1. Determine whether the following Rust code follows Rust ownership rules, and if not why?

fn modify(var: &i32) {

\*var += 1;

}

fn main() {

let mut num: i32 = 42;

modify(&num);

println!("{}", num);

}

2. Determine whether the following Rust code follows Rust ownership rules, and if not why?

fn modify(var: &mut i32) {

\*var += 1;

}

fn print(var: &i32) {

println!(“{}”, var);

}

fn main() {

let mut num: i32 = 42;

let r: &i32 = &num;

modify(&mut num);

print(r);

}

3. Determine whether the following Rust code follows Rust ownership rules, and if not why?

fn modify(var: &mut i32) {

\*var += 1;

}

fn print(var: &i32) {

println!(“{}”, var);

}

fn main() {

let mut num: i32 = 42;

modify(&mut num);

print(&num);

}

4. Determine whether the following Rust code follows Rust ownership rules, and if not why? fn life() -> &i32 {

let answer: i32 = 42;

&answer

}

fn main() {

println!(“{}”, \*life());

}

Answers:

1. The modify() function attempts to mutate an immutable reference

2. Function modify() attempts to borrow a mutable reference when other references already exist of that variable

3. Valid

4. Function life() attempts to return a reference to a local variable which is going out of scope