TA Guide: Rust Lab 03 – Mastering Variables and Logic in Rust

Date: 15/7/2025

1.1: Variable Basics and Mutability

🔷 Code Example:

```
fn main() {
    let year = 2025;
    let mut month = 7;
    month = 12;
    println!("Year: {}, Month: {}", year, month);

    // The following line causes error and must be fixed
    // year = 2026; // ❌ cannot assign to immutable variable
    // Fix: declare year as mutable if change is needed
    // let mut year = 2025; year = 2026;
}
```

🔶 TA Note:

Check if student declared `year` as immutable, used `mut` for `month`, and handled the reassign error for `year` properly.

1.2: Type Inference and Consistency

🔷 Code Example:

```
fn main() {
    let mut price = 99.99;
    // price = 100; // ❌ mismatched types: expected f64, found integer
    price = 100.0;
    let discount_applied = price < 100.0;
    println!("Discount applied: {}", discount_applied);
}
```

🔶 TA Note:

Ensure student shows type mismatch and fixes it properly (e.g. by using 100.0), and uses inferred boolean variable.

1.3: Logic Expressions and Boolean Connectives

🔷 Code Example:

```
fn main() {
    let available = true;
    let mut in_stock = false;
    let rating = 4.5;
```

```
    let is_good = available && rating > 4.0 && in_stock;
    println!("Is good (first): {}", is_good);

    in_stock = true;
    let is_good = available && rating > 4.0 && in_stock;
    println!("Is good (second): {}", is_good);

    println!("!available = {}", !available);
    println!("available || in_stock = {}", available || in_stock);
    println!("rating < 3.0 || rating > 4.0 = {}", rating < 3.0 ||
rating > 4.0);
}
```

🔶 TA Note:

Verify student reassigns `in_stock`, evaluates all logic expressions, and understands short-circuit evaluation.

1.4: Shadowing and Redeclaration

🔷 Code Example:

```
fn main() {
    let mut score = 80;
    score = score + 10;
    let score = score > 85;
    let score = if score { "Passed" } else { "Failed" };
    println!("Score status: {}", score);
}
```

🔶 TA Note:

Check for correct mutability, shadowing with boolean and string types, and that the final print reflects expected value.

1.5: Scope and Lifetime

🔷 Code Example:

```
fn main() {
    let a = 10;
    {
        let b = a + 5;
        println!("Inner block: a = {}, b = {}", a, b);
    }
    // println!("b = {}", b); // ❌ Error: `b` does not live long enough

    let b = 15; // Shadow b outside block
    println!("b shadowed: {}", b);

    {
```

```
        let mut x = 1;
        println!("x = {}", x);
        let x = true;
        println!("x shadowed as bool = {}", x);
        let x = "done";
        println!("x shadowed as str = {}", x);
    }
}
```

🔶 TA Note:

Ensure variable `b` is not printed outside scope unless shadowed, and student shows valid type and mutability shadowing.

1.6: Code Review and Commenting

🔷 Code Example:

```
// Students should annotate their own previous code sections with:
// - Variable purpose
// - Why mutability is used
// - How shadowing is applied
// - What errors occurred and how they were resolved
```

🔶 TA Note:

Ensure each section from Part 1–5 has comments explaining the student's reasoning and handling of errors/shadowing/types.

2. Event Ticket Discount Checker

🔷 Code Example:

```
fn main() {
    let base_price = 150.0;
    let mut discount = 0.0;

    let is_student = true;
    let is_early_bird = false;
    let has_coupon = true;

    if is_student { discount += 0.10; }
    if is_early_bird { discount += 0.20; }
    if has_coupon { discount += 0.05; }

    let final_price = base_price - (base_price * discount);

    {
        let free_entry = final_price < 50.0;
        println!("Free entry: {}", free_entry);
    }
```

```
    println!("Base ticket price: ${}", base_price);
    println!("Student discount applied: {}", is_student);
    println!("Early bird discount applied: {}", is_early_bird);
    println!("Coupon used: {}", has_coupon);
    println!("Final ticket price: ${}", final_price);

    // println!("Free entry: {}", free_entry); // ❌ Error: `free_entry`
out of scope
}
```

🔶 TA Note:

Confirm correct use of all booleans, discount accumulation, shadowing of `final_price`, and scoping rule for `free_entry`.