Part 1: multiple question choice on Rust semantics

1. A:
   Will the following code compile? why?

```
fn main() {
    let x = String::from("hello");
    let y = x; // this is line 3
    println!("{}, world!", y);
    println!("{}, world!", x);

    let x = 10;
    let y = x;
    println!("y={}", y);
    println!("x={}", x);
}
```

   Answers:

   1. yes
   2. no, x is moved
   3. no, x and y are defined twice
   4. no, x dose not implement the clone trait
   5. no, x is borrowed twice

   B:
   If not, how to fix it?

   Answers:
   1. replace line 3 with let y = x.clone();
   2. replace line 3 with let y: &String = x;
   3. add y.drop(); between the first and the second println!
   4. rename x and y in the second half of the code to x_1 and y_1
   5. delete println!("{}, world!", y);
2. A:
   Will the following code compile? why?

```
let x = &[1,2,3,4];
let y = x; // this is line 2
println!("{:?}", y);
println!("{:?}", x);
```

Answers:

1. yes
2. no, x is moved
3. no, [i32;4] dose not implement the copy trait
4. no, x does not implement Debug

B:
If not, how to fix it?

Answers:

1. replace line 2 with let y = &x;
2. replace line 2 with let y = x.clone();
3. add y.drop(); between the 2 println!
4. replace line 3 with println!("{:?}", y.clone());
5. switch the order of the two println!

3. A:
The owner of vec's original data at HERE is ?

```
fn foo(vec: Vec<i32>) { // this is line 1
    let x = vec.clone();
    let y = vec;
    let z = &y;
    let w = y;
    // HERE
}
```

1. x
2. y
3. z
4. w
5. the code doesn't compile

B:
When is the value of `x` dropped?

Answers:

1. after line 2
2. after line 3
3. after line 4
4. after line 5

4. A:
Will the following code compile? why?

```
fn main() {
    let s1 = String::from("hello ");
    let s2 = String::from("world!");
    let s3 = concat(s1,s2);
    println!("{} + {} = {}",s1,s2,s3); // this is line 5
}

fn concat(s1:String, s2: String) -> String {
    s1 + s2.as_str()
}
```

Answers:

1. yes
2. no, s1 and s2 are moved inside the function
3. no, is not possible co concatenate to s1 since s1 is not mutable
4. no, the String type dose not support the + operator

B:
If not, how to fix it?

Answers:

1. replace line 4 with let s3 = concat(s1.clone(),s2.clone())
2. the function should contain return (s1 + s2.as_str()).clone();
3. replace line 5 with println!("{} + {} = {}",&s1,&s2,&s3);

5. Does the following code compile ?

```
{
    let x: &u8;
    let v = 10;
    match v {
        0..=10 => x = &v,
        _ => {}
    }
    println!("{x}");
}
```

Answers:

1. Yes,
2. No, the x reference must be initialized at the first line
3. No, the x reference is not mutable and can't be overwritten with &v
4. No, x is possibly uninitialized

6. Does the following code compile ?

```
let mut value: u8 = 5;
{
    let second = &mut value;
    {
        let third = 12;
        *second = third; //this is line 6
    }
}
println!("{}", value);
```

Answers:

1. No, third has type i32, and can't be assigned to a u8 type.
2. No, third's life time ends after line 6, and can't be borrowed at line 9.
3. Yes, it displays 5
4. Yes, it displays 12

7. Given the following code:

```
        let s = [1,2,3,4];
        // This is line 2
        let x = &s;
    println!( "{:?}", x);
```

what can i add at line 2 without breaking the code?

1. s[0] = 0;
2. let b = &mut s;
3. let slice = &s[0..2];
4. s.push(5)
5. None of the above

Part 2: coding Rust concepts

1. Write a function `prev_str` that takes an `&str` as input and returns a `String`.
   This function converts the `&str` by replacing each character with its predecessor.
   For example:

   - 'b' becomes 'a'
   - 'f' becomes 'e'
   - 'B' becomes 'A'
   - 'A' remains 'A'
   - 'a' remains 'a'

   If the character is not a letter, it remains unchanged.

2. Write a struct `X` with two fields: `s` (an `Option<String>`) and `i` (an `i32`). Then, implement the following methods for `X`:
   - `new`: takes a `&str` and an `i32` and returns an `X` instance
   - `take_str`: takes a mutable reference to self and returns the `s` field of `X`, replacing it with `None`
3. Create a function named `replace_surname` that takes a `NameSurname` struct (a struct with the field `name: String` and `surname: String`) and a String as input and returns (guess what?) a String. The function should replace the surname of the `NameSurname` struct with the `String` and return the old surname.
   use `std::mem::swap;`
4. Write a struct `Student` with two fields: `name` (a `String`) and `id` (a `u32`). Then, implement the following methods for `Student`:

   - new: takes a `&str` and a `u32` and returns a `Student` instance
   - Display: implement the `Display` trait for `Student` so that it prints the name and the id of the student

   Then write a struct `University` with two fields: `name` (a `String`) and `students` (a `Vec<Student>`). Then, implement the following methods for `University`:
   - new: takes a `&str` and a `&[Student]` and returns a `University` instance
   - remove_student: takes an `id: u32` and returns a `Result<Student, &str>`:
     - `Ok(Student)` if a student with the given id is found and removed
     - `Err("Not found")` otherwise
   - Display: implement the `Display` trait for `University` so that it prints the name and the list of students of the university
5. Write a struct `AirFleet` that contains a vector of Airplane. Airplane is a struct that contains a company (An enum called `AirplaneCompany` with options: Airbus or Boeing) and a model (String). Implement the following methods for `AirFleet`:
   - `remove_boeing`: remove all the airplanes of the company Boeing from the fleet
   - `add_airplane`: add an airplane to the fleet
   - `search_airplane`: search an airplane by model and return the company of the airplane, it must return a `Result<AirplaneCompany, String>`
     The function must return OK if the airplane is found, Err if the airplane is not found, or the fleet is empty.
6. Create the module `hashmaps` that contains a `struct Maps` with a field `map` of type `HashMap<Unumber, String>`, then create the module `unumber` that contains a type `Unumber` of type `usize`
   In another module create a function `string_to_tuple` that takes a `Maps` and returns a `HashMap<Unumber, (Unumber, String)>`. The function should convert the `HashMap<Unumber, String>` to a `HashMap<Unumber, (Unumber, String)>`, the key remains the same, and the value is a tuple, its first field is the length of the `String` and its second field is the `String` itself

7. Write a `struct Size` that has two fields: `width` and `height`, both of type `u32`. Then, implement the following methods for `Size`:
    - `new`: takes two `u32` arguments and returns a `Size` instance
    - `area`: takes a reference to self and returns the area of the `Size` instance
    - `compare`: takes a reference to self and another `Size` instance and returns an `Option<bool>`:
        - `None` if the two `Size` instances have the same area
        - `Some(true)` if the area of the first `Size` instance is greater than the area of the second `Size` instance
        - `Some(false)` if the area of the first `Size` instance is less than the area of the second Size instance
8. Write a struct `MaybePoint` that has two fields: `x` and `y`, both of type `Option<i32>`. Then, implement the following methods for `MaybePoint`:
    - new: takes two `Option<i32>` arguments and returns a `MaybePoint` instance
    - is_some: takes a reference to self and returns `true` if both `x` and `y` are `Some` values, `false` otherwise
    - maybe_len: takes a reference to self and returns an `Option<f32>`:
        - `None` if `x` or `y` are `None`
        - `Some(len)` where `len` is the length of the vector from `(0, 0)` to `(x, y)`
9. Write a function `res1` that takes an `i32` and returns a `Result<i32, String>`. The function returns:
    - `Ok(n)` if `n` is divisible by 10, `Err("error")` otherwise. Then write a function `res2` that takes a `Result<i32, &str>` and returns a `Result<i32, String>`.
    - The function returns `Ok(n)` if `n` is divisible by 5, `Err("error")` otherwise.
    - Then write a `wrapper` function that takes an `i32` and returns a `Result<i32, String>`.
    - The function returns `Ok(n)` if `n` is divisible by 10 and 5, `Err("error")` otherwise. Errors should be propagated.
10. Create a function `order` that take a `Vec` of `String`s and returns a vector of `String`s, where each string is prepended by its index in the vector followed by a dash and a space.
    For example, given the vector `["How", "Are", "You"]` the function should return `["0 - How", "1 - Are", "2 - You"]`.
11. Define two `enum`s both called `X`, place them in two different modules, `modx` and `mody`. Define the `enum`s like this:

    - With 2 variants `S` with a `char` and `C` with a `String`
    - With 1 variant `F` with a `f64` and a `usize`
      Write a function `sum` in the module `modsum` that takes a `X` from `modx` and a `X` from `mody`, the function returns the `u8` equivalent of the `char` or the length of the `String` based on the variant, summed with the product of `f64` by the `usize`.

The modules can be put in the same file, in this way

```
        mod xyz {
        //insert your enum
}

mod zyx {
        //insert your enum
}

mod modsum{

                use super::xyz;
        use super::zyx;

        fn sum(){
        //insert your code here
        }
}
```