Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

1. One major disadvantage of Rust is that it automatically adds runtime type safety checks, which negatively affects its runtime performance. **T** **F**

2. In Rust, ownership is a mechanism to check for memory leaks at runtime. **T** **F**

3. Rust literals are first class citizens. **T** **F**

4. The `as` operator in Python and Rust does the same thing. **T** **F**

5. The Rust compiler will attempt to copy an object *only after* it decides that moving the object is not permissible. **T** **F**

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following keyword can be used as part of an expression in Rust?

i. `if`

ii. `let`

iii. `loop`

iv. `match`

v. `use`

b) Which of the following types of bugs are absent from Rust programs?

i. Data race

ii. Null pointer exception

iii. Divide by zero error

iv. Dangling pointers

v. Buffer overflow

Question 3. Short Answer

a) What is the difference between the following two sets of statements?

```
// set 1
let x = 5;
let x = x + 2;
let x = x * 3;

// set 2
let mut x = 5;
x = x + 2;
x = x * 3;
```

In set 1, there are three immutable variables with the same name. The second replaces the first and the third replaces the second.

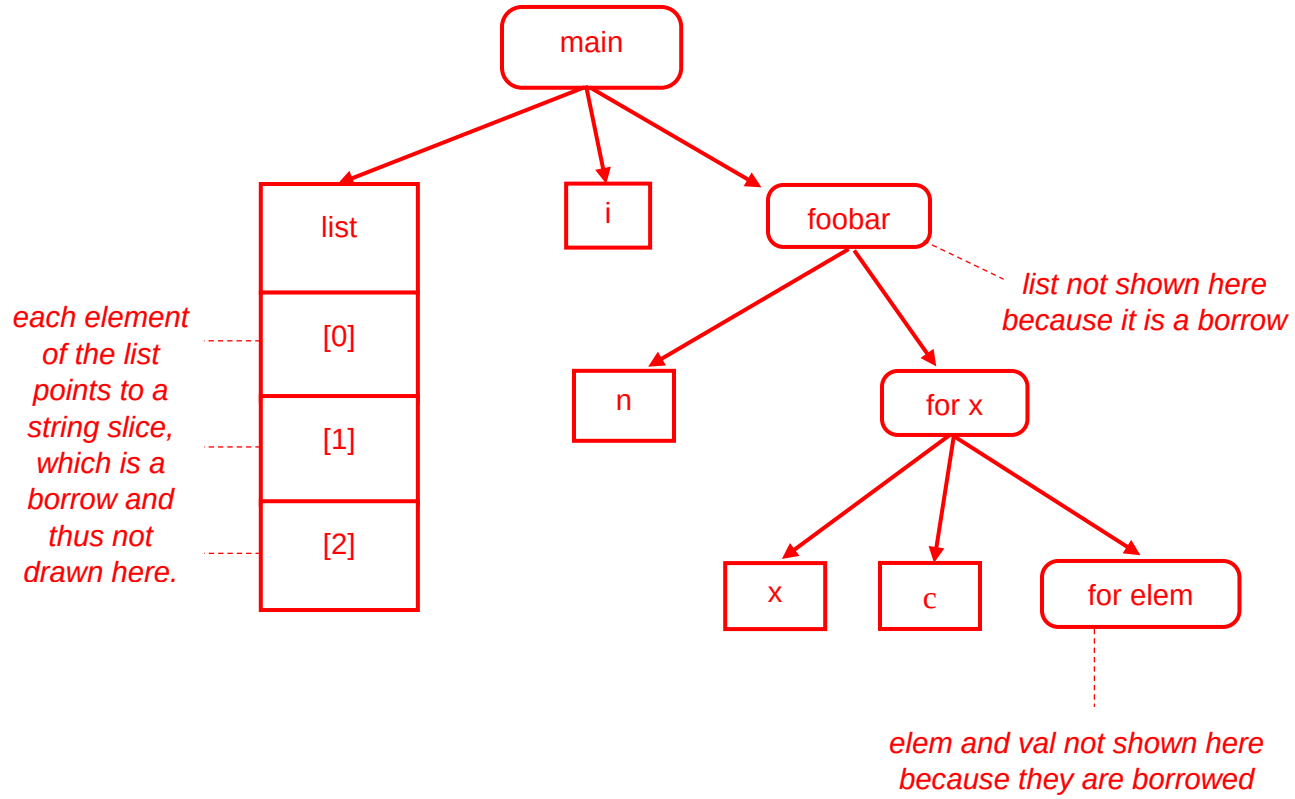
In set 2, x is a mutable variable and you are free to change its value.

b) Given the following program:

```
fn foobar(n: i32, list: &Vec<Option<& str>>) {
    for x in {0..n} {
        let mut c = 0;
        for elem in list {
            let val = elem.unwrap();
            println!("{}.{}: {}", x, c, val); /* here */
            c += 1;
        }
    }
}

fn main() {
    let i = 5;
    let list = vec![ Some("hello"), Some("new"), Some("world") ];
    foobar(i, &list);
}
```

Draw an ownership diagram in the form of a tree at the point when "0.2: world" is printed (the line with the comment "here"). Do **not** show borrowed objects.



Question 4. Programming Questions

a) Given the following definition for Shape:

```
enum Shape {
    Rectangle(f64, f64), /* width, height */
    Triangle(f64, f64),  /* height, base */
    Circle(f64),         /* radius */
}
```

Complete the following function, which sums the area of a list of Shapes. Assume constant `PI` is defined.

```
use Shape::{Rectangle, Triangle, Circle};
use std::f64::consts::PI;

fn total_area(list: &Vec<Shape>) -> f64 {

    let mut sum = 0.0;

    for s in list {
        sum += match s {
            Rectangle(w, h) => w * h,
            Triangle(b, h) => b * h / 2.,
            Circle(r) => PI * r * r,
        };
    }

    sum

}
```

b) Create the following function that will parse a csv file filled with integers and return a 2-dimensional vector of integers. Hint: you can parse a string to an integer using the parse method. Return an io::Error with ErrorKind::Other if a value cannot be parsed to an integer.

```
fn parse_csv(filename: &str) -> Result<Vec<Vec<isize>>, io::Error> {



let mut f = File::open(filename)?;
    let mut s = String::new();
    f.read_to_string(&mut s)?;

    let mut ret = Vec::new();
    let rows = s.split("\n");
    for row in rows {
        if row.trim().len() == 0 {
            continue;
        }

        let cols = row.split(",");
        let mut retrow = Vec::new();
        for col in cols {
            match col.trim().parse::<isize>() {
                Ok(val) => retrow.push(val),
                Err(e) => return Err(Error::new(ErrorKind::Other,
                                        format!("{}", e))),
            }
        }

        ret.push(retrow);
    }

    Ok(ret)









}
```

c) Write a function, reverse_vector, that takes an input vector of integers and will return an output vector whose elements are reversed. To spice things up, you are only allowed to use recursion to solve this problem. Hint: you will need a helper function to do this.

```
fn reverse_vector(v: & Vec<i32>) -> Vec<i32> {



    // turn v into a slice
    let s = &v[..];

    // start recursion
    return reverse_vector_helper(s);



}




fn reverse_vector_helper(v: & [i32]) -> Vec<i32> {
    if v.len() == 0 {
        return Vec::new();
    }

    let i = v[0];
    let v = &v[1..];
    let mut ret = reverse_vector_helper(v);

    ret.push(i);
    return ret;
}
```