Blockchain Lab

Assignment-1

Name: Aditya Pathak
Subgroup: 3NC2
Roll Number: 102115044

1.

```
// SPDX-License-Identifier: GPL-3.0

//GPL -> General Public License Version 3.0
//GPL License basically means that the code is open source and must adhere to this license

pragma solidity >=0.4.16 <0.9.0;
//Pragma is basically a statement used to instruct the compiler about how to treat the source code.
//Here, pragma is basically defining the versions of solidity for which the code would work.


// A contract in solidity is similar to a class in OOPS
contract basicDataTypesAndArrays
{
    uint unsignedIntegerVal;


    int integerVal;


    bool booleanVal;


    string stringVal;

     //The address type is used to store 20byte addresses of nodes on the ethereum blockchain
     //Since, smart contracts also have their own unique addresses, the address may be of a contract or it may be the
     //public key of an accouunt on the ethereum blockchain.
    address hexadecimalAddressVal;
    // The address type is a 160-bit value
    // that does not allow any arithmetic operations.
    // 0x8685De4413c68DCf55dE6A566252517B2696affB



    function setVals(uint val1,int val2,bool val3, string memory val4, address val5) public     infinite gas
    {
        unsignedIntegerVal=val1;
        integerVal=val2;
        booleanVal=val3;
        stringVal=val4;
        hexadecimalAddressVal=val5;
    }

    function getVals() public view returns(uint,int,bool,string memory,address)     infinite gas
    {
        return (unsignedIntegerVal,integerVal,booleanVal,stringVal,hexadecimalAddressVal);
    }

    uint[3] fixedSizeArray=[1,3,5];

    function getFixedArrayVal(uint index) public view returns(uint)     infinite gas
```

```
    {
        uint val=fixedSizeArray[index];
        return val;
    }

    int[] dynamicArray=[-6,14,57,-64];

    function getDynamicArrayVal(uint index)public view returns(int)     4882 gas
    {
        return dynamicArray[index];
    }
}
```

2.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.4.16 <0.9.0;

contract functionBasics
{
    uint public data1;
    uint private data2;

    //Public Function without returns.
    //Public functions are basically accesible outside the contract.
    function setval1(uint val1)public    44702 gas
    {
        data1=val1;
        setval2();
    }

    // Private function without returns. This function
    //  will not be visible when contract is deployed.
    function setval2()private    22123 gas
    {
        data2=99;
    }
    // View Modifier-----------------------------
    // The view modifier is used to indicate that a function will only read the state of the contract
    // and not modify it. It's essentially a promise that the function won't change the state.
    // Functions marked as view can be called without sending a transaction, meaning they can be
    // executed by other functions or contracts without consuming gas
    // VIEW->Transactionless and no state change.

    function fun1() public view returns(uint)    2437 gas
    {
        return data1;
    }
    function fun2() public view returns(uint)    2415 gas
    {
        return data2;
    }


    // Pure Modifier------------------------------
    // The pure modifier is even more restrictive than view. It indicates that a function does not read or modify
    // the state of the contract. It only works with the arguments provided to it and local variables
    // within the function. Functions marked as pure can also be called without sending a transaction.
    function returnSumPriv() private pure returns (uint)    36 gas
    {
        uint ans=55+47;
        return ans;
    }
    function returnSum(uint a,uint b) public pure returns (uint,uint)    infinite gas
    {
        return ( a+b,returnSumPriv() );//This returns the value of both the private and public return sum functions
    }
}
```

3.

```
// Stack.sol
// Smart contract for stack operations

pragma solidity ^0.8.0;

contract Stack {
    uint[] private stack;

    function push(uint value) public {    46829 gas
        stack.push(value);
    }

    function pop() public returns (uint) {    infinite gas
        require(stack.length > 0, "Stack is empty");
        uint value = stack[stack.length - 1];
        stack.pop();
        return value;
    }

    function getStack() public view returns (uint[] memory) {    infinite gas
        return stack;
    }
}

// Queue.sol
// Smart contract for queue operations

pragma solidity ^0.8.0;

contract Queue {
    uint[] private queue;

    function enqueue(uint value) public {    46829 gas
        queue.push(value);
    }

    function dequeue() public returns (uint) {    infinite gas
        require(queue.length > 0, "Queue is empty");
        uint value = queue[0];
        for (uint i = 0; i < queue.length - 1; i++) {
            queue[i] = queue[i + 1];
        }
        queue.pop();
        return value;
    }

    function getQueue() public view returns (uint[] memory) {    infinite gas
        return queue;
    }
}
```

4.

```
// ArrayOperations.sol
// Smart contract for multidimensional array operations

pragma solidity ^0.8.0;

contract ArrayOperations {
    uint[][] public matrix;

    function setMatrix(uint[][] memory _matrix) public {    infinite gas
        matrix = _matrix;
    }

    function getMatrix() public view returns (uint[][] memory) {    infinite gas
        return matrix;
    }

    function getElement(uint row, uint col) public view returns (uint) {    infinite gas
        return matrix[row][col];
    }
}
```

5.

```
// QuadraticSolver.sol
// Smart contract for solving quadratic equations

pragma solidity ^0.8.0;

contract QuadraticSolver {
    function solveQuadraticEquation(uint a, uint b, uint c) public pure returns (int, int) {    infinite gas
        int discriminant = int(b**2 - 4 * a * c);

        require(discriminant >= 0, "No real roots");

        int root1 = (-int(b) + discriminant) / (2 * int(a));
        int root2 = (-int(b) - discriminant) / (2 * int(a));

        return (root1, root2);
    }
}

// Caller.sol
// Smart contract for calling QuadraticSolver function

pragma solidity ^0.8.0;

contract Caller {
    QuadraticSolver public quadraticSolver;

    constructor(address _quadraticSolver) {    infinite gas 188000 gas
        quadraticSolver = QuadraticSolver(_quadraticSolver);
    }

    function callQuadraticSolver(uint a, uint b, uint c) public view returns (int, int) {    infinite gas
        return quadraticSolver.solveQuadraticEquation(a, b, c);
    }
}
```