**3D Part 1: Libraries**

**📄 Create a file:**

CopyEdit

LibrariesDemo.sol

**✅ Paste this code:**

solidity

CopyEdit

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

library Search {

function indexOf(uint[] storage self, uint value) internal view returns (uint) {

for (uint i = 0; i < self.length; i++) {

if (self[i] == value) {

return i;

}

}

return type(uint).max; // Not found

}

}

contract TestSearch {

uint[] data;

constructor() {

data.push(1);

data.push(2);

data.push(3);

data.push(4);

data.push(5);

}

function isValuePresent(uint value) external view returns (uint) {

uint index = Search.indexOf(data, value);

return index;

}

}

library MathLibrary {

function square(uint num) internal pure returns (uint) {

return num \* num;

}

}

contract SquareContract {

using MathLibrary for uint;

function calculateSquare(uint num) external pure returns (uint) {

return num.square();

}

}

**🧪 Deploy and Test:**

✅ Deploy TestSearch

* Call isValuePresent(4) — should return 3.
* Call isValuePresent(99) — should return a very large number (not found).

✅ Deploy SquareContract

* Call calculateSquare(7) — should return 49.

🎉 Libraries done!

**🎯 3D Part 2: Inline Assembly**

**📄 Create:**

CopyEdit

AssemblyDemo.sol

**✅ Paste this:**

solidity

CopyEdit

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

library SumLibrary {

function sumUsingInlineAssembly(uint[] memory \_data) public pure returns (uint sum) {

for (uint i = 0; i < \_data.length; ++i) {

assembly {

let value := mload(add(add(\_data, 0x20), mul(i, 0x20)))

sum := add(sum, value)

}

}

}

}

contract TestAssembly {

uint[] data;

constructor() {

data.push(1);

data.push(2);

data.push(3);

data.push(4);

data.push(5);

}

function getSum() external view returns (uint) {

return SumLibrary.sumUsingInlineAssembly(data);

}

}

**🧪 Deploy and Test:**

✅ Deploy TestAssembly

* Call getSum() — should return 15.

🎉 Assembly done!

**🎯 3D Part 3: Events**

**📄 Create:**

CopyEdit

EventsDemo.sol

**✅ Paste this:**

solidity

CopyEdit

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract EventExample {

event Deposit(address indexed from, uint256 amount);

event Withdraw(address indexed to, uint256 amount);

mapping(address => uint256) public balances;

function deposit() public payable {

require(msg.value > 0, "Must deposit more than 0 ether");

balances[msg.sender] += msg.value;

emit Deposit(msg.sender, msg.value);

}

function withdraw(uint256 amount) public {

require(balances[msg.sender] >= amount, "Insufficient balance");

balances[msg.sender] -= amount;

payable(msg.sender).transfer(amount);

emit Withdraw(msg.sender, amount);

}

}

**🧪 Deploy and Test:**

✅ Deploy EventExample

**Test deposit:**

1. In Remix:
   * In **Value**, enter e.g. 1 ether
2. Click deposit()
3. Check logs—should see a Deposit event.

**Test withdraw:**

1. Call withdraw(1000000000000000000) (1 ether in wei)
2. Should emit Withdraw event.

🎉 Events done!

**🎯 3D Part 4: Error Handling**

**📄 Create:**

CopyEdit

ErrorHandlingDemo.sol

**✅ Paste this:**

solidity

CopyEdit

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract ErrorHandlingExample {

constructor() payable {

// Allow the contract to receive Ether during deployment

}

function divide(uint256 numerator, uint256 denominator) external pure returns (uint256) {

require(denominator != 0, "Division by zero is not allowed");

return numerator / denominator;

}

function withdraw(uint256 amount) external {

require(amount <= address(this).balance, "Insufficient balance");

payable(msg.sender).transfer(amount);

}

function assertExample() external pure {

uint256 x = 5;

uint256 y = 10;

assert(x < y);

}

function tryCatchExample() external view returns (bool) {

try this.divide(10, 5) returns (uint256) {

return true;

} catch Error(string memory) {

return false;

}

}

}

**🧪 Deploy and Test:**

✅ Deploy ErrorHandlingExample (make sure to deploy with some ether if you plan to test withdraw()).

✅ Call divide(10,0)—should **revert** with error Division by zero.

✅ Call tryCatchExample()—should return true.

✅ Call assertExample()—should succeed.

🎉 Error handling done!