3_Slices_en

May 2, 2023

1 The slice type

Slices let you reference a contiguous sequence of elements in a collection rather than the whole collection.

String literals are slices.

[ ]:
```
let s = "Hello, world!";
```

The type of s here is **`&str`**: it's a slice pointing to that specific point of the binary. This is also why string literals are immutable; **`&str`** is an immutable reference.

1.1 String slices

A string slice is a reference to part of a String, and it looks like this:

[ ]:
```
{
    let s = String::from("hello world");
    let hello = &s[0..5];
    let world = &s[6..11];
}
```

With Rust's **..** range syntax, if you want to start at the first index (zero), you can drop the value before the two periods. In other words, these are equal:

[ ]:
```
{
    let s = String::from("hello");
    let slice = &s[0..2];
    let slice = &s[..2];
}
```

By the same token, if your slice includes the last byte of the String, you can drop the trailing number. That means these are equal:

[ ]:
```
{
    let s = String::from("hello");

    let len = s.len();

    let slice = &s[3..len];
```

```
    let slice = &s[3..];
}
```

2 Exercises

Exercise 1: The program bellow defines a function that takes a string and returns the first word it finds in that string. If the function doesn't find a space in the string, the whole string must be one word, so the entire string should be returned. Write the missing lines in the following code:

[ ]:
```
fn first_word(s: &String) -> &str {
    let bytes = s.as_bytes();

    for (i, &item) in bytes.iter().enumerate() { // iterate on the String's␣
 ↪bytes
        if item == b' ' { //check if the byte is equal to blank space, which␣
 ↪means we've got a word
            // TO DO (1) Write the statement that returns the first word
            return
        }
    }
    // TO DO (2) Write the statement that returns the entire string

}

fn main() {
    let mut s = String::from("hello world");

    let word = first_word(&s);

    //TO DO (3) uncomment the following line and try to explain what happens
    // s.clear();

    println!("the first word is: {}", word);
}
main();
```

Solution:

[ ]:
```
fn first_word(s: &String) -> &str {
    let bytes = s.as_bytes(); // convert our String to an array of bytes

    for (i, &item) in bytes.iter().enumerate() { // iterate on the String's␣
 ↪bytes
        if item == b' ' { //check if the byte is equal to blanc space, which␣
 ↪means we've got a word
            // TO DO (1) Write the statement that returns the first word
            return &s[0..i];
```

```
        }
    }
    // TO DO (2) Write the statement that returns the entire string
    &s[..]
}

fn main() {
    let mut s = String::from("hello world");

    let word = first_word(&s);

    //TO DO (3) uncomment the following line and try to explain what happens
    // s.clear(); // this empties the String, making it equal to ""

    println!("the first word is: {}", word);
}
main();
```

The error in (3): if we have an immutable reference to something, we cannot also take a mutable reference. Because `clear` needs to truncate the String, it tries to take a mutable reference, which fails.

Exercise 2: Re-write the code for the fizz buzz game using the "if in a let" syntax and using a single println!

[ ]:
```
fn main() {
}
main();
```

Solution:

[ ]:
```
fn main() {
    for number in 1..20 {
        let value: String;
        let result = if number % 3 == 0 {
            if number % 5 == 0 {"Fizz Buzz"} else { "Fizz"}
        } else if number % 5 == 0 {
            "Buzz"
        } else {
            value = number.to_string();
            &*value
        };
        println!("Result {}", result);
    }
}
main();
```