Rust プログラミングを試す

Rust はここ数年で注目を集めているプログラミング言語です。2006 年、Mozilla の従業員が個人プロジェクトとして始め、後に会社の支援を受けて 2010 年にこのプロジェクトを発表しました。このプロジェクトの名前が Rust です。

次に、Rust の最初のプログラムを実行してみましょう。公式サイトを開いて、プログラムを実行する方法を見てみましょう。

公式サイトでは以下のスクリプトを提供しています：

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

このコマンドは、Rust プログラミング言語のインストールスクリプトをダウンロードして実行します。`curl` コマンドを使用して、`https://sh.rustup.rs` からスクリプトを取得し、`sh` コマンドでそのスクリプトを実行します。`--proto '=https'`と`--tlsv1.2` オプションは、セキュアな HTTPS 接続を使用することを保証します。`-sSf` オプションは、進捗表示を抑制し、エラーが発生した場合にのみ出力を表示します。

Mac 上でも、Mac システムのパッケージ管理ツールである Homebrew を使用してインストールすることができます。以下のコマンドを実行してください：

```
brew install rust
```

ここでは Homebrew を使って rust をインストールします。インストール中に、引き続き公式サイトを見ていきましょう。

次に、公式サイトに Cargo というものが登場しました。これは Rust のビルドツール兼パッケージ管理ツールです。

公式ウェブサイトには以下のように記載されています：

- `cargo build` でプロジェクトをビルドする
- `cargo run` でプロジェクトを実行する
- `cargo test` でプロジェクトをテストする

Cargo プログラムの構築、実行、テスト方法を教えてくれます。

実行：

```
brew install rust
```

上記のコマンドは、macOS 用のパッケージマネージャーである Homebrew を使用して、Rust プログラミング言語をインストールするためのものです。このコマンドを実行すると、Homebrew が自動的に Rust の最新バージョンをダウンロードし、システムにインストールします。

出力：

```
==> Downloading https://homebrew.bintray.com/bottles/rust-1.49.0_1.big_sur.bottle.tar.gz
==> Downloading from https://d29vzk4ow07wi7.cloudfront.net/5a238d58c3fa775fed4e12ad74109deff54a82a06cb6a
######################################################################## 100.0%
==> Pouring rust-1.49.0_1.big_sur.bottle.tar.gz
==> Caveats
Bashの補完機能が以下の場所にインストールされました:
  /usr/local/etc/bash_completion.d
==> Summary
   /usr/local/Cellar/rust/1.49.0_1: 15,736ファイル, 606.2MB
```

これでインストールが成功しました。

端末で `cargo` を実行すると、以下のような出力が得られます：

```
Rustのパッケージマネージャー

使用方法: cargo [オプション] [サブコマンド]

OPTIONS: -V, –version バージョン情報を表示して終了します –list インストールされているコマンドを一覧表示します –explain `rustc --explain CODE` を実行します -v, –verbose 詳細な出力を使用します (-vv は非常に詳細/ビルド.rs の出力) -q, –quiet stdout に出力を表示しません –color カラーリング: auto, always, never –frozen Cargo.lock とキャッシュが最新であることを要求します –locked Cargo.lock が最新であることを要求します –offline ネットワークにアクセスせずに実行します -Z … Cargo の不安定（nightly 専用）フラグ、詳細は ‘cargo -Z help’ を参照 -h, –help ヘルプ情報を表示します

一般的な Cargo コマンドには以下のものがあります（すべてのコマンドは–list で確認できます）：
build, b 現在のパッケージをコンパイルする check, c 現在のパッケージを分析してエラーを報告するが、オブジェクトファイルはビルドしない clean ターゲットディレクトリを削除する doc このパッケージとその依存関係のドキュメントをビルドする new 新しい Cargo パッケージを作成する init 既存のディレクトリに新しい Cargo パッケージを作成する run, r ローカルパッケージのバイナリまたは例を実行する test, t テストを実行する bench ベンチマークを実行する update Cargo.lock にリストされている依存関係を更新する search レジストリでクレートを検索する publish このパッケージをパッケージ化してレジストリにアップロードする install Rust バイナリ
```

をインストールする。デフォルトの場所は $HOME/.cargo/bin uninstall Rust バイナリをアンインストールする

特定のコマンドに関する詳細情報は、‘cargo help ’ を参照してください。

すべてのコマンドを理解する必要はありません。よく使うコマンドを知っていれば十分です。`build` と `run` コマンドは特

引き続き公式ドキュメントを見ていきましょう：

```c
新しいRust開発環境で小さなアプリケーションを書いてみましょう。まず、Cargoを使って新しいプロジェクトを作成します。
```

```
cargo new hello-rust
```

これにより、`hello-rust` という名前の新しいディレクトリが生成され、以下のファイルが含まれます:

hello-rust |- Cargo.toml |- src |- main.rs Cargo.toml は Rust のマニフェストファイルです。プロジェクトのメタデータや依存関係をここに記述します。

`src/main.rs` は、私たちがアプリケーションのコードを書く場所です。

これはプロジェクトの作成方法について説明しています。次に、実際に作成します。

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

バイナリ（アプリケーション）`hello-rust` パッケージを作成しました

プロジェクトをVSCodeで開きます。

```rust
main.rs:

fn main() {
    println!("Hello, world!");
}
```

この Rust のコードは、コンソールに「Hello, world!」と表示するシンプルなプログラムです。Rust の基本的な構文を示しています。

次に、当然の流れとして、プログラムをビルドして実行することを考えます。

```
$ cargo build
```

エラー: /Users/lzw/ideas/curious-courses/program/run/rust またはその親ディレクトリ内に Cargo.toml が見つかりませんでした

エラーが発生しました。なぜでしょうか。これは、cargoがプロジェクトのディレクトリ内でのみ実行できることを示しています

この時、もし直接実行したらどうなるかと考えます。

```shell
$ cargo run

   Compiling hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
    Finished dev [unoptimized + debuginfo] target(s) in 4.43s
     Running `target/debug/hello-rust`
Hello, world!
```

このコードは、Rust のプロジェクトをコンパイルして実行するためのコマンドとその結果を示しています。cargo run コマンドを使用して、Rust のプロジェクトをビルドし、実行しています。ビルドが成功すると、Hello, world! というメッセージが表示されます。

よし、成功だ。文字列が出力され、プログラムが動き始めた。

プログラムを変更してみます。

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

この Rust のコードは、main 関数内で println! マクロを使用して、2 + 3 の結果を出力しようとしています。しかし、このコードにはいくつかの問題があります。

1. println! マクロはフォーマット文字列を必要とします。単に 2 + 3 を渡すと、コンパイルエラーになります。
2. 2 + 3 の結果を出力するためには、フォーマット文字列内でプレースホルダーを使用する必要があります。

正しいコードは以下のようになります：

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

このコードでは、println! マクロにフォーマット文字列"{}" を渡し、その中に 2 + 3 の結果を埋め込んでいます。これにより、5 が出力されます。

cargo run を実行した後、以下のようなエラーが発生しました：

```
   Compiling hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
エラー: フォーマット引数は文字列リテラルでなければなりません
 --> src/main.rs:2:14
  |
2 |     println!(2+3);
  |              ^^^
  |
ヘルプ: フォーマットするための文字列リテラルが不足している可能性があります
  |
2 |     println!("{}", 2+3);
  |              ^^^^^

エラー: 以前のエラーのため中断しています

エラー: hello-rust をコンパイルできませんでした

詳細を確認するには、--verbose オプションを付けて再度コマンドを実行してください。
```

まだRustの文法を学んでいません。私たちの直感に従ってコードを変更したところ、エラーが発生しました。このエラーメッセ

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

この Rust のコードは、main 関数内で println! マクロを使用して、2 + 3 の結果を出力します。具体的には、2 + 3 の計算結果である 5 が出力されます。

今度は正しく修正され、確かに 5 が出力されました。

そうだ、build はどうなるんだろう。

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

（注：このコードブロックはコマンドの実行結果を示しており、翻訳の必要はありません。）

なぜ build が必要なのでしょうか。それは、単に実行可能なプログラムを生成したいだけで、実際に実行したくない場合があるからです。特に大規模なプログラムの場合、実行には時間がかかることがあります。また、ローカルで生成した後、リモートサーバーに転送して実行したい場合もあるでしょう。

私たちはすでに Rust プログラムを実行することができました。次に、Rust 言語のより多くの構文に慣れ親しむことで、「コンピュータサイエンスの謎解き」で説明した変数、関数、関数呼び出し、式などの概念に対応する Rust のシンボル表現を見つけ出します。

小练习

- 上記のように、学生が自分のコンピュータで Rust プログラミングを試してみる。
- 練習が終わったら、100 字以内の要約またはこの記事への補足を提出することができる。
