University of Toronto, Department of Electrical and Computer Engineering

ECE 1724-F1 — Performant Software Systems with Rust

Quiz 5 — Sample Quiz 3

Wednesday, November 25, 2026

Instructions: Choose exactly one answer per question. Each complete program is independent. Ignore compiler warnings; assume Rust 2024. These teaching examples use standard-library APIs and patterns; no knowledge of library internals is required.

1. Why does this complete program fail to compile?

```
fn first<'a>(left: &'a str, _right: &'a str) -> &'a str {
    left
}
fn main() {
    let outer = String::from("long lived");
    let result;
    {
        let inner = String::from("brief");
        result = first(&outer, &inner);
    }
    println!("{}", result);
}
```

(A) A function cannot return an input reference.

(B) The signature ties the result to both input lifetimes, but inner is dropped before the result is used.

(C) Every reference argument must have a different named lifetime.

(D) outer is moved into first.

2. Which statement explains why the return type of choose needs no explicit lifetime annotation?

```
struct Label {
    text: String,
}
impl Label {
    fn choose(&self, _fallback: &str) -> &str {
        &self.text
    }
}
fn main() {
    let label = Label {
        text: String::from("ready"),
    };
    println!("{}", label.choose("unknown"));
}
```

(A) The method elision rule ties the output lifetime to the borrow of self.

(B) The output must have a static lifetime because unknown is a string literal.

(C) The output lifetime is tied to _fallback instead of self.

(D) Rust never requires annotations when there are two input references.

3. What does this complete program print? unwrap_or_else calls its closure for None.

```
fn main() {
    let saved = String::from("cached");
    let fallback = move || saved;
    let requested: Option<String> = None;
    let name = requested.unwrap_or_else(fallback);
    println!("{}", name);
}
```

Ⓐ It fails to compile because closures cannot own Strings.

Ⓑ It prints an empty string.

Ⓒ It prints cached.

Ⓓ It fails to compile because unwrap_or_else must call a closure twice.

4. What does this complete program print?

```
fn main() {
    let values = vec![2, 4, 6];
    let total: i32 = values.iter().map(|value| value + 1).sum();
    println!("{} {}", total, values.len());
}
```

Ⓐ 12 3

Ⓑ 15 0

Ⓒ It fails to compile because sum consumes values.

Ⓓ 15 3

5. Which statement about this complete program is correct?

```
fn main() {
    let first = Box::new(String::from("log"));
    let second = first;
    println!("{}", second);
}
```

Ⓐ Assigning first to second copies the String into a second heap allocation.

Ⓑ Ownership of the box moves to second; the program prints log.

Ⓒ The assignment shares ownership, so both names can still be used afterward.

Ⓓ It fails to compile because Box values cannot be moved.