

ALGORITHME & PROGRAMMATION STRUCTURÉE et ORIENTÉE OBJETS

UTOPIOS / Ib Formation



Objectif de ce module

■ Indispensable avant la programmation

- ➔ Apprendre les concepts de base de l'algorithme et de la programmation
- ➔ Etre capable de mettre en œuvre ces concepts pour analyser des problèmes simples et écrire les programmes correspondants

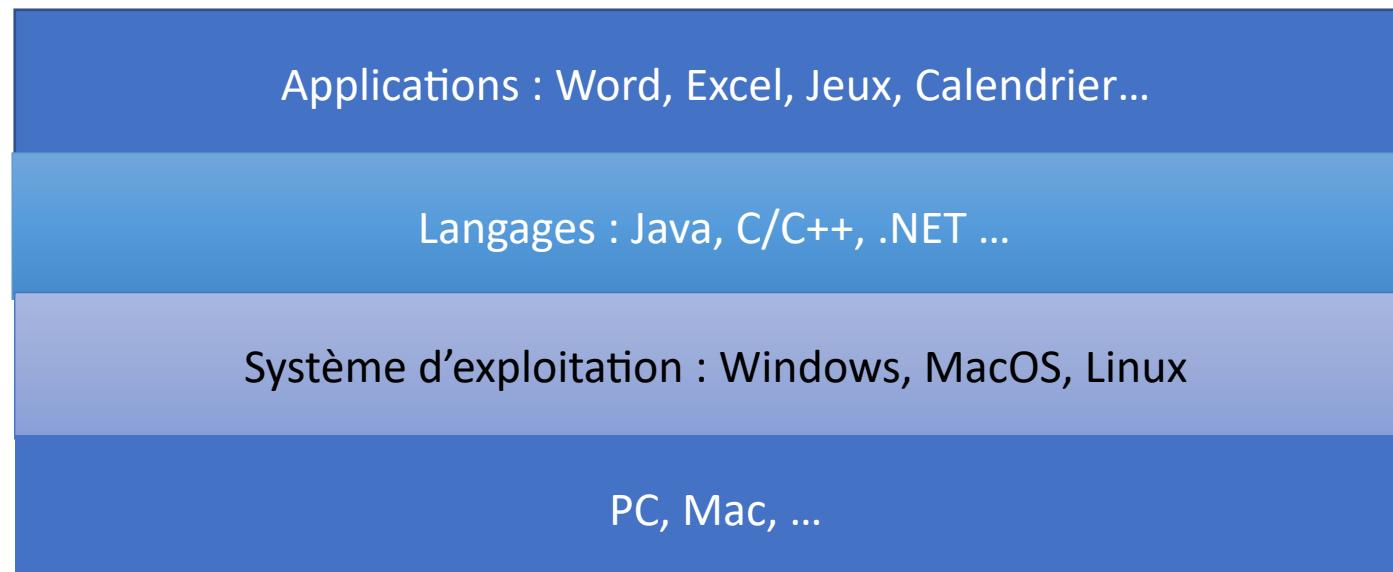
A complex network graph is visible in the background, composed of numerous small, semi-transparent blue dots connected by thin lines, creating a mesh-like pattern.

Pourquoi fait-on
des algorithmes ?

Généralités en informatique

L'informatique : c'est quoi ?

- ➔ Techniques du traitement **automatique** de l'**information** au moyen des ordinateurs
- ➔ Eléments d'un système informatique



Généralités en informatique

■ Les éléments d'un ordinateur

→ Unité centrale (le boîtier)

- Processeur ou CPU (*Central Processing Unit*)
- Mémoire centrale
- Disque dur, lecteur ...
- Cartes spécialisées (cartes graphiques, réseau, ...)
- Interfaces d'entrée-sortie (USB, HDMI, VGA)

→ Périphériques

- Moniteur, clavier, souris
- Modem, imprimante, scanner, ...

Généralités en informatique

■ Qu'est-ce qu'un système d'exploitation ?

→ Ensemble de programmes qui gère le matériel et contrôle les applications :

- **Gestion des périphériques**

Affichage à l'écran, lecture du clavier, pilotage d'une imprimante, ...

- **Gestion des utilisateurs et de leurs données**

Comptes, partage des ressources, gestion des fichiers et répertoires, ...

- **Interface avec l'utilisateur**

Textuelle ou graphique : Interprétation des commandes

- **Contrôle des programmes**

Découpage en tâches, partage du temps processeur, ...

Généralités en informatique

■ Qu'est-ce qu'un langage informatique ?

- ➡ Un langage informatique est un outil permettant de donner des ordres (**instructions**) à la machine à chaque instruction correspond une action du processeur.
- ➡ **Intérêt** : écrire des programmes (suite consécutive d'instructions) destinés à effectuer une tache donnée
- ➡ **Exemple** : un programme de gestion de comptes bancaires
- ➡ **Contrainte** : être compréhensible par la machine

Généralités en informatique

■ Qu'est-ce que le langage machine ?

- ➔ Langage **binnaire**: l'information est exprimée et manipulée sous forme d'une suite de bits
- ➔ Un **bit (binary digit)** = 0 ou 1 (2 états électriques)
- ➔ Une combinaison de 8 bits = 1 **Octet (byte)** => $2^8= 256$ possibilités qui permettent de coder tous les caractères alphabétiques, numériques, et symboles tels que ?, *, &, ...
 - Le code **ASCII** (*American Standard Code for Information Interchange*) donne les correspondances entre les caractères alphanumériques et leurs représentation binaire, Ex. A= 01000001 etc...
- ➔ Les opérations logiques et arithmétiques de base (addition, multiplication, ...) sont effectuées en binaire

Généralités en informatique

L'assembleur

- Problème : le langage machine est difficile à comprendre par l'humain
 - Idée : trouver un langage compréhensible par l'homme qui sera ensuite converti en langage machine
- Assembleur** (1er langage) : exprimer les instructions élémentaires de façon symbolique



- + : déjà plus accessible que le langage machine
- - : dépend du type de la machine (n'est pas **portable**)
- - : pas assez efficace pour développer des applications complexes



Apparition des langages évolutés

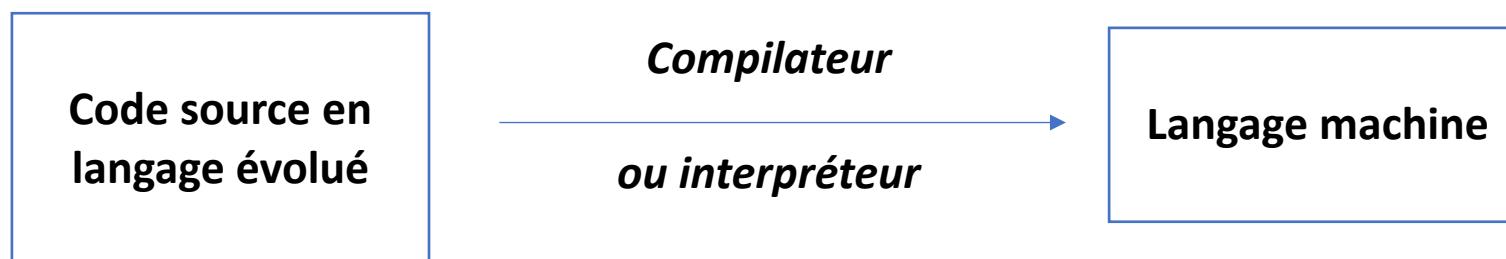
Généralités en informatique

■ Langage haut niveau

➡ Intérêts multiples pour le haut niveau :

- proche du langage humain «anglais» (compréhensible)
- permet une plus grande portabilité (indépendant du matériel)
- Manipulation de données et d'expressions complexes (réels, objets, a^*b/c , ...)

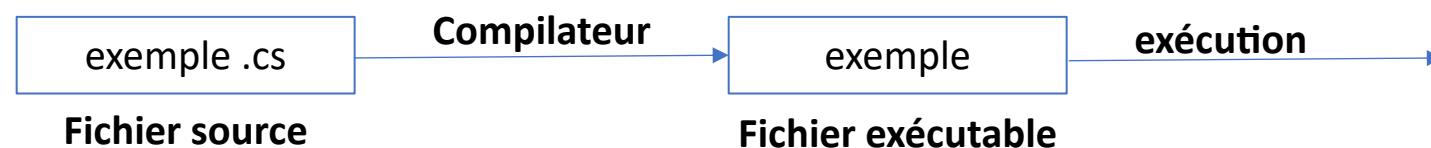
➡ Nécessité d'un **traducteur** (compilateur/interpréteur), exécution plus ou moins lente selon le traducteur



Généralités en informatique

Compilateur

➡ **Compilateur** : traduire le programme entier une fois pour toutes



- + plus rapide à l'exécution
- + sécurité du code source
- - il faut recompiler à chaque modification

Généralités en informatique

■ Interpréteur

➡ **Interpréteur** : traduire au fur et à mesure les instructions du programme à chaque exécution



- + exécution instantanée appréciable pour les débutants
- - exécution lente par rapport à la compilation

Généralités en informatique

■ Un programme

- ➔ Un programme correspond à une méthode de résolution pour un problème donné.
- ➔ Cette méthode de résolution est effectuée par une suite d'instructions d'un langage de programmation
- ➔ Ces instructions permettent de traiter et de transformer les **données** (entrées) du problème à résoudre pour aboutir à des résultats (sorties)
- ➔ Un programme n'est pas une solution en soi mais une **méthode à suivre** pour trouver des solutions.
- ➔ La programmation est l'ensemble des activités orientées vers la conception, la réalisation, le test et la maintenance de programmes

Généralités en informatique

■ Langages de programmation

➡ Types de langages :

- Langages procéduraux
- Langages orientés objets

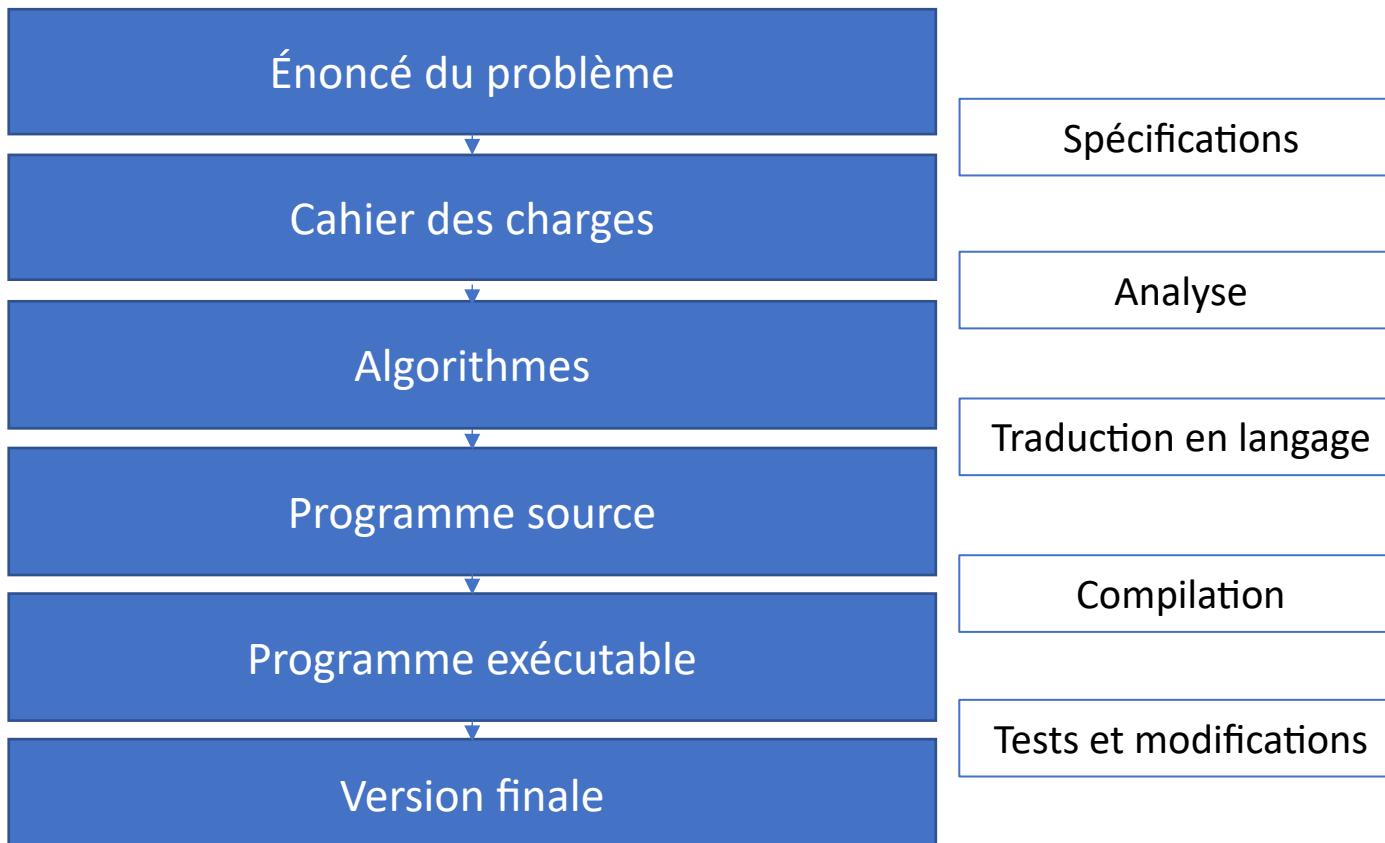
➡ Exemple :

- C, PHP
- C++, Java, C# ...

➡ Choix d'un langage ?

Généralités en informatique

Etapes de la réalisation d'un programme



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

Généralités en informatique

L'algorithmique

- Le terme **algorithme** vient du nom du mathématicien arabe **Al-Khawarizmi** (820 après J.C.)
- Un algorithme est une description complète et détaillée des actions à effectuer et de leur séquencement pour arriver à un résultat donné
 - Intérêt : séparation analyse/codage (pas de préoccupation de syntaxe)
 - Qualités : **exact** (fournit le résultat souhaité), **efficace** (temps d'exécution, mémoire occupée), **clair** (compréhensible), **général** (traite le plus grand nombre de cas possibles), ...
- Une bonne connaissance de l'algorithmique permet d'écrire des algorithmes **exacts et efficaces**

Généralités en informatique

■ Algorithme et programmation

→ Un algorithme est une suite d'instructions ayant pour but de résoudre un problème donné
Un algorithme peut se comparer à une recette de cuisine

→ L'élaboration d'un algorithme précède l'étape de programmation

- Un programme est un algorithme
- Un langage de programmation est un langage compris par l'ordinateur
- L'algorithme est la résolution brute d'un problème informatique

Généralités en informatique

■ Représentation d'un algorithme

- ➡ **L'organigramme** : représentation graphique avec des symboles (carrés, losanges, etc.)
 - offre une vue d'ensemble de l'algorithme
 - représentation quasiment abandonnée aujourd'hui

- ➡ **Le pseudo-code** : représentation textuelle avec une série de conventions ressemblant à un langage de programmation (sans les problèmes de syntaxe)
 - plus pratique pour écrire un algorithme
 - représentation largement utilisée

Notions de bases en algorithmique

Notions de base

■ Notions de variable

- ➡ Dans les langages de programmation une **variable** sert à stocker la valeur d'une donnée
- ➡ Une variable désigne en fait un emplacement mémoire dont le contenu peut changer au cours d'un programme (d'où le nom variable)
- ➡ Règle : Les variables doivent être **déclarées** avant d'être utilisées, elle doivent être caractérisées par :
 - un nom (**Identificateur**)
 - un **type** (entier, réel, caractère, chaîne de caractères, ...)

Notions de base

Choix des identificateurs

→ Le choix des noms de variables est soumis à quelques règles qui varient selon le langage, mais en général:

- Un nom doit commencer par une lettre alphabétique **exemple valide: A1** **exemple invalide: 1A**
- Il doit être constitué uniquement de lettres, de chiffres et du soulignement _
valides: cdi2016, cdi_2016 **invalides: cdi 2016,cdi-2016,cdi;2016**
- Il doit être différent des mots réservés du langage
(par exemple en Java: **int, float, else, switch, case, default, for, main, return**, ...)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé

Notions de base

■ Choix des identificateurs

- ➡ **Conseil :** pour la lisibilité du code, choisir des noms significatifs qui décrivent les données manipulées
exemples: `TotalVentes2016`, `Prix_TTC`, `Prix_HT`

- ➡ **Remarque :** en pseudo-code algorithmique, on va respecter les règles citées,
même si on est libre dans la syntaxe

Notions de base

Type de variable

→ Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre, les types offerts par la plupart des langages sont:

→ Type numérique (entier ou réel)

- **Byte** (codé sur 1 octet): de 0 à 255
- **Entier court** (codé sur 2 octets) : -32 768 à 32 767
- **Entier, long** (codé sur 4 ou 8 octets)
- **Réel simple précision** (codé sur 4 octets)
- **Réel double précision** (codé sur 8 octets)

→ Type logique ou booléen : deux valeurs VRAI ou FAUX

→ Type caractère : lettres majuscules, minuscules, chiffres, symboles, ...**exemples: 'A', 'a', '1', '?', ...**

→ Type chaîne de caractère : toutes suites de caractères **exemples: " Nom, Prénom", "code postal: 1000", ...**

Notions de base

Déclaration des variables

→ Toutes variables utilisées dans un programme doit avoir fait l'objet d'une déclaration préalable

→ En pseudo-code, on va adopter la forme suivante pour la déclaration de variables

Variables liste d'identificateurs : type

→ Exemple:

Variables i, j,k: entier

x, y : réel

Vrai / Faux: booléen

ch1, ch2 : chaîne de caractères

→ Remarque: pour le type numérique on va se limiter aux entiers et réels sans considérer les sous types

Notions de base

L'instruction d'affectation

→ L'**affectation** consiste à attribuer une valeur à une variable
c'est le fait de remplir où de modifier le contenu d'une zone mémoire

→ En pseudo-code, l'affectation se note avec le signe \leftarrow

Var \leftarrow e: attribue la valeur de e à la variable Var

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

→ Ex valides : i \leftarrow 1 j \leftarrow i k \leftarrow i+j
 x \leftarrow 10.3 OK \leftarrow FAUX ch1 \leftarrow "SMI"
 ch2 \leftarrow ch1 x \leftarrow 4 x \leftarrow j

→ non valides : i \leftarrow 10.3 OK \leftarrow "SMI » j \leftarrow x

Notions de base

Attention

→ Beaucoup de langages de programmation (C/C++, Java, ...) utilisent le signe égal = pour l'affectation \leftarrow .

Ne pas confondre :

- l'affectation n'est pas commutative : $A=B$ est différent de $B=A$
- l'affectation est différente d'une équation mathématique :
 - $A=A+1$ a un sens en langage de programmation
 - $A+1=2$ n'est pas possible en langage de programmation et n'est pas équivalente à $A=1$

→ Certains langages donnent des valeurs par défaut aux variables déclarées.

Pour éviter tout problème il est préférable **d'initialiser les variables** déclarées

Exercice 1 (simple sur l'affectation)

→ Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

Variables A, B, C : Entier

Début algorithme

A \leftarrow 3

B \leftarrow 7

A \leftarrow B (devient 7)

B \leftarrow A + 5 (devient $7+5=12$)

C \leftarrow A + B (devient $7+12=19$)

C \leftarrow B - A (devient $12-7=5$)

Fin algorithme

Exercice 2 (simple sur l'affectation)

→ Donnez les valeurs des variables A et B après exécution des instructions suivantes ?

Variables A, B : Entier

Début

A ← 1

B ← 2

A ← B (devient 2)

B ← A

Fin

Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ?

Exercice 3 (simple sur l'affectation)

→ Ecrire un algorithme permettant d'échanger les valeurs de deux variables A et B

Exercice simple sur l'affectation 3/3

→ Ecrire un algorithme permettant d'échanger les valeurs de deux variables A et B

Variables A, B, TMP : Entier

Début

A ← 1

B ← 2

TPM ← B (2)

B ← A (1)

A ← TMP (2)

Fin

Notions de base

■ Expressions et opérateurs

- ➡ Une **expression** peut être une valeur, une variable ou une opération constituée de variables reliées par des **opérateurs** exemples: **1, b, a*2, a+ 3*b-c, ...**
- ➡ L'évaluation de l'expression fournit une valeur unique qui est le résultat de l'opération
- ➡ Les **opérateurs** dépendent du type de l'opération, ils peuvent être:
 - **des opérateurs arithmétiques:** +, -, *, /, % (**modulo : reste de la division euclidienne**), ^ (**puissance**)
 - **des opérateurs logiques :** **NON, OU, ET**
 - **des opérateurs relationnels:** =, ≠, <, >, <=, >=
 - **des opérateurs sur les chaînes:** & (**concaténation**)
- ➡ Une expression est évaluée de gauche à droite mais en tenant compte de **priorités**

Notions de base

Priorités des opérateurs

→ Pour les opérateurs arithmétiques donnés ci-dessus, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire) :

1. ^ élévation à la puissance
2. * , / multiplication, division
3. % modulo
4. + , - addition, soustraction

exemple: $2 + 3 * 7$ vaut 23

→ En cas de besoin (ou de doute), on utilise les parenthèses pour indiquer les opérations à effectuer en priorité

exemple: $(2 + 3) * 7$ vaut 35

Notions de base

■ Instructions d'entrée/Sortie : lecture et écriture

- ➔ Les instructions de lecture et d'écriture permettent à la machine de communiquer avec l'utilisateur
- ➔ La **lecture** permet d'entrer des données à partir du clavier

En pseudo-code, on note: **lire (var)**

la machine met la valeur entrée au clavier dans la zone mémoire nommée var

- ➔ Remarque: Le programme s'arrête lorsqu'il rencontre une instruction Lire et ne se poursuit qu'après la frappe d'une valeur au clavier et de la touche Entrée

Notions de base

■ Instructions d'entrée/Sortie : lecture et écriture

→ **L'écriture** permet d'afficher des résultats à l'écran (ou de les écrire dans un fichier)

En pseudo-code, on note: **écrire (var)**

la machine affiche le contenu de la zone mémoire var

→ 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 frapper

Instructions d'entrée/Sortie : lecture et écriture

→ Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui calcule et affiche le double de ce nombre

→ Algorithme Calcul_double

variables A, B : entier

Début

```
    écrire ("entrer le nombre ")
    lire (A)
    B ← 2*A
    écrire ("le double de ", A, "est :", B)
```

Fin

Instructions d'entrée/Sortie : lecture et écriture

→ Ecrire un algorithme qui vous demande de saisir votre nom puis votre prénom et qui affiche ensuite votre nom complet

→ **Algorithme AffichageNomComplet**

variables Nom, Prenom, Nom_Complet: **chaîne de caractères**

Début

```
écrire("entrez votre nom")
lire(Nom)
écrire("entrez votre prénom")
lire(Prenom)
Nom_Complet ← Nom & Prenom
écrire("Votre nom complet est : ", Nom_Complet)
```

Fin

Conditions composées

■ Les opérateurs logiques combinés

- ➡ Une condition composée est une condition formée de plusieurs conditions simples reliées par des opérateurs logiques: ET, OU, OU exclusif (XOR) et NON

- ➡ 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 $(b = c)$

Conditions composées

■ Les tables de vérités

- L'évaluation d'une condition composée se fait selon des règles présentées généralement dans ce qu'on appelle des 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

Structures conditionnelles

Instruction conditionnelle

C'est quoi ?

→ Les **instructions conditionnelles** servent à exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée.

→ On utilisera la forme suivante :

- la condition ne peut être que vraie ou fausse
- si la condition est vraie, ce sont les instructions1 qui seront exécutées
- si la condition est fausse, ce sont les instructions2 qui seront exécutées
- la condition peut être une condition simple ou une condition composée de plusieurs conditions

```
Si condition alors  
instruction ou suite d'instructions1  
Sinon  
instruction ou suite d'instructions2  
Finsi
```

Instruction conditionnelle

C'est quoi ?

→ La partie **Sinon** n'est pas obligatoire, quand elle n'existe pas et que la condition est fausse, aucun traitement n'est réalisé

```
Si condition alors  
instruction ou suite d'instructions1  
Finsi
```

Instruction conditionnelle

EXEMPLE

■ Exemple Si... Alors... Sinon

Algorithme AffichageValeurAbsolue (version1)

Variable x : réel

Début

Ecrire ("Entrez un réel: ")

Lire (x)

Si (x < 0) alors

Ecrire ("la valeur absolue de ", x, "est:", -x)

Sinon

Ecrire ("la valeur absolue de ", x, "est:", x)

Finsi

Fin

Instruction conditionnelle

EXEMPLE

■ Exemple Si... Alors

Algorithme AffichageValeurAbsolue (version2)

Variable x,y: réel

Début

Ecrire("Entrez un réel: ")

Lire (x)

y \leftarrow x

Si (x < 0) **alors**

 y \leftarrow -x

Finsi

Ecrire ("la valeur absolue de ", x, "est:",y)

Fin

Instruction conditionnelle

EXERCICE

Exercice 4 Si... Alors

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est divisible par 3

Instruction conditionnelle

EXERCICE
CORRIGÉ

■ Exemple Si... Alors

Instructions conditionnelles imbriquées

C'est quoi ?

→ Les tests imbriqués / instructions conditionnelles peuvent avoir un degré quelconque d'imbriques

```
Si condition1 alors
    Si condition2 alors
        instructionsA
    Sinon
        instructionsB
    Finsi
Sinon
    Si condition3 alors
        InstructionsC
    Finsi
Finsi
```

Tests imbriqués

EXEMPLE

■ Exemple Si... Alors... Sinon imbriqué

Variable n : entier

Début

Ecrire ("entrez un nombre: ")

Lire (n)

Si (n < 0) alors

Ecrire("Ce nombre est négatif")

Sinon

Si (n = 0) alors

Ecrire ("Ce nombre est nul")

Sinon

Ecrire ("Ce nombre est positif")

Finsinon

Finsi

Fin

Tests imbriqués

EXEMPLE

Exemple Si... Alors... Sinon imbriqué

Variable n : entier

Début

```
Ecrire ("entrez un nombre: ")
Lire (n)
Si (n < 0) alors
    Ecrire("Ce nombre est négatif")
Finsi
Si (n = 0) alors
    Ecrire ("Ce nombre est nul")
Finsi
Si (n > 0) alors
    Ecrire ("Ce nombre est positif")
Finsi
Fin
```

Remarque : dans la version 2 on fait trois tests systématiquement alors que dans la version 1, si le nombre est négatif on ne fait qu'un seul test

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

Exercice 5

Le prix de photocopies dans une reprographie varie selon le nombre demandé :

- 0,5 euros la copie pour un nombre de copies inférieur à 10,
- 0,4 euros pour un nombre compris entre 10 et 20,
- et 0,3 euros au-delà.

Ecrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées, qui calcule et affiche le prix à payer

Test imbriqué

EXERCICE
CORRIGÉ

Exercice Si... Alors... Sinon

Exercice 6

1. Déterminer le montant d'un capital c placé à un taux fixe t pendant un nombre n d'années. On suppose que c, t, n sont lus,

Exemple de calcul (le taux est de 4% , soit 0,04)

$$C_n = 10\,000 \times (1+0,04)^5 = 12\,166 \text{ euros, soit un gain de 2\,166 euros.}$$

Test imbriqué

EXERCICE

Exercice 7

Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur.
Ensuite, il l'informe de sa catégorie pour une licence sportive :

- « Baby » de 3 à 6 ans
- « Poussin » de 7 à 8 ans
- « Pupille » de 9 à 10 ans
- "Minime" de 11 à 12 ans
- "Cadet" à partir de 13 ans