UNIVERSITÉ LIBRE DE BRUXELLES Juin 2014
École polytechnique de Bruxelles

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 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 ainsi que des commentaires pertinents.
- Vous pouvez ajouter des fonctions si cela vous semble nécessaire.
- Sauf mention contraire, vous ne pouvez utiliser aucune fonction de librairies (pas d'`import`).

Question 1 - Python (3 points)

Avec les éléments de Python vu au cours, décrivez en illustrant avec des petits exemples, huit types d'erreurs différents qu'un programme Python peut avoir. Expliquez les *bonnes pratiques* et les mécanismes pour éviter ou gérer chacune de ces erreurs.

Question 2 - Complexité (4 points)

Pour chacun des codes suivants, en supposant que `n` est un paramètre de type entier naturel strictement positif, donnez la complexité maximale en terme de $\mathcal{O}()$ des codes suivants :

1. ```
   t = []
   for i in range(n):
      t=t+[i]
   for i in t:
      j=0
      while j < i:
         print(t[j], end= ' ')
         j = j+1
      print('*')
   ```
2. ```
   i=1
   while i < n:
      print(i)
      i=2*i
   ```
3. ```
   i=2
   while i < n:
      print(i)
      i = i*i
   ```
4. Sachant que `prefixe` est initialement un string vide et `s` un string à n caractères :
   ```
   def Permutations(prefixe, s):
      if len(s) == 0:
         print(prefixe)
      else:
         for i in range(len(s)):
            Permutations(prefixe + s[i], s[:i]+s[i+1:])
   ```
5. Sachant que `graphe` est un dictionnaire de n noms de *personnes* sous forme de strings de longueur maximale de 10 caractères, où la valeur correspondante à chaque personne est une liste de ses amis (listes de noms de personnes), et `nom` est un nom de personne dans `graphe`, donnez la complexité moyenne et maximale de la fonction `connaissance_indirecte`.
   ```

```
def connaissance_indirecte(graphe,nom):
   s = set(nom)
   to_do = set(nom)
   while len(to_do) > 0:
      z = to_do.pop()
      for y in graphe[z]:
        if y not in s:
           s.add(y)
           to_do.add(y)
   return s - {nom}
```

Exemple :

```
graphe = { "Jean"       : ["Germaine"] ,
           "Germaine"   : ["Catherine"],
           "Catherine"  : ["Jean"] ,
           "Luc"        : [],
           "Michel"     : ["Luc"],
           "Bernadette" : ["Luc", "Michel", "Jules"],
           "Jules"      : ["Bernadette"] }
nom = "Bernadette"
```

Question 3 - Approximation de $cosh(x)$ (4 points)

La fonction $cosh(x)$ peut être définie par la série suivante :

$$cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + ... = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$$

On vous demande d'écrire une fonction `cosh(x)` qui reçoit un `x` et renvoie une approximation du cosinus hyperbolique correspondant.

Les calculs s'arrêteront quand la valeur absolue du dernier terme ajouté est inférieure à `EPS` ou lorsque le nombre de termes considérés atteint la borne `NMAX`. `EPS` et `NMAX` sont considérées comme des constantes définies globalement.

Question 4 – Opérations sur les fichiers (4 points)

On vous demande d'écrire une fonction `no_doubles(fin,fout)` qui élimine toutes les lignes qui apparaissent plus d'une fois dans un fichier texte donné.

La fonction doit prendre deux *noms* de fichier comme paramètres : le premier (`fin`) identifie le fichier à lire et à "nettoyer" ; le second (`fout`) donne le nom du fichier qui sera créé par la fonction et qui ne contiendra plus aucune ligne doublon.

Les lignes vides ne seront pas prises en compte.

Afin d'offrir une complexité acceptable à votre solution, envisagez d'utiliser un `set` ou un `dict`.

Exemple :

Fichier donné (“in.txt”)

```
Une valse à mille temps
Une valse à mille temps
Une valse a mis l'temps

De patienter vingt ans
Pour que tu aies vingt ans
Et pour que j'aie vingt ans

Une valse à mille temps
Une valse à mille temps
Une valse à mille temps

Offre seule aux amants
Trois cent trente-trois
fois l'temps
De bâtir un roman
```

→ **no_doubles**("in.txt","out.txt") →

Fichier résultat (“out.txt”)

```
Une valse à mille temps
Une valse a mis l'temps

De patienter vingt ans
Pour que tu aies vingt ans
Et pour que j'aie vingt ans


Offre seule aux amants
Trois cent trente-trois
fois l'temps
De bâtir un roman
```

Question 5 – Récursivité (5 points)

Une expression arithmétique peut être représentée de façon simplifiée comme suit :

- Une valeur numérique forme une expression. *Par exemple, 123 est une expression de valeur 123.*
- Une liste de la forme [`op`, `expr1`, `expr2`], dans laquelle `op` est l'un des 4 caractères '+', '-', '*', '/' et `expr1` et `expr2` sont des expressions, forme une expression de valeur `expr1 op expr2`. *Par exemple,* `['+', 5, 2]` *est une expression de valeur 5 + 2 = 7.*

Ecrivez une fonction `eval(expr)` *récursive* qui permet d'évaluer une expression `expr` donnée sous la forme décrite ci-dessus.

Remarque : Vous pouvez partir du principe que le paramètre `expr` reçu par la fonction représente une expression *valide* ; vous ne devez pas le vérifier.

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

Exemples :

```
>>> print(eval( 123 ))
123

>>> print(eval( ['+', 5, 2] ))
7

>>> print(eval( ['*', ['/', 9, 3], ['+', 5, 2]] ))    #  (9/3) * (5+2) = 21
21
```