PREMIÈRE / COURS

Algorithmique et programmation en Python

Astrolab Cours · astrolabcours.fr

01 Le problème

Une population de 1 000 bactéries double toutes les heures. Au bout de combien d'heures dépasse-t-elle un million ? On pourrait calculer à la main, heure après heure. Mais s'il faut recommencer avec 10 000 bactéries, ou un milliard comme seuil, c'est long.

Un **algorithme** fait ce travail pour nous : on écrit une fois la méthode, et l'ordinateur la répète autant qu'il faut.

01 L'idée : la recette de cuisine

Un algorithme, c'est une **recette** : une liste d'instructions précises, exécutées dans l'ordre.

- Les **variables**, ce sont les bols : chacun porte une étiquette (son nom) et contient un ingrédient (sa valeur). On peut vider un bol et le remplir avec autre chose.
- Les **conditions**, ce sont les « si la pâte est trop épaisse, ajoute du lait ».
- Les **boucles**, ce sont les « répète 3 fois » ou « remue **tant que** la sauce n'a pas épaissi ».
- Les **fonctions**, ce sont les recettes de base qu'on réutilise : « prépare une pâte brisée ».

01 Les variables et l'affectation

En Python, le signe `=` ne veut pas dire « égal » : il veut dire « range dans le bol ».

```
a = 5          # le bol a contient 5
a = a + 3      # a contient maintenant 8
print(a)       # affiche 8
```

L'instruction `a = a + 3` n'a aucun sens en maths, mais elle en a un en Python : on calcule à droite, puis on range à gauche.

Les types courants : les entiers (`int`), les décimaux (`float` , avec un **point** : `2.5`), les textes (`str`) et les booléens (`True` , `False`).

Les conditions

```
if note >= 10:
    print("admis")
else:
    print("refusé")
```

© Astrolab Cours · astrolabcours.fr/maths/premiere/algorithmique/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

Les comparaisons s'écrivent `==` (égal), `!=` (différent), `<`, `<=`, `>`, `>=`. L'**indentation** (le décalage vers la droite) est obligatoire : elle dit quelles lignes sont dans le `if`.

La boucle for : répéter un nombre connu de fois

`for k in range(n):` répète le bloc indenté n fois, avec `k` qui vaut successivement $0, 1, \ldots, n-1$.

Exemple : calculer u_{10} pour la suite $u_0 = 2$ et $u_{n+1} = 3u_n - 1$.

```
u = 2
for k in range(10):
    u = 3 * u - 1
print(u)
```

La boucle fait 10 tours : au premier, `u` devient u_1 ; au dixième, u_{10}.

La boucle while : répéter tant qu'une condition est vraie

`while condition:` répète le bloc indenté **tant que** la condition est vraie. On l'utilise quand on ne sait pas à l'avance combien de tours il faudra.

Retour aux bactéries, c'est un **algorithme de seuil** :

```
p = 1000
n = 0
while p <= 1000000:
    p = 2 * p
    n = n + 1
print(n)
```

On obtient 10 heures : $1\,000 \times 2^{10} = 1\,024\,000$. Attention à bien faire avancer la condition, sinon la boucle ne s'arrête jamais.

Les fonctions

```
def f(x):
    return x**2 - 3*x + 1

print(f(4))    # affiche 5
```

`def` crée une fonction, `return` renvoie le résultat. En Python, la puissance s'écrit `**`, pas `^`.

Les listes

Une liste range plusieurs valeurs dans un même bol, numérotées à partir de **0**.

```
L = [3, 8, 1]
L.append(5)       # L devient [3, 8, 1, 5]
```

© Astrolab Cours · astrolabcours.fr/maths/premiere/algorithmique/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits

```
print(L[0])       # affiche 3, le premier élément
print(len(L))     # affiche 4, le nombre d'éléments
```

On peut construire une liste en compréhension : `[k**2 for k in range(5)]` donne `[0, 1, 4, 9, 16]`.

© Astrolab Cours · astrolabcours.fr/maths/premiere/algorithmique/ · Tous droits réservés : usage personnel uniquement, republication et usage commercial interdits