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

Prénom : Nom : Matricule :

Remarques préliminaires

- On vous demande de répondre à **chaque question sur une feuille séparée**.
- N'oubliez pas d'inscrire votre prénom, nom et numéro de matricule sur chaque feuille.
- Vous disposez de trois heures et vous ne pouvez pas utiliser de notes.
- Vous pouvez utiliser le verso des feuilles pour répondre.
- 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 - Fonctionnement de Python (2 points)

Expliquer le mode de fonctionnement d'un compilateur et d'un interpréteur. Python de base est-il interprété ou compilé ? Expliquez quels en sont les avantages et les inconvénients pour ce langage.

Solution. → Voir syllabus.

Question 2 - Code Python (5 points)

Pour chacun des codes suivants,

- donnez ce qu'il imprime (avec les appels qui sont donnés) et expliquez de façon générale ce que fait la fonction ou la classe (si elle résout un problème, lequel ...) ;
- donnez l'état du programme lors de chaque (groupe de) `print` grâce à des diagrammes d'état (comme fait au cours) ;
- sachant que les types des paramètres sont,
  - a et n : nombre naturel,
  - b, c, d : liste d'entiers que vous pouvez supposer de taille n,

  donnez la complexité **moyenne** ($\mathcal{O}()$) de la **fonction** `foo_`i ($i = 1..4$), et des méthodes de la classe Hell (sous-question 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).

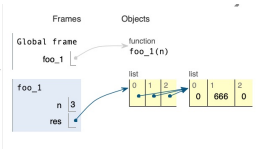
Notez bien : pour expliquer ce que le code fait et donner la complexité, vous pouvez annoter les codes fournis (compléter les docstrings, commenter, ...)

1.
```python
def foo_1(n):                # O(n^2)
    res = [[0]*n]*n            # O(n) car crée une sous-liste + le liste principale
    res[1][1] = 666            # O(1)
    print(res)                 # O(n^2)
    return res                 # O(1)

print(foo_1(3))
```

Solution :
Imprime :

```
[[0, 666, 0], [0, 666, 0], [0, 666, 0]]
[[0, 666, 0], [0, 666, 0], [0, 666, 0]]
```

Diagramme d'état :



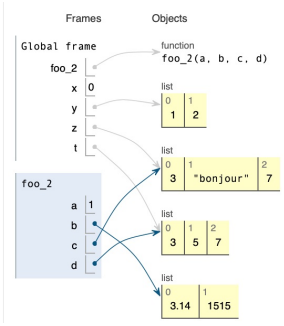
Le second diagramme d'état, après la sortie de la fonction, est moins intéressant (voir PythonTutor).
Complexité moyenne de la fonction `foo_1` = $O(n^2)$ (voir code) à cause du print. Notez que la partie d'instruction `[[0]*n]*n` est en complexité moyenne (et maximale) en $\mathcal{O}(\backslash)$ étant donné que le code ne crée qu'une seule sous liste ($\mathcal{O}(\backslash)$) et ensuite la liste principale ($\mathcal{O}(\backslash)$) ; la somme des deux est en $\mathcal{O}(\backslash)$.

2.
```python
def foo_2(a, b, c, d): # O(1)
    a = 1                  # O(1)
    b = [3.14, 1515]       # O(1)
    c[1] = "bonjour"       # O(1)
    d[:] = [3, 5, 7]       # O(1)
    print(a)               # O(1)
    print(b)               # O(1)
    print(c)               # O(1)
    print(d)               # O(1)

x = 0
y = [1,2]
z = [3,5,7]
t = [11, 13, 17]
foo_2(x, y, z, t)
print(x)
print(y)
print(z)
print(t)
```

Solution :
Imprime :

```
1
[3.14, 1515]
[3, 'bonjour', 7]
[3, 5, 7]
0
[1, 2]
[3, 'bonjour', 7]
[3, 5, 7]
```

Diagramme d'état :



Le second diagramme d'état est moins intéressant (cf PythonTutor).
Ce code ne semble pas résoudre de problème spécifique.
Complexité moyenne de la fonction `foo_2` : $O(1)$. Justification supplémentaire : cf support de notes.

3.
```python
def foo_3(n):                  # O(n^2)
    return  [i for i in range(3*n) if i % 3 != 0] + [666]*(n**2) # O(n^2)

print(foo_3(3))
```

Solution :
Imprime :

```
[1, 2, 4, 5, 7, 8, 666, 666, 666, 666, 666, 666, 666, 666, 666]
```

Diagrammes d'état :

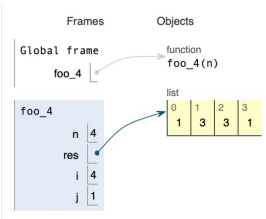
Complexité moyenne : $O(n)$.

4.
```python
def foo_4(n):                           # O(n^2)
    res = [1]                           # O(1)
    for i in range(1, n+1):             # (répétitions : O(n)) : O(n^2)
        print(res)                       # O(n)
        res.append(1)                    # O(1)
        for j in range(i-1,0,-1):        # (répétition : O(n)) : O(n)
            res[j] = res[j] + res[j-1]    # O(1)
    return res                          # O(1)


print(foo_4(4))
```

Solution :
Imprime :

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
```

Diagrammes d'état :



Les autres diagrammes d'état est moins intéressant (voir pythontutor pour l'obtenir).
Ce code produit une liste avec les coefficient de polynôme $(x+1)^n$ (triangle de Pascal)
Complexité : Chaque itération du `for` prend un temps moyen en $O(i)$ (avec $i = 1, 2, n$). Au total $O(n + (n-1) + (n-2) + ...) = O(n^2)$

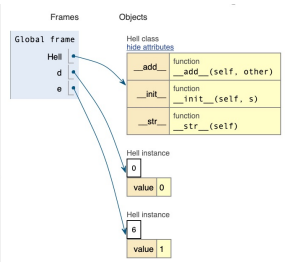
5. **Solution :**

```python
class Hell(object):

    def __init__(self, s): # O(1)
        self.value = s     # O(1)

    def __str__(self):     # O(1)
        return str(self.value * 6)  # O(1)

    def __add__(self, other):       # O(1)
        return Hell(111 * (self.value + other.value)) # O(1)

d = Hell(0)
e = Hell(1)
print(d)
print(e)
print(e+d)
```

Solution :
Imprime :

```
0
6
666
```

Diagrammes d'état :



Ce code crée une nouvelle classe d'objet Hell ainsi que 3 objets de cette classe (`d`, `e` et `e+d`).

Question 3 - Fichiers (5 points)

Soit un fichier contenant les notes des étudiants pour différents travaux. Nous supposons que chaque étudiant possède un nom unique qui s'écrit en un seul mot. Le fichier est structuré avec un étudiant par ligne, chaque ligne commençant par son nom suivi par ses différentes notes, le tout séparé par des espaces. Les étudiants ne doivent pas tous avoir le même nombre de notes.

Par exemple :

```
Paul 16 18 20 18
John 12 9 3
Ringo 3.5 5 8
Georges 20 16
```

Ecrivez une fonction `compute_means(file_name_in, file_name_out)` qui lit le fichier de notes nommé `file_name_in` et qui écrit le fichier des moyennes nommé `file_name_out` structuré de la façon suivante :

Le fichier de moyenne contiendra une ligne pour chaque note moyenne rencontrée, dans l'ordre des moyennes croissantes, et pour chaque moyenne, la suite des noms d'étudiants ayant obtenu cette moyenne, le tout séparé par des espaces. Pour le fichier de notes ci-dessus, le fichier de moyennes produit sera :

```
5.5 Ringo
8.0 John
18.0 Paul Georges
```

Solution.

```python
def moyenne(line):
    return sum([float(x) for x in line]) / len(line)

def file_in_to_dict(file_name_in):
    """
        renvoie un dictionnaire avec pour chaque moyenne rencontrée,
        la liste des noms d'étudiants correspondants
        """
    d = {}
    for line in open(file_name_in):
        line = line.split()
        d.setdefault(moyenne(line[1:]), []).append(line[0])
    return d

def write_dict_to_file(d, file_name_out):
    """
        crée le fichier de moyennes au départ du dictionnaire
        """
    fout = open(file_name_out, 'w')
    for moy in sorted(d.keys()):
        fout.write(str(moy))
        for nom in d[moy]:
            fout.write(' ' + nom)
        fout.write('\n')
    fout.close()

def compute_means(file_name_in, file_name_out):
    d = file_in_to_dict(file_name_in)
    write_dict_to_file(d, file_name_out)

compute_means("in.txt", "out.txt")
```

Question 4 - Tri (4 points)

Vous partez en tour du monde et décidez de choisir un compagnon qui parlera le plus de langues pour vous aider à communiquer avec les locaux tout au long du voyage. Dans la liste de vos compagnons, sont indiquées les langues qu'ils parlent chacun.

Nous supposons aussi, pour simplifier, qu'une seule langue est parlée dans chaque pays.

Ecrivez une fonction `my_sort(companions, countries)` qui va trier la liste de vos compagnons dans l'ordre du plus utile au moins utile. Le compagnon jugé le plus utile sera celui qui parlera la langue du plus grand nombre de pays que vous allez visiter.

Les compagnons sont représentés par un tuple `(nom, [langues])`, avec leur nom et la liste de langues qu'ils parlent.

Tous les pays que vous comptez visiter sont stockés dans le dictionnaire `countries` avec la langue du pays.

Vous êtes libres d'utiliser le tri vu au cours ou au TP de votre choix.

Par exemple :

```
countries = {
    "Argentine" : "espagnol",
    "Brésil" : "portugais",
    "Etats-Unis" : "anglais",
    "Equateur" : "espagnol"
}

companions = [
    ("jean", ["espagnol", "anglais"]),
    ("jp", ["italien"]),
    ("ken", ["portugais", "anglais"]),
    ("alain", ["italien", "anglais", "portugais", "grec"])
]

>>> my_sort(companions, countries)
>>> print(companions)
[('jean', ['espagnol', 'anglais']), ('ken', ['portugais', 'anglais']),
 ('alain', ['italien', 'anglais', 'portugais', 'grec']), ('jp', ['italien'])]
```

En effet, Jean est le premier de la liste puisqu'il parle la langue de trois pays visités, suivi de Ken et Alain qui parlent la langue de deux pays visités et enfin JP qui ne parle aucune langue des pays visités.

Solution.

```
def nb_langues(a, pays):
    """Dans combien de pays, a parle t-il la langue"""
    cpt = 0
    for lg in pays.values():
        if lg in a[1]:
            cpt += 1
    return cpt

def vient_avant(a, b, pays):
    return nb_langues(a, pays) > nb_langues(b, pays)

def min_pos_from(ls, start, pays):
    res = start
    for j in range(start + 1, len(ls)):
        if vient_avant(ls[j], ls[res], pays):
            res = j
    return res

def swap(ls, i1, i2):
       ls[i1], ls[i2] = ls[i2], ls[i1]

def my_sort(ls, pays):
    for i in range(len(ls) - 1):
        pos_min = min_pos_from(ls, i, pays)
        swap(ls, i, pos_min)
```

Question 5 - Récursivité (4 points)

Sur nos ordinateurs, les fichiers sont organisés en dossiers (folders). Un dossier peut contenir des fichiers et d'autres (sous-)dossiers.

Un dossier est représenté par un tuple `(nom, [contenu])` formé de son nom et d'une liste de contenus. Un fichier est représenté par un tuple `(nom, taille)` formé de son nom et de sa taille en octets (un entier).

Écrivez une fonction **récursive** `biggest_file(folder)` qui reçoit un dossier `folder` et qui renvoie un tuple formé du nom et de la taille du plus "lourd" fichier contenu dans le dossier (sous-dossiers compris). Par fichier le plus lourd, on entend celui dont la taille est la plus grande. Si plusieurs fichiers ont une taille maximale, vous pouvez renvoyer n'importe lequel parmi ceux-là.

Vous pouvez tester si une variable `x` est une liste en utilisant l'expression booléenne (`type(x) == list`).

Par exemple :

```
folder = ("Programmes", [
    ("Bloc-Notes", 123),
    ("Chrome", [
        ("Chrome", 206),
        ("Favoris", 28),
    ]),
    ("Calculatrice", 37),
    ("Jeux", [
        ("Démineur", 56),
        ("Réussite", 52)
```

```
    ]),
    ("MS Office", [
        ("Word", [
            ("Word", 198),
            ("Dictionnaire", 273)
        ]),
        ("Lanceur", 107),
        ("Excel", [
            ("Exemple_1", 174),
            ("Exemple_2", 66)
        ])
    ])

])

>>> biggest_file(folder)
('Dictionnaire', 273)
```

Solution.

```
def bigger_file(folder):
    """Le nom et la taille du fichier le plus lourd"""
    res = ("File not found", 0)       # Le résultat si folder vide
    for elem in folder[1]:            # folder[1] est la liste contenu qu'on veut parcourir
        if type(elem[1]) == list:     # Si elem[1] est une liste, elem est un sous-dossier
            elem = bigger_file(elem) # On retient le plus grand elem du sous-dossier
        if elem[1] > res[1]:          # Et quoiqu'il en soit, on retient le plus grand de tout le parcours
            res = elem
    return res
```