**National University of Technology**

**Computer Science Department**

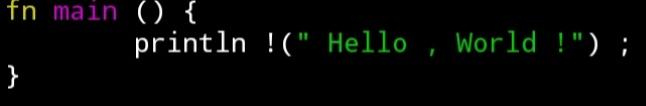
**Program:** Artificial intelligence

**Course:** PFAI (LAB)

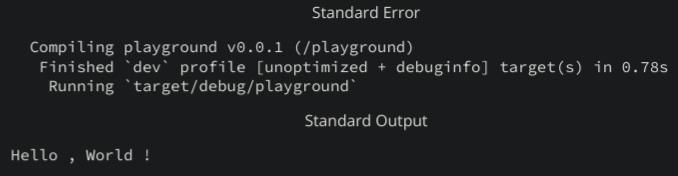
**Lab Report – 07**

|  |  |
| --- | --- |
| Submitted To: | Submitted By: |
| Lec. Umer Aftab | Muhammad Imran – F23607042 |
|  | Inshal Bint-e-Adil – F23607038 |

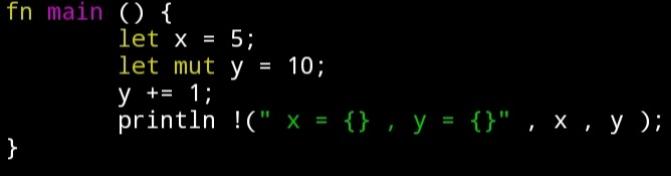
**Exercise 1: Hello, World!**

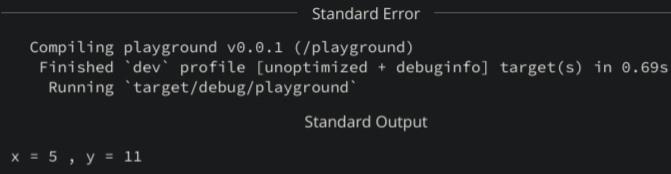
**CODE**

* fn main() defines the main function where the program starts running.
* Inside it, println!("Hello, World!"); prints the text "Hello, World!" to the console.
* println! is a macro in Rust used to print output.

**OUTPUT**

**Exercise 2: Variables & Mutability**

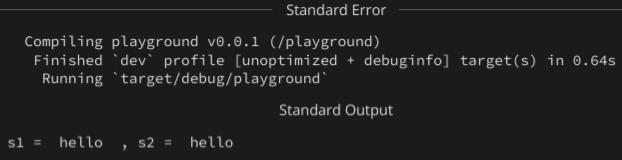
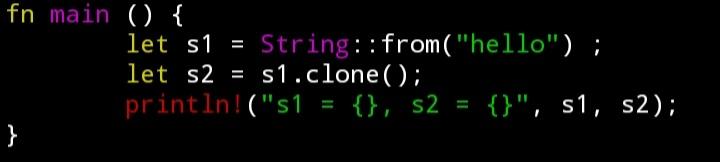
**CODE**

**OUTPUT**

This Rust program does the following:

* fn main() starts the main function.
* let x = 5; creates a constant variable x with value 5.
* let mut y = 10; creates a mutable variable y with value 10 (meaning y can be changed).
* y += 1; increases y by 1, making y = 11.
* println!("x = {}, y = {}", x, y); prints the values of x and y.

**Exercise 3: Ownership & Borrowing**

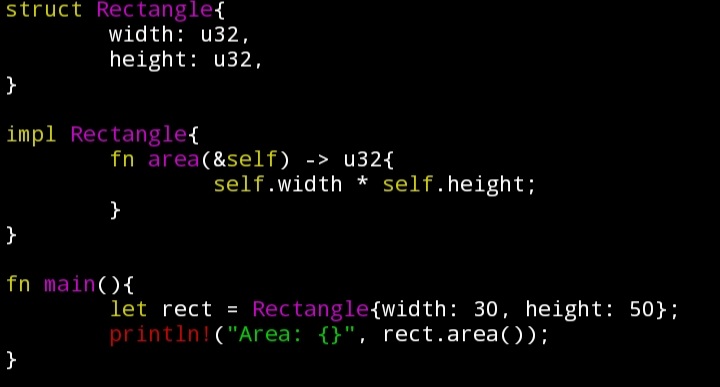
**CODE **

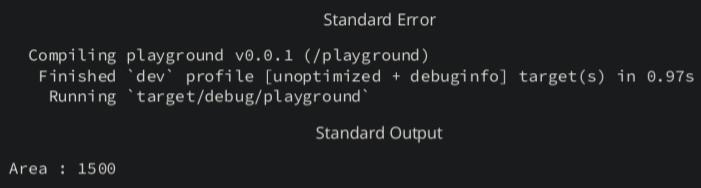
**OUTPUT**

This Rust program works like this:

* fn main() defines the main function.
* let s1 = String::from("hello"); creates a String named s1.
* let s2 = s1.clone(); makes a copy (clone) of s1 into s2.
* println!("s1 = {}, s2 = {}", s1, s2); prints both s1 and s2.

**Exercise 4: Structs & Methods**

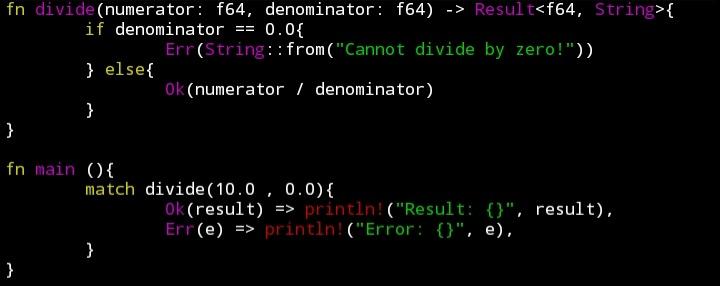
**CODE **

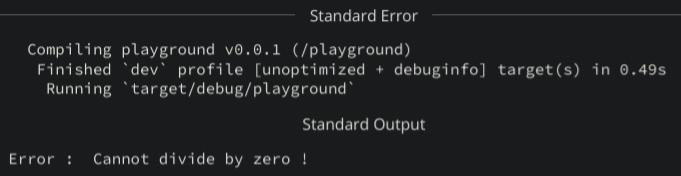
**OUTPUT**

This Rust program defines and uses a Rectangle structure:

* struct Rectangle defines a structure with width and height fields of type u32.
* impl Rectangle implements methods for Rectangle.
* fn area(&self) -> u32 defines a method that calculates and returns the area (width \* height).
* In main(), a Rectangle instance rect is created with width 30 and height 50.
* println!("Area: {}", rect.area()); prints the area, which will be 1500.

**Exercise 5: Error Handling**

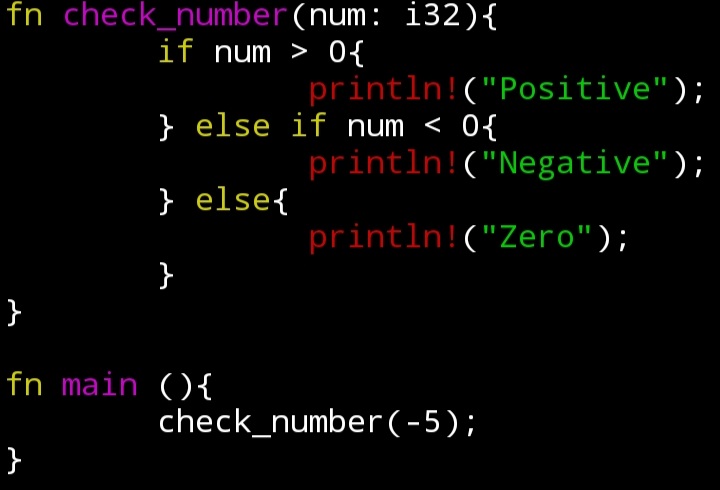
**CODE **

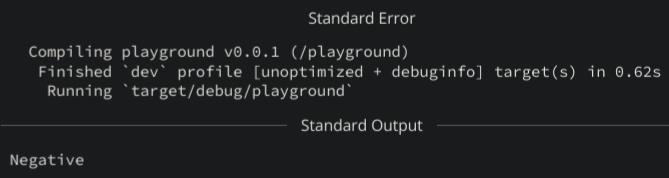
**OUTPUT**

This Rust program safely handles division:

* fn divide(numerator: f64, denominator: f64) -> Result<f64, String> defines a function that returns either a division result (Ok) or an error message (Err) if dividing by zero.
* If denominator == 0.0, it returns an error Err("Cannot divide by zero!").
* Otherwise, it returns the division result inside Ok.
* In main(), match is used to check if the division was successful (Ok) or if there was an error (Err), and prints the result accordingly.

**Exercise 6: Functions & Control Flow**

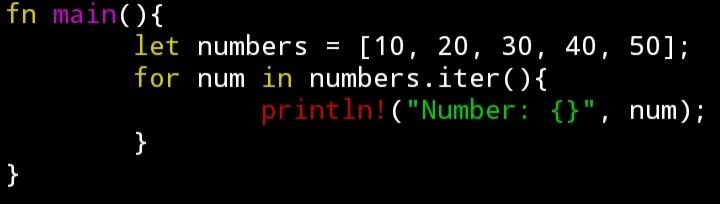
**CODE**

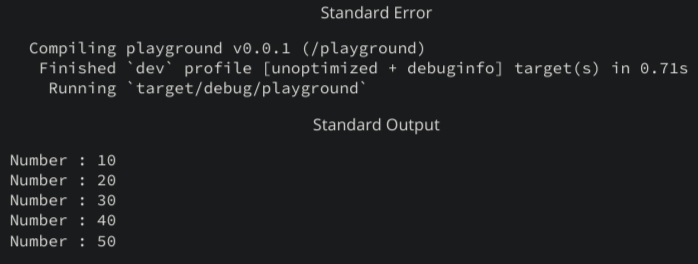
**OUTPUT**

This Rust program checks if a number is positive, negative, or zero:

* fn check\_number(num: i32) defines a function that takes an integer num.
* It uses if, else if, and else to check:
  + If num > 0, it prints "Positive".
  + If num < 0, it prints "Negative".
  + Otherwise (if num == 0), it prints "Zero".
* In main(), it calls check\_number(-5), so it will print "Negative".

**Exercise 7: Arrays & Loops**

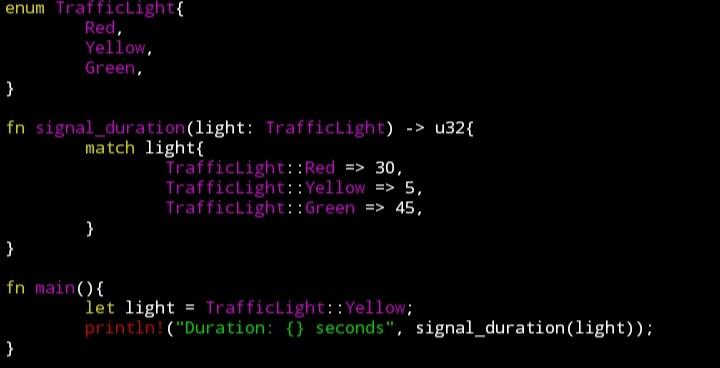
**CODE **

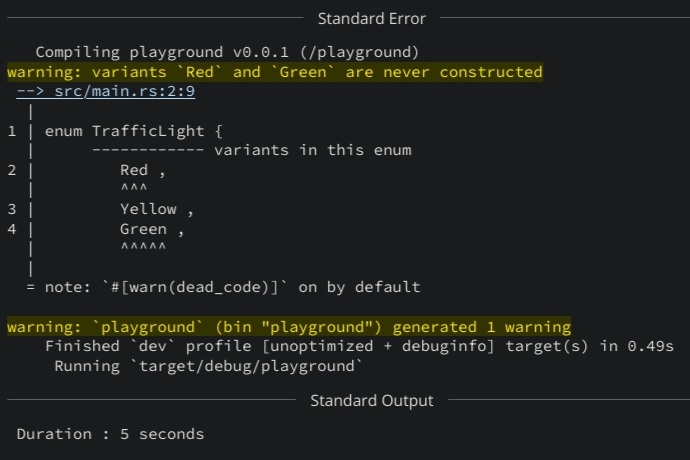
**OUTPUT**

This Rust program prints each element of an array:

* let numbers = [10, 20, 30, 40, 50]; creates an array of integers.
* for num in numbers.iter() loops through each element using .iter().
* Inside the loop, println!("Number: {}", num); prints each number one by one.

**Exercise 8: Enums & Pattern Matching**

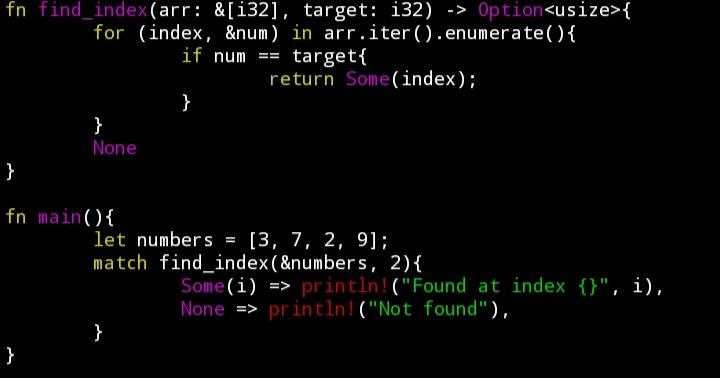
**CODE**

**OUTPUT**

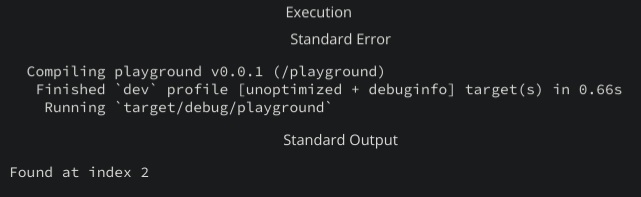
This Rust program uses an enum to represent traffic light colors:

* enum TrafficLight defines three possible values: Red, Yellow, and Green.
* fn signal\_duration(light: TrafficLight) -> u32 takes a TrafficLight value and returns how long (in seconds) the light stays on, using match.
  + Red lasts 30 seconds, Yellow 5 seconds, and Green 45 seconds.
* In main(), it sets light to Yellow and prints its duration (5 seconds).

**Exercise 9: The Option Type**

**CODE**

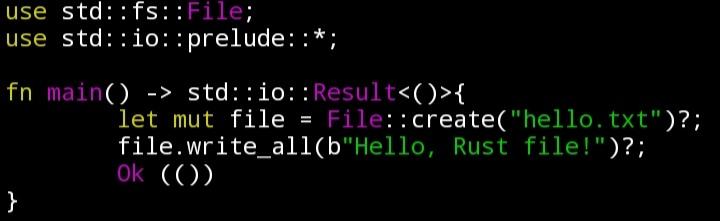
**OUTPUT**

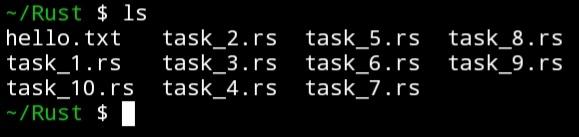
****

This Rust program searches for a number in an array:

* fn find\_index(arr: &[i32], target: i32) -> Option<usize> looks for target in arr.
* It loops through the array with enumerate() to get both the index and value.
* If it finds the target, it returns Some(index).
* If not found after the loop, it returns None.
* In main(), it looks for 2 in [3, 7, 2, 9].
* Since 2 is found at index 2, it prints "Found at index 2".

**Exercise 10: Basic File I/O**

**CODE**

**OUTPUT**

* use std::fs::File; and use std::io::prelude::\*; import necessary modules for file operations.
* fn main() -> std::io::Result<()> means main can return an input/output (I/O) result.
* File::create("hello.txt")?; creates a new file named hello.txt. The ? handles errors automatically.
* file.write\_all(b"Hello, Rust file!")?; writes the text into the file.