Discussion 2

February 6, 2023

Slides: Neil Kaushikkar

Goals For Today

- MP0 Syntax/Control Flow Tips
- Work Time

Tuple Destructuring

- **pub fn get_equation_tuple(line: &String) -> (Option<&str>, Option<&str>)**
- When calling the function, store the result in a destructured tuple:
  - **let (left, right) = get_equation_tuple(...)**
- Both left and right are now **Option<&str>**
- PLEASE DO NOT USE .0, .1, etc to get tuple values
  - At the end of the day it just looks ugly and is harder to read

Use Operation::from_char to find the Operation

- Call the function with **Operation::from_char(c)** where c is some **char**
- Use a for loop to loop through the characters in the equation **String** and call **Operation::from_char(c)** on each character
  - You will get **None** if the current char is not an **Operation**
  - Stop the loop with break when you reach a valid **Operation**
  - What does **Operation::from_char(c)** return on a valid **Operation?**
  - Loop through all characters with: **for c in line.chars() { … }**

Parsing Strings to f64

- Many similar ways to parse Strings:
  - **let val = match s.parse::<f64>() { … }**
  - **let val: f64 = match s.parse() { … }**
  - If you specify a type for your variable, Rust can tell which version of **parse()** you want to use (i.e. the second example)
- https://doc.rust-lang.org/std/primitive.str.html#method.parse

Use a Nested Match Statement

- Each element of the tuple returned by **get_equation_tuple(line: &String)** is an Option<&str> and we want to parse the &str inside the Option
- First match the Option, then parse, then match the result of the parse
- ```
  let left_float: f64 = match left_option {
      Some(s) => match s.parse() {
          Ok(f) => f,
          Err(_) => return … // handle this error case
      },
      None => return … // handle this error case
  }
  ```
  ```

Q&A / Time to Collaborate