DS n° 03 – DS03

Les codes en python doivent être commentés et les indentations dans le code doivent être visibles.

I Exercice – Traduction et compréhension d'un algorithme

1. À l'aide des opérateurs `//` et `%` écrire en Python :
   — une fonction `ent(x)` renvoyant la partie entière d'un nombre réel x positif ;
   — une fonction `par(n)` renvoyant 0 si un entier naturel n est pair et 1 sinon.
2. On considère l'algorithme suivant, qui s'applique à deux entiers naturels i et j et utilise une variable auxiliaire t qui porte le résultat final :

Algorithme 1 : Algorithme à implanter en Python.

```
   Données : i et j deux entiers naturels
   Résultat : t
 1 t ← 0
 2 tant que i ≥ 1 faire
 3 |   si par(i) = 1 alors
 4 |   |   t ← t + j
 5 |   sinon
 6 |   |   Ne rien faire
 7 |   fin
 8 |   j ← j × 2
 9 |   i ← ent(i/2)
10 fin
```

   Écrire en Python une fonction `mystere(i,j)` traduisant l'algorithme. On considérera les fonctions `ent(n)` et `par(n)` connues et on pourra les utiliser directement dans le code proposé.
3. On exécute `mystere(8,9)`. Proposer sous forme de tableau à trois colonnes les valeurs prises pas à pas par les trois entiers (i, j, t) *à la fin* de chacune des itérations, chaque ligne représentant une itération de la boucle itérative utilisée dans le code.
4. Même question pour `mystere(9,8)`.
5. Dans le cas général, quel est le résultat final retourné par l'algorithme ?
6. On se propose de prouver le résultat précédent en étudiant le terme $i \times j + t$. Pour cela :
   (a) prouver que ce terme est égal à $i \times j$ au début de la première itération ;
   (b) prouver que la propriété précédente reste vraie à la fin d'une itération quelconque (invariance) ;
   (c) prouver que le résultat final de l'algorithme est bien le résultat attendu.

II Question de cours – Méthode des rectangles

On calcule l'intégrale numérique en réalisant une somme de surfaces de rectangles. Le domaine d'intégration est découpé en intervalles et on fait comme si la fonction restait constante sur chaque intervalle avec comme valeur sa valeur au premier point de l'intervalle.

On souhaite approximer la valeur de $\int_0^{\frac{3\pi}{2}} \cos(x)\mathrm{d}x$, grâce à la méthode précédente, en utilisant 2000 valeurs de la fonction pour des valeurs d'angle régulièrement espacées.
Proposer un code en Python calculant la valeur approchée recherchée et l'affichant à l'écran.

III Exercice – Travaux pratiques

1. On considère la suite définie par récurrence comme suit : $\begin{cases} u_0 = 0 \\ u_{n+1} = 2u_n + 3 \end{cases}$

   Écrire en Python un programme qui calcule les dix premiers termes de la suite et les affiche tous à l'écran.

2. On considère maintenant la suite de Fibonacci définie par :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \forall n \in \mathbb{N}, \quad F_{n+2} = F_{n+1} + F_n \end{cases}$$

   Écrire un programme qui permet de calculer les dix premiers termes de la suite $(F_n)_{n\in\mathbb{N}}$.

Correction DS n° 03 – DS03

I Exercice – Traduction et compréhension d'un algorithme

1. Utilisation des opérateurs de la division euclidienne.

Fonction par

```
def par(n):
    return n%2!=0 # renvoie la réponse booléenne à la question:
                  # "n n'est-il pas pair" ?
```

Cette fonction renvoie `True` ou `False` qui, dans des comparaisons, sont interprétés respectivement comme 1 ou 0. On peut préférer répondre plus explicitement à la question, avec cette fonction :

```
def par(n):
    if n%2==0:   # si n est pair
        return 0 # Équivalent à False
    else:        # sinon
        return 1 # Équivalent à True
```

Fonction ent

```
def ent(x):
    return x//1 # renvoie le quotient de la division
                # euclidienne de x par 1
                # c'est-à-dire sa partie entière
```

2.
```
def mystere(i,j):
    t=0              # initialisation de t
    while i >=1:     # sortie de la boucle pour i < 1
        if par(i)==1:   # si i est impair
            t=t+j          # on ajoute j à t
                           # pour toutes les itérations :
        j=j*2              # on multiplie j par 2
        i=ent(i/2)         # on met à jour i avec la partie
                           # entière de sa moitié
#       print(i,j,t)    # à décommenter pour afficher les valeurs
    return t         # on retourne le résultat final

print(mystere(9,8))
```

3. Pour $i = 8$ et $j = 9$, valeurs *à la fin* de chaque itération :

itération k	i_k	j_k	t_k
0	4	18	0
1	2	36	0
2	1	72	0
3	0	144	72

4. Pour $i = 9$ et $j = 8$, valeurs *à la fin* de chaque itération :

itération k	i_k	j_k	t_k
0	4	16	8
1	2	32	8
2	1	64	8
3	0	128	72

5. Dans un cas général, le résultat final retourné par l'algorithme est le produit de deux entiers : $i \times j$.
6. (a) Au début de la première itération, $i_0 = i$, $j_0 = j$ et $t_0 = 0$, donc $i_0 \times j_0 + t_0 = i \times j$.

   (b) Supposons la propriété précédente vraie au début de la k^{e} itération : $i_k \times j_k + t_k = i \times j$. Il y a alors deux possibilités :
   — si i_k est pair, alors $t_{k+1} = t_k$, $j_{k+1} = 2j_k$, et $i_{k+1} = \frac{i_k}{2}$, d'où :

   $$i_{k+1} \times j_{k+1} + t_{k+1} = \frac{i_k}{2} \times 2j_k + t_k = i_k \times j_k + t_k = i \times j$$

   — si i est impair, alors $t_{k+1} = t_k + j_k$, $j_{k+1} = 2j_k$ et $i_{k+1} = \frac{i_k - 1}{2}$, d'où :

   $$i_{k+1} \times j_{k+1} + t_{k+1} = \frac{i_k - 1}{2} \times 2j_k + t_k + j_k = i_k \times j_k + t_k = i \times j$$

   (c) Le résultat final est la dernière valeur de t, c'est-à-dire la valeur de t quand la boucle « tant que » est terminée. Cette boucle est terminée à l'itération f telle que lorsque $i_f < 1$ donc telle que $i_f = 0$ puisque i_f est un entier naturel. On a donc alors $i \times j = i_f \times j_f + t_f = t_f$. Le résultat final est donc bien le produit de i par j.