

Algorithmique & Python

Cours détaillé avec codes, en français, et l'explication en arabe

ملخص الخوارزميات والبرمجة - الثلاثي الأول - شعبة علوم الإعلامية

Chapitre	L'idée en une phrase
1. Sous-programmes	Découper un problème en fonctions et procédures réutilisables.
2. Tableaux et matrices	Stocker plusieurs valeurs du même type et les parcourir avec des boucles.
3. Enregistrements et fichiers	Regrouper des champs différents et conserver les données sur le disque.
4. Récursivité	Un sous-programme qui s'appelle lui-même, avec une condition d'arrêt.
5. Tris et recherche dichotomique	Ranger un tableau, puis le fouiller en divisant par deux à chaque étape.

LA DÉMARCHE POUR RÉSOUDRE UN EXERCICE

- **1. Analyse du programme principal** : quels résultats ? quelles données ? quels modules faut-il ?
- **2. Analyse de chaque module** (fonction ou procédure) : rôle, paramètres, résultat, algorithme.
- **3. Écriture de l'algorithme**, puis **4. traduction en Python**.
- Toujours **tester à la main** sur un petit exemple (exécution pas à pas).

ابدأ دائما بتحليل البرنامج الرئيسي ثم تحليل كل وحدة على حدة، واكتب الخوارزمية ثم ترجمها إلى بايثون، وجرب النتيجة يدويا على مثال صغير.

Traduction Algorithme → Python

Algorithme	Python	Remarque
Lire(x)	<code>x = int(input())</code>	int, float ou rien selon le type
Ecrire(x)	<code>print(x)</code>	<code>print("a =", a)</code> pour un message
Si ... Alors ... Sinon ... FinSi	<code>if ... : ... else : ...</code>	elif pour Sinon Si
Pour i de a à b Faire	<code>for i in range(a, b + 1):</code>	range exclut la borne finale
Tant que cond Faire	<code>while cond:</code>	indentation obligatoire
Répéter ... Jusqu'à cond	<code>while True: ... if cond: break</code>	ou <code>while not cond</code>
a DIV b ; a MOD b	<code>a // b ; a % b</code>	quotient et reste entiers
T[i] (tableau)	<code>T[i]</code>	indices de 0 à n - 1
M[i, j] (matrice)	<code>M[i][j]</code>	ligne i, colonne j

FONCTION OU PROCÉDURE ?

- Une **fonction** calcule et **retourne une seule valeur** (elle s'utilise dans une expression). Une **procédure** exécute une tâche (affichage, saisie, modification d'un tableau) et ne retourne rien directement.
- Les **paramètres formels** sont dans l'en-tête du sous-programme ; les **paramètres effectifs** sont les valeurs données à l'appel.
- **Passage par valeur** : le sous-programme reçoit une copie (la variable de l'appelant ne change pas). **Passage par adresse** : le sous-programme peut modifier la variable de l'appelant. En Python, un **tableau** passé en paramètre est modifiable ; pour retourner plusieurs résultats, on utilise **return a, b**.
- Les **variables locales** n'existent que dans le sous-programme qui les déclare.

SYNTAXE EN ALGORITHMIQUE (USUELLE) ET EN PYTHON

Algorithmique

```

DEF FN Carre (n : Entier) : Entier
DEBUT
    Retourner n * n
FIN

DEF PROC Afficher (T : Tableau, n : Entier)
DEBUT
    Pour i de 0 à n - 1 Faire
        Ecrire (T[i])
    FinPour
FIN

```

Python

```

def carre(n):
    return n * n

def afficher(T, n):
    for i in range(n):
        print(T[i])

def min_max(T, n):
    # retourne deux résultats
    return min(T), max(T)

m, M = min_max(T, n)

```

BONNES PRATIQUES

- Un module = **un seul rôle** bien précis, avec un nom explicite.
- Ne pas lire ni afficher dans une fonction de calcul : séparer le calcul de l'affichage.
- Appeler une fonction : **y = carre(5)**. Appeler une procédure : **afficher(T, n)** (jamais dans une affectation).

ARABE

البرنامج الكبير يقسم إلى وحدات صغيرة تسمى برامج فرعية، وهذا يسهل الفهم والتصحيح ويمكن من إعادة استعمال نفس الجزء عدة مرات. الدالة تحسب وتعيد قيمة واحدة ونستعملها داخل عبارة، أما الإجراء فينفذ مهمة مثل العرض أو تعديل جدول ولا يعيد قيمة مباشرة.

الوسائط الشكلية هي الموجودة في تعريف البرنامج الفرعي، والوسائط الفعلية هي القيم التي نمررها عند الاستدعاء. في التمرير بالقيمة يعمل البرنامج الفرعي على نسخة فلا يتغير المتغير الأصلي، وفي التمرير بالعنوان يمكنه تعديل المتغير الأصلي. في بايثون يمكن تعديل الجداول المرسله كوسيط، ولإرجاع عدة نتائج نستعمل الإرجاع المتعدد. أما المتغيرات المحلية فلا يعرفها إلا البرنامج الفرعي الذي عرفها.

LE TABLEAU (VECTEUR)

- Il regroupe **n valeurs du même type** sous un seul nom. L'élément d'indice i s'écrit **T[i]**. En Python, les indices vont de **0 à n - 1** : sortir de ces bornes provoque une erreur.
- On traite un tableau avec **une boucle** (Pour) ; une **matrice** (l lignes, c colonnes) avec **deux boucles imbriquées**.

Saisie et maximum d'un tableau

```
from numpy import array

T = array([0] * n)          # tableau de n entiers
for i in range(n):
    T[i] = int(input("T[" + str(i) + "] = "))

def maximum(T, n):
    mx = T[0]
    for i in range(1, n):
        if T[i] > mx:
            mx = T[i]
    return mx
```

RECHERCHE SÉQUENTIELLE

- On parcourt le tableau jusqu'à trouver la valeur. On s'arrête dès que c'est trouvé : utiliser **while** avec un booléen, pas un Pour complet.

Python

```
def recherche(T, n, x):
    i = 0
    while i < n and T[i] != x:
        i += 1
    return i < n          # True si x est dans T
```

LA MATRICE

Création, saisie et diagonale

```
M = array([[0] * c for i in range(l)])      # matrice l x c

for i in range(l):
    for j in range(c):
        M[i][j] = int(input("M[" + str(i) + "][" + str(j) + "] = "))

def somme_diag(M, n):                        # matrice carrée n x n
    s = 0
    for i in range(n):
        s += M[i][i]                        # diagonale principale : i = j
    return s
```

- **Matrice carrée** : $l = c$. Diagonale principale : $i = j$. Diagonale secondaire : $i + j = n - 1$.
- **Exemple** : une matrice est « multiple d'un nombre premier » si le PGCD de tous ses éléments est premier (ex : $[[6, 12], [18, 24]]$ a un PGCD de 6, qui n'est pas premier).

ARABE

الجدول يجمع عدة قيم من نفس النوع تحت اسم واحد، ونصل إلى كل عنصر بواسطة رتيته التي تبدأ من الصفر في بايثون، لذلك فآخر رتبة هي الحجم ناقص واحد، والخروج عن هذه الحدود يسبب خطأ. المصفوفة جدول يبعدن له أسطر وأعمدة، ونحدد عنصرها برتبة السطر ثم رتبة العمود. لمعالجة جدول نستعمل حلقة واحدة، ولمعالجة مصفوفة نستعمل حلقتين متداخلتين. ننشئ الجداول والمصفوفات بواسطة مكتبة نمباي، وهي تخزن عناصر من نفس النوع بحجم ثابت. وعند البحث عن قيمة نتوقف بمجرد العثور عليها باستعمال حلقة مادام، ولا نكمل المرور على كل الجدول بدون فائدة.

3 Enregistrements et fichiers

السجلات والملفات

L'ENREGISTREMENT : PLUSIEURS CHAMPS DE TYPES DIFFÉRENTS

- Un enregistrement regroupe des **champs** de types différents décrivant **un seul objet** (un élève : nom, âge, moyenne). On accède à un champ par son nom.

Algorithmique

```
TYPE Eleve = Enregistrement
    nom : Chaîne
    age : Entier
    moy : Réel
Fin Eleve

e : Eleve
e.nom ← "Ahmed"
```

Python (dictionnaire)

```
e = {"nom": "Ahmed",
     "age": 17,
     "moy": 14.5}

print(e["nom"])
e["moy"] = 15
```

Selon ton professeur, l'enregistrement peut aussi s'écrire avec une classe. Le principe reste le même.

LES FICHIERS

- Un fichier conserve les données **de façon permanente** sur le disque. On l'ouvre avec un **mode**, on lit ou on écrit, puis on le **ferme** avec `close()`.
- Modes : **r** lecture, **w** écriture (efface le contenu existant), **a** ajout à la fin. Pour un fichier **binnaire**, on ajoute b : **rb**, **wb**, **ab**.

Algorithmique (fichier texte)	Python
Lire_ligne(F, ch)	<code>ch = F.readline()</code>
Ecrire(F, ch)	<code>F.write(ch)</code>
Ecrire_nl(F, ch)	<code>F.write(ch + "\n")</code>
Fin_fichier(F)	pas d'équivalent direct : <code>readline()</code> retourne "" à la fin

Fichier texte : lire toutes les lignes

```
f = open("notes.txt", "r")
ligne = f.readline()
while ligne != "":
    print(ligne.strip())
    ligne = f.readline()
f.close()
```

Fichier binaire : écrire puis relire des enregistrements

```
from pickle import dump, load

f = open("eleves.dat", "wb")          # ab pour ajouter à la fin
dump(e, f)                          # écrit un enregistrement
f.close()

f = open("eleves.dat", "rb")
try:
    while True:
        e = load(f)                  # lit le suivant
        print(e["nom"])
except EOFError:                     # fin du fichier
    pass
f.close()
```

ARABE

السجل نوع مركب يجمع عدة حقول من أنواع مختلفة تصف كيانا واحدا، مثل تلميذ له اسم و سن ومعدل، ونصل إلى كل حقل باسمه. الملف يخزن المعطيات بصفة دائمة على القرص، على عكس المتغيرات التي تضع عند إغلاق البرنامج.

الملف النصي يتكون من أسطر نقرؤها أو نكتبها. أما الملف الثنائي فنخزن فيه سجلات كاملة، ونقرؤها واحدا بعد الآخر إلى أن نصل إلى نهاية الملف. وضع القراءة يفتح الملف للقراءة فقط، ووضع الكتابة ينشئ الملف ويمسح محتواه القديم، ووضع الإضافة يضيف في النهاية دون مسح. لا تنس غلق الملف بعد الانتهاء حتى تحفظ المعطيات.

PRINCIPE

- Un sous-programme est **récuratif** s'il **s'appelle lui-même**. Il lui faut obligatoirement :
 - **1) une condition d'arrêt** (cas de base) qui donne le résultat sans nouvel appel ;
 - **2) un appel récuratif** qui rapproche le problème du cas de base à chaque fois.
- Sans condition d'arrêt, les appels ne finissent jamais : erreur de dépassement de la pile (RecursionError).

EXEMPLES CLASSIQUES

Python

```
def fact(n):                # n! = n x (n - 1)! et 0! = 1
    if n == 0:
        return 1
    return n * fact(n - 1)

def pgcd(a, b):             # Euclide : pgcd(a, b) = pgcd(b, a mod b)
    if b == 0:
        return a
    return pgcd(b, a % b)

def fib(n):                 # Fibonacci : F(n) = F(n-1) + F(n-2)
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
```

EXÉCUTION À LA MAIN : FACT(3)

- **Descente** : $\text{fact}(3) = 3 \times \text{fact}(2) \rightarrow \text{fact}(2) = 2 \times \text{fact}(1) \rightarrow \text{fact}(1) = 1 \times \text{fact}(0) \rightarrow \text{fact}(0) = 1$ (arrêt).
- **Remontée** : $\text{fact}(1) = 1 \times 1 = 1$; $\text{fact}(2) = 2 \times 1 = 2$; $\text{fact}(3) = 3 \times 2 = 6$.
- Chaque appel est mémorisé dans la **pile d'exécution** et les résultats se calculent **au retour**.

RÉCURSIF OU ITÉRATIF ?

- La version récursive est courte et proche de la définition mathématique, mais elle consomme plus de mémoire. **fib** récursif est très lent : il recalcule les mêmes valeurs (le nombre d'appels double environ à chaque étape).
- Méthode : écrire la **définition récurrente**, repérer le **cas de base**, puis traduire en code.

ARABE

العودية هي أن يستدعي برنامج فرعي نفسه. يجب أن تتوفر فيها دائما حالة توقف تعطي النتيجة مباشرة دون استدعاء جديد، واستدعاء ذاتي يقرب المسألة من حالة التوقف في كل مرة. إذا غابت حالة التوقف فإن الاستدعاءات لا تنتهي ويحدث خطأ في الذاكرة.

عند كل استدعاء يحفظ الحاسوب حالة الاستدعاء في ذاكرة تسمى الرصة، ثم تحسب النتائج بالترتيب العكسي عند الرجوع، كما رأينا في حساب الجداء العاملي. العودية تجعل الحل قصيرا وواضحا لكنها أكثر استهلاكاً للذاكرة وقد تكون أبطأ، خاصة في متتالية فيبوناتشي لأن نفس الحسابات تتكرر عدة مرات.

TRI PAR SÉLECTION

- À chaque tour i , on cherche le **minimum** de $T[i..n-1]$ et on l'échange avec $T[i]$. Au plus $n-1$ échanges.

Python

```
def tri_selection(T, n):
    for i in range(n - 1):
        imin = i
        for j in range(i + 1, n):
            if T[j] < T[imin]:
                imin = j
        T[i], T[imin] = T[imin], T[i]
```

TRI À BULLES

- On compare les voisins et on les échange s'ils sont mal rangés ; on **recommence tant qu'il y a eu un échange**. Les plus grandes valeurs « remontent » à la fin.

Python

```
def tri_bulles(T, n):
    echange = True
    while echange:
        echange = False
        for i in range(n - 1):
            if T[i] > T[i + 1]:
                T[i], T[i + 1] = T[i + 1], T[i]
                echange = True
```

TRI PAR INSERTION

- On garde la partie gauche triée. On prend $T[i]$, on **décale** vers la droite les éléments plus grands, puis on insère $T[i]$ à sa place.

Python

```
def tri_insertion(T, n):
    for i in range(1, n):
        x = T[i]
        j = i - 1
        while j >= 0 and T[j] > x:
            T[j + 1] = T[j]
            j -= 1
        T[j + 1] = x
```

TRI SHELL

- C'est un **tri par insertion avec un pas h** : on trie les éléments distants de h, puis on diminue h ($n//2$, $n//4$, ... jusqu'à 1). Plus rapide que l'insertion simple.

Python

```
def tri_shell(T, n):
    h = n // 2
    while h > 0:
        for i in range(h, n):
            x = T[i]
            j = i
            while j >= h and T[j - h] > x:
                T[j] = T[j - h]
                j -= h
            T[j] = x
        h //= 2
```

SEGMENTATION (BASE DU TRI RAPIDE)

- On prend un **pivot** $e = T[0]$. On range à gauche les éléments $\leq e$ et à droite les éléments $> e$, puis le pivot est placé à sa **position définitive**. Le **tri rapide** applique ensuite la même méthode, de façon récursive, sur la partie gauche et sur la partie droite.
- Exemple** : $T = [7, 5, 10, 3, 8]$ avec $e = 7$ donne $[3, 5, 7, 10, 8]$ (plusieurs résultats valides existent, selon l'ordre des échanges).

Python

```
def segmenter(T, n):
    e = T[0]
    k = 0 # nombre d'éléments <= e rencontrés
    for i in range(1, n):
        if T[i] <= e:
            k += 1
            T[i], T[k] = T[k], T[i]
    T[0], T[k] = T[k], T[0] # le pivot va à sa place
    return k
```

RECHERCHE DICHOTOMIQUE (TABLEAU TRIÉ OBLIGATOIRE)

- On compare x avec l'élément du milieu $m = (d + f) \text{ DIV } 2$; on garde la moitié qui peut contenir x. À chaque étape, la zone de recherche est **divisée par deux**.

Python

```
def dichot(T, n, x):
    d, f = 0, n - 1
    while d <= f:
        m = (d + f) // 2
        if T[m] == x:
            return m # trouvé à l'indice m
        elif T[m] < x:
            d = m + 1 # x est à droite
        else:
            f = m - 1 # x est à gauche
    return -1 # absent
```

COMPARAISON

Algorithme	Idée	Complexité
Sélection	min échangé en tête	$O(n^2)$
Bulles	échanges de voisins	$O(n^2)$
Insertion	décalages dans la partie triée	$O(n^2)$, $O(n)$ si déjà trié
Shell	insertion avec pas décroissant	meilleur que $O(n^2)$
Tri rapide	segmentation + récursivité	$O(n \log n)$ en moyenne
Recherche séquentielle	parcours complet	$O(n)$
Recherche dichotomique	moitié éliminée à chaque étape	$O(\log n)$

ARABE

الفرز هو ترتيب عناصر جدول تصاعدياً أو تنازلياً. في الفرز بالانتقاء نبحث عن أصغر عنصر ونضعه في مكانه الصحيح في كل مرة. في الفرز الفقاعي نقارن العناصر المتجاورة ونبدل مكانها حتى لا يبقى أي تبديل. في الفرز بالإدراج ندخل كل عنصر في مكانه داخل الجزء المرتب. وفرز شال هو فرز بالإدراج بين عناصر متباعدة بخطوة تتناقص إلى الواحد، وهو أسرع من الإدراج العادي.

التقطيع يعتمد على عنصر محوري: نضع الأصغر منه أو المساوي له على اليسار والأكبر على اليمين، فيصبح العنصر المحوري في مكانه النهائي. والفرز السريع يكرر نفس العملية بالعودية على الجزأين، وتعقيده في المتوسط أحسن من الخوارزميات البسيطة التي تعقيدها تربيعي.

البحث الثنائي لا يعمل إلا على جدول مرتب. في كل مرة نقارن القيمة المطلوبة بالعنصر الأوسط ونحذف النصف الذي لا يمكن أن توجد فيه، لذلك يكون تعقيده لوغاريتمياً وهو سريع جداً حتى مع الجداول الكبيرة.

ERREURS FRÉQUENTES

- Commencer les indices à 1 : en Python, le premier indice est **0** et le dernier est **n - 1**.
- Oublier le **return** dans une fonction, ou afficher au lieu de retourner.
- Récursivité sans **cas de base**, ou appel qui ne rapproche pas du cas de base.
- Utiliser la recherche dichotomique sur un tableau **non trié**.
- Écrire `range(a, b)` en croyant que `b` est inclus : il faut **`range(a, b + 1)`**.
- Oublier de **fermer le fichier** (`close`) ou d'ouvrir avec le mauvais mode (`w` efface le contenu).

تنبيه: رتب عناصر الجدول تبدأ من الصفر، ولا تنس حالة التوقف في العودية، ولا تستعمل البحث الثنائي إلا على جدول مرتب.

Ce résumé reprend les chapitres d'algorithmique et de programmation généralement vus en début d'année en 4ème Sciences de l'informatique. La syntaxe algorithmique et l'ordre des chapitres peuvent varier selon ton professeur : compare avec ton cours.