

Crate | tracing

■ *tracing*

- ◆ The *tracing* crate generates structured diagnostics of your code
- ◆ Features include:
 - Log levels
 - Async support
 - Spans
 - Proc macro helpers
- ◆ *tracing* is a framework, so additional crates are needed for extra features

■ Instrumenting a Function

```
#[tracing::instrument]  
fn do_stuff() {  
    // ...  
}
```

■ Adding Logs

```
use tracing::{info, debug};

#[tracing::instrument(skip(n))]
fn test(s: &str, n: usize) {
    info!("testing");
    debug!(n, "this is a debug msg");
}
```

```
2022-08-13T18:01:21.203761Z  INFO test{s="sample"}:
crate_lecture_tracing: testing
2022-08-13T18:01:21.203800Z  DEBUG test{s="sample"}:
crate_lecture_tracing: this is a debug msg n=10
```

■ Spans

- ◆ A span represents a period of time
- ◆ Anything logged during a *span* will also contain the span information
- ◆ Example usage:
 - Associating IP addresses with logs
 - Showing the current stage in a multi-staged processing pipeline
 - Logging user names when resources are accessed

■ Creating a Span

```
use tracing::{Level, span};
```

```
#[tracing::instrument]
```

```
fn example() {
```

```
    let _span = span!(Level::DEBUG, "showing example")  
                .entered();
```

```
    test("msg", 10);
```

```
    // ...
```

```
}
```

```
#[tracing::instrument(skip(n))]
```

```
fn test(s: &str, n: usize) {
```

```
    info!("testing");
```

```
    debug!(n, "this is a debug msg");
```

```
}
```

■ Output

```
2022-08-13T18:11:43.453682Z  INFO example:  
showing example:test{s="msg"}: crate_lectu  
re_tracing: testing
```

```
2022-08-13T18:11:43.453722Z  DEBUG example:  
showing example:test{s="msg"}: crate_lectu  
re_tracing: this is a debug msg n=10
```

■ Subscriber

- ◆ By default, *tracing* does not log anything
- ◆ A *subscriber* is used to determine how logging should occur
- ◆ *Subscribers* can be customized so only relevant items are logged
- ◆ *Subscribers* should only be used in *application* code
 - Never use in library code

■ *fmt* subscriber

- ◆ Basic terminal logger
- ◆ *cargo add tracing_subscriber*

```
fn main() {  
    let subscriber = tracing_subscriber::fmt()  
        .with_max_level(Level::TRACE)  
        .init();  
    // ...  
}
```

■ *EnvFilter*

- ◆ Allows configuring logs using environment variables
- ◆ *cargo add tracing_subscriber -F env-filter*

```
fn main() {  
    use tracing_subscriber::{EnvFilter, fmt, prelude::*};  
  
    tracing_subscriber::registry()  
        .with(fmt::layer())  
        .with(EnvFilter::from_env("MYAPP_LOG"))  
        .init();  
    // ...  
}
```

■ Recap

- ◆ *tracing* is a logging instrumentation framework
- ◆ By default, all function parameters are logged using `#[tracing::instrument]`
 - To skip parameters, use `#[tracing::instrument(skip(name))]`
- ◆ *spans* are periods of time that will be associated with log events
- ◆ *subscribers* determine how events should be logged
 - Only use a *subscriber* in application code
- ◆ *EnvFilter* allows configuring logs at runtime