BLOCK NONCE #9: Use Early Return

Why?

Because your main logic shouldn't be wrapped in layers of `if`:

```
fn watch_movie(user: User) -> Result<(), WatchMovieError> {
    if let AuthStatus::Authenticated(plan) = user.authenticate() {
        if plan.is_premium_plan() {
            if user.pay().is_ok() {
                return user.watch_movie();
            }
        }
    }
    Err(WatchMovieError::AccessDenied)
}
```

And imagine the main logic wasn't just one line... pretty messy, right?

How?

Just flip the condition, add an early return and let the main logic breath.

```
fn watch_movie(user: User) -> Result<(), WatchMovieError> {
    let AuthStatus::Authenticated(plan) = user.authenticate() else {
        return Err(WatchMovieError::NotAuthenticated);
    };
    if !plan.is_premium_plan() {
        return Err(WatchMovieError::NonPremiumPlan);
    }
    user.pay()?;
    user.watch_movie()
}
```

This code is easier to understand - a simple list of checks, each with its corresponding error, followed by the main logic.

TIP💡

1. The `?` operator is another form of early return. It's a shortcut for:

   ```
   match user.pay() { Ok(x) => x, Err(e) => { return Err(e); } }
   ```

2. **Note:** `continue` = early return for loops!
3. If your function needs to handle one enum variant, use the let-else syntax as shown in the example above.
4. Early returns are preferred at the beginning of a function. If you find yourself adding one in the middle — **stop and think**: Is it complicating the flow? Consider refactoring that part into its own function first.