IC Devoir 4 - Mercredi 24 avril de 17h30 à 18h30

Exercice 1 Considérant que

$$1-\cos(x) \underset{0}{\sim} \frac{x^2}{2} \quad \text{et} \quad \cos(2x) = 2\cos(x)^2 - 1$$

proposer une fonction récursive, `c(x)` qui donne une approximation de $\cos(x)$.

Exercice 2

1. Donner la représentation de 115 en base 2. Expliquer la démarche et détailler les étapes.
2. Considérons la représentation binaire des nombres entiers relatifs sur 5 bits par la **méthode du complément à 2**.

a) Donner l'intervalle de $\mathbb{Z}$ contenant les nombres pouvant être représentés.

b) Donner la représentation binaire de 13 et de -9.

Exercice 3 Considérons la liste `L` : `[2,6,1,5,4,2,4]`.

1. Décrire le tri par insertion.
2. Détailler les étapes du tri par insertion sur la liste `L`.
3. Décrire le tri par sélection.
4. Détailler les étapes du tri par sélection sur la liste `L`.

Tri fusion et ordre lexicographique

On définit une relation d'ordre dans l'ensemble E_n des listes de n entiers par :

$a \leq b$ si $a = b$ ou bien s'il existe $k \in [\![0; n-1]\!]$ tel que

- $\forall j \in [\![0, k-1]\!], \quad a_j = b_j$
- et $a_k < b_k$

C'est l'ordre qui est utilisé pour classer les mots dans un dictionnaire.

Exercice 4 *Tri fusion*

1. Compléter la fonction `inf_ou_egal(a,b)` qui compare deux listes `a` et `b` dans E_n et qui renvoie `True` si $a \leq b$ et `False` sinon.

Tri fusion suivant l'ordre lexicographique

```
1 def inf_ou_egal(a,b):
2     for k in range(len(a)):
3         if a[k]<b[k]:
4             return ...
5         elif ...
6             return False
7     return ...
```

2. Soient deux listes `L1` et `L2` supposées triées par ordre croissant avec la relation définie ci-avant et dont les éléments sont des éléments de E_n.
Écrire une fonction `fusion(L1,L2)` qui fusionne ces deux listes en renvoyant une liste triée par ordre croissant réunissant `L1` et `L2`.
Cette fonction utilisera la fonction `inf_ou_egal`.
3. Ecrire une fonction `tri_fusion(L)` qui trie par ordre croissant la liste `L` en utilisant la méthode du tri par fusion.
Cette fonction utilisera la fonction fusion.

Tri rapide

Méthode

➜ Le tri rapide consiste à :

- si la liste contient au plus un élément, alors on retourne la liste ; sinon, on considère le premier élément ;
- on crée deux sous-listes, une contenant les éléments qui lui sont inférieurs et l'autre pour les autres éléments (ceux qui lui sont strictement supérieurs) ;
- on applique le tri aux deux sous-listes obtenues et on concatène le résultat.

Exercice 5 **Prog**

1. Compléter la suite des appels récursifs et les étapes de la remontée du **tri rapide** sur l'exemple suivant :

Profondeur	Liste
0	`[5,4,8,4,3,7,2,6,1,9]`
1	...
2	...
⋮	⋮
1	...
0	`[1,2,3,4,4,5,6,7,8,9]`

2. Écrire un script récursif de ce tri : `tri_rapide(L)`.
3. Proposer une version itérative (non récursive) de ce tri.
Vous pourrez vous inspirer/compléter la version suivante qui travaille directement sur la liste `L` en introduisant une liste `T` contenant la position des sous-listes non encore triée.
Dans l'exemple `L = [5,4,8,4,3,7,2,6,1,9]`

- `T` est initialisée à `[(0,10)]` car toute la liste `L` est à trier.
- L'instruction de la ligne 7 retire le dernier élément de `T` (qui est maintenant vide)
- Après constitution des deux sous-listes `I=[4,4,3,2,1]` et `S=[8,7,6,9]` on ajoute dans `T` les couples associés, respectivement `(0,5)` et `(6,10)`, afin que le travaille puisse continuer.
- Tant qu'il y a des couples dans `T`, on applique la découpe sur les sous listes associées.
- Lorsque `T` est vide, la liste `L` est triée.

```
1  def tri_rapide_ite(L):
2      if len(L)>1:
3          T=[(0,len(L))]
4      else:
5          T=...
6      while len(T)>0:
7          i,j=T.pop()
8          I,S=...
9          for e in L[i+1:j]:
10             if ...:
11                 I.append(e)
12             else:
13                 ...
14         L[i:j]=I+[L[i]]+S
15         if len(I)>1:
16             T.append(...)
17         if len(S)>1:
18             T.append(...)
19     return L
```


Lycée Saint-Exupery – maffre.saintexupery@gmail.com – MPSI

IPT Devoir 4- Proposition de solutions

Solution 1
On rappelle que le 0 des nombres réels en PYTHON est de l'ordre 10^{-15}. Or la condition d'arrêt nécessite le calcul de x^2, donc si $|x| \leq 10^{-8}$ alors $x^2 \leq 10^{-15}$ et la valeur retournée est 1 qui est un point fixe de l'autre relation

$$1 = 2 \times 1^2 - 1$$

donc la valeur finale retournée serait toujours 1.

Attention ! La condition d'arrêt doit être moins précise que `abs(x)<1e-8`.

```
def c(x):
    if abs(x)<1e-4:
        return 1-x**2/2
    else:
        return 2*c(x/2)**2-1
```

Solution 2 1. La représentation d'un nombre en base b consiste à effectuer des divisions successives d'un nombre par b jusqu'à obtenir un quotient nul. La représentant est donnée la collection des restes dans l'ordre inverse de l'obtention.
Écriture de 115 en base 2 :

115	2						
1	57	2					
↓	1	28	2				
a_0	↓	0	14	2			
	a_1	↓	0	7	2		
		a_2	↓	1	3	2	
			a_3	↓	1	1	1
				a_4	↓	1	0
					a_5	↓	
						a_6	

Alors $115 = \overline{1110011}^2$.
En effet, $2^6 + 2^5 + 2^4 + 2^1 + 2^0 = 64 + 32 + 16 + 2 + 1 = 115$.
2. Considérons la représentation binaire des nombres entiers relatifs sur 5 bits par la **méthode du complément à 2**.
a) L'intervalle des nombres représentés et

$$[\![-2^4, 2^4 - 1]\!] = [\![-16, 15]\!].$$

b) $13 = 8 + 4 + 1 = \overline{01101}^2$ et -9 est représenté par

$$-9 + 2^5 = 23 = 16 + 4 + 2 + 1 = \overline{10111}^2.$$

Solution 3 1. Le tri par **insertion** consiste à : **Méthode**

- parcourir la liste de gauche à droite ;
- à chaque étape, l'élément considéré, est classé parmi les éléments qui le précèdent (et donc qui sont déjà ordonnés).

2. Tri appliqué à `L` :

i	L
X	[2,6,1,5,4,2,4]
1	[2,6,1,5,4,2,4]
2	[1,2,6,5,4,2,4]
3	[1,2,5,6,4,2,4]
4	[1,2,4,5,6,2,4]
5	[1,2,2,4,5,6,4]
6	[1,2,2,4,4,5,6]

3. Le tri par sélection consiste à : **Méthode**

- on cherche le minimum de la liste et, par échange, on le place à la première place
- puis on place le minimum des éléments restants à la second place
- on recommence, ainsi de suite, jusqu'à ce que les éléments restants soient réduits à un élément.

4. Tri appliqué à `L` :

i	min(L[i:])	L
X	X	[2,6,1,5,4,2,4]
0	1	[1,2,6,5,4,2,4]
1	2	[1,2,6,5,4,2,4]
2	2	[1,2,2,6,5,4,4]
3	4	[1,2,2,4,6,5,4]
4	4	[1,2,2,4,4,6,5]
5	5	[1,2,2,4,4,5,6]

Lycée Saint-Exupery – maffre.saintexupery@gmail.com – MPSI

Solution 4 *Tri fusion suivant l'ordre lexicographique*

1.

Tri fusion suivant l'ordre lexicographique

```
def inf_ou_egal(a,b):
    for k in range(len(a)):
        if a[k]<b[k]:
            return True
        elif a[k]>b[k]:
            return False
    return True
```

Exemple :

```
--> inf_ou_egal([1,2,3],[1,2,4])
True
--> inf_ou_egal([1,2,3],[1,2,0])
False
```

2. Fusion de deux listes triées :

Fusion récursive de deux listes déjà triées

```
def fusion_rec(L1,L2):
    if len(L1)==0:
        return L2
    if len(L2)==0:
        return L1
    if inf_ou_egal(L1[0],L2[0]):
        return L1[:1]+fusion_rec(L1[1:],L2)
    else:
        return L2[:1]+fusion_rec(L1,L2[1:])
```

Exemple :

```
-->fusion_rec([[1,2],[1,5]],[[0,2],[1,4]])
[[0, 2], [1, 2], [1, 4], [1, 5]]
```

3. Méthode du tri par fusion :

Tri par fusion

```
def tri_fusion(L):
    L,n=list(L),len(L)
    if n==1:
        return L
    else:
        L1=tri_fusion(L[:n//2])
            L2=tri_fusion(L[n//2:])
        return fusion_rec(L1,L2)
```

Exemple :

```
--> tri_fusion([[1,2],[2,1],[1,0],[1,1]])
[[1, 0], [1, 1], [1, 2], [2, 1]]
```

Solution 5 Tri rapide

1. Suite des appels récursifs et les étapes de la remontée :

Profondeur	Liste
0	[5,4,8,4,3,7,2,6,1,9]
1	[4,4,3,2,1],[5],[8,7,6,9]
2	[4,3,2,1],[4],[],[5],[7,6],[8],[9]
3	[3,2,1],[4],[],[4],[],[5],[6],[7],[],[8],[9]
4	[2,1],[3],[],[4],[],[4],[],[5],[6],[7],[],[8],[9]
5	[1],[2],[],[3],[],[4],[],[4],[],[5],[6],[7],[],[8],[9]
4	[1,2],[3],[],[4],[],[4],[],[5],[6],[7],[],[8],[9]
3	[1,2,3],[4],[],[4],[],[5],[6],[7],[],[8],[9]
2	[1,2,3,4],[4],[],[5],[6,7],[8],[9]
1	[1,2,3,4,4],[5],[6,7,8,9]
0	[1,2,3,4,4,5,6,7,8,9]

2. Script basique :

tri_rapide.py

```
def tri_rapide(L):
    if len(L)<2:
        return(L)
    I,S=[],[]
    for e in L[1:]:
        if e<=L[0]:
            I.append(e)
        else:
            S.append(e)
    return tri_rapide(I)+[L[0]]+tri_rapide(S)
```

3. Version itérative :

```
1  def tri_rapide_ite(L):
2      if len(L)>1:
3          T=[(0,len(L))]
4      else:
5          T=[]
6      while len(T)>0:
7          i,j=T.pop()
8          I,S=[],[]
9          for e in L[i+1:j]:
10             if e<=L[i]:
11                 I.append(e)
12             else:
13                 S.append(e)
14         L[i:j]=I+[L[i]]+S
15         if len(I)>1:
16             T.append((i,i+len(I)))
17         if len(S)>1:
18             T.append((len(I)+i+1,j))
19     return L
```