

Exercices

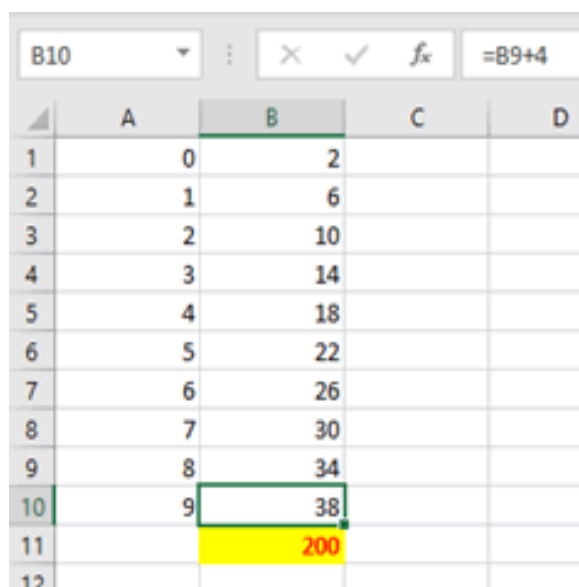
1.1 Exercice 0 :

1.1.1 suites arithmétiques

Une suite arithmétique (u_n) de raison r peut être définie par la formule de récurrence :

$$u_{n+1} = u_n + r$$

Le tableau suivant montre 2 listes calculées à partir de suites arithmétiques.



	A	B	C	D
1	0	2		
2	1	6		
3	2	10		
4	3	14		
5	4	18		
6	5	22		
7	6	26		
8	7	30		
9	8	34		
10	9	38		
11		200		
12				

FIGURE 1 – tableau Excel

1. Proposer une formule générale pour le calcul de la somme des termes d'une suite arithmétique.
2. Calculer la somme des termes de la colonne de gauche
3. Calculer la somme des termes de la colonne de droite
4. Soit $u(n)$ la fonction python qui calcule et retourne la valeur de u au rang n . Ecrire une fonction `somme_arithmetique` en python qui calcule la somme des n premiers termes de la suite $u(n)$ et retourne la valeur calculée. Cette fonction utilisera une boucle bornée, et non la formule proposée au 1.

1.1.2 Fonction logarithmique

Le logarithme en base a d'un nombre x strictement positif peut être calculé à partir de :

$$\log_a(x) = \frac{\log(x)}{\log(a)}$$

En informatique, on utilise préférentiellement la fonction logarithmique en base 2.

1 Calculer les logarithmes en base 2 de 8, 16, 32, 64. Que remarquez-vous? 2. Combien de fois successives faut-il diviser 8 par 2 pour arriver à 1? 3. Même question pour 16 puis 32. Conclure : Exprimer d'une autre manière ce que signifie le logarithme en base 2 d'un nombre.

1.2 Exercice 1

```

1 def multiplie1(b,n):
2     L=[]
3     for i in range(n):
4         L.append(b*i)
5     return L

```

Le programme exécute n fois la ligne 4. Le nombre d'opérations significatives effectuée est $T(n) = 2 \times n$:
On aura pour multiplie1 une classe de complexité algorithmique

$$O(n)$$

.

Si on ajoute des lignes dans la boucle `for`, pour faire par exemple :

```

1 def multiplie2(b,n):
2     L=[]
3     for i in range(n):
4         y = b * i
5         L.append(y)
6     return L

```

Le nombre d'opérations significatives effectuée par multiplie2 est $T(n) = 3 \times n$.

C'est deux fois plus que multiplie1. Or cette différence ne vient que d'une différence des **détails d'implémentation** du même algorithme, et ne doit pas être considérée pour le calcul de la complexité.

On aura aussi pour multiplie2 une classe de complexité algorithmique

$$O(n)$$

.

Enfin, ce même algorithme peut être implémenté avec une boucle non bornée :

```

1 def multiplie3(b,n):
2     L=[]
3     i = n - 1
4     while i >= 0 :
5         y = b * i
6         L.append(y)
7         i -= 1
8     return L

```

1. Pour les fonctions multiplie1 et multiplie2 : Énoncer un ensemble de règles pour déterminer $T(n)$.
2. Pour les fonctions multiplie1 et multiplie2 : Énoncer une règle pour évaluer la complexité algorithmique $O(g(n))$ à partir de $T(n)$.
3. Déterminer $T(n)$ pour multiplie3. Vérifier que l'on obtient aussi une classe de complexité $O(n)$ pour multiplie3.