

## 1. Hello World:

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is active. It displays a 'GAS LIMIT' of 3000000, a 'VALUE' of 0 wei, and the 'CONTRACT' 'HelloWorld - HelloWorld.sol'. A 'Deploy' button is visible, along with a 'Publish to IPFS' checkbox. Below, it shows 'Transactions recorded' (1) and 'Deployed Contracts' (1). The main editor on the right shows the 'HelloWorld.sol' contract code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.7;
3
4 contract HelloWorld {
5     string public myString = "hello world";
6 }
```

The bottom status bar shows '0' gas, a 'listen on network' checkbox, and a search bar for transaction hashes or addresses.

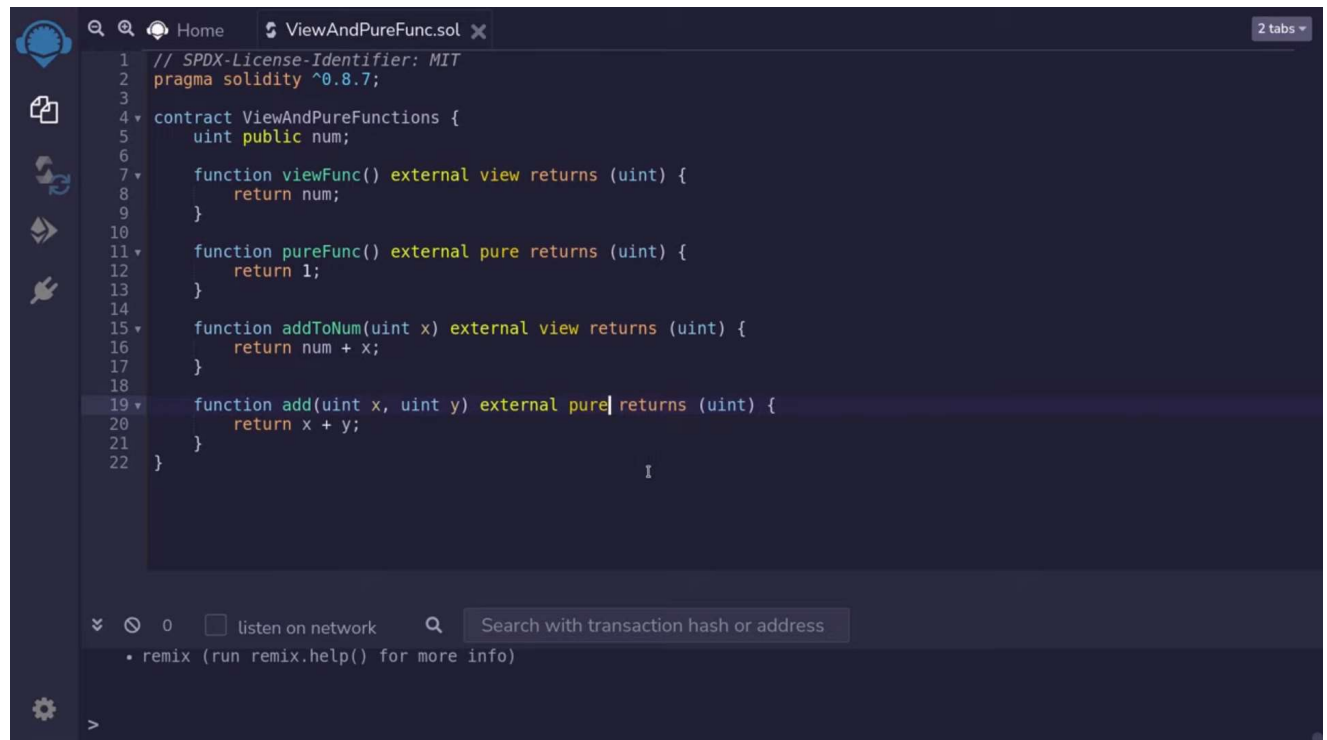
## 2. Data Types:

The screenshot shows the Remix IDE interface with the 'Data.sol' contract loaded. The main editor displays the following Solidity code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.7;
3
4 // Data types - values and references
5
6 contract ValueTypes {
7     bool public b = true;
8     uint public u = 123; // uint = uint256 0 to 2**256 - 1
9                          // uint8 0 to 2**8 - 1
10                         // uint16 0 to 2**16 - 1
11     int public i = -123; // int = int256 -2**255 to 2**255 - 1
12                        // int128 -2**127 to 2**127 - 1
13     int public minInt = type(int).min;
14     int public maxInt = type(int).max;
15     address public addr = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
16     bytes32 public b32 = 0x89c58ced8a9078bdef2bb60f22e58eeff7dbfed6c2dff3e7c508b629295926fa;
17 }
```

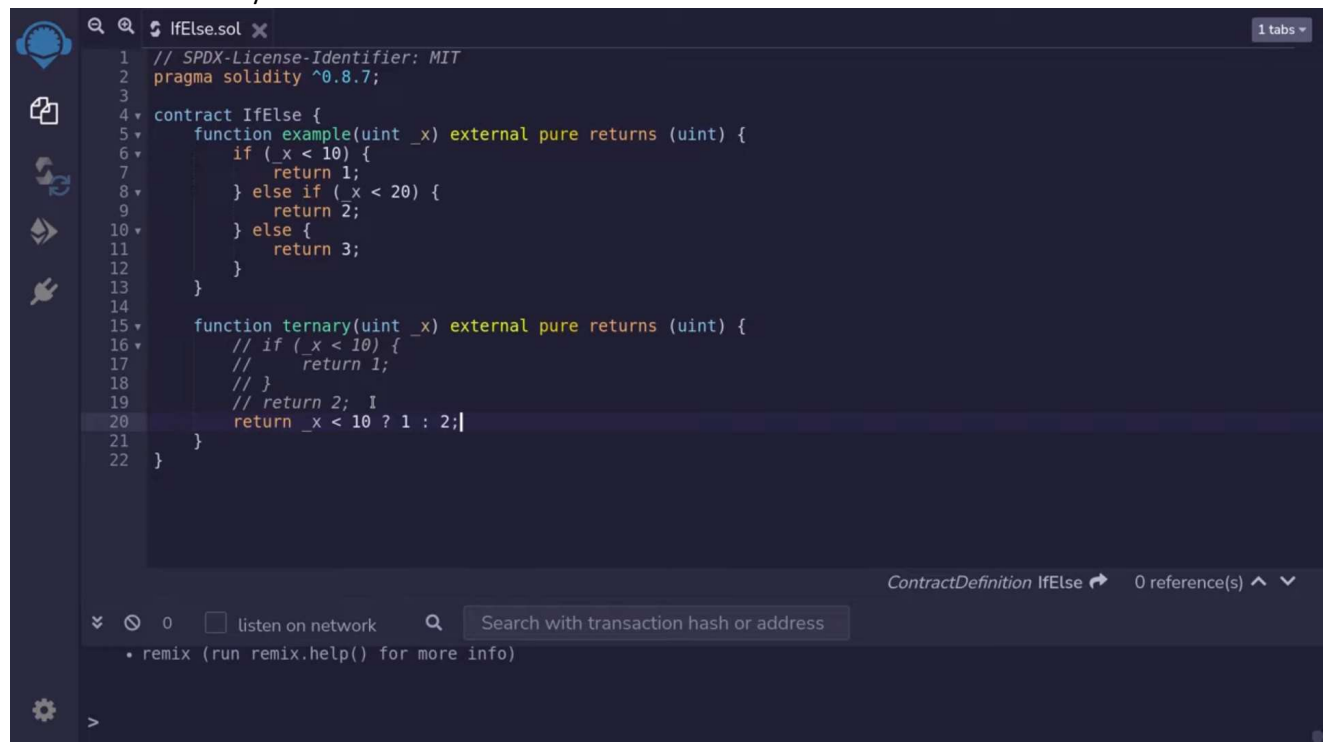
The bottom status bar shows '0' gas, a 'listen on network' checkbox, and a search bar for transaction hashes or addresses. A message at the bottom indicates 'remix (run remix.help() for more info)'.

### 3. Pure and View Functions:



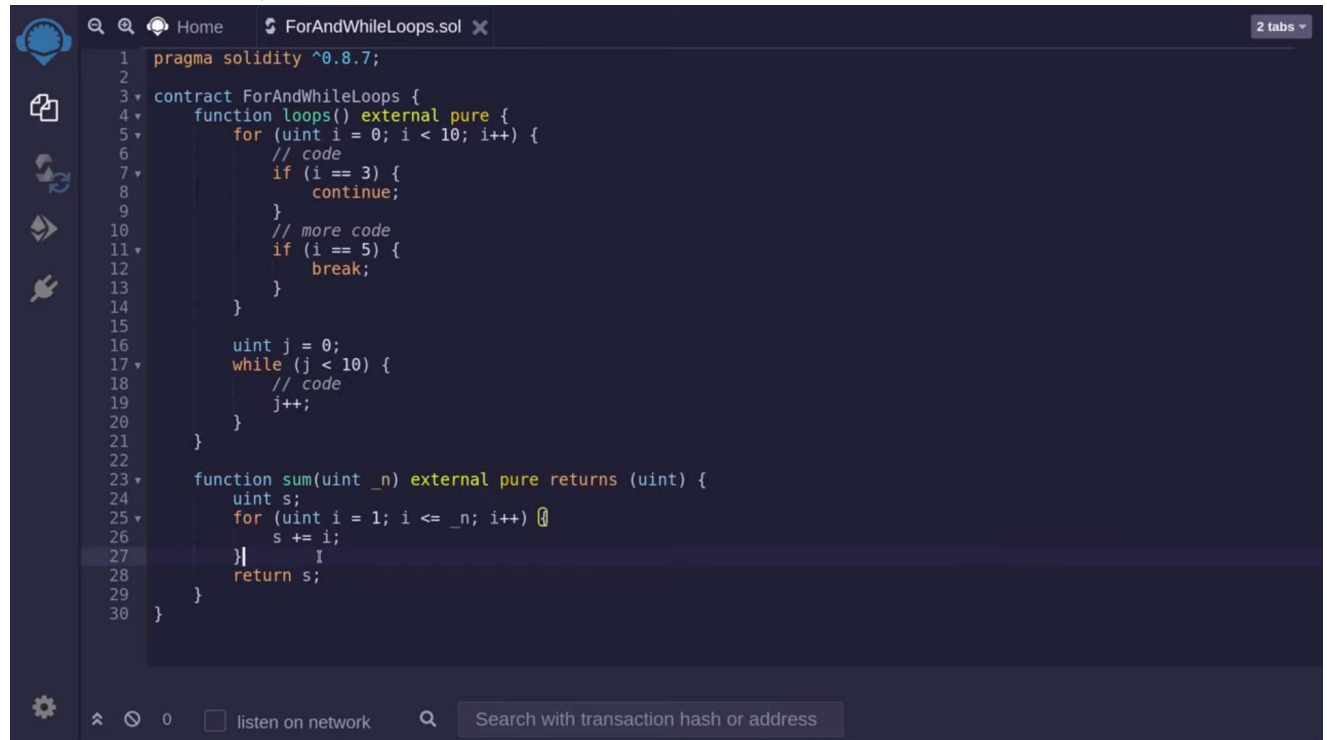
```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.7;
3
4 contract ViewAndPureFunctions {
5     uint public num;
6
7     function viewFunc() external view returns (uint) {
8         return num;
9     }
10
11    function pureFunc() external pure returns (uint) {
12        return 1;
13    }
14
15    function addToNum(uint x) external view returns (uint) {
16        return num + x;
17    }
18
19    function add(uint x, uint y) external pure returns (uint) {
20        return x + y;
21    }
22 }
```

### 4. If Else and Ternary:



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.7;
3
4 contract IfElse {
5     function example(uint _x) external pure returns (uint) {
6         if (_x < 10) {
7             return 1;
8         } else if (_x < 20) {
9             return 2;
10        } else {
11            return 3;
12        }
13    }
14
15    function ternary(uint _x) external pure returns (uint) {
16        // if (_x < 10) {
17        //     return 1;
18        // }
19        // return 2;
20        return _x < 10 ? 1 : 2;
21    }
22 }
```

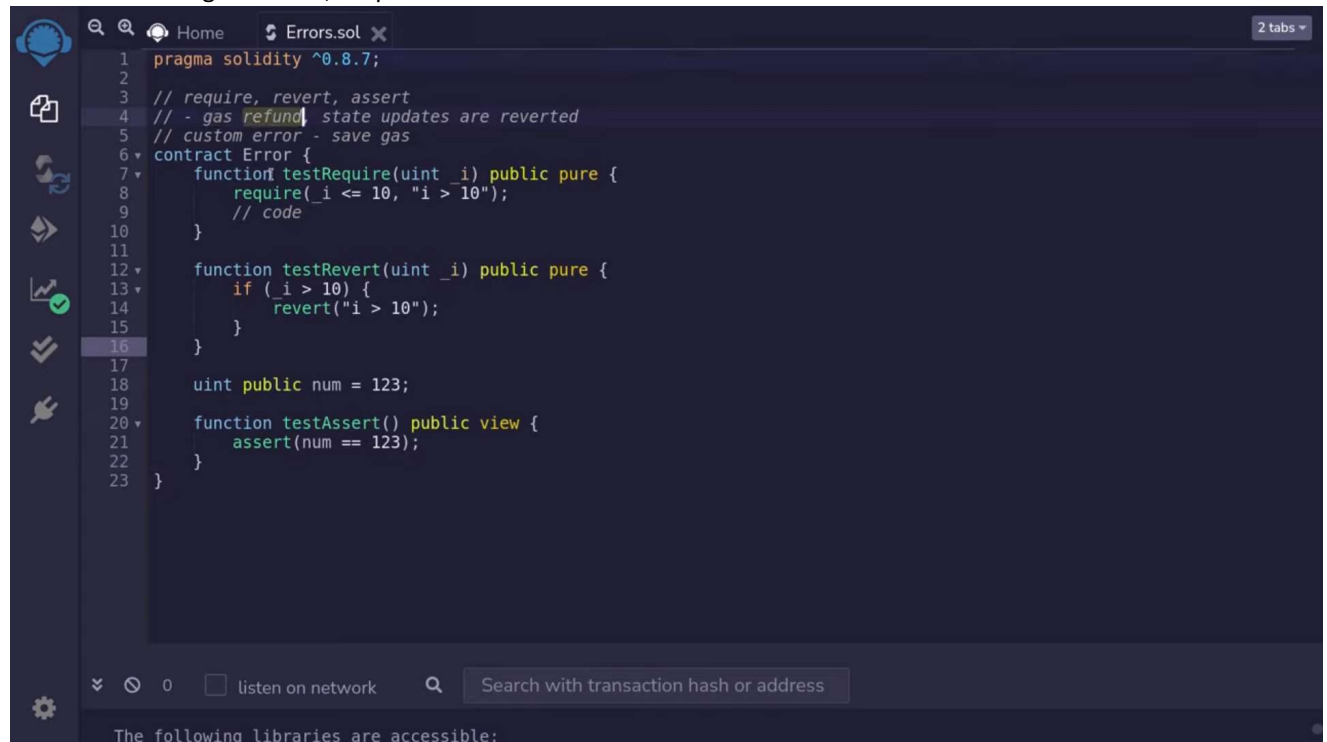
## 5. While and For Loop:



The screenshot shows a Solidity IDE with a file named 'ForAndWhileLoops.sol'. The code defines a contract 'ForAndWhileLoops' with two functions: 'loops()' and 'sum()'. The 'loops()' function uses a 'for' loop to iterate from 0 to 10, with a 'continue' statement at i=3 and a 'break' statement at i=5. It also uses a 'while' loop to iterate from 0 to 10. The 'sum()' function uses a 'for' loop to calculate the sum of numbers from 1 to n.

```
1 pragma solidity ^0.8.7;
2
3 contract ForAndWhileLoops {
4     function loops() external pure {
5         for (uint i = 0; i < 10; i++) {
6             // code
7             if (i == 3) {
8                 continue;
9             }
10            // more code
11            if (i == 5) {
12                break;
13            }
14        }
15
16        uint j = 0;
17        while (j < 10) {
18            // code
19            j++;
20        }
21    }
22
23    function sum(uint _n) external pure returns (uint) {
24        uint s;
25        for (uint i = 1; i <= _n; i++) {
26            s += i;
27        }
28        return s;
29    }
30 }
```

## 6. Error handling => Assert, Require and Revert:

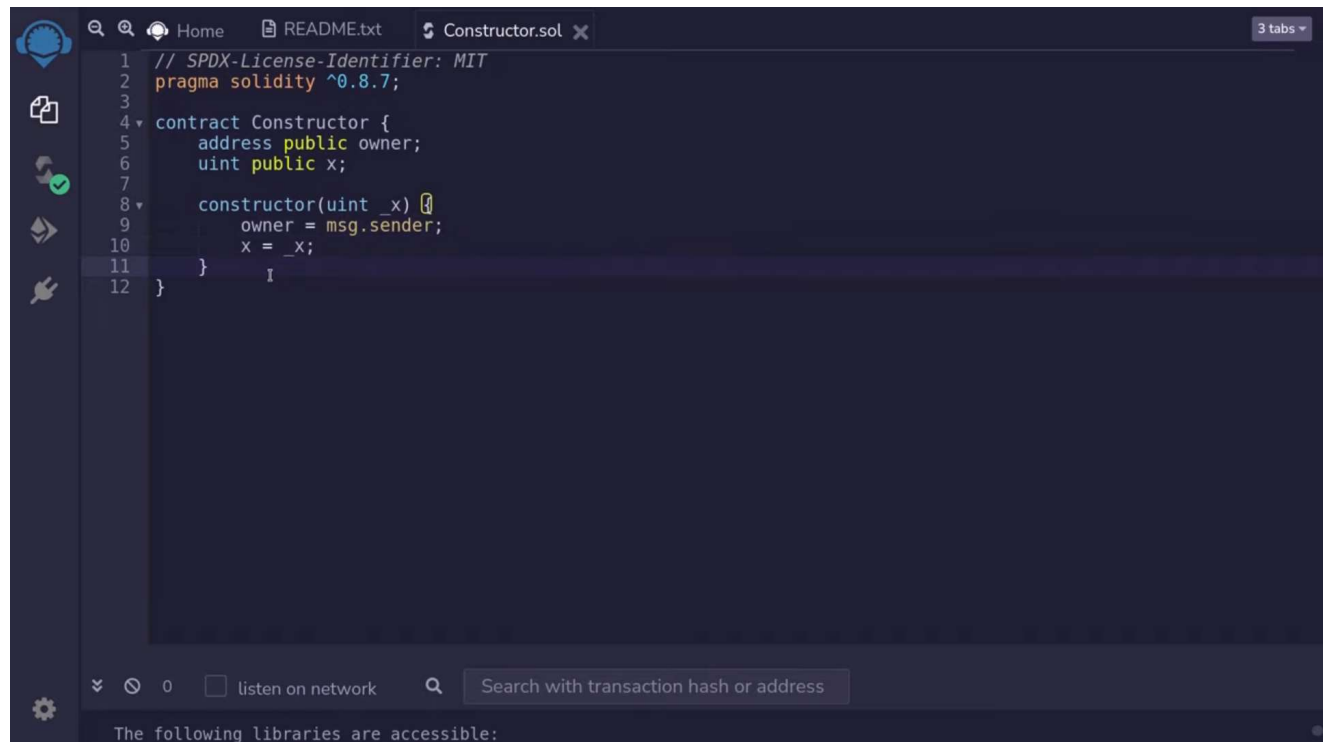


The screenshot shows a Solidity IDE with a file named 'Errors.sol'. The code defines a contract 'Error' with three functions: 'testRequire()', 'testRevert()', and 'testAssert()'. The 'testRequire()' function uses the 'require' statement to check if i is less than or equal to 10. The 'testRevert()' function uses the 'revert' statement to revert the state if i is greater than 10. The 'testAssert()' function uses the 'assert' statement to check if the number 123 is equal to itself.

```
1 pragma solidity ^0.8.7;
2
3 // require, revert, assert
4 // - gas refund, state updates are reverted
5 // custom error - save gas
6 contract Error {
7     function testRequire(uint _i) public pure {
8         require(_i <= 10, "i > 10");
9         // code
10    }
11
12    function testRevert(uint _i) public pure {
13        if (_i > 10) {
14            revert("i > 10");
15        }
16    }
17
18    uint public num = 123;
19
20    function testAssert() public view {
21        assert(num == 123);
22    }
23 }
```

The following libraries are accessible:

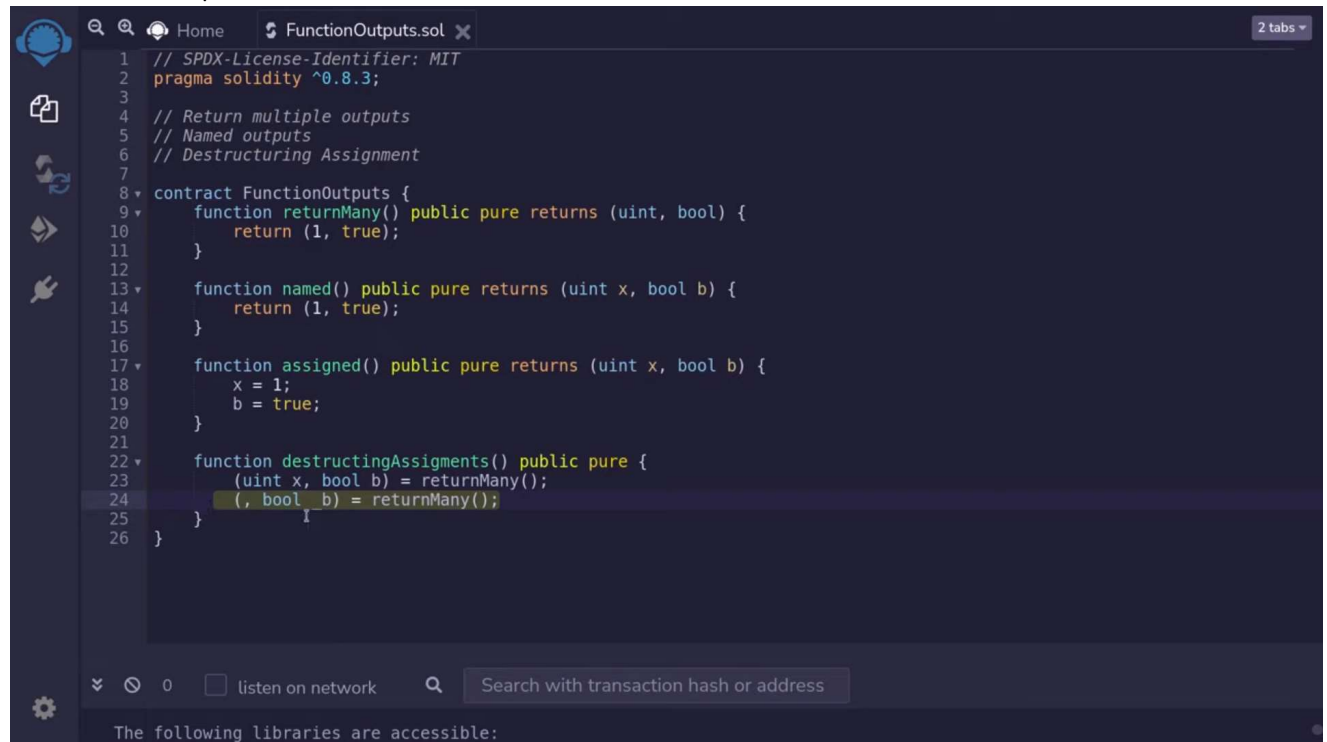
## 7. Constructor:



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.7;
3
4 contract Constructor {
5     address public owner;
6     uint public x;
7
8     constructor(uint _x) {
9         owner = msg.sender;
10        x = _x;
11    }
12 }
```

The following libraries are accessible:

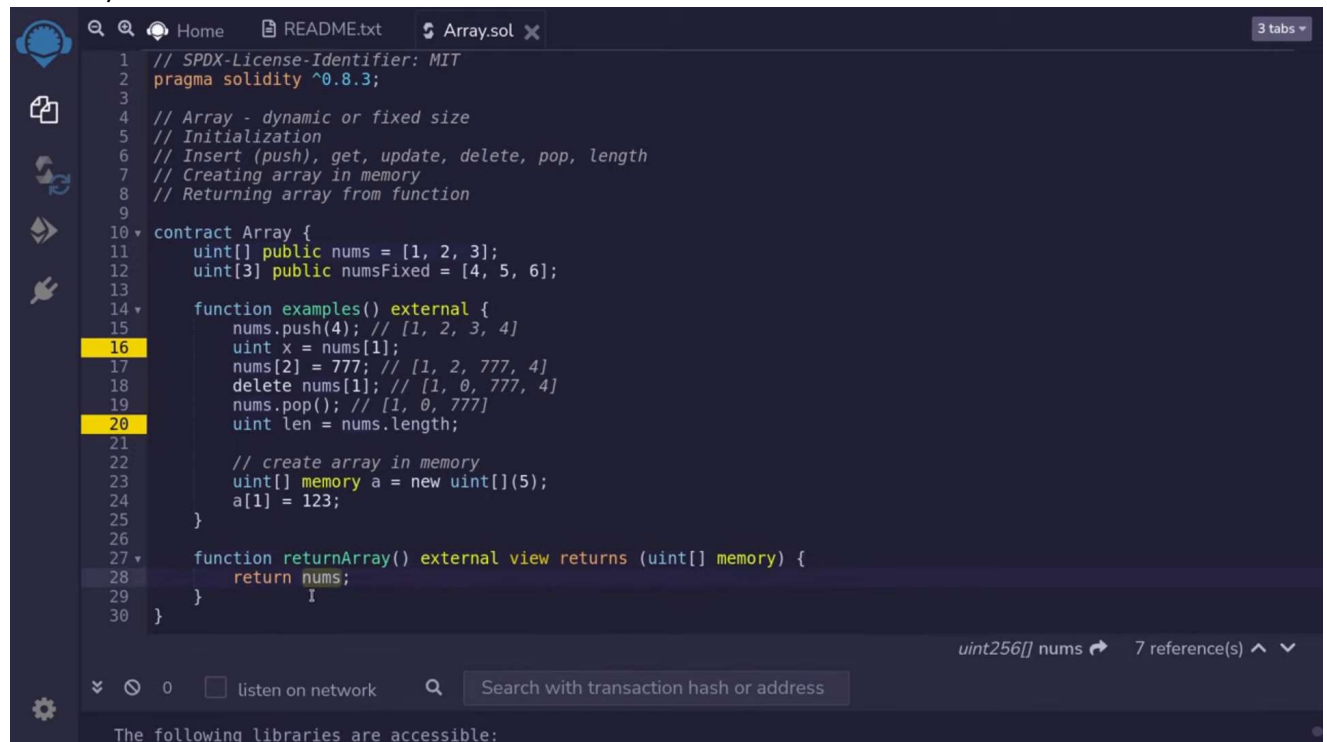
## 8. Function Outputs:



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 // Return multiple outputs
5 // Named outputs
6 // Destructuring Assignment
7
8 contract FunctionOutputs {
9     function returnMany() public pure returns (uint, bool) {
10        return (1, true);
11    }
12
13     function named() public pure returns (uint x, bool b) {
14        return (1, true);
15    }
16
17     function assigned() public pure returns (uint x, bool b) {
18        x = 1;
19        b = true;
20    }
21
22     function destructuringAssignments() public pure {
23        (uint x, bool b) = returnMany();
24        (, bool b) = returnMany();
25    }
26 }
```

The following libraries are accessible:

## 9. Arrays:



The screenshot shows the Solidity IDE with a file named `Array.sol` open. The code defines a contract `Array` with two arrays: `uint[] public nums = [1, 2, 3];` and `uint[3] public numsFixed = [4, 5, 6];`. It includes a function `examples()` that demonstrates array operations like `push`, `pop`, `length`, and `delete`. A `returnArray()` function is also shown, which returns a reference to a memory array. The IDE interface includes a sidebar with icons for Explorer, Search, Run and Debug, and Extensions. The bottom status bar shows '0' and 'listen on network'.

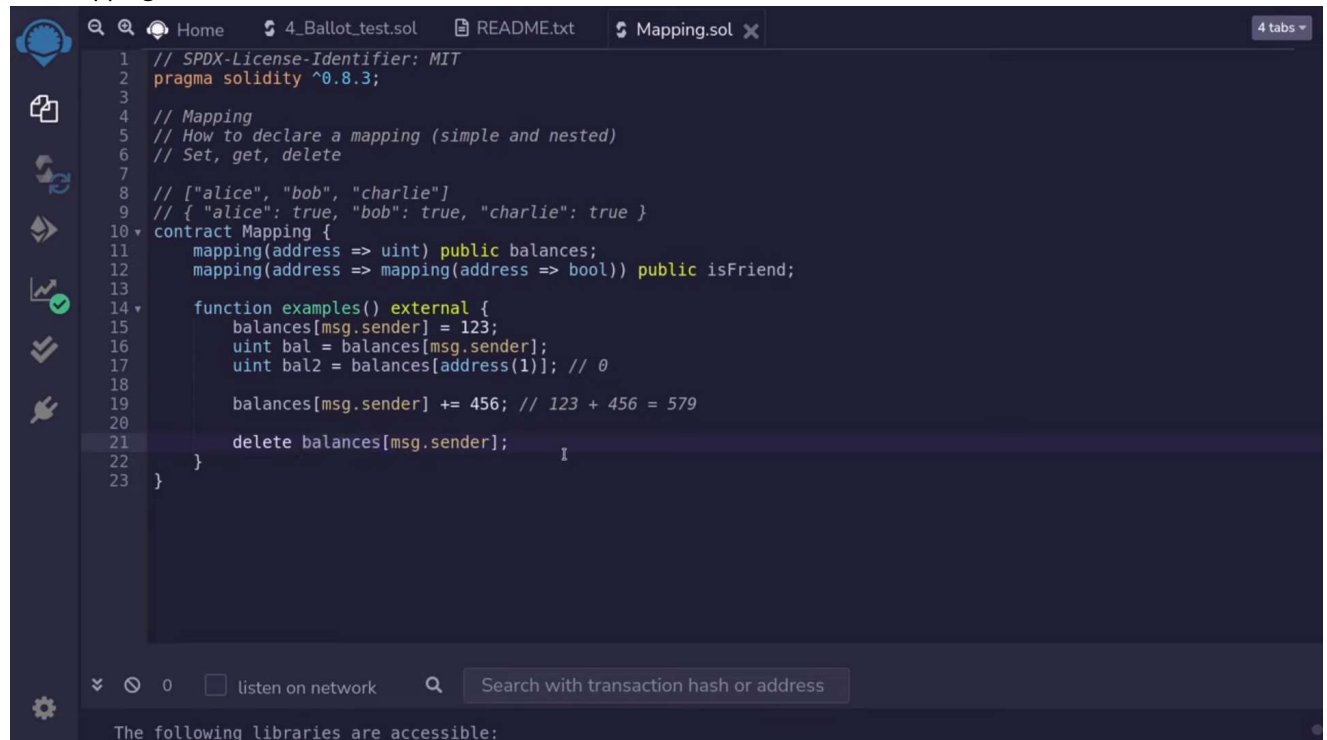
```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 // Array - dynamic or fixed size
5 // Initialization
6 // Insert (push), get, update, delete, pop, length
7 // Creating array in memory
8 // Returning array from function
9
10 contract Array {
11     uint[] public nums = [1, 2, 3];
12     uint[3] public numsFixed = [4, 5, 6];
13
14     function examples() external {
15         nums.push(4); // [1, 2, 3, 4]
16         uint x = nums[1];
17         nums[2] = 777; // [1, 2, 777, 4]
18         delete nums[1]; // [1, 0, 777, 4]
19         nums.pop(); // [1, 0, 777]
20         uint len = nums.length;
21
22         // create array in memory
23         uint[] memory a = new uint[](5);
24         a[1] = 123;
25     }
26
27     function returnArray() external view returns (uint[] memory) {
28         return nums;
29     }
30 }
```

uint256[] nums 7 reference(s)

0 ☐ listen on network Search with transaction hash or address

The following libraries are accessible:

## 10. Mappings:



The screenshot shows the Solidity IDE with a file named `Mapping.sol` open. The code defines a contract `Mapping` with two mappings: `mapping(address => uint) public balances;` and `mapping(address => mapping(address => bool)) public isFriend;`. It includes a function `examples()` that demonstrates mapping operations like `balances[msg.sender] = 123;`, `balances[msg.sender] += 456;`, and `delete balances[msg.sender];`. The IDE interface is similar to the previous screenshot, showing the same sidebar and status bar.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.3;
3
4 // Mapping
5 // How to declare a mapping (simple and nested)
6 // Set, get, delete
7
8 // ["alice", "bob", "charlie"]
9 // { "alice": true, "bob": true, "charlie": true }
10 contract Mapping {
11     mapping(address => uint) public balances;
12     mapping(address => mapping(address => bool)) public isFriend;
13
14     function examples() external {
15         balances[msg.sender] = 123;
16         uint bal = balances[msg.sender];
17         uint bal2 = balances[address(1)]; // 0
18
19         balances[msg.sender] += 456; // 123 + 456 = 579
20
21         delete balances[msg.sender];
22     }
23 }
```

0 ☐ listen on network Search with transaction hash or address

The following libraries are accessible:

## 11. Struct:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.7;

contract test {
    struct Employee {
        uint id;
        string name;
        uint salary;
    }
    Employee employee;

    function addEmployee() public {
        employee = Employee(1,'john', 5000);
    }

    function getEmployeeId() public view returns (uint) {
        return employee.id;
    }
}
```

## 12. Enum:

```
// Solidity program example for Enum Type declaration
pragma solidity ^0.5.0;

// Defining contract
contract EnumTest {
    enum Number { One,Two,Three}
    Number public number=Number.Three;
    constructor() public {
    }

    function getNumber() public view returns (uint) {
        return uint(number);
    }
}
```

### 13. Strings:

String literals are enclosed in double quotes(") or single quotes(')

If it is declared in a global contract, called a global scope variable

```
pragma solidity ^0.5.0;

// Create contract for String type testing
contract StringTest {
    string name;
    string role="admin";
}
```

There is another way to declare string types with byte32. Byte32 works with string declaration as well as string literal syntax.

```
pragma solidity ^0.5.0;

// Create contract for String type testing
contract StringTest {
    byte32 name;
    byte32 role="admin";
}
```

Both `string` and `byte32` do store the string data.

---

### 14. State and Local Variables:

```
pragma solidity ^0.5.0;
contract VariablesContractTest {
    /*
    *State variables are declared inside a contract ie global scope
    */
    uint age; // State variable
    constructor() public {
        age = 10;
    }
    function add() public view returns(uint){
        // local variables are declared and accessed inside a function
        uint localVariable1 = 12; // local variable
        uint localVariable2 = 2; // local variable
        uint sum = localVariable1 + localVariable2; // local variable
        return sum; // access the local variable
    }
}
```