Iterators | Implementing ***Iterator***

Iterator

- Iteration is provided by the ***Iterator*** trait
  - Only one function to be implemented
  - Provides ***for..in*** syntax
  - Access to all iterator adapters
    - ***map, take, filter,*** etc
- Can be implemented for any structure

Iterator Trait

```rust
trait Iterator {
    type Item;
    fn next(&mut self) -> Option<Self::Item>;
}
```

Example - Iterator

```rust
impl Iterator for Odd {
    type Item = isize;

    fn next(&mut self) -> Option<Self::Item> {
        self.number += 2;
        if self.number <= self.max {
            Some(self.number)
        } else {
            None
        }
    }
}
```

```rust
struct Odd {
    number: isize,
    max: isize,
}
```

Example - Iterator

```rust
impl Odd {
    fn new(max: isize) -> Self {
        Self { number: -1, max }
    }
}

let mut odds = Odd::new(7);
println!("{:?}", odds.next());
println!("{:?}", odds.next());
println!("{:?}", odds.next());
println!("{:?}", odds.next());
println!("{:?}", odds.next());
```

```
Some(1)
Some(3)
Some(5)
Some(7)
None
```

Example - *for..in*

```rust
let mut odds = Odd::new(7);
for o in odds {
    println!("odd: {}", o);
}
```

```
odd: 1
odd: 3
odd: 5
odd: 7
```

Example – Adapters

```rust
let mut evens = Odd::new(8);
for e in evens.map(|odd| odd + 1) {
    println!("even: {}", e);
}
```

```
even: 2
even: 4
even: 6
even: 8
```

Recap

- Implementing ***Iterator*** provides access to ***for..in*** syntax and iterator adapters
    - Set the output type using the ***Item*** associated type as part of the ***Iterator*** trait
    - Return ***Some*** when data is available and ***None*** when there are no more items to iterate
- Data structure must:
    - Be mutable
    - Have a field to track iteration