Formation Rust.

Sparks.

Objectifs pédagogiques.

Lors de cette formation Rust de 3 jours, les apprenants découvriront les spécificités du langage, et comprendront pourquoi celui-ci devient l'une des technologies préférées des développeurs et développeuses. Après avoir exploré la syntaxe Rust, ils réaliseront différents ateliers pratiques, pour apprendre les bonnes pratiques de code, debug et test en Rust. Les principaux concepts de programmation mis en œuvre par Rust seront abordés de manière progressive tout au long de la formation : ownership, borrowing, lifetimes, mutabilité, etc.

Public

Développeurs, Architectes logiciels

Durée

3 jours

Prérequis

Des connaissances en programmation et sur les systèmes Unix.

Dates 2021

Distanciel : 13/04, 14/06, 20/09, 07/11

Programme.

Introduction à la formation Rust

Présentation générale et objectifs de cette formation Rust
Principales caractéristiques du langage, avantages et positionnement
Fonctionnement de Rust (vitesse, erreurs, sûreté, multithreading) et paradigmes de programmation
Installer Rust et Cargo, le gestionnaire de packages pour gérer dépendances et compilation
Démarrer un nouveau projet : crates et structure de base
Outils de développement
Documentation
Exemples de cas pratiques : tour de table, échanges sur les différents contextes professionnels et les perspectives d'application des nouvelles compétences, installation de rustup et mise en place de différents outils de développement. Codage d'un premier programme en Rust.

Syntaxe et concepts basiques de programmation Rust

Blocs de code, accolades, structures… découvrir la syntaxe de Rust
Variables, constantes et (im)mutabilité
Types primitifs : entiers, flottants, strings, tuples…
Expressions if, if else, boucles et boucles conditionnelles
Fonctions : déclaration, paramètres et retours
Définir un module
Tableaux et slices
Introduction aux macros
Commentaires
Exemples de cas pratiques : création de divers programmes en Rust (conversion de température, génération du nième nombre de la suite Fibonacci…) pour manipuler les concepts de base du langage.

Propriété (ownership) et emprunt (borrowing)

Principe de fonctionnement et intérêt de l'ownership : mémoire et allocation
Comprendre les règles de l'ownership
Transférer la propriété : les moves
Références et borrowing
Opérateurs & et & mut
Exemples de cas pratiques : résolution de problèmes de programmation pour comprendre et mettre en application les concepts de propriété, emprunt et les slices. Echanges sur l'apport de tels principes dans les programmes Rust en termes de sécurité et de gestion de la mémoire, comparaison avec d'autres langages.

Types personnalisés : structures (structs) et énumérations (enums)

Utiliser les structs pour organiser la donnée : définition et instanciation
Méthodes et fonctions
Enumérations : Option et autres usages communs
Filtrage par motif (pattern matching) avec l'expression match
Déstructuration : tuples, énumérations, pointeurs…
Exemples de cas pratiques : écrire des programmes utilisant structs et enums, corrections de morceaux de code pour faire passer des tests (rustlings GitHub),mise en œuvre de patterns et anti-patterns de Rust.

Gestion des erreurs

Les deux types d'erreurs sous Rust : recoverable et unrecoverable
Apports de l'approche Rust pour la gestion des erreurs
Utiliser la macro panic!
Result, principes de fonctionnement
La propagation d'erreurs
Gérer les erreurs dans main()
Exemples de cas pratiques : utilisation de la macro panic, résolution de problèmes de programmation avec l'enum Option, la méthode unwrap et les combinateurs, l'enum Result et ses alias, gestion de multiples types d'erreur avec try.

Généricité en Rust, traits et lifetimes

Principes de fonctionnement des traits dans Rust
Créer et implémenter un trait : Into, Drop, Iterator, Clone…
L'attribut derive
La surcharge d'opérateurs
Références et concept de durée de vie (lifetime)
Durées de vie statiques et associées
La généricité en Rust : types, fonctions, implémentations, restrictions…
Exemples de cas pratiques : implémentations de différents types de traits, surcharge de l'opérateur d'addition avec le trait Add, mise en œuvre de types et fonctionnalités génériques, utilisation de la clause where pour expliciter une restriction.

Closures et itérateurs

Les closures (aussi appelées lambdas) : caractéristiques et syntaxe
Closures passées en paramètres
Types anonymes
Fonctions passées en paramètres
Fonctions d'ordre supérieur (HOF)
Créer ses propres itérateurs avec le trait Iterator
Exemples de cas pratiques : capture de variables par référence, par valeur, utilisation des closures avec des outils de la bibliothèque standard (itérateurs any, find), refactoring et mise en œuvre de la méthode next.

Multi-threading

Créer des threads
Transfert de données entre thread : message passing
Les traits Sync et Send
Exemples de cas pratiques : création de threads avec spawn, communication entre threads et manipulation de concepts de programmation concurrente.

Quelques concepts de programmation avancée en Rust

Fonctionnement des macros
Contourner la sûreté de Rust avec les blocs unsafe
Interfaçage avec C/C++

0 805 950 800 demande@sparks-formation.com

Toutes nos formations sont disponibles à distance