Rust □□□□□

Rust ist in den letzten Jahren eine beliebte Programmiersprache geworden. Im Jahr 2006 begann ein Mitarbeiter von Mozilla ein persönliches Projekt, das später die Unterstützung des Unternehmens erhielt und im Jahr 2010 veröffentlicht wurde. Dieses Projekt heißt Rust.

Als Nächstes starten wir das erste Rust-Programm. Öffnen Sie die offizielle Website, um zu sehen, wie man das Programm zum Laufen bringt.

Die offizielle Website bietet ein Skript an:

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

Auf dem Mac kann auch das Paketverwaltungstool Homebrew des Mac-Systems zur Installation verwendet werden. Sie können den folgenden Befehl ausführen:

```
brew install rust
```

Ich verwende hier Homebrew, um Rust zu installieren. Während der Installation lasst uns weiterhin die offizielle Website betrachten.

Als nächstes sehen wir auf der offiziellen Website etwas namens Cargo, das Build-Tool und Paketmanager von Rust.

Die offizielle Website besagt:

- Erstellen Sie Ihr Projekt mit `cargo build`
- Führen Sie Ihr Projekt mit `cargo run` aus
- Testen Sie Ihr Projekt mit `cargo test`

Erklären Sie, wie man ein Cargo-Programm erstellt, ausführt und testet.

Ausführen:

```
brew install rust
```

Ausgabe:

```
==> Herunterladen von https://homebrew.bintray.com/bottles/rust-1.49.0_1.big_sur.bottle.tar.gz
==> Herunterladen von https://d29vzk4ow07wi7.cloudfront.net/5a238d58c3fa775fed4e12ad74109deff54a82a06cb
######################################################################## 100.0%
```

```
==> Entpacken von rust-1.49.0_1.big_sur.bottle.tar.gz
==> Hinweise
Die Bash-Vervollständigung wurde installiert unter:
  /usr/local/etc/bash_completion.d
==> Zusammenfassung
   /usr/local/Cellar/rust/1.49.0_1: 15.736 Dateien, 606,2MB
```

Das bedeutet, die Installation war erfolgreich.

Wenn Sie `cargo` im Terminal ausführen, wird die folgende Ausgabe angezeigt:

```
Rusts Paketmanager
```

VERWENDUNG: cargo [OPTIONEN] [UNTERBEFEHL]

OPTIONEN: -V, –version Versionsinformation anzeigen und beenden –list Installierte Befehle auflisten –explain `rustc --explain CODE` ausführen -v, –verbose Ausführliche Ausgabe verwenden (-vv sehr ausführlich/build.rs Ausgabe) -q, –quiet Keine Ausgabe auf stdout –color Farbgebung: auto, immer, nie –frozen Erfordert, dass Cargo.lock und Cache aktuell sind –locked Erfordert, dass Cargo.lock aktuell ist –offline Ohne Netzwerkzugriff ausführen -Z … Instabile (nur nightly) Flags für Cargo, siehe ‘cargo -Z help’ für Details -h, –help Hilfeinformationen anzeigen

Einige gängige Cargo-Befehle sind (alle Befehle mit –list anzeigen): build, b Aktuelles Paket kompilieren check, c Aktuelles Paket analysieren und Fehler melden, aber keine Objektdateien erstellen clean Zielverzeichnis entfernen doc Dokumentation dieses Pakets und seiner Abhängigkeiten erstellen new Neues Cargo-Paket erstellen init Neues Cargo-Paket in einem bestehenden Verzeichnis erstellen run, r Binärdatei oder Beispiel des lokalen Pakets ausführen test, t Tests ausführen bench Benchmarks ausführen update In Cargo.lock aufgeführte Abhängigkeiten aktualisieren search In der Registry nach Crates suchen publish Paket erstellen und in die Registry hochladen install Rust-Binärdatei installieren. Standardort ist $HOME/.cargo/bin uninstall Rust-Binärdatei deinstallieren

Siehe ‘cargo help ’ für weitere Informationen zu einem bestimmten Befehl.

```
Es ist nicht notwendig, alle Befehle zu verstehen. Es reicht aus, die häufig verwendeten Befehle zu kenn

Weiter geht es mit der offiziellen Dokumentation:

```c
Lassen Sie uns eine kleine Anwendung mit unserer neuen Rust-Entwicklungsumgebung schreiben. Um zu beginn
```
```

```
cargo new hello-rust
```

Dies erzeugt ein neues Verzeichnis namens `hello-rust` mit den folgenden Dateien:

hello-rust |- Cargo.toml |- src |- main.rs Cargo.toml ist die Manifest-Datei für Rust. Hier werden die Metadaten für Ihr Projekt sowie die Abhängigkeiten gespeichert.

`src/main.rs` ist der Ort, an dem wir unseren Anwendungscode schreiben werden.

```
Dies erklärt, wie man ein Projekt erstellt. Als nächstes wird es erstellt.
```

```
$ cargo new hello-rust
```

Binäres (Anwendungs-)Paket `hello-rust` erstellt

```
Wir öffnen das Projekt mit VSCode.
```

```
main.rs:
```

```rust
fn main() {
    println!("Hallo, Welt!");
}
```

Als Nächstes liegt es nahe, daran zu denken, das Programm zu **builden** und auszuführen.

```
$ cargo build
```

Fehler: `Cargo.toml` konnte nicht in `/Users/lzw/ideas/curious-courses/program/run/rust` oder in einem übergeordneten Verzeichnis gefunden werden

```
Es ist ein Fehler aufgetreten. Warum? Dies deutet darauf hin, dass Cargo nur in Verzeichnissen innerhall
```

```
In diesem Moment dachte ich, was passieren würde, wenn ich es einfach direkt ausführe.
```

```shell
$ cargo run

   Kompiliere hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
```

```
    Fertigstellung dev [unoptimized + debuginfo] Ziel(e) in 4.43s
     Ausführung von `target/debug/hello-rust`
Hallo, Welt!
```

Gut, es hat geklappt. Der String wurde ausgegeben, und das Programm funktioniert jetzt.

Versuchen Sie, das Programm zu ändern.

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

(Der Code bleibt auf Englisch, da es sich um eine Programmiersprache handelt und die Syntax nicht übersetzt wird.)

Nachdem `cargo run` ausgeführt wurde, erschien:

```
   Compiling hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
error: Format-Argument muss ein String-Literal sein
 --> src/main.rs:2:14
  |
2 |     println!(2+3);
  |              ^^^
  |
help: Möglicherweise fehlt ein String-Literal für die Formatierung
  |
2 |     println!("{}", 2+3);
  |              ^^^^^

Fehler: Abbruch aufgrund eines vorherigen Fehlers

Fehler: hello-rust konnte nicht kompiliert werden

Um mehr zu erfahren, führen Sie den Befehl erneut mit --verbose aus.
```

```
Ich habe noch keine Rust-Syntax gelernt. Basierend auf unserer Intuition haben wir den Code geändert, wa


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

(Der Code bleibt auf Englisch, da es sich um eine Programmiersprache handelt und die Syntax nicht übersetzt wird.)

Diesmal war es richtig, und tatsächlich wurde 5 ausgegeben.

Was passiert mit dem Build?

```
$ cargo build
    Fertigstellung dev [unoptimized + debuginfo] Ziel(e) in 0.00s
```

Warum gibt es den `build`-Befehl? Weil es Situationen gibt, in denen wir nur die ausführbare Datei generieren möchten, ohne sie direkt auszuführen. Bei einigen umfangreichen Programmen kann die Ausführung zeitaufwändig sein. Vielleicht möchten wir die Datei lokal erstellen und dann auf einen Remote-Server übertragen, um sie dort auszuführen.

Wir haben das Rust-Programm bereits zum Laufen gebracht. Als Nächstes geht es darum, die Syntax der Rust-Sprache weiter zu vertiefen, um in Rust die symbolischen Darstellungen der Konzepte wie Variablen, Funktionen, Funktionsaufrufe und Ausdrücke zu finden, die wir in „Demystifying Computer Science“ behandelt haben.

Kleine Übung

- Versuchen Sie, wie oben gezeigt, Rust-Programmierung auf Ihrem eigenen Computer auszuprobieren.
- Nach Abschluss der Übungen können Sie eine Zusammenfassung von maximal hundert Wörtern oder Ergänzungen zu diesem Artikel einreichen.
