Comment se former à Rust : un guide complet pour aider les débutants à maîtriser la programmation Rust

Par Vitaly Bragilevsky - Delphine Massenhove (traducteur)

Date de publication : 3 mars 2025

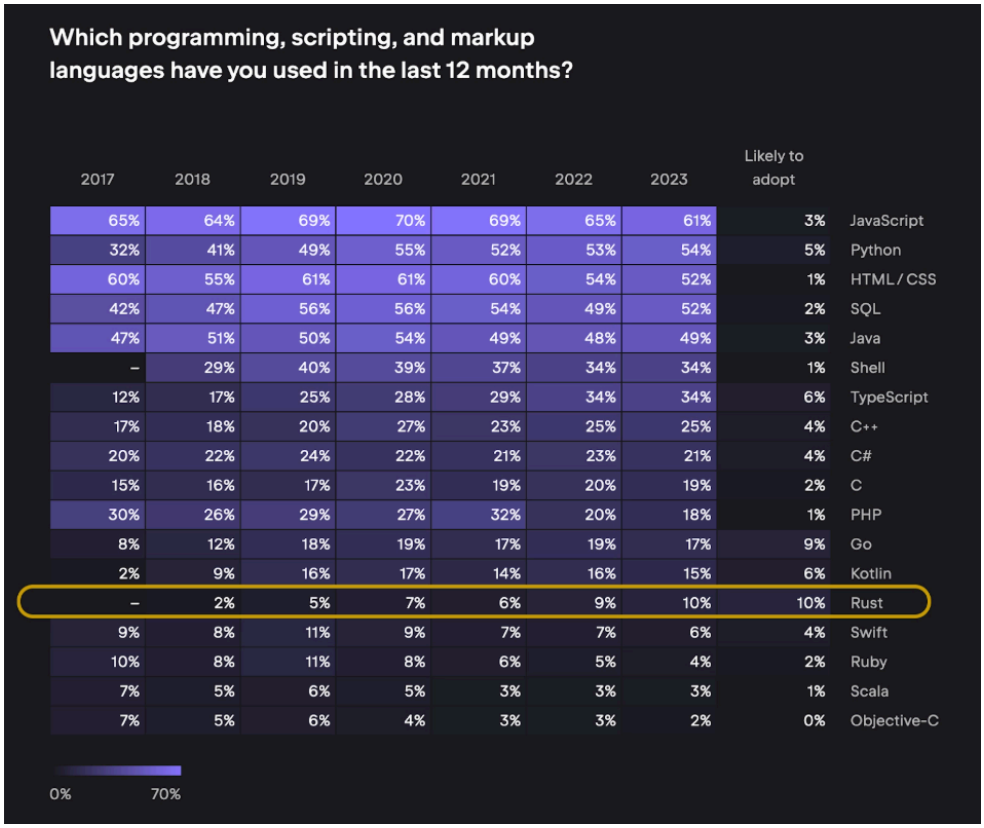
Introduction

Vous envisagez d'apprendre un nouveau langage de programmation et pensez à Rust, mais n'avez pas encore pris votre décision. Vous savez déjà écrire du code et avez une certaine expérience avec au moins un langage de programmation, probablement Python ou JavaScript. Vous avez déjà entendu un certain nombre de choses à propos de Rust. De nombreuses personnes en parlent comme d'un langage de programmation système moderne, qui apporte sécurité et performances et résout des problèmes difficiles à éviter avec d'autres langages de programmation (tels que C ou C++). Vous avez aussi sûrement entendu dire que Rust devenait de plus en plus populaire. Vous voyez régulièrement des articles de blog expliquant que Google, Microsoft, Amazon et d'autres grands acteurs du secteur l'ont déjà adopté et que son adoption est aussi en augmentation dans plusieurs autres secteurs d'activité. Mais vous avez aussi entendu dire que Rust n'est pas le langage le plus facile à apprendre et vous vous demandez si vous pourrez y arriver.

Si vous vous reconnaissez dans la description ci-dessus, cet article est fait pour vous. Mon objectif est de répondre à toutes les questions que vous vous posez à propos de l'apprentissage de Rust. Je commencerai par présenter les avantages de Rust pour les développeurs et ferai une comparaison avec d'autres langages de programmation concurrents. Je poursuivrai par une présentation des fonctionnalités et concepts de Rust les plus importants, qui font qu'ils se démarquent des autres langages de programmation. Je partagerai ensuite des recommandations pour bien démarrer avec Rust, des suggestions sur les formations les plus efficaces et diverses ressources pédagogiques utiles (livres, guides, tutoriels et cours), ainsi que des conseils pour vous aider à relever les défis que vous pourrez rencontrer durant votre parcours d'apprentissage. Enfin, j'évoquerai les domaines du développement logiciel dans lesquels l'utilisation de Rust fait vraiment la différence et vous expliquerai pourquoi et comment.

I - Pourquoi apprendre Rust en 2024 ?

Les dernières enquêtes menées auprès des développeurs ont montré que Rust est l'un des 12 langages de programmation les plus utilisés dans le secteur de l'informatique. Selon les données de ces enquêtes, 10 à 13 % des développeurs de logiciels travaillent régulièrement avec Rust. Si on prend seulement en considération ceux qui sont en phase d'apprentissage de la programmation, la proportion d'utilisateurs de Rust est alors de 11 à 18 %. Les personnes qui se forment à Rust constituent un puissant moteur de croissance. **L'Enquête sur l'état de l'écosystème des développeurs** révèle que Rust a connu une croissance constante au cours des six dernières années et ne montre aucun signe de stagnation :



Vous remarquerez que Rust est le seul langage ayant obtenu un résultat à deux chiffres dans la colonne « Likely to adopt » !

L'engouement des développeurs pour Rust est également significatif. Comment puis-je le savoir ? Les résultats de l'enquête sur les développeurs de StackOverflow en attestent :

- Rust continue d'être **le langage qui suscite le plus d'intérêt** dans la communauté de développeurs, et ce depuis de nombreuses années consécutives (huit ans ? neuf ans ? On ne sait même plus exactement !).
- Les développeurs qui travaillent déjà avec Rust **n'envisagent aucunement de passer** à un autre langage de programmation.

En tant que participant régulier à des conférences pour les développeurs Rust, je peux également témoigner de cet engouement pour Rust. Les utilisateurs de Rust constituent l'une des communautés de développeurs les plus dynamiques et conviviales.

D'un autre côté, l'enquête de StackOverflow **rapporte** une baisse du salaire annuel médian au cours des deux dernières années, mais cette tendance s'observe également pour d'autres langages de programmation populaires. Malgré son adoption croissante par de grandes entreprises et des entreprise de taille moyenne du secteur informatique, nombreuses petites entreprises n'utilisent pas encore Rust, il pourrait donc y avoir davantage d'offres d'emploi pour des développeurs Rust à l'avenir (je vous invite à consulter **cette passionnante discussion** pour plus d'informations sur ce sujet).

Si on examine plus en détail les fonctionnalités et l'écosystème du langage, on comprend immédiatement les chiffres mentionnés plus haut. Rust a été conçu pour offrir à la fois davantage de sécurité en matière de mémoire et de meilleures performances. Les choix uniques qu'ont fait les concepteurs du langage favorisent la diminution des erreurs d'accès à la mémoire dans le code, ce qui réduit les risques de failles de sécurité sans que les performances n'en pâtissent. Alors, quel est donc le « prix à payer » pour cela ? Eh bien, c'est tout le temps que vous devez passer à convaincre le compilateur Rust que votre code est vraiment sûr. Une fois que le compilateur est d'accord avec vous, vous êtes tranquille. Ensemble, la sécurité de la mémoire et les performances du langage permettent d'offrir une expérience beaucoup plus fluide aux utilisateurs de solutions logicielles (c'est-à-dire quasiment tout le

monde de nos jours). La bibliothèque standard de Rust, d'autres bibliothèques essentielles et ensembles d'outils améliorent également l'expérience de développement. Par exemple, développer des applications concurrentes hautes performances en Rust n'est pas difficile, car l'écosystème est mature et bien équipé pour prendre en charge de telles tâches.

Les développeurs apprécient également le grand nombre de secteurs d'activité et de domaines technologiques dans lesquels Rust est applicable. **L'enquête annuelle sur Rust** la plus récente a notamment identifié les domaines suivants :

- Applications côté serveur et autres applications backend.
- Applications, services et infrastructures de cloud computing.
- Systèmes distribués.
- Réseaux informatiques.
- Sécurité informatique.
- Développement embarqué.
- Développement de jeux.
- Frontends web (y compris WebAssembly).
- Et bien d'autres.

Rust n'est évidemment pas le seul langage de programmation utilisé dans ces domaines. Il a des concurrents dans chacun d'eux. Il est souvent comparé à C++ ou à Go, car ils ont de nombreux points communs, mais ils diffèrent sur plusieurs points, notamment en ce qui concerne les garanties en matière de sécurité de la mémoire, les performances, la disponibilité des outils et des bibliothèques, la simplicité du langage et les secteurs d'activités dans lesquels ils sont le plus utilisés. Rust n'obtient pas les meilleurs résultats à tous les niveaux, mais offre des performances aussi bonnes que C++ et se distingue par sa capacité à permettre d'éviter les pièges liés à l'accès à la mémoire facilement et de créer des applications sans erreurs.

II - Comprendre les concepts fondamentaux de Rust

Prenons de la hauteur pour avoir une vue d'ensemble de Rust et de ses fonctionnalités. Rust n'est ni un langage de programmation orienté objet, ni un langage de programmation fonctionnel. Il ne comporte pas de classes et ne prend pas directement en charge les modèles orientés objet. Il peut opérer avec des fonctions comme valeurs de première classe, mais cette capacité est limitée par rapport aux langages fonctionnels à part entière (comme Haskell ou OCaml).

II-1 - Écrire du code avec des fonctions, des valeurs et des types

Les programmes en Rust sont structurés comme des collections de fonctions combinées en modules. Les fonctions manipulent les valeurs. Les valeurs sont typées de manière statique, ce qui signifie que chaque valeur ou expression doit avoir un type attribué au moment de la compilation. Rust fournit à la fois des types primitifs et composés (tableaux et structures), et sa bibliothèque standard fournit de nombreux types supplémentaires pour les collections de valeurs. Il prend en charge les types génériques, ce qui permet d'éviter d'avoir à mentionner des types spécifiques et de fournir des définitions plus générales. Rust fournit également des traits comme collections de méthodes (c'est-à-dire de fonctions) qui peuvent être implémentés pour des types spécifiques. Les traits permettent à Rust d'atteindre les mêmes niveaux d'abstraction logicielle que les langages orientés objet qui prennent en charge l'héritage et le polymorphisme.

II-2 - Gestion de la mémoire

L'approche de Rust en matière de gestion de la mémoire repose sur le principe suivant : **le compilateur Rust doit savoir précisément à quels emplacements la mémoire est allouée dans le code, comment on y accède, et là où elle n'est plus nécessaire**. Ces informations permettent de contrôler l'accès à la mémoire et de libérer automatiquement la mémoire allouée en insérant les instructions correspondantes directement dans le code généré, ce qui contribue à éviter de nombreux pièges courants auxquels on est exposé avec d'autres langages. Cette approche diffère de la gestion automatique de la mémoire (comme en JavaScript, Python, Java ou C#), dans laquelle

les fragments de mémoire qui ne sont plus nécessaires sont détectés au moment de l'exécution et supprimés par le ramasse-miettes. Rust économise ainsi le temps nécessaire à l'exécution des algorithmes correspondants au moment de l'exécution et atteint un double objectif de sécurité de la mémoire et de performance.

Pour pouvoir inférer des connaissances sur l'accès à la mémoire, Rust fixe des limites concernant ce qui peut être fait avec la mémoire et définit des règles strictes pour garantir l'exactitude :

- chaque fragment de mémoire doit appartenir à une seule variable : le *modèle de propriété* de Rust repose sur ce point ;
- la mutation d'un fragment de mémoire nécessite un accès exclusif (par opposition à une simple lecture de la mémoire) ;
- Rust permet de créer des références mutables et immuables à des fragments de mémoire (en les empruntant) mais utilise un *vérificateur d'emprunt* pour garantir l'exactitude (par exemple, en interdisant plus d'une référence mutable) ;
- le compilateur Rust calcule et vérifie les durées de vie de chaque variable dans un programme, de l'emplacement où elle est créée à celui où elle est supprimée (où elle devient inaccessible).

Les exigences du compilateur peuvent être perçues comme trop strictes. C'est une cause de frustration courante, en particulier lors de l'apprentissage du langage : le compilateur Rust peut refuser un fragment de code qui est logiquement correct.

Pour faciliter la compréhension de ces concepts, examinons l'équivalent Rust du programme Python suivant :

```
1. def print_list(numbers):
2.     for number in numbers:
3.         print(str(number) + " ", end="")
4.     print()
5.
6.
7. def add_one(numbers):
8.     numbers.append(1)
9.
10.
11. def main():
12.     numbers = [1, 1, 1]
13.     print_list(numbers)
14.     add_one(numbers)
15.     print_list(numbers)
16.
17.
18. if __name__ == '__main__':
19.     main()
```

Nous avons donc un fragment de mémoire (la liste Python) avec trois éléments. Nous l'imprimons, ajoutons un élément supplémentaire, et l'imprimons à nouveau. Nous ne mentionnons jamais aucun type ici. Python n'a besoin d'aucun signe de gestion de la mémoire dans le programme, même si la mémoire doit être allouée et libérée à un moment donné. De plus, nous transmettons la liste numbers facilement, sans avoir à penser au contrôle d'accès à la mémoire.

Le même code en Rust diffère non seulement par sa syntaxe, mais aussi dans son approche des types et de la gestion de la mémoire :

```
1. // Here we take a vector by reference (&).
2. // We are not allowed to mutate elements.
3. // We don't take ownership; we just borrow.
4. fn print_vec(numbers: &Vec) {
5.     for number in numbers {
6.         print!("{} ", number);
7.     }
8.     println!()
9. }
10.
```

```
11.
12. // Here we take a vector by mutable reference (&mut).
13. // We are now allowed to mutate elements and the vector itself.
14. // We still don't take ownership; we just borrow.
15. fn add_one(numbers: &mut Vec) {
16.     numbers.push(1)
17. }
18.
19.
20. fn main() {
21.     let mut numbers = vec![1,1,1];
22.     // We pass a reference
23.     print_vec(&numbers);
24.     // We pass a mutable reference
25.     add_one(&mut numbers);
26.     // We pass a reference again
27.     print_vec(&numbers);
28. }
```

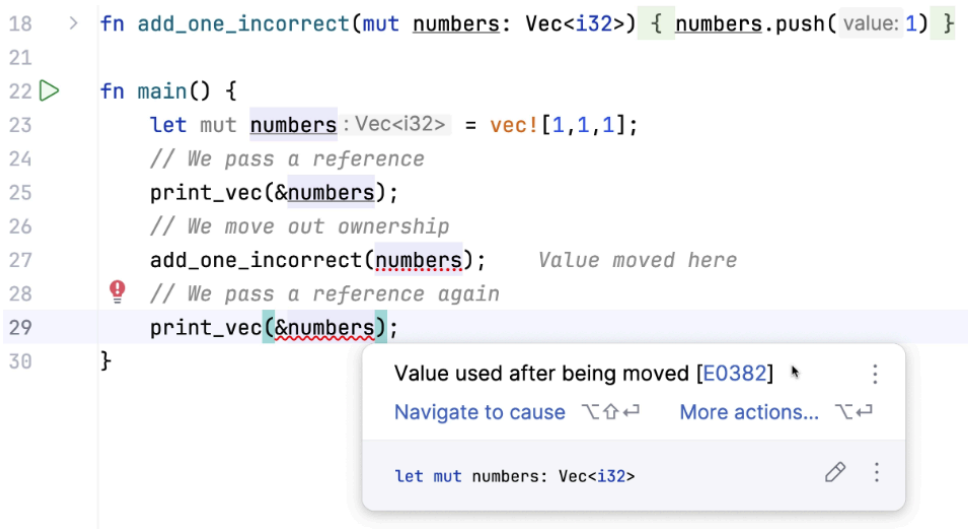
Je vous invite à explorer ces deux fragments de code et à trouver par vous-même les similitudes et les différences entre eux. Même sans comprendre Rust, vous pouvez vous en faire une idée générale simplement en examinant ce code.

Malgré le passage des références en paramètres, la variable *numbers* reste propriétaire de la mémoire allouée. Rust utilise par défaut un accès mémoire en lecture seule et requiert une spécification explicite pour l'accès en écriture. Rust garantit également que la mémoire soit libérée après la dernière utilisation dans le deuxième appel à `print_vec`.

Voici une variante de la fonction add_one qui prend possession d'un vecteur et rend l'ensemble du programme incorrect :

```
fn add_one_incorrect(mut numbers: Vec) {
    numbers.push(1)
}
```

Quel est le problème ? Eh bien, après avoir appelé la fonction `add_one_incorrect`, la variable *numbers* ne possède plus la mémoire ; sa durée de vie est terminée, nous ne pouvons donc rien imprimer ! La capture d'écran ci-dessous montre quel est le problème :



Avant l'introduction de la version incorrecte, le vérificateur d'emprunt de Rust pouvait garantir que le passage des références ne posait aucun problème. Après l'introduction d'une erreur, cela n'est plus possible. Ici, le comportement du langage montre à quel point l'accès à la mémoire est strictement contrôlé.

II-3 - Concurrence

La concurrence est la capacité d'un système à exécuter plusieurs tâches ou processus simultanément ou pendant des périodes qui se chevauchent pour améliorer l'efficacité et les performances. Cela peut impliquer une exécution parallèle sur de multiples processeurs ou une exécution entrelacée sur un seul processeur, ce qui permet à un système de traiter plusieurs opérations simultanément et de gérer plusieurs tâches plus efficacement.

Les Rustacés disent généralement de la concurrence de Rust qu'elle est impressionnante. Plusieurs facteurs expliquent cette perception :

- le modèle de propriété ajoute plus de contrôle sur le partage des données entre les threads concurrents ;
- l'adoption de structures de données immuables simplifie l'implémentation d'algorithmes concurrents et contribue à la sécurité des threads ;
- la transmission des messages via les canaux adoptés en Rust réduit considérablement les complexités liées à la concurrence d'état partagé ;
- l'approche de Rust en matière de durée de vie et de gestion de la mémoire rend généralement le code qui utilise des primitives de concurrence telles que des verrous, des sémaphores ou des barrières, plus concis et plus sûr ;
- dans de nombreux cas, l'approche de Rust en matière de programmation asynchrone évite d'utiliser des schémas de concurrence complexes et permet d'écrire du code concis et clair.

Bien que la concurrence ne soit pas la première chose que les débutants apprennent lorsqu'ils abordent Rust, elle reste plus facile à comprendre que dans de nombreux autres langages de programmation. Plus important encore, Rust vous aide à écrire un code concurrent moins propice aux erreurs.

III - Premiers pas avec Rust

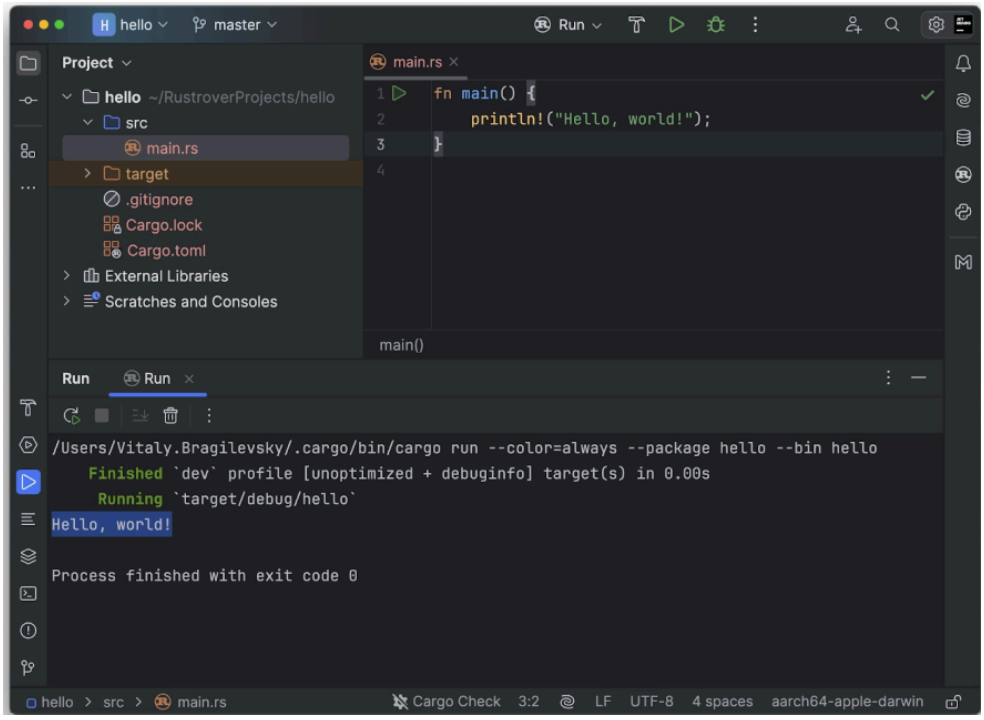
Il suffit d'installer deux choses sur votre ordinateur pour démarrer avec Rust :

- La chaîne d'outils Rust : une collection d'outils qui comprend le compilateur Rust et d'autres utilitaires.
- Un éditeur de code ou un IDE (environnement de développement intégré).

La première étape est simple : allez sur **rustup.rs** et exécutez la commande qu'ils suggèrent, ou téléchargez un installateur, en fonction de votre système d'exploitation.

La deuxième étape nécessite de faire un choix. Vous pouvez utiliser un éditeur de code configuré pour prendre en charge Rust (par exemple, Visual Studio Code prend en charge Rust mais requiert un peu de configuration) ou opter pour un IDE Rust dédié (tel que **RustRover**, l'IDE de JetBrains gratuit pour une utilisation à des fins d'apprentissage).

Si vous choisissez RustRover, vous n'avez rien à configurer. Vous pouvez le lancer, créer un nouveau projet (à l'aide du bouton *New Project* sur l'écran d'accueil), et exécuter le programme *Hello, world!* généré par défaut (utilisez l'icône triangulaire verte dans la gouttière pour exécuter la fonction main) :



Si vous rencontrez un problème, consultez le guide de démarrage rapide de RustRover (**Quick Start Guide**).

À moins qu'il ne s'agisse d'une tâche vraiment simple, vous devrez ajouter des dépendances externes. Les projets Rust s'appuient sur Cargo, un gestionnaire de paquets. En Rust, les paquets s'appellent des **crates**. Les dépendances sont spécifiées dans le fichier *Cargo.toml*, par exemple :

```
[package]
name = "hello"
version = "0.1.0"
edition = "2021"

[dependencies]
chrono = "0.4.38"
```

Après avoir ajouté la crate `chrono` à nos dépendances, nous pouvons utiliser les fonctionnalités qu'elle fournit dans notre code, par exemple :

```
1. use chrono::{DateTime, Local};
2.
3. fn main() {
4.     println!("Hello, world!");
5.     let local: DateTime = Local::now();
6.     println!("Today is {}", local.format("%A"));
7. }
```

Dans ce programme, nous obtenons la date et l'heure (locales) actuelles, les formatons comme un jour de la semaine, puis affichons le résultat. L'exécution de ce programme donne par exemple :

```
1. Hello, world!
2. Today is Thursday
```

L'apprentissage de la syntaxe de base, des types de données et des structures de contrôle (conditions if, boucles, etc.) en Rust est un processus parallèle à l'apprentissage des crates disponibles et de ce qu'elles fournissent. Une fois que vous pourrez exécuter des programmes Rust, vous serez prêt à travailler avec des dépendances externes. Parlons des ressources qui simplifient l'apprentissage de Rust.

IV - Parcours d'apprentissage et ressources

Si vous avez déjà appris une langue étrangère, vous savez que diverses compétences sont requises, comme la compréhension écrite, l'expression écrite, la compréhension orale et l'expression orale. Plusieurs types d'activités permettent de développer ces compétences. Les enseignants vous encouragent toujours à lire des textes, à regarder des vidéos, à écouter des podcasts, à parler autant que possible, à publier sur les réseaux sociaux, etc. C'est la même chose pour apprendre un langage de programmation. Pour apprendre comment coder en Rust (ou avec tout autre langage de programmation), vous devrez :

Lire des livres, de la documentation, des articles de blog, les annonces de nouvelles versions et des discussions sur les réseaux sociaux (ce qui est essentiel pour rester informé et connaître le contexte actuel).

- Regarder des vidéos de blogueurs spécialisés.
- Examiner des exemples de code (ce qui nécessite de les lire et de les modifier !).
- Écrire du code pour vous exercer.
- Suivre des tutoriels pratiques.
- Développer des projets simples à partir de zéro (c'est extrêmement important : l'apprentissage basé sur la réalisation de projets est toujours utile pour développer des compétences en programmation de façon indépendante).
- Peut-être même contribuer à des projets open source (il y a toujours quelque chose à faire, même pour les débutants, il suffit de demander !).

Il est important de maintenir un équilibre entre le temps passé à lire des contenus et écrire du code. Il est impossible de maîtriser un langage de programmation sans le pratiquer réellement en écrivant du code. Dans le même temps, il est essentiel de consulter des livres ou autres documents pour parvenir à une compréhension approfondie des détails spécifiques au langage. Tout cas complexe pourrait devenir extrêmement difficile à résoudre sans aide extérieure (ce qui peut vous ralentir dans votre parcours pour devenir un développeur professionnel).

Lorsque vous écrivez du code, il est important de reproduire des exercices, mais aussi de suivre vos propres idées (ou de vous inspirer d'autres idées) pour développer des applications simples. Assurez-vous que vos exercices ne se limitent pas à la compréhension d'algorithmes, mais enseignent également les fonctionnalités du langage et les fonctions de la bibliothèque. En développant des applications simples, vous apprendrez à choisir la fonctionnalité du langage ou la bibliothèque qui répond le mieux à votre objectif. C'est une compétence essentielle en soi.

Ainsi, la clé d'un apprentissage réussi est de combiner toutes ces approches. Je connais une série de ressources qui peuvent vous y aider.

IV-1 - Livres sur Rust

Il existe beaucoup de bons livres pour apprendre Rust :

- **The Rust Programming Language** est un livre officiel sur le langage régulièrement actualisé, destiné aux personnes qui apprennent Rust. Il est connu sous le nom de The Book au sein de la communauté. Je recommande également **une autre version** de ce livre qui apporte plus d'interactivité, avec des quiz, des visualisations et des exercices. Cette version a été développée par Will Crichton, Gavin Gray et Shriram

Krishnamurthi, chercheurs à l'Université Brown. Il existe également **une interview** avec Will Crichton, dans laquelle il explique cette expérimentation pour améliorer l'expérience des personnes apprenant Rust.	

- **Rust in Action** par Tim McNamara (également connu sous le nom de timClicks ; mon conseil est de le suivre partout).
- **Hands-on Rust** par Herbert Wolverson : le livre qui plaît particulièrement à ceux qui apprécient l'apprentissage par projet (vous pouvez également trouver de nombreuses vidéos gratuites d'Herbert sur plusieurs concepts de Rust ; je ne saurais trop vous les recommander).
- **Rust for Rustaceans** par Jon Gjengset, un livre sur Rust plébiscité par de nombreux membres de la communauté.
- **Code Like a Pro in Rust** par Brenden Matthews.
- **Learn Rust in a Month of Lunches** par David MacLeod.
- Après quelques mois d'apprentissage de Rust, vous devriez jeter un œil à **Rust Atomics and Locks**, par Mara Bos. C'est un chef-d'œuvre en matière d'explication des concepts de bas niveau derrière la concurrence en Rust.

IV-2 - Tutoriels Rust

Pour ceux qui préfèrent une approche plus pratique de l'apprentissage, il existe également des tutoriels et des cours avec exercices :

- **Comprehensive Rust** est un tutoriel Rust développé en interne chez Google mais accessible à tous. C'est la meilleure ressource pour ceux qui souhaitent apprendre rapidement tous les détails nécessaires.
- **Rustlings** (et sa variation pour une **expérience d'apprentissage dans l'IDE**, un **cours** sur Rust de **JetBrains Academy**).
- **100 exercises to learn Rust** : un cours génial avec des exercices développés par **Luca Palmieri** (encore un autre Rustacé à suivre).
- Eleftheria Batsou a présenté **un bel ensemble d'idées de projets simples** sur lesquels travailler tout en apprenant Rust.

IV-3 - Rust sur YouTube

Pour les Rustacés qui aiment apprendre sur YouTube, je recommande d'y suivre plusieurs chaînes :

- **Let's Get Rusty** : je ne sais pas pourquoi, mais il est difficile d'arrêter de regarder ces vidéos.
- **Jeremy Chone** est très doué pour expliquer des concepts complexes.
- **Chris Biscardi** parle du développement de jeux en Rust avec le framework Bevy, entre autres. Créer un jeu est un excellent projet pour tout le monde !
- **Jon Gjengset** : pendant les deux premières années de votre parcours avec Rust, il peut être difficile de suivre ses streams techniques approfondis, mais je recommande de commencer à les regarder assez tôt !

IV-4 - La Communauté Rust

La **communauté Rust sur Reddit** mérite d'être suivie.

En tant que débutant, vous pouvez poser vos questions sur le **serveur Discord de la communauté Rust**.

This Week in Rust est la ressource communautaire la plus importante pour lire les actualités concernant Rust. Je le lis chaque semaine sans exception.

Bien entendu, de nombreuses autres ressources d'apprentissage sont disponibles, mais il est impossible de toutes les répertorier. La bonne nouvelle est que vous n'arrêtez jamais d'apprendre Rust, car le langage évolue constamment.

V - Défis courants et comment les surmonter

Apprendre Rust peut être complexe. Vous pouvez toutefois surmonter ces difficultés et parvenir à maîtriser le langage en adoptant la bonne stratégie. Voici quelques conseils pour vous aider tout au long de votre parcours :

- **Comprenez bien le modèle de propriété**. Le modèle de propriété de Rust, qui inclut des concepts tels que la propriété, l'emprunt et la durée de vie, est souvent l'aspect le plus difficile pour les débutants. Commencez par de petits exemples et entraînez-vous fréquemment. Cherchez surtout à comprendre les raisons pour lesquelles Rust applique ces règles plutôt que de vous concentrer sur la manière de contourner les erreurs du compilateur.
- **Prenez votre temps avec le vérificateur d'emprunt**. Le vérificateur d'emprunt peut être frustrant, car il est strict et peut conduire à des erreurs de compilation déroutantes. Lorsque vous rencontrez des problèmes avec le vérificateur d'emprunt, prenez du recul et analysez ce que Rust essaie d'éviter. Expérimentez avec des approches telles que le référencement ou le clonage de données (mais essayez de ne pas tout cloner !). N'hésitez pas à demander de l'aide sur des forums comme Reddit ou le serveur Rust Community sur Discord. Il est vraiment utile de comprendre ce qu'est précisément un vérificateur d'emprunt. Consultez, par exemple, cette **excellente explication** de Nell Shamrell-Harrington.
- **Tirez parti de la documentation de Rust**. Rust est un langage relativement nouveau, donc certaines choses peuvent ne sont pas aussi facilement compréhensibles qu'avec d'autres langages plus anciens. Utilisez la documentation Rust, qui est réputée pour être complète et bien écrite. La communauté Rust contribue également à d'excellentes ressources telles que le guide **Rust by Example**.
- **Commencez par des projets simples**. La complexité de Rust peut donner l'impression d'être dépassé au début. Commencez par implémenter des programmes basiques et passez progressivement à des programmes plus complexes. Voici quelques exemples de projets simples : la création d'un outil de ligne de commande avec **clap**, l'écriture d'analyseurs de fichiers simples ou la création d'un serveur web de base à l'aide d'un framework tel que **Rocket**.
- **Familiarisez-vous avec votre éditeur de code ou votre IDE**. Écrire du code Rust manuellement peut entraîner des erreurs. Les outils sont là pour vous aider pour la complétion, le linting et le formatage du code.
- **Rejoignez la communauté Rust**. Cela vous évitera de vous sentir isolé. Impliquez-vous en contribuant à des projets open source, ou en participant à des discussions et des forums. La communauté Rust est connue pour être très accueillante et aider les débutants.
- **Apprenez de vos erreurs**. Le compilateur de Rust est particulièrement strict et vous pouvez avoir l'impression de batailler avec lui en permanence. Considérez chaque erreur du compilateur comme une opportunité d'apprentissage. Les messages d'erreur de Rust sont souvent très descriptifs et vous guident vers la bonne solution. Au fil du temps, vous constaterez que ces erreurs vous aideront à écrire un code de meilleure qualité et plus sûr.
- **Soyez patient**. On peut facilement se décourager avec Rust. Rappelez-vous que Rust est conçu pour être sûr et efficace, ce qui demande d'apprendre des concepts complexes. Faites des pauses lorsque vous en ressentez le besoin et n'hésitez pas à revenir sur des sujets plusieurs fois jusqu'à ce que vous soyez sûr de les avoir bien compris.
- **Rejoignez des groupes d'apprentissage de Rust**. De nombreux développeurs apprennent Rust et rejoindre un groupe peut vous apporter soutien et motivation. Recherchez des meetups ou des groupes d'étude en ligne, ou apprenez en même temps q'un ami.

Maîtriser Rust prend du temps, mais le jeu en vaut la chandelle. Les avantages de Rust, telles que les garanties qu'il offre en matière de sécurité de la mémoire et ses performances, en font un langage puissant pour la programmation système et le développement web, entre autres. Persévérez et vous constaterez que l'utilisation de Rust peut être plaisante et gratifiante.

VI - Créer des applications concrètes avec Rust

Examinons maintenant rapidement les avantages de Rust pour la création d'applications concrètes. Nous allons évoquer plusieurs des domaines technologiques pour lesquels Rust est pertinent, avec des cas d'usages qui mettent en évidence ses atouts et son utilité.

VI-1 - Rust pour le développement web

L'écosystème de Rust est parfaitement adapté au développement web backend. De nombreux frameworks et bibliothèques effectuent des tâches générales de développement web (routage de requêtes HTTP, gestion de formulaires et de JSON, création de modèles, accès aux bases de données, authentification et autorisation, journalisation et traçage, etc.), avec des performances adéquates pour la production. Je vous recommande de consulter le site web **AreWeWebYet?** pour plus de détails ainsi que des liens utiles. Spoiler : la réponse à la question que pose le site est oui.

La plupart des frameworks web pour Rust sont assez faciles à utiliser dès le départ et offrent une documentation de qualité, en partie du fait de la concurrence qui règne entre eux. La communauté **Actix Web** propose notamment une **collection d'exemples** intéressante et à jour. Mon collègue Khalid Abuhakmeh a écrit un **tutoriel pour les débutants** sur la création d'une application web HTML simple qui utilise Rocket, un autre framework web pour Rust, comme backend. Luca Palmieri, que j'ai mentionné plus tôt, est l'auteur du livre **Zero To Production in Rust**, qui est un guide pour devenir développeur web backend. Il travaille également sur **Pavex**, un framework web émergent et prometteur. Fait notable concernant ce dernier projet, Luca documente soigneusement les choix de conception dans **une série de rapports d'avancement**. Il est réellement intéressant de découvrir et de comprendre les raisons de ces choix. Consultez **cette comparaison de frameworks web** pour comprendre à quel point les frameworks web Rust se développent.

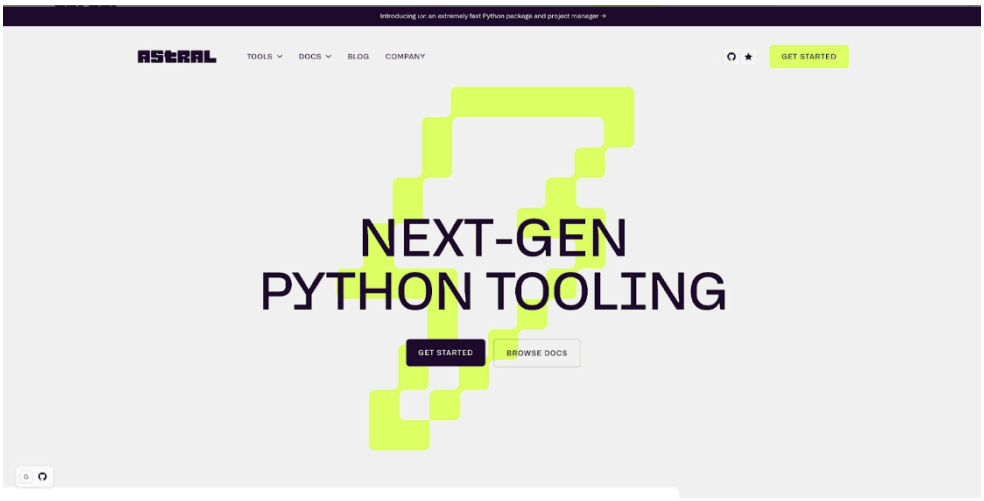
Rust est également actif dans le frontend web, même s'il n'est peut-être pas encore prêt à 100 % pour la production. Néanmoins, ses avancées concernant WebAssembly et les frameworks comme **egui** ou **Dioxus** sont impressionnantes.

VI-2 - Intégration de Rust avec d'autres langages de programmation

Je me demande si les concepteurs de Rust avaient imaginé que Rust deviendrait l'un des langages de prédilection des personnes développant des outils et des bibliothèques pour d'autres langages, en particulier pour Python et JavaScript. Rust peut offrir de meilleures performances, et il est suffisamment attrayant pour que les développeurs des écosystèmes d'autres langages se mettent à l'apprendre et à l'adopter.

Le projet **PyO3** permet aux développeurs d'implémenter des modules Python natifs en Rust et d'accéder à Python dans les binaires Rust. Grâce à PyO3, vous pouvez fournir une implémentation Rust extrêmement efficace d'une bibliothèque Python pour les développeurs Python. Rust et JavaScript peuvent être connectés de façon similaire via WebAssembly et **wasm-bindgen**.

Astral et son créateur, Charlie Marsh, prouvent qu'il est possible de développer des outils Python qui fonctionnent à la vitesse de la lumière (enfin, nous savons tous que c'est une affirmation marketing, bien sûr, mais le résultat est vraiment fascinant).



Deno, un runtime JavaScript développé en Rust, offre de très bonnes performances. Ce n'est pas le seul exemple dans l'écosystème JavaScript. Il existe même une **collection d'outils JavaScript** implémentés en Rust.

Implémenter une interopérabilité entre Rust et d'autres langages de programmation est également possible.

VI-3 - Systèmes et programmation embarquée en Rust

Rust brille incontestablement dans le domaine de la programmation de systèmes. Les intervenants de Google et de Microsoft **confirment** que l'introduction de Rust dans leurs bases de code réduit le nombre de vulnérabilités de sécurité, augmente les performances et maintient ou voire améliore la productivité des équipes de développeurs de logiciels. Amazon prend non seulement en charge le **développement Rust sur AWS**, mais l'utilise également directement pour **implémenter sa propre infrastructure**. Cloudflare, une entreprise qui crée des réseaux de diffusion de contenu et d'autres infrastructures réseau, **utilise Rust** dans de nombreux projets de bas niveau et contribue à l'écosystème Rust avec ses **frameworks**. Ferrous Systems, une société qui soutient le développement de rust-analyzer, développe également **Ferrocene**, une suite d'outils Rust certifiée pour les applications critiques. Cela permet d'utiliser Rust dans les industries automobile et aérospatiale. On trouve notamment des rapports concernant l'utilisation de Rust par **Volvo** et **Renault** pour leurs logiciels embarqués.

VII - Conclusion et prochaines étapes

Rust est un langage prometteur, qui a l'avantage de disposer de nombreuses ressources d'apprentissage et d'une communauté active et accueillante pour les débutants. Même s'il présente certains défis, compte tenu de ses concepts fondamentaux uniques, il existe également des moyens de les relever. Un fois que vous maîtriserez son utilisation, de nombreuses opportunités d'emploi dans différents domaines s'ouvriront à vous.

J'espère que les conseils et les ressources que j'ai présentés dans cet article vous aideront dans votre parcours. Je vous recommande surtout de vous fixer un objectif avant de vous lancer : trouvez un projet open source auquel vous souhaitez contribuer ou développez une idée pour votre projet dès le début. Vous serez étonné de la rapidité avec laquelle vous parviendrez à vous rapprocher de l'atteinte de cet objectif.

Je vous souhaite le meilleur dans votre aventure d’apprentissage de Rust !