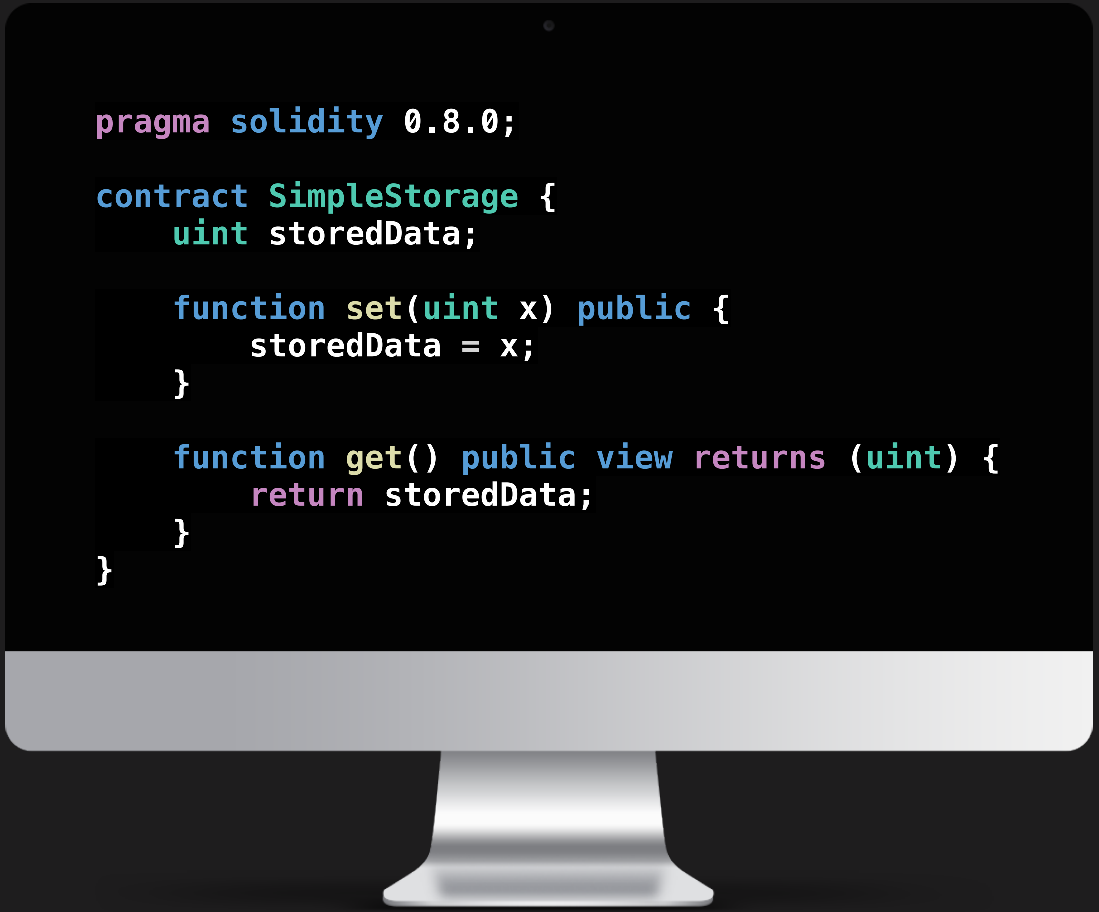
Smart Contract

- BChen, Bill Hsu @ Binance Research -

目錄



```
pragma solidity 0.8.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Gas的設計

- 每種運算都有其相對應的成本
- Gas Price
  - 每個單位 Gas 的價格
  - 1 Gwei = 0.000000001 ETH
- Gas Limit
  - 單筆交易中所願意支付 Gas 單位的最大數量
- Tx Fee
  - 最多為 Gas Limit * Gas Price

1. 礦工的選擇
2. 超鉅額手續費

合約的部署

1. 寫好合約
2. 編譯合約
    - Bytecode
    - ABI
3. 透過線上 IDE 部署 or 其他

呼叫合約

1. Function 的識別碼
2. 放上需要的參數

Function : setName(string)

$\downarrow$

Kecaak-256

$\downarrow$

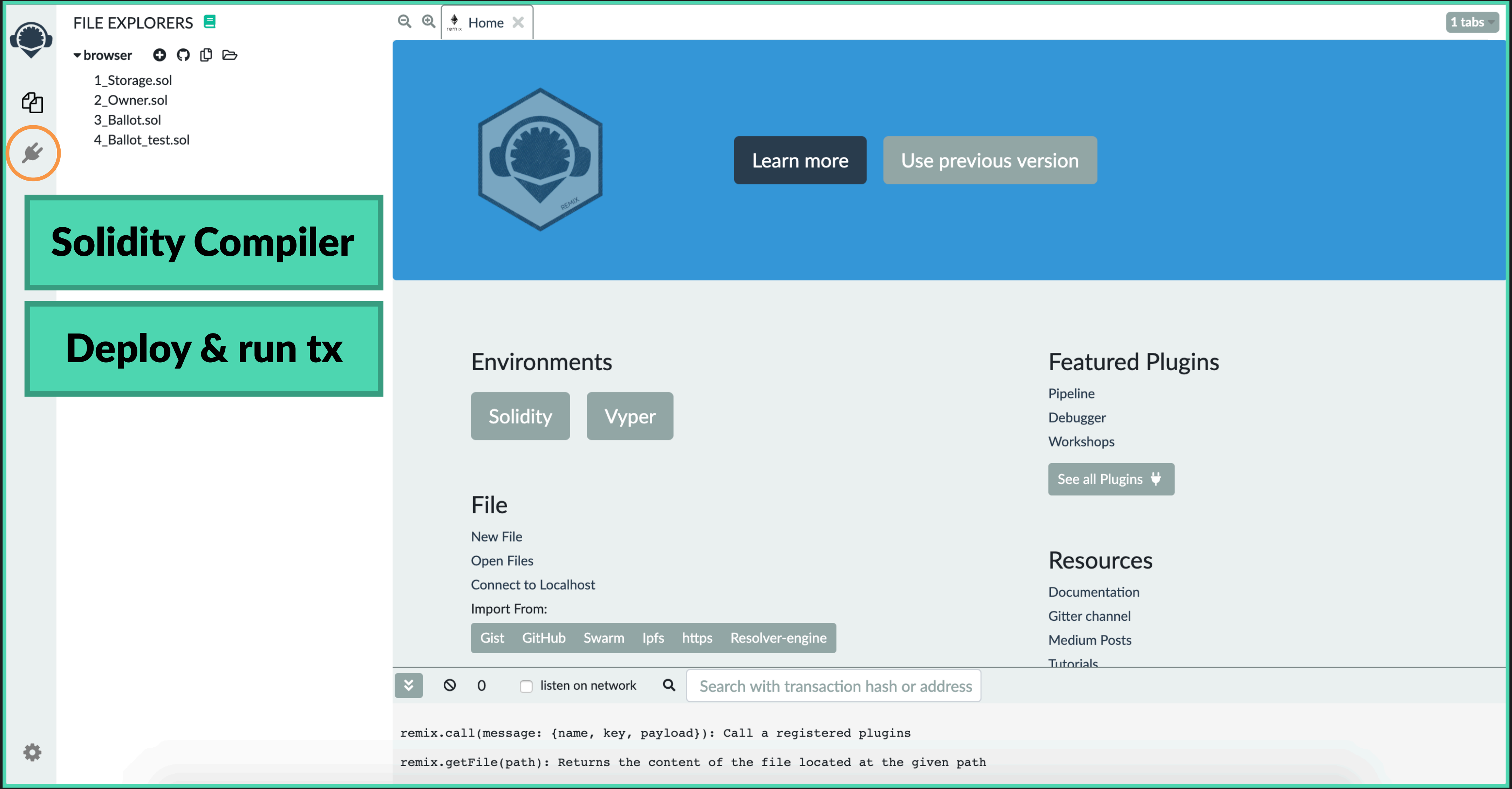
c47f0027........

- 開發環境介紹 -

編譯合約

- Remix - 線上 IDE
  - http://remix.ethereum.org
- 安裝本機版 Remix IDE
  - `npm install remix-ide -g`
- Solc (Solidity Compiler) - localhost
  - https://www.npmjs.com/package/solc

Remix - 線上 IDE

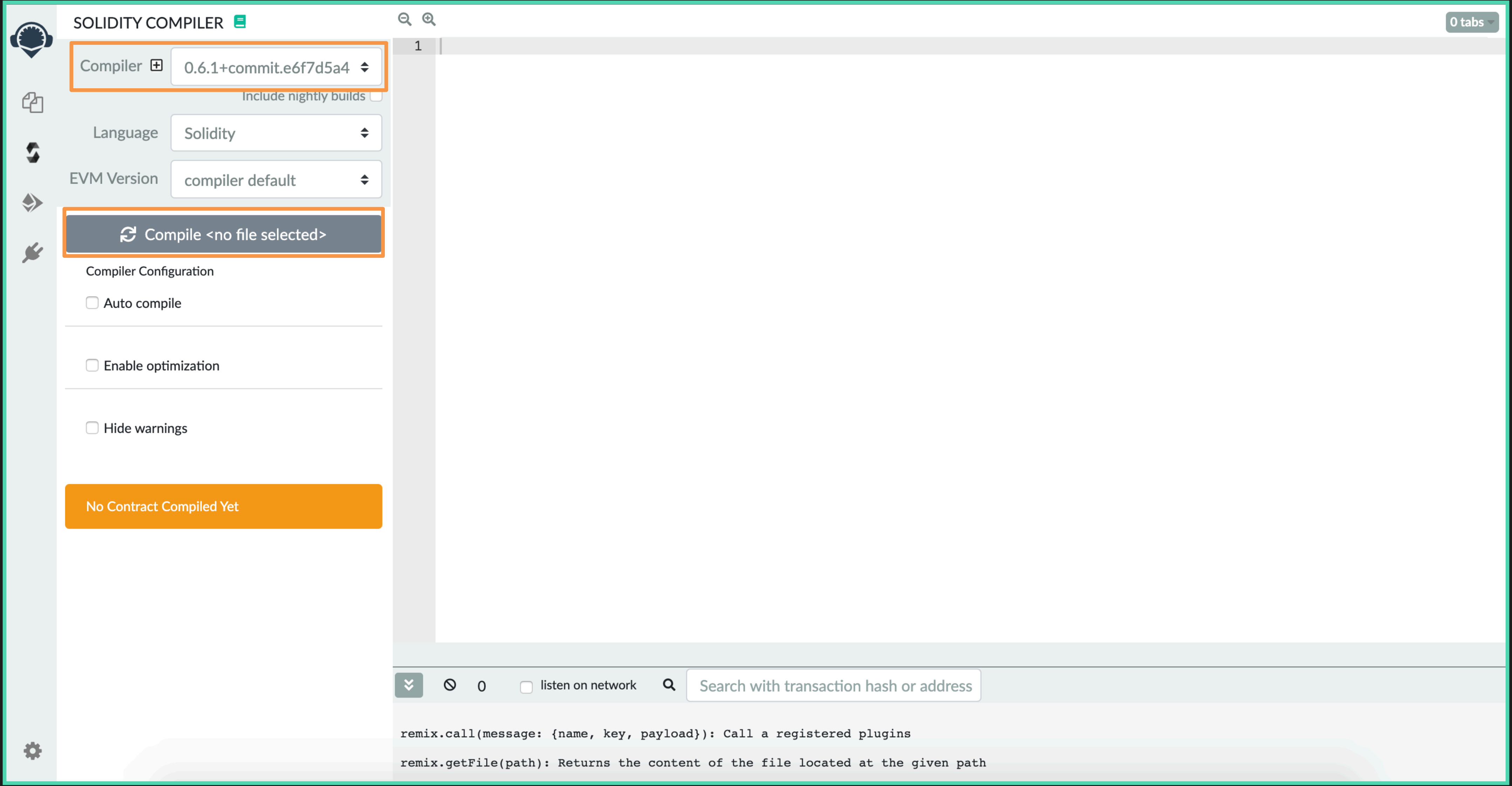


```
remix.call(message: {name, key, payload}): Call a registered plugins
remix.getFile(path): Returns the content of the file located at the given path
```

Solidity Compiler

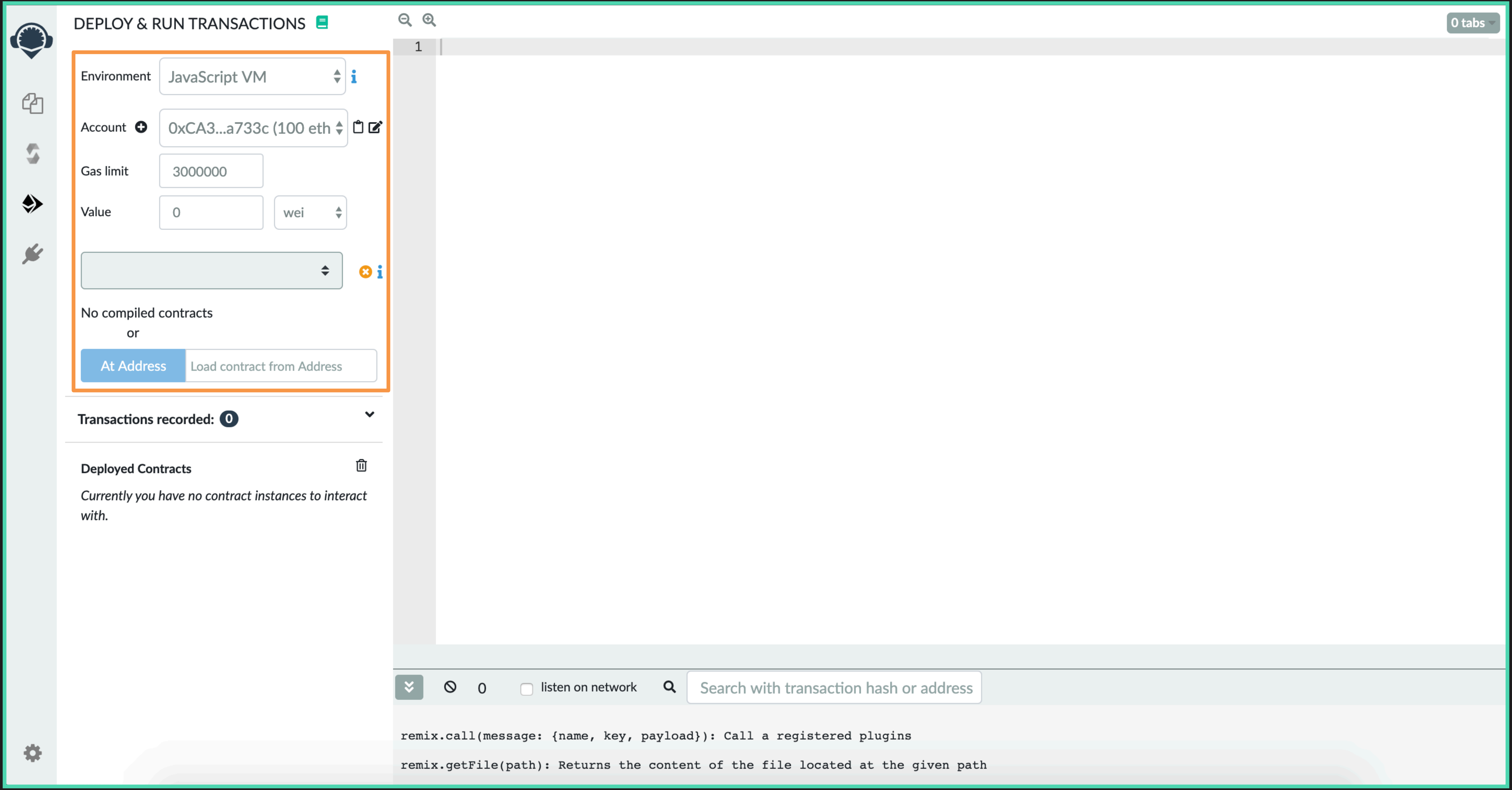
Deploy & run tx

Remix - 線上 IDE



```
remix.call(message: {name, key, payload}): Call a registered plugins
remix.getFile(path): Returns the content of the file located at the given path
```

Remix - 線上 IDE


DEPLOY & RUN TRANSACTIONS
0 tabs
At Address
Load contract from Address
listen on network
Search with transaction hash or address

Environment: JavaScript VM

Account: 0xCA3...a733c (100 eth

Gas limit: 3000000

Value: 0 wei

No compiled contracts
or

Transactions recorded: 0

Deployed Contracts

Currently you have no contract instances to interact with.

```
remix.call(message: {name, key, payload}): Call a registered plugins
remix.getFile(path): Returns the content of the file located at the given path
```

- 智能合約架構簡介 -

基礎架構

- 官方文檔：https://solidity.readthedocs.io/en/v0.6.0/index.html

```
pragma solidity 0.8.0;

contract SimpleStorage {
    uint256 storedData;  ←—— 變數宣告

    function set(uint256 x) public {   ←—— 很多函數
        storedData = x;
    }

    function get() public view returns (uint256) {←—— 很多函數
        return storedData;
    }
}
```

變數宣告

- 變數型態
  - bool
  - int / uint
  - bytes
  - address
  - string
  - array
  - mapping

+

- 能見度
  - public
  - private
  - internal
  - external

+

- 變數名稱

```
int8 public age;
bool private isOwner;
string name;
```

變數宣告

- address

```
address payable public bank;
```

- mapping

```
mapping(address => uint256) public balances;
balances[address] = 10;
uint256 balance = balances[address];
```

- array
    - push
    - pop
    - length

```
uint256[4] fixArr;
uint256[] dynamicArr;
```

特殊變數

- Coin
  - wei
  - gwei
  - finney
  - ether
- Time
  - now
  - seconds
  - minutes
  - hours
  - days
  - weeks
  - years
- Tx
  - tx.orgin
  - tx.gasPrice
- msg
  - msg.sender
  - msg.value
  - msg.data

$$\begin{tikzcd}\textbf{User} \arrow[r] & \textbf{Contract A} \arrow[r] & \textbf{Contract B}\end{tikzcd}$$

特殊變數

- Address
  - address.balance
  - address.tranfser
  - address.send
  - address.call
- Block
  - block.number
  - block.timestamp
  - block.difficulty
  - blockhash (uint)

函數宣告

Storage ⟷ Memory ⟷ Calldata

● 函數名稱(參數) + ● 能見度 + ● 回傳值

- ○ public
- ○ private
- ○ internal
- ○ external ← this.funtion( )

```
function funName() private {…}
function funName2(uint num) external returns(uint8) {…}
function deposit() public payable {…}
```

函數宣告

- View function
- Pure function

  - 不改變合約狀態
  - 函數執行不消耗 gas
  - 不需經過礦工驗證

```
function viewFun(uint256 a, uint256 b) public view returns (uint256) {
    return a * (b + 42) + now;
}

function pureFun(uint256 a, uint256 b) public pure returns (uint256) {
    return a * (b + 42);
}
```

Error Handling

- Assert
  - 燒掉所有 gas
  - 常用於處理非變量
  - 常用於處理溢位
  - 驗證改變後的狀態
  - 一般用於函數結尾
- Require
  - 退回剩餘 gas
  - 常用於驗證 input
  - 常用於驗證條件狀態
  - 一般用於函數開頭
  - 允許 error message
- Revert
  - 退回剩餘 gas
  - 搭配 if / else
  - 允許 error message

特殊函數

revert →

```
function buy(uint amount) public payable {
    if (amount > msg.value / 2 ether)
        revert("Not enough Ether provided.");
}
```

require、assert

↓

```
function sendHalf(address payable addr) public payable returns (uint256 balance) {
    require(msg.value % 2 == 0, "Even value required.");
    uint256 balanceBeforeTransfer = address(this).balance;
    addr.transfer(msg.value / 2);
    assert(address(this).balance == balanceBeforeTransfer - msg.value / 2);
    return address(this).balance;
}
```

特殊函數

- Constructor
  - 合約建構子
  - 只會執行一次
  - 非必須
- Selfdestruct
  - 合約自殺
  - 唯一參數為地址
  - 把合約剩餘的錢給該地址

```
contract shop {
    address payable owner;

    constructor() {
        owner = msg.sender;
    }

    function close() public {
        require(owner == msg.sender);
        selfdestruct(owner);
    }

}
```

特殊函數

Fallback / Receive [payable]

- 沒有 function 宣告
- 沒有參數與回傳值
- 必須是 external
- 預設只有 2300 gas
- 非必要
- 觸發條件：
  1. 單純的轉帳
  2. 呼叫合約沒有的函數

```
contract StandardFallback {

    receive() external payable {}

    fallback() external {}
}
```

Struct

- 自定義變數型態
- mapping 也可以用 (**only value**)
- 可複製，但是 mapping 部分無法

```
struct 變數名稱 {
        成員型態 成員變數名稱;
        成員型態 成員變數名稱;
}
```

Struct 練習

```
contract StructExample1 {
    struct Student {
        string studentName;
    }

    Student student;

    function setStudent(string calldata studentName) external {
        student.studentName = studentName;
    }

    function getStudent() external view returns (string memory) {
        return student.studentName;
    }
}
```

Struct 練習 – mapping

```
contract StructExample2 {
    struct student {
        string studentId;
        string studentName;
    }

    mapping(address => student) public studentAdd;

    function addStudent(string studentId, string studentName) {
        studentAdd[msg.sender] = student(studentId, studentName);
    }
}
```

Struct 練習 – mapping 複製

```
contract StructExample3 {
    struct student {
        address studentAdd;
        mapping(string => string) idToName;
    }

    student student1;
    student student2;

    function setStudent() public{
       student1.idToName["0612221"] = "Gaga";
       student1.studentAdd = msg.sender;
       student2 = student1;
    }
}
```

Event

- 合約內部函數觸發
- 將觸發參數存進 log 中
- Contract 無法直接取 log 的資料
- 額外的儲存空間，很便宜
- 方便 DAPP 監聽事件
- 搭配 **emit** 使用

```
event 事件名稱(參數型態1 參數名稱1, 參數型態2 參數名稱2, ... );
```

Event 練習

```
contract EventExample {
    event buyer(string buyerName, address buyerAdd);

    function register(string calldata name, address add) external {
        emit buyer(name, add);
    }
}
```

Function Modifiers

- 提供函數執行前的檢查、預先處理。
- 支援繼承屬性
- 可接收參數
- 可被覆寫
- 可以多個 modifier

```
modifier 名稱() {
    條件檢查式; //可以有很多個
    _;
}
```

Function Modifiers 練習

```
contract FunctionModifers {
    address payable owner;

    modifier isOwner() {
        require(msg.sender == owner);
        _;
    }

    function kill() public isOwner {
        selfdestruct(owner);
    }
}
```

Getter Function

- 對於所有 public 變數宣告後會自動生成
- Getter function 為 external

```
contract Getter {
    uint8 public num = 10;

    function getNum() public view returns (uint8) {
        return num;
    }

    function getNum2() public view returns (uint8) {
        return this.num();
    }
}
```

Function Overloading

```
function fun1(uint8 num1) external pure returns (uint8) {
        return num1;
    }

function fun1(uint8 num1, uint8 num2) external pure returns (uint8) {
    return num1 + num2;
}
```

Function Multiple Returns

```
function multipleReturn() external view returns (uint256, bool, address) {
    return (block.number, block.number % 2 == 0, msg.sender);
}
```

Inheritance

- 支援多重繼承
- virtual 函數在繼承中可以被修改
- 要修改 virtual 函數，要先宣告 override

```
contract Owned {
    address payable owner;

    constructor() public {
        owner = msg.sender;
    }
}
```

```
contract Shop is Owned {
    string shopName;

    function setName(string calldata _shopName) external virtual {
        if (msg.sender == owner)
            shopName = _shopName;
    }
}
```

Inheritance

```
contract Shop is Owned {
    string shopName;

    function setName(string calldata _shopName) external virtual {
        if (msg.sender == owner)
            shopName = _shopName;
    }
}
contract Mall is Owned, Shop {
    string[] shopArr;

    function setName(string calldata _shopName) external override {
        if (msg.sender == owner) {
            shopName = _shopName;
            shopArr.push(_shopName);
        }
    }
}
```

Abstract Contract

- 合約內有至少一個函數未實現
- 未實現的合約要宣告為 virtual
- 如果繼承者也未完全實現全部函數，也應宣告為 abstract

```
abstract contract Abstract {
    uint8 num;
    function setNum(uint8) public virtual;
    function getNum() public view returns (uint8) {
        return num;
    }
}
```

Interface

- 與 abstract contract 相似，但它沒有實現任何函數
- 不能繼承其他合約或介面
- 不能定義 constructor、變數，**但是 struct 、 enum 、event可以**
- 所有的 function 必須是 **external**，且預設都是 **virtual**
- 就像是蓋房子前的藍圖

```
interface Member {
    function setName(string calldata) external;
    function setAge(uint8) external;
    function getInfo() external returns (string memory, uint8);
}
```

Library

- 函式庫合約不能有狀態儲存
- 不能繼承或被繼承
- 不能被 destroyed
- 不能接受 Ether

OpenZeppelin

```
library Math {
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}
```

- 撰寫第一個智能合約 -

練習1

- 理解 external 和 public 的實際差異
  - 變數宣告：
    - `mapping (字串 → 地址) public students;`
  - 函數宣告：
    - `function publicFun(memory 字串, 地址) public {…}`
    - `function externalFun(calldata 字串, 地址) external {…}`
    - `function callPublicFun(calldata 字串, 地址) external {…}`
    - `function callExternalFun(calldata 字串, 地址) external{…}`

練習2

- 理解 array 操作 with **view**
  - 變數宣告：
    - `address[] students;`
  - 函數宣告：
    - `function addStudent(地址) {…}`
    - `function deleteStudent(Indexs) {…}`
    - `function getStudentLen() view returns(長度){…}`

練習3

- 理解 Constructor 和 Fallback 函數 with msg
  - 變數宣告：
    - `address public payable owner;`
  - 函數宣告：
    - `constructor () {owner = sender}`
    - `fallback () {如果觸發者為 owner 則自殺並且把錢轉給 owner}`
    - `receive () {只要觸發就把錢轉給特定地址}`

練習4

- 在合約中呼叫其他合約
  - 合約1：
    - `function sqr(數字) {回傳平方值};`
    - `function mul(數字1,數字2) {回傳相乘值};`
  - 合約2：
    - `合約1 名稱 = new 合約1();`
    - `function callSqr(數字) {呼叫合約1};`
    - `function callMul(數字1,數字2) {呼叫合約1};`

Bank contract on BSC

- Constructor → 設定合約擁有者
- 存錢 → `function deposit()`
- 提錢 → `function withdraw(uint withdrawAmount)`
- 轉帳 → `function transfer(uint transferAmount, address transferTo)`
- 餘額查詢 → `function getBalance()`
- 銀行資產查詢 → `function getBankBalance()`
- 帳戶註冊 → `function enroll(string studentId)` // mapping(string => address)
- Fallback → 確認是 Owner 卷款錢逃
- Constructor → 設定 Owner
- 防呆說明 → Error message

- END -