한 번에 끝내는 블록체인 개발 A to Z

Chapter 1

Rust Introduction

Chapter 2 Rust Introduction

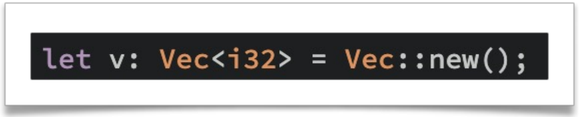
Collections

Vector

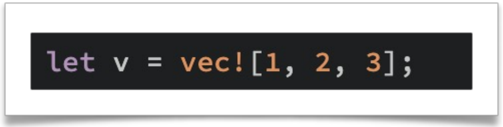
Vector

Vectors allow you to store more than one value in a single data structure that puts all the values next to each other in memory.

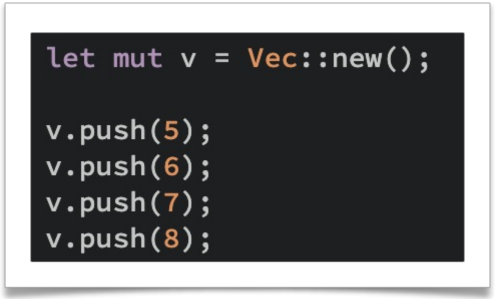
Vectors are implemented using generics.



```rust
let v: Vec<i32> = Vec::new();
```



```rust
let v = vec![1, 2, 3];
```

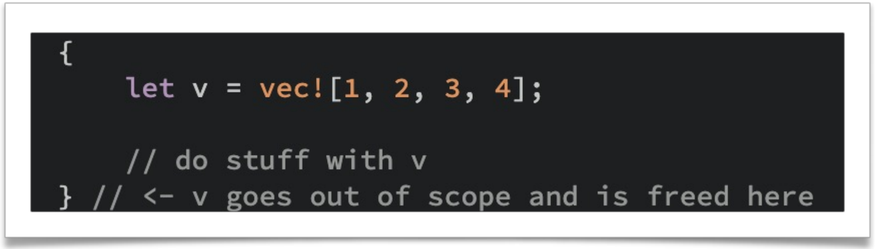


```rust
let mut v = Vec::new();

v.push(5);
v.push(6);
v.push(7);
v.push(8);
```

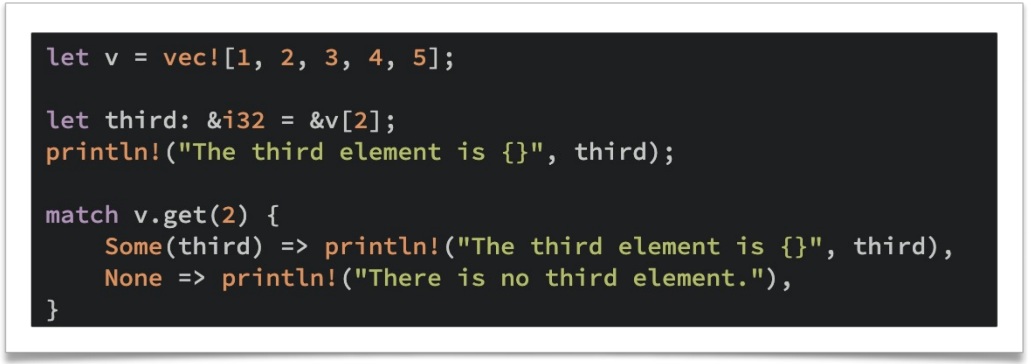
Vector

Like any other struct, a vector is freed when it goes out of scope.

There are two ways to reference a value stored in a vector: via indexing or using
the get method. When the get method is passed an index that is outside the vector, it returns None without panicking.



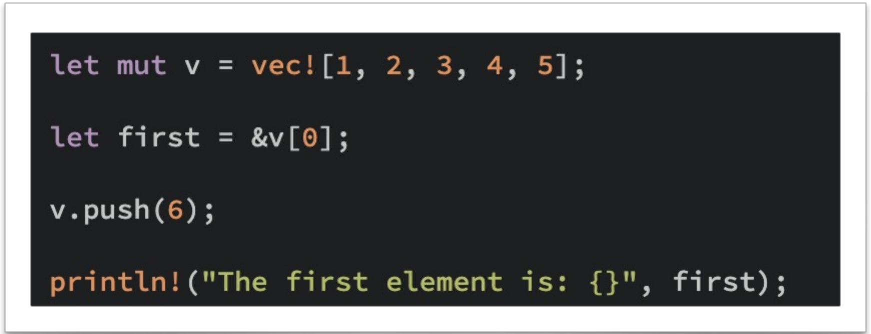
```
{
    let v = vec![1, 2, 3, 4];

    // do stuff with v
} // <- v goes out of scope and is freed here
```



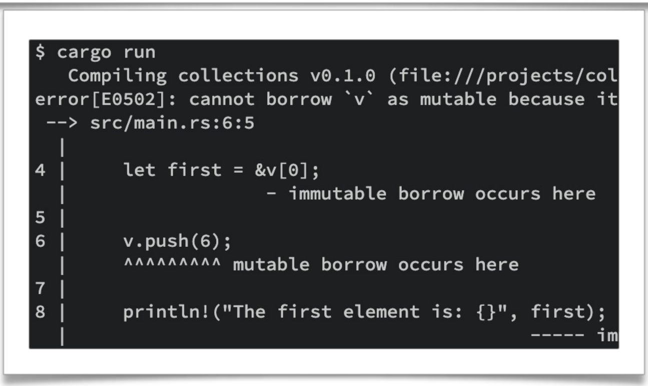
```
let v = vec![1, 2, 3, 4, 5];

let third: &i32 = &v[2];
println!("The third element is {}", third);

match v.get(2) {
    Some(third) => println!("The third element is {}", third),
    None => println!("There is no third element."),
}
```

Vector

When the program has a valid reference, the borrow checker enforces the ownership and borrowing rules to ensure this reference and any other references to the contents of the vector remain valid. Recall the rule that states you can’t have mutable and immutable references in the same scope.



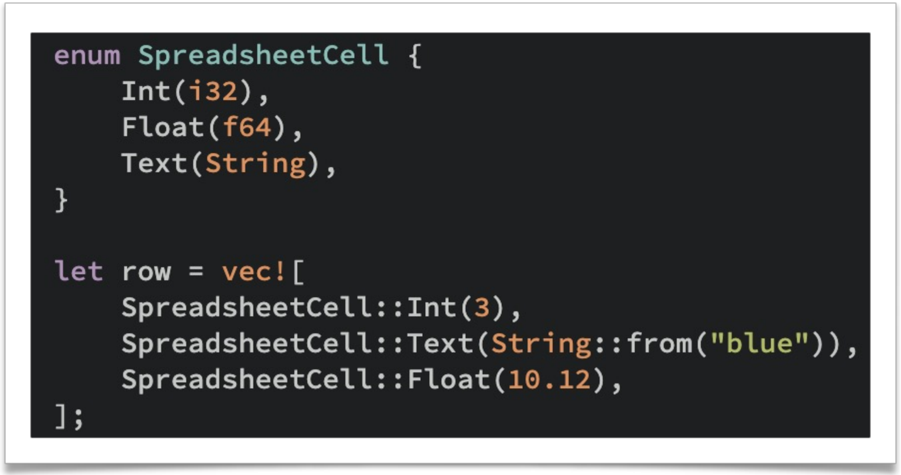
```rust
let mut v = vec![1, 2, 3, 4, 5];

let first = &v[0];

v.push(6);

println!("The first element is: {}", first);
```



```
$ cargo run
   Compiling collections v0.1.0 (file:///projects/col
error[E0502]: cannot borrow `v` as mutable because it
 --> src/main.rs:6:5
  |
4 |     let first = &v[0];
  |                  - immutable borrow occurs here
5 |
6 |     v.push(6);
  |     ^^^^^^^^^ mutable borrow occurs here
7 |
8 |     println!("The first element is: {}", first);
  |                                          ----- im
```

Using an Enum

Vectors can only store values that are the same type. This can be inconvenient; there are definitely use cases for needing to store a list of items of different types. Fortunately, the variants of an enum are defined under the same enum type, so when we need one type to represent elements of different types, we can define and use an enum!

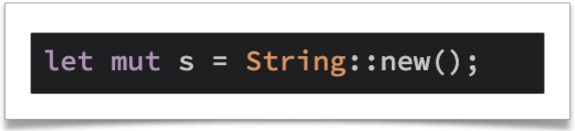


```rust
enum SpreadsheetCell {
    Int(i32),
    Float(f64),
    Text(String),
}

let row = vec![
    SpreadsheetCell::Int(3),
    SpreadsheetCell::Text(String::from("blue")),
    SpreadsheetCell::Float(10.12),
];
```

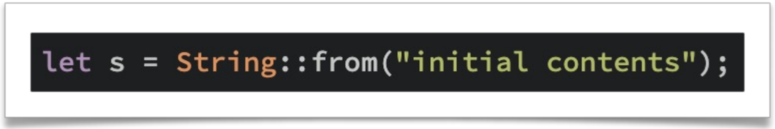
String

Create String

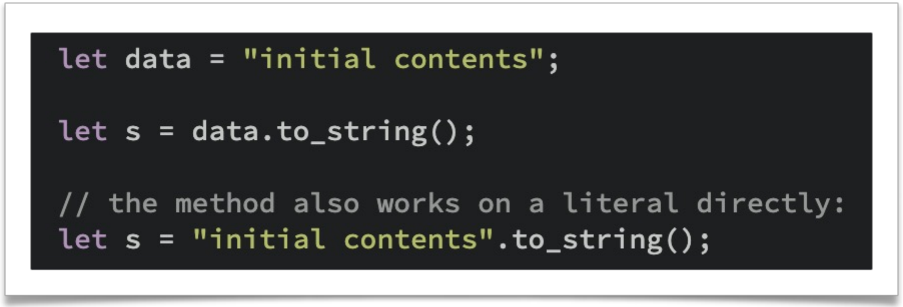
Many of the same operations available
with Vec<T> are available with String as well.



```rust
let mut s = String::new();
```

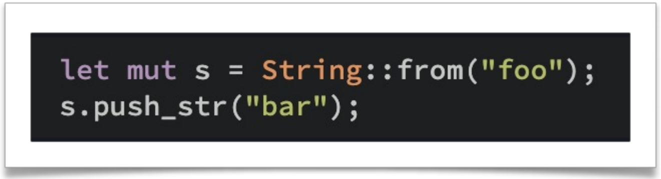


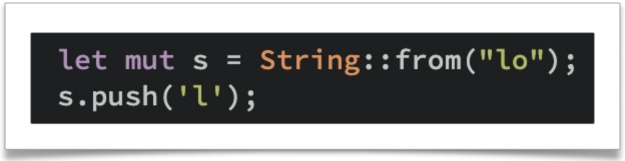
```rust
let s = String::from("initial contents");
```

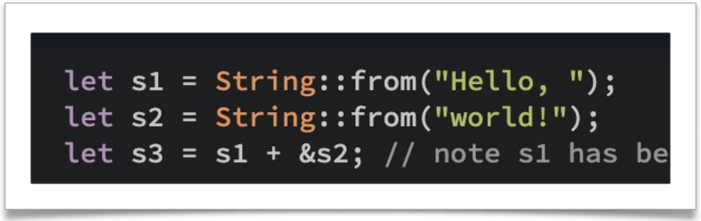


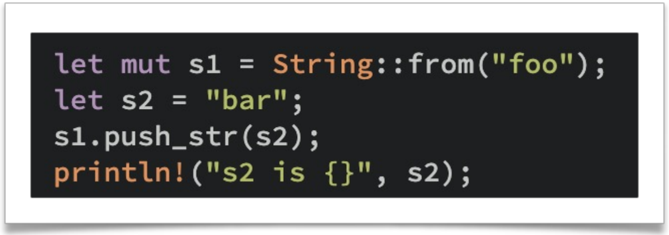
```rust
let data = "initial contents";

let s = data.to_string();

// the method also works on a literal directly:
let s = "initial contents".to_string();
```

Update


```rust
let mut s = String::from("foo");
s.push_str("bar");
```

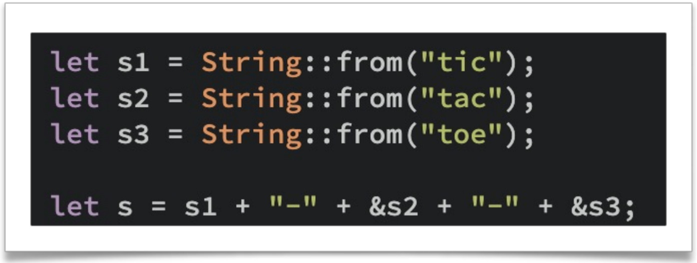

```rust
let mut s = String::from("lo");
s.push('l');
```


```rust
let s1 = String::from("Hello, ");
let s2 = String::from("world!");
let s3 = s1 + &s2; // note s1 has be
```

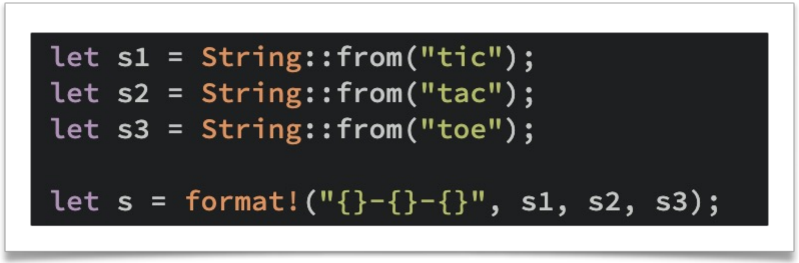

```rust
let mut s1 = String::from("foo");
let s2 = "bar";
s1.push_str(s2);
println!("s2 is {}", s2);
```

After these two lines, s will contain foobar.
The push_str method takes a string slice because we don't necessarily want to take ownership of the parameter. For example, in the code in Listing 8-16, we want to be able to use s2 after appending its contents to s1.

Formatting

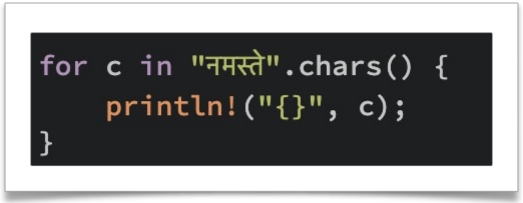


```rust
let s1 = String::from("tic");
let s2 = String::from("tac");
let s3 = String::from("toe");

let s = s1 + "-" + &s2 + "-" + &s3;
```

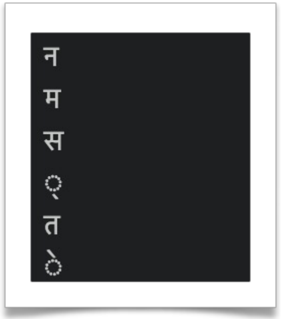


```rust
let s1 = String::from("tic");
let s2 = String::from("tac");
let s3 = String::from("toe");

let s = format!("{}-{}-{}", s1, s2, s3);
```

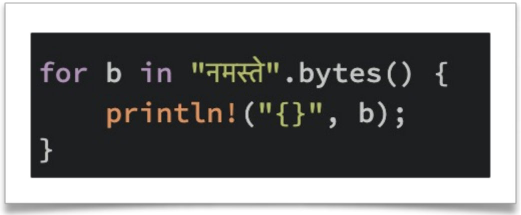
Iterating



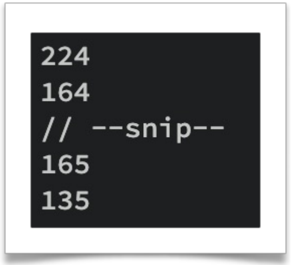
```rust
for c in "नमस्ते".chars() {
    println!("{}", c);
}
```



```
न
म
स
्
त
े
```



```rust
for b in "नमस्ते".bytes() {
    println!("{}", b);
}
```



```
224
164
// --snip--
165
135
```