¿Qué es Rust?.

Rust es un lenguaje de programación de sistemas de código abierto que se puede usar para desarrollar software seguro y eficaz. Con Rust,se puede administrar la memoria y controlar otros detalles de bajo nivel. Pero también se pueden aprovechar los conceptos de alto nivel, como la iteración y las interfaces. Estas características distinguen a Rust de los lenguajes de bajo nivel, como C y C++.

Rust también ofrece las siguientes ventajas que lo hacen ideal para una amplia gama de aplicaciones:

- Type Safe: El compilador garantiza que no se aplicará ninguna operación a una variable de tipo incorrecto.

- Memory Safe: Los punteros de Rust (conocidos como *referencias*) siempre hacen referencia a la memoria válida.

- Data Race Free: El comprobador de préstamos de Rust garantiza la seguridad para subprocesos asegurándose de que varias partes de un programa no puedan mutar el mismo valor al mismo tiempo.

- Zero-cost abstractions: Rust permite el uso de conceptos generales, como la iteración, las interfaces y la programación funcional, con un costo de rendimiento mínimo o nulo. Las abstracciones funcionan tan bien como si hubiera escrito el código subyacente a mano.

- Minimal Runtime: Rust tiene un entorno de ejecución mínimo (y opcional). Con el fin de administrar la memoria de forma eficaz, el lenguaje tampoco tiene ningún recolector de elementos no utilizados. De este modo, Rust se parece más a lenguajes como C y C++.

- Tagets bare metal: Rust puede tener como destino la programación insertada y sin sistema operativo, lo que lo hace adecuado para escribir un kernel de sistema operativo o controladores de dispositivo.

Según la encuesta de desarrolladores de Stack Overflow de 2021, Rust ha sido el lenguaje más apreciado durante varios años seguidos. Los

desarrolladores disfrutarán de la programación con Rust. Muchos tipos de organizaciones, desde las startups hasta las grandes empresas, usan Rust en sus casos de uso exclusivos. Desde la creación de herramientas, hasta la escritura de aplicaciones web, el trabajo en servidores o la creación de sistemas insertados, las posibilidades son infinitas.

Características únicas de Rust.

Para averiguar si un lenguaje de programación es adecuado para un proyecto, debe conocer las características y las limitaciones de estos, una vez hecho esto podra comparar los lenguajes posibles y elegir el que mejor se adapte a sus necesidades.

En esta unidad, revisaremos algunas de las características y limitaciones de Rust:

- El sistema de módulos de Rust: módulos, crates y rutas
- Bibliotecas estándar de Rust y crates de terceros
- La herramienta Cargo de Rust y el administrador de dependencias
- Cuándo se debe usar Rust.

Administración de código con el sistema de módulos de Rust.

Rust ofrece una colección de características que le ayudarán a administrar y organizar el código. Estas características se conocen como "sistema de módulos de Rust". El sistema se compone de crates, módulos y rutas, así como herramientas para trabajar con estos elementos.

- Crates: Un crate de Rust es una unidad de compilación. Es el fragmento de código más pequeño que puede ejecutar el compilador de Rust. El código de un crate se compila en conjunto para crear un archivo ejecutable binario o una biblioteca. En Rust, solo los crates se compilan como unidades reutilizables. Un crate contiene una jerarquía de módulos de Rust con un módulo implícito de nivel superior sin nombre.
- Módulos: Los módulos de Rust ayudan a organizar el programa, ya que permiten administrar el ámbito de los elementos de código individuales dentro de un crate. Los elementos de código relacionados o los elementos que se usan juntos se pueden agrupar

en el mismo módulo. Las definiciones de código recursivas pueden abarcar otros módulos.

- Rutas: En Rust, puede usar rutas para dar nombre a los elementos del código. Por ejemplo, una ruta puede ser una definición de datos, como un vector, una función de código o incluso un módulo.

  La característica de módulo también le ayuda a controlar la privacidad de las rutas. Puede especificar las partes del código a las que se puede acceder públicamente frente a las partes privadas. Esta característica le permite ocultar los detalles de implementación.

Uso de crates y bibliotecas de Rust.

La biblioteca estándar de Rust, (std), contiene código reutilizable para las definiciones y operaciones fundamentales de los programas de Rust. Esta biblioteca tiene definiciones para tipos de datos principales como, por ejemplo, String y Vec<T>, operaciones para primitivas de Rust, código para funciones de macro usadas con frecuencia, compatibilidad con acciones de entrada y salida, y muchas otras áreas de funcionalidad.

Hay decenas de miles de bibliotecas y crates de terceros disponibles para su uso en los programas Rust; para acceder a la mayoría de ellas se puede usar el repositorio de crates de terceros de Rust, crates.io. Más adelante veremos cómo acceder a estos crates desde nuestro proyecto, pero por ahora estas son algunas de las crates que se usan en los ejercicios de programación:

- std: Biblioteca estándar de Rust. En los ejercicios de Rust, verá que aparecen los siguientes módulos:
  - std::collections: definiciones de tipos de colección, como HashMap.
  - std::env: Funciones para trabajar con el entorno.
  - std::fmt: Funcionalidad para controlar el formato de salida.
  - std::fs: Funciones para trabajar con el sistema de archivos.
  - std::io: Definiciones y funcionalidad para trabajar con entradas y salidas.
  - std::path: definiciones y funciones que permiten trabajar con datos de ruta de acceso del sistema de archivos.
- structopt: crate de terceros para analizar argumentos de línea de comandos fácilmente.
- chrono: crate de terceros para controlar los datos de fecha y hora.

- regex: crate de terceros para trabajar con expresiones regulares.
- serde: crate de terceros con operaciones de serialización y deserialización de estructuras de datos de Rust.

De manera predeterminada, la biblioteca std está disponible para todos los crates de Rust. Para acceder al código reutilizable en un crate o biblioteca, implementamos la palabra clave use. Con la palabra clave use, el código del crate o biblioteca se "incluye en el ámbito" para que pueda acceder a las definiciones y funciones en el programa.

Se accede a la biblioteca estándar en instrucciones use con la ruta std, como en (use std::fmt). Se accede a otros crates o bibliotecas con su nombre, como por ejemplo (use regex::Regex).

Creación y administración de proyectos con Cargo.

Aunque se puede usar el compilador de Rust (rustc) directamente para crear crates, en la mayoría de los proyectos se usa la herramienta de compilación de Rust y un administrador de dependencias llamado Cargo.

Cargo hace gran cantidad de cosas, entre las que se incluyen las siguientes:

- Crear nuevas plantillas de proyecto con el comando cargo new.
- Compilar un proyecto con el comando cargo build.
- Compilar y ejecutar un proyecto con el comando cargo run.
- Probar un proyecto con el comando cargo test.
- Comprobar los tipos de proyecto con el comando cargo check.
- Compilar la documentación de un proyecto con el comando cargo doc.
- Publicar una biblioteca para crates.io con el comando cargo publish.
- Para agregar crates dependientes a un proyecto, agregue el nombre del crate al archivo Cargo.toml.

Cuándo se debe usar Rust

El lenguaje Rust tiene numerosos puntos a favor que se deben tener en cuenta al elegir el mejor lenguaje para un proyecto:

- Rust permite controlar el rendimiento y el consumo de recursos de los programas y bibliotecas escritos en el lenguaje al mismo nivel que C y C++, al tiempo que mantiene la memoria protegida,

ya que elimina todas las clases de errores comunes de manera predeterminada.

- Rust tiene características de abstracción muy completas que permiten a los desarrolladores codificar muchos de los aspectos invariables de sus programas en código, que luego el compilador se encarga de comprobar en lugar de depender de convenciones o documentaciones. Esta característica suele dar lugar a la impresión de que "si se compila, funciona".

- Rust tiene herramientas integradas para compilar, probar, documentar y compartir código, así como un ecosistema completo de herramientas y bibliotecas de terceros. Gracias a estas herramientas, algunas tareas que son difíciles en algunos lenguajes, como crear dependencias, resultan fáciles de llevar a cabo y productivas en Rust.