Folding Laundry

From zipping, filtering and folding – when your laundry gets Rusty

Mert Yildiz, 2024/09/25

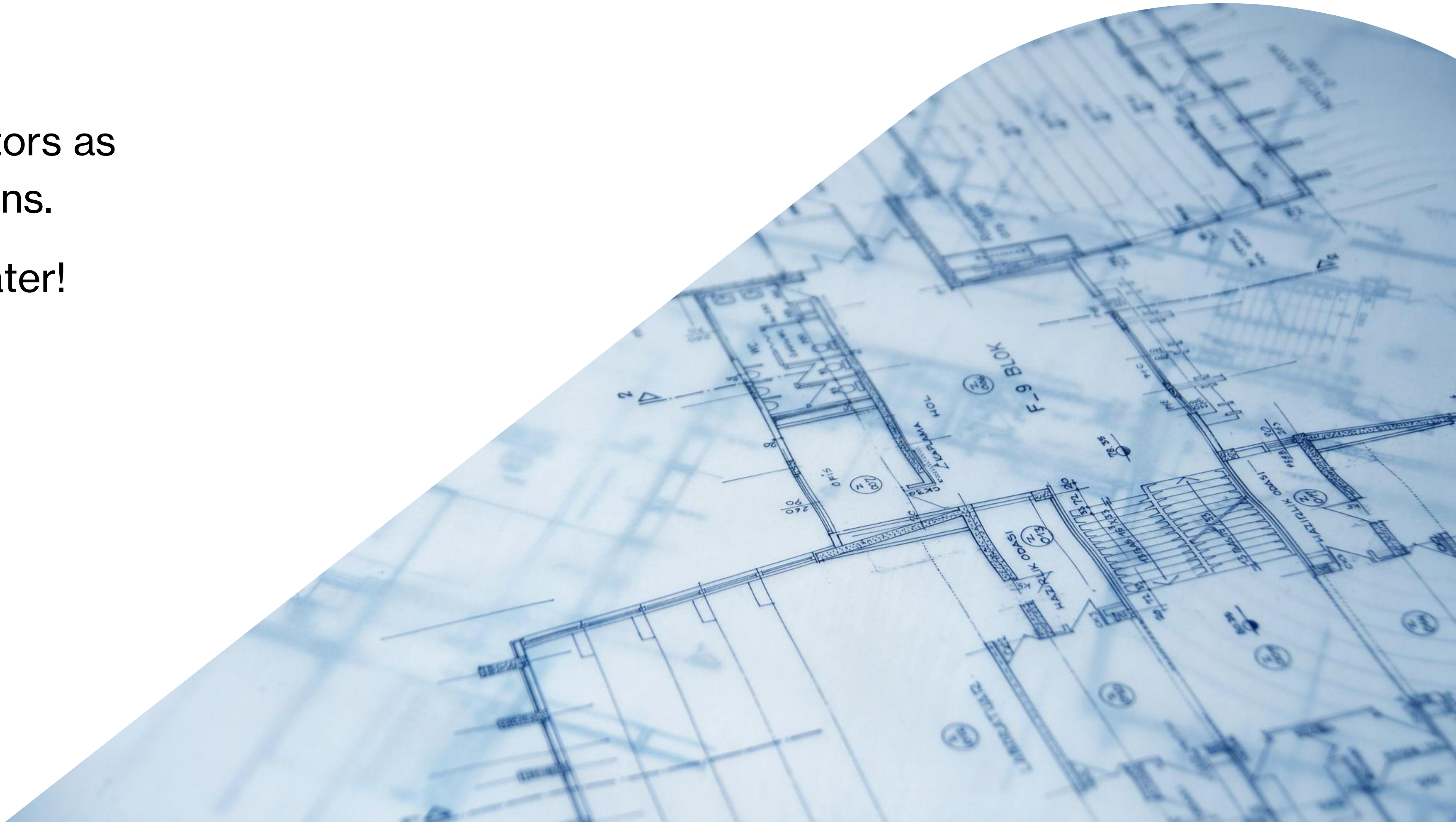
Agenda

1. (Quick) Introduction to iterators
2. (Quick) Introduction to chaining combinators
3. Exercises

1. Iterators as Blueprints

We can understand iterators as blueprints for computations.

Create them now – pay later!



1. Iterators as Blueprints

```
struct Data(u8);
let data = vec![Data(45), Data(54)];
```

1. Iterators as Blueprints

```rust
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0);
```

1. Iterators as Blueprints

```rust
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0); // nothing happened yet!
```

1. Iterators as Blueprints

```rust
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0); // nothing happened yet!
let iter = iter.filter(|&b| b > 10);
```

1. Iterators as Blueprints

```
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0); // nothing happened yet!
let iter = iter.filter(|&b| b > 10); // still nothing happened!
```

1. Iterators as Blueprints

```rust
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0); // nothing happened yet!
let iter = iter.filter(|&b| b > 10); // still nothing happened!
let iter = iter.filter_map(|b| {
    if b % 2 == 0 {
        Some(b.to_string())
    } else {
        None
    }
});
```

1. Iterators as Blueprints

```rust
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0); // nothing happened yet!
let iter = iter.filter(|&b| b > 10); // still nothing happened!
let iter = iter.filter_map(|b| {
    if b % 2 == 0 {
        Some(b.to_string())
    } else {
        None
    }
}); // still nothing happened!
```

1. Iterators as Blueprints

```rust
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0); // nothing happened yet!
let iter = iter.filter(|&b| b > 10); // still nothing happened!
let iter = iter.filter_map(|b| {
    if b % 2 == 0 {
        Some(b.to_string())
    } else {
        None
    }
}); // still nothing happened!
let result: Vec<_> = iter.collect();
```

1. Iterators as Blueprints

```rust
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0); // nothing happened yet!
let iter = iter.filter(|&b| b > 10); // still nothing happened!
let iter = iter.filter_map(|b| {
    if b % 2 == 0 {
        Some(b.to_string())
    } else {
        None
    }
}); // still nothing happened!
let result: Vec<_> = iter.collect(); // compute!
```

1. Iterators as Blueprints

```rust
struct Data(u8);
let data = vec![Data(45), Data(54)];
let iter = data.into_iter().map(|d| d.0); // nothing happened yet!
let iter = iter.filter(|&b| b > 10); // still nothing happened!
let iter = iter.filter_map(|b| {
    if b % 2 == 0 {
        Some(b.to_string())
    } else {
        None
    }
}); // still nothing happened!
let result: Vec<_> = iter.collect(); // compute!
println!("{:#?}", result);
```

```
[
    "54",
]
```

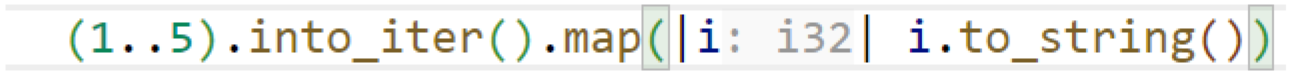
2. Working with iterators

Iterators are most useful when we instruct them
with combinators like

2. Working with iterators

Iterators are most useful when we instruct them with combinators like

- `map()` – Transform type A to type B.

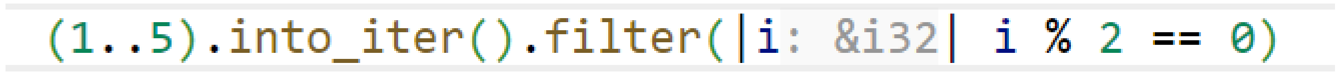


```
(1..5).into_iter().map(|i: i32| i.to_string())
```

2. Working with iterators

Iterators are most useful when we instruct them with combinators like

- `map()` – Transform type A to type B.
- `filter()` – Filter out elements.



```
(1..5).into_iter().filter(|i: &i32| i % 2 == 0)
```

2. Working with iterators

Iterators are most useful when we instruct them with combinators like

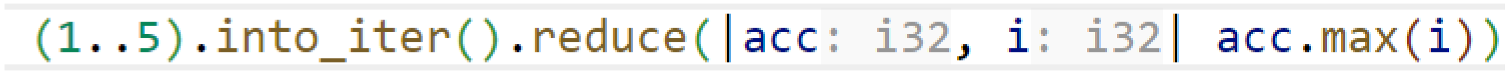
- `map()` – Transform type A to type B.
- `filter()` – Filter out elements.
- `filter_map()` – Transform and filter in one go!

```
(1..5).into_iter().filter_map(|i| {
    if i % 2 == 0 {
        Some(i.to_string())
    } else {
        None
    }
})
```

2. Working with iterators

Iterators are most useful when we instruct them with combinators like

- `map()` – Transform type A to type B.
- `filter()` – Filter out elements.
- `filter_map()` – Transform and filter in one go!
- `reduce()` – Boil down n elements to only one!

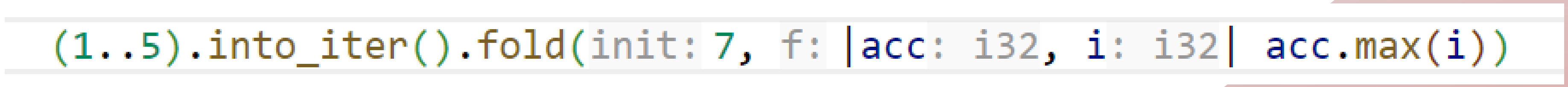


```
(1..5).into_iter().reduce(|acc: i32, i: i32| acc.max(i))
```

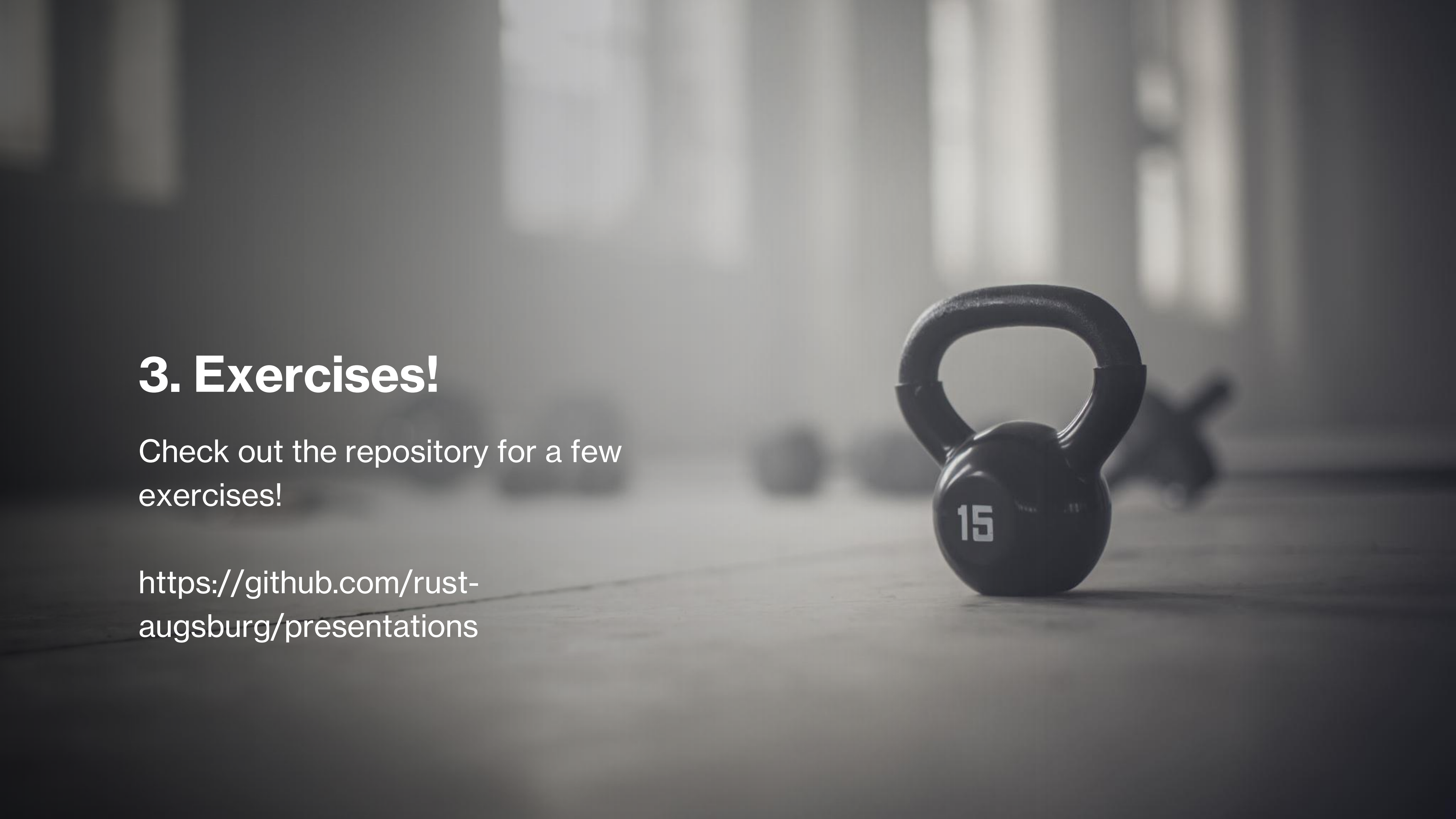
2. Working with iterators

Iterators are most useful when we instruct them with combinators like

- `map()` – Transform type A to type B.
- `filter()` – Filter out elements.
- `filter_map()` – Transform and filter in one go!
- `reduce()` – Boil down n elements to only one!
- `fold()` – Like `reduce`, but you provide an inital value!



```
(1..5).into_iter().fold(init: 7, f: |acc: i32, i: i32| acc.max(i))
```



3. Exercises!

Check out the repository for a few exercises!

https://github.com/rust-augsburg/presentations

Contact

mert.yildiz@posteo.net