Structs

Goals For Today

- Learn what a struct is
- Learn how to define and instantiate a struct
- Learn how to use a struct

What is a struct?

- Structs are custom made data types that hold multiple values
- Similar to tuples, structs had hold data of multiple types without issue
  - Unlike tuples, you must declare the type of data that you are storing
- An Object of type struct is called an instance of that struct
- Structs are functionally similar to classes
  - Instances do not share variables
  - Can declare local functions using traits or as methods with impl
  - Declared prior to the methods they are used in

```
struct User {
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64,
}
```

How do I create a struct?


```
struct User {
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64,
}
```

```
fn main() {
    let mut user1 = User {
        email: String::from("someone@example.com"),
        username: String::from("someusername123"),
        active: true,
        sign_in_count: 1,
    };

    user1.email = String::from("anotheremail@example.com");
}
```

How do I declare a struct?

- To get or change a specific value in an instance, we use dot notation
- Notice that the entire instance needs to be mutable to edit specific values
  - You CANNOT select certain values to be immutable and others to be mutable
- Structs can be returned by functions (remember ownership rules!)
  - Notice that we use String instead of &str. &str implies data is borrowed but we want the struct instance to own its data!

```
fn build_user(email: String, username: String) -> User {
    User {
        email: email,
        username: username,
        active: true,
        sign_in_count: 1,
    }
}
```

Works because function transfers ownership of User upon return

```
fn build_user(email: String, username: String) -> User {
    User {
        email,
        username,
        active: true,
        sign_in_count: 1,
    }
}
```

Uses a concept called Field Init which is possible because the fields inside the struct are the same name as the variables they are being initialized to

Struct Update Syntax

- Say we want to create a second struct that has all the fields same as the first except for one field
- Is there a more efficient way to do it instead of retyping the entire struct?

```
fn main() {
    // --snip--

    let user2 = User {
        active: user1.active,
        username: user1.username,
        email: String::from("another@example.com"),
        sign_in_count: user1.sign_in_count,
    };
}
```

```
fn main() {
    // --snip--

    let user2 = User {
        email: String::from("another@example.com"),
        ..user1
    };
}
```

Struct Methods

- Say we want to add functions to our structs
- How do we go about making a struct owned function?
  - Key word impl
  - Indicates the implementation of a function for a struct, can have multiple impl
  - Function accepts a borrowed instance to access data in the instance
  - All Methods in impl require you to pass &self irrespective of use

```
#[derive(Debug)]
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}
```

Associated functions

- The only exception to needing to pass &self as a parameter are associated functions
- These are functions not called by the dot notation but with :: instead
- Typically used for constructors are demonstrated below
- Self is an alias for Rectangle in this case, rather than an instance

```
impl Rectangle {
    fn square(size: u32) -> Self {
        Self {
            width: size,
            height: size,
        }
    }
}
```

Struct Comparison

- Say I want to compare two instances of Structs
- Can I use == as a comparator? Why or why not?
- How does one implement the == operator on an object? Is == a function?
- How about !=?
- What is comparison for Strings or other non integer objects?

Struct Comparison

- == is in fact a function call. It is an implicit function call of function eq()
- != is also a function call of function ne()
- You can implement comparison for your Structs using impl
- You need to place #[derive(PartialEq)] at the top of your file
    - impl PartialEq for <struct_name>
    - declare functions for eq and/or ne and use them as comparators
    - Notice we return a bool and that our second parameter is passed by reference

```
enum BookFormat {
    Paperback,
    Hardback,
    Ebook,
}

struct Book {
    isbn: i32,
    format: BookFormat,
}

impl PartialEq for Book {
    fn eq(&self, other: &Self) -> bool {
        self.isbn == other.isbn
    }
}
```

Reminders:

Chapter 5 in the rust docs: Using Structs to Structure Related Data - The Rust Programming Language (rust-lang.org)

HW 6 has been released

MP1 is due next week