Explorando la programación en Rust

Rust es un lenguaje de programación que ha ganado mucha popularidad en los últimos años. En 2006, un empleado de Mozilla comenzó un proyecto personal, que luego recibió el apoyo de la empresa y fue lanzado en 2010. Este proyecto se llama Rust.

A continuación, vamos a ejecutar el primer programa en Rust. Abre el sitio web oficial y veamos cómo hacer que el programa funcione.

El sitio web oficial proporciona un script:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

En Mac también puedes usar Homebrew, el gestor de paquetes del sistema, para instalar. Puedes ejecutar el siguiente comando:

```
brew install rust
```

Aquí estoy utilizando Homebrew para instalar Rust. Mientras esperamos a que se complete la instalación, sigamos revisando el sitio web oficial.

A continuación, vemos que en el sitio oficial aparece algo llamado Cargo, que es la herramienta de construcción y gestión de paquetes de Rust.

El sitio web oficial dice:

- compila tu proyecto con `cargo build`
- ejecuta tu proyecto con `cargo run`
- prueba tu proyecto con `cargo test`

Nos dice cómo construir, ejecutar y probar programas con Cargo.

Ejecutar:

```
brew install rust
```

Nota: El comando anterior es en inglés y no necesita traducción, ya que es una instrucción técnica para instalar Rust utilizando Homebrew en sistemas macOS.

Salida:

```
==> Descargando https://homebrew.bintray.com/bottles/rust-1.49.0_1.big_sur.bottle.tar.gz
==> Descargando desde https://d29vzk4ow07wi7.cloudfront.net/5a238d58c3fa775fed4e12ad74109deff54a82a06cb
######################################################################## 100.0%
==> Extrayendo rust-1.49.0_1.big_sur.bottle.tar.gz
==> Notas
La finalización de Bash se ha instalado en:
  /usr/local/etc/bash_completion.d
==> Resumen
   /usr/local/Cellar/rust/1.49.0_1: 15,736 archivos, 606.2MB
```

Esto significa que la instalación se ha completado con éxito.

Al ejecutar `cargo` en la terminal, la salida es la siguiente:

```
El gestor de paquetes de Rust
```

USO: cargo [OPCIONES] [SUBCOMANDO]

OPCIONES: -V, –version Imprime la información de la versión y sale –list Lista los comandos instalados –explain Ejecuta `rustc --explain CÓDIGO` -v, –verbose Usa salida detallada (-vv muy detallada/salida de build.rs) -q, –quiet No imprime salida en stdout –color Coloreado: auto, siempre, nunca –frozen Requiere que Cargo.lock y la caché estén actualizados –locked Requiere que Cargo.lock esté actualizado –offline Ejecuta sin acceder a la red -Z … Banderas inestables (solo para nightly) de Cargo, consulta ‘cargo -Z help’ para más detalles -h, –help Imprime la información de ayuda

Algunos comandos comunes de cargo son (ver todos los comandos con –list): build, b Compila el paquete actual check, c Analiza el paquete actual y reporta errores, pero no genera archivos objeto clean Elimina el directorio target doc Genera la documentación de este paquete y sus dependencias new Crea un nuevo paquete de cargo init Crea un nuevo paquete de cargo en un directorio existente run, r Ejecuta un binario o ejemplo del paquete local test, t Ejecuta las pruebas bench Ejecuta los benchmarks update Actualiza las dependencias listadas en Cargo.lock search Busca en el registro por crates publish Empaqueta y sube este paquete al registro install Instala un binario de Rust. La ubicación predeterminada es $HOME/.cargo/bin uninstall Desinstala un binario de Rust

Consulta ‘cargo help ’ para obtener más información sobre un comando específico.

```
No es necesario entender todos los comandos. Solo necesitas conocer los comandos más comunes. Los coman
```

Continuemos revisando la documentación oficial:

```c
Escribamos una pequeña aplicación con nuestro nuevo entorno de desarrollo en Rust. Para comenzar, usare

cargo new hello-rust
```

Esto generará un nuevo directorio llamado `hello-rust` con los siguientes archivos:

hello-rust |- Cargo.toml |- src |- main.rs

Cargo.toml es el archivo de manifiesto para Rust. Es donde se guardan los metadatos del proyecto, así como las dependencias.

`src/main.rs` es donde escribiremos el código de nuestra aplicación.

```
Esto explica cómo crear el proyecto. A continuación, procedemos a crearlo.

$ cargo new hello-rust
```

Se ha creado el paquete binario (aplicación) `hello-rust`

```
Abrimos el proyecto con VSCode.

main.rs: (no se traduce, ya que es un nombre de archivo en inglés)
```

```rust
fn main() {
    println!("¡Hola, mundo!");
}
```

A continuación, es natural pensar en compilar y ejecutar el programa.

```
$ cargo build
```

error: no se pudo encontrar `Cargo.toml` en `/Users/lzw/ideas/curious-courses/program/run/rust`
o en ningún directorio superior

```
Ocurrió un error. ¿Por qué? Esto indica que `cargo` solo puede ejecutarse en el directorio del proyecto
```

En este momento, me pregunto qué pasaría si lo ejecuto directamente.

```shell
$ cargo run

   Compilando hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
    Finalizado dev [sin optimización + debuginfo] objetivo(s) en 4.43s
     Ejecutando `target/debug/hello-rust`
¡Hola, mundo!
```

¡Bien, lo logré! El programa ha comenzado a funcionar y ha generado la cadena de texto.

Intenta modificar el programa.

```rust
fn main() {
    println!(2+3);
}
```

Nota: El código en Rust no se traduce, ya que es un lenguaje de programación y su sintaxis debe permanecer en inglés.

Después de ejecutar `cargo run`, apareció:

```
   Compilando hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
error: el argumento de formato debe ser un literal de cadena
 --> src/main.rs:2:14
  |
2 |     println!(2+3);
  |              ^^^
  |
help: podrías estar olvidando un literal de cadena para formatear
  |
2 |     println!("{}", 2+3);
  |              ^^^^^

error: abortiendo debido a un error previo

error: no se pudo compilar `hello-rust`

Para obtener más información, ejecuta el comando nuevamente con –verbose.
```

```
Aún no he aprendido ninguna sintaxis de Rust. Siguiendo nuestra intuición para modificar el código, com
```

```rust
fn main() {
    println!("{}", 2+3);
}
```

Esta vez lo hicimos bien, efectivamente se imprimió 5.

Por cierto, ¿qué pasa con el build?

```
$ cargo build
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
```

¿Por qué necesitamos un `build`? Porque es posible que solo queramos generar el programa ejecutable sin ejecutarlo. Tal vez, para algunos programas grandes, la ejecución sea costosa en términos de tiempo. O quizás queremos generar el programa localmente y luego transferirlo a un servidor remoto para su ejecución.

Ya hemos logrado ejecutar un programa en Rust. A continuación, se trata de familiarizarnos con más sintaxis del lenguaje Rust, para encontrar en Rust las representaciones simbólicas correspondientes a los conceptos que hemos discutido en “Desentrañando la Ciencia de la Computación”, como variables, funciones, llamadas a funciones y expresiones.

Pequeño ejercicio

- Intenta, como en el ejemplo anterior, que los estudiantes prueben la programación en Rust en sus propias computadoras.
- Después de practicar, pueden enviar un resumen de cien palabras o una adición al artículo.
