

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.2 <0.9.0;
```

```
contract TestPayable {
    uint x;
    uint y;
    // This function is called for all messages sent to
    // this contract, except plain Ether transfers
    // (there is no other function except the receive function).
    // Any call with non-empty calldata to this contract will execute
    // the fallback function (even if Ether is sent along with the call).
    fallback() external payable { x = 1; y = msg.value; }

    // This function is called for plain Ether transfers, i.e.
    // for every call with empty calldata.
    receive() external payable { x = 2; y = msg.value; }
}
```

```
contract StudentRegister {
    mapping (uint => Student) private students;
    address public owner;

    constructor() public payable {
        /* Set the owner to the creator of this contract */
        owner = msg.sender;
    }

    /// Only the `owner` can access - modifier
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    /// Student structure
    struct Student {
        uint studentId;
        string name;
        /* Marks array */
        uint[] marks;
        uint percentage;
        bool exist;
    }

    /// @notice Register a student in the record
    /// @return The percentage of the student
    function register(
        uint studentId,
        string memory name,
        uint[] memory marks
    ) public onlyOwner returns (uint) {
        require(students[studentId].exist == false, "Student data already exist.");
        require(marks.length == 3, "Only 3 subjects are available. Array length should be 3.");

        uint totalMarks = getArraySum(marks);
```

```

    uint percentage = (totalMarks * 100) / 150;

    students[studentId] = Student(
        studentId,
        name,
        marks,
        percentage,
        true
    );

    return percentage;
}

/// @notice Get student details from the record
/// @return Student id, name, marks, percentage of the student
function getStudentDetails(
    uint studentId
) public view returns (uint, string memory, uint[] memory, uint) {
    require(students[studentId].exist == true, "No student data
available.");

    /* Access student from the registered using studentId */
    Student memory student = students[studentId];

    return(
        student.studentId,
        student.name,
        student.marks,
        student.percentage
    );
}

/// @notice Get sum of the array
/// @return sum of the array
function getArraySum(uint[] memory array) private pure returns (uint
sum) {
    sum = 0;
    for (uint i = 0; i < array.length; i++) {
        require(0 <= array[i] && array[i] <= 100, "Marks should be
between 0 and 100.");
        sum += array[i];
    }
}

function callTestPayable(TestPayable test) public returns (bool) {
    (bool success,) =
address(test).call(abi.encodeWithSignature("nonExistingFunction()"));
    require(success);
    // results in test.x becoming == 1 and test.y becoming 0.
    (success,) = address(test).call{value:
1}(abi.encodeWithSignature("nonExistingFunction()"));
    require(success);
    // results in test.x becoming == 1 and test.y becoming 1.

    // If someone sends Ether to that contract, the receive function
in TestPayable will be called.

```

```
        // Since that function writes to storage, it takes more gas than
is available with a
        // simple ``send`` or ``transfer``. Because of that, we have to
use a low-level call.
        (success,) = address(test).call{value: 2 ether}("");
        require(success);
        // results in test.x becoming == 2 and test.y becoming 2 ether.

        return true;
    }
}
```