Stepping-Stone Task: Basic Syntax and Variables

Objective

The goal of this task is to introduce you to basic programming concepts in Rust, focusing on variables, data types, and basic operations.

What are Variables?

A **variable** is like a container that holds a value. You can think of it as a labeled box where you can store information. For example, you can have a box labeled `age` that holds the number 15. You can use variables to store numbers, text, and other types of information so that you can use or change them later in your program.

What Does ”Declare” Mean?

To **declare** a variable means to create it and give it a name. Declaring a variable also tells the computer what kind of value it will hold (e.g., a number, a word, or true/false). You declare a variable so that you can use it in your program. Think of it as setting up a new box to store a specific type of information.

What are Data Types?

In programming, **data types** define what kind of information a variable can hold. Here are some basic data types in Rust:

- **Integer (`i32`)**: A whole number, like 5 or 10. The `i32` type means it's a 32-bit integer, which can hold positive or negative whole numbers.
- **Boolean (`bool`)**: A value that can be either true or false, used for yes/no or on/off situations.
- **String (`&str`)**: A sequence of characters, like ”Hello, Rust!” or ”123 Main St”. In Rust, strings are represented as `&str`, which is a reference to text.

Task Overview

You will write a simple Rust program that:

- Declares variables of different data types (e.g., integers, strings, booleans).
- Performs basic arithmetic operations.
- Prints the results to the screen.

Instructions for Codespaces

1. **Open Your Project in GitHub Codespaces**
   - Go to your forked `virtual-robot-maze` repository on GitHub.
   - Click the `Code` button and select `Codespaces`.
   - Create a new codespace, or open an existing one.
2. **Navigate to the `basic-programming` directory**
   - In Codespaces, open the terminal (usually at the bottom of the screen).
   - Navigate to the `basic-programming` folder that already exists in the project:

   ```
                 cd basic-programming
   ```

3. **Create a new Rust project for this task**
   - Inside the `basic-programming` directory, create a new Rust project:

   ```
                 cargo new basic_syntax
                 cd basic_syntax
   ```

   - This will create a folder named `basic_syntax` where you will work on this task.
4. **Declare Variables**
   - Declare variables of different types, like integers, strings, and booleans.
   - Use the `let` keyword to define variables:

   ```
                 let x: i32 = 5;        // Integer
                 let name: &str = "Rust"; // String
                 let is_learning: bool = true; // Boolean
   ```

   - Modify the values of these variables to understand how they work.
5. **Perform Basic Arithmetic**
   - Add two integer variables together and store the result in a new variable:

```rust
                let y: i32 = 10;
                let sum = x + y;
                println!("The sum is: {}", sum);
```

- Try other operations like subtraction, multiplication, and division.

6. **Print Results to the Screen**

    - Use the `println!` macro to display the values of the variables and results of operations:

```rust
                println!("Name: {}", name);
                println!("Is learning Rust: {}", is_learning);
```

What to Expect

- After running the program, you should see output similar to:

```
The sum is: 15
Name: Rust
Is learning Rust: true
```

- Modify the values of variables and re-run the program to observe different results.

Testing and Debugging

- To run your program, use:

```
            cargo run
```

- If you see an error message, try to understand what it says—Rust's error messages are often helpful.
- If the program runs successfully, try adding new variables or operations.

Hints

- Variables must have the correct data types, so `i32`, `&str`, and `bool` must match the values you assign.
- Experiment with different arithmetic operations (e.g., `-`, `*`, `/`) and see what results you get.
- If you get stuck, break the problem into smaller steps—start with one variable and build up from there.

Next Steps

Once you've completed this task, you will have a better understanding of how variables and data types work in Rust. This foundation will be helpful for the upcoming tasks in the virtual robot project.