BLOCK NONCE #7: Rust enums are great!

What?

Rust enums let you encode valid states in the type system, these states are enforced at compilation time.

Why?

The compiler then enforces correctness: you can't create invalid combinations of fields, you avoid runtime checks and redundancy.

How?

```
struct LoggingConfig {
    enabled: bool,
    level: LogLevel, // What should be here when logging is disabled?
}
```

Let's try to make it optional!

```
struct LoggingConfig {
    enabled: bool,
    level: Option<LogLevel>,
}
impl LoggingConfig {
    fn log(&self, message: &str) {
        if self.enabled {
            // Why taking the risk of panicking here?
            let level = self.level.unwrap();
            log(level, message);
        }
    }
}
```

We should do that instead!

```
enum LoggingConfig {
    Disabled,
    Enabled(LogLevel),
}
impl LoggingConfig {
    fn log(&self, message: &str) {
          if let Self::Enabled(level) = self { // No risk of panicking.
              log(level, message);
        }
    }
}
```

TIP💡 You should consider using structs over enums for (de)serializable objects, as the schema for enums is less compatible with other programming languages (e.g Python).