_2(liste):                                                  =>   O(n^2)          O(n^3)
    dico = {}                                                           O(1)           O(1)
    for elem in liste:                   au total de la boucle for          O(n^2)          O(n^3)
        dico[liste.count(elem)] ...                                     O(n)           O(n^2)
    print(dico)                                                         O(n)           O(n)
    return dico                                                         O(1)           O(1)
```

Complexité moyenne : `O(n^2)`
Complexité maximale : `O(n^3)`

3. Imprime la seconde diagonale de la matrice carrée `matrix`
   Imprime :
   13
   10
   7
   4



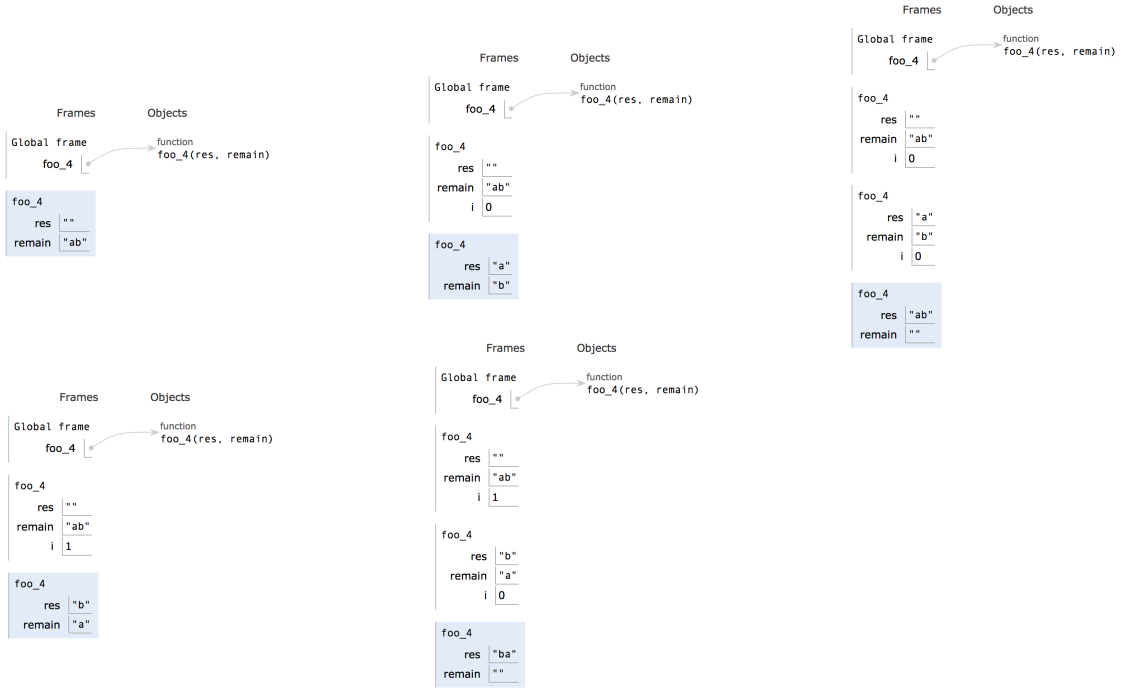
Complexités
Hypothèses : n = len(matrix) = len(matrix[i]) pour toutes les sous listes de matrix et les éléments sont des données simples

```
                                        complexités                    moyenne        max
def foo_3(matrix):                                                      O(n)           O(n)
    for i in range(len(matrix)):                                        O(n)           O(n)
        print(matrix[-i-1][i])                                        O(1)           O(1)
```

Complexité moyenne = Complexité maximale : `O(n)`

4. Imprime les permutations de remain ainsi que les sous séquences de ces permutations
   Imprime :
   (string vide)
   a

ab
b
ba



Complexités

Hypothèses : n = len(remain) de départ et remain est une séquence d'éléments simples (caractères par exemple)

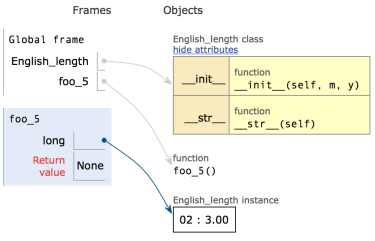
La complexité moyenne et maximale est la même :

```
def foo_4(res, remain):                                    O(n . n!)
    print(res)                                             O(n)
    for i in range(len(remain)):                          multiplie par n
        foo_4(res + remain[i], remain[:i] + remain[i+1:])  appel est en O(n)
```

`foo_4` appelle n instances d'elle même (récursivement) dont chacune appelle $n-1$ instances ... : au total $n!$ instances ; à chaque instance le `print(res)` est en `O(n)` : la complexité est dont en `O(n.n!) = O(n^n)`

5. Imprime la valeur de l'objet `long` ensuite une erreur car pour construire Long2, il faut 2 arguments qui sont manquants :

```
TypeError: __init__() missing 1 required positional argument: 'y'
```



Complexités

Si les paramètres sont simples, tout est en O(1) (en moyenne ou au maximum). Sinon, la complexité de l'impression vaut la taille des champs (miles ou yards)

Question 2 - Manipulation de fichiers et de données (5 points)

Considérons un fichier de données contenant les résultats numériques d'un ensemble d'expériences. Chaque ligne de ce fichier correspond à une expérience et chaque colonne correspond à l'un des résultats.
Ecrivez une fonction `file_stats(file_name)` qui reçoit en paramètre le *nom* d'un fichier `file_name` et qui renvoie une liste contenant, pour chaque colonne, des informations statistiques (le minimum, le maximum et la moyenne) pour l'ensemble des expériences. Ces informations statistiques seront stockées sous forme de dictionnaire, en utilisant les clés `min`, `max` et `avg`.
Nous supposerons que les différentes valeurs d'une ligne sont séparées par un point-virgule (" ;").

Exemple. Soit le fichier de données `experiment.dat`, contenant les données suivantes :

```
0.1; -12;  5
0.9;   2; 10
0.6;  -1;  8
0.4;   3;  3
```

Pour ce fichier, la fonction que nous vous demandons d'écrire doit renvoyer le résultat suivant :

```
>>> stats = file_stats("experiment.dat")
>>> print(stats)
[{'max': 0.9, 'min': 0.1, 'avg': 0.5},
 {'max': 3.0, 'min': -12.0, 'avg': -2.0},
 {'max': 10.0, 'min': 3.0, 'avg': 6.5}]
```

Solution.

Solution sans compréhension de listes

```
def clean(data):
   """Lit toutes les lignes du fichier data composées de lignes de valeurs
   float séparées par des et crée une matrice avec les valeurs float  """
   res = []
   for line in data:
      line = line.strip().split(';')
      float_line = []
      for x in line:
         float_line.append(float(x))
      res.append(float_line)
   return res

def transpose(data):
   """Renvoie la transposée de la matrice data reéçue en paramètre"""
   res  = []
   for i in range(len(data[0])):
      line = []
      for row in data:
         line.append(row[i])
      res.append(line)
   return res

def stat(ls):
   """Renvoie un triple avec le minimum, le maximum et la moyenne des valeurs de la liste ls"""
   return(min(ls), max(ls), sum(ls)/len(ls))

def file_stats(file_name):
   """Lit un fichiers de tuples et crée, une liste avec, pour chaque colonne, un dictionnaire
   contenant le min, max et la moyenne pour les valeurs de la colonne"""
   res = []
   data = open(file_name).readlines()
   data = clean(data)
   data = transpose(data)
   for line in data:
      (min, max, avg) = stat(line)
      res.append({'min' : min, 'max' : max, 'avg' : avg})
   return res
```

Solution avec compréhensions de listes

```
def clean(data):
   """Lit toutes les lignes du fichier data composées de lignes de valeurs float séparées par des ';'
   et crée une matrice avec les valeurs float  """
   return [[float(x) for x in line.strip().split(';')] for line in data]

def transpose(data):
   """Renvoie la transposée de la matrice data reçue en paramètre"""
   return [[row[i] for row in data] for i in range(len(data[0]))]

def stat(ls):
   """Renvoie un triple avec le minimum, le maximum et la moyenne des valeurs de la liste ls"""
   return(min(ls), max(ls), sum(ls)/len(ls))

def file_stats(file_name):
   """Lit un fichiers de tuples et crée, une liste avec, pour chaque colonne, un dictionnaire
   contenant le min, max et la moyenne pour les valeurs de la colonne"""
   res = []
   data = open(file_name).readlines()
   data = clean(data)
   data = transpose(data)
   for line in data:
      (min, max, avg) = stat(line)
      res.append({'min' : min, 'max' : max, 'avg' : avg})
   return res
```

Question 3 - Tri (4 points)

On considère une liste de mots et on vous demande d'écrire la fonction `len_sort(ls)` qui trie la liste `ls` dans l'ordre croissant de la longueur des mots en utilisant le tri *par insertion*.

Exemple.

```
>>> words = ['ceci','est','un','exemple']
>>> len_sort(words)
>>> print(words)
['un', 'est', 'ceci', 'exemple']
```

Solution.

```
def len_sort(words):
    """Trie la liste de mots par taille croissante"""
    for i in range(1, len(words)):
        val = words[i]
        j = i
        while j > 0 and len(words[j - 1]) > len(val):
            words[j] = words[j - 1]
            j -= 1
        words[j] = val
```

Question 4 - Récursivité (4 points)

Soit une liste ne contenant que des nombres ou des sous-listes elles-même ayant semblable contenu. Écrivez la fonction `recursive_sum(ls)` qui calcule la somme de tous les nombres contenus dans la liste `ls` et ses sous-listes :

Exemple.

```
>>> ls = [1, [], [2, 3, [4]]]
>>> print(recursive_sum(ls))
10
```

Note. Pour rappel, il est possible de vérifier qu'une variable `x` est une liste en utilisant l'expression booléenne `type(x) == list`.

Solution.

```
def recursive_sum(ls):
    res = 0
    for x in ls:
        if type(x) == list:
            x = recursive_sum(x)
        res += x
    return res
```