

UNIVERSITE OFFICIELLE DE BUKABU(UOB)

Ecole des mines

Licence 1

Cours d'Informatique et technologie de l'information I

Pr. NIYONSABA Thérance

2021/2022

Table des matières

0. Introduction	3
1. Classification des langages de programmation	6
1.1 Généralités	6
1.2 Généalogie des langages de programmation	7
1.3 Les grandes classes des langages de programmation	9
Chapitre 1. Les notions de bases de l’algorithmique	12
1.1. Structures générale de l’algorithme.....	13
1.2 Les notions de base.....	14
1.2.1 Les identificateurs	14
1.2.2. Les types de données.....	14
1.2.3 La déclaration.....	15
1.2.4 L’affectation	16
1.2.5 Les expressions algébriques	16
Chap 2 Les instructions conditionnelles et les boucles.....	17
2.1 La boucle tant que	18
2.2 Les boucles répéter... JUSQU'A	19
2.3 La boucle POUR...FAIRE	19
2.4 Les instructions conditionnelles si.... si non.....	21
2.5 Structure conditionnelle à choix multiple	23
Chapitre. 3 Les tableaux	24
« 3.1. Les listes	24
3.1.1. Déclaration	25
3.2. Les matrices.....	29
3.2.1. Déclaration d’un tableau bidimensionnel	30
3.3 Quelques algorithmes de tri.....	33
3 .3.1Algorithme de tri à bulles.....	33
3.3.2. Algorithme de tri par insertion.....	35
3.3.3. Algorithme de tri rapide	36
Chapitre. 4 Programmation graphique avec Python	38
4.1. Notions générales	38
4.2. Programmation graphique	40
Bibliographie.....	43

0. Introduction

L'informatique se définit comme une science de traitement automatique de l'information. Ce traitement se réalise dans un système informatique (système d'information). Ce dernier étant un ensemble organisé de ressources, de personnes et de structures qui évoluent dans une organisation et dont le comportement coordonné vise à atteindre un but commun. Les systèmes d'information sont censés aider les utilisateurs dans leurs activités : stocker et restaurer l'information, faire des calculs, permettre une communication efficace, ordonnancer et contrôler des tâches, etc.

Le **système d'information** ou SI, peut être défini comme étant l'ensemble des moyens humains, matériels et immatériels mis en œuvre afin de gérer l'information au sein d'une unité, une entreprise par exemple.

Il ne faut toutefois pas confondre un **système d'information** avec un **système informatique**. En effet, les systèmes d'information ne sont pas toujours totalement informatisés et existaient déjà avant l'arrivée des nouvelles technologies de l'information et des communications dont l'informatique fait partie intégrante. Le système d'information possède 4 fonctions essentielles :

- La **saisie** ou **collecte** de l'information
- La **mémorisation** de l'information à l'aide de fichier ou de base de données
- Le **traitement** qui concerne les quatre opérations fondamentales à savoir l'insertion, suppression, modification et visualisation qui forment l'abréviation CRUD : Create, Read, Update, Delete. A ces opérations s'ajoutent les opérations de consultation, organisation, mise à jour, calculs pour obtenir de nouvelles données, ...)
- La **diffusion** de l'information.

La conception d'un système d'information n'est pas évidente car il faut réfléchir à l'ensemble de l'organisation que l'on doit mettre en place. La phase de conception nécessite des méthodes permettant de mettre en place un modèle sur lequel on va s'appuyer.

Une des branches de l'informatique consiste à écrire des programmes pour résoudre des problèmes. Avant la phase d'écriture d'un programme et de son implémentation, il faut d'abord bien définir le problème (et les données associées) et c'est l'algorithmique qui permet de le résoudre.

On ne peut réduire la notion d'algorithme aux seules méthodes informatiques de résolution de problèmes. L'algorithmique est un terme dont l'origine remonte au XIII^{ème} siècle, du nom d'un mathématicien perse Al-Khwarizmi. Mais on peut faire remonter la pratique algorithmique bien avant, avec les savants babyloniens de 1800 avant J.C.

Le concept d'algorithmique, sera finalement théorisé par Alan Turing en 1936, avec la machine qui porte son nom, et qui à l'heure actuelle est encore (et plus que jamais) fréquemment invoquée dans de nombreux travaux d'informatique théorique. La seconde moitié du XX^e siècle, qui verra un développement exponentiel des calculateurs, constituera une formidable mise en application des principes développés jusque lors. L'on se gardera cependant de penser hâtivement que l'algorithmique a été rendue superflue par l'apparition des ordinateurs, et par le développement de logiciels d'édition de code toujours plus sophistiqués et intelligents.

D'ailleurs, Michael Fellows avait écrit : "Computer science is not about machines, in the same way that astronomy is not about telescopes", indiquant par là que l'ordinateur n'est qu'un moyen d'observation de la science informatique.

Il existe de nombreuses définitions du mot algorithme. Dans ce cours, nous utiliserons l'acception suivante :

Un algorithme est une séquence d'opérations visant à la résolution d'un problème en un temps fini (mentionner la condition d'arrêt). Exemples d'algorithmes : l'algorithme de multiplication scalaire, l'algorithme d'Euclide (calcul du pgcd de deux entiers),...etc.

Un algorithme doit donc être :

- Lisible: l'algorithme doit être compréhensible même par un non-informaticien ;
- Universel: l'algorithme doit pouvoir être traduit en n'importe quel langage de programmation, il ne doit donc pas faire appel à des notions techniques relatives à un programme particulier ou bien à un système d'exploitation donné
- Précis: chaque élément de l'algorithme ne doit pas porter à confusion, il est donc important de lever toute ambiguïté
- Structuré: un algorithme doit être composé de différentes parties facilement identifiables

La conception d'un algorithme se fait toujours par étapes de plus en plus détaillées. La première version de l'algorithme est autant que possible indépendante de son implémentation, la représentation des données n'est donc pas fixée. A ce niveau, les données sont considérées de façon abstraite : un nom, les valeurs prises, les opérations possibles et les propriétés de ces opérations. On parle souvent de type abstrait de données (ou TAD). La conception de l'algorithme se fait en utilisant les opérations du TAD.

La représentation concrète du type de données est définie en termes d'objets du langage de programmation utilisé. Il se peut que ce type soit déjà défini dans le langage (c'est le cas par exemple des entiers et leurs opérations) ou qu'il faut le définir comme on le fera pour les listes chaînées.

Une fois l'algorithme et les structures de données définies, on les code en un langage informatique et on obtient un programme qui est défini comme une suite d'instructions permettant de réaliser une ou plusieurs tâche(s), de résoudre

un problème, de manipuler des données. On parle souvent de source (au masculin). Il s'agit en fait du texte source du programme, texte original d'un programme, dans le langage informatique compréhensible par un être humain, donc destiné à être compilé.

Aujourd'hui avec le développement des sciences et technologies, nous sommes parvenus au monde de la programmation à la 4^{ème} génération. Nous sommes passés du binaire à l'assembleur puis des langages procéduraux, aux langages évènementiels et d'objets.

Derrière toutes ces innovations, aussi complexes qu'elles soient, nous répétons toujours le même processus pour résoudre un problème en informatique. Les étapes de cette résolution peuvent être schématisées ainsi qu'il suit :

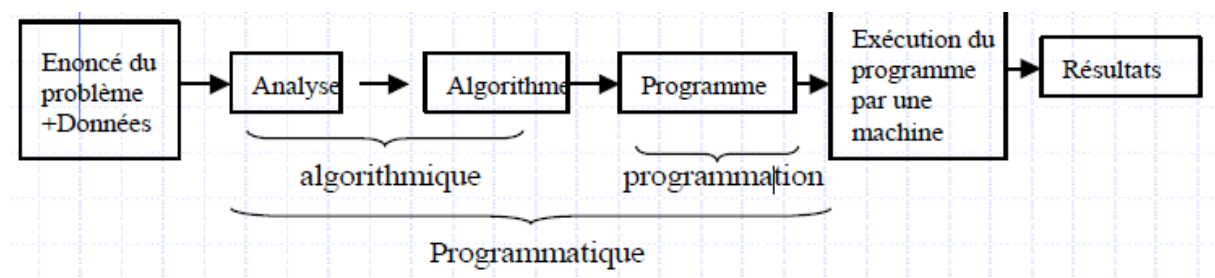


Fig. 1. Généralisation des étapes de résolution

La compilation est le passage du code source d'un programme à un exécutable à l'aide du logiciel approprié (le compilateur). L'exécutable est le programme en code binaire directement compréhensible par le processeur.

1. Classification des langages de programmation

1.1 Généralités

Un langage de programmation est une « langue » qui permet de décrire des traitements, des tâches à effectuer, sous la forme d'un programme qui sera traduit par un « traducteur » (compilateur, interpréteur) pour être exécuté par l'ordinateur.

Comme tout langage, un langage de programmation a une syntaxe. Il comporte les éléments suivants :

- Des verbes qui ordonnent une action
- Des objets, données ou objets matériels sur lesquels on opère ou qu'on définit
- Des attributs spécifiant la manière d'opérer
- Des étiquettes permettant de repérer un endroit du texte-programme

Un langage de programmation est non-ambigu (en anglais " context-free ") : TOUT y est défini, à la différence des langues naturelles.

1.2 Généalogie des langages de programmation

Premier langage de haut niveau créé, le FORTRAN est et resté le grand langage de calcul scientifique malgré ses faiblesses syntaxiques. Pour les applications de gestion, le COBOL (COMmon Business Oriented Language) reste un standard international.

Fortran et Cobol sont encore utilisés 40 ans après leur apparition. Dans les années 1960 Fortran, Cobol sont évolués et s'ajoute et Basic.

Trois langages sont restés importants dans les années 90 :

- 1) Pascal était le langage enseigné dans les universités. Il n'est plus enseigné dans les universités comme le langage de base bien qu'il soit pédagogique.
- 2) Le langage C est le langage des programmeurs systèmes et a été avec Unix le langage des stations de travail. Il est devenu C++ qui a servi ensuite à créer Java.
- 3) Ada a démarré fin 70, Ada 83 et Ada 95 (le langage et ses évolutions sont venus

toujours trop tard). Tous ces langages ont eu une extension orientée objet.

Algol a joué une profonde influence sur la définition et la conception de langages. Défini par un comité européen et américain (International Algebraic

Language IAL, ALGOrithmic Language). Proche du langage mathématique et lisible.

Algol 58 au début mais jamais implémenté. Concept Algol 60 a conduit à définir des compilateurs dirigés par la syntaxe. La sémantique était définie en anglais et avait des ambiguïtés. Algol 60 est structuré en blocs, les variables ne sont pas visibles en dehors des blocs. Algol 60 a les instructions de contrôle structurées améliorées par rapport au Fortran.

Début 70 grâce au travail de Niklaus Wirth sur Algol. Pour défier le Fortran utilisé aux USA. Dimension du tableau statique (contrairement à l'Algol) car on améliore ainsi l'exécution des programmes. On introduit les types de données qui peuvent être construits à partir des types non structurés integer, booléen, char et énumération.

Type structuré par l'utilisateur : tableaux, enregistrement, ensembles et fichiers.

Types de pointeur avec des enregistrements pour faire des arbres et des listes liées.

Pascal a été adopté par les Universités dans les années 70 pour enseigner l'informatique.

Et mi 80 c'est le langage enseigné pour les étudiants. C'est le langage qui introduit un ensemble complet de structures de données et qui encourage un bon style de

programmation. Pascal a influencé les autres langages. Wirth conçut ensuite Modula puis Modula-2, c'était l'introduction du concept des modules, une

extension de Pascal. Pascal a profondément influencé Ada.

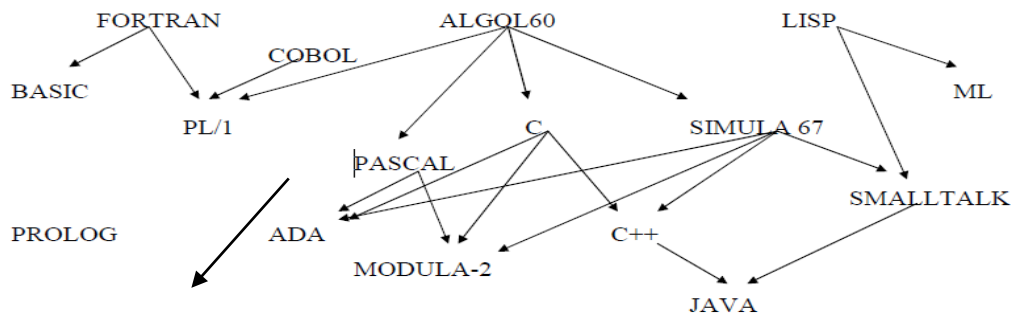


Fig 1. Généalogie des langages de programmation

1.3 Les grandes classes des langages de programmation

Même si tous les langages de programmation finissent par traduire leur code en langage machine, il existe plusieurs niveaux de langage de programmation, qui peuvent être plus simples à utiliser pour un programmeur. Plus un langage est de haut niveau, plus il faut abstraction du matériel, alors qu'un langage de base est plus proche du matériel de l'ordinateur. Ces différents niveaux peuvent être séparés en : le langage machine et le langage assembleur, qui sont des langages de bas niveau, le langage de haut niveau et le langage de quatrième génération.

Le langage machine

Le langage machine, dit langage de première génération, est ce qui pourrait être considéré comme le langage de programmation du plus bas niveau. Il est écrit en binaire, une suite de 0 et 1, qui se regroupent en groupe de 8 pour former ce que l'on appelle des octets. Chaque « 0 » veut dire non, ou faux, et chaque « 1 » veut dire oui, ou vrai. C'est le langage de programmation le plus simple à comprendre pour l'ordinateur, car chaque processeur est muni d'une palette d'instruction en langage machine qui lui est propre, un langage natif. Si un processeur A peut exécuter toutes les instructions d'un processeur B, le

processeur A est compatible avec le processeur B. Le contraire n'est pas forcément vrai, car le processeur A peut posséder des instructions que le processeur B ne possède pas. Le langage machine est donc le seul langage de programmation qui peut être directement

« compris » par l'ordinateur. Si c'est le seul qui peut être compris par l'ordinateur, pourquoi il y a-t-il d'autre niveau de langage de programmation ? La réponse est simple, le langage machine est un langage qui est très difficile à comprendre pour un programmeur et le débogage peut être très fastidieux. C'est pourquoi il existe un langage de programmation tel que le langage assembleur.

Le langage assembleur

Le langage assembleur, dit langage de deuxième génération, est le langage de programmation qui se rapproche le plus du langage machine. Il remplace la plupart des instructions binaire du processeur par des mots clés, comme MOV. Ce langage facilite grandement la programmation, car le programmeur n'a plus besoin de connaître toutes les lignes de bits qui réfèrent à une instruction du processeur par cœur. Par contre, parce que chaque processeur possède un langage machine qui lui est propre, il doit aussi y avoir un langage assembleur propre à chaque processeur. Parce qu'il est très proche du langage machine, le langage assembleur peut être traduit en langage machine et être exécuté rapidement par l'ordinateur. Par contre, pour la même raison qu'il peut être exécuté rapidement, le langage assembleur reste un langage difficile à comprendre et à déboguer pour un programmeur.

Le langage de haut niveau

Le langage de haut niveau, dit langage de troisième génération, est un niveau langage de programmation qui se rapproche beaucoup plus d'un langage dit naturel (par exemple l'anglais) que du langage machine. Le code d'un

programme fait avec un langage de haut niveau utilise des mots issus d'un langage plus familier pour l'être humain en faisant abstraction des caractéristiques techniques du matériel. C'est l'équivalent du langage assembleur, mais à un niveau d'abstraction beaucoup plus élevé. Cela rend le langage de haut niveau beaucoup plus compréhensible et facile d'utilisation pour un programmeur. Par contre, l'utilisation d'un langage de haut niveau comporte un certain défaut, car il faut traduire le code du programme en langage machine avant de l'exécuter, ce qui rend le programme plus lent à l'exécution et à la compilation. De plus, le langage de haut niveau ne comporte pas le même problème que les langages machine et assembleur ; il n'est pas propre au processeur. Un programme fait avec un langage de haut niveau peut donc facilement être fonctionnel sur plusieurs ordinateurs. Par contre, certains programmes peuvent être uniques à un système d'exploitation (Windows, Linux, ...). Pour être exécuté, un programme utilisant un langage de haut niveau doit être compilé ou interprété.

Dans le cas où le programme est compilé, le code source du programme est traduit à son équivalent en langage machine par un programme appelé *compilateur*. Lorsque la compilation du programme est terminée, le code en langage machine résultant est sauvegardé séparément et peut alors être exécuté indépendamment du code source en tout temps. A chaque modification du code source, il faut alors recompiler le code pour faire une nouvelle version exécutable du programme en langage machine. Le code machine résultant de cette compilation est moins efficace (plus lent) que l'équivalent qui aurait été fait en langage assembleur, mais il est beaucoup plus rapide et simple pour un programmeur de faire un programme en langage de haut niveau qu'en langage assembleur ou en langage machine. Dans le cas où le programme est interprété, le programme reste en mémoire alors qu'il est interprété ligne par ligne, par un programme appelé *interpréteur*, et exécuté directement, sans faire de traduction préalable en langage machine.

On peut donc exécuter le programme pour le tester, y apporter quelque modification et le réexécuter directement. Ceci peut faire avancer grandement le processus de développement et de test comparativement au langage compilé, qui lui doit être recompilé à chaque fois qu'une modification est apportée au code source. Par contre, parce que le programme doit rester en mémoire en même temps que l'interpréteur et que l'interpréteur doit réinterpréter le code source du programme à chaque exécution, un programme interprété requiert plus de mémoire et est plus lent à l'exécution qu'un programme compilé.

Le langage de quatrième génération

Le langage de quatrième génération (L4G, ou 4GL en anglais) est un langage qui se rapproche encore plus du langage naturel que le langage de troisième génération. Le langage de quatrième génération est un langage qui est conçu pour résoudre des problèmes spécifiques, contrairement au langage polyvalent de troisième génération. Le langage de quatrième génération est non procédural et est plus simple à apprendre qu'un langage de troisième génération. Ce niveau de langage permet au programmeur de programmer une application spécifique à un sujet et de le faire avec une grande lisibilité et très peu de code, avec le défaut d'être moins polyvalent et d'avoir moins de possibilité d'optimisation. Le langage de quatrième génération est souvent utilisé pour accéder aux bases de données et pour faire des requêtes concernant celles-ci. Parce que ce langage est dédié à un domaine spécifique, les opérations spécifiques concernant ce domaine sont déjà incluses dans le lexique du langage, contrairement au langage de troisième génération où ces opérations seraient incluses dans une bibliothèque logiciel complémentaire.

Chapitre 1. Les notions de bases de l'algorithmique

1.1. Structures générale de l'algorithme

L'algorithme est composé par trois parties essentielles à savoir :

- L'en-tête
- La partie déclarative
- Le corps de l'algorithme

L'en-tête peut contenir le nom de l'auteur, la date d'écriture de l'algorithme et les fonctions de l'algorithme.

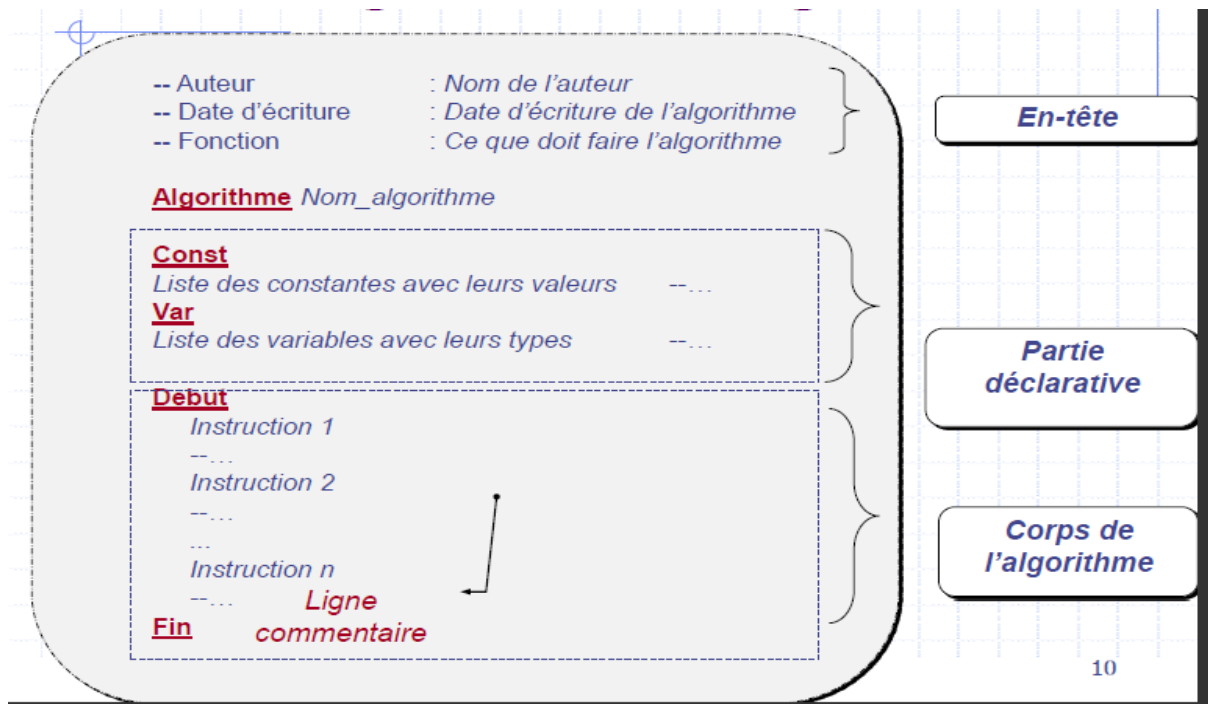
Avant d'utiliser n'importe quel objet de l'algorithme, il faut le déclarer au niveau de la partie déclarative de l'algorithme. Les objets de l'algorithme sont les données et les résultats. On distingue :

- Les objets en entrée: Ce sont les données fournies à l'algorithme.
- Les objets en sortie: Ce sont les résultats produits par l'algorithme.
- Les objets internes ou intermédiaires: Ce sont les objets de manœuvre de l'algorithme servant aux manipulations internes (compteurs, objets intermédiaires de stockage,...).

Le corps de l'algorithme contient l'ensemble des instructions applicables sur l'ensemble des objets algorithmiques déjà déclarés au niveau de la partie précédente (déclarative). Ces instructions se divisent essentiellement en trois catégories:

- Les instructions simples: Entrée de données, sortie de résultats, affectation.
- Les structures décisionnelles (conditionnelles): Ce sont des structures de contrôle permettant de choisir entre les traitements.
- Les structures itératives (répétitives): Ce sont des structures de contrôle permettant de répéter un ensemble de traitements autant de fois qu'on veut.

La figure suivante, montre la structure générale de l'algorithme.



1.2 Les notions de base

1.2.1 Les identificateurs

Tout programme informatique a pour vocation première de manipuler des données (bien souvent des quantités numériques, mais nous verrons que l'on peut également manipuler des chaînes de caractères), afin de produire un résultat. Ces quantités sont rangées en mémoire et la grande majorité des langages de programmation permettent d'y accéder via des noms de variables, que l'on appelle identificateurs.

Un identificateur commence nécessairement par une lettre, mais il peut contenir également des chiffres. VAR2 sera donc acceptable, contrairement à 2VAR. On proscrira également l'utilisation de caractères spéciaux (excepté le blanc souligné) ainsi que des espaces. Pour plus de lisibilité, on pourra remplacer l'identificateur MAVARIABLE par MA_VARIABLE.

1.2.2. Les types de données

Nous avons vu dans la section précédente, que les identificateurs permettent de référencer des données. Intéressons-nous à présent au type de ces

données. On distingue en général les données dites scalaires (constituées d'un seul élément) des données structurées (vecteurs, tableaux...).

1.2.2.1 Les données scalaires

On parle aussi de données élémentaires. Elles sont constituées d'un unique élément, qui peut être :

Un entier, positif ou négatif, sans restriction sur sa taille. En pratique, dans la plupart des langages de programmation, un entier (ou integer) est codé sur 4 octets (1 octet = 8 bits), et est donc contraint dans l'intervalle $[-2^{31}, 2^{31} - 1]$ (un bit est réservé pour le signe).

Un nombre décimal (cette définition inclut ici également les nombres rationnels et irrationnels), sans restriction, ni sur sa taille, ni sur son nombre de décimales. Les nombres flottants en simple précision (float), codés 4 octets et les nombres en double précision (double) codés sur 8 octets.

Un booléen (bool) permettant de symboliser les valeurs logiques VRAI et FAUX. En machine, 1 bit seulement est nécessaire à leur stockage en mémoire.

Une chaîne de caractères (string) de taille illimitée, notée entre double cotes (e.g. "ceci est un chaîne de caractères").

1.2.2.2 Les données dimensionnées

On parle aussi de données structurées. Dans cette première partie de cours, nous ne considérerons que les tableaux. Il est possible de définir d'autres types de données dimensionnées.

Un tableau est un ensemble d'éléments scalaires de même type, organisés suivant des colonnes et des lignes.

1.2.3 La déclaration

En informatique, la déclaration indique la partie du code où on annonce que l'on va utiliser un identificateur de variable.

Elle comprend trois informations principales : le nom de l'identificateur, la nature (variable ou constante) et son type (type de données). Lorsque l'on

déclare une donnée structurée, on spécifie également son nombre de dimensions et sa taille.

1.2.4 L'affectation

L'affectation consiste à associer une donnée à un identificateur. On la note : IDENTIFICATEUR \leftarrow constante ou expression algébrique

Notons que la plupart des langages de programmation ont adopté le signe = pour symboliser l'affectation, ce qui introduit bien souvent des confusions parmi les étudiants. Contrairement à l'égalité mathématique, l'affectation n'est pas symétrique. La partie gauche de l'instruction contient obligatoirement un identificateur, tandis que la partie droite est au choix une constante numérique, un identificateur, ou bien encore une expression algébrique (comportant alors des constantes, des identificateurs et, comme nous le verrons plus tard, des fonctions).

Exemples :

- $PI \leftarrow 3.14$
- $B \leftarrow PI$

1.2.5 Les expressions algébriques

Les expressions algébriques opèrent sur les données numériques (entières ou réelles). Les opérations algébriques comprennent opérations arithmétiques, logiques, et autres. Les opérateurs arithmétiques sont les suivants :

+	Addition	A+B
-	Soustraction	A-B
*	Multiplication	A*B
/	Division	A/B
↑	Puissance	A**B
-	Moins unaire (signe)	-A

Exemple : A partir de la saisie de trois nombres écrire l'algorithme permettant de calculer leur somme, leur produit et leur moyenne.

Algorithme Math

Var

N1, N2, N3, Som, Pro, Moy (R  el)

D  but

  crire ("Entrer le premier nombre")
lire (N1)
  crire ("Entrer le deuxi  me nombre")
lire (N2)
  crire ("Entrer le troisi  me nombre")
lire (N3)
Som $\leftarrow N1 + N2 + N3$
*Pro $\leftarrow N1 * N2 * N3$*
Moy $\leftarrow Som / 3$ -- ou Moy $\leftarrow (N1+N2+N3) / 3$
  crire (" La somme est: ", Som)
  crire (" Le produit est : ", Pro)
  crire (" la moyenne est : ", Moy)

Fin

Op  rateurs de comparaison

Les op  rateurs de comparaison (relationnels) servent    comparer deux variables et    fournir un r  sultat vrai ou faux.

<	Inf��rieur ��
<=	Inf��rieur ou ��gal ��
>	Sup��rieur ��
>=	Sup��rieur ou ��gal ��
=	��gal ��
<>	Diff��rent de

Chap 2 Les instructions conditionnelles et les boucles

Les instructions de contr  le permettent de modifier le flux d'ex  cution du code. Notons que contrairement aux instructions d  taill  es pr  c  demment, ce nouveau type d'instruction ne modifie pas directement l'  tat de la machine, mais seulement l'ordre d'ex  cution des instructions   l  mentaires.

Il en existe deux types principaux :

- Les boucles sont utilisées pour répéter un grand nombre de fois une série d'instructions (on parle d'itération pour définir une passe dans la boucle).
- Les débranchements permettent de quitter prématurément l'exécution d'une boucle d'instructions.

Ces deux types d'instructions de contrôle sont alors combinés pour créer un programme complexe.

2.1 La boucle tant que

Il s'agit d'une variation de la boucle à compteur, permettant de transcrire une séquence d'instructions du type :

Tant que (condition) alors faire quelque chose

En d'autres termes, on répète une liste d'instructions exécutables tant qu'une condition (formulée sous forme d'expression algébrique à valeur booléenne) est vérifiée.

Syntaxe :

Tant que <condition> Faire

Séquence d'instructions

Fin Tant que

Exemple 1 :

<code>i ← 1</code>		10
<code>Tant que i <= 5 FAIRE</code>		20
<code> ECRIRE (i*10)</code>		30
<code> i ← i + 1</code>		40
<code>Fin tant que</code>		50

Exemple 2 :

<code>...</code>		
<code>i ← 1</code>		10
<code>Tant que i <= 5 FAIRE</code>		20
<code> ECRIRE (i*10)</code>		40
<code> i ← i * 2</code>		
<code>Fin tant que</code>		

Exemple d'un programme de calcul de la commission. Si le chiffre d'affaire est supérieur à 500 millions, le taux de commission est de 5%, s'il est inférieur mais supérieur à 100 millions, le taux est de 3%, sinon elle est de 0%. Le programme est écrit en langage de haut niveau Python

```

import os
import sys
f=float(input("entrez le chiffre d affaire: "))
if f>500000000:
    commission=f*5/100
    print(commission)
elif f>100000000:
    commission=f*3/100
    print(commission)
else:
    commission=0
    print(commission)

```

2.2 Les boucles répéter... JUSQU'A

A la différence de la boucle tant que, la boucle jusqu'à ce que n'est plus réitérée dès que la condition exprimée. La syntaxe est la suivante :

Répéter

<Séquence d'instructions>

JUSQU'A <Condition>

L'exécution de l'algorithme, toutes les instructions écrites entre Répéter et Jusqu'à sont exécutées au moins une fois et leur exécution sera répétée jusqu'à ce que la condition placée derrière Jusqu'à soit satisfaite.

2.3 La boucle POUR...FAIRE

Pour<compteur>de<valeur initiale>à<valeur finale>[par pas de p] faire

<sequenced'instructions>

Fin pour

Ou

Compteur une variable entière de contrôle appelée index

Valeur Initiale est une expression à valeur entière dont la valeur sera la valeur initiale

Valeur Finale est une expression à valeur entière dont la valeur sera la valeur finale

P est une expression dont la valeur sera le pas d'avancement. Il est facultatif, s'il est omis, le passera pris par défaut égal à 1.

Le paramètre compteur reçoit la valeur initiale au moment d'accès à la boucle, puis à chaque parcours, il passe automatiquement à la valeur suivante dans son domaine jusqu'à atteindre la valeur finale.

Exemple 1:

```
Algorithme ESSAI
Var
  i, N (entier)
Début
  N ← 5
  Pour i de 1 à N faire
    écrire (i)
  N ← 0
Fin pour
fin
```

L'algorithme ci-dessous affiche 1,2, 3, 4, 5,

Exemple 2 :

L'algorithme lit 10 notes et calcule leur somme et leur moyenne

```
Algorithme somme_moyenne
VAR
  note, moyenne, somme(réel)
  i (entier)
Début
  somme ← 0
  Pour i de 1 à 10 faire
    Lire (note)
    somme ← somme + note
  Fin pour
  Ecrire ("la somme est", somme)
  moyenne ← somme /10
  Ecrire ("la moyenne est", moyenne)
Fin
```

Exemple de programme (en Python) qui affiche la somme des éléments inférieur au nombre n saisi.

```
import os
import sys
```

```

somme=0
n=int(input("entre la valeur de n = "))
for i in range(1,n+1):
    somme=somme+i
print(somme)

```

2.4 Les instructions conditionnelles si..... si non

Ils permettent de faire les choix conditionnés dans un algorithme.

Exemple

```

variables
age : entier
debut
    ecrire "veuillez donner votre age"
    lire age
    si (age < 13)
        alors ecrire "trop jeune, allez voir un autre film !"
        sinon ecrire "entrez, bonne soiree."
    fin

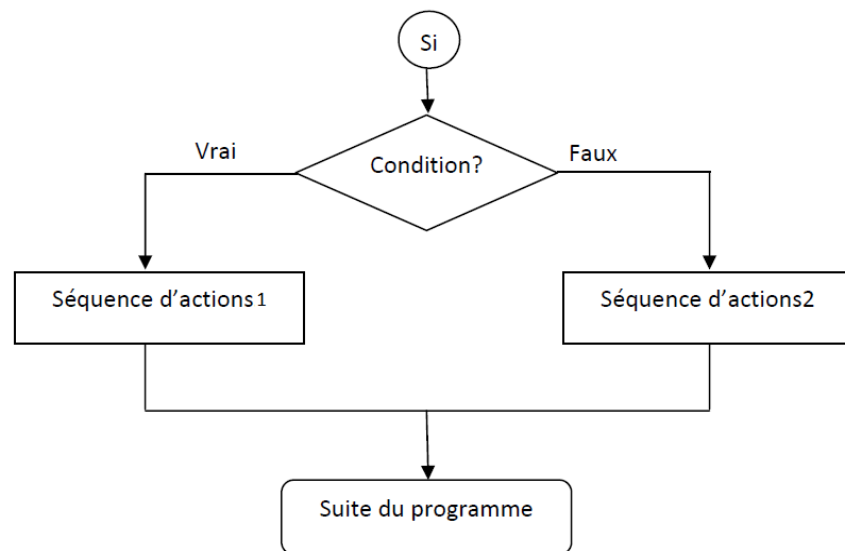
```

L'instruction SI (AGE < 13) ALORS écrire "Trop jeune, allez voir un autre film !" SINON écrire "Entrez, bonne soirée." est une instruction conditionnelle alternative: l'un ou l'autre des messages "Trop jeune, allez voir un autre film !" et "Entrez, bonne soirée." s'affiche selon la valeur de la condition AGE < 13.

Pour une meilleure lisibilité de votre algorithme, les mots ALORS et SINON doivent être décalées vers la droite par rapport au mot SI.

Lorsque plusieurs instructions doivent être effectuées après un même alors ou même sinon, nous indiquons qu'elles forment une séquence en les plaçant

entre crochet ouvrant. L'exécution de cette instruction se déroule selon



l'organigramme

Forme imbriquée de si est la suivante :

Si condition 1 Alors

instruction (s)1

Sinon

Si condition 2 Alors

instruction(s)2

Sinon

Si condition N-1 Alors

instruction(s)N-1

Sinon

instruction(s)N

Fin si

Exemple :

Ecrire un algorithme qui permet de saisir deux entiers A et B puis teste si $A > B$ ou $A < B$ ou $A = B$.

Exemple de calcul des racines d'une équation

```
import os
```

```
import sys
```

```

import math
a= int(input("saisir la valeur de a:"))
b= int(input("saisir la valeur de b:"))
c= int(input("saisir la valeur de c:"))
det=math.pow(b,2)-4*a*c
print(det)
if det>0:
    x1=(-b-math.sqrt(det))/2*a
    x2=(-b+math.sqrt(det))/2*a
    print(x1)
    print(x2)
elif(det==0):
    x1=-b/2*a
else:
    print("pas de racine")

```

2.5 Structure conditionnelle à choix multiple

Syntaxe

Selon <sélecteur> faire

<liste de valeurs1> : <traitement 1>

<liste de valeurs2> : <traitement 2>

.....

<liste de valeursN> : <traitement N>

Sinon

<traitement N+1>

Fin selon

Exemple:

Ecrire un algorithme qui permet de lire un numéro de jour de la semaine (compris entre 1 et 7) et d'afficher le nom du jour en toute lettre.

Chapitre. 3 Les tableaux

De nombreux objets traités par les programmes ne peuvent pas être représentés à l'aide d'un seul nombre ou d'une chaîne, mais sont constitués naturellement de plusieurs informations. Les données peuvent être représentées sous la forme de :

- Vecteur ou (tableau uni (mono)dimensionnel)
- matrice
- Pile;
- File simple ou double
- Liste
- Les arbres
- ...etc

Une structure de données est une manière d'organiser et de stocker l'information. Pour en faciliter l'accès ou dans d'autres buts. Une structure de données a une interface qui consiste en un ensemble de procédures pour ajouter, effacer, accéder, réorganiser, etc. les données.

Une structure de données conserve des données et éventuellement des métadonnées. Par exemple : un tas utilise un tableau pour stocker les clés et une variable A.

« 3.1. Les listes

Les listes sont appelées vecteurs ou tableaux unidimensionnels. La gestion d'une collection de données (nombres, booléens ou chaînes de caractères) se faisait systématiquement via un tableau (que l'on appelle aussi parfois improprement liste).

L'avantage d'une telle gestion des données est de permettre un accès simple à toute donnée, sous réserve bien entendu de connaître l'indice de sa

position dans le tableau. Ainsi, chaque élément peut être récupéré en un temps constant, indépendamment de la taille de la liste.

Il s'agit de la structure de données la plus simple possible, présente dans tout langage de programmation digne de ce nom (généralement sous l'appellation array).

Exemple d'affichage d'une liste en python

```
import os
import sys
liste=[]
n=int(input("saisir n"))
for i in range(n):
    element=int(input("saisir"))
    liste.append(element)
print(liste)
```

3.1.1. Déclaration

Dans tous les langages, le compilateur doit nécessairement connaître le nombre d'éléments d'un tableau (sa taille). Cette information lui permet de réserver l'emplacement mémoire correspondant. De même, comme pour une variable. Exemple de déclaration

```
Var
i, j (entier)
T1 (Tableau [1..10] de entier)
```

Le Tableau T1 comporte 10 cases pouvant contenir des entiers, les indices de ces cases varient de 1 à 10.

Exemple 1 :

Soit un tableau T ayant N éléments entiers. Calculer la somme de ses éléments.

```

...
Début
  lire (N)
  pour i de 1 à N faire
    lire T[i]
  fin pour
  S ← 0
  pour i de 1 à N faire
    S ← S + T[i]
  fin pour
  écrire (S)
Fin

```

Exemple 2 :

Soit un tableau Tab ayant M éléments entiers. Compter le nombre d'éléments nuls.

```

Début
  lire (M)
  pour i de 1 à M faire
    lire (Tab[i])
  fin pour
  NB ← 0
  pour i de 1 à M faire
    Si Tab[i] = 0 alors
      NB ← NB + 1
    fin si
  fin pour
  écrire (NB)
Fin

```

Exemple 3 :

Soit Tab le tableau monodimensionnel. Ecrire un algorithme pour retrouver l'élément minimal :

```

Fonction minimum
Tab[ 1..n]  (entier)
var i, min (entier)
Début
  Ecrire (Tab[i])
  Lire(Tab[i])
  min←Tab[1]
  pour i allant de 2 à n faire
    si min>Tab[i] alors
      min←T[i]
    fin si
  fin pour
  Ecrire (min)
Fin

```

Exemple 4 :

Soit Tab le tableau unidimensionnel. Ecrire un algorithme qui range par ordre croissant les éléments du tableau:

```
Tab[ 1..n]  (entier)
var i, j, t (entier)
Début
    Ecrire (T[i])
    Lire (Tab[i])
    pour i allant de 1 à n faire
        pour j allant de 2 à n faire
            si Tab[i]>Tab[i+1] alors
                t←Tab[j]
                Tab[j]←Tab[j+1]
                Tab[j+1]←t
            Fin si
        Fin pour j
    Ecrire (Tab[i])
Fin pour i
Fin
```

Exemple 5 : Fusion de tableaux triés

```
Fonction fusion (T1 [1..N1] (entier)
T2 [1..N2] (entier)
T3[1..N1+N2] (entier)
var I1,I2,i (entier)
début
    I1←1
    I2←1
    pour i allant de 1 à N1+N2 faire
        si T1[I1]≤T2[I2] alors
            T3[i]←T1[I1];
            I1←I1+1
        sinon T3[i]←T2[I2]
            I2←I2+1;
        Fin si non
    Fin si
    Ecrire(T3[i])
Fin pour;
Fin
```

Quelques methodes définies en Python, appliquée aux listes

Le type de données liste possède d'autres méthodes. Voici toutes les méthodes des objets listes :

append(x)

Equivalent à `a.insert(len(a), x)`.

extend(L)

Rallonge la liste en ajoutant à la fin tous les éléments de la liste donnée ; équivaut à `a[len(a):] = L`.

insert(i, x)

Insère un élément à une position donnée. Le premier argument est l'indice de l'élément avant lequel il faut insérer, donc `a.insert(0, x)` insère au début de la liste, et `a.insert(len(a), x)` est équivalent à `a.append(x)`.

remove(x)

Enlève le premier élément de la liste dont la valeur est x. Il y a erreur si cet élément n'existe pas.

pop([i])

Enlève l'élément présent à la position donnée dans la liste, et le renvoie. Si aucun indice n'est spécifié, `a.pop()` renvoie le dernier élément de la liste. L'élément est aussi supprimé de la liste.

index(x)

Retourne l'indice dans la liste du premier élément dont la valeur est x. Il y a erreur si cet élément n'existe pas.

Quelques exercices sur les listes

1. Affichage des éléments de la liste

```
import os

import sys

liste=[]

n=int(input("Saisir la taille du vecteur n = "))

for i in range(n):

    element=int(input("Element du vecteur = "))

    liste.append(element)

print("le premier element est ",liste[0]) // Affichage du premier élément

print("le dernier element est ",liste[n-1]) // Affichage du dernier élément

print("L element minimal est ",min(liste)) // Affichage de l'élément minimal

t=min(liste)

q=liste.index(t)

m=max(liste)

l=liste.index(m)

print("l indice de l element minimal",q)
```

```

print("LA LISTE TRIEE EST LA SUIVANTE")

liste.sort()  // Tri des éléments de la liste

print(liste)

print("=====
=====")

liste2=[]

n=int(input("Saisir la taille de la deuxieme liste n = "))

for i in range(n):

    element=int(input("Element du vecteur = "))

    liste2.append(element)  // Fusion de deux matrice

    longueur=len(liste2)  //Longueur de la matrice

print("La longueur de la matrice",longueur)

print("=====
=====")

print("la concatenation de deux liste est")

liste.extend(liste2)

print(liste)

```

3.2. Les matrices

Les matrices ou tableaux bidimensionnels contiennent un certain nombre de cases, auxquelles on peut accéder par un index. Les cases du tableau bidimensionnel sont organisées sous la forme d'un certain nombre de lignes et de colonnes. On peut créer des tableaux avec autant de dimensions que l'on souhaite, même si la plupart du temps, on se limitera à deux ou trois.

Pour déclarer une variable comme étant un tableau à plusieurs dimensions, on généralise le principe de la déclaration à une dimension, en

plaçant entre crochets, après l'identifiant, le nombre de cases dans chaque dimension.

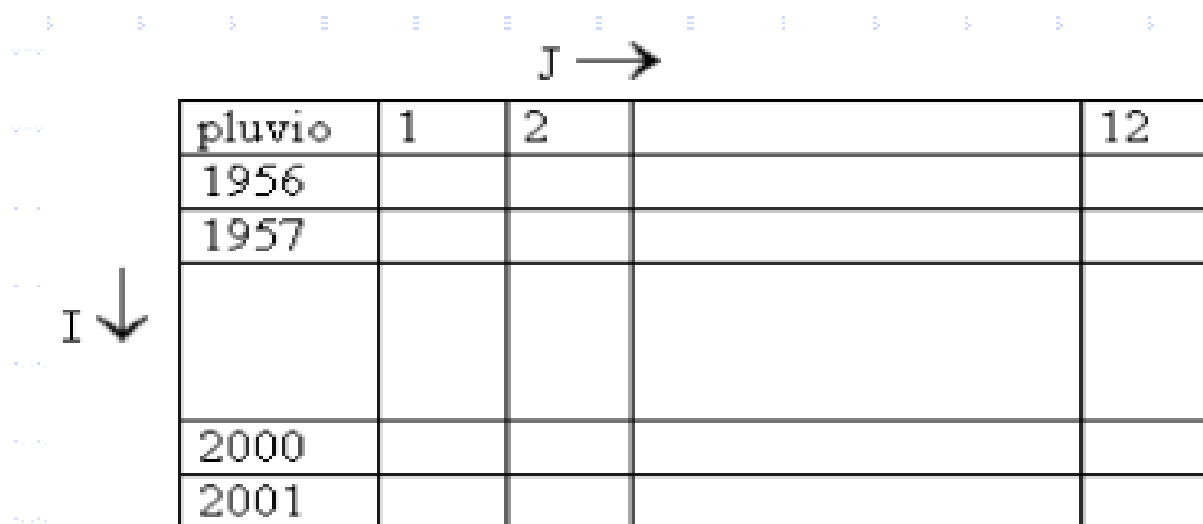
Les tableaux bidimensionnels sont des structures des données de mêmes types constitués d'un certain nombre de cases, qui forment les lignes et les colonnes auxquelles on peut accéder par un index.

Exemple applicatif des matrices:

Considérons qu'on s'intéresse aux relevés pluviométriques d'une certaine station climatique de 1956 à 1998, et, pour chaque année, on dispose des relevés mois par mois.

On parle alors de tableau bidimensionnel car pour localiser un relevé pluviométrique, deux indices sont nécessaires :

- un premier indice de lignes **I**, qui va repérer l'année
- un deuxième indice de colonnes, noté par **J**, qui repère le mois.



The diagram illustrates a 2D array structure. Above the table, a horizontal arrow labeled 'J' points to the right, indicating the column index. To the left of the table, a vertical arrow labeled 'I' points downwards, indicating the row index. The table itself has 6 rows and 5 columns. The first row is the header, with the first column labeled 'pluvio' and the last column labeled '12'. The second and third rows are labeled '1956' and '1957' respectively in the first column. The fifth and sixth rows are labeled '2000' and '2001' respectively in the first column. The second and third columns are labeled '1' and '2' in the header row. The fourth column is empty in the header row.

pluvio	1	2		12
1956				
1957				
2000				
2001				

Les tableaux bidimensionnels sont très utilisés en particulier en mathématiques avec la notion de matrice (calcul matriciel, analyse numérique, par exemple résolution de systèmes d'équations).

Un tableau de **NL**-lignes et **NC**-colonnes est une suite de **NL X NC** variables **T [L,C]**.

3.2.1. Déclaration d'un tableau bidimensionnel

Il est indispensable de déclarer un tableau avant une première utilisation:

Id_tableau = tableau [1..NL, 1..NC] de type-element

Les instructions habituelles (lecture, écriture, affectation,..) s'appliquent aussi aux tableaux à 2 dimensions.

Exemple 1 : Lecture des éléments d'une matrice :

```
Pour i de 1 à N faire
  Pour j de 1 à N faire
    Lire T[i,j]
  Fin pour
Fin pour
```

Exemple 2, de traitement des éléments de la matrice :

L'algorithme de calcul de nombre d'éléments négatifs par colonnes :

```
A [1..N][1..M] (entier)
var j,i (entier)
B[1..K]
début
  Ecrire la matrice A[i,j]
  lire la matrice A[i,j]
  // le nombre d'éléments négatifs par colonnes
  pour i allant de 1 à N faire
    pour j allant de 1 à M faire
      si A[i][j]<0
        B[i]++
      Fin si
    fin pour j
  B[i]
Fin pour i
fin
```

Exemple 3 : Dans un tableau T de N lignes et M colonnes, combien d'éléments sont nuls?

```
Compt ← 0
Pour i de 1 à N faire
  Pour j de 1 à M faire
    Si T[i,j] = 0 alors
      Compt ← Compt + 1
    Fin si
  Fin pour
Fin pour
```

Quelques fonctions du langage python appliquées aux matrices et exemple d'application

```
import numpy as np // Importation de la bibliotheque numpy pour creer la matrice

a = np.array([[1,2],[3,1],[1,4]])

print(a)

print(a.dtype) //type de données de la matrice

print(a.size), // le nombre d'éléments donc ligne fois colonne

a = np.arange(0,2).reshape(1,2) // Composition de la matrice par les éléments situe dans l
interval (0 ,2)

print(a)

a = np.array([3,4,6,9,5,2])

b = a.reshape(3,2) //Transformation de la liste en matrice

print(b)

// ajout des colonnes

a = np.array([[1,2,5],[3,2,1],[1,1,4]])

d = np.array([[7],[6],[4]])

print(np.append(a,d,axis=1))

// Redimensionnement

d = np.resize(a,new_shape=(4,3))

print(d)

//Somme des colonnes

somm= np.sum(d,axis=0)

print(somm)

//sommendes lignes

somm= np.sum(d,axis=1)

print(somm)

// Element maxi par lignes

print(np.max(d,axis=1))

// Element maxi par colonnes

print(np.max(d,axis=0))
```



```

// Moyenne des colonnes
print(np.mean(d,axis=0))

// Moyenne des lignes
print(np.mean(d,axis=1))

// Transpose de la matrice
print(np.transpose(d))

// Produit de deux matrice
print(np.dot(a,m))

// Calcul du determinant
print(np.linalg.det(m))

//matrice symétrique
o = np.dot(np.transpose(t),t)
print(o)

```

3.3 Quelques algorithmes de tri

3.3.1 Algorithme de tri à bulles

Le tri à bulles est un algorithme de tri classique. Son principe est simple, et il est très facile à implémenter.

On considère un tableau de nombres T , de taille N . L'algorithme parcourt le tableau, et dès que deux éléments consécutifs ne sont pas ordonnés, les échange. Après un premier passage, on voit que le plus grand élément se situe bien en $_n$ de tableau. On peut donc recommencer un tel passage, en s'arrêtant à l'avant-dernier élément, et ainsi de suite. Au i -ème passage on fait remonter le i -ème plus grand élément du tableau à sa position définitive, un peu à la manière de bulles qu'on ferait remonter à la surface d'un liquide, d'où le nom d'algorithme de tri à bulles.

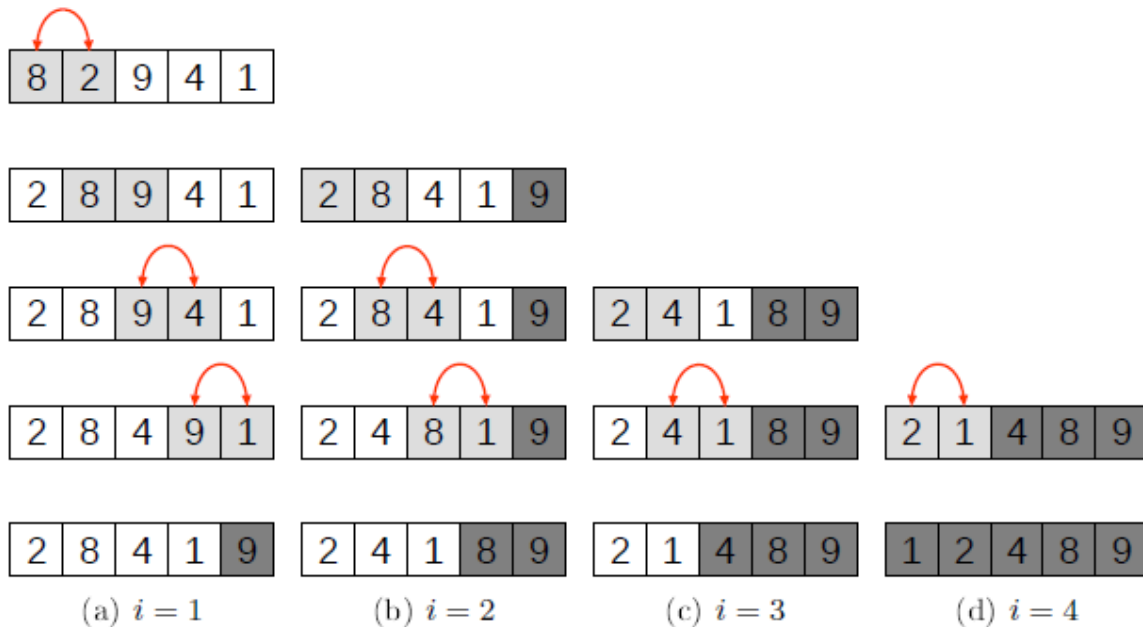
On donne ci-dessous l'algorithme de tri à bulles en pseudo-code.

```

procedure TRI_BULLES( $T$ )
   $N \leftarrow$  taille de  $T$ 
  pour  $i$  de  $N - 1$  à 1 faire
    pour  $j$  de 0 à  $i - 1$  faire
      si  $T[j] > T[j + 1]$  alors
        échanger  $T[j]$  et  $T[j + 1]$ 
      fin si
    fin pour
  fin pour
fin procedure

```

L'exécution de l'algorithme de tri à bulles se fait selon le schéma suivant :



Les cases gris clair représentent les éléments comparés, les flèches rouges les échanges d'éléments, et les case gris sombre les éléments placés définitivement.

L'algorithme de tri à bulles a une complexité en temps en (N^2) en pire cas, où N est la taille du tableau. Le pire cas correspond ici au cas où le tableau est initialement trié par ordre décroissant : dans ce cas l'algorithme doit faire remonter chaque élément jusqu'à la i -ème place à chaque étape i , en effectuant à chaque fois un échange. La complexité en temps indique ici que le temps pris

par l'algorithme pour trier un tableau de taille N augmente quadratiquement en fonction de N lorsque N est très grand.

On peut parler également de complexité en temps en moyenne, qui correspond pour simplifier au temps que prend l'algorithme pour un tableau de taille N tiré au hasard.

Ici, la complexité en moyenne du tri à bulles est également en (N^2) , ce qui le rend peu efficace comparé à d'autres algorithmes de tri comme le tri fusion ou le tri rapide. Le tri à bulle est en fait un des algorithmes de tri les plus lents, il est donc rarement utilisé en pratique.

3.3.2. Algorithme de tri par insertion

Le tri par insertion est également un algorithme de tri classique, simple à implémenter et intuitif, puisqu'il est celui que les joueurs de cartes utilisent naturellement pour trier leurs cartes.

On considère un tableau de nombres T de taille N qu'il s'agit de trier par ordre croissant.

Le principe de l'algorithme est le suivant. On parcourt le tableau du début à la fin ($i = 1$ à $N-1$), et à l'étape i , on considère que les éléments de 0 à $i-1$ du tableau sont déjà triés. On va alors placer le i -ème élément à sa bonne place parmi les éléments précédents du tableau, en le faisant « redescendre » jusqu'à atteindre un élément qui lui est inférieur.

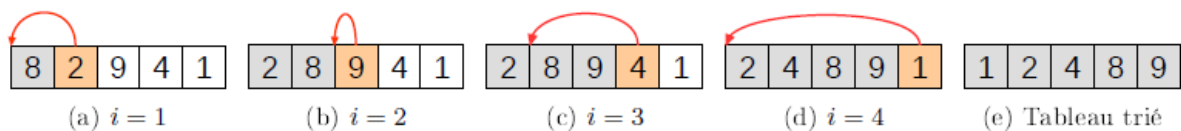
On donne ci-dessous l'algorithme de tri par insertion en pseudo-code.

```

procedure TRI_INSERTION( $T$ )
   $N \leftarrow$  taille de  $T$ 
  pour  $i$  de 1 à  $N - 1$  faire
     $x \leftarrow T[i]$ 
     $j \leftarrow i$ 
    tant que  $j > 0$  et  $T[j - 1] > x$  faire
       $T[j] \leftarrow T[j - 1]$ 
       $j \leftarrow j - 1$ 
    fin tant que
     $T[j] \leftarrow x$ 
  fin pour
fin procedure

```

L'exécution de l'algorithme de tri à bulles se fait selon le schéma suivant :



L'algorithme de tri par insertion a une complexité en temps en (N^2) en pire cas et en moyenne, où N est la taille du tableau. Le pire cas correspond au cas où le tableau est initialement trié par ordre décroissant : dans ce cas l'algorithme doit faire redescendre jusqu'au début le i -ème élément du tableau, à chaque étape i . Le tri par insertion est donc également en moyenne moins efficace que d'autres algorithmes de tri, mais il est tout de même rapide lorsque le tableau considéré est déjà « presque » trié, ou de petite taille.

3.3.3. Algorithme de tri rapide

L'algorithme de tri rapide (quicksort) a été inventé par Charles Antony Richard Hoare (C.A.R. Hoare) en 1961, et est basé sur le principe « diviser pour régner ».

Ce principe, souvent utilisé en algorithmique, consiste à diviser le problème à résoudre en sous-problèmes, eux-mêmes divisés à leur tour, et ainsi de suite, jusqu'à arriver à des problèmes simples, dont la résolution permet d'obtenir simplement la solution du problème de départ.

On considère un tableau T de N nombres. On désigne le dernier élément du tableau comme pivot, et on va alors parcourir le tableau jusqu'à l'avant-dernier élément pour séparer les éléments strictement inférieurs au pivot et les éléments supérieurs au pivot, en créant deux nouveaux tableaux. Cette étape est appelée partitionnement. Une fois cette opération effectuée, on peut trier chacun des deux tableaux de manière récursive (la fonction de tri s'appelle elle-même, mais sur des tableaux strictement plus courts), puis reconstituer le tableau trié en concaténant le tableau trié des éléments plus petits que le pivot, le pivot, et le tableau trié des éléments plus grands que le pivot.

On voit que la fonction de tri va s'appeler elle-même sur des tableaux de plus en plus courts, jusqu'à arriver à des tableaux de 0 ou 1 élément, qui sont naturellement triés. C'est vraiment le principe « diviser pour régner ».

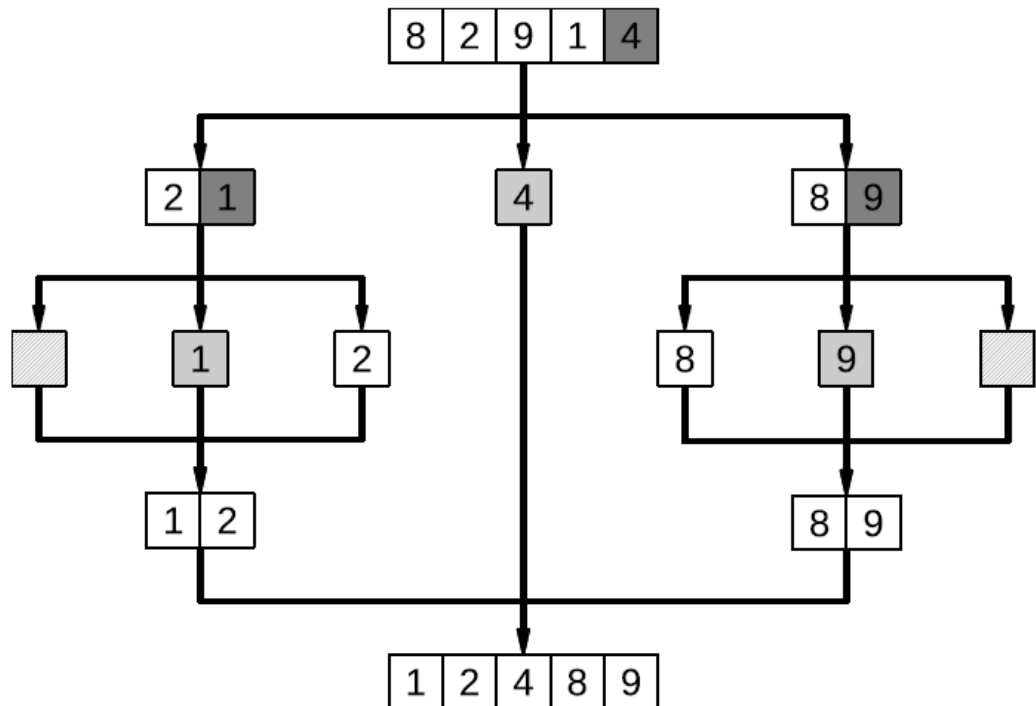
On donne donc ci-dessous l'algorithme de tri rapide en pseudo-code.

```

procedure TRI_RAPIDE( $T$ )
   $N \leftarrow \text{longueur}(T)$ 
  si  $N > 1$  alors
     $\text{pivot} \leftarrow T[N - 1]$ 
     $T_{\text{inf}}, T_{\text{sup}} \leftarrow \text{partitionner}(T)$ 
     $T_{\text{trie}} \leftarrow \text{concaténer } T_{\text{inf}}, \text{pivot et } T_{\text{sup}}$ 
    retourner  $T_{\text{trie}}$ 
  sinon
    retourner  $T$ 
  fin si
fin procedure

```

Arbre représentant l'exécution de l'algorithme de tri rapide est la suivante :



L'algorithme de tri rapide a une complexité en temps en pire cas en (N^2), lequel a lieu lorsque le tableau est trié par ordre décroissant, mais une complexité en moyenne en ($N\log(N)$), ce qui le rend nettement plus efficace que les deux algorithmes vus précédemment. Le tri rapide est donc largement utilisé en pratique.

Chapitre. 4 Programmation graphique avec Python

4.1. Notions générales

Le langage de programmation Python est un langage de haut niveau. Il présente aussi les avantages suivants :

- facile à apprendre, à lire, à comprendre et à écrire ;
- portable (fonctionne sous de nombreux systèmes d'exploitation) ;
- adapté aussi bien pour des scripts, des petits ou gros projets ;
- doté d'une façade objet bien conçue et puissante ;
- possède une communauté active autour du langage.

Python est un langage puissant, à la fois facile à apprendre et riche en possibilités. Il dispose de nombreuses fonctionnalités intégrées.

Ainsi, il existe ce qu'on appelle des bibliothèques qui aident le développeur à travailler sur des projets particuliers. Plusieurs bibliothèques peuvent ainsi être installées pour, par exemple, développer des interfaces graphiques en Python.

Python est un langage de programmation interprété, c'est-à-dire que les instructions que vous lui envoyez sont « transcrites » en langage machine au fur et à mesure de leur lecture. D'autres langages (comme le C / C++) sont appelés « langages compilés » car, avant de pouvoir les exécuter, un logiciel spécialisé se charge de transformer le code du programme en langage machine. On appelle cette étape la « compilation ». À chaque modification du code, il faut rappeler une étape de compilation. Les avantages d'un langage interprété sont la simplicité (on ne passe pas par une étape de compilation avant d'exécuter son programme) et la portabilité (un langage tel que Python est censé fonctionner aussi bien sous Windows que sous Linux ou Mac OS, et on ne devrait avoir à effectuer aucun changement dans le code pour le passer d'un système à l'autre).

La méthode import

Lorsque vous ouvrez l'interpréteur Python, les fonctionnalités du module math ne sont pas incluses. Il s'agit en effet d'un module, il vous appartient de l'importer si vous vous dites. Nous allons voir une première syntaxe d'importation.

```
>>> import math
```

La syntaxe est facile à retenir : le mot-clé import, qui signifie « importer » en anglais, suivi du nom du module, ici math. Après l'exécution de cette instruction, rien ne se passe... en apparence. En réalité, Python vient d'importer le module math.

Toutes les fonctions mathématiques contenues dans ce module sont maintenant accessibles. Pour appeler une fonction du module, il faut taper le

nomdu module suivi d'un point « . » puis du nomde la fonction. C'est la même syntaxe pour appeler des variables du module.

4.2. Programmation graphique

Nous allons créer des interfaces graphiques à l'aide d'un module présent par défaut dans Python : Tkinter. Ce module permet de créer des interfaces graphiques en offrant une passerelle entre Python et la bibliothèque Tk. On pourra pouvoir à créer des fenêtres, créer des boutons, faire réagir nos objets graphiques à certains évènements...

Présentation de Tkinter Tkinter (Tk interface) est un module intégré à la bibliothèque standard de Python, bien qu'il ne soit pas maintenu directement par les développeurs de Python. Il offre un moyen de créer des interfaces graphiques via Python. Tkinter est disponible sur Windows et la plupart des systèmes Unix. Les interfaces que vous pourrez développer auront donc toutes les chances d'être portables d'un système à l'autre. Notez qu'il existe d'autres bibliothèques pour créer des interfaces graphiques. Tkinter a l'avantage d'être disponible par défaut, sans nécessiter une installation supplémentaire. Pour savoir si vous pouvez utiliser le module Tkinter via la version de Python installée sur votre système, tapez dans l'interpréteur en ligne de commande Python : `from tkinter import *`

Ci bas le code minimal pour créer une fenêtre avec Tkinter.

```
from tkinter import *  
fenetre = Tk()  
champ_label = Label(fenetre, text="Salut a tous et toutes !")  
champ_label.pack()  
fenetre.mainloop()
```

Nos objets graphiques (boutons, champs de texte, cases à cocher, barres de progression...) sont appelés des widgets. On peut préciser plusieurs options lors de la construction de nos widgets. Ici, on définit l'option text de notre Label à Salut a tous et toutes

Il existe d'autres options communes à la plupart des widgets (la couleur de fond bg, la couleur du widget fg, etc.) et d'autres plus spécifiques à un certain type de widget. Le Label par exemple possède l'option text représentant le texte affiché par le Label.

On peut modifier des options lors de la création du widget. Mais on peut aussi en modifier après :

```
>>> champ_label["text"] 'Salut les Zér0s !'  
>>> champ_label["text"] = "Maintenant, au revoir !"  
>>> champ_label["text"] 'Maintenant, au revoir !' >>>
```

. Les widgets les plus communs

Les labels

C'est le premier widget que nous avons vu, hormis notre fenêtre principale qui en est un également. On s'en sert pour afficher du texte dans notre fenêtre, du texte qui ne sera pas modifié par l'utilisateur.

```
champ_label = Label(fenetre, text="contenu de notre champ label")  
champ_label.pack()
```

Les boutons

Les boutons sont des widgets sur lesquels on peut cliquer et qui peuvent déclencher des actions ou commandes comme nous le verrons ultérieurement plus en détail.

```
bouton_quitter = Button(fenetre, text="Quitter",  
command=fenetre.quit)  
bouton_quitter.pack()
```

Une ligne de saisie

En fait de zone, ils'agit d'une ligne simple. On préférera créer une variable Tkinter associée au champ de texte.

```
var_texte = StringVar()  
ligne_texte = Entry(fenetre, textvariable=var_texte, width=30)  
ligne_texte.pack()
```

Les cases à cocher

Les cases à cocher sont définies dans la classe `Checkbox`. Là encore, on utilise une variable pour surveiller la sélection de la case. Pour surveiller l'état d'une case à cocher (qui peut être soit active soit inactive), on préférera créer une variable de type `IntVar` plutôt que `StringVar`, bien que ce ne soit pas une obligation.

```
var_case = IntVar()
case = Checkbox(fenetre, text="Ne plus poser cette question",
variable=var_case)
case.pack()
```

Les boutons radio

Les boutons radio (radio buttons en anglais) sont des boutons généralement présentés en groupes. C'est, à proprement parler, un ensemble de cases à cocher mutuellement exclusives : quand vous cliquez sur l'un des boutons, celui-ci sélectionne et tous les autres boutons du même groupe se désélectionnent. Ce type de bouton est donc surtout utile dans le cadre d'un regroupement.

```
var_choix = StringVar()
choix_rouge = Radiobutton(fenetre, text="Rouge", variable=var_choix,
value="rouge") choix_vert = Radiobutton(fenetre, text="Vert", variable=var_choix,
value="vert") choix_bleu = Radiobutton(fenetre, text="Bleu", variable=var_choix,
value="bleu")
choix_rouge.pack()
choix_vert.pack()
choix_bleu.pack()
```

Les listes déroulantes

Ce widget permet de construire une liste dans laquelle on peut sélectionner un ou plusieurs éléments. Le fonctionnement n'est pas tout à fait identique aux boutons radio. Ici, la liste comprend plusieurs lignes et non un groupe de boutons. Créer une liste se fait assez simplement, vous devez commencer à vous habituer à la syntaxe :

```
liste = Listbox(fenetre)
```

```
liste.pack()
```

On insère ensuite des éléments. La méthode `insert` prend deux paramètres : 1. la position à laquelle insérer l'élément ; 2. l'élément même, sous la forme d'une chaîne de caractères. Si vous voulez insérer des éléments à la fin de la liste, utilisez la constante `END` définie par Tkinter

```
liste.insert(END, "Pierre")  
liste.insert(END, "Feuille")  
liste.insert(END, "Ciseau")
```

Pour accéder à la sélection, utilisez la méthode `curselection` de la liste. Elle renvoie un tuple de chaînes de caractères, chacune étant la position de l'élément sélectionné. Par exemple, si `liste.curselection()` renvoie `('2',)`, c'est le troisième élément de la liste qui est sélectionné (Ciseau en l'occurrence).

Bibliographie

- 1 Jean-Michel Adam – Université Grenoble Alpes – UFR SHS – Département IMSS , Traduction de la notation algorithmique en langage Python
- 2 Guido van Rossum Fred L. Drake, Jr., editor, Tutoriel Python
- 3 Ricco Rakotomalala, Introduction a Python, programmation , les bases du langage, R.R. – Université Lyon 2
- 4 Abdelhamid DJEFFAL, Cours d'Algorithmique et structures de données 1, Université Mohamed Khider – Biskra, 2012/2013
- 5 M. Delest, initiation à l'algorithmique, Bordeaux ,2007 ;
- 6 Yann MENEROUX, Introduction `a l'Algorithmie; Ecole Nationale des Sciences Géographiques,2018 ;
- 7 Didier Müller, Introduction à la théorie des graphes ;
- 8 Florent Hivert ; Algorithmique Structures de données ; Université Paris Sud ;
- 9 Renaud Dumont , Algorithmique, Ulg ;2009-2010
- 10 Apprenez à programmer en Python // www.openclassrooms.com