ASSIGNMENT – 6

**Aim**: Understanding the Syntactical Details of Solidity Language

# Theory:

Solidity is a high-level, statically typed programming language designed for writing smart contracts on the Ethereum blockchain. It was influenced by popular programming languages like JavaScript, C++, and Python. As a contract-oriented language, Solidity enables developers to encode logic for decentralized applications (dApps) and automate interactions on the blockchain.

# Code:

1. **Version Pragma:** Specifies the compiler version. pragma solidity ^0.8.0;
2. **Contract Declaration:** Similar to a class in OOP. contract MyContract {

// Code goes here

}

1. **State Variables:** Stored permanently on the blockchain. uint public myNumber;

address owner;

# Data Types

* + **Value Types**: uint, int, bool, address, bytes
  + **Reference Types**: arrays, structs, mappings

1. **Functions:** Perform operations, can be internal or external. function setNumber(uint \_number) public {

myNumber = \_number;

}

1. **Modifiers:** Add conditions or logic before function execution. modifier onlyOwner() {

require(msg.sender == owner, "Not the owner");

\_;

}

1. **Events:** Facilitate logging on the blockchain. event NumberSet(uint newNumber);
2. **Constructors:** Initialize state variables at deployment. constructor() {

owner = msg.sender;

}

1. **Payable Functions:** Allow receiving Ether. function deposit() public payable {}

# Visibility Specifiers

* + Public
  + Private
  + Internal
  + External.

# Memory and Storage

* + memory: Temporary, used for variables in functions.
  + storage: Persistent, used for state variables.

# Control Structures

* + if, else, for, while, break, continue.

# Error Handling

* + require, assert, revert.

require(balance >= amount, "Insufficient balance");

# Output:

