# **EPIGRAPHE**

Quand je marche dans la vallée de l’ombre de la mort, je ne crains aucun mal car tu es avec moi, ta houlette et ton bâton me rassurent. Tu dresses devant moi une table, en face de mes adversaire, tu oins d’huile ma tête et ma coupe déborde.

PSAUMES 23 : 4 - 5

# **DEDICACE**

A mon père **DJOUNA MUKANGU**, toi qui as porté sur tes épaules tant des sacrifices pour que je grandisse dans la dignité, toi qui as tout donné pour me voir avancer, ton courage, ton amour, ta patience et ton soutien ont bâti les fondations de mon avenir, c’est grâce à toi que j’ai appris la valeur de l’effort et de la persévérance. Ce travail est pour toi papa, je te le dédie du fond du cœur

A ma mère **CLEMENCIA KABUNDU MUKANGU** tu es mon pilier, mon refuge et mon inspiration depuis ma naissance tu es restée à me cotés toujours présente, toujours forte et aimante, tu as tout donné pour me voir arrivé là sans jamais rien attendre en retour. Ce travail je te le dois maman et je te les dédie du fond de mon cœur.

Pace à ma grand-mère, **MARIE KUSEHUKA**, toi qui a toujours prié pour moi, veillé sur moi avec douceur et sagesse. Ton amour et tes bénédictions m’accompagnent chaque jour. Ce travail est aussi pour toi.

**MUKANGU MALAMBA Marleine**

# **REMERCIEMENTS**

Nos remerciement le plus sincère et considérable à notre directeur ANGOMA MONGA SINDANI Blaise, qui malgré ses nombreuses occupations à accepter de diriger ce travail. Ce conseil et sa rigueur ont permis d’orienter ce travail et de le mener à son terme. Merci infiniment prof

Merci à notre co-directeur MAYI LUKONGO YALEM d’avoir accepté de travailler ensemble avec nous. Merci à vous CT DANNY KAMBERE, CT MAFUKU. Nous remercions le corps académiques et scientifique de L’ISPT/KIN pour leur volonté et leur apport durant quatre ans pour notre formation impeccable.

A vous ma famille, RACHEL GAMESA, PATRICIA MUKANGU, MOISE GAMESA, SUNAMITH MUKANGU, CHADRACK MUKANGU, PERSIDE MUKANGU, BLESSING MUKANGU et ARDI MUKANGU merci pour votre amour, vos soutiens et vos encouragement tout au long de ce parcours.

Merci à vous aussi mes enfants triomphe MUKANGU, brayan MUKANGU, Rayan MAKANGU, Félicien MUSHIBONGA, oracle NDONGALA, Nephtalie NDONGALA, chance NDONGALA, alora NDONGALA, clémence MUKASSA, judelia MUKASSA, christiania MUKUSSA.

A toi ma mère du home BERTHANIE NTOYA, ton amour m’a enveloppé comme celui d’une vraie. Tu m’as comprise, supporté même dans mes moments les plus difficiles, ton cœur tendre, ta douceur et ta patience resteront gravés à moi, Merci infiniment d’avoir été ce refuge plein d’amour merci.

Merci Jaspe MAKANGALA, Mike MAWETE icha, Christel GASHIMBA, Archileine KALALA, Merveille VANGU, tonton DORA KUBASIKA, henock KABUNDU, MALAMBA NOTE, regina LUTONDO, a toute la famille LULENDO merci beaucoup pour votre amour et soutien. Rudith MUKIDI (dith couture) merci pour les conseils ma grande.

A mes très chers camarades et compagnons de lutte Bénie LULENDO, plamedie NSELELE, plamedi kalema, MULA Kap’s, Joas NGOLOMINGI, Phoebe KIBONGE, Prisca KHETE, exaucé NGITUKA, ben LASANGA pour l’amour ainsi que la solidarité estudiantine durant notre parcours académiques.

Merci à tous nos collègues avec qui nous avons entretenus des bonnes relations durant nôtres passages à L’ISPT-KIN et partout ailleurs. Que ceux dont les noms n’ont pas été cités, trouve ici l’expression de notre profondes gratitude.

**MUKANGU MALAMBA Marleine**

# **LISTE DES ABREVIATIONS**

BFS: Breadth-first search

CI/CD: Continuous integration / Continuous deployment

CPU: Central processing unit (le processeur, cerveau de la machine)

CT: chef des travaux

DFS: Depth-first search

GIL: Global interpreter lock

GPU: graphics processing unit

HTTP: Hypertext transfer protocol

IA: Intelligence artificielle

IDE: Integrated development environment

IOT: Internet of things

I/O: Input / Output (entrée/sortie)

ISPT: institut supérieur pédagogique et technique

LLVP: Low-level virtual machine

NLP: Natural language processing

PC: personal computer (ordinateur personnel)

POO: Programmation orientée objet

RAM: Random Access memory (mémoire vive)

RGPD: Règlement général sur la protection des données

SE: Systeme exploitation

SQL: Structured query language

SSD: Solid-state drive

XSS: Cross-site Scripting

# **LISTE DES FIGURES**

FIGURE 3.1 : Code Rust ………………………………………………..P61

FIGURE 3.2 : Code python …………………………………………….P62

# **LISTE DES TABLEAUX**

Tableau 2.1 : Comparaison Rust et python…………………………….P49

Tableau 3.1 : Comparaison d’utilisation mémoire ……………………..P67

Tableau 3.2 : Comparaison d’utilisation de processeur…………………P67

Tableau 3.3 : Comparaison d’efficacité des E/S………………………….P67

# **INTRODUCTION GENERALE**

## **Présentation du sujet**

Dans le monde de l’informatique, le choix d’un langage de programmation est une décision importante qui peut avoir un impact significatif sur les performances d’une application ainsi que sur l’utilisation des ressources systèmes. L’évolution rapide des technologies de l’information a conduit l’émergence d’un grand nombre de langages de programmation.

Avec la multitude de langages disponibles, chacun ayant ces propres caractéristiques, concept et domaines d’application, il est essentiel de comprendre comment ces facteurs influencent l’efficacité des programmes. Cette étude comparative vise à analyser comment les différents langages se comportent dans des contextes variés, notamment en matière de vitesse d’exécution, de gestion de mémoire et d’efficacité des ressources. Les langages de programmation, tels que « c++, java, Rust, python présentes de caractéristique distinctes qui influencent leur adaptation des taches spécifiques, notamment dans le domaine de l’IA et du développement de logiciel en général.

## **0.2 Problématique**

La problématique centrale de cette étude est de déterminer comment les différents langages de programmation affectent les performances des applications et l’utilisation des ressources. Plus précisément, cette initiative nous pousse à nous poser des questions suivantes :

* Quels langages offrent les meilleures performances pour des taches spécifiques, comme le traitement en temps réel ou le calcul intensif ?
* Comment la gestion de la mémoire varie-t-elle entre les langages et quel impact cela a-t-il sur la performance globale ?
* Dans quelle mesure la facilité de développement et la productivité des programmeurs influencent-elles le choix d’un langage, même si cela peut entrainer des compromis en termes de performances ?

## **0.3 Hypothèse**

L’hypothèse de cette étude est que le choix du langage de programmation a un impact significatif sur les performances d’une application et l’utilisation des ressources. En d’autres termes certains langages peuvent offrir des avantages en termes de vitesse et d’efficacité, tandis que d’autres peuvent faciliter le développement et la maintenance.

## **Choix et intérêt du sujet**

### **0.4.1 Choix**

Le choix de cette étude comparative de langages de programmation en termes de performance et l’utilisation des ressources est motivé par la diversité croissante des langages disponibles et leur impact sur le développement de logiciel. Avec l’évolution rapide des technologies, il est important de comprendre comment ces langages se distinguent en matière d’efficacité, de vitesse d’exécution et de gestion des ressources, afin de faire choix éclairés lors du développement d’application.

### **0.4.2 Intérêt**

* **Intérêt personnel** : Sur le plan personnel, cet intérêt découle d’une passion pour la programmation et le désir d’approfondir mes connaissances sur les différents langages. Comprendre les forces et les faiblesses de chaque langage me permettra d’améliorer mes compétences en développement et de choisir les outils les plus adaptés pour mes projets du futur.
* **Intérêt scientifique** : D’un point de vue scientifique, cette étude contribue à la recherche sur l’optimisation des performances de logiciels. En analysant les langages de programmation sous l’angle des performances et l’utilisation des ressources, cette recherche peut fournir des compréhensions précieuses pour les chercheurs et les praticiens, en les aidants à choisir les langages les plus approprié pour des applications spécifiques, notamment dans les domaines de l’IA et le développement web.
* **Intérêt social :** Dans un monde de plus en plus numérique, le choix du langage de programmation peut influencer la qualité des applications utilisées par le grand public. De plus, la compréhension de langages de programmation et de leurs performances peut encourager l’éducation et la formation dans le domaine de l’informatique, contribuant ainsi à la création d’une main d’œuvre qualifiée capable de répondre aux besoins technologiques croissants de la société.
* **Information en temps réel :** l’utilisation de langage adaptés peut également améliorera fiabilité et la stabilité des systèmes d’information, ainsi que les utilisateurs reçoivent des données précises et a jour, ce qui est fondamental dans des situations ou chaque seconde compte.

## **Objectif**

### **Objectif principal**

L’objectif principal de cette étude est d’évaluer et de comparer les performances et l’utilisation des ressources de différents langages de programmation, afin de déterminer lequel est le plus adapté à des tâches spécifiques et à des environnements de développement variés.

### **Objectif spécifique**

* Analyser les performances des langages de programmation dans des scenarios de traitement intensif et de gestion de la mémoire.
* Evaluer l’impact de la syntaxe et des paradigmes de programmation sur la productivité des développeurs.
* Identifier les langages les plus efficaces pour des applications spécifiques, comme le développement web, le traitement de données et l’IA.

## **0.6 Méthodes et technique**

### **O.6.1 Méthode**

* **Méthode historique :** la méthode historique consiste à examiner l’évolution des langages de programmation fil du temps, en analysant comment leurs performances et leur utilisation des ressources ont changé. Cette approche permet de comprendre les raisons derrière l’émergence des nouveaux langages et les améliorations apportées aux langages existant. En étudiant les versions antérieures et les innovations technologiques, on peut identifier les tendances et les besoins qui ont façonné le développement des langages
* **Méthode structure-fonctionnelle :** elle se concentre sur l’analyse des structures des langages de programmation et de leurs fonctions. Cette approches permet d’évaluer comment les caractéristique syntaxique et sémantiques d’un langage influencent ses performances et son efficacité en termes d’utilisation des ressources.

### **Techniques**

* **Technique documentaire** : elles impliquent la collecte et l’analyse de documents pertinents, tels que les articles de recherche, des livres, des rapports techniques et des publications en ligne. Ces documents fournissent des informations sur les performances de langages de programmation, les études antérieures et les meilleures pratiques.
* **Techniques d’Interview**: les techniques d’interview peuvent être utilisées pour recueillir des informations auprès de développeurs et d’experts en programmation. Ces pour entretiens peuvent fournir des perspectives qualitatives sur les expériences des utilisateurs avec les différents langages. Ainsi que des opinions sur les performances et l’utilisation des ressources.
  1. **Délimitation du sujet**

Cette étude se concentrera sur une sélection de langages de programmation python, Rust. Elle exclura les langages moins courants ou spécialisées, afin de se concentrer sur ceux qui largement utilisés dans l’industrie et la recherche. De plus l’analyse se limitera aux performances en termes de vitesse d’exécution et d’utilisation de la mémoire, sans aborder d’autres aspects tels que la sécurité ou la facilite d’apprentissage.

## **0.8 Subdivision du sujet**

Le sujet comprend trois chapitres

* **Chapitre I** : «  Généralité sur la programmation informatique »

Explique la genèse des langages de programmation et présentent des modelés ainsi que les logiciels relatifs.

* **Chapitre 2** : « Atouts et inconvénient des langages de programmation : étude des cas »

Comparaison de quelques langages de programmation choisie.

* **Chapitre 3** : « Analyse de performance et d’utilisation des ressources »

# **CHAPITRE 1. GENERALITES SUR LA PROGRAMMATION INFORMATIQUE**

## **Introduction**

La programmation informatique est l’art de concevoir des instructions que les ordinateurs peuvent exécuter pour réaliser des taches spécifiques et le traduire en un langage compréhensibles.

Dans un monde de plus en plus numérique, la programmation joue un rôle central dans le développement des logiciels, d’applications web, des jeux vidéo et des systèmes.

Depuis l’émergence des premiers ordinateurs dans les années 1940, les langages de programmation ont évolué de manière significative, permettant aux développeurs de créer les logiciels et les applications de plus en plus complexes et performantes. Ce chapitre vise à poser des bases nécessaires pour comprendre les langages de programmation en mettant des points sur leurs performances et leur utilisation des ressources, une étude comparative permet d’éclaircir les forces et les faiblesses de divers langages en facilitant ainsi le choix d’un langage approprié en fonction des besoins spécifiques d’un projet.

## **Genèse des langages de programmation[[1]](#footnote-1)**

Les langages de programmation ont évolué depuis les premier langages machine, qui étaient spécifiques à chaque architecture matérielle jusqu’aux langages de haut niveau qui abstraient les détails matériels. Le développement des langages a été motivé par le besoin d’améliorer la productivité des développeurs et de rendre la programmation plus accessible. La programmation informatique trouve ses origines bien avant l’apparition des ordinateurs modernes, son histoire remonte aux premières tentatives de mécanisation des calculs et de l’automatisation des taches

* **Années 1950 – 1960 :** apparition des premier langages comme **fortran et cobol,** conçus pour des applications scientifique et commerciales. Ces langages ont permis un niveau de généralisation plus élevée, facilitant les développements des logiciels complexes.
* **Années 1970 :** Introduction de langages comme C qui a permis un contrôle plus fin sur le matériel tout en restant relativement accessible.
* **Années 1980 – 1990 :** Emergences de langages orientés objet comme C++ et java qui ont introduit des concepts de modularité et de réutilisation du code.
* **Années 2000 :** développement des langages modernes comme Python, Rust qui mettent l’accent sur la simplicité et la sécurité.

## **Modelés des langages de programmation et leurs classifications**

Les langages de programmation peuvent être classés selon différents critères, notamment leur approche conceptuelle, leur mode d’exécution ou leur domaine d’application.

Chaque langage repose sur un modèle de programmation qui définit la manière dont les instructions sont structurées et exécutées

Voici les grandes catégories qui permettent de mieux comprendre comment ces langages sont organisés :

1. **Modèles**

* **Programmation impérative :** basée sur une séquence d’instructions que l’ordinateur exécute étape par étape. Exemple : c, python, java
* **Programmation orientée objet (POO) :** organisé autour de classes et d’objet qui modélisent des éléments du monde réel, elle utilise des concepts comme l’encapsulation, l’héritage et le polymorphisme. Exemple : java, c++, python
* **Programmation fonctionnelle :** elle met l’accent sur l’utilisation des fonctions pures sans modifier l’état global, elle favorise une meilleure gestion des erreurs et une optimisation de code. Exemple : Lisp
* **Programmation logique :** celle-ci est basée sur des règles et fraits permettant de résoudre des problèmes raisonnement, elle est principalement utilisée en intelligence artificielle. Exemple : prolog
* **Programmation événementielle :** elle exécute le code en réponse à des événements (clics, touches pressés). Très utilisé dans le développement web. Exemple : JavaScript
* **Programmation parallèle et concurrente :** conçu pour exécuter plusieurs taches simultanément et utilisée dans les systèmes distribués et les applications temps réel. Exemple : Rust
* **Programmation déclarative :** définit comment l’utilisateur spécifie les résultats attendus sans détailler l’algorithme.

Chaque modèle est adapté à un type de problème. La POO est idéale pour concevoir des applications évolutives tandis que la programmation fonctionnelle excelle dans le traitement des grandes quantités de données. Les langages modernes tendent d’ailleurs à combiner plusieurs modèles pour gagner en flexibilité.

1. **Classifications**

* **Selon le niveau d’abstraction**
* **Langages de bas niveau :** proches du langage machine, il permet un contrôle précis du matériel mais sont complexes à manipuler (langage assembleur c).
* **Langages de haut niveau :** plus proches des langages humains, ils sont faciles à comprendre et à utiliser (python, java, JavaScript).
* **Selon le mode d’exécution**
* **Langages compilés :** le code est transformé en un fichier exécutable avant l’exécution (c, c++).
* **Langages interprétés :** le code est exécuté ligne par ligne par un interpréteur, ce qui facilite le débogage mais peut ralentir l’exécution (python, JavaScript).
* **Langages hydrides :** elles combinent la compilation et l’interprétation pour optimiser la vitesse et la flexibilité (java).
* **Selon leurs usages**
* **Langages pour le développement web :** elle est spécialisée dans la création des sites et applications web (Html, css, javascript).
* **Langages pour l’intelligence artificielle et la data science :** optimisés pour le traitement de données et l’apprentissage automatique (python, Julia).
* **Langages pour les systèmes embarqués :** adaptés aux microcontrôleurs et aux objets connectés (c, Rust).
* **Langages de base de données :** utilisés pour manipuler et gérer des données (SQL)

Les langages de programmation sont organisés selon leur niveau d’abstraction, leurs modes d’exécution et leur modèle de programmation et le choix d’un langage dépend du projet à réaliser, qu’il s’agisse de développement web, intelligence artificielle, optimisation des performances ou gestion de base de données.

## **Aspects et défis de la programmation**

La programmation informatique est au cœur de l’évolution technologique. Elle permet de créer les logiciels, des systèmes embarqués, des intelligences artificielles et bien plus encore. Elle présente plusieurs aspects et défis qui influencent son développement et son adoption

### **Les principaux aspects de la programmation informatique**

1. **Performance et optimisation**

La rapidité d’exécution des programmes est importante, surtout dans les domaines des jeux vidéo. L’aspect est de concevoir des algorithmes efficaces qui minimisent la consommation des ressources (mémoire, processeur) tout en offrant des performances optimales.

1. **Sécurité informatique**

Avec la croissance des cyberattaques, sécuriser le code est primordial. Les failles de sécurité comme les erreurs de mémoire, la perte de confidentialité, des interruptions de service et la vulnérabilité des logiciels peuvent être exploitées par des hackers. Des langages comme « Rust » et des bonnes pratiques comme le chiffrement des données contribuent à améliorer cette protection.

1. **Accessibilité et démocratisation**

De fois réservée aux experts, la programmation est devenue plus accessible grâce à des langages intuitifs comme ‘’python’’ et aux plateformes éducatives interactives. Le défi es de rendre encore plus simple l’apprentissage du code pour que d’avantage de personne puissent en bénéficier.

1. **Développement durable et efficacité énergétique**

Les infrastructures informatiques consomment énormément d’énergie. Optimiser le code pour réduire l’impact envirommental est devenu un aspect majeur tels que dans le Cloud computing et l’IOT (internet des objets).

1. **Intelligence artificielle et automatisation**

L’évolution de l’IA modifie la façon dont nous programmons. Des outils comme copilot ou chatGpt assistent les développeurs mais posent aussi la question de la **créativité humaine** face à l’automatisation, comment garder un équilibre entre assistance et innovation.

### **1.4.2 Les défis de la programmation**

1. **Complexité du développement**

Les logiciels sont avancés, plus ils deviennent difficiles à coder et à maintenir. La gestion de la complexité est un défi constant notamment avec l’essor des micros services et des architectures distribuées.

1. **Compatibilité entre technologie**

Avec des milliers de langages disponibles, il est parfois difficile d’assurer la compatibilité entre les technologies. Un logiciel développé aujourd’hui peut rapidement devenir obsolète si l’évolution technologique n’est pas prise en compte.

1. **Sécurité des données et protection de la vie privée**

Le respect de la confidentialité des utilisateurs est un défi majeur, surtout avec l’essor du big data de l’IA. Les réglementations comme le RGPD impose des normes strictes et les développeurs doivent adapter leurs pratiques pour garantir la protection des données

1. **Evolution rapide des langages et outils**

Les langages évoluent sans cesse. Un développeur doit continuellement mettre à jour ses 5compétences pour suivre les nouvelles tendances. Qui demande un apprentissage permanant.

### **1.4.3 Concepts fondamentaux de la programmation**

La programmation informatique repose sur plusieurs concepts clés qui permettent se structurer et d’exécuter des programmes efficacement. Ces concepts sont universels et s’appliquent à tous les langages de programmation bien qu’ils puissent être implémentés différemment

### **1.4.4 Variables et types de données**

Une variable est un espace mémoire destiné à stoker une valeur. Chaque variable possède un type de données qui définit le type d’information qu’elle peut contenir :

* Types numériques : entiers (Int), réels (float, double)
* Types textuels : chaines des caractères (string)
* Types logiques : booléen (true, false)
* Types complexes : tableaux (array), objets (class)

### **1.4.5 Structures de contrôle**

Les structures de contrôle permettent de gérer de déroulement du programme

1. **Les conditions :**

Les structures conditionnelles permettent d’exécuter un bloc de code en fonction d’une condition donnée. **If, else, switch :** elles exécutent le code selon une condition spécifique.

1. **Les boucles :**

Les boucles permettent de répéter un bloc de code plusieurs fois. **For, white :** permet de répéter une série d’instructions jusqu’à ce qu’une condition soit remplie.

### **1.4.6 Fonctions et modularité**

Une fonction un bloc de code qui exécute une tache spécifique et peut être réutilisé plusieurs fois

**Modularité :** diviser le code en sous parties facilite sa gestion

**Réutilisation :** un même code peut être utilisé plusieurs endroits sans duplication

## **Gestion de la mémoire en programmation**

La gestion de mémoire est un aspect important du développement des logiciels, elle influence directement les performances et la stabilité des programme. Une mauvaise gestion de mémoire peut entrainer des pertes de mémoire, des erreurs critiques. Dans cette étude approfondie nous analyserons les mécanismes de gestion de mémoire, les techniques d’optimisation et les défis rencontrés par les développeurs

La programmation informatique repose sur l’utilisation efficace de cette ressource afin d’assurer la rapidité d’exécution et l’optimisation des performances

1. **Enjeux de gestion de la mémoire**

**Performance :** Une allocation efficace permet d’accélérer l’exécution du programme.

**Sécurité :** une mauvaise gestion exposée les programmes aux problèmes de sécurité.

**Stabilité :** des erreurs de mémoire peuvent provoquer des comportements imprévisibles.

1. **Types de mémoire et organisation**

* **La mémoire vive (RAM) :** la RAM est utilisées pour stocker temporairement les données dont le processeur a besoin. Elle est volatile ce qui signifie que son contenu est perdu lorsque l’ordinateur s’éteint.
* **La mémoire cache :** située dans le processeur, la mémoire cache permet un accès rapide aux donnée fréquemment utilisées.
* **La mémoire virtuelle :** si la RAM est saturée, une partie du disque dur du SSD est utilisée comme extension temporaire de la mémoire (pagefile sous Windows, swap sous linux).
* **Segmentation de la mémoire :** l’espace mémoire d’un programme est divisé en plusieurs zones :
* **Pile (stack) :** stocke les variables locales et gère les appels de fonction.
* **Tas (Heap) :** gère la mémoire allouée dynamiquement (création d’objets, tableaux dynamique).

1. **Modes de gestion de la mémoire en programmation**

**Allocation statique :** les variables sont déclarées à la compilation et restent en mémoire durant toute l’exécution. Exemple : le variable global en C, constante en java

**Allocation dynamique :** permet d’allouer et de libérer la mémoire à la demande. Elle est souvent dans les langages comme C, (malloc (), free()) et C++ (new ; delete).

**Garbage collection (ramasse-miettes) :** ce sont des systèmes automatiques de gestion mémoire qui libère les objets inutilisés. Présent dans le java, python, C# pour éviter les erreurs de mémoire.

1. **Problèmes et défis liés à la gestion de la mémoire**

* **Fuites de mémoire :** se produisent lorsque la mémoire est allouée mais jamais libérée, consommant inutilement des ressources.
* **Débordement de mémoire (Buffer overflow) :** lorsqu’un programme écrit au-delà de la mémoire allouée, il peut corrompre des données et exposer le système à des failles de sécurité.
* **Fragmentation de la mémoire :** l’allocation et la libération répétées de blocs de mémoire peuvent fragmenter l’espace disponible, réduisant l’efficacité du programme.

1. **Optimisation et bonnes pratiques**

**Utilisation efficace des structures de données :** des choix judicieux comme utiliser des listes chainées au lieu de tableaux statiques, peuvent améliorer la gestion mémoire

**Gestion manuelle ou automatique selon le besoin :** pour des programmes nécessitant un contrôle précis (systèmes embarqués), l’allocation manuelle est préférable. Pour les applications web ou mobiles un Garbage collector est souvent plus efficace.

**Techniques avancées :** Référence intelligent (weak référence) pour éviter les cycles de dépendance

La gestion de mémoire est essentielle pour assurer la stabilité, la rapidité et l’efficacité des programmes. Les développeurs doivent comprendre les mécanismes sous-jacents pour éviter les erreurs et concevoir des applications performantes.

## **Entrées et sortie en programmation**

Les entrées et sorties (I/O Input/Output) jouent un rôle fondamental en programmation car elles permettent aux programmes d’interagir avec l’utilisateur, les fichiers, les bases de données et le réseau. Une gestion efficace des I/O influence directement la rapidité et la fiabilité des logiciels.

1. **Enjeux des I/O en programmation**

**Fluidité :** Une bonne gestion des I/O améliore l’expérience de l’utilisateur.

**Performance :** les opérations entrée/sortie peuvent ralentir un programme si elles ne sont pas optimiser.

**Sécurité :** Une mauvaise gestion des données d’entrée peut exposer le programme à de risque.

1. **Types d’entrées et de sorties**

* **Entrées utilisateur :** les données peuvent provenir de l’utilisateur par le biais du clavier ou d’autres périphériques

**Exemple en python**

Nom : Input (‘’Entrez votre nom : ’) ;

Print (‘’Bonjour, ‘’nom) ;

**Exemple en c**

Char nom [50] ;

Printf (‘’Entrez votre nom : ’’) ;

Scanf (‘’ %s’’, nom) ;

Printf (‘’Bonjour, %s\n’’, nom) ;

* **Entrées et sorties via fichiers :** les programmes peuvent lire et écrire des données dans des fichiers (texte, binaire, basse données)

**Lecture d’un fichier en python**

With open (data.txt’’, ‘’r’’) as fil contenu = file.read () Print (contenu)

1. **Optimisation et bonnes pratiques**

**Utilisation efficace des I/O :** limiter les accès aux fichiers pour éviter les ralentissements. Utiliser les buffers pour accélérer le traitement des données.

* **Gestion des erreurs :** toujours vérifier que les fichiers existent avant de les lire, traiter les erreurs réseau pour éviter les crashs.
* **Sécurité des entrées :** la validation des données pour éviter les injections SQL ou les XSS, l’utilisation d’entrées sécurisées comme ‘’prepared statement en bases de données.

Les entrées et des sorties sont une composante essentielle de tout programme, leurs gestions efficace améliore la performance, la sécurité et l’interaction d’utilisateur. Des bonnes pratiques comme la validation des entées et l’optimisation des accès aux fichiers garantissent la stabilité des programmes modernes.

## **Algorithme et structure de données fondement et applications**

L’algorithme et le structure de donnée sont les piliers essentiels de la programmation informatique. Ils permettent d’organiser les informations efficaces et d’exécuter des opérations de manière optimisée, comprendre ces concepts est fondamental pour améliorer la rapidité et la fiabilité des logiciels.

Un algorithme est une séquence d’instructions permettant de résoudre un problème ou d’exécuter une tache. Il définit comment traiter les données et obtenir un résultat.

Une structure de données est une manière d’organiser et de stocker des données afin de faciliter leur manipulation. Un programme efficace repose sur une bonne utilisation des structures de donnée adaptées à ses besoins.

**Ces concepts sont importants pour :**

* L’optimisation : un bon algorithme réduit le temps d’exécution et améliore l’efficacité du code.
* Gestion des ressources : une structure bien pensée minimise la consommation de mémoire
* Fiabilité : des algorithmes robustes garantissent de résultat précis et prévisible

### **1.6.1 Principaux types d’algorithme**

1. **Algorithme de recherche**

Ces algorithmes permettent de retrouver un élément dans une liste ou un tableau

* Recherche linéaire : parcourt les éléments un par un (O(n)).
* Recherche binaire : divise la liste en deux parties à chaque étape (O (log n)).

1. **Algorithme de tri**

Le tri est important pour organiser les données rapidement

* Tri par sélection (O (n²)) : trouve le plus petit élément et le place en premier.
* Tri rapide (Quicksort) (O (n log n)) : divise la liste en sous-parties pour trier plus vite.
* Tri fusion (Mergesort) (O (n log n)) : sépare les éléments, les trie, puis le fusionne.

1. **Algorithme de parcours de graphes**

Les graphes sont largement utiliser en intelligence artificielle (I.A) et en réseau

* BFS (breadth-first search) : parcours en largeur, adapté aux chemins courts.
* DFS (Depth-first search) : parcours en profondeur, idéal pour explorer toutes les solutions.
* Dijkstra : trouve les chemins le plus courts entre deux points.

### **1.6.2 Principales structures de données**

* **Tableaux et listes**

Un tableau est une structure de données qui stocke une collecte des éléments de même type dans une séquence voisine en mémoire

* Taille fixe : la taille d’un tableau est généralement définie lors de sa création et ne peut pas être modifié
* Accès direct : les éléments d’un tableau peuvent être accédés directement via leur index ce qui permet un accès rapide (complexité O(1))
* Homogénéité : tous les éléments d’un tableau doivent être du même type (exemple : tous les entiers ou des caractères.

Les listes est une structure de données qui peut contenir une collecte des éléments, souvent des types différents et qui redimensionnée ou dynamiquement

* Taille dynamique : les listes peuvent diminuer en taille au fur et à mesure que des éléments sont ajoutés ou supprimés.
* Accès séquentiel : l’accès aux éléments se fait généralement par index, mais il peut être moins rapide que pour les tableaux en raison de la gestion dynamique de la mémoire.
* Hétérogénéité : les liste peuvent contenir les éléments de différent type (exemple : un entier, une chaine de caractère, un objet, etc…)

A part les tableaux et les listes, nous pouvons trouver d’autres structures des données telles que les dictionnaires, les tables de hachage, les graphes, les arbres….

## **Critères de performance de langages de programmation**

La performance d’un langage de programmation est un facteur clé qui influence la rapidité et l’efficacité des logiciels. Elle dépend de plusieurs aspects comme gestion de la mémoire, le temps d’exécution, l’optimisation du code et l’utilisation des ressources matérielles. Cet approfondissement détaillé explore les critères qui déterminent la performance la performance des langages dans le développement informatique.

La programmation informatique repose sur des langages qui traduisent les instructions en un format compréhensible par la machine. Certains langages sont conçus pour optimiser la rapidité d’exécutions tandis que privilégient la simplicité et la flexibilité au détriment des performances. La performance est important pour :

* vitesse d’exécution : un langage rapide permet de traiter les opérations efficacement
* gestion optimale des ressources : l’utilisation intelligente de la mémoire et du CPU impacte l’efficacité des programmes.
* Expérience utilisateur : un programme fluide et réactif est plus agréable à l’utiliser.
* Cout opérationnel : un logiciel optimisé réduit la consommation énergétique et le frais d’infrastructure.

### **1.7.1 Critères techniques de performance**

Lorsqu’on parle des langages de programmation, deux modes d’exécutions principaux se distinguent : la compilation et l’interprétation. Chacun de ces modes a ces propres caractéristiques, avantages et inconvénients.

1. **Compilation**

La compilation est le processus par lequel un programme source écrit dans un langage de programmation (généralement de haut de niveau) est traduit en code machine (binaire) par un compilateur. Ce code machine peut ensuite être exécuté directement par l’ordinateur. Les caractéristiques de la compilation sont :

* Etape unique : le code source et traduit en un fichier exécutable avant d’être exécuté.
* Vitesse d’exécution : les programme compilés s’exécutent généralement plus rapidement, car le code machine est directement exécuté par le processeur.
* Optimisation : les compilateurs peuvent appliquer diverse optimisation au code améliorant ainsi les performances.
* Erreurs : les erreurs de syntaxe et de type sont détectées lors de la compilation, ce qui permet de corriger les problèmes avant l’exécution.

Exemple des langages compilés : C, C++, Rust, Go

Avantages : exécution de la performance est rapide grâce à la pré-compilation, erreurs sont souvent détectées avant l’exécution et la distribution est facile sous forme de fichiers exécutables.

Inconvénients : Le temps de compilation peut être long surtout pour des grands pour des grands projets, le code compilé est souvent spécifique à une architecture ou un système d’exploitation particulier.

1. **Interprétation**

L’interprétation est le processus par lequel un interpréteur exécute le code source ligne par ligne ou instruction par instruction. Sans le traduire en code machine au préalable, l’interpréteur lu et exécute le code directement. Elles sont caractériser par :

L’exécution immédiate : pas de fichier exécutable intermédiaire ; le code est exécuté directement.

* Flexibilité : permet des modifications et des tests rapides, idéal pour le développement interactif.
* Erreurs : les erreurs sont détectées à l’exécution ce qui peut rendre le débogage plus difficile.

Exemple des langages interprétés : Python, JavaScript, PHP

Avantages : développement rapide est idéal pour les prototypes et les scripts, la portabilité et que le même code peut s’exécuter sur les différents plateformes tant que l’interpréteur est disponible. L’utilisation est plus simple à utiliser pour les développeurs avec des environnements interactifs.

Désavantages : les performances sont généralement plus lentes que le code compilé car chaque ligne doit être analysée et exécutée à la volé. Les erreurs ne sont souvent détectées qu’à l’exécution ce qui peut entrainer des comportements inattendus.

Le choix entre compilation et interprétation dépend des besoins spécifiques du projet, les langages compilés sont souvent préférés pour des applications nécessitant des performances optimales tandis que les langages interprétés idéaux pour le développement rapide et la flexibilité. Certains langages modernes comme java et C# utilisent une approche hybride ou le code est d’abord compilé en bytecode puis interprété ou compilé à la volée par une machine virtuelle.

### **1.7.2 Concurrence et parallélisme de la programmation**

La concurrence et le parallélisme sont deux concepts important en programmation qui permettent d’améliorer l’efficacité des programmes, surtout dans un monde ou les applications doivent gérer de plus en plus de taches simultanément. Bien qu’ils soient souvent confondus, ces deux concepts ont des significations distinctes et des applications différentes.

#### **1.7.2.1 Concurrence**

La concurrence désigne la capacité d’un programme à gérer plusieurs taches en même temps, mais pas nécessairement de manière simultanée. Cela signifie qu’un programme peut passer d’une tache à un autre, donnant l’illusion que plusieurs taches sont exécutées en même temps. Exemple de concurrence

* Applications web : un serveur web peut gérer plusieurs requêtes d’utilisateurs en utilisant des techniques de concurrence. Lorsque plusieurs utilisateurs accèdent à un site web, le serveur peut traiter chaque requête de manière concurrente même si une seule requête est traitée à la fois.
* Système d’exploitation : le S.E modernes utilise la concurrence pour gérer plusieurs processus et threads. Par exemple un utilisateur peut écouter de la musique tout en naviguant sur internet.

#### **1.7.2.2 Techniques de concurrence**

Multi-threading : permet d’exécuter plusieurs threads (unités d’exécution) au sein d’un même processus. Chaque thread peut gérer une tache différente. Un navigateur web utilise peut utiliser des threads pour changer plusieurs éléments d’une page web en même temps

Asynchrone : utilise des appels non bloquants pour permettre à un programme de continuer à s’exécuter tout en attendant qu’une opération (d’entrée/sortie) se termine.

#### **1.7.2.3 Parallélisme**

Le parallélisme consiste à exécuter plusieurs taches simultanément. Cela nécessite généralement plusieurs unités de traitement (comme le cœur de processeur) qui travaillent en même temps sur les différentes tâches. Exemples de parallélisme

* Calcul scientifique : dans les applications comme la simulation climatique ou le rendu graphique, des calculs lourds peuvent être répartis sur plusieurs cœurs de processus pour accélérer le traitement.
* Traitement d’image : Les algorithmes de traitement d’image peuvent traiter plusieurs pixels en parallèle, améliorant ainsi la vitesse de traitement.

**L’avantage de concurrence** : Les applications peuvent rester réactives même lorsqu’elles effectuent des taches longues, elle permet une meilleure utilisation des ressource système, en particulier dans les applications I/O-bound.

**Désavantage de concurrence :** La gestion de la concurrence peut introduit des bugs difficiles à détecter, tels que les conditions de course. La création et la gestion de threads peuvent entrainer une surcharge.

#### **1.7.2.4 Techniques de parallélisme**

Multi-processing : Utilise plusieurs processus distincts qui s’exécutent en parallèles sur les différents cœurs de processus.

GPU (unité de traitement graphique : Utilise des milliers des cœurs pour effectuer des calculs en parallèle, particulièrement efficace pour les taches nécessitant beaucoup de calcul comme l’apprentissage automatique.

**Avantage :** les taches lourdes peuvent être exécuté plus rapide grâce à l’exécution simultanée, les applications peuvent être conçues pour tirer parti de l’augmentation du nombre des cœurs de processus.

**Désavantage :** La coordination entre les taches parallèle peut être complexe et source d’erreurs nécessitant des ressources matérielles supplémentaire pour tirer parti du parallélisme.

#### **1.7.2.5 Applications pratiques**

1. **Développement web**

Dans le développement web, la concurrence est souvent utilisée pour gérer les requêtes utilisateur. Un serveur web peut utiliser des threads pour traiter chaque requête de manière concurrente, permettant ainsi à plusieurs utilisateurs d’accéder au site en même temps.

1. **Traitement de données**

Les applications de traitement de donnée comme celles utilisées en science des données ou en apprentissage automatique bénéficient du parallélisme. Les algorithmes peuvent être conçus pour traiter des grandes quantités de données en parallèle ce qui accélère le processus d’analyse.

1. **Jeux vidéo**

Les jeux vidéo modernes utilisent à la fois la concurrence et le parallélisme. La logique de jeu peut être gérée de manière concurrente tandis que le rendu graphique peut être effectué en parallèle sur le GPU.

* Java : offre des bibliothèques pour le Multi-threading et la programmation concurrente.
* Python : Utilise des modules comme treading et le multiprocessus pour gérer la concurrence et le parallélisme.
* C++ : fournit des bibliothèques comme open-MP pour le parallélisme.

### **1.7.3 L’impact du Cloud et de l’IA en programmation**

L’évolution technologique a transformé le paysage de la programmation parmi les changements les significatifs. L’émergence du Cloud computing et de l’IA a eu un impact profond sur la manière dont les développeurs conçoivent, construisent et déploient des applications.

#### **1.7.3.1 Cloud computing**

Le Cloud computing désigne la fourniture de service informatique via internet. Ce service inclut le stockage, le traitement de données et des applications accessibles à distance, au lieu de dépendre d’un serveur local. Les utilisateurs peuvent accéder à des ressources informatiques sur des serveurs distants.

1. **Types de services Cloud**

* Infrastructure as a service (Iaas) : fournit des ressources informatiques virtualisées sur internet (ex : Google Cloud compute).
* Platform as a service (paas) : offre des plateformes pour développer, tester et déployer des applications (Google App Engine).
* Software as a service (Saas) : propose des applications accessibles en ligne (Google workspace, Salesforce).

1. **Avantage du Cloud computing**

* Accessibilité : elle a accès aux applications et aux données depuis n’ importe où avec une connexion internet.
* Mise à jour automatique : les fournisseurs gèrent les mises à jour et la maintenance.
* Elle donne la possibilité d’ajuster les ressources en fonction des besoins, moins de dépenses liées à l’infrastructure physique.

1. **L’impact du Cloud sur la programmation**

**Développement d’application :** Le Cloud a révolutionné le développement d’application en permettant aux développeurs de se concentrer sur le code plutôt que sur l’infrastructure. Grâce aux services Paas, les développeurs peuvent déployer des applications sans se soucier de la gestion des serveurs.

**Collaboration et travaille à distance :** Le Cloud facilite la collaboration entre les équipes de développement quel que soit leur emplacement. Les outils de gestion de projet et de visionnage sont basés sur le Cloud.

**Déploiement continu :** les pratiques de déploiement continu (CI/CD) sont facilitées par le Cloud. Les développeurs peuvent automatiser le processus de test et de déploiement, ce qui accélère la mise en production des nouvelles fonctionnalités.

**Sécurité et sauvegarde :** les fournisseurs de Cloud offrent des solutions de sécurité avancées et des options de sauvegarde. Permettant aux développeurs de se concentrer sur le développement sans soucier des pertes de données.

#### **1.7.3.2 L’intelligence Artificielle**

L’IA désigne la simulation de l’intelligence humaine par des machines. Cela inclut des techniques comme l’apprentissage automatique (machine Learning), le traitement du langage naturel (NLP) et la vision par le pc

1. **Types de l’IA**

* IA faible : conçue pour des taches spécifiques (ex : assistants virtuels).
* IA forte : capable d’effectuer des taches intellectuelles humaines (encore théorique).

1. **Avantage de l’IA**

* Automatisation : réduction du temps et des efforts nécessaires pour accomplir des tâches répétitives.
* Précision : Amélioration de la précision dans des domaines comme la médecine, la finance la logistique.
* Analyse de données : la capacité à traiter et à analyser des grandes quantités de données rapidement.

1. **L’impact de l’IA sur la programmation**

**Outils de développement :** L’IA fournit des outils qui aident les développeurs à écrire le code plus efficacement. Ex : des assistants de codage comme GitHub copilot utilisent l’IA pour suggérer des lignes de code en fonction du contexte.

**Tests automatisés :** L’IA peut automatiser le processus de test, en générant des tests unitaire et en identifiant les bogues potentiels dans le code. Cela permet de réduire le temps de développement et d’améliorer la qualité du logiciel.

**Analyse prédictive :** L’IA permet d’analyser les données des utilisateurs pour prédire les comportements futurs. Cela aide les développeurs à concevoir des applications plus adaptées aux besoins des utilisateurs.

**Personnalisation :** l’IA facilite la personnalisation des applications (exemple : les recommandations de produit sur les sites de commerce électroniques sont souvent alimentées par des algorithmes d’’IA qui analysent le comportement des utilisateurs.

#### **1.7.3.3 Cas d’utilisation du Cloud et de l’IA en programmation**

**Application web :** les applications web modernes utilisent souvent de service Cloud pour le stockage et le traitement de données. L’IA est intégrée pour offrir des fonctionnalités comme la recherche intelligente et l’analyse des données de l’utilisateur.

**Applications mobiles :** les applications mobiles exploitent le Cloud pour synchroniser les données et l’IA pour offrir les expériences personnalisées comme des recommandations basées sur les préférences de l’utilisateur.

**Big data et analyse :** les entreprise utilisent des solutions Cloud pour stocker des grandes quantités de données et l’IA est utilisée pour analyser ces données et en tirer des informations précieux pour la prise de décision.

**Automatisation des processus :** l’intégration de l’IA dans le Cloud permet d’automatiser des processus métier réduisant ainsi les couts et augmentant l’efficacité.

#### **1.7.3.4 Défis et limites**

**Sécurité des données :** Les stockagesdes donnéessensibles dans le Cloud soulève des préoccupations en matière de sécurité. Les développeurs de Cloud doivent s’assurer que les données sont protégées contre les violations de sécurité.

**Dépendance au fournisseur** : les entreprises peuvent devenir dépendantes d’un fournisseur de Cloud, ce qui peut poser des problèmes en cas de changement de politique ou du prix.

**Biais de l’IA :** l’IA peut introduire des biais si les données utilisées pour l’entrainer ne sont pas représentatives. Cela peut conduire à des décisions injustes ou inexactes.

#### **1.7.3.5 Tendance future**

**Cloud hybrique :** les entreprises adoptent le plus en plus des solutions de Cloud hybrique, combinant des infrastructures sur le site et de service Cloud pour plus de flexibilité.

**IA éthique :** avec la montée des préoccupations liées à l’éthique de l’IA les développeurs devront intégrer des principes éthiques dans la conception et utilisation des systèmes de l’IA.

**Automatisation et DevOps :** l’automatisation des processus de développement continuera de croitre avec des outils de l’IA intégrés pour améliorer la productivité et la qualité du code.

L’impact de Cloud et de l’IA sur la programmation est indéniable, ces technologies transforment la manière dont les développeurs conçoivent, construisent et déploient des applications. En tirant parti des avantages du Cloud et de l’IA, les développeurs peuvent créer des solutions plus efficaces, évolutives et personnalisées.

## **Importance de l’optimisation en programmation**

L’optimisation en programmation est un processus important qui vise à améliorer les performances d’un programme, en termes de rapidité d’exécution que l’utilisation de ressources. Dans un monde ou les utilisateurs attendent les applications réactives et efficaces, l’optimisation devient une priorité pour les développeurs.

L’optimisation en programmation consiste à modifier un algorithme ou un code pour améliorer son efficacité, cela peut inclure :

* Réduction du temps d’exécution : rendre un programme plus rapide
* Minimisation de l’utilisateur de la mémoire : réduire la quantité de mémoire nécessaire pour exécuter un programme
* Amélioration de la réactivité : rendre une application plus réactive aux actions de l’utilisateur

**Types d’optimisation**

* Optimisation de l’algorithme : choisir des algorithmes plus efficaces pour résoudre un problème.
* Optimisation du code : Améliorer le code source pour qu’il s’exécute plus rapidement ou utilise moins des ressources.
* Optimisation des ressources : Gérer efficacement les ressources systèmes, comme la mémoire et le processus.

### **1.8.1 Pourquoi l’optimisation est-elle importante**

1. **Amélioration des performances :** l’optimisation permet de rendre les programmes plus rapides et plus efficaces. Dans des domaines comme le traitement de données, la performance est importante. Exemple : un algorithme tri optimisé peut réduire le temps nécessaire pour trier des grandes quantités des données.
2. **Expérience utilisateur :** une application réactive est rapide améliore l’expérience de l’utilisateur. Les utilisateurs sont plus susceptibles de rester sur une application qui répond rapidement à leurs actions. L’optimisation contribue donc à la satisfaction et à la fidélisation des utilisateurs.
3. **Utilisation efficace des ressources :** les programmes optimisés utilisent moins de ressources, ce qui peut réduire les couts d’exploitation, dans les environnements Cloud où les frais sont souvent basés sur l’utilisation des ressources. Cela permet aussi de prolonger la durée de la vie des appareils en réduisant la charge sur le matériel.
4. **Scalabilité :** les applications optimisées sont plus faciles à faire évoluer. Lorsqu’une application doit gérer un nombre croissant d’utilisateurs ou des données, une bonne optimisation permet de s’assurer qu’elle peut évoluer sans dérations des performances.

### **1.8.2 Défis de l’optimisation**

**Compromis entre lisibilité et performance :** l’optimisation peut parfois rendre le code plus complexe et moins lisible. Il est important de trouver un équilibre entre la performance et la clarté du code, surtout dans les projets collaboratifs.

**Cout de l’optimisation :** l’optimisation peut nécessiter un invertissent en temps et en ressources. Parfois le gain de performance ne justifie pas le cout de l’optimisation, surtout si le programme fonctionne déjà de manière satisfaite.

**Changement de contexte :** l’optimisation qui fonctionne dans le contexte particulier peut ne pas être efficace dans un autre. Pour une petite base de données, une optimisation peut ne pas fonctionner pour une base de données massive.

**Evolution des technologies :** les technologies évoluent rapidement, ce qui signifie que les techniques d’optimisation doivent également évoluer. Ce qui est considéré comme une bonne pratique aujourd’hui peut devenir obsolète demain.

L’optimisation en programmation est un aspect essentiel du développement logiciel, elle permet d’améliorer les performances, d’optimiser l’utilisation des ressources et d’offrir une meilleure expérience utilisateur. En adoptant des techniques d’optimisations appropriées, les développeurs peuvent créer des applications efficaces réactives et évolutives répondant ainsi aux exigences croissantes des utilisateurs et du marché.

## **1.9. CONCLUSION**

La programmation informatique est un outil essentiel qui permet de créer des logiciels et de donner les instructions aux machines. Elle repose sur des langages de programmation et applications utiles dans des nombreux domaines, comprendre les bases de la programmation nous permet d’explorer des nombreuses possibilités et de créer des choses nouvelles.

# **CHAPITRE 2 : ATOUTS ET INCONVENIENTS DES LANGAGES DE PROGRAMMATION**

**« Étude de cas : comparaison entre RUST et PYTHON »**

## **2.1 Introduction**

Dans le monde la programmation, le choix du langage peut avoir un impact significatif sur le développement d’un projet. Chaque langage de programmation possède ses propres caractéristiques, avantage, inconvénient qui les rendent plus ou moins adapté à des tâches spécifiques. Dans ce chapitre, nous allons examiner deux langages populaires : RUST et PYTHON

Rust est souvent salué pour sa performance et sa sécurité, notamment grâce à son système de gestion de mémoire. Python est reconnu pour sa simplicité et sa lisibilité, ce qui en fait un excellent choix pour le débutant ainsi que pour le prototypage rapide. Cette étude de cas vise à comparer les atouts et les inconvénients de Rust et python, en mettant en lumière leurs caractéristiques respectives, leurs performances, leurs courbes d’apprentissages ainsi que leurs écosystème. En analysant ces deux langages, nous espérons fournir des idées claires qui aideront les développeurs à choisir le langage le plus adapté à leur besoin spécifiques.

## **2.2 Présentation des langages**

**Rust :** Rust est langage de programmation systématique conçu pour être rapide, sécurisé et concurrent. C’est un langage compilé développé par **Mozilla Research**, Rust met l’accent sur la sécurité de la mémoire et la prévention des erreurs courantes telles que les fuites de mémoire et les accès concurrents nos sécurisé.

**Python :** Python est langage de programmation interprété, orienté objet polyvalent et de haut niveau créé par **Guido van Rossum** et lancé en 1991. Il est connu pour sa syntaxe claire lisible, ce qui en fait un choix populaire pour les débutants et les développeurs expérimentés. Grace à sa syntaxe simple et son large écosystème, il est utilisé dans de nombreux domaines. Développement web, data science, automatisation, intelligence artificielle etc…

## **2.3. Avantages et désavantages de RUST**

### **2.3.1. Avantages**

* **Sécurité de la mémoire**

Rust utilise un système de propriété. Chaque variable a un propriétaire et lorsque ce propriétaire sort de portée, la mémoire est automatiquement libérée. Cela signifie qu’il n’y’a pas besoin de Garbage collector, ce qui rend Rust plus performant.

* Les avantage de cette sécurité est que les erreurs liées à la mémoire sont réduite, ce qui diminue le nombre de bogues dans le code.
* Applications fiables : Les programmes écrits en Rust sont plus surs et fiable, ce qui est important pour des applications critiques comme celles utilisées dans la santé ou l’aéronautique.
* **Performance élevée**

Rust est langage compilé, ce qui signifie que le code source est transformé en code machine avant d’être exécuté. Cela permet à Rust d’atteindre les performances similaires à celle de C et C++. Rust optimise le code pour le rendre encore plus rapide.

* Réduction des temps de réponse : les applications peuvent répondre plus rapidement aux utilisateurs.
* Efficacité des ressources : moins de consommation de mémoire et de processeur, ce qui est particulièrement important pour les appareils avec les ressources limitées.
* **Concurrence sécurisée**

Rust utilise son système de propriété pour s’assurer que les données ne sont pas modifiées par plusieurs threads en même temps. Cela évite les conditions de course qui sont des erreurs courantes dans les programmes multithread.

* Simplicité : les développeurs peuvent écrire du code concurrent sans se soucier des erreurs de mémoire.
* Performance amélioré : les applications peuvent tirer parti des processeurs multicoeurs sans compromettre la sécurité.
* **Écosystème de bibliothèque**

Rust dispose d’un gestionnaire de paquets appelé CARGO, cargo facilite l’installation et la gestion des bibliothèques. Il existe de nombreuses bibliothèques disponibles pour des taches variées comme le traitement de données, le développement web et bien plus encore.

* Gain de temps**:** Les développeurs peuvent utiliser des bibliothèques existantes au lieu de réécrire le code
* Communauté active : Une communauté dynamique signifie que les bibliothèques sont régulièrement mises à jour et améliorées.
* **Documentation de qualité**

Rust et connu pour sa documentation claire et complète, chaque bibliothèque et chaque fonction sont bien expliquées. Avec des exemples de code, cela rend l’apprentissage et l’utilisation de Rust beaucoup plus facile

* Facilite l’apprentissage : les nouveaux développeurs peuvent comprendre rapidement comment utiliser Rust.
* Réduit les erreurs : une bonne documentation aide à éviter les malentendus et les erreurs dans le code.
* **Interopérabilité**

Rust permet d’appeler des informations écrites dans d’autres langages grâce à des bindings. Cela signifie que vous pouvez utiliser des bibliothèques C dans vos projets, Rust sans trop de difficulté.

* Flexibilité : les développeurs peuvent tirer parti des bibliothèques existantes dans d’autres langages.
* Migration facile : les projets peuvent être progressivement migrés vers Rust sans avoir à tout réécrire.
* **Outils développement**

Rust est livré avec un compilateur performant appelé **RUSTC** et un gestionnaire de de paquets appelé **CARGO.** De plus il existe de nombreux outils pour le débogage et l’analyse de code.

* Amélioration de la productivité : les développeurs peuvent se concentrer sur la création de code plutôt que sur la configuration des outils.
* Débogage plus facile : les outils permettent d’identifier rapidement les erreurs dans le code.
* **Communauté active**

La communauté Rust est connue pour être accueillante et utile, il existe de nombreux forums, groupe de discussion et évènement où les développeurs peuvent échanger des idées et poser des questions.

* Soutien continu : les développeurs peuvent obtenir de l’aide rapidement lorsqu’ils rencontrent des problèmes.
* Partage des connaissances : les membres de la communauté partagent les ressources et des tutoriels, ce qui facilite l’apprentissage.

Rust est un langage de programmation qui offre des nombreux avantages en matière des sécurités, performances, de gestion de la concurrence. Son écosystème de bibliothèque, sa documentation de qualité, ses outils de développement et sa communauté active en font un choix idéal pour les développeurs. Que ce soit pour des applications critiques ou des projets personnels, Rust est un langage qui mérite d’être exploré et utilisé.

### **2.3.2. Désavantages**

* **Courbe d’apprentissage**

Rust utilise un système de propriété complexe qui est diffèrent de nombreux autres langages, ce système est conçu pour garantir la sécurité de la mémoire mais il peut être déroutant pour les débutant.

* Difficulté pour les débutants : les développeurs qui débutent en programmation peuvent trouver Rust plus difficile à maitriser que des langages python, javascript. Les messages des erreurs peuvent être longs et techniques, ce qui peut frustrer les nouveaux apprenants.
* Temps d’apprentissage plus long : les équipes doivent investir plus de temps pour former les nouveaux développeurs sur Rust.
* **Temps de compilation**

Rust effectue de nombreuses vérifications lors de la compilation pour garantir la sécurité de la mémoire et la performance. Ces vérifications peuvent ralentir les processus de compilation, le compilateur vérifie que le code respecte les règles de propriété et d’emprunt

* Cycle de développement plus lent : les développeurs doivent attendre plus longtemps pour avoir les résultats de leurs modifications.
* Frustration : les temps d’attente peuvent entrainer des frustrations surtout lors du débogage.
* **Ecosystème et bibliothèques**

Certaines bibliothèques de Rust ne sont pas aussi matures ou bien documentées que celle d’autres langages.

* Difficulté à trouver des solutions : les développeurs peuvent avoir du mal à trouver des bibliothèques pour des taches spécifiques.
* Besoins de réinventer la roue : les développeurs peuvent être contrains d’écrire leur propre code pour des fonctionnalités courantes au lieu d’utiliser des bibliothèques

Existants.

* **Complexité syntaxique**

Rust utilise des macros qui peuvent être difficile à comprendre pour les nouveaux utilisateurs, les macros permettent de générer le code mais leur syntaxe peut être déroutante. Elle propose des types de données avancés comme (les enums et les traits) qui peuvent être difficile à maitriser

* Difficulté à écrire le code : les développeurs peuvent avoir du mal à écrire les codes correct en raison de complexité de la syntaxe.
* Erreurs fréquentes : les erreurs de syntaxe peuvent être plus fréquentes, ce qui entraine des frustrations.
* **Outils et support**

Certains environnements de développement intégré (IDE) ne prennent pas encore en charge Rust aussi bien que d’autres langages, certains outils manquent de fonctionnalités avancées comme l’autocomplétion ou l’analyse de code en temps réel.

* Productivité réduite : les développeurs peuvent perdre du temps à configurer leurs outils ou à travailler sans les fonctionnalités dont ils ont besoin.
* Frustration : l’absence des bons outils peuvent entrainer des frustrations surtout pour les développeurs habitués à des environnements de développement plus riches.
* **Adoption dans l’industrie**

Rust est moins courant dans les entreprises, cela peut rendre difficile les recherches d’emplois pour les développeurs Rust. De nombreux entreprises n’ont pas encore formé leurs équipes à Rust ce qui limite son utilisation.

* Moins d’opportunité d’emplois : les développeurs Rust peuvent avoir moins d’opportunités d’emplois que ceux qui maitrisent des langages courants.
* Difficulté à trouver les mentors : les nouveaux développeurs peuvent avoir du mal à trouver les mentors ou des ressources pour apprendre Rust.
* **Intégration avec des projets existants**

Bien que Rust puisse interagir avec d’autres langages, intégration n’est pas toujours simple et nécessiter des efforts supplémentaire. Créer des bindings pour utiliser des bibliothèques C ou C++ peut être complexe et prend du temps

* Difficulté à migrer : les entreprises peuvent hésiter à adopter Rust pour des objets existants en raison des défis d’intégration.
* Couts supplémentaire : l’intégration peut nécessiter des ressources supplémentaires ce qui augmente les couts de projet.
* **Documentation et ressources**

Certaines parties de la documentation peuvent être trop techniques pour les débutants ce qui rend l’apprentissage plus difficile. Parfois les exemples fournis ne couvrent pas toutes les situations possibles ce qui peut laisser les développeurs dans l’incertitude.

* Difficulté à trouver des informations : les développeurs peuvent avoir du mal à trouver les informations dont ils ont besoin pour résoudre des problèmes.
* Frustration lors de l’apprentissage : les nouveaux utilisateurs peuvent se sentir perdus en raison d’un manque de clarté dans la documentation.

Bien que Rust soit un langage de programmation puissant avec de nombreux avantages, il présente également les désavantages qui peuvent poser les défis aux développeurs. Sa courbe d’apprentissage raide, son temps de compilation long, ses limites d’écosystème, sa complexité syntaxique et d’autres aspects peuvent rendre son utilisation difficile.

## **2.4. Avantages et désavantages de PYTHON**

### **2.4.1. Avantages**

* **Simplicité et lisibilité**

Python est conçu pour être intuitif, ce qui permet aux développeurs de se concentrer sur la logique plutôt que sur la syntaxe. Elle utilise l’indentation pour structurer les codes, cela rend les codes lisibles et aide à éviter les erreurs de syntaxe.

* Facilite l’apprentissage : les débutants peuvent apprendre rapidement les bases de la programmation.
* Maintenance simple : un code lisible est plus facile à maintenir et à mettre à jour.
* **Polyvalence**

Python est très populaire pour l’analyse de données grâce à des bibliothèques comme PANDAS et NUMPY, des bibliothèques comme TENSORFLOW et KERAS permettent de créer des modelés d’apprentissage automatique. Python est souvent utilisé pour écrire des scripts qui automatisent des tâches répétitives.

* Un seul langage pour plusieurs projets : Les développeurs peuvent utiliser python pour des différents types des projets, ce qui réduit les besoins d’apprendre plusieurs langages.
* Adaptabilité : python peut être utilisé dans des environnements variés, des petits scripts aux grandes applications.
* **Large écosystème de bibliothèques**

Python possède un vaste écosystème de bibliothèque pour les calculs numériques, l’analyse de données, pour la visualisation de données, pour faire des requetés HTTP facilement.

* Gain de temps : Les développeurs peuvent utiliser des bibliothèques existant au lieu de réécrire les codes.
* Communauté active : une grandes communauté signifie que les bibliothèques sont régulièrement mise à jour et améliorées.
* **Communauté active**

Des plateformes comme STACK OVERFLOW et REDDIT permettent aux développeurs de poser des questions et d’obtenir des réponses rapidement. De nombreux tutoriels gratuits et payants sont disponibles pour apprendre python.

* Soutien continu : les développeurs peuvent obtenir de l’aide rapidement lorsqu’ils rencontrent des problèmes.
* Partage de connaissance : Les membres de la communauté partagent des ressources des astuces et des meilleures pratiques.
* **Interopérabilité**

Python peut appeler des fonctions écrites en C ou C++, ce qui permet l’utilisation des bibliothèques performantes. Elle peut interagir facilement avec des API RESTFUL et des services web, ce qui facilite l’intégration avec d’autres systèmes.

* Flexibilité : les développeurs peuvent tirer parti des bibliothèques existantes dans d’autres langages.
* Migration facile : Les projets peuvent être progressivement migrés vers python sans avoir à tout réécrire.
* **Outils de développement**

Les outils de développement comme les éditeurs de code et les débogueurs sont essentiels pour aider les développeurs à écrire et à tester leurs codes. Python dispose de nombreux outils efficaces. Un environnement interactif pour le développement et l’analyse de données, un éditeur de code léger avec de nombreuses extensions pour python.

* Amélioration de la productivité : Les développeurs peuvent écrire et tester leur code plus rapidement.
* Débogage facilité : les outils permettent d’identifier rapidement les erreurs dans le code
* **Documentation et qualité**

**Python** est connu pour sa documentation de qualité, bien structurée et facile à comprendre. De nombreux exemples sont fournis pour illustrer les concepts.

* Facilite l’apprentissage : les débutants peuvent comprendre rapidement comment utiliser python.
* Réduit les erreurs : une bonne documentation aide à éviter les malentendus et les erreurs dans les codes.
* **Performance**

Python est interprété mais il compile le code en BYTECODE, ce qui améliore la vitesse d’exécution. De nombreuses bibliothèques python comme NUMPY, sont écrites en C pour des performances optimales.

* Suffisante pour de nombreuses applications : pour de nombreux projets, la performance de python est suffisante.
* Facilité de prototypage : les développeurs peuvent rapidement créer des prototypages et des applications fonctionnelles.
* **Utilisation dans l’éducation**

Les étudiants peuvent explorer des différents domaines, comme le développement web, la science de données et l’IA. Sa syntaxe claire et lisible le rend idéal pour les débutants.

* Formation rapide : Les étudiants peuvent acquérir des compétences en programmations rapidement.
* Préparation à l’industrie : Les compétences en python sont très demandées sur le marché du travail.
* **Flexibilité**

Python prend en charge la programmation orientée objet (P.O.O) ce qui permet de créer des programmes modulaires. Elle permet également d’utiliser des concepts de programmation fonctionnelle, comme les fonctions de première classe et les expressions LAMBDA.

* Adaptabilité : Les développeurs peuvent choisir le style de programmation qui convient le mieux à leur projet.
* Réutilisation du code : la flexibilité permet de créer des bibliothèques et des modules réutilisables.
* **Support multiplateforme**

Le code de python peut généralement être exécuté sur n’importe quelle plateforme sans modification, elle peut être installée facilement sur la plupart des systèmes d’exploitation (S .E).

* Accessibilité : Les développeurs peuvent travailler sur des différents systèmes sans se soucier de la compatibilité.
* Déploiement simplifié : Les applications peuvent être déployées sur les différentes plateformes sans avoir à réécrire les codes.

Python est un langage de programmation puissant et polyvalent qui offre de nombreux avantagent. Sa simplicité, sa flexibilité, sa large écosystème, et communauté active en font un choix idéal pour les développeurs de tous niveaux. Que ce soit pour le développement web, la science de données ou l’automatisation, python est un langage qui mérite d’être exploré et utilisé. Grace à ses nombreux avantages, elle continue de gagner la popularité et de s’imposer comme un outil essentiel dans le monde de la programmation moderne.

### **2.4.2. Désavantages**

* **Performance**

Python est un langage interprété, ce qui signifie que les codes sont exécutés ligne par ligne. Cela peut ralentir l’exécution par rapport à des langages compilés. Elle utilise un Garbage collector pour gérer la mémoire, ce qui peut introduire des délais supplémentaires pendant l’exécution.

* Application gourmandes en ressources : pour des applications nécessitant une exécution rapide comme les jeux vidéo ou les systèmes embarqués, python ne pas le meilleur choix.
* Problèmes de Scalabilité : Dans des applications à grande échelle, la lenteur d’exécutions peut devenir un obstacle.
* **Gestion de la mémoire**

Le garbage collector peut provoquer des pauses dans l’exécution du programme, ce qui peut être problématique pour les applications en temps réel. Bien que python gère la mémoire automatiquement, des fuites de mémoire peuvent survenir surtout si des références circulaires existent.

* Ralentissements : les pauses dues au garbage collector peut ralentir les applications surtout celle nécessitant une réactivité rapide.
* Complexité accrue : les développeurs doivent être conscients de la gestion de la mémoire, ce qui peut ajouter une couche de complexité à leurs codes.
* **Utilisation de la mémoire :** python utilise des types de données dynamiques ce qui nécessite plus de mémoire pour stocker les informations sur les types.

Application lourds : python peut nécessiter plus de ressource, ce qui peut être problématique sur des systèmes avec des ressources limité.

* Couts supplémentaire : pour des applications nécessitant beaucoup de mémoire, cela peut entrainer des couts supplémentaires en infrastructure.
* **Concurrence et multithreads**

Python utilise un mécanisme appelé GIL qui empêche plusieurs threads d’exécuter les codes python en même temps, cela limite les performances des programmes multithread. La gestion de la concurrence en python peut être complexe et nécessite une bonne compréhension des concepts de threading.

* Performances réduites : les applications qui nécessitent une exécution parallèle peuvent ne pas tirer parti de la puissance des processeurs multicoeurs.
* Complexité accrue : Les développeurs doivent souvent recourir à des solutions alternatives comme l’utilisation de processus au lieu de threads, ce qui complique le développement.
* **Dépendances et Gestion des paquets**

Parfois, l’installation de certaines bibliothèques peut être compliquée en particulier celles qui nécessitent des compilations spécifiques. Les projets peuvent rencontrer des conflits entre des différentes versions de bibliothèques ce qui complique la gestion des dépendances

* Difficultés lors du développement : Les conflits de dépendances peuvent entrainer des erreurs difficiles à diagnostiquer.
* Temps perdu : les développeurs peuvent passer beaucoup de temps à résoudre des problèmes liés aux dépendances au lieu de se concentrer sur le développement.
* **Limitations de l’écosystème**

Bien que python ait un écosystème riche, il présente des limitations. Certaine bibliothèques python ne sont pas aussi performantes que leurs homologuent dans d’autres langages, ce qui peut limiter leur utilisation dans des applications critiques.

* Difficulté de l’utilisation : les développeurs peuvent rencontrer des problèmes en essayant d’utiliser des bibliothèques mal documentées ou non optimisées. Bien que python ait une bonne documentation, certaines bibliothèques tierces peuvent avoir une documentation incomplète ou confuse.
* Frustration : les limitations de l’écosystème peuvent entrainer de la frustration surtout pour les projets complexes.
* **Problème des sécurités**

La sécurité en programmation concerne la protection des applications contre les menaces et les vulnérabilités, python comme tout autre langage présente des défis en matière de sécurité. Certaines bibliothèques python peuvent contenir de vulnérabilité qui peut être exploitées par des attaques. La manipulation de données sensibles comme les mots de passe nécessite des précautions spécifiques qui ne sont pas toujours évidentes en python

* Risques accrus : les applications python peuvent être exposées à des risques si les développeurs ne sont pas attentifs aux problèmes.
* Couts de remédiation : les failles de sécurité peuvent entrainer des couts élevés pour les entreprises en cas de violation de données.
* **Typage dynamique**

Le typage dynamique signifie que les types de données d’une variable sont déterminés au moment de l’exécution plutôt qu’à la compilation, bien que cela ait des avantages, cela présente également des inconvénients. Les erreurs liées aux types peuvent ne pas être détectées avant l’exécution, ce qui peut entrainer de bogues difficiles à diagnostiquer. Le code peut devenir difficile à maintenir surtout dans les projets de grande taille, car les types ne sont pas explicitement définis.

* Problème de débogage : les développeurs peuvent passer beaucoup de temps à déboguer des erreurs liées aux types.
* Complexité accrue : la maintenance de code peut devenir plus complexe, ce qui peut ralentir le développement.
* **Difficulté d’intégration avec des systèmes existant**

**L’intégration** de système fait référence à la capacité d’un langage à fonctionner avec d’ autres systèmes ou langage, bien que python interagir avec d’autre langage l’intégration n’est pas toujours simple et peut nécessiter des efforts supplémentaires.

* Difficulté à migrer : les entreprises peuvent hésiter à adopter python pour des projets existants en raison des défis d’intégration.
* Coût supplémentaire : l’intégration peut nécessiter des ressources supplémentaires, ce qui augmente les coûts du projet.
* **Dépendance à l’égard des versions**

Les mises à jour de bibliothèque peuvent introduire des incompatibilités ce qui peut casser le code existants, certaines bibliothèque peuvent devenir obsolète ou ne plus être maintenue ce qui complique la mise à jour des projets.

* Difficulté de mise à jour : les développeurs peuvent rencontrer des problèmes lors de la mise à jour de leurs projets vers de nouvelles versions de bibliothèque.
* Temps perdu : la gestion des versions peut entrainer des retards dans le développement.
* **Manque de support pour le multithreads**

Comme mentionné précédemment les threads s’exécute du code python simultanément ce qui limite l’efficacité du multithreading, la gestion des threads en python peut être complexe et nécessite une bonne compréhension des concepts de concurrence.

* Performance réduite : les applications qui nécessitent une exécution parallèle peuvent ne pas tirer parti de la puissance des processeurs multicoeurs.
* Complexité accrue : les développeurs doivent souvent recourir à des solutions alternatives comme l’utilisation de processus au lieu de threads ce qui complique le développement
* **Problèmes de compatibilité**

La compatibilité fait référence à la capacité d’un langage à fonctionner sur les différentes plateformes et version, les différences entre python 2 et python 3 peuvent pose des problèmes des compatibilités surtout pour les anciens projets. Certains bibliothèques peuvent ne pas être mises à jour pour fonctionner avec les derniers versions de python.

* Difficulté de migration : les développeurs peuvent rencontrer des difficultés lors de la migration des projets vers de nouvelle version de python.
* Temps perdu : les problèmes de compatibilité peuvent entrainer des retards dans le développement.
* **Limitation de python dans le développement de jeux**

Les jeux nécessitent souvent des performances élevées et python peut ne pas être en mesure de répondre à ces exigences, bien qu’il existe des bibliothèques comme pygame, elle n’est pas aussi robuste que celle disponible dans les langages c# ou c++.

* Opportunités limitées : les développeurs python peuvent manquer d’opportunités dans le domaine du développement de jeux.
* Difficultés de déploiement : les jeux développés en python peuvent rencontrer des difficultés lors du déploiement sur des plateformes de jeu.

Bien que python soit un langage de programmation puissant et polyvalent, il présente aussi des nombreux désavantages, sa plateforme, sa gestion de la mémoire, ses limitations en matière de concurrence et d’autres facteurs peuvent poser des défis aux développeurs. Il est important de poser ces inconvénients par rapport aux avantages offert par python en comprenant ces défis les développeurs peuvent mieux se préparer à travailler avec ce langage et maximiser son potentiel dans leurs projets.

## **2.5. Comparaison entre RUST et PYTHON (syntaxe, code, ide, éditeurs, mots clés, hard, soft,** ….)

## **Tableau 2.1 : Comparaison Rust et python**

|  |  |  |
| --- | --- | --- |
| **Critère** | **Rust** | **Python** |
| Syntaxe | Strict, plus verbeux, typée statiquement, proche de **C/C++** | Simple, lisible, typée dynamiquement |
| Exemple de code | fn main () {println! (‘’Hello, world!’’) ;} | Print (‘’Hello, world !’’) |
| Paradigmes | Orienté système, fonctionnel, impératif | Orienté objet, fonctionnel, impératif |
| Mots-clés | Fn, let, struct, enum, match, impl | Def, class, import, if, for, while |
| IDE populaires | Visual studio code, intellij Rust, Clion | Pycharm, Visual studio code, jupyter notebook |
| Editeurs de texte | Vim, Emacs, sublime text, vs code | Vs code, atom, sublime text, thonny |
| Gestionnaire de paquets | Cargo (intégré, puissant) | Pip, conda |
| Hard | Performance élevées, compilé, idéal pour les systèmes | Interprété, moins performant pour les applications lourdes |
| Soft | Concurrence sécurisée, gestion de la mémoire | Facilité d’utilisation, riche écosystème de bibliothèque |
| Performance | Très élevée, sans garbage collector | Moyenne, dépendante du GIL et de l’interpréteur |
| Sécurité mémoire | Modèle d’emprunt et de propriété, erreurs détectées à la compilation | Garbage collector, erreurs souvent détectées à l’exécution |
| Utilisation | Développement de système, applications performantes | Développement web, science de données, automatisation |
| Apprentissage | Courbe d’apprentissage plus rapide, nécessite rigueur | Facile à apprendre, idéal pour débutants |

RUST : est préférable pour les applications où la performance et la sécurité de la mémoire sont importantes, bien qu’il nécessite un investissement de temps plus important pour l’apprentissage.

PYTHON : est idéal pour les débutants et pour les développements des taches nécessitant un développement rapide et une grande bibliothèque de ressources.

## **2.6. Disposition**

La disposition ou l’architecture des langages python et Rust se réfère à la manière dont ils sont structuré et utilisés.

### **2.6.1 Disposition de Rust**

1. **Architecture**

**Compilé** : Rust est un langage compilé, ce qui signifie que le code est transformé en code machine avant l’exécution, offrant ainsi de meilleures performances.

**Typage statique** : les types de données sont vérifiés à la compilation, ce qui permet de détecter les erreurs avant l’exécution.

1. **Module et crate**

**Module**: Rust utilise un système de modules pour organiser le code, permettant une encapsulation et une réutilisation.

**Crate**: les bibliothèques Rust sont appelées ‘’Crates’’ elles peuvent être partagées et utilisées via le gestionnaire de paquets cargo.

1. **Bibliothèque**

**Standard Library :** Rust dispose également d’une bibliothèque standard, mais elle est plus petite que celle de python

**Ecosystème :** l’écosystème Rust est en pleine expansion, avec des Crates pour de nombreux domaine (webAssembly, système etc…)

1. **Environnement d’exécution**

**Compilation :** Rust utilise **rustc** pour compiler le code. Les projets sont souvent gérés avec cargo, qui facilite la gestion des dépendances et la compilation.

### **2.6.2 Disposition de python**

1. **Architecture**

**Interprété :** python est un langage interprété, ce qui signifie que le code est exécuté ligne par ligne par un interpréteur.

**Typage dynamique :** les types de données sont déterminés à l’exécution, ce qui facilite la flexibilité mais peut introduire des erreurs à l’exécution.

1. **Module et paquets**

**Module :** python utilise des modules pour organiser le code chaque fichier python peut être un module.

**Paquets :** les modules peuvent être regroupés en paquets pour une meilleure organisation.

1. **Bibliothèque**

**Standard Library :** python dispose d’une vaste bibliothèque standard qui couvre des nombreux domaines (réseaux, traitement de données etc….)

**Ecosystème :** des bibliothèques tierces comme NumPy, Pandas et Flask enrichissent encore les fonctionnalités.

1. **Environnement d’exécution**

**Interpréteurs :** python peut être exécuté dans des divers environnements, y compris les IDE comme (Pycharm) les notebooks (jupyter) et les environnements en ligne

Le choix entre les deux dépendra des besoins spécifiques du projet, des compétences de l’équipe et des objectifs de performance.

## **2.7. Conclusion**

En conclusion, il n’existe pas de langage parfait. Rust et python ont chacun leurs avantages et les inconvénients. Le choix entre les deux doit être fait en fonction des exigences spécifiques des projets, des compétences des développeurs et des objectifs à long terme. En comprenant ces différences, les développeurs peuvent mieux s’orienter dans le paysage des langages de programmation et choisir celui qui convient le mieux à leurs besoins.

# **CHAPITRE 3 ANALYSE DE PERFORMANCE ET D’UTILISATION DES RESSOURCES (Rust et python)**

## **3.1 Introduction**

Dans le développement de logiciel, la performance et l’utilisation efficace des ressources sont des aspects importants. Ce chapitre se penche sur deux langages de programmation populaire « RUST et PYTHON ». Chacun de ces langages à ses propres caractéristiques, avantages, inconvénients en matière de performance et d’utilisation des ressources. Rust est souvent salué pour sa rapidité et sa gestion de la mémoire tandis que python est apprécié pour sa simplicité et sa flexibilité.

## **3.2 Consommation de mémoire**

La consommation de mémoire est un aspect essentiel à considérer lors du développement d’applications. Elle influence non seulement les performances d’un programme mais aussi son efficacité et capacité à fonctionner sur les différentes plateformes.

### **3.2.1 La gestion de la mémoire en Rust**

1. **Modèle de propriété**

Rust adopte un modèle de gestion de la mémoire basé sur la propriété, chaque valeur à un propriétaire et ce propriétaire est responsable de la libération de la mémoire. Ce modèle présente plusieurs avantages

* Sécurité : Rust garantit qu’il n’y’a pas de fuites de mémoire ou d’accès à des zones de mémoire non valides
* Pas de ramasse-miettes : contrairement à d’autres langages, Rust ne nécessite pas de ramasse-miette, ce qui réduit la surcharge en mémoire.

1. **Emprunts et références**

Rust utilise également des emprunts pour remettre à plusieurs parties d’accéder à une même valeur sans le copier. Cela contribue à une utilisation efficace de la mémoire

* Emprunts immuable : permettent à plusieurs parties de lire une valeur sans la modifier
* Emprunts mutable : permettent à une seule partie de modifier une valeur, garantissant ainsi la sécurité des données.

1. **Allocation**

Rust utilise un système d’allocation à la compilation pour gérer la mémoire

* Allocation sur la pile : les types de données simples sont généralement alloués sur la pile, ce qui est rapide et efficace.
* Allocation sur le tas : pour des structures de données plus complexes, Rust utilise le tas mais avec une gestion stricte de la mémoire pour éviter les fuites.

### **3.2.2 La gestion de la mémoire en Python**

1. **Modèle de gestion de la mémoire**

Python adopte une approche différente pour la gestion de la mémoire

* Ramasse-miette : python utilise un ramasse-miette pour gérer la mémoire. Cela signifie que le langage libère automatiquement la mémoire qui n’est plus utilisée. Bien que cela simplifie la gestion de mémoire pour les développeurs, cela peut introduire des pauses dans l’exécution du programme.

1. **Objets et allocation**

Dans python, tout est un objet ce qui a des implications sur la consommation de mémoire

* Surcoût des objets : chaque objet en python à un certain surcout en mémoire en raison de la gestion des métadonnées (comme le type et la référence).
* Allocation dynamique : la mémoire est allouée dynamiquement, ce qui peut entrainer des fragments de mémoire et une utilisation moins efficace

1. **Gestion des cycles de référence**

Python gère les cycles de référence lorsque deux objets se réfèrent mutuellement grâce à un compteur de référence

* Compteur de référence : chaque objet en python à un compteur de référence qui suit combien de références pointe vers lui, lorsque ce compteur atteint zéro, l’objet est libéré.
* Ramasse-miette pour les cycles : un ramasse-miette supplémentaire est nécessaire pour gérer les cycles de référence, ce qui peut ajouter une surcharge.

### **3.2.3 Comparaison de la consommation de mémoire**

1. **Allocation de mémoire**

RUST : la mémoire est allouée de manière statique ou dynamique, mais avec un contrôle strict. Les allocations sur la pile sont rapides et efficaces.

PYTHON : la mémoire est allouée dynamiquement ce qui peut entrainer une fragmentation et un surcout en mémoire.

1. **Utilisation de la mémoire**

RUST : en raison de son modelé de propriété, Rust utilise généralement moins de mémoire pour des structures de données similaires. La gestion stricte de la mémoire permet d’éviter les fuites.

PYTHON : la consommation de mémoire est souvent plus élevée en raison de la nature des objets et de L’OVERHEAD (elle désigne les ressources supplémentaires, temps, mémoire et complexité) associé à chaque objet.

### **3.2.4 Implication pour le développement**

1. **Choix du langage**

Le choix entre Rust et Python dépend fortement des exigences de l’application

Application critique en mémoire : Rust est généralement préférable pour le système nécessitant un contrôle strict de la mémoire

Développement rapide : Python est souvent choisi pour sa simplicité et sa rapidité de développement, même si cela implique une consommation plus élevée.

1. **Optimisation**

Rust : les développeurs doivent être conscients de la gestion de la mémoire, mais ils bénéficient d’une performance accrue.

Python : les développeurs peuvent utiliser des bibliothèques optimisées pour réduire la consommation de mémoire, mais ils doivent être conscients des limitations du ramasse-miettes.

### **3.2.5 Outils de mesure de la consommation de mémoire**

1. **Outil pour Rust**

Valgrind**:** un outil pour détecter les fuites de mémoire et analyse l’utilisation de la mémoire

Heaptrack : permet d’analyser l’utilisation de la mémoire et de visualiser les allocations

1. **Outils pour python**

Memory \_ profilé : un module pour mesurer la consommation de mémoire d’un programme python

Objgraph : un outil pour visualiser les objets et les références dans un programme python.

La consommation de mémoire est un facteur clé à considérer lors du choix entre Rust et Python. Rust offre une gestion de la mémoire plus efficace grâce à son modèle de propriété, tandis que python bien qu’il soit plus simple à utiliser, peut avoir une consommation de mémoire plus élevée en raison de son ramasse-miettes et de la nature des objets.

## **3.3 Optimisation spécifiques des langages (Rust & Python**)

L’optimisation du code est essentielle pour améliorer les performances des applications, chaque langage de programmation à ses propres techniques et stratégies d’optimisation, adaptées à ses caractéristiques et à son utilisation.

### **3.3.1 Optimisation dans Rust**

1. **Modelé de propriété et gestion de la mémoire**

Rust se distingue par son modèle de propriété, qui joue un rôle important dans l’optimisation. Pas de ramasse-miettes en éliminant le besoin d’un ramasse-miette, Rust réduit la surcharge de la gestion de mémoire ce qui améliore les performances.

Gestion statique de la mémoire, les allocations de mémoire son effectuées à la compilation permettant des optimisations l’exécution.

1. **Concurrence**

Rust facilite l’écriture de programmes concurrentiels grâce à son modèle de sécurité. La sécurité des données, grâce à la gestion des emprunts et des références, Rust permet de créer des applications concurrentes sans risque de conditions.

Utilisation de threads : threads en Rust sont légers et permettent une utilisation efficace des ressources système.

1. **Optimisation au niveau du compilateur**

Le compilateur Rust (**Rustc**) applique plusieurs optimisations

Inlining : le compilateur peut remplacer les appels de fonction par le code de la fonction elle-même pour réduire le temps d’exécution.

L’optimisation des boucles : les boucles peuvent être optimisées pour réduire le nombre d’interactions ou améliorer la vitesse d’exécutions.

1. **Utilisation de bibliothèques optimisées**

Crates : les crates (bibliothèque) comme rayon pour le traitement parallèle permettent d’améliorer les performances des applications.

Bibliothèque c : Rust peut appeler des bibliothèques c, permettant d’utiliser des algorithmes optimisés.

### **3.3.2 Optimisation dans Python**

1. **Gestion de la mémoire**

Python utilise un ramasse-miettes pour gérer la mémoire, ce qui peut parfois entrainer des ralentissements. Bien que cela simplifie la gestion de la mémoire, il peut introduire des pauses. Pour optimiser, il est important de minimiser les allocations inutiles.

Types de données appropriés : utiliser les types de données intégrés les plus efficaces comme ‘’ les tuples au lieu des listes’’ peut réduire la consommation de mémoire.

1. **Utilisation de bibliothèques optimisées**

Python dispose de nombreuses bibliothèques optimisées pour améliorer les performances.

NumPy : pour les calculs numériques, NumPy utilise des tableaux multidimensionnels et est beaucoup plus rapide que les listes python standard.

Cython : permet de compiler le code python en C, ce qui améliore considérablement la vitesse d’exécution.

1. **Optimisation des algorithmes**

L’optimisation des algorithmes est important en python, utiliser des algorithmes efficaces (comme les algorithmes de tri optimisés) peut réduire le temps d’exécution. Utiliser des compréhensions de liste au lieu de boucles for classique peut améliorer la lisibilité et les performances.

1. **Profilage et analyse**

Pour optimiser le code python, il est essentiel de mesurer les performances

Cprofile : un module intégré pour profiler le code et identifier les goulets d’étranglement.

Timeit : un module pour mesurer le temps d’exécution de petits morceaux de code.

### **3.3.4 Comparaison des optimisations**

1. **Facilité d’utilisation**

**Rust :** bien que Rust offre d’excellentes performances, sa courbe d’apprentissage peut être plus rapide en raison de son modèle de propriété et de sa gestion de la mémoire.

**Python :** python est réputé pour sa simplicité et sa lisibilité, ce qui facilite le développement rapide, même si cela peut implique des compromis sur les performances.

1. **Cas d’utilisation**

**Rust :** idéal pour les applications nécessitant des performances élevées, comme les systèmes embarqués, les moteurs de jeux et les applications concurrentes.

**Python :** préféré pour le développement rapide l’analyse de données et le prototypage bien que des optimisations soient nécessaires pour les applications à forte intensité de calcul.

1. **Conseils d’optimisation**

**Rust :** elle privilégier les emprunts pour les éviter les copies inutiles et optimiser l’utilisation de la mémoire. Rust utilise des crates bien établies pour le bénéficier d’optimisations déjà mises en place.

**Python :** Python profile le code pour identifier les goulets d’étranglement et les optimiser. Elle choisit les types de données les plus efficaces pour réduire la consommation de mémoire.

L’optimisation de code est un aspect fondamental du développement logiciel et chaque langage à ses propres techniques et stratégies. Rust offre des performances élevées grâce à son modèle de propriété et à une gestion efficace de la mémoire tandis que Python bien qu’il soit plus simple à utiliser, peut nécessiter des optimisations supplémentaires pour atteindre des performances similaires. En comprenant les optimisations spécifiques à chaque langage, les développeurs peuvent choisir le bon outil pour leurs besoins et créer des applications plus efficaces et performantes. Que ce soit pour des projets à grande échelle ou des prototypes rapides, la connaissance des meilleures pratiques d’optimisation est essentielle pour réussir dans le développement logiciel moderne.

## **Performances**

La performance désigne la rapidité d’un programme et sa capacité à utiliser efficacement les ressources. Dans le monde de la programmation, la performance d’un langage est un critère essentiel à considérer lors du développement d’applications. Les performances peuvent influencer la vitesse d’exécution, l’utilisation de la mémoire et la capacité à gérer des charges de travail élevées.

### **3.4.1 Rust**

Rust est un langage de programmation système conçu pour être rapide, sur et concurrent. Il a été créé pour éviter les erreurs courantes de gestion de la mémoire, comme les fuites et les accès non autorisés. Quelques caractéristique clés de Rust

Sécurité de la mémoire : Rust utilise un système de propriété qui garantit la sécurité de la mémoire sans avoir besoin d’un ramasse-miettes.

Performance : Rust est compilé en code machine, ce qui le rend très rapide.

Concurrence : Rust facilite l’écriture de programmes concurrentiels sans risque de conditions de course.

### **3.4.2 Python**

Python est un langage de programmation interprété, connu pour sa simplicité et sa lisibilité. Il est largement utilisé dans le développement web, l’analyse de données, l’intelligence artificielle (I.A) et bien d’autres domaines. Quelques caractéristique clés de Python.

Facilité d’utilisation : python est facile à apprendre et à utiliser, ce qui en fait un choix populaire pour les débutants.

Interprété : python est un langage interprété, ce qui signifie qu’il est généralement plus lent que les langages compilés comme Rust.

Écosystème riche : python dispose d’une vaste bibliothèque de module et de Framework qui facilite le développement.

### **3.4.3 Comparaison des performances**

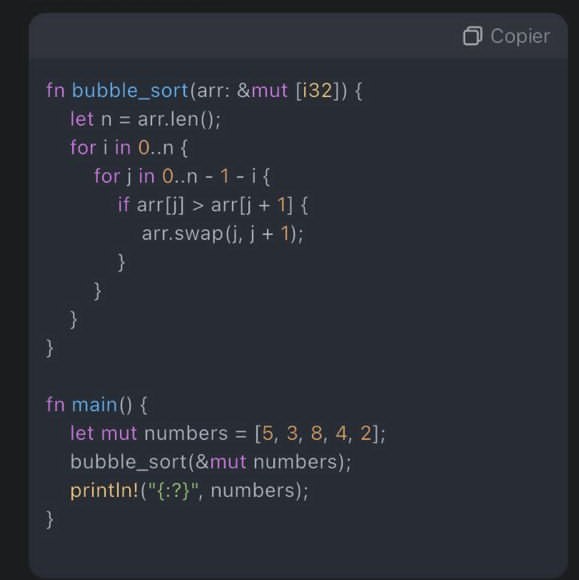
Pour comparer les performances de Rust et python, nous allons examiner plusieurs (le temps d’exécution, l’utilisation de la mémoire, la Scalabilité et la facilité de développement.

1. **Temps d’exécution**

Le temps d’exécution est l’un de critères le plus importants pour évaluer les performances d’un langage.

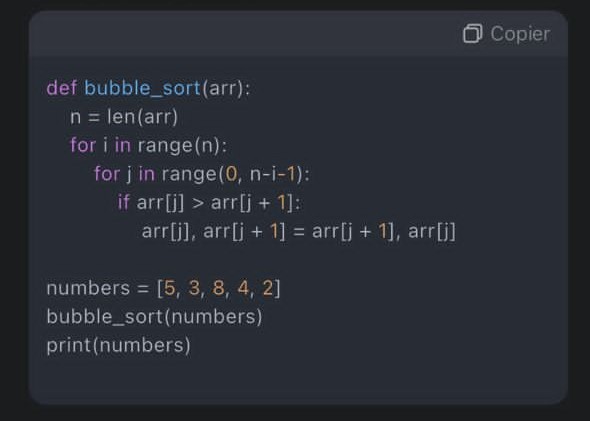
* **Exemple** : Nous allons mesurer le temps nécessaire pour exécuter des algorithmes similaires dans les deux langages. En compara l’exécution d’un algorithme tri simple tel que : le tri à bulles (bubble), dans les deux langages.

**RUST**



# **FIGURE 3.1 : code Rust**

**PYTHON**

****

**FIGURE 3.2 : code python**

* **Résultat :** en exécutant ces deux algorithmes de tri, nous pouvons mesurer le temps d’exécution. En général, le tri en Rust sera plus rapide en raison de la compilation et des optimisations du code.

1. **Utilisation de la mémoire**

C’est un indicateur clé des performances, nous allons analyser la consommation de mémoire des programmations de mémoire des programmes écrits en Rust et Python.

* **Rust**: permet d’analyser l’utilisation de la mémoire à l’aide de d’outils comme **valgrind.** En exécutant le programme tri, nous pouvons obtenir des informations sur la mémoire
* **Python**: en python nous pouvons utiliser le module memory\_profiler pour mesurer la consommation de mémoire

**Résultat :** en général, Rust aura une consommation de mémoire plus faible que python pour des algorithmes similaire, en raison de sa gestion stricte de la mémoire.

1. **Scalabilité**

La Scalabilité est la capacité d’un programme à gérer une augmentation de la charge de travail

* **Rust : elle** est conçue pour être performant même avec une charge de travail élevée. Les programmes Rust peuvent tirer parti de la concurrence et des threads pour améliorer les performances.
* **Python :** python peut également gérer des charges de travail élevées mais il peut rencontrer des limitations en raison du GIL (Global interpreter lock) qui empêche l’exécution simultanée de plusieurs threads.

**Résultat :** Rust est mieux adapté pour des applications nécessitant une haute scalabilité grâce à sa gestion efficace des ressources et à sa capacité à exécuter des taches en parallèle. Python bien qu’il soit capable de gérer des charges de travail élevées, est limités par le GIL, ce qui peut affecter ses performances dans des scénarios hautement concurrentiels.

1. **Facilité de développement**

* Rust : elle est réputée pour sa courbe d’apprentissage plus rapide, en raison de son système de propriété et de son modèle de gestion de la mémoire. Cependant une fois maitrisé, il permet de développer des applications très performantes.
* Python : est très accessible et facile à apprendre, ce qui en fait un excellent choix pour les débutants. La syntaxe claire et concise permet un développement rapide même si cela peut parfois se faire au détriment des performances.

**Résultat :** Rust et python répondent à des besoins différents en matière de facilité d’utilisation. Rust est plus difficile à maitriser mais offre des performances exceptionnelles tandis que python est accessible et rapide à apprendre ce qui en fait un excellent choix pour le débutants et pour le développement rapide d’application, même si cela peut parfois affecter les performances.

1. **Cas d’utilisation**

Rust : est souvent utiliser dans des applications nécessitant des performances élevées et un contrôle précis de la mémoire comme :

* Système embarqué : pour des applications où les ressources sont limitées.
* Jeux vidéo : pour des moteurs de jeu nécessitant des performances optimales.
* Applications serveur : pour gérer des charges de travail élevées avec une consommation de mémoire réduite.

Python : est largement utilisé dans des domaines où la rapidité de développement est important tels que :

* Analyse de données : grâce à des bibliothèques comme pandas et NumPy.
* Développement web : avec des Framework comme Django et Flask.
* Intelligence artificielle : grâce à des bibliothèques comme Tensorflow et pyTorch.

1. **Outils de mesure des performances**

**Outils pour Rust**

* Cargo : le gestionnaire de paquets de Rust, qui inclut des outils de benchmarking.
* Valgrind : pour analyser l’utilisation de la mémoire et détecter les fuites.

**Outils pour python**

* Cprofile : un module intégré pour profiler le code python.
* Timeit : pour mesurer le temps d’exécution de petits morceaux de code.
* Memory\_profiler : pour analyser la consommation de mémoire.

Rust et python ont des performances très différentes en fonction de leur conception et de leur utilisation. Rust est généralement plus rapide et plus efficace en termes de gestion de la mémoire, ce qui en fait un choix idéal pour les applications nécessitant des performances élevée. Python est plus facile à apprendre et à utiliser ce qui en fait un excellent choix pour le développement rapide et l’analyse de données.

Le choix entre Rust et python dépendra des besoins spécifiques de chaque projet. Les développeurs doivent évaluer les exigences en matière des performances, de scalabilité et de facilité de développement pour choisir le langage le mieux adapté à leur application.

## **3.5 Utilisation des ressources**

L’utilisation des ressources fait référence à la manière dont un programme utilise la mémoire, le processeur, et d’autres ressources matérielles. Une utilisation efficace des ressources est essentielle pour :

* Performance : des programmes réactifs et rapides.
* Scalabilité : capacité à gérer des charges de travail croissantes sans dégradation des performances.
* Durabilité : réduction de la consommation d’énergie et des couts d’exploitation.

**Types des ressources**

* Mémoire : quantité de RAM utilisée par le programme.
* Processeur : utilisation du CPU pour exécuter des instructions.
* Entrées/sorties : temps pris pour lire ou écrire des données.

### **3.5.1 Utilisation des ressources en Rust**

1. **Gestion de la mémoire**

Rust utilise un modèle de propriété qui régule comment la mémoire est allouée et libérée. Voici quelques caractéristique clés

* Pas de ramasse-miette : Rust ne dispose pas d’un ramasse-miette, ce qui réduit la surcharge liées à la gestion de la mémoire.
* Sécurité de la mémoire : grâce au système de propriété, Rust garantit qu’il n’y’a pas des fuites de mémoire, d’accès concurrent non sécurisé ou de données non valides.

1. **Performance du processeur**

* Optimisation : le compilateur Rust (LLVM) optimise le code à la compilation, ce qui permet d’obtenir des performances proches du c ou du c++
* Concurrence : Rust facilite l’écriture de code concurrent et parallèle permettant une utilisation efficace des processeurs multicoeurs.

1. **Efficacité des entrées/sorties**

* Traitement asynchrone : Rust propose des bibliothèques comme **tokio** pour gérer les opérations d’E/S de manière asynchrone, ce qui permet d’améliorer les performances des applications nécessitant des opérations d’E/S intensive.
* Gestion des ressources : Rust permet de gérer les ressources de manière fine, ce qui est particulièrement utile pour les applications systèmes et embarquées.

### **3.5.2 Utilisation des ressources en python**

1. **Gestion de la mémoire**

Python utilise un ramasse-miettes pour gérer la mémoire

* Ramasse-miette : cela signifie la gestion de la mémoire, mais peut introduire des pauses imprévisibles lors de l’exécution du programme.
* Fuites de mémoire : python peut parfois rencontrer des fuites de mémoire, notamment avec des références circulaires.

1. **Performance du processeur**

* Interprété : python est un langage interprété, ce qui signifie qu’il est généralement plus lent que les langages compilés comme Rust.
* Global interpreter lock : le GIL empêche l’exécution simultanée de plusieurs threads ce qui limite l’utilisation des ressources CPU dans les applications multithreadées.

1. **Efficacité des E/S**

* Bibliothèques d’E/S : python dispose des bibliothèques robustes pour gérer les entrées/sorties comme (asyncio) qui permet une programmation asynchrone.
* Facilité d’utilisation : les opérations d’E/S en python sont généralement plus faciles à mettre en œuvre mais peuvent être moins performantes que celles en Rust

### **3.5.3 Comparaison directe entre Rust et Python**

1. **Utilisation mémoire**

**3.1 Tableau de comparaison d’utilisation mémoire**

|  |  |  |
| --- | --- | --- |
| **Critère** | **Rust** | **Python** |
| Gestion de la mémoire | Propriété, sans ramasse miette | Ramasse-miettes, peut avoir des fuites |
| Performance | Très efficace | Moins efficace |

1. **Utilisation du processeur**

**3.2 Tableau de comparaison d’utilisation de processeur**

|  |  |  |
| --- | --- | --- |
| **Critère** | **Rust** | **Python** |
| Exécution | Compilé, optimisé | Interprété, limité par le GIL |
| Concurrence | Excellente (sans GIL) | Limité par le GIL |

1. **Efficacité des E/S**

**Tableau de comparaison d’efficacité des E/S**

|  |  |  |
| --- | --- | --- |
| **Critère** | **Rust** | **Python** |
| Asynchrone | Très efficace | Prise en charge, mais moins performant |
| Facilité d’utilisation | Plus complexe | Très accessible |

### **3.5.4 Cas d’utilisation**

1. **Application nécessitant des performances élevées**

**Rust**: idéalpour les systèmes embarqués, les moteurs de jeux et les applications nécessitant une gestion fine des ressources.

**Python**: moins adapté mais peut être utilisé pour des prototypes ou des applications moins exigeants.

1. **Développement rapide**

**Rust** peut nécessiter plus de temps pour le développement en raison de la complexité du langage. **Python** excellente option pour le prototypage rapide et le développement agile.

Rust et python ont chacun leur forces et faiblesse en matière d’utilisation des ressources, Rust excelle dans les performances et gestion de la mémoire tandis que python offre une facilité d’utilisation et une rapidité de développement.

Le choix entre les deux dépendra des besoins spécifiques du projet de l’expérience de l’équipe de développement et des exigences en matière des performances.

## **3.6. Exemple d’algorithme traduit dans les deux langages**

### **3.6.1. Algorithme de tri par sélection en pseudo-code**

Pour i allant de 0 à n-1 :

Trouver l’indice du plus petit élément dans la partie non triée

Échanger cet élément avec l’élément à la position i

### **3.6.2. Programme équivalent en Python**

def selection\_sort(arr):

n = len(arr)

for i in range(n):

min\_index = i

for j in range(i+1, n):

if arr[j] < arr[min\_index]:

min\_index = j

arr[i], arr[min\_index] = arr[min\_index], arr[i]

return arr

Exemple d'utilisation

liste = [64, 25, 12, 22, 11]

print("Liste triée :", selection\_sort(liste))

### **Programme équivalent en Rust**

fn selection\_sort(arr: &mut Vec<i32>) {

let n = arr.len();

for i in 0..n {

let mut min\_index = i;

for j in (i+1)..n {

if arr[j] < arr[min\_index] {

min\_index = j;

}

}

arr.swap(i, min\_index);

}

}

fn main() {

let mut liste = vec![64, 25, 12, 22, 11];

selection\_sort(&mut liste);

println!("Liste triée : {:?}", liste);

}

**3.7 Conclusion**

Dans ce chapitre, nous avons appris à comprendre et à optimiser l’utilisation de la mémoire et les performances de nos programmes. Nous avons comparé les performances de différentes approches et de structure de données, afin de créer des programmes plus rapide et plus fluides en maitrisant ces concepts, vous serez en mesure de développer des applications performantes et économes en ressources.

# **Conclusion général**

Commençons par le chapitre 1, ou nous avons abordé les généralités sur la programmation informatique, nous avons appris ce qu’est un langage de programmation informatique et les différent processus de développement logiciel.

Ensuite le chapitre 2, nous avons étudié les avantages et les inconvénients de deux langages populaires : Rust et python. Nous avons vu comment ces langages ces distinguent en termes de syntaxe, de concept et de domaine d’application, cette comparaison nous a aider à saisir les critères à prendre en compte mors du choix d’un langage de programmation adapté à nos besoins.

Enfin le chapitre 3 s’est concentré sur l’analyse de la performance et de l’utilisation des ressources, en mettant l’accent sur Rust et Python. Nous avons appris à comprendre la consommation de mémoire de nos programmes, à gérer efficacement la mémoire optimiser les performances.

# **BIBLIOGRAPHIE**

1. **Ouvrages**

* GRATWITZ M., méthode de recherche en science social, 8eme édition, Ed DALLOZ, paris 1996, cité par KIMWENA Jackin
* CASTELLANI X., méthode générale d’analyse des applications informatiques, Ed PUF, Paris, 1986, cité par KIMWENA Jackin

1. **Notes de cours inédit**

* TAHONDO Joseph, algorithmique, LPTA1 II, ispt-kin, KINSHASA, 2022
* KAPUTU George, POO, LPTA2 II, ispt-kin, Kinshasa, 2023

1. **TFC, mémoires, projets**

Didier MAYI, « Etude et mis en place d’un système informatique pour l’inventaire » cas de l’institut saint joseph/Bonsomi, travail de fin de cycle, ISPT-KIN, Kinshasa, 2017

1. **Sites internet**

* [fr.m.wikipedia](http://www.wikipedia).org, consulté  le 30/04/2025 à 19h’
* Rust programming language [https://doc.rust-lang.org/book/] <https://.doc.rust-lang.org/book/>
* Python.documentation.[https://docs.python.org/3/] <https://.docs.python.org/3/>
  + - * <https://fr.wikipedia.org/wiki/Hypertexte>
      * https :fr.wikipedia.org/wiki/Internet
      * <http://www.ifeda.org.tn/files_uploads/16329d571905336d78aac74f50e7d464.pdf>
      * <https://www.uv.es/nemiche/cursos/polycopies/5%20Merise.pdf>
      * [20est%20un%20ensemble,le%20papier%20ou%20le%20textile](https://fr.wikipedia.org/wiki/Imprimerie#:~:text=L'imprimerie%20est%20un%20ensemble,le%20papier%20ou%20le%20textile)
      * <https://pageperso.lislab.fr/~arnaud.labourel/programmation1_2022/doc/doc_idea.pdf>
      * <http://salihayacoub.com/420Keh/Semaine%2010/2015IntellijIdeaGUI.pdf>

# **TABLE DES MATIERES**

[**EPIGRAPHE** 1](#_Toc207228923)

**[DEDICACE](#_Toc207228924)** [2](#_Toc207228924)

[**REMERCIEMENTS** 3](#_Toc207228925)

[**LISTE DES ABREVIATIONS** 4](#_Toc207228926)

[**LISTE DES FIGURES** 5](#_Toc207228927)

[**LISTE DES TABLEAUX** 6](#_Toc207228928)

[**INTRODUCTION GENERALE** 7](#_Toc207228929)

[**0.1** **Présentation du sujet** 7](#_Toc207228930)

[**0.2 Problématique** 7](#_Toc207228931)

[**0.3 Hypothèse** 7](#_Toc207228932)

[**0.4** **Choix et intérêt du sujet** 8](#_Toc207228933)

[**0.4.1 Choix** 8](#_Toc207228934)

[**0.4.2 Intérêt** 8](#_Toc207228935)

[**0.5** **Objectif** 9](#_Toc207228936)

[**0.5.1** **Objectif principal** 9](#_Toc207228937)

[**0.5.2** **Objectif spécifique** 9](#_Toc207228938)

[**0.6 Méthodes et technique** 9](#_Toc207228939)

[**O.6.1 Méthode** 9](#_Toc207228940)

[**0.6.2** **Techniques** 9](#_Toc207228941)

[**0.8 Subdivision du sujet** 10](#_Toc207228942)

[**CHAPITRE 1. GENERALITES SUR LA PROGRAMMATION INFORMATIQUE** 11](#_Toc207228943)

[**1.1** **Introduction** 11](#_Toc207228944)

[**1.2** **Genèse des langages de programmation** 11](#_Toc207228945)

[**1.3** **Modelés des langages de programmation et leurs classifications** 12](#_Toc207228946)

[**1.4** **Aspects et défis de la programmation** 14](#_Toc207228947)

[**1.4.1** **Les principaux aspects de la programmation informatique** 14](#_Toc207228948)

[**1.4.2 Les défis de la programmation** 15](#_Toc207228949)

[**1.4.3 Concepts fondamentaux de la programmation** 15](#_Toc207228950)

[**1.4.4 Variables et types de données** 15](#_Toc207228951)

[**1.4.5 Structures de contrôle** 16](#_Toc207228952)

[**1.4.6 Fonctions et modularité** 16](#_Toc207228953)

[**1.4.7** **Gestion de la mémoire en programmation** 16](#_Toc207228954)

[**1.5** **Entrées et sortie en programmation** 18](#_Toc207228955)

[**1.6** **Algorithme et structure de données fondement et applications** 19](#_Toc207228956)

[**1.6.1 Principaux types d’algorithme** 20](#_Toc207228957)

[**1.6.2 Principales structures de données** 20](#_Toc207228958)

[**1.7** **Critères de performance de langages de programmation** 21](#_Toc207228959)

[**1.7.1 Critères techniques de performance** 22](#_Toc207228960)

[**1.7.2 Concurrence et parallélisme de la programmation** 23](#_Toc207228961)

[**1.7.3 L’impact du Cloud et de l’IA en programmation** 25](#_Toc207228962)

[**1.8** **Importance de l’optimisation en programmation** 29](#_Toc207228963)

[**1.8.1 Pourquoi l’optimisation est-elle importante** 29](#_Toc207228964)

[**1.8.2 Défis de l’optimisation** 30](#_Toc207228965)

[**1.9. CONCLUSION** 30](#_Toc207228966)

[**CHAPITRE 2 : ATOUTS ET INCONVENIENTS DES LANGAGES DE PROGRAMMATION** 31](#_Toc207228967)

[**2.1 Introduction** 31](#_Toc207228968)

[**2.2 Présentation des langages** 31](#_Toc207228969)

[**2.3. Avantages et désavantages de RUST** 31](#_Toc207228970)

[**2.3.1. Avantages** 31](#_Toc207228971)

[**2.3.2. Désavantages** 34](#_Toc207228972)

[**2.4. Avantages et désavantages de PYTHON** 36](#_Toc207228973)

[**2.4.1. Avantages** 36](#_Toc207228974)

[**2.4.2. Désavantages** 39](#_Toc207228975)

[**2.5. Comparaison entre RUST et PYTHON (syntaxe, code, ide, éditeurs, mots clés, hard, soft,** ….) 43](#_Toc207228976)

[**Tableau 2.1 : Comparaison Rust et python** 43](#_Toc207228977)

[**2.6. Disposition** 44](#_Toc207228978)

[**2.6.1 Disposition de Rust** 44](#_Toc207228979)

[**2.6.2 Disposition de python** 44](#_Toc207228980)

[**2.7. Conclusion** 45](#_Toc207228981)

[**CHAPITRE 3 ANALYSE DE PERFORMANCE ET D’UTILISATION DES RESSOURCES (Rust et python)** 46](#_Toc207228982)

[**3.1 Introduction** 46](#_Toc207228983)

[**3.2 Consommation de mémoire** 46](#_Toc207228984)

[**3.2.1 La gestion de la mémoire en Rust** 46](#_Toc207228985)

[**3.2.2 La gestion de la mémoire en Python** 47](#_Toc207228986)

[**3.2.3 Comparaison de la consommation de mémoire** 48](#_Toc207228987)

[**3.2.4 Implication pour le développement** 48](#_Toc207228988)

[**3.2.5 Outils de mesure de la consommation de mémoire** 48](#_Toc207228989)

[**3.3 Optimisation spécifiques des langages (Rust & Python**) 49](#_Toc207228990)

[**3.3.1 Optimisation dans Rust** 49](#_Toc207228991)

[**3.3.2 Optimisation dans Python** 50](#_Toc207228992)

[**3.3.4 Comparaison des optimisations** 51](#_Toc207228993)

[**3.4** **Performances** 52](#_Toc207228994)

[**3.4.1 Rust** 52](#_Toc207228995)

[**3.4.2 Python** 52](#_Toc207228996)

[**3.4.3 Comparaison des performances** 53](#_Toc207228997)

[**FIGURE 3.1 : code Rust** 53](#_Toc207228998)

[**3.5 Utilisation des ressources** 56](#_Toc207228999)

[**3.5.1 Utilisation des ressources en Rust** 57](#_Toc207229000)

[**3.5.2 Utilisation des ressources en python** 57](#_Toc207229001)

[**3.5.3 Comparaison directe entre Rust et Python** 58](#_Toc207229002)

[**3.5.4 Cas d’utilisation** 58](#_Toc207229003)

[**3.6. Exemple d’algorithme traduit dans les deux langages** 59](#_Toc207229004)

[**3.6.1. Algorithme de tri par sélection en pseudo-code** 59](#_Toc207229005)

[**3.6.2. Programme équivalent en Python** 59](#_Toc207229006)

[**3.6.3.** **Programme équivalent en Rust** 59](#_Toc207229007)

[**Conclusion général** 61](#_Toc207229008)

[**BIBLIOGRAPHIE** 62](#_Toc207229009)

[TABLE DES MATIERES 63](#_Toc207229010)

1. [fr.m.wikipedia](http://www.wikipedia).org, consulté  le 30/04/2025 à 19h’ [↑](#footnote-ref-1)