Iterators | Implementing ***IntoIterator*** Using An Existing Collection

Iterator Trait

- By default requires mutable access to structure
  - Inconvenient
  - Not always possible
  - Mutation not always needed
- Solution:
  - Implement ***IntoIterator*** trait & call ***.iter()*** on inner collection
    - Vector, HashMap

***IntoIterator* Trait**

- Yields an ***Iterator*** (yield items/values)
  - Implementation details determine how items are accessed
    - Borrow, mutable, move

IntoIterator Trait

```rust
trait IntoIterator {
    type Item;
    type IntoIter;
    fn into_iter(self) -> Self::IntoIter;
}
```

Move

```rust
struct Friends {
    names: Vec<String>,
}

impl IntoIterator for Friends {
    type Item = String;
    type IntoIter = std::vec::IntoIter<Self::Item>;
    fn into_iter(self) -> Self::IntoIter {
        self.names.into_iter()
    }
}
```

```rust
struct Friends {
    names: Vec<String>,
}

impl IntoIterator for Friends {
    type Item = String;
    type IntoIter = std::vec::IntoIter<Self::Item>;
    fn into_iter(self) -> Self::IntoIter {
        self.names.into_iter()
    }
}

for f in friends {
    println!("{:?}", f);
}
```

Value Moved - Error!

```
for f in friends {
    println!("{:?}", f);
}

for f in friends {
    println!("{:?}", f);
}
```

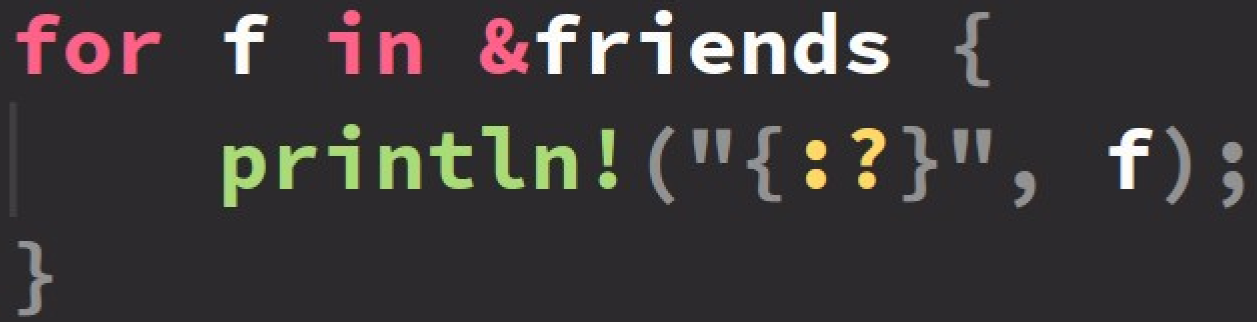
Error Details

```
for f in friends {
         -------
         |
         `friends` moved due to this implicit
          call to `.into_iter()`

for f in friends {
         ^^^^^^^ value used here after move
```

Borrow

```rust
struct Friends {
    names: Vec<String>,
}

impl<'a> IntoIterator for &'a Friends {
    type Item = &'a String;
    type IntoIter = std::slice::Iter<'a, String>;
    fn into_iter(self) -> Self::IntoIter {
        self.names.iter()
    }
}
```

Iteration



```
for f in &friends {
    println!("{:?}", f);
}
```

Mutable Borrow

```rust
struct Friends {
    names: Vec<String>,
}

impl<'a> IntoIterator for &'a mut Friends {
    type Item = &'a mut String;
    type IntoIter = std::slice::IterMut<'a, String>;
    fn into_iter(self) -> Self::IntoIter {
        self.names.iter_mut()
    }
}
```

Iteration

```rust
struct Friends {
    names: Vec<String>,
}
```

```rust
let names = vec![
    "Albert".to_owned(),
    "Sara".to_owned()
];
let mut friends = Friends{ names };

for f in &mut friends {
    *f = "Frank".to_string();
    println!("{:?}", f);
}
```

Iter Methods

- Convention for exposing iteration is to provide up to two methods:
  - *.iter()*
    - Iteration over borrowed values
  - *.iter_mut()*
    - Iteration over borrowed mutable values
- Implement these by simply calling ***into_iter()*** after implementing the ***IntoIterator*** trait
- These are optional, but allow for easy combinator usage without the ***for*** loop

Example

```rust
impl Friends {
    fn iter(&self) -> std::slice::Iter<'_, String> {
        self.into_iter()
    }
    fn iter_mut(&mut self) -> std::slice::IterMut<'_, String> {
        self.into_iter()
    }
}

let total = friends.iter().count();
```

Recap

- ***IntoIterator*** trait yields iterators
  - Allows control over borrows & mutability
- Implementation of ***IntoIterator*** requires:
  - An ***Item*** type - yielded value
  - An ***IntoIter*** type - mutable struct which tracks iteration progress / proxy to data structure
- The ***IntoIter*** type can be retrieved from the documentation on your inner collection