LESSON 8

main.rs

```
let text =
  fs::read_to_string(p)?;
Ok(text.len())
```

Rust

Result & ?

Learn it all @MyWebUniversity



A FREE MYWEBUNIVERSITY COURSE

RUST PROGRAMMING

LESSON 8

Error Handling with Result

panic!, Result, the ? operator, custom errors, thiserror and anyhow

panic!: Unrecoverable Errors

terminal

```
$ cargo run
thread 'main' panicked at src/main.rs:4:37:
config.toml must exist: Os { code: 2, kind: NotFound }
note: run with `RUST_BACKTRACE=1` to display a backtrace
```

- panic stops the program with a message. It is for bugs, situations that should never happen.
- unwrap and expect panic on None or Err. They are fine in quick examples and tests, but ...

01

Result

main.rs

```rust
use std::fs;

match fs::read_to_string("scores.txt") {
    Ok(text) => println!("{} lines", text.lines().count()),
    Err(e) => eprintln!("Could not read file: {e}"),
}
let n: i32 = "42".parse().unwrap_or(0);
```

- Functions that can fail return Result: Ok with the value, or Err with an error. Like …
- match on the result, or use methods like unwrap_or_else and map_err.

02

The ? Operator

main.rs

```rust
fn average_from_file(path: &str) -> Result<f64, Box<dyn std::error::Error>> {
    let text = std::fs::read_to_string(path)?;
    let nums: Vec<f64> = text.lines()
        .map(|l| l.trim().parse::<f64>())
        .collect::<Result<_, _>>()?;
    Ok(nums.iter().sum::<f64>() / nums.len() as f64)
}
```

- The question mark after a Result returns the error early if there is one, or gives you the …
- It turns long match chains into clean, short code. main can return Result too.

03

Custom Error Types

error.rs

```rust
#[derive(Debug, thiserror::Error)]
enum GradeError {
    #[error("cannot read file: {0}")]
    Io(#[from] std::io::Error),
    #[error("bad score on line {line}")]
    BadScore { line: usize },
}

fn load(p: &str) -> Result<Vec<f64>, GradeError> { /* ... ? ... */ }
```

- For libraries, define an error enum with one variant per failure. The thiserror crate …
- From conversions let the question mark convert lower-level errors automatically.

04

anyhow for Applications

main.rs

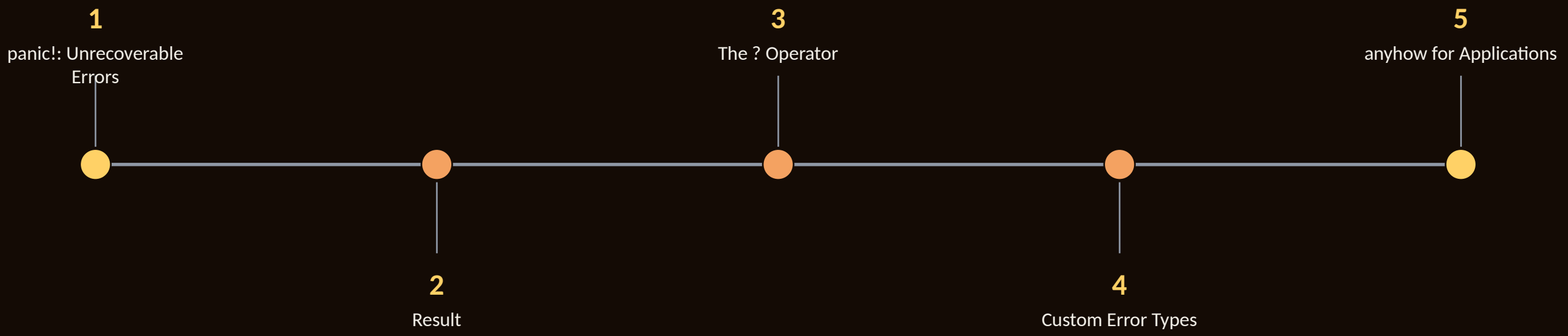
```rust
use anyhow::{Context, Result};

fn main() -> Result<()> {
    let cfg = std::fs::read_to_string("config.toml")
        .context("reading config.toml")?;
    let port: u16 = cfg.trim().parse().context("port must be a number")?;
    println!("port {port}");
    Ok(())
}
```

- In applications, the anyhow crate gives one easy error type. context adds a helpful message …
- Rule of thumb: thiserror for libraries, anyhow for apps.

05

Lesson Map


1
panic!: Unrecoverable Errors
2
Result
3
The ? Operator
4
Custom Error Types
5
anyhow for Applications

Sources and Further Reading

terminal

```
$ cargo run
thread 'main' panicked at src/main.rs:4:37:
config.toml must exist: Os { code: 2, kind: NotFound }
note: run with `RUST_BACKTRACE=1` to display a backtrace
```

RUST PROGRAMMING · 8

main.rs

```rust
fn average_from_file(path: &str) -> Result<f64, Box<dyn std::error::Error>> {
    let text = std::fs::read_to_string(path)?;
    let nums: Vec<f64> = text.lines()
        .map(|l| l.trim().parse::<f64>())
        .collect::<Result<_, _>>()?;
    Ok(nums.iter().sum::<f64>() / nums.len() as f64)
}
```

Thank you for watching

MyWebUniversity.com

Free technical education since 2000 — English and Dari

@MyWebUniversity · @MyWebUniversity-Dari