Practical 3 Code -

//SPDX-License-Identifier:MIT

pragma solidity >=0.5.0 <0.9.0;

contract EtherWallet{

address payable public owner;

constructor(){

owner = payable(msg.sender); // by default msg.sender is not payable so we cast it

}

function Withdraw(uint \_amount) public {

require(msg.sender == owner,"Only the owner can invoke this function");

payable(msg.sender).transfer(\_amount);

}

function getBalance() external view returns(uint){

return address(this).balance;// return balance in this contract

}

receive() external payable {} // default function came after sol version 0.6.0 that allows contract to receive funds

}

Practical 4 Code -

//SPDX-License-Identifier:MIT

pragma solidity >=0.4.0<=0.9.0;

contract StudentRegister{

address public owner;

mapping (address=>student)students;

constructor() {

owner=msg.sender;

}

modifier onlyOwner {

require(msg.sender==owner);

\_;

}

struct student{

address studentId;

string name;

string course;

uint256 mark1;

uint256 mark2;

uint256 mark3;

uint256 totalMarks;

uint256 percentage;

bool isExist;

}

function register(address studentId,string memory name,string memory course,uint256 mark1,uint256 mark2,uint256 mark3) public onlyOwner {

require(students[studentId].isExist==false,"Fraud Not Possible,student details already registered and cannot be altered");

uint256 totalMarks;

uint256 percentage;

totalMarks=(mark1+mark2+mark3);

percentage=(totalMarks/3);

students[studentId]=student(studentId,name,course,mark1,mark2,mark3,totalMarks,percentage,true);

}

function getStudentDetails(address studentId) public view returns (address,string memory,string memory,uint256,uint256){

return(students[studentId].studentId,students[studentId].name,students[studentId].course,students[studentId].totalMarks,students[studentId].percentage);

}

}