1. Write a program to display output using print!

**Program:**

fn main(){

print!("Hi This is {}","Rust Programming")

}

**Output:** Hi This is Rust Programming

2. Write a program to display Output following pattern using Placeholders

1

22

333

4444

55555

**Program:**

fn main(){

println!("{}","1");

println!("{}","22");

println!("{}","333");

println!("{}","4444");

println!("{}","55555");

}

**Output:**

1

22

333

4444

55555

3. Write a program to do the following

a. Declare a variable x and store value 1000 in it.

b. Declare a variable y and store value “Programming” in it

c. Print the values of x and y

d. Change the value of x to 1100

e. Print the values of x and y

**Program:**

fn main(){

let mut x=1000;

let y ="Programming";

println!("{}",x);

println!("{}",y);

x=1100;

println!("{}",x);

println!("{}",y);

}

**Output:**

1000

Programming

1100

Programming

4. Write a program to implement the Scope and Shadowing

**Program:**

fn main(){

let outer\_variable = 112;

{

let inner\_variable = 222;

println!("Outer Varible is {}.",outer\_variable);

println!("Inner Varible is {}.",inner\_variable);

}

println!("Inner Varible is {}.",inner\_variable);

// ABOVE LINE WILL GIVE US AN ERROR AS IT IS DECLARED INSIDE THE SCOPE SO IT SHOULD BE ECLARED AS GLOBAL TO ACCESS

}

5. Write a program to implement the following

a. Implicit type declaration

b. Explicit type declaration

**Program:**

fn main() {

//Implicit Type Declaration in Rust

let a = 23;

let b = 3.12;

println!("{}",a);

println!("{}",b);

//Explicit Type Declaration in Rust

let a:i64 = 33; //Explicitly Declaring that it is a integer of 64 bit size

let b:f32 = 3.14;

println!("{}",a);

println!("{}",b);

}

**Output:**

23

3.12

33

3.14

6. Write Program to Declare an array, arr, of size 6 that has numbers divisible by 2

ranging from 0 to 10 and Print the value of arr.

**Program:**

fn main() {

// define an array

let arr:[i32;6] = [0, 2, 4, 6, 8, 10];

// print the values of array

print!("{},{},{},{},{},{}",arr[0], arr[1], arr[2], arr[3], arr[4], arr[5]);

}

**Output:**

0,2,4,6,8,10

7. Write a program to create and access a tuple.

**Program:**

fn main() {

let tuple = ("Rust",'c',5);

print!("{} {} {}",tuple.0,tuple.1,tuple.2);

}

**Output:**

Rust c 5

8. Write a program to create an array of 10 elements and implement the following

a. Create a of 2nd and 3rd element

b. Omit the start index of the slice

c. Omit the End Index of the Slice

d. Omit both Start and End Index of the Slice

**Program:**

fn main() {

let my\_array: [i32; 10] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let slice\_2nd\_3rd = &my\_array[1..3];

println!("Slice of 2nd and 3rd elements: {:?}", slice\_2nd\_3rd);

let omit\_start = &my\_array[1..];

println!("Slice omitting the start index: {:?}", omit\_start);

let omit\_end = &my\_array[..8];

println!("Slice omitting the end index: {:?}", omit\_end);

let omit\_both = &my\_array[..];

println!("Slice omitting both start and end index (entire array): {:?}", omit\_both);

}

**Output:**

Slice of 2nd and 3rd elements: [2, 3]

Slice omitting the start index: [2, 3, 4, 5, 6, 7, 8, 9, 10]

Slice omitting the end index: [1, 2, 3, 4, 5, 6, 7, 8]

Slice omitting both start and end index (entire array): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

9. Write a program to create different types of constants print it in the output

**Program:**

fn main(){

const name:i32 = 34; //Global Constant

{

const name2:i32 = 64; //Local Constant

println!("{}",name);

println!("{}",name2);

}

}

**Output:**

34

64

10. Declaring String Object and converting String Literal to String Object

**Program:**

fn main(){

let str\_obj = String::from("String Name is Rust");

let str\_lit = str\_obj.to\_string();

println!("{}",str\_lit);

}

**Output:**

String Name is Rust

11. Write a program to implement Type Casting Operator.

**Program:**

fn main(){

let a = 15;

let b = (a as f64) / 2.0; //Type Casting converting to Foat

println!("a: {}", a);

println!("b: {}", b);

}

**Output:**

a: 15

b: 7.5

12. Write a program to implement Borrowing and Dereferencing Operators

**Program:**

//Borrowing and Dereferencing Operators

fn main() {

let x = 10;

let mut y = 13;

let a = &x;

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

println!("Value of x:{}", x); // x value remains the same since it is immutably borrowed

//mutable reference to a variable

let b = &mut y;

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

\*b = 11; // derefencing

println!("Value of b:{}", b); // updated value of b

println!("Value of y:{}", y); // y value can be changed as it is mutuably borrowed

}

**Output:**  
Value of a:10

Value of x:10

Value of b:13

Value of b:11

Value of y:11

13. Write a program to check if a number is positive or negative

**Program:**

fn main(){

let a = 5;

if a>0{

println!("{a} is positive",a=a);

}

else if a<0{

println!("{a} is negative",a=a);

}

else{

println!("{a} is neither positive nor negative it is zero",a=a);

}

}

**Output:**

5 is positive

14. Write a program to determine if a number is even or odd

**Program:**

fn main(){

let a = 6;

if a%2==0{

println!("{} is even",a);

}

else{

println!("{} is odd",a);

}

}

**Output:**

6 is postive

15. Write a program to make a calculator using Match Expression

**Program:**

let a =5;

let b = 10;

fn test(a: i32, operator: char ,b: i32) {

match operator {

'+' => {

println!("{}", a + b);

},

'-' => {

println!("{}", a - b);

},

'\*' => {

println!("{}", a \* b);

},

'/' => {

if b == 0{

println!("Division by 0 is undefined");

}

else {

println!("{}", a / b);

}

},

'%' => {

if b == 0{

println!("Mod 0 is undefined");

}

else {

println!("{}", a % b);

}

},

\_ => println!("{}", "invalid operator"),

}

}

fn main(){

print!("3 + 2: ");

test(3,'+',2);

print!("3 - 2: ");

test(3,'-',2);

print!("3 \* 2: ");

test(3,'\*',2);

print!("3 / 2: ");

test(3,'/',2);

print!("3 % 2: ");

test(3,'%',2);

print!("3 ( 2: ");

test(3,'(',2);

print!("3 ( 0: ");

test(3, '/', 0)

}

**Output:**

3 + 2: 5

3 - 2: 1

3 \* 2: 6

3 / 2: 1

3 % 2: 1

3 ( 2: invalid operator

3 ( 0: Division by 0 is undefined

16. Write a program to Match a pattern using If Let Expression

**Program:**

fn main() {

*// define a variable*

let x = 5;

*// define match expression*

match x {

1 => println!("Java"),

2 => println!("Python"),

3 => println!("C++"),

4 => println!("C#"),

5 => println!("Rust"),

6 => println!("Kotlin"),

\_ => println!("Some other value"),

};

}

**Output:**

Rust

17. Write a program to Print First 10 Natural Numbers using Loop

**Program:**

fn main(){

for i in 1..11{

println!("{}",i);

}

}

**Output:**

1

2

3

4

5

6

7

8

9

10

18. Write a program to Multiplication Table using Loop Labels

**Program:**

fn main(){

//Will arise warnings if the loop labels are not usede

'outer: for i in 1..5{

println!("Multiplication Table of {} is",i);

'inner: for j in 1..5{

println!("{}\* {} = {}",i,j,i\*j);

}

}

}

**Output:**

Multiplication Table of 1 is

1\* 1 = 1

1\* 2 = 2

1\* 3 = 3

1\* 4 = 4

Multiplication Table of 2 is

2\* 1 = 2

2\* 2 = 4

2\* 3 = 6

2\* 4 = 8

Multiplication Table of 3 is

3\* 1 = 3

3\* 2 = 6

3\* 3 = 9

3\* 4 = 12

Multiplication Table of 4 is

4\* 1 = 4

4\* 2 = 8

4\* 3 = 12

19. Write a program to Count Iterations of a Loop Until a Condition

## **Example: Problem Statement**

* A variable x is provided to you.
* Repeatedly decrease the value of the variable x by 3 each time, as long as x is greater than or equal to 0.
* Print the number of times the iteration runs.

**Program:**

fn test(mut x:i32) {

// define a mutable variable

let mut count = 0;

// define a while loop

while x >= 0 {

x = x - 3; // decrement the value of x by 3

count = count + 1;

}

println!("{}", count);

}

fn main(){

print!("Iterations when x = 21 :");

test(21);

print!("Iterations when x = 33 :");

test(33);

}

**Output:**

Iterations when x = 21 :8

Iterations when x = 33 :12

20. Write a program to Print the following patterns

&

&&

&&&

&&&&

&&&&&

**Program:**

fn main(){

for i in 1..6{

for \_j in 0..i{

print!("&");

}

println!();

}

}

**Output:**

&

&&

&&&

&&&&

&&&&&

21. Write a program to print the values in a collection using iter() method

**Program:**

fn main() {

*// Create a vector as an example collection*

let numbers = vec![1, 2, 3, 4, 5];

*// Create an iterator for the vector using iter()*

let iter = numbers.iter();

*// Use a for loop to print the values in the collection*

println!("Printing values in the collection using iter():");

for value in iter {

println!("{}", value);

}

}

**Output:**

Printing values in the collection using iter():

1

2

3

4

5

22. Write a program to Find The Factorial using functions.

**Program:**

fn test(n:i32) {

// Write code here!

let mut prod:i32 = 1;

if n<0{

print!("{}",0);

}

else if n==0{

print!("{}",1);

}

else{

for i in 1..n+1{

prod\*=i

}

print!("{}",prod);

}

}

fn main(){

print!("factorial (4) : ");

test(4);

println!();

print!("factorial (6) : ");

test(6);

}

**Output:**

factorial (4) : 24

factorial (6) : 720

23. Write a function test\_divisibility\_by\_3\_4 which will check whether a given integer

number is divisible by 3 or 4.

a. If the number is divisible by both return 0

b. If the number is divisible by 3 only return 1

c. If the number is divisible by 4 only return 2

d. If the number is not divisible by both, return -1

**Program:**

fn test\_divisibility\_by\_3\_4(a:i32) -> i32{

// Write code here

if a%3==0 && a%4==0{

return 0;

}

else if a%3==0{

return 1;

}

else if a%4==0{

return 2;

}

return -1;

}

fn main(){

println!(" Number = 12 : {}", test\_divisibility\_by\_3\_4(12));

println!(" Number = 9 : {}", test\_divisibility\_by\_3\_4(9));

println!(" Number = 8 : {}", test\_divisibility\_by\_3\_4(8));

println!(" Number = 23 : {}", test\_divisibility\_by\_3\_4(23));

}

**Output:**

Number = 12 : 0

Number = 9 : 1

Number = 8 : 2

Number = 23 : -1

24. Write a program to demonstrate the Pass by Value and Pass by Reference

**Program:**

fn square(n:&mut i32){

\*n = \*n \* \*n;

println!("The value of n inside function : {}", n);

}

fn main() {

let mut n = 4;

println!("The value of n before function call : {}", n);

println!("Invoke Function");

square(&mut n);

println!("The value of n after function call : {}", n);

}

**Output:**

The value of n before function call : 4

Invoke Function

The value of n inside function : 16

The value of n after function call : 16

25. Write a function calculate\_area\_perimeter() that takes x and y( length and width of a

rectangle) as a parameter to the function and returns a tuple (area, perimeter).

**Program:**

fn calculate\_area\_perimeter(x: f64, y: f64) -> (f64, f64) {

let area = x \* y;

let perimeter = 2.0 \* (x + y);

(area, perimeter)

}

fn main() {

let length = 5.0;

let width = 3.0;

let (area, perimeter) = calculate\_area\_perimeter(length, width);

println!("Length: {}", length);

println!("Width: {}", width);

println!("Area: {}", area);

println!("Perimeter: {}", perimeter);

}

**Output:**

Length: 5

Width: 3

Area: 15

Perimeter: 16

26. Write a function arr\_square() that returns the Array of Squares

**Program:**

fn arr\_square() -> [i32;5] {

let mut square:[i32;5] = [1, 2, 3, 4, 5]; // mutable array

for i in 0..5 { // compute the square of each element

square[i] = square[i] \* square[i];

}

square

}

fn main(){

println!("Updated Array : {:?}",arr\_square());

}

**Output:**

Updated Array : [1, 4, 9, 16, 25]

27. write a recursive function fibonacci that takes a positive integer number n as a

parameter and returns the nth Fibonacci term in that range.

**Program:**

fn fibonacci(term: i32) -> i32 {

match term {

0 => 0,

1 => 1,

\_ => fibonacci(term-1) + fibonacci(term-2),

}

}

fn main(){

println!("fibonacci(4)={}",fibonacci(4));

}

**Output:**

fibonacci(4)=3

28. Write a program to Creating a String

**Program:**

fn main() {

let course1 = String::new();

let s\_course1 = course1.to\_string();

println!("This is an empty string {}.", s\_course1);

println!("This is a length of my empty string {}.", s\_course1.len());

let course2 = "Rust Programming";

let s\_course2 = course2.to\_string();

println!("This is a string literal : {}.", s\_course2);

println!("This is a length of my string literal {}.", s\_course2.len());

let course3 = String::from("Rust Language");

println!("This is a string object : {}.", course3);

println!("This is the length of my string object {}.", course3.len());

}

**Output:**

This is an empty string .

This is a length of my empty string 0.

This is a string literal : Rust Programming.

This is a length of my string literal 16.

This is a string object : Rust Language.

This is the length of my string object 13.

29. Implement the string manipulation operations using Core Methods of String Objects

a. str.capacity()

b. str.contains("sub\_str")

c. str.replace(replace\_from, replace\_to)

d. string.trim()

**Program:**

fn main() {

// define a growable string variable

let str = String::from("Rust Programming");

println!("This is a beginner course in {}.", str);

//capacity in bytes

println!("Capacity: {}.", str.capacity());

let sub\_str = String::from("Rust");

// find if string contains a substring

println!("{} is a substring of {}: {}.", sub\_str, str, str.contains("Rust"));

let replace\_from = "Programming";

let replace\_to = "Language";

// find if string contains a substring

let result = str.replace(replace\_from, replace\_to);

println!("{} now becomes {}.", str, result);

let string = " Rust Programming ".to\_string();

let trim\_string = string.trim();

// get characters at 5,6,7,8,9,10 and 11 ndexes

println!("Trimmed\_string : {}", trim\_string);

}

**Output:**

This is a beginner course in Rust Programming.

Capacity: 16.

Rust is a substring of Rust Programming: true.

Rust Programming now becomes Rust Language.

Trimmed\_string : Rust Programming

30. Write a program to tokenize and iterate over a string

**Program:**

fn main() {

// define a String object

let str = String::from("Educative, course on, Rust, Programming");

// split on token

for token in str.split(","){

println!("{}", token);

}

}

**Output**:

Educative

course on

Rust

Programming