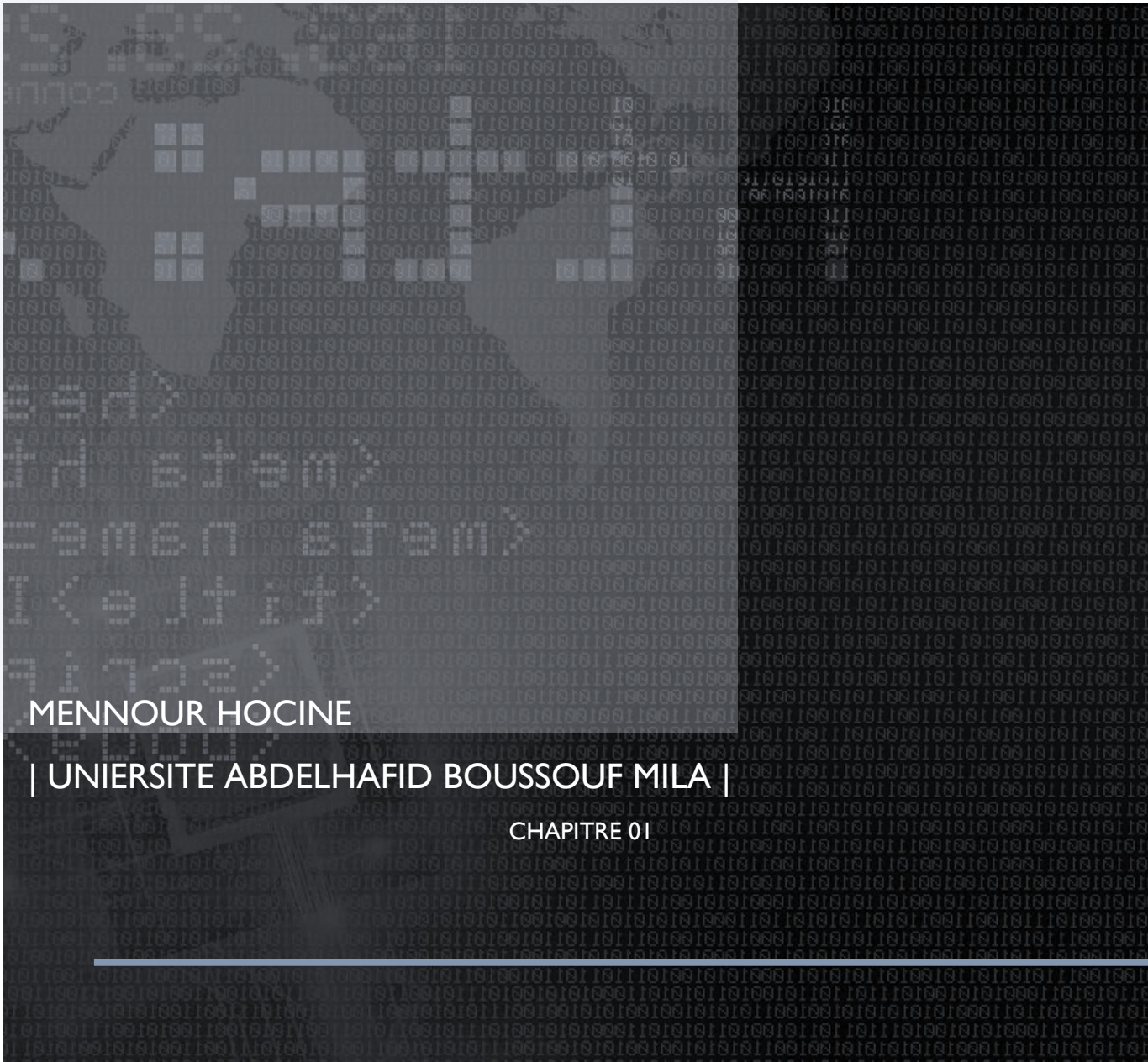
PROGRAMMATION INFORMATIQUE

APPLIQUEE AUX SCIENCES ET TECHNOLOGIES



MENNOUR HOCINE

| UNIERSITE ABDELHAFID BOUSSOUF MILA |

CHAPITRE 01

CHAPITRE I : INTRODUCTION A LA PROGRAMMATION SCIENTIFIQUE

1- **L'Invention de la Méthode (L'Algorithme)**
   - Le problème initial
   - La Naissance du Raisonnement Mécanique (IXe siècle)

2- **La Mécanisation du Calcul (La Machine)**
   2.1 Le Modèle Universel de Turing (1936)
   2.2 De la Théorie à la Réalité Physique
   2.3 La Barrière de la Communication

3- **La Communication (Les Langages de Programmation)**
   3.1 Le Problème du Fossé Sémantique
   3.2 Les Trois Structures Universelles

4- **Introduction aux langages Python et R**
   4.1 Python : Le langage universel
   4.2 R : Le Langage des Statisticiens
   4.3 Qu'est-ce qu'une Bibliothèque ?
   4.4 Les Environnements de Développement (IDE)
      4.4.1 RStudio
      4.4.2 Anaconda et Jupyter
      4.4.3 VS Code

5- **Variables et Fonctions**
   5.1 Les Variables (Le stockage des données)
   5.2 Les Fonctions (L'automatisation des calculs)

6- **Structures de Contrôle : Logique et Répétition**
   6.1 Les Structures Conditionnelles (La prise de décision)
   6.2 Les Boucles (L'Automatisation)

7- **Structures de données fondamentales**
   7.1 Les Séquences : Listes et Vecteurs
   7.2 Les Mappages : Dictionnaires et Listes Nommées
   7.3 Les Ensembles (Sets) et l'Unicité

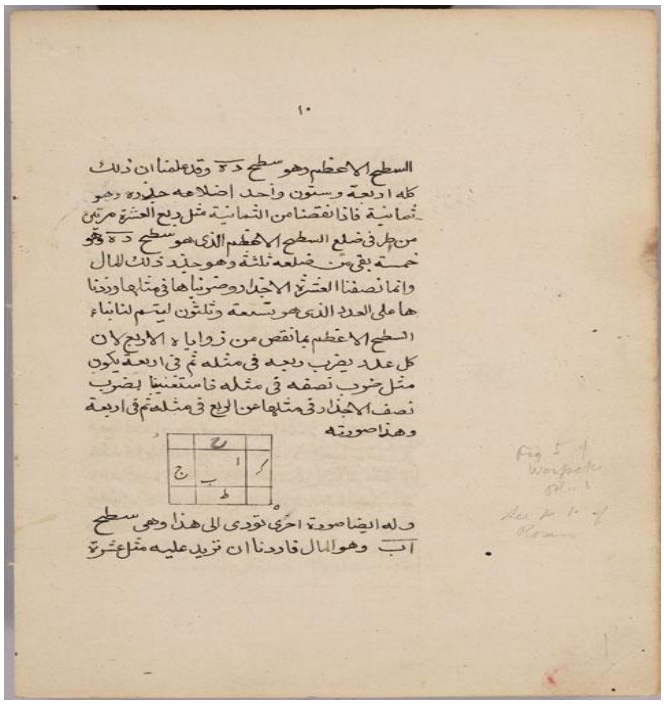
1- L'INVENTION DE LA METHODE (L'ALGORITHME)

Le problème initial

Comment résoudre un problème complexe sans avoir besoin d'une intuition géniale à chaque fois ?

La Naissance du Raisonnement Mécanique (IXe siècle)

L'histoire de la programmation ne commence pas avec l'ordinateur, mais avec une rupture intellectuelle majeure. Jusqu'au IXe siècle, les mathématiques étaient un art difficile : pour résoudre une équation, il fallait recourir à des constructions géométriques et trouver une astuce spécifique à chaque cas.



C'est à Bagdad, vers l'an 825, que le mathématicien **Al-Khwārizmī** bouleverse cette approche. Dans son traité ***« Kitāb al-Mukhtasar fī Hisāb al-Jabr wa-al-Muqābala »***, il formalise une idée révolutionnaire : il est possible de résoudre des problèmes complexes en suivant une suite d'étapes simples, logiques et disciplinées. Il n'est plus nécessaire de comprendre la géométrie sous-jacente ; il suffit d'appliquer correctement la méthode.

Cette idée marque la naissance de ce que l'on appelle aujourd'hui **l'algorithme**.

Définition : Un algorithme est une suite finie, ordonnée et non ambiguë d'instructions qui permet, à partir de données d'entrée, d'obtenir un résultat de sortie pour résoudre un problème donné.

Pour être valide scientifiquement, un algorithme doit respecter trois critères fondamentaux :

- **Finitude** : la procédure doit s'arrêter après un nombre fini d'étapes (absence de boucle infinie).
- **Déterminisme** : pour une même entrée, l'algorithme doit toujours produire la même sortie.
- **Non-ambiguïté** : chaque étape doit être définie de manière claire et précise, sans interprétation possible.

Pour illustrer cette logique, prenons l'exemple même utilisé par Al-Khwārizmī : la résolution d'équations de la forme :

$$x^2 + bx = c$$

Par exemple :

$$x^2 + 10x = 39$$

Al-Khwārizmī propose un **algorithme**, c'est-à-dire une procédure étape par étape.

Début :

1. Diviser le coefficient de (x) par deux :
   (var1 ←10 / 2 = 5)
2. Mettre ce résultat au carré :
   (var2 ← 5 x 5 = 25)
3. Ajouter la constante (c) :
   (var3 ← 25 + 39 = 64)
4. Prendre la racine carrée :
   (var4 ← $\sqrt{64}$ = 8)
5. Soustraire le résultat de l'étape 1 :
   (Résultat ← 8 - 5 = 3)

Cette procédure illustre la naissance du **logiciel** : une suite d'opérations élémentaires (addition, division, racine carrée…etc) qui transforme des données **d'entrée** en un **résultat**, sans aucune ambiguïté.

À ce stade de l'histoire, la méthode est conceptuellement parfaite. Cependant, elle présente une faiblesse majeure : **l'exécutant**.

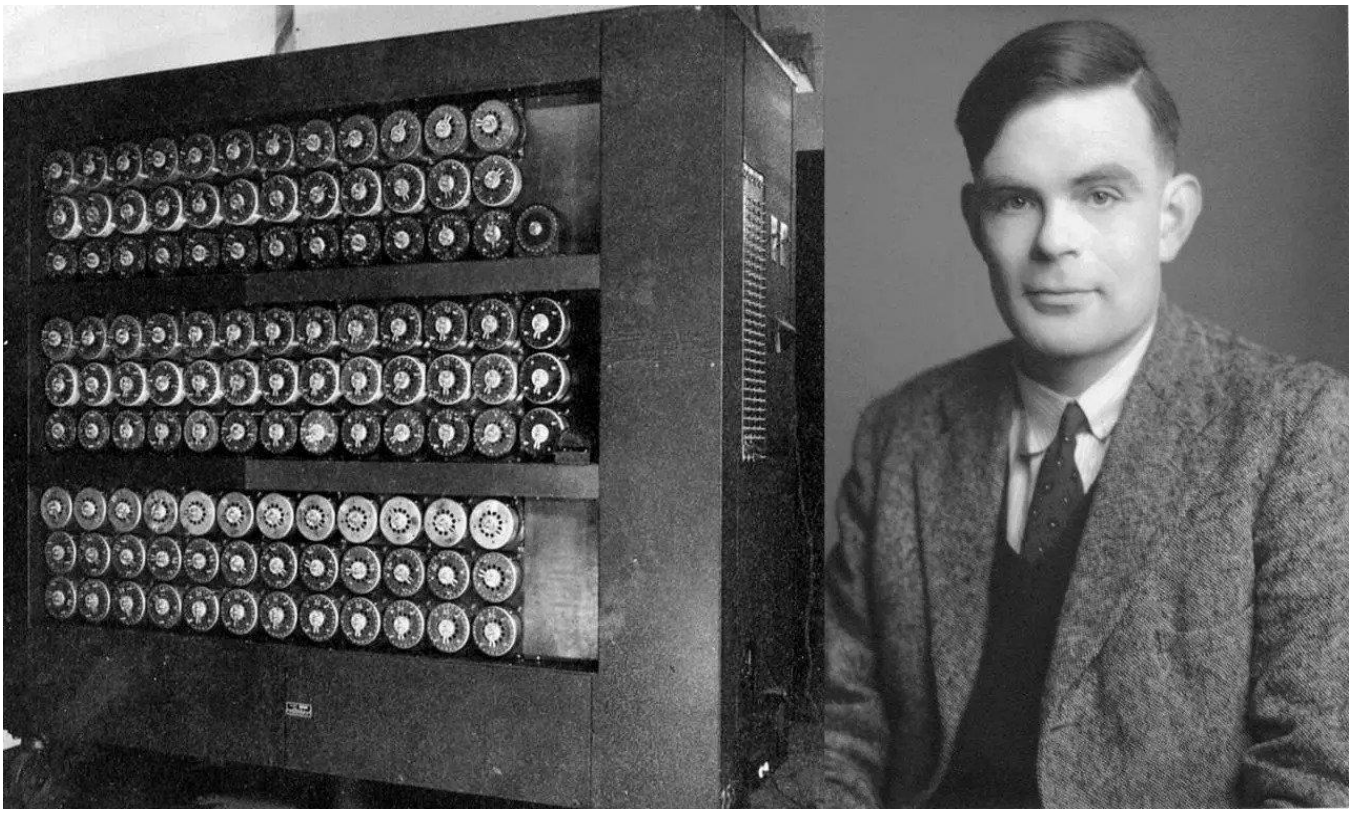
Pendant des siècles, l'exécution des algorithmes était confiée à des humains, souvent appelés *calculateurs*. Or, l'être humain est ***(Lent, Fatigable, Faillible)***

Pour que la science moderne puisse véritablement émerger, il devenait nécessaire de confier ces recettes à un exécutant infatigable et fiable. C'est ce besoin fondamental d'automatiser l'algorithme qui nous conduit naturellement à la **deuxième partie** de cette histoire.

2- LA MECANISATION DU CALCUL (LA MACHINE)

2.1 Le Modèle Universel de Turing (1936)

Pendant plus de mille ans, l'exécution des algorithmes est restée manuelle. Le tournant décisif a lieu en 1936, grâce au mathématicien britannique **Alan Turing**.



Turing se pose une question fondamentale : ***"Est-il possible de construire une machine qui pourrait exécuter n'importe quel algorithme, sans intervention humaine ?"***

Il théorise alors la **Machine de Turing**. Ce n'est pas encore un objet physique avec des fils et des écrans, c'est un modèle mathématique. Il prouve qu'une machine peut tout calculer si elle possède trois éléments simples :

1. **Une Mémoire (Le Ruban) :** Une bande infinie de papier divisée en cases, où l'on peut écrire des symboles (0 ou 1).
2. **Un Processeur (La Tête) :** Un lecteur capable de lire une case, l'effacer, écrire par-dessus, et se déplacer à gauche ou à droite.
3. **Un Programme (La Table de Règles) :** Des instructions élémentaires (ex: *"Si tu lis 1, écris 0 et déplace-toi à droite"*).

Le concept clé : Turing prouve la séparation entre le **Matériel** (la machine physique qui ne change pas) et le **Logiciel** (le ruban qu'on change pour faire un calcul différent). C'est la naissance de l'ordinateur universel.

2.2 De la Théorie à la Réalité Physique

Comment passe-t-on du ruban de papier de Turing à votre ordinateur portable ?

L'architecture moderne (dite de Von Neumann) est simplement la version électronique de la machine de Turing :

Composant de Turing	Composant Moderne	Rôle dans le calcul
Le Ruban	**RAM (Mémoire)**	Stocker les variables (x, b, c) et les résultats intermédiaires.
La Tête	**CPU (Processeur)**	Effectuer les opérations (+, -, /, x) à très haute vitesse.
Les Symboles	**Bits (0 et 1)**	Tout est traduit en impulsions électriques (0 = 0V, 1 = 5V).

Exemple Fil Rouge : L'Exécution Binaire

Reprenons notre équation du second degré $\boldsymbol{x^2 + 10x = 39}$

Dans la Partie 1, nous l'avons décrite avec des phrases humaines. Mais la Machine de Turing (votre processeur) ne comprend ni le français, ni les maths. Elle ne comprend que le courant électrique (ON/OFF).

Pour que la machine exécute l'algorithme d'Al-Khwārizmī, chaque étape doit être brisée en instructions microscopiques appelées **Langage Machine**.

Voici à quoi ressemble notre algorithme "vu" par le processeur :

1. **LOAD 00001010** : Le CPU charge la valeur 10 (en binaire) dans un registre.
2. **SHR 1** : Il effectue un décalage de bits (ce qui revient à diviser par 2). *Résultat : 5*.
3. **MUL R1, R1** : Il multiplie le registre par lui-même. *Résultat : 25*.
4. **ADD 00100111** : Il ajoute 39 (en binaire). *Résultat : 64*.
5. **SQRT R1** : Il calcule la racine carrée (via une série de soustractions). *Résultat : 8*.
6. **STORE 0x4F** : Il stocke le résultat final dans la mémoire RAM à l'adresse 4F.

2.3 La Barrière de la Communication

Nous avons fait un progrès immense : la machine exécute l'algorithme à la vitesse de la lumière (des milliards d'opérations par seconde), sans jamais se fatiguer.

Regardez l'étape 2.3 ci-dessus. C'est illisible pour un humain.

Pour demander à la machine de résoudre une équation, le scientifique doit-il vraiment écrire des millions de lignes de codes binaires (010101...) ? C'est impossible et source d'erreurs. Il manque une dernière pièce au puzzle : un **Traducteur** qui nous permettrait d'écrire l'algorithme en langage humain et qui le convertirait automatiquement en langage machine.

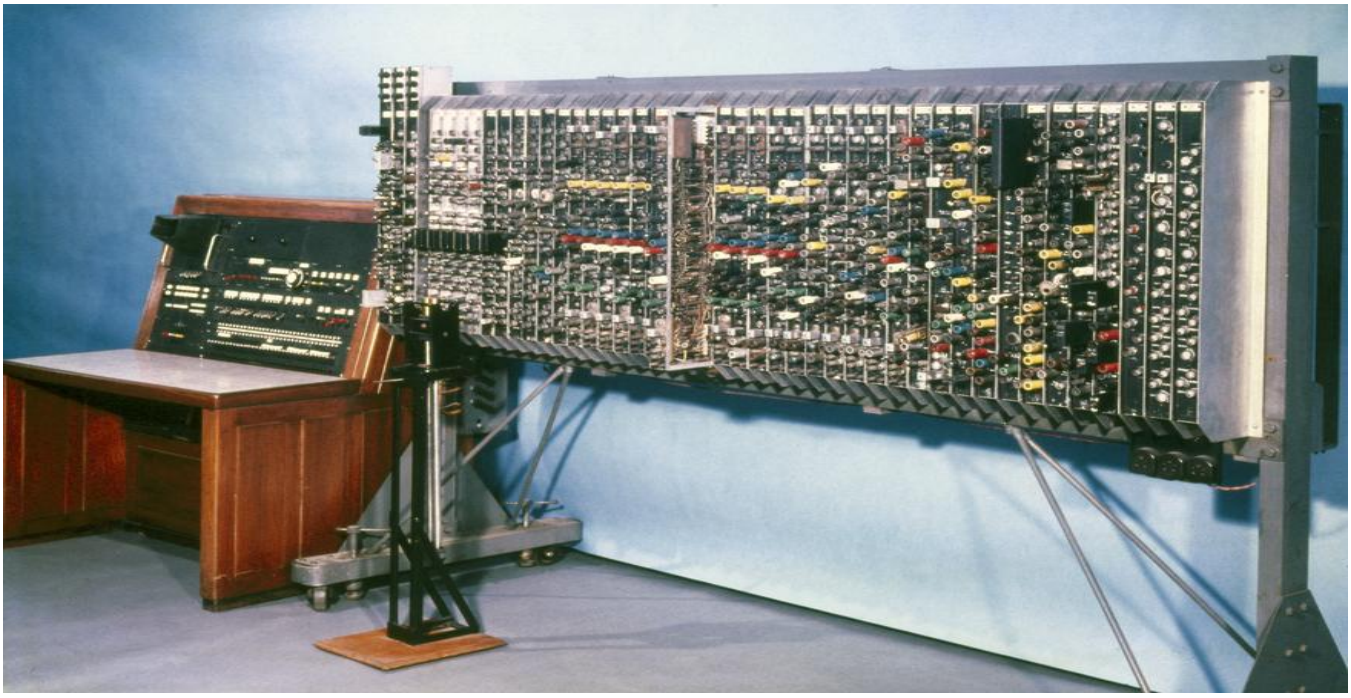
3- LA COMMUNICATION (LES LANGAGES DE PROGRAMMATION)

3.1 Le Problème du Fossé Sémantique

À la fin des années 1940, les premiers ordinateurs deviennent opérationnels. Cependant, pour "programmer" une équation aussi simple que celle d'Al-Khwārizmī, les scientifiques devaient manipuler des câbles ou rédiger de longues suites de 0 et de 1. Cette tâche, aussi essentielle qu'ingrate, les éloignait de la véritable réflexion scientifique.

Un gouffre séparait alors :

- **Le langage du scientifique** : clair et abstrait, comme $x^2 + 10x = 39$
- **Le langage de la machine** : brut, binaire, comme 01001011 10100010



Pour combler ce fossé, les informaticiens ont inventé une couche intermédiaire : **les langages de programmation de haut niveau** (Fortran, puis C, et aujourd'hui Python, R, etc.). Grâce à eux, l'humain peut enfin s'exprimer dans une langue proche de la logique et des mathématiques, sans se soucier de l'électronique.

Un langage de programmation agit comme un **traducteur diplomatique**.

- Le scientifique rédige son **code source**, clair et structuré.
- Un **compilateur** ou un **interpréteur** lit ce code et le traduit automatiquement en une succession de 0 et 1 compréhensible par le processeur.

Définition : Un langage de programmation est un langage formel destiné à écrire des programmes informatiques. Il est composé d'instructions spécifiques capables de traiter des entrées, d'effectuer différentes opérations et de produire des résultats.

Voyons comment s'unissent l'héritage d'Al-Khwārizmī et la puissance de la machine moderne à travers un exemple en **R**. Reprenons l'équation du second degré $\boldsymbol{x^2 + 10x = 39.}$

```
# Importation des modules mathématiques (équivalent de "import math" en Python)
library(base)  # Chargé par défaut, contient sqrt()

# --- Entrées ---
b <- 10
c <- 39

# --- Algorithme (la méthode d'Al-Khwārizmī traduite en R) ---
etape1 <- b / 2
etape2 <- etape1^2
etape3 <- etape2 + c
etape4 <- sqrt(etape3)
x <- etape4 - etape1

# --- Sortie ---
print(paste("La solution est x =", x))

# Ou version plus élégante en R :
cat("La solution est x =", x, "\n")
```

Ce code, lisible et logique pour un humain, sera transformé à l'exécution en signaux électriques interprétés par la machine.

3.2 Les Trois Structures Universelles

Quel que soit le langage utilisé (Python, C++, Java…), tout programme repose sur **trois structures de base**, démontrées par le théorème de **Böhm-Jacopini** :

- **La Séquence (l'ordre)** : les instructions s'exécutent de haut en bas
  Exemple : on calcule `etape1`, puis `etape2`.
- **La Sélection (la condition)** : permet d'adapter le comportement du programme
  Exemple : `if delta < 0: print("Pas de solution")`
- **L'Itération (la boucle)** : permet de répéter une action
  Exemple : `while erreur > 0.001:` (on répète jusqu'à atteindre la précision voulue)

4- INTRODUCTION AUX LANGAGES PYTHON ET R POUR LA PROGRAMMATION SCIENTIFIQUE

Une fois que nous avons compris qu'un ordinateur a besoin d'instructions précises (algorithmes), il faut choisir la langue pour lui parler. Dans le monde de la recherche scientifique actuelle, deux langages se partagent le monopole : **Python** et **R**.

4.1 PYTHON : LE LANGAGE UNIVERSEL

Python est un langage de programmation créé au début des années 90 par le néerlandais Guido van Rossum. Son nom ne vient pas du serpent, mais de la troupe d'humoristes les *Monty Python*.

Sa philosophie tient en une phrase : **"La lisibilité compte."**

Contrairement aux anciens langages (C, C++) qui exigeaient des lignes de code complexes pour afficher une simple phrase, Python est conçu pour être lu presque comme de l'anglais. C'est ce qui en fait le langage le plus populaire au monde aujourd'hui, et le meilleur choix pour débuter.

Pourquoi utiliser Python en Écologie ?

Pour un spécialiste de l'écologie, Python est le **"Couteau Suisse"**. Il ne sert pas qu'à faire des calculs, il sert à tout :

1. **Manipulation de Données (Data Science) :** Python remplace Excel. Il peut ouvrir un fichier de 10 millions de lignes (ce qui ferait planter Excel) en une fraction de seconde pour nettoyer et trier des relevés de terrain.
2. **Cartographie et SIG :** Python est le moteur caché derrière des logiciels comme QGIS ou ArcGIS. Il permet d'automatiser l'analyse d'images satellites pour suivre la déforestation ou l'évolution du trait de côte.
3. **Automatisation :** Vous devez télécharger les données météo de 50 stations tous les jours ? Python peut le faire automatiquement pendant que vous dormez.

4.2 R : LE LANGAGE DES STATISTICIENS

R est bien plus qu'un simple langage de programmation. C'est un **environnement logiciel** dédié au calcul statistique et aux graphiques.

Il a été créé au milieu des années 90 par deux néo-zélandais, Ross Ihaka et Robert Gentleman (d'où le nom "R").

Contrairement à Python qui a été pensé par des informaticiens pour construire des logiciels, R a été pensé **par des statisticiens pour analyser des données**. Sa logique est donc très proche de celle des mathématiques et des tableaux de données.

B. Pourquoi R est-il le standard en Écologie ?

Si vous ouvrez une publication scientifique récente en écologie (dans *Nature* ou *Ecology Letters*), il y a 90% de chances que les analyses aient été faites sous R. Voici pourquoi :

1. **La Puissance Statistique Native :** R n'a pas besoin d'extensions pour faire des statistiques. Les tests de Student, les ANOVA, les modèles linéaires (GLM) sont intégrés au cœur du langage.
2. **La Visualisation de Données (ggplot2) :** R possède un système de création graphique extrêmement puissant. Il permet de créer des figures complexes, élégantes et prêtes pour la publication.
3. **La Communauté Scientifique :** C'est un cercle vertueux. Comme tous les chercheurs utilisent R, les nouvelles méthodes d'analyse (ex: indices de biodiversité fonctionnelle) sont toujours publiées sous forme de "paquets R" (Packages).

4.3 QU'EST-CE QU'UNE BIBLIOTHEQUE ?

Lorsque vous installez Python ou R, vous installez le langage "de base". Il sait faire des additions, des boucles et gérer du texte. Mais si vous voulez faire des statistiques complexes ou tracer des cartes, le langage de base ne suffit pas. C'est là qu'interviennent les **bibliothèques**.

Définition Formelle Une **Bibliothèque** (souvent appelée *Librairie*, *Module* ou *Package*) est un ensemble de fonctions pré-programmées par d'autres développeurs et mises à disposition de la communauté. C'est un "paquet" de code prêt à l'emploi que l'on greffe au programme principal pour lui ajouter de nouvelles fonctionnalités spécifiques.

A. L'Analogie de l'Atelier de Bricolage Pour bien comprendre, imaginez que le langage (Python ou R) est un **établi vide**.

- Avec l'établi seul, vous pouvez faire des choses simples (plier, poser).
- Si vous voulez faire de la plomberie, vous devez apporter la **boîte à outils "Plomberie"** (clés à molette, joints...).
- Si vous voulez faire de l'électricité, vous rangez la plomberie et vous ouvrez la **boîte à outils "Électricité"**.

En programmation, ces "boîtes à outils" sont les bibliothèques.

B. Le Principe : "Ne pas réinventer la roue" C'est le principe fondamental de la programmation scientifique. Un mathématicien a déjà passé 10 ans à écrire le code parfait pour calculer une *Transformée de Fourier*.

- **Sans bibliothèque :** Vous devez recréer ce code mathématique complexe vous-même (et risquer de faire des erreurs).
- **Avec bibliothèque :** Vous importez la bibliothèque (ex: NumPy), vous tapez une seule commande, et le calcul se fait parfaitement.

C. Comment ça s'utilise ? Une bibliothèque ne fonctionne pas toute seule, il faut demander au programme de la charger au début du script.

- *En Python :* On utilise le mot-clé import.

Python

```
import pandas  # J'apporte la boîte à outils "Tableaux Excel"
```

- *En R :* On utilise la fonction library().

R

```
library(ggplot2)  # J'apporte la boîte à outils "Graphiques"
```

4.4 LES ENVIRONNEMENTS DE DEVELOPPEMENT (IDE)

Le choix d'un langage de programmation (Python ou R) constitue la première étape de la démarche informatique. Cependant, pour être opérationnel, le scientifique doit disposer d'une infrastructure logicielle permettant la rédaction, le test et l'exécution du code. C'est le rôle de l'Environnement de Développement.

Définition : Environnement de Développement Intégré (IDE) Un Environnement de Développement Intégré (en anglais *Integrated Development Environment* ou IDE) est un logiciel qui rassemble, dans une interface graphique unique, l'ensemble des outils nécessaires à la programmation. Il vise à maximiser la productivité du programmeur en automatisant les tâches répétitives et en simplifiant la navigation dans le code.

Un IDE complet regroupe généralement trois composants fondamentaux :

1. **L'Éditeur de code source :** Un traitement de texte spécialisé qui offre des fonctionnalités avancées telles que la coloration syntaxique (mise en évidence des mots-clés du langage par des couleurs), l'indentation automatique et l'autocomplétion (suggestion de fonctions pendant la frappe).
2. **L'Interpréteur ou le Compilateur :** Le système qui traduit le code humain en instructions machine exécutables. L'IDE permet de lancer cette exécution directement depuis l'éditeur.
3. **Le Déogueur (Debugger) :** Un outil d'analyse permettant de tester le programme étape par étape pour identifier et corriger les erreurs (bugs) dans la logique du code.

Dans le contexte de la science des données, les IDE intègrent également des outils de visualisation pour afficher directement les graphiques et les tableaux de données.

4.4.1 RStudio (L'interface de référence pour R)

RStudio est l'environnement de développement standard pour le langage R. Il est spécifiquement conçu pour répondre aux besoins de l'analyse statistique et de la recherche reproductible. Son interface est divisée en quatre quadrants fonctionnels qui permettent de gérer l'intégralité du flux de travail scientifique :

- **L'Éditeur de Source (Haut-Gauche) :** C'est l'espace de rédaction des scripts (.R). Il permet d'enregistrer les séquences d'instructions pour les réutiliser ultérieurement. Il supporte également le format **RMarkdown**, qui permet de mêler texte explicatif et code pour générer des rapports automatisés.
- **La Console (Bas-Gauche) :** C'est l'interface directe avec le moteur R. Les instructions y sont exécutées immédiatement et les résultats textuels (calculs, résumés statistiques) s'y affichent. C'est l'espace privilégié pour l'exploration rapide des données.
- **L'Environnement et l'Historique (Haut-Droite) :** Cet espace liste tous les "objets" (variables, tableaux de données, fonctions) actuellement stockés dans la mémoire vive de la session. Il permet de visualiser la structure des données importées sans avoir à écrire de code.

- **Fichiers, Graphiques, Packages et Aide (Bas-Droite) :** Cette zone multifonctionnelle permet de naviguer dans les fichiers de l'ordinateur, de visualiser les graphiques générés **(Plots)**, de gérer les extensions installées **(Packages)** et de consulter la documentation officielle des fonctions **(Help).**

4.4.2 Anaconda et Jupyter (L'écosystème Data Science)

Pour le langage Python, l'approche diffère légèrement. Plutôt qu'un logiciel unique, la communauté scientifique privilégie souvent une distribution complète nommée **Anaconda**.

A. La Distribution Anaconda Anaconda est une plateforme de distribution libre et open source pour les langages Python et R. Son objectif principal est de simplifier la gestion des paquets (bibliothèques) et le déploiement. Elle inclut :

- Le noyau du langage Python.
- Plus de 1500 bibliothèques scientifiques pré-installées (dont NumPy, Pandas, Matplotlib, Scikit-learn).
- Un gestionnaire de paquets (conda) qui assure la compatibilité entre les différentes versions des bibliothèques.

B. Jupyter Notebook Au sein d'Anaconda, l'outil de développement le plus utilisé pour l'analyse exploratoire et l'enseignement est **Jupyter Notebook**. Il s'agit d'une application web (s'exécutant dans le navigateur) qui repose sur le concept de programmation littéraire. Un document Jupyter (.ipynb) est constitué d'une suite de **cellules** pouvant contenir :

- Du code exécutable (Python ou R).
- Du texte enrichi (Markdown) pour les explications, les formules mathématiques (LaTeX) et les titres.
- Les sorties du code (graphiques, tableaux) affichées directement sous la cellule qui les a générées.

Cette structure favorise la traçabilité des analyses et le partage des résultats scientifiques.

4.4.3 VS Code (L'éditeur de code extensible)

Visual Studio Code (VS Code) est un éditeur de code source développé par Microsoft. Contrairement à RStudio qui est spécialisé, ou Jupyter qui est orienté vers la présentation, VS Code est un éditeur généraliste et modulaire.

Sa caractéristique principale réside dans son système d'**extensions**. De base, VS Code est un éditeur léger. Pour le transformer en un véritable environnement de développement scientifique, l'utilisateur installe des modules spécifiques :

- **Extension Python / R :** Ajoute le support complet du langage (coloration, exécution, débogage).
- **IntelliSense :** Un système avancé de complétion de code qui suggère les variables et les fonctions en fonction du contexte, accélérant considérablement l'écriture.
- **Git intégré :** Permet de gérer les versions du code (système de sauvegarde chronologique) directement depuis l'interface.

VS Code est particulièrement prisé pour les projets complexes nécessitant l'usage de plusieurs langages simultanément (par exemple, un projet alliant une analyse de données en Python et une interface web en HTML/JavaScript).

5- VARIABLES ET FONCTIONS

Pour programmer une analyse scientifique, nous n'avons besoin de maîtriser que deux éléments fondamentaux : les données que l'on observe (les Variables) et les opérations que l'on effectue dessus (les Fonctions).

5.1 Les Variables (Le stockage des données)

En écologie, vous manipulez constamment des grandeurs qui changent selon le site ou le moment (température, pH, abondance). En informatique, nous devons stocker ces informations.

Définition : Une variable est un espace de stockage dans la mémoire de l'ordinateur, identifié par un nom unique, et contenant une valeur (un nombre, du texte, etc.). Contrairement à une constante, le contenu d'une variable peut être modifié tout au long du programme.

A. L'Affectation en R : La Flèche (<-)

Pour créer une variable ou modifier son contenu, on utilise une opération appelée **l'affectation.** Le langage R utilise un symbole spécifique : la flèche orientée vers la gauche `<-`. Cela se lit : **"Mettre la valeur dans la variable".**

Exemple :

```
# Création de la variable 'temperature' avec la valeur 25.5
temperature <- 25.5

# Création de la variable 'site' avec le nom du lieu
site <- "Lac Oubeira"
```

B. Les Types de Données

L'ordinateur doit connaître la nature de ce qu'il stocke. On ne calcule pas la moyenne d'un mot. Voici les trois types fondamentaux en R :

Type (Nom R)	Définition	Exemple Scientifique
Numeric	Nombre (entier ou décimal). Sert aux calculs.	ph <- 7.2
Character	Chaîne de caractères (Texte). Sert aux descriptions.	espece <- "Parus major"
Logical	Booléen (Vrai/Faux). Sert aux conditions logiques.	espece_protegee <- TRUE

5.2 Les Fonctions (L'automatisation des calculs)

Une fois les données stockées, nous devons les traiter. Plutôt que de réécrire les calculs manuellement à chaque fois, nous utilisons des fonctions.

Définition : **Une fonction** est un bloc de code nommé et réutilisable qui effectue une tâche spécifique. Elle prend des données en entrée (appelées Arguments), leur applique un traitement (l'algorithme), et renvoie un résultat en sortie.

Exemple d'application : Calcul de Densité

Imaginons que nous voulions calculer la densité d'une population ($D = N/S$). Au lieu de faire le calcul manuellement pour chaque site, nous créons une fonction.

Syntaxe en **R** :

```
# 1. DÉFINITION DE LA FONCTION
# On crée un outil nommé 'calculer_densite'
# Il a besoin de deux arguments : 'nb_individus' et 'surface'
calculer_densite <- function(nb_individus, surface) {

  resultat <- nb_individus / surface  # Le Traitement (La formule)
  return(resultat)                    # La Sortie (Le renvoi du résultat)
}
# 2. UTILISATION (APPEL) DE LA FONCTION
# On applique notre outil sur des données réelles
site_A <- calculer_densite(500, 20)  # R fait le calcul 500/20
site_B <- calculer_densite(100, 50)  # R fait le calcul 100/50

# Affichage
print(site_A) # Affiche 25
```

6- STRUCTURES DE CONTROLE : LOGIQUE ET REPETITION

Jusqu'à présent, nos programmes étaient linéaires : ils s'exécutaient de la ligne 1 à la ligne 10 sans dévier. Or, la nature n'est pas linéaire. Un écologue doit prendre des décisions ("Si le sol est acide, alors...") et traiter des milliers de données.

Pour cela, **nous avons besoin de la Sélection (Conditions) et l'Itération (Boucles).**

6.1 Les Structures Conditionnelles (La prise de décision)

Un programme doit être capable d'adapter son comportement en fonction des données qu'il analyse. C'est la structure **SI ... ALORS ... SINON.**

Définition : **Une structure conditionnelle** est une instruction qui permet d'exécuter un bloc de code uniquement si une condition précise est remplie (Vraie).

A. La Syntaxe en R

Instruction	Rôle
if (condition)	Introduit la condition à tester
else	Gère tous les autres cas si la condition est fausse
else if	Teste une condition intermédiaire (équivalent elif Python)

B. Les Opérateurs de Comparaison

Pour poser une condition, il faut savoir comparer deux valeurs. L'ordinateur utilise des symboles mathématiques stricts :

Symbole	Signification	Exemple
==	Égal à (**double égal** pour comparer)	ph == 7
!=	Différent de	site != "A"
< / >	Inférieur / Supérieur	temp > 25
<= / >=	Inférieur ou égal / Supérieur ou égal	polluant <= 0.05

C. Exemple Scientifique : Diagnostic de qualité de l'eau

Imaginons un script qui analyse automatiquement **le taux d'oxygène dissous (mg/L)** pour déterminer la santé d'un cours d'eau.

```
# Donnée relevée sur le terrain
taux_o2 <- 3.5

# Prise de décision automatique
if (taux_o2 < 4) {
  print("ALERTE : Zone en hypoxie (Manque d'oxygène)")

} else if (taux_o2 < 6) {
  print("ATTENTION : Qualité médiocre")

} else {
  print("NORMAL : Bonne oxygénation")
}
```

Ici, comme 3.5 est inférieur à 4, le programme affiche l'alerte et ignore le reste du code.

6.2 Les Boucles (L'Automatisation)

C'est ici que réside la véritable puissance de l'ordinateur. Là où un humain fatigue après 10 calculs, l'ordinateur peut répéter l'opération un million de fois sans erreur et en une fraction de seconde.

Définition : **Une boucle** est une structure qui permet de répéter un bloc d'instructions.

On distingue deux types principaux :

A. La Boucle `for` (Pour chaque...)

On l'utilise quand on connaît à l'avance les éléments à traiter. Elle parcourt une collection de données (liste de relevés).

Usage : *"Pour chaque échantillon dans ma liste, fais ce calcul."*

Exemple : Conversion de températures pour plusieurs sites

```
# On définit une séquence de 1 à 5
nombres <- 1:5
# Pour chaque 'i' dans notre séquence
for (i in nombres) {
  carre <- i^2
  print(paste("Le carré de", i, "est", carre))
}_T)
```

B. La Boucle `while` (Tant que...)

On l'utilise quand on ne sait pas quand le phénomène va s'arrêter. Elle répète l'action tant qu'une condition reste vraie. C'est la boucle reine pour la modélisation et la simulation temporelle.

Exemple : Simulation de croissance bactérienne

```
population <- 100  # Population initiale de bactéries

# Tant que la population n'a pas atteint le seuil critique de 1000
while (population < 1000) {
  population <- population * 2   # La population double
  print(paste("Population actuelle :", population))
}

print("Seuil critique atteint !")
```

7. STRUCTURES DE DONNEES FONDAMENTALES

Jusqu'à présent, nous avons manipulé des variables simples contenant une seule valeur (un nombre, une chaîne de caractères). Cependant, dans le cadre d'études écologiques, les données se présentent rarement de manière isolée. Nous manipulons des séries temporelles, des inventaires de biodiversité ou des tableaux de relevés.

Pour gérer ces collections d'informations, nous utilisons des **structures de données**.

Définition : Structure de Données

Une structure de données est une manière spécifique d'organiser, de gérer et de stocker des données dans la mémoire d'un ordinateur afin qu'elles puissent être utilisées efficacement. Elle définit la relation entre les données et les opérations qui peuvent être effectuées sur celles-ci.

Nous comparerons ici comment ces structures s'écrivent dans nos deux langages : **Python** et **R**.

7.1 Les Séquences : Listes et Vecteurs

Les séquences sont des collections ordonnées d'éléments. Chaque élément possède une position précise, appelée **index**.

Attention à l'indexation :

- En **Python**, on commence à compter à **0**.
- En **R**, on commence à compter à **1**.

A. Les Listes et Vecteurs : La collection modifiable

C'est la structure la plus courante. Elle permet de stocker une série de mesures. Sa caractéristique principale est la **mutabilité** : on peut ajouter, supprimer ou modifier des éléments après la création.

- **Syntaxe Python :** Crochets [...] (Liste)
- **Syntaxe R :** Fonction c(...) (Vecteur)
- **Usage scientifique :** Stocker une série de mesures expérimentales susceptibles d'être corrigées ou complétées.

Exemple : Relevés de pH

Python

```
# Liste de mesures
releves_ph = [7.1, 6.8, 7.2, 6.9]

# Modification : On corrige la DEUXIÈME valeur (index 1 en Python)
releves_ph[1] = 6.5

# Ajout : On ajoute une mesure à la fin
releves_ph.append(7.0)

print(releves_ph)
# Résultat : [7.1, 6.5, 7.2, 6.9, 7.0]
```

R

```
# Vecteur de mesures
releves_ph <- c(7.1, 6.8, 7.2, 6.9)

# Modification : On corrige la DEUXIÈME valeur (index 2 en R !)
releves_ph[2] <- 6.5

# Ajout : On combine le vecteur existant avec une nouvelle valeur
releves_ph <- c(releves_ph, 7.0)

print(releves_ph)
# Résultat : 7.1 6.5 7.2 6.9 7.0
```

B. Les Tuples : La collection immuable

Le tuple est une liste "verrouillée". Une fois créé, il ne peut plus être modifié (ni ajout, ni suppression, ni changement de valeur).

- **Syntaxe Python :** Parenthèses (...)
- **Syntaxe R :** Pas d'équivalent strict (on utilise un vecteur classique).
- **Usage scientifique :** Stocker des données constantes (coordonnées GPS, constantes physiques).

Exemple : Coordonnées GPS

Python

```
# Coordonnées (Latitude, Longitude)
station_gps = (36.189, 5.403)

# Tentative de modification
# station_gps[0] = 36.190
# --> ERREUR ! Python bloque la modification pour protéger la donnée.
```

R

```
# En R, on utilise un vecteur classique
station_gps <- c(36.189, 5.403)

# R n'a pas de protection "immuable" par défaut.
# C'est au scientifique de faire attention à ne pas écraser ses données.
```

7.2 Les Mappages : Dictionnaires et Listes Nommées

Contrairement aux listes qui utilisent des numéros (index) pour accéder aux données, les dictionnaires utilisent un système de **Clé-Valeur**. C'est une structure d'association.

Définition : Dictionnaire

Une collection où chaque valeur est associée à une **clé** (un nom) unique. Il permet de récupérer une information non pas par sa position, mais par son sens sémantique.

- **Syntaxe Python :** Accolades { clé : valeur } (Dictionnaire)
- **Syntaxe R :** Fonction list(clé = valeur) (Liste nommée)
- **Usage scientifique :** Associer des attributs à une entité biologique ou décrire les métadonnées d'un site.

Exemple : Traits d'une espèce

Python

```
# Création du dictionnaire
espece_info = {
    "nom_scientifique": "Cedrus atlantica",
    "type": "Persistant",
    "statut_uicn": "En danger"
}

# Accès par la CLÉ (et non le numéro)
print(espece_info["statut_uicn"])
# Résultat : En danger
```

R

```
# Création d'une liste nommée
espece_info <- list(
    nom_scientifique = "Cedrus atlantica",
    type = "Persistant",
    statut_uicn = "En danger"
)

# Accès par le signe $ suivi du nom de la clé
print(espece_info$statut_uicn)
# Résultat : "En danger"
```

7.3 Les Ensembles (Sets) et l'Unicité

L'ensemble en informatique est l'implémentation directe de l'ensemble mathématique. C'est une collection non ordonnée d'éléments **uniques**.

Définition : Ensemble

Un ensemble est une collection qui ne contient **aucun doublon**. Il est optimisé pour les opérations ensemblistes (union, intersection).

- **Syntaxe Python :** Accolades {...} ou fonction set()
- **Syntaxe R :** Fonction unique() sur un vecteur.
- **Usage scientifique :** Éliminer les doublons d'une liste brute (Calcul de richesse spécifique).

Exemple : Nettoyage d'inventaire

Python

```
# Liste brute avec doublons
obs_brutes = ["Chêne", "Pin", "Chêne", "Cèdre", "Pin"]

# Conversion en SET pour éliminer les doublons
especes_uniques = set(obs_brutes)

print(especes_uniques)
# Résultat : {'Cèdre', 'Chêne', 'Pin'} (L'ordre est aléatoire)
```

R

```
# Vecteur brut avec doublons
obs_brutes <- c("Chêne", "Pin", "Chêne", "Cèdre", "Pin")

# Application de la fonction unique()
especes_uniques <- unique(obs_brutes)

print(especes_uniques)
# Résultat : "Chêne" "Pin" "Cèdre"
```

Résumé Comparatif des Structures

Structure	**Type Python**	**Type R**	**Accès aux données**	**Usage Principal**
Séquence	**Liste** [...]	**Vecteur** c(...)	Par Index ([0] vs [1])	Séries de mesures, colonnes de données.
Constante	**Tuple** (...)	*(Vecteur classique)*	Par Index	Coordonnées fixes.
Association	**Dictionnaire** {k:v}	**Liste Nommée** list(k=v)	Par Clé / Nom ($)	Base de données, paramètres.
Unicité	**Ensemble** set(...)	Fonction unique()	Impossible (pas d'ordre)	Filtrer les doublons.