



Co-funded by the
Erasmus+ Programme
of the European Union



**SUSTAINABLE
SHIP
AND SHIPPING
4.0**

Erasmus Mundus Joint Master
Degree

Excellence in education for a new generation of Naval Architects and Marine Engineers

INDUSTRY 4.0 ENABLING TECHNOLOGIES

Smart Contract for Incentivizing Valid Usage of Energy

Team

Abisade Folarin
Ahsan Ullah Jawad
Muhammad Fuad Shofly
Pritam Kumar Patel
Saeid Panahi

Course Teacher

Professor Dr. Tiago M. Fernández
Professor Dr. Paula Fraga Lamas



1. Introduction

As a continuation of the IoT assignment on Intelligent lighting system of the Industry 4.0 Enabling Technologies course, a smart contract has been written based on Ethereum to send incentives for valid usage of energy. In this contract, all sensor input can be stored with a unique id, various valid conditions can be checked, and Wei can be sent inward and outward the contract if a certain condition is met.

2. Case

Numerous light sources are the substantial part of the energy consumption of a ship. It has been observed that reduction of the usage of these lights can significantly reduce energy consumption, carbon emissions, and light pollution, as well as increase the longevity of the devices leading to achieve a sustainable society. Apart from the personal or organizational urge, monetary incentive has always been proven as an effective tool to achieve a target. Hence, a smart contract has been written to send incentives only for valid usage of light sources.

3. Users

Resembling a regular contract, this smart contract has two parties. One is the ship crew, and the other is the ship owner.

4. Technical Parameter

This smart contract has been written in the Solidity programming language that runs on Ethereum Virtual Machine. To compile the code Remix IDE has been used and the code has been run on JavaScript VM(London) environment.

5. Contents

The contract is focused on detecting the necessary usage of the light source. Only in case of a valid reason for light source usage, the ship owner can send incentives for example WEI to the ship crew.

In the ship, Push button, RTC, LDR, and PIR sensors will gather the information about the surrounding. The algorithm of the code will confirm the validity. Valid reasons include emergency, nighttime, darkness, and the presence of a human. All these sensors generate the value along with the status of the light source. These values are the input of the code. A struct “Incident” will gather all the sensors’ value. Each time these inputs are logged, “generateincidentid” will create a unique id using the hash function. The function “new_incident” will create an incident and add each of the incidents into the array of the “Incident” struct. An Event “incident_register” will emit each time a incident is logged along with the index number. The shipowner can see the input log history using the index number.

A function “check_validity” contains the if-else condition that checks the validity of the led usage. There are a few cases when all sensors’ data might not be necessary. In such a case, the value of the sensors should be “-1”. The function “Validity” will show the validity of the input.

A smart contract is more impactful when it can transact money i.e., Ether in this case. Therefore, the ship owner can send Wei from his account to this contract using the “receivewe” function. Each time of transection the amount of the Wei should be a minimum of 500. If the sender sends less than this value transaction will be failed displaying an error message. Each time the owner sends money to the contract, it will be added to the current balance.

To check the current balance of the contract, the “checkbalance” function can be used.

Finally, the ship owner can send the Wei to the ship crew using “sendwei” function. This function only executes the transaction if there is a valid reason for energy usage. An error will be displayed with a message if the reason is invalid. Moreover, a minimum balance should be more than 100 Wei, otherwise, the transaction will not be executed along with another error message.

6. Testing

The code has been tested on different scenarios as mentioned below.

a) Scenario 1: Push Button: On, Disregard other sensors, Led: On, Reason: Valid

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

contract intelligent_light {
    uint public validity = 0;
    uint public totalReceivedWei = 0;
    address public shipowner;

    struct Incident {
        uint pb_value;
        uint rtc_value;
        uint ldr_value;
        uint pir_value;
        uint led;
        uint time;
    }
    Incident[] incidents;

    function check_validity() public view returns(uint) {
        if (_pb_value == 1 && _rtc_value == -1 && _ldr_value == -1 && _pir_value == -1 && _led == 1 && _time == 1000) {
            return validity;
        } else {
            return 0;
        }
    }

    //--function for receiving wei from ship owner to this contract--
    mapping (address=> uint) totalReceivedWei;
    function receivewe() external payable {
        require(msg.value>500,"Please send atleast 500 wei");
        shipowner=msg.sender;
        totalReceivedWei[msg.sender]+=msg.value;
    }

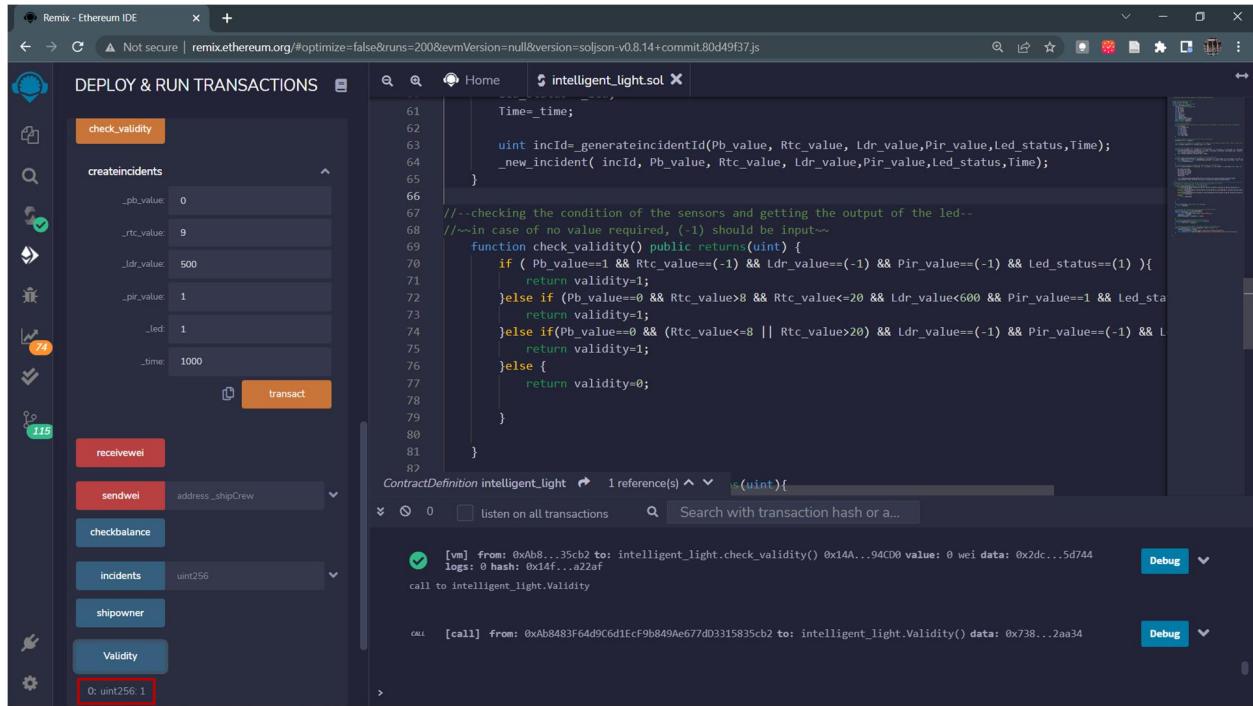
    //~~function for checking balance of the contract~~
    function checkbalance() external view returns(uint){
        return address(this).balance;
    }

    function createincidents(Incident memory incident) public {
        incidents.push(incident);
    }

    function shipowner() public view returns(address) {
        return shipowner;
    }

    function Validity() public view returns(uint) {
        return validity;
    }
}
```

b) Scenario 2: Push Button: Off, Time: Day, Light: Dark, Pir: On, Led: On, Reason: Valid



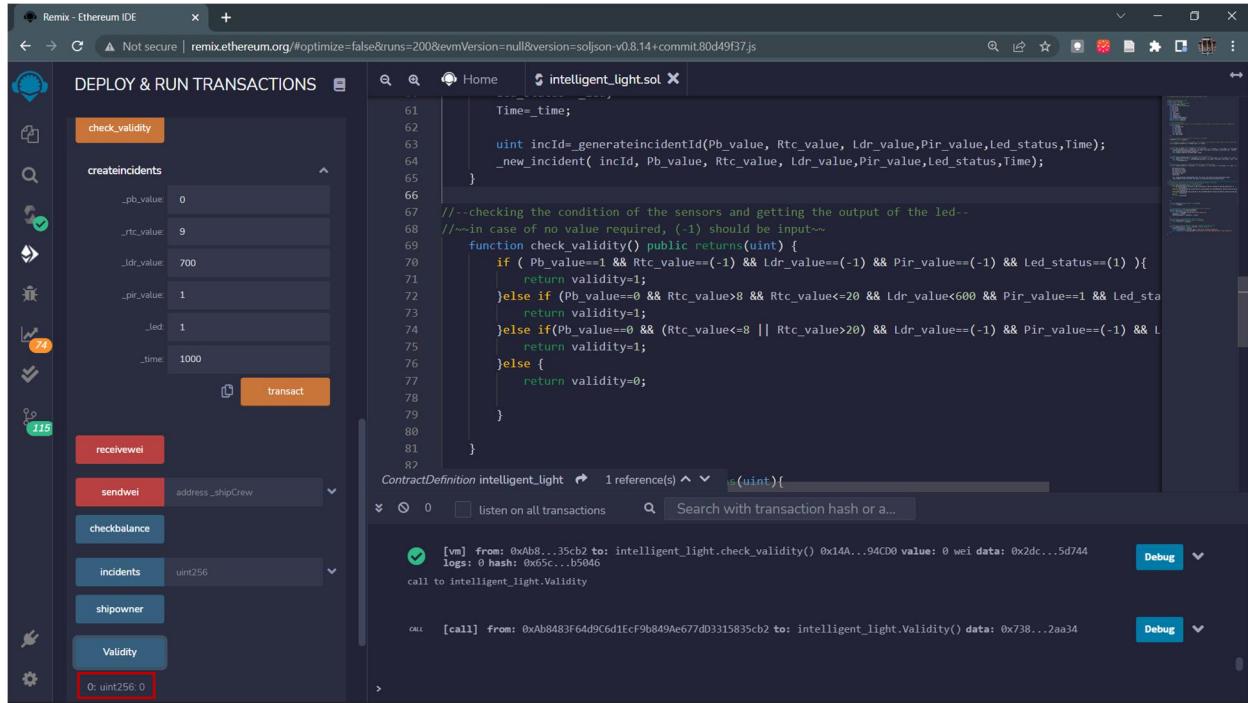
The screenshot shows the Remix Ethereum IDE interface. The left sidebar has tabs for Deploy & Run Transactions, Home, and intelligent_light.sol. The Deploy & Run Transactions tab is active, showing a transaction history with one entry: [call] from: 0xAb8...35cb2 to: intelligent_light.check_validity() 0x14A...94C00 value: 0 wei data: 0x2dc...5d744. Below this, there are buttons for receivewei, sendwei, checkbalance, incidents, shipowner, and Validity. The incidents button is highlighted. The Validity button has a red border around its value field. The right side of the interface shows the Solidity code for the intelligent_light contract, specifically the check_validity function.

```

61     Time=_time;
62
63     uint incId=_generateincidentId(Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);
64     _new_incident( incId, Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);
65   }
66
67   //--checking the condition of the sensors and getting the output of the led--
68   //~~in case of no value required, (-1) should be input~~
69   function check_validity() public returns(uint) {
70     if ( Pb_value==1 && Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1) ){
71       return validity=1;
72     }else if (Pb_value==0 && Rtc_value>8 && Rtc_value<=20 && Ldr_value<600 && Pir_value==1 && Led_status==1){
73       return validity=1;
74     }else if(Pb_value==0 && (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==1){
75       return validity=1;
76     }else {
77       return validity=0;
78     }
79   }
80
81 }
82

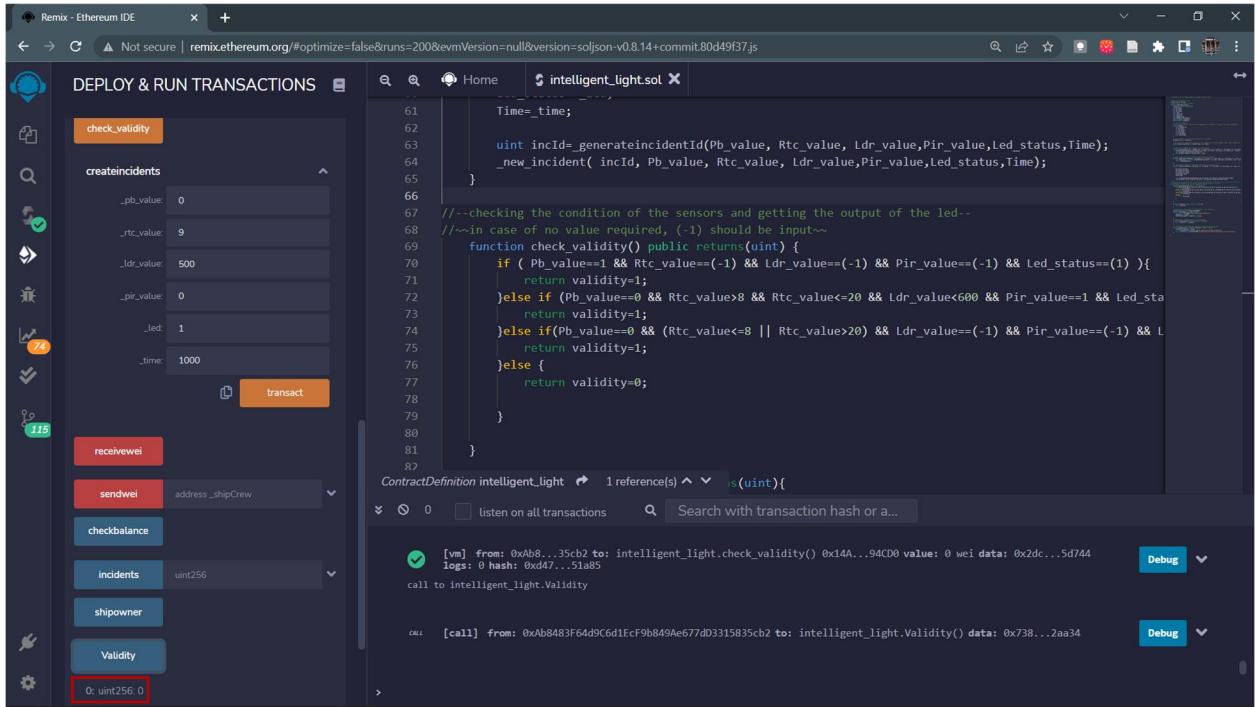
```

c) Scenario 3: Push Button: Off, Time: Day, Light: Bright, Pir: On, Led: On, Reason: Invalid



The screenshot shows the Remix Ethereum IDE interface, identical to the previous one but with different sensor values. The incidents button is highlighted. The Validity button has a red border around its value field. The right side shows the same Solidity code as before, but the transaction logs indicate an error: [call] from: 0xAb8...35cb2 to: intelligent_light.check_validity() 0x14A...94C00 value: 0 wei data: 0x2dc...5d744, logs: 0 hash: 0x65c...15046. This indicates that the transaction failed due to an invalid input (bright light).

d) Scenario 4: Push Button: Off, Time: Day, Light: Dark, Pir: off, Led: On, Reason: Invalid



The screenshot shows the Remix Ethereum IDE interface. The left sidebar displays a list of transactions: `check_validity`, `createincidents`, `receiveweis`, `sendwei`, `checkbalance`, `incidents`, `shipowner`, and `Validity`. The `Validity` transaction has a red border around its value field, which contains `0: uint256: 0`. The right panel shows the Solidity code for the `intelligent_light.sol` contract, specifically the `check_validity()` function. The function logic checks for specific sensor values to determine validity. In this scenario, the conditions for a valid return (light dark, pir off, led on) are not met, resulting in a return value of 0.

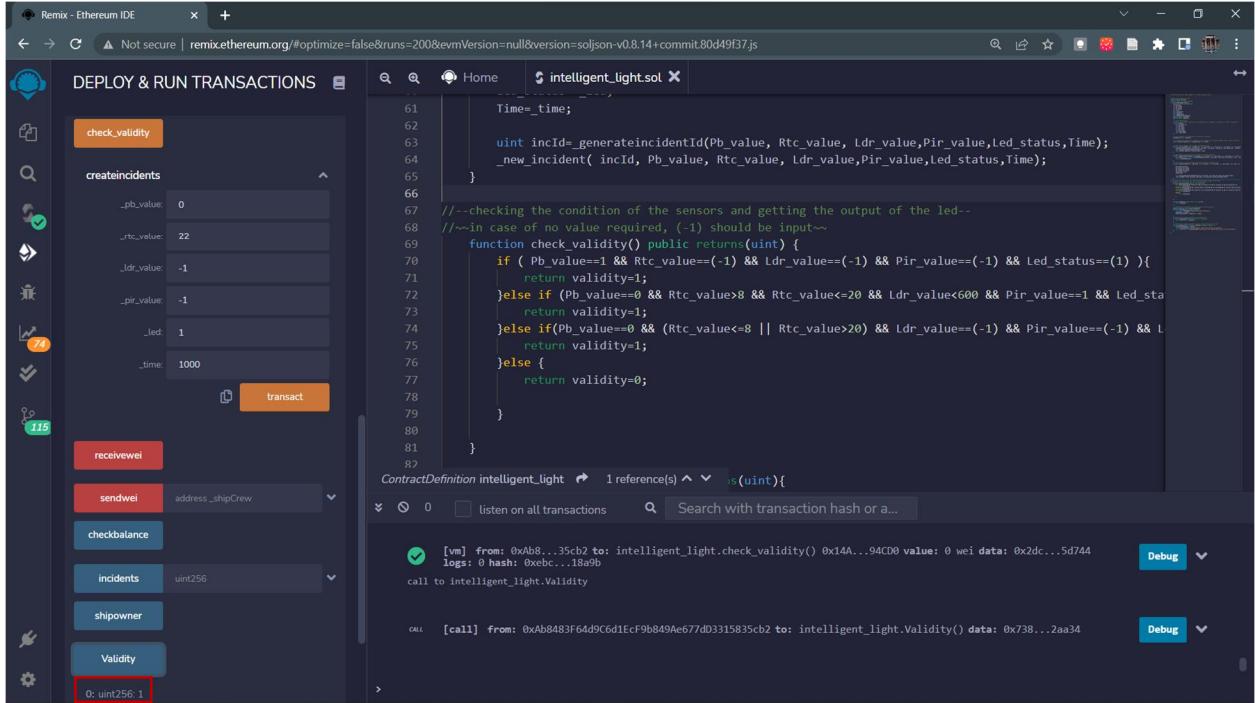
```

Time=_time;
uint incId=_generateincidentId(Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);
_new_incident( incId, Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);

//--checking the condition of the sensors and getting the output of the led--
//~~in case of no value required, (-1) should be input~~
function check_validity() public returns(uint) {
    if ( Pb_value==1 & Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1) ){
        return validity=1;
    }else if (Pb_value==0 & Rtc_value>8 && Rtc_value<=20 && Ldr_value<600 && Pir_value==1 && Led_status==(-1) ){
        return validity=1;
    }else if(Pb_value==0 & (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(-1) ){
        return validity=1;
    }else {
        return validity=0;
    }
}

```

e) Scenario 5: Push Button: Off, Time: Night, Disregard other sensors, Led: On, Reason: Valid



The screenshot shows the Remix Ethereum IDE interface. The left sidebar displays a list of transactions: `check_validity`, `createincidents`, `receiveweis`, `sendwei`, `checkbalance`, `incidents`, `shipowner`, and `Validity`. The `Validity` transaction has a red border around its value field, which contains `0: uint256: 1`. The right panel shows the Solidity code for the `intelligent_light.sol` contract, specifically the `check_validity()` function. In this scenario, the light being dark and the pir sensor being off (values -1) satisfy the condition for a valid return (light dark, pir off, led on), resulting in a return value of 1.

```

Time=_time;
uint incId=_generateincidentId(Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);
_new_incident( incId, Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);

//--checking the condition of the sensors and getting the output of the led--
//~~in case of no value required, (-1) should be input~~
function check_validity() public returns(uint) {
    if ( Pb_value==1 & Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1) ){
        return validity=1;
    }else if (Pb_value==0 & Rtc_value>8 && Rtc_value<=20 && Ldr_value<600 && Pir_value==1 && Led_status==(-1) ){
        return validity=1;
    }else if(Pb_value==0 & (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(-1) ){
        return validity=1;
    }else {
        return validity=0;
    }
}

```

f) Scenario 6: Condition: Not meet, Receiving Wei: Failed

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'Deploy & Run Transactions' sidebar lists several functions: check_validity, createincidents, receiveweи, sendwei, checkbalance, incidents, shipowner, and Validity. The 'sendwei' function is selected, showing its parameters: address _shipCrew. The main pane displays the Solidity code for the intelligent_light contract. A red box highlights the value '0: uint256 1' at the bottom of the code editor. In the bottom right corner of the code editor, there is a red error message: '[vm] from: 0xAb8...35cb2 to: intelligent_light.receivewei() 0x14A...94CD0 value: 500 wei data: 0x537...b560d logs: 0 hash: 0xd5c...247f9'. Below the code editor, a status bar shows 'revert' and 'The transaction has been reverted to the initial state. Reason provided by the contract: "Please send atleast 500 wei". Debug the transaction to get more information.'

```

Rtc_value=_rtc_value;
Ldr_value=_ldr_value;
Pir_value=_pir_value;
Led_status= _led;
Time=_time;

uint incId=_generateIncidentId(Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);
_new_incident( incId, Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);

//--checking the condition of the sensors and getting the output of the led--
//~~in case of no value required, (-1) should be input~~
function check_validity() public returns(uint) {
    if ( Pb_value==1 && Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1) ){
        return validity=1;
    }else if ( Pb_value==0 && Rtc_value>8 && Rtc_value<=20 && Ldr_value<600 && Pir_value==1 && Led_status==(-1) ){
        return validity=1;
    }else if(Pb_value==0 && (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(-1) ){
        return validity=1;
    }else {
        return validity=0;
    }
}

```

g) Scenario 7: Condition: meet, Receiving Wei: Executed

This screenshot is similar to the previous one but shows a successful transaction. The 'sendwei' function is selected, and the value '0: uint256 1' is highlighted. The status bar now shows a green checkmark and the message 'transact to intelligent_light.receivewei pending ...'. Another transaction entry is visible below it: '[vm] from: 0xAb8...35cb2 to: intelligent_light.receivewei() 0x14A...94CD0 value: 1000 wei data: 0x537...b560d logs: 0 hash: 0xe87...6fcfd'. The bottom right corner of the code editor shows a green success message: '[vm] from: 0xAb8...35cb2 to: intelligent_light.receivewei() 0x14A...94CD0 value: 1000 wei data: 0x537...b560d logs: 0 hash: 0xe87...6fcfd'.

```

Rtc_value=_rtc_value;
Ldr_value=_ldr_value;
Pir_value=_pir_value;
Led_status= _led;
Time=_time;

uint incId=_generateIncidentId(Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);
_new_incident( incId, Pb_value, Rtc_value, Ldr_value,Pir_value,Led_status,Time);

//--checking the condition of the sensors and getting the output of the led--
//~~in case of no value required, (-1) should be input~~
function check_validity() public returns(uint) {
    if ( Pb_value==1 && Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1) ){
        return validity=1;
    }else if ( Pb_value==0 && Rtc_value>8 && Rtc_value<=20 && Ldr_value<600 && Pir_value==1 && Led_status==(-1) ){
        return validity=1;
    }else if(Pb_value==0 && (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(-1) ){
        return validity=1;
    }else {
        return validity=0;
    }
}

```

h) Scenario 8: Balance Check

The screenshot shows the Remix Ethereum IDE interface. The left sidebar displays the "DEPLOY & RUN TRANSACTIONS" section with a list of deployed contracts under "INTELLIGENT_LIGHT AT 0x14A...94CD0". One of the buttons, "checkbalance", is highlighted with a red box. The transaction details show a value of "0: uint256: 1000". The right side of the interface shows the Solidity code for the contract, specifically the `checkbalance()` function, which returns a validity value based on sensor inputs.

```
Rtc_value=_rtc_value;
Ldr_value=_ldr_value;
Pir_value=_pir_value;
Led_status=_led;
Time=_time;

uint incId=_generateIncidentId(Pb_value, Rtc_value, Ldr_value, Pir_value, Led_status, Time);
_new_incident(incId, Pb_value, Rtc_value, Ldr_value, Pir_value, Led_status, Time);

//--checking the condition of the sensors and getting the output of the led--
//~~in case of no value required, (-1) should be input~~
function check_validity() public returns(uint) {
    if (Pb_value==1 && Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1)) {
        return validity=1;
    } else if (Pb_value==0 && Rtc_value>8 && Rtc_value<=20 && Ldr_value<600 && Pir_value==1 && Led_status==(-1)) {
        return validity=1;
    } else if (Pb_value==0 && (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(-1)) {
        return validity=1;
    } else {
        return validity=0;
    }
}
```

i) Scenario 9: Wei Sender Address Check

The screenshot shows the Remix Ethereum IDE interface, similar to scenario 8. The "checkbalance" button is highlighted with a red box. The transaction details show a value of "0: address: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2". The right side shows the Solidity code for the `shipowner()` function, which returns the address of the sender.

```
Rtc_value=_rtc_value;
Ldr_value=_ldr_value;
Pir_value=_pir_value;
Led_status=_led;
Time=_time;

uint incId=_generateIncidentId(Pb_value, Rtc_value, Ldr_value, Pir_value, Led_status, Time);
_new_incident(incId, Pb_value, Rtc_value, Ldr_value, Pir_value, Led_status, Time);

//--checking the condition of the sensors and getting the output of the led--
//~~in case of no value required, (-1) should be input~~
function check_validity() public returns(uint) {
    if (Pb_value==1 && Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1)) {
        return validity=1;
    } else if (Pb_value==0 && Rtc_value>8 && Rtc_value<=20 && Ldr_value<600 && Pir_value==1 && Led_status==(-1)) {
        return validity=1;
    } else if (Pb_value==0 && (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(-1)) {
        return validity=1;
    } else {
        return validity=0;
    }
}
```

j) Scenario 10: Reason: Invalid, Sending Wei: Failed

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar has a dropdown set to 'createincidents'. The transaction inputs are: _pb_value: "0", _rtc_value: "9", _ldr_value: "500", _pir_value: "0", _led: "'1'", and _time: "'1000'". Below the inputs are three buttons: 'receivewei' (disabled), 'sendwei' (highlighted in red), and 'checkbalance'. The 'sendwei' button has a tooltip '0xAb8483F64d9C6d1EcF9b849' and a dropdown menu showing the same address. The transaction status is '0: uint256: 1000'. The right panel displays the Solidity code for the 'intelligent_light.sol' contract. A revert message is shown: 'The transaction has been reverted to the initial state. Reason provided by the contract: "Invalid reason. They do not deserve incentive". Debug the transaction to get more information.' A log entry is present: '[vm] from: 0xAb8...35cb2 to: intelligent_light.sendwei(address) 0x14A...94CD0 value: 200 wei data: 0xbe0...35cb2 logs: 0 hash: 0xd21...3f0f8'. A 'Debug' button is visible at the bottom right.

```

Rtc_value=_rtc_value;
Ldr_value=_ldr_value;
Pir_value=_pir_value;
Led_status=_led;
Time=_time;

uint incId=_generateIncidentId(Pb_value, Rtc_value, Ldr_value, Pir_value, Led_status, Time);
_new_incident( incId, Pb_value, Rtc_value, Ldr_value, Pir_value, Led_status, Time);

//--checking the condition of the sensors and getting the output of the led--
//~~in case of no value required, (-1) should be input~~
function check_validity() public returns(uint) {
    if ( Pb_value==1 & Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1) ) {
        return validity=1;
    } else if ( Pb_value==0 && Rtc_value>8 && Rtc_value<=20 && Ldr_value<=600 && Pir_value==1 && Led_status==(-1) ) {
        return validity=1;
    } else if(Pb_value==0 & (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(-1) ) {
        return validity=1;
    } else {
        return validity=0;
    }
}

```

k) Scenario 10: Reason: Valid, Sending Wei: Executed

The screenshot shows the Remix Ethereum IDE interface. The setup is identical to the previous scenario, with the 'createincidents' option selected in the sidebar. The transaction inputs remain the same. The 'sendwei' button is now highlighted in green, indicating success. The transaction status is '0: uint256: 1000'. The right panel shows the same Solidity code as before. A log entry is present: '[call] from: 0xAb8483F64d9C6d1EcF9b849 to: intelligent_light.Validity() data: 0x738...2aa34'. Another log entry is shown: '[vm] from: 0xAb8...35cb2 to: intelligent_light.sendwei(address) 0x14A...94CD0 value: 200 wei data: 0xbe0...35cb2 logs: 0 hash: 0x738...2aa34'. A 'Debug' button is visible at the bottom right.

```

Rtc_value=_rtc_value;
Ldr_value=_ldr_value;
Pir_value=_pir_value;
Led_status=_led;
Time=_time;

uint incId=_generateIncidentId(Pb_value, Rtc_value, Ldr_value, Pir_value, Led_status, Time);
_new_incident( incId, Pb_value, Rtc_value, Ldr_value, Pir_value, Led_status, Time);

//--checking the condition of the sensors and getting the output of the led--
//~~in case of no value required, (-1) should be input~~
function check_validity() public returns(uint) {
    if ( Pb_value==1 & Rtc_value==(-1) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(1) ) {
        return validity=1;
    } else if ( Pb_value==0 && Rtc_value>8 && Rtc_value<=20 && Ldr_value<=600 && Pir_value==1 && Led_status==(-1) ) {
        return validity=1;
    } else if(Pb_value==0 & (Rtc_value<=8 || Rtc_value>20) && Ldr_value==(-1) && Pir_value==(-1) && Led_status==(-1) ) {
        return validity=1;
    } else {
        return validity=0;
    }
}

```

I) Scenario 11: Balance: Insufficient, Sending Wei: Failed

The screenshot shows the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar has several buttons: 'check_validity', 'createincidents', 'receiveweи', 'sendwei', 'checkbalance', 'incidents', 'shipowner', and 'Validity'. The 'sendwei' button is highlighted in red. The main pane displays the Solidity code for the 'intelligent_light.sol' contract. A transaction log at the bottom indicates a failure:

```
[vm] from: 0xAb8...35cb2 to: intelligent_light.sendwei(address) 0xad1...803fb value: 200 wei data: 0xbe0...35cb2 logs: 0 hash: 0x4bc...f2abc
```

The transaction revert message states: "The transaction has been reverted to the initial state. Reason provided by the contract: "Insufficient wei, Please add more wei to the contract". Debug the transaction to get more information."

m) Scenario 12, 13: Checking Log History

The screenshot shows the Remix Ethereum IDE interface. The 'incidents' button in the sidebar is highlighted in red. The main pane displays the same Solidity code as before. A transaction log at the bottom shows a successful deployment of a new incident:

```
[vm] from: 0xAb8...35cb2 to: intelligent_light.createincidents(int256,int256,int256,int256,uint256,uint256) 0xad1...803fb value: 0 wei data: 0x01b...004db logs: 1 hash: 0x71a...6dfa
```

The log details the creation of a new incident with the following parameters:

- 0: uint256: incident_id 23
- 1: int256: pb_value 0
- 2: int256: rtc_value 22
- 3: int256: ldr_value 700
- 4: int256: pir_value 1
- 5: uint256: led_status 1
- 6: uint256: time_stamp 1200

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;

contract intelligent_light {
    uint256 _pb_value;
    uint256 _rtc_value;
    uint256 _ldr_value;
    uint256 _pir_value;
    uint8 _led;
    uint256 _time;

    mapping (address=> uint) totalreceivedwei;
    function receiveweis() external payable {
        require(msg.value>500,"Please send atleast 500 wei");
        shipowner=msg.sender;
        totalreceivedwei[msg.sender]+=msg.value;
    }
    //~function for checking balance of the contract~
    function checkbalance() external view returns(uint){
        return address(this).balance;
    }
    //~function for sending 100 wei from this contract to the ship crew--
    function sendwei(address payable _shipCrew) payable external {
        _shipCrew.transfer(msg.value);
        require((validity==1),"Invalid reason.. They do not deserve incentive");
    }
}

library incidents {
    struct incident {
        uint256 incident_id;
        int256 pb_value;
        int256 rtc_value;
        int256 ldr_value;
        int256 pir_value;
        uint8 led_status;
        uint256 time_stamp;
    }
}

```

The screenshot shows the Remix Ethereum IDE interface. On the left, there's a sidebar with various icons for managing contracts, variables, and transactions. The main area has tabs for "Home" and the current file "intelligent_light.sol". The code editor displays the Solidity code for the "intelligent_light" contract, which includes functions for receiving and sending Wei, and a library for managing incidents. The transaction history panel on the right shows two recent calls to the "sendwei" function. A sidebar on the left also lists the "incidents" library and its fields.

7. Conclusion:

Blockchain is a contemporary and revolutionary technology. The scope of the smart contract will eliminate the necessity of intermediary, increase trust, and security and streamline various processes. Solidity is a popular computer language to write the smart contract. Our solidity code is working as expected. However, it can be further improved to perform more complex tasks. This assignment has instigated our interest and is a stepping stone in the world of blockchain. While writing this code we received support from Professor Tiago M. Fernández Caramés and Dr. Paula Fraga Lamas. Various resources from the internet also helped us to modify and debug the code.