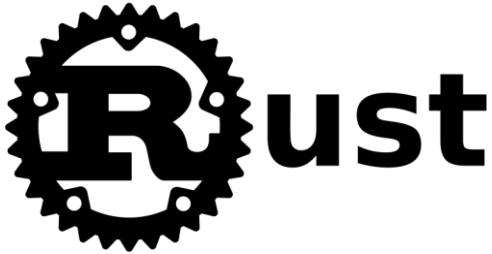
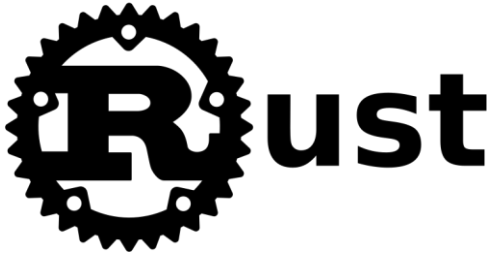
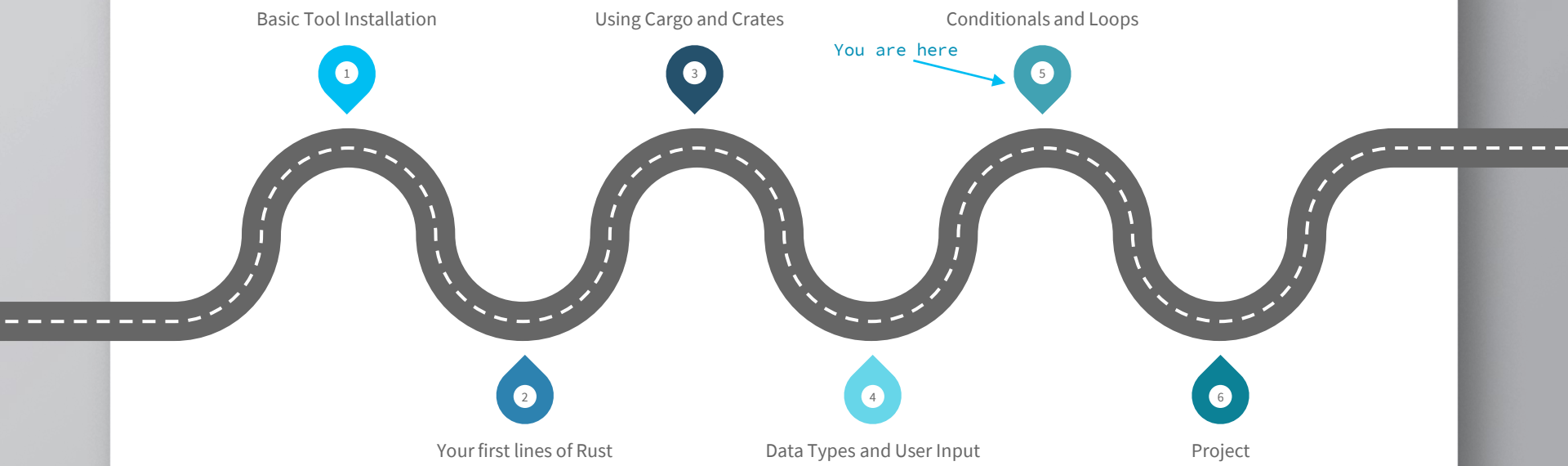
Introduction to Rust



Chapter 5



ROADMAP


Basic Tool Installation
1
Your first lines of Rust
2
Using Cargo and Crates
3
Data Types and User Input
4
Conditionals and Loops
You are here
5
Project
6

5.

Conditionals and Loops

Conditionals

A fundamental function of every programming language is to execute different commands according to the program's state e.g. the contents of the program's variables. The first command we are going to discuss is the **match Control Flow construct**.

The general syntax is:

```
match <variable> {
        x => <command>,
        y => <command>,
        other => <command>,
}
```

Let's make everything clear with an example.

Conditionals

Let's say we have the code from our previous lecture where we read a u8 number from the user and store it in the variable "number".Instead of just printing the number again we can use **match** on the variable to have a different behaviour. Let's use **cargo new conditionals-and-loops** and open the project in VSCode.

We will use the same code from the previous lecture up until the println! statement. Now let's **match** the variable number and print different statements depending on the value:

```
match number {
    2 => println!("You typed the number two"),
    8 => println!("You typed the number eight"),
    _other: u8 => println!("You did not type 2 or 8"),
}
```

Use **cargo run** and try some values to see the output!

(using the underscore _ before a variable name explains to the compiler that we are not going to use it anywhere else)

Conditionals

What we did now is kind of silly functionally so let's try something more useful. We will use the Ordering functionality of the standard cmp (compare) crate. In order to do that we have to include on the top:

use std::cmp::Ordering;

Now we can use the method .cmp() to compare two numbers and change the program's behaviour with the match statement. Let's declare a u8 variable called "ten" and give it the value 10. Then we can use the match statement in the expression **number.cmp(&ten)** and act accordingly on every **Ordering** possibility:

```
10      let ten: u8 = 10;
11      match number.cmp(&ten) {
12          Ordering::Less => println!("Smaller than 10."),
13          Ordering::Greater => println!("Bigger than 10."),
14          Ordering::Equal => println!("Exactly 10."),
15      }
```

Now by using **cargo run** the program tells us if our input is smaller than, bigger than, or exactly 10.

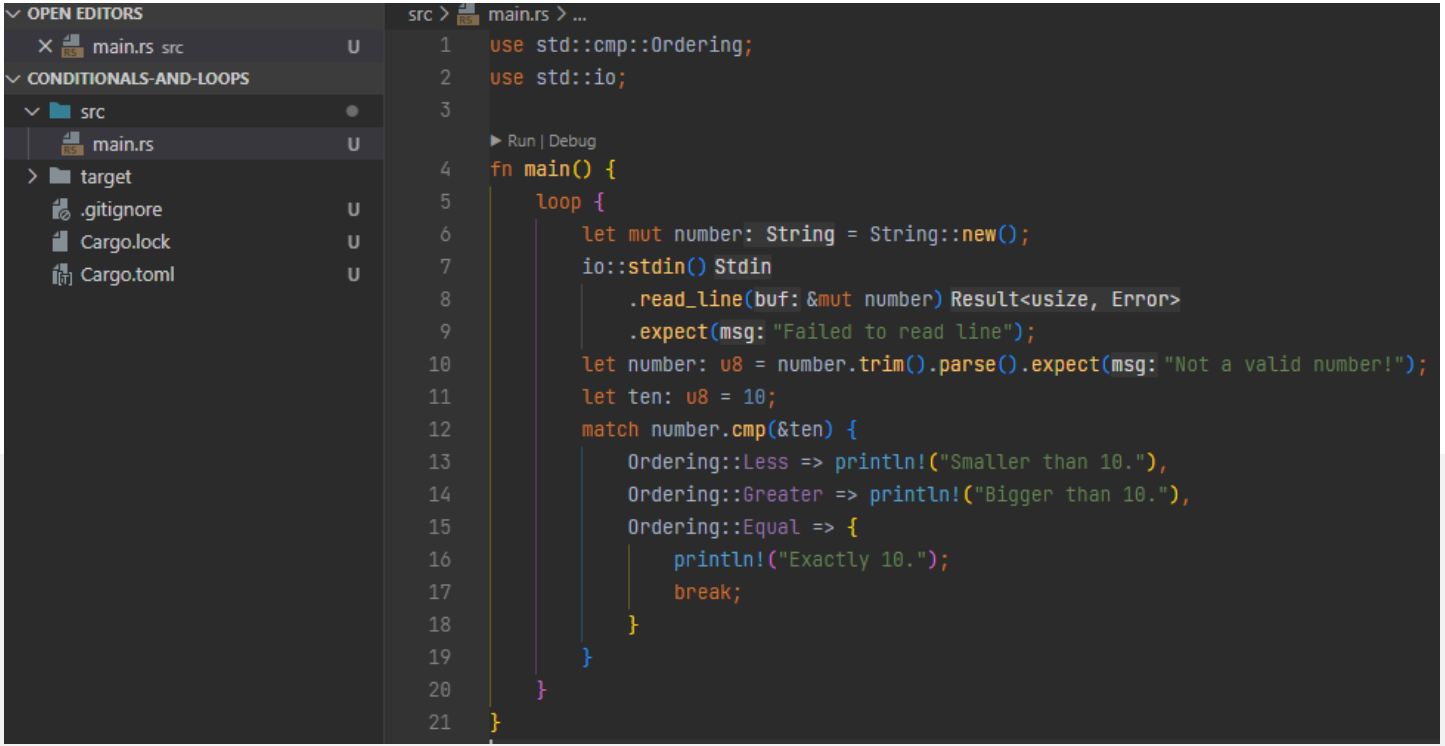
Loops

The most basic loop is literally **loop{}** . We can just surround our code with the brackets of the command loop and the program will continuously ask as for an input and tell us if it's bigger, smaller or equal to 10. If you try it you can see that the program never ends, so you will have to kill it manually by CTRL+C. We can manually end a loop by using the command **break;** Let's change the match expression on the Ordering::Equal line to:

```
Ordering::Equal => {
        println!("Exactly 10.");
        break;
}
```

Now when we enter the number 10 the program will print "Exactly 10." and then exit the loop, terminating it.

Our final program should look like this:



You are now ready to tackle the project!

Any questions?

You can find me at:

- vlasopoulos.v@gmail.com