Rust Strings & Operators

Week 6

SCHOOL OF SCIENCE AND TECHNOLOGY

PAN-ATLANTIC UNIVERSITY

Rust String

The String data type in Rust can be classified into the following:

String Literal(&str)

String Object(String)

String Literal

String literals (&str) are used when the value of a string is known at compile time.

String literals are a set of characters, which are hardcoded into a variable. For example, *let department="Computer Science ".*

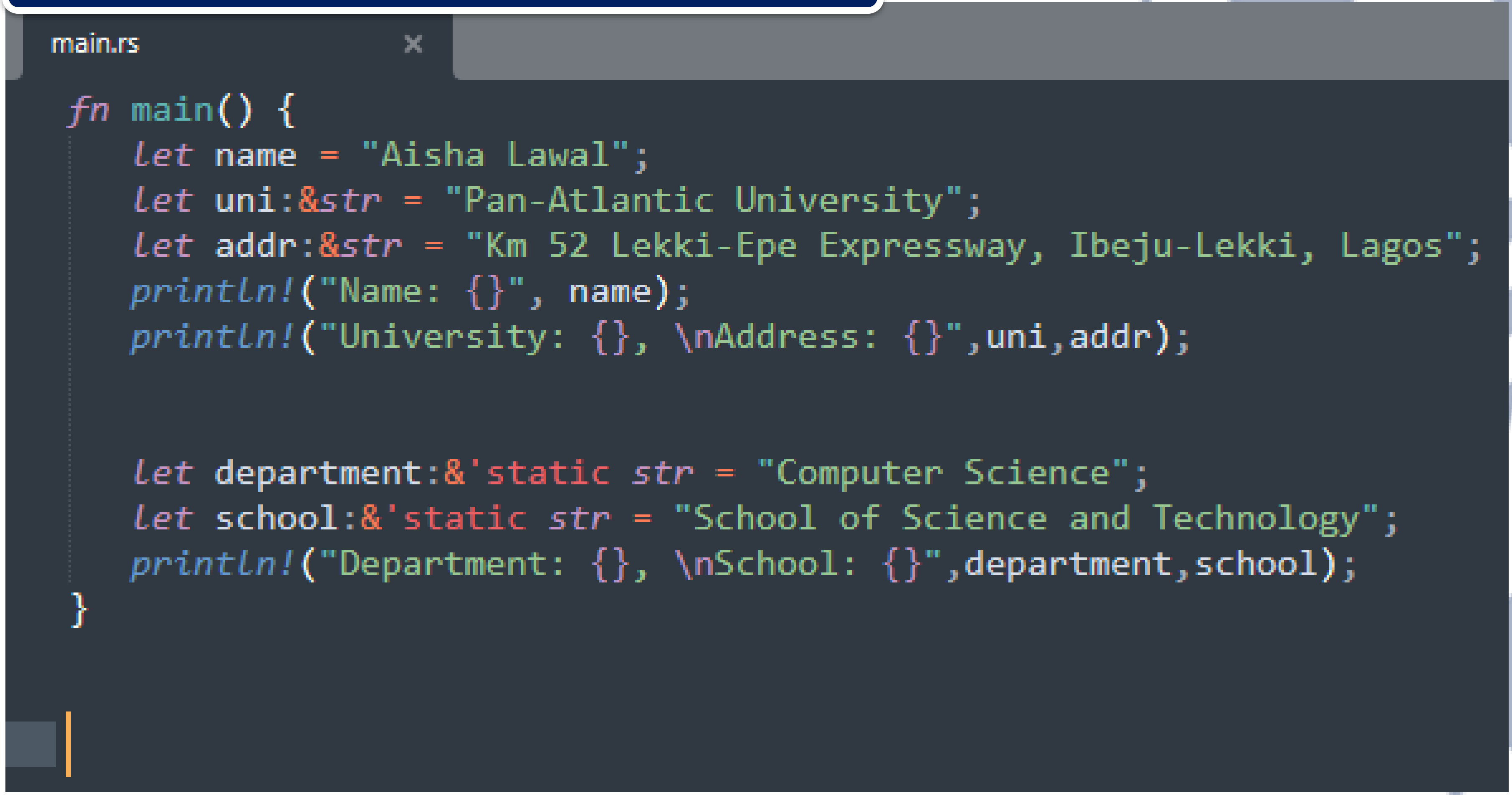
String literals are found in module std::str.

String literals are also known as string slices.

String literals are static by default. This means that string literals are guaranteed to be valid for the duration of the entire program.

String Literal

Practice 1: week-6/practice_1/src/main.rs



main.rs

```rust
fn main() {
    let name = "Aisha Lawal";
    let uni:&str = "Pan-Atlantic University";
    let addr:&str = "Km 52 Lekki-Epe Expressway, Ibeju-Lekki, Lagos";
    println!("Name: {}", name);
    println!("University: {}, \nAddress: {}",uni,addr);


    let department:&'static str = "Computer Science";
    let school:&'static str = "School of Science and Technology";
    println!("Department: {}, \nSchool: {}",department,school);
}
```

String Object

The String object type is provided in Standard Library. Unlike string literal, the string object type is not a part of the core language.

It is defined as public structure in standard library *pub struct String*. String is a growable collection.

It is mutable and UTF-8 encoded type.

The **String** object type can be used to represent string values that are provided at runtime. String object is allocated in the heap.

String Object Syntax

To create a String object, we can use any of the following syntax –

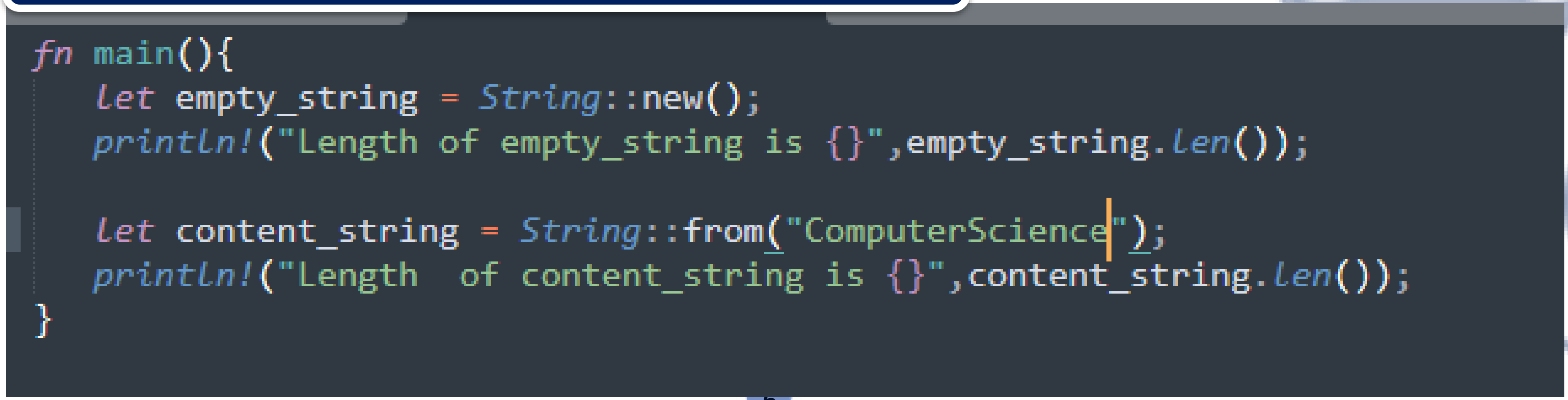
- **String::new()**

The above syntax creates an empty string

- **String::from()**

This creates a string with some default value passed as parameter to the **from()** method.

Practice 2: week-6/practice_2/src/main.rs



```
fn main(){
    let empty_string = String::new();
    println!("Length of empty_string is {}",empty_string.len());

    let content_string = String::from("ComputerScience");
    println!("Length  of content_string is {}",content_string.len());
}
```

Common Methods - String Object

Sr.No.	Method	Signature	Description
1	new()	pub const fn new() → String	Creates a new empty String.
2	to_string()	fn to_string(&self) → String	Converts the given value to a String.
3	replace()	pub fn replace<'a, P>(&'a self, from: P, to: &str) → String	Replaces all matches of a pattern with another string.
4	as_str()	pub fn as_str(&self) → &str	Extracts a string slice containing the entire string.
5	push()	pub fn push(&mut self, ch: char)	Appends the given char to the end of this String.
6	push_str()	pub fn push_str(&mut self, string: &str)	Appends a given string slice onto the end of this String.

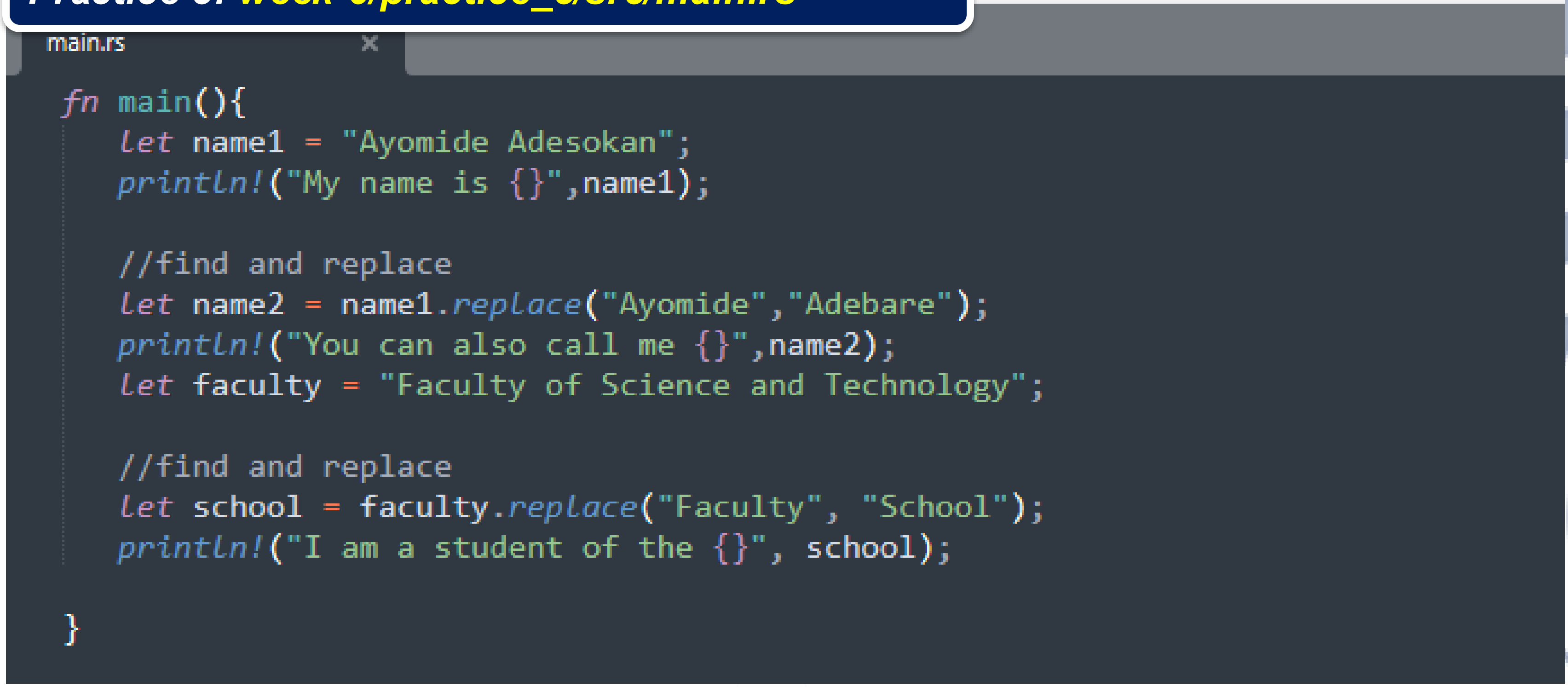
Common Methods - String Object

7	len()	pub fn len(&self) → usize	Returns the length of this String, in bytes.
8	trim()	pub fn trim(&self) → &str	Returns a string slice with leading and trailing whitespace removed.
9	split_whitespace()	pub fn split_whitespace(&self) → SplitWhitespace	Splits a string slice by whitespace and returns an iterator.
10	split()	pub fn split<'a, P>(&'a self, pat: P) → Split<'a, P> , where P is pattern can be &str, char, or a closure that determines the split.	Returns an iterator over substrings of this string slice, separated by characters matched by a pattern.
11	chars()	pub fn chars(&self) → Chars	Returns an iterator over the chars of a string slice.

Illustration: replace()

The ***replace()*** function takes two parameters – the first parameter is a string pattern to search for and the second parameter is the new value to be replaced.

Practice 3: week-6/practice_3/src/main.rs



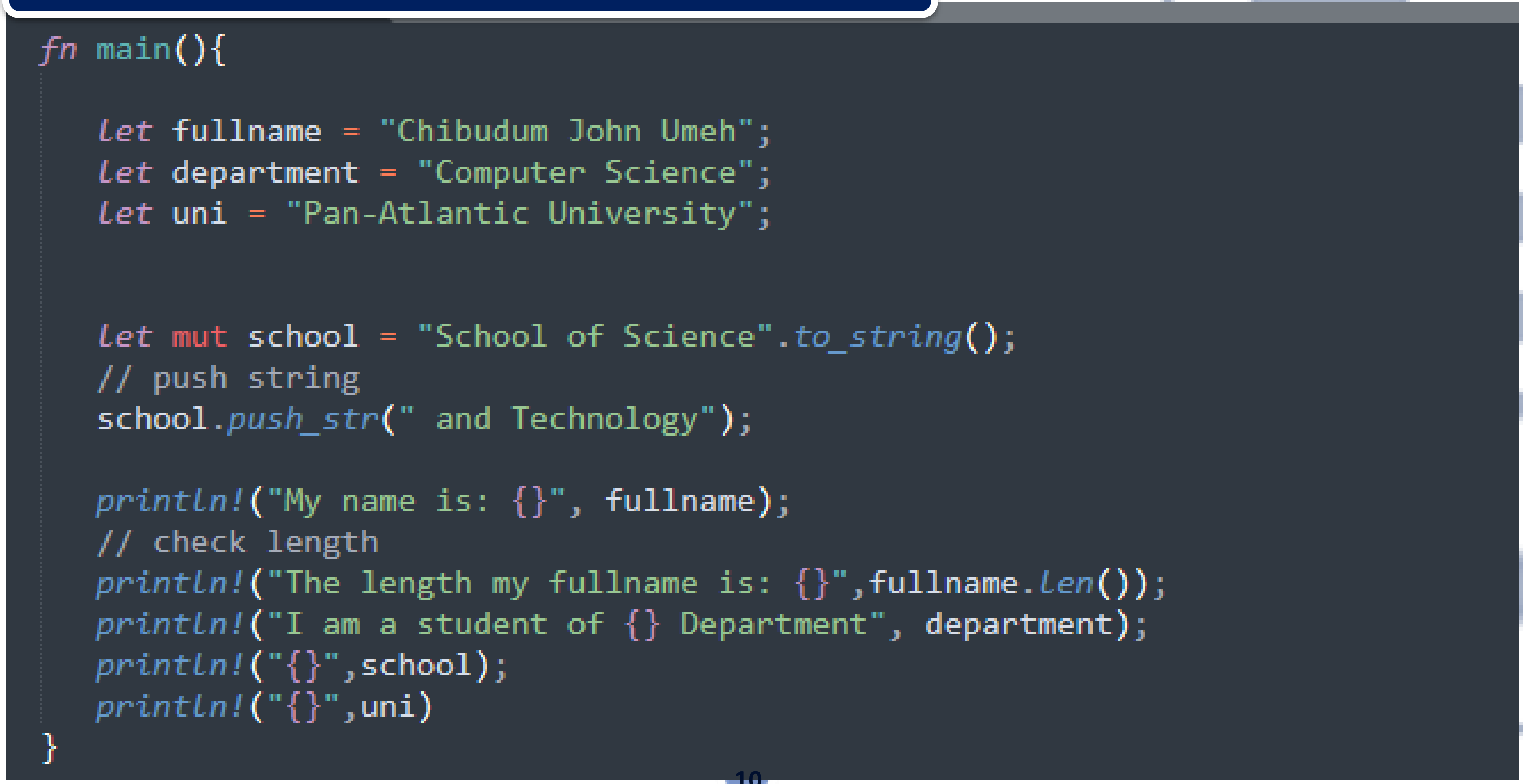
main.rs

```
fn main(){
    let name1 = "Ayomide Adesokan";
    println!("My name is {}",name1);

    //find and replace
    let name2 = name1.replace("Ayomide","Adebare");
    println!("You can also call me {}",name2);
    let faculty = "Faculty of Science and Technology";

    //find and replace
    let school = faculty.replace("Faculty", "School");
    println!("I am a student of the {}", school);

}
```

Illustration: push_str()

The **push_str()** function appends a given string slice onto the end of a String.

Practice 4: week-6/practice_4/src/main.rs

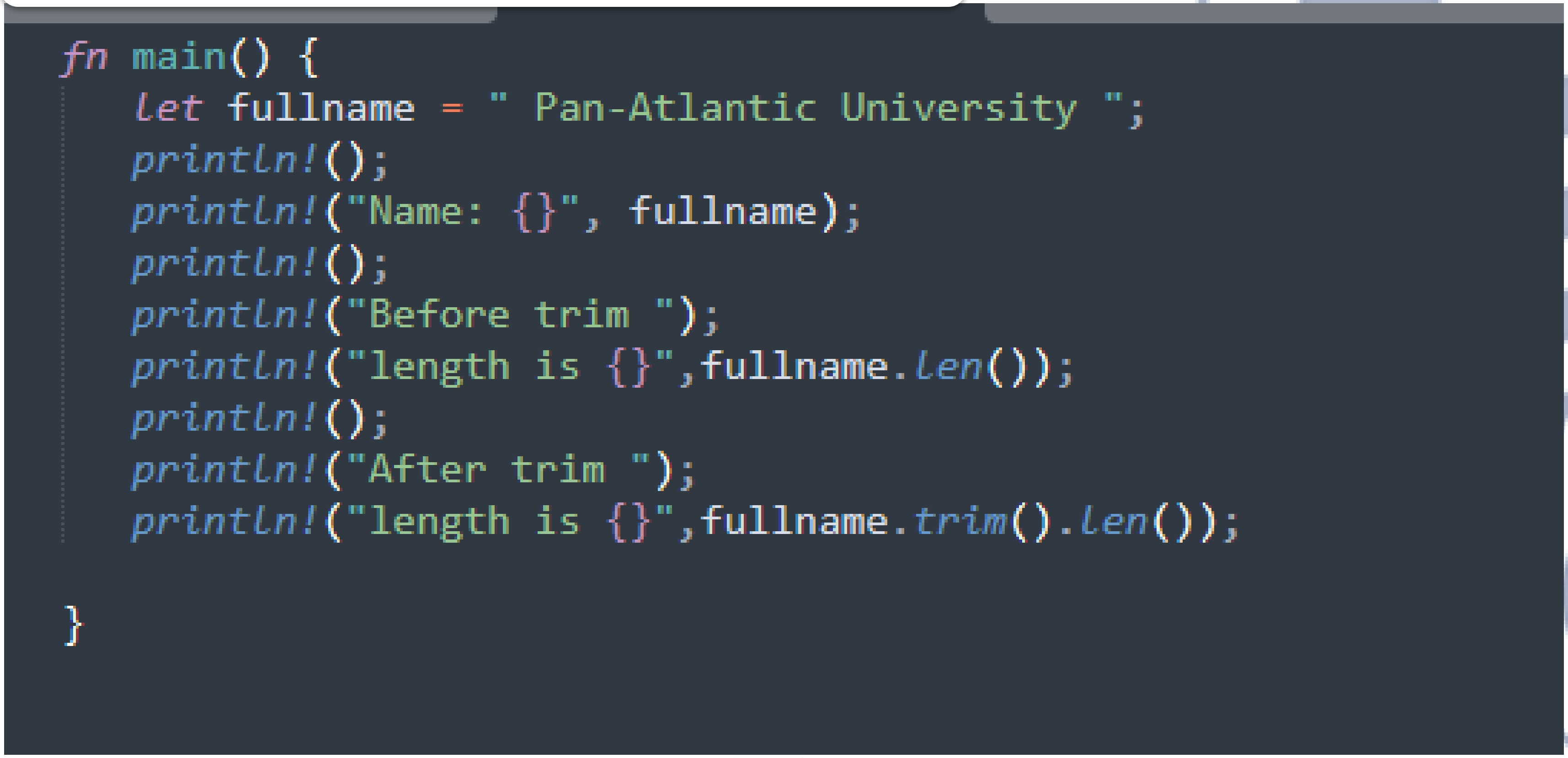


```rust
fn main(){

    let fullname = "Chibudum John Umeh";
    let department = "Computer Science";
    let uni = "Pan-Atlantic University";


    let mut school = "School of Science".to_string();
    // push string
    school.push_str(" and Technology");

    println!("My name is: {}", fullname);
    // check length
    println!("The length my fullname is: {}",fullname.len());
    println!("I am a student of {} Department", department);
    println!("{}",school);
    println!("{}",uni)
}
```

Illustration: trim()

The **trim()** function removes leading and trailing spaces in a string. NOTE that this function will not remove the inline spaces.

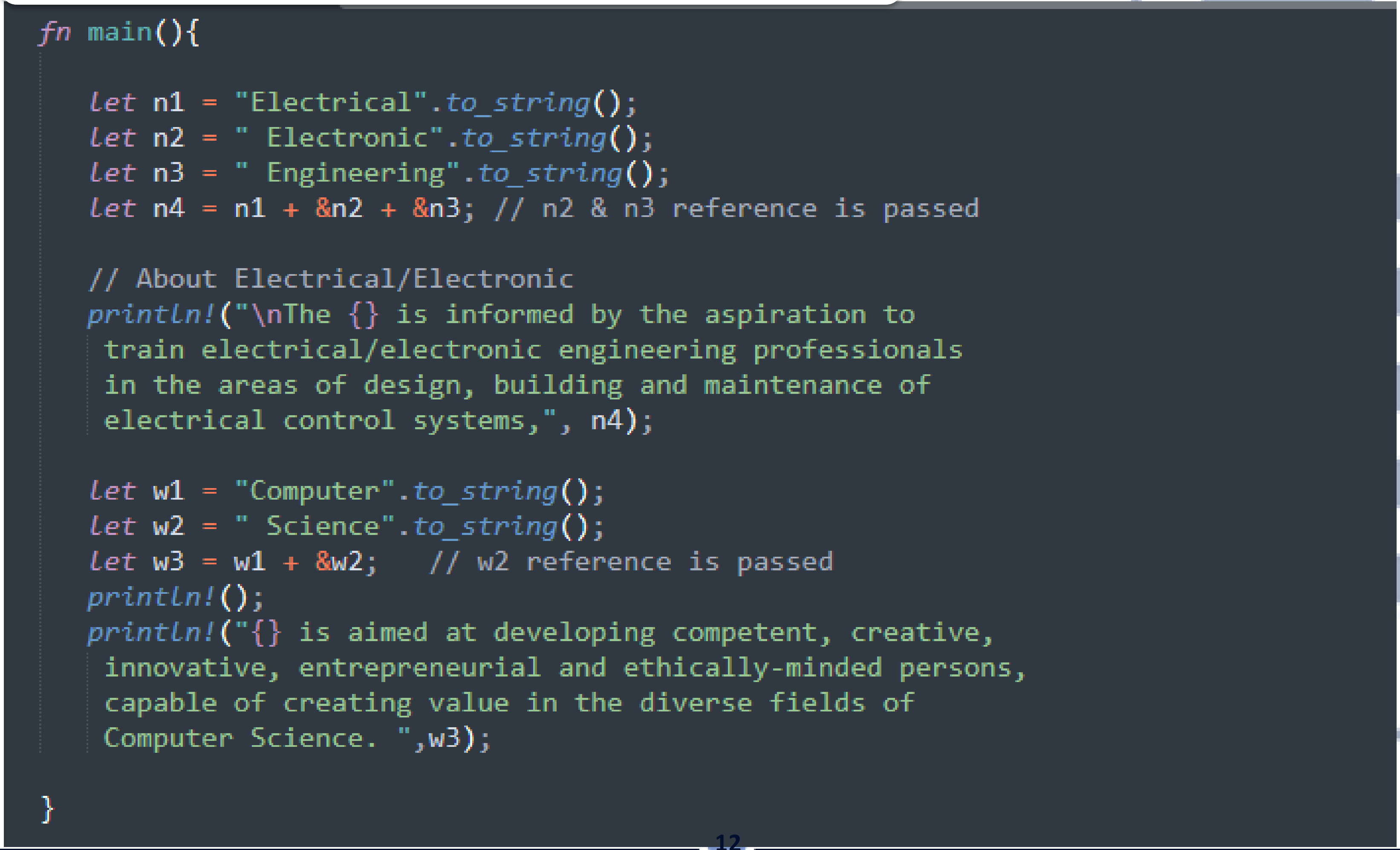
Practice 5: week-6/practice_5/src/main.rs



```rust
fn main() {
    let fullname = " Pan-Atlantic University ";
    println!();
    println!("Name: {}", fullname);
    println!();
    println!("Before trim ");
    println!("length is {}",fullname.len());
    println!();
    println!("After trim ");
    println!("length is {}",fullname.trim().len());

}
```

Illustration: String Concatenation

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

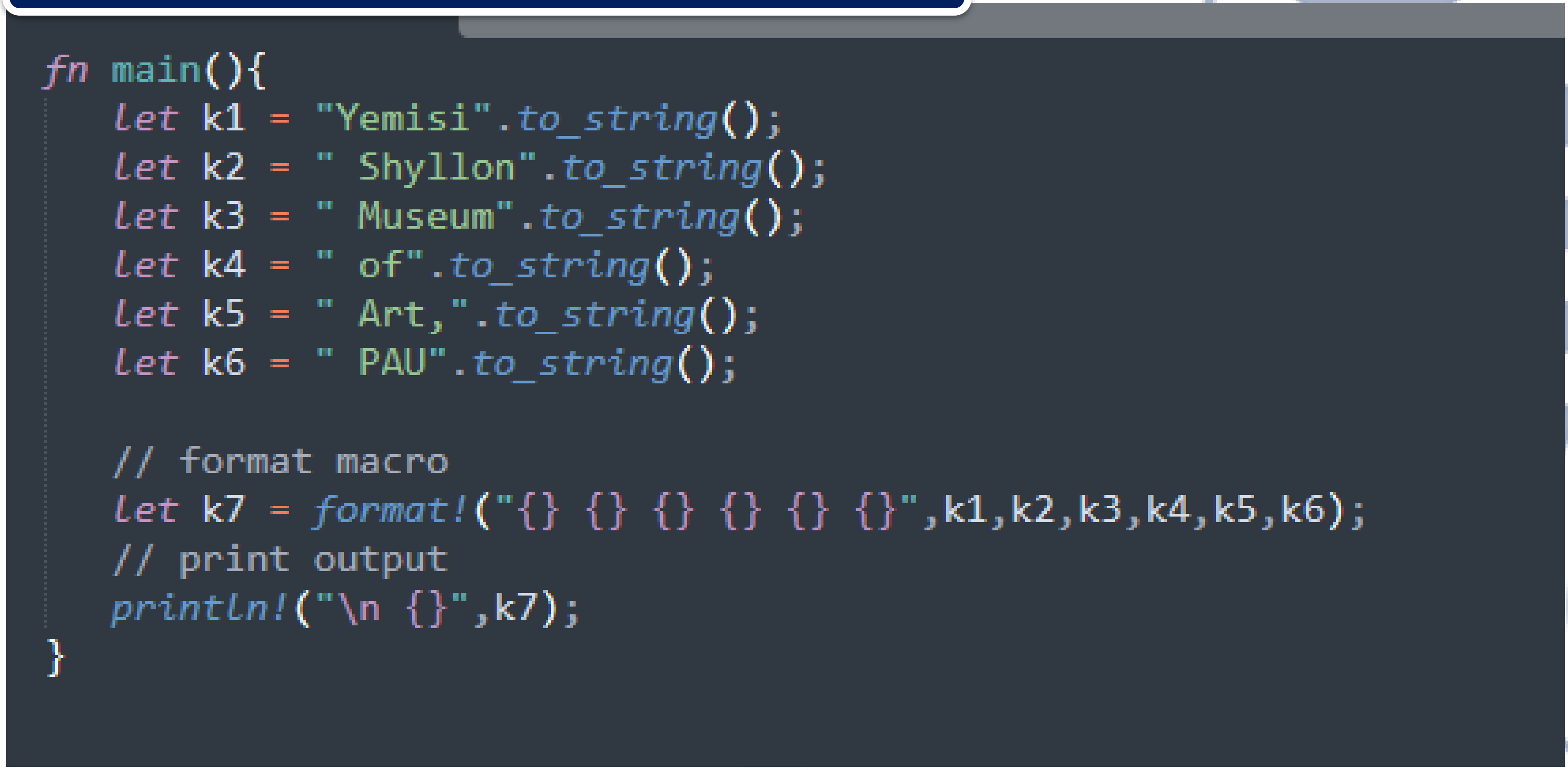


```
fn main(){

    let n1 = "Electrical".to_string();
    let n2 = " Electronic".to_string();
    let n3 = " Engineering".to_string();
    let n4 = n1 + &n2 + &n3; // n2 & n3 reference is passed

    // About Electrical/Electronic
    println!("\nThe {} is informed by the aspiration to
     train electrical/electronic engineering professionals
     in the areas of design, building and maintenance of
     electrical control systems,", n4);

    let w1 = "Computer".to_string();
    let w2 = " Science".to_string();
    let w3 = w1 + &w2;   // w2 reference is passed
    println!();
    println!("{} is aimed at developing competent, creative,
     innovative, entrepreneurial and ethically-minded persons,
     capable of creating value in the diverse fields of
     Computer Science. ",w3);

}
```

Illustration: Format! Macro

Another way to add to String objects together is using a macro function called format.

Practice 7: week-6/practice_7/src/main.rs



```rust
fn main(){
    let k1 = "Yemisi".to_string();
    let k2 = " Shyllon".to_string();
    let k3 = " Museum".to_string();
    let k4 = " of".to_string();
    let k5 = " Art,".to_string();
    let k6 = " PAU".to_string();

    // format macro
    let k7 = format!("{} {} {} {} {} {}",k1,k2,k3,k4,k5,k6);
    // print output
    println!("\n {}",k7);
}
```

Operator

An operator defines some function that will be performed on the data. The data on which operators work are called operands. Consider the following expression −

- 7 + 5 = 12

Here, the values 7, 5, and 12 are operands, while + and = are operators.

The major operators in Rust can be classified as −

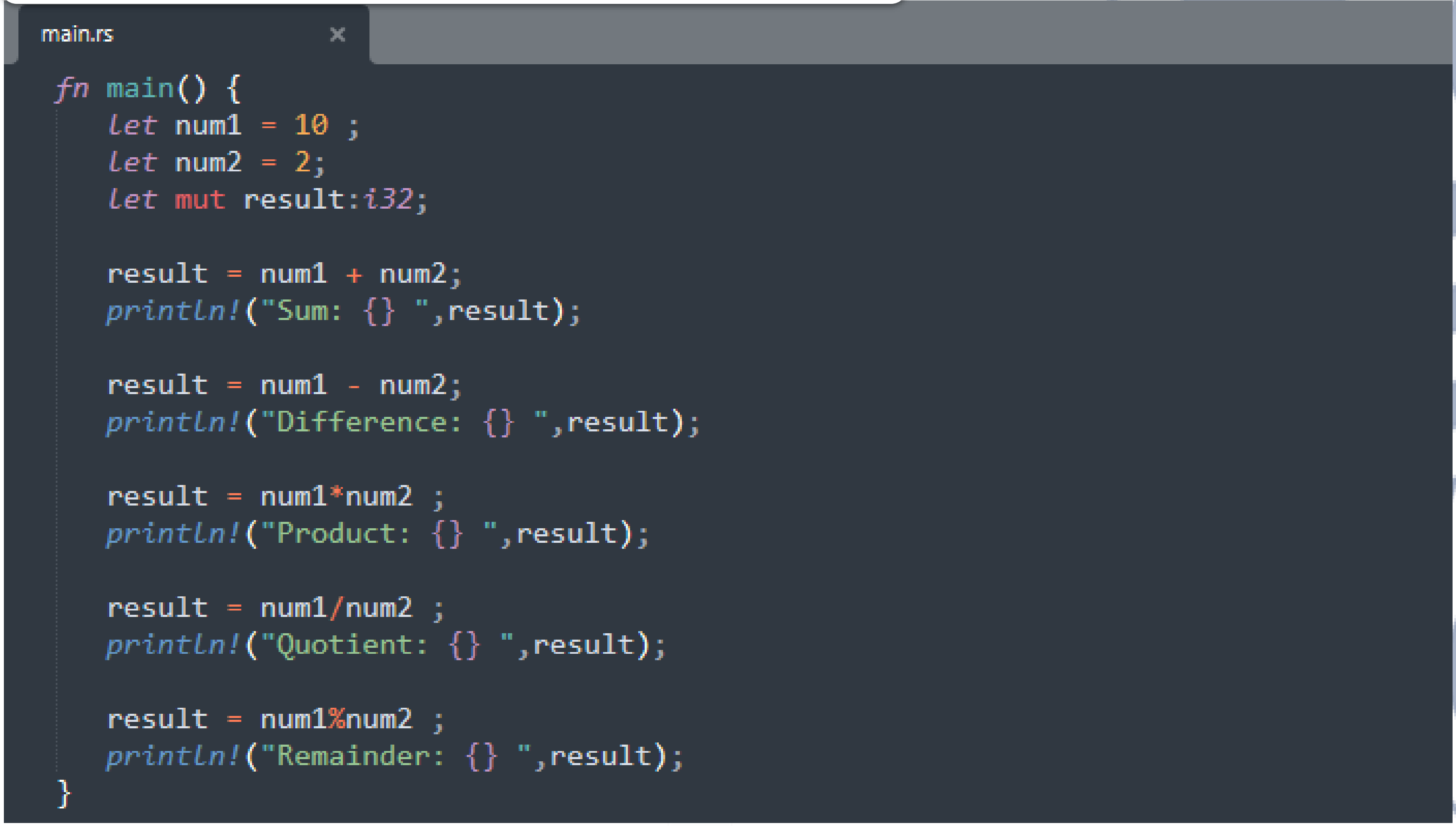
- Arithmetic
- Comparison
- Logical
- Bitwise
- Conditional

Arithmetic Operators

Sr.No	Operator	Description	Example
1	+(Addition)	returns the sum of the operands	a+b is 15
2	-(Subtraction)	returns the difference of the values	a-b is 5
3	* (Multiplication)	returns the product of the values	a*b is 50
4	/ (Division)	performs division operation and returns the quotient	a / b is 2
5	% (Modulus)	performs division operation and returns the remainder	a % b is 0

Arithmetic Operators illustration

***Practice 8:** week-6/practice_8/src/main.rs*



```rust
fn main() {
    let num1 = 10 ;
    let num2 = 2;
    let mut result:i32;

    result = num1 + num2;
    println!("Sum: {} ",result);

    result = num1 - num2;
    println!("Difference: {} ",result);

    result = num1*num2 ;
    println!("Product: {} ",result);

    result = num1/num2 ;
    println!("Quotient: {} ",result);

    result = num1%num2 ;
    println!("Remainder: {} ",result);
}
```

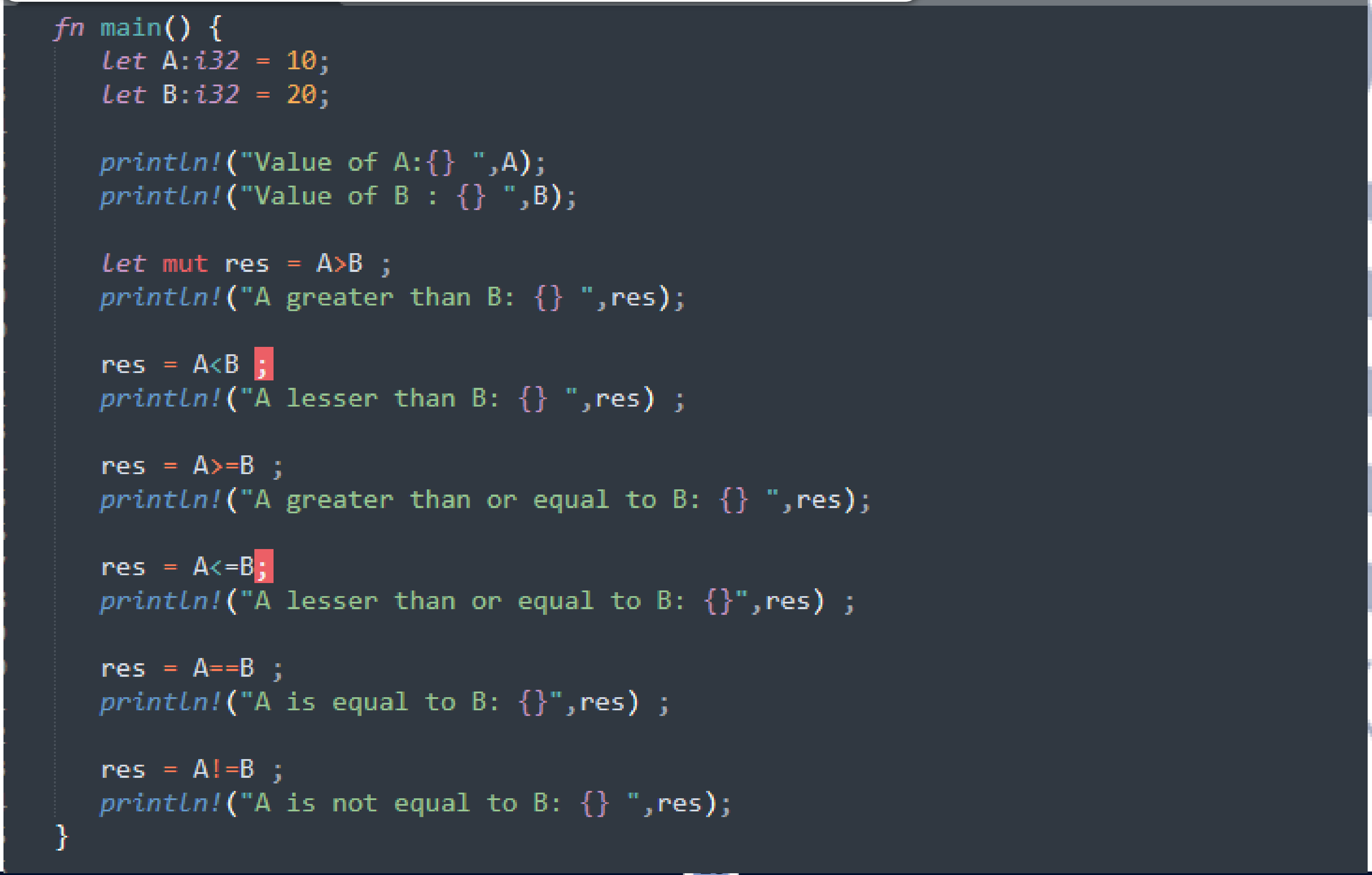
Relational Operators

Relational Operators test or define the kind of relationship between two entities. Relational operators are used to compare two or more values. Relational operators return a Boolean value − true or false.

Sr.No	Operator	Description	Example
1	>	Greater than	(A > B) is False
2	<	Lesser than	(A < B) is True
3	>=	Greater than or equal to	(A >= B) is False
4	<=	Lesser than or equal to	(A <= B) is True
5	==	Equality	(A == B) is False
6	!=	Not equal	(A != B) is True

Relational Operators illustration

Practice 9: *week-6/practice_9/src/main.rs*



```
fn main() {
    let A:i32 = 10;
    let B:i32 = 20;

    println!("Value of A:{} ",A);
    println!("Value of B : {} ",B);

    let mut res = A>B ;
    println!("A greater than B: {} ",res);

    res = A<B ;
    println!("A lesser than B: {} ",res) ;

    res = A>=B ;
    println!("A greater than or equal to B: {} ",res);

    res = A<=B;
    println!("A lesser than or equal to B: {}",res) ;

    res = A==B ;
    println!("A is equal to B: {}",res) ;

    res = A!=B ;
    println!("A is not equal to B: {} ",res);
}
```

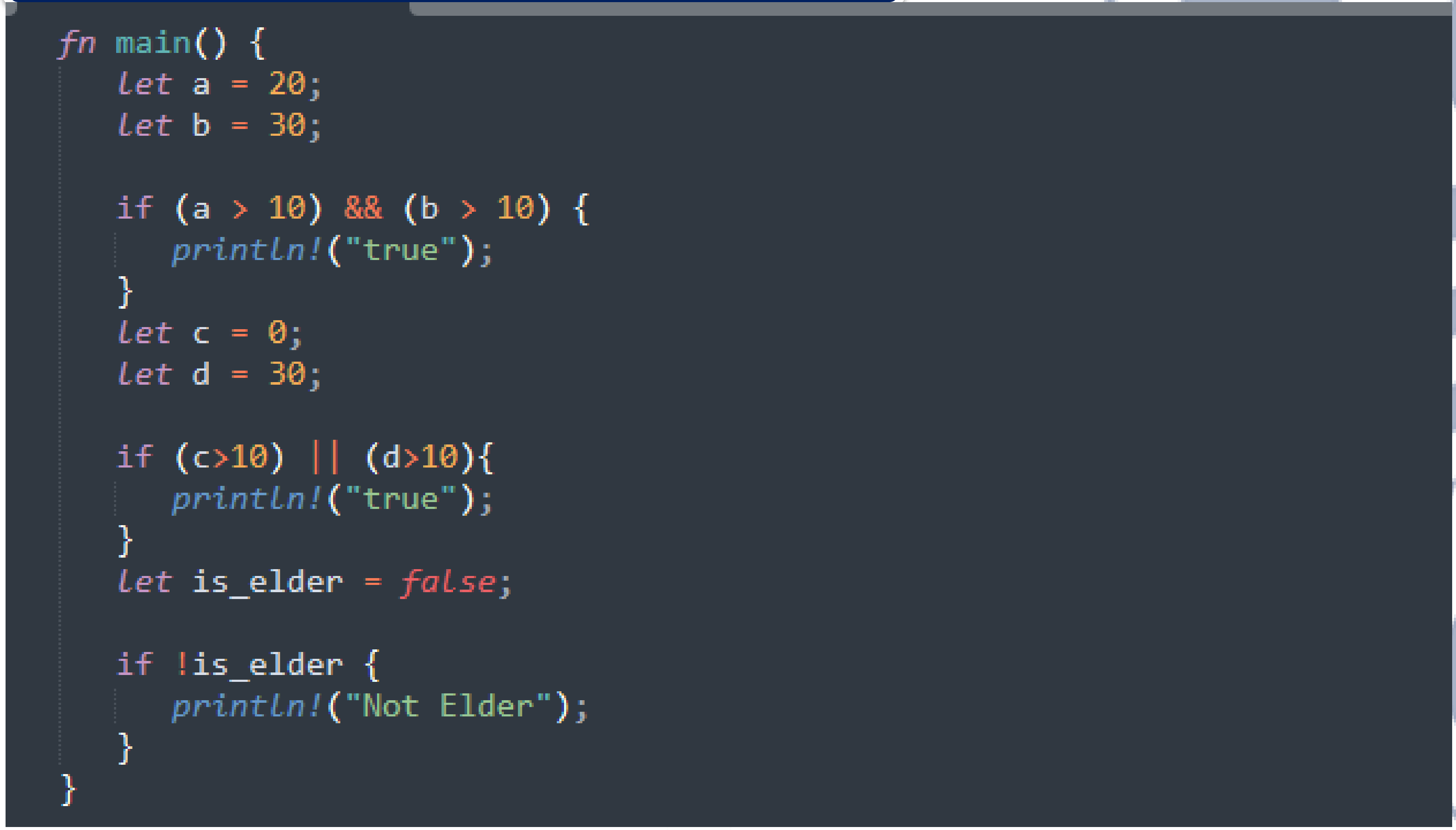
Logical Operators

Logical Operators are used to combine two or more conditions. Logical operators too, return a Boolean value. Assume the value of variable A is 10 and B is 20.

Sr.No	Operator	Description	Example
1	&& (And)	The operator returns true only if all the expressions specified return true	(A > 10 && B > 10) is False
2	||(OR)	The operator returns true if at least one of the expressions specified return true	(A > 10 || B >10) is True
3	! (NOT)	The operator returns the inverse of the expression's result. For E.g.: !(>5) returns false	!(A >10) is True

Logical Operators illustration

Practice 10: ***week-6/practice_10/src/main.rs***



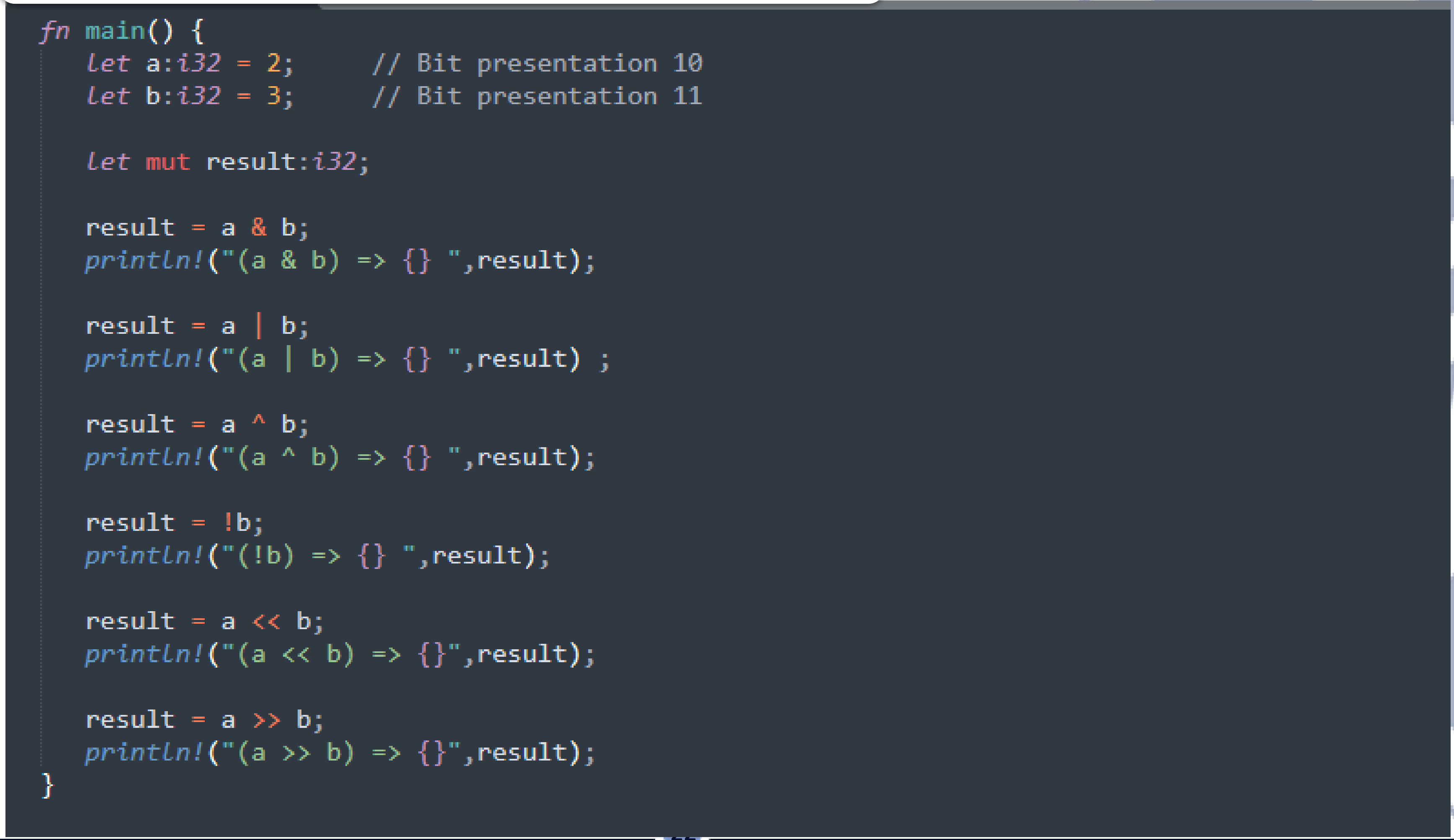
```rust
fn main() {
    let a = 20;
    let b = 30;

    if (a > 10) && (b > 10) {
        println!("true");
    }
    let c = 0;
    let d = 30;

    if (c>10) || (d>10){
        println!("true");
    }
    let is_elder = false;

    if !is_elder {
        println!("Not Elder");
    }
}
```

Bitwise Operators

Sr.No	Operator	Description	Example
1	& (Bitwise AND)	It performs a Boolean AND operation on each bit of its integer arguments.	(A & B) is 2
2	| (BitWise OR)	It performs a Boolean OR operation on each bit of its integer arguments.	(A | B) is 3
3	^ (Bitwise XOR)	It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.	(A ^ B) is 1
4	! (Bitwise Not)	It is a unary operator and operates by reversing all the bits in the operand.	(!B) is -4
5	<< (Left Shift)	It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.	(A << 1) is 4
6	>> (Right Shift)	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	(A >> 1) is 1

Bitwise illustration

Practice 11: week-6/practice_11/src/main.rs



```rust
fn main() {
    let a:i32 = 2;     // Bit presentation 10
    let b:i32 = 3;     // Bit presentation 11

    let mut result:i32;

    result = a & b;
    println!("(a & b) => {} ",result);

    result = a | b;
    println!("(a | b) => {} ",result) ;

    result = a ^ b;
    println!("(a ^ b) => {} ",result);

    result = !b;
    println!("(!b) => {} ",result);

    result = a << b;
    println!("(a << b) => {}",result);

    result = a >> b;
    println!("(a >> b) => {}",result);
}
```

Class Projects

SCHOOL OF SCIENCE AND TECHNOLOGY
PAN-ATLANTIC UNIVERSITY

Project I

Develop a Rust program that displays the following menu for the food items available to take order from the customer.

The program inputs the type of food and quantity. It finally displays the total charges for the order according to price criteria. If the total order is greater than N10,000, give a discount of 5%.

Menu	Price
P = Poundo Yam/Edinkaiko Soup	- N3,200
F = Fried Rice & Chicken	- N3,000
A = Amala & Ewedu Soup	- N2,500
E = Eba & Egusi Soup	- N2,000
W = White Rice & Stew	- N2,500

SCHOOL OF
SCIENCE AND
TECHNOLOGY

PAN-ATLANTIC UNIVERSITY