```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract shipping {
    address public buyer;
    address public seller;
    address public carrier;
    uint public price;
    uint public  deliverytime;
    bool public delivered;

    constructor(address _buyer, address _seller, uint _price, uint
_deliverytime){
        buyer = _buyer;
        seller = _seller;
        deliverytime = _deliverytime;
        price = _price;
    }
    function assignCarrier(address _carrier) public {
        require(msg.sender==seller,"only seller can assign the carrier");
        carrier = _carrier;
    }

    function confirmDelivery() public {
        require(msg.sender==buyer,"only buyer can confirm delivery");
        require(carrier!=address(0),"assign carrier before delivery can be
confirmed");
        require(block.timestamp >= deliverytime);
        delivered = true;
    }

    function withdraw() public payable {
        require(msg.sender==seller || msg.sender==carrier);
        require(delivered==true);
        uint amount = price;
        payable(msg.sender).transfer(amount);
    }
}
```

EXP:2)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.8;

struct Book {
    string name;
    string writter;
    uint id;
    bool available;
}

contract Types {
    uint256 public number;

    constructor() {
        number = 10;
    }

    function getResult() public pure returns (uint256){
        uint256 number2 = 10;
        uint256 number3 = 20;
        uint256 result = number2+number3;
        return result;
    }

    Book book1;

    function setBookDetails_Book1() public {
        book1 = Book("Building Ethereum DApps", "Roberto Infante",2,false);
    }

    Book book2;

    function setBookDetails_Book2() public {
        book2 = Book("Intro to Ethereum and Solidity","Alex",3,true);
    }

    function book1_info() public view returns (string memory, string memory,
uint256, bool){
        return(book1.name,book1.writter,book1.id,book1.available);
    }

    function book2_info() public view returns (string memory, string memory,
uint256, bool){
        return(book2.name,book2.writter,book2.id,book2.available);
    }
}
```

EXP-3A)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

contract Arr {
    uint256[] private arr;

    function addData(uint256 num) public {
        arr.push(num);
    }

    function getData() public view returns(uint256[] memory) {
        return arr;
    }

    function getLength() public view returns(uint256) {
        return arr.length;
    }

    function search(uint IndexNumber) public view returns(uint256){
        require(IndexNumber <= arr.length,"Please enter index number within
range");

        return arr[IndexNumber];

    }

    function getSum() public view returns(uint256){
        uint sum = 0;
        for(uint index = 0; index < arr.length; index++){
            sum = sum + arr[index] ;
        }
        return sum;
    }


}
```

EXP-3B)

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

struct student{
        string name;
        string subject;
        uint8 marks;

    }

contract Mapping {

    mapping (address => student) public results;
    address[] public students;

    function set(address stuAdd, string memory name, string memory subject, 
uint8 marks) public {
        results[stuAdd] = student(name,subject,marks);
        students.push(stuAdd);
    }

    function getStudents() public view returns(uint256){
        return students.length;
    }

}
```

EXP:4)

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

contract Exp4 {
    uint256 number;
    string value;

    function set(uint num) public {
        number = num;
        if (number % 2 == 0){
            value = "even";
        }
        else {
            value ="odd";
        }
    }

    function get() public  view returns(string memory){
        return value;
    }
}
```

EXP:5)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

contract Hotel {
    enum status{vaccant, occupied}
    status currentStatus;
    address payable public owner;
    event occupy(address _occupant, uint value);

    constructor() {
        owner = payable (msg.sender);
        currentStatus = status.vaccant;
    }

    modifier onlyWhileVacant {
        require(currentStatus == status.vaccant);
        _;
    }

    modifier costs(uint _amount){
        require(msg.value >= _amount);
        _;
    }

    function book() public payable onlyWhileVacant costs(2 ether) {
        currentStatus = status.occupied;
        owner.transfer(msg.value);
        emit occupy(msg.sender, msg.value);
    }

    function getStatus() public view returns(status){
        return currentStatus;
    }

    receive() external payable { }

}
```

EXP:6)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

contract A {
    uint256 internal a;
    function setA(uint256 _value) external  {
        a = _value;
    }
}

contract B {
    uint256 internal b;
    function setB(uint256 _value1) external {
        b = _value1;
    }
}

contract C is A,B {
    function getSum() public view returns(uint256){
        return a+b;
    }
}

// contract caller {
//     C = new C();
//     function testInheritance() public  returns(uint256){
//         c.setA(10);
//         c.setB(20);
//         return c.getSum();

//     }

// }
```

Exp:7)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

contract MarketPlace {

    struct Item {
        uint id;
        string name;
        uint price;
        address payable owner;
        bool sold;
    }

    uint public itemCount;
    address payable public owner;

    mapping(uint => Item) public items;

    constructor() {
        owner = payable(msg.sender);
    }

    modifier onlyOwner(){
        require(msg.sender == owner);
        _;
    }

    event ItemAdded(uint id, string name, uint price, address payable owner);
    event ItemSold(uint id, address buyer, uint price);

    function addItem(string memory _name, uint _id, uint _price) public
onlyOwner {
        itemCount++;
        items[itemCount] = Item(_id, _name,_price,payable(msg.sender),false);
        emit ItemAdded(_id, _name, _price, owner);

    }

    function buyItem(uint _id) public payable {
        require(items[_id].id>0 && items[_id].id<=itemCount,"Invalid Item
Id");
        require(!items[_id].sold,"Item sold");
        require(msg.value >= items[_id].price);
        // items[_id].owner.transfer(items[_id].price);
        owner.transfer(msg.value);
        items[_id].sold = true;
        emit ItemSold(_id, msg.sender, items[_id].price);
    }

    function getItemStatus(uint256 _id) public view returns(bool){
        return items[_id].sold;
    }

    receive() external payable { }
}
```