Intro to Rust

Carol (Nichols || Goulding)
@carols10cents

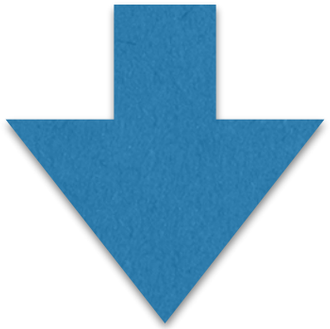
Agenda

- Guessing game - max 1 hr
- Ownership - min 40 min
- Announcements - 5 min

Setup

- rust somewhere around 1.10.0
- cargo (should come with rust)
- terminal
- your favorite text editor
- internet

```
$ cd ~/wherever-you-want
$ cargo new guessing_game --bin
$ cd guessing_game
$ cargo run
```



```
 Compiling guessing_game v0.1.0
   (file:///wherever-you-want/guessing_game)
     Running `target/debug/guessing_game`
Hello, world!
```

Guessing Game

Demo

Generate

- Random number

Input

- Guesses from the user

Output

- Correct
- Secret number is higher
- Secret number is lower

Let’s code!

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```


MATH IS HARD
LET'S GO SHOPPING!
quickmeme.com

https://crates.io

Search

CARGO

Search

Browse All Crates | Docs ▾ | Log in with GitHub

The Rust community's crate host

Install Cargo

Getting Started

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

56,629,401 Downloads

5,498 Crates in stock

New Crates

- quantum (0.1.0)
- appdirs (0.1.0)

Most Downloaded

- libc (0.2.14)
- winapi (0.2.8)

Just Updated

- json (0.10.0)
- engine-io (0.1.2)

What you learned

- Output values
- Input from stdin
- Var bindings
- Mutability
- Using a crate
- match
- Result
- parse
- loop
- break

Ownership

Memory management

Java

Java

C

C

C

Common memory errors

- Use after free
- Double free
- Null pointers
- Memory leak

string literal:

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

`String`:

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

```
fn main() {

}
```

```rust
fn main() {

    let s1 = String::from(
        "hello"
    );

}
```

```rust
println!("s1 is {}", s1);
fn main() {
    println!("s1 is {}", s1);
    let s1 = String::from(
        "hello"
    );
    println!("s1 is {}", s1);
}
println!("s1 is {}", s1);
```

```rust
fn main() {

    let s1 = String::from(
        "hello"
    );
    println!("s1 is {}", s1);
}
```

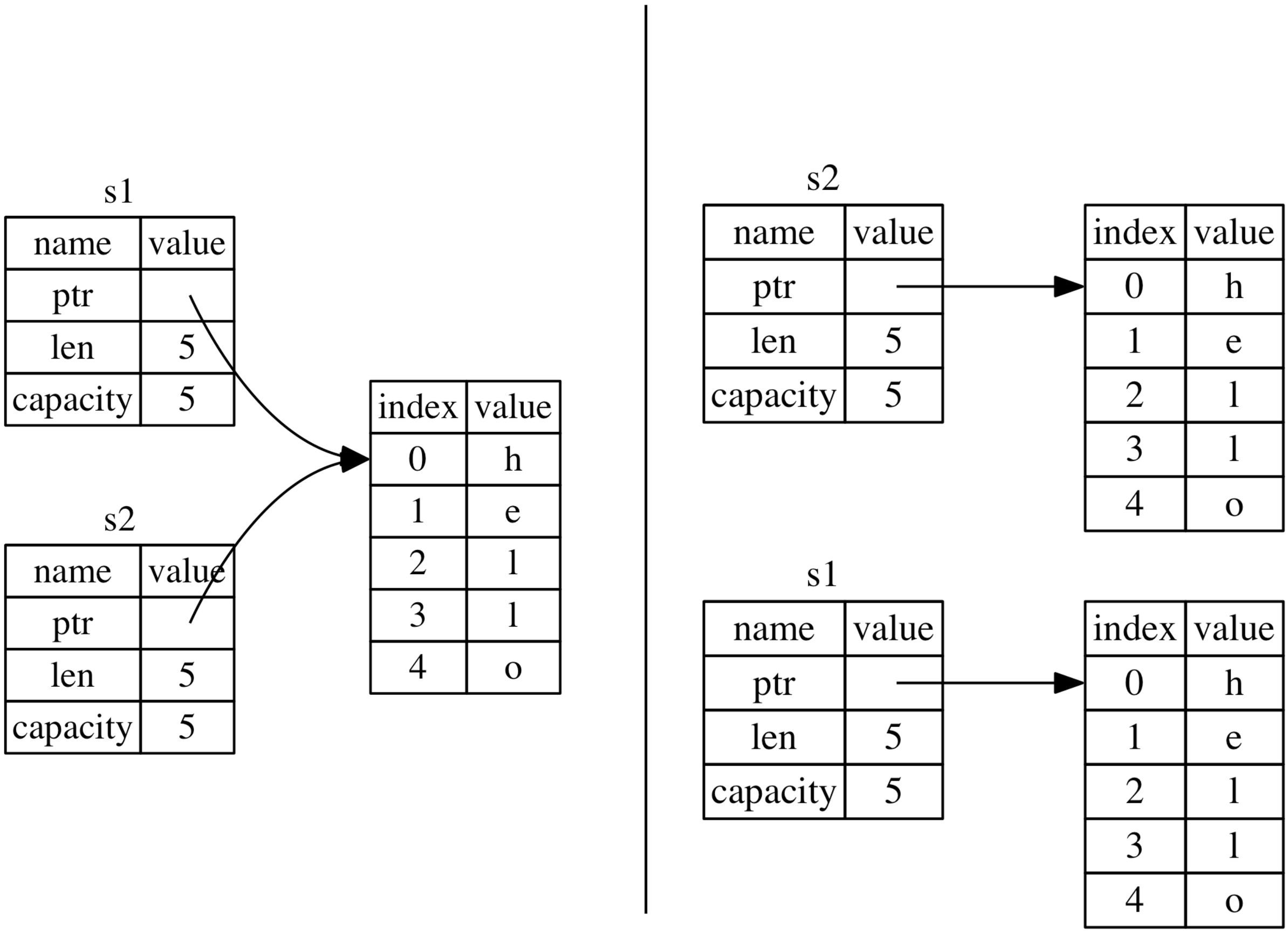
```
fn main() {

    let s1 = String::from(
        "hello"
    );
    println!("s1 is {}", s1);
}
```

s1

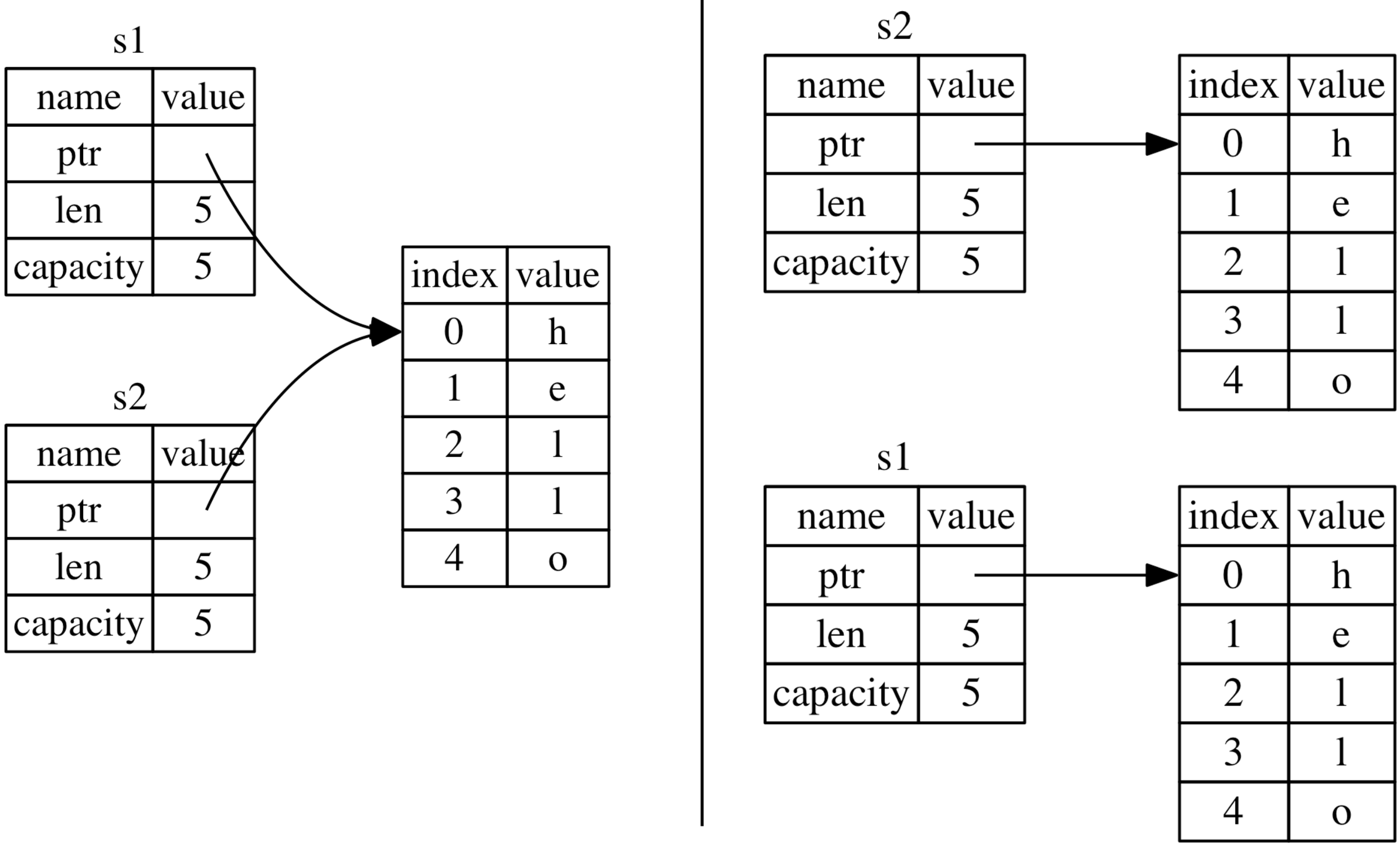
name	value
ptr	
len	5
capacity	5

index	value
0	h
1	e
2	l
3	l
4	o

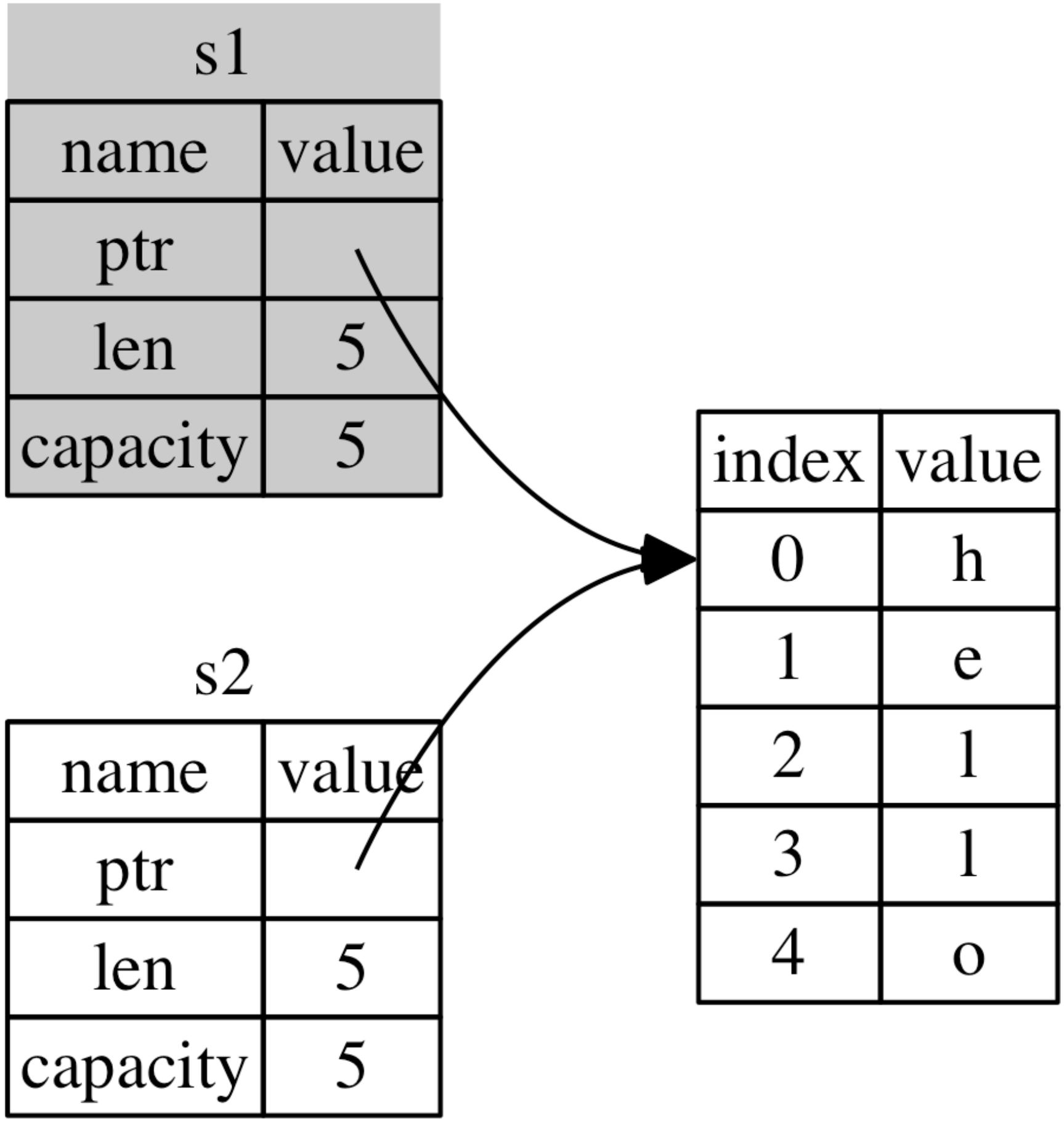
```rust
fn main() {
    let s1 = String::from(
        "hello"
    );

    let s2 = s1;



}
```


s1
name
value
ptr
len
5
capacity
5
s2
name
value
ptr
len
5
capacity
5
index
value
0
h
1
e
2
l
3
l
4
o
s2
name
value
ptr
len
5
capacity
5
index
value
0
h
1
e
2
l
3
l
4
o
s1
name
value
ptr
len
5
capacity
5
index
value
0
h
1
e
2
l
3
l
4
o

Inefficient :(



```
fn main() {
    let s1 = String::from(
        "hello"
    );

    let s2 = s1;


}
```


s1
name
value
ptr
len
5
capacity
5
s2
name
value
ptr
len
5
capacity
5
index
value
0
h
1
e
2
l
3
l
4
o

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );

    let s2 = s1;

    println!("s1 is {}", s1);
```

```
                                  ^^
error: use of moved value: `s1`
```

```rust
}
```

The Ownership Rules

1. Each value in Rust has a variable binding that's called its 'owner'.

2. There can only be one owner at a time.

3. When the owner goes out of scope, the value will be `drop()`ped.

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );
    something(s1);

}

fn something(s: String) {
    println!("i got {}", s);
}
```

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );
    something(s1);

}

fn something(s: String) {
    println!("i got {}", s);
}
```

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );
    something(s1);
    something(s1);
}

fn something(s: String) {
    println!("i got {}", s);
}
```

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );
    something(s1);
    something(s1);
}
```

```
                    ^^
error: use of moved value: `s1`
```

```rust
fn something(s: String) {
    println!("i got {}", s);
}
```

```
fn main() {
    let s1 = String::from(
        "hello"
    );
    something(s1);
    something(s1);
}

fn something(s: String) {
    println!("i got {}", s);
}
```

```
                ^^
error: use of moved value: `s1`
```

SO ANNOYING

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );
    let s1 = something(s1);
    something(s1);
  }

fn something(s: String)->String {
    println!("i got {}", s);
    s
}
```

```
fn main() {
    let s1 = String::from(
        "hello"
    );
    let s1 = something(s1);
    something(s1);
}

fn something(s: String) -> String {
    println!("i got {}", s);
    s
}
```

ALSO

ANNOYING

Borrowing to the rescue!

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );
    something(&s1);
    something(&s1);
}

fn something(s: &String) {
    println!("i got {}", s);
}
```

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );
    something(&s1);
}

fn something(s: &String) {
    s.push_str(", world!");
}
```

```rust
fn main() {
    let s1 = String::from(
        "hello"
    );
    something(&s1);
}

fn something(s: &String) {
    s.push_str(", world!");
}
```

```
    ^^
 error: cannot borrow immutable
borrowed content `*s` as mutable
```

```rust
fn main() {
    let mut s1 = String::from(
        "hello"
    );
    something(&mut s1);
}

fn something(s: &mut String) {
    s.push_str(", world!");
}
```

```rust
fn main() {
    let mut s1 = String::from(
        "hello"
    );
    let r1 = &mut s1;
    let r2 = &mut s1;
}
```

```rust
fn main() {
    let mut s1 = String::from(
        "hello"
    );
    let r1 = &mut s1;
    let r2 = &mut s1;
}
```

```
error: cannot borrow `s1` as
mutable more than once at a time
```

```rust
fn main() {
    let mut s1 = String::from(
        "hello"
    );
    let r1 = &s1;
    let r2 = &s1;
}
```

```rust
fn main() {
    let mut s1 = String::from(
        "hello"
    );
    let r1 = &s1;
    let r2 = &s1;
    let r3 = &mut s1;
}
```

```rust
let mut list = vec![1, 2, 3];
for i in &list {
    println!("i is {}", i);
    list.push(i + 1);
}
```

```
let mut list = vec![1, 2, 3];
for i in &list {
    println!("i is {}", i);
    list.push(i + 1);
}
```

```
^ error: cannot borrow `list`
as mutable because it is also
borrowed as immutable
```

Rules of References

1. At any given time, you may have either, but not both of:
    1. One mutable reference.
    2. Any number of immutable references.
2. References must always be valid.

I LIED TO YOUUUU!

&String isn't idiomatic Rust.

Slices

- Also don't have ownership.
- Reference a contiguous sequence of elements in a collection: string slices, vec slices, etc.

&str

Function to get the 1st word

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

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

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

    for (i, &byte) in bytes.iter().enumerate() {
        if byte == 32 {
            return i;
        }
    }

    s.len()

}
```

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

let end_of_first_word = first_word(&s);

s.clear();

// end_of_first_word is still 5 here…
```

Good news!

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

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

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

    for (i, &byte) in bytes.iter().enumerate() {
        if byte == 32 {
            return &s[0..i];
        }
    }

    &s[..]
}
```

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

let first_word = first_word(&s);

s.clear();
```

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

let first_word = first_word(&s);

s.clear();
```

```
^ error: cannot borrow `s` as
mutable because it is also
borrowed as immutable
```

string literal:

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

`String`:

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

`&str`:

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

`String`:

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

```rust
fn something(s: &String) {
    println!("i got {}", s);
}
```

```rust
fn something(s: &str) {
    println!("i got {}", s);
}
```

```rust
let s = "hello";
let t = String::from("hi");
something(s);
something(&t[..]);
```

Further reading

- https://carols10cents.github.io/trpl/
  - We did chapters 2 & 3 today
  - Give me feedback!
- https://doc.rust-lang.org/stable/book/
- http://rust-lang.github.io/book/