Rust in Replit

By Shaun Hamilton

Introduction

01 **Rust Overview**

02 Basics of Rust

03 CLI Calculator

04 Image Combiner

Introduction

01 Rust Overview

02 **Basics of Rust**

03 CLI Calculator

04 Image Combiner

Introduction

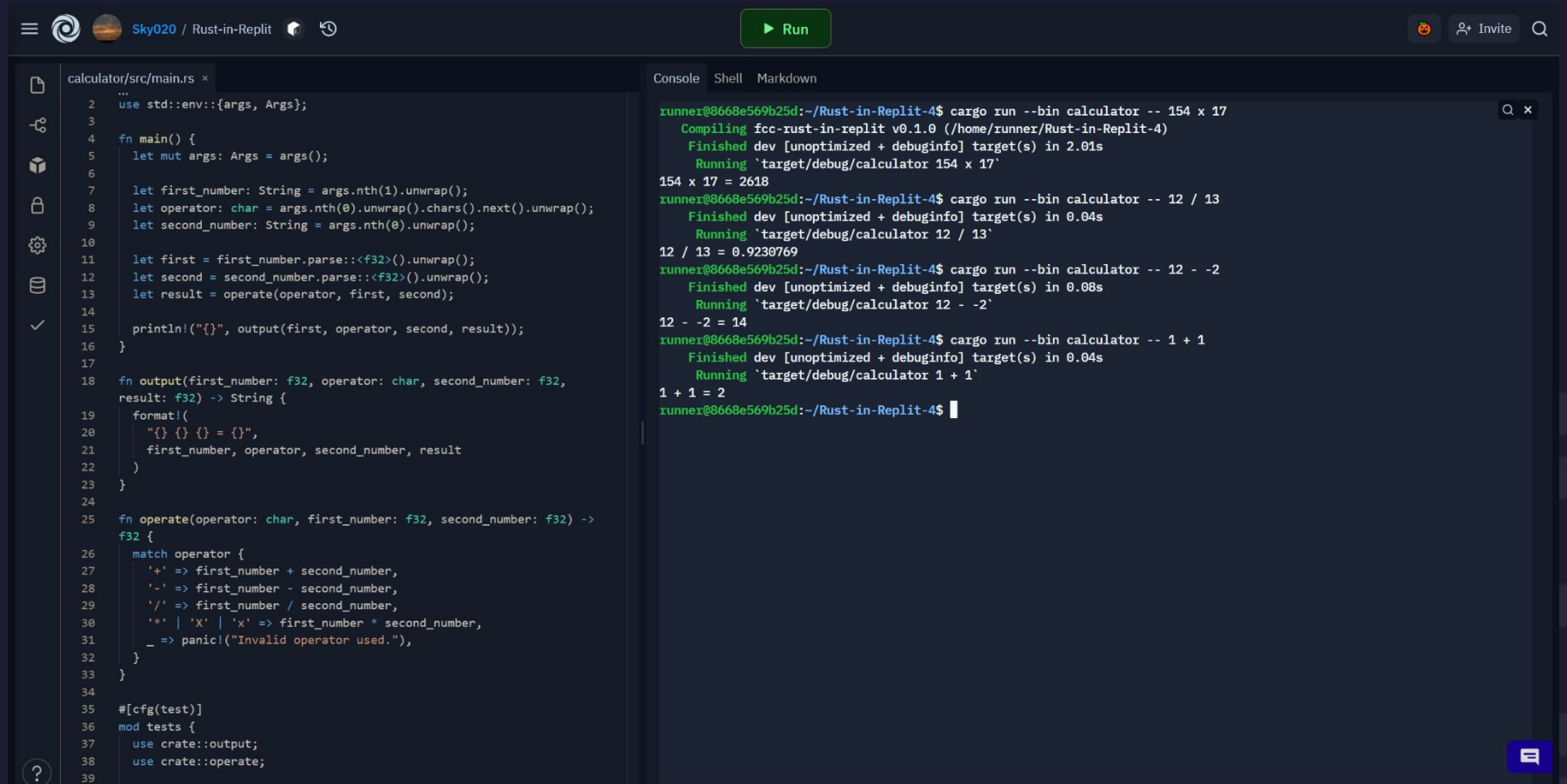
01 Rust Overview

02 Basics of Rust

03 **CLI Calculator**

04 Image Combiner

CLI Calculator



Introduction

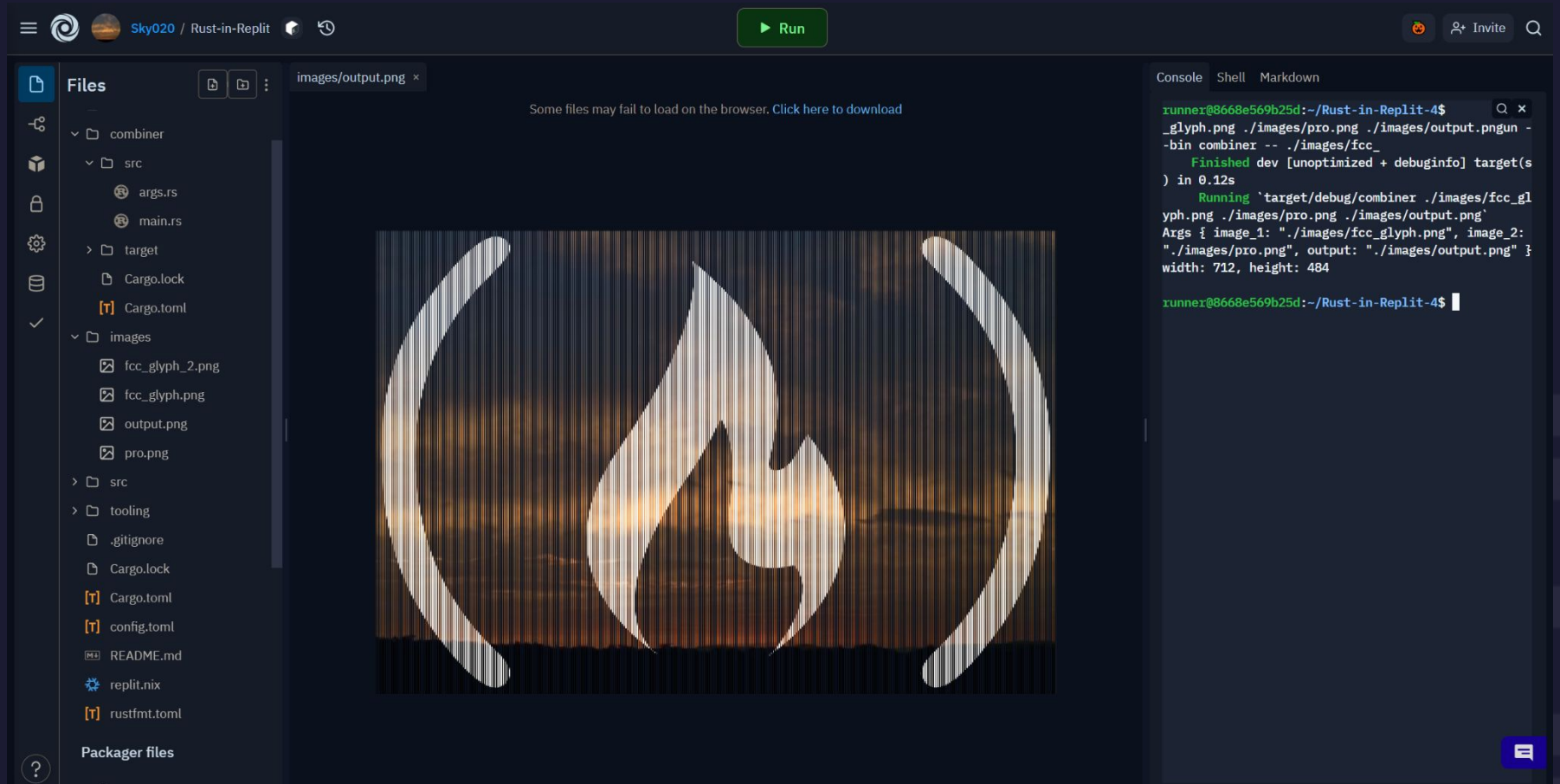
01 Rust Overview

02 Basics of Rust?

03 CLI Calculator

04 **Image Combiner**

Image Combiner



Rust Overview

"[Rust] deals with low-level details of memory management, data representation, and concurrency."

"... the language is designed to guide you naturally towards reliable code that is efficient in terms of speed and memory usage."

(Source: Nicholas Matsakis and Aaron Turon)

Rust Overview

- rustc - The compiler which takes your Rust code and compiles it into binary (machine readable code)
- rustup - The command line utility to install and update Rust
- cargo - The Rust build system and package manager

Basics of Rust - Variables

```
let my_variable = 0;
const MY_CONSTANT: u8 = 0;
static MY_STATIC: u8 = 0;

let mut my_mutable_variable = 0;
```

Basics of Rust - Functions

```rust
fn main() -> () { // Unnecessary return type
  my_func();
}

fn my_func() -> u8 {
  return 0;
}
```

Basics of Rust - Functions

```
fn my_func() -> u8 {
  0
}
```

Basics of Rust - Functions

```
fn main() {
  let _unused_variable = my_func(10);
}

fn my_func(x: u8) -> i32 {
  x as i32
}
```

Basics of Rust - String and Slices

```rust
let my_str: &str = "Hello, world!";

let my_string: String = String::from("Hello, world!");
```

Basics of Rust - String and Slices

```
let my_string = String::from("The quick brown fox");
let my_str: &str = &my_string[4..9]; // "quick"

let my_arr: [usize; 5] = [1, 2, 3, 4, 5];
let my_arr_slice: &[usize] = &my_arr[0..3]; // [1, 2, 3]
```

Basics of Rust - The `char` Type

```
let my_str: &str = "Hello, world!";

let collection_of_chars: &str = my_str.chars().as_str();
```

Basics of Rust - Number Types

There are many types of number in Rust:

- Unsigned Integers: `u8`, `u16`, `u32`, `u64`, `u128`
- Signed Integers: `i8`, `i16`, `i32`, `i64`, `i128`
- Floating Point Numbers: `f32`, `f64`

Unsigned integers only represent positive whole numbers.
Signed integers represent both positive and negative whole numbers.
Floats only represent positive and negative fractions.

Basics of Rust - Structs

```
struct String {
  vec: Vec<u8>,
}
```

Basics of Rust - Structs

```
struct MyStruct {
  field_1: u8,
}

let my_struct = MyStruct { field_1: 0, };
```

Basics of Rust - Structs

```
impl String {
  fn from(s: &str) -> Self {
    String {
      vec: Vec::from(s.as_bytes()),
    }
  }
}
```

Basics of Rust - Structs

```
struct MyUnitStruct;
struct MyTupleStruct(u8, u8);
```

Basics of Rust - Enums

```rust
enum MyErrors {
  BrainTooTired,
  TimeOfDay(String)
  CoffeeCupEmpty,
}
fn work() -> Result<(), MyErrors> { // Result is also an enum
  if state == "missing semi-colon" {
    Err(MyErrors::BrainTooTired)
  } else if state == "06:00" {
    Err(MyErrors::TImeOfDay("It's too early to work".to_string()))
  } else if state == "22:00" {
    Err(MyErrors::TimeOfDay("It's too late to work".to_string()))
  } else if state == "empty" {
    Err(MyErrors::CoffeeCupEmpty)
  } else {
    Ok(())
  }
}
```

Basics of Rust - Macros

```
let my_str = "Hello, world!";
println!("{}", my_str);
```

Basics of Rust - Macros

```rust
let am_i_an_error = true;

if (am_i_an_error) {
  panic!("There was an error");
}
```

```
$ cargo run
   Compiling fcc-rust-in-replit v0.1.0 (/home/runner/Rust-in-Replit)
    Finished dev [unoptimized + debuginfo] target(s) in 1.66s
     Running `target/debug/calculator`
thread 'main' panicked at 'There was an error', src/main.rs
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

Basics of Rust - Ownership

An important concept in Rust is *_ownership_*. There are three main ownership rules:

> - **Each value in Rust has a variable that's called its *_owner_***.
> - **There can only be one owner at a time.**
> - **When the owner goes out of scope, the value will be dropped.**
> ([Source: The Rust Book](https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html?highlight=heap#ownership-rules))

This is how Rust gets away without having a typical garbage collector, whilst also not requiring the programmer to explicitly manage memory.

Basics of Rust - Ownership

```
fn main() { // first_string is not declared yet -> has no value
  let first_string = String::from("freeCodeCamp"); // first_string
is now owner of the value "freeCodeCamp"
  let second_string = first_string; // second_string takes
ownership of the value "freeCodeCamp"

  println!("Hello, {}!", first_string); // first_string is NOT
valid, because the value was moved to second_string
}
```

Basics of Rust - Ownership

```rust
fn main() {
  let first_string: String = String::from("freeCodeCamp");
  let second_string: &String = &first_string; // first_string is still the owner of the value "freeCodeCamp"

  println!("Hello, {}!", first_string);
}
```