Table of Contents

1. Structure of a Rust Program

```
fn main() {
    println!("Hello, World!");  // Print to console
}
```

---

2. Variables and Data Types

Variable Declaration

```
let age = 25;                   // Immutable by default
let mut height = 5.9;           // Mutable variable
const PI: f64 = 3.14159;        // Constant (explicit type required)
```

Data Types

```
let x: i32 = 10;           // Integer (i32, i64, u32, u64)
let y: f64 = 5.8;          // Floating point
let is_active: bool = true; // Boolean
let grade: char = 'A';     // Character
```

---

3. Strings

```
let s1 = String::from("Hello");  // Growable string
let s2 = "World";                // String slice
let s3 = format!("{} {}", s1, s2);  // String formatting
println!("{}", s3);               // Output: Hello World
```

---

4. Control Flow

If-Else Statements

```
let age = 18;

if age >= 18 {
    println!("Adult");
} else {
    println!("Minor");
}
```

Match (Switch Alternative)

```
let number = 3;

match number {
    1 => println!("One"),
    2 => println!("Two"),
    3 => println!("Three"),
    _ => println!("Other"),
}
```

---

5. Loops

For Loop

```
for i in 1..=5 {
    println!("{}", i);
}
```

While Loop

```
let mut count = 0;

while count < 5 {
    println!("{}", count);
    count += 1;
}
```

Infinite Loop (Loop + Break)

```
let mut n = 0;

loop {
    println!("{}", n);
    n += 1;
    if n == 5 {
        break;
    }
}
```

---

6. Functions

```
fn add(a: i32, b: i32) -> i32 {
    a + b  // Implicit return (no semicolon)
}
fn main() {
    let result = add(10, 5);
    println!("Sum: {}", result);
}
```

---

7. Arrays and Tuples

Arrays

```
let numbers = [1, 2, 3, 4, 5];
println!("{}", numbers[0]);  // Access array element
```

Tuples

```
let person = ("Alice", 30);
println!("Name: {}, Age: {}", person.0, person.1);
```

---

8. Structs (Custom Data Types)

```
struct Person {
    name: String,
    age: u32,
}

fn main() {
    let person = Person {
        name: String::from("Bob"),
        age: 25,
    };

    println!("{} is {} years old.", person.name, person.age);
}
```

---

9. Enums

```
enum Direction {
    Up,
```

```
    Down,
    Left,
    Right,
}

fn move_character(dir: Direction) {
    match dir {
        Direction::Up => println!("Move Up"),
        Direction::Down => println!("Move Down"),
        Direction::Left => println!("Move Left"),
        Direction::Right => println!("Move Right"),
    }
}

fn main() {
    move_character(Direction::Left);
}
```

---

10. Option and Result (Error Handling)

Option

```
fn divide(a: f64, b: f64) -> Option<f64> {
    if b != 0.0 {
        Some(a / b)
    } else {
        None
    }
}

fn main() {
    let result = divide(10.0, 2.0);
    match result {
        Some(value) => println!("Result: {}", value),
        None => println!("Cannot divide by zero"),
    }
```

```
}
```

Result

```
fn divide(a: f64, b: f64) -> Result<f64, String> {
    if b != 0.0 {
        Ok(a / b)
    } else {
        Err(String::from("Division by zero"))
    }
}

fn main() {
    let result = divide(10.0, 0.0);
    match result {
        Ok(value) => println!("Result: {}", value),
        Err(e) => println!("Error: {}", e),
    }
}
```

---

11. Vectors (Dynamic Arrays)

```
let mut nums = vec![1, 2, 3];
nums.push(4);  // Add element
nums.remove(0); // Remove first element

for n in &nums {
    println!("{}", n);
}
```

---

12. Ownership and Borrowing

Ownership (Move Semantics)

```
let s1 = String::from("Hello");
let s2 = s1;  // s1 is moved to s2

// println!("{}", s1);  // Error: s1 no longer valid
println!("{}", s2);    // s2 owns the value
```

Borrowing (References)

```
fn greet(name: &String) {
    println!("Hello, {}", name);
}

fn main() {
    let s1 = String::from("Alice");
    greet(&s1);  // Pass reference
    println!("{}", s1);  // s1 is still valid
}
```

---

13. Mutable References

```
fn change(text: &mut String) {
    text.push_str(" World!");
}

fn main() {
    let mut s = String::from("Hello");
    change(&mut s);  // Mutable borrow
    println!("{}", s);  // Hello World!
}
```

---

14. Lifetimes

```
fn longest<'a>(s1: &'a str, s2: &'a str) -> &'a str {
    if s1.len() > s2.len() {
        s1
    } else {
        s2
    }
}
```

---

15. Traits (Interfaces)

```
trait Animal {
    fn speak(&self);
}

struct Dog;

impl Animal for Dog {
    fn speak(&self) {
        println!("Woof!");
    }
}

fn main() {
    let dog = Dog;
    dog.speak();
}
```

---

16. File I/O

```
use std::fs;

fn main() {
    let content = fs::read_to_string("example.txt")
        .expect("Failed to read file");
    println!("{}", content);
}
```

---

17. Modules and Packages

```
mod utils {
    pub fn greet(name: &str) {
        println!("Hello, {}!", name);
    }
}

fn main() {
    utils::greet("Alice");
}
```