Solidity Contract for Enums

```
pragma solidity ^0.5.0;

contract Example {
  // creating an enum
  enum Button {ON, OFF}

  // declaring a variable of type enum
  Button button;

  // function to turn on the button
  function buttonOn() public {
    // set the value of button to ON
    button = Button.ON;
  }

  // function to turn off the button
  function buttonOff() public {
    // set the value of button to OFF
    button = Button.OFF;
  }

  // function to get the value of the button
  function getbuttonState() public view returns(Button) {
    // return the value of button
    return button;
  }
}
```

Solidity Contract for addition of uint Value types

```
pragma solidity ^0.5.0;
contract SolidityTest {

   function getResult() public view returns(uint){
      uint a = 1;
      uint b = 2;
      uint result = a + b;
      return result;
   }
}
```

Solidity Contract for String Value types

```
pragma solidity >=0.4.16 <0.8.0;
contract FirstContract{
    string public _name;
    function setName(string memory name) public{
        _name = name;
    }
    function getName() view public returns(string memory){
        return _name;
    }
 }
```

```
pragma solidity ^0.5.0;
contract LearningStrings
{
    string public text;
    // Assigning the text directly
    function setText() public
    {
        text = 'hello';
    }
    // Assigning the text by passing the value in the function
    function setTextByPassing(string memory message) public
    {
        text = message;
    }
    // Function to get the text
    function getText() view public returns (string memory)
    {
        return text;
    }
}
```

Solidity Contract for HelloWorld

```
pragma solidity >=0.4.16 <0.8.0;
contract helloWorld {
  string public hello = "Hello World!";
}
```

Solidity Contract for Dynamic Array

```
pragma solidity ^0.8.13;

contract Array {
    // Several ways to initialize an array
    uint[] public arr;

    // Solidity can return the entire array.
    // But this function should be avoided for
    // arrays that can grow indefinitely in length.
    function getArr() public view returns (uint[] memory) {
        return arr;
    }

    function push(uint i) public {
        // Append to array
        // This will increase the array length by 1.
        arr.push(i);
    }

    function pop() public {
        // Remove last element from array
        // This will decrease the array length by 1
        arr.pop();
    }

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

    function remove(uint index) public {
        // Delete does not change the array length.
        // It resets the value at index to it's default value,
        // in this case 0
        delete arr[index];
    }


}
```

Solidity Contract for Structure

```
pragma solidity ^0.5.0;

contract test {
   struct Book {
      string title;
      string author;
      uint book_id;
   }
   Book book;

   function setBook() public {
      book = Book('Learn Java', 'TP', 1);
     // book = Book('Learn Python', 'PS', 2);
   }
   function getBookId() public view returns (uint) {
      return book.book_id;
   }
}
```

Solidity Contract for Array of Structure

```
pragma solidity ^0.8.0;

contract Collections
{
    struct user
    {
        string Name;
        string Address;

    }

    user[] users;

    function setUser(string calldata _name, string calldata _address) public
    {
        users.push(  // resize the array and store new item
        user(         // of type `student`
            _name,
            _address
        )
     );
    }

    function getUser1()public view returns(string[] memory, string[] memory){
       string[] memory Name = new string[](users.length);
        string[] memory Address = new string[](users.length);

        for(uint i=0; i<users.length; i++){
            Name[i] = users[i].Name;
            Address[i] = users[i].Address;
        }
        return(Name, Address);
    }

}
```

Solidity Contract for Array of Structure

```
pragma solidity >=0.7.0 <0.9.0;

contract studentRecord
{
    address owner;

    constructor()
    {
        owner = msg.sender;
    }

    struct student
    {
        string Name;
        string Address;

    }

    student[] public StudentRecord;

    function setStudentRecords(string calldata _name, string calldata _address)
public
    {
        StudentRecord.push(  // resize the array and store new item
        student(          // of type `student`
            _name,
            _address
        )
     );
    }


    function GetStudentRecord(uint index) public view returns(student memory)
    {
        return StudentRecord[index];
    }

    function studentCount() public view returns(uint)
    {
        return StudentRecord.length;
    }
}
```

Solidity Contract for Function Visibility: Private

```
pragma solidity ^0.8.0;

contract VisibilitySpecifier{


    function testAccessiblePrivate() public pure returns(uint){
       return privateFun();
    }

    function privateFun() private pure returns(uint) {
        return 30;
    }
}
```

Solidity Contract for Function Visibility: Internal

```
pragma solidity ^0.8.0;

contract Base {
    // Internal function can be called
    // - inside this contract
    // - inside contracts that inherit this contract
    function internalFunc() internal pure returns (uint) {
        return 25;
    }
   function testInternalFunc() public pure virtual returns (uint) {
    return internalFunc();
   }
}

contract Child is Base {

    // Internal function call be called inside child contracts.
    function testInternalFunc() public pure override returns (uint) {
        return internalFunc();
    }
}
```

Solidity Contract for Function Visibility: Public

```
contract VisibilitySpecifier{

    //function testAccessiblePublic() public pure returns(uint){
   //     return publicFun();
    //}

    function publicFun() public pure returns(uint) {
        return 20;
    }
}
```

Solidity Contract for Function Visibility: External

```
pragma solidity ^0.8.0;

contract VisibilitySpecifier{


    // function testAccessibleExternal() public pure returns(uint){
    //    return externalFun();
    // }


    function externalFun() external pure returns(uint) {
        return 15;
    }
}

contract CheckExternalVisibility{
    VisibilitySpecifier ext = new VisibilitySpecifier();

    function testAccessibleExternal() public view returns(uint){
       return ext.externalFun();
    }
}
```

Solidity Contract for Function Visibility: Public with Inheritance

```
pragma solidity ^0.8.13;


contract BaseContract{

  function hello() public pure returns (string memory){
      return "Hello Motor";
  }
}


contract A is BaseContract {
    function sayHello() public pure returns (string memory){
        //we are calling the hello function in the BaseContract
        return hello();
    }

}
```

Solidity Contract for Mapping

```
pragma solidity ^0.8.4;

contract MyContract {
    mapping(uint => string) public names;
    constructor() public {
        names[101] = "Jon";
        names[102] = "Sara";
        names[103] = "Paul";
    }
}
```

Solidity Contract for Nested Mapping

```
pragma solidity ^0.6.0;

contract MyContract {
    // Mappings
    mapping(uint => string) public names;
    mapping(uint => Book) public books;
    mapping(address => mapping(uint => Book)) public myBooks;

    struct Book {
        string title;
        string author;
    }

    constructor() public {
        names[1] = "Adam";
        names[2] = "Bruce";
        names[3] = "Carl";
    }

    function addBook(uint _id, string memory _title, string memory _author) public
{
        books[_id] = Book(_title, _author);
    }

    function addMyBook(uint _id, string memory _title, string memory _author)
public {
        myBooks[msg.sender][_id] = Book(_title, _author);
    }
}
```

Solidity Contract for Fallback Function

```
pragma solidity >=0.7.0 <0.9.0;

contract A{
    fallback() external{
      return 5;
    }
}

contract B{
    function foo() external {
        A a = new A();
        a.functionThatDoesNotExists();
    }
}
```

Solidity Contract for Single Inheritance

```
pragma solidity ^0.8.0;

contract A {
    uint public a;
    constructor(){
        a=100;
    }
    function funA() public{
        a=10;
    }
}

contract B is A{
    uint public b;
    constructor(){
        b=100;
    }
    function funB() public{
        b=10;
    }
}
```

Solidity Contract for Multiple Inheritance

```
pragma solidity ^0.8.0;

contract A {
    uint public a;
    constructor(){
        a=100;
    }
    function funA() public{
        a=10;
    }
}

contract B{
    uint public b;
    constructor(){
        b=100;
    }
    function funB() public{
        b=10;
    }
}

contract C is A,B{

}
```

Solidity Contract for Multi-Level Inheritance

```
pragma solidity ^0.8.0;

contract A {
    uint public a;
    constructor(){
        a=100;
    }
    function funA() public{
        a=10;
    }
}

contract B is A{
    uint public b;
    constructor(){
        b=100;
    }
    function funB() public{
        b=10;
    }
}

contract C is B{
    uint public c;
    constructor(){
        c=100;
    }
    function funC() public{
        c=10;
    }
}
```

Solidity Contract for Hierarchical Inheritance

```
pragma solidity ^0.8.0;

contract A {
    uint public a;
    constructor(){
        a=100;
    }
    function funA() public{
        a=10;
    }
}

contract B is A{
    uint public b;
    constructor(){
        b=100;
    }
    function funB() public{
        b=10;
    }
}

contract C is A{
    uint public c;
    constructor(){
        c=100;
    }
    function funC() public{
        c=10;
    }
}
```

Solidity Contract for Abstract Contract

```
pragma solidity ^0.5.0;

contract A {
   function getResult() public view returns(uint);
}
contract B is A {
   function getResult() public view returns(uint) {
      uint a = 1;
      uint b = 2;
      uint result = a + b;
      return result;
   }
}
```

Solidity Contract for Interface

```
pragma solidity ^0.8.13;

interface Base {
    function get() pure external returns (uint);
}

contract MyContract is Base{
    function get() public pure override returns (uint) {
        return 10;
    }
}
```

Solidity Contract for Modifier (user defined)

```
pragma solidity ^0.7.0;

//contract name is MyFirstContract
contract MyFirstContract {

//create two variables.  A sting and an address

    address owner;
    string private name;

//constructor sets the creator of the contract to the owner variable
   constructor() {
      owner = msg.sender;
   }


//modifier checks that the caller of the function is the owner
    modifier onlyOwner() {
         require(msg.sender == owner, 'Not Owner');
         _;
    }

//set name.  Only the owner of the contract can call because a modifier is
specified
    function setName(string memory newName) public onlyOwner{
        name = newName;
    }

//get the name
    function getName () public view returns (string memory) {
        return name;
    }


}
```

Solidity Contract for Event

```
pragma solidity ^0.4.21;

// Creating a contract
contract eventExample {

    // Declaring state variables
    uint256 public value = 0;

    // Declaring an event
    event Increment(address owner);

    // Defining a function for logging event
    function getValue(uint _a, uint _b) public {
        emit Increment(msg.sender);
        value = _a + _b;
    }
}
```

Solidity Contract for Error Handling

```
pragma solidity ^0.8.13;

contract ErrorTest{
    function TestRevert(uint n) external pure{
        if(n>20){
            revert('Value cannot greater than 20');
        }
    }

    function TestRequire(uint n) external pure{
        require(n<20,'N shound be less than 20 ');
    }

    function TestAssert(uint n) external pure {
        assert(n<20);
    }
}
```

Solidity Contract for Contract Communication

```
pragma solidity ^0.8.0;

contract Calculator{
    function add(int a, int b) external pure returns(int){
        return a+b;
    }

    function multiply(int a, int b) external pure returns(int){
        return a*b;
    }
}

contract Foo{
    Calculator calc = new Calculator();

    function FourTimesSix() external view returns(int){
        return calc.multiply(4,6);
    }

    function FourPlusSix() external view returns(int){
        return calc.add(4,6);
    }
}
```

Solidity Contract for Lottery App

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract Lottery {
    address payable public manager;
    address payable[] public players;
    uint public lotteryId;
    mapping (uint => address payable) lotteryHistory;

    constructor() public payable {
        manager = payable(msg.sender);
        lotteryId = 1;
    }

    function enter() public payable {
        require((msg.value) > 1 ether);
        players.push(payable(msg.sender));
    }

     function getBalance() public view returns (uint) {
        return address(this).balance;
    }

    function random() private view returns(uint) {
       return uint(keccak256(abi.encodePacked(block.difficulty, block.timestamp,
players)));
    }

    function winner() public payable restricted {
        uint index = random() % players.length;
        players[index].transfer(address(this).balance);
        lotteryHistory[lotteryId] = players[index];
        lotteryId++;
        players=new address payable [](0);
    }

    function allplayers() public view returns(address payable[] memory) {
        return players;
    }
```

```
    function getWinnerByLottery(uint lottery) public view returns (address
payable) {
        return lotteryHistory[lottery];
    }

    modifier restricted() {
        require(msg.sender == manager);
        _;
    }
}
```