

# Ownership in Rust

## Why Learn another language?

- Not all languages stay popular forever
  - There may be interesting work in other languages
  - You can apply the patterns of another language to your current language
- 

## Why Rust?

- Memory safe
  - Fast
  - Correctness as a feature
  - Statically Linked Binaries
  - Strong type system
  - Functional programming
  - Fearless Concurrency
  - **Unique ownership system**
- 

## Memory Safety

- Does this compile?
- what does it print?

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    vector<tuple<int, int>> vec = {{5, 10}};
    const auto &first_elem = vec.front();
    for (int i = 0; i < 10; ++i) vec.push_back(first_elem);
    for (const auto [x, y]: vec) cout << x << ' ' << y << '\n';
}
```

---

## Memory Safety Cont.

It does, rather confusingly:

```
5 10
5 10
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
```



## Iterator Invalidation

- Does this compile?
- what does it print?

```
#include <bits/stdc++.h>
using namespace std;
```

```
int main() {
    vector<string> strings = {"Hi"};
    const auto& elem = strings[0];
    strings.push_back("World");
    cout << elem << '\n';
}
```

---

### Iterator Invalidation Cont.

- This is undefined behavior. It could print nothing or segfault or uninitialized memory.



Let's try this in rust.

---

### Rust Memory Safety

- Does this compile?
- If so, what does it print?

```
fn main() {
    let mut vec = vec![(5, 10)];
    let first_elem = &vec[0];
}
```

```

    for _ in 1..10 { vec.push(*first_elem); }
    for p in vec { println!("{}", p.0, p.1); }
}

```

---

```

error[E0502]: cannot borrow `vec` as mutable because
it is also borrowed as immutable
--> src/main.rs:4:22

```

```

|
3 |     let first_elem = &vec[0];
|                       --- immutable borrow occurs here
4 |     for _ in 1..10 { vec.push(*first_elem); }
|                               ^
|                               |
|                               | immutable borrow
|                               | later used here
|                               | mutable borrow occurs here

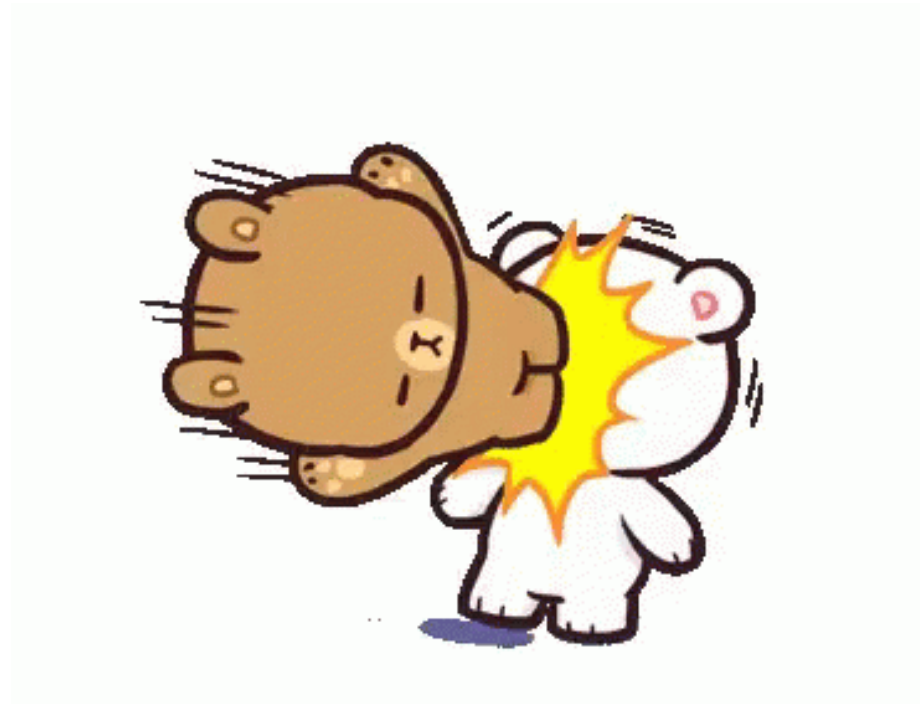
```

```

error: aborting due to previous error

```

---



## Strings

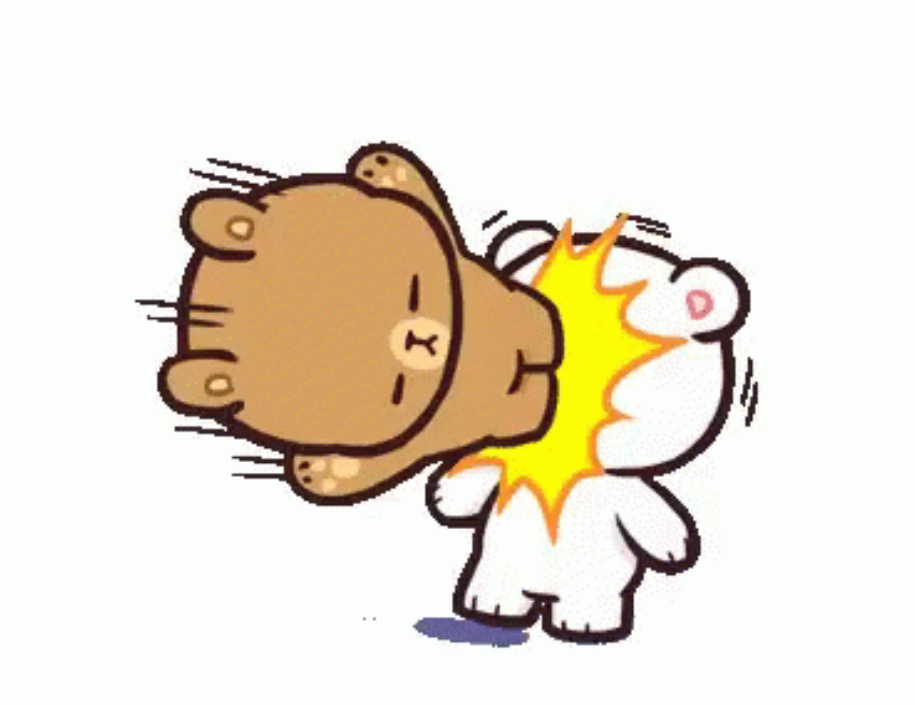
- Does this compile?
- If so, what does it print?

```
fn main() {  
    let mut vec = vec![String::from("Hi")];  
    let first_elem = &vec[0];  
    vec.push(first_elem.to_string());  
    println!("{}", first_elem);  
}
```

---

```
error[E0502]: cannot borrow `vec` as mutable because  
it is also borrowed as immutable  
--> src/main.rs:4:5  
   |  
3 |     let first_elem = &vec[0];  
   |                      --- immutable borrow occurs here  
4 |     vec.push(first_elem.to_string());  
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ mutable borrow occurs here  
5 |     println!("{}", first_elem);  
   |                      ----- immutable borrow later used here
```

---



---

### Ownership Rules

- You can give away as many copies as you want to.
- You are allowed to have as many immutable references as you want **or**
- You can loan out one mutable reference.

---

### Define Ownership

- A copy gives ownership of a copy to someone.
  - I give you a copy of my book
- A move gives ownership to someone.
  - I give you my book
- A reference shares ownership of code with someone.
  - We share the book

---

### Is this program ok?

```
#include <bits/stdc++.h>
using namespace std;
```

```
void print_arr(const vector<int> &vec);

int main() {
    print_arr({1,2,3,4});
}
```

---

Does it look fine now?

```
#include <bits/stdc++.h>
using namespace std;

void print_arr(const vector<int> &vec) {
    for (const auto elem: vec) cout << elem << '\n';
    delete &vec;
}

int main() {
    print_arr({1,2,3,4});
}
```

---

Types can't save us

- We have a problem where we try to delete a vector through a reference. This compiles.
  - Our program crashes immediately.
- 

In Rust

```
fn print_vec(vec: &Vec<i32>) {
    for item in vec {
        println!("{}", item);
    }
    drop(*vec);
}

fn main() {
    print_vec(&vec![1,2,3,4]);
}
```

---

```

error[E0507]: cannot move out of `*vec` which is behind
a shared reference
--> src/main.rs:5:10
5 |         drop(*vec);
  |         ^^^^^ move occurs because `*vec` has type
  |               `std::vec::Vec<i32>`,
  |               which does not implement the `Copy` trait

```

---

## Fearless Concurrency

```

#include <bits/stdc++.h>
using namespace std;

vector<int> nums;
void writeToNums() {
    for (int i = 0; i < 3; i++)
        nums.push_back(i);
}

int main() {
    vector<thread> threads;
    for (int i = 0; i < 20; i++) threads.emplace_back(writeToNums);
    for (auto& th: threads) th.join();
    for (auto item: nums) cout << item << ' ';
}

```

---

Undefined behavior again.

---

## In Rust

```

use std::thread;
fn write_to_nums(nums: &mut Vec<i32>) {
    for i in 0..3 {
        nums.push(i);
    }
}

fn main() {
    let mut nums = vec![];
    let mut threads = vec![];
}

```



```

for _ in 0..20 {
    threads.push(thread::spawn(move || {
        write_to_nums(&mut nums);
    }));
}
for thread in threads {
    let _ = thread.join();
}
for num in nums {
    println!("{}", num);
}
}

```

---

```

error[E0382]: use of moved value: `nums`

```

```

--> src/main.rs:13:32

```

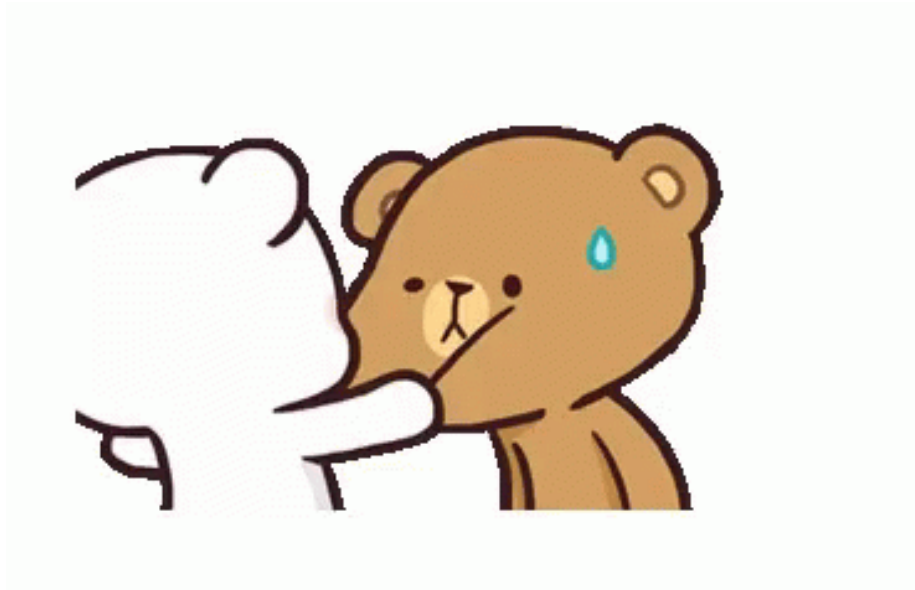
```

9 |   let mut nums = vec![];
  |   ----- move occurs because `nums` has type
  |   ----- `std::vec::Vec<i32>`, which does
  |   ----- not implement the `Copy` trait
...
13 |   threads.push(thread::spawn(move || {
  |                               ^^^^^^^ value moved into
  |                               closure here, in previous iteration of loop
14 |   write_to_nums(&mut nums);
  |               ^^ use occurs due to use in closure

```

---

The Borrow Checker at first



The Borrow Checker by the end

