FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Algorithmique et Programmation en Python

Licence : Physique Chimie
Filières : Chimie
Semestre : 3

Pr: Youssef Ouassit

Objectifs du module

Objectifs du module :

- Comprendre les **bases de l'algorithmique** : savoir décrire les étapes d'un raisonnement ou d'un calcul de façon logique.
- **Apprendre à résoudre des problèmes** scientifiques à l'aide d'un langage de programmation.
- Découvrir et **utiliser le langage Python**, largement employé en physique, chimie et data science.

Descriptif horaire & évaluation

- 22 heures de cours ;
- 12 heures de travaux dirigés ;
- 12 heures de travaux pratiques ;
- Évaluation : Examen final

FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 1:
Introduction à l'algorithmique

Licence : Physique Chimie
Filières : Chimie

Pr: Youssef Ouassit

Plan Séance 1

1. **Introduction à l'Algorithmique**
   a. Qu'est-ce qu'un algorithme?
   b. L'importance de l'algorithmique dans la résolution de problèmes.

2. **Conception des algorithmes**
   a. Définition du problème.
   b. Analyse d’un problème
   c. Structure générale d’un algorithme

Pourquoi on donne une telle d'importance à l'outil informatique et à la programmation de cet outil?

Une des grandes caractéristiques des systèmes informatiques est :

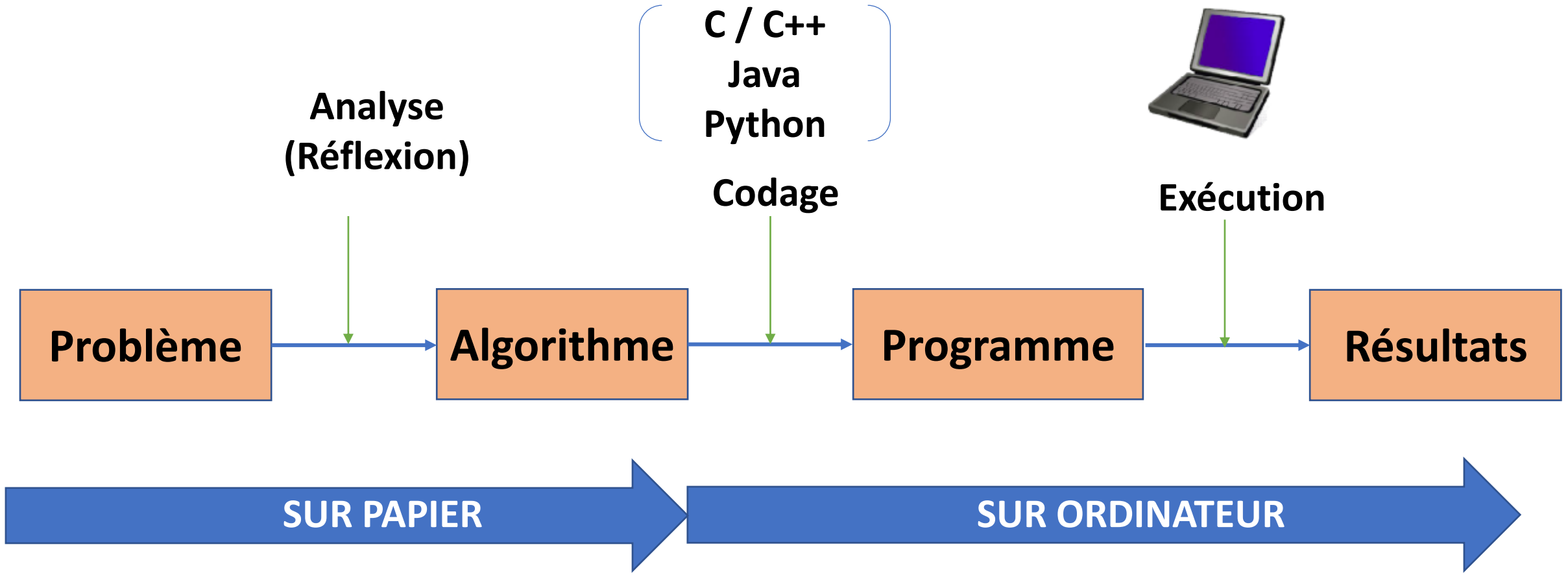
La rapidité d'exécution des tâches

Problème : Calculer 1000 !

- Combien de temps est nécessaire pour calculer ce nombre ?

Introduction à l'Algorithmique

Qu'est ce qu'un algorithme ?



Exemples des algorithmes :

Dans la vie courante, un algorithme peut prendre la forme :

- Une recette de cuisine;
- Méthode pour préparer du café
- Méthode de résolutions des équations de second degré
- etc…

Qu'est ce qu'un algorithme ?

Définition : Un algorithme est une séquence d'instructions (d'actions) **finie** et **ordonnée** permettant de transformer les données en entrées vers des données en sortie afin de résoudre un problème donné. Les données en sortie représentent la solution du problème.

Algorithme = Σ *Données* + Σ *Instructions*

Origine du mot algorithme ?

Algorithmes الخوارزميات

Le mot **algorithme** tire son origine du nom d'un mathématicien perse du IXe siècle, **Muhammad ibn Musa al-Khwarizmi**. Né vers 780 dans la région de Khwarazm (aujourd'hui en Ouzbékistan), al-Khwarizmi est célèbre pour ses travaux en mathématiques, en astronomie et en géographie.

Introduction à l'Algorithmique

Histoire des algorithmes

La "minute culturelle"

Algorithmes **sans ordinateurs** :

Introduction à l'Algorithmique

Histoire des algorithmes

La "minute culturelle"

Algorithmes **sans ordinateurs** :

- Euclide (vers -300) : calcul du PGCD de 2 nombres

Introduction à l'Algorithmique

Histoire des algorithmes

La "minute culturelle"

Algorithmes **sans ordinateurs** :

- Euclide (vers -300) : calcul du PGCD de 2 nombres
- Al-Khuwārizmī (825) : résolution d'équations

Introduction à l'Algorithmique

Histoire des algorithmes

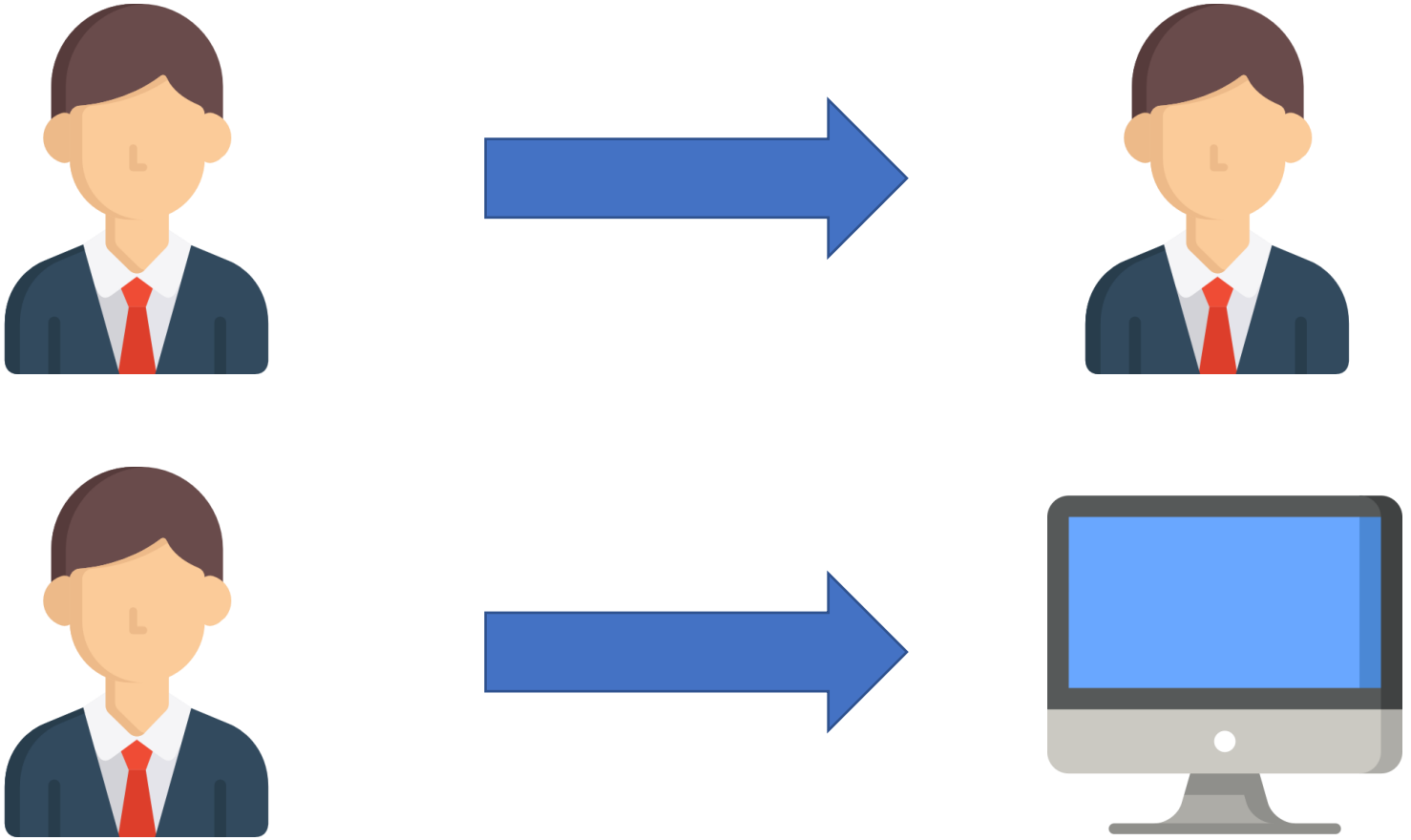
La "minute culturelle"

Algorithmes **sans ordinateurs** :

- Euclide (vers -300) : calcul du PGCD de 2 nombres
- Al-Khuwārizmī (825) : résolution d'équations
- Ada Lovelace (1842) : calcul des nombres de Bernoulli sur la *machine analytique* de Charles Babbage

A quoi sert un algorithme ?

Transmettre du savoir faire :



A quoi sert un algorithme ?

A décrire les étapes de résolution d'un problème:

- De façon structurée et compacte
- A partir des opérations de base
- Indépendamment du langage de programmation

A quoi sert un algorithme ?

A décrire les étapes de résolution d'un problème:

- **De façon structurée et compacte**
- A partir des opérations de base
- Indépendamment du langage de programmation

Méthode de résolution d'un problème :

- Facile à comprendre
- Facile à transmettre

A quoi sert un algorithme ?

A décrire les étapes de résolution d'un problème:

- De façon structurée et compacte
- **A partir des opérations de base**
- Indépendamment du langage de programmation

Méthode de résolution d'un problème :

- Adapté au moyens à disposition
- Adapté aux connaissances de celui qui l'utilise

A quoi sert un algorithme ?

A décrire les étapes de résolution d'un problème:

- De façon structurée et compacte
- A partir des opérations de base
- **Indépendamment du langage de programmation**

Méthode de résolution d'un problème :

- Adapté pour les problème qui se traitent sans ordinateurs
- Compréhensible sans apprendre un langage de programmation

Comment représenter un algorithme ?

Trois façons adopté dans notre cours pour décrire nos algorithmes:

- Pseudo-code
- Organigramme
- Python

Qu’est ce qu’un problème ?

Un problème en algorithmique peut être défini comme une tâche ou une question à résoudre, formulée de manière claire et précise, où l'objectif est de trouver une solution efficace et correcte en utilisant une série d'étapes logiques et bien définies.

Exemples :

- Calculer le montant d’une marchandise
- Trouver la valeur maximale dans liste de valeurs
- Résolution des systèmes linéaires
- Résolution des équations différentielles
- …

Problème VS Instance d’un problème

Exemple d'un problème :

Donner le temps nécessaire pour aller d’une ville X à une ville Y.

Une instance du problème 1:

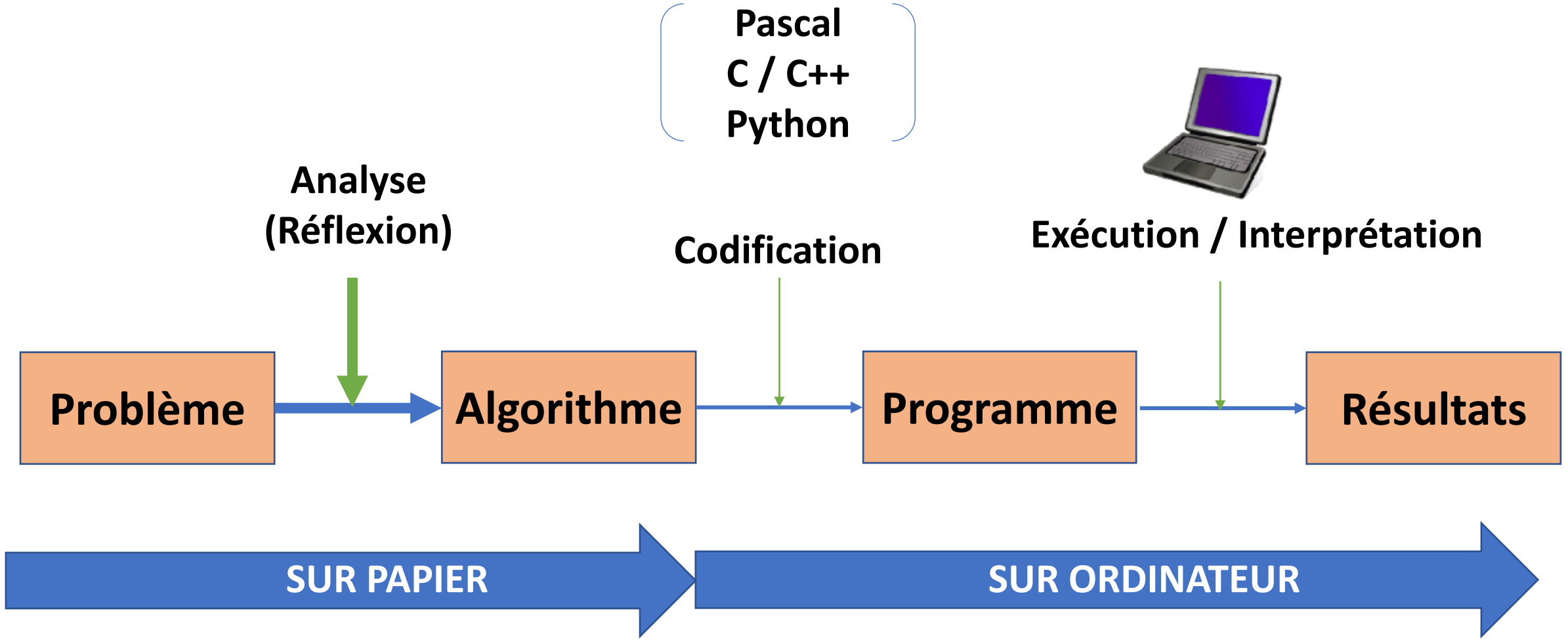
Donner le temps nécessaire pour aller de Casablanca à Marrakech

Une autre instance du problème 1:

Donner le temps nécessaire pour aller de Casablanca à Tanger

Structure Générale d'un Algorithme

Etapes de la programmation



Analyser un problème :

Analyser un problème sert à décrire le problème par ces composantes :

Les données d'entré : C'est les données indispensables à connaître pour pourvoir résoudre le problème.

Les données de sortie: Se sont les résultats recherchés.

Le traitement : L'ensemble des opérations à appliquer sur les données d'entré pour aboutir aux résultats.

Analyser un problème : Exemple 1

Problème:

Calculer le montant TTC à payer pour un lot de PC portable sachant le taux de TVA est de 20%.

Question : Analyser le problème

Analyser un problème : Exemple 1

Données d'entré :

Donnée	Identificateur	Type	Catégorie
Prix Unitaire Hors Taxe	PUHT	Réel	Variable
Nombre de PC	NP	Entier	Variable
Taux de TVA	TVA	Réel	Constante

Données de sortie :

Donnée	Identificateur	Type	Catégorie
Montant TTC	MTTC	Réel	Variable

Analyser un problème : Exemple 1

Traitement :

MTTC = NP x PUHT x (1 + TVA)

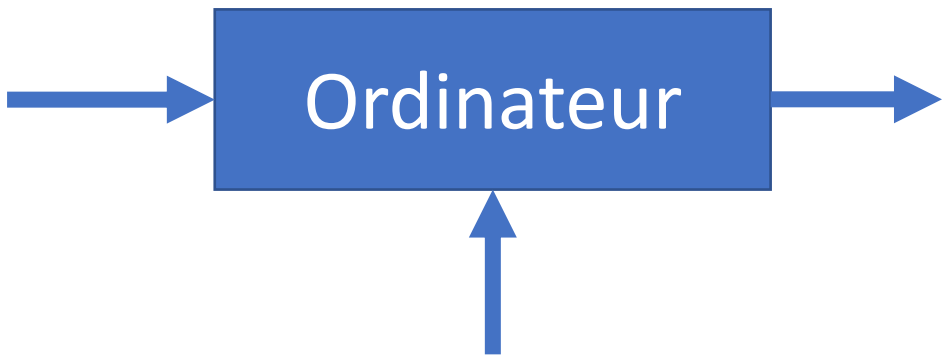
Analyser un problème : Exemple 1

Données :

TVA

NP

PUHT



Résultats:

MTTC

Traitement :

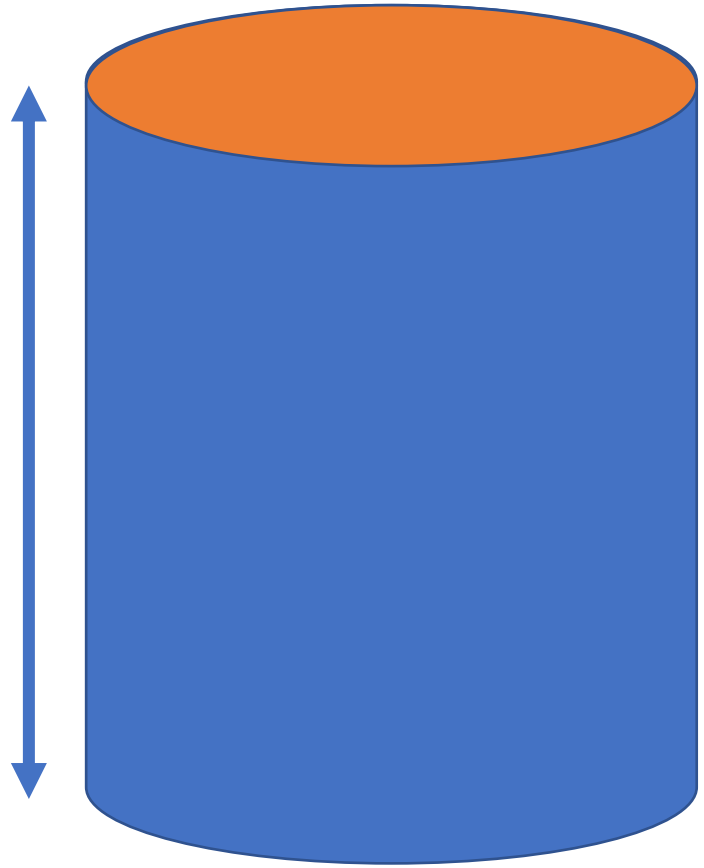
MTTC = NP x PUHT x (1 + TVA)

Analyser un problème : Exemple 2

Problème :

Calculer le volume d’un cylindre.

Question : Analyser le problème



Analyser un problème : Exemple 2

Données d'entré :

Donnée	Identificateur	Type	Catégorie

Données de sortie :

Donnée	Identificateur	Type	Catégorie

Analyser un problème : Exemple 2

Traitement :

$V = Pi \times R \times R \times H$

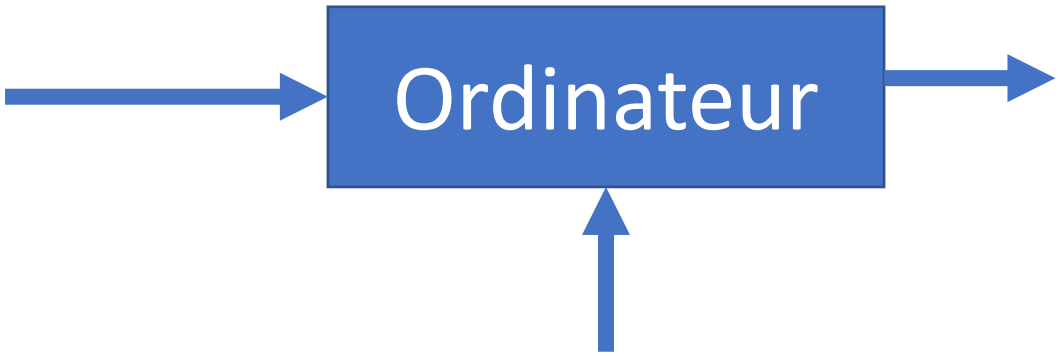
Analyser un problème : Exemple 2

Données :

Pi

R

H



Résultats:

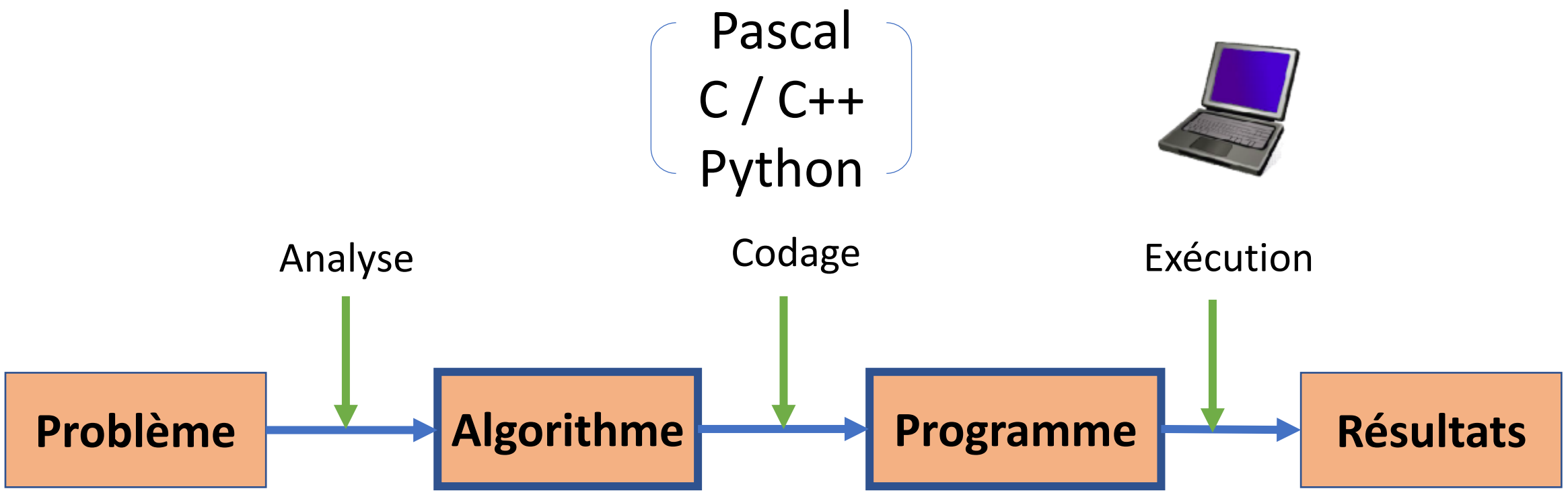
V

Traitement :

V = Pi x R x R x H

Structure Générale d'un Algorithme

Analyser un problème



Structure Générale d'un Algorithme en pseudo-code

L'en-tête

Algorithme NomAlgorithme

La partie déclarative

Constantes Identificateur = Valeur

Variables Identifiant1 : Type1
Identifiant2 : Type2

Le corps de l'algorithme

Début

Instruction 1;
Instruction 2;
….
Instruction n;

Fin

Structure Générale d'un Algorithme en pseudo-code

Exemple 1

Algorithme Montant TTC
Constantes TVA = 0.2
Variables PUHT, MTTC : Réel
NP : Entier
Début
MTTC = PUHT x NP x (1 + TVA)
Fin

NB : Cet algorithme n'est pas complet

Structure Générale d'un Algorithme en pseudo-code

Exemple 2

Algorithme Volume Cylindre
Constantes Pi = 3.14
Variables R, H, V : Réel
Début
    V = Pi x R x R x H
Fin

NB : Cet algorithme n'est pas complet

Les variables

Définition

Il s'agit d'un **nom symbolique** utilisé pour représenter une valeur ou une information que le programme manipule. Cette valeur peut changer ou être réassignée au cours de l'exécution d'un programme, d'où le terme "variable".

Dans un langage de programmation, une variable est un espace de stockage.

Caractéristiques :

- Un identificateur
- Un type

Nommage des variables

Règles pour nommer une variable :

En partie par obligation liée au langage, et en partie par convention pour l'enseignement, un nom de variable doit :

1. Contenir que des lettres ou des chiffres
2. Débuter avec une lettre
3. Utilisation du caractère souligné : _ (underscore)

Nommage des variables

Exemples :

Variable	Accepté ?
Prix	
1prix	
prix_2	
prix@	

Types des variables :

Nous distinguons 4 types de base des variables :

1. **Entier** : les valeurs relatives n ϵ Z (exemple: -1, 0, 100).
2. **Flottant (Réel)** : les valeurs réels x ϵ IR (0.12, 10.46, -13.5).
3. **Chaîne de caractères** : "AC", "1B", "D", "Bonjour Ahmed"
4. **Booléen**: support deux valeurs Vrai (ou bien 1) Faux (ou bien 0)

Le type Flottant :

Opérations	Symbole	Exemple a=5 et b=4
Addition	+	a + b = 9
Soustraction	-	a – b = 1
Multiplication	*	a * b = 2
Division réelle	/	a / b = 1.25
Exposant	^	A^b = 625
Comparaisons	<, = , > , <= , >=, !=	a=b => False

Le type Entier :

Opérations	Symbole	Exemple a=5 et b=4
Addition	+	a + b = 9
Soustraction	-	a – b = 1
Multiplication	*	a * b = 2
Division réelle	/	a / b = 1.25
Exposant	^	A^b = 625
Comparaisons	<, = , > , <= , >=, !=	a=b => False
Quotient division entière	div	a div b = 1
Reste division entière	mod	a mod b = 1

Le type Chaîne de caractères :

Opérations	Symbole	Exemple a = "ab" et b="bc"
Concaténation	+	a + b = "abcd"
Multiplication	*	a * 2 = "abab"
Comparaisons	<, = , > , <= , >=, !=	a=b => False

Le type Booléen:

Opérations	Symbole	Exemple a = True et b=False
Et	and	a ET b → False
Ou	or	a OU b → True
Non	not	Non a → False

Exemple 3

Problème :

Calculer la moyenne de deux notes, les notes ont des coefficients différents.

Exemples

Exemple 3

Données d'entré :

Donnée	Identificateur	Type	Catégorie
La première note	N1	Réel	Variable
La deuxième note	N2	Réel	Variable
Le coefficient de la 1ère note	C1	Entier	Variable
Le coefficient de la 2ème note	C2	Entier	Variable

Données de sortie :

Donnée	Identificateur	Type	Catégorie
La moyenne	M	Réel	Variable

Exemple 3

Traitement :

$$M = (N1*C1 + N2*C2) / (C1+C2)$$

Exemple 3

Algorithme Moyenne
Variables N1, N2: Réel
C1, C2 : Entier
Début
M = (N1*C1 + N2*C2) / (C1 + C2)
Fin

NB : Cet algorithme n'est pas complet

Exemple 4

Problème :

Calculer la surface d'un rectangle.

Exemples

Exemple 4

Données d'entré :

Donnée	Identificateur	Type	Catégorie
La longueur	L	Réel	Variable
La largeur	R	Réel	Variable

Données de sortie :

Donnée	Identificateur	Type	Catégorie
La surface	S	Réel	Variable

Exemple 4

Traitement :

$$S = L * R$$

Exemple 4

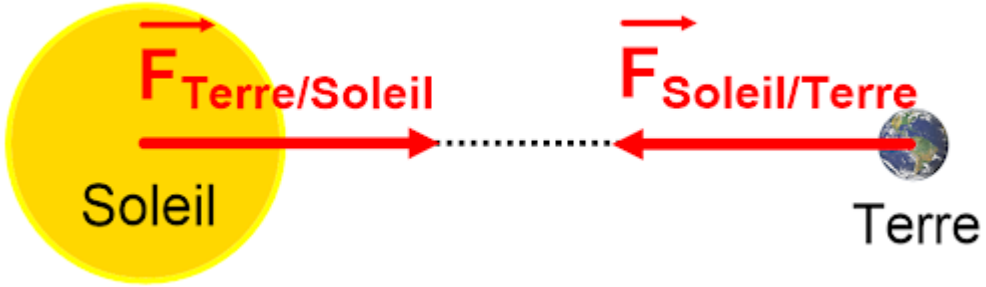
Algorithme Surface Rectangle
Variables L, R, S: Réel
Début
    S = L * R
Fin

NB : Cet algorithme n'est pas complet

Exemple 5

Problème :

Calculer la force gravitationnelle entre deux masses en utilisant la loi de la gravitation universelle de Newton.



$$F(\mathrm{A/B}) = F(\mathrm{B/A}) = \frac{G \cdot m(\mathrm{A}) \cdot m(\mathrm{B})}{d^2}$$

Exemples

Exemple 5

Données d'entré :

Donnée	Identificateur	Type	Catégorie
La masse du premier objet	m1	Réel	Variable
La masse du deuxième objet	m2	Réel	Variable
La distance entre les centres	d	Réel	Variable
La constante gravitationnelle	G	Réel	Constante

Données de sortie :

Donnée	Identificateur	Type	Catégorie
La force	f	Réel	Variable

Exemple 5

Traitement :

$$f = G*m1*m2 / (d^2)$$

Exemple 5

```
Algorithme   Force
Variables    m1, m2, f, d: Réel
Constantes   G = 6.67 * 10^(−11)
Début
    f  = G*m1*m2 / (d^2)
Fin
```

NB : Cet algorithme n'est pas complet

FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 2:
Les instructions de base

Licence : Physique Chimie
Filières : Chimie

Pr: Youssef Ouassit

Plan Séance 2

1. **Les instructions de base**
   a. L’affectation
   b. L’instruction d’Entré / Lecture / Saisit
   c. L’instruction de Sortie / Ecriture / Affichage

2. **Exemples**

Plan Séance 2

Problème :

Ecrire un algorithme qui calcule l'air total d'un cylindre.

Plan Séance 2

Algorithme Air Cylindre
Constantes Pi = 3.14
Variables R, H, SB , SL, A : Réel
Début
    SB = Pi*R^2
    SL = 2*Pi*R*H
    A = 2*SB + SL
Fin

A COMPLETER

Instruction d'affectation

Affectation

Quel problème dans cet algorithme ?

```
Algorithme Air Cylindre
Constantes Pi = 3.14
Variables R, H, SB , SL,  A : Réel
Début
        SB = Pi*R^2
        SL =  2*Pi*R*H
        A = 2*SB + SL
Fin
```

Affectation

Définition :

L’affectation permet d’affecter une valeur à une variable. Physiquement elle permet de stocker une valeur dans une case mémoire référencée par le nom de la variable.

Symbolisée en algorithmique par " ← ", elle précise le sens de l’affectation.

Syntaxe :

Variable ← Expression

Affectation

Exemples :

A ← 5

B ← 5 + 4

C ← 6 * 2 * (10 + 4)

D ← A + B

E ← A * B + C

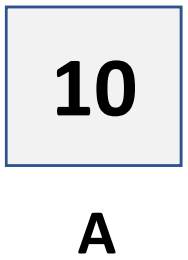
Affectation

Remarque 1:

Une variable ne peut contenir qu'une valeur à la fois.

$A \leftarrow 4$

$A \leftarrow 10$

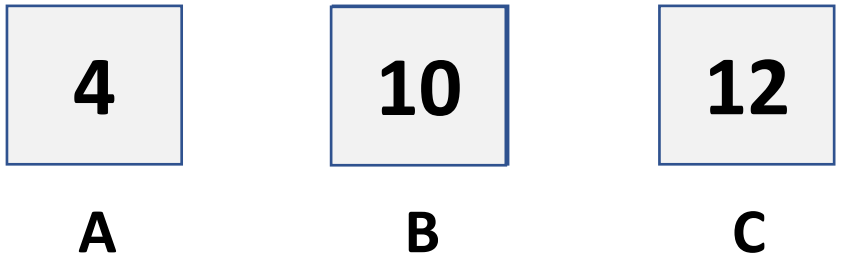


Affectation

Remarque 2:

Les valeurs des variables évoluent au cours de l'exécution, et les cases mémoire des variables sont indépendantes :

A ← 4
B ← 8
C ← A+B
B ← 10



Les instructions de base

Affectation

Exercice 1:

Quelles seront les valeurs des variables a, b et c après exécution des instructions suivantes :

a ← 1

b ← 5

c ← a – b

a ← 2

c ← a + b

Réponse : $a = 2$; $b = 5$; $c = 7$

Affectation

Exercice 2:

Quel seront les valeurs des variables A, B et C après l'exécution des instructions d'affectation suivantes :

	A	B	C
A ← 2			
B ← 6			
C ← A+ B			
C ← C + 1			
A ← C			
B ← B+C			

Affectation

Exercice 3:

Quel seront les valeurs des variables A, B et C après l'exécution des instructions d'affectation suivantes :

	A	B	C
A ← 2	2	?	?
B ← A+1	2	3	?
C ← B div 3	2	3	1
C ← C+1	2	3	2
A ← A mod 2	0	3	2
B ← B mod 10	0	3	2

Affectation

Corriger cet algorithme avec l'affectation :

```
Algorithme Air Cylindre
Constantes Pi = 3.14
Variables R, H, SB , SL,  A : Réel
Début
        SB = Pi*R^2
        SL =  2*Pi*R*H
        A = 2*SB + SL
Fin
```

Affectation

Algorithme corrigé :

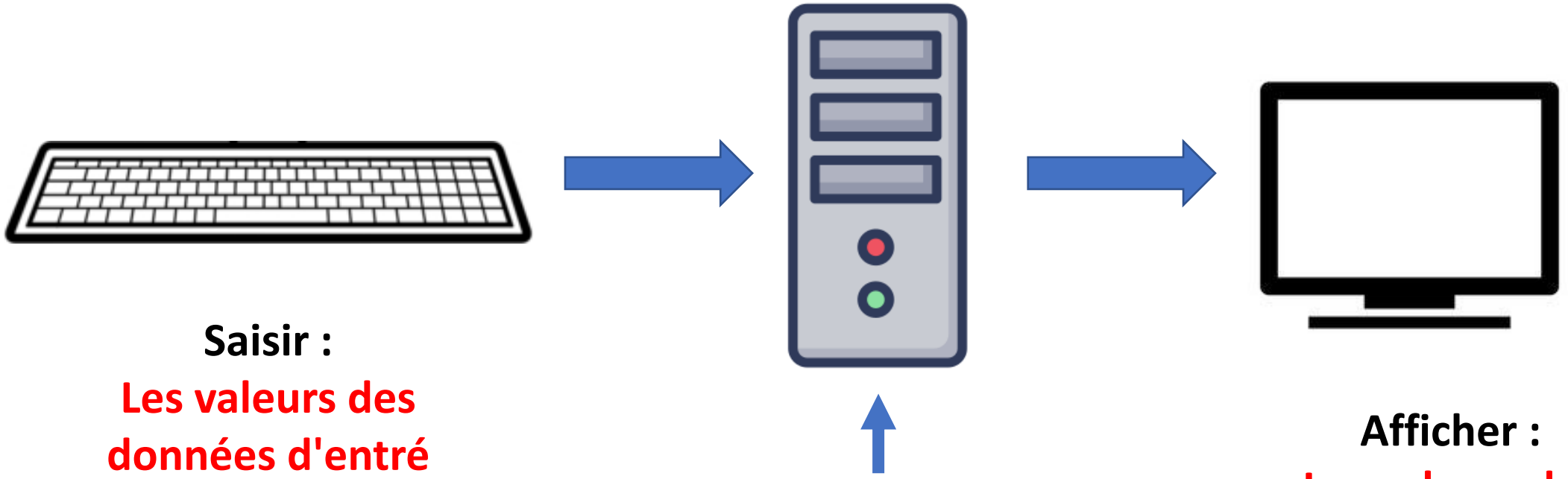
```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
Fin
```

A COMPLETER

Instruction d'entré / lecture

Les instructions de base

Instruction d’entré / lecture



Saisir :

Les valeurs des données d'entré

PROGRAMME

Afficher :

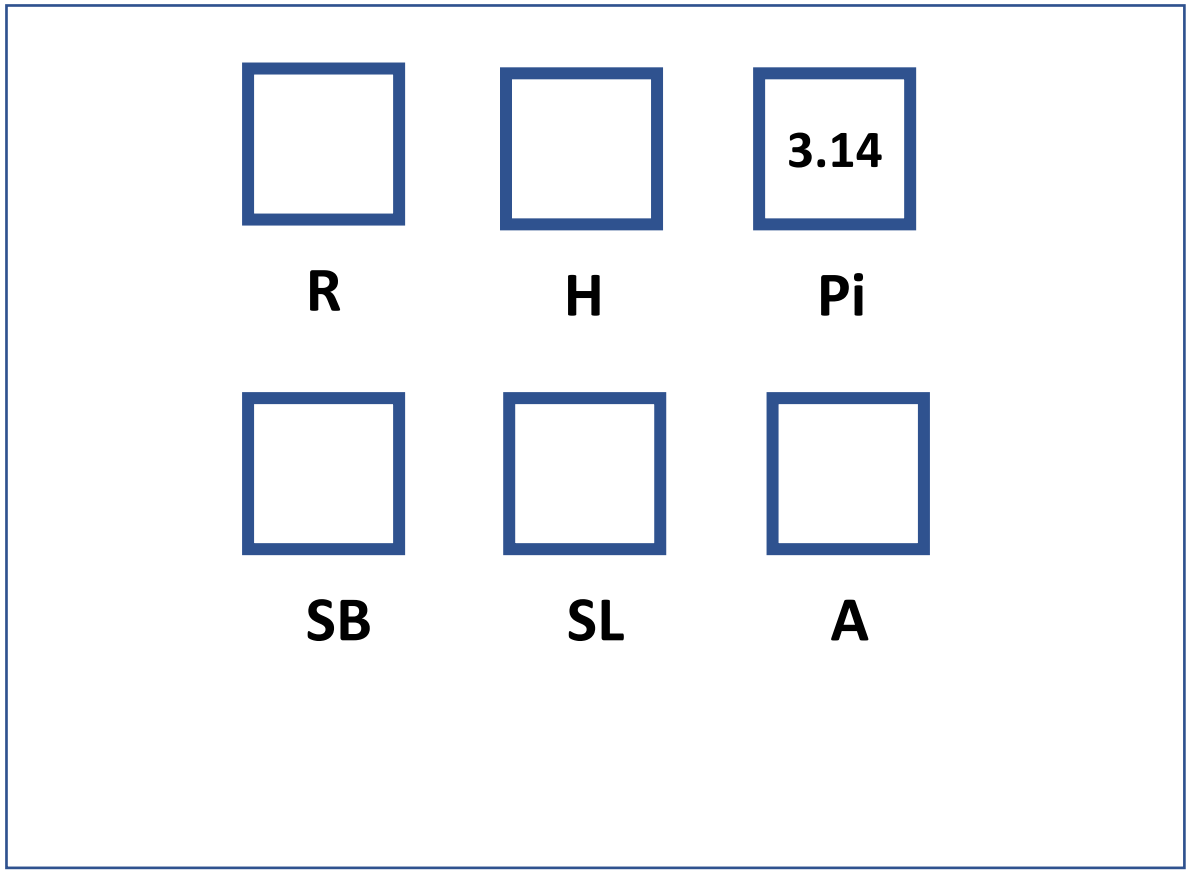
Les valeurs des données de sortie (résultats)

Instruction d'entré / lecture

Exemple illustratif:

```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
Fin
```

Etape 1 : Réservation des cases mémoire pour les variables et constantes déclarés



LA Mémoire RAM

Les instructions de base

Instruction d'entré / lecture / saisit

L'instruction d'entrée ou de lecture permet à l'utilisateur de saisir des données au clavier pour qu'elles soient utilisées par l'algorithme.

Cette instruction assigne (affecte) une valeur entrée au clavier dans une variable.

Syntaxe : Lire(variable)

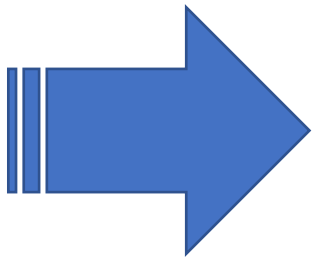
Exemple :

Lire(B) : Cette instruction permet à l'utilisateur de saisir une valeur au clavier qui sera affectée à la variable B.

Instruction d'entré / lecture

Corriger l'algorithme suivant :

```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
Fin
```



```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        Lire(R)
        Lire(H)
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
Fin
```

Instruction d’entré / lecture

Remarque :

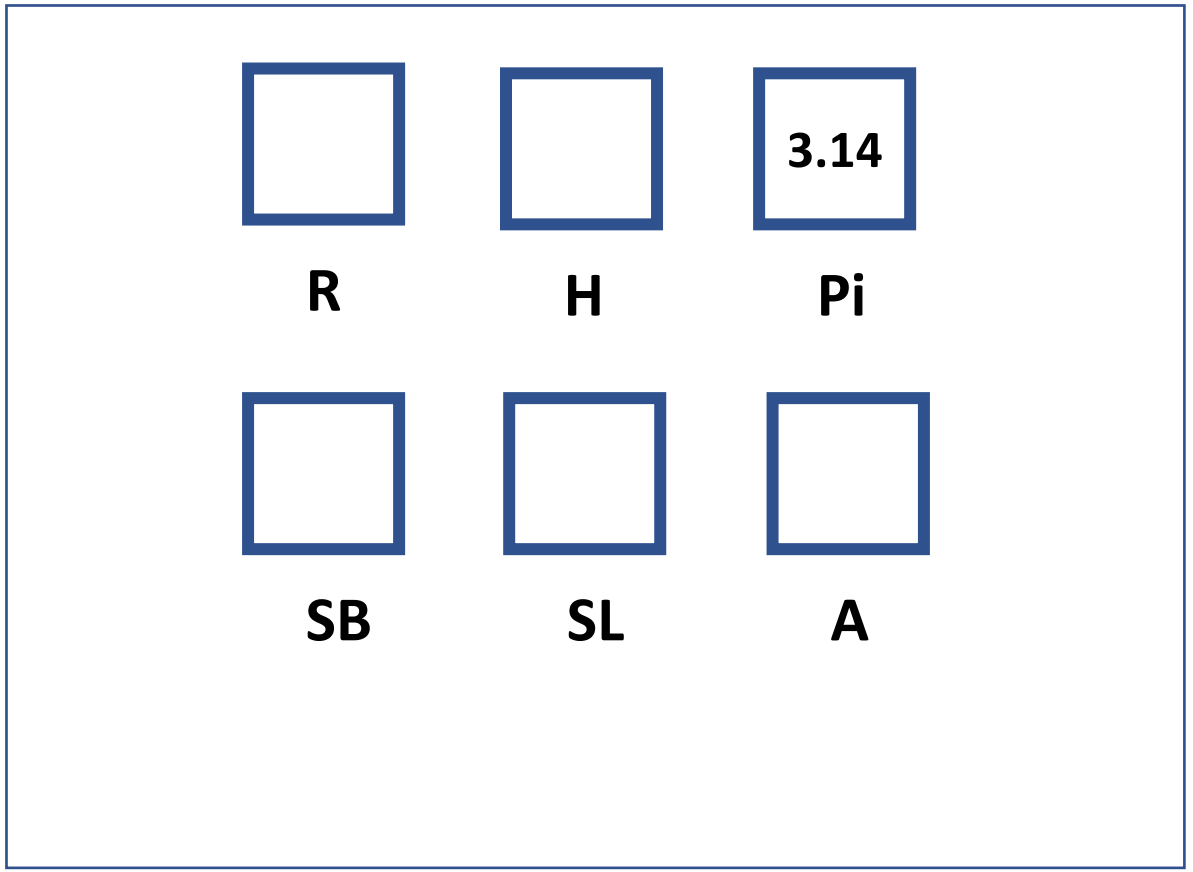
Lorsque le programme rencontre cette instruction, **l’exécution s’interrompt** et attend que l'utilisateur tape une valeur. Cette valeur est rangée en mémoire dans la variable désignée.

Instruction de sortie / écriture

Instruction d'entré / lecture

Exemple illustratif:

```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        Lire(R)
        Lire(H)
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
Fin
```



La Mémoire RAM

Les instructions de base

Instruction de sortie / écriture

L'instruction de sortie (d'écriture) permet d'afficher des informations à l'utilisateur à travers l'écran.

Syntaxe : Ecrire(expression)

Expression peut être une valeur, un résultat, un message, le contenu d'une variable...

Exemple :

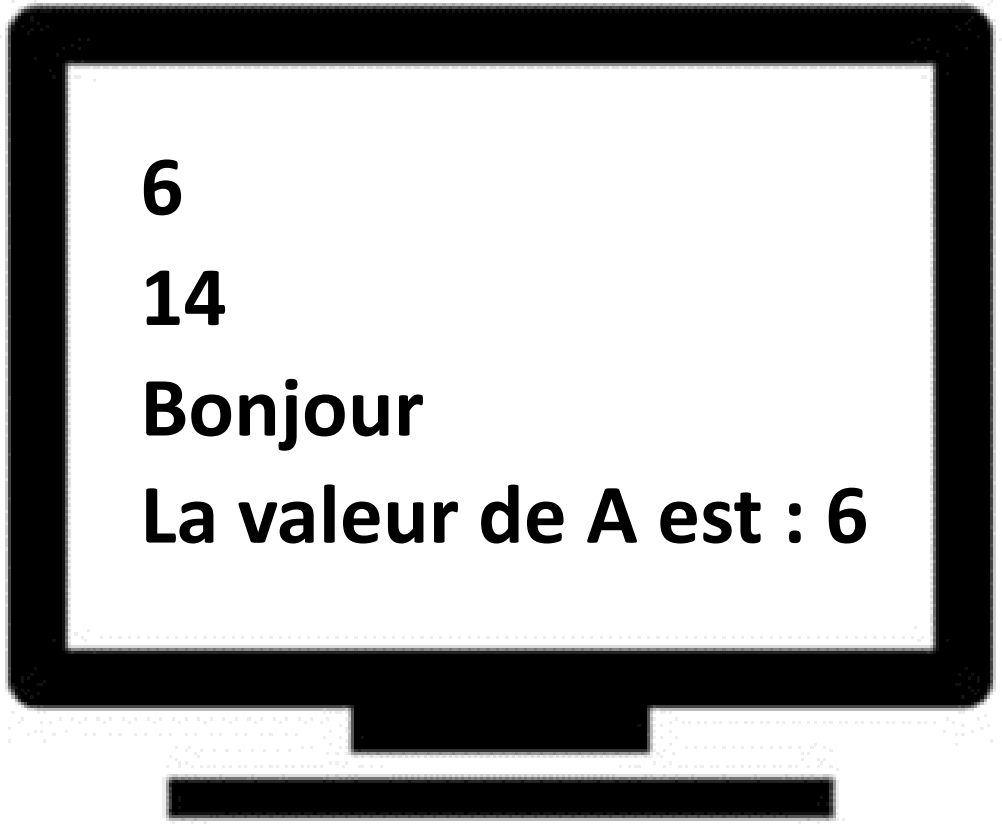
Ecrire(A) : Cette instruction permet d'afficher à l'écran la valeur de la variable A.

Instruction de sortie / écriture

Exemples:

A ← 6
B ← 8

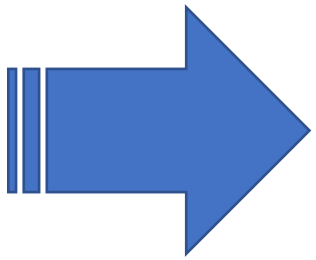
Ecrire(A)
Ecrire(A+B)
Ecrire("Bonjour")
Ecrire("La valeur de A est : ", A)



Instruction d'entré / lecture

Corriger l'algorithme suivant :

```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        Lire(R)
        Lire(H)
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
Fin
```

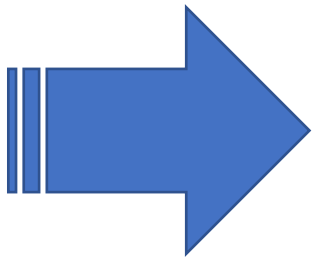


```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        Lire(R)
        Lire(H)
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
        Ecrire("L'air est : ", A)
Fin
```

Instruction d'entré / lecture

Améliorer l'algorithme :

```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        Lire(R)
        Lire(H)
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
        Ecrire(A)
Fin
```



```
Algorithme Air Cylindre
Constantes Pi ← 3.14
Variables R, H, SB , SL,  A : Réel
Début
        Ecrire("Donner le rayon : ")
        Lire(R)
        Ecrire("Donner la hauteur : ")
        Lire(H)
        SB ← Pi*R^2
        SL ←  2*Pi*R*H
        A ← 2*SB + SL
        Ecrire("L'air est : ", A)
Fin
```

Les instructions de base

Exemples :

Compléter les algorithme des deux premiers problèmes :

- Calculer le montant TTC
- Calculer la surface d’un cylindre

Instruction d'entré / lecture : Exemple 1

Première correction : Utiliser le symbole d'affectation

Algorithme Montant TTC

Constantes TVA = 0.2

Variables PUHT, MHT, MTVA, MTTC : Réels

NP : Entier

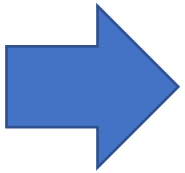
Début

MHT = PUHT * NP

MTVA = MHT * TVA

MTTC = MHT * MTVA

Fin



Algorithme Montant TTC

Constantes TVA ← 0.2

Variables PUHT, MHT, MTVA, MTTC : Réels

NP : Entier

Début

MHT ← PUHT * NP

MTVA ← MHT * TVA

MTTC ← MHT * MTVA

Fin

Instruction d'entré / lecture : Exemple 1

Deuxième correction : Utiliser les instructions de lecture/écriture

```
Algorithme  Montant TTC
Constantes  TVA ← 0.2
Variables   PUHT, MHT, MTVA, MTTC : Réels
                NP : Entier
Début
        Ecrire("Donner le prix unitaire HT :")
        Lire (PUHT)
        Ecrire("Donner le nombre de PC :")
        Lire (NP)
        MHT ← PUHT * NP
        MTVA ← MHT * TVA
        MTTC ← MHT * MTVA
        Ecrire("Le montant TTC est  ", MTTC, "DH")
Fin
```

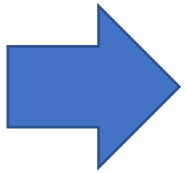
Instruction d'entré / lecture : Exemple 2

Première correction : Utiliser le symbole d'affectation

Algorithme Volume Cylindre
Constantes Pi = 3.14
Variables R, H, SB, V : Réels
Début

SB = Pi*R*R
V = SB * H

Fin



Algorithme Volume Cylindre
Constantes Pi ← 3.14
Variables R, H, SB, V : Réels
Début

SB ← Pi*R*R
V ← SB * H

Fin

Instruction d'entré / lecture : Exemple 2

Deuxième correction : Ajouter les instructions de lecture et écriture

```
Algorithme  Volume Cylindre
Constantes  Pi ← 3.14
Variables     R, H, SB, V : Réels
Début
        Ecrire("Donner le rayon de la base :")
        Lire (R)
        Ecrire("Donner la hauteur :")
        Lire (H)
        SB ← Pi*R*R
        V ← SB * H
        Ecrire("Le Volume est  ", V, "m3")
Fin
```

Instruction d’entré / lecture : Exemple 3

Exemple 3:

Ecrire un algorithme qui calcule l'air d'un rectangle

Données d'entré : L , R
Données de sortie : A
Traitement : A = L * R

Instruction d'entré / lecture : Exemple 3

Exemple 3: Air d'un rectangle

```
Algorithme   Air Rectangle
Variables    L, R, A: Réel
Début
        Ecrire("Donner la longueur")
        Lire (L)
        Ecrire("Donner la largeur")
        Lire (R)
        A ← L * R
        Ecrire("La surface est", A , "m2")
Fin
```

FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 3:
Les structures conditionnelles

Licence : Physique Chimie
Filières : Chimie

Pr: Youssef Ouassit

Plan Séance 3

Les structures de contrôles :

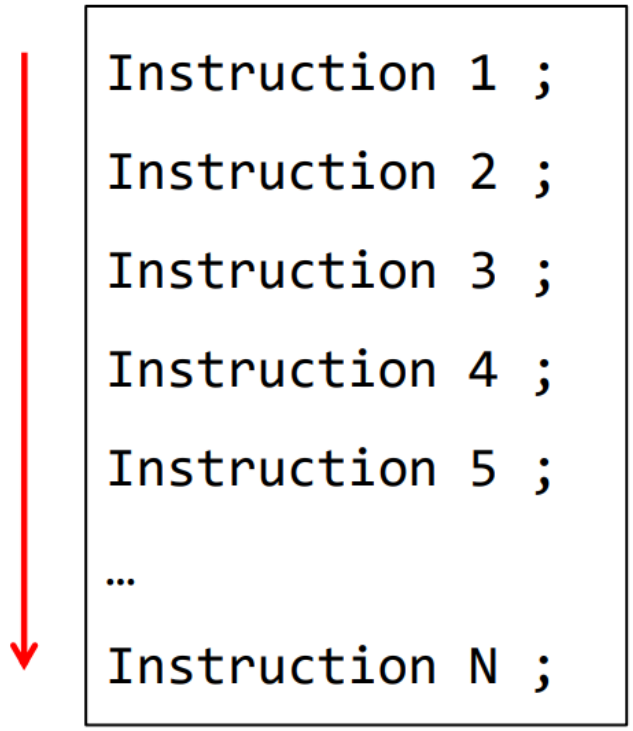
1. **Les structures conditionnelles (de choix)**
   a. L’instruction conditionnelle simple
   b. L'instruction conditionnelle alternative
   c. Structure conditionnelle multiple

2. **Les instructions d’itération ou de répétition (les boucles)**
   a) La boucle Pour … Faire
   b) La boucle Tant Que

Introduction

Structures séquentielles :

Une **structure algorithmique séquentielle** est la forme la plus simple d'organisation d'un programme où les instructions sont exécutées l'une après l'autre, dans l'ordre dans lequel elles sont écrites. Cette structure ne contient ni branches ni boucles; chaque étape est suivie par la suivante sans saut ni répétition.

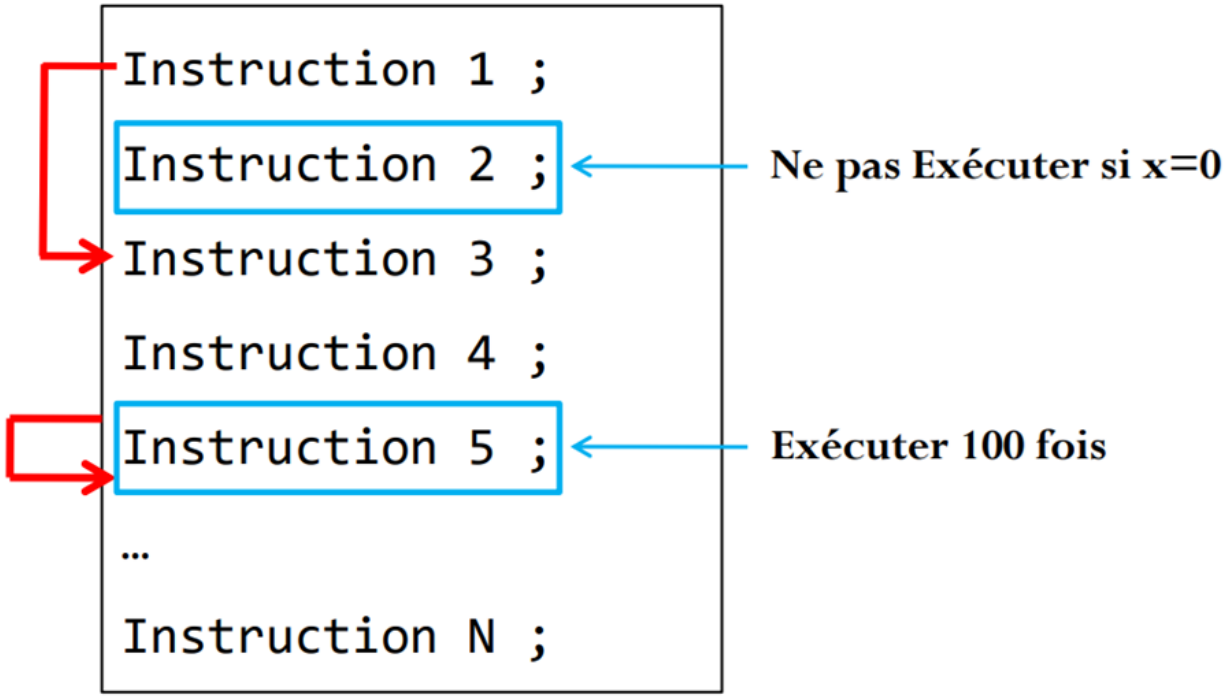


Introduction

Dans Certains Cas, Nous avons besoin de modifier l'ordre d'exécution des instructions :

Par exemple :

- Ne Pas Exécuter l' Instruction 2 si la variable x vaut 0.
- Exécuter L' Instruction 5 100 fois

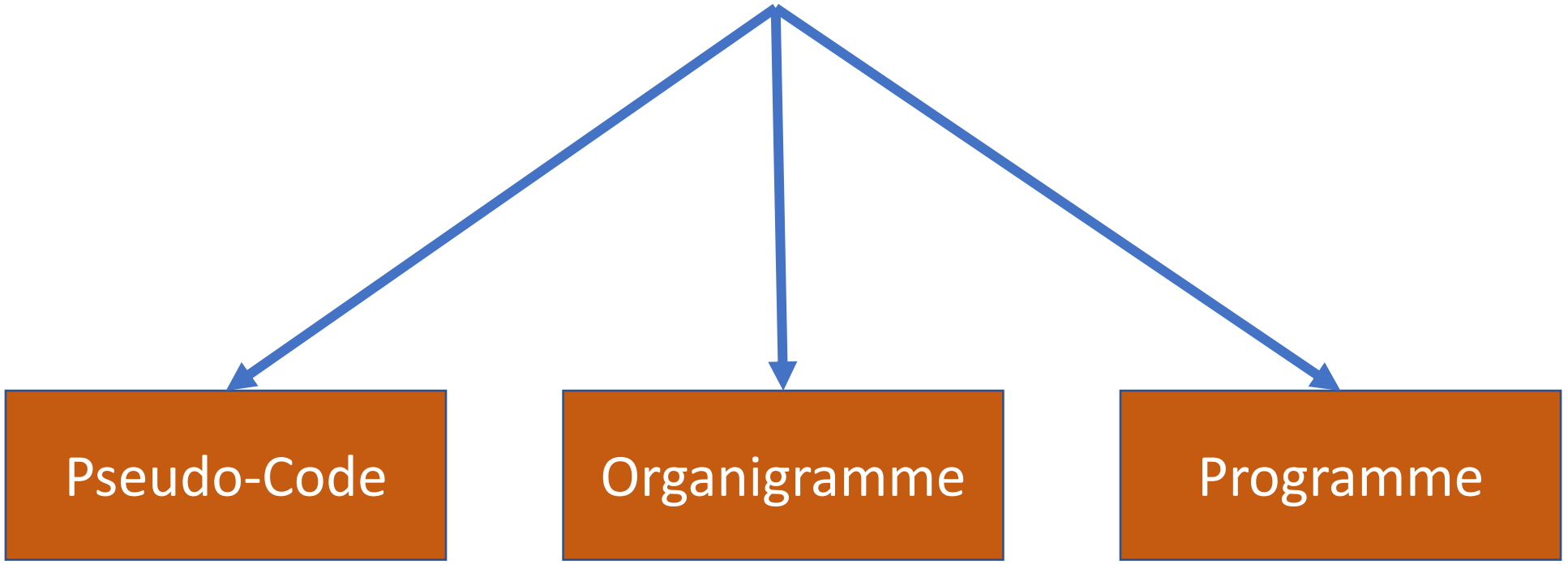


Structure Conditionnelle Alternative

Structures conditionnelles

Représentation des algorithmes

Trois façons pour représenter un algorithme:



Structures conditionnelles

Représentation des algorithmes

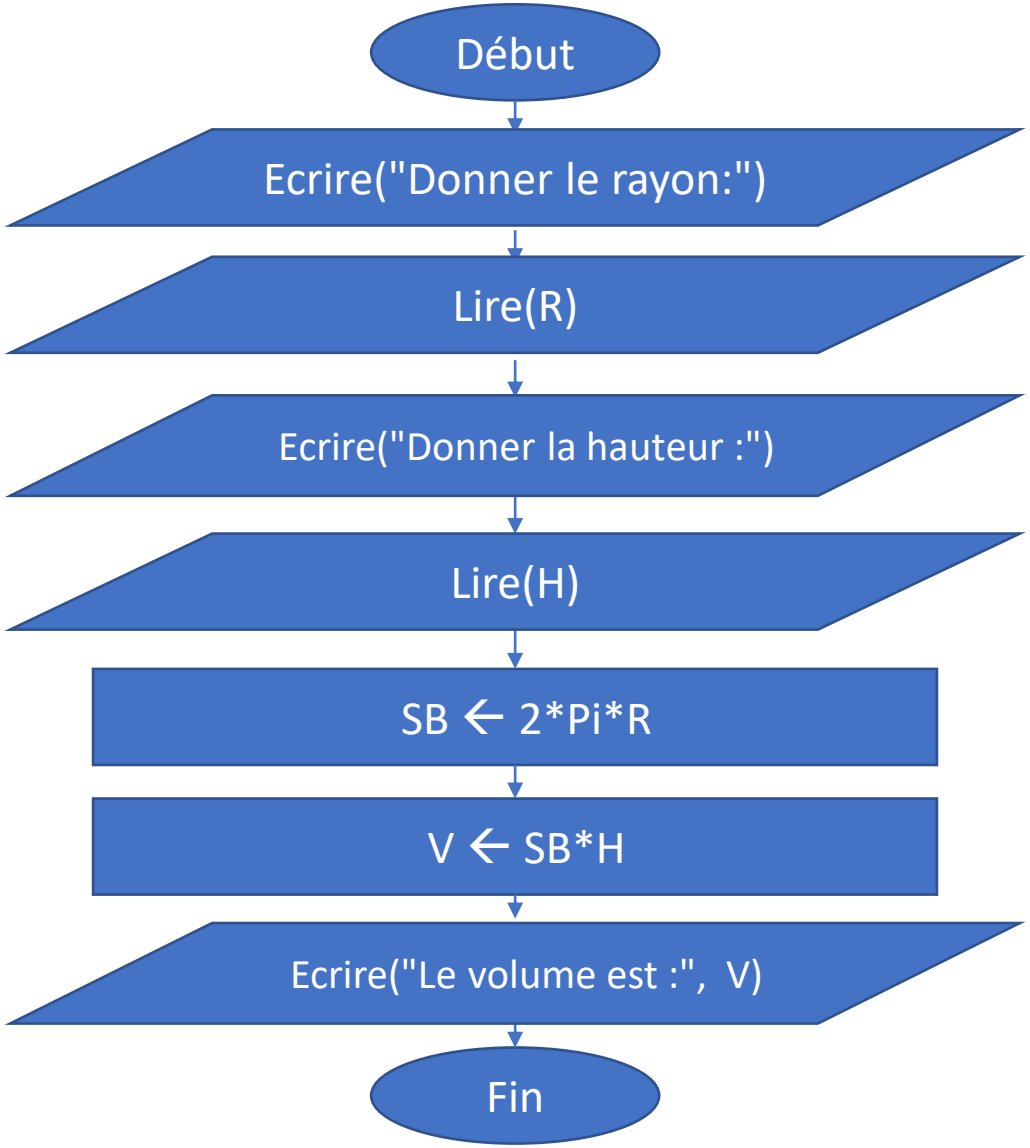
Un **organigramme d'algorithme** (ou diagramme de flux) est une représentation graphique des étapes d'un algorithme. Il utilise des symboles pour décrire les actions et les décisions prises au fil du déroulement de l'algorithme. Voici les principaux symboles utilisés dans un organigramme :

1) **Ovale** : Représente le début ou la fin de l'algorithme.
2) **Rectangle** : Représente une action ou un processus, comme une opération de calcul.
3) **Parallélogramme** : Représente une entrée ou une sortie (lecture ou écriture de données).
4) **Losange** : Représente une décision (ou un test conditionnel), avec deux flèches sortantes : une pour "vrai" et une pour "faux".

Structures conditionnelles

Organigramme

```
Algorithme Volume
Variables : R, H, V, SB : Réel
Constantes : Pi = 3.14
Début
        Ecrire("Donner le rayon: ")
        Lire(R)
        Ecrire("Donner la hauteur: ")
        Lire(H)
        SB ← 2*Pi*R
        V ← SB*H
        Ecrire("Le volume est " , V)
Fin
```



Structures conditionnelles alternatives

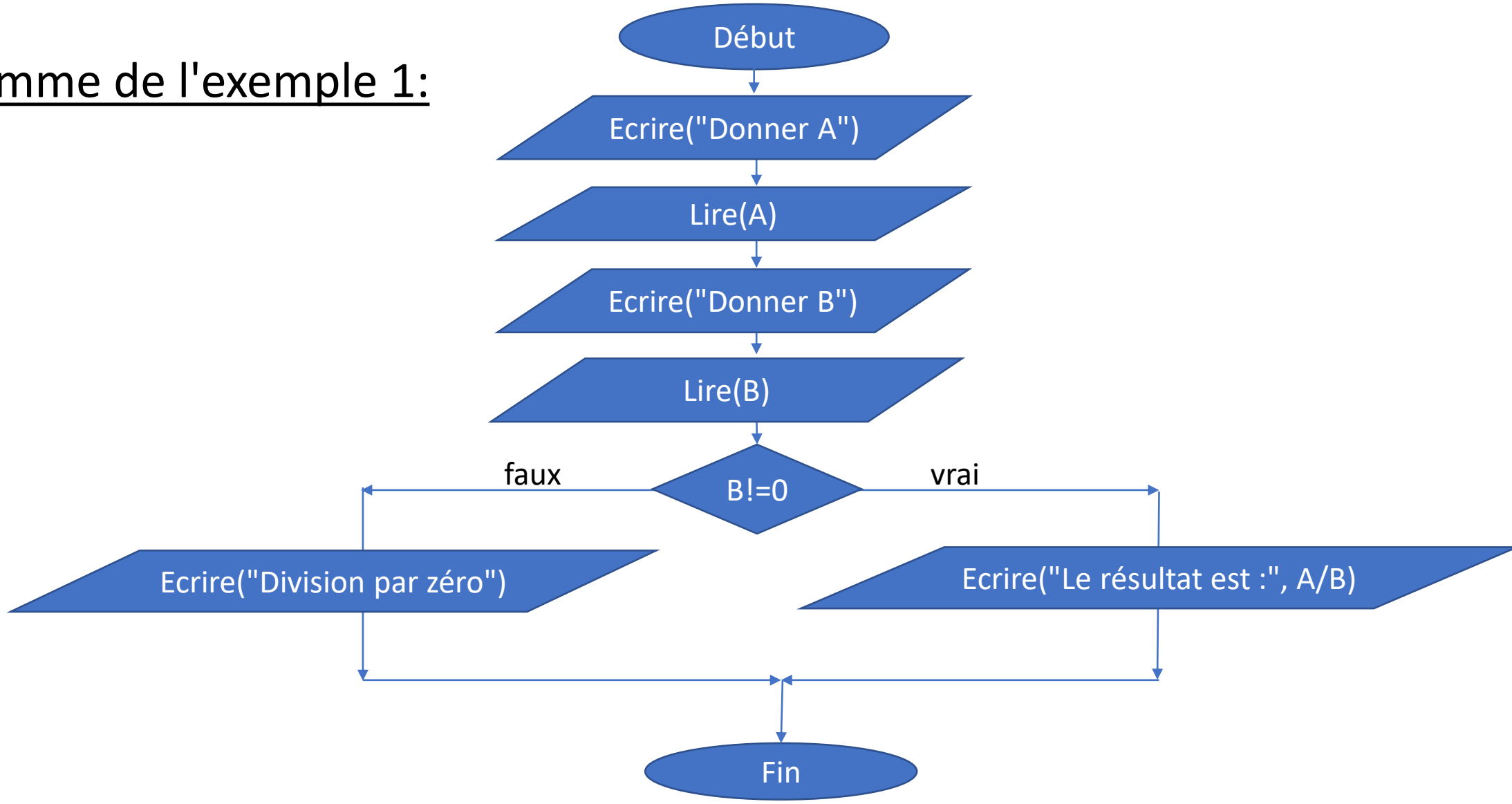
Exemple 1:

Un algorithme qui calcul la division de deux nombres.

```
Algorithme Division
Variables : A, B : Réel
Début
        Ecrire("Donner A: ")
        Lire(A)
        Ecrire("Donner B:")
        Lire(B)
        Ecrire("Le résultat : ", A/B)
        Ecrire("Division par zéro")
Fin
```

Structures conditionnelles alternatives

Organigramme de l'exemple 1:



Structures conditionnelles alternatives

Syntaxe :

Si condition Alors

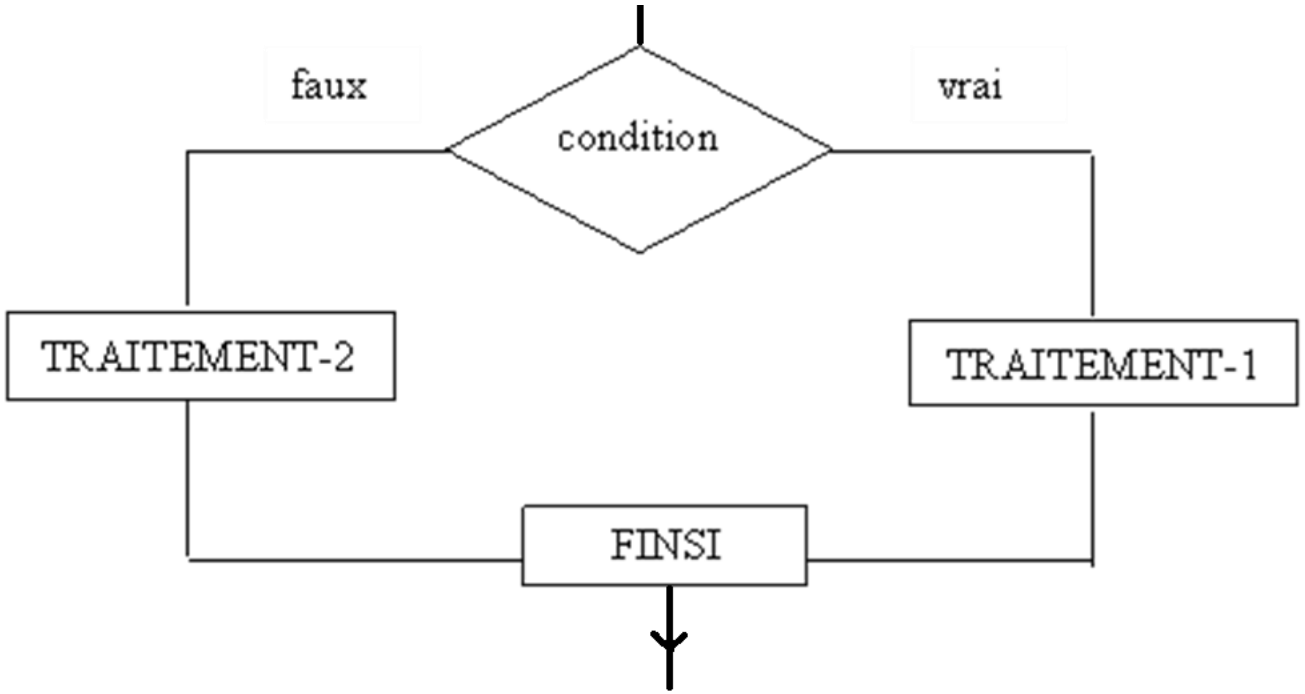
TRAITEMENT 1

SiNon

TRAITEMENT 2

FinSI

Organigramme:



Structures conditionnelles

Structures conditionnelles alternatives

Syntaxe :

```
Si B!=0 Alors
        Ecrire("Le résultat:", A/B)
SiNon
        Ecrire("Division par zéro")
FinSI
```

Algorithme de l'exemple 1:

```
Algorithme Division
Variables : A, B : Réel
Début
        Ecrire("Donner A: ")
        Lire(A)
        Ecrire("Donner B:")
        Lire(B)
        Si B!=0 Alors
                Ecrire("Le résultat:", A/B)
        SiNon
                Ecrire("Division par zéro")
        FinSI
Fin
```

Structures conditionnelles alternatives

Qu'est ce qu'une condition ?

Une condition est une expression booléenne, elle est soit vraie soit fausse.
La condition est soit simple, soit composée.

1- Condition simple : La condition simple est composée de deux opérandes et un opérateur de comparaison.
Les opérateurs de comparaison : , <=, >=, =, <>.

Exemples:

X = 2*Y X <= 4 X != 0 A mod 2 = 0

Structures conditionnelles alternatives

2- Condition composée : La condition composée comporte plus qu'une condition simple qui sont reliées par des opérateurs logiques : ET, OU, NON.

Exemples :

- $A \in [0,20]$ est exprimé par : **(A >=0) ET (A <=20)**
- $A \notin [0,20]$ est exprimé par : **(A <0) OU (A > 20)**
- $A \notin [0,20]$ est exprimé par : **Non ((A >=0) ET (A <=20))**
- X est un multiple de 5 ou 7 : **(X mod 5 = 0) OU (X mod 7 =0)**

Structures conditionnelles alternatives

Exercice:

Soit x, y et z des entiers, Donner les expressions booléennes correspondant aux situations suivantes :

- x, y et z sont identiques.
  - **x=y et y=z**
- x est pair
  - **x mod 2 = 0**
- x est impair.
  - **x mod 2 = 1**
- $x \in [y, z]$
  - **(A >=0) ET (A <=20)**
- x est un multiple de y
  - **x mod y = 0**
- x est un nombre divisible par 5 mais pas par 10
  - **(x mod 5 = 0) ET (x mod 10 != 0)**

Structure Conditionnelle Simple

Structure conditionnelle simple

Exemple 2: Un étudiant est accepté

Un Algorithme qui lit la note d'un étudiant et détermine si il est admis ou non.

Et si on ne s'intéresse qu'au cas des admis ?

```
Algorithme Admis
Variables : M : Réel
Début
        Ecrire("Votre moyenne : ")
        Lire(M)
        Si M >= 10 Alors
                Ecrire("Vous êtes admis")
        SiNon
                Ecrire("Vous n'êtes pas admis")
        FinSi
Fin
```

Structures conditionnelles

Structure conditionnelle simple

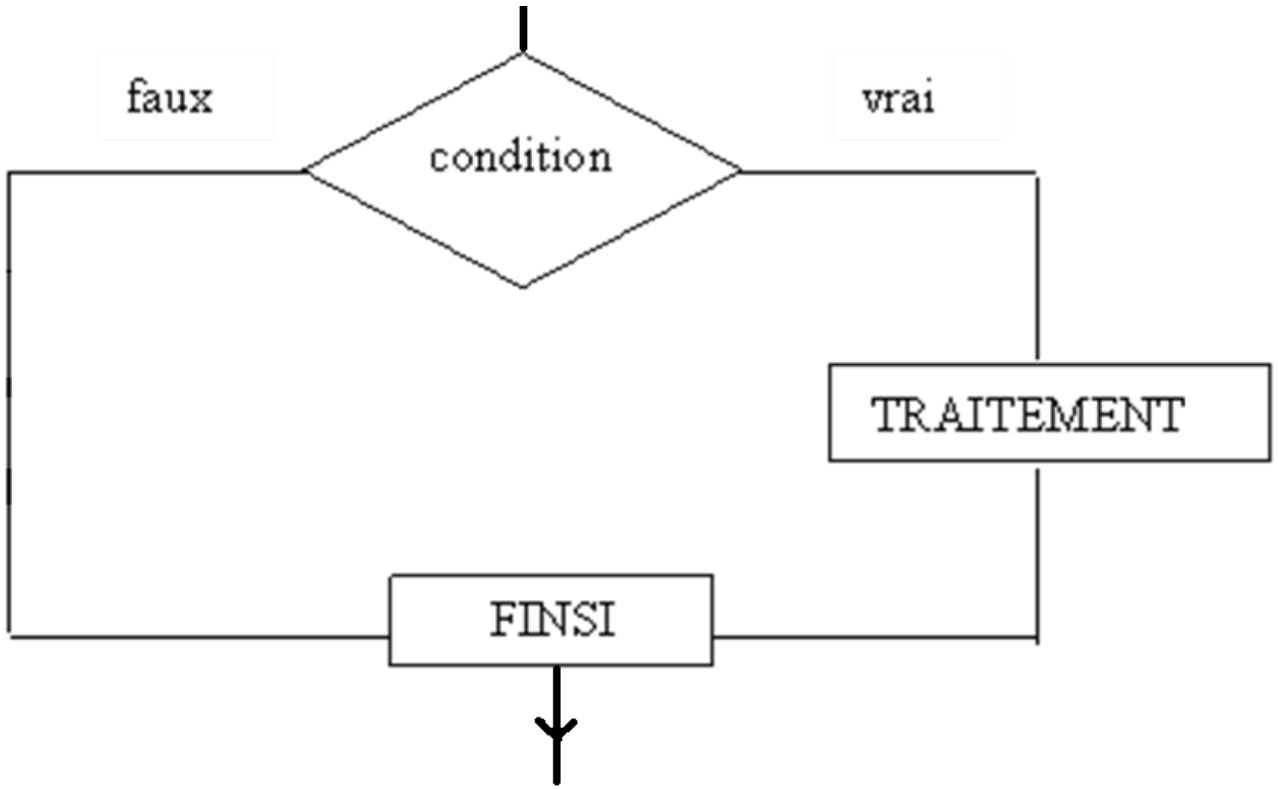
Syntaxe :

Si condition Alors

TRAITEMENT

FinSI

Organigramme:



Structure conditionnelle simple

Exemple 2 :

```
Algorithme Moyenne
Variables : M : Réel
Début
        Ecrire("Votre moyenne : ")
        Lire(M)
        Si M >= 10 Alors
                Ecrire("Vous êtes admis")
        SiNon
                Ecrire("Vous n'êtes pas admis")
        FinSi
Fin
```

```
Algorithme Moyenne
Variables : M : Réel
Début
        Ecrire("Votre moyenne : ")
        Lire(M)
        Si M >= 10 Alors
                Ecrire("Vous êtes admis")
        FinSi
Fin
```

Structures conditionnelles

Structure conditionnelle simple

Equivalence entre les deux syntaxes:

Alternative:

```
Si condition Alors
        Traitement 1
SiNon
        Traitement 2
FinSI
```

Simple:

```
Si condition Alors
        Traitement 1
FinSi
Si Non(condition) Alors
        Traitement 2
FinSI
```

Structure conditionnelle simple

Exemple 2: Avec des structures conditionnelles simple

```
Algorithme Moyenne
Variables : M : Réel
Début
        Ecrire("Votre moyenne : ")
        Lire(M)
        Si M >= 10 Alors
                Ecrire("Vous êtes admis")
        SiNon
                Ecrire("Vous n'êtes pas admis")
        FinSi
Fin
```

```
Algorithme Moyenne
Variables : M : Réel
Début
        Ecrire("Votre moyenne : ")
        Lire(M)
        Si M >= 10 Alors
                Ecrire("Vous êtes admis")
        FinSi
        Si M < 10 Alors
                Ecrire("Vous n'êtes pas admis")
        FinSi
Fin
```

Structure conditionnelle simple

Exemple 3 :

Ecrire un algorithme qui affiche la valeur absolue d'un nombre, proposer deux solutions, une avec un SI alternative et une solution avec un SI simple.

Données d'entrés:

N : Réel

Données de sortie :

V : Réel

Traitement :

V = N si N >=0

V = -N si N < 0

Structures conditionnelles simples

Exemple 3 :

Avec SI alternative :

```
Algorithme Valeur Absolue
Variables : N, V : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        Si N >= 0 Alors
                V ← N
        SiNon
                V ← - N
        FinSi
        Ecrire("La valeur absolue est : ", V)
Fin
```

Avec deux SI simples :

```
Algorithme Valeur Absolue
Variables : N, V : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        Si N >= 0 Alors
                V ← N
        FinSi
        Si N < 0 Alors
                V ← -N
        FinSi
        Ecrire("La valeur absolue est : ", V)
Fin
```

Structure conditionnelle simple

Exemple 3 :

```
Algorithme Valeur Absolue
Variables : N, V : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        Si N >= 0 Alors
                V ← N
        SiNon
                V ← - N
        FinSi
        Ecrire("La valeur absolue est : ", V)
Fin
```

Avec une SI simple :

```
Algorithme Valeur Absolue
Variables : N, V : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        V ← N
        Si N < 0 Alors
                V ← -N
        FinSi
        Ecrire("La valeur absolue est : ", V)
Fin
```

Structures Conditionnelles Imbriquées

Structures conditionnelles simples

Exemple 4 :

Ecrire un algorithme qui détermine le signe d'un nombre entré par l'utilisateur.
On distingue deux cas : Strictement Positif ou Négatif

```
Algorithme Signe
Variables : N : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        Si N > 0 Alors
                Ecrire("Nombre strictement positif")
        SiNon
                Ecrire("Nombre négatif")
        FinSi
Fin
```

Structures conditionnelles imbriquées

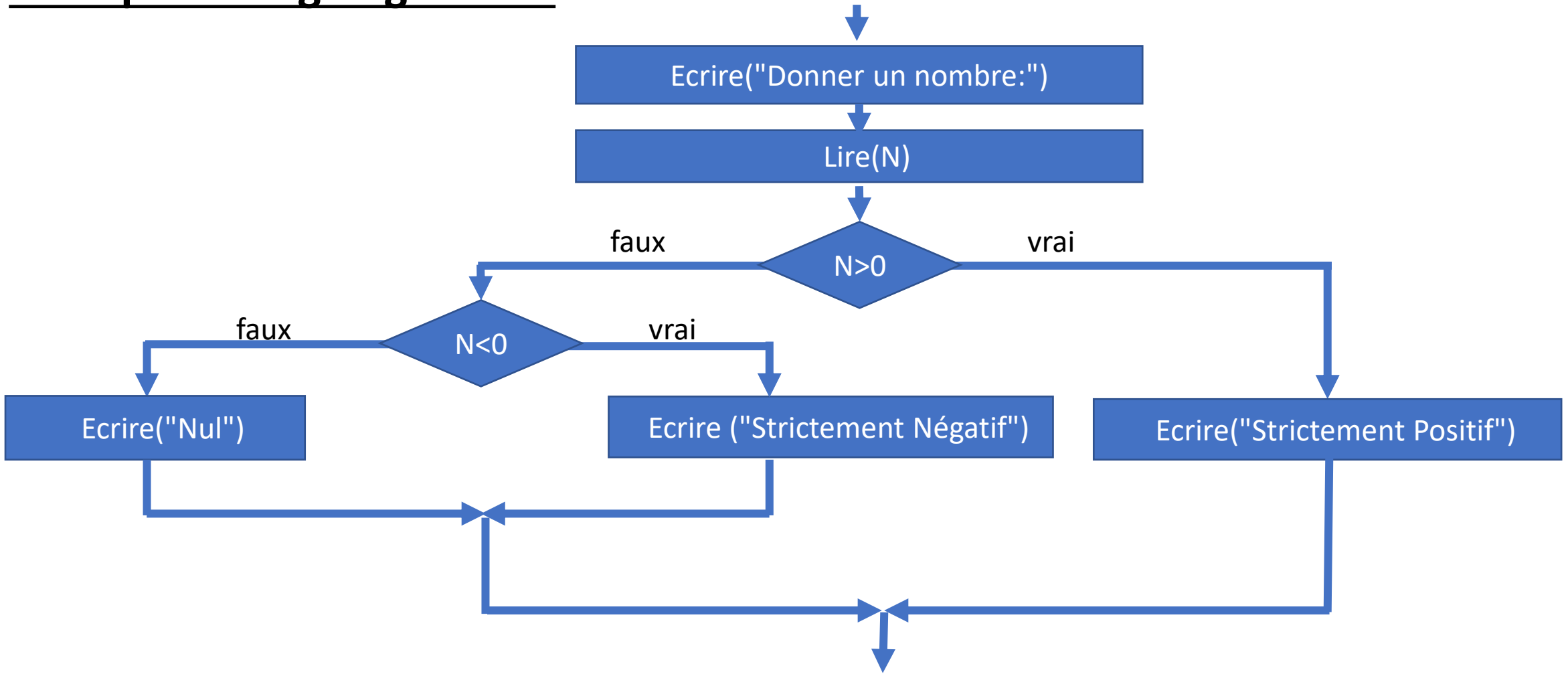
Exemple 5:

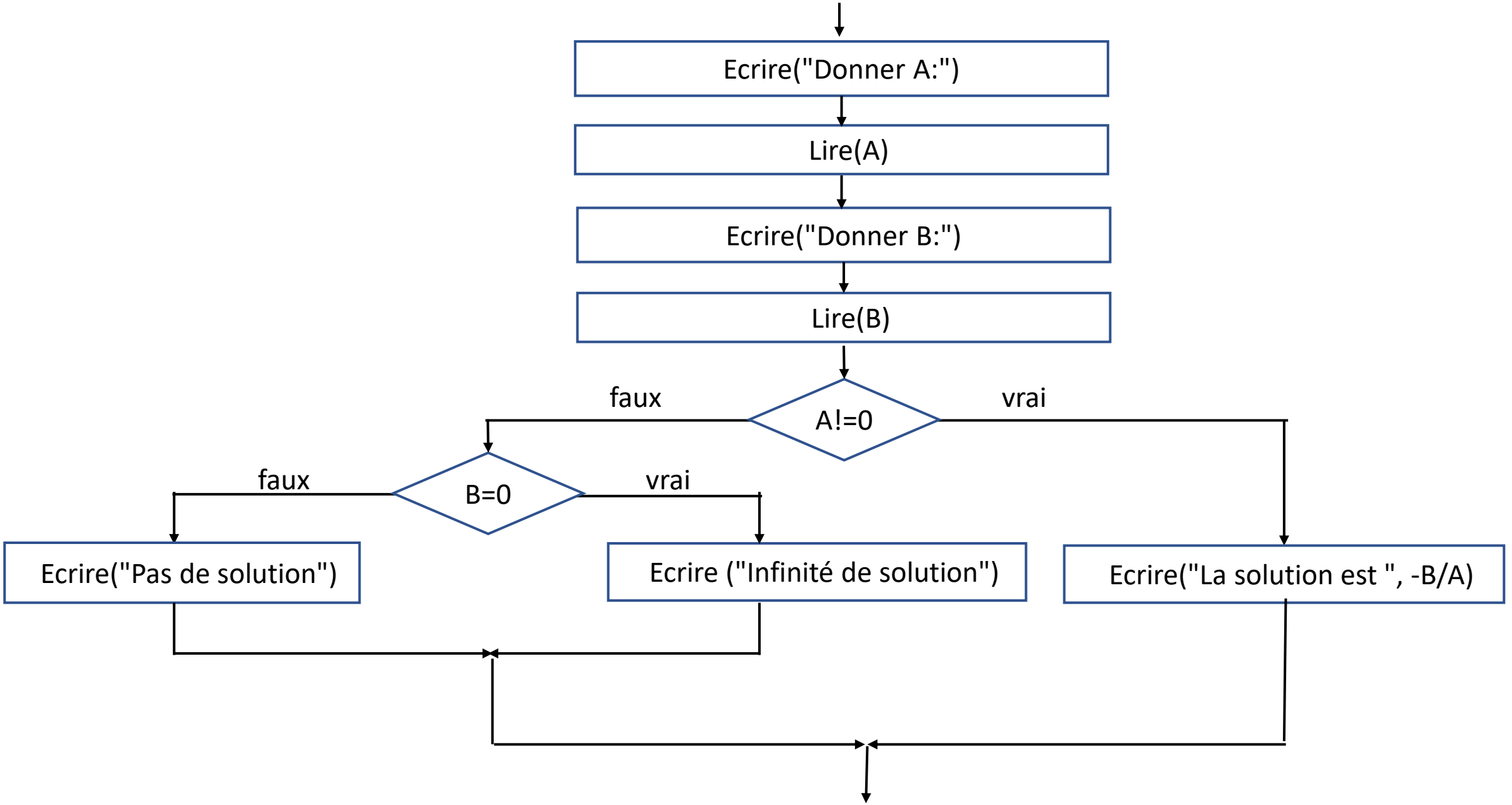
Améliorer l'algorithme du signe pour traiter les trois cas suivants :

- Strictement positif
- Strictement négatif
- nul

Structures conditionnelles imbriquées

Exemple 5: Organigramme




Ecrire("Donner A:")
Lire(A)
Ecrire("Donner B:")
Lire(B)
A!=0
faux
vrai
B=0
faux
vrai
Ecrire("Pas de solution")
Ecrire ("Infinité de solution")
Ecrire("La solution est ", -B/A)

Structures conditionnelles imbriquées

Exemple 5: Algorithme en Pseudo-code

```
Algorithme Signe
Variables : N : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        Si N > 0 Alors
                Ecrire("Nombre strictement positif")
        SiNon
                Si N < 0 Alors
                        Ecrire("Nombre négatif")
                SiNon
                        Ecrire("Nul")
                FinSi
        FinSi
Fin
```

Structures conditionnelles

Structure conditionnelle multiple

Définition:

La structure conditionnelle multiple (Si...Sinon Si...Sinon / If...Elif...Else) est utilisée pour évaluer plusieurs conditions successives. Elle permet de tester plusieurs situations l'une après l'autre jusqu'à ce qu'une soit vraie. Si aucune des conditions n'est vraie, le bloc else est exécuté.

Syntaxe :

```
Si condition1 Alors
        Traitement 1
SiNon Si condition2 Alors
        Traitement 2
SiNon Si condition 3 Alors
        Traitement 3
…

SiNon
        Traitement par défaut
FinSi
```

Structures conditionnelles

Structure conditionnelle multiple

```
Algorithme Signe
Variables : N : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        Si N > 0 Alors
                Ecrire("Nombre strictement positif")
        SiNon  Si N < 0 Alors
                Ecrire("Nombre négatif")
        SiNon
                Ecrire("Nul")
        FinSi
Fin
```

Structure conditionnelle multiple

Exemple 6 :

Écrivez un algorithme qui demande à l'utilisateur de saisir trois notes (entre 0 et 20) et qui calcule et affiche la moyenne de ces trois notes. Ensuite, le programme devrait afficher la mention associée à cette moyenne selon les critères suivants :

- Si la moyenne est inférieure à 10, affichez "Insuffisant".
- Si la moyenne est entre 10 inclus et 12 exclus, affichez "Passable".
- Si la moyenne est entre 12 inclus et 14 exclus, affichez "Assez bien".
- Si la moyenne est entre 14 inclus et 16 exclus, affichez "Bien".
- Si la moyenne est supérieure ou égale à 16, affichez "Très bien".

Structures conditionnelles

Structure conditionnelle multiple

Exemple 6 : Algorithme en pseudo-code

```
Algorithme Mention
Variables note1, note2, note3 : Réel
                mention : chaîne
Début
    Ecrire("Entrez la première note : ")
    Lire(note1)
    Ecrire("Entrez la deuxième note : ")
    Lire(note2)
    Ecrire("Entrez la troisième note : ")
    Lire(note3)
    moyenne = (note1 + note2 + note3) / 3
    Si moyenne < 10 Alors
       mention ← "Insuffisant"
    Sinon si moyenne < 12  Alors
       mention ← "Passable"
    Sinon si moyenne < 14 Alors
       mention ← "Assez bien"
    Sinon si moyenne < 16 Alors
       mention ← "Bien"
    Sinon
        mention ← "Très bien"
    FinSi

    Ecrire("La moyenne est de ", moyenne)
    Ecrire("la mention est : ", mention)
Fin
```

Structure conditionnelle multiple

Exemple 7 :

Écrivez un algorithme résout les équations du premier degré de la forme :

A X + B = 0

- **Les donnée d'entré :**

Les coefficients A et B

- **Les données de sortie :**

La solution de l'équation X

- **Traitement :**

S = -B/A Si A!= 0

S = $\mathbb{R}$ SI A =0 ET B=0

S = $\emptyset$ SI A =0 ET B!=0

Structure conditionnelle multiple

Exemple 7 : Algorithme

```
Algorithme Equation
Variables A,B,S: Réel
Début
    Ecrire("Donner A: ")
    Lire(A)
    Ecrire("Donner B: ")
    Lire(B)
    Si A ! = 0 Alors
        S ←-B/A
        Ecrire("La solution est ",  S)
    Sinon si B=0  Alors
        Ecrire("Infinité de solutions")
    Sinon
        Ecrire("Pas de solutions")
    FinSi
Fin
```

FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 4:
Les structures répétitives (les boucles)

Licence : Physique Chimie
Filières : Chimie

Pr: Youssef Ouassit

Plan Séance 4

Les structures de contrôles :

1. **Les structures conditionnelles (de choix)**
   a. L’instruction conditionnelle simple
   b. L'instruction conditionnelle alternative
   c. L'instruction conditionnelle sélective

2. **Les instructions d’itération ou de répétition (les boucles)**
   a) La boucle Pour … Faire
   b) La boucle Tant Que

Structures répétitives (Itératives ou Boucles)

Définition :

- Une boucle est une structure de contrôle qui permet de répéter une séquence d'instructions.

Pourquoi les utiliser ?

- Pour automatiser des tâches répétitives.
- Réduire la duplication de code.
- Traiter des structures de données comme les listes, tuples, chaînes de caractères,...

Structures répétitives (Itératives ou Boucles)

Deux types principaux :

Structure déterministe :

- La boucle: Pour

Répétez un nombre de fois déterminé
Exemple : Répéter un mouvement 13 fois

Structure indéterministe :

- La boucle : TantQue

Répétez tant que une condition est vrai
Exemples :

- **Répéter un mouvement tant qu'il n y a pas d'insuffisance musculaire**
- **Préparer le crème pâtissière**

La boucle POUR

La boucle Pour

Motivation :

Afficher le mot Informatique 5 fois

Solution naïve (sans boucle) :

```
Algorithme Répéter
Début
        Ecrire("Informatique")
        Ecrire("Informatique")
        Ecrire("Informatique")
        Ecrire("Informatique")
        Ecrire("Informatique")
Fin
```

Inconvénients :

- Duplication de code
- Difficile à maintenir
- Cette approche est peu flexible, non réutilisable, et difficile à modifier.

La boucle Pour

Motivation :

Si on veut calculer la somme des 6 premiers nombres entiers :

Solution naïve (sans boucle) :

```
Algorithme Somme
Variables S : Entier
Début
    S ← 1 + 2 + 3 + 4 + 5 + 6
    Ecrire("La somme est ", S)
Fin
```

Inconvénients :

- Si on veut calculer la somme des 100 premiers entiers, ou plus, il faudrait écrire une longue série d'additions.
- Duplication de code
- Difficile à maintenir
- Cette approche est peu flexible, non réutilisable, et difficile à modifier.

La boucle Pour

Motivation : La table de multiplication d'un entier :

Solution naïve (sans boucle) :

```
Algorithme Table Multiplication
Variables N : Entier
Début
        Ecrire("Donner un entier : ")
        Lire(N)
        Ecrire(N, "x 1 = ", 1*N)
        Ecrire(N, "x 2 = ", 2*N)
        Ecrire(N, "x 3 = ", 3*N)
        Ecrire(N, "x 4 = ", 4*N)
        Ecrire(N, "x 5 = ", 5*N)
        Ecrire(N, "x 6 = ", 6*N)
        Ecrire(N, "x 7 = ", 7*N)
        Ecrire(N, "x 8 = ", 8*N)
        Ecrire(N, "x 9 = ", 9*N)
        Ecrire(N, "x 10 = ", 10*N)
Fin
```

Inconvénients :

- Il y a beaucoup de répétitions dans le code.
- Si vous voulez afficher la table de multiplication pour un plus grand nombre d'itérations (par exemple, jusqu'à 20), vous devrez tout réécrire ou modifier manuellement chaque ligne.

La boucle Pour

Définition :

En informatique, la boucle Pour est une structure de contrôle qui permet de répéter l'exécution d'une séquence d'instructions.

Dans cette forme de boucle, une variable prend des valeurs successives sur un intervalle. Cette forme est souvent utilisée pour exploiter les données d'une collection indexée.

La boucle Pour

Syntaxe de base:

```
Pour compteur ← VI à VF Faire
    SEQUENCE
FinPour
```

- Une « boucle for » a deux parties : une entête qui spécifie la manière de faire l'itération, et un corps qui est exécuté à chaque itération.
- Le compteur i varie de valeur_initiale (VI) à valeur_finale (VF).
- Les instructions à l'intérieur de la boucle sont exécutées pour chaque valeur du compteur.

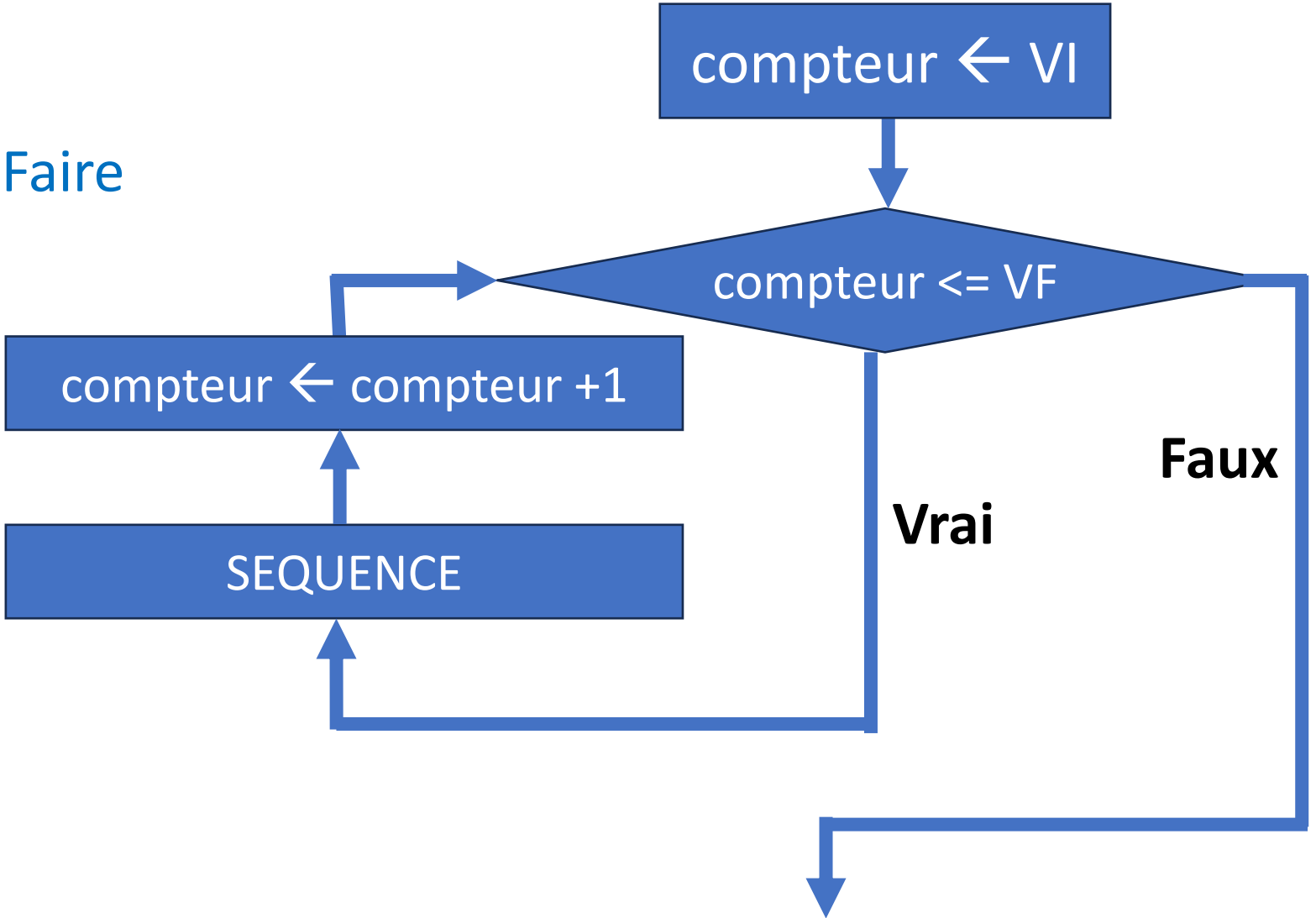
La boucle Pour

Organigramme:

Pour compteur ← VI à VF Faire

SEQUENCE

FinPour



Structures répétitives (Les boucles)

La boucle Pour

Exemple 1: Répéter une séquence 10 fois

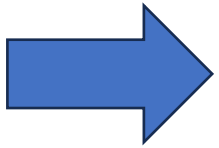
```
Pour i ← 1 à 10 Faire
        SEQUENCE
FinPour
```

Fonctionnement :
Pour chaque valeur i de l'intervalle [1, 10] exécuter la SEQUENCE

La boucle Pour

Exemple 2: Afficher le mot informatique 5 fois

```
Pour i ← 1 à 5 Faire
        Ecrire("Informatique")
FinPour
```



5 répétitions :

Informatique
Informatique
Informatique
Informatique
Informatique

La boucle Pour

Exemple 2: Afficher le mot informatique 500 fois

Il suffit de changer la valeur finale 5 par 500:

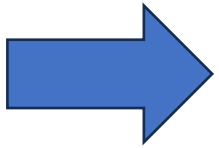
```
Pour i ← 1 à 500 Faire
        Ecrire("Informatique")
FinPour
```

La boucle Pour

Exemple 3:

Il est possible de choisir n'importe quel valeur initiale:

```
Pour i ← 3 à 7 Faire
        Ecrire("Bonjour")
FinPour
```

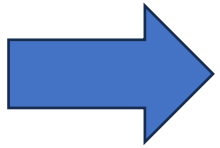


```
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
```

La boucle Pour

Exemple 4 : Affiches la suite des nombres de 1 à 10

```
Pour i ← 1 à 10 Faire
        Ecrire(i)
FinPour
```



```
1
2
3
4
5
6
7
8
9
10
```

La boucle Pour

Exemple 5 : Un algorithme qui affiche la suite des nombres de 1 à un entier N entré par l'utilisateur

```
Algorithme Nombres
Variables i, N : Entier
Début
        Ecrire(" Donner le nombre  : ")
        Lire(N)
        Pour  i ← 1 à N Faire
                Ecrire(i)
        FinPour
Fin
```

La boucle Pour

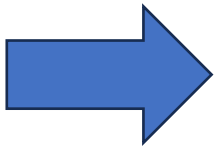
Exemple 6 : Un algorithme qui affiche les nombres multiple de 3 et de 7 entre 1 à un entier N entré par l'utilisateur

```
Algorithme Nombres
Variables i, N : Entier
Début
        Ecrire(" Donner le nombre  : ")
        Lire(N)
        Pour  i ← 1 à N Faire
                Si i mod 3=0 OU i mod 7=0 Alors
                        Ecrire(i)
                FinSi
        FinPour
Fin
```

La boucle Pour

Exemple 7 : Un algorithme qui affiche la table de multiplication d'un entier N.

```
Algorithme Table Multiplication
Variables i, N : Entier
Début
        Ecrire(" Donner le nombre  : ")
        Lire(N)
        Pour  i ← 1 à 10 Faire
                Ecrire(N, "x", i , "=", N*i)
        FinPour
Fin
```



```
Donner un nombre: 5
5x1=5
5x2=10
5x3=15
5x4=20
5x5=25
5x6=30
5x7=35
5x8=40
5x9=45
5x=10=50
```

La boucle Pour

Exemple 8 : Un algorithme qui calcule et affiche la somme des nombres de 1 à un entier N entré par l'utilisateur.

```
Algorithme Somme
Variables i, N, S : Entier
Début
        Ecrire(" Donner le nombre  : ")
        Lire(N)
        S ← 0
        Pour  i ← 1 à N Faire
                S ← S + i
        FinPour
        Ecrire("La somme est ", S)
Fin
```

La boucle Pour

Exemple 9 : Un algorithme qui calcule et affiche la somme des nombres de pairs dans l'intervalle 1 et N entré par l'utilisateur.

```
Algorithme Somme
Variables i, N, S : Entier
Début
        Ecrire(" Donner le nombre  : ")
        Lire(N)
        S ← 0
        Pour  i ← 1 à N Faire
                Si i mod 2 = 0 Alors
                        S ← S + i
                FinSi
        FinPour
        Ecrire("La somme est ", S)
Fin
```

Structures répétitives (Les boucles)

La boucle Pour

Syntaxe complet:

```
Pour compteur ← VI à VF  pas VP  Faire
    SEQUENCE
FinPour
```

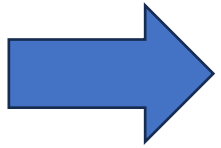
- VI : Valeur initiale
- VF : Valeur finale
- VP : Valeur du pas (Par défaut c'est 1)

La pas c'est la valeur avec laquelle le compteur de la boucle avance après chaque itération.

La boucle Pour

Exemple 10 : Compter avec un pas égal à 2

```
Algorithme Tester Pas
Variables i, N : Entier
Début
        Pour  i ← 1 à 10 pas 2    Faire
                Ecrire(i)
        FinPour
Fin
```



```
1
3
5
7
9
```

La boucle TantQue

Motivation :

Vous voulez écrire un programme qui demande à un utilisateur d'entrer un mot de passe. Tant que le mot de passe n'est pas correct, il doit redemander le mot de passe. Le nombre de tentatives est inconnu à l'avance.

Combien de fois l'utilisateur peut se tromper ?

Structures répétitives (Les boucles)

La boucle TantQue

Définition :

La boucle TantQue est une structure de contrôle en programmation qui permet d'exécuter un bloc de code répétitif tant qu'une condition donnée est vraie. Cela signifie que le bloc de code à l'intérieur de la boucle TantQue est exécuté de manière répétée tant que la condition spécifiée reste vraie. Une fois que la condition devient fausse, l'exécution de la boucle s'arrête et le programme passe à l'instruction suivante après la boucle.

Structures répétitives (Les boucles)

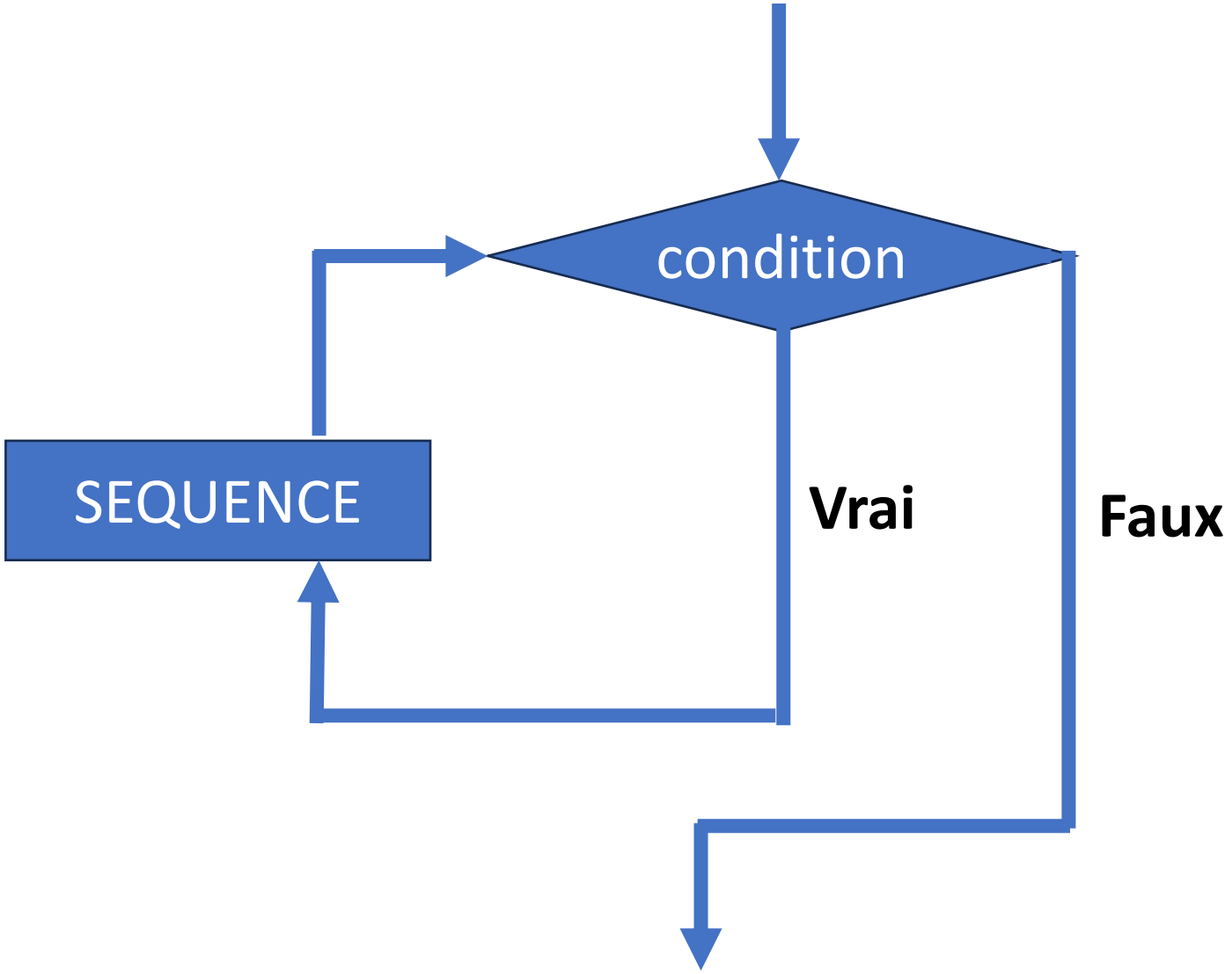
La boucle TantQue

Syntaxe :

TantQue condition Faire

SEQUENCE

FinTanque



Structures répétitives (Les boucles)

La boucle TantQue

Exemple 1 : Mot de passe

```
Algorithme Indemnités
Variables pwd : Entier
Début
        mot_de_passe_correct ← "abc123"
        Ecrire(" Votre mot de passe:")
        Lire(pwd)
        TantQue pwd != mot_de_passe_correct Faire
                Ecrire("Mot de passe incorrect, réessayez.")
                Lire(pwd)
         FinTantQue
         Ecrire("Accès autorisé.")
Fin
```

La boucle TantQue vérifie la validité du mot de passe.

La boucle TantQue

Exemple 2 :

Calculer les indemnités sociales à verser sachant que pour chaque enfant l'employer bénéfice d'une somme de 300 DH.

```
Algorithme Indemnités
Variables N : Entier
Début
        Ecrire(" Donner le nombre de vos enfants : ")
        Lire(N)
        S = N*300
        Ecrire("Le montant à verser est :", S)
Fin
```

La boucle TantQue

Exemple 2 :

Nous avons ajouté un contrôle de saisie

```
Algorithme Indemnités
Variables N : Entier
Début
        Ecrire(" Donner le nombre de vos enfants : ")
        Lire(N)
        TantQue N < 0 Faire
                Ecrire("Erreur, donnez une valeur valide : ")
                Lire(N)
        FinTantQue
        S = N*300
        Ecrire("Le montant à verser est :", S)
Fin
```

La boucle TantQue

Exemple 3 :

Écrire un algorithme qui calcule la somme des valeurs saisies par l'utilisateur. La saisie se termine lorsque l'utilisateur entre la valeur 0.

La boucle TantQue

Exemple 3 :

```
Algorithme Somme Plusieurs Valeurs
Variables N, S : Entier
Début
    S ← 0
    N ← -1
    TantQue N!=0  Faire
        Ecrire("Donner un autre nombre: ")
        Lire(N)
        S ← S + N
    FinTantQue
    Ecrire("La somme est ", S)
Fin
```

Equivalence entre les boucles

Une boucle TantQue équivalente à la boucle Pour:

```
Pour i ← 1 à 5 Faire
        Ecrire("Bonjour")
FinPour
```

```
i ← 1
TantQue i <= 5 Faire
        Ecrire("Bonjour")
        i ← i + 1
FinTantQue
```

FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 5:
Introduction à Python

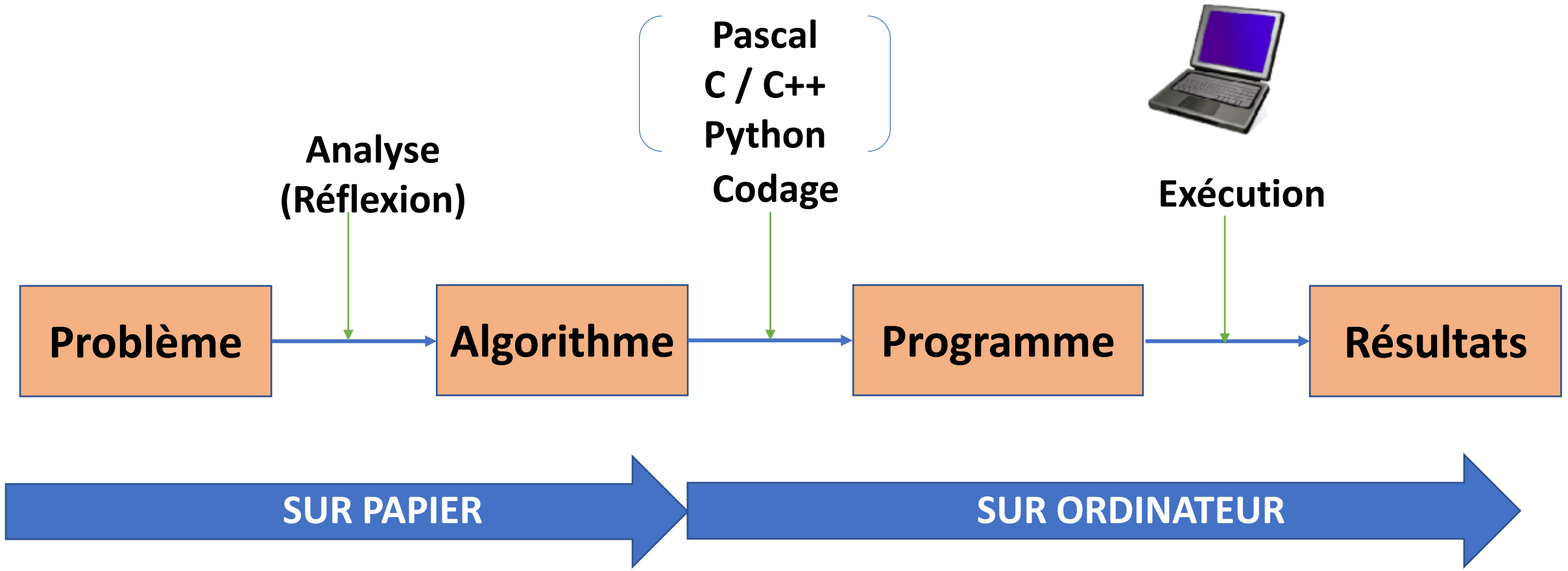
Licence : Physique Chimie
Filière : Chimie – S3

Pr: Youssef Ouassit

Plan Séance 1

1. **Introduction à Python**
    a. Historique
    b. Syntaxe de base
    c. Installation

Etapes de la programmation



Un peu d’histoire

Python est un langage de programmation **polyvalent** et **puissant**, largement utilisé dans divers domaines, notamment le développement web, l'analyse de données, l'intelligence artificielle, la science des données, l'automatisation, et bien plus encore.

Un peu d’histoire

Origines (1980s - 1990)

Créateur : Python a été créé par **Guido van Rossum**, un programmeur néerlandais, à la fin des années 1980. Van Rossum travaillait au Centrum Wiskunde & Informatica (CWI) aux Pays-Bas lorsqu'il a commencé à développer Python.

Nom : Le nom "Python" ne vient pas du serpent, mais plutôt du groupe comique britannique Monty Python. Van Rossum voulait un nom court, unique et un peu mystérieux.

Un peu d’histoire

La version 0.9.0 (1991)

Première version publique : Python 0.9.0 a été publié en février 1991. Cette version incluait déjà des fonctionnalités clés telles que les classes avec héritage, les exceptions, et les fonctions avec arguments nommés. Les structures de données intégrées comme les listes, les dictionnaires et les chaînes de caractères étaient également présentes.

Un peu d’histoire

La version 1.0 (1994)

Python 1.0 a été publié en janvier 1994. Cette version a introduit des fonctionnalités importantes comme les modules, les expressions **lambda**, les fonctions **map**, **filter**, et **reduce**.

Adoption : À cette époque, Python a commencé à gagner en popularité grâce à sa syntaxe simple et à sa capacité à s'intégrer facilement à d'autres langages et outils.

Un peu d'histoire

La version 2.x (2000)

Python 2.0 a été publié en octobre 2000. Cette version a introduit de nombreuses améliorations, notamment la collecte des ordures (**garbage collection**) pour gérer la mémoire, la **liste des compréhensions**, et le support complet de **Unicode**.

Python 2.x a continué à évoluer avec de nombreuses versions mineures, ajoutant des fonctionnalités et améliorant la performance. Python 2.7, publié en 2010, est devenu la version finale de la série Python 2.

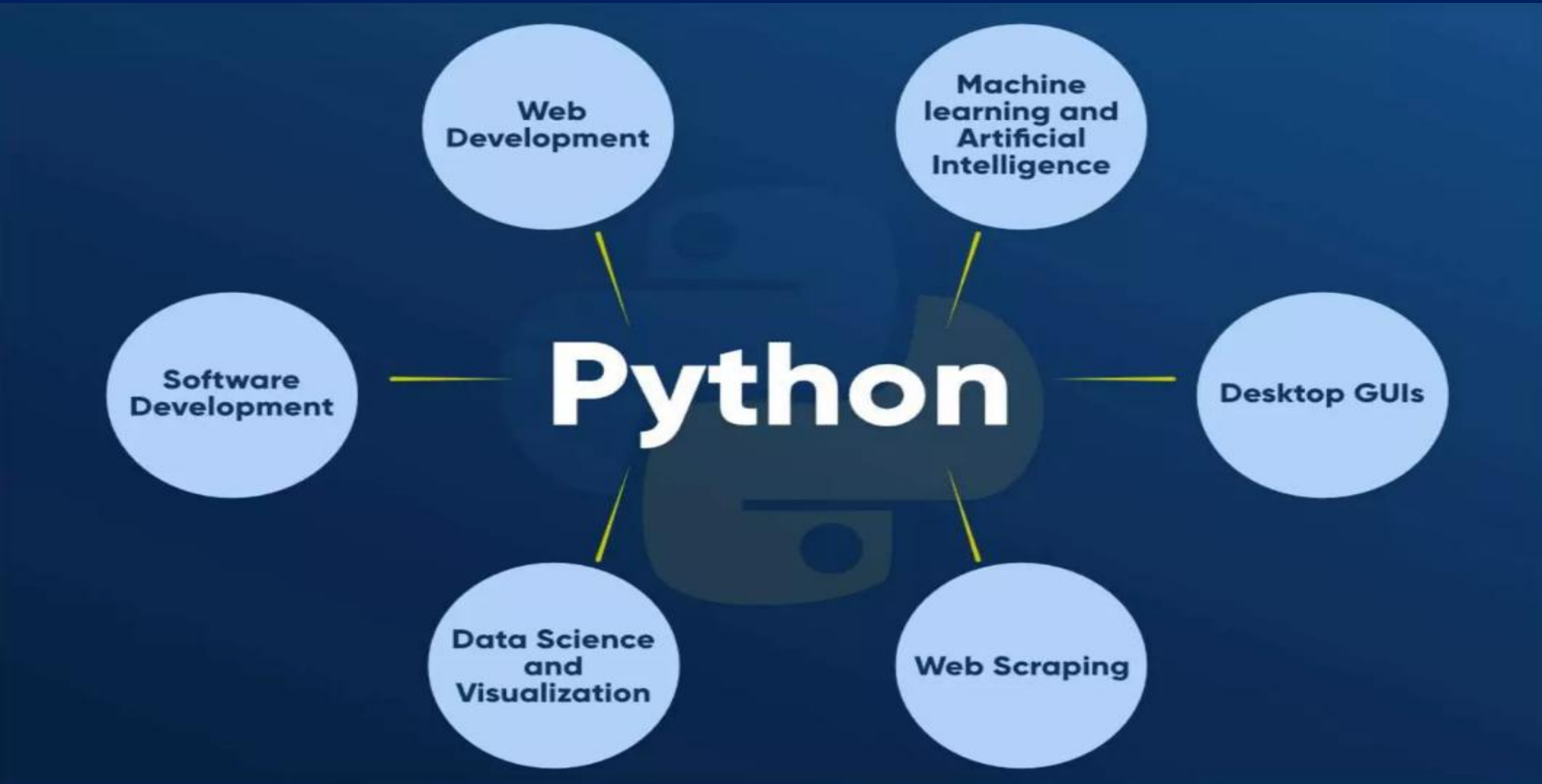
Un peu d’histoire

La version 3.x (2008 à présent)

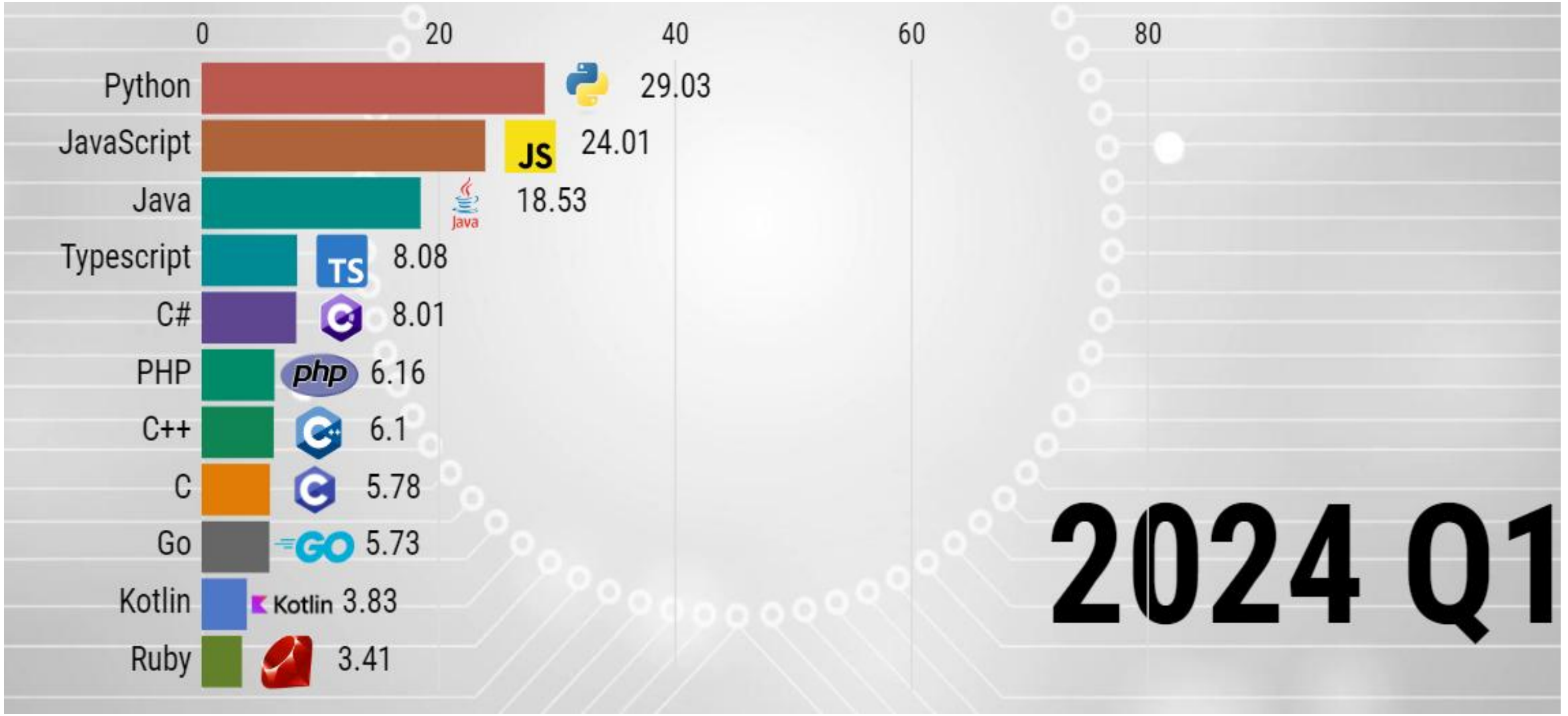
Python 3.0 a été publié en décembre 2008. Cette version a introduit des changements majeurs qui ont **rompu** la compatibilité avec Python 2.x, rendant la transition difficile pour certains projets.
L'objectif principal de Python 3 était de corriger les défauts fondamentaux du langage, en rendant la syntaxe plus cohérente et en supprimant les fonctionnalités obsolètes.

Adoption de Python 3 : Bien que la transition ait été lente au départ, Python 3 est maintenant la version principale utilisée par la majorité de la communauté Python. Le support officiel de Python 2 a pris fin le 1er janvier 2020, marquant la fin d'une ère.

Domaines d’applications



Un peu d’histoire



statisticsanddata.org

Pourquoi Python ?

Python est :

Portable : disponible sous toutes les plateformes (Unix, Windows, …)

Simple: avec une syntaxe qui privilège la lisibilité, libéré de celle de C/C++.

Riche: il incorpore plusieurs possibilités de langage: Programmation Impérative, POO

Pourquoi Python, ses caractéristiques ?

Il est :

- **dynamique** : il n'est pas nécessaire de déclarer le type d'une variable dans le source. Le type est associé lors de l'exécution du programme ;

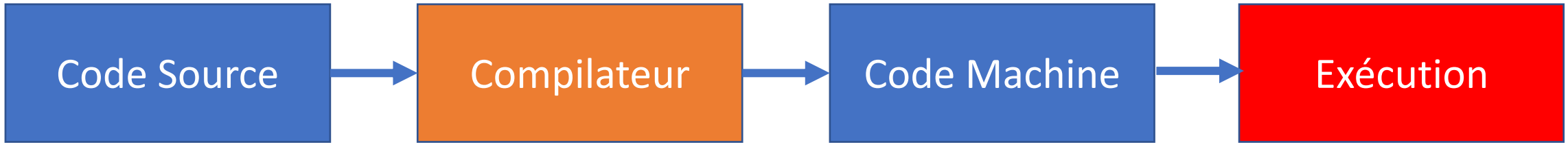
- **fortement typé** : les types sont toujours appliqués (un entier ne peut être considéré comme une chaîne sans conversion explicite, une variable possède un type lors de son affectation).

- **compilé/interprété** à la manière de Java. Le source est compilé en bytecode (pouvant être sauvegardé) puis exécuté sur une machine virtuelle.

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

Un langage compilé:



- Le code source est directement traduit en code machine natif spécifique à une plateforme.
- Exécution rapide car tout est compilé en amont.
- Nécessite une recompilation pour chaque type de machine.
- Exemple : C, C++

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

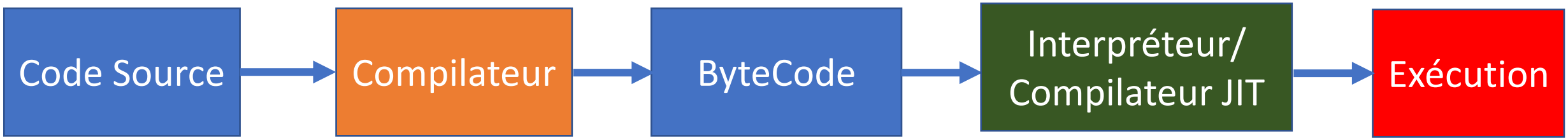
Un langage interprété:

- Le code source est interprété ligne par ligne à l'exécution.
- Facilité de développement et de debugging, mais exécution plus lente.
- Aucune compilation préalable en code machine.

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

Un langage Pseudo-Compilé:

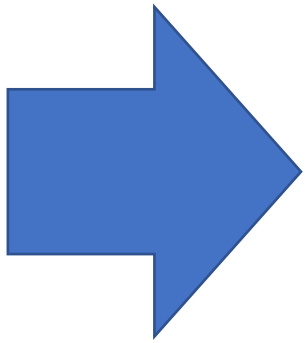


- Le code source est d'abord compilé en bytecode indépendant de la plateforme.
- Le bytecode est ensuite interprété ou compilé en code machine par la VM lors de l'exécution.
- Portabilité élevée car le bytecode peut être exécuté sur toute machine avec une VM.

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

Un langage Pseudo-Compilé:



Python est pseudo-compilé, il est irrémédiablement lent, mais… on peut lui associer des librairies intégrant des fonctions compilées qui, elles, sont très rapides.

Syntaxe de base

Les types de base :

int : Représente les nombres entiers, positifs ou négatifs, sans décimale.
Exemples : 5, -10, 0
Taille à partir de : 28 octets

float : Représente les nombres à virgule flottante, c'est-à-dire les nombres décimaux.
Exemples : 3.14, -0.001, 2.0
Taille à partir de : 24 octets

Syntaxe de base

Les types de base :

bool : Représente les valeurs de vérité, soit True soit False.
Exemples : True, False
Taille à partir de : 28 octets

complex : Représente les nombres complexes, qui ont une partie réelle et une partie imaginaire.
Exemples : 1 + 2j, 3 - 4j
Taille à partir de : 32 octets

Syntaxe de base

Les structure de données:

str : Représente une chaîne de caractères, utilisée pour stocker du texte.

list : Représente une collection ordonnée et modifiable d'éléments, qui peut contenir des éléments de différents types.

tuple : Similaire à une liste, mais immuable (les éléments ne peuvent pas être modifiés après la création).

range : Représente une séquence d'entiers, souvent utilisée dans les boucles.

dict : Représente une collection non ordonnée de paires clé-valeur. Les clés doivent être uniques et immuables, mais les valeurs peuvent être de n'importe quel type.

set : Représente une collection non ordonnée d'éléments uniques, sans doublons.

Syntaxe de base – Instructions de base

Pseudo code

Affectation simple:

A ← 5

Affectation multiple :

A ← 5

B ← 5

Affectation parallèle :

A ← 4

B ← 5

Python

Affectation simple:

En Python : A = 5

Affectation multiple :

A = B = 5

Affectation parallèle :

A, B = 4, 5

Syntaxe de base – Instructions de base

Affectation – Typage automatique

➢ `a = 1.2`

a est une variable, en interne elle a été automatiquement typée en flottant «float» parce qu'il y a un point décimal. a est l'identifiant de la variable (attention à ne pas utiliser le mots réservés comme identifiant), = est l'opérateur d'affectation

Calcul

➢ `a = 1.2`
➢ `d = a + 3`

d sera un réel contenant la valeur 4.2

Syntaxe de base – Instructions de base

Enchainement des instructions

La plus couramment utilisé

1 instruction = 1 ligne

```
#même valeur pour plusieurs variables
a = 1
b = 5
d = a + b
```

Autres possibilités

Peu utilisé

```
a = 1;  b = 5 ; d = a + b;
```

```
a = 1;
b = 5;
d = a + b;
```

Syntaxe de base – Instructions de base

Pseudo code

Syntaxe:

Ecrire(expressions)

Exemples :

```
Ecrire("Bonjour")
A ← 5
Ecrire("La valeur est : ", A)
```

Python

Syntaxe:

print(expressions)

Exemples :

```
print("Bonjour")
A = 5
print("La valeur est : ", A)
```

Concepts de base

Transtypage:

Conversion en numérique

```
a = "12 "      # a est de type chaîne caractère
b = int(a)     #b est de type int
c = float(a)   #c est de type float
```

N.B. Si la conversion n'est pas possible ex. float("toto"), Python renvoie une erreur

Conversion en chaîne de caractères

```
a = str(15) # a est de type chaîne et contient « 15 »
```

Syntaxe de base – Instructions de base

Pseudo code

Syntaxe:

Lire(variable)

Exemples (chaîne de caractères):

```
Ecrire("Votre nom : ")
Lire(nom)
```

Python

Syntaxe:

variable = input(message)

Exemples :

```
nom = input("Votre nom : ")
```

Syntaxe de base – Instructions de base

Pseudo code

Syntaxe:

Lire(variable)

Exemples (Entier):

```
Ecrire("Votre âge : ")
Lire(a)
```

Python

Syntaxe:

variable = input(message)

Exemples :

```
a = input("Votre âge : ")
a = int(a)
```

Ou :

```
a = int(input("Votre âge : "))
```

Syntaxe de base – Instructions de base

Pseudo code

Syntaxe:

Lire(variable)

Exemples (Réer):

```
Ecrire("Votre salaire : ")
Lire(s)
```

Python

Syntaxe:

```
variable = input(message)
```

Exemples :

```
s = input("Votre salaire : ")
s = float(s)
```

Ou :

```
s = float(input("Votre salaire : "))
```

Syntaxe de base – Instructions de base

Pseudo code **Python**

En Algorithmique	En Python
Ecrire("Donner un entier : ") Lire(a)	a = int(input("Donner un entier : "))
Ecrire("Donner un réel : ") Lire(b)	b = float(input("Donner un réel : "))
Ecrire("Donner une chaîne : ") Lire(c)	c = input("Donner une chaîne : ")

Concepts de base

Opérateurs Arithmétiques:

Opération	Algorithmique	Python
Addition	+	+
Soustraction	-	-
Multiplication	*	*
Division réel	/	/
Division entière	div	//
Reste de la division	mod	%
Exposant	^	**

Concepts de base

Opérateurs de comparaison:

Opération	Algorithmique	Python
Supérieur	>	>
Inférieur	<	<
Supérieur ou égal	>=	>=
Inférieur ou égal	<=	<=
Différent	!=	!=
Egal	=	==

Concepts de base

Opérateurs logiques:

Opération	Algorithmique	Python
Et	Et	And
Ou	Ou	Or
Non	Non	Not

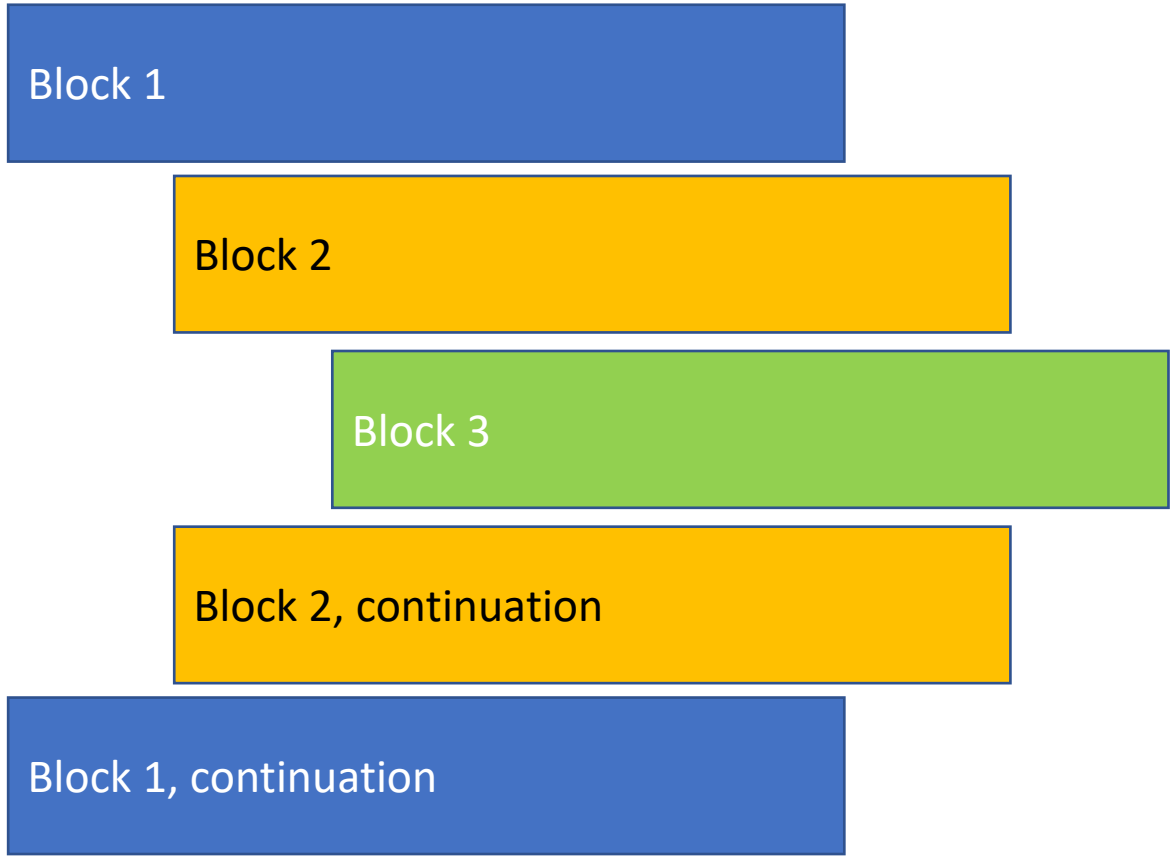
Concepts de base

La notion d’un block:

En Python, un bloc de code est un groupe de lignes de code qui sont exécutées ensemble en fonction d'une condition, d'une boucle ou d'une fonction. La notion de blocs en Python est étroitement liée à l'indentation, car c'est l'indentation qui délimite les blocs de code, contrairement à d'autres langages où des accolades {} ou des mots-clés spécifiques sont utilisés.

Concepts de base

La notion d’un block:



Indentation :

- Python utilise l'indentation (généralement 4 espaces) pour délimiter les blocs de code.
- Chaque ligne de code appartenant à un même bloc doit être indentée au même niveau.

Concepts de base

Structures conditionnelles

Les structures conditionnelles permettent d'exécuter du code uniquement si certaines conditions sont remplies.

- La structure if
- La structure if … else ….
- La structure if… elif …else

Concepts de base

Structures conditionnelles

Pseudo code

```
Si condition alors
        BI1
SiNon
        BI2
FinSi
```

Python

```
if condition :
        BI1
else :
        BI2
```

Concepts de base

Structures conditionnelles

Pseudo code

```
Si condition alors
        BI1
FinSi
```

Python

```
if condition :
        BI1
```

Concepts de base

Structures conditionnelles

Pseudo code

```
Si condition1 alors
        BI1
SiNon Si condition2 Alors
        BI2
SiNon Si condition3 Alors
        BI3
SiNon
        BI4
FinSi
```

Python

```
if condition1 :
        BI1
elif condition2:
        BI2
elif condition2:
        BI3
else:
        B4
```

Exemple :

Ecrire un programme qui détermine le signe d'un nombre :

- Strictement positif
- Strictement négatif
- nul

Concepts de base

Structures conditionnelles

```
Algorithme Signe
Variables : N : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        Si N > 0 Alors
                Ecrire("Nombre strictement positif")
        SiNon  Si N < 0 Alors
                Ecrire("Nombre négatif")
        SiNon
                Ecrire("Nul")
        FinSi
        Ecrire("Aurevoir")
Fin
```

Structures répétitives (Itératives ou Boucles)

Structure indéterministe :

- La boucle : while

Le nombre de répétitions dépend d'une condition

Structure déterministe :

- La boucle: for

Le nombre de répétitions est connu à l'avance

La boucle while

Définition :

La boucle while est une structure de contrôle en programmation qui permet d'exécuter un bloc de code répétitif tant qu'une condition donnée est vraie. Cela signifie que le bloc de code à l'intérieur de la boucle while est exécuté de manière répétée tant que la condition spécifiée reste vraie. Une fois que la condition devient fausse, l'exécution de la boucle s'arrête et le programme passe à l'instruction suivante après la boucle.

Structures répétitives

La boucle while

Pseudo code

```
TantQue condition Faire
    TRAITMENT
FinTantQue
```

Python

```
while condition :
    TRAITEMENT
```

La boucle while

Exemple 1 :

```
N=int(input("Donner le nombre de vos enfants : "))

while N<0:
        N = int(input("Erreur, donner une valeur >= 0 " ))

S = N*300
print("Le montant à verser est :", S)
```

La boucle while

Exemple 2 :

```
R= input("Voulez-vous un café ? O/N : ")
# contrôler la validité des données d'entrés
while R!='O' and R!='N' :
        R = input(« Réessayez .Voulez-vous un café ? O/N : " )

if R=='O' :
        print("Boisson servi !")
else :
        print("Aurevoir!")
```

La boucle for

Définition :

La boucle for est une structure de contrôle en programmation qui permet d'itérer une séquence de valeurs.

Les séquences en Python :

- Les chaînes de caractères
- Les listes
- Les tuples
- Les clés ou items ou valeurs des dictionnaires

Structures répétitives

La boucle while

Pseudo code

```
Pour i←1 à 10 Faire
    TRAITMENT
FinPour
```

Python

```
for i in range(1,11) :
    TRAITEMENT
```

La boucle for

Exemple 2:

```
for  i in range(1, 5):
     print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
```

La boucle for

Itérer les valeurs d'un intervalle [a, b]:

La fonction **range** permet de générer une séquence de nombre entiers :

La syntaxe est :

```
range( début,  fin,  pas)     # Le pas est optionnel et a la valeur 1 par défaut
```

La boucle for

Itérer les valeurs d'un intervalle [a, b]:

Exemples :

range(1, 6) va générer la séquence 1, 2, 3, 4, 5
range(1, 11) va générer la séquence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
range(-2, 3) va générer la séquence -2, -1, 0, 1, 2
range(1, 11, 2) va générer la séquence 1, 3, 5, 7, 9
range(1, 11, 3) va générer la séquence 1, 4, 7, 10
range(11) va générer la séquence 0,1, 2, 3, 4, 5, 6, 7, 8, 9, 10
range(11, 1) va générer la séquence VIDE
range(4, 1, -1) va générer la séquence 4, 3, 2
range(-4, -1) va générer -4, -3, -2

La boucle for

Syntaxe générale :

```
for i  in séquence :
    TRAITEMENT
```

Fonctionnement :
Pour chaque élément i de la séquence exécuter le TRAIEMENT

La boucle for

Itérer une chaîne de caractères :

```
for i in "Maroc" :
    print(i)
```

M
a
r
o
c

L'instruction break

L'instruction break permet d'interrompre l'exécution d'une boucle et de passer à la partie suivante du script.

Exemple : La boucle suivante s'arrête lorsque i atteint 5.

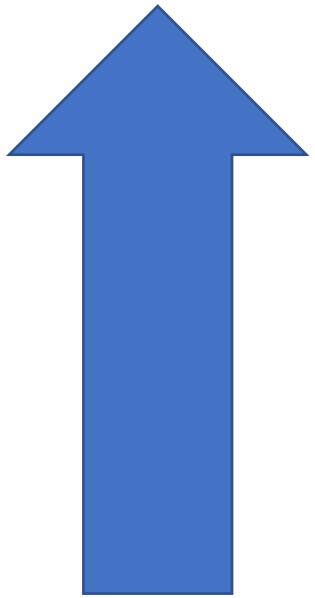
```
for i in range(10):
    if i == 5:
        print("Sortie de la boucle à i =", i)
        break
```

Priorités des opérateurs

La priorité des opérateurs en Python détermine l'ordre dans lequel les opérations sont effectuées dans une expression.

Les opérateurs
Les parenthèses : ( )
La puissance : **
Opérateurs unaire : -5 ou +6
Multiplication, division, modulo, division entière : * , / , % , //
Addition, soustraction : + , -
Opérateurs de comparaison : ==, !=, <, <=, >, >=
Opérateurs logique : and, or, not

Plus prioritaire



Priorités des opérateurs

Exemples :

(2 + 3) * 4	Donne : 20
2 + 3 ** 2	Donne : 11
10 + 2 * 3	Donne : 16
10 - 5 + 2	Donne : 7
5*3//3	Donne : 5
2 + 3 ** 2 * 4	Donne : 38
1 + 2 * (3 - 1) ** 2 // 5	Donne : 2
10 - 3 < 5	Donne : False
10 – (3 < 5)	Donne : 9
2 + 3 * 4 - 8 / 2**2	Donne : 12.0

Installation de Python – Méthode 1

Installation sur Windows:

Téléchargement : Rendez-vous sur le site officiel de Python www.python.org et téléchargez la dernière version stable.

Exécution de l'Installateur : Lancez l'installateur. Assurez-vous de cocher la case "Add Python to PATH" avant de cliquer sur "Install Now".

Installation de pyzo:

Téléchargement : Rendez-vous sur le site officiel de Python www.pyzo.org et téléchargez la dernière version.

Exécution de l'Installateur : Lancez l'installateur.

Installation de Python – Méthode 2

Accéder à :

https://colab.research.google.com/

Cliquez sur :

Nouveau Notebook

FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 6:
Les listes

Licence : Physique Chimie
Filière : Chimie – S3

Pr: Youssef Ouassit

Structures de données

Définition:

Les structures de données sont des manières d'organiser et de stocker des données dans un ordinateur afin de les utiliser efficacement. Voici un aperçu des principales structures de données :

- Les tableaux
- Les listes
- Les graphes
- Les arbres
- …

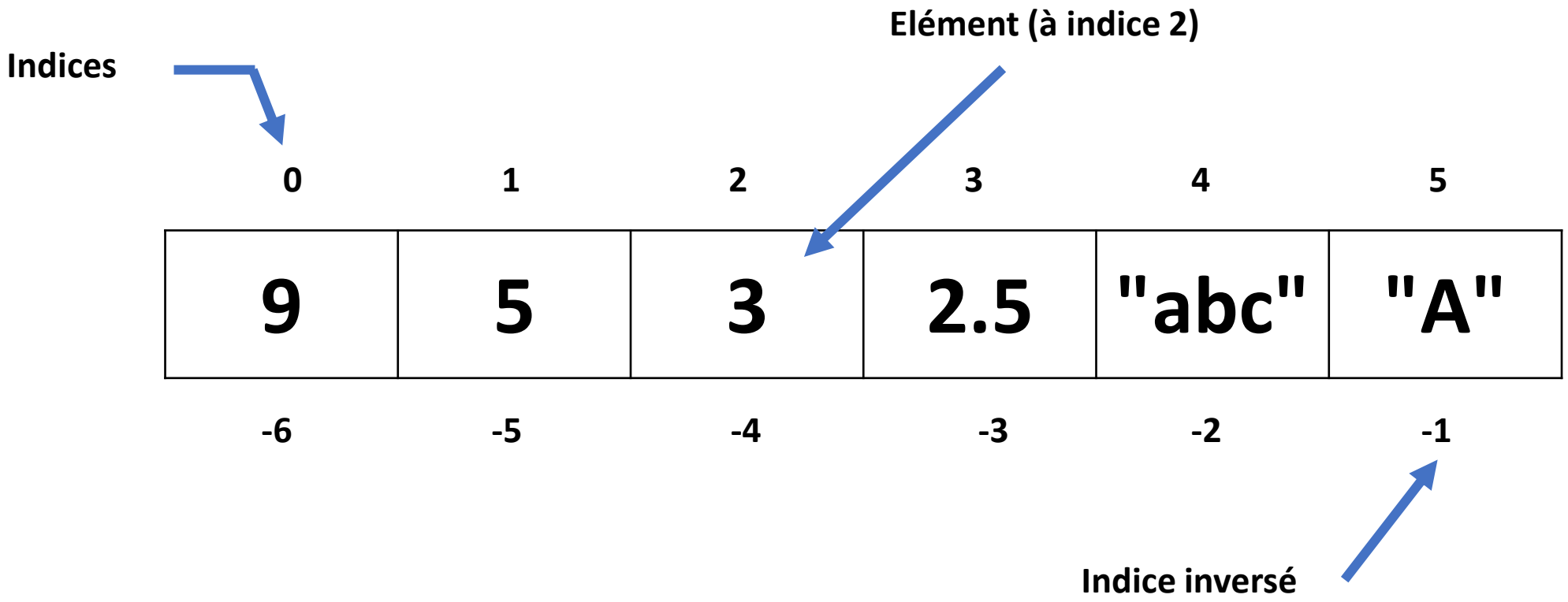
Structures de données

Comment choisir une structure de données:

Choisir la bonne **structure de données** est une étape cruciale dans la conception d'un programme ou d'un système, car cela impacte la **performance**, la **lisibilité**, et la **maintenabilité** du code. Pour faire ce choix, il est important de considérer plusieurs facteurs, tels que les **opérations** que vous devez effectuer, les **caractéristiques** des données, les **contraintes de performance**, et l'**usage prévu** de la structure.

Définition

Une liste est une séquence d'éléments linéaire, que nous pouvons représenter graphiquement sous la forme suivante :



Définition

En Python, les listes sont des structures de données très flexibles et largement utilisées. Voici un aperçu de leurs caractéristiques, de leur utilisation et de quelques opérations courantes.

Caractéristiques des Listes en Python

1. **Dynamique :** Les listes peuvent contenir un nombre variable d'éléments. Tu peux ajouter ou retirer des éléments facilement.
2. **Ordonnée :** Les éléments d'une liste conservent leur ordre d'insertion.
3. **Hétérogène :** Les listes peuvent contenir des éléments de différents types (nombres, chaînes, objets, etc.).
4. **Mutabilité :** Les listes sont mutables, ce qui signifie que tu peux modifier leurs éléments après leur création.

Déclaration

Pour déclarer une liste vide:

```
L = [ ]
```

Pour déclarer une liste initialisée:

```
A = [ 9, 16 , 2 , 7 , 5 , 30 ]
B = [ 9, 5 , 3 , 2.5 , "abc" , "A" ]
```

Les opérateurs sur les listes

Opérateur d'indexation:

Pour accéder à un élément on utilise l'opérateur d'indexation [ ]:

L = [9, 1, 3, 9, 5]

- print(L[2]) # affiche la valeur 3
- L[4] = 30 # permet de modifier la valeur de la case d'indice **4** par **30**.

Les opérateurs sur les listes

Opérateur d'indexation:

Pour accéder à un élément on utilise l'opérateur d'indexation []:

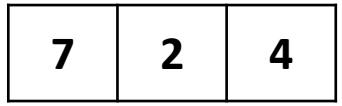
```
fruits = ["pomme", "banane", "cerise"]

print(fruits[0])  # 'pomme'
print(fruits[-1])  # 'cerise'
```

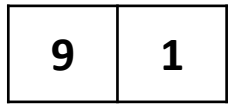
Les opérateurs sur les listes

Opérateur de concaténation :

A = [7 , 2 , 4]



B = [9 , 1]



C = A + B

Résultat :

C = [7 , 2 , 4 , 9 , 1]

Les opérateurs sur les listes

Opérateur d'appartenance :

Pour vérifier si une valeur appartient à une liste, on utilise l'opérateur **in** :

```
L = [ 5  ,  8 ,  7  , 2  , 4  ]

5 in L   # donne True
1 in L   # donne False
```

Les opérateurs sur les listes

Opérateur de duplication :

Nous pouvons multiplier une liste par un entier, les éléments de la liste seront multipliés:

```
L = [ 5  ,  8 ,  1 ]

A = L * 2
A = [ 5  ,  8 ,  1 ,5  ,  8 ,  1 ]
```

Les opérateurs sur les listes

Opérateur de slicing (les tranches) :

Il est possible d'extraire une partie d'une liste avec l'opérateur L[debut : fin : pas] :

```
L = [0, 1, 2, 3, 4, 5]
L1 = L[1:4]  # [1, 2, 3]
L2 = L[:3]   # [0, 1, 2]
L3 = L[::2]  # [0, 2, 4]
L4 = L[:]  # [0, 1, 2, 3, 4, 5]
```

Les fonctions sur les listes

Nombre des éléments:

La fonction **len** permet de calculer le nombre des éléments d'une liste:

```
L = [ 5  ,  8 ,  1 ]

n = len(L)
n = 3
```

Les fonctions sur les listes

Valeur maximale d'une liste:

La fonction **max** permet de calculer le maximum des éléments d'une liste:

```
L = [ 5  , 8 , 1 ]

n = max(L)
n = 8
```

Les fonctions sur les listes

Valeur minimale d'une liste:

La fonction **min** permet de calculer le minimum des éléments d'une liste:

```
L = [ 5  , 8 , 1 ]

n = min(L)
n = 1
```

Les fonctions sur les listes

Somme des éléments d'une liste:

La fonction **sum** permet de calculer la somme des éléments d'une liste:

```
L = [ 5  , 8 , 1 ]

n = sum(L)
n = 14
```

Les fonctions sur les listes

Trier les éléments d'une liste:

La fonction **sorted** permet de trier les éléments d'une liste:

```
L = [ 5  ,  8 ,  1 ]

A = sorted(L)
A = [1, 5, 8]
B = sorted(L, reverse=True)
B = [8, 5, 1]
```

Les méthodes des listes

Méthode `append()`

Description : Ajoute un élément à la fin de la liste.

Syntaxe : `liste.append(élément)`

Exemple :

```python
fruits = ["pomme", "banane"]
fruits.append("cerise")  # ["pomme", "banane", "cerise"]
```

Les méthodes des listes

Méthode `extend()`

Description : Ajoute les éléments d'une autre liste à la fin de la liste courante.

Syntaxe : `liste.extend(autre_liste)`

Exemple :

```python
liste1 = [1, 2, 3]
liste2 = [4, 5, 6]
liste1.extend(liste2)  # [1, 2, 3, 4, 5, 6]
```

Les méthodes des listes

Méthode `insert()`

Description : Insère un élément à une position spécifique de la liste.

Syntaxe : `liste.insert(index, élément)`

Exemple :

```python
fruits = ["pomme", "cerise"]
fruits.insert(1, "banane")  # ["pomme", "banane", "cerise"]
```

Les méthodes des listes

Méthode `remove()`

Description : Supprime la première occurrence d'un élément dans la liste.

Syntaxe : `liste.remove(élément)`

Exemple :

```python
fruits = ["pomme", "banane", "cerise", "banane"]
fruits.remove("banane")  # ["pomme", "cerise", "banane"]
```

Les méthodes des listes

Méthode `pop()`

Description : Supprime et retourne l'élément à une position donnée (par défaut, le dernier).

Syntaxe : `liste.pop([index])`

Exemple :

```python
fruits = ["pomme", "banane", "cerise"]
dernier_fruit = fruits.pop()  # "cerise"
```

Les méthodes des listes

Méthode `clear()`

Description : Supprime tous les éléments de la liste.

Syntaxe : `liste.clear()`

Exemple :

```python
fruits = ["pomme", "banane", "cerise"]
fruits.clear()  # []
```

Les méthodes des listes

Méthode `index()`

Description : Retourne l'index de la première occurrence d'un élément dans la liste.

Syntaxe : `liste.index(élément)`

Exemple :

```python
fruits = ["pomme", "banane", "cerise"]
index_banane = fruits.index("banane")  # 1
```

Les méthodes des listes

Méthode `count()`

Description : Compte le nombre de fois qu'un élément apparaît dans la liste.

Syntaxe : `liste.count(élément)`

Exemple :

```python
fruits = ["pomme", "banane", "cerise", "banane"]
nombre_banane = fruits.count("banane")  # 2
```

Les méthodes des listes

Méthode `sort()`

Description : Trie les éléments de la liste en place.

Syntaxe : `liste.sort([key=None, reverse=False])`

Exemple :

python Copier le code

```python
chiffres = [3, 1, 4, 1, 5, 9]
chiffres.sort()  # [1, 1, 3, 4, 5, 9]
```

Les méthodes des listes

Méthode `reverse()`

Description : Inverse l'ordre des éléments de la liste en place.

Syntaxe : `liste.reverse()`

Exemple :

```python
chiffres = [1, 2, 3]
chiffres.reverse()  # [3, 2, 1]
```

Les méthodes des listes

Méthode `copy()`

Description : Retourne une copie superficielle de la liste.

Syntaxe : `liste.copy()`

Exemple :

python Copier le code

```python
fruits = ["pomme", "banane", "cerise"]
fruits_copie = fruits.copy()  # ["pomme", "banane", "cerise"]
```

Parcourir une liste

Parcours par indice :

```
for i in range(len(L)):
    print(L[i])
```

Parcours par valeur :

```
for a in L:
    print(a)
```

Les tuples

Définition d’un tuple

Un tuple est une séquence ordonnée d'éléments, qui peuvent être de types différents (par exemple, entiers, chaînes de caractères, listes, etc.).

Contrairement aux listes, les tuples sont immuables, ce qui signifie que leurs éléments ne peuvent pas être modifiés après leur création. Une fois qu'un tuple est défini, il ne peut pas être modifié, étendu ou réduit.

Les tuples

Déclaration

```
# Tuple vide
tuple_vide = ()

# Tuple avec des éléments
mon_tuple = (1, 2, 3, 4, 5)

# Tuple avec des éléments sans parenthèses
mon_tuple = 1, 2, 3, 4, 5

# Tuple avec des types différents
mon_tuple_melangee = (1, "Python", 3.14, True)
```

Les tuples

Déclaration

```
# Tuple vide avec tuple()
mon_tuple_vide = tuple()

# Tuple à partir d'une chaîne de caractères
mon_tuple_chaine = tuple("Python") # ('P','y','t','h','o','n’)

# Tuple à partir d’une liste
mon_tuple_tuple = tuple([1, 2, 3])  # (1, 2, 3)

# Tuple avec un seul élément
mon_tuple_vide = (1, )
```

Les tuples

Opérateurs de base

Opérateur d'Appartenance **in**

```
fruits = ("pomme", "banane", "cerise" )
print("pomme" in fruits)  # True
print("orange" in fruits)  # False
```

Opérateur de concaténation **+**

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
tuple3 = tuple1 + tuple2  # (1, 2, 3, 4, 5, 6)
```

Les tuples

Opérateurs de base

Opérateur de répétition *

```
tuple = [1, 2, 3]
resultat = tuple * 3  # (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Opérateur d’indexation []

```
fruits = ("pomme", "banane", "cerise« )
print(fruits[0])  # 'pomme'
print(fruits[-1])  # 'cerise'
```

Les tuples

Opérateurs de base

Opérateur de Slicing **[debut : fin : pas]**

```
chiffres = (0, 1, 2, 3, 4, 5)
sous_tuple = chiffres[1:4]  # (1, 2, 3)
sous_tuple2 = chiffres[:3]   # (0, 1, 2)
sous_tuple3 = chiffres[::2]  # (0, 2, 4)
sous_tuple4 = chiffres[:]  # (0, 1, 2, 3, 4, 5)
```

Les tuples

Les méthodes

Méthode : `count`

Description : Retourne le nombre d'occurrences d'un élément spécifique dans le tuple.

Syntaxe : `tuple.count(element)`

Exemple :

```python
mon_tuple = (1, 2, 2, 3)
print(mon_tuple.count(2))  # Affiche : 2
```

Les tuples

Les méthodes

Méthode : `index`

Description : Retourne l'index de la première occurrence d'un élément spécifique dans le tuple. Lève une exception `ValueError` si l'élément n'est pas trouvé.

Syntaxe : `tuple.index(element, start=0, end=len(tuple))`

Exemple :

```python
mon_tuple = (1, 2, 3)
print(mon_tuple.index(2))  # Affiche : 1
```

Les tuples

Les fonctions

Les fonctions sur les listes sont aussi applicable sur les tuples :

- len
- min
- max
- sum
- sorted
- reversed

FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 7:
Les chaînes

Licence : Physique Chimie
Filière : Chimie – S3

Pr: Youssef Ouassit

Les str

Définition d’une chaîne de caractères

Le type **str** en Python est une séquence d'unités de caractères Unicode.
Les chaînes de caractères sont utilisées pour stocker et manipuler du texte.

Les chaînes de caractères sont définies en utilisant des guillemets simples ('), des guillemets doubles ("), ou des triples guillemets (''' ou """) pour les chaînes multilignes.

Les chaînes de caractères

Déclaration

```
# Définir une chaîne

chaine1 = 'Bonjour'

chaine2 = "le monde"

chaine3 = '''Ceci est une chaîne qui s'étend sur
plusieurs lignes.'''
```

Les chaînes de caractères

Opérateurs de base

Opérateur d'Appartenance **in**

```
ch = "pomme"
print("po" in ch)  # True
print("ora" in ch)  # False
```

Opérateur de concaténation **+**

```
ch1 = "Bonjour "
ch2 = "Ahmed"
ch3 = ch1 + ch2  # "Bonjour Ahmed"
```

Les chaînes de caractères

Opérateurs de base

Opérateur de répétition *

```
ch = "bon"
resultat = ch * 2  # "bonbon"
```

Opérateur d'indexation []

```
ch = "pomme"
print(ch[0])  # 'p'
print(ch[-1])  # 'e'
```

Les chaînes de caractères

Opérateurs de base

Opérateur de Slicing **[debut : fin : pas]**

```
ch = "Bonjour"
sous_chaine = ch[:3]  # Bon
sous_chaine2 = ch[1:3]   # on
sous_chaine3 = ch[::2]  # Bnor
sous_chaine4 = ch[:]  # Bonjour
```

Les chaînes de caractères

Les méthodes

Méthode : `upper`

Description : Retourne une nouvelle chaîne où tous les caractères sont convertis en majuscules.

Syntaxe : `str.upper()`

Exemple :

```python
texte = "bonjour"
print(texte.upper())  # Affiche : BONJOUR
```

Les chaînes de caractères

Les méthodes

Méthode : `lower`

Description : Retourne une nouvelle chaîne où tous les caractères sont convertis en minuscules.

Syntaxe : `str.lower()`

Exemple :

```python
texte = "BONJOUR"
print(texte.lower())  # Affiche : bonjour
```

Les chaînes de caractères

Les méthodes

Méthode : `replace`

Description : Remplace toutes les occurrences d'une sous-chaîne par une autre dans la chaîne d'origine.

Syntaxe : `str.replace(old, new[, count])`

Exemple :

python

```python
texte = "bonjour monde"
print(texte.replace("monde", "tout le monde"))  # Affiche : bonjour tout le monde
```

Les chaînes de caractères

Les méthodes

Méthode : `split`

Description : Divise la chaîne en une liste de sous-chaînes en utilisant le séparateur spécifié. Si aucun séparateur n'est fourni, les espaces sont utilisés par défaut.

Syntaxe : `str.split([separator[, maxsplit]])`

Exemple :

```python
texte = "bonjour tout le monde"
print(texte.split())  # Affiche : ['bonjour', 'tout', 'le', 'monde']
```

Les chaînes de caractères

Les méthodes

Méthode : `` `find` ``

Description : Retourne l'index de la première occurrence de la sous-chaîne spécifiée dans la chaîne. Retourne `` `-1` `` si la sous-chaîne n'est pas trouvée.

Syntaxe : `` `str.find(sub[, start[, end]])` ``

Exemple :

python Copier le code

```python
texte = "bonjour"
print(texte.find("jour"))  # Affiche : 3
```

Les chaînes de caractères

Les méthodes

Méthode : `count`

Description : Retourne le nombre de fois qu'une sous-chaîne apparaît dans la chaîne.

Syntaxe : `str.count(sub[, start[, end]])`

Exemple :

```python
texte = "bonjour tout le monde"
print(texte.count("o"))  # Affiche : 3
```

Les chaînes de caractères

Les fonctions

On retrouve les même fonctions déjà vu :

- len
- max
- min
- …

Des fonctions spécifiques aux chaînes de caractères:

- **ord** : Retourne le code Unicode d'un caractère.
- **chr** : Retourne le caractère correspondant à un code Unicode
- **eval :** Évalue une chaîne comme une expression Python.
- **input** : Demande une entrée utilisateur.