# Trying Out Rust Programming

Rust is a relatively popular programming language in recent years. In 2006, an employee at Mozilla started a personal project, which later received company support and was released in 2010. This project is called Rust.

Let’ s start by running our first Rust program. Visit the official website to see how to get the program running.

The official website provides a script:

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

On a Mac, you can also use the Mac system’ s package management tool Homebrew to install it. Run the command:

```
brew install rust
```

Here, I’ ll use Homebrew to install Rust. While it’ s installing, let’ s continue exploring the official website.

Next, we see Cargo mentioned on the website. Cargo is Rust’ s build and package management tool.

The official website states:

- Build your project with `cargo build`
- Run your project with `cargo run`
- Test your project with `cargo test`

These commands show us how to build, run, and test a Cargo program.

Run:

```
brew install rust
```

The output is:

```
==> Downloading https://homebrew.bintray.com/bottles/rust-1.49.0_1.big_sur.bottle.tar.gz
==> Downloading from https://d29vzk4ow07wi7.cloudfront.net/5a238d58c3fa775fed4e12ad74109deff54a82a06cb6
######################################################################## 100.0%
==> Pouring rust-1.49.0_1.big_sur.bottle.tar.gz
==> Caveats
Bash completion has been installed to:
  /usr/local/etc/bash_completion.d
==> Summary
   /usr/local/Cellar/rust/1.49.0_1: 15,736 files, 606.2MB
```

This means the installation was successful.

When running `cargo` in the terminal, the output is as follows:

```
Rust's package manager


USAGE:
    cargo [OPTIONS] [SUBCOMMAND]


OPTIONS:
    -V, --version           Print version info and exit
        --list              List installed commands
        --explain <CODE>    Run `rustc --explain CODE`
    -v, --verbose           Use verbose output (-vv very verbose/build.rs output)
    -q, --quiet             No output printed to stdout
        --color <WHEN>      Coloring: auto, always, never
        --frozen            Require Cargo.lock and cache are up to date
        --locked            Require Cargo.lock is up to date
        --offline           Run without accessing the network
    -Z <FLAG>...            Unstable (nightly-only) flags to Cargo, see 'cargo -Z help' for details
    -h, --help              Prints help information


Some common cargo commands are (see all commands with --list):
    build, b    Compile the current package
    check, c    Analyze the current package and report errors, but don't build object files
    clean       Remove the target directory
    doc         Build this package's and its dependencies' documentation
    new         Create a new cargo package
    init        Create a new cargo package in an existing directory
    run, r      Run a binary or example of the local package
    test, t     Run the tests
    bench       Run the benchmarks
    update      Update dependencies listed in Cargo.lock
    search      Search registry for crates
    publish     Package and upload this package to the registry
    install     Install a Rust binary. Default location is $HOME/.cargo/bin
    uninstall   Uninstall a Rust binary


See 'cargo help <command>' for more information on a specific command.
```

No need to understand all the commands. Just knowing the commonly used ones is enough. The build and run commands are crucial.

Continuing with the official documentation:

```
Let’ s write a small application with our new Rust development environment. To start, we’ ll use Cargo

cargo new hello-rust

This will generate a new directory called hello-rust with the following files:

hello-rust
|- Cargo.toml
|- src
  |- main.rs
Cargo.toml is the manifest file for Rust. It’ s where you keep metadata for your project, as well as de

src/main.rs is where we’ ll write our application code.
```

This explains how to create a project. Let’ s proceed.

```
$ cargo new hello-rust

Created binary (application) `hello-rust` package
```

We’ ll use VSCode to open the project.

main.rs:

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

Next, we naturally want to build and run the program.

```
$ cargo build

error: could not find `Cargo.toml` in `/Users/lzw/ideas/curious-courses/program/run/rust` or any parent
```

An error occurred. Why? This indicates that Cargo can only run in the project’ s directory. So, navigate to the subdirectory by running `cd hello-rust`.

Now, let’ s see what happens if we run it directly.

```
$ 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!
```

Success! The string was output, and the program is working.

Let' s try modifying the program.

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

After running `cargo run`, the output was:

```
   Compiling hello-rust v0.1.0 (/Users/lzw/ideas/curious-courses/program/run/rust/hello-rust)
error: format argument must be a string literal
 --> src/main.rs:2:14
  |
2 |     println!(2+3);
  |              ^^^
  |
help: you might be missing a string literal to format with
  |
2 |     println!("{}", 2+3);
  |              +++++

error: aborting due to previous error

error: could not compile `hello-rust`

To learn more, run the command again with --verbose.
```

We haven' t learned any Rust syntax yet. Our intuitive code modification caused an error. The error message was helpful and suggested how to fix it.

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

This time, it worked, and it output 5.

Now, what about the build command?

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

Why have a build command? Because sometimes we just want to generate an executable without running it. For large programs, execution can be time-consuming. We might want to generate the executable locally and then transfer it to a remote server for execution.

We’ ve successfully run a Rust program. Next, we’ ll familiarize ourselves with more Rust syntax and find the corresponding symbols for concepts like variables, functions, function calls, and expressions as discussed in “Decoding Computer Science.”

---

**Exercise**

- Try using Rust programming on your computer as described above.
- After practicing, submit a summary or additions to this article within 100 words.

---