Ownership, References & Borrowing, Slices Handout

The borrowing and ownership mechanism can be simplified down to:

- Passing a variable by value will move ownership, dropping the original variable from memory. Note: applies to not copy types.
- Passing a variable by reference will keep the original variable.
- Passing a variable by mutable reference will keep the original variable, but allow you to modify the variable.
- You may only borrow a variable mutably once at a time, and you may not immutably borrow while mutably borrowing.
- You may have as many immutable borrows as you want, so long as you aren't modifying that value.
- You may mutably borrow a field in a struct, and then mutably borrow a different field in the same struct simultaneously, so long as you aren't also mutably borrowing the overall struct.
- You may mutably borrow multiple slices from the same array simultaneously so long as there is no overlap. Note: safe code cannot convince the compiler that two slices don't overlap. You‘ll have to use unsafe code.
- Safe memory practices means that instead of mutably borrowing the same variable in multiple places, you queue the changes to make in a separate location and apply them serially one after another.

```rust
// Source: http://intorust.com/tutorial/ownership/
/*
Lines are uncommented so it can bee compiled
remove comments on  10,11 17, 47 to start the exercise
proceed with the Goals
*/

fn main() {
    let (adjective, name) = two_words();
    let name = format!("{} {}", adjective, name);
    print_out(name);
}

fn two_words() -> (String, String) {
    (format!("fellow"), format!("Rustaceans"))
}


fn remove_vowels(name: String) -> String {
    // Goal #1: What is needed here to make this compile?
    let output = String::new();
    for c in name.chars() {
        match c {
            'a' | 'e' | 'i' | 'o' | 'u' => {
                // skip vowels
            }
            _ => {
                output.push(c);
            }
        }
    }
    output
}

fn print_out(name: String) {
    let devowelized_name = remove_vowels(name);
    println!("Removing vowels yields {:?}", devowelized_name);

    // Goal #2: What happens when you uncomment the `println` below?
    // Can you change the code above so that the code below compiles
    // successfully?
    //
     //println!("Removing vowels from {:?} yields {:?}",
     //         name, devowelized_name);

    // Extra credit: Can you do it without copying any data?
    // (Using only ownership transfer)
}
```