1. Explain the purpose of generics and give one concrete example of their benefit

Generics allow for greater reusability. With generics we don’t need to rebuild the same functionality for 40 different types. Instead, we implement it once and simply pass different type parameters. For instance, if generics didn’t exist, Option would be impossible to create, and instead we would need to separately and manually implement Option\_i31, Option\_f32, Option\_String, and Option\_any\_other\_type\_we\_use.

1. What happens to generics at compile time in Rust

At compile time, generics are monomorphized. This means that the compiler finds every type parameter given and separately compiles the generic for that specific type parameter.

1. Implement a generic function called print\_thing which simply prints the provided instance of its single type parameter with a label

fn print\_thing<T>(thing: T) {

println!(“Thing: {}”, thing);

}

1. Explain why the following code will fail to compile

fn count<T>(list: Vec<T>, search: T) {

let mut counter = 0;

for item in list {

if search == item {

counter += 1;

}

}

return counter;

}

Since we don’t know the type of parameter T, we cannot say for certain if the equality operator is defined for it or what that definition might be. So, Rust can’t compile an operation it doesn’t know how to perform. Furthermore, it would allow types for which the operation isn’t defined to be passed to the function.

1. Declare a generic struct called Pair, which takes one type parameter and has contains two instances of that type as fields

struct Pair<T> {

first: T,

second: T,

}

1. Implement a method to return a reference to the larger of the pair when T is f32

impl Pair<f32> {

fn larger(&self) -> &f32 {

if first > second {

first

}

second

}

}