**Practical: 1**

**Aim: Explore Control flow and looping**

**Code:**

use std::io;

fn main() {

let mut num = String::new();

println!("Enter a number:");

io::stdin().read\_line(&mut num).unwrap();

let num: u32 = num.trim().parse().unwrap();

match is\_prime(num) {

true => {

println!("{} is prime", num);

}

false => {

println!("{} is not prime", num);

}

}

}

fn is\_prime(num: u32) -> bool {

if num <= 1 {

return false;

}

let mut counter = 0;

for y in 1..num + 1 {

if num % y == 0 {

counter += 1;

//println!("{} is divisible by {}", num, y);

}

}

//println!("{}", counter);

if counter == 2 {

true

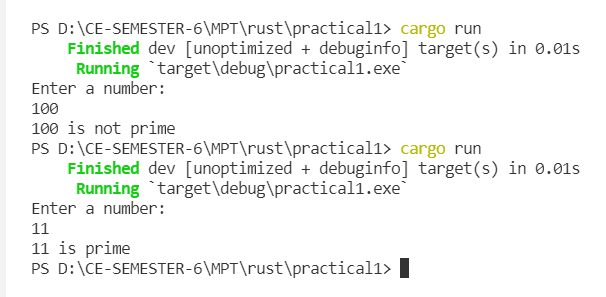
} else {

false

}

}

**Output:**



**Practical: 2**

**Aim: Explore slice type.**

**Code:**

fn main() {

let name = String::from("Utsav Balar");

let bytes = name.as\_bytes();

let mut idx = 0;

for (i, &item) in bytes.iter().enumerate() {

if item == b' ' {

idx = i;

}

}

let first\_name = &name[..idx];

let last\_name = &name[idx + 1..];

println!("First Name: {} \nLast Name: {}", first\_name, last\_name);

println!("Full name: {}", combine\_name(&first\_name, &last\_name));

}

fn combine\_name(first\_name: &str, last\_name: &str) -> String {

let mut full\_name = String::new();

full\_name.push\_str(first\_name);

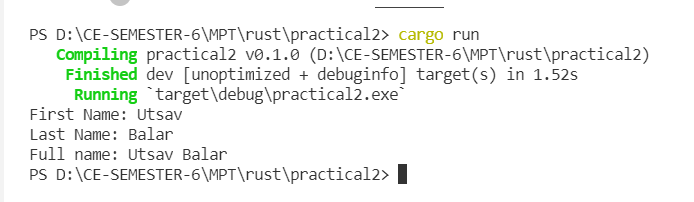
full\_name.push\_str(" ");

full\_name.push\_str(last\_name);

full\_name

}

**Output:**

****

**Practical: 3**

**Aim: Explore referencing and borrowing.**

**Code:**

fn main() {

let owner = String::from("Utsav Balar");

let new\_owner = give\_me\_ownership(owner);

// println!("Owner: {}", owner); <- Error: owner has been moved

println!("{}", new\_owner);

let owner = String::from("Utsav Balar");

let new\_owner = give\_me\_ownership\_without\_move(&owner); // <- Pass owner by reference

println!("{}", owner); // <- Works fine

println!("{}", new\_owner);

let name = &owner; // <- borrow owner

println!("{:#?}", split\_name(&name)); // <- pass name by reference

}

fn split\_name(name: &str) -> (&str, &str) {

let bytes = name.as\_bytes();

let mut idx = 0;

for (i, &item) in bytes.iter().enumerate() {

if item == b' ' {

idx = i;

}

}

let first\_name = &name[..idx];

let last\_name = &name[idx + 1..];

(first\_name, last\_name)

}

fn give\_me\_ownership\_without\_move(owner: &String) -> String {

owner.to\_string()

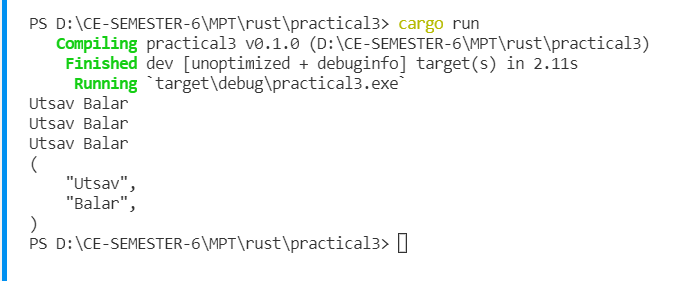
}

fn give\_me\_ownership(owner: String) -> String {

owner

}

**Output:**



**Practical: 4**

**Aim: Explore enum.**

**Code:**

#[allow(dead\_code)]

enum Payment {

Cash(f64),

Card(String, u8),

Crypto { currency: String, amount: f64 },

UPI(UpiData),

}

struct UpiData {

upi\_id: String,

phone\_number: String,

}

fn main() {

let payment = Payment::Card(String::from("Utsav Balar"), 12);

match payment {

Payment::Cash(amount) => println!("Cash Payment: {}", amount),

Payment::Card(name, amount) => {

println!("Card Owner: {}, Card Payment: {}", name, amount)

}

Payment::Crypto { currency, amount } => {

println!("Crypto Currency: {}, Amount: {}", currency, amount)

}

Payment::UPI(data) => println!(

"UPI ID: {}, Phone Number: {}",

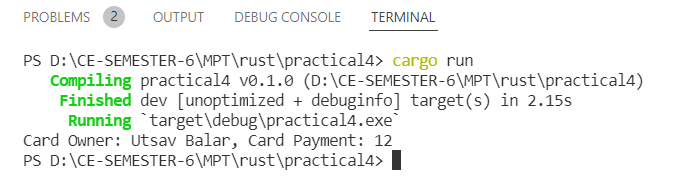
data.upi\_id, data.phone\_number

),

}

}

**Output:**

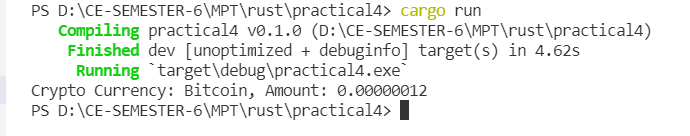
****

let payment = Payment::Crypto {

currency: String::from("Bitcoin"),

amount: 0.00000012,

};

****