

Slice Type

Slice type

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

It is also a reference so there is no ownership, no moving.

Let's start with a string slice that reference to a part of a String:

```
let s = String::from("hello world");

let hello = &s[0..5];
let world = &s[6..11];
```

To create a slice you specify the collection you are creating the slice reference from along with the `&` to indicate that it is an reference. Then you provide the range `[starting_index..ending_index]`, `starting_index` is the first position in the slice and `ending_index` is the ending element index, excluding that element.

Why slice?

The reason why we are using slice because say we are finding a particular index in a String, after finding that index number, that String suddenly changed, but you still used the old index number that isn't valid anymore, this will result in a code bug. However, using slice type, your compiler will ensure that the index number or particular information you retrieve about that String remains valid before it is changed.

```
let mut s = String::from("hello world");

let word = first_word(&s); // get the reference to "hello" in s

s.clear(); // error, because we are using mutable reference here, but there is a immutable
reference existing!

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

Thus using slice type will prevent any mutable changes from happening before the slice type gets used.

String literals

```
let s = "hello world";
```

The type of `s` here is `&str`, a slice that points to that string literal in binary. It is also immutable because there is no `mut` modifier.

Using string slice as parameters

By making the parameter of a string from

```
fn first_word(s: &String) -> &str
```

Into to

```
fn first_word(s: &str) -> &str
```

We are able to take slices of `String` whether partial or whole, or on the entire reference, because slice type are reference themselves.

We can also slice type string literal, because string literal themselves are also string slices you can also pass them in directly, or you can slice them too.

```
fn main() {
    let my_string = String::from("hello world");

    // `first_word` works on slices of `String`s, whether partial or whole
    let word = first_word(&my_string[0..6]);
    let word = first_word(&my_string[..]);
    // `first_word` also works on references to `String`s, which are equivalent
    // to whole slices of `String`s
    let word = first_word(&my_string);

    let my_string_literal = "hello world";

    // `first_word` works on slices of string literals, whether partial or whole
    let word = first_word(&my_string_literal[0..6]);
    let word = first_word(&my_string_literal[..]);
```

```
// Because string literals *are* string slices already,  
// this works too, without the slice syntax!  
let word = first_word(my_string_literal);  
}
```

Revision #1

Created 28 January 2023 17:47:18 by Tamarine

Updated 28 January 2023 19:24:31 by Tamarine