الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعليم العالي والبحث العلمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
جامعة محمد بوضياف بالمسيلة
UNIVERSITE MOHAMED BOUDIAFDE M'SILA
كلية الرياضيات والإعلام الآلي
Faculté des Mathématiques et de l'Informatique
جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila
قسم الإعلام الآلي
Département de l'informatique

رقم: 04 /ق.إ.آ/ 2025

المسيلة في:2025/10/07

مستخلص من محضر اجتماع اللجنة العلمية لقسم الإعلام الآلي
رقم: 2025/03 المنعقد بتاريخ 2025/10/05 - دورة عادية

في يوم الاحد الخامس من شهر اكتوبر لسنة ألفين وخمسة وعشرون تم انعقاد اجتماع اللجنة العلمية لقسم الإعلام الآلي على الساعة التاسعة ونصف صباحا ، وكان من بين نقاط جدول الأعمال التطرق لمطبوعة الدروس للأستاذ **شترة محمد** المعنونة:

Programmation Informatique et d'Intelligence Artificielle

وبعد الاطلاع على تقريري الخبرة الإيجابيين وافقت اللجنة على اعتماد المطبوعة واستعمالها من قبل صاحبها في حدود ما يسمح به القانون.

رفعت الجلسة في نفس اليوم على الساعة الحادية عشر صباحا.

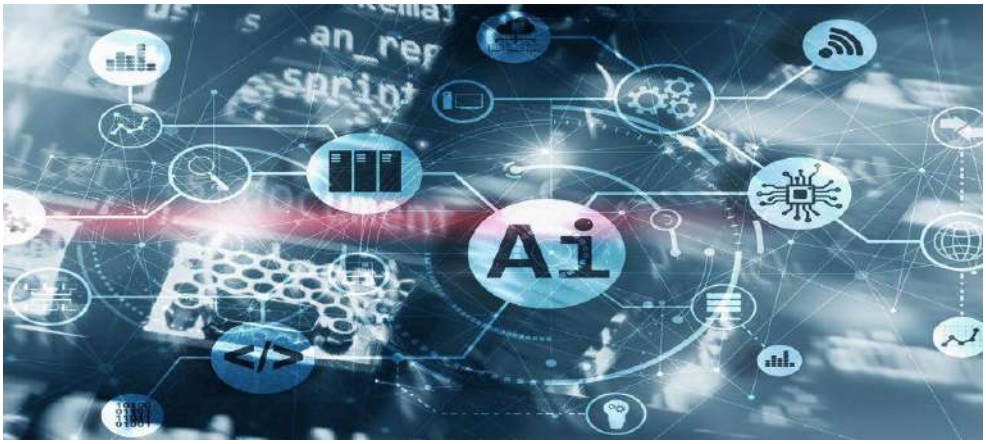
بركات عبد الباسط
رئيس اللجنة العلمية
لقسم الإعلام الآلي

جامعة محمد بوضياف بالمسيلة
رئيس اللجنة العلمية لقسم الإعلام الآلي
كلية الرياضيات والإعلام الآلي

الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعليم العالي و البحث العلمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOHAMED BOUDIAF - M'SILA

Faculté des Mathématiques et Informatique

Département d'informatique



Programmation informatique et d'intelligence artificielle

Cours pédagogique

1ière Année Master

Dr. Mohamed CHATRA

2025

Avant-propos

Ce support de cours « programmation informatique et d'intelligence artificielle » est destiné aux étudiants de première année master toutes les spécialités en Sciences et Technologie. Élaboré par le docteur CHATRA Mohamed, ce polycopié a servi de support aux cours qu'il a professés durant l'année 2024-2025 au département de Sciences de la Nature et de la Vis (SNV) de la faculté de Sciences université Mohamed BOUDIAF de M'Sila.

Compétences visées :

- Utilisation des outils informatiques pour l'acquisition, le traitement, la production et la diffusion de l'information
- Compétences en Python et gestion de projets,
- Compétences en automatisation et visualisation de données.

Objectifs :

- Acquérir les bases de solides en informatique.
- Apprendre à programmer en Python, Excel
- Maitriser l'automatisation de tâches
- Maitriser un logiciel de gestion de projets

Méthode pédagogique

- Cours magistraux.
- Travaux dirigés.
- Travaux pratiques.
- Interactivité maximale enseignant – étudiants.
- Travail personnel (surtout pour la programmation).
- Evaluation (identique aux textes en vigueur).

Répartition horaire :

- 2ème semestre de Master (M1)
- Durée (12 semaines):
  - Cours magistraux (1h30/semaine).
  - Travaux dirigés (1h30/semaine).
  - Travaux pratiques (3h/semaine)

Matériels nécessaires :

- Un ordinateur avec Python installé,
- Bibliothèques Python :
  - NumPy
  - Pandas,
  - Scikit-learn
  - Matplotlib
  - os.listdir
  - os.path.exists)
  - os.mkdir, os.rmdir)
  - Matplotlib, Seaborn, Plitly
  - Request
  - Beautiful Soup
  - Tkinter, PyQT
- Tensorflow ou PyTorch3

Programme :

Chapitre 1 : Fondamentaux de l'informatique et outils numériques (1 semaine à 2 séances : cours + 1 séance de TP convertie en cours)

- Variables, types de données et opérateurs
- Structures de contrôle :
  - Conditions
  - Boucles
- Fonctions
- Les fichiers :
  - Lecture,
  - écriture,
  - modification,
  - suppression.
- Les listes :
  - Ajout,
  - Suppression,
  - Modification
- Présentation de la notion de système d'exploitation :
  - Rôle,
  - Prise en charge des travaux,
  - Les types de systèmes d'exploitation (Linus et Windows),
  - Gestion des priorités,
  - Etc.
- Présentation des réseaux informatiques :
  - Principes de fonctionnement,
  - Adresse IP,
  - DNS,
  - Internet.

Chapitre 2 : Langage de programmation Python (4 semaines)

- La syntaxe de base :
  - Variables et types de variables
  - Opérations et opérateurs
- Structure de contrôles :
  - Conditions,
  - Boucles.
- Les chaines de caractères
- Les listes
- Fichiers
- Fonctions et modules (math, random),
- Les bibliothèques Pandas, NumPy, etc.
- Introduction à la programmation objet,
- Lire et traiter des fichiers textes (recherche, tri)
- Gérer des rapports simples (PDF, Excel)

Chapitre 3 : Programmation et automatisation (3 semaines)

- Principes d'Automatisation de tâches
  - Bibliothèques Python pour l'automatisation :
    - Os, shutil : manipulation de fichiers et dossiers
    - Openpyxl ou pandas : travail avec des fichiers Excel ou CSV
  - Définitions et exemples d'automatisation (envoi de mails,…)
- Manipulation de fichiers avec Python :
  - Utiliser les librairies pour :

- Parcourir un dossier (os.listdir)
- Vérifier l'existence d'un fichier ou dossier (os.path.exists)
- Créer ou supprimer des dossiers (os.mkdir, os.rmdir)
- Visualiser des données : Matplotlib, Seaborn, Plitly
- Request pour réagir avec des API
- Beautiful Soup pour le Scraping de données
- Tkinter, PyQT pour visualiser des données graphiques

➢ Copier ou déplacer des fichiers avec shutil…

Chapitre 4 : Apprentissage avancé d'Excel (2 semaines)

- Principes des macros et création d'une macro simple,
- Tableaux croisés dynamiques,
- Histogrammes,
- Diagrammes en barres,
- Araignée,
- Etc.

Chapitre 5 : Apprentissage de GanttProject (2 semaines)

- Introduction à la gestion de projets :
  - ➢ Qu'est-ce qu'un projet ?
  - ➢ Quels sont les enjeux de gestion d'un projet ?
  - ➢ Interface de GanttProject
- Les tâches (création, modification, organisation)
- Gestion du temps ((dates de début ou de fin de projet)
- Gestion des ressources

Table des matières

Introduction

Ce programme de formation vise à développer des compétences clés en informatique, en programmation, et en gestion de projets, avec un accent particulier sur l'utilisation d'outils comme Python, Excel, et des logiciels de gestion de projets. Voici une synthèse des éléments clés et des suggestions pour optimiser la formation :

Compétences visées

- Acquisition, traitement, production et diffusion de l'information.
- Maîtrise des outils de base (Excel, logiciels de gestion de projets) et des langages de programmation (Python).
- Apprentissage de la programmation en Python pour résoudre des problèmes concrets.
- Utilisation d'un logiciel de gestion de projets (comme Trello, Asana, ou Microsoft Project) pour organiser et suivre les tâches.
- Automatisation de tâches répétitives avec Python (scripts, macros, etc.).
- Visualisation de données avec des outils comme Matplotlib, Seaborn, ou Power BI.

Objectifs

- Comprendre les concepts fondamentaux de l'informatique et de la programmation.
- Savoir manipuler des données et utiliser des outils informatiques de manière efficace.
- Python : syntaxe de base, structures de données, fonctions, bibliothèques (pandas, numpy, etc.).
- Excel : formules, tableaux croisés dynamiques, macros (VBA).
- Développer des scripts pour automatiser des processus répétitifs.
- Utiliser des outils comme Selenium pour l'automatisation web ou des bibliothèques Python pour l'automatisation de fichiers.
- Apprendre à planifier, organiser et suivre des projets.
- Utiliser des fonctionnalités avancées comme les diagrammes de Gantt, les dépendances de tâches, etc.

Ce programme est bien structuré pour développer des compétences techniques et pratiques en informatique, programmation, et gestion de projets. En combinant théorie et pratique, et en favorisant l'interactivité, il permettra aux étudiants d'acquérir des compétences solides et directement applicables dans un contexte professionnel.

Chapitre 1 : Fondamentaux de l'informatique et outils numériques

Le premier chapitre est une introduction essentielle pour poser les bases nécessaires à la compréhension des concepts informatiques et à la maîtrise des outils numériques.

1. Variables, types de données et opérateurs

1.1. Variables

Une variable est un espace mémoire utilisé pour stocker une valeur. Elle est définie par un nom et peut contenir différents types de données.

1.1.1. Déclaration et affectation

En Python, une variable est créée lorsqu'une valeur lui est affectée :

```
x = 10              # Variable entière
nom = "Alice"       # Variable chaîne de caractères
pi = 3.14           # Variable flottante
est_vrai = True     # Variable booléenne
```

- Pas besoin de déclarer le type (Python est un langage à typage dynamique).
- Le nom d'une variable doit commencer par une lettre ou un underscore (_) et ne peut pas contenir d'espaces ni de caractères spéciaux autres que _.

1.1.2. Types de données

Les types de base les plus courants en Python :

Type	Exemple	Description
int (entier)	x = 5	Nombre entier
float (flottant)	y = 3.14	Nombre à virgule flottante
str (chaîne)	nom = "Alice"	Chaîne de caractères
bool (booléen)	vrai = True	Valeur logique (True ou False)
list (liste)	nombres = [1, 2, 3]	Collection modifiable d'éléments
tuple (tuple)	coord = (10, 20)	Collection immuable d'éléments
dict (dictionnaire)	pers = {"nom": "Alice", "âge": 25}	Clés-valeurs
set (ensemble)	ens = {1, 2, 3}	Collection unique d'éléments non ordonnés

Conversion de types :

```
a = 10
b = float(a)  # Convertir en flottant -> 10.0
c = str(a)    # Convertir en chaîne -> "10"
```

1.1.3. Opérateurs

Les opérateurs permettent de manipuler des valeurs.

Opérateurs arithmétiques

Opérateur	Description	Exemple
+	Addition	5 + 3 # 8
-	Soustraction	5 - 3 # 2
*	Multiplication	5 * 3 # 15
/	Division	5 / 2 # 2.5
//	Division entière	5 // 2 # 2
%	Modulo (reste de la division)	5 % 2 # 1
**	Puissance / Exponentiation	2 ** 3 # 8

Opérateurs de comparaison

Renvoient **True** ou **False** selon la condition.

Opérateur	Description	Exemple
==	Égal à	5 == 5 # True
!=	Différent de	5 != 3 # True
<	Inférieur à	3 < 5 # True
>	Supérieur à	5 > 3 # True
<=	Inférieur ou égal	3 <= 3 # True
>=	Supérieur ou égal	5 >= 3 # True

Opérateurs logiques

Permettent de combiner des conditions.

Opérateur	Description	Exemple
and	Vrai si toutes les conditions sont vraies	(5 > 3) and (10 > 5) # True
or	Vrai si au moins une condition est vraie	(5 > 3) or (10 < 5) # True
not	Inverse la condition	not (5 > 3) # False

Opérateurs d'assignation

Utilisés pour attribuer une valeur à une variable.

Opérateur	Équivalent à	Exemple
=	Affectation	x = 5
+=	x = x + 3	x += 3 # x = 8
-=	x = x - 3	x -= 3 # x = 2
*=	x = x * 3	x *= 3 # x = 15
/=	x = x / 3	x /= 3 # x = 5.0
//=	x = x // 3	x //= 3 # x = 1
%=	x = x % 3	x %= 3 # x = 2
**=	x = x ** 3	x **= 3 # x = 125

Opérateurs d'appartenance

Testent si un élément est présent dans une séquence (liste, chaîne, etc.).

Opérateur	Description	Exemple
in	Vrai si l'élément est présent	"a" in "apple" # True
not in	Vrai si l'élément est absent	"b" not in "apple" # True

```
fruits = ["pomme", "banane", "cerise"]
if "banane" in fruits:
    print("Banane trouvée !")
```

Opérateurs d'identité

Comparaison d'objets en mémoire.

Opérateur	Description	Exemple
is	Vrai si les objets sont identiques	a is b
is not	Vrai si les objets sont différents	a is not b

Exemple :

```
x = [1, 2, 3]
y = x
if x is y:
    print("x et y sont identiques")
```

Exemples d'application

```
# Déclaration de variables
a = 10
b = 3.5
nom = "Alice"
est_valide = True
# Opérations arithmétiques
somme = a + b
produit = a * b
# Conditions et comparaison
if a > b:
    print("a est plus grand que b")
# Boucles et listes
nombres = [1, 2, 3, 4, 5]
for n in nombres:
    print(n)
# Manipulation de fichiers
with open("test.txt", "w") as fichier:
    fichier.write("Bonjour")
```

1.2. Conditions (if, elif, else)

Elles permettent d'exécuter un bloc de code si une condition est remplie.
Syntaxe de base

```
if condition:
    # Code exécuté si la condition est vraie
elif autre_condition:
    # Code exécuté si la première est fausse mais celle-ci est vraie
else:
    # Code exécuté si aucune des conditions précédentes n'est vraie
```

Exemples

Condition simple

```
age = 20
if age >= 18:
    print("Vous êtes majeur")
```

Condition avec **elif**

```
note = 17
if note >= 18:
    print("Excellent")
elif note >= 16:
    print("Bien")
elif note >= 10:
    print("Passable")
else:
    print("Échec")
```

Conditions combinées (**and**, **or**, **not**)

```
temp = 25
humidite = 60
if temp > 20 and humidite < 70:
    print("Temps agréable")
if temp > 30 or humidite > 80:
    print("Temps inconfortable")
if not temp < 10:
    print("Il ne fait pas froid")
```

Condition imbriquée

```
age = 25
revenu = 3000
if age > 18:
    if revenu > 2500:
        print("Crédit accordé")
    else:
        print("Revenu insuffisant")
else:
    print("Vous devez être majeur")
```

1.3. Boucles (for, while)

Les boucles permettent d'exécuter plusieurs fois une même instruction.

Boucle for

Elle est utilisée pour parcourir des séquences (listes, chaînes, plages de nombres).
Syntaxe de base

```
for variable in sequence:
    # Code exécuté à chaque itération
```

Exemples
Boucle **for** sur une liste

```
nombres = [1, 2, 3, 4, 5]
for nombre in nombres:
    print(nombre)
```

Boucle **for** sur une chaîne de caractères

```
mot = "Python"
for lettre in mot:
    print(lettre)
```

Boucle avec **range()** (itération sur une plage de nombres)

```
for i in range(5):  # de 0 à 4
    print(i)
for i in range(1, 10, 2):  # de 1 à 9 avec un pas de 2
    print(i)
```

Boucle **for** avec **enumerate()**

```
fruits = ["pomme", "banane", "cerise"]
for index, fruit in enumerate(fruits):
    print(f"Indice {index} : {fruit}")
```

Boucle **for** avec **zip()** (parcourir plusieurs listes en parallèle)

```
prenoms = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 22]
for prenom, age in zip(prenoms, ages):
    print(f"{prenom} a {age} ans")
```

Boucle while

Elle exécute un bloc de code tant qu'une condition est vraie.
Syntaxe de base

```
while condition:
    # Code exécuté tant que la condition est vraie
```

Exemples

Boucle simple

```
compteur = 0
while compteur < 5:
    print(compteur)
    compteur += 1
```

Demander une saisie utilisateur jusqu'à une condition spécifique

```
mot_de_passe = ""
while mot_de_passe != "secret":
    mot_de_passe = input("Entrez le mot de passe : ")
print("Accès autorisé")
```

1.3.1. Instructions de contrôle de boucle

Ces instructions permettent d'agir sur le comportement des boucles.
break : arrêter une boucle immédiatement

```
for i in range(10):
    if i == 5:
        break  # Arrête la boucle lorsque i vaut 5
    print(i)
```

```
while True:
    entree = input("Tapez 'stop' pour quitter : ")
    if entree == "stop":
        break
```

continue : passer à l'itération suivante

```
for i in range(5):
    if i == 2:
        continue  # Ignore l'affichage de 2
    print(i)
```

else après une boucle
Le bloc **else** est exécuté seulement si la boucle se termine normalement (sans **break**).

```
for i in range(5):
    print(i)
else:
    print("Boucle terminée sans interruption")
```

2. Fonctions en Python

Une fonction est un bloc de code réutilisable qui exécute une tâche spécifique. Elle permet d'éviter la répétition de code et d'améliorer la structure d'un programme.

2.1. Définition et appel d'une fonction

Syntaxe de base

```
def nom_de_la_fonction(paramètres):
    # Instructions
    return valeur  # (Optionnel)
```

Exemple simple

```
def saluer():
    print("Bonjour !")
saluer()  # Appel de la fonction
```

2.2. Paramètres et arguments

Les fonctions peuvent accepter des paramètres pour être plus flexibles.

2.2.1. Fonction avec un paramètre

```
def dire_bonjour(nom):
    print(f"Bonjour, {nom} !")
dire_bonjour("Alice")
dire_bonjour("Bob")
```

2.2.2. Fonction avec plusieurs paramètres

```
def additionner(a, b):
    return a + b
resultat = additionner(3, 5)
print(resultat)  # 8
```

2.3. Valeurs de retour (return)

Une fonction peut retourner une valeur à l'aide de return.
Exemple avec retour de valeur

```
def carre(nombre):
    return nombre ** 2
resultat = carre(4)
print(resultat)  # 16
```

Une fonction peut retourner plusieurs valeurs sous forme de tuple.

```
def calculs(a, b):
    somme = a + b
    produit = a * b
    return somme, produit
s, p = calculs(3, 4)
print(s, p)  # 7 12
```

2.4. Types de paramètres

2.4.1. Paramètres avec valeur par défaut

Si aucun argument n'est fourni, la valeur par défaut est utilisée.

```
def bonjour(nom="inconnu"):
    print(f"Bonjour, {nom} !")
bonjour()  # Bonjour, inconnu !
bonjour("Alice")  # Bonjour, Alice !
```

2.4.2. Paramètres nommes (key=value)

Les arguments peuvent être passés en précisant leur nom.

```
def informations(nom, age):
    print(f"Nom: {nom}, Âge: {age}")
informations(age=30, nom="Alice")  # Ordre des arguments inversé
```

2.4.3. Paramètres avec un nombre variable d'arguments

***args** : Nombre variable de valeurs
Permet de passer un nombre illimité d'arguments sous forme de tuple.

```
def somme(*nombres):
    return sum(nombres)
print(somme(1, 2, 3, 4))  # 10
print(somme(10, 20))  # 30
```

****kwargs** : Nombre variable d'arguments nommés
Permet de passer des arguments sous forme de dictionnaire.

```
def afficher_infos(**infos):
    for cle, valeur in infos.items():
        print(f"{cle}: {valeur}")
afficher_infos(nom="Alice", age=30, ville="Paris")
```

2.5. Portée des variables (local vs global)

Une variable déclarée dans une fonction est locale et inaccessible en dehors.

```
def test():
    x = 10          # Variable locale
    print(x)
test()
# print(x)          # Erreur : x n'existe pas en dehors de la fonction
```

Pour modifier une variable globale dans une fonction, utilisez global.

```
x = 5
def modifier():
    global x
    x = 10
modifier()
print(x)  # 10
```

2.6. Fonctions lambda (fonctions anonymes)

Les **fonctions lambda** sont des fonctions anonymes définies en une seule ligne.

```
carre = lambda x: x ** 2
print(carre(5))  # 25
```

Exemple avec **map()** et **filter()**

```
nombres = [1, 2, 3, 4, 5]
# Doubler chaque nombre
double = list(map(lambda x: x * 2, nombres))
print(double)  # [2, 4, 6, 8, 10]
# Filtrer les nombres pairs
pairs = list(filter(lambda x: x % 2 == 0, nombres))
print(pairs)  # [2, 4]
```

2.7. Fonctions récursives

Une fonction récursive s'appelle elle-même pour résoudre un problème.
Exemple : Factorielle d'un nombre

```
def factorielle(n):
    if n == 0:
        return 1
    return n * factorielle(n - 1)
print(factorielle(5))  # 120
```

Les fonctions sont essentielles pour structurer et optimiser un programme.

3. Manipulation des fichiers en Python

La manipulation des fichiers est une compétence essentielle en programmation. Elle permet de lire, écrire, modifier et supprimer des fichiers, ce qui est utile pour stocker des données, générer des rapports ou interagir avec des fichiers externes. Voici une explication détaillée de ces opérations en Python.

3.1. Ouvrir un fichier (open())

La fonction open() permet d'ouvrir un fichier. Elle prend deux arguments :

- Le nom du fichier
- Le mode d'ouverture

Modes d'ouverture

Mode	Description
"r"	Lecture seule (par défaut). Erreur si le fichier n'existe pas
"w"	Écriture (écrase le contenu). Crée un fichier s'il n'existe pas
"a"	Ajout en fin de fichier. Crée un fichier s'il n'existe pas
"x"	Création. Erreur si le fichier existe
"r+"	Lecture et écriture
"w+"	Écriture et lecture (efface le fichier existant)
"a+"	Ajout et lecture

3.2. Lecture d'un fichier (read(), readline(), readlines())

Lire tout le contenu

```
with open("fichier.txt", "r") as fichier:
    contenu = fichier.read()
    print(contenu)
```

Lire ligne par ligne (**readline()**)

```
with open("fichier.txt", "r") as fichier:
    ligne = fichier.readline()  # Lit la première ligne
    print(ligne)
```

Lire toutes les lignes sous forme de liste (**readlines()**)

```
with open("fichier.txt", "r") as fichier:
    lignes = fichier.readlines()
    print(lignes)  # Liste contenant chaque ligne du fichier
```

Lire un fichier ligne par ligne avec une boucle

```
with open("fichier.txt", "r") as fichier:
    for ligne in fichier:
        print(ligne.strip())  # .strip() pour enlever les sauts de ligne
```

3.3. Écriture dans un fichier (write())

3.3.1. Écraser et écrire (w)

```
with open("fichier.txt", "w") as fichier:
    fichier.write("Bonjour\n")
    fichier.write("Python est puissant\n")
```

Si le fichier existait, son contenu est supprimé avant l'écriture.

3.3.2. Ajouter du texte à la fin (a)

```
with open("fichier.txt", "a") as fichier:
    fichier.write("Nouvelle ligne ajoutée\n")
```

3.4. Modification d'un fichier

Python n'offre pas de modification directe. Il faut lire, modifier, puis réécrire le fichier.

3.4.1. Modifier une ligne spécifique

```
# Lire et modifier
with open("fichier.txt", "r") as fichier:
    lignes = fichier.readlines()
lignes[1] = "Ligne modifiée\n"  # Modifier la deuxième ligne
# Réécrire le fichier
with open("fichier.txt", "w") as fichier:
    fichier.writelines(lignes)
```

3.4.2. Rechercher et remplacer du texte

```
with open("fichier.txt", "r") as fichier:
    contenu = fichier.read()
contenu = contenu.replace("ancien texte", "nouveau texte")
with open("fichier.txt", "w") as fichier:
    fichier.write(contenu)
```

3.5. Suppression d'un fichier (os.remove())

La suppression d'un fichier se fait avec le module os.

```
import os
if os.path.exists("fichier.txt"):
    os.remove("fichier.txt")
    print("Fichier supprimé")
else:
    print("Le fichier n'existe pas")
```

La manipulation des fichiers est une compétence clé en programmation. En maîtrisant la lecture, l'écriture, la modification et la suppression de fichiers, vous pouvez interagir avec des données externes et créer des programmes plus puissants et flexibles. Utilisez les bonnes pratiques pour éviter les erreurs et optimiser votre code.

4. Manipulation des listes en Python

Les listes sont l'un des types de données les plus utilisés en Python. Elles permettent de stocker une collection ordonnée et modifiable d'éléments. Les listes sont très flexibles et offrent de nombreuses méthodes pour ajouter, supprimer et modifier des éléments.

4.1. Création d'une liste

Une liste est définie en entourant les éléments par des crochets [] et en les séparant par des virgules.

Exemple :

```
# Liste vide
liste_vide = []
# Liste de nombres
nombres = [1, 2, 3, 4, 5]
# Liste de chaînes de caractères
fruits = ["pomme", "banane", "cerise"]
# Liste mixte
mixte = [10, "texte", 3.14, True]
```

4.2. Ajout d'éléments

Il existe plusieurs méthodes pour ajouter des éléments à une liste.

4.2.1. Ajouter un élément à la fin (append())

La méthode append() ajoute un élément à la fin de la liste.
Exemple :

```
fruits = ["pomme", "banane"]
fruits.append("orange")
print(fruits)  # ['pomme', 'banane', 'orange']
```

4.2.2. Insérer un élément à un indice précis (insert())

La méthode insert() permet d'ajouter un élément à un index spécifique.
Exemple :

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

4.2.3. Ajouter plusieurs éléments (extend())

Fusionne une autre liste ou une séquence à la fin de la liste actuelle.

```
fruits = ["pomme", "banane"]
fruits.extend(["orange", "kiwi"])
print(fruits)  # ['pomme', 'banane', 'orange', 'kiwi']
```

4.3. Suppression d'éléments

Il existe plusieurs méthodes pour supprimer des éléments d'une liste.

4.3.1. Supprimer un élément par sa valeur (remove())

La méthode remove() supprime la première occurrence d'un élément spécifié.
Exemple :

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

Si la valeur n'existe pas, une erreur est levée.

4.3.2. Supprimer un élément par son index (pop())

Supprime et retourne un élément à une position donnée.
Exemple :

```
fruits = ["pomme", "banane", "cerise"]
element = fruits.pop(1)  # Supprime 'banane'
print(element)  # 'banane'
print(fruits)  # ['pomme', 'cerise']
```

Si aucun index n'est donné, supprime le dernier élément.

```
fruits.pop()
```

4.3.3. Supprimer un élément avec del

```
fruits = ["pomme", "banane", "cerise"]
del fruits[1]  # Supprime 'banane'
print(fruits)  # ['pomme', 'cerise']
```

4.3.4. Vider complètement la liste (clear())

```
fruits.clear()
print(fruits)  # []
```

4.4. Modification d'éléments

4.4.1. Modifier un élément par son index

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

4.4.2. Modifier plusieurs éléments

```
nombres = [1, 2, 3, 4, 5]
nombres[1:3] = [20, 30]  # Remplace les indices 1 et 2
print(nombres)  # [1, 20, 30, 4, 5]
```

4.4.3. Remplacer plusieurs éléments par une autre liste

```
nombres = [1, 2, 3, 4, 5]
nombres[1:4] = [100, 200]
print(nombres)  # [1, 100, 200, 5] (remplace 2, 3, 4)
```

Les listes en Python sont des structures de données puissantes et flexibles. En maîtrisant les méthodes d'ajout, de suppression et de modification, vous pouvez manipuler efficacement des collections d'éléments. Ces compétences sont essentielles pour résoudre des problèmes complexes et créer des programmes robustes.

5. Présentation de la notion de système d'exploitation

Le système d'exploitation (OS) est un logiciel central qui gère les ressources matérielles et logicielles d'un ordinateur. Il sert d'intermédiaire entre l'utilisateur, les applications et le matériel. Voici une explication détaillée de ses rôles, de son fonctionnement, des types de systèmes d'exploitation, et de la gestion des priorités.

5.1. Rôle du système d'exploitation

Le système d'exploitation permet de :

- Gérer les **ressources matérielles** (processeur, mémoire, disque, périphériques).
- Assurer la **communication entre les logiciels et le matériel**.
- Gérer **l'exécution des programmes** et des processus.
- Administrer les **utilisateurs et leurs droits**.
- Offrir une **interface utilisateur** (graphique ou en ligne de commande).
- Garantir la **sécurité des données** et des accès.

5.2. Prise en charge des travaux

Un OS organise et exécute les tâches demandées par l'utilisateur et les logiciels. Il prend en charge :

Gestion des processus

- Exécuter plusieurs programmes en **multitâche**.
- Assurer la **priorisation** des processus.
- Gérer le **passage entre les tâches** (changement de contexte).

Gestion de la mémoire

- Allouer de la mémoire aux processus en cours.
- Libérer la mémoire des programmes terminés.
- Utiliser la **mémoire virtuelle** pour optimiser l'usage de la RAM.

Gestion des fichiers

- Organiser le stockage des fichiers sur les disques.
- Gérer les droits d'accès aux fichiers.
- Assurer la **lecture, écriture, suppression** des fichiers.

Gestion des périphériques

- Communiquer avec les **claviers, souris, imprimantes, écrans, etc.**
- Installer et gérer les **pilotes** pour chaque matériel.

5.3. Types de systèmes d'exploitation

Les principaux types d'OS sont :

Type	Description
Monotâche	Exécute un seul programme à la fois (ex. MS-DOS).
Multitâche	Exécute plusieurs programmes simultanément (ex. Windows, Linux).
Monoutilisateur	Un seul utilisateur à la fois (ex. Windows 98).
Multiutilisateur	Plusieurs utilisateurs simultanés (ex. Linux, Windows Server).
Temps réel	Réagit instantanément aux événements (ex. OS pour avions, robots).

5.4. Systèmes d'exploitation : Linux vs Windows

5.4.1. Linux

- **Open-source** et gratuit.
- Sécurisé et stable.
- Personnalisable.
- Utilisé pour les **serveurs, développeurs, supercalculateurs**.
- Plusieurs distributions : Ubuntu, Debian, Fedora, Arch Linux.

5.4.2. Windows

- **Propriétaire** et payant.
- Facile à utiliser, interface graphique intuitive.
- Large compatibilité avec les logiciels.
- Moins sécurisé que Linux.
- Utilisé pour **bureautique, jeux, entreprise**.

5.5. Gestion des priorités

Le système d'exploitation attribue un **niveau de priorité** aux processus pour optimiser l'utilisation du processeur.

Méthodes courantes

- **FIFO (First In, First Out)** : Exécute les processus dans l'ordre d'arrivée.
- **Round Robin** : Accorde un temps limité à chaque processus avant de passer au suivant.
- **Priorité statique** : Certains processus ont toujours une priorité plus élevée.
- **Priorité dynamique** : Le système ajuste les priorités en fonction de l'utilisation.

5.6. Autres concepts importants

Gestion des utilisateurs

- Différents niveaux d'accès (administrateur, utilisateur standard, invité).
- Sécurisation par mots de passe, permissions, restrictions.

Sécurité et gestion des erreurs

- Antivirus, pare-feu, mises à jour de sécurité.
- Gestion des crashes système et récupération.

Interfaces utilisateur

- Interface Graphique (GUI) : Fenêtres, icônes, menus (ex. Windows, macOS).
- Interface en ligne de commande (CLI) : Saisie de commandes (ex. Terminal Linux, CMD Windows).

Le système d'exploitation est le cœur d'un ordinateur, c'est essentiel pour le bon fonctionnement des ordinateurs et la gestion des ressources. Il gère les ressources matérielles, exécute les applications, assure la sécurité et fournit une interface utilisateur. Les systèmes comme Linux et Windows offrent des fonctionnalités différentes, adaptées à des besoins spécifiques. La gestion des priorités et des processus permet d'optimiser les performances et de garantir un fonctionnement fluide. Comprendre ces concepts est essentiel pour utiliser efficacement un ordinateur et développer des applications performantes.

6. Présentation des réseaux informatiques

Les réseaux informatiques sont des systèmes qui permettent à des appareils (ordinateurs, smartphones, serveurs, etc.) de communiquer entre eux. Ils sont essentiels pour partager des ressources, des données et des services. Voici une explication détaillée des principes de fonctionnement, des adresses IP, du DNS et d'Internet.

6.1. Principes de fonctionnement

Un réseau fonctionne en reliant plusieurs équipements via des câbles, le Wi-Fi ou d'autres technologies de communication. Il permet :

- **Le partage de ressources** (imprimantes, fichiers, bases de données).
- **La communication** entre utilisateurs (e-mails, messagerie, appels vidéo).
- **L'accès à Internet** et aux services en ligne.

6.1.1. Types de réseaux

Type de réseau	Description
LAN (Local Area Network)	Réseau local, limité à un bâtiment (ex. entreprise, maison).
MAN (Metropolitan Area Network)	Réseau urbain couvrant une ville.
WAN (Wide Area Network)	Réseau étendu couvrant plusieurs pays (ex. Internet).
WLAN (Wireless LAN)	Réseau local sans fil (Wi-Fi).

6.1.2. Éléments d'un réseau

- **Serveurs** : Stockent et distribuent des données.
- **Clients** : Appareils connectés au réseau (PC, smartphones).
- **Routeurs** : Dirigent le trafic entre réseaux et assurent la connexion à Internet.

- **Switches** : Connectent plusieurs appareils et gèrent le trafic interne.
- **Modems** : Convertissent les signaux Internet (ADSL, fibre optique).

6.2. Adresse IP

Une **adresse IP (Internet Protocol)** identifie un appareil sur un réseau. Elle est unique sur un même réseau.

6.2.1. Types d'adresses IP

Type	Description
IPv4	Format : 192.168.1.1 (4 nombres entre 0 et 255). Limité à environ 4 milliards d'adresses.
IPv6	Format : 2001:0db8:85a3::8a2e:0370:7334. Plus d'adresses disponibles.

IP privée vs IP publique

- **IP privée** : Utilisée à l'intérieur d'un réseau local (ex. 192.168.1.1).
- **IP publique** : Utilisée sur Internet, attribuée par un **FAI (Fournisseur d'Accès Internet)**.

Attribution des IP

- **Statique** : Adresse IP fixe.
- **Dynamique** : IP attribuée temporairement par un serveur **DHCP**.

6.3. DNS (Domain Name System)

Le **DNS** traduit un nom de domaine (ex. google.com) en adresse IP.

Fonctionnement

1. L'utilisateur tape un nom de domaine dans son navigateur.
2. Le DNS recherche l'adresse IP associée.
3. L'ordinateur se connecte au serveur via cette IP.

6.3.1. Types de serveurs DNS

- **Serveurs racine** : Gèrent les extensions (.com, .org, .fr).
- **Serveurs TLD (Top-Level Domain)** : Gèrent des domaines spécifiques (ex. .fr, .com).
- **Serveurs de noms** : Stockent les correspondances noms-IP.

Exemple de résolution DNS

google.com → DNS → 142.250.190.78 → Connexion au site

6.4. Internet

Internet est un **réseau mondial** reliant des millions de réseaux locaux. Il permet l'échange d'informations et l'accès aux services en ligne.

6.4.1. Principaux services d'Internet

Service	Description
Web (WWW)	Pages accessibles via un navigateur (ex. Google, YouTube).
E-mail	Communication par messagerie électronique (ex. Gmail, Outlook).
FTP (File Transfer Protocol)	Transfert de fichiers entre machines.
VoIP (Voice over IP)	Appels vocaux via Internet (ex. Skype, Zoom).
Cloud Computing	Stockage et applications en ligne (ex. Google Drive, AWS).

6.4.2. Connexion à Internet

- **Fournisseurs d'Accès Internet (FAI)** : Orange, Free, SFR…
- **Technologies** : Fibre optique, ADSL, 4G/5G, satellite.

Les réseaux informatiques sont indispensables à la communication moderne. Comprendre les principes de fonctionnement, les adresses IP, le DNS et Internet est essentiel pour naviguer dans le monde numérique. Ces concepts permettent de connecter des appareils, d'accéder à des services en ligne et de partager des informations à l'échelle mondiale.

7. Conclusion

Ce chapitre permet aux étudiants de se familiariser avec les concepts fondamentaux de l'informatique et de la programmation, tout en découvrant les systèmes d'exploitation et les réseaux. Les séances de cours et de TP sont conçues pour être interactives et pratiques, afin de faciliter l'acquisition des compétences.

Chapitre 02 : Langage de programmation Python

Python est un langage de programmation interprété, simple à lire et à écrire. Il a été créé à la fin des années 1980 par Guido van Rossum. Le langage de programmation Python est utilisé pour le développement web, l'analyse de données, l'intelligence artificielle, l'automatisation, les scripts, etc. leur syntaxe claire, ce qui le rend facile à apprendre. Il fonctionne sur plusieurs plateformes (Windows, macOS, Linux). Il est très riche par des grandes bibliothèques standards et de nombreux modules externes. Enfin, prise en charge de la programmation orientée objet, fonctionnelle et impérative.

1. La syntaxe de base

La syntaxe de base en Python désigne l'ensemble des règles qui définissent comment écrire correctement des instructions dans ce langage.
Elle impose un style simple, lisible et structuré par l'indentation.
Éléments principaux de la syntaxe

- Les instructions s'écrivent ligne par ligne.
- L'indentation (espaces au début de la ligne) remplace les blocs avec des accolades.
- Les commentaires commencent par #.
- Les chaînes de caractères sont délimitées par des guillemets simples ou doubles.
- Il n'est pas nécessaire de déclarer le type d'une variable.
- Les blocs comme if, for, while, def se terminent par : et utilisent une indentation.

1.1 Variables et types de variables

Une variable est un nom donné à une valeur stockée en mémoire. Elle permet de garder et de manipuler des données pendant l'exécution d'un programme. En Python, les types de variables sont déterminés dynamiquement.

- La création se fait par simple affectation (`nom = valeur`).
- Le type de la variable est automatiquement détecté.
- Une variable peut changer de type en cours d'utilisation.

Les types courants de variables

- int : nombres entiers → age = 25
- float : nombres décimaux → prix = 19.99
- str : chaînes de caractères → nom = "Ali"
- bool : valeurs logiques → actif = True

Exemples :

```
age = 25          # Entier (int)
prix = 19.99      # Flottant (float)
nom = "Ali"       # Chaîne de caractères (str)
est_majeur = True # Booléen (bool)
```

1.2 Opérations et opérateurs

Les opérations permettent de manipuler les données.
Les opérateurs sont les symboles utilisés pour effectuer ces opérations.
Les opérateurs arithmétiques : Ils servent à faire des calculs.

- `+` addition
- `-` soustraction
- `*` multiplication
- `/` division
- `%` reste (modulo)
- `**` puissance

Opérateurs de comparaison : Ils comparent deux valeurs et donnent un résultat booléen.

- == égal
- != différent
- > plus grand
- < plus petit
- >= plus grand ou égal
- <= plus petit ou égal

Les opérateurs logiques : Ils combinent des conditions.

- `and` vrai si les deux sont vraies
- `or` vrai si au moins une est vraie
- `not` inverse le résultat

Exemple :

```
a = 10
b = 20
print(a + b)  # Addition : 30
print(a > b)  # Comparaison : False
```

2. Structures de contrôle

Les structures de contrôle permettent de diriger l'exécution du programme. Elles définissent quel code doit s'exécuter, quand et combien de fois.

2.1 Les structures conditionnelles

Elles permettent d'exécuter un bloc de code si une condition est vraie.

2.1.1 Structure : if, elif, else.

Exemple :

```
age = 18
if age >= 18:
    print("Majeur")
else:
    print("Mineur")
```

2.2 Les Boucles

Elles permettent de répéter des instructions.
La boucle for : répète pour chaque élément d'une séquence. Pour itérer sur une séquence (liste, chaîne, etc.).

```
for i in range(5):
    print(i)  # Affiche 0, 1, 2, 3, 4
```

La boucle while : Répète tant qu'une condition est vraie.

```
x = 5
while x > 0:
      print(x)
      x -= 1  # Affiche 5, 4, 3, 2, 1
```

3. Les chaînes de caractères

Une chaîne de caractères est une séquence de caractères entourée de guillemets (" ou ').

Fonctions utiles

- len(chaine) : donne la longueur
- chaine.upper() : transforme en majuscules
- chaine.lower() : transforme en minuscules
- chaine.replace("a", "b") : remplace
- chaine[0] : accède à un caractère

Exemples :

```
nom = "Alice"
print(nom[0])       # Affiche 'A' (premier caractère)
print(nom.upper())  # Affiche 'ALICE'
print(len(nom))     # Affiche 5 (longueur de la chaîne)
```

4. Les listes

Une liste est une collection ordonnée qui peut contenir plusieurs éléments. Elle peut contenir des nombres, des chaînes, des booléens ou d'autres listes.

Caractéristiques des listes

- Définies entre crochets []
- Les éléments sont séparés par des virgules
- Modifiables (on peut ajouter, changer ou supprimer des éléments)
- Indexées à partir de 0

Fonctions courantes

- append(valeur) : ajoute à la fin
- remove(valeur) : supprime la première occurrence
- insert(index, valeur) : insère à une position
- pop() : enlève le dernier élément
- len(liste) : donne le nombre d'éléments
- liste[index] : accède à un élément

Exemples :

```
fruits = ["pomme", "banane", "cerise"]
print(fruits[1])       # Affiche 'banane'
fruits.append("orange") # Ajoute 'orange' à la fin
fruits.remove("banane") # Supprime 'banane'
```

5. Les fichiers

Un fichier est un espace de stockage utilisé pour enregistrer des données sur le disque. Python permet de lire, écrire ou modifier ces fichiers à l'aide de fonctions intégrées.

Types d'accès

- "r" : lecture
- "w" : écriture (efface le contenu existant)
- "a" : ajout à la fin
- "r+" : lecture et écriture

Fonction de base

- open(nom_fichier, mode) : ouvre le fichier
- read() : lit tout le contenu
- readline() : lit une ligne
- readlines() : lit toutes les lignes dans une liste
- write() : écrit du texte
- close() : ferme le fichier (automatique avec with)

5.1 Lecture

```
# Lecture
with open("fichier.txt", "r") as f:
    contenu = f.read()
# Écriture
with open("fichier.txt", "w") as f:
    f.write("Bonjour")
# Ajout
with open("fichier.txt", "a") as f:
    f.write("\nNouvelle ligne")
```

6. Les fonctions et les modules

6.1 Les fonctions

Une fonction est un bloc de code qui effectue une tâche précise et qu'on peut réutiliser.
Caractéristiques

- Déclarées avec **def**
- Peuvent recevoir des paramètres
- Peuvent renvoyer une valeur avec return

Exemple :

```
def addition(a, b):
      return a + b
print(addition(3, 5))  # Affiche 8
```

6.2 Les modules

Un module est un fichier contenant des fonctions, des classes ou des variables qu'on peut importer.
Les types des modules :

- **Modules intégrés** (déjà inclus avec Python) : math, random, os, etc.

```
import math # Fonctions mathématiques.
print(math.sqrt(16))  # Affiche 4.0
```

```
import random # Génération de nombres aléatoires
print(random.randint(1, 10))  # Affiche un nombre entre 1 et 10
```

- **Modules externes** (à installer) : pandas, numpy, etc.
- **Modules personnalisés** (fichiers .py créés par vous)

7. Les bibliothèques Pandas, NumPy, etc.

Une bibliothèque est un ensemble de fonctions et de classes prêtes à l'emploi. Elles permettent de gagner du temps et d'éviter de tout programmer soi-même.

7.1 Pandas

Pandas est une bibliothèque pour l'analyse de données. Elle permet de créer, lire, modifier et analyser des tableaux appelés DataFrame.
Exemple :

```
import pandas as pd
data = {"Nom": ["Alice", "Bob"], "Âge": [25, 30]}
df = pd.DataFrame(data)
print(df)
```

7.2 NumPy

NumPy est une bibliothèque pour le calcul numérique. Elle est surtout utilisée pour les tableaux, les vecteurs et les opérations mathématiques rapides.
Exemple :

```
import numpy as np
tableau = np.array([1, 2, 3])
print(tableau * 2)  # Affiche [2, 4, 6]
```

Autres bibliothèques utiles

- `matplotlib` : pour les graphiques
- `scikit-learn` : pour l'apprentissage automatique
- `seaborn` : pour les visualisations avancées

8. Introduction à la programmation objet

La programmation orientée objet (POO) permet de structurer le code en objets.
Concepts clés :

- **Classe** : Modèle pour créer des objets.
- **Objet** : Instance d'une classe.
- **Attributs** : Variables d'un objet.
- **Méthodes** : Fonctions d'un objet.

Exemple :

```
class Personne:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

    def se_presenter(self):
        print(f"Je m'appelle {self.nom} et j'ai {self.age} ans.")

p = Personne("Alice", 25)
p.se_presenter()  # Affiche "Je m'appelle Alice et j'ai 25 ans."
```

9. Lire et traiter des fichiers textes

9.1 Recherche

Rechercher une chaîne dans un fichier :

```
with open("fichier.txt", "r") as f:
    for ligne in f:
        if "mot-clé" in ligne:
            print(ligne)
```

9.2 Tri

Trier les lignes d'un fichier :

```
with open("fichier.txt", "r") as f:
    lignes = f.readlines()
    lignes.sort()
    for ligne in lignes:
        print(ligne)
```

10. Gérer des rapports simples (PDF, Excel)

Gérer des rapports simples signifie créer ou modifier automatiquement des documents comme des fichiers PDF ou Excel à l'aide de Python. Cela permet de générer des fiches, tableaux ou bilans sans traitement manuel.

10.1 Générer un PDF

Un fichier PDF peut être généré avec du texte, des titres ou des lignes. La bibliothèque **reportlab** permet d'écrire du contenu directement dans un document.
Utiliser la bibliothèque **reportlab** :

```
from reportlab.pdfgen import canvas
c = canvas.Canvas("rapport.pdf")
c.drawString(100, 750, "Bonjour, monde !")
c.save()
```

10.2 Manipuler des fichiers Excel

Un fichier Excel peut contenir des données organisées en lignes et colonnes. La bibliothèque openpyxl permet de créer des feuilles, remplir des cellules, trier ou enregistrer des valeurs.
Utiliser la bibliothèque **openpyxl** :

```
from openpyxl import Workbook
wb = Workbook()
ws = wb.active
ws["A1"] = "Nom"
ws["B1"] = "Âge"
wb.save("rapport.xlsx")
```

Chapitre 03 : Programmation et automatisation avec Python

1. Principes d'automatisation de tâches

L'automatisation consiste à utiliser des scripts pour exécuter des tâches répétitives ou complexes de manière automatisée. Python est un langage idéal pour cela grâce à sa simplicité et à ses nombreuses bibliothèques.
Exemples sur les tâches qu'on peut automatiser :

- Lire et modifier des fichiers
- Extraire des données d'un site web
- Envoyer des e-mails automatiquement
- Générer des rapports (PDF, Excel)
- Organiser des dossiers et fichiers
- Planifier des scripts pour qu'ils s'exécutent seuls

Pourquoi Python est adapté à l'automatisation :

- Syntaxe simple
- Bibliothèques prêtes à l'emploi
- Grande communauté et documentation
- Compatible avec Windows, Linux, macOS

2. Bibliothèques Python pour l'automatisation

Python propose plusieurs bibliothèques pour automatiser des tâches sans effort manuel.

2.1 os et shutil

Gérer les fichiers et dossiers (Créer, supprimer, déplacer, renommer)

- **os** : Pour interagir avec le système d'exploitation.
- **shutil** : Pour manipuler des fichiers et des dossiers (copie, déplacement, suppression).

```
import os, shutil
os.rename("ancien.txt", "nouveau.txt")
shutil.copy("source.txt", "copie.txt")
```

2.2 datetime

Gérer les dates et les heures.
Planifier une exécution, générer des noms de fichiers par date
Exemple :

```
from datetime import datetime
print(datetime.now())
```

2.3 schedule et time

Lancer des tâches à des moments précis.
Exemple :

```
import schedule, time
def dire_bonjour():
    print("Bonjour")
schedule.every(10).seconds.do(dire_bonjour)
while True:
    schedule.run_pending()
    time.sleep(1)
```

2.4 smtplib

Envoyer des e-mails automatiquement.
Exemple :

```
import smtplib
serveur = smtplib.SMTP("smtp.example.com", 587)
serveur.starttls()
serveur.login("adresse@example.com", "motdepasse")
serveur.sendmail("adresse@example.com", "destinataire@example.com", "Message")
serveur.quit()
```

2.5 openpyxl, reportlab, pandas

Créer ou modifier des fichiers Excel, PDF, CSV.

2.6 selenium, requests, beautifulsoup4

Naviguer sur internet, remplir des formulaires, extraire des données.

3. Travail avec des fichiers Excel ou CSV

Travailler avec des fichiers Excel ou CSV signifie lire, modifier, créer ou analyser des tableaux de données à l'aide de Python.
Un fichier CSV contient des données organisées sous forme de texte, séparées par des virgules. Il est simple à lire, léger et compatible avec de nombreux logiciels.
Un fichier Excel est structuré en feuilles, lignes et colonnes. Il permet de stocker des données, d'ajouter des formules, des formats ou des graphiques.
Pourquoi utiliser Python

- Lire ou écrire automatiquement des fichiers
- Trier ou filtrer des données
- Générer des rapports à partir de données
- Gagner du temps sur des traitements répétitifs
- **openpyxl** : Pour manipuler des fichiers Excel.
- **pandas** : Pour manipuler des données tabulaires (Excel, CSV, etc.).

Exemples d'automatisation

- **Envoi de mails** : Utiliser la bibliothèque smtplib.
- **Tâches planifiées** : Utiliser schedule ou cron (sur Linux).

3.1. Manipulation de fichiers avec Python

Python offre des fonctionnalités puissantes pour manipuler des fichiers, qu'il s'agisse de fichiers texte, CSV, Excel, ou autres. Voici les opérations les plus courantes.

3.1.1. Parcourir un dossier

Utiliser **os.listdir** pour lister les fichiers d'un dossier.

```
import os
fichiers = os.listdir("chemin/du/dossier")
for fichier in fichiers:
    print(fichier)
```

3.1.2. Vérifier l'existence d'un fichier ou dossier

Utiliser os.path.exists.

```
if os.path.exists("chemin/du/fichier.txt"):
        print("Le fichier existe.")
else:
        print("Le fichier n'existe pas.")
```

Créer ou supprimer des dossiers

Créer un dossier : os.mkdir.

```
os.mkdir("nouveau_dossier")
```

Supprimer un dossier : os.rmdir.

```
os.rmdir("dossier_a_supprimer")
```

Copier ou déplacer des fichiers

Copier un fichier : shutil.copy.

```
import shutil
shutil.copy("source.txt", "destination.txt")
```

Déplacer un fichier : shutil.move.

```
shutil.move("source.txt", "nouvel_emplacement/source.txt")
```

4. Visualisation de données

Matplotlib

Pour créer des graphiques simples.

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.show()
```

Seaborn

Pour des visualisations statistiques avancées.

```
import seaborn as sns
sns.lineplot(x=[1, 2, 3, 4], y=[10, 20, 25, 30])
plt.show()
```

Plotly

Pour des graphiques interactifs.

```
import plotly.express as px
fig = px.line(x=[1, 2, 3, 4], y=[10, 20, 25, 30])
fig.show()
```

5. Interaction avec des API

Requêtes HTTP avec requests

Pour interagir avec des API web.

```
import requests
response = requests.get("https://api.github.com")
print(response.json())  # Affiche la réponse en JSON
```

Scraping de données avec BeautifulSoup

Pour extraire des données de pages web.

```
from bs4 import BeautifulSoup
import requests
url = "https://exemple.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")
print(soup.title.text)  # Affiche le titre de la page
```

6. Interfaces graphiques

Tkinter

Pour créer des interfaces graphiques simples.

```
import tkinter as tk
fenetre = tk.Tk()
fenetre.title("Mon application")
label = tk.Label(fenetre, text="Bonjour, monde !")
label.pack()
fenetre.mainloop()
```

PyQt

Pour des interfaces graphiques plus avancées.

```
from PyQt5.QtWidgets import QApplication, QLabel
app = QApplication([])
fenetre = QLabel("Bonjour, monde !")
fenetre.show()
app.exec_()
```

7. Exemples pratiques d'automatisation

7.1. Envoi de mails

Utiliser smtplib pour envoyer des mails.

```
import smtplib
from email.mime.text import MIMEText

msg = MIMEText("Ceci est un mail automatique.")
msg["Subject"] = "Sujet du mail"
msg["From"] = "expediteur@exemple.com"
msg["To"] = "destinataire@exemple.com"

with smtplib.SMTP("smtp.exemple.com") as server:
    server.login("utilisateur", "motdepasse")
    server.sendmail("expediteur@exemple.com", "destinataire@exemple.com", msg.as_string())
```

7.2. Manipulation de fichiers Excel

Utiliser pandas pour lire et écrire des fichiers Excel.

```
import pandas as pd

# Lire un fichier Excel
df = pd.read_excel("fichier.xlsx")

# Écrire dans un fichier Excel
df.to_excel("nouveau_fichier.xlsx", index=False)
```

7.3. Tâches planifiées

Utiliser schedule pour planifier des tâches.

```
import schedule
import time
def tâche():
    print("Tâche exécutée !")
schedule.every(10).seconds.do(tâche)
while True:
    schedule.run_pending()
time.sleep(1)
```

Chapitre 04 : Apprentissage avancé d'Excel

1. Principes des macros et création d'une macro simple

Une **macro** est un ensemble d'instructions automatisées qui permettent d'exécuter des tâches répétitives dans Excel. Les macros sont écrites en **VBA (Visual Basic for Applications)**, un langage de programmation intégré à Excel.

1.1. Créer une macro simple

1. Activer l'onglet **Développeur** :
   Aller dans **Fichier > Options > Personnaliser le ruban**.
   Cocher **Développeur** dans la colonne de droite.
2. Enregistrer une macro :
   Cliquer sur **Développeur > Enregistrer une macro**.
   Donner un nom à la macro (ex: MacroTest).
   Effectuer les actions à automatiser (ex: mettre en gras une cellule).
   Cliquer sur **Arrêter l'enregistrement**.
3. Exécuter la macro :
   Cliquer sur **Développeur > Macros**.
   Sélectionner la macro et cliquer sur **Exécuter**.

Exemple de code VBA

- Ouvrir l'éditeur VBA avec **Alt + F11**.
- Ajouter un module et écrire du code VBA :

```
Sub MacroTest()
    Range("A1").Font.Bold = True
    Range("A1").Value = "Bonjour, monde !"
End Sub
```

2. Tableaux croisés dynamiques

Qu'est-ce qu'un tableau croisé dynamique (TCD) ?
Un **TCD** permet de résumer, analyser et présenter des données de manière interactive.
Créer un TCD

1. Sélectionner les données à analyser.
2. Aller dans **Insertion > Tableau croisé dynamique**.
3. Choisir l'emplacement du TCD (nouvelle feuille ou emplacement existant).
4. Glisser les champs dans les zones **Lignes**, **Colonnes**, **Valeurs** et **Filtres**.

Exemple

- Données : Une liste de ventes avec les colonnes **Date**, **Produit**, **Quantité**, **Prix**.
- TCD : Résumer les ventes par produit et par mois.

3. Histogrammes

Qu'est-ce qu'un histogramme ?

- Un **histogramme** est un graphique qui montre la distribution des données en regroupant les valeurs en intervalles (bins).

Créer un histogramme

1. Sélectionner les données.
2. Aller dans **Insertion > Graphique statistique > Histogramme**.

3. Personnaliser l'histogramme (intervalles, couleurs, etc.).

Exemple

- Données : Une liste de notes d'étudiants.
- Histogramme : Visualiser la distribution des notes (ex: 0-10, 10-20, etc.).

4. Diagrammes en barres

Qu'est-ce qu'un diagramme en barres ?

- Un **diagramme en barres** compare des valeurs à l'aide de barres horizontales ou verticales.

Créer un diagramme en barres

1. Sélectionner les données.
2. Aller dans **Insertion > Graphique à barres**.
3. Choisir le type de diagramme (barres verticales ou horizontales).

Exemple

- Données : Ventes mensuelles de produits.
- Diagramme : Comparer les ventes de chaque produit par mois.

5. Diagrammes en araignée (Radar)

Qu'est-ce qu'un diagramme en araignée ?

- Un **diagramme en araignée** (ou radar) permet de comparer plusieurs variables sur un graphique circulaire.

Créer un diagramme en araignée

1. Sélectionner les données.
2. Aller dans **Insertion > Graphique radar**.
3. Personnaliser le graphique (couleurs, légendes, etc.).

Exemple

- Données : Performances d'un employé dans différentes compétences (ex: communication, technique, gestion).
- Diagramme : Visualiser les forces et faiblesses.

6. Autres graphiques avancés

Graphiques en courbes

- Pour visualiser des tendances sur une période.
- **Insertion > Graphique en courbes**.

Graphiques en secteurs (camembert)

- Pour montrer les proportions d'un tout.
- **Insertion > Graphique en secteurs**.

Graphiques en nuage de points

- Pour analyser les relations entre deux variables.
- **Insertion > Graphique en nuage de points**.

7. Bonnes pratiques pour les graphiques

- **Titres** : Ajouter un titre clair et descriptif.
- **Légendes** : Expliquer les couleurs et les séries.
- **Étiquettes** : Afficher les valeurs directement sur le graphique.
- **Mise en forme** : Utiliser des couleurs contrastées et une police lisible.

Chapitre 05 : Apprentissage de GanttProject

1. Introduction et prise en main de GanttProject

1.1. Introduction à la gestion de projets

Qu'est-ce qu'un projet ?

- Un projet est un ensemble d'activités temporaires et organisées pour atteindre un objectif spécifique dans un délai et avec des ressources définis.
- Exemples de projets : Construction d'un bâtiment, lancement d'un produit, organisation d'un événement.

Quels sont les enjeux de gestion d'un projet ?

- **Planification** : Définir les étapes, les délais et les ressources.
- **Suivi** : Contrôler l'avancement et ajuster les plans si nécessaires.
- **Communication** : Assurer une bonne coordination entre les parties prenantes.
- **Risques** : Identifier et gérer les risques potentiels.

Pourquoi utiliser GanttProject ?

- GanttProject est un outil simple et gratuit pour créer des diagrammes de Gantt, gérer les tâches et les ressources.
- Il est idéal pour les petits et moyens projets.

1.2. Interface de GanttProject

Installation

- Télécharger GanttProject depuis le site officiel : GanttProject.
- Installer le logiciel sur votre ordinateur.

Présentation de l'interface

- **Barre de menus** : Fichier, Édition, Affichage, Projet, etc.
- **Barre d'outils** : Boutons pour ajouter des tâches, des ressources, etc.
- **Vue Gantt** : Diagramme de Gantt pour visualiser les tâches et les délais.
- **Vue Ressources** : Liste des ressources (personnes, équipements) affectées aux tâches.

1.3. Les tâches

Création de tâches

1. Cliquer sur **Ajouter une tâche** dans la barre d'outils.
2. Saisir le nom de la tâche (ex: "Conception du site web").
3. Définir la durée de la tâche (ex: 5 jours).

Modification de tâches

- Double-cliquer sur une tâche pour modifier ses propriétés :
  - **Nom** : Modifier le nom de la tâche.
  - **Durée** : Ajuster la durée.
  - **Dépendances** : Définir les tâches précédentes (prérequis).

Organisation des tâches

- **Hiérarchie** : Créer des sous-tâches pour organiser les tâches en phases.
  - Exemple : Tâche principale "Développement" avec des sous-tâches "Front-end", "Back-end".
- **Glisser-déposer** : Réorganiser les tâches dans le diagramme de Gantt.

2. Gestion du temps et des ressources

2.1. Gestion du temps

Dates de début et de fin de projet

- Définir la date de début du projet dans **Projet > Propriétés du projet**.
- La date de fin est calculée automatiquement en fonction des tâches.

Dépendances entre tâches

- Relier les tâches pour définir l'ordre d'exécution.
  - Exemple : La tâche "Tests" ne peut commencer que lorsque la tâche "Développement" est terminée.
- Pour créer une dépendance, cliquer sur une tâche et la relier à une autre avec la flèche dans le diagramme de Gantt.

Jalons (Milestones)

- Les jalons marquent des événements clés dans le projet (ex: livraison, réunion).
- Créer un jalon en définissant une tâche avec une durée de 0 jour.

2.2. Gestion des ressources

Création de ressources

1. Aller dans la **Vue Ressources**.
2. Cliquer sur **Ajouter une ressource**.
3. Saisir le nom de la ressource (ex: "Développeur", "Designer").
4. Définir le coût horaire (optionnel).

Affectation des ressources aux tâches

1. Double-cliquer sur une tâche.
2. Aller dans l'onglet **Ressources**.
3. Sélectionner la ressource à affecter.

Suivi de la charge de travail

- Visualiser la charge de travail des ressources dans la **Vue Ressources**.
- Ajuster les affectations pour éviter les surcharges.

3. Exportation et partage du projet

Exporter le diagramme de Gantt

- Aller dans **Fichier > Exporter au format PDF** pour partager le planning.

Exporter le projet

- Exporter le projet au format **.gan** pour le sauvegarder ou le partager avec d'autres utilisateurs de GanttProject.

Exercices pratiques

Exercice 1 : Créer un projet simple

- Créez un projet pour organiser un événement (ex: conférence, mariage).
- Ajoutez au moins 5 tâches avec des dépendances et des jalons.
- Affectez des ressources à chaque tâche.

Exercice 2 : Gérer les ressources

- Créez un projet de développement logiciel.
- Ajoutez des ressources (ex: développeurs, testeurs) et affectez-les aux tâches.
- Ajustez les charges de travail pour équilibrer les ressources.

Références bibliographiques

[01] Ouvrages : Introduction à l'informatique, Principes de la programmation, Gestion de bases de données,)

[02] Duteil-Mougel Carine (2005) : Les mécanismes persuasifs des textes politiques, Corpus N°4. Lien : http://corpus.revues.org/357

[03] J. Guilhaumou (2002) : Le corpus en analyse de discours, perspectives historiques. Corpus N°1. Lien : http://corpus.revues.org/8

[04] Ouvrages classiques sur les variables quantitatives et qualitatives.

[05] E.Schultz et M.Bussonnier (2020) : Python pour les SHS. Introduction à la programmation de données. Presses Universitaires de Rennes.

[06] C.Paroissin, (2021) : Pratique de la data science avec R : arranger, visualiser, analyser et présenter des données. Paris : Ellipses, DL 2021.

[07] S.Balech et C.Benavent : NLP texte minig V4.0, (Paris Dauphine – 12/2019) : lien : https://www.researchgate.net/publication/337744581_NLP_text_mining_V40_une_introduction_-_cours_programme_doctoral

[08] Ganascia, J.Gabriel (2024) : l'IA expliquée aux humains. Paris France- Edition le Seuil.

[09] Anglais, Lise, Dilhac, Antione, Dratwa, Jim et al. (2023) : L'éthique au cœur de l'IA. Quebec Obvia.

[10] J.Robert (2024) : Natural Language Processing (NLP) : définition et principes Datasciences. Lien : https://datascientest.com/introduction-au-nlp-natural-language-processing

[11] Qu'est-ce que le traitement du langage naturel. Lien : https://aws.amazon.com/fr/what-is/nlp/

[12] M.Journe : Eléments de Mathématiques discrètes – Ellipses

[13] F.Challet : L'apprentisage profond avec Python – Eyrolles

[14] H.Bersini (2024) : L'intelligence artificielle en pratique avec Python – Eyrolles

[15] B.Prieur (2024) : Traitement automatique du langage naturel avec Python – Eyrolles

[16] V.Mathivet (2024) : Implémentation en Python avec Scikit-learn – Eyrolles

[17] G.Dubertret (2023) : Initiation à la cryptographie avec Python – Eyrolles

[18] S.Chazallet (2023) : Python 3 – Les fondamentaux du langage - Eyrolles

[19] H.Belhadef, I.Djemal : Méthode TALN – Cours de l'unievrsité de Msila - Algérie

[20] Documentation officielle Python : Python.org / Bibliothèque os / Bibliothèque shutil

[21] Tutoriels Python : W3Schools Python / Real Python

[22] Bibliothèques populaires : NumPy : numpy.org / Pandas : pandas.pydata.org / ReportLab : reportlab.com / OpenPyXL : openpyxl.readthedocs.io / Matplotlib : matplotlib.org / Seaborn : seaborn.pydata.org / Plotly : plotly.com / Requests : docs.python-requests.org / BeautifulSoup : www.crummy.com/software/BeautifulSoup/ / Tkinter : tkdocs.com / PyQt : riverbankcomputing.com

[23] Programmation orientée objet : Livre : "Python 3 - Les fondamentaux du langage" de Sébastien Chazallet.

[24] Documentation officielle Microsoft : Support Microsoft Excel / Guide des graphiques Excel

[25] Tutoriels vidéo : YouTube : Tutoriels Excel avancés / LinkedIn Learning : Excel avancé / YouTube : Tutoriels GanttProject

[26] "Excel 2019 pour les Nuls" de Greg Harvey

[27] "Excel avancé : Techniques et macros" de Pierre Rigollet

[28] Formations en ligne : Coursera : Excel Skills for Business / Udemy : Excel avancé / Coursera : Gestion de projets / Udemy : Gestion de projets avec GanttProject.

[29] Formations en ligne :

[30] Documentation officielle de GanttProject : Guide utilisateur GanttProject.
[31] "Le Chef de projet efficace" de Michel Barabel et Olivier Meier.
[32] "Gestion de projets pour les Nuls" de Stanley E. Portny.