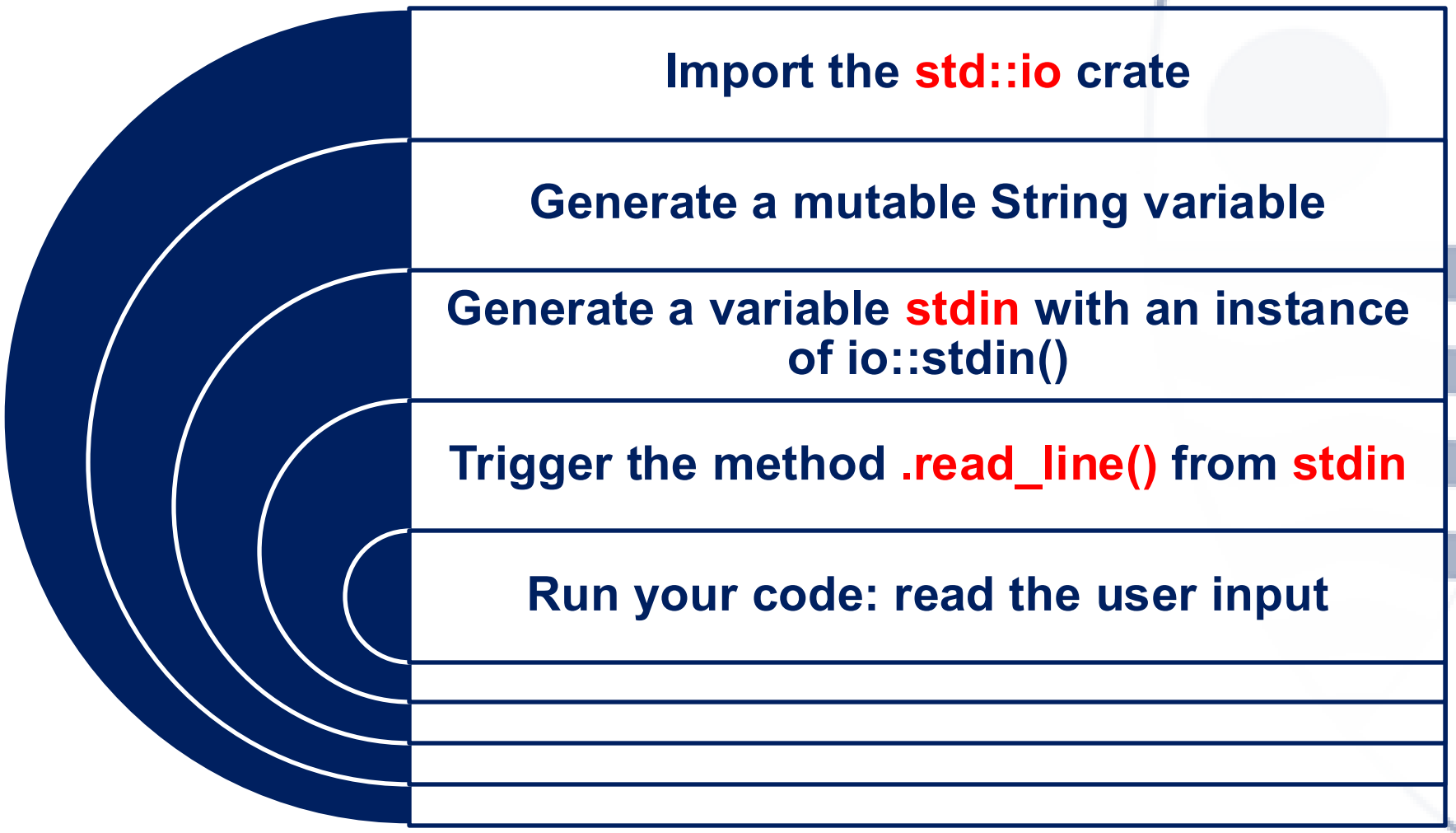
Rust - Decision Making & Loops

Week 4

SCHOOL OF
SCIENCE AND
TECHNOLOGY
PAN-ATLANTIC UNIVERSITY

How to Read User Input in Rust



Rust - Standard Input

Practice 1: week-4/practice_1/src/main.rs

```rust
// Rust program to output name and age

use std::io;

fn main() {
    println!("\nStudent Information Management System!");

    // input name
    println!("\nPlease Enter your name.");
    let mut name = String::new();
        io::stdin()
        .read_line(&mut name)
        .expect("Failed to read input");
    println!("Your name is: {}", name);

    // input age
    println!("\nEnter your age.");
    let mut age = String::new();
        io::stdin().read_line(&mut age).expect("Failed to read input");
    let age:i32 = age.trim().parse().expect("Input not an integer");
    println!("Your age is: {}", age);
}
```

Another Standard Input illustration:

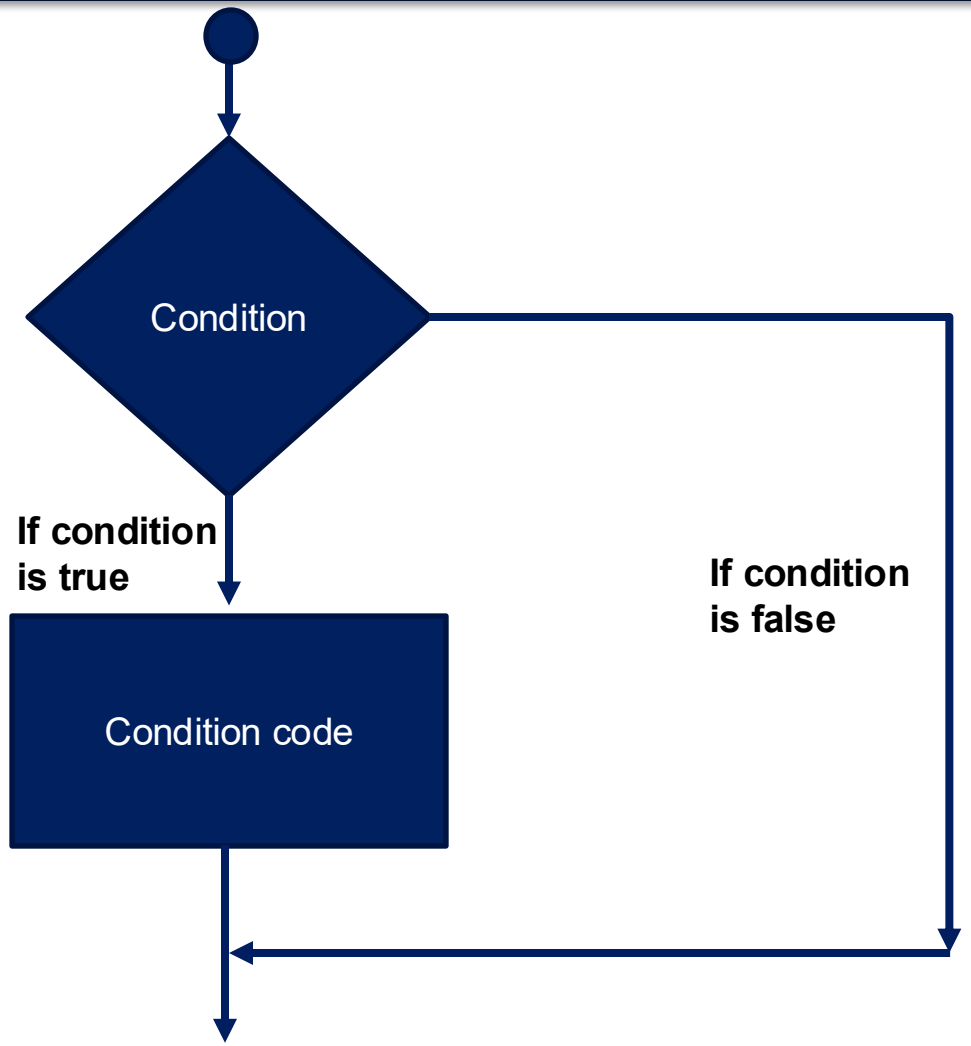
Practice 2: *week-4/practice_2/src/main.rs*

```rust
// Rust program to calculate the area of a triangle given three sides

use std::io;

fn main()
{
    let mut input1 = String::new();
    let mut input2 = String::new();
    let mut input3 = String::new();

    println!("Enter first edge of triangle: ");
    io::stdin().read_line(&mut input1).expect("Not a valid string");
    let a:f32 = input1.trim().parse().expect("Not a valid number");

    println!("Enter second edge of triangle: ");
    io::stdin().read_line(&mut input2).expect("Not a valid string");
    let b:f32 = input2.trim().parse().expect("Not a valid number");

    println!("Enter third edge of triangle: ");
    io::stdin().read_line(&mut input3).expect("Not a valid string");
    let c:f32 = input3.trim().parse().expect("Not a valid number");

    let s:f32 = (a + b + c) / 2.0;
    let mut area:f32 = s * (s - a) * (s - b) * (s - c);
    area = area.sqrt();

    println!("Area of a triangle: {}", area);
}
```

Rust – Decision Making

Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Rust – Decision Making



Rust – Decision Making

Sr.No	Statement & Description
1	**if statement** An *if* statement consists of a Boolean expression followed by one or more statements.
2	**if...else statement** An *if* statement can be followed by an optional *else* statement, which executes when the Boolean expression is false.
3	**else...if and nested ifstatement** You can use one *if* or *else if* statement inside another *if* or *else if* statement(s).
4	**match statement** A *match* statement allows a variable to be tested against a list of values.

Rust – If Statement

The *if…else* construct evaluates a condition before a block of code is executed.

Syntax:

```
if boolean_expression {
   // statement(s) will execute if the boolean expression is true
}
```

If the Boolean expression evaluates to true, then the block of code inside the if statement will be executed. If the Boolean expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

If Statement illustration:

Practice 3: week-4/practice_3/src/main.rs

```rust
// Rust program to calculate the area of a
// triangle for a given base and height

use std::io;

fn main()
{
    let mut input1 = String::new();
    let mut input2 = String::new();

    println!("Enter base: ");
    io::stdin().read_line(&mut input1).expect("Not a valid string");
    let base:f32 = input1.trim().parse().expect("Not a valid number");

    println!("Enter height: ");
    io::stdin().read_line(&mut input2).expect("Not a valid string");
    let height:f32 = input2.trim().parse().expect("Not a valid number");

    if base > 0.0 {
        let area:f32 =(base * height) / 2.0;
        println!("Area of a triangle: {}", area);
    }
}
```

Rust – If…else Statement

An **if** can be followed by an optional **else** block. The else block will execute if the Boolean expression tested by the if statement evaluates to false.

Syntax:

```
if boolean_expression {
   // statement(s) will execute if the boolean expression is true
} else {
   // statement(s) will execute if the boolean expression is
false
}
```

- The **if** block guards the conditional expression. The block associated with the if statement is executed if the Boolean expression evaluates to true.
- The if block may be followed by an optional else statement. The instruction block associated with the else block is executed if the expression evaluates to false.

If…else Statement illustration:

Practice 4: week-4/practice_4/src/main.rs

```rust
// Rust program to determine age pass

use std::io;

fn main() {

    let mut input1 = String::new();
    let mut input2 = String::new();

    println!("Enter your name: ");
    io::stdin().read_line(&mut input1).expect("Not a valid string");

    println!("Enter your age: ");
    io::stdin().read_line(&mut input2).expect("Not a valid string");
    let age:i32 = input2.trim().parse().expect("Not a valid number");

    if age >= 18 {
        println!("Welcome to the party {}!", input1);
    } else {
        println!("Oops, you are not of age to enter the party {}", input1);
    }
}
```

Nested If Statement

The **else…if** ladder is useful to test multiple conditions.

Syntax:

```
if boolean_expression1 {
    //statements if the expression1 evaluates to true
} else if boolean_expression2 {
   //statements if the expression2 evaluates to true
} else {
   //statements if both expression1 and expression2 result to false
}
```

When using if…else…if and else statements, there are a few points to keep in mind.

- An if can have zero or one else's and it must come after any else..if.
- An if can have zero to many else..if and they must come before the else.
- Once an else..if succeeds, none of the remaining else..if or else will be tested.

Nested If Statement

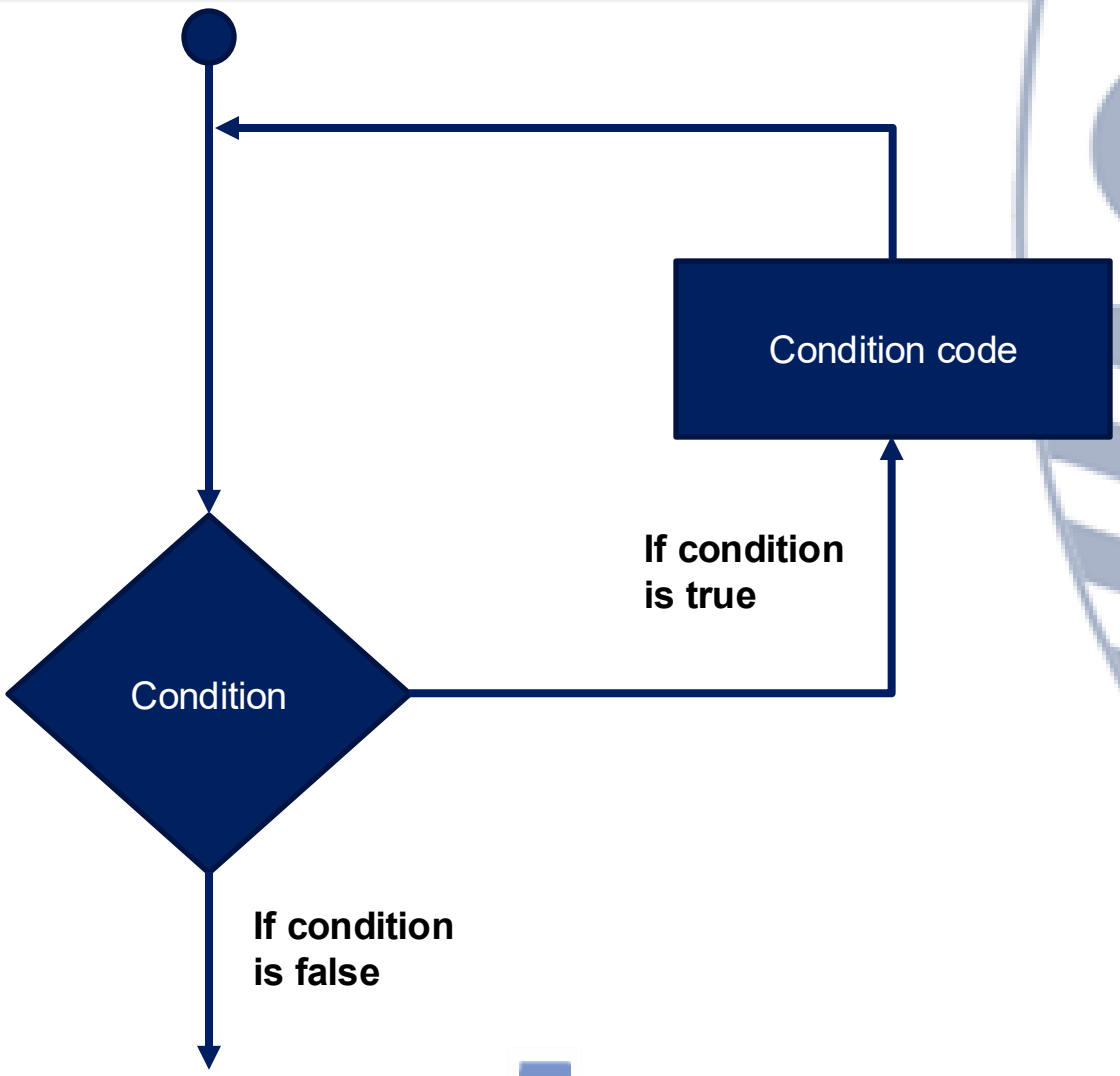
```rust
// Rust program to read the height of a person
// and then print if person is tall, dwarf,
// or average height person

use std::io;

fn main()
{
    let mut input = String::new();

    println!("\nEnter Your Height (in centimetres):");
    io::stdin().read_line(&mut input).expect("Not a valid string");
    let height:f32 = input.trim().parse().expect("Not a valid number");

    if height >= 150.0 && height <= 170.0
    {
        println!("You are of average height person");
    }
    else if height > 170.0 && height <= 195.0
    {
        println!("You are tall");
    }
    else if height < 150.0 && height > 100.0
    {
        println!("You are dwarf");
    }
    else
    {
        println!("Abnormal height");
    }
}
```

Practice 5: week-4/practice_5/src/main.rs

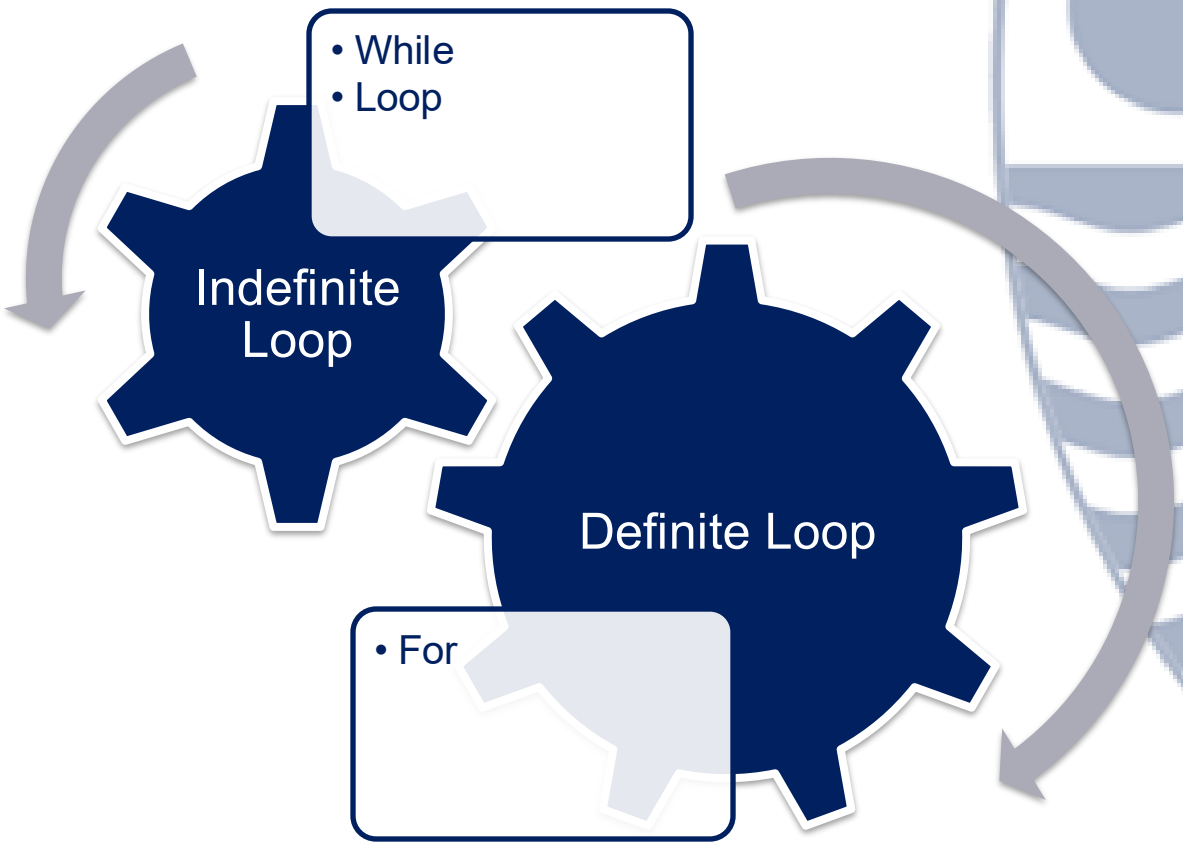
Loops

A loop statement allows us to execute a statement or group of statements multiple times. Programming languages provide various control structures that allow for more complicated execution paths. There may be instances, where a block of code needs to be executed repeatedly. In general, programming instructions are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

General form of a Loops



Types of Loops



Definite Loop

A loop the number of iterations of which is definite/fixed is termed as a definite loop. The **for** loop is an implementation of a definite loop.

For Loop

The for loop executes the code block for a specified number of times. It can be used to iterate over a fixed set of values, such as an array.

Syntax:

```
for temp_variable in lower_bound..upper_bound {
    //statements
}
```

For Loop illustration:

Practice 6: week-4/practice_6/src/main.rs

```rust
// Rust program to count numbers

use std::io;

fn main(){

    println!("Enter lower bound");
    let mut input1 = String::new();
    io::stdin().read_line(&mut input1).expect("Failed to read input");
    let lower_bound:i32 = input1.trim().parse().expect("Failed to input");

    println!("Enter upper bound");
    let mut input2 = String::new();
    io::stdin().read_line(&mut input2).expect("Failed to read input");
    let upper_bound:i32 = input2.trim().parse().expect("Failed to input");

    for x in lower_bound..upper_bound{ // upper_bound is not inclusive

      println!("Count Level is {}",x);
   }
}
```

Indefinite Loop

An indefinite loop is used when the number of iterations in a loop is indeterminate or unknown.
Indefinite loops can be implemented using:

Sr.No	Name & Description
1	While The while loop executes the instructions each time the condition specified evaluates to true
2	Loop The loop is a while(true) indefinite loop

While Loop illustration:

Practice 7: week-4/practice_7/src/main.rs

```rust
use std::io;

fn main() {

    println!("Enter a number");
    let mut input1 = String::new();
    io::stdin().read_line(&mut input1).expect("Failed to read input");
    let mut num:i32 = input1.trim().parse().expect("Failed to input");

    while num < 10 {

        println!("inside loop number value is {}",num);
        num+=1;
    }
    println!("outside loop number value is {}",num);
}
```

Loop illustration:

Practice 8: week-4/practice_8/src/main.rs

```rust
fn main(){

    //while true

    let mut x = 0;
    loop {
        x+=1;
        println!("x={}",x);

        if x==15 {
            break;
        }
    }
}
```

The **break** statement is used to take the control out of a construct. Using break in a loop causes the program to exit the loop.

Continue Statement illustration:

The **continue** statement skips the subsequent statements in the current iteration and takes the control back to the beginning of the loop. Unlike the break statement, the continue does not exit the loop. It terminates the current iteration and starts the subsequent iteration.

Practice 9: week-4/practice_9/src/main.rs

```
fn main() {

    let mut count = 0;

    for num in 1..21 {
        if num > 10 {
          println!("{:?}",num);
           continue;
        }
        count+=1;
    }
    println! (" The count of values greater than 10 (between 1 and 20) is: {} ",count);
    //outputs 10
}
```

Class Projects

SCHOOL OF
SCIENCE AND
TECHNOLOGY
PAN-ATLANTIC UNIVERSITY

Project I

Given the values of *a*, *b* and *c*, find the roots of a quadratic equation using Rust program.

Hints:

- Input the values from the keyboard
- Find the discriminant. The discriminant determines the number and nature of the roots.
- If the discriminant is positive, then there are two distinct roots.
- If the discriminant is zero, then there is exactly one real root.
- If the discriminant is negative, then there are no real roots.

Project II

Develop a program in Rust that takes as input the experience and age of an employee to determine the annual incentive, with the following criteria:

- If the employee is experienced and his/her age is equal to or more than 40, then the incentive of the employee is N1,560,000.
- If the employee is experienced and his/her age is equal to or more than 30 but less than 40, then the incentive should be N1,480,000.
- For experienced employee below 28 years of age the incentive should be N1,300,000 per month.
- For inexperienced employee the incentive should be N100,000.