# Module 1: Introduction to Java Fundamentals

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology
Chennai Campus

*premanand.s@vit.ac.in*

January 5, 2026

# Module 1: Introduction to Java Fundamentals

- OOP Paradigm and Features of Java
- JVM, Bytecode, Java Program Structure
- Data Types, Variables, Naming Conventions
- Operators, Control and Looping Constructs
- One- and Multi-dimensional Arrays
- Enhanced for-loop
- Strings, StringBuffer, StringBuilder, Math Class
- Wrapper Classes

# What is a Variable?

A **variable** is a named memory location used to store data.

# What is a Variable?

A **variable** is a named memory location used to store data.
**In simple words:**

- Variable = Container for values
- Value can change during program execution

# What is a Variable?

A **variable** is a named memory location used to store data.

**In simple words:**

- Variable = Container for values
- Value can change during program execution

**Example (Real life):**

- Water bottle → holds water
- Variable → holds data

# Variables: Python vs Java

| **Feature** | **Python** | **Java** |
|---|---|---|
| Type declaration | Not required | Mandatory |
| Example | `x = 10` | `int x = 10;` |
| Type checking | Mostly runtime | Compile-time |
| Flexibility | More flexible | More structured and strict |

# Variable Declaration in Java

**Syntax:**

```
dataType variableName = value;
```

# Variable Declaration in Java

**Syntax:**

```
dataType variableName = value;
```

**Examples:**

```
int age = 20;
float marks = 85.5f;
char grade = 'A';
boolean pass = true;
```

# What is a Data Type?

A **data type** specifies:

- What type of data a variable can store
- How much memory is allocated
- What operations are allowed

# What is a Data Type?

A **data type** specifies:

- What type of data a variable can store
- How much memory is allocated
- What operations are allowed

**Why data types matter?**

- Prevents invalid data usage
- Improves performance
- Helps detect errors early

# Types of Data Types in Java

Java data types are classified into:

# Types of Data Types in Java

Java data types are classified into:

- **Primitive Data Types**
  - int, float, double, char, boolean, byte, short, long

# Types of Data Types in Java

Java data types are classified into:

- **Primitive Data Types**
  - int, float, double, char, boolean, byte, short, long
- **Non-Primitive (Reference) Data Types**
  - String
  - Arrays
  - Classes
  - Interfaces

# Data Type: int

**What is int?**

# Data Type: int

**What is int?**

- Used to store whole numbers
- No decimal values allowed

# Data Type: int

**What is int?**

- Used to store whole numbers
- No decimal values allowed

**Purpose:**

- Counting values
- Loop counters
- Indexing arrays

# Data Type: int

**What is int?**

- Used to store whole numbers
- No decimal values allowed

**Purpose:**

- Counting values
- Loop counters
- Indexing arrays

**Code Example:**

```
int age = 20;
int count = 100;
```

# Data Type: float

**What is float?**

# Data Type: float

**What is float?**

- Stores decimal values
- Single precision (less accurate than double)

# Data Type: float

**What is float?**

- Stores decimal values
- Single precision (less accurate than double)

**Purpose:**

- Memory-efficient decimal storage
- Used when precision is not critical

# Data Type: float

**What is float?**

- Stores decimal values
- Single precision (less accurate than double)

**Purpose:**

- Memory-efficient decimal storage
- Used when precision is not critical

**Code Example:**

```
float temperature = 36.5f;
float average = 78.25f;
```

# Data Type: double

**What is double?**

# Data Type: double

**What is double?**

- Stores decimal values with high precision
- Default type for decimals in Java

# Data Type: double

**What is double?**

- Stores decimal values with high precision
- Default type for decimals in Java

**Purpose:**

- Scientific calculations
- Financial and engineering applications

# Data Type: double

**What is double?**

- Stores decimal values with high precision
- Default type for decimals in Java

**Purpose:**

- Scientific calculations
- Financial and engineering applications

**Code Example:**

```
double pi = 3.14159;
double distance = 12345.678;
```

# Data Type: char

**What is char?**

# Data Type: char

**What is char?**

- Stores a single character
- Uses single quotes

# Data Type: char

**What is char?**

- Stores a single character
- Uses single quotes

**Purpose:**

- Storing grades, symbols, letters

# Data Type: char

**What is char?**

- Stores a single character
- Uses single quotes

**Purpose:**

- Storing grades, symbols, letters

**Code Example:**

```
char grade = 'A';
char gender = 'M';
```

# Data Type: boolean

**What is boolean?**

# Data Type: boolean

**What is boolean?**

- Stores only two values: true or false

# Data Type: boolean

**What is boolean?**

- Stores only two values: true or false

**Purpose:**

- Decision making
- Conditional statements

# Data Type: boolean

**What is boolean?**

- Stores only two values: true or false

**Purpose:**

- Decision making
- Conditional statements

**Code Example:**

```
boolean isPassed = true;
boolean isEligible = false;
```

# Data Type: long

**What is long?**

# Data Type: long

**What is long?**

- Stores very large whole numbers

# Data Type: long

**What is long?**

- Stores very large whole numbers

**Purpose:**

- Bank account numbers
- Population count
- Large IDs

# Data Type: long

**What is long?**

- Stores very large whole numbers

**Purpose:**

- Bank account numbers
- Population count
- Large IDs

**Code Example:**

```
long population = 1400000000L;
long accountNumber = 9876543210L;
```

# Data Types: byte and short

**What are byte and short?**

# Data Types: byte and short

**What are byte and short?**

- Used for small-range integers
- Memory efficient

# Data Types: byte and short

**What are byte and short?**

- Used for small-range integers
- Memory efficient

**Purpose:**

- Embedded systems
- Large arrays where memory matters

# Data Types: byte and short

**What are byte and short?**

- Used for small-range integers
- Memory efficient

**Purpose:**

- Embedded systems
- Large arrays where memory matters

**Code Example:**

```
byte level = 5;
short year = 2024;
```

# Choosing the Right Data Type

- Use **int** for normal whole numbers
- Use **double** for decimals
- Use **boolean** for conditions
- Use **char** for single characters
- Use **long** for very large values

# Choosing the Right Data Type

- Use **int** for normal whole numbers
- Use **double** for decimals
- Use **boolean** for conditions
- Use **char** for single characters
- Use **long** for very large values

**Correct data type = efficient + error-free program**

# Java Naming Conventions

Java follows strict naming rules and conventions.

# Java Naming Conventions

Java follows strict naming rules and conventions.
**Rules:**

- Must start with a letter, _ or $
- Cannot start with a digit
- Cannot use keywords

# Java Naming Conventions

Java follows strict naming rules and conventions.

**Rules:**

- Must start with a letter, _ or $
- Cannot start with a digit
- Cannot use keywords

**Conventions:**

- Variable names $\rightarrow$ camelCase (studentMarks)
- Class names $\rightarrow$ PascalCase (StudentDetails)
- Constants $\rightarrow$ UPPER_CASE (MAX_SIZE)

# Common Mistakes Students Make

- Forgetting data type during declaration
- Using wrong suffix (f for float, L for long)
- Using keywords as variable names
- Confusing char ('A') with String ("A")

# What is an Operator?

An **operator** is a symbol that performs an operation on one or more operands.

# What is an Operator?

An **operator** is a symbol that performs an operation on one or more operands.

**In simple words:**

- Operator $\rightarrow$ action
- Operand $\rightarrow$ value or variable

# What is an Operator?

An **operator** is a symbol that performs an operation on one or more operands.

**In simple words:**

- Operator $\rightarrow$ action
- Operand $\rightarrow$ value or variable

**Example:**

- In `a + b`
- `+` is the operator
- `a` and `b` are operands

# Types of Operators in Java

Java supports several types of operators:

# Types of Operators in Java

Java supports several types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Unary Operators
- Bitwise Operators
- Ternary Operator

# Types of Operators in Java

Java supports several types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Unary Operators
- Bitwise Operators
- Ternary Operator

We will start with the most commonly used ones.

# Arithmetic Operators

Arithmetic operators are used to perform basic mathematical calculations.

# Arithmetic Operators

Arithmetic operators are used to perform basic mathematical calculations.

| Operator | Meaning | Example |
|---|---|---|
| + | Addition | a + b |
| - | Subtraction | a - b |
| * | Multiplication | a * b |
| / | Division | a / b |
| % | Modulus (Remainder) | a % b |

# Arithmetic Operators

Arithmetic operators are used to perform basic mathematical calculations.

| Operator | Meaning | Example |
|---|---|---|
| + | Addition | a + b |
| - | Subtraction | a - b |
| * | Multiplication | a * b |
| / | Division | a / b |
| % | Modulus (Remainder) | a % b |

**Note:**

- Division between two integers gives an integer result
- Modulus operator returns the remainder

# Arithmetic Operators

```
public class ArithmeticOperators {
    public static void main(String[] args) {

        int a = 10;
        int b = 3;

        System.out.println("Addition: " + (a + b));
        System.out.println("Subtraction: " + (a - b));
        System.out.println("Multiplication: " + (a * b));
        System.out.println("Division: " + (a / b));
        System.out.println("Modulus: " + (a % b));
    }
}
```

# Relational Operators in Java

Relational operators are used to compare two values.

# Relational Operators in Java

Relational operators are used to compare two values.

| Operator | Meaning | Example |
|---|---|---|
| == | Equal to | `a == b` |
| != | Not equal to | `a != b` |
| > | Greater than | `a > b` |
| < | Less than | `a < b` |
| >= | Greater than or equal to | `a >= b` |
| <= | Less than or equal to | `a <= b` |

# Relational Operators in Java

Relational operators are used to compare two values.

| Operator | Meaning | Example |
|---|---|---|
| == | Equal to | `a == b` |
| != | Not equal to | `a != b` |
| > | Greater than | `a > b` |
| < | Less than | `a < b` |
| >= | Greater than or equal to | `a >= b` |
| <= | Less than or equal to | `a <= b` |

**Note:**

- Relational operators always return a **boolean** value
- Do not confuse = (assignment) with == (comparison)

# Relational Operators

```
public class RelationalOperators {
    public static void main(String[] args) {

        int a = 10;
        int b = 5;

        System.out.println("a == b : " + (a == b));
        System.out.println("a != b : " + (a != b));
        System.out.println("a > b  : " + (a > b));
        System.out.println("a < b  : " + (a < b));
        System.out.println("a >= b : " + (a >= b));
        System.out.println("a <= b : " + (a <= b));
    }
}
```

# Logical Operators in Java

Logical operators are used to combine or modify boolean expressions.

# Logical Operators in Java

Logical operators are used to combine or modify boolean expressions.

| Operator | Meaning | Example |
|---|---|---|
| && | Logical AND | `(a > b) && (a > c)` |
| \|\| | Logical OR | `(a > b) \|\| (a < c)` |
| ! | Logical NOT | `!(a > b)` |

# Logical Operators in Java

Logical operators are used to combine or modify boolean expressions.

| Operator | Meaning | Example |
|---|---|---|
| `&&` | Logical AND | `(a > b) && (a > c)` |
| `\|\|` | Logical OR | `(a > b) \|\| (a < c)` |
| `!` | Logical NOT | `!(a > b)` |

**Note:**

- Logical operators work only with boolean expressions
- Result of logical operations is always **true** or **false**

# Logical Operators

```
public class LogicalOperators {
  public static void main(String[] args) {

    int a = 10;
    int b = 5;
    int c = 20;

    // Logical AND
    System.out.println("(a > b) && (a > c) : "
    + ((a > b) && (a > c)));

    // Logical OR
    System.out.println("(a > b) || (a > c) : "
    + ((a > b) || (a > c)));

    // Logical NOT
    System.out.println("!(a > b) : " + !(a > b));
  }
}
```

# Unary Operators in Java

Unary operators operate on a **single operand**.

# Unary Operators in Java

Unary operators operate on a **single operand**.

| Operator | Meaning | Example |
|---|---|---|
| + | Unary plus | `+a` |
| - | Unary minus | `-a` |
| ++ | Increment (increase by 1) | `++a` **or** `a++` |
| -- | Decrement (decrease by 1) | `--a` **or** `a--` |
| ! | Logical NOT | `!flag` |

# Unary Operators in Java

Unary operators operate on a **single operand**.

| Operator | Meaning | Example |
|---|---|---|
| + | Unary plus | `+a` |
| - | Unary minus | `-a` |
| ++ | Increment (increase by 1) | `++a` **or** `a++` |
| -- | Decrement (decrease by 1) | `--a` **or** `a--` |
| ! | Logical NOT | `!flag` |

**Note:**

- Unary operators work on only one variable
- ++ and -- modify the value of the variable
- Prefix and postfix forms behave differently

# Unary Operators

```
public class UnaryOperators {
    public static void main(String[] args) {
        int a = 10;

        // Unary plus and minus
        System.out.println("Unary plus (+a): " + (+a));
        System.out.println("Unary minus (-a): " + (-a));

        // Increment operator
        System.out.println("Pre-increment (++a): " + (++a));
        System.out.println("Post-increment (a++): " + (a++));
        System.out.println("Value of a after post-increment: " + a)

        // Decrement operator
        System.out.println("Pre-decrement (--a): " + (--a));
        System.out.println("Post-decrement (a--): " + (a--));
        System.out.println("Value of a after post-decrement: " + a)
    }
}
```

# Bitwise Operators in Java

Bitwise operators perform operations directly on the **binary representation** of numbers.

# Bitwise Operators in Java

Bitwise operators perform operations directly on the **binary representation** of numbers.

| Operator | Meaning | Example |
|---|---|---|
| & | Bitwise AND | `a & b` |
| \| | Bitwise OR | `a \| b` |
| ^ | Bitwise XOR | `a ^ b` |
| ~ | Bitwise NOT | `~a` |
| << | Left shift | `a << 1` |
| >> | Right shift | `a >> 1` |

# Bitwise Operators in Java

Bitwise operators perform operations directly on the **binary representation** of numbers.

| Operator | Meaning | Example |
|---|---|---|
| & | Bitwise AND | `a & b` |
| \| | Bitwise OR | `a \| b` |
| ^ | Bitwise XOR | `a ^ b` |
| ~ | Bitwise NOT | `~a` |
| << | Left shift | `a << 1` |
| >> | Right shift | `a >> 1` |

**Note:**

- Bitwise operators work at the **bit level**
- Mostly used in low-level programming and optimization

# Bitwise Operators

```
public class BitwiseOperators {
    public static void main(String[] args) {
        int a = 5;   // Binary: 0101
        int b = 3;   // Binary: 0011
        // Bitwise AND
        System.out.println("a & b  = " + (a & b));   // 1 (0001)
        // Bitwise OR
        System.out.println("a | b  = " + (a | b));   // 7 (0111)
        // Bitwise XOR
        System.out.println("a ^ b  = " + (a ^ b));   // 6 (0110)
        // Bitwise NOT
        System.out.println("~a     = " + (~a));      // -6
        // Left shift
        System.out.println("a << 1 = " + (a << 1));  // 10 (1010)
        // Right shift
        System.out.println("a >> 1 = " + (a >> 1));  // 2 (0010)
    }
}
```

| Operator | Meaning | Example |
|---|---|---|
| `?:` | Conditional (Ternary) Operator | `result = (a > b) ? a : b;` |

# Ternary Operator

```
public class TernaryOperator {
 public static void main(String[] args) {

  int a = 10;
  int b = 20;

  // Find maximum of two numbers
  int max = (a > b) ? a : b;
  System.out.println("Maximum value: " + max);

  // Check even or odd
  int number = 15;
  String result = (number % 2 == 0) ? "Even" : "Odd";
  System.out.println("Number is: " + result);
 }
}
```

# Assignment Operators in Java

Assignment operators are used to assign values to variables.

# Assignment Operators in Java

Assignment operators are used to assign values to variables.

| Operator | Meaning | Example |
|---|---|---|
| = | Assign value | a = 10 |
| += | Add and assign | a += 5 (a = a + 5) |
| -= | Subtract and assign | a -= 3 (a = a - 3) |
| *= | Multiply and assign | a *= 2 (a = a * 2) |
| /= | Divide and assign | a /= 4 (a = a / 4) |
| %= | Modulus and assign | a %= 2 (a = a % 2) |

# Assignment Operators in Java

Assignment operators are used to assign values to variables.

| Operator | Meaning | Example |
|---|---|---|
| = | Assign value | a = 10 |
| += | Add and assign | a += 5 (a = a + 5) |
| -= | Subtract and assign | a -= 3 (a = a - 3) |
| *= | Multiply and assign | a *= 2 (a = a * 2) |
| /= | Divide and assign | a /= 4 (a = a / 4) |
| %= | Modulus and assign | a %= 2 (a = a % 2) |

**Note:**

- Assignment operators simplify expressions
- Commonly used in loops and calculations

# Assignment Operators

```
public class AssignmentOperators {
    public static void main(String[] args) {
        int a = 10;
        System.out.println("Initial value of a: " + a);
        a += 5;   // a = a + 5
        System.out.println("After a += 5 : " + a);
        a -= 3;   // a = a - 3
        System.out.println("After a -= 3 : " + a);
        a *= 2;   // a = a * 2
        System.out.println("After a *= 2 : " + a);
        a /= 4;   // a = a / 4
        System.out.println("After a /= 4 : " + a);
        a %= 3;   // a = a % 3
        System.out.println("After a %= 3 : " + a);
    }
}
```

# Operator Precedence in Java

Operator precedence determines the **order in which operators are evaluated** in an expression.

# Operator Precedence in Java

Operator precedence determines the **order in which operators are evaluated** in an expression.

| **Precedence Level** | **Operators** |
|---|---|
| Highest | (), ++, --, ! |
| | *, /, % |
| | +, - |
| | <, <=, >, >= |
| | ==, != |
| | && |
| | \|\| |
| Lowest | =, +=, -=, *=, /= |

# Operator Precedence in Java

Operator precedence determines the **order in which operators are evaluated** in an expression.

| Precedence Level | Operators |
|---|---|
| Highest | (), ++, --, ! |
| | *, /, % |
| | +, - |
| | <, <=, >, >= |
| | ==, != |
| | && |
| | \|\| |
| Lowest | =, +=, -=, *=, /= |

**Note:**

- Operators with higher precedence are evaluated first
- Parentheses () can be used to change evaluation order

# Why Do We Need User Input?

- So far, we used **fixed (hardcoded) values**
- Real programs should work with **dynamic data**
- User input allows programs to:
  - Take values from keyboard
  - Work for different users
  - Solve real-world problems

# Why Do We Need User Input?

- So far, we used **fixed (hardcoded) values**
- Real programs should work with **dynamic data**
- User input allows programs to:
  - Take values from keyboard
  - Work for different users
  - Solve real-world problems

**Key Idea**

Programs should not assume values — they should **ask the user**.

# User Input: Python vs Java

| **Python** | **Java** |
|---|---|
| input() function | Scanner class |
| Simple | More structured |
| Dynamic typing | Strict data types |

# User Input: Python vs Java

| Python | Java |
|---|---|
| input() function | Scanner class |
| Simple | More structured |
| Dynamic typing | Strict data types |

**Important**

Java does NOT have a direct `input()` function like Python.

# Taking Input in Java – Scanner Class

Java uses the **Scanner** class to take input from the user.

**Steps to use Scanner:**

1. Import Scanner class
2. Create Scanner object
3. Read input values

# Taking Input in Java – Scanner Class

Java uses the **Scanner** class to take input from the user.

**Steps to use Scanner:**

1. Import Scanner class
2. Create Scanner object
3. Read input values

## Analogy

Scanner is like a **measuring instrument** Keyboard is the **input source**

# Step 1: Import Scanner Class

To use Scanner, we must import it:

```
import java.util.Scanner;
```

# Step 1: Import Scanner Class

To use Scanner, we must import it:

```
import java.util.Scanner;
```

- Scanner belongs to `java.util` package
- Without import, Java will give an error

# Step 2: Create Scanner Object

We create a Scanner object to read input:

```
Scanner sc = new Scanner(System.in);
```

# Step 2: Create Scanner Object

We create a Scanner object to read input:

```
Scanner sc = new Scanner(System.in);
```

- `sc` → Scanner object
- `System.in` → Keyboard input

# Step 3: Reading Input Values

Scanner provides different methods:

| **Method** | **Data Type** |
|---|---|
| nextInt() | int |
| nextDouble() | double |
| next() | String (single word) |
| nextLine() | String (full line) |

# Step 3: Reading Input Values

Scanner provides different methods:

| Method | Data Type |
|---|---|
| nextInt() | int |
| nextDouble() | double |
| next() | String (single word) |
| nextLine() | String (full line) |

**Note**

Java input depends strictly on the data type.

# Example: Reading Integer Input

```
import java.util.Scanner;

public class UserInputDemo {
 public static void main(String[] args) {

  Scanner sc = new Scanner(System.in);

  System.out.print("Enter your age: ");
  int age = sc.nextInt();

  System.out.println("Your age is: " + age);
 }
}
```

# How This Program Works

1. Program asks user for input
2. User enters value using keyboard
3. Scanner reads the value
4. Value is stored in a variable
5. Program uses the value

# How This Program Works

1. Program asks user for input
2. User enters value using keyboard
3. Scanner reads the value
4. Value is stored in a variable
5. Program uses the value

## Key Thinking

Java programs wait for user input before continuing.

# Example

```
import java.util.Scanner;

public class BeamArea {
 public static void main(String[] args) {

  Scanner sc = new Scanner(System.in);

  System.out.print("Enter length of beam: ");
  double length = sc.nextDouble();

  System.out.print("Enter width of beam: ");
  double width = sc.nextDouble();

  double area = length * width;

  System.out.println("Area of beam = " + area);
 }
}
```

# next() vs nextLine()

- `next()` $\rightarrow$ Reads only one word
- `nextLine()` $\rightarrow$ Reads full sentence

# next() vs nextLine()

- `next()` → Reads only one word
- `nextLine()` → Reads full sentence

```
String name = sc.next();       // Premanand
String name = sc.nextLine();   // Premanand S
```

# next() vs nextLine()

- `next()` → Reads only one word
- `nextLine()` → Reads full sentence

```
String name = sc.next();       // Premanand
String name = sc.nextLine();   // Premanand S
```

**Warning**

This is a very common beginner mistake.

# Example: next() Method

```
import java.util.Scanner;

public class NextExample {
 public static void main(String[] args) {

  Scanner sc = new Scanner(System.in);

  System.out.print("Enter your name: ");
  String name = sc.next();

  System.out.println("Name entered: " + name);
 }
}
```

# Example: nextLine() Method

```
import java.util.Scanner;

public class NextLineExample {
 public static void main(String[] args) {

  Scanner sc = new Scanner(System.in);

  System.out.print("Enter your full name: ");
  String name = sc.nextLine();

  System.out.println("Full name entered: " + name);
 }
}
```

# Common Input Trap

```
import java.util.Scanner;

public class InputTrap {
 public static void main(String[] args) {

  Scanner sc = new Scanner(System.in);

  System.out.print("Enter age: ");
  int age = sc.nextInt();

  System.out.print("Enter name: ");
  String name = sc.nextLine();   // Problem

  System.out.println("Age: " + age);
  System.out.println("Name: " + name);
 }
}
```

# Why Does This Error Occur?

- `nextInt()` reads only the number
- The Enter key remains in the input buffer
- `nextLine()` immediately reads that leftover newline

# Why Does This Error Occur?

- `nextInt()` reads only the number
- The Enter key remains in the input buffer
- `nextLine()` immediately reads that leftover newline

**Result**

The name input appears to be skipped.

# Correct Way to Fix the Problem

```
System.out.print("Enter age: ");
int age = sc.nextInt();
sc.nextLine();   // Clear buffer

System.out.print("Enter name: ");
String name = sc.nextLine();
```

# Correct Way to Fix the Problem

```
System.out.print("Enter age: ");
int age = sc.nextInt();
sc.nextLine();   // Clear buffer

System.out.print("Enter name: ");
String name = sc.nextLine();
```

## Key Rule

After numeric input, always clear the buffer before using `nextLine()`.

# Corrected Full Example: nextInt() + nextLine()

```
import java.util.Scanner;

public class CorrectInputExample {
 public static void main(String[] args) {

  Scanner sc = new Scanner(System.in);

  System.out.print("Enter age: ");
  int age = sc.nextInt();

  sc.nextLine();

  System.out.print("Enter full name: ");
  String name = sc.nextLine();

  System.out.println("Age: " + age);
  System.out.println("Name: " + name);
 }
}
```

# Golden Rule for Scanner Input

After `nextInt()`, `nextDouble()`, etc.,
always use one extra `nextLine()`
before reading string input.

# Thank You!

## Stay Connected


## Premanand S

**Email:** premanand.s@vit.ac.in
**Phone:** +91-7358679961

**LinkedIn:** linkedin.com/in/premsanand
**Instagram:** instagram.com/premsanand
**WhatsApp Channel:** anandsDataX

**Google Scholar:** Google Scholar Profile
**GitHub:** github.com/anandprems