Please enter this text to replace the highlighted text that starts with “However,” at the bottom of page 156 and top of page 157.

However, because File::create could also fail, we need a second arm in the inner match expression. When the file can’t be created, a different error message is printed. The second arm of the outer match stays the same, so the program panics on any error besides the missing file error.

That’s a lot of match! The match expression is very useful but also very much a primitive. In Chapter 13, you’ll learn about closures; the Result<T, E> type has many methods that accept a closure and are implemented using match expressions. Using those methods will make your code more concise. A more seasoned Rustacean might write this code instead of Listing 9-5:

prod: check xref

use std::fs::File;

use std::io::ErrorKind;

fn main() {

let f = File::open("hello.txt").unwrap\_or\_else(|error| {

if error.kind() == ErrorKind::NotFound {

File::create("hello.txt").unwrap\_or\_else(|error| {

panic!("Tried to create file but there was a problem: {:?}", error);

})

} else {

panic!("There was a problem opening the file: {:?}", error);

}

});

}

Although this code has the same behavior as Listing 9-5, it doesn’t contain any match expressions and is cleaner to read. Come back to this example after you’ve read Chapter 13, and look up the unwrap\_or\_else method in the standard library documentation. Many more of these methods can clean up huge nested match expressions when you’re dealing with errors.

prod: check xref