**Name:** Muhammad Auria Ahmad

**Roll #** PIAIC90066

1. Define Traits in your own words?

Traits are like a certain properties of datatypes. They also look like the behavior of datatypes

1. Illustrate with an example how using Traits can help you organize your code and reduce  
   duplication in your program.

Suppose we have some different struct (different self defined data type) and these all data types have some same behavior. You define behavior in a trait and implement those behaviors for datatype. You don’t need to write same behavior again and again for different data type. E.g.

struct **Superman**{

  name: **String**,

}

struct **Batman**{

  name: **String**,

}

struct **Hulk**{

  name: **String**,

}

struct **Spiderman**{

  name: **String**,

}

pub trait **Power** {

  fn **power\_score**(&self)->**u8**{

    50

  }

}

impl **Power** for **Superman**{

  fn **power\_score**(&self) ->**u8** {

      100

  }

}

impl **Power** for **Batman**{

  fn **power\_score**(&self) ->**u8** {

      80

  }

}

impl **Power** for **Hulk**{}

impl **Power** for **Spiderman**{}

fn **main**(){

 let super\_man = **Superman**{

   name:**String**::**from**("Superman")

 };

 let bat\_man = **Batman**{

   name:**String**::**from**("Batman")

 };

 let hulk = **Hulk**{

   name:**String**::**from**("Hulk")

 };

 let spider\_man = **Spiderman**{

   name:**String**::**from**("Spiderman")

 };

**println!**("{}",super\_man.**power\_score**());

**println!**("{}",bat\_man.**power\_score**());

**println!**("{}",hulk.**power\_score**());

**println!**("{}",spider\_man.**power\_score**());

}

1. Write a rust program:  
   a. Define a struct IOT\_student with attributes (name, age, education).  
   b. Define another struct IOT\_instructor (name, age).  
   c. Define a trait Questions with method ask\_Questions with a default  
   implementation which prints (“​Zoom session will be LIVE, Zoom recording willnot be available. Quarter 2 studio recorded videos are available on Portal.​”).  
   d. Impl trait Questions for IOT\_instructor which overrides the default implementation  
   of method ask\_question, takes student name as a parameter and prints on  
   screen (“{} ​In case of any issue email to education@piaic.org​”).  
   e. Create instances of both the structs and call Method ask\_question.

*// a. Define a struct IOT\_student with attributes (name, age, education).*

#[**derive**(**Debug**)]

struct **IotStudent**{

  name: **String**,

  age: **i32**,

  education: **String**,

}

*// b. Define another struct IOT\_instructor (name, age).*

#[**derive**(**Debug**)]

struct **IotInstructor**{

  name: **String**,

  age: **i32**,

}

*// c. Define a trait Questions with method ask\_Questions with a default*

*// implementation which prints (“​Zoom session will be LIVE, Zoom recording will*

*// not be available. Quarter 2 studio recorded videos are available on Portal.​”).*

trait **Questions** {

  fn **ask\_questions**(&self, student\_name: **String**)->**String**{

**String**::**from**("Zoom session will be LIVE, Zoom recording will not be available. Quarter 2 studio recorded videos are available on Portal.")

  }

}

*// d. Impl trait Questions for IOT\_instructor which overrides the default implementation*

*// of method ask\_question, takes student name as a parameter and prints on*

*// screen (“{} ​In case of any issue email to education@piaic.org​”).*

impl **Questions** for **IotInstructor**{

  fn **ask\_questions**(&self, student\_name: **String**) ->**String** {

**format!**("{} ​In case of any issue email to education@piaic.org", student\_name)

  }

}

*// e. Create instances of both the structs and call Method ask\_question.*

fn **main**(){

let student = **IotStudent**{

  name: **String**::**from**("Orya"),

  age: 25,

  education: **String**::**from**("Electronics Engineering"),

};

let instructor = **IotInstructor**{

  name: **String**::**from**("Sir Hanzala"),

  age: 23,

};

**println!**("{}",instructor.**ask\_questions**(student.name));

}

1. Go through the solution of the largest function given at the end of 10.2 in the book and  
   rewrite the solution but this time returning the smallest item instead largest.

**Note:** Last example is this, 2nd last example of section 10.2 is after this

use **std**::**fmt**::**Display**;

struct **Pair**<**T**>{

  x:**T**,

  y:**T**,

}

impl <**T**> **Pair**<**T**>{

  fn **new**(x: **T**, y: **T**) -> **Self** {

**Self**{x, y}

  }

}

impl <**T**: **Display** + **PartialOrd**> **Pair**<**T**>{

  fn **cmp\_display**(&self){

    if self.x <= self.y{

**println!**("The smallest number is x = {}", self.x);

    }else{

**println!**("The smallest number is y = {}", self.y);

    }

  }

}

fn **main**(){

  let nums = **Pair**::**new**(3,1);

  nums.**cmp\_display**();

}

**2nd last example**

fn **smallest** <**T**: **PartialOrd** + **Copy**> (list: &[**T**]) -> **T** {

  let mut smallest = list[0];

  for &item in list {

    if item < smallest {

      smallest = item;

    }

  }

  smallest

}

fn **main**(){

  let number\_list = **vec!**[32,50,25,100,65];

  let result = **smallest**(&number\_list);

**println!**("the smallest number is {}", result);

}

**End of assignment 2**