



## I - Suites

**Une suite est une famille d'éléments indexés par les entiers naturels. Chaque élément de la suite se nomme un terme.** Il y a un premier élément, puis un deuxième, un troisième...

Exemple : 10, 7, 4, 1, -2, -5

- Le terme  $u_0$  est le \_\_\_\_
- Le terme  $u_1$  est le \_\_\_\_
- Le terme  $u_2$  est le \_\_\_\_

**SUITE ARITHMETIQUE** : suite dans laquelle un terme  $u_{n+1}$  se calcule **en additionnant** le terme précédent  $u_n$  et une valeur appelée la raison  $r$ .

Exemple : 5, 15, 25, 35... est une suite arithmétique

- de **valeur initiale**  $u_0 = 5$  (on part de 5)
- de **raison**  $r = 10$  (on rajoute 10 à chaque fois)

A partir de ces deux informations, on peut obtenir les valeurs successives des éléments de la suite.

- $u_0 = 5$
- $u_1 = 15$  car  $u_1 = u_0 + 10$
- $u_2 = 25$  car  $u_2 = u_1 + 10$
- $u_3 = 35$  car  $u_3 = u_2 + 10$
- $u_4 = 45$  car  $u_4 = u_3 + 10$

**Formule 1** : calcul du successeur :  $u_{N+1} = u_N + r$

On notera que sur l'exemple, la raison  $r$  est :  $r = +10$

**Formule 2** : calcul général  $u_N = u_0 + r \cdot n$

On notera que sur l'exemple,  $u_0 = 5$  et  $r = +10$

### Evolution (en math)

Si la **raison  $r$  est positive non nulle**, on aura une valeur de plus en plus grande : **la suite est croissante**.

Si la **raison  $r$  est négative**, on aura une valeur de plus en plus petite : **la suite est décroissante**.

### Attention aux flottants en informatique

Si  $r$  est inférieure à la plus petite valeur représentable avec des flottants, il est possible que la raison soit en réalité nulle : la suite est alors constante !

### Théorème général

**Toute suite d'entiers positifs strictement décroissante ne peut prendre qu'un nombre fini de valeurs.**

**SUITE GEOMETRIQUE** : suite dans laquelle un terme  $v_{n+1}$  se calcule en **multipliant** le terme précédent  $v_n$  et une valeur appelée la raison  $q$ .

**Exemple** : 5, 10, 20, 40... est une suite géométrique :

- de **valeur initiale**  $v_0 = 5$  (on part de 5)
- de **raison**  $q = 2$  (on multiplie par 2 à chaque fois qu'on passe au successeur)

A partir de ces deux informations, on peut obtenir les valeurs successives des éléments de la suite.

- $v_0 = 5$
- $v_1 = 10$  car  $v_1 = v_0 * 2$
- $v_2 = 20$  car  $v_2 = v_1 * 2$
- $v_3 = 40$  car  $v_3 = v_2 * 2$
- $v_4 = 80$  car  $v_4 = v_3 * 2$

**Formule 1** : calcul du successeur

$$v_{N+1} = v_N * q$$

On notera que sur l'exemple, la raison  $q$  est :  $q =$

**Formule 2** : calcul général

$$v_n = v_0 * q^n$$

On notera que sur l'exemple,  $v_0 =$

et  $q =$

### Evolution (en math)

Si la **raison est strictement supérieure à 1**, la **suite géométrique est croissante**. Avec une raison de 10 : 5, 50, 500, 5000, 50000...

Si la **raison est strictement inférieure 1**, la **suite géométrique est décroissante**. Avec une raison de 0.1 : 5, 0.5, 0.05, 0.005, 0.0005...

### Attention aux flottants en informatique

Avec une raison de 0.5 et des FLOTTANTS : 10, 5, 2, 1, 0.5, 0.25, 0.125, 0.0625... On n'attendra jamais 0. Voilà pourquoi on préfère les entiers.

## II - VARIANT : Preuve de terminaison ou preuve d'arrêt

La **preuve de terminaison** d'un algorithme permet d'affirmer qu'il s'arrête sur toutes les entrées valides qu'on lui fournit.

**Terminaison d'une boucle** : si les propositions A et B sont vraies alors la boucle se termine.

Proposition A : la boucle peut s'écrire **TQ VARIANT > 0**

Proposition B : le **VARIANT** est une suite d'entiers strictement **décroissante**.

**1. AVANT** : On cherche les valeurs des variables impliquées dans la boucle AVANT le premier tour.

**2. PENDANT** : On cherche l'évolution des variables à chaque tour de boucle.

**3. CONDITION** : On utilise les points 1 et 2 pour écrire la condition sous la forme **TQ VARIANT > 0**

**4. CONCLUSION** : On étudie le variant  $u_n$  pour voir s'il s'agit bien d'une suite d'entiers décroissante.

**Exercice de cours** : Montrer la terminaison de l'algorithme suivant en

1  $x = 3$

montrant l'existence d'une condition du type **TQ VARIANT > 0** pour laquelle

2 **while**  $x < 100$  :

le variant est bien une **suite d'entiers positifs strictement décroissante**. 3  $x = x + 10$

```
1 def exercice(seuil) :
2     '''Fonction-exercice qui ne sert à rien
3     :: param seuil(int) :: un entier positif
4     :: return (int)     :: un résultat
5     '''
6     x = 0
7     while 5*x < seuil :
8         x = x + 1
9
10    return x
```

**Exercice 05** : Montrer la terminaison de l'algorithme si on remplace la ligne 8 par ceci. 8  $x = x - 1$

**Exercice 06** : Montrer la terminaison de l'algorithme de recherche linéaire.

```
1 def rechercher(t:list, x:int):
4     i = 0
5     longueur = len(t)
6     while i < longueur and t[i] != x: # début de boucle si on arrive ici
7         i = i + 1
8     if i < longueur:
9         reponse = i
10    else:
11        reponse = -1
12    return reponse
```

### III - INVARIANT : Preuve de correction

**3.1 TESTER N'EST PAS DEMONTRER !** Sauf si le nombre de cas est fini et qu'on les teste tous.

**3.2 Preuve de correction** : La preuve de correction d'un algorithme permet d'affirmer

- qu'il fournit toujours une **réponse correcte**
- sur toutes les **entrées valides** qu'on lui fournit.

Pour faire la preuve de correction d'un algorithme, il faut trouver un **INVARIANT** : une propriété P vérifiable qui reste vraie lors des différentes étapes de l'algorithme.

Un **INVARIANT DE BOUCLE** est une propriété qui est vraie avant la première itération et reste vraie à chaque itération d'une boucle. Elle est donc vraie après avoir effectué le dernier tour de boucle.

**Démonstration en 3 phases** : Initialisation - Conservation - Terminaison

**Phase d'initialisation** :  $P_0$  est VRAI.

On doit prouver que l'**INVARIANT** est VRAI avant le premier tour de boucle.

**Phase de conservation** :  $P_k \Rightarrow P_{k+1}$

Hypothèse : on fait l'hypothèse que l'invariant  $P_k$  est vraie après un certain nombre de tours. On doit exécuter le nouveau tour de boucle et montrer que l'invariant  $P_{k+1}$  est toujours vraie.

On peut donc écrire  $P_k \Rightarrow P_{k+1}$

**Phase de terminaison**

On doit prouver qu'après le dernier tour, on obtient la situation voulue. Ni trop tôt, ni trop tard. C'est pour cela qu'on nomme cette partie la phase de terminaison, à ne pas confondre avec la preuve de terminaison / d'arrêt :

$$P_0 \Rightarrow P_1 \Rightarrow P_2 \Rightarrow P_3 \dots \Rightarrow P_{\text{final}}$$

### IV - Coût (et complexité)

#### Définition

Le **coût d'un algorithme** exprime la façon dont l'algorithme réagit globalement à l'augmentation du nombre  $n$  de données à traiter.

On évalue souvent le coût dans le pire des cas car il borne le comportement de l'algorithme

- de type <b>constant</b> ( en 1)	FACILE
- de type <b>logarithmique</b> (en $\log n$ )	
- de type <b>linéaire</b> (en $n$ )	
- de type <b>quadratique</b> (en $n^2$ )	
- de type <b>polynomial</b> (en $n^k$ )	
- de type <b>exponentiel</b> (en $2^n$ )	DIFFICILE
- de type <b>factoriel</b> (en $n!$ )	