//Exercise 1

// Complete this function to return the bigger number!

// Do not use:

// - another function call

// - additional variables

pub fn bigger(a:i32,b:i32)-> i32{

if a>b

{

return a

}

return b

}

//Exercise 2

// Input: Provide an arbitrary value of number

// Check number is Positive or Negative or Zero

// Output: &str

fn check\_number(number: i32) -> &'static str {

if number > 0 {

return "Positive";

} else if number < 0 {

return "Negative";

} else {

return "Zero";

}

}

// Exercise 3

// Step 1: Make me compile!

// Step 2: Get the bar\_for\_fuzz and default\_to\_baz tests passing!

pub fn foo\_if\_fizz(fizzish: &str) -> &str {

if fizzish == "fizz" {

return "foo";

}

else if fizzish=="literally anythingliterally anything"

{

return "bar";

}

else {

return "bar";

}

}

// Exercise 4

// Determine if a given year is a leap year

// Implement logic

fn is\_leap\_year(year: i32) -> bool {

if year % 4 != 0 {

return false;

} else if year % 100 != 0 {

return true;

} else if year % 400 != 0 {

return false;

}

true

}

// Exercise 5

// Calculate the factorial of a number

// Implement logic

fn factorial(n: u32) -> u32 {

if n == 0 {

1

} else {

n \* factorial(n - 1)

}

}

// Exercise 6

// Check if a number is prime

// Implement logic

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

if n <= 1 {

return false;

}

let sqrt\_n = (n as f64).sqrt() as u32;

for i in 2..=sqrt\_n {

if n % i == 0 {

return false;

}

}

true

}

// Don't mind this for now :)

#[cfg(test)]

mod tests {

use super::\*;

// Test for exercise 1

#[test]

fn ten\_is\_bigger\_than\_eight() {

assert\_eq!(10, bigger(10, 8));

}

// Test for exercise 1

#[test]

fn fortytwo\_is\_bigger\_than\_thirtytwo() {

assert\_eq!(42, bigger(32, 42));

}

// Test for exercise 2

#[test]

fn test\_check\_number\_positive() {

let result = check\_number(10);

assert\_eq!(result, "Positive");

}

// Test for exercise 2

#[test]

fn test\_check\_number\_negative() {

let result = check\_number(-5);

assert\_eq!(result, "Negative");

}

// Test for exercise 2

#[test]

fn test\_check\_number\_zero() {

let result = check\_number(0);

assert\_eq!(result, "Zero");

}

// Test for exercise 3

#[test]

fn foo\_for\_fizz() {

assert\_eq!(foo\_if\_fizz("fizz"), "foo")

}

// Test for exercise 3

#[test]

fn bar\_for\_fuzz() {

assert\_eq!(foo\_if\_fizz("fuzz"), "bar")

}

// Test for exercise 3

#[test]

fn default\_to\_baz() {

assert\_eq!(foo\_if\_fizz("literally anything"), "baz")

}

// Test for exercise 4

#[test]

fn test\_leap\_year() {

assert\_eq!(is\_leap\_year(2020), true);

assert\_eq!(is\_leap\_year(2000), true);

assert\_eq!(is\_leap\_year(1600), true);

}

// Test for exercise 4

#[test]

fn test\_non\_leap\_year() {

assert\_eq!(is\_leap\_year(2021), false);

assert\_eq!(is\_leap\_year(1900), false);

assert\_eq!(is\_leap\_year(1800), false);

}

// Test for exercise 5

#[test]

fn test\_factorial() {

assert\_eq!(factorial(0), 1);

assert\_eq!(factorial(1), 1);

assert\_eq!(factorial(5), 120);

assert\_eq!(factorial(10), 3628800);

}

// Test for exercise 6

#[test]

fn test\_prime\_number() {

assert\_eq!(is\_prime(2), true);

assert\_eq!(is\_prime(7), true);

assert\_eq!(is\_prime(13), true);

assert\_eq!(is\_prime(19), true);

}

// Test for exercise 6

#[test]

fn test\_non\_prime\_number() {

assert\_eq!(is\_prime(1), false);

assert\_eq!(is\_prime(4), false);

assert\_eq!(is\_prime(10), false);

assert\_eq!(is\_prime(15), false);

}

}