



# RUST PROGRAMMING

## ITERATORS

SPOT ON

Kamal kumar mukiri,  
Apt Computing Labs



# Use of Iterators

---



## Problems with “loops”:

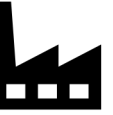
**Increased Risk of Errors:** Manual loops require careful handling of index bounds

**Lower Performance:** Without iterators, manual loops might not benefit as much from compiler optimizations

**Reduced Code Clarity:** Code written with manual loops can become verbose and harder to read, especially as complexity grows.

# How does Rust Compiler optimize “ITERATORS”?

---



## 1. Inlining of Iterator Methods



# How does Rust Compiler optimize “ITERATORS”?



## 2. Loop Fusion:

```
let result: Vec<_> = (1..100)
    .map(|x| x * 2)
    .filter(|x| x % 3 == 0)
    .collect();
```

The compiler can optimize this into a **single pass** over the range, performing both the **map and filter operations in the same loop**.

# How does Rust Compiler optimize “ITERATORS”?



## 3. Removal of Unused Iterations (Dead Code Elimination)

## 4. Avoiding Intermediate Allocations

```
let result: Vec<_> = (1..10)
    .map(|x| x + 1)
    .collect();
```

Instead of creating temporary lists, Rust collects elements directly into result, reducing allocation overhead.

# Iterator and Intolerator Traits:



```
1  ✓ trait Iterator {  
2      type Item;  
3  ✓  fn next(&mut self) -> Option<Self::Item> {  
4      // Login to return an item or None  
5      }  
6  }
```

**“Item” : Type of the value the iterator produces.**

**”next” : Either returns Some(v) or None**

# Writing own iterator:



```
1 struct Counter {
2     current: i32,
3     limit: i32
4 }
5
6 impl Counter {
7     fn new(a: i32) -> Self {
8         Counter {current:0, limit: a}
9     }
10 }
11
12 impl Iterator for Counter {
13     1 type Item = i32;
14
15     2 fn next(&mut self) -> Option<Self::Item>{
16         if self.current < self.limit {
17             self.current += 1;
18             return Some(self.current-1);
19         } else {
20             return None;
21         }
22     }
23 }
```

## Implement Iterator for Counter:

1. Specifies the type of items the iterator will yield.
2. Required method to implement.

Each call to next should:

- Return Some(value) while there are more items to iterate.
- Return None once the iterator has finished (when current  $\geq$  limit).

## Documenting:

All mark down features are supported.

More Options: <https://doc.rust-lang.org/cargo/commands/cargo-doc.html>



Thank you !!!