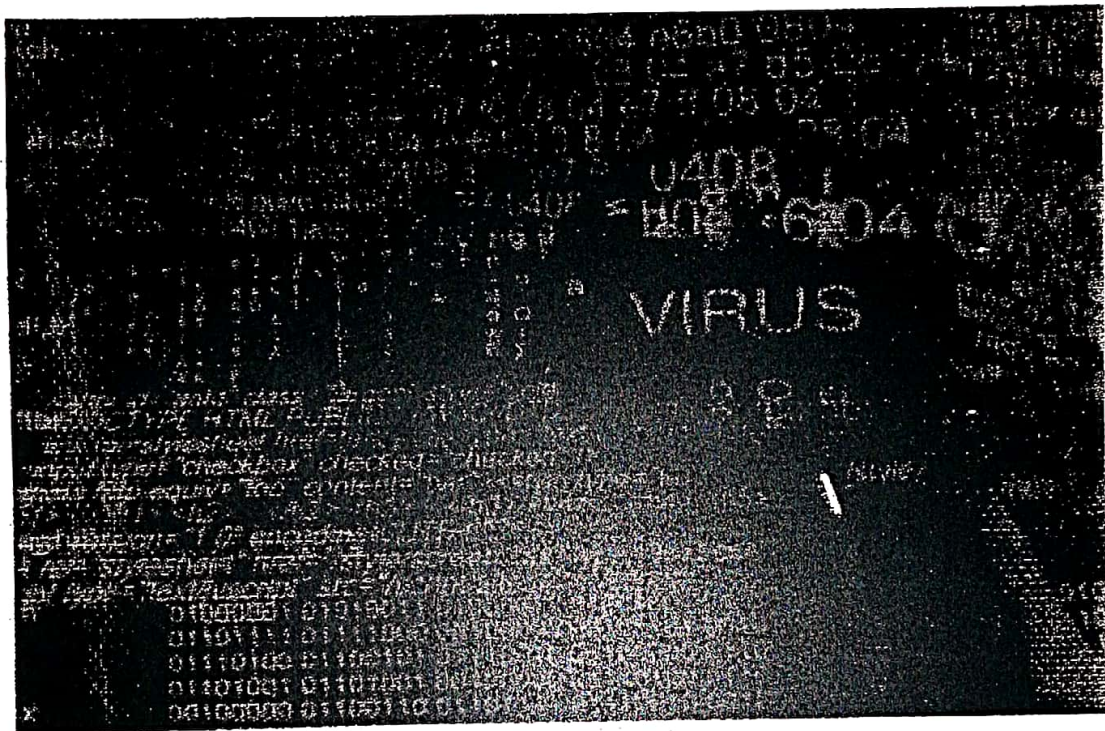


INFORMATIQUE

Cours Complet



PROF HAMOUDI
Sup/Spé

2016-2017

60

Programme 1^{re} année :

1. Généralités et structures de contrôle,
 - if-else (structures conditionnelles)
 - boucles

2. les fonctions et récursivité

3. structures de données
 - liste
 - tuple
 - ensemble
 - dictionnaire

4. la programmation orientée objet

5. les fichiers

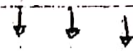
6. les piles et les files

7. L'ingénierie numérique et simulation

Culture générale 3 informatique

→ le rôle du langage est le traitement des données

les entrées → la machine lit (input)



Traitement.



Affectation

→ structures conditionnelles

→ boucles.

les sorties → la machine écrit (print)

→ Il y a donc 3 sortes de variables :

- variables d'entrées
- variables des sorties
- variables de traitement.

→ Chaque variable a un type :

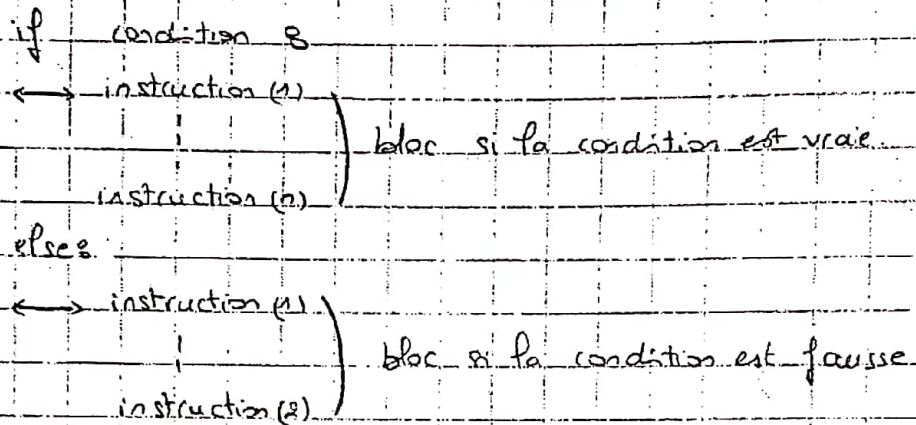
int = entier

float = réel

str = chaîne de caractère

les structures conditionnelles

① if - else :



Exemple :

Ecrire un programme qui donne
la valeur absolue d'un réel

```
n = float(input("Saisir un réel"))  
if n >= 0 :  
    y = n  
else :  
    y = -n  
print('la valeur absolue de, "a", est "y")
```

⚠ Tu peux aussi écrire quand tu as une seule condition :

```
if n >= 0 : y = n  
else : y = -n
```

⚠ Symboles à connaître :

$\geq$	$\leq$	$>$	$<$	$=$	et	$\&$	and
$\leq$	$\geq$	$<$	$>$	$=$	ou	$\ $	or
égalité	$\&$	$=$	$=$	$=$	Division euclidienne	$a \div b$	$r \mid q$
$\neq$	$\&$	$\neq$	$\neq$	$\neq$	q	$\&$	$q // b$
$\bar{p}$	$\&$	not	p	p	r	$\&$	$q \% r$

Exercice (1) :

Écrire un programme qui affiche si un entier est pair ou impair.

```
→ a = int(input('Saisir un entier : '))
if a % 2 == 0 :
    b = 'pair'
else :
    b = 'impair'
print(a, 'est', b)
```

Exercice (2) :

Écrire un programme qui affiche le max de 2 réels.

```
→ a = float(input('--- '))
b = float(input('--- '))
if a > b : m = a
else : m = b
print('le max est', m)
```

2

⚠ 6. Really important :

(1) si tu veux permuter a et b,

DON'T DO THIS :

c = a

a = b

b = c



⇒ Do this : a, b = b, a

Tu peux même le faire pour 3 éléments :

a, b, c = b, a, c

(2) $\left. \begin{array}{l} a = 2 \\ b = 2 \end{array} \right\} \Leftrightarrow a = b = 2$

↑
This is better

⊗ if, else, elif :

if condition (1) :

↳ instr (1)

⋮

instruct (n)

) bloc (1) ⇔ condition (1)

elif condition (2) :

instr (1)

⋮

instruct (2)

) bloc (2) ⇔ cond (1) et cond (2)
càd cond (1) non réalisée

else :

instr (1)

⋮

instruct (n)

) bloc (3) ⇔ cond (1) et cond (2)

Exercice 8

Écrire un programme qui retourne le max de 3 réels

```
a = float(input(' '))
b = float(input(' '))
c = float(input(' '))
if a > b and a > c :
    m = a
elif b > c :
    m = c
else :
    m = b
print('le max est', m)
```

Exercice 8

Écrire un programme qui affiche le produit de 2 réels
est positif, négatif ou nul

```
a = float(input(' '))
b = float(input(' '))
if (a > 0 and b > 0) or (a < 0 and b < 0) :
    s = 'positif'
elif a == 0 or b == 0 :
    s = 'nul'
else :
    s = 'negatif'
print('le produit est', s)
```

A savoir

python(2) $\frac{1}{2} = 0$ $\Rightarrow$ so it's better to write $1.0/2$
python(3) $\frac{1}{2} = 0.5$

les boucles

nbr d'itérations

Déterminé

(Dépend d'un itérateur)



for élément in itérateur :

instr (1)

instr (2)

Indéterminé

dépend d'une condition



while condition :

instr

instr

Associer

un itérateur en python est tout structure de données qui se composent de plusieurs éléments :

- liste
- tuple
- range
- dictionnaire
- ensembles

→ La fonction range retourne une suite de valeurs entières

• $\text{range}(a, b) = [a, b[$ si $a < b$, sinon $[],$

$\text{range}(2, 5) = 2, 3, 4$

• $\text{range}(n) = [0, n[$ si $n > 0$

$\text{range}(5) = 0, 1, 2, 3, 4$

• $\text{range}(a, b, c)$ si c est le pas

$\text{range}(4, 10, 2) = 4, 6, 8, 10$

$\text{range}(3, 0, -1) = 3, 2, 1$

→ les classiques :

① la somme :

$$\sum_{i=a}^b f(i) \Rightarrow \begin{array}{l} S = 0 \\ \text{for } i \text{ in range}(a, b+1) : \\ \quad S += f(i) \end{array}$$

⚠ : Be careful : c'est beaucoup mieux d'écrire :
 $S += f(i)$ ☺
que d'écrire $S = S + f(i)$ ☹

② le produit :

$$\prod_{i=a}^b f(i) \Rightarrow \begin{array}{l} P = 1 \\ \text{for } i \text{ in range}(a, b+1) : \\ \quad P *= f(i) \end{array}$$

⚠ : Again, it's better to write : $P *= f(i)$
que d'écrire $P = P * f(i)$

③ Une suite :

$$\begin{array}{l} u_0 = a \\ u_{n+1} = f(u_n) \end{array} \Rightarrow \begin{array}{l} u = a \\ \text{for } i \text{ in range}(1, n+1) : \\ \quad u = f(u) \end{array}$$

A savoir :

Toujours donner la priorité, quand c'est possible,
à for, au lieu de while.

A savoir :

Python revient automatiquement à la ligne après print.

Exercice 8

1. Ecrire un programme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit : (exemple : utilisateur entre le nombre 7) :

Table de 7 :

$$7 \times 1 = 7$$

⋮

$$7 \times 10 = 70$$

2. Faire une variante du programme précédent qui affiche la table de multiplication de tous les chiffres compris entre 2 et 9. Pensez à laisser une espace entre 2 tables de multiplication.

```
1. a = int(input(' - '))  
for i in range(1, 11):  
    p = a * i  
    print(a, 'x', i, '=', p)
```

A savoir :

Pour faire un saut de ligne, tu fais print().

```
2. for a in range(2, 10):  
    for i in range(1, 11):  
        print(a, 'x', i, '=', a * i)  
    print()
```

Always write the stupid stuff of the answer even if you don't know the whole answer, like:
def, s=0
you'll get some points

A savoir

tu peux être sanctionné si tu fais des lignes en plus, du genre:

```
s = e
print(s)
Ecris e
print(i)
```

A savoir

1 tabulation, c'est un espace régulier

A savoir

Pour printer plusieurs éléments dans la même ligne, si tu veux

→ une tabulation

```
print(n, end = '\t')
```

→ espace normal

```
print(n, end = ' ')
```

Exercice 8

Écrire un programme qui lit un nombre entier positif et affiche tous ses diviseurs

```
a = int(input(' '))
if a == 0: print('tous les entiers >= 1 sont des diviseurs')
else:
    if a < 0: a = -a
    for i in range(1, a//2 + 1):
        if a % i == 0: print(i, end = '\t')
    if a > 1: print(a)
```

6

A savoir :

par défaut :

end = '\n' et retour à la ligne.

Exercice 8

Écrire un programme qui lit un nbr entier positif n , et affiche s'il est parfait ou non. Un nbr est dit parfait s'il est égal à la somme de ses diviseurs stricts.
Exemple : $28 = 1 + 2 + 4 + 7 + 14$.

```
n = int(input(' '))
s = 0
for i in range(1, n//2 + 1):
    if n % i == 0:
        s += i
if n != 0 and s == n: print(n, 'est parfait')
else:
    print(n, 'est non parfait')
```

Exercice 8

Écrire un programme qui lit un nombre entier positif n et qui affiche tous les nbr parfaits inférieurs ou égaux à n .

```
n = int(input(' '))
for a in range(1, n + 1):
    s = 0
    for i in range(1, a//2 + 1):
        if a % i == 0: s += i
    if a == s: print(a)
```

Exercice 6

Écrire un programme qui effectue la multiplication de 2 nombres entiers en se basant uniquement sur des additions répétitives

```
a = int(input(' '))
b = int(input(' '))
if b > a: a, b = b, a
s = 0
for i in range(b):
    s += a
print('le produit est', s)
```

Exercice 7

Écrire un programme qui lit 2 entiers n et y , et divise ensuite n par y en utilisant uniquement les opérateurs d'addition et de soustraction. Le programme affiche ensuite le quotient et le reste

```
a = int(input(' '))
b = int(input(' '))
if b == 0: print("Erreur de division par zéro")
q, r = 0, a
while r >= b:
    q, r = q + 1, r - b
print('le quotient est', q)
print('le reste est', r)
```

⚠ Par défaut, la division sur 0, donne erreur.

Les fonctions

- Une fonction est identifiée par :
 - son nom
 - ses arguments

- Etape (1) :

Définir la fonction

- Etape (2) :

Appeler la fonction → exécuter la fct.

- 1. la définition

def nom_fonction (p₁, ..., p_n) :

inst(1)
|
|
inst(n) } le corps de la fct.

→ pour donner une valeur n à la fonction,
il faut utiliser "return n ".

⚠ s. Une fois exécuté, return arrête l'exécution
de la fonction.

• la fct renvoie = la fct return.

A savoir :

Ds les autres langages, on doit faire :

$a > b$ and $b > c$

Mais dans python, tu peux faire :

$a > b > c$

• 2 exemple

⇒ calculer $n!$

```
def factoriel(n):  
    p = 1  
    for i in range(2, n+1):  
        p *= i  
    return p
```

```
a = int(input('---'))  
print('le factoriel de', a, 'est', factoriel(a))
```

Exercices

1. Définir la fonction qui donne le max entre a et b
2. Définir la fonction qui donne le max entre a, b, et c

```
1. def Max_2(a, b):  
    if a > b: return a  
    return b
```

⚠ : on n'a pas besoin d'écrire else, avec return, c'est implicite, genre si on passe à return(b), c'est-à-dire, la 1^{re} condition n'est pas réalisée.

```
2. def Max_3(a, b, c):  
    if a > b and a > c: return a  
    if b > c: return b  
    return c
```

A savoir

une fct booléenne une fct qui retourne soit True soit False

8

Exercice 8

Écrire les fcts suivantes

1. Somme Diviseurs qui prend un entier n et renvoie la somme de ses diviseurs
2. Est-perfait qui prend un entier n qui affiche True si un nbr est parfait, False sinon.
3. afficher-perfait qui prend un entier n et affiche tous les nbr parfait $\leq n$.

```
1 def Somme Diviseurs (n) :  
    s = 0  
    for i in range (1, n//2 + 1) :  
        if n % i == 0 :  
            s += i  
    return s
```

A savoir :

return s'intéresse à la nature de ce qu'on lui donne, genre si je lui donne une condition qui est soit True or False, return va retourner True soit False, si je lui donne un entier, elle retourne un entier.
Cà dire genre si je fais
return (a = n > y)
si on a en effet, $n > y$, on aura True
sinon, False.

2. def EstParfait(n):
 return SommeDiviseurs(n) == n

⚠ Si la négation d'une fonction, genre max
is not max.

A savoir

Pour appeler les fcts qui ne retournent pas des valeurs, on utilise des ordres
par exemple, si je veux appeler une fct LOL qui ne retourne pas les valeurs, j'écris à la fin du programme LOL(n)

3. def afficherParfaits(n):
 for i in range(1, n+1):
 if EstParfait(i): print(i)

n = int(input(' '))
print('les nbr parfaits', n, 'sont : ')
afficherParfaits(n)

⚠ Rmq : on n'a pas fait
if EstParfait(i) == True :
it's wrong to do it !

Exercices

Ecrire les fcts suivantes :

1. Pgcd qui prend 2 entiers et renvoie leur pgcd avec les 2 méthodes :
 - Méthode d'Euclide
 - Méthode de soustractions successives

2. Puissance qui renvoie x^n , avec $(n, x) \in \mathbb{R} \times \mathbb{R}$

3. Somme chiffres qui prend un entier et retourne la somme des chiffres décimaux qui le composent

4. Est-premier qui renvoie si un entier n est premier ou non.

5. Est-divisible par 9, qui prend un entier n et renvoie si il est divisible par 9 ou non, en utilisant Q3 sans utiliser l'opérateur %

Explication

→ Méthode euclidienne

si b/a $\text{pgcd}(a, b) = b$

sinon : $r = a \% b$

donc $\text{pgcd}(a, b) = \text{pgcd}(b, r)$

→ Méthode soustraction

si $a = b$, $\text{pgcd}(a, b) = a$, sinon, on continue à soustraire le plus petit des plus grand jusqu'à ce que $a = b$.

$a = 24, b = 18$

$a = 6, b = 18$

$a = 6, b = 12$

$a = 6, b = 6$

1 # Méthode euclidienne :

```
def pgcd(a, b) :
```

```
    if a == b == 0 : return None
```

```
    if b == 0 : return a
```

```
    while a % b != 0 :
```

```
        a, b = b, a % b
```

```
    return b
```

Méthode soustraction :

```
def pgcd(a, b) :
```

```
    if a == b == 0 : return None
```

Δ genre arithmétique → if a + b == 0 : return a + b

Théorème de Bézout while a != b :

```
    if a > b : a = a - b
```

```
    else : b = b - a
```

```
    return a
```

2 def puissance(n, n) :

```
    if n == 0 :
```

```
        return None
```

```
    if n in [0, 1] :
```

```
        return 1
```

```
    p = 1
```

```
    for i in range(1, n + 1) :
```

```
        p *= n
```

```
    return p
```

m

3. Δ : Explication

$n = 235 \mid 10$
 $n = 23 \mid 10$
 $n = 2 \mid 10$
 $n = 0 \quad \checkmark$

def Somme_chiffres(n):

S = 0

while n != 0:

S += n % 10

n //= 10 $\rightarrow n = n // 10$

return S

4. Δ : Explications

$n = 1820$

somme = 11

somme = 2 $\neq 0, 9$

donc non divisible par 9.

def Est_divisible_par_9(n):

while n > 9:

n = Somme_chiffres(n)

if n in {0, 9}: return True

return False

Autre, tu fais :

return n in {0, 9}

(genre si en effet, $n = 0$ ou $n = 9$, il va retourner True, sinon False)

4 - def Est_premier(n):

if $n < 2$: return False

for i in range(2, int($n^{0.5}$) + 1):

if $n \% i == 0$: return False

return True

(parce que les diviseurs de $n \in [2, \sqrt{n}]$).

3 - les variables locales et les variables globales:

→ Variable locale:

• propre à une fct

• elle est définie dans la fct ou dans les paramètres

→ Variable globale:

• partagée par les fcts qui suivent sa définition

• Déclarée hors les fcts dans le programme principal.

A savoir:

Quand la variable locale existe, la fct does not care anymore about la globale ☹️

→ Exemple (1):

$a = 2$ # global

def f(n):

return $a * n$

$y = f(5) \rightarrow y = 10$

→ Exemple (2) :

a = 2 # global

def f(n) :

a = 3 # local

return a * n

y = f(5) → y = 15 (3 × 5)

print(a) → 2

→ Exemple (3) :

a = 2

def f(n) :

a = a + 1

return a * n

→ Erreur car a n'est pas encore définie !

He can't use a = 2 because he has to work with the local one !

→ Exemple (4) :

a = 2

def changer(b) :

a = b # a est locale

changer(5)

print(a) → 2

je veux que print(a) → 5 ☹️

que faire ?

a = 2

def changer(b) :

global a

a = b

changer(5)

print(a) → 5 😊

accès en
écriture

les structures de données :

les éléments de structures python peut être de type différent :

↳ Listes :

- ordre : oui.
- indexation : oui, avec des entiers
- Répétition de valeurs : oui.
- modifiable : en taille et en valeurs

↳ Tuples :

Comme les listes sauf que les tuples ne sont pas modifiable

↳ Ensemble :

- Ordre : NON.
- indexation : NON.
- répétition : NON
- modifiable : modifiable en taille seulement.

↳ Dictionnaire :

- Ordre : non.
- Indexation : par clés (sous forme de chaîne de caractère)
à chaque valeur on associe une clé
- Répétition :
 - clé : non
 - valeur : oui
- Modifiable :
 - Taille : oui
 - Valeur : oui
 - clé : non.

(1) les listes

I Définir les listes

① liste de valeurs à préciser

$l = [5, -7, 2.5, 'salut']$

$\begin{matrix} -4 & & -2 & & -1 \\ 0 & 1 & 2 & 3 \end{matrix}$

• $l[1] = -7$; $l[-1] = 'salut'$

• $\text{len}(l) = 4$ (taille)

• $l = [] \rightarrow$ liste vide

② la fct liste

• $l = \text{range}(1, 5)$

$\Rightarrow \text{len}(l) = 4$

• $l[0] = \text{range}(1, 5)$

• $l = \text{list}(\text{range}(1, 5)) \Rightarrow l = [1, 2, 3, 4]$

⚠ : En général :

• $l = \text{list}(\text{itérateur})$

• $l = \text{list}('Bonjour')$

$\Rightarrow l = ['B', 'o', 'u', 'j', 'o', 'u', 'r']$

• $l = \text{list}(1, 2, 3) \rightarrow$ erreur

(car 1, 2, 3 n'est pas un itérateur)

③ Par compréhension :

$$L = [f(n) \text{ for } n \text{ in itérateur if } P(n)]$$

Ex : en maths :

$$E = \{2^i / i \in [0, r[\}$$

$$E = [2^i * i \text{ for } i \text{ in range(5)}]$$

$$\Rightarrow E = [1, 2, 4, 8, 16]$$

$$B = [i+1 \text{ for } i \text{ in } E \text{ if } i \% 3 = 1]$$

$$\Rightarrow B = [2, 5, 17]$$

II - le parcours d'une liste :

① Parcours par indice :

$$\text{for } i \text{ in range(len(L)) :$$

$\Rightarrow i$: l'indice

$L[i]$: la valeur

Ex : Pour modifier les valeurs, ce parcours est nécessaire.

② Parcours par valeurs :

$$\text{for } i \text{ in } L :$$

$\Rightarrow i$: valeurs

III - Opérateurs :

Ex : + ne change pas l'ordre :

① Opérateur + : la concaténation

$$L = [1, 2, 3] + [4, 5]$$

$$\Rightarrow [1, 2, 3, 4, 5]$$

$$[4, 3] + [2]$$

$$\Rightarrow [4, 3, 2]$$

② Opérateur x :

$$L = [1, 2] * 3$$

$$\Rightarrow [1, 2, 1, 2, 1, 2]$$

③ Opérateur de comparaison

On compare les jères valeurs, mais attention Δ : les 2 listes should have the same type d'at les éléments. (genre both des chiffres for example)

$[0, 5] > [0, 8, 9, 10]$

$[1, 8] < [1, 8, 9, 10]$

IV - les sous listes

$l = \begin{matrix} -6 & -5 & -4 & -3 & -2 & -1 \\ 10 & 20 & 30 & 40 & 50 & 60 \\ 0 & 1 & 2 & 3 & 4 & 5 \end{matrix}$

$A = l[4:6] \Rightarrow [20, 30, 40]$

$B = l[4:8-1] \Rightarrow [20, 30, 40, 50] = l[4:5]$

$C = l[2:] \Rightarrow [30, 40, 50, 60]$

$D = l[:3] \Rightarrow [10, 20, 30]$

Δ
-1, 2e pas.

$E = l[4:1] \Rightarrow E = []$

$F = l[4:1:-1] \Rightarrow [50, 40, 30]$

$G = l[1:-2] \Rightarrow [20, 30, 50]$

$I = l[:] \Rightarrow I = [10, 20, 30, 40, 50, 60]$

But Δ : $I \neq l$ (oui)

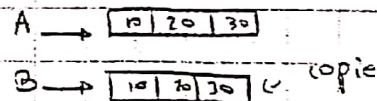
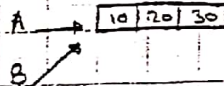
Explication :

$A = [20, 30, 40]$

$\Rightarrow A$ est la référence de la liste $[20, 30, 40]$

$B = A$

$\neq B = A[0] \Leftrightarrow B = A[0].copy()$



A savoir :

① $A = [10, 20, 30]$

$A = A + [40]$

$\Rightarrow A = [10, 20, 30, 40]$

$A \neq [10, 20, 30]$

$\rightarrow [10, 20, 30, 40]$

Non!

$A += [40]$

$\Rightarrow A = [10, 20, 30, 40]$

$A \rightarrow [10, 20, 30, 40]$

$A.append(40)$

Good! $\rightarrow$ c'est mieux.

②

def Ajouter(n, l) :

$l = l + [n]$

$A = [10, 20, 30]$

Ajouter(40, A) $\rightarrow$ $\left. \begin{array}{l} n = 40 \\ l = A \end{array} \right\}$

$\Rightarrow A = [10, 20, 30]$

$A \rightarrow [10, 20, 30]$

$l \rightarrow [10, 20, 30, 40]$

def Ajouter(n, l) :

$l = [n]$

$A = [10, 20, 30]$

Ajouter(40, A) $\rightarrow$ $\left. \begin{array}{l} n = 40 \\ l = A \end{array} \right\}$

$\Rightarrow A = [10, 20, 30, 40]$

$A \rightarrow [40, 20, 30, 40]$

$l \rightarrow$

$\rightarrow$ Exos classiques :

① Somme(l) $\Leftrightarrow$ sum(l) ② Max(l) $\Leftrightarrow$ max(l)

def Somme(l) :

$S = 0$

for i in l :

$S = i$

return S

def Max(l) :

$m = l[0]$

for i in l[1:] :

if $i > m$:

$m = i$

return m

③ indice Max(l) $\Leftrightarrow$ l.index(max(l))

def max_indice(l) :

$m = 0$

for i in range(1, len(l)) :

if $l[m] < l[i]$: $m = i$

return m

14

④ Recherche si $n \in$ liste :

```

def Recherche(n, l):
    for y in l:
        if y == n:
            return True
    return False

def indice(n, l):
    for i in range(len(l)):
        if n == l[i]:
            return i
    return None
  
```

return True si $n \in$ liste
return False sinon

IV - les méthodes associées aux listes :

① Ajout :

→ $l.append(n) \Rightarrow l_+ = [n]$
 ⇒ Ajout n à la fin de l

→ $l.insert(i, n)$
 ⇒ insérer n ds la position i
 $l = [10, 20, 30]$
 $l.insert(1, 40)$
 ⇒ $l = [10, 40, 20, 30]$

→ $l_+ = [40, 50]$
 ⇒ $l = [10, 20, 30, 40, 50]$
 ⇕
 $l.extend([40, 50])$

→ $l.append([40, 50])$
 ⇒ $l = [10, 20, 30, [40, 50]]$

② Suppression :

$l = [10, 20, 30, 20, 40]$
 → $u = l.pop(i)$: elle supprime et retourne la valeur $l[i]$

$u = l.pop(2) \Rightarrow \begin{cases} l = [10, 20, 40] \\ u = 30 \end{cases}$

→ $l.remove(n)$: elle supprime n mais n doit exister Δ
 $l.remove(20) \Rightarrow l = [10, 30, 40, 40]$

③ les fonctions et méthodes supplémentaires

→ `reversed(l)` → renvoie la liste / tuple / chaîne de caractère inversée

→ $A = [1, 2, 3]$

$B = \text{reversed}(A) \rightarrow B = [3, 2, 1]$

⚠️ Il fait une copie et ensuite la modifie.
c'est pour ça que c'est valable pour un tuple.

→ `l.reverse()` → inverse `l`, la liste.

⚠️ on l'utilise seulement pour les listes

→ `l.index(n)` → l'indice de `n` de `l`

→ `l.copy`

→ `l.sort()` → Trie la liste `l`.

$l = [3, 1, 4, 2]$

`l.sort()` → $l = [1, 2, 3, 4]$

`l.sort(reverse=True)` → $l = [4, 3, 2, 1]$

→ `l.count(n)` → nbr d'occurrences de `n` dans `l`.

⚠️ si $n \notin l$, il affiche 0.

Question: Comment parcourir une liste du dernier au 1^{er} élément

① Par indice:

`for i in range(len(l)-1, -1, -1):`

→ `i`: indice

`l[i]`: valeur

② Par valeurs

`for x in reversed(l):`

→ `x`: valeur

V - la création d'une liste

la taille (nbr d'éléments)

Déterminé: n

$l = [0] * n \Rightarrow l = [0, \dots, 0]$

for i in range(n):

calcul d'une valeur

$l[i] = \text{cette valeur}$

indéterminé

$l = []$

Boucle for ou while

si cette valeur vérifie la condition :

$l.append(\text{cette valeur})$

Exercices

1. Écrire la fct $\text{parfait}(n)$, la fct qui renvoie la liste des nbrs parfaits $\leq n$ en utilisant la fct Est_parfait .

2. $U_n = 3n + 1$

a. Écrire la fct $\text{Termes}(n)$ qui retourne la liste de U_0 à U_n .

b. Écrire la fct $\text{Som}(n)$ qui retourne la liste des termes $U_i \leq n$.

3. Suite de Fibonacci

$U_0 = U_1 = 1$; $U_{n+2} = U_{n+1} + U_n$

Écrire la fct $\text{termes_Fib}(n)$ qui renvoie la liste des termes de U_0 à U_n .

1 # Na méthode

```
def parfait(n):
```

```
    l = []
```

```
    for i in range(n+1):
```

```
        if est_parfait(i):
```

```
            l.append(i)
```

```
    # return l
```

Méthode du prof

```
def parfait(n):
```

```
    return [i for i in range(1, n+1) if EstParfait(i)]
```

2 a/

Méthode du prof

```
def Termes(n):
```

```
    return [3*i + 1 for i in range(n+1)]
```

Na méthode

```
def Termes(n):
```

```
    l[0] = n+1
```

```
    for i in range(n+1):
```

```
        l[i] = 3*i + 1
```

```
    return l
```

2 b/

Méthode (1)

```
def Termes_inf(n):
```

```
    u, i = 1, 0
```

```
    l = []
```

```
    while u <= n:
```

```
        l.append(u)
```

```
        i += 1
```

```
        u = 3*i + 1
```

```
    return l
```

Méthode (2)

```
def
```

```
    i = 0
```

```
    l = []
```

```
    while 3*i + 1 <= n:
```

```
        l.append(3*i + 1)
```

```
        i += 1
```

```
    return l
```


Méthode (3) :

def Termes inf(n) :

l = []

i = 0

while True :

u = 3*i + 1

if u <= n : l.append(u)

else : break

i += 1

return l

→ si tu fais
pas "break",
tu auras une
boucle infinie

3 - # Méthode (1) :

def fibo(n) :

a = b = 1

for i in range(2, n+1) :

a, b = b, a+b

return b

def Termes Fibo(n) :

return [fibo(i) for i in range(n+1)]

Méthode (2) (mieux)

def Termes Fibo(n) :

l = [1] * (n+1)

for i in range(2, n+1) :

l[i] = l[i-1] + l[i-2]

return l

→ utiliser la récursivité dans fibonacci est
déconseillé, voire sanctionné car la
complexité est très $\uparrow$

Exercice :

1. Ecrire une fct booléenne d'argument n, entier naturel qui renvoie True si n est premier, False sinon.
2. Ecrire une fct d'argument n renvoyant la liste des entiers premiers inférieurs ou égaux à n qui sont congrus à 1 modulo 4.

1 - def premier(n):
 if n < 2: return False
 for i in range(2, int((n**0.5) + 1)):
 if n % i == 0:
 return False
 return True

2 - # Méthode (1) is Trivial
 def List(n):
 L = []
 for i in range(1, n+1):
 if premier(i) and i % 4 == 1:
 L.append(i)
 return L

Méthode (2) is better
 def List(n):
 L = []
 for i in range(1, n+1):
 if i % 4 == 1 and premier(i):
 L.append(i)
 return L

Δ c'est plus facile to check si $n = 4k + 1$ que si n est premier, donc vaut mieux commencer avec $4k + 1$.

* Méthode (3) is even better
 Au lieu que n varie de (1, n+1),
 on prend directement les n qui
 vérifient $n = 4k + 1$, mais on
 élimine 1 car il n'est pas premier.

def List(n):
 L = []
 for i in range(5, n+1, 4):
 L.append(i)

Méthode (4) is even more better.
 def List(n):
 return [i for i in range(5, n+1, 4) if i premier(i)]

Exercices

1. Écrire la fct inverse(L) qui retourne la liste L inversée
2. Écrire la fct inverser(L) qui modifie la liste L en inversant l'ordre de ses éléments.

Q1/ sans modifier L

```
def Inverse(L):  
    n = len(L)  
    A = [0] * n  
    for i in range(n):  
        A[i] = L[n-1-i]  
    return A
```

Q2/ en modifiant L

```
def Inverse(L):  
    n = len(L)  
    for i in range(n//2):  
        L[i], L[n-1-i] = L[n-1-i], L[i]
```

(2) : Implémentation des matrices

- Une matrice est une liste de lignes
- Chaque ligne est une liste de nbrs
→ Une matrice est une liste de listes

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \Rightarrow M = \left[\underbrace{[1, 2, 3]}_0, \underbrace{[4, 5, 6]}_1 \right]$$

↖ ↗
même taille 

- $M[i]$ est une ligne (liste de nbr)
- $M[i][j]$ est le nbr de ligne i , et colonne j
→ $M[1][2]$ est 6.

Question

Questions : Comment écrire une matrice nulle de n lignes et m colonnes.

You would think :

$$M = [[0] * m] * n \leftarrow \text{FAUX} \quad \text{😞} \quad \text{!!!}$$

Réponse correcte :

$$M = [[0] * m \text{ for } i \text{ in range}(n)] \quad \text{😄} \quad \checkmark$$

Exercice 8

1. Écrire la fct Dimension(M) qui retourne une liste contenant le nbr de lignes et le nbr de colonnes de la matrice M .
2. Écrire la fct Somme(A, B) qui renvoie la somme des 2 matrices A et B si c'est possible.
3. Écrire la fct Produit(A, B) qui renvoie le produit des 2 matrices A et B .


```
1 - def Dimension (n):
    return [len(n), len(n[0])]
```

```
2 - def Somme (A, B):
    if Dimension(A) != Dimension(B): return None
    [n, m] = Dimension(A)
    C = [0]*m for i in range(n)
    for i in range(n):
        for j in range(m):
            C[i][j] = A[i][j] + B[i][j]
    return C
```

Meilleure méthode :

```
def Somme (A, B):
    return [A[i][j] + B[i][j] for j in range(m)] for i in range(n)]
```

```
3 - def Produit (A, B):
    [n, m], [m, p] = Dimension(A), Dimension(B)
    if m != m: return None
    C = [0]*p for i in range(n)
    for i in range(n):
        for j in range(p):
            for k in range(m):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

Meilleure méthode :

```
return [sum([A[i][k] * B[k][j] for k in range(m)]) for j in range(p)]
for i in range(n)]
```

(3) : les tuples

$$t = (\overset{-3}{5}, \overset{-2}{-3}, \overset{-1}{9})$$

$$\Rightarrow \text{len}(t) = 3$$

$$\Rightarrow t[0] = 5$$

$$\Rightarrow t[0] = 1 \Rightarrow \text{Erreur.}$$

$$t = \text{tuple}(\text{range}(4, 5)) \Rightarrow 4, 5$$

$$t = \text{tuple}([4, 2, 3]) \Rightarrow (4, 2, 3)$$

$$t = \text{tuple}() \Rightarrow \text{tuple vide}$$

$$t = (f(x) \text{ for } x \text{ in itérateur if } P(x))$$

c'est la méthode pour créer un tuple par compréhension :
"the classy method"

→ les fonctions et méthodes :

$$\text{reversed}(t) \quad | \quad t.\text{index}(x) \quad | \quad t.\text{count}(x)$$

→ Opérateurs :

$$t = (4, 2, 3) + (4, 5) \Rightarrow (4, 2, 3, 4, 5)$$

$$t = (4) + (2) = (3) \text{ car } (4) \text{ n'est pas un tuple } \Delta$$

$$t = (4, 1) + (2, 1) = (4, 2) \text{ car } (4, 1) \text{ est un tuple } \Delta$$

$$t = (4, 2) * 3 \Rightarrow t = (4, 2, 4, 2, 4, 2)$$

$$t = (4, 2)$$

$$t_1 = (3, 4) \Rightarrow t = (4, 2, 3, 4) \Leftrightarrow t = t + (3, 4)$$

$$t = (\underbrace{(4, 2)}_0, \underbrace{3}_1)$$

$$\Rightarrow t[0] = (4, 2, 4) \Rightarrow \text{Erreur}$$

$$\Rightarrow t[0].\text{append}(4) \Rightarrow t = ((4, 2, 4), 3)$$

(u) les ensembles

I Définir un ensemble

① $E = \{5, -1, 5, 2, 3\}$

$\Rightarrow E = \{5, -1, 2, 3\}$

$\text{len}(E) = 4$

② la fact set

$E = \text{set}('Bonjour')$

$\Rightarrow E = \{'B', 'o', 'u', 'r', 'j', 'o', 'u', 'r'\}$

$E = \text{set}([4, 2, -1, 3, 2, 5])$

$\Rightarrow E = \{4, 2, -1, 3, 5\}$

③ Compréhension

$E = \{f(x) \text{ for } x \text{ in itérateur if } f(x)\}$

II Opérateurs

① Comparaison

$E == F \Rightarrow$ m^ê éléments

$E != F$

$E \subseteq F$ si $E \subset F$

② ensembliste

$A - B \Rightarrow A \setminus B$

$A \& B \Rightarrow A \cap B$

$A | B \Rightarrow A \cup B$

$A \wedge B \Rightarrow A \Delta B = A \cup B \setminus A \cap B$

III Parcourir un ensemble

for x in $E \Rightarrow x$ est un élément de E .

IV Méthodes

• $E.add(x)$

$\Rightarrow$ ajouter x à E si $x \notin E$

sinon, nothing happens.

• $E.remove(x)$

$\Rightarrow$ Supprime x de E si $x \in E$,

sinon, Erreur.

• $E.discard(x)$

$\Rightarrow$ supprime x de E mais sans effet si $x \notin E$.

(3) : les dictionnaires :

I Définir un dictionnaire :

$$\textcircled{1} D = \{ 'a' : 2, 'b' : 3, 'c' : 2 \}$$

clé valeur

• $\text{len}(D) = 3$

• $D['a'] = 2$

• $D['a'] = 5 \Rightarrow D = \{ 'a' : 5, 'b' : 3, 'c' : 2 \}$

• $D = \{ a = 1, b = 2, c = 3 \}$ Des variables

$\Rightarrow D = \{ 'a' : 1, 'b' : 2, 'c' : 3 \}$

Don peut avoir plusieurs types de variables

Etudiant = $\{ 'Prénom' : 'Ali', 'Age' : 20, 'note' : 17.5 \}$

\textcircled{2} la 1^{ère} dict :

$$l = [('a', 1), ('b', 2), ('c', 3)]$$

⚠ c'est important d'avoir la clé et la valeur, soit dans une liste ou un tuple parce qu'on a besoin de l'ordre, car il considère toujours le 1^{er} élément = la clé et le 2^{ème} élément = la valeur.

le tuple (clé, valeur) est appelé : item.

\textcircled{3} par compréhension :

• $D = \{ \text{cle}(x) : f(x) \text{ for } x \text{ in itérateur if } f(x) \}$

Exemple :

Ecrire par compréhension : $\{ 'a' : 0, 'b' : 1, 'c' : 2 \}$

ch = 'abc' 0 1 2

$D = \{ ch[i] : i \text{ for } i \text{ in range(len(ch)) \}$

• $D = \{ 'a' : 1, 'b' : 2, 'c' : 3 \}$
 → `print(D['a'])` → 1
 → `print(D['e'])` → erreur
 → $D['a'] = 5 \Rightarrow D = \{ 'a' : 5, 'b' : 2, 'c' : 3 \}$
 → $D['e'] = 4 \Rightarrow$ l'item ('e', 4) sera ajouté.
 ⚠ si 'a' ∈ D, il la modifie, si 'a' ∉ D, elle est ajoutée.

Mais :

→ `D.setdefault('f', 6)`
 → l'item ('f', 6) sera ajouté si 'f' n'est pas une clé de D. sinon, D ne sera pas modifié. ⚠

II - le parcours d'un dictionnaire D :

• par clé :

if $x \in D \Rightarrow x$ est une clé de D
 ⇒ $\begin{cases} x : \text{clé} \\ D(x) : \text{valeur} \end{cases}$

• par valeurs :

for i in `D.values()` :
 → $i = \text{valeur}$

• parcours par Items :

for i in `D.items()` :
 ⇒ $\begin{cases} i = \text{item : tuple (clé, valeur)} \\ i[0] = \text{clé} \\ i[1] = \text{valeur} \end{cases}$

Meilleure méthode :

for c, v in `D.items()` :
 ⇒ $\begin{cases} c = \text{clé} \\ v = \text{valeur} \end{cases}$

Exercice 6

Les dictionnaires de cet exercice sont de la forme
Nom : note, correspond à chaque étudiant.

1. Écrire la fct Moyenne_générale (classe)
avec classe est un dictionnaire
2. Écrire une fct plus_grande_note (classe)
3. Écrire une fct majorants (classe) qui renvoie
l'ensemble des noms des étudiants qui ont eut
la plus grande note.
4. Écrire Réussi (classe), une fct contenant les
étudiants qui ont une note > 10 .

① # Na méthode

```
def Moyenne_générale (classe):  
    S = 0  
    for i in classe.values():  
        S += i  
    return (S / len (classe))
```

Méthode du prof

```
def Moyenne_générale (classe):  
    return sum (i for i in classe.values()) / len (classe)
```

② def plus_grande_note (classe):

```
    n = 0  
    for i in D.values():  
        if i > n: n = i  
    return n
```

A savoir

$E = \{ \}$ $\rightarrow$ Dictionnaire vide
 $E = \text{set} ()$ $\rightarrow$ Ensemble vide

③ def Majorants (classe):

E = set()

n = plus grande note (classe):

for i in classe:

if classe[i] == n:

E.add(i)

return E

meilleure méthodes

for c, v in classe.items():

if v == n:

E.add(c)

④ def Reussis (classe):

D = {}

for i in classe:

if classe[i] >= 10:

D[i] = classe[i]

return D

for c, v in classe.items():

if v >= 10:

D[c] = v

* for comprehension

Q3/

def Majorants (classe):

n = plus grande note (classe):

return { c for c, v in classe.items() if v == n }

Q4/ def Reussis (classe):

return { c : v for c, v in classe.items() if v >= 10 }

Les collections

On suppose avoir défini et rempli un dictionnaire `biblio` contenant des informations sur des livres d'une bibliothèque, telles que les clés sont les titres des livres, et les valeurs sont des dictionnaires dont les clés sont : `Auteur`, `prix`, `AnneeEdition`.

- 1- Ecrire une fonction `Livres_en_Annee(AnEdit)` qui prend en paramètre une année `AnEdit` et retourne un ensemble de tous les livres, du dictionnaire `biblio`, qui ont été édités à cette année. Chaque livre est représenté par un couple (titre, auteur).
- 2- Ecrire une fonction `NbrLivres(NomAuteur)` qui retourne le nombre de livres du même auteur dont le nom est `NomAuteur`.
- 3- Ecrire une fonction `Livres_communs(biblio1, biblio2)` qui prend en paramètres deux dictionnaires et retourne l'ensemble des titres des livres communs.
- 4- Ecrire une fonction `Plus_cher()` qui retourne le livre le plus cher de la bibliothèque, sous forme du tuple (titre, auteur, prix, année d'édition).
- 5- Ecrire une fonction `Meilleur_Prix(lstbib, TitreLivres)` qui, prend en paramètres une liste de dictionnaires représentant plusieurs bibliothèques et le titre d'un livre, et retourne l'indice de la bibliothèque qui offre le meilleur prix de ce livre.
- 6- Ecrire une fonction `MAJ(n, taux)` qui met à jour les prix des livres du dictionnaire `biblio` (voir plus haut), en faisant une réduction (de `-taux%`) sur les prix de tous les livres datés de plus de `n` années. On suppose que l'année en cours est stockée dans une variable globale `AnCourant`.
- 7- Ecrire une fonction `PrixAchat(Ens_Livres)` qui, prend en paramètres un ensemble de couples (Titre du livre, quantité), et calcule le montant global à payer pour acheter ces livres (on ignore les livres qui ne sont pas disponibles dans la bibliothèque `biblio`). Si la quantité dépasse 10, il faut appliquer une réduction de 10%.
- 8- Ecrire une fonction `Enregistrer()` qui enregistre les livres du dictionnaire `biblio` dans le fichier « `c:\livres.txt` » tels que chaque livre prend une ligne du fichier sous la forme suivante :

Titre ; auteur ; prix ; année d'édition

```
# Méthode (1)
1 - def Livres_en_Annee(AnEdit):
    E = set()
    for i in biblio:
        if biblio[i]['AnneeEdition'] == AnEdit:
            E.add((i, biblio[i]['Auteur']))
    return E
```


Méthode des listes

def Livres En Année (AnEdit):

```
return [c,v['Auteur']], for c,v in biblio.items():  
    if v['AnnéeEdition'] == AnEdit
```

2 def Nbr_Livres (NomAuteur):

N = 0

for i in biblio:

if biblio[i]['Auteur'] == NomAuteur

N += 1

return N

3 Méthode (1):

def Livres communs (biblio1, biblio2):

E = set()

for i in biblio1:

for j in biblio2:

if i == j:

E.add(i)

break

return E

Méthode (2): (Nouveau)

def Livres

E = set()

for i in biblio1:

if i in biblio2:

E.add(i)

return E

Méthode (3): (Encore mieux)

def

return [i for i in biblio1 if i in biblio2]

Méthode (4): (Excellent)

def

return set(biblio1) & set(biblio2)

(F.) : les fichiers :

on a

(2 types de fichiers)

Texte :

→ facile à programmer,
mais prend bcp d'espace.

→ Accès séquentiel : c-à-d
que pour trouver un élément, il
faut passer par les autres
éléments.

binaires

→ Dur à programmer
mais ne prend pas bcp
d'espace.

→ Accès directe : on sait
exactement où se trouve
un élément, car chaque
élément vaut 4 octets.

→ les étapes de traitement du fichier : à connaître.

1. Ouverture du fichier.
2. Lecture et/ou écriture des données.
3. Fermeture du fichier.

⚠ Dans l'exam, even
if you don't know
comment écrire
le traitement
(étape (2)), écris
l'étape (1) et (3)

I - Ouverture un fichier :

`f = open( NomFichier, mode )`

str : chaîne de
caractère.

→ 3 modes de base :

• 'r' : Read : lecture seule. ⚠ le fichier doit exister,
sinon Erreur.

• 'w' : Write : Écriture seule.

→ si le fichier ∅ il sera créé.

→ si le fichier ∃ il sera écrasé

• 'a' : Append : Ajout → Écriture à la fin seulement et automatiquement.

▷ si le fichier n'existe pas, il le crée.
si le fichier existe, on ne peut pas le modifier,
on peut juste écrire à la fin.

→ Rmq :

ces modes ouvrent le fichier en mode Texte.

→ Exemple

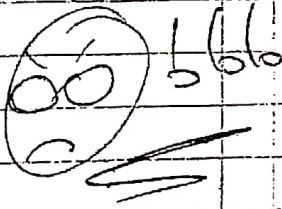
On a le fichier, dont le nom (le chemin) est :
c:\documents\cpge.txt

donc, you would do, just like me :

```
f = open('c:\documents\cpge.txt', 'a')
```

BUT

▷ We have a problem.



Pour python, antislash est //, et non \.

Donc pour lui, \ n'existe pas.

donc on remplace \ par //, même si dans l'énoncé de l'exo, ils écrivent le chemin du chemin avec un seul \.

Donc, la réponse correcte est :

```
f = open('c // documents // cpge .txt', 'a')
```

IV - Fermer le fichier :

```
f.close()
```

II - lecture d'un fichier texte

- Un fichier texte est une suite de caractères.
- Pour python, c'est une suite de lignes (c'est un itérateur de lignes).
- Une ligne est chaîne de caractères qui se termine avec '\n' '\n' est retour à la ligne.

① Lire une ligne :

ligne = f.readline()

⚠ Il lit la ligne en cours et se positionne au début de la ligne suivante.

→ Exemples

①	Plus on sait
②	Plus on doute
③	

① : ligne = 'Plus on sait \n'

③ : ligne = chaîne vide
" "

Si tu as :

1. la ligne vide, mais chaîne N'EST pas vide, car on a : ligne = '\n'.

② Parcourir ligne par ligne :

for ch in f :

→ ch est une ligne

⚠ ça commence à partir de l'emplacement du curseur.

③ Lire une liste de lignes :

• l = f.readlines()

⇒ Lire toutes les lignes à partir du curseur à la fin.

• l = f.readlines(n)

⇒ liras au max, n lignes, à partir du curseur.

⇒ l = ['plus... \n', 'plus... \n']

④ Tout le contenu sous forme d'une chaîne :

• ch = f.read()

⇒ ch = "plus... \n plus..."
chaîne de caractères

• ch = f.read(4)

⇒ 4 caractères :
"plus"

III Ecriture dans un fichier texte

① Ecriture une chaîne
f.write (chaîne)

② Ecrire liste de chaînes

f.writeLines (l) avec l est une liste de chaîne

l = ['plus on sait', 'plus on sait']

plus
on sait plus on

Les fichiers (exercices de base)

Exercice 1 :

Ecrire la fonction **copier(source, destination)** qui prend en paramètres les noms de deux fichiers textes et copie le contenu du premier dans le deuxième. On suppose que le premier fichier existe et que le deuxième est à créer.

Exercice 2 :

On suppose qu'on dispose d'un fichier texte de nom "c:\entiers.txt" contenant des nombres entiers (chacun sur une ligne).

- 1- Ecrire la fonction **somme()** qui retourne la somme des nombres de ce fichier.
- 2- Ecrire la fonction **recherche(x)** qui retourne True si x fait partie des nombres du fichier, ou False sinon.
- 3- Ecrire la fonction **ajouter(l)** qui prend en argument une liste d'entiers l et les ajoute au fichier "c:\entiers.txt"
- 4- Ecrire la fonction **copierPaires()** qui copie les nombres pairs de notre fichier dans un autre fichier à créer de nom "c:\pairs.txt" tel que 5 nombres par ligne séparés par espaces.

(7) les chaînes de caractères

C'est un itérateur de caractères

- Ordre des éléments : Oui
- Indexés : Oui
- Répétition : Oui
- modifiable : NON

→ Interesting stuff

Caractère	Code (entier)
256 caractères	ASCII (0, 255)
'0'	48
'1'	49
'A'	65
'B'	66
'a'	97
'b'	98

32

• `ord('A') = 65`

• `chr(65) = 'A'`

• `str(65) = '65'`

• `ord('1') = 49`

`int('1') = 1`

`ord(caractère)` donne l'unicode du caractère

⇒ 'A' est une chaîne de caractère de `len = 1`

⇒ on ne peut pas faire `ord('AB')`

Exo 1.5

```
def copier(source, destination):
    f = open(source, 'r')
    g = open(destination, 'w')
    for i in f:
        # i est une ligne
        g.write(i)
    f.close()
    g.close()
```

Exo 2

```
2 - def recherche(n):
    f = open('c:\\entiers.txt', 'r')
    for i in f:
        if int(i) == n:
            f.close()
            return True
    f.close()
    return False
```

⚠ Toujours close avant de faire return.

str(n) → n en type de chaîne de caractère.

```
3 - def ajouter(l):
    f = open('c:\\entiers.txt', 'a')
    for i in l:
        f.write(str(i) + '\n')
    f.close()
```

4 - def somme():

```
f = open('c:\\entiers.txt', 'r')
S = 0
for i in f:
    S += int(i)
f.close()
return S
```

5 - def copierPairs():

```
f = open('c:\\entiers.txt', 'r')
g = open('c:\\entiers.txt', 'w')
k = 0
for i in f:
    if int(i) % 2 == 0:
        k += 1
        if k % 5 == 0:
            g.write(i)
        else:
            g.write(i[:-1] + ' ')
f.close()
g.close()
```

⚠ quand on a fait : g.write(i), le \n est incl. dans i car on l'a copié de f

⚠ quand on a fait i[:-1] + ' ', on a enlevé le \n et on a ajouté un espace.

① Fonctions et méthodes :

③ Remplacer les sous chaîne

```
# ch8
chaîne
• for x in ch8
    ⇒ x est un caractère de ch
• if x in ch8
    ⇒ Teste si x est une sous-chaîne de ch
    ⇒ jour in 'Bonjour' : True
• i = ch.index(x)
    ⇒ l'indice de début de la 1ère sous-chaîne x
    ⇒ i = 'Bonjour'.index('jour')
    ⇒ i = 3
```

Adm • 'jour' in 'Bonjour' ⇒ False car J est majuscule

```
• i = ch.count(jour) = 1
    avec ch = 'Bonjour'
```

② Conversion en maj ou min

```
ch = 'Bonjour'
• u = ch.upper()
    ⇒ u = 'BONJOUR'
• y = ch.lower()
    ⇒ y = 'bonjour'
• z = ch.capitalize()
    ⇒ z = 'Bonjour'
```

on ne modifie pas la chaîne, on modifie la référence

```
R = ch.replace(x, y)
```

* y remplace tous les x

```
→ Texte = "Bonjour, c'est un agréable jour"
```

```
Texte = texte.replace('jour', 'soir')
```

```
⇒ 'Bonsoir, c'est un agréable soir'
```

→ Pour la suppression :

```
Texte = texte.replace('jour', "")
chaîne vide
```

④ Suppression de gauche et de droite ⇒ Nettoyage

```
ch = " python est un langage in"
```

```
ch = ch.strip()
```

```
⇒ ch = "python est un langage"
```

Par défaut, supprime les espaces et retour à la ligne.

```
ch = "! ! ? python, est ? ! ! "
```

```
ch = ch.strip('! ? , ')
```

• lstrip ⇒ left strip

•rstrip ⇒ right strip

⑤ Subdiviser une chaîne en liste de sous chaînes.

ch = 'Python est un langage'

• l = ch.split() *(par défaut, le séparateur est l'espace)*

→ l = ['Python', 'est', 'un', 'langage']
 0 1 2 3

• l = ch.split('n')

→ l = ['pytho', ' est u', ' la', '']

Suite exo collections :

Titre ; Auteur ; prix ; An Ed

Q4/ def enregistrier():

f = open('coll livres.txt', 'w')

c'est genre quand on plus de place

for i in biblio:

f.write(i + ';' + biblio[i]['Auteur'] + ';' + str(biblio[i]['prix'])
+ ';' + str(biblio[i]['Année Edition']) + '\n')

f.close()

Q1/ on répond en utilisant le fichier de la Q4

def livres_en_années(AnEdit):

E = set()

f = open('coll livres.txt', 'r')

for ch in f:

ch = ch.strip()

l = ch.split(';')

if int(l[3]) == AnEdit:

E.add((l[0], l[1]))

f.close()

return E

I. Tableau à 1 dimension & vecteur

→ Comment importer de la bibliothèque ?

- import numpy as np
- import scipy as sp
- import matplotlib.pyplot as plt

① Array

• $T = np.array(\begin{bmatrix} 5 & -7 & 8 \\ 0 & 1 & 2 \end{bmatrix})$

⇒ dtype = int

• $T[0] = 5$

• $T[0] = 3.2 \Rightarrow T[0] = 3$

• $T = np.array([3.2, -7, 8])$

⇒ dtype = float

genre if a
tout transformé
en float.

• $T = array([3.2, -7.0, 8.0])$

• len(T) = 3

• np.size(T) = 3

• np.shape(T) = (3,) ← Tuple

② Tableau de zéros

• $T = np.zeros(5) \Rightarrow array([0., 0., 0., 0., 0.])$

▷ par défaut : dtype = float

• $T = np.zeros(5, int) \Rightarrow array([0, 0, 0, 0, 0])$

→ En général :

$T = np.zeros(dimension, type)$

$T = np.zeros(3, bool)$

⇒ $array([False, False, False])$

• $T[0] = 5 \Rightarrow T[0] = True$

L'ingénierie numérique et simulation

le calcul scientifique

numérique
logiciels: Matlab, Scilab

symbolique ou formel
Maple, Mathematic

• Bibliothèques Python :

numpy : extension de scipy = module scipy intégré et numpy.linalg) algèbre linéaire

• Bibliothèque pour la représentation graphique d'objets
matplotlib.pyplot

Scilab : langage de script, non de programmation

→ la bibliothèque numpy :

la notion principale est "le tableau" (array)

1 dimension

vecteur

2 dimensions

matrice

Tableau (array)

Liste (list)

Similitude

- Valeur modifiable
- Indexation de 0 à len (Tableau)

- valeurs modifiable
- Indexation de 0 à len (Tableau)

Différence

- Taille statique
- Éléments de même type
- type statique
- Type est atomique
(entier, réel, booléen, caractère)

- Taille dynamique
- Éléments peuvent être de type ≠
- Type est dynamique
- N'importe quel type

→ Remarque

les fcts maths usuelles de Numpy renvoient l'image d'un tableau

$y = \text{np.sin}(x)$ où x est un tableau de nbres.

→ y : un tableau tq $\text{shape}(x) = \text{shape}(y)$

avec $y[i] = \sin(x[i])$

Applications

Tracer la courbe des fcts suivantes :

1 la courbe de $\sin$ sur $[-\pi, \pi]$

2 les courbes de $\sin$ et $\cos$ sur $[-\pi, \pi]$

1 `import numpy as np`

`import matplotlib.pyplot as plt`

`x = np.linspace(-np.pi, np.pi, 100)`

`y = np.sin(x)`

`plt.plot(x, y)`

`plt.show()`

2 `import numpy as np`

`import matplotlib.pyplot as plt`

`x = np.linspace(-np.pi, np.pi, 100)`

nom_courbe → `plt.plot(x, np.sin(x), label='sinus')`

nom_courbe → `plt.plot(x, np.cos(x), label='cosinus')`

`plt.legend()`

nom_graphe → `plt.title('le graphe de sin et cos')`

`plt.xlabel('L'axe des abscisses')`

`plt.ylabel('L'axe des ordonnées')`

`plt.show()`

③ Tableau des uns

$T = \text{np.zeros}(\text{dimension}, \text{type})$

④ Arange

• $T = \text{np.arange}(4, 5) \Rightarrow \text{array}([1, 2, 3, 4])$

* 4 est le pas • $T = \text{np.arange}(4, 2, 5, 1) \Rightarrow \text{array}([4.2, 2.2, \dots, 4.2])$

0.3 est le pas • $T = \text{np.arange}(1, 2, 0.3) \Rightarrow \text{array}([1, 1.3, 1.6, 1.9])$

par défaut,

le pas = 1.

⑤ Linspace int ou float

• $T = \text{np.linspace}(a, b, n)$ int : n est le nbr de valeurs de $[a, b]$, b inclus.

• $T = \text{np.linspace}(1, 2, 3) \Rightarrow T = \text{array}([1, 1.5, 2])$

$\text{linspace}(a, b, n) \Rightarrow \text{pas} = \frac{b-a}{n-1}$

II. Opérateurs associés au tableau

$A + B$: somme terme à terme (A et B de m dimension)

$A * B$: Produit de chaque terme par B.

$A + 3$: somme de chaque terme par 3

$A * B$: Produit terme à terme (A et B de m dim)

→ Les fonctions maths usuelles

`import math`

$x = \text{math.sqrt}(2)$

$y = \text{np.sqrt}(2)$ } $\Rightarrow x = y$

$A = \text{np.array}([1, 4, 9])$

$B = \text{math.sqrt}(A) \Rightarrow \text{Erreur}$

(le paramètre de `math.sqrt` doit être float ou int)

$B = \text{np.sqrt}(A)$

$\Rightarrow \text{array}([1, 2, 3])$ $3 = \sqrt{9}$
 $2 = \sqrt{4}$

28

III - les tableaux à 2 dimensions : (Matrices)

① Array :

```
n = np.array([[4, 2, 3], [4, 5, 6]])  
print(n) → array([[4, 2, 3], [4, 5, 6]])
```

len(n) = 2

np.size(n) = 6

np.shape(n) = (2, 3)

⚠ : $A * B$ → produit terme à terme

• np.dot(A, B) → produit matriciel $A \times B$

② Matrice nulle :

$n = np.zeros((n, m), type)$

③ Matrice de 1 :

$n = np.ones((n, m), type)$

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

④ Matrice identité :

• $n = np.eye(3) \Rightarrow n = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

• $n = np.eye(3, 2) \Rightarrow n = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$

• $n = np.eye(3, 3, 1)$ décalage de la diagonale
 $\Rightarrow n = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$

⑤ Fonctions usuelles :

• np.linalg.det(n) → déterminant

• np.linalg.inv(n) → Matrice inversée

• n.transpose() → la transposée de n à retourner

• Valeurs propres et vecteurs propres :

valp, vectp = np.eig(n)

→ valp = Tableau de n valeurs propres

→ vectp = Tableau de n vecteurs propres

Tableau
 $\text{vectp}[i]$ est associé à $\text{vclp}[i]$

$\Rightarrow$ vectp est une matrice

• Soit n une matrice

$n[i]$ est la ligne d'indice i de Tableau

$n[i, j] = n[i][j]$

valable pour tableau only!

valable pour tableau et liste!

Tableau

[10, 20, 30, 40]

Liste

$T = \text{np.array}([10, 20, 30, 40])$

• $A = T[483]$

$\Rightarrow A = \text{array}([20, 30])$

• $T[483] = 0$

$\Rightarrow T = \text{array}([10, 0, 0, 40])$

• $A = \ell[483]$

$\Rightarrow A = [20, 30]$

• $\ell[483] = 0 \Rightarrow$ Erreur

$T[i] = 0 \Leftrightarrow \text{for } i \text{ in range}(\text{len}(T)) \text{ et } \ell[i] = 0$

Tableau

$n = \text{np.array}([[-4, 2, 3], [4, 5, 6]])$

• $A = n[0]$

$\Rightarrow A = \text{array}([4, 2, 3])$

• $n = n[0, 0] \Rightarrow n = 4$

• $n[0] = 0 \Rightarrow$ Erreur

• $n[0] = \text{np.zeros}(3) \Rightarrow$ Erreur

• $A = n[i] \Rightarrow$ la ligne i de tableau

• $B = n[i, :]$ les éléments de la ligne i de tableau

• $n[i] = 0 \Rightarrow$ Erreur

• $n[i, :] = 0 \Rightarrow$ les éléments de la ligne i deviennent nuls

• $C = n[:, j] \Rightarrow$ les éléments de la colonne j

Liste

$n = [[4, 2, 3], [4, 5, 6]]$

• $A = n[0] \Rightarrow A = [4, 2, 3]$

• $n = n[0][0] \Rightarrow n = 4$

• $n[0] = 0 \Rightarrow n = [0, [4, 5, 6]]$

• $n[0] = [0, 0, 0]$

$\Rightarrow n = [[0, 0, 0], [4, 5, 6]]$

$C = [n[i][j] \text{ for } i \text{ in range}(\text{len}(n))]$

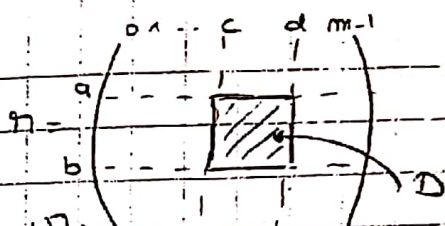
• Tableaux

$$D = n[a:b, c:d]$$

• Liste

$$D = [n[j][i] \text{ for } j \text{ in range}(c,d)]$$

$$\text{for } i \text{ in range}(a,b)]$$



Algorithme de résolution numérique

I - Calcul d'intégrales

① la fct `scipy.integrate.quad`

`from scipy.integrate import quad`

$$S, \text{err} = \text{quad}(f, a, b) \Rightarrow \int_a^b f(x) dx$$

$$S = \int_a^b f(x) dx$$

err = la marge d'erreur

Exemple

$$1 - \int_0^{\pi} \sin(x) dx ; 2 - \int_1^2 (2x+1) dx$$

1/ `from scipy.integrate import quad`

`from numpy import sin, pi`

$$S, \text{err} = \text{quad}(\sin, 0, \pi)$$

A savoir

Si tu fais `import numpy as np`, tu dois écrire `np.sin`, mais si tu importes `numpy`, write `sin`

2/ ⚠ on doit définir la fct $x \rightarrow 2x+1$

def `f(x)`:

return `2*x+1`

$$S, \text{err} = \text{quad}(f, 1, 2)$$

⚠ Meilleure méthode

$$S, \text{err} = \text{quad}(\lambda x. 2*x+1, 1, 2)$$

Si tu avais `sin(2x)`, il aurait fallu définir $x \rightarrow 2x$

② la méthode des rectangles :

$$S_{\text{rectangle}} = f(i) \times h$$

def integrale (f, a, b, h) :

$$S = 0$$

$$i = a$$

while $i < b$:

$$S_+ = f(i) \times h$$

$$i_+ = i + h$$

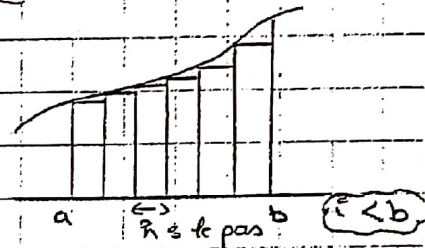
return S

genre on a sommé les surface

des mini rectangle de dimension

$h \times f(i)$, avec i varie de a à b .

pour que la définition de $\int$ est la surface balayée par la courbe



③ la méthode des trapèzes :

Rappel primaire :

$$S_{\text{trapèze}} = \frac{\text{قاعدة كبرى} + \text{قاعدة صغرى}}{2} \times \text{ارتفاع}$$

def integrale (f, a, b, h) :

$$S = 0$$

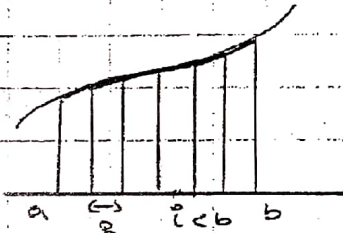
$$i = a$$

while $i < b$:

$$S_+ = (f(i) + f(i+h)) \times h / 2$$

$$i_+ = i + h$$

return S



II - le zéro d'une fct & résoudre $f(x) = 0$

① méthode de dichotomie :

la valeur approchée de x à ϵ près

Résoudre $f(x) = 0$ dans $[a, b]$ tq $f(a) \cdot f(b) < 0$.

def Dichotomie (f, a, b, eps = 10^{-6}) :

if $f(a) * f(b) > 0$: return None

while $\text{abs}(a - b) > \text{eps}$:

$m = (a + b) / 2$

if $f(a) * f(m) < 0$: $b = m$

else : $a = m$

return $(a + b) / 2$

② Méthode de Newton :

def Newton (f, Df, x0, n, eps = 10^{-6}) :

$F = \text{lamba} : x \rightarrow f(x) / Df(x)$

$x_1 = x_0$

for k in range(n) :

$x_{k+1} = F(x_k)$

if $\text{abs}(x_k - x_{k+1}) < \text{eps}$: return x_k

$x_k = x_{k+1}$

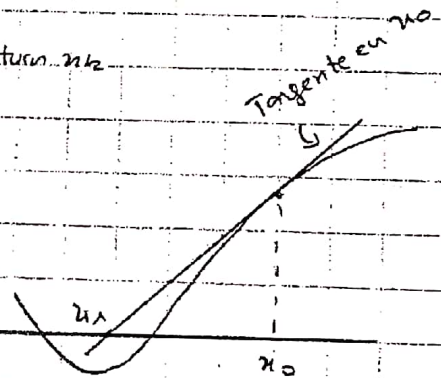
• Avantage : converge rapidement.

• Inconvénients : la cv n'est pas sûre.

• Nécessité la dérivée.

Suite récurrente :

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad \text{A connaître}$$



III - Résoudre les équ. diff.

④ la fct scpy. integrate.odeint :

① Une équ. diff. du 1^{er} ordre :

$$y' = f(y, t)$$

$$y(t_0) = y_0$$

Résoudre l'équation sur $[t_0, t_{fin}]$

→ une fct f est définie numériquement à l'aide de 2 tableaux x et y de même taille tq : $y[i] = f(x[i])$

31

Étape (1) est définir la fct qui calcule y'

```
def f(y, t):
    # l'ordre est important, do NOT write f(t, y) !!
    return y'
```

```
t = linspace(t0, tf, n)
sol = odeint(f, y0, t)
y = sol[:, 0] # y est la ière colonne
plt.plot(t, y)
plt.show()
```

Exemple :

$$\begin{aligned} y' &= 2y + \sin(t) = 0 \\ y(0) &= 1 \end{aligned} \quad \text{sur } [0, \pi]$$

```
import matplotlib.pyplot as plt
from numpy import sin, pi, linspace
from scipy.integrate import odeint
def f(y, t):
    return 2*y - sin(t)
t = linspace(0, pi, 1000)
sol = odeint(f, 1, t) # 1 = y(0)
y = sol[:, 0]
plt.plot(t, y)
plt.show()
```

ⓑ un système d'équa du 1^{er} ordre

```
def f(y, t):
    [y, z] = y
    return np.array([f1(y, z, t), f2(y, z, t)])
```

on résout le système



```
t = np.linspace(t0, tf, n)
sol = odeint(f, np.array([y0, z0]), t)
y = sol[:, 0] # 1ère colonne
z = sol[:, 1] # 2ème colonne
```

$$\begin{aligned} y' &= f_1(y, z, t) \\ z' &= f_2(y, z, t) \end{aligned} \Rightarrow \begin{aligned} y &= [y, z] \\ y' &= [f_1, f_2] \end{aligned}$$

$$\begin{aligned} y(t_0) &= y_0 \\ z(t_0) &= z_0 \end{aligned} \quad \text{sur } [t_0, t_f]$$

Exemple

$$\begin{aligned}y' &= -2y + z - t \\z' &= z + y \\y(0) &= 0 \\z(0) &= 1\end{aligned} \quad \text{sur } [0, 2]$$

```
def f(Y, t): # Y = [y, z]
    return np.array([-2 * Y[0] + Y[1] - t, Y[1] + Y[0]])
```

ou bien

```
def f(Y, t): #
    [y, z] = Y
    return np.array([-2 * y + z - t, z + y])
```

```
t = np.linspace(0, 2, 100)
sol = odeint(f, [0, 1], t)
y, z = sol[:, 0], sol[:, 1]
plt.plot(t, y, label='y')
plt.plot(t, z, label='z')
```

③ Equa. diff de 2^{ème} ordre

Il faut l'écrire sous forme d'un système de 2 équ. du 1^{er} ordre

$$y'' = f(y, y', t) \Rightarrow \begin{cases} y' = z \\ z' = f(y, z, t) \end{cases}$$
$$\begin{cases} y(t_0) = y_0 \\ y'(t_0) = y'_0 \end{cases}$$

```
def F(Y, t): # Y = [y, y'] = [y, z]
    return np.array([Y[1], f(Y[0], Y[1], t)])

t = np.linspace(t0, tf, n)
sol = odeint(F, np.array([y0, y0']), t)
y = sol[:, 0]
y_prime = sol[:, 1]
```


Exemple 8

$$y'' - 2y + \sin(y') - 3t = 0$$

$$\Rightarrow \begin{cases} y' = z \\ z' = 2y - \sin(z) + 3t \end{cases} \quad \begin{cases} y(0) = 0 \\ y'(0) = 0 \end{cases} \quad \text{sur } [0, \pi].$$

```
def F(Y, t): # Y = [y, z]
    return np.array([-Y[1], 2*Y[0] - sin(Y[1]) + 3*t])
t = np.linspace(0, np.pi, 100)
sol = odeint(F, np.array([0, 0]), t)
y, z = sol[:, 0], sol[:, 1]
plt.plot(t, y, label='y')
plt.plot(t, z, label='y_prime')
plt.legend()
plt.show()
```

Exercice 8

Résoudre :

$$1. \quad \begin{cases} y'' + \sin(y) - 2\cos(y') - t = 0 \\ y(0) = y'(0) = 0 \end{cases} \quad \text{sur } [0, 1]$$

et représenter y en fct de t .

$$2. \quad \begin{cases} y'' = y - z + y' \\ z'' = 2z - 3y + z' - t \\ y(0) = z(0) = 0 \\ y'(0) = z'(0) = 1 \end{cases} \quad \text{sur } [0, 5]$$

et représenter y et z en fct de t .

```

2 - { y' = a
      z' = b
      a' = y - z + a
      b' = 2*z - 3*y + b - t
def f(Y, t):
    [y, z, a, b] = Y
    return np.array([a, b, y - z + a, 2*z - 3*y + b - t])
t = np.linspace(0, 5, 1000)
sol = odeint(f, [0, 0, 1, 1], t)
y, z = sol[:, 0], sol[:, 1]
plt.plot(t, y, label='y')
plt.plot(t, z, label='z')
plt.legend()
plt.show

```

① la méthode d'Euler

y_0 est donné

$$y_{i+1} = y_i + h \cdot f(y_i, t_i)$$

• L'expression explicite d'Euler la fct d'Euler

→ prend f, y_0, t et renvoie y .

h et le pas.

→ prend f, y_0, t_0, t_f, n ou h et renvoie t et y .

```

4 - def Euler(f, y0, t):
    n = len(t)
    y = [0] * n
    y[0] = y0
    for i in range(n-1):
        y[i+1] = y[i] + (t[i+1] - t[i]) * f(y[i], t[i])
    return y

```

le pas.

Exemple 8

Résoudre $y' = 2y - t$
 $y(0) = 0$ sur $[0, 2]$
à l'aide de `odeint` et `euler`.

```
def f(y, t):  
    return 2*y - t  
t = np.linspace(0, 2, 1000)  
y = Euler(f, 0, t)  
sol = odeint(f, 0, t)  
plt.plot(t, y, label='Euler')  
plt.plot(t, sol[:, 0], 'ro', label='odeint')  
plt.legend()  
plt.show
```

→ 'ro' c'est rouge et avec des ronds.

→ On a dessiné le même tableau en utilisant les 2 méthodes pour voir quelle méthode est la meilleure.

```
2 def Euler(f, y0, to, tfm, h):  
    t = [to]  
    y = [y0]  
    a = to  
    while a < tfm:  
        y.append(y[-1] + h * f(y[-1], t[-1]))  
        a += h  
        t.append(a)  
    return t, y
```

méthode (1)

```
def Euler(f, y0, to, tfin, h):
```

```
    t, y = to, y0
```

```
    tt, yy = [to], [y0]
```

```
    while t < tfin:
```

```
        y = y + h * f(y, t)
```

```
        t = t + h
```

```
        tt.append(t)
```

```
        yy.append(y)
```

```
    return tt, yy
```

Example :

resolution de :

$$y' = 2y - t$$

$$y(0) = 0$$

sur $[0, 2]$

```
def f(y, t):
```

```
    return 2 * y - t
```

```
t, y = Euler(f, 0, 0, 2, 0.001)
```

```
sol = odeint(f, 0, t)
```

```
3. def Euler(f, y0, to, tfin, n):
```

```
    h = (tfin - to) / (n - 1)
```

```
    t = [to + i * h for i in range(n)]
```

```
    y = [0] * n
```

```
    y[0] = y0
```

```
    for i in range(n - 1):
```

```
        y[i + 1] = y[i] + h * f(y[i], t[i])
```

```
    return t, y
```


IV - la résolution d'un système linéaire $AX=B$

① la fct numpy.linalg.solve

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \times \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

• les entrées :

A : matrice carré de $n \times n$

B : vecteur (tableau à 1 dim) de taille n .

$x = \text{np.linalg.solve}(A, B)$

• la sortie :

x = vecteur de taille n tq :

$$x_1 = x[0], \dots, x_n = x[n-1]$$

→ Exemples

$$\begin{cases} x_1 - x_2 + x_3 = 0 \\ 2x_1 + x_2 + 3x_3 = 1 \\ 3x_1 - 2x_2 - x_3 = -5 \end{cases}$$

$$\Rightarrow A = \text{np.array}([[-1, 1, 1], [2, 1, 3], [3, -2, -1]])$$

$$B = \text{np.array}([0, 1, -5])$$

$$x = \text{np.linalg.solve}(A, B)$$

② la méthode de pivot de Gauss

le concepte on a la matrice comme ceci :

Explication

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \times \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

on veut annuler la partie li fiba $\square n, n$
pour chaque $i < n-1$

$$\Rightarrow a_{ji} = 0 \text{ pour } j > i$$

$$a_j = a_j - \frac{a_{ji}}{a_{ii}} \times a_i \leftarrow \text{Transvection de lignes}$$

Explication

Mais, on a besoin que a_{ii} soit très grand, donc on doit chercher la plus grande valeur en i de la colonne i .

quand on la trouve, on l'appelle "pivot": et on permute la ligne i si $i \neq p$ avec la ligne p si $i \neq p$.

△ Faut permuter b aussi.

$$a_i \leftrightarrow a_p \quad b_i \leftrightarrow b_p$$

$$b_j = b_j - \frac{a_{ji}}{a_{ii}} \cdot b_i$$

$$u_i = \frac{b_i - \sum_{j=i+1}^{n-1} a_{ij} \cdot u_j}{a_{ii}}$$

• def Pivot(A, i) :

$p = i$

for j in range($i+1$, len(A)) :

if abs(A[j][i]) > abs(A[p][i]) :

$p = j$

return p.

• def Permuter_lignes(A, i, j) # permuter A_i et A_j

$m = \text{len}(A[0])$ # Nbr de colonne

for k in range(m) :

$$A[i][k], A[j][k] = A[j][k], A[i][k]$$

• def Transvection_lignes(A, j, i, coef) # coef = $\frac{a_{ji}}{a_{ii}}$

for k in range(len(A[0])) :

$$A[j][k] = coef * A[i][k]$$


```

def solution(A, B):
    n = len(A)
    a = [l.copy() for l in A] # copie de matrice A.
    b = B.copy()
    for i in range(n-1):
        p = Pivot(a, i)
        if p != i:
            permuter_lignes(a, i, p)
            b[i], b[p] = b[p], b[i]
        for j in range(i+1, n):
            coef = a[j][i] / a[i][i]
            Transvection_lignes(a, j, i, coef)
            b[i] = b[i] * coef
    # Calculer x.
    x = [0] * n
    for i in range(n-1, -1, -1):
        s = 0
        for j in range(i+1, n):
            s += a[i][j] * x[j]
        x[i] = (b[i] - s) / a[i][i]
    return x

```

la programmation orientée objet

→ Python est totalement orienté objets.

- $x = 5 \Rightarrow \text{type}(x) = \text{int}$

$\Rightarrow x$ est un objet de classe int.

- $y = \text{'Bonjour'}$ $\Rightarrow y$ est un objet de classe str.

En python, le type $\Leftrightarrow$ une chose.

- $x = 5 \Rightarrow x$ est un objet de classe int.

$\Rightarrow x$ est une instance de classe int.

- Une classe est caractérisée par :

→ ses attributs & variables

→ ses méthodes & fonctions

- Définir un objet (une instance) $\Leftrightarrow$ instancier la classe

$\Rightarrow$ initialiser les attributs.

le rôle du constructeur de la classe.

→ Exemple : la classe complexe.

Un complexe est caractérisé par :

→ Ses attributs : Re et Im (de classe float)

→ Méthodes :

1 - le module de ce complexe.

2 - le conjugué de ce complexe.

3 - la somme de 2 complexes.

Remarque :

L'objet lui-même toujours fait partie des paramètres des méthodes. C'est le 1^{er} paramètre.

Par convention, on le nomme : "self".

$\Rightarrow$ On va maintenant construire la classe complexe.

Classe Complexe

```
def __init__(self, a=0, b=0):
    self.Re = float(a)
    self.Im = float(b)
def module(self):
    return (self.Re**2 + self.Im**2)**0.5
def conjugué(self):
    return complexe(self.Re, self.Im)
def somme(self, autre):
    return complexe(self.Re, self.Im +
                    autre.Im)
# on définit l'égalité de la classe complexe
def __eq__(self, autre):
    return self.Re == autre.Re and self.Im == autre.Im
```

Exercice : (Énoncé dans la feuille suivante)

1. Classe Point

```
def __init__(self, x=0, y=0):
    self.x = float(x)
    self.y = float(y)
def distance(self, autre):
    return ((self.x - autre.x)**2 + (self.y - autre.y)**2)**0.5
def milieu(self, autre):
    return point((self.x + autre.x)/2, (self.y + autre.y)/2)
```

2. Classe droite

```
def __init__(self, a, b):
    self.a = float(a)
    self.b = float(b)
def Appartient(self, p):
    if p.y == self.a * p.x + self.b:
        return True
    return False
```

```

def prend_point (self, n) :
    return point (n, self.a * n + self.b)

def parallele (self, p) :
    return droite (self.a, p.y - self.a * p.x)

```

3. Class Cercle :

```

def __init__ (self, a=0, b=0, c=0) :
    self.O = point (a, b)
    self.R = float(R)

```

* On ne peut

pas faire :

~~self.O = autre.O~~

car on ne sait

pas comparer

2 pt, genre on

ne la pas défini.

```

def Appartient (self, p) :

```

```

    return self.O.distance(p) == self.R

```

```

def Conformes (self, autre) :

```

```

    return self.O.x == autre.O.x and

```

```

        self.O.y == autre.O.y and

```

```

        self.R == autre.R

```

Q - comment savoir si une variable à une certaine classe, genre sa nature ?

→ savoir si n est un float ?

• if is_instance (n, float)

or if type(n) == float

Savoir si n est un float ou int :

```

if isinstance (n, int) or isinstance (n, float)

```

⇕

```

if type (n) in {int, float}

```


4 - def Drosite - 2 points (A, B):

if A.x != B.x:

$\alpha = (A.y - B.y) / (A.x - B.x)$

b = A.y - $\alpha \cdot A.x$

return drosite(a, b)

5 - def Circle 2 point (A, B):

if is_instance(A, point) and is_instance(B, point):

return Circle(A.x, A.y, A.distance(B))

Programmation orientée objets

Exercice :

I- Définir la classe Point dont :

1. Les attributs : les coordonnées x et y.
2. Les méthodes :
 - Distance qui renvoie la distance entre deux points.
 - Milieu qui retourne le point milieu entre 2 points.

II- Créons maintenant une autre classe, pour représenter des droites du plan, non verticales. Ces droites sont assimilables aux équations de type $y=ax+b$.

1. Les attributs représentant nos objets droite seront donc a et b.
← titre de la classe.
2. Ecrire les méthodes :
 - Appartient(p) qui renvoie si un point p appartient à la droite ou non.
 - prendPoint(x) qui renvoie le point de la droite ayant comme abscisse x.
 - parallele(p) qui renvoie la droite parallèle passant par le point p.

III- définir la classe Cercle dont :

1. les attributs : O et R
 - O : le centre, de classe Point.
 - R : le rayon, un réel

Le constructeur de la classe doit prendre 3 réels a, b et c. les deux premiers représentent les coordonnées du centre O, et le troisième représente la valeur du rayon.

2. Les méthodes :
 - Appartient(p) qui renvoie si un point p appartient à la droite ou non.
 - Confondus : fonction booléenne qui teste si deux cercles sont confondus ou non.

IV- écrire la fonction Droite2points(A,B) qui retourne la droite passant par les deux points A et B si c'est possible.

V- Ecrire la fonction Cercle2points(A,B) qui prend 2 points A et B, et renvoie le cercle de centre A et qui passe par B. La fonction doit s'assurer du type de A et B.

28

Les piles et les files :

Ce sont 2 structures de données dont 4 opérations de base sont autorisées, appelées primitives :

(1) Tester si elle est vide

(2) Ajouter un élément à la fin

(3) Supprimer un élément et renvoyer sa valeur

→ le dernier de Pile

→ le premier de File

(4) Vider la structure :

Supprimer tous les éléments

→ Bonq :

L'implémentation en python se fait à l'aide des listes.

I. les piles :

• Ajout à la fin $\Rightarrow$ Au sommet

$\Rightarrow$ Empiler $\Rightarrow$ push

• Suppression à la fin $\Rightarrow$ Au sommet

$\Rightarrow$ Dépiler $\Rightarrow$ pop

La pile est du type LIFO : Last In, First Out.

(Dernier arrivé est le premier servi)

① la pile de la taille dynamique (illimitée)

Cas (1) : le sommet de la pile correspond au dernier élément de la liste

→ Définition des fcts primitives :

def NouvellePile() :

return []

def Est_vide (pile): # it says True or False
return pile == []

def Empiler (n, pile): # complexité est.
pile.append(n)

def vider (pile):
pile.clear()

def Depiler (pile): # Elle supprime le dernier
return pile.pop() élément et le retourne.

→ Définition des fcts secondaires

def Hauteur (pile):
return len (pile)

def Val_sommet (pile): # renvoie la valeur
return pile[-1] du dernier élément sans
le supprimer

Cas (2) : le sommet de la pile correspond au 1^{er} élément
de la liste

def Nouvelle_Pile ():
return []

def Est_vide (pile):
return pile == []

def Empiler (n, pile):
return pile.insert(0, n)

def Depiler (pile):
return pile.pop()

def vider (pile):
return pile.clear()

def Hauteur (pile):
return len (pile)

def Val_sommet (pile):
return pile[0]

Exercice 6

en disposant uniquement des fcts ci-dessus.

Ecrire les fcts :

(1) Somme (P) qui renvoie la somme des éléments (nbrs) d'une pile P.

(2) Inverse (P) qui renvoie la pile P inversée.

(1) def Somme (P) :

S = 0

while not Est_vide (P) :

S += Depiler (P)

return S

Par récursivité :

def Somme (P) :

if Est_vide (P) : return 0

n = Depiler (P)

return n + Somme (P)

(2) def Inverse (P) :

L = Nouvelle_pile (L)

while not Est_vide (P) :

Empiler (Depiler (P), L)

return L

⚠ c'est FAUX de faire

Empiler (Depiler (P), Nouvelle_pile (L))

car il va créer à chaque fois

une nouvelle pile

→ la pile en tant qu'une classe

L'attribut est une liste L d'éléments.

class Pile :

def __init__ (self) :

pile = L = []

def Est_vide (self) :

return self.L == []

def Empiler (self, n) :

Sommet

self.L.append(n)

def Depiler (self) :

return self.L.pop()

def vider (self) :

self.L.clear()

```

def hauteur (self):
    return len(self.l)
def valsommet (self):
    return self.l[-1]

```

⇒ On refait l'exercice avec P étant une classe :

1. def Somme (P):	2. def Inverse (P):
S = 0	L = Pile()
while not P.Est_vide():	while not P.Est_vide():
S += P.Depiler()	L.Empiler(P.Depiler())
return S	return L

Exercice 8 (classique)

Tester si une expression mathématique (chaîne de caractère) est correctement parenthésée.
une parenthèse est : '(', '[', '{', ')', '}', ']', ']'

Le concepteur on a expression du genre : $[(5+9)*3]$
on parcourt l'expression, on trouve au début '[', alors on crée une pile et on pose dans cette pile, l'inverse de '[' ']
Donc on continue de parcourir l'expression. Alors, si on trouve '(' ou '{', on pose encore l'inverse de '(' ou '{' dans la pile ']. Si on trouve une parenthèse qui ferme '(' ou '[', on la compare avec le sommet de la pile. If they are the same, on dépile. Sinon, l'expression n'est pas bien parenthésée. A la fin, on doit avoir une pile vide.

Méthode (1) :

def Est_parenthésée (exp) :

P = Nouvelle pile()

A = ['(', '[', '{']

B = [')', ']', '}']

for u in exp :

if u in A :

Empiler (B[A.index(u)], P)

elif u in B :

if Est_vide(P) or u != val_sommet(P) :

return False

else : Depiler (P)

return Est_vide(P)

Méthode (2) : Dictionnaire

def Est_parenthésée (exp) :

P = Nouvelle pile()

D = {'(' : ')', '[' : ']', '{' : '}'}

for u in exp :

if u in D.keys() :

Empiler (D[u], P)

elif u in D.values() :

if Est_vide(P) or u != val_sommet(P) :

return False

else : Depiler (P)

return Est_vide(P)

Méthode (3) : Sans définir les primitives

def Est_parenthésée (exp) :

pile = []

D = {'(' : ')', '[' : ']', '{' : '}'}

for u in exp :

if u in D.keys() :

pile.append (D[u])

elif u in D.values() :

if pile == [] or u != pile[-1] :

return False

else : pile.pop()

return pile == []

Exercice (2) : Classique

La notation polonaise inversée
(notation post fixe) (NPI).

→ Elle est rapide dans l'exécution

→ On n'a pas besoin d'utiliser des ()

• $x + y \Rightarrow$ unifixé

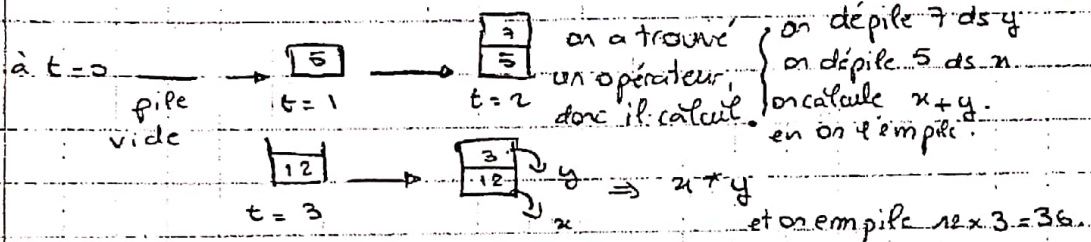
• $+ xy \Rightarrow$ préfixé

• $xy + \Rightarrow$ post fixe

→ Comment ça marche ??

Elle est basée sur une pile.

$57 + 3 *$



on utilise 2 fcts, à écrire :

1 - Calcul (op, x, y) qui retourne la valeur de " $x op y$ " avec $op \in \{ '+', '-', '*', '/' \}$.

2 - Evaluer ($expr$) tq $expr$ est une liste qui évalue une expression en (NPI).

genre notre expression est $[5, 7, '+', 3, '*']$

1 - def Calcul (op, x, y) :

if $op == '+'$: return $x + y$

if $op == '-'$: return $x - y$

if $op == '*'$: return $x * y$

return x / y


```

2 def Evaluer(expr):
    pile = Nouvelle_Pile()
    op = ['+', '-', '*', '/', '^']
    for i in expr:
        if i not in op: Empiler(i, pile)
        else:
            y = Depiler(pile)
            x = Depiler(pile)
            Empiler(calcul(i, x, y), pile)
    return Depiler(pile)

```

→ Comment faire si je n'ai pas les jats définies ?

```

def Evaluer(expr):
    pile = []
    op = ['+', '-', '*', '/', '^']
    for i in Expr:
        if i not in op: pile.append(i)
        else:
            y = pile.pop()
            x = pile.pop()
            pile.append(calcul(i, x, y))
    return pile[0]

```

② Pile de taille statique (limitée)

Let's work with les classes

Class Pile:

```

def __init__(self, n):
    self.Nmax = n
    self.l = [0] * n
    self.sommet = -1

```

```

def Est_vide(self):
    return self.sommet == -1

def Est_pleine(self):
    return self.sommet == self.Nmax - 1

def Empiler(self, u):
    self.sommet += 1
    self.l[self.sommet] = u

def Depiler(self):
    self.sommet -= 1
    return self.l[self.sommet + 1]

def vider(self):
    self.sommet = -1

def hauteur(self):
    return self.sommet + 1

def vsommet(self):
    return self.l[self.sommet]

```

II les files

- Ajouts à la fin $\Rightarrow$ En queue de la file
 $\Rightarrow$ Enfiler
- Suppressions premier $\Rightarrow$ En tête de la file
 $\Rightarrow$ Defiler.

De type FIFO: First in, First OUT
(Implémentation à l'aide d'une liste.)

① File dynamique.

Enfiler(n, file) $\Leftrightarrow$ file.append(n)

Defiler(file) $\Leftrightarrow$ file.pop(0)

② File statique

Les attributs :

- N_{max} : Taille de la liste
- l : Liste de taille N_{max}
- Tête : indice du 1^{er} élément
- Queue : indice du dernier élément

def Nouvelle File(n) :

return { 'Nmax' : n, 'l' : [0]*n, 'Tête' : 0, 'Queue' : -1 }

def Est_vide (file) :

return file['Tête'] > file['Queue']

def Est_pleine (file) :

return file['Queue'] == file['Nmax'] - 1

def Enfiler (n, file) :

file['Queue'] += 1

file['l'][file['Queue']] = n

def Defiler (file) :

n = file['l'][file['Tête']]

file['Tête'] += 1

return n

def vider (file) :

file['Tête'] = 0

file['Queue'] = -1

def longueur (file) :

return file['Queue'] - file['Tête'] + 1

La récursivité

Répétition d'exécution

- Boucles itérative (plus rapide)
- En python, itérative est 10x plus rapide que récursive
- Récursivité récursive

→ Une fct récursive = fct qui fait appel à elle-même & répétition d'exécution, d'où la nécessité d'une condition d'arrêt.

Procédure C/C : Une fct récursive se compose de :

- Une condition d'arrêt (souvent implicite en cas d'une procédure)
- Appel récursif = Appel de la fct à elle-même.

fct qui ne retourne rien!

→ Exemple : Calculer $n!$

Méthode itérative

$n! = 1 \times 2 \times \dots \times n$: boucle

Méthode récursive

$n! = \begin{cases} 1 & \text{si } n < 2 \text{ : cond d'arrêt.} \\ n(n-1)! & \text{sinon : appel récursif} \end{cases}$

def fact(n) :

if $n < 2$: return 1

return $n * \text{fact}(n-1)$

Exercice : Écrire les fcts récursives

1 - la fct puissance (x, n) qui renvoie x^n , avec $(x, n) \in \mathbb{R} \times \mathbb{N}, \geq 0, 05$

2 - Soit U défini par :

$$U_0 = 5$$

$$U_{n+1} = 2U_n - 1$$

Écrire la fct Suite(n) qui renvoie U_n .

3. $\text{pgcd}(a, b)$ avec les 2 méthodes :

→ Euclide

→ Soustraction successive

4. Afficher $\text{Dec}(n)$ qui affiche $n, n-1, \dots, 1$, séparés par des espaces.

5. Afficher $\text{Croissant}(n)$ qui affiche $1, \dots, n$.

6. $\text{Somme}(l)$ qui renvoie des éléments (n bc) d'une liste l .

7. $\text{Recherche}(n, l)$ qui renvoie True si $x \in l$ ou False sinon.

8. $\text{Rech_dichot}(n, l)$ qui effectue la recherche dichotomique de x dans l . On suppose que l est triée.

9. $\text{liste_croissante}(n)$ qui renvoie $[1, 2, \dots, n]$.

10. $\text{liste_Decroissante}(n)$ qui renvoie $[n, n-1, \dots, 1]$.

11. $\text{Inverse}(l)$ qui renvoie la liste inversée.

l 2 e normal

3/ # Méthode Euclide

```
def pgcd(a, b):  
    if a % b == 0: return b  
    return pgcd(b, a % b)
```

4/ def Afficher Dec(n):

```
    if n > 0:  
        print(n, end=" ")  
        Afficher Dec(n-1)
```

5/ def Afficher Crois(n):

```
    if n > 0:  
        Afficher Crois(n-1)  
        print(n, end=" ")
```

Méthode soustraction

```
def pgcd(a, b):  
    if a == b: return a  
    if a < b: a, b = b, a  
    return pgcd(a-b, b)
```

6/ def Somme(l):

```
    if l == []: return 0  
    return l[0] + Somme(l[1:])
```

7/ def Recherche(n, l):

```
    if l == []: return False  
    if n == l[0]: return True  
    return Recherche(n, l[1:])
```

```

9/ def ListeDecroissante(n):
    if n <= 0: return []
    l = ListeDecroissante(n-1)
    l.insert(0, n)
    return l

```

Autre méthode :

```

11/ def inverse(l):
    if l == []: return []
    return [l[-1]] + inverse(l[0:-1])

```

⚠ Ne j'ai pas retourné l

```

8/ def Rech_dichot(n, l):
    if l == []: return False
    m = len(l) // 2
    if n == l[m]: return True
    if n < l[m]:
        return Rech_dichot(n, l[0:m])
    return Rech_dichot(n, l[m+1:])

```

l est list

Programme 2^{ème} années

1 - les bases de données

2 - les Tri

3 - Complexité des programmes

4 - les exceptions

5 - Les arbres

6 - les graphes

→ correction de quelques concours

45

Le Tri

I la méthode prédéfinie de python : sort

• Cas (1) : $l = [5, -1, 2, -2, 3]$

• $l.sort() \Rightarrow [-2, -1, 2, 3, 5]$

• $l.sort(reverse = True) \Rightarrow$ Tri décroissant.

• Cas (2) : $l = [('Karim', 15), ('Ali', 21), ('Ibrahim', 18)]$

• si je fais $l.sort()$, what happens ?

on a des tuples, donc pour comparer des tuples,

on compare les 1^{ers} éléments des tuples (les lettres).

Si ils sont semblables, on passe au 2^{èmes} éléments des tuples.

$\Rightarrow l = [('Ali', 18), ('Ibrahim', 18), ('Karim', 15)]$

• et si je veux un tri croissant selon l'âge ?

$l.sort(key = lambda t : t[1])$

• et si je veux un tri croissant selon l'âge et en

cas d'égalité, par ordre alphabétique des noms ?

$l.sort(key = lambda t : (-t[1], t[0]))$

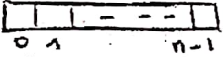
• Et si je veux un tri décroissant selon l'âge et en

cas d'égalité, tri croissant d'ordre alphabétique des noms ?

$l.sort(key = lambda t : (-t[1], -t[0]))$

II les algorithmes de Tri standards par sélection - bulle - par insertion.

① Tri par sélection

- le concepte : on a 
on cherche l'indice de l'élément min de la liste,
et on le permute avec la case 0.
on cherche ensuite le min de 
et on permute avec la case 1.

1^{ère} étape : comment trouver le min ?

def IndiceMin(l) :

n = len(l)

m = 0

for j in range(1, n) :

m = j

return m

2^{ème} étape : trouver le min d'une
liste mais on commence
à partir de l'indice i.

def IndiceMin(l, i) :

n, m = len(l), i

for j in range(i+1, n) :

if l[j] < l[m] :

m = j

return m

⇒ Écrivons maintenant le programme du Tri

def Tri_Selection(l) :

n = len(l)

for i in range(n-1) :

m = i

for j in range(i+1, n) :

if l[j] < l[m] :

m = j

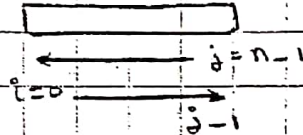
l[i], l[m] = l[m], l[i]

Exercice :

Écrire une fct Est_Triée qui teste si une liste est
triée ou non.

② Tri Bulle

- le concepte: on permute $l[i]$ et $l[i+1]$ si $l[i] > l[i+1]$. Mais, on remarque que quand on fait ceci à toute la liste pour la 1^{ère} fois, la plus grande valeur se trouve dans la dernière case. Donc on doit travailler avec 2 variables i, j .



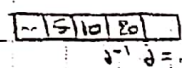
En utilisant nbr de permutation: En utilisant la jct Est trié

```
def Tri_Bulle(l):
    n = len(l)
    for j in range(n-1, 0, -1):
        Nbr_permutations = 0
        for i in range(j):
            if l[i] > l[i+1]:
                l[i], l[i+1] = l[i+1], l[i]
                Nbr_permutations += 1
        if Nbr_permutations == 0: break
```

```
def Tri_Bulle(l):
    n = len(l)
    for j in range(n-1, 0, -1):
        Est_Trie = True
        for i in range(j):
            if l[i] > l[i+1]:
                l[i], l[i+1] = l[i+1], l[i]
                Est_Trie = False
        if Est_Trie: break
```

③ Le Tri par insertion

- le concepte: on a une liste triée et on veut juste insérer n dans une liste triée. Et pour faire cela, c'est comme ci: on crée une case vide à la fin de la liste.



i : l'emplacement de n .

on compare n avec $l[j-1]$. Si $n < l[j-1]$:

on décale $l[j]$ vers la case i et on compare n avec $l[i]$.

de m^{ême} on décale $l[i]$ vers l'ancien emplacement de

$l[i]$ si $n < l[i]$. on s'arrête si $n > l[i]$.

Mais m^{ême}, on veut faire ça même si notre liste n'est pas triée en utilisant le m^{ême} concepte. Comment ??

On sait que la 1^{ère} case est toujours triée. Donc, on choisit d'arrêter à partir de la 2^{ème} case, et on fait la même chose au début avec i qui parcourt aussi la liste. Les conditions d'arrêts : $j = 0$ ou $l[j-1] \leq l[j]$.

Ecrivons le programme :

def Tri-Insertion(l) :

$n = \text{len}(l)$

for i in range(1, n) :

$j, u = i, l[i]$

while $j > 0$ and $l[j-1] > u$:

$l[j] = l[j-1]$

$j = j - 1$

$l[j] = u$

complexité (next lesson)

$\Theta(n)$

meilleur cas :
 $\Theta(n)$

le pire cas :
 $\Theta(n^2)$

Pour Tri-bulle avec la fct est triée :

meilleure $\leftarrow \Theta(n)$ $\rightarrow$ pire :

$\Theta(n)$: liste triée

$\Theta(n^2)$: liste décroissante

III - les algorithmes de Tri avancés :

basés sur le principe "Diviser pour régner"

Exercice 1 :

Ecrire la fct de fusion(A, B) qui prend 2 listes A et B supposées déjà triées, et renvoie leur fusion triée en temps linéaire de $\text{len}(A) + \text{len}(B)$ ($\Theta(n) = \Theta(n)$) où n est la taille de la liste.


```

def fusion(A, B):
    n, m = len(A), len(B)
    C = [0] * (n + m)
    i = j = 0
    for k in range(n + m):
        if j == m: C[k], i = A[i], i + 1
        elif i == n: C[k], j = B[j], j + 1
        elif A[i] <= B[j]: C[k], i = A[i], i + 1
        else: C[k], j = B[j], j + 1

```

→ meilleure méthode :

```

if j == m or (i < n and A[i] <= B[j]):
    C[k], i = A[i], i + 1
else: C[k], j = B[j], j + 1

```

④ Tri fusion :

- le concept : on a une liste, on la divise en 2, puis on fusionne les 2 moitiés, mais elles doivent être triées. Comment les trier ? on les divise encore en 2, jusqu'à ce qu'il ne reste qu'une seule case.

Ecrivons le programme : "fusionner $\ell(a : b + 1)$ et $\ell(b + 1 : c + 1)$ supposés déjà triés et mettre la fusion dans $\ell(a : c + 1)$ qui sera triée.

```

def fusionner(l, a, b, c):
    A = l[a : b + 1]
    i, j = 0, b + 1
    for k in range(a, c + 1):
        if j > c or (i <= b - a and A[i] <= l[j]):
            l[k], i = A[i], i + 1
        else: l[k], j = l[j], j + 1
    A.clear()

```

```
def Trifusion (l, debut = 0, fin = -1) :
```

```
    if fin == -1 : fin = len(l) - 1
```

```
    if debut < fin :
```

```
        m = (debut + fin) // 2
```

$O(n) = O(n \log_2(n))$ pour

```
        Trifusion (l, debut, m)
```

le meilleur et pour

```
        Trifusion (l, m + 1, fin)
```

le pire.

```
    fusionner (l, debut, m, fin)
```

② Tri rapide :

le concepte : on prend un élément au hasard de la liste & on call it le pivot. A la fin, on veut avoir :

< | pivot | >

Mais le pivot n'est pas le milieu.

Ecrivons un programme "partitionne $l[a : b + 1]$ en mettant le pivot choisi à sa position finale p , de telle sorte que $\forall x \in l[a : p] : x \leq \text{pivot}$.

et $\forall x \in l[p + 1 : b + 1] : x > \text{pivot}$.

Mais par convention, on choisit le pivot & le 1^{er} élément de la liste.

```
def partitionner (l, a, b) :
```

```
    p, pivot = a, l[a]
```

```
    for i in range (a + 1, b + 1) :
```

```
        if l[i] < pivot :
```

```
            p += 1
```

```
            l[p], l[i] = l[i], l[p]
```

```
    l[p], l[a] = l[a], l[p]
```

```
    return p
```

```
def Tri_Rapide (l, debut = 0, fin = -1) :
```

```
    if fin == -1 : fin = len(l) - 1
```

```
    p = partition (l, debut, fin)
```

```
    Tri_Rapide (l, debut, p - 1)
```

```
    Tri_Rapide (l, p + 1, fin)
```

43

la complexité des programmes

$$\Theta(n) = \Theta(f(n))$$

En temps Θ

Le nbr de répétitions
d'exécutions d'un bloc
d'instructions simples.

En espace Θ

Le nbr de cases
mémoires occupées par
le programme.

I. la complexité en temps

① En temps linéaire

$$\Theta(n) = \Theta(n) \Leftrightarrow \Theta(n) = a \cdot n + b$$

• Généralement :

une boucle simple à $\sim n$ itérations.

• Exemples :

• fact(n)

• puiss(n, m)

• somme(l) est en $\Theta(n)$ avec $n = \text{len}(l)$

• Recherche séquentielle de n ds l est :

→ $\Theta(1)$ pour le meilleur des cas ($n = l[0]$)

→ $\Theta(n)$ pour le pire ($n \notin l$)

② Quadratique

$$\Theta(n) = \Theta(n^2) \Leftrightarrow \Theta(n) = a \cdot n^2 + b \cdot n + c$$

• Généralement :

2 boucles imbriquées.

• Exemples :

• Parcours d'une matrice carrée d'ordre n

• Vérifier si tous les éléments d'une liste sont

2 à 2 distincts.

def Deux à Deux Distincts (L) :

$n = \text{len}(L)$

for i in range(n-1) :

for j in range(i+1, n) :

if $L[i] == L[j]$: return False

return True

$$E(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n = \Theta(n^2)$$

③ Polynomiale :

$$E(n) = \Theta(n^k) \quad \text{tq } k \geq 2$$

Exemples :

le produit de 2 matrices carrées est $\Theta(n^3)$

④ Logarithmique :

Généralement :

$$E(n) = \Theta(\log_2(n)) = \Theta(\ln(n))$$

$\Theta(\log_2(n))$ si pour chaque étape $n \leftarrow n/2$

Exemple :

Recherche dichotomique

Calcul de puissance * méthode (2)

def puiss(n, n) :

if $n == 0$: return 1

if $n \% 2 == 0$: return puiss($n/2$, $n/2$)

return n * puiss($n/2$, $n/2$)

⑤ En temps exponentielle :

$$E(n) = \Theta(k^n) \quad \text{tq } k \geq 2 \quad \text{Non accepté, sauf pour } n \text{ très b.}$$

Exemples :

la suite fibonnaci

$$U_0 = 1, U_1 = 1$$

$$U_n = U_{n-1} + U_{n-2} \quad \text{pour } n \geq 2$$

49

itérative

```
def fibo_iter(n):
    a, b = 1, 1
    for i in range(2, n+1):
        a, b = b, a+b
    return b
     $\Theta(n)$ 
```

récursive

```
def fibo_rec(n):
    if n < 2: return 1
    return fibo_rec(n-1) + fibo_rec(n-2)
     $\Theta(2^n)$ 
```

II La complexité en Espace

Exem de : somme(l)

la $\Theta(n)$	itérative	Réursive
avec $n = \text{len}(l)$	<pre>S = 0 for i in range(len(l)): S += l[i] return S</pre>	<pre>if l == []: return 0 return l[0] + somme(l[1:])</pre>
en temps	$\Theta(n)$	$\Theta(n)$
en Espace	$\Theta(n)$	$E(n) = \Theta(n^2)$

► Pour calculer $E(n)$ en espace, c'est comme si on calcule le nbr de case utilisé.

A savoir:

$\Theta(n^2)$ en temps est acceptable

$\Theta(n^2)$ en Espace est non acceptable.

on ne change pas la liste on joue sur le i.

→ Réduire la complexité en espace de fct récursive

```
def somme(l, i=0):
    # somme de l à partir de l[i]
    if i == len(l): return 0
    return l[i] + somme(l, i+1)
```

Exercices

- (1). Corriger Recherche (n, l)
- (2). Corriger Inverse (l)
- (3). Corriger Rech dich (n, l)

```
(1) def Recherche (n, l, i=0):
    if i == len(l): return False
    if l[i] == n: return True
    return Recherche(n, l, i+1)
```

```
(3) def inverse (l, i=0):
    if i == len(l): return []
    A = inverse(l, i+1)
    A.append(l[i])
    return A
```

⚠ Pour (3), c'est faux de faire `return inverse(l, i+1) + l[i]`
car on aura un problème de complexité en espace!
on aura trop de liste --

2) Si on fait `Rech_dich(n, l, i=0, len(l)-1)`
Au moment de l'affectation des valeurs, l'affectation
se fait en même temps car c'est un tuple.
L'utilisateur choisit (S/A)

donc $n \rightarrow n=5, l \rightarrow A, i \rightarrow 0, \text{len}(l)-1 \rightarrow \text{len}(l)-1$

Donc on utilise une fct auxiliaire.

non défini (30)

Méthode (1), avec fct auxiliaire;

Méthode (2) (sans fct auxiliaire)

```
def Rech_dich_Aux(n, l, i, j):
    if i > j: return False
    m = (i+j)//2
    if n == l[m]: return True
    if n < l[m]: return Rech_dich_Aux(n, l, i, m-1)
    return Rech_dich_Aux(n, l, m+1, j)
def Rech_dich(n, l):
    return Rech_dich_Aux(n, l, 0, len(l)-1)
```

```
def Rech_dich(n, l, i=0, j=len(l)-1):
    if j == -1 & j == len(l)-1:
    if i > j: return False
    m = (i+j)//2
    if n == l[m]: return True
    if n < l[m]: return Rech_dich(n, l, i, m-1)
    return Rech_dich(n, l, m+1, j)
```


les exceptions & Gestion des erreurs

while true

try :

$a = \text{float}(\text{input}('Saisir un réel '))$ $\Rightarrow$ Erreur au cas où on introduit une chaîne non convertible en float $\Rightarrow$ Valeur erreur

$b = \text{float}(\text{input}('Saisir un 2^{ème} réel '))$

$q = a/b$ $\Rightarrow$ Erreur si $b=0 \Rightarrow$ L'exception: Zero Division Error sera levée par le programme

print(a)

break

except ValueError

print("Vous devez saisir des nbres réels")

except ZeroDivisionError

print("Le 2^{ème} nbr doit être non nul")

$\Rightarrow$ Les erreurs qu'on a eues $\ell = \begin{bmatrix} -3 & -2 & -1 \\ 5 & 20 & -3 \\ 0 & 1 & 2 \end{bmatrix}$

• $\text{print}(\ell[1]) \Rightarrow 20$

• $\text{print}(\ell[5]) \Rightarrow$ IndexError / Key Error pour les dictionnaires.

• $\text{print}(\ell['a']) \Rightarrow$ TypeError

Try :

ID & interne/outside,

genre de fichier

n'existe pas.

$f = \text{open}(\text{nom}, 'r')$

except IOError

print("Erreur d'ouverture du fichier")

Exemple :

```
def Somme():
```

```
    try :
```

```
        f = open('c:\entiers.txt', 'r')
```

```
    except IOError:
```

```
        print("Fichier n'existe pas")
```

```
        return (genre to break)
```

```
    S = 0
```

```
    try :
```

```
        for ch in f:
```

```
            S += int(ch)
```

```
    return S
```

```
except ValueError:
```

```
    print("Une des lignes ne contient pas un entier")
```

```
finally :
```

```
    f.close()
```

Remq. - on a fait return S before close, because here, on a finally + les exceptions.

→ Comment lever une exception raise NomException(paramètre)

Exemple :

```
def fact(n):
```

```
    if not isinstance(n, int): raise TypeError()
```

```
    if n < 0: raise ValueError()
```

```
    p = 1
```

```
    for i in range(2, n+1):
```

```
        p *= i
```

```
    return p
```

```
l = [5, -3, 'salut', 20, 4, 3, 2]
```

```
for e in l:
```

```
    try :
```

```
        print(fact(i))
```

```
    except TypeError: print("cet élément n'est pas un entier")
```

```
    except ValueError: print("cet entier n'est pas négatif")
```

ici, on intercepte l'exception.
(si possible)

51

les arbres

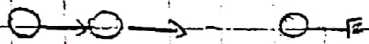
Il y a plusieurs types de structures de données :

→ Soit où les données sont regroupées dans un tableau (liste) $[1, 1, \dots]$

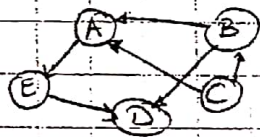
• Cas particuliers File et File

→ Soit où les données sont dans des listes :

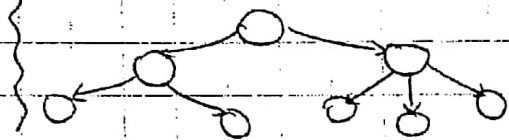
• Liste simplement chaînée :



• Graphique :



• Arbre :



les arbres

Un arbre est une structure de données composée d'éléments appelés nœuds tq :

- chaque nœud porte une valeur appelé sa clé
- chaque nœud peut avoir plusieurs fils et un seul parent
- les nœuds qui n'ont pas de fils sont appelés feuilles
- le nœud (unique) qui n'a pas de parent s'appelle la racine de l'arbre (batman go)
- Un arbre est référencé par sa racine
- chaque nœud fils est la racine d'un sous-arbre

⇒ Un arbre est une structure récursive :

• On distingue 2 types d'arbres :

Arbre binaire :

chaque nœud a au max
2 fils

Arbre n-aires :

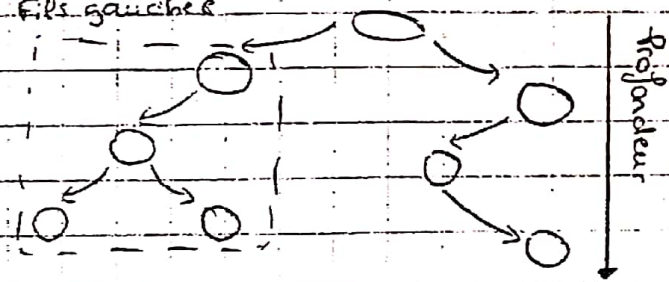
chaque nœud a au max
n fils tq $n > 2$

I. Les arbres binaires

⚠ Même si un nœud a un seul fils, on doit préciser s'il est droit ou gauche.

Fils gauches

arbre gauche



• La profondeur

nbr de niveaux

• chaque nœud est caractérisé par :

- sa valeur (sa clé)
- son fils gauche (FG)
- son fils droit (FD)

④ Représentation d'un arbre binaire

Ⓐ À l'aide d'une liste

Nœud = [sa clé, Fils G, Fils D] (chaque fils : une liste)

⇒ Arbre Binaire = AB = [clé de la racine, Arbre G, Arbre D]

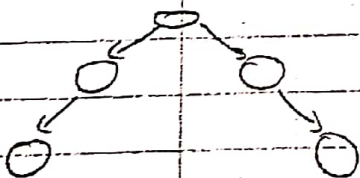
AG = arbre gauche

- $AB[0]$ = la valeur du nœud
- $AB[1]$ = AG = FG = liste de la même forme
- $AB[2]$ = AD = FD = liste de la même forme
- Arbre vide = []
- Feuille = [clé, [], []]

À savoir :

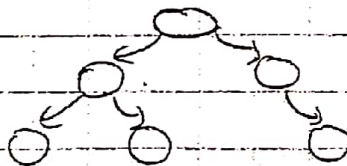
Arbre équilibré :

même nbr de niveaux et éléments



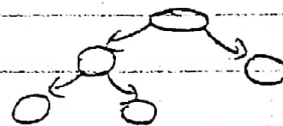
Arbre presque équilibré :

même nbr de niveaux



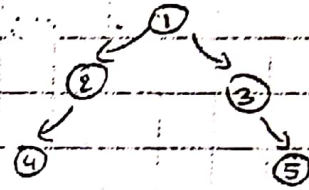
Arbre complet :

genre dichotomique complet à partir de la gauche.



Exemples

$AB = [1, [2, [4, [], []], []], [3, [], [5, [], []]]]$



ⓑ A l'aide d'un dictionnaire

$AB = \{ 'val': \text{clé de la racine}, 'G': AG = FG, 'D': AD = FD \}$

- $AB['val'] = \text{la valeur}$
- $AB['G'] = AG$: dictionnaire de la même forme
- $AB['D'] = AD$ " " "
- Arbre vide $\in \{ \}$: dictionnaire vide

• Exemple: c (L'arbre précédent)

$AB = \{ 'val': 1, 'G': \{ 'val': 2, 'G': \{ 'val': 4, 'G': \{ \}, 'D': \{ \{ \}, 'D': \{ \{ \} \} \}, 'D': \{ 'val': 3, 'G': \{ \}, 'D': \{ 'val': 5, 'G': \{ \}, 'D': \{ \{ \} \} \} \}$

ⓒ A l'aide des classes

Il faut 2 classes :

- classe Noeud dont les attributs sont :
 - key = sa clé (sa valeur)
 - $left$ = son fils gauche de classe noeud (None si $\emptyset$)
 - $right$ = son fils droit est de classe noeud (None si $\emptyset$)
- la classe Arbre dont les attributs sont :
 - $racine$ = de classe noeud (None si l'arbre est vide)

Classe Noeud :

```
def __init__(self, val, g=None, d=None):
```

```
    self.key = val
```

```
    self.left = g
```

```
    self.right = d.
```

2) Si l'arbre n'est pas vide, on enfile la racine :
file.append(Racine)

Tu peux faire inside.append ce que tu veux :
liste, éléments ou, et si on a par exemple A et L,
2 listes et tu fais A.append(L), la longueur de
A devient len(A) + 1, NOT len(A), len(L)
on ajoute à A la référence de L, dans une seule case.

3) Tant que la file est non vide :

- on définit un nœud : p = file.pop(0)
- on traite sa clé
- on enfile ses fils s'il y en a :

 - si fils gauche existe, on l'ajoute
 - si fils droit existe, on l'ajoute

Exemple : Affichage des clés selon le parcours en largeur :

def Afficher_largeur(AB) : # AB = [clé, G, D.]

file = []

if AB != [] : file.append(AB)

while file != [] :

p = file.pop(0)

print(p[0], end=' ')

if p[1] != [] : file.append(p[1])

if p[2] != [] : file.append(p[2])

Exercice :

(1) Refaire la fct en considérant qu'un arbre
est représenté par la classe Arbre

(2) la fct liste_largeur(AB) qui renvoie la liste
des clés selon le parcours en largeur

(3) Somme(AB)

(1) def Afficher (AB) :

file = []

if AB.racine != None : file.append (AB.racine)

while file != [] :

p = file.pop(0)

print (p.key, end = ' ')

if p.left :

if p.right :

(2) def Liste_Largeur (AB) :

L = []

file = []

if AB != [] : file.append (AB)

while file != [] :

p = file.pop(0)

L.append (p[0])

genre, if

if p[1] :

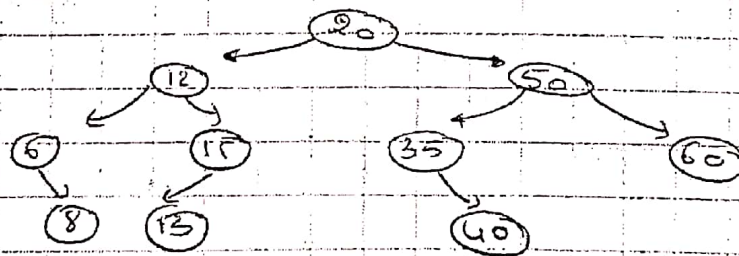
if p[2] :

return L

II - Arbre binaire de Recherche :

Pour chaque nœud : les valeurs de l'arbre gauche < sa clé.
les valeurs de l'arbre droit > sa clé.

Exemple :



Avantages :

- Si l'arbre est équilibré :
- la recherche se fait au pire en $O(\log_2(n))$
- De même, pour l'insertion d'un nouveau élément (sous forme d'une feuille)

Exercice 8

ABR est un arbre binaire de recherche, représenté à l'aide d'une liste [clé, G, D]

- (1) Écrire la fct Recherche (n, ABR)
- (2) Écrire la procédure Insérer (n, ABR) qui insère n dans ABR. cette procédure doit modifier l'arbre (pas sa référence).
- (3) La fct insertion (n, ABR) qui renvoie l'arbre résultant de l'insertion de n dans l'arbre.

```
(1) def Recherche (n, ABR) :  
    if ABR == [] : return False  
    if n == ABR[0] : return True  
    if n < ABR[0] : return Recherche (n, ABR[1])  
    return Recherche (n, ABR[2])
```

```
(2) def Insérer (n, ABR) :  
    if ABR == [] :  
        ABR.extend ([n, [], []])  
    elif n < ABR[0] : insérer (n, ABR[1])  
    elif n > ABR[0] : insérer (n, ABR[2])
```


(3) def insertion(n, ABR) :

if ABR == [] : return [n, [], []]

if n == ABR[0] : return ABR

if n < ABR[0] : return [ABR[0], insertion(n, ABR[1], ABR[2])]

return [ABR[0], ABR[1], insertion(n, ABR[2])]

(4) → Ecrire la fct Liste triée (ABR) qui renvoie une liste triée des clés de ABR

def Liste triée(ABR) :

if ABR == [] : return []

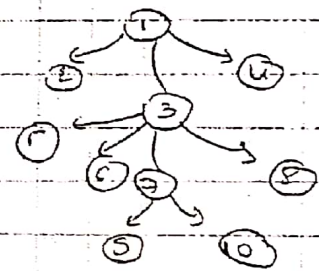
return Liste triée(ABR[1]) + [ABR[0]] + Liste triée(ABR[2])

III - Les arbres n-aires :

chaque noeud est caractérisé par :

• sa clé

• la liste de ses fils



Arbre 4 aires.

① Représentation :

Ⓐ par classe :

Class Noeud :

def init_(self, n) :

self.valeur = n

self.fils = []

Class Arbre :

def init_(self, R=None) :

self.racine = R

→ Exemple :

R = Noeud(1)

R.fils = [Noeud(2), Noeud(3), Noeud(4)]

l = R.fils

l[1].fils = [Noeud(5), Noeud(6), Noeud(7), Noeud(8)]

l = l[1].fils

l[2].fils = [Noeud(9), Noeud(10)]

A = Arbre(R)

② Par une liste :

Forme (1) :

$A = [clé, [fils_1, \dots, fils_n]]$

- $A[0]$ est la clé
- $A[1]$ est la liste des fils
- Feuille = $[n, []]$

Forme (2) :

$A = [clé, fils_1, \dots, fils_n]$

- $A[0] = \text{la clé}$
- $A[i]$ avec $i \geq 1$ est fils
- $A[1:]$ la liste des fils
- Feuille = $[n]$

Exemple (forme (1)) :

$A = [1, [[2, []], [3, [[5, []], [6, []], [7, [[8, []], [9, []], [10, []], [11, []], [12, []]]]]]$

Exemple (forme (2)) :

$A = [1, [2], [3, [5], [6], [7, [8], [9], [10], [11], [12]]], [4]]$

② Le parcours :

① En largeur & niveau par niveau

② En profondeur :

- Préfixé & clé $\rightarrow$ les fils
- Post fixé & les fils $\rightarrow$ la clé

Exercice : Arbre n-aires

Soit A un arbre n-aires dont les clés sont des nbres. Écrire les fonctions :

1. Afficher largeur (A) qui affiche les valeurs (clés) selon le parcours en largeur
2. Afficher Préfixé (A) qui affiche les valeurs (clés) selon le parcours en profondeur préfixé
3. Somme (A) qui renvoie la somme des clés
4. Nbr Noeuds (A) qui renvoie le nbr de noeuds
5. Profondeur (A) qui renvoie la profondeur de l'arbre (nbr de niveaux)

6. Nbre Airc(A) qui renvoie le nbre d'airs.
Pour les cas suivants :

- A est représenté par une liste de forme (1).
- A est représenté par une liste de forme (2).
- A est de classe Arbre (forme (3)).

Forme (1)

def Afficher_longueur(A) :

file = []

if A != [] : file.append(A)

while file != [] :

P = file.pop()

print(P[0], end = '\t')

for f in P[1:] :

file.append(f)

Forme (2) :

(la seule différence est qu'on
écrit : for f in P[1:] :

Forme (3) :

def Affichage_longueur(A) :

file = []

if A.racine != None : file.append(A, racine)

while file != [] :

P = file.pop()

print(P.racine, end = '\t')

for f in P.fils :

file.append(f)

Forme (1) :

def Afficher_Prefixe(A) :

if A != [] :

print(A[0], end = '\t')

for n in A[1:] :

Afficher_Prefixe(n)

Forme (2) :

def Afficher_Prefixe(A) :

remplace A[1] par A[1:] :

Forme (3)

def Afficher_Prefixe(A) :

if A.racine != None :

print(A.racine, end = '\t')

for f in A.racine.fils :

Afficher_Prefixe(A.fils(f))

3/ # Méthode (1)

→ Forme (1):

def Somme(A):

if A == []: return 0

S = A[0]

for u in A[1]:

return S

→ Forme (2): tu fais A[1:]

4/ # Méthode (1)

def Nbr Nœuds(A):

if A == []: return 0

n = 1

for f in A[1]:

n += Nbr Nœuds(f)

return n

Méthode (2)

def Somme(A):

S = 0

file = []

if A != []: file.append(A)

while file != []:

P = file.pop(0)

S += P[0]

for f in P[1]:

file.append(f)

Méthode (2)

tu fais comme méthode (2)

en haut, mais au lieu de

faire $S += P[0]$, tu fais

$S += 1$

5/ def profondeur(A):

if A == []: return 0

if A[1] == []: return 1

return 1 + max([profondeur(f) for f in A[1]])

6/ def Nbrs Aircs(n):

if A == []: return None

return max([len(A[1]) + 1

Nbr Aircs(f) for f in A[1]])

Exercice (1) :

Soit AB un arbre binaire. Écrire les fcts :

- (1) Afficher Préfixe qui affiche les valeurs selon le parcours en profondeur préfixé
- (2) Afficher Infixe " " " infixe
- (3) Afficher postfixe " " " postfixe

(1) def Aff-Préfixe (AB) :

if AB != [] :

print (AB[0], end = ' ')

Aff-Préfixe (AB[1])

Aff-Préfixe (AB[2])

(2) def Aff-Infixe (AB) :

if AB != [] :

Aff-Infixe (AB[1])

print (AB[0], end = ' ')

Aff-Infixe (AB[2])

(3) def Aff-Postfixe (AB) :

if AB != [] :

Aff-Postfixe (AB[1])

Aff-Postfixe (AB[2])

print (AB[0], end = ' ')

Exercice 2 :

Soit AB un arbre binaire. Écrire les fcts :

1. Liste-Préfixe (AB) qui affiche les éléments de AB avec un parcours préfixe
2. Liste-Infixe (AB) " " " parcours infixe


```

1 def Liste_Prefixe(AB) e
  if AB == [] : return []
  return [AB[0]] + Liste_Prefixe(AB[1]) + Liste_Prefixe(AB[2])

```

```

2 def Liste_Infixe(AB) e
  if AB == [] : return []
  return Liste_Infixe(AB[1]) + [AB[0]] + Liste_Infixe(AB[2])

```

△ Et si on travaille avec AB comme étant un dictionnaire e.g. val : dé, 'G' : AG, 'D' : AD, nous en retourne dictchys ds une liste ?

```

def Liste_Prefixe(AB) e
  if AB == {} : return []
  return [AB[val]] + Liste_Prefixe(AB['G']) + Liste_Prefixe(AB['D'])

```

△ Et si AB est une classe ?

```

def Liste_Prefixe(AB) e
  if AB.racine == None : return []
  return [AB.racine.key] +
    Liste_Prefixe(Arbre(AB.racine.left)) +
    Liste_Prefixe(Arbre(AB.racine.right))

```

Exercice 8

Soit AB un arbre binaire. Ecrire les fctrs

1. Somme(AB) qui calcule la somme des valeurs de l'arbre
2. Nbr Noeuds(AB) qui calcule le nbr de noeuds de l'arbre
3. Profondeur(AB) qui calcule la profondeur de l'arbre

1 Recherche(x, AB) qui retourne True si $x \in AB$,
False sinon

4 def Somme(AB) :
if $AB == []$: return 0
return $AB[0] + \text{Somme}(AB[1:]) + \text{Somme}(AB[2:])$

5 Si on travaille avec AB comme classe ?

def Somme(AB) : # AB classe
if $AB.\text{racine} == None$: return 0
return $AB.\text{racine}.\text{key} + \text{Somme}(\text{Arbre}(AB.\text{racine}.\text{left}))$
+ $\text{Somme}(\text{Arbre}(AB.\text{racine}.\text{right}))$

2 def NbrNoeuds(AB) :
if $AB == []$: return 0
return $1 + \text{NbrNoeuds}(AB[1:]) + \text{NbrNoeuds}(AB[2:])$

3 def Profondeur(AB) :
if $AB == []$: return 0
return $1 + \max(\text{Profondeur}(AB[1:]), \text{Profondeur}(AB[2:]))$
 $\Theta(n) = \Theta(n)$ car il répète l'opération n fois.

1 def Recherche(x, AB) :
if $AB == []$: return False
if $x == AB[0]$: return True
return Recherche($x, AB[1:]$) or Recherche($x, AB[2:]$)

Au pire :

$x \notin AB \Rightarrow \Theta(n)$

Au meilleur :

si $x \in AB$ et x est la clé de la racine $\Rightarrow \Theta(1)$

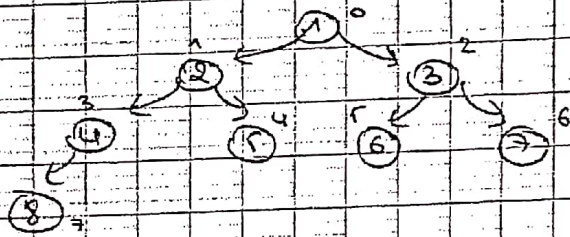
* 2 opérations simples sont considérées être 1 seule.

Le Tri par Tas

Le concept en gros (résumé du document)

Alors tu veux trier la liste, tu la considère comme un arbre binaire, ainsi, si tu as :

$L = [4, 2, 3, 1, 5, 6, 7, 8]$, et elle devient :



ainsi, un nœud est référencé par son indice i

tg & sa clé & $L[i]$

fil G & $2i + 1$

fil D & $2i + 2$

Do le doc, ils ont

fil G & est 2n car ils ont

commencé l'indexation de la liste à partir de 1, noté

* i l'indice du nœud de départ à tamiser

n l'indice à ne pas dépasser

Ainsi, pour trier la liste, on va tamiser

puis inverser la valeur de l'indice avec la dernière valeur

def Tamiser (Arbre, i, n) :

$j = 2 * i + 1$ # FG

while $j \leq n$

if $j < n$ and $Arbre[j+1] > Arbre[j]$:

$j = j + 1$

if $Arbre[j] > Arbre[i]$:

$Arbre[i], Arbre[j] = Arbre[j], Arbre[i]$

$i = j$

$j = 2 * i + 1$

else break

def Tri par Tas (L) :

$n = \text{len}(L)$

for i in range($n//2 - 1, -1, -1$) :

Tamiser(L, $i, n - 1$)

for i in range($n - 1, 0, -1$) :

$L[0], L[i] = L[i], L[0]$

Tamiser(L, 0, $i - 1$)

Tri par tas

Une exécution de l'algorithme du tri par tas (Heapsort) trie une partie des valeurs permutées au hasard. Dans un premier temps, les éléments sont réarrangés pour respecter les conditions de tas. Avant le tri à proprement parler, la structure de l'arbre en tas est montrée brièvement par l'illustration.

Le tri par tas est un algorithme de tri par comparaisons. Cet algorithme est de complexité asymptotiquement optimale, c'est-à-dire que l'on démontre qu'aucun algorithme de tri par comparaison ne peut avoir de complexité asymptotiquement meilleure. Sa complexité est proportionnelle à $n \log n$ où n est la longueur du tableau à trier. Le tri par tas se fait *en place*, c'est-à-dire qu'il ne nécessite pas l'allocation d'une zone mémoire supplémentaire.

Son inconvénient majeur est sa lenteur comparée au tri rapide (qui est en moyenne deux fois plus rapide) : sur un tableau de taille importante, il sera amené à traiter un nombre élevé d'emplacements mémoire dont l'éloignement peut dépasser la capacité du cache, ce qui ralentit l'accès à la mémoire et l'exécution de l'algorithme.

Principe

L'idée qui sous-tend cet algorithme consiste à voir le tableau comme un arbre binaire. Le premier élément est la racine, le deuxième et le troisième sont les deux descendants du premier élément, etc. Ainsi le n^e élément a pour enfants les éléments $2n$ et $2n + 1$. Si le tableau n'est pas de taille $(2^n - 1)$, les branches ne se finissent pas toutes à la même profondeur.

Dans l'algorithme, on cherche à obtenir un tas, c'est-à-dire un arbre binaire vérifiant les propriétés suivantes (les deux premières propriétés découlent de la manière dont on considère les éléments du tableau) :

- la différence maximale de profondeur entre deux feuilles est de 1 (i.e. toutes les feuilles se trouvent sur la dernière ou sur l'avant-dernière ligne) ;
- les feuilles de profondeur maximale sont « tassées » sur la gauche.
- chaque nœud est de valeur supérieure (resp. inférieure) à celles de ses deux fils, pour un tri ascendant (resp. descendant).

Il en découle que la racine du tas (le premier élément) contient la valeur maximale (resp. minimale) de l'arbre. Le tri est fondé sur cette propriété.

Comme expliqué plus haut, un tas ou un arbre binaire presque complet peut être stocké dans un tableau, en posant que les deux descendants de l'élément d'indice n sont les éléments d'indices $2n$ et $2n + 1$ (pour un tableau indexé à partir de 1). En d'autres termes, les nœuds de l'arbre sont placés dans le tableau ligne par ligne, chaque ligne étant décrite de gauche à droite. Pour la suite, nous considérons que l'on trie par ordre croissant.

L'opération de base de ce tri est le tamissage, ou percolation, d'un élément, supposé le seul « mal placé » dans un arbre qui est presque un tas. Plus précisément, considérons un arbre $A = A[1]$ dont les deux sous-arbres ($A[2]$ et $A[3]$) sont des tas, tandis que la racine est éventuellement plus petite que ses fils. L'opération de tamissage consiste à échanger la racine avec le plus grand de ses fils, et ainsi de suite récursivement jusqu'à ce qu'elle soit à sa place. Pour construire un tas à partir d'un arbre quelconque, on tamise les racines de chaque sous-tas, de bas en haut et de droite à gauche. On commence donc par les sous-arbres « élémentaires » — contenant deux ou trois nœuds, donc situés en bas de l'arbre. La racine de ce tas

est donc la valeur maximale du tableau. Puis on échange la racine avec le dernier élément du tableau, et on restreint le tas en ne touchant plus au dernier élément, c'est-à-dire à l'ancienne racine ; on a donc ainsi placé la valeur la plus haute en fin de tableau (donc à sa place), et l'on n'y touche plus. Puis on tamise la racine pour créer de nouveau un tas, et on répète l'opération sur le tas restreint jusqu'à l'avoir vidé et remplacé par un tableau trié.

Pseudo-code

On fait l'hypothèse que l'arbre est un tableau indexé entre 1 et longueur. $\text{arbre}[i]$ désigne le i -ème élément de ce tableau.

```

fonction tamiser(arbre, nœud, n) :
  (* descend arbre[nœud] à sa place, sans dépasser l'indice n *)
  k := nœud
  j := 2k
  tant que j ≤ n
    si j < n et arbre[j] < arbre[j+1]
      j := j+1
    fin si
    si arbre[k] < arbre[j]
      échanger arbre[k] et arbre[j]
      k := j
      j := 2k
    sinon
      terminer
    fin si
  fin tant que
fin fonction

fonction triParTas(arbre, longueur) :
  pour i := longueur/2 à 1
    tamiser(arbre, i, longueur)
  fin pour
  pour i := longueur à 2
    échanger arbre[i] et arbre[1]
    tamiser(arbre, 1, i-1)
  fin pour
fin fonction

```

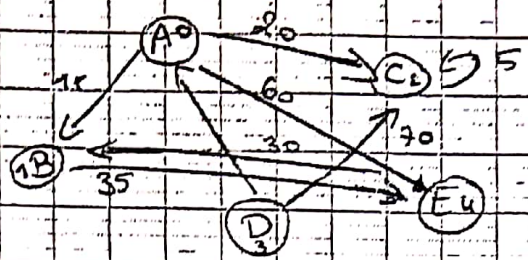
À la fin de la fonction triParTas le tableau arbre est trié suivant l'ordre croissant. Il suffit d'inverser les opérateurs de comparaison pour obtenir un tri dans l'ordre décroissant.

Analyse

Cet algorithme permet de trier sur place les éléments d'un tableau en un temps de l'ordre de $n \log_2 n$, où n est le nombre d'éléments à trier. La complexité entre le meilleur des cas et le pire des cas ne varie que d'un facteur constant¹. L'étape la plus coûteuse de l'algorithme est la seconde boucle, c'est-à-dire l'extraction des éléments du tas. La première étape, consistant à construire le tas, est effectuée en temps linéaire en n .

Les principaux atouts de cette méthode sont la faible consommation mémoire et l'efficacité, optimale étant donné qu'on ne fait aucune hypothèse sur la nature des données à trier.

Les graphes



Un graphe $G = (S, A)$ tqe

S est l'ensemble des sommets (ensemble ou liste non ordonné)
 A est l'ensemble des liens

les sommets possèdent des identifiants

on a 2 types de graphes

- graphe orienté lien = arc (tuple)
- graphe non orienté lien = arête (ensemble tuple liste)

on a 2 types de graphes (again)

- Pondéré à chaque lien de A , on associe une valeur appelé son poids.

→ A est une liste et poids est une liste de même taille que A tqe

$\text{poids}[i]$ est la valeur associée à $A[i]$

- Non pondéré les liens n'ont pas de poids

④ la représentation (S, A)

$S = [A, B, C, D, E]$

$A = [A, B], [A, C], [A, E], [B, E], [C, D], [C, D], [A, D], [E, B]$

ou

$A = [0, 1], [0, 2], [0, 4], [1, 4], [2, 2], [2, 3], [0, 3], [4, 1]$

$G = (S, A)$ ou $G = (S, A)$

pois = [15, 20, 50, 35, 50, 50, 30]

② la représentation par la matrice d'adjacence

n = le nbr de sommets.

A chaque sommet, on associe un indice unique $\{1, \dots, n\}$

n est la matrice adjacence $n \times n$

• n matrice carrée.

• $n_{ij} = 1$ si un lien existe de i à j } non pondéré

• $n_{ij} = 0$ sinon

• n_{ij} = le poids s'il y a un lien de i à j } Pondéré

• $n_{ij} = 0$ sinon

Ainsi :

	0	1	2	3	4
0	None	15	20	50	60
1	None	None	None	None	35
2	None	None	5	30	None
3	None	None	None	None	None
4	None	30	None	None	None

⇒ Remq :

Si le graphe est non orienté, la matrice d'adjacence sera symétrique.

Exercice :

Soit G un graphe représenté à l'aide d'un tuple (S, A) , avec A est une liste de liens, chacun est de la forme d'une liste $[i, j]$, qui représente un lien du sommet $S[i]$ à $S[j]$ (i et j sont des indices

de sommets). Les poids sont stockés dans une liste
 poids de e poids (k) et le poids du lien $A(k)$
 → Écrire la fct Matrice adjacence (G , poids)
 qui renvoie la matrice d'adjacence. On suppose que
 le graphe est non orienté.

② Quand le graphe est non orienté, A n'a pas de liens
 double, genre on n'a pas la liaison $[A, B]$ et $[B, A]$,
 juste $[A, B]$

def Matrice adjacence (G , poids):

$s, A = G$

$n = \text{len}(s)$

$M = [None] * n$ for i in range(n)

for k in range($\text{len}(A)$):

$(i, j) = A[k]$

$M[i][j] = M[j][i] = \text{poids}(k)$

return M

⇒ $B(n) = \Theta(\text{len}(A))$

③ Représentation par les listes adjacences

- À chaque sommet, on associe la liste de ses voisins et
 1^{er} cas : graphe non pondéré

• Sommet = son indice dans s

→ Représentation du graphe sous forme d'une liste :

$G = [[1, 2, 3, 4], [4], [2, 3], [], [1]]$

$G[i]$ est la liste d'adjacence du sommet i .

• Sommet = chaîne de caractères ⇒ G = dictionnaire

$$G = \{A' \in \{B', C', D', E'\}, B' \in \{E'\}, C' \in \{C', D'\}, D' \in \{C'\}, E' \in \{B'\}\}$$

2^{ème} cas : Graphique pondéré

• Sommet = indice

$$G = [(1, 1), (2, 2), (3, 1), (4, 6)], [(1, 3), (2, 5), (3, 7), (4, 3)]$$

$G[i]$ = liste d'adjacence de i

$\rightarrow G[i][j][0]$ = voisin

$G[i][j][1]$ = poids

• Sommet = chaîne

$$G = \{A' \in \{B', C', D', E'\}, B' \in \{E'\}, C' \in \{C', D'\}, D' \in \{C'\}, E' \in \{B'\}\}$$

$$G[A][B] = \text{poids du lien } [A, B]$$

Algorithme de Dijkstra

DD Check la description sur Wikipedia, le concept est très bien expliqué

Cas (1) : $G = (S, A)$

le graphe $G = (S, A)$ tq :

So l'ensemble des sommets identifiés par des chaînes de caractères

A : liste des liens, chaque est un tuple (s_1, s_2) représente

le lien de S_1 vers S_2 (graphe orienté), ou représente les liens de S_1 à S_2 et de S_2 à S_1 (graphe non orienté).
 poids = liste de même taille que A , tel que $\text{poids}[k]$ est le poids associé au lien $A[k]$.

→ les structures à utiliser :

- P regroupe les sommets déjà sélectionnés
 ⇒ sommet exclus

- Q regroupe les sommets déjà mis à jour leurs distance et qu'on va en sélectionner le sommet ayant la plus petite distance.

(file à priorité)

- d associe aux sommets leurs distances

- prédecesseurs associe à chaque sommet son prédecesseur dans le plus court chemin, à partir du sommet de départ

- $\text{def voisins}(n, G, \text{poids})$: voisins du sommet n et z
 $S, A = G$

$\text{return} [(A[k][1], \text{poids}[k])] \text{ for } k \text{ in range}(\text{len}(A)) \text{ if } A[k][0] == n]$

- $\text{def Trouve_min}(Q, d)$: Q ensemble / dictionnaire

$S_{\min} = \text{None}$

for x in Q :

if $S_{\min} = \text{None}$ or $d[x] < d[S_{\min}]$:

$S_{\min} = x$

return $S_{\min}$


```

def Dijkstra (G, poids, Sdeb, Sfin) :
    P = set() # ensemble
    Q = {Sdeb} # ensemble
    d = {Sdeb: 0} # dict
    predecesseur = {} # dict
    while Q != set() :
        x = trouve_min(Q, d)
        if x == Sfin : break
        Q.remove(x)
        if x not in P :
            P.add(x)
            for y, pxy in voisins(x, G, poids) : # y voisin ; pxy = poids(x, y)
                if y not in P :
                    dy = d[x] + pxy
                    if y not in d or dy < d[y] :
                        d[y] = dy
                        Q.add(y)
                        predecesseur[y] = x
            if Sfin not in predecesseur : return None # pas de chemin de Sdeb à Sfin
    chemin = [Sfin]
    x = Sfin
    while x != Sdeb :
        chemin.append(x)
        x = predecesseur[x]
    chemin.reverse() # cuz dirch y sort genre [J, H, C, A] alors
    return chemin, d[Sfin] # que we want [A, C, H, J]

```


2^{ème} cas : G = Matrice d'adjacence

le sommet est identifié par un entier $\in [0, \text{len}(G)-1]$

def voisins(n, G) :

return [y for y in range($\text{len}(G)$) if $G[n][y] \neq \text{None}$]

def trouve_min(Q, d) :

$S_{\min} = Q[0]$

for n in Q :

if $d[n] < d[S_{\min}]$: $S_{\min} = n$

return $S_{\min}$

def Dijkstra($G, S_{\text{deb}}, S_{\text{fin}}$) :

$n = \text{len}(G)$

$P, Q = [], [S_{\text{deb}}]$

$d = [\text{None}] * n \neq \text{None} \Leftrightarrow \text{infini}$

$d[S_{\text{deb}}] = 0$

prédecesseur = $[\text{None}] * n \neq \text{None} \Leftrightarrow$ pas de précédent

while $Q \neq []$:

$u = \text{trouve_min}(Q, d)$

if $u == S_{\text{fin}}$: break

$Q.\text{remove}(u)$

$P.\text{append}(u)$

for y in voisins(u, G) : # y = voisin, poids(u, y) = $G[u][y]$

if y not in P :

$d_y = d[u] + G[u][y]$

if $d[y] == \text{None}$ or $d_y < d[y]$:

$d[y] = d_y$

if y not in Q : $Q.\text{append}(y)$

prédecesseur[y] = u

if prédecesseur[S_{fin}] == None : return None

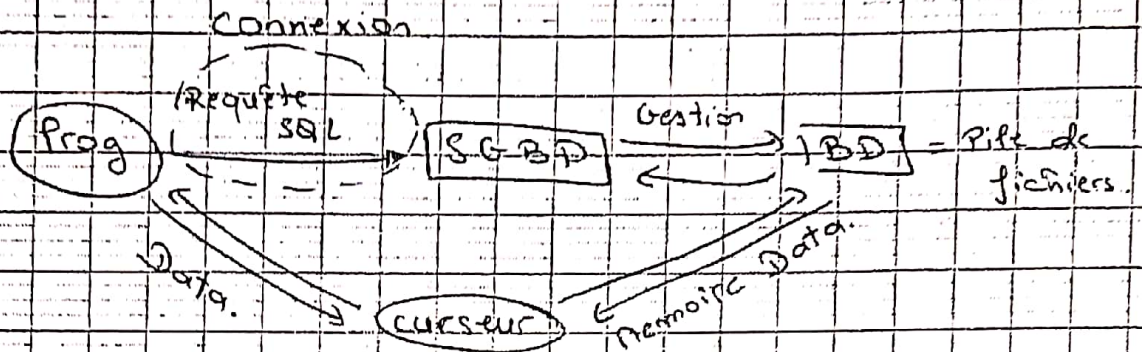
chemin = [S_{fin}]


```
n = Sfin
while n != Sdeb e
    chemin.append(n)
    n = predecesseur(n)
chemin.reverse()
return chemin, d[Sfin]
```


Les bases de données

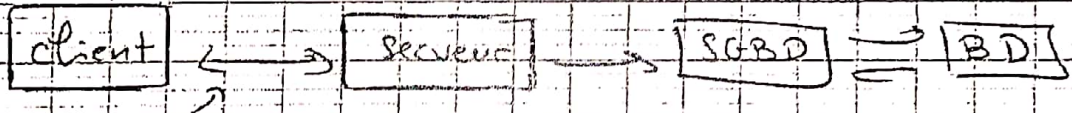
SQL = Structure Query Language = langage de requête structure

→ Architecture 2 tiers :



SGBD = système de Gestion de Bases de données

→ Architecture 3 tiers :



double flèche car le serveur peut répondre au client.

→ Les SGBD :

N°1 = MS Access, Sqlite & La BD est un simple fichier local.
→ intégré en python.

N°2 = mysql, PostgreSQL

→ Gratuit, le plus utilisé sur internet

N°3 = Oracle, MS SQL server & plus puissant professionnel.

Modèle de données

logique ou conceptuel : Données telles qu'elles sont en pratique

physique : Données telles qu'elles seront ds la BD :

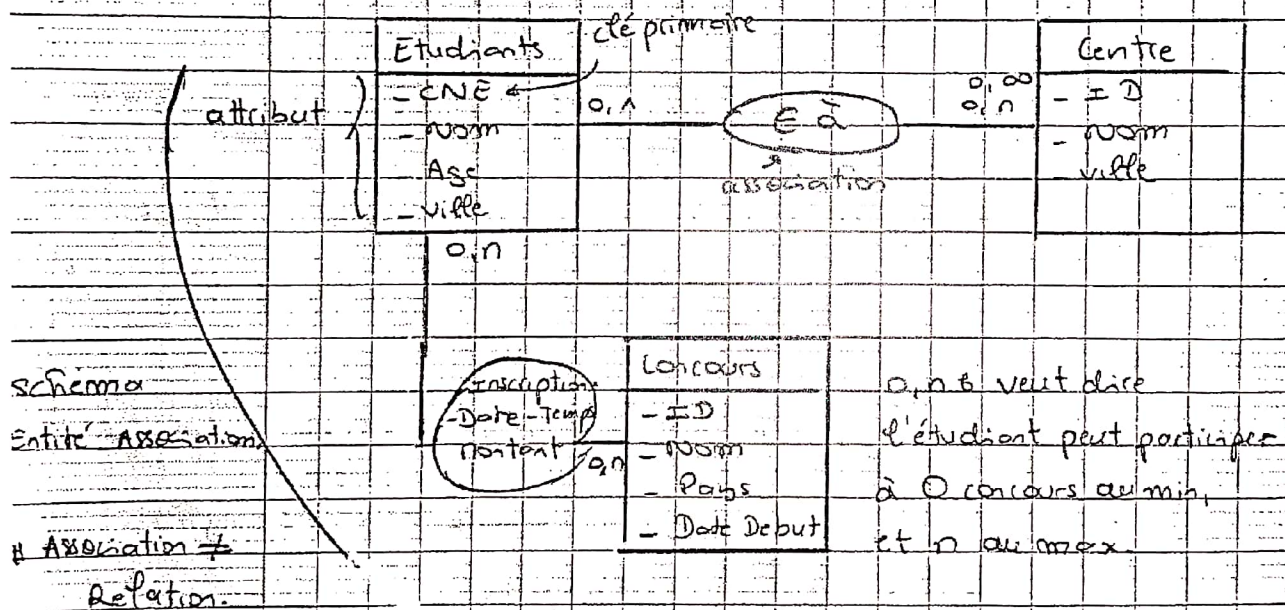
• Réarchivé : le plus ancien.

• Relationnel : le plus utilisé, basé sur les notions de maths relationnel.

→ Passage de modèle logique NCD au modèle physique relationnel

Exemple : Gestion des étudiants leur centre et leurs inscriptions aux concours

Donc 3 entités : Etudiant, Centre, concours



• le domaine de l'attribut c'est son type :

(entier, réel, caractère, chaîne de caractère, Texte, Date, time, ...)

Le modèle relationnel :

- La BD est constituée de tables ; chaque entité devient une table ou relation.

- Attributs $\rightarrow$ champ ou colonne de la table.

$\Rightarrow$ Les tables de notre exemple :

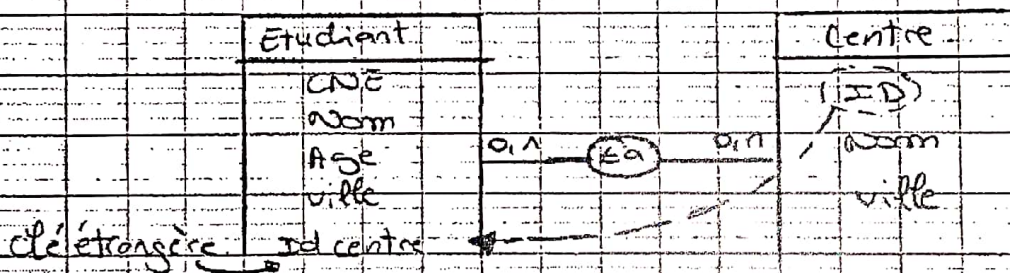
Centre		
ID	Nom	Ville
—	—	—

• Centre (ID, Nom, ville)

• Etudiant (CNE, Nom, Age, ville)

• Concours (ID, Nom, Pays, Date, Debut)

⚠ Trais : il n'y a pas de lien entre Etudiant et centre
Que faire ?? ☹



On a un centre à plusieurs élèves, donc ID centre va migrer vers Etudiant.

Donc, la table devient :

Etudiant (CNE, Nom, Age, ville, # ID centre)

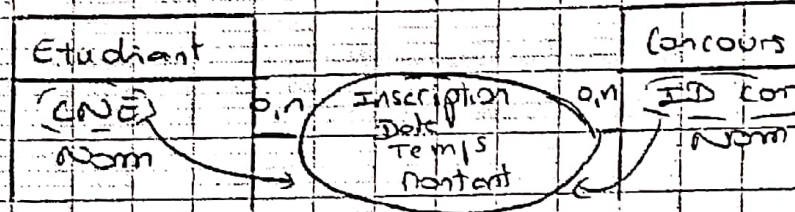
$\nearrow$ c'est un champ, mais pas un attribut.

⚠ Attribut $\Rightarrow$ champ Réciproque Fausse !!

A savoir :

En général (souvent, # it's not the case), on a la clé primaire et on # la clé étrangère

16) Mais, what about le lien entre concours et Etudiant? ils sont tous les 2 D.O, what to do? (??)
ils vont se rencontrer dans l'association



et on ajoute une table :

Inscription (Montant, Date temps, # CNE, # IDc)

→ Donc association n à n devient une table

16) Attention :

le champ ne peut être qu'une seule valeur
et cannot de une liste. Mais il peut être
float, integer, autre

→ Langage SQL :

Type de données	mysql	SQLite
Entier	int, integer	integer
Réel	float, Real	Real
caractère	CHAR	CHAR
chaîne de caractères	Varchar (n)	CHAR(n)
Texte	Text, tinyText, longText	TEXT
Date	DATE	?
Temps	TIME	—
Date et temps	DATE TIME	—

A savoir

on écrit Date like this
'aaaa-mm-jj'
on écrit time like this
'hh:mm'

① Crée une table :

CREATE TABLE Centre (

ID INTEGER PRIMARY KEY NOT NULL,

Nom CHAR(50) ^{genre au max 50 caractères},

Ville CHAR(30);

CREATE TABLE Etudiant (

CNE CHAR(10) PRIMARY KEY NOT NULL,

Nom TEXT,

Age INTEGER,

Ville CHAR(30),

ID Centre INTEGER

FOREIGN KEY (ID Centre) REFERENCES Centre (ID))

CREATE TABLE Inscription (

Date

Montant

CNE Etudiant CHAR(10) NOT NULL,

ID Concom NOT NULL,

FOREIGN KEY (CNE)

FOREIGN KEY (ID Concom)

PRIMARY KEY (CNE, ID Concom));

⚠ Si on nous dit de créer la table si elle n'existe pas,
que faire ? 🤔

→ CREATE TABLE IF NOT EXISTS Centre (

Centre		
ID	Nom	Ville

Ds chaque table

il y a → enregistrement
on dit à insérer un enregistrement.

Insertion → ② Ajouter un enregistrement dans une table

```
INSERT INTO Centre  
VALUES (123, 'NY', 'Rabat')
```

③ Modifier un (des) champ(s) d'un (des)
enregistrement(s) & Mise à jour

```
UPDATE Centre  
SET Nom = 'Douglas Young', ville = 'Rabat'  
WHERE ID = 123
```

④ Suppression d'enregistrements(s) d'une table

```
DELETE FROM Centre  
WHERE UPPER(ville) = 'RABAT'
```

genre on supprime everything UPPER to avoid problems.

⑤ Si je veux supprimer 2 villes?

Tu fais WHERE ville = 'Rabat' or ville = 'Fes'

⑥ Interroger la BD & extraction de données &
consultation

```
SELECT liste de champs  
FROM liste de table  
WHERE condition
```


Exemple (1) : Donner tous les champs de tous les élèves

SELECT * -- veut dire everything } la relation étudiant,
FROM Etudiant; -- c'est son contenu.

Exemple (2) : Donner le nom et l'âge de tous les étudiants

SELECT nom, Age } Projection
FROM Etudiant; -- $\pi_{nom, Age}(Etudiant)$

Exemple (3) : Donner tous les champs des étudiants
âgés de moins de 20 ans, strictement.

SELECT * } Sélection
FROM Etudiant } $\sigma_{Age < 20}(Etudiant)$
WHERE Age < 20

Exemple (4) : Donner le nom et la ville de tous les
étudiants âgés de moins de 20 ans.

SELECT nom, ville } Projection d'une sélection :
FROM Etudiant } $\pi_{nom, ville}(\sigma_{Age < 20}(Etudiant))$
WHERE Age < 20;

La table étudiant est une relation

son contenu...

son schéma :

CNE, nom, Age, ville

Etudiants : $\{(E_1, N_1, 15, V_1, C_1),$
 $(E_2, N_2, 21, V_2, C_2)\}$

projection

$\pi_{nom, Age}(Etudiant) = \{(N_1, 15), (N_2, 21), (N_3, 18)\}$

$\sigma_{Age < 20}(Etudiant) = \{(E_1, N_1, 15, V_1, C_1), (E_3, N_3, 18, V_3, C_3)\}$

check
page suivante,
le tableau
pour
comprendre.

CNE	Nom	Age	ville	IDcentre
E1	N1	19	V1	C1
E2	N2	21	V2	C2
E3	N3	18	V3	C3

Exemple (r) : Tous les éléments (Nom, ville)
ainsi que le nom du centre de chaque
étudiant.

⇒ C'est FAUX de faire :

```
SELECT E.Nom, E.ville, C.Nom,
FROM Etudiant, Centre ← produit cartésien
WHERE Etudiant.IDcentre = Centre.ID
```

→ we don't like produit cartésien (∞)

⇒ Il est recommandé de faire une jointure de
2 tables :

$$A \bowtie B = \{ (a, b, c) / (a, b) \in A, (b, c) \in B \}$$

```
SELECT E.Nom, E.ville, C.Nom
FROM Etudiant E JOIN Centre C
ON IDcentre = C.ID
```

→ Projection d'une jointure :

E = Etudiant

C = Centre

R = π

E.Nom, E.ville, C.Nom

Exemple (6) : Nom, Age des étudiants du centre
NY

```
SELECT E.Nom, E.Age
FROM Etudiant E JOIN Centre C
ON E.ID_Centre = C.ID
WHERE C.Nom = 'Nicolas Youssef'
```

⇒ les jointures

JOINTURE
interne

INNER JOIN

JOINTURE externe

LEFT OUTER JOIN X

RIGHT OUTER JOIN X

FULL OUTER JOIN X

JOINTURE Naturelle
A B

Correction de TD : (~~Page~~) (Après 3 feuilles)

a) SELECT NumPil
FROM Pilote P) NumPil (Pilote)

e) SELECT NumAv
FROM Avion
WHERE Capacité > 350) NumAv (Capacité (avion))

g) En service veut dire qu'ils ont un vol
SELECT NumPil, Ville DEP
FROM Pilote P INNER JOIN Vol V
ON P.NumPil = V.NumPil

⚠ Si on fait P LEFT OUTER JOIN, on obtient les résultats de INNER JOIN, les pilotes sans avions

⚠ Si on fait RIGHT OUTER JOIN, on obtient les résultats de INNER JOIN, avions sans pilote.

⚠ Si on fait P FULL OUTER JOIN, on obtient les résultats de INNER + pilotes sans avions + avions sans pilotes

A savoir

Par défaut, JOIN = INNER JOIN

j) SELECT NumPil
FROM Pilote
WHERE NumPil NOT IN (SELECT NumPil
FROM Vol);

k) SELECT NumVol, Ville Dep
FROM Vol
WHERE NumPil IN (100, 204)

⚠ si je veux faire compris entre 100 et 204
NumPil BETWEEN 100 AND 204

m) SELECT NumVol
FROM V.Vol JOIN Pilote P
ON V.NumPil = P.NumPil
WHERE Ville Dep = 'Nice' AND ADR = 'Nice';

➡ Nieux FROM Pilote Natural JOIN Vol.

130

g) SELECT Numvol
 FROM (Pilote P JOIN vol V
 ON P.Numpil = V.Numpil) JOIN AVION A
 ON V.NumAV = A.NumAV
 WHERE ADR = 'Nice' AND loc = 'Paris' AND
 (ville = 'Nice' OR ville = 'Paris')

Better way

FROM Pilote Natural JOIN vol
 Natural JOIN AVION

A saupic

Quand on fait :

- Like 'John%', ça veut dire 'John quelque chose'
- Like '%John', ça veut dire 'quelque chose John'
- Like '%John%', " " " 'quelque chose John quelque chose'

c) SELECT Numpil, Numpil
 FROM Pilote
 WHERE ADR = (SELECT ADR
 FROM Pilote

WHERE Numpil Like '%Dupont%' AND
 Numpil NOT Like '%Dupont%')

d) SELECT DISTINCT ville, ADR
 FROM VOL
 WHERE ville, Dep IN (SELECT ville, ADR
 FROM VOL
 WHERE ville, Dep = 'Paris');

```

b) SELECT DISTINCT Ville DEP
FROM VOL
UNION
SELECT DISTINCT Ville Arr
FROM VOL

```

⇒ en algèbre relationnelle

$$\pi_{\text{Ville-Dep}}(\text{vol}) \cup \pi_{\text{Ville-Arr}}(\text{vol})$$

```

c) SELECT DISTINCT Ville DEP
FROM VOL
WHERE Ville DEP IN (SELECT Ville Arr
FROM VOL);

```

```

d) SELECT DISTINCT Ville DEP
FROM VOL
WHERE Ville DEP NOT IN (SELECT Ville Arr
FROM VOL);

```

→ Une vue

Une vue = une Table virtuelle.

Elle n'existe pas physiquement, elle n'est présente qu'en mémoire et au moment de son appel.

It's really useful & on just the question to write it

```

CREATE VIEW Arr
AS SELECT DISTINCT Ville Arr
FROM VOL;

b) SELECT DISTINCT Ville DEP
FROM VOL
UNION
Arr;

```

(4)

c) SELECT DISTINCT ville - DEP
FROM vol
WHERE ville DEP IN Acc

d) SELECT DISTINCT ville - DEP
FROM vol
WHERE ville DEP NOT IN Acc

→ Question 8

Afficher Numvol, ville DEP, ville Acc,
Nompil, NomAV, H-DEP, H-Acc
SELECT Numvol, ville DEP, ville Acc, Nompil, NomAV, H-DEP, H-Acc
FROM (vol V LEFT OUTER JOIN Pilote P
ON V.Nompil = P.Nompil) JOIN AV ON A
ON B.NomAV = A.NomAV
ORDER BY H-DEP, ville DEP

A sous-requête

Order By : dans l'ordre
GROUP BY : se regroupe d'abord

v) SELECT DISTINCT A.Nompil, A.Nompil
FROM Pilote A JOIN Pilote P
ON A.Nompil = P.Nompil AND A.Nompil < P.Nompil
GROUP BY A.Nompil

> veut dire

#

Suite des cours :

→ les fonctions d'agrégation
(on les utilise avec SELECT and Having,
mais pas avec where)

- COUNT(*) ou COUNT(champ)
= Nbr d'éléments

- SUM(champ) = somme des valeurs d'une colonne

- AVG(champ) = la moyenne " "

- MAX(champ) = le max " "

- MIN(champ) = le min " "

(1) Exemple : le nbr de pilote
SELECT COUNT(*)
FROM Pilote

(2) Exemple : la totale capacité des avions faisant
le départ de Paris

SELECT SUM(capacité)
FROM AVION NATURAL JOIN Vol
WHERE Ville - DEP = 'Paris'

Exercices

12) les numéros et noms de pilote ayant des sautoires
maximaux

13) le num et nom et le nbr de vols de chaque pilote

14) les num et noms des pilotes assurant
au moins 2 vols à partir de Paris

12) SELECT Numpil, Numpil
FROM Pilote
WHERE SAL = (SELECT MAX(SAL)
FROM Pilote);

13) SELECT DISTINCT P.Numpil, Numpil, COUNT(Numpil)
FROM Pilote P LEFT OUTER JOIN VOLV
ON P.Numpil = V.Numpil
GROUP BY P.Numpil;

14) SELECT DISTINCT P.Numpil, P.Numpil
FROM Pilote P JOIN VOLV
ON P.Numpil = V.Numpil
WHERE Ville DEP = 'Paris'
GROUP BY P.Numpil
HAVING COUNT(*) >= 2

Exercice 5 Soit la base d'une compagnie aérienne donnée par les tables suivantes :

- PILOTE (NUMPIL, NOMPIL, ADR, SAL)
- AVION (NUMAV, NOMAV, CAPACITE, LOC)
- VOL (NUMVOL, NUMPIL, NUMAV, VILLE_DEP, VILLE_ARR, H_DEP, H_ARR)

Avec les significations suivantes pour les attributs :

- NUMPIL : clé de PILOTE, nombre entier
- NOMPIL : nom du pilote, chaîne de caractères
- ADR : ville de la résidence du pilote, chaîne de caractères
- SAL : salaire du pilote, nombre entier
- NUMAV : clé de AVION, nombre entier
- NOMAV : nom de AVION, chaîne de caractères
- CAPACITE : nombre de places d'un avion, nombre entier
- LOC : ville de l'aéroport d'attache de l'avion, chaîne de caractères
- NUMVOL : clé de VOL, nombre entier
- VILLE_DEP : ville de départ du vol, chaîne de caractères
- VILLE_ARR : ville d'arrivée du vol, chaîne de caractères
- H_DEP : heure de départ du vol, nombre entier entre 0 et 23
- H_ARR : heure d'arrivée du vol, nombre entier entre 0 et 23

Donnez les requêtes en algèbre relationnelle correspondants aux expressions suivantes :

- (1) (a) Donnez la liste des noms des pilotes;
- (b) Donnez la liste des villes desservies (arrivée ou départ);
- (c) Donnez la liste des villes qui sont desservies en arrivée et en départ;
- (d) Donnez la liste des villes qui sont desservies uniquement en départ;
- (2) (e) Donnez la liste des avions (leurs numéros) dont la capacité est supérieure à 350 passagers;
- (f) Quels sont les numéros et noms des avions localisés à Nice?
- (3) (g) Quels sont les ^{noms} numéros des pilotes en service et les villes de départ de leurs vols?
- (h) Quel est le nom des pilotes domiciliés à Paris dont le salaire est supérieur à 2500 euros?
- (i) Liste des vols au départ de Nice allant à Paris après 18 heures?
- (u) (j) Quels sont les numéros des pilotes qui ne sont pas en service?
- (5) (k) Quels sont les vols (numéro, ville de départ) effectués par les pilotes de numéro 100 et 204? or not and
- (l) Quels sont les numéros des pilotes en service qui ne s'appellent pas Durand?
- (6) (m) Donnez le numéro des vols effectués au départ de Nice par des pilotes Niçois?
- (n) Quels sont les vols effectués par un avion qui n'est pas localisé à Nice?
- (o) Quels sont les pilotes (numéro et nom) assurant au moins un vol au départ de Nice avec un avion de capacité supérieure à 300 places?
- (p) Quels sont les noms des pilotes domiciliés à Paris assurant un vol au départ de Nice avec un Airbus A380?
- (7) (q) Quels sont les numéros des vols effectués par un pilote Niçois au départ ou à l'arrivée de Nice avec un avion localisé à Paris?
- (8) (r) Quels sont les pilotes (numéro et nom) habitant dans la même ville que le pilote Dupont (on suppose qu'il n'y en a qu'un)?
- (9) (s) Quelles sont les villes desservies à partir de la ville d'arrivée d'un vol au départ de Paris? L'NC 2016
- (t) Quels sont les appareils (leur numéro) localisés dans la même ville que l'avion numéro 100?
- (u) Quels sont les numéros et noms des pilotes qui effectuent un vol au départ de leur ville de résidence?
- (10) (v) Donnez les numéros et noms des pilotes homonymes (même nom).

Exploiter une Base de Données à partir de Python

Exemple : SGBD SQLite (intégré en python) et la BD est un simple fichier local)

- Étape (0) : Importer la bibliothèque python correspondant au SGBD

`import sqlite3 / import mysql.connector.`

- Étape (1) : Établir la connexion

→ se connecter au SGBD et sélectionner la BD

`con = sqlite3.connect (chemin de la BD)`

or `con = mysql.connector.connect (ip, login, password, BD)`

→ créer le curseur de la connexion

zone mémoire intermédiaire entre le prog et le SGBD

- Étape (2) : Exécuter les requêtes SQL
- `cur.execute (Requête SQL)`

→ Remarque :

les requêtes autre que `SELECT` nécessitent une validation à l'aide de l'instruction

`con.commit()`

le résultat de `SELECT` est stocké dans le curseur sous forme d'une suite de tuples

- Étape (3) : Fermer le curseur et la connexion

`cur.close()`

`con.close()`

Comment récupérer le résultat d'une requête

SELECT e

```
cur.execute("""SELECT NomPil  
FROM pilote""")
```

Méthode (1) : Parcourir le curseur tuple par tuple
for t in cur:

→ t est un tuple # t[0] = NomPil
t[1] = NomPil.

➤ Après le parcours, le curseur est vide.

Méthode (2) : Mettre tous les tuples dans une liste

```
l = cur.fetchall()
```

Méthode (3) : Extraire du 1^{er} tuple

```
t = cur.fetchone()
```

→ il affecte à t le premier tuple et le supprime du curseur.

Exemple :

```
import sqlite3
```

```
def liste_Noms():
```

```
    con = sqlite3.connect("C:\A2\Gestion")
```

```
    cur = con.cursor()
```

```
    cur.execute("""SELECT NomPil, NomPil  
FROM Pilote""")
```

```
    l = cur.fetchall()
```

```
    cur.close()
```

```
    con.close()
```

```
    return l
```


def NombrePilotes():

```
    cur.execute(""" SELECT COUNT(*)  
                    FROM Pilote """)
```

```
    t = cur.fetchall()
```

```
    cur.close()
```

```
    con.close()
```

```
    return t[0]
```

Concours - 8

ÉCOLE POLYTECHNIQUE
ÉCOLE SUPÉRIEURE DE PHYSIQUE ET CHIMIE INDUSTRIELLES

CONCOURS D'ADMISSION 2008

FILIÈRE MP - OPTION PHYSIQUE ET SCIENCES DE L'INGÉNIEUR

FILIÈRE PC

COMPOSITION D'INFORMATIQUE

(Durée : 2 heures)

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

Ave Cesar (zud bdrzq)

On cherche à crypter un texte t de longueur n composé de caractères en minuscules (soit 26 lettres différentes) représentés par des entiers compris entre 0 et 25 ($0 \leftrightarrow a, 1 \leftrightarrow b, \dots, 25 \leftrightarrow z$). Nous ne tenons pas compte des éventuels espaces.

Ainsi, le texte `ecolepolytechnique` est représenté par le tableau suivant où la première ligne représente le texte, la seconde les entiers correspondants, et la troisième les indices dans le tableau t .

e	c	o	l	e	p	o	l	y	t	e	c	h	n	i	q	u	e
4	2	14	11	4	15	14	11	24	19	4	2	7	13	8	16	20	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Codage de César

Ce codage est le plus rudimentaire que l'on puisse imaginer. Il a été utilisé par Jules César (et même auparavant) pour certaines de ses correspondances. Le principe est de décaler les lettres de l'alphabet vers la gauche de 1 ou plusieurs positions. Par exemple, en décalant les lettres de 1 position, le caractère `a` se transforme en `z`, le `b` en `a`, ... le `z` en `y`. Le texte `avec Cesar` devient donc `zudbdrzq`.

Question 1 Que donne le codage du texte `maitrecorbeau` en utilisant un décalage de 5 ?

Question 2 Écrire la fonction `codageCesar(t, n, d)` qui prend en arguments le tableau t , sa longueur n et un entier d ; et qui retourne un tableau de même taille que t contenant le texte t décalé de d positions.

Question 3 Écrire de même la fonction `decodageCesar(t, n, d)` prenant les mêmes arguments mais qui réalise le décalage dans l'autre sens.

Pour réaliser ce décodage, il faut connaître la valeur du décalage. Une manière de la déterminer automatiquement est d'essayer de deviner cette valeur. L'approche la plus couramment employée est de regarder la fréquence d'apparition de chaque lettre dans le texte crypté. En effet, la lettre la plus fréquente dans un texte suffisamment long en français est la lettre e.

Question 4 Écrire la fonction $\text{frequences}(t', n)$ qui prend en argument un tableau t' de taille n représentant le texte crypté; et qui retourne un tableau de taille 26 dont la case d'indice i contient le nombre d'apparitions du nombre i dans t ($0 \leq i < 26$).

Question 5 Écrire la fonction $\text{decodageAuto}(t', n)$ qui prend en argument le tableau t' de taille n représentant le texte crypté; et qui retourne le texte t d'origine (en calculant la clé pour que la lettre e soit la plus fréquente dans le texte décrypté).

Codage de Vigenère

Au XVIème siècle, Blaise de Vigenère a modernisé le codage de César très peu résistant de la manière suivante. Au lieu de décaler toutes les lettres du texte de la même manière, on utilise un texte clé qui donne une suite de décalages.

Prenons par exemple la clé concours. Pour crypter un texte, on code la première lettre en utilisant le décalage qui envoie le a sur le c (la première lettre de la clé). Pour la deuxième lettre, on prend le décalage qui envoie le a sur le o (la seconde lettre de la clé) et ainsi de suite. Pour la huitième lettre, on utilise le décalage a vers s, puis, pour la neuvième, on reprend la clé à partir de sa première lettre. Sur l'exemple *ecolepolytechnique* avec la clé *concours*, on obtient : (la première ligne donne le texte, la seconde le texte crypté et la troisième la lettre de la clé utilisée pour le décalage)

e	c	o	l	e	p	o	l	y	t	e	c	h	n	i	q	u	e
g	q	b	n	s	j	f	d	a	h	r	e	v	h	z	i	w	s
c	o	n	c	o	u	r	s	c	o	n	c	o	u	r	s	c	o

Question 6 Donner le codage du texte *becunfromage* en utilisant la clé de codage *jean*.

Question 7 Écrire la fonction $\text{codageVigenere}(t, n, c, k)$ qui prend comme arguments un tableau t de taille n représentant le texte à crypter, et un tableau d'entiers c de longueur k donnant la clé servant au codage; et qui retourne un tableau de taille n contenant le texte crypté t' .

Maintenant, on suppose disposer d'un texte t' assez long crypté par la méthode de Vigenère, et on veut retrouver le texte t d'origine. Pour cela, on doit trouver la clé c ayant servi au codage. On procède en deux temps : 1) détermination de la longueur k de la clé c , 2) détermination des lettres composant c .

La première étape est la plus difficile. On remarque que deux lettres identiques dans t espacées de $\ell \times k$ caractères (où ℓ est un entier et k la taille de la clé) sont codées par la même lettre dans t' . Mais cette condition n'est pas suffisante pour déterminer la longueur k de la clé c puisque des répétitions peuvent apparaître dans t' sans qu'elles existent dans t . Par exemple, les lettres t et n sont toutes deux codées par la lettre h dans le texte crypté à partir de *ecolepolytechnique* avec *concours* comme clé. Pour éviter ce problème, on recherche les répétitions non pas d'une

Correction 8

Question (2) :

```
def codage_Cesar(t, d):  
    n = len(t)  
    tt = [0] * n  
    for i in range(n):  
        tt[i] = t[i] - d  
        if tt[i] < 0: tt[i] += 26  
    return tt
```

↳ mieux : $tt[i] = (t[i] - d) \% 26$

Encore mieux : $\text{return}[(n-d) \% 26 \text{ for } z \text{ in } t]$

Question (3) :

```
def decodage_Cesar(t, d):  
    return codage_Cesar(t, -d)
```

↳ ça marche only si j'écris 82 avec z
Encore mieux :

Question (4) :

```
def fréquences(t):  
    l = [0] * 26  
    for i in range(26):  
        for j in t:  
            if i == j:  
                l[i] += 1  
    return l
```

Better way :

```
def fréquences(t):  
    l = [0] * 26  
    for i in t:  
        l[i] += 1  
    return l
```


Question (1) :

```
def decodage_Cesar_Auto(t) :  
    F = frequences(t)  
    m = 0  
    for i in range(26) :  
        if F[i] > F[m] :  
            m = i  
    F.clear()  
    d = (m - 1) % 26 # 'a' - 1  
    return decodage_Cesar(t, d)
```

Question (2) :

```
def Codage_Vigenere(t, c) :  
    n, k = len(t), len(c)  
    tp = [0] * n  
    for i in range(n) :  
        tp[i] = (t[i] + c[i % k]) % 26  
    return tp
```

(# Better :

```
return [(t[i] + c[i % k]) % 26 for i in range(n)]
```

Question (3) :

Already done before.

(Please check the page suivante, suivante...)

Question (13)

```
def decodage Vigenere = Auto (tp) <math>\epsilon</math>  
  k = longueur De la cle (tp) <math>\epsilon</math>  
  c = [0] * k  
  for i in range (k) <math>\epsilon</math>  
    l = tp[i <math>\epsilon</math> k]  
    F = frequencies (l)  
    m = F.index(max(F))  
    c[i] = (m - <math>\epsilon</math>) % 26  
    l.clear()  
    F.clear()  
  n = len(tp)  
  return [(tp[i] - c[i % k]) % 26 for i in range (n)]
```

AS

lettre mais de séquences de lettres dans t' puisque deux séquences de lettres répétées dans t , dont les premières lettres sont espacées par $\ell \times k$ caractères, sont aussi cryptées par deux mêmes séquences dans t' .

Dans la suite de l'énoncé, on ne considère que des séquences de taille 3 en supposant que toute répétition d'une séquence de 3 lettres dans t' provient *exclusivement* d'une séquence de 3 lettres répétée dans t . Ainsi, la distance séparant ces répétitions donne des multiples de k .

La valeur de k est obtenue en prenant le PGCD de tous ces multiples. Si le nombre de répétitions est suffisant, on a de bonnes chances d'obtenir la valeur de k . On suppose donc que cette assertion est vraie.

Question 8 Écrire la fonction $\text{pgcd}(a, b)$ qui calcule le PGCD des deux entiers strictement positifs a et b par soustractions successives de ses arguments.

Question 9 Écrire la fonction $\text{pgcdDesDistancesEntreRepetitions}(t', n, i)$ qui prend en argument le texte crypté t' de longueur n et un entier i ($0 \leq i < n - 2$) qui est l'indice d'une lettre dans t' ; et qui retourne le pgcd de toutes les distances entre les répétitions de la séquence de 3 lettres $\langle t[i], t[i + 1], t[i + 2] \rangle$ dans la suite du texte $\langle t[i + 3], t[i + 4], \dots, t[n - 1] \rangle$. Cette fonction retourne 0 s'il n'y a pas de répétition.

Question 10 Écrire la fonction $\text{longueurDeLaCle}(t', n)$ qui prend en argument le texte crypté t' de longueur n ; et qui retourne la longueur k de la clé de codage.

Question 11 Donner le nombre d'opérations réalisées par la fonction longueurDeLaCle en fonction de la longueur n ? (On ne comptera que le nombre d'appels à la fonction PGCD).

Question 12 Une fois la longueur de la clé connue, donner une idée d'algorithme permettant de retrouver chacune des lettres de la clé. (Il s'agit de décrire assez précisément l'algorithme plutôt que d'écrire le programme).

Question 13 Écrire la fonction $\text{decodageVigenereAuto}(t', n)$ qui prend en argument le tableau t' de taille n représentant le texte crypté; et qui retourne le texte t d'origine. (On n'hésitera pas à recopier des parties de texte dans des tableaux intermédiaires).

Problème en Informatique

Ce problème est inspiré de l'épreuve d'informatique du concours d'admission 2011 de l'école polytechnique, filières PSI et PT. Dans toute l'épreuve le mot « tableau » signifie « liste » en python, ainsi « matrice » signifie « liste de listes ».

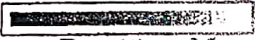
Images en gris



Image A

Les ordinateurs et de nombreux dispositifs électroniques (caméras numériques, écrans, etc.) représentent les images comme des matrices de nombres entiers. Dans ce problème, on se limite aux images rectangulaires en teintes de gris.

En machine, une telle image est la donnée d'une *hauteur* H , d'une *largeur* L , et d'une matrice M d'entiers de H lignes et L colonnes. L'image est divisée en éléments ou *pixels* définis par leurs numéros de ligne l et de colonne c . Chaque entier de la matrice définit le *ton de gris* associé au pixel de coordonnées (l, c) . Ce ton varie entre zéro, qui rend l'absence de lumière et donc le noir, et une valeur maximale dite *profondeur* P qui rend le blanc. Les valeurs intermédiaires rendent diverses teintes de gris de plus en plus claires. On notera que le pixel de coordonnées $(0, 0)$ est conventionnellement situé en haut et à gauche de l'image.

Par exemple, pour cette image , une échelle de 16 teintes de gris allant du noir au blanc, on a $H = 1$, $L = 16$, $P = 15$ et M ne possède qu'une seule ligne qui contient les 16 entiers de 0 à 15 en ordre croissant.

Pour représenter une image, nous allons définir une classe image dont les attributs sont : H (hauteur), L (largeur), P (profondeur) et M (matrice), comme suit :

```
class image:
    def __init__(self, h, l, p):
        self.H = h
        self.L = l
        self.P = p
        self.M = [ [p]*l for i in range(h)] #initialisation des pixels en couleur blanche
```

Ainsi, l'appel `image(100,100,255)` crée une image blanche de résolution 100x100 et de profondeur 255. On accédera aux composants d'une image i par les notations $i.H$ (hauteur), $i.L$ (largeur), $i.P$ (profondeur) et $i.M$ (matrice). Dans l'énoncé, on accédera aux éléments de la matrice par la notation $i.M[l, c]$ avec l est un indice de ligne compris entre 0 et $H-1$ au sens large, tandis que c est un indice de colonne compris entre 0 et $L-1$ au sens large.

Partie I. Opérations élémentaires

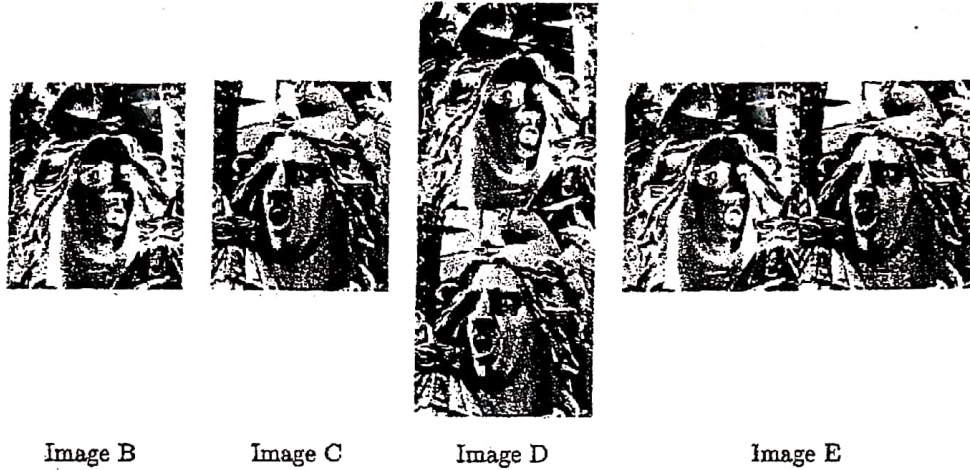


FIGURE 1 – Opérations élémentaires

Question 1 Écrire une fonction $\text{inverser}(i)$ qui renvoie l'image inverse de i . C'est-à-dire que le ton d'un pixel de la nouvelle image est $i.P - v$, où v est le ton du pixel correspondant de l'image d'origine. Par exemple, l'image B de la figure 1 résulte de l'application de inverser à l'image A de l'introduction.

Question 2 Écrire une fonction $\text{flipH}(i)$ qui renvoie la transformée de l'image i par la symétrie d'axe vertical passant par le milieu de l'image. Par exemple, l'image C de la figure 1 résulte de l'application de flipH à l'image A de l'introduction.

Question 3 Écrire une fonction $\text{poserV}(i_1, i_2)$ qui prend en arguments deux images i_1 et i_2 de même largeur et profondeur, et qui renvoie la nouvelle image obtenue en posant i_1 sur i_2 . Par exemple, l'image D de la figure 1 résulte de l'application de poserV aux images B et C.

Question 4 Écrire une fonction $\text{poserH}(i_1, i_2)$ qui prend en arguments deux images i_1 et i_2 de même hauteur et profondeur, et qui renvoie la nouvelle image obtenue en posant i_2 à droite de i_1 . Par exemple, l'image E de la figure 1 résulte de l'application de poserH aux images B et C.

Partie II. Transferts

Certaines transformations des images sont simplement l'application d'une fonction aux tons, dont on rappelle qu'ils sont des entiers compris entre 0 et P (profondeur de l'image) au sens large. Une telle fonction de transfert peut s'appliquer vers des images dont la profondeur n'est pas nécessairement P , mais une nouvelle profondeur. Une fonction de transfert est représentée par la donnée de la profondeur cible P' et d'un tableau d'entiers t de taille $P + 1$, dont les cases contiennent des entiers entre 0 et P' au sens large.

Question 5 Écrire une fonction $\text{transferer}(i, P', t)$ qui prend en arguments une image i , ainsi qu'une fonction de transfert donnée par un entier P' et un tableau d'entiers t . La fonction transferer renvoie une nouvelle image, de même taille que i , de nouvelle profondeur P' et dont chaque pixel (ℓ, c) a pour ton $t[i.M[\ell, c]]$.

Question 6 Écrire une nouvelle fonction inverser (Question 1) qui utilise la fonction transferer de la question précédente.



Image A

Image F

Image G

Image H

Image I

FIGURE 2 - Transferts

L'histogramme d'une image de profondeur P est un tableau h d'entiers de taille $P + 1$ tel que $h[v]$ compte le nombre de pixels de l'image dont le ton est v .

Question 7 Écrire une fonction $\text{histogramme}(i, h)$ qui prend en arguments une image i et un tableau d'entiers h correctement dimensionné, et qui affecte les cases de h pour en faire l'histogramme de i . On notera que le contenu initial des cases de h est inconnu.

Soit i une image de hauteur H , de largeur L et de profondeur P . Soit h son histogramme et soit $v_{\min}$ le ton de gris le plus sombre se trouvant dans l'image i . On égalise les tons en transformant chaque ton v en v' défini ainsi :

$$v' = P \times \frac{\left(\sum_{k=0}^{k=v} h[k] \right) - h[v_{\min}]}{H \times L - h[v_{\min}]}, \text{ pour } v_{\min} \leq v \leq P.$$

On note que v' n'est pas défini pour v tel que $0 \leq v < v_{\min}$, ce qui n'a pas d'importance, ces tons v étant absents de l'image. On note surtout que la valeur de v' ci-dessus n'est généralement pas un entier. Les candidats seront attentifs à cette difficulté, qu'ils résoudront dans le langage choisi. Ils pourront utiliser la primitive $\text{arrondir}(x)$ qui prend un nombre flottant x ou un rationnel en argument et renvoie l'entier le plus proche de x .

Question 8 Écrire la fonction $\text{egaliser}(i)$ qui prend une image i en argument et qui renvoie une nouvelle image qui est i dont les tons de gris sont égalisés. Par exemple, l'image F de la figure 2 résulte de l'application de egaliser à l'image A .

Question 9 Que renvoie la fonction `egaliser` appliquée à une image uniformément blanche?

On cherche maintenant à réduire la profondeur d'une image de P vers P' , avec $P' \leq P$. Une technique consiste à remplacer un ton v par v' , tel que v'/P' est le plus proche possible de v/P .

Question 10 Écrire une fonction `reduire( $i, P'$ )` qui renvoie une nouvelle image qui est i dont la profondeur est réduite à P' . Par exemple, les images G, H et I de la figure 2 résultent des réductions à 1, 4 et 16 de la profondeur de l'image A.

Partie III. Tramage

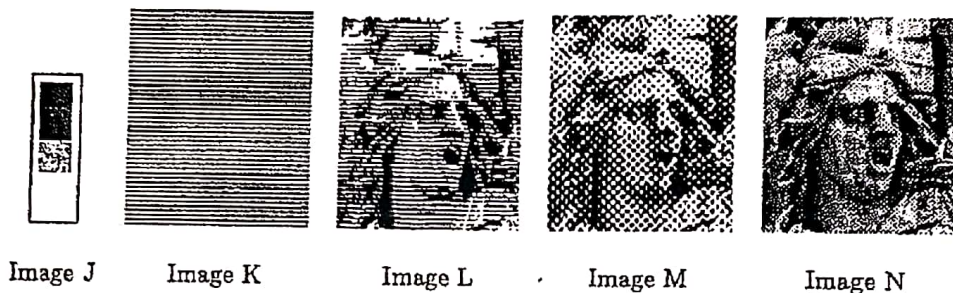


FIGURE 3 – Tramage

Lorsqu'il s'agit d'imprimer nos images, on se retrouve confronté à une difficulté : l'encre est noire, le papier est blanc. Il faut donc transformer les images en tons de gris en images en noir et blanc au sens strict. L'appel `reduire( $i, 1$ )` est un moyen de procéder, toutefois le résultat n'est pas très satisfaisant — voir l'image G de la figure 2. Cette partie examine la technique du tramage qui produit des images en noir et blanc (c'est-à-dire de profondeur 1) plus plaisantes, telles les images L, M et N de la figure 3.

On peut voir l'image G comme produite par seuillage, les tons de gris plus grands qu'un certain seuil deviennent blancs, tandis que les autres deviennent noirs. Une idée pour améliorer le rendu des images consiste à faire varier le seuil selon les pixels. Cela revient à représenter les seuils par une matrice, en fait par une image que l'on appelle une *trame*. Une trame est le plus souvent constituée par la répétition d'une petite image, dite *motif*, répétition selon les axes de coordonnées qui finit par paver toute l'image. Par la suite, le seuillage selon une trame sera simplement désigné comme le seuillage selon le motif dont dérive cette trame.

Par exemple, l'image J de la figure 3 est une échelle de gris verticale de profondeur 3 ($H = 4$, $L = 1$, $P = 3$ et la matrice M est un vecteur colonne contenant les entiers de 0 à 3 du haut vers le bas). Sa répétition produit l'image K. On notera que l'image J est présentée agrandie par rapport à l'image K. Le motif J est adéquat pour seuiller les images de profondeur 4, par exemple l'image H de la figure 2, ce qui donne l'image L de la figure 3.

Question 11 Écrire une fonction `tramer( $i, m$ )` qui prend deux images i et m en arguments, et qui renvoie l'image de profondeur 1 obtenue en seuillant l'image i selon le motif m . Le seuillage est défini précisément ainsi : étant donnés un ton v de l'image et un ton w de la trame, on obtient un pixel blanc si et seulement si $v > w$, et un pixel noir sinon.

Question 12 Écrire une fonction `tramerTelevision(i)` qui prend en argument une image i de profondeur P , et qui renvoie l'image de profondeur 1 obtenue en seuillant l'image i selon l'échelle de gris verticale de profondeur $P - 1$.

En imprimerie on utilise rarement des trames constituées de lignes horizontales comme la trame K. On préfère les trames constituées de lignes inclinées de points, ou trames diagonales. On obtient une trame diagonale de profondeur 15 par répétition du motif 16×16 représenté ci-dessous, à droite :

$$\begin{pmatrix} 1 & 5 & 10 & 14 \\ 3 & 7 & 8 & 12 \\ 13 & 9 & 6 & 2 \\ 15 & 11 & 4 & 0 \end{pmatrix} = \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix} \xrightarrow{?} \begin{matrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{matrix}$$

Le motif de droite dérive de l'image 4×4 représentée à gauche. On supposera par la suite qu'une variable globale `deuxQuarts` contient l'image de gauche.

Question 13 Écrire une fonction `tramerJournal(i)` qui prend en argument une image i de profondeur 16, et qui renvoie l'image de profondeur 1 obtenue en seuillant l'image i selon la trame diagonale de profondeur 15. Par exemple, l'image M de la figure 3 résulte de l'application de `tramerJournal` à l'image I de la figure 2.

Question 14 Indiquer un procédé simple qui permet d'obtenir l'image N de la figure 3 à partir de l'image I de la figure 2 et à l'aide de la trame diagonale de profondeur 15. On notera que l'image N est bien une image de profondeur 1, et qu'elle est représentée réduite (d'un facteur 2) par rapport à l'image M. Coder ensuite votre procédé comme une fonction qui prend une image de profondeur arbitraire en argument.

Si $P = 1$: 0 noir, 1 blanc

Si $P = 2$: 0 noir, 1 gris, 2 blanc

Si $P = 3$: 0 noir, 1 gris foncé, 2 gris clair, 3 blanc

Question 1) `def invoker(i)`

`def invoker(i) :`

`j = image(i.h, i.l, i.p)`

`for p in range(i.h) :`

`for c in range(i.l) :`

`j.h[p][c] = i.p[i.h[p][c]]`

Question (2) is flip H

```
def flipH(i):  
    j = image(i.H, i.L, i.P)  
    for l in range(i.H):  
        for c in range(i.L):  
            j.p[l][c] = i.p[l][i.L-1-c]  
    return j
```

Question (3) is poscv

```
def poscv(i1, i2):  
    i = image(i1.H, i2.H, i1.L, i2.P)  
    for l in range(i1.H):  
        for c in range(i1.L):  
            i.p[l][c] = i1.p[l][c]  
    for l in range(i1.H, i.H):  
        for c in range(i.L):  
            i.p[l][c] = i2.p[l-i1.H][c]  
    return i
```

merge (i)

```
for l in range(i.H):  
    for c in range(i.L):  
        if l < i1.H:  
            i.p[l][c] = i1.p[l][c]  
        else:  
            i.p[l][c] = i2.p[l-i1.H][c]
```

Question (4) e - Porcett

```
def porcett(i, i2):  
    i = image(i, H, i2.L, i2.P)  
    for l in range(i.H):  
        for c in range(i.L):  
            i.m[l][c] = i2.m[l][c]  
    for c in range(i2.L, i2.L):  
        i.m[l][c] = i2.m[l][c - i2.L]  
    return i
```

DA

Tu ne peux
pas écrire
en python
P'

Question (5) e

```
def Transférer(i, Pprime, t):  
    j = image(i, H, i.L, Pprime)  
    for l in range(i.H):  
        for c in range(i.L):  
            j.m[l][c] = t.m[l][c]  
    return j
```

Question (6) e

```
def inverser(i):  
    t = [i.P - v for v in range(i.P, 1)]  
    return Transférer(i, i.P, t)
```

Question (7) e

R est un compteur, c'est pourquoi il faut le remplir
avec des 0. (on l'initialise)

DA A chaque fois qu'ils te donnent un tableau et tu veux
l'initialiser, c'est FAUX de faire $R = [0] * (\text{len}(B))$
parce que si tu as fait une copie de R
il faut faire $\text{for } v \text{ in range}(\text{len}(R))$
 $R[v] = 0$

05 / 11
10


```

def Histogramme(i, R): # len(R) = i.p + 1
    for v in range(len(R)):
        R[v] = 0
    for l in range(i.H):
        for c in range(i.L):
            R[i.H * l + c] = 1

# Si R n'était pas donné par créer R, on
# aurait fait R = [0] * (i.p + 1)

```

TD (3) : Requêtes SQL

Requêtes simples :

```

(i) CREATE TABLE etudiant (
    numero VARCHAR(10) PRIMARY KEY NOT NULL,
    nom CHAR(50),
    prenom CHAR(50),
    age INT ;

```

```

CREATE TABLE xance (
    cours VARCHAR(20),
    numero INT,
    type VARCHAR(2),
    date DATE,
    salle VARCHAR(10),
    heure Debut TIME,
    heure fin TIME,
    enseignant VARCHAR(10),

```

```
PRIMARY KEY (cours, numero),  
FOREIGN KEY (cours) REFERENCE cours (sigle),  
FOREIGN KEY (enseignant) REFERENCES Enseignant (id);
```

```
iii) INSERT INTO inscription  
VALUES ('10372', 'COGDIR')
```

```
iii) SELECT nom, prenom  
FROM etudiant  
WHERE age < 20
```

```
iv) SELECT nom, prenom  
FROM enseignant E JOIN cours C  
ON C.responsable = E.id  
WHERE intitule = 'Statistiques'
```

```
v) SELECT nom, prenom  
FROM Etudiant E  
JOIN inscription I ON E.Numero = I.etudiant  
JOIN cours C ON C.sigle = I.cours  
WHERE C.intitule = 'Probabilités';
```

```
vi) SELECT COUNT (DISTINCT enseignant)  
FROM Seance S JOIN cours C  
ON S.cours = C.sigle  
WHERE intitule = 'Modélisation Stochastique';
```

5/0

TD Bases de Données et Ingénierie des Systèmes d'Information TD 3 – Requêtes SQL

TD sur les requêtes SQL

3 décembre 2008

Prérequis : Modèle conceptuel de données (entité-association), modèle relationnel, bases du langage SQL.

Durée : 1 h 50

TD 3 – Requêtes SQL

Description du système d'informations

La direction des études des Mines de Nancy a décidé d'informatiser la gestion des emplois du temps. Chaque étudiant est caractérisé par son numéro d'étudiant, son nom, son prénom et son âge. Chaque cours est identifié de façon unique par un sigle (SI033, MD021, ...) et possède un intitulé (bases de données, mathématiques discrètes, ...) ainsi qu'un enseignant responsable. On connaît également le nombre de séances de chaque cours. Les enseignants sont caractérisés par un identifiant alphanumérique, leur nom et leur prénom. Enfin, chaque séance est identifiée par le cours ainsi que le numéro de la séance (séance 3 du cours SI033, séance 1 du cours de MD021, ...), le type d'intervention (CM, TD, TP), la date, l'heure de début et l'heure de fin auxquelles la séance a lieu ainsi que la salle et l'enseignant qui dispense la séance. Les étudiants s'inscrivent aux cours auxquels ils souhaitent assister.

Schéma relationnel retenu

Les clés primaires sont soulignées et les clés étrangères sont en italique.

etudiant (numero, nom, prenom, age) *fait référence à l'id de l'enseignant*
enseignant (id, nom, prenom)
cours (sigle, intitule, *responsable*, nombreSeances)
seance (cours, numero, type, date, salle, heureDebut, heureFin, *enseignant*)
inscription (*etudiant*, *cours*)

Requêtes simples

- i) Écrire les requêtes de création des tables « Etudiant » et « Séance ».
- ii) Inscrivez l'étudiant ('10372', 'Léponge', 'Bob', 20) au cours ('LOG015', 'Logique', 'jh1908').
- iii) Cherchez le nom et le prénom de tous les étudiants de moins de 20 ans.
- iv) Cherchez le nom et le prénom de l'enseignant responsable du cours de Statistiques.
- v) Cherchez le nom et le prénom de tous les étudiants inscrits au cours de Probabilités.
- vi) Déterminez le nombre d'enseignants intervenant dans le cours de Modélisation Stochastique.
- vii) Où et quand a lieu le premier cours d'Algèbre linéaire ?
- viii) Affichez un « emploi du temps » du cours de Logique.
- ix) Pour chaque enseignant, indiquez le nombre de cours dans lesquels il intervient (restreignez les réponses à l'ensemble des enseignants qui interviennent dans au moins deux cours).

Requêtes imbriquées

- i) Ajoutez un cours magistral de Logique le 14 décembre avec Jacques Herbrand en salle S250 de 14h à 18h.

TD Bases de Données et Ingénierie des Systèmes d'Information TD 3 – Requêtes SQL

- ii) Listez les étudiants inscrits à aucun cours.
- iii) Combien d'étudiants (différents) ont assistés à au moins une séance animée par Leonhard Euler?

Syntaxe SQL

Sélection

```
SELECT liste_d_attributs [DISTINCT, AS  
+ , +, *, %  
AVG, MAX, MIN, SUM, COUNT  
FROM noms_de_tables  
[ WHERE liste_de_critères ] { <, <=, >, >=, =, <>  
BETWEEN, IN, IS NULL  
LIKE, AND, OR, NOT  
[ GROUP BY liste_d_attributs ]  
[ HAVING liste_de_critères ]  
[ ORDER BY liste_d_attributs ]
```

Exemple :

```
SELECT SUM(p.gain)  
FROM Participe p, Jockey j  
WHERE p.Numero_jockey = j.Numero_jockey  
AND j.nom like 'Jean-Claude Dusse';
```

Création de tables

```
CREATE TABLE nom_de_la_table (  
  nom_de_l'attribut type [liste_de_contraintes_d'attribut]  
  nom_de_l'attribut type [liste_de_contraintes_d'attribut]  
  liste_de_contraintes_de_table  
);
```

où :

- type $\in \left\{ \text{VARCHAR}(n) \text{ où } n \in \mathbb{N}, \text{INT}, \text{DATE}, \text{TIME} \right\}$
- contrainte_d'attribut $\in \left\{ \begin{array}{l} \text{NULL, NOT NULL,} \\ \text{DEFAULT valeur,} \\ \text{CHECK(nom_de_l'attribut in domaine_de_définition)} \end{array} \right\}$
- contrainte_de_table $\in \left\{ \begin{array}{l} \text{PRIMARY KEY(liste_d'attributs),} \\ \text{FOREIGN KEY(nom_de_l'attribut) ...} \\ \text{... REFERENCES nom_de_la_table(nom_de_l'attribut)} \end{array} \right\}$

TD Bases de Données et Ingénierie des Systèmes d'Information TD 3 – Requêtes SQL

Exemple :

```
CREATE TABLE cours (  
  sigle          VARCHAR(20) NOT NULL,  
  intitule       VARCHAR(128) NOT NULL,  
  responsable    VARCHAR(50) NOT NULL,  
  nombreSeances INT          NOT NULL DEFAULT '0',  
  PRIMARY KEY (sigle),  
  FOREIGN KEY (responsable) REFERENCES enseignant(id)  
);
```

Suppression de table

```
DROP TABLE nom_de_la_table;
```

Insertion

```
INSERT INTO nom_de_la_table (attribut_1, attribut_2, ...)  
VALUES (valeur_1, valeur_2, ...);
```

Requêtes imbriquées / sous-requêtes

Une sous-requête est une commande SELECT dans une autre commande. Par exemple :

```
SELECT * FROM table1 WHERE id IN (SELECT id FROM table2);
```

On dit que la sous-requête est imbriquée dans la requête externe. Il est possible d'imbriquer des requêtes dans des sous-requêtes. Une sous-requête doit toujours être entre parenthèses. Voici un exemple de commande qui montre les principaux avantages des sous-requêtes et de leur syntaxe :

```
SELECT r1  
FROM t1  
WHERE s11 = (  
  SELECT COUNT(*) FROM t2  
  WHERE NOT EXISTS (  
    SELECT * FROM t3  
    WHERE r3 =  
      (SELECT s0,11*s1 FROM t4 WHERE r5 IN (SELECT * FROM t5) AS t5)  
  );
```

EXISTS teste simplement si la requête interne retourne une ligne. NOT EXISTS teste si la requête interne ne retourne aucun résultat.


```

vii) SELECT Salle, date, Heure Debut
      FROM Sance S JOIN Cours C
      ON S.cours = C.sigle
      WHERE intitule = 'Algèbre linéaire'
      AND S.nombre = 1;

```

```

ix) SELECT id, nom, prenom, COUNT(S.cours)
      FROM Enseignant E JOIN Sance S
      ON E.id = S.enseignant
      GROUP BY id
      HAVING COUNT(DISTINCT S.cours) > 1;

```

Requêtes imbriquées :

```

i) INSERT INTO Sance
   VALUES ((SELECT sigle FROM Cours WHERE intitule = 'logique'),
            (SELECT nombre Sances + 1
             FROM Cours
             WHERE intitule = 'logique'),
            'CD',
            '2017-12-14',
            'S2C0',
            '16:00',
            '18:00',
            (SELECT id
             FROM Enseignant
             WHERE nom = 'Herbrand' AND prenom = 'Jacques'));

```

```

UPDATE Cours
SET nombre Sances = nombre Sances + 1
WHERE intitule = 'logique';

```

32

