Faculté des Sciences
كلية العلوم

Département d'Informatique
Filière MIP - Semestre 2
2023-2024

Informatique II

Algorithmique II / Python

Pr. Khadija Louzaoui

Grandes lignes du cours

- Introduction à l'algorithmique
- Les fonctions et les procédures
- La récursivité
- Les enregistrements et les fichiers
- La complexité
- La preuves des algorithmes

Faculté des Sciences
كلية العلوم

Département d'Informatique
Filière MIP - Semestre 2
2023-2024

Module : Informatique II (Algorithmique II / Python)

Chapitre 1

Introduction à l'algorithmique

Pr. Khadija Louzaoui

Introduction à l'algorithmique

- Introduction
- Structure générale d'un algorithme
- Les variables et les constantes
- Les instructions élémentaires
- Les instructions conditionnelles (les alternatives)
- Les instructions itératives (Les boucles)
- Les tableaux

Processus de réalisation d'un programme

La réalisation de programmes passe par l'écriture d'**algorithmes**

D'où l'intérêt de **l'algorithmique**

Où on apprend réellement à **programmer**

5

Algorithme

❑ Le terme **algorithme** est employé en **informatique** pour décrire une méthode de résolution de problèmes programmables sur machine.

❑Un **algorithme** décrit ce qui doit faire l'ordinateur pour arriver un but bien précis. Ce sont **les instructions** qu'on doit lui donner.

❑**Un algorithme** est un **ensemble d'instructions (nécessaires, ordonnées, précises , qui se terminent dans un temps fini)** et qui agissent sur un ensemble de **données** pour avoir un **résultat** .

❑La notion **d'algorithme** est à la **base** de toute la **programmation** informatique.

6

Algorithme

❑**L'algorithme** est un moyen pour le **programmeur** de présenter son approche d'un problème donné à d'autres personnes, dans un **langage** clair et compréhensible par l'être humain.

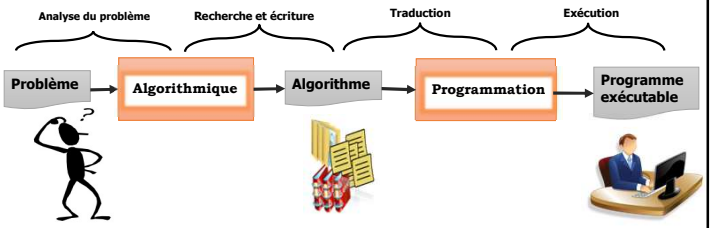
❑C'est un **pseudo-langage** qui est conçu pour résoudre les problèmes et applications sans aucune contrainte due aux langages de programmation et aux spécificités de la machine.

7

Algorithmique

❑L'algorithmique est **la science des algorithmes**. Elle s'intéresse à l'art de construire des algorithmes ainsi qu'à déterminer leur validité, leur robustesse, leur réutilisabilité, leur complexité ou leur efficacité.

❑L'algorithmique permet ainsi de passer d'un **problème** à résoudre à un **algorithme** qui décrit la démarche de résolution du problème.



Deux phases nécessaires pour obtenir un programme exécutable.

8

Phase d'algorithmique

- **Phase d'algorithmique** qui implique la recherche et l'écriture d'un **algorithme.**
- On doit définir les **données** qu'on dispose et les **objectifs** qu'on souhaite atteindre, ainsi que prévoir des réponses à tous les cas possibles.

Il s'agit de **repérer** les **données nécessaire** à la **résolution** du **problème**.

Il s'agit de **déterminer** toutes les **étapes** des **traitements** à faire et donc des " **instructions**" à **développer** pour arriver aux résultats.

Les **résultats** obtenus peuvent être **affichés** sur écran, ou **imprimés** sur papier, ou bien encore **conservés** dans un fichier.

Principe du traitement automatisé

Phase d'algorithmique

Lorsqu'on écrit un **algorithme**, les questions suivantes doivent être considérées :

- Quel est le résultat attendu ? (données de sortie)
- Quelles sont les données nécessaires (informations requises) ?
- Comment faire (le traitement à réaliser) ?

Exemple : Résolution d'une équation du premier degré **ax+b=0.**

- **Les données d'entrées sont :** a, b
- **La donnée de sortie est :** x
- **Traitement:**

  Etudier les cas suivants :

  - a=0 et b≠0
  - a=0 et b=0
  - a ≠ 0

Phase de programmation

- **Phase de programmation** qui consiste à traduire l'algorithme obtenu en un programme à l'aide d'un **langage de programmation** (**code source**).

  1-Le processeur **extrait** les *données* à traiter à partir de la source indiquée dans le programme (soit auprès de l'utilisateur qui devrait les introduire au moyen du clavier, soit en mémoire secondaire ou centrale).

  2- Il **exécute**, ensuite, la *série d'opérations* élémentaires de manière séquentielle (dans l'ordre prévu par le programme) et mémorise tous les résultats intermédiaires.

  3- Il **renvoie** enfin le ou les *résultats* attendus à la destination indiquée dans le programme.

Langage de programmation

❑Un **programme**, lorsqu'il est sous la forme de **code source**, est un texte qui exprime un **algorithme**, en vue de permettre **l'exécution** de ce dernier sur une machine.

❑Un **langage de programmation** permet d'écrire un **programme** (JavaScript, Python, Java, Typescript, C#, C++, PHP, C, Ruby…)

❑On peut distinguer deux grands types de langages : **les langages interprétés** et **les langages compilés**.

- Dans un **langage interprété**, le même code source pourra marcher directement sur **tout ordinateur**. Avec un **langage compilé**, il faudra (en général) tout **recompile**r à chaque fois ce qui pose parfois des soucis.
- Dans un **langage compilé**, le programme est directement **exécuté** sur l'ordinateur, donc il sera en général **plus rapide** que le même programme dans un **langage interprété**.

Langage de programmation

Python

Le langage de programmation Python a été créé en 1989 par Guido van Rossum.

Ce langage de programmation présente de nombreuses caractéristiques intéressantes :

- Multiplateforme.
- Gratuit.
- Langage de haut niveau.
- Langage interprété.
- Langage orienté objet.
- Simple à prendre.
- Très utilisé en bio-informatique et plus généralement en analyse de données et IA.

Toutes ces caractéristiques font que Python est désormais enseigné dans de nombreuses formations, du lycée à l'enseignement supérieur.

Structure générale d'un algorithme

Un algorithme est composé de trois parties principales :

L'en-tête : cette partie sert à donner un nom à l'algorithme et identifie le problème à résoudre.

- Elle est précédée par le mot clé ***Algorithme***.

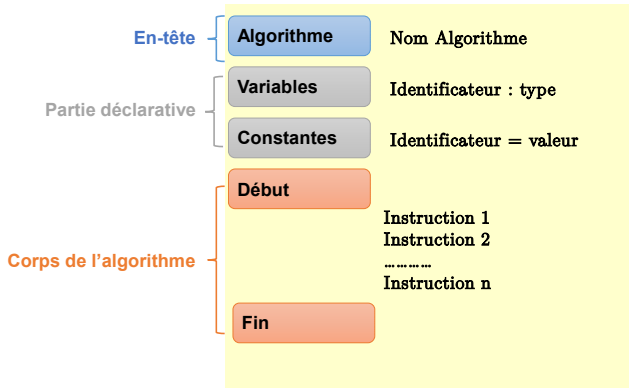
La partie déclarative : cette partie rassemble les déclarations des différents objets non primitifs que l'algorithme utilise dans son traitement.

- Elle est précédée par les mots ***variables***, ***constantes***, etc..

Le corps de l'algorithme : cette partie décrit le traitement de l'algorithme, elle est constituée d'une ou plusieurs séquences ***d'instructions*** faisant appel à des opérations de base à exécuter par l'ordinateur.

- Elle est délimitée par les mots **Début** et **Fin**.

Structure générale d'un algorithme



Structure générale d'un algorithme

Exemple : Un algorithme qui calcul la surface d'un cercle

Notation algorithmique

```
Algorithme SurfaceCercle
Variables r,s : réel
Constante pi = 3,141592
Début
    Ecrire ("Donner la valeur du rayon:")
    Lire (r)
    s ← r * r * pi
    Ecrire ("La surface du cercle est :", s )
Fin
```

Python

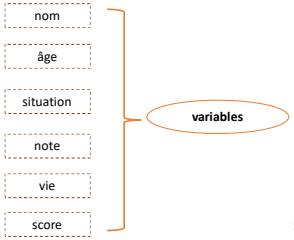
```
In [1]: pi=3.141592
        r= int(input("Donner la valeur du rayon :  :"))
        s=pi*r
        print("la surface du cercle est :",s)

        Donner la valeur du rayon :  :8
        la surface du cercle est : 25.132736
```

Notion de variable

❑Dans un programme, on va avoir en permanence besoin de stocker provisoirement des **valeurs**.

- Des données issues du disque dur, fournies par l'utilisateur (frappées au clavier).
- Des résultats obtenus par le programme, intermédiaires ou définitifs.
- Ces données peuvent être de plusieurs types : des nombres, du texte, etc.

❑Toujours dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une **variable**.



Notion de variable

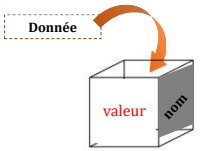
❑Pour employer une image, **une variable** est une **boîte**, que le programme (l'ordinateur) va repérer par une **étiquette** (**nom**).

❑Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.

❑On peut à chaque fois changer le contenu de la boite.

❑Une **variable** sert à stocker la **valeur** d'une **donnée** dans un langage de programmation.

❑Une **variable** désigne un **emplacement mémoire** dont le contenu peut changer au cours d'un programme (d'où le nom de **variable**).



Notion de variable

Variable et mémoire

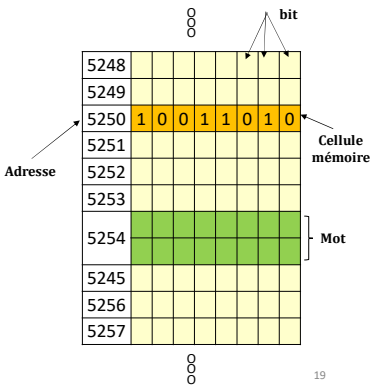
❑Une **variable** est **un espace mémoire** **nommé** de taille fixe, prenant au cours de déroulement du programme un nombre indéfini de **valeurs** différentes.

❑**L'élément unitaire** de stockage de l'information est appelé **bit**. Un bit ne peut avoir que deux états distincts : **0** ou **1** .

❑Dans la **mémoire** de l'ordinateur, les données sont manipulées par groupes de 8 bits (**octet**,**Byte**), ou plus (**mots** de 16, 32, 64 bits,...).

❑Une **case mémoire** (**cellule**) est donc appelée **mot** (word).

❑Pour que l'unité centrale puisse **stocker** une information et la **retrouver** dans la mémoire. Chaque mot a un **numéro** qui permet d'y faire référence de façon unique : c'est **l'adresse mémoire**.



Notion de variable

Variable et mémoire

❑ Il y a donc deux façons de voir la mémoire centrale de l'ordinateur : côté **programmeur** et côté **ordinateur.**

❑Dans la **programmation**, les **adresses** mémoire sont représentées par des **noms**. Le programmeur ne connait pas donc **l'adresse** d'une case mais plutôt son **nom**.

Mémoire

Adresses	Mots
10000000	6
10000001	8
10000010	7
10000011	
...	

x
y
moyenne
Variables

Côté ordinateur Côté programmeur

Déclaration des variables

❑La partie déclaration consiste à énumérer toutes les variables dont on aura besoin au cours de l'algorithme.

❑Chaque déclaration doit comporter le **nom** de variable (**identificateur**) et son **type**.

- ❑Un **identificateur** est le nom donné à une variable, une fonction, etc. Ce nom doit obligatoirement commencer par une lettre suivie d'une suite de lettres et les chiffres et il ne doit pas contenir d'espace.

❑Les variables se déclarent au début de l'algorithme, avant le programme lui-même mais après le mot clé «**variable**» ou «**var**».

Syntaxe :

variable identificateur : type
variables liste d'identificateurs : type

Déclaration des variables

Types de donnée

❑**Type de donnée** permet de déterminer :

- L'ensemble des **valeurs** que peut prendre une variable.
- L'ensemble des **opérations** qu'on peut les appliquer sur les variables.
- Déterminer **l'espace mémoire** nécessaire au stockage de cette variable dans la mémoire.

❑Les **types** les plus connus sont :

- Le type **entier** : sert à manipuler les nombres entiers positifs ou négatifs
- Le type **réel** : sert à manipuler les nombres à virgule.
- Le type **caractère** : sert à manipuler des caractères alphabétiques et numériques.
- Le type **chaine de caractères** : sert à manipuler des chaines de caractères permettant de représenter des mot ou des phrases.
- Le type **booléen** : utilise les expressions logiques (vrai/ faux)

Déclaration des variables

Types de donnée

Type de données	Numérique		Alphanumérique		Booléen
Algorithmique	Entier	Réel	Caractères	Chaîne de caractères	Booléen
Exemples	-45 19	-5,6 9,2	'A,' '@' '6 ' '? '	"bonjour" "G453929"	False True

Exemple:

Variables a , b : entier
x : réel
preom : chaine de caractères
absent : booléen

Déclaration des variables

Les opérations sur des variables

Type	Opérations	Symboles	Exemples
Entier	Addition	+	2 + 5 = 7
	Soustraction	-	8 – 5 = 3
	Multiplication	*	2 * 4 = 8
	Division	/	10 / 4 = 2,5
	Division entier	Div	10 Div 4 = 2
	Modulo (le reste de la division entier)	Mod	10 Mod 3 = 1
	Comparaison	≤ ≥ > < = ≠	7 > 2 vrai
Réel	Addition	+	2 + 5,4 = 7,4
	Soustraction	-	8,5 – 5 = 3,5
	Multiplication	*	2 * 4 = 8
	Division	/	10,6 / 4 = 2,65
	Comparaison	≤ ≥ > < = ≠	7 < 2 faux
Caractère	Comparaison	≤ ≥ > < = ≠	'B' < 'K ' faux
Chaine	Concaténation	& +	"ok " & " " & "by"
	Comparaison	≤ ≥ > < = ≠	'moh' < 'an ' vrai
Booléen	Logiques	et non ou	non(7>1) faux

Déclaration des variables

Les opérations sur des variables

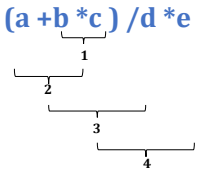
Priorité	Opérateur	Signification
La plus élevée	- (unaire)	Négation algébrique
	^	Puissance
	* / div mod	Multiplication, division, division entière et modulo
	+ -	Addition et soustraction
	&	Concaténation de chaînes
	< <= > >=	Opérateurs de comparaison
	= <>	Opérateurs d'égalité
	Non	Négation logique
	Et	Et logique
La plus basse	Ou	Ou logique

Déclaration des variables

Les opérations sur des variables

❑En cas de besoin, on utilise les parenthèses pour indiquer les opérations à effectuer en priorité.
❑à priorité égale, l'évaluation de l'expression se fait de gauche à droite.

Exemple : 7-5+2 **vaut : 4 -->** (7-5 puis 2+2)
7-(5+2) **vaut : 0 -->** (5+2 puis 7-7)
7-5*2 **vaut: 3 -->** (5*2 puis 7-10)



Les constantes

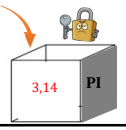
❑Une **constante** est une variable dont la valeur ne doit pas changer au cours de l'exécution du programme. Par convention, on la nomme en MAJUSCULES.

❑Une constante doit toujours **recevoir une valeur dès sa déclaration,** elle est caractérisée par un **identificateur** et une **valeur**

❑En pseudo-code, la déclaration des constantes est effectuée après le mot clé «**constante**» ou «**const**» .

Syntaxe :
Constante identificateur = valeur

Exemple:
Constante PI = 3,14;
Constante COEF = 5;



❑Un algorithme, par définition, est un ensemble **d'instructions** qui peuvent être simples ou complexes.

❑Une **instruction** est une action élémentaire commandant à l'ordinateur un calcul, ou une communication avec l'un de ses périphériques d'entrées ou de sorties.

❑Les **instructions de base** permettent la manipulation des variables telles que **l'affectation**, **la lecture** et **l'écriture**.

Instruction d'affectation

❑**L'affectation** est l'action élémentaire principale puisque c'est par son intermédiaire que l'on peut modifier la valeur d'une variable.

❑**L'affectation** consiste à **attribuer** une **valeur** à une **variable**, elle est symbolisée en algorithmique par le signe « ← »

Var ← val

- **val** peut être une **valeur**, une **autre variable** ou une **expression**.
- **Var** et **val** doivent être de **même type** ou de types compatibles.
- l'affectation **ne modifie que ce qui est à gauche** de la flèche

Remarque:

❑ les déclarations suivantes sont invalides :

Var = val ou **Var → val**

❑**Une constante** ne figure jamais à gauche d'une instruction d'affectation.

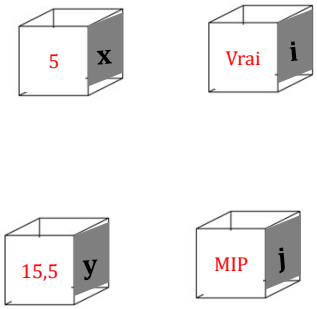
Exemple d'instruction fausse :

Constante z = 1 ;

- **z ← 2 ; « Faux »**

Instruction d'affectation

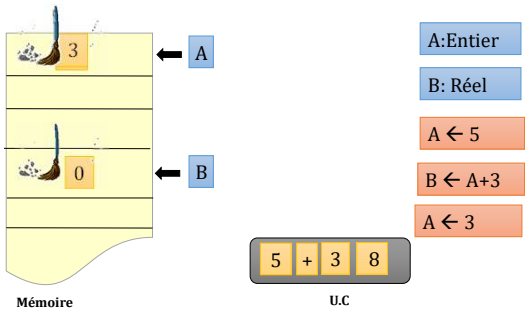
Exemples:



x ← 5
y ← x+10,5
i ← Vrai
j ← " MIP "

Instruction d'affectation

❑Après une affectation, l'ancien contenu d'une variable est substitué (**écrasé**) par le nouveau contenu.



Instruction d'affectation

❑Une instruction d'affectation doit se faire entre deux **types compatibles**.

```
Algorithme Affectations
Variables   i, j, k : entier;
            x, a, b, c, delta :réel;
            ok :booléen;
            ch1,ch2 :chaine de caractères;
Début
    i ←1;
    j ←i;
    k ←i+j;
    x ←10.3 ;
    ok ←FAUX ;
    ch1 ←"SMIA";
    ch2 ←ch1 ;
    x ←4;
    x ←j
    delta ← b*b - 4*a*c;
Fin
```

❑Exemples non valides:

```
i ←10.3 ;
OK ←"SMI";
 j ←x;
ch1 ←delta;
```

Instruction d'affectation

❑Les langages de programmation C, Python, C++, Java, … utilisent le signe égal = ou := pour l'affectation ←.

❑ Lors d'une affectation, **l'expression** de **droite** est **évaluée** et la **valeur** trouvée est **affectée** à la **variable** de **gauche**.
Ainsi, A←B ; est différente de B←A;

❑l'affectation est **différente** d'une équation **mathématique** :
- **A+1 ← 3 n'est pas possible en langages de programmation** et n'est pas équivalente à **A ← 2**.
- Les opérations x ← x+1 et x ← x-1 ont un sens en programmation et se nomment respectivement **incrémentation** et **décrémentation**.
- Certains langages donnent des **valeurs par défaut** aux variables déclarées.
- Pour éviter tout problème, il est préférable **d'initialiser** les variables déclarées.

33

Instruction d'affectation

Exercices

Faite le déroulement des algorithmes suivants en donnant la valeur finale de chaque variable :

```
Algorithme Test1
Variables a,b,c: booléens
Debut
    a ← vrai
    b ← faux
    c ← b
fin
```

```
Algorithme Test2
Variables a,b,c: entiers
Debut
    a ← 10
    b ← 30
    c ← a+b
    b ← a+2
    a ← a*b
fin
```

34

Les instructions d'entrée et de sortie

L'algorithme a besoin de données en entrée, et fournit un résultat en sortie.

Lorsqu'on utilise un ordinateur:
- le **clavier** permet de **saisir** les données
- l'**écran** permet d'**afficher** un résultat ou des textes qui donnent des directives sur les données à fournir.

35

L'instruction d'entrée

❑**L'instruction de lecture** permet d'introduire **(entrer)** une ou plusieurs données à partir du clavier (**la saisie**), puis les sauvegarder dans leurs cases mémoires correspondantes.

❑ L'instruction d'entrée donne la main à l'utilisateur pour **saisir** une donnée au clavier. La **valeur** saisie sera **affectée** à une **variable**.

Syntaxe :
Lire (identificateur)
Lire (identificateur1, identificateur2,…)

Exemple:
- **Lire(x)** : lit et stocke **une** valeur donnée dans la case mémoire associée à **x**.
- **Lire(x, y)** : lit et stocke **deux** valeurs, la première dans **x** et la deuxième dans **y**.

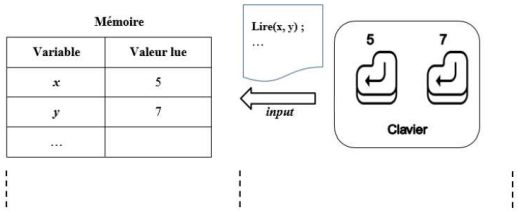
36

L'instruction d'entrée

Lire (var)

Se déroule en trois étapes :

1) Le **programme s'arrête** lorsqu'il rencontre une instruction **Lire** et **ne se poursuit** qu'après la **saisie** de l'entrée attendue par le clavier.
2) La touche **Entrée** signale la **fin** de la **saisie**.
3) La machine **place** la **valeur** entrée au clavier (ou saisie) dans la zone mémoire nommée **var**.



37

L'instruction d'entrée

Remarques:

- **Lire** une valeur ne correspondant pas au type de la variable où elle doit être stockée.
- Conseil: Avant de lire une variable, il est fortement conseillé d'**écrire** des **messages** à l'écran, afin de **prévenir l'utilisateur** de ce qu'il doit taper (**sinon longue attente**).
- **Attention une constante n'est jamais lue.**

38

L'instruction d'entrée

Exercices

lire(5)	Erreur
lire(R)	Lire et stocker la valeur saisie au clavier dans la variable R
lire('a')	erreur
lire(a,b)	Lire et stocker respectivement les deux valeurs saisies au clavier dans les variables a et b
Lire (a+b)	erreur
lire (x←5)	erreur
lire("le montant est ",M)	erreur

39

L'instruction de sortie

- **L'instruction d'écriture** permet **d'écrire** en sortie (**output**) les données **résultant** d'un traitement effectué par l'algorithme (valeur, texte, ...) en les **affichant** par exemple sur un périphérique de **sortie** tel que l'écran.
- Cette donnée à afficher peut être :
  - Un texte (un commentaire ou un message).
  - Une constante.
  - Le contenu d'une variable.
  - Mélange de texte et de valeurs.
  - Le résultat d'une expression arithmétique.
  - Le résultat d'une expression logique.

Syntaxe :

Ecrire (Expression)

40

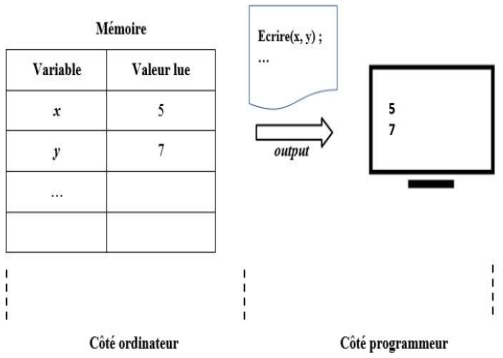
L'instruction de sortie

Exemple :

- **Ecrire** (" texte à afficher")
- **Ecrire** (var)
- **Ecrire** (var1,var2,…)
- **Ecrire** (exp)
- **Ecrire** (exp1,exp2,…)
- **Ecrire** (var,exp)
- **Ecrire** (" texte à afficher" ,var, exp)

▪La virgule sépare les chaines de caractères et la variable.
▪Tout le texte contenu entre des guillemets est écrit à l'écran, alors que lorsqu'une variable apparait dans l'instruction Ecrire c'est sa valeur qui est afficher.
▪Dans le cas d'écriture d'une expression, c'est le résultat d'évaluation de cette expression qui est affiché et non pas l'expression elle-même.

L'instruction de sortie



L'instruction de sortie

Exemple:

Ecrire(Nom)
▪Signifie : Affiche en sortie le contenu de la variable Nom

Ecrire("Entrer votre Nom : ")
▪Signifie : afficher sur l'écran le message : « Entrer votre Nom »

Ecrire("Votre Nom est : ", Nom)
▪Affiche en sortie le message : « Votre Nom est : le contenu de la variable Nom »

Ecrire(x+y)
▪Affiche en sortie le résultat d'addition de x et y
(soient x=5 et y=7, le résultat est 12)

L'instruction de sortie

Exercices

Algorithme	Affichage après Exécution
Ecrire(" Bonjour ")	Bonjour
Ecrire(R)	La valeur de R
Ecrire(3+2)	5
Ecrire("3+2=7")	3+2=7
Ecrire("3+2=3+2")	3+2=3+2
Ecrire("3+2=",3+2)	3+2=5
Ecrire(X+Y)	La valeur de X+Y
Ecrire("X+Y")	X+Y
Ecrire ("la somme est ",S)	La somme est : la valeur de S
Ecrire (a, b+3," message")	La valeur de a , la valeur de (a+b) , message

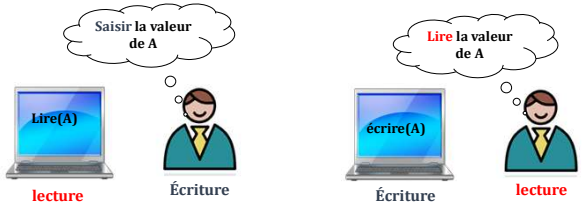
L'instruction d'entrée et de sortie

Remarque :

il faut bien comprendre la lecture et l'écriture du côté de la machine et non coté utilisateur ça veut dire :

La lecture pour la machine c'est **l'écriture** pour l'utilisateur.

L'écriture pour la machine c'est **la lecture** pour l'utilisateur.



Les commentaires

- Lorsqu'un algorithme devint long, il est conseillé d'ajouter des lignes de **commentaires** dans l'algorithme, c'est-à-dire des lignes qui ont pour but de donner des indications sur les instructions effectuées et d'expliquer le fonctionnement d'algorithme (programme) sans que le compilateur ne les prenne en compte.
- Un **commentaire** commence toujours par deux symboles « /* » et se termine par les symboles « */ » sur plusieurs lignes.
- Ou bien commence par le symbole « // » sur une seul ligne.

Exemple:

- */* Ceci est un commentaire sur plusieurs lignes */*
- *// Ceci est un commentaire sur une ligne*

Remarque :

Parfois on utilise les commentaires pour annuler l'action de quelques instructions dans un algorithme ou un programme au lieu de les effacer :

```
Variable i: Entier
// Variable j: Réel
```


Exemple :

Ecrire un algorithme qui **demande** un nombre à l'utilisateur, puis qui **calcule** et **affiche** le **carré** de ce nombre.

```
Algorithme Exe2
Variables nb, carr : Entier;
Début
    Ecrire ("Entrez un nombre :");
    Lire (nb);
    carr ← nb * nb;
    Ecrire ("Son carré est : ", carr);

/* En fait, on pourrait tout aussi bien économiser la variable
carr en remplaçant les deux avant-dernières lignes par */

    Ecrire ("Son carré est : ", nb * nb);
Fin
```


Les structures alternatives

- Contrairement au traitement séquentiel, **la structure alternative** ou **conditionnelle** permet **d'exécuter** ou non une série d'instructions selon la valeur d'une **condition**.

- Une **condition** est une expression logique ou une variable logique évaluée à **Vrai** ou **Faux**.
  - Si la condition est **vérifiée** (sa valeur est : **Vrai**).
  - Si la condition est **n'est pas vérifiée** (sa valeur est : **Faux**).
- Une **condition** est une **comparaison**, elle est composée de trois éléments :
  - une valeur
  - un opérateur de comparaison
  - une autre valeur

Les valeurs peuvent être a priori de n'importe quel type (numériques, caractères…). Mais si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du **même** type !

Structure alternative simple (un choix)

Dans cette forme, une **action** qui correspond à une ou plusieurs **instructions**, est **exécuté si** une **condition** est **vérifiée**. **Sinon** l'algorithme passe directement au bloc d'instruction qui suit immédiatement le bloc conditionnel.

Syntaxe :

```
Si Condition Alors
    Instructions
Fin Si
```

Remarque :
La condition évaluée après l'instruction « **Si** » est **une variable** ou **une expression booléenne** qui, à un moment donné, est **Vraie** ou **Fausse**. par exemple : x=y ; x <= y ; ...

Structure alternative simple (un choix)

Exemple:

```
x ← 5
y ← 9
Si (x = y) Alors
  Ecrire (″x est égale à y″)
FinSi
```

le message « x est égale à y » ne sera pas affiché puisque la condition (x = y) **n'est pas vérifiée**.

Structure alternative complète (deux choix)

Cette forme permet de choisir entre deux actions selon qu'une condition est vérifiée ou non.

Syntaxe :

```
Si Condition Alors
    Instructions 1
Si non
    Instructions 2
Fin Si
```

Si la condition est **vraie** alors le bloc d'instructions1 sera **exécuté**, et le bloc d'instructions2 sera **ignoré**, **sinon** le bloc instructions2 sera **exécuté** et le bloc d'instructions1 sera **ignoré**.

Structure alternative complète (deux choix)

Exemple :

```
x ← 5
y ← 9
Si (x = y) Alors
  Ecrire (″x est égale à y″)
Sinon
  Ecrire (″x est différente de y″)
FinSi
```

On peut traiter les deux cas possibles. Si la condition (x=y) est **vérifiée**, le **premier message est affiché**, si elle n'est **pas vérifiée**, le **deuxième message est affiché**.

Structure alternative imbriquée (multiple choix)

- La forme complète permet de choisir entre plusieurs actions en imbriquant des formes simples selon la syntaxe suivante:

Syntaxe :

```
Si condition1 Alors
    Instructions 1
Sinon
    Si condition2 Alors
        Instructions 2
    Sinon
        Instructions 3
    Fin si
Fin si
```


Structure alternative imbriquée (multiple choix)

Exemple :

```
Algorithme Nature_Nombre
variables
        n: réel
Début
    Ecrire("Algorithme qui détermine la nature d'un nombre:")
    Ecrire("Veuillez entrer un nombre :")
     Lire n
    Si (n > 0) alors
       Ecrire("Ce nombre est positif")
    Sinon
      Si (n = 0) alors
        Ecrire("Ce nombre est nul")
       Sinon
        Ecrire("Ce nombre est négatif")
       FinSi
     FinSi
Fin
```


Structure alternative imbriquée (multiple choix)

L'utilisation de tests imbriqués permet de :

- **Simplifier le (pseudo-code) :** à travers l'imbrication nous n'avons utilisé que deux conditions simples au lieu de trois conditions dont une est composée.

  ➔Un algorithme (ou programme) plus simple et plus lisible.

- **Optimiser le temps d'exécution :** dans le cas où la première condition est vérifiée, l'algorithme passe directement à la fin, sans tester le reste qui est forcément faux.

  ➔Un algorithme (ou programme) plus performant à l'exécution.

Structure alternative imbriquée (multiple choix)

➔Les tests peuvent avoir un degré quelconque d'imbrications

```
Si (condition1) Alors instruction(s) 1
Sinon
      Si (condition2) Alors instruction(s) 2
       Sinon
         Si (condition3) Alors instruction(s) 3
         ...
          Sinon instruction(s) N
          FinSi
      FinSi
FinSi
```

Conseil :
utiliser les tests imbriqués pour **limiter** le **nombre** de **tests** et **placer d'abord** les **conditions** les plus **probables**.

Conditions composées

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme d'une simple comparaison, par exemple:

- La condition x ∈ [0, 1[s'exprime par la combinaison de deux conditions **x >= 0 ET x < 1** qui doivent être vérifiées en même temps.
- le cas « A est inclus entre 5 et 8 », A une variable de type entier, elle revient à dire que « A est supérieur à 5 ET A est inférieur à 8 ».

Il y a donc bien là deux conditions, reliées par ce qu'on appelle un **opérateur logique**.

Conditions composées

- Une condition **composée** est une condition formée de plusieurs conditions **simples reliées** par des **opérateurs logiques**: **ET**, **OU**, **OU exclusif** (XOR) et **NON**.
- Ordre de priorité des opérateurs logiques
  1. NON
  2. ET
  3. OU
  4. OUexclusif

Exemples :

- x compris entre 2 et 6 : **(x >= 2) ET (x < =6)**
- n divisible par 3 ou par 2 : **(n%3=0) OU (n%2=0)**
- deux valeurs et deux seulement sont identiques parmi a, b et c : **(a=b) XOR (a=c) XOR (b=c)**
- (A = faux) ⇔ non A
- (A = vrai) ⇔ A

Conditions composées

Tables de vérité

L'évaluation d'une condition composée se fait selon des règles présentées généralement dans ce qu'on appelle tables de vérité.

C1	C2	C1 ET C2
VRAI	VRAI	**VRAI**
VRAI	FAUX	**FAUX**
FAUX	VRAI	**FAUX**
FAUX	FAUX	**FAUX**

C1	C2	C1 OU C2
VRAI	VRAI	**VRAI**
VRAI	FAUX	**VRAI**
FAUX	VRAI	**VRAI**
FAUX	FAUX	**FAUX**

C1	C2	C1 XOR C2
VRAI	VRAI	**FAUX**
VRAI	FAUX	**VRAI**
FAUX	VRAI	**VRAI**
FAUX	FAUX	**FAUX**

C1	NON C1
VRAI	**FAUX**
FAUX	**VRAI**

Conditions composées

Nous avons les équivalences suivantes :

- **NON** (A **ET** B) ⇔ **NON** A **OU NON** B
- **NON** (A **OU** B) ⇔ **NON** A **ET NON** B

Ainsi, toute structure de test avec l'opérateur logique **ET** peut être exprimée d'une manière équivalente avec l'opérateur logique **OU** et vice-versa. Par conséquent, les deux alternatives suivantes sont équivalentes :

```
Si A ET B Alors
    Instructions 1
Sinon
    Instructions 2
Fin si
```

⇔

```
Si Non A OU Non B Alors
    Instructions 1
Sinon
    Instructions 2
Fin si
```

Structure sélective (Conditionnelle à choix multiples)

Exemple:

```
Algorithme jours
variables j: entier
Début
   Ecrire("Entrer un nombre entre 1 et 7:")
   lire j
   Si (j = 7) alors
      écrire("Dimanche")
   Sinon
      Si (j = 6) alors
         écrire("Samedi")
      Sinon
         Si (j = 5) alors
            écrire("Vendredi")
         Sinon
            Si (j = 4) alors
               écrire("Jeudi")
            Sinon
               Si (j = 3) alors
                  écrire("Mercredi")
               Sinon
                  Si (j = 2) alors
                     écrire("Mardi")
                  Sinon
                     Si (j = 1) alors
                        écrire("Lundi")
                     FinSi
                  FinSi
               FinSi
            FinSi
         FinSi
      FinSi
   FinSi
Fin
```


Structure sélective (Conditionnelle à choix multiples)

- Plusieurs conditions à traiter dans un algorithme.
- Comparer une même variable avec plusieurs valeurs.
- Imbrication des alternatives devient importante.

⇓

La **structure sélective** est la solution la plus appropriée car elle est plus facile à représenter.

- **La structure sélective** est une représentation simplifiée des conditions imbriquées (au lieu d'exprimer beaucoup d'imbrications de conditions Si - Sinon).

- La **structure sélective**
  - ou **structure conditionnelle à choix multiples**
  - ou encore **structure Selon**
  - appelée parfois **structure Cas**

Structure sélective (Conditionnelle à choix multiples)

Syntaxe :

```
Selon sélecteur Faire
   valeur 1 : Traitement 1
   valeur 2 : Traitement 2
   valeur 3 : Traitement 3
   valeur 4 : Traitement 4
   ...
   valeur n : Traitement n
   Sinon : autre Traitement
FinSelon
```

ou

```
Cas sélecteur Vaut
   valeur 1 : Traitement 1
   valeur 2 : Traitement 2
   valeur 3 : Traitement 3
   valeur 4 : Traitement 4
   ...
   valeur n : Traitement n
   Sinon : autre Traitement
FinCas
```


Structure sélective (Conditionnelle à choix multiples)

Exemple:

```
Algorithme jours
variables  j: entier
Début
   Ecrire("Algorithme qui affiche le jour :")
   Ecrire("Veuillez entrer un nombre entre 1 et 7:")
   Lire j
   selon j faire
       1:Ecrire("lundi")
       2:Ecrire("Mardi")
       3:Ecrire("Mercredi")
       4:Ecrire("Jeudi")
       5:Ecrire("Vendredi")
       6:Ecrire("Samedi")
       7:Ecrire("Dimanche")
       Sinon: écrire("Entrer un nombre entre 1 et 7")
   FinSelon
Fin
```

Exercice:

1- Ecrire un algorithme qui permet de lire deux variables numériques a et b, de les afficher avant et après leur permutation.
Par exemple, avant : a=15 et b=6, après : a=6 et b=15.

2- Proposer un algorithme qui réalise la permutation de deux variables numériques sans avoir utiliser une troisième variable.

3- Ecrire un algorithme qui permet la lecture de trois entiers a, b et c et qui détermine le minimum des trois nombres.

4- Ecrire un algorithme qui permet la résolution de l'équation ax+b=0.

5- Ecrire un algorithme qui permet de résoudre l'équation du deuxième degré $ax^2+bx+c=0$.

Boucle

- Dans un algorithme on a souvent besoin de répéter un même bloc d'instructions plusieurs fois.
- Au lieu d'effectuer cette répétition manuellement on utilise les structures itératives ou **répétitives** (**boucles**).
- Une **boucle** (ou **itération**) est une instruction de contrôle qui permet de répéter plusieurs fois un ensemble d'instructions.
- Les boucles participent à ce qu'on appelle la **factorisation** du code. Elles permettent de n'écrire qu'une fois un morceau d'algorithme qui peut néanmoins être exécuté plusieurs fois.

Boucle

- En algorithmique il existe trois types principaux de structures répétitives à savoir:
  - la boucle **Pour** qui permet de répéter une instruction un certain nombre de fois.
  - la structure **Tant que ... Faire** qui permet d'effectuer une instruction tant qu'une **condition** est satisfaite.
  - la boucle **Répéter ... jusqu'a** qui permet de répéter une instruction jusqu'à ce qu'une **condition** soit satisfaite.

Nombre de répétitions est connu → Boucle **Pour**

Nombre de répétitions dépend d'une condition → Boucle **Tant que** / Boucle **Répéter ... Jusqu'à ...**

Structure POUR

- La boucle **Pour** permet d'exécuter une séquence d'instructions **un nombre de fois connu fixé** à l'avance.
- Elle utilise une variable **indice** (**compteur**) de contrôle d'itérations caractérisé par: **valeur initiale**, **valeur finale** , **pas** de variation

Syntaxe :

```
Pour indice variant de initial à final pas valP
    Instructions
FinPour
```

- **indice** : est une variable compteur de **type entier** (ou caractère). Elle doit être **déclarée**.
- **valP** : est un **entier** qui peut être **positif** ou **négatif**.
  - **pas** peut ne pas être mentionné, car par défaut sa valeur est égal à **1**. Dans ce cas
- **le nombre d'itérations** est égal à **(final - initial+1)**.
- **initial** et **final** peuvent être des **valeurs**, des **variables** définies avant le début de la boucle ou des **expressions** de même type que **indice**.

Structure POUR

Exemple

```
Algorithme Message
Début
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
Fin
```

```
Algorithme Message
Variable i:entier
Début
    Pour i de 0 à 9 pas 1
        écrire("Bonjour")
    FinPour
Fin
```


Structure POUR

Remarques

- Il faut éviter de modifier la valeur du **compteur (indice)** (et de **final**) à l'intérieur de la boucle.
- En effet, une telle action :
  - perturbe le nombre d'itérations prévu par la boucle Pour
  - rend difficile la lecture de l'algorithme
  - présente le risque d'aboutir à une boucle infinie

Exemple :

```
Pour i de 1 à 5 pas 1
i ←i-1;
écrire(" i = ", i);
FinPour
```


Structure Tantque … Faire

- La boucle **Tant que … Faire**, permet de **tester une condition** et **répéter** le traitement associé tant que cette condition est **vérifiée**.
- Une fois cette condition là devient **fausse** alors on **quitte** la boucle pour **poursuivre** l'exécution du reste du traitement.

Syntaxe :

```
Tantque condition Faire
    Instructions
Fintantque
```


Structure Tant que … Faire

Remarques:

- Il est possible que les instructions à **répéter** ne soient jamais exécutées.
- Le nombre **d'itérations** dans une boucle TantQue **n'est pas connu** au moment d'entrée dans la boucle. Il **dépend** de l'évolution de la valeur de la condition.
- Une des instructions du corps de la boucle doit absolument **changer la valeur de la condition** de **vrai** à **faux** (après un certain nombre d'itérations), sinon le programme va tourner indéfiniment (boucles infinies).

Structure Tant que ... Faire

Exemple (Message)

```
Algorithme Message
Début
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
Fin
```

??????????????

Boucles infinies

Condition

Point de Sortie de la boucle

```
Algorithme Message
Début
    Tantque Vrai Faire
        écrire("Bonjour")
    FinTantque
Fin
```


Structure Tant que ... Faire

Exemple (Message)

```
Algorithme Message
Début
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
    écrire("Bonjour")
Fin
```

Déclarer un Compteur

Initialiser le Compteur

Vérifier la condition

Incrémenter du Compteur

Répéter 10 fois

Sortir de la boucle

```
Algorithme Message
Variable i:entier
Début
i ← 0
Tantque i < 10 Faire
    écrire("Bonjour")
    i ← i+1
FinTantque
Fin
```


Structure Tant que ... Faire

Exemple

```
Algorithme Message
Variable i:entier
Début
i ← 0
Tantque i < 10 Faire
    écrire("Bonjour")
    i ← i+1
FinTantque
Fin
```

i ← 0		Ecrire("Bonjour")	i ← i+1
Itération 1:	i 0	Bonjour	i 1
Itération 2:	i 1	Bonjour	i 2
Itération 3:	i 2	Bonjour	i 3
Itération 4:	i 3	Bonjour	i 4
...		...	
Itération 10:	i 9	Bonjour	i 10
Fin tanque	i 10	Point de sortie la boucle	

Structure Tant que ... Faire

Exemple

```
Algorithme Message
Variable i:entier
Début
i ← 0
Tantque i < 10 Faire
    écrire("Bonjour")
    i ← i+1
FinTantque
Fin
```

```
Algorithme Message
Variable i:entier
Début
i ← 10
Tantque i>0 Faire
    écrire("Bonjour")
    i ← i-1
FinTantque
Fin
```

```
Algorithme Message
Variable i:entier
Début
i ← 1
Tantque i =< 10 Faire
    écrire("Bonjour")
    i ← i+1
FinTantque
Fin
```

```
Algorithme Message
Variable i:entier
Début
i ← 9
Tantque i>=0 Faire
    écrire("Bonjour")
    i ← i-1
FinTantque
Fin
```

Structure Tant que ... Faire

Boucles Infinie

```
Algorithme Message
Variable i:entier
Début
i ← 0
Tantque i < 10 Faire
   écrire("Bonjour")
FinTantque
Fin
```

```
Algorithme Message
Variable i:entier
Début
i ← 1
Tantque i > 0 Faire
   écrire("Bonjour")
   i ← i+1
FinTantque
Fin
```


Structure Tant que ... Faire

Exemple (contrôle de saisie)

Contrôle de **saisie** d'une **lettre** alphabétique jusqu'à ce que le **caractère entré** soit **valable**

```
Algorithme Contrôle_saisie
variables C : Caractère
Début
 écrire("Entrer une lettre :")
 lire C
Tantque ((C < 'A') ou (C > 'Z')) Faire
   écrire ("Saisie erronée. Recommencez")
   Lire C
FinTantque
   écrire ("La saisie est valable ")
Fin
```


Lien entre Pour et Tant que ... Faire

- La boucle **Pour** est **un cas particulier** de **Tantque** (cas où le nombre d'itérations est connu et fixé).
- Tout ce qu'on peut écrire avec **Pour** peut être remplacé avec **Tantque** (**la réciproque est fausse**)

```
Pour i de initial à final pas 1
   Instructions
FinPour
```

```
i ← initial
Tantque i <= final Faire
   Instructions
   i ← i+1
FinTantque
```


Lien entre Pour et Tant que ... Faire

Exemple (Puissance)

un algorithme qui permet de calculer et d'affiche x^n (x est un réel , n est un entier positive).

```
Algorithme puissance
variables x,p  : réel
           n,i  : entier
Début
   écrire(" Donner la base x:" )
   lire x
   écrire(" Donner l'exposant n:" )
   lire n
   p  ← 1
Pour i de 1 à n pas 1
   p  ← p*x
FinPour
   écrire(x," à la puissance ",n," est:",p )
Fin
```

```
   p  ← 1
   i  ← 1
Tantque (i<=n) Faire
   p  ← p*x
   i  ← i+1
FinTantque
```

Structure Répéter … Jusqu'à …

❑La boucle **Répéter… Jusqu'à…** , permet de **répéter** un bloc d'instructions jusqu'à ce qu' une **condition** soit **vérifiée**.

Syntaxe :

```
Répéter
    Instructions
Jusqu'à condition
```

- La **vérification** de la **condition** s'effectue après l'exécution des instructions.
- les **instructions** entre **Répéter** et **jusqu'a** sont **exécutées au moins une fois** et leur exécution est répétée jusqu'à ce que la **condition** devienne **fausse**.

Structure Répéter … Jusqu'à …

Exemple

Ecrire un algorithme qui permet de demander à l'utilisateur un nombre N et qui calcule la somme des N entiers positifs .
Par exemple, si l'utilisateur entre le nombre 5, l'algorithme doit calculer 1+2+3+4+5 .

```
Algorithme Message
Variable N,s,i:entier
Début
Écrire ("Donner N:")
Lire N
i ← 1
s← 0
Répéter
   s← s+i
   i ← i+1
Jusqu'à (i > N)
Écrire ("la somme est :" ,s)
Fin
```

Déclarer un Compteur
Initialiser le Compteur
Exécuter le traitement pour la 1 fois
{ Exécuter le traitement à répéter
Incrémenter du Compteur
Vérifier la condition
Répéter N fois }
Sortir de la boucle

Structure Répéter … Jusqu'à …

Exemple (Division de deux nombres)

```
Algorithme Division
variables a,b,c  : réel
Début
   écrire(" Entrer le nombre a :" )
   lire a
   Répéter
   écrire(" Donner le nombre b:" )
   lire b
   Jusqu'à (b <> 0)
   c  ← a/b
   écrire(" à la résultat de la division est:",c)
Fin
```


Tantque … Faire et Répéter… jusqu'à…

Différences entre les boucles Tant que… Faire et Répéter … Jusqu'à…

- La séquence d'instructions est exécutée au moins **une fois** dans la boucle **Répéter … Jusqu'à…**, alors qu'elle peut ne pas être exécutée dans le cas du **Tantque … Faire**.
- Dans les deux cas, la séquence d'instructions est exécutée si la **condition** est **vraie**.
- Dans les deux cas, la séquence d'instructions doit nécessairement **faire évoluer** la **condition**, faute de quoi on obtient une boucle infinie.

Tant que … Faire et Répéter… jusqu'à…

Exemple

Ecrire un algorithme qui demande à l'utilisateur de saisir un entier supérieur strictement à 1. et qui calcule la somme des entiers jusqu'à a ce nombre.
Par exemple, si l'utilisateur entre le nombre 5, l'algorithme doit calculer 1+2+3+4+5 .

```
Algorithme Message
Variable N,s,i:entier
Début
Écrire ("Donner N:")
Lire N
i ← 1
s← 0
Répéter
   s← s+i
   i ← i+1
Jusqu'à (i > N)
Écrire ("la somme est :" ,s)
Fin
```

```
i  ← 1
s  ← 0
Tantque (i=<N) Faire
   s ← s+i
   i ← i+1
FinTantque
```


Notion du compteur

- Un **compteur** est une variable associée à la boucle dont la valeur est incrémentée de un à chaque itération. Elle sert donc à **compter le nombre d'itérations** (répétitions) de la boucle.
- La notion du **compteur** est associée particulièrement aux deux boucles : « Répéter…jusqu'à » et « Tant que…faire ». Par contre, dans la boucle « Pour », c'est l'indice qui joue le rôle du compteur.
- L'utilisation du **compteur** dans les deux premières boucles est exprimée ainsi :

Bloc de la boucle

```
compt ← 0
Répéter
    Instructions
     ...
    compt ← compt+1
Jusqu'à condition
```

```
compt ← 0
Tantque condition Faire
    Instructions
     ...
    compt ← compt+1
FinTantque
```


Notion du compteur

Remarque :

- Il faut toujours initialiser le **compteur** avant de commencer le comptage.
- La variable « **compt** » a été initialisée à zéro (0) avant le début de chaque boucle.
- L'instruction « **compt ← compt +1** » incrémente la valeur de « **compt** » de un (1). Elle peut être placée n'importe où à l'intérieur du bloc de la boucle.

Exemple :

```
compt ← 0
Répéter
    Ecrire(i)
    compt ← compt+1
Jusqu'à (i=5)
```

Résultat d'exécution : 0,1,2,3,4

```
compt ← 0
Tantque (i<5) Faire
    Ecrire(i)
    compt ← compt+1
FinTantque
```

Résultat d'exécution : 0,1,2,3,4

Boucles imbriquées

- Les boucles peuvent être imbriquées les unes dans les autres. Deux ou plusieurs boucles imbriquées peuvent être aussi les mêmes ou différentes.

Exemples :

```
Pour i de 1 à 2  pas 1
    Ecrire("i=",i)
   Pour j de 1 à 3 pas 1
      Ecrire("j=",j)
   FinPour
FinPour
```

boucle 1 / boucle 2

Résultat d'exécution :

```
i=1
j=1
j=2
j=3
i=2
j=1
j=2
j=3
```

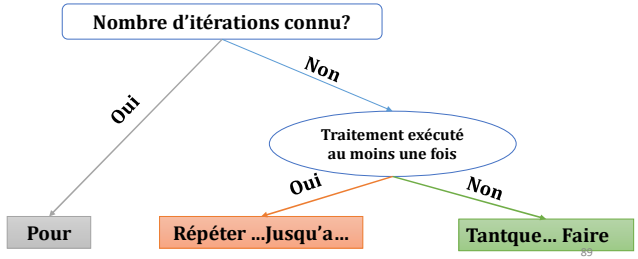
```
Algorithme boucle_imbriquée
Variables i,j:entier
Début
Pour i de 1 à 5 pas 1
   Pour j de 1 à i
        écrire("O")
   FinPour
   écrire("K")
FinPour
Fin
```

Résultat d'exécution :

```
OK
OOK
OOOK
OOOOK
OOOOOK
```

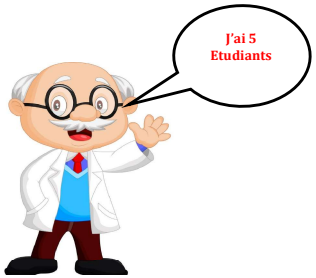
Choix d'un type de boucle

- Si on peut déterminer le **nombre d'itérations** avant l'exécution de la boucle, il est plus naturel d'utiliser la boucle **Pour**.
- S'il **n'est pas** possible de **connaître** le nombre d'itérations avant l'exécution de la boucle, on fera appel à l'une des boucles **Tant que… Faire** ou **Répéter … Jusqu'à….**



Activité

❑ Supposons qu'on veut calculer la moyenne des notes d'une classe de 5 étudiants :



Stocker la note de chaque étudiants
Calculer la moyenne

Etudiant	Note
Amina	10,5
Ahmed	9
Meryem	14
Mouad	15
Rachid	2

Activité

Jusqu'ici, nous avons employé des **variables** pour **stocker** une seule valeur de types primitifs.

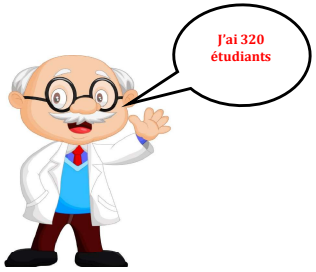
Etudiant	Variable	Note
Amina	**N1**	10,5
Ahmed	**N2**	9
Meryem	**N3**	14
Mouad	**N4**	15
Rachid	**N5**	2

Activité

Imaginons maintenant le cas pour une promotion de 320 étudiants, Vous pouvez remarquer que c'est un peu **lourd** de déclarer, manipuler une centaine de variables (avec 320 fois de lecture/écriture distinctes) et calculer la moyenne donnera obligatoirement une atrocité du genre :

Moyenne ← (N1+N2+N3+…+N319+N320)/320

Stocker la note des 320 étudiants???



Etudiant	Variable	Note
Amina	**N1**	10,5
Ahmed	**N2**	9
Meryem	**N3**	14
…	**…**	…
Mouad	**N319**	15
Rachid	**N320**	2

320

Activité

❑ Jusqu'à présent, le seul moyen pour le faire:

- **Déclarer** 320 variables désignant les notes N1, …, N320.
- La **saisie** de ces notes nécessite 320 instructions **lire(Ni)**.
- Le **calcul** de la moyenne est : **l'addition** de 320 notes diviser par 320
- Le calcul du nombre des **notes>10** se fait par une suite de tests de 320 instructions Si :
  - nbre ← 0
  - Si (N1 >10) alors nbre ←nbre+1 FinSi
  - …
  - Si (N320>10) alors nbre ←nbre+1 FinSi

Cette façon n'est pas très pratique

C'est pourquoi la notion de **tableau** a été alors inventée.

Tableaux

❑ En algorithmique (et en programmation), on peut **regrouper** toutes ces **variables** en une **seule structure de donnée** qui s'appelle **tableau**.

❑ Un **tableau** est un ensemble de **variables** de **même type** ayant toutes le même **identificateur**.

Comment peut-on différencier entre des variables ayant le même nom ?

Chaque élément du tableau est repéré par un indice. Ce dernier est un numéro qui permet de différencier chaque élément du tableau des autres.

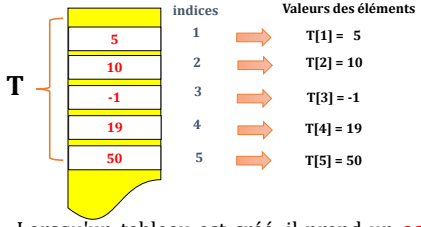
❑ Les éléments du **tableau** ont tous le même **nom**, mais pas le même **indice**. Pour accéder à un élément d'un tableau, on utilise le nom du tableau suivi de l'indice de l'élément entre crochets […].

Tableaux

❑ **L'indice** est une variable entière permet d'indiquer la **position** d'un **élément** donné au sein du tableau et de déterminer sa **valeur**.

Exemple

Soit le tableau **T** contenant les valeurs suivantes : 5, 10, -1, 19 et 50 :
L'organisation du tableau **T** dans la mémoire peut être représentée comme suit :



Remarque : Lorsqu'un tableau est créé, il prend un **espace contigu** en mémoire : les cases sont les unes à la suite des autres, Il n'y a pas de « trou » au milieu.

Types tableaux

Les éléments d'un tableau sont rangés selon un ou plusieurs **axes** appelés **dimensions** du tableau.

❑ **Tableau** à une dimension.

❑ **Tableau** à plusieurs dimensions.

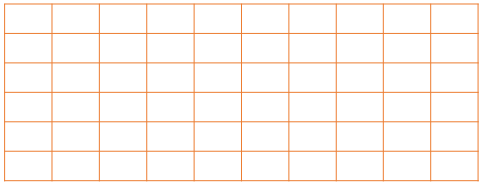


Tableau à une dimension

- Dans les **tableaux à une dimension** (qui permettent de représenter par exemple des vecteurs au sens mathématique du terme), chaque élément est accessible pour lecture ou modification par un seul **indice**.
- Un **tableau à une dimension** est un **vecteur** d'éléments de même **type**.
- Il peut être représenté sous la forme suivante :

T	T[1]	T[2]	T[3]	T[4]	...	T[n]

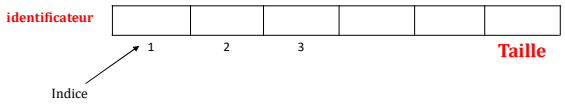
  - Dimension du tableau : **1**
  - Taille du tableau : **n**
  - Les **T[i]**, i=**1, 2, ..., n** doivent être de même type

Déclaration

- La **déclaration** d'un tableau à une dimension s'effectue en précisant le **type** de ses éléments et sa **taille** (le nombre de ses éléments)

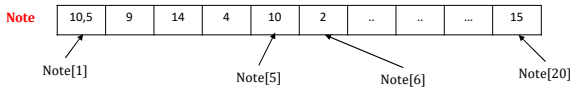
Syntaxe :

Tableau identificateur[taille] : type



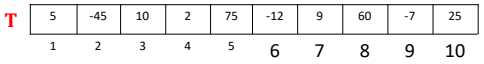
- **Exemple :**

Tableau note[20] : réel



Accéder aux éléments d'un tableau

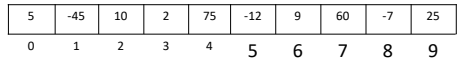
Tableau T[10] : entier



- Ce tableau est de longueur 10, car il contient 10 emplacements.
- Chacun des dix nombres du tableau est repéré par son rang, appelé **indice**.
- Pour accéder à un élément du tableau, il suffit de préciser entre crochets **l'indice de la case contenant cet élément**.
- Pour accéder au 5ème élément (75), on écrit : **T[5]**

Accéder aux éléments d'un tableau

Tableau T[10] : entier



- Selon les langages de programmation, le premier indice du tableau est soit 0, soit 1. Le plus souvent c'est **0**.
- Pour accéder au **i**ème élément, on écrit **T[i-1]** (avec 0<=i<10)
- Dans ce cas, **T[i]** désigne l'élément **i+1** du tableau **T**.

Une fois déclaré, un tableau peut être manipulé comme un ensemble de **variables** simples. Les trois manipulations de base sont **l'affectation**, la **lecture** et **l'écriture**.

Affectation

L'affectation d'une valeur **v** à un élément **i** d'un tableau **T** se fait par :

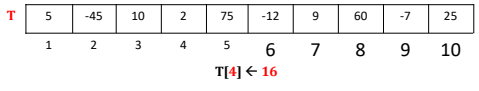
T[i] ← v

Syntaxe d'affectation:

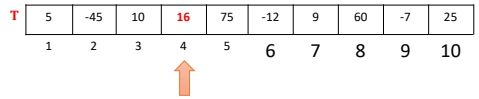
identificateur[indice] ← valeur

Exemple:

- Affectation du nombre 16 au 4éme élément du tableau:



- Cette instruction a modifié le contenu du 4ème élément du tableau (16 au lieu de 2).



- La variable **x** prend la valeur du premier élément du tableau (x vaut 5).

x ←T[1]

Lecture

Il est possible aussi d'affecter des valeurs aux éléments d'un tableau par une instruction de **lecture**.

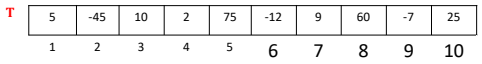
Syntaxe de lecture:

Lire(identificateur [indice])

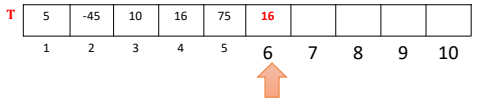
Exemple :

- Utilisation de la lecture pour saisir un nombre au 6éme élément du tableau

Lire(T[6])



- Cette instruction initialise le contenu du 6ème élément du tableau par la valeur 16



Ecriture

De même que la lecture, l'écriture de la valeur d'un élément du tableau s'écrira comme suit :

Ecrire T[i]

Cette instruction permet d'afficher la valeur de l'élément **i** du tableau **T**.

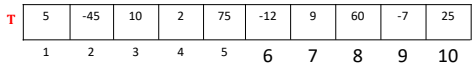
Syntaxe d'écriture:

Ecrire(identificateur[indice])

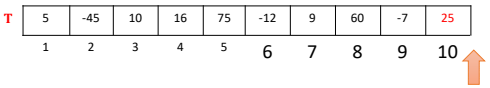
Exemple :

- affichage de la valeur du nombre du dernier élément du tableau

Ecrire(T[10])



- Cette instruction affiche le contenu du 10ème élément du tableau (la valeur 25).



Tableaux et boucles

- Un grand **avantage** des tableaux est qu'on peut traiter les données qui y sont stockées de façon simple en utilisant des **boucles**.
- Les **boucles** sont extrêmement utiles pour les algorithmes associés aux tableaux, pour **parcourir** les éléments du tableau selon l'ordre croissant (ou décroissant) des indices on utilise les boucles.
- Le traitement de chacun des éléments étant souvent le même, seule la valeur de l'indice est amenée à changer, une **boucle** est donc parfaitement adaptée à ce genre de traitements.

initialisation d'un tableau

- Pour initialiser un tableau on peut utiliser par exemple les instructions suivantes :
  - T1[4] = { 2, 25, -7, 3}
  - T2[9] = { 10, 17, -5, 43, 55, -26, 70, 88, 19 }

Saisie et affichage des éléments d'un tableau

```
Algorithme Tableau
Variables  n, i : entier
           Tableau T[100]: réel

Début
   Ecrire(" Donner la taille du tableau : " )
      Lire(n)
   Pour i de 1 à n pas 1
      Ecrire("Saisie de l'élément  ", i, " : " )
      Lire(T[i])
   FinPour
   Pour i de 1 à n pas 1
      Ecrire ("T[",i,"]=", T[i] )
   FinPour
fin
```

Déclaration d'un tableau avec une taille maximale de 100

Lire la taille effective du tableau

Lire les éléments du tableau un par un

Afficher les éléments du tableau un par un

Remarque

Ne pas confondre :

- **Taille maximale** : une constante (capacité)

  Constante Max=100

  Variables tableau T[Max]: réel

- **Taille effective** : nombre de cases réellement utilisées lors de la manipulation du tableau (une variable)

Exemple:

```
Algorithme Tableau
Variables  n, i : entier
           Tableau T[100]: réel
Début
  Ecrire(" Donner la taille du tableau:" )
  Lire(n)
   pour i ← 1 à n pas 1 faire
   ...
   FinPour
Fin
```

Taille maximale : 100
Taille effective : n

Exemple 1

```
Algorithme note
Variables  n, i : entier
      moyenne, somme :réel
           tableau Note[50]: réel
Début
   Ecrire(" Entrer le nombre des notes:" )
      Lire(n)
   Pour i de 1 à n pas 1
      Ecrire("Donner la note de l'étudiant num ", i, ":" )
      Lire(Note[i])
   FinPour
   somme ← 0
   Pour i de 1 à n pas 1
      somme ← somme+ Note[i]
   FinPour
   Moyenne ← somme/n
  Ecrire("la moyenne des notes est :", moyenne )
Fin
```

Exemple 2

Un algorithme qui permet de calculer le nombre d'étudiants (20 étudiants) ayant une note supérieure à 12 avec les tableaux.

```
Algorithme note
Variables tableau Note[20]: réel
        nbr, i : entier
Début
      Pour i ← 1 à 20 pas 1
          Ecrire("Donner la note de l'étudiant num ", i, ":" )
          Lire(Note[i])
      FinPour
      Nbr ← 0
      Pour i ← 1 à 20 pas 1
          Si (Note[i] >12) alors
            nbr ← nbr+1
          FinSi
      FinPour
     Ecrire("le nombre des notes supérieur à 12 est :", nbr )
Fin
```


Activité

Stocker
Moyenne
Max et Min

Etudiant	Analyse	Algèbre	Informatique
Amina	10,5	15	19
Ahmed	9	5	10
Meryem	14	10	4
Mouad	15	8	12
Rachid	2	4	13

Activité

Tableau Analyse[5]:réel

Tableau Algèbre[5]:réel

Tableau Informatique[5]:réel

Etudiant	Analyse	Algèbre	Informatique	Moyenne
Amina	10,5	15	16	13,8
Ahmed	9	5	10	8
Meryem	14	10,5	4	9,5
Mouad	15	8	12	11,6
Rachid	2	4	13,5	6,5
Max	15	15	16	
Min	2	4	4	

L'utilité d'un **tableau à deux dimensions** réside dans la possibilité de déclarer un **seul** tableau au lieu de déclarer **plusieurs** tableaux identiques.

Tableaux à deux dimension

Un tableau à plusieurs dimensions est une **matrice** d'éléments de même type.

- Les langages de programmation permettent de déclarer des tableaux dans lesquels les valeurs sont repérées par deux **indices**.
  - Le premier indice représente le numéro de **ligne**
  - Le deuxième indice représente le numéro de **colonne**

A

A[1,1]	A[1,2]	A[1,3]	A[1,4]	A[1,5]	A[1,6]
A[2,1]	A[2,2]	A[2,3]	A[2,4]	A[2,5]	A[2,6]
A[3,1]	A[3,2]	A[3,3]	A[3,4]	A[3,5]	A[3,6]
A[4,1]	A[4,2]	A[4,3]	A[4,4]	A[4,5]	A[4,6]
A[5,1]	A[5,2]	A[5,3]	A[5,4]	A[5,5]	A[5,6]
A[6,1]	A[6,2]	A[6,3]	A[6,4]	A[6,5]	A[6,6]

- Dimension du tableau : **2**
- Taille du tableau : **L,C**
- Les **A[i][j]**, i=**1, 2, ...,L** , **j**=1, 2, ...,C doivent être de même type

Déclaration

- En pseudo code :
  Tableau identificateur[NbrLigne] [NbrColonne] : type

- Exemple :
  Tableau Notes[5][3] : réel

		1	2	3
	1	10,5	15	16
	2	9	5	10
Notes	3	14	10,5	4
	4	15	8	12
	5	2	4	13,5

Accéder aux éléments de la matrice

- Les éléments sont rangées dans un tableau à deux entrées.

A		1	2	3	4	5	6
	1	15	28	44	9	90	5
	2	23	20	51	12	3	19
	3	36	21	60	65	18	10

- Ce tableau a 3 lignes et 6 colonnes.
- Les éléments du tableau sont repérés par leur numéro de **ligne** désignés en bleu et leur numéro de colonne désignés en vert .
  - Par exemple **A[2][4]** vaut **12**.

A[i][j] permet d'accéder à l'élément de la matrice qui se trouve à l'intersection de la ligne i et de la colonne j

Accéder aux éléments de la matrice

- Selon les langages de programmation, les premiers indices de la matrice est 0, soit 1. Le plus souvent c'est **0**.

A		0	1	2	3	4	5
	0	15	28	44	9	90	5
	1	23	20	51	12	3	19
	2	36	21	60	65	18	10

- Ce tableau a 3 lignes et 6 colonnes.
- Les éléments du tableau sont repérés par leur numéro de **ligne** désignés en bleu et leur numéro de colonne désignés en vert .
  - Par exemple **A[1][3]** vaut **12**.

A[i][j] permet d'accéder à l'élément de la matrice qui se trouve à l'intersection de la ligne i+1 et de la colonne j+1

- Un tableau à deux dimensions est manipulé de la même façon qu'un tableau simple (à une seule dimension) que ce soit pour **l'affectation**, la **lecture** ou **l'écriture**.

Affectation

- Syntaxe d'affectation:
  A[NbrLigne][NbrColone] ← valeur

- Exemple:
  - Affectation de la note 16 à l'étudiant num 4 dans la matière num 2.

A[4][2] ← 16

A		1	2	3
	1	10,5	15	16
	2	9	5	10
	3	14	10,5	4
	4	15	16	12
	5	2	4	13,5

Lecture

- Syntaxe de lecture:

Lire (A[NbrLigne][NbrColone]

- Exemple :
  - saisie de la note 10 à l'étudiant num 3 dans la matière num 1.

Lire (A[3][1])

A	1	2	3
1	10,5	15	16
2	9	5	10
3	**10**	10,5	4
4	15	16	12
5	2	4	13,5

Ecriture

- Syntaxe d'écriture:

Ecrire (A[NbrLigne][NbrColone]

- Exemple :
  - Afficher de l'étudiant num 2 dans la matière num 2.

Lire (A[2][2])

A	1	2	3
1	10,5	15	16
2	9	**5**	10
3	10	10,5	4
4	15	16	12
5	2	4	13,5

Saisie et affichage des éléments de la matrice

```
Algorithme Matrice
Variables m, n, i, j : entier
          Tableau M[100][100]: entier
Début
   Ecrire(" Entrer le nombre de lignes de la matrice:" )
   Lire(m)
   Ecrire(" Entrer le nombre de colonnes de la matrice:" )
   Lire(n)
   Pour i ← 1 à m pas 1
      Pour j ← 1 à n pas 1
         Ecrire ("Entrez l'élément : ligne ", i, " et colonne ", j)
         Lire (M[i][j])
      FinPour
   FinPour
   Pour i ← 0 à m pas 1
      Pour j ← 0 à n pas 1
         Ecrire ("M[",i,",",j,"]=" , M[i][j])
      FinPour
   FinPour
Fin
```

Déclaration d'une matrice avec une taille maximale de (100, 100)

Lire la taille effective de la matrice m et n

Lire les éléments de la matrice

Afficher les éléments de la matrice

initialisation d'une matrice

- Pour initialiser une matrice on peut utiliser par exemple les instructions suivantes :
  - T1[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}}
  - T2[3][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Exemple

Ecrire un algorithme qui permet de demander à l'utilisateur de saisir la notes des étudiants (5 étudiants) dans chaque matière (3 matières).
Cet algorithme permet de calculer et afficher la moyenne de chaque étudiant.

Etudiant	Analyse	Algèbre	Informatique	Moyenne
Amina	10,5	15	16	13,8
Ahmed	9	5	10	8
Meryem	14	10,5	4	9,5
Mouad	15	8	12	11,6
Rachid	2	4	13,5	6,5

121


```
Algorithme Moyenne
Variables    m,n,i,j : entier
        moyenne, somme :réel
        Tableau M[50][50]: réel
Début
      Ecrire(" Entrer le nombre de lignes de la matrice:" )
      Lire(m)
      Ecrire(" Entrer le nombre de colonnes de la matrice:" )
      Lire(n)
      pour i ←1 à m pas 1
            pour j ← 1 à n pas 1
              Ecrire ("Donner la note de l' étudiant num ; ", i, " dans la matière num : ", j)
              lire (M[i][j])
            FinPour
      Fin pour
      somme ← 0
      Moyenne ← 0
      pour i ← 1 à m pas 1
          pour j ← 1 à n pas 1
             somme← somme+ M[i][j]
          FinPour
        Moyenne ← somme/n
        Ecrire("la moyenne de l'étudiant num :",i," est :" , moyenne )
        somme←0
      FinPour
Fin
```

122