Fixed Arrays

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

contract SumCalculator {
    // External function to calculate the sum of a fixed-size array
    function sum(uint[5] memory numbers) external pure returns (uint) {
        uint total = 0;

        // Loop through the array and calculate the sum
        for (uint i = 0; i < 5; i++) {
            total += numbers[i];
        }

        return total;
    }
}
```

Structs

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

// Define the Choices enum
enum Choices { Yes, No, Maybe }

// Define the Vote struct
struct Vote {
    Choices choice;
```

```
    address voter;
}

contract Voting {
    // Public storage variable to hold the vote
    Vote public vote;

    // Function to create a new vote
    function createVote(Choices _choice) external {
        // Create a new instance of Vote and store it in the vote variable
        vote = Vote({
            choice: _choice,
            voter: msg.sender
        });
    }
}
```

Mapping

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Contract {
    // Public mapping from address to bool to track membership status
    mapping(address => bool) public members;

    // External function to add an address as a member
    function addMember(address _member) external {
```

```
        members[_member] = true;
    }
}
```

Mapping Retrieval

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Contract {
    // Mapping to track membership status
    mapping(address => bool) public members;

    // Adds an address as a member
    function addMember(address _member) external {
        members[_member] = true;
    }

    // Returns true if the address is a member
    function isMember(address _addr) external view returns (bool) {
        return members[_addr];
    }
}
```

Mapping Removal

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Contract {
    // Mapping to track membership status
```

```
    mapping(address => bool) public members;

    // Adds an address as a member
    function addMember(address _member) external {
        members[_member] = true;
    }

    // Checks if an address is a member
    function isMember(address _addr) external view returns (bool) {
        return members[_addr];
    }

    // Removes an address from membership
    function removeMember(address _member) external {
        members[_member] = false;
    }
}
```