# Introductory Rust (2026): Traits, Generics, Iterators

Lecture 4

---

Lukáš Hozda

Winter 2026/27

MFF CUNI

# Reusable Abstractions

# Generic Functions

```rust
fn max_of<T: Ord + Copy>(a: T, b: T) -> T {
    if a > b { a } else { b }
}
```

- Type parameter + trait bounds.
- Monomorphized at compile time.

# Trait Definition

```
trait Score {
    fn score(&self) -> i64;
}
```

- Behavior contract for multiple types.

# Trait Implementation

```
struct Item { weight: i64, value: i64 }

impl Score for Item {
    fn score(&self) -> i64 {
        self.weight * self.value
    }
}
```

# Iterator Pipeline

```
let xs = vec![1, 2, 3, 4, 5, 6];
let even_sum: i64 = xs.iter().copied().filter(|x| x % 2 == 0).sum();
println!("{even_sum}");
```

- Declarative data flow.

# Iterator Adaptors

- `map`, `filter`, `take`, `skip`, `fold`.
- Most adaptors are lazy.
- `collect` materializes results.

# Generic Struct

```
struct Pair<T> { left: T, right: T }
```

- Generic containers keep API compact.

# Trait Bounds On Impl

```
impl<T: std::fmt::Display> Pair<T> {
    fn show(&self) {
        println!("{} {}", self.left, self.right);
    }
}
```

# Static Dispatch

- Default trait use is static dispatch.
- Fast and inlinable.
- Code size grows with many monomorphizations.

# Practical Rule

- Start with concrete types.
- Introduce generics where repetition appears.
- Prefer clear trait contracts over clever type tricks.