EA4 – Éléments d'algorithmique
TP nº 1

Exercice 1 : Expressions en Python

Pour chacune des expressions suivantes, expliquer le résultat retourné par Python.

```
5 + 3 * 10
13 / 4
13.0 / 4
13 // 4
13 % 4
3 ** 4
i = 5
i
i = i + 4
i
j = 0
k
bonjour = salut
bonjour = "salut"
bonjour
```

```
i < j
p = (j <= 3)
p
p and (i == 9)
p or (j <= 3)
not p
i <> j
i != j
if j < 10:
       print ("j < 10")
else:
       print ("j >= 10")
```

```
#list seulement si Python 3
range(5)
list(range(5))
range(1, 5)
list(range(1, 5))
range(-3, 4)
list(range(-3, 4))
range(1, 21, 2)
list(range(1, 21, 2))
for i in range(10):
       print(i)
for i in range(2, 11):
       print(i)
```

Exercice 2 : Listes

1. Écrire un programme qui affiche la liste des 10 premiers carrés.
2. Écrire une fonction itérative qui retourne la liste des n premiers carrés. La fonction débutera par la création d'une liste de n éléments.
3. En écrire une version récursive construisant la liste des carrés progressivement.
4. Écrire une fonction qui retourne la liste des n premiers carrés qui sont aussi des cubes.

Exercice 3 : Fibonacci reloaded

1. On rappelle les fonctions `fibo_1`, `fibo_2` et `fibo_3` vues en cours :

```
def fibo_1(n) :
  if n < 0 : return 0
  if n < 2 : return 1
  return fibo_1(n - 1) + fibo_1(n - 2)

def fibo_2(n) :
  if n < 0 : return 0
  liste = [1, 1]
  for i in range(1, n) :
    liste.append(liste[i - 1] + liste[i])
  return liste[n]
```

```
def fibo_3(n) :
  if n < 0 : return 0
  previous, last = 1, 1
  for i in range(1, n) :
    previous, last = last, previous + last
  return last
```

Modifier ces fonctions en ajoutant un compteur qui comptabilisera le nombre d'opérations élémentaires faites lors d'un appel à chacune d'entre elles. Tester ces fonctions sur de grands nombres. Ces mesures reflètent-elles les conclusions du cours ?

2. À partir des fonctions suivantes qui calculent respectivement le produit de 2 matrices de dimension 2 et la puissance n-ième d'une matrice, écrire une fonction `fibo_4` en utilisant la relation

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

Comparer son efficacité avec les versions précédentes.

```
def produit_matrice_2_2 (M1, M2) :
  res = [ [0, 0], [0, 0] ]
  res[0][0] = M1[0][0] * M2[0][0] + M1[0][1] * M2[1][0]
  res[1][0] = M1[1][0] * M2[0][0] + M1[1][1] * M2[1][0]
  res[0][1] = M1[0][0] * M2[0][1] + M1[0][1] * M2[1][1]
  res[1][1] = M1[1][0] * M2[0][1] + M1[1][1] * M2[1][1]
  return res

def puissance_matrice_2_2 (M, n) :
  if n == 0 : return [ [1, 0], [0, 1] ]
  if n == 1 : return M
  tmp = puissance_matrice_2_2(M, n // 2)
  carre = produit_matrice_2_2(tmp, tmp)
  if n % 2 == 1 : return produit_matrice_2_2(M, carre)
  else : return carre
```

Exercice 4 : Cœfficients binomiaux

1. Écrire une fonction qui calcule les cœfficients binomiaux en utilisant la relation de récurrence $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ et en affichant les appels intermédiaires. Comment améliorer cette fonction en mémorisant les valeurs déjà calculées dans une matrice ?
2. En écrire une version itérative utilisant la formule directe de calcul des cœfficients $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ et en affichant les valeurs intermédiaires calculées. Comment améliorer cette fonction en minimisant le nombre de multiplications à effectuer ?
3. À l'aide de compteurs, comparer l'efficacité de ces fonctions.