Week 11: Rust Basics

-- General --

Rust:

→ Is an Imperative language with features of Functional
- Uses a sequence of statements to determine how to reach a certain goal
- Favors functional principle of immutability, has high order functions (map / fold) and closures
- But not purely functions

→ Is Statically Typed
- Compiled language, types checked at compile time

→ Has Explicit Declarations
- `pub fn foo(num:i32) -> String`

Basic Types

→ i32 - a 32 bit integer
→ f64 - a 64 bit floating point number
→ bool
→ String
→ tuples
→ () - no type just like in OCaml

If Expressions and Loops

→ Branches of an If must have same type
→ Else branches are not required

```
let x = if a && b { 10 } else { 5 };
if a && b { x = 17 } else { y = 10 }
```

→ notice the second example … both types are () because it's an assignment
→ For loops can use range types to index

```
for i in (0..5) { println!("{}", i); } // prints 0 through 4
while ctr < 5 { println!("{}", ctr); ctr += 1; }
```

→ For each loops use iterators

```
for ele in arr.iter() { println!("{}", ele); }
```

-- Ownership Rules --

(1) There can only be one owner of a piece of DATA a time
(2) If the DATA has the "Copy trait" we can assign without transferring ownership
(3) When the owner goes out of scope, the DATA is dropped

Examples:

```
(1) let x = String::from(“hello”);  // x is owner of “hello”
    let y = x;                      // y is owner of “hello”

(2) let x = 5;               // x owns 5
    let y = x;              // y owns a separate 5

(3) {
          let x = String::from(“hello”)   // x owns “hello”
    }    // “hello” is dropped / freed
```

-- Referencing Rules --

(1) Can only make one mutable reference [to a mutable variable], blocking USE of original
(2) Can make many immutable references, blocking MUTATION of original
(3) Can't have both mutable references and immutable references simultaneously

Examples:

```
(1) let mut x = String::from(“hello”)
    let y = &mut x;
    let z = &mut x;          // ERROR
    println!(“{}”, y);
    println!(“{}”, x);       // ERROR

(2) let mut x = String::from(“hello”)
    let y = &x;
    let z = &x;              // OK
    println!(“{}”, y);
    println!(“{}”, x);       // OK
    x.push_str(“ world”);    // ERROR

(3) let mut x = String::from(“hello”)
    let y = &x;
    let z = &mut x;          // ERROR
```

-- Understanding the Error --

Use what you know about ownership rules, referencing, and mutability to determine what the errors are in these code snippets. If you aren't sure then try it out in rust playground!

```
/* Immutable referencing */
fn one() {
    let x = 1;
    let p = &x;

    // *p = 17;          // Why is this an error?
    // Is x still the owner of the original data?

    //let q = &x;        // Can we do this?
    //let q = &mut x;    // What about this?

    if *p == 17 {
        println!("p is 17")
    } else {
        println!("p is not 17")
    }
    // p == 17 compares an integer to a reference . . . error!

    println!("p is {}", p);
    println!("x is {}", x);
}

/* Mutable referencing */
fn two() {
    let mut x = 1;
    let p = &mut x;
    *p = 17;                    // Now this is OK!
    //let q = &x;               // Why is this an error?
    //let q = &mut x;           // This?

    if *p == 17 {
        println!("It worked!")
    } else {
        println!("It failed!")
    }

    println!("p is {}", p);
    //println!("x is {}", x);     // Why is this an error?
}
```

```
/* Ownership */
fn three() {
    let x = String::from("three");
    {
        let y = x;                  // Owner is y
        //let z = x;                // Why is this wrong?
        let z = y;                  // Owner is z

        //println!("x is {}", x);   // Error
        //println!("y is {}", y);   // Error
        println!("z is {}", z);     // Fine
    }
    // who is the owner of the data at this line?  Is there an owner??
    // println!("x is {}", x);      // why is this an error?

    let a = String::from("three_more");
    {
       let y = a;
       // a = y;       Is this OK??
    }
    //println!("a is {}", a);        // still problematic

    let mut b = String::from("three_again");
    {
        let y = b;
        b = y;                      // Why is this OK now?
    }
    println!("b is {}", b);          // Why is this OK now?
}

/* More Ownership */
fn four() {
    let x = String::from("four");
    {
        // let w = &mut x    // why is this an error?
        let y = &x;          // x is still owner of the data
        let z = &x;
        println!("x is {}", x);
        println!("y is {}", y);
        println!("z is {}", z);
    }
    println!("x is {}", x);  // OK: data not freed
}
```

```
/* Last Ownership */
fn five() {
    let mut x = String::from("five");
    {
        let y = &mut x;
        // x.push_str(" functions");    // why is this an error?
        // println!("x is {}", x);
        y.push_str(" functions");
        println!("y is {}", y);
        // let z = &x;                  // what about this one?
    }
    x.push_str("!");                    // Now this is OK!

    {
        let z = &x;
        // z.push_str("!");             // this is an error
        println!("z is {}", z);
        println!("x is {}", x);
    }
    println!("x is {}", x);
}
```