



# 區塊鏈實務與安全

資安應用與認知課程教學資源與推廣中心  
許鈞昆

# 大綱

1

Solidity基本語法介紹

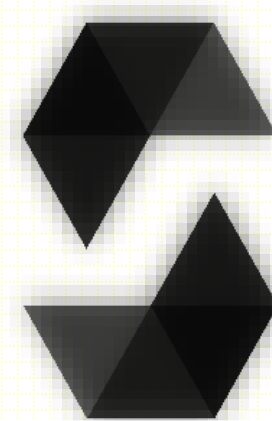
2

智慧合約範例介紹

# Solidity基本語法介紹

# 什麼是Solidity?

- Solidity為一種實現以太坊智慧合約的程式語言(Viper、Serpent)
  - 靜態型別
  - 支援多重繼承(inheritance)、結構(struct)、映射(mapping)
- 類似JavaScript
- 目前仍持續開發中，各個版本之間其語法有所差異
  - 目前最新版本為0.8.17



# Solidity介紹

- 合約第一行 *pragma solidity*
  - 告訴編譯器，編譯原始碼的版本
- 定義合約使用 *contract*

```
// SPDX-License-Identifier: GPL-3.0
```

```
pragma solidity >=0.6.0 <0.8.0;
```

```
contract Example {  
    // your code  
    // hello world  
}
```

授權宣告

版本需大於等於0.6.0，小於0.8.0

合約主體，*Example*為合約名稱

# Solidity介紹

## 資料型態(別)

- 字串 `string`
- 整數 `uint / int`
- 位元組陣列 `bytes`
- 地址 `address`
- 布林 `bool`
- 列舉 `enum`
- 結構 `struct`
- 陣列 `array`
- 映射 `mapping`

## 規則限制

- 沒有浮點數
- 沒有`switch`、`go`
- 新版本不可用中文
- 時間以秒為單位
- 虛擬貨幣單位可為  
`wei`、`szabo`、`finney`或`ether`

# 貨幣單位 & 時間單位

## 貨幣單位

1	<b>wei</b>
$10^3$	ada / kwei
$10^6$	lovelace / mwei
$10^9$	shannon / gwei
$10^{12}$	<b>szabo</b> / microether
$10^{15}$	<b>finney</b> / milliether
$10^{18}$	<b>ether</b>

基礎單位是wei

## 時間單位

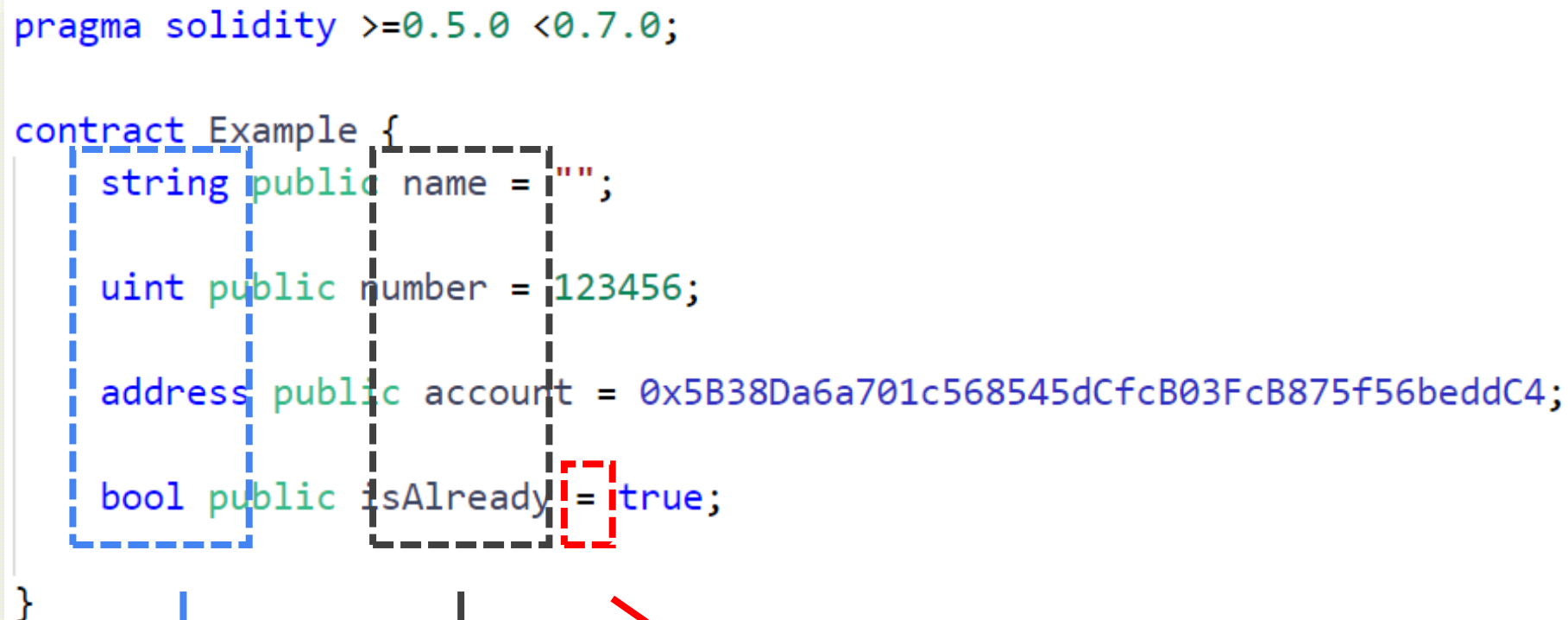
秒	seconds
分	minutes
時	hours
日	days
週	weeks
年	years

基礎單位是seconds

# 資料型態與變數

```
pragma solidity >=0.5.0 <0.7.0;

contract Example {
    string public name = "";
    uint public number = 123456;
    address public account = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
    bool public isAlready = true;
}
```



資料型態

變數名稱

指派(把=右邊的值指給=左邊的變數存起來)

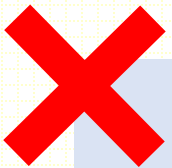


# Solidity資料型態



## 字串 string

- 保存多個字元的資料類型
- 不能直接進行字串串接
- 沒有直接取得字串長度的方法
- 沒有直接比較字串的方法



string.length  
string.length()  
strA + strB  
strA.equals(strB)

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <0.7.0;

contract Example {
    string public nothing;
    string public plus = "+";
    string public hello = "hello";
    string public korean = "안녕하세요";
}
```

# Solidity資料型態



## 整數 uint / int

- uint (unsigned integers)  
為無符號整數，只能記錄正整數
- int (integers)  
可以用來記錄正負數的整數型態

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <0.7.0;

contract Example {
    uint8 unum8 = 255;
    uint256 unum256 = 12345678;

    int8 num8 = 127;
    int256 num256 = -12345678;
}
```

如何得知整數型態的數字範圍？



# Solidity資料型態



## 地址 address

- 存放帳戶地址或合約地址
- 長度160bits
- 以16進制表示

ifier: GPL-3.0

```
pragma solidity >=0.5.0 <0.7.0;
```

```
contract Example {
```

```
    // 外部地址
```

```
    address eoa = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
```

```
    // 合約地址
```

```
    address ca = address(this);
```

```
}
```

# Solidity資料型態



## 位元組陣列 bytes

- 存放byte資料



可變陣列 bytes



固定陣列 bytes1 ~ bytes32

```
// SPDX-License-Identifier: GPL-3.0  
pragma solidity >=0.5.0 <0.7.0;
```

```
contract Example {  
    // 0x61  
    bytes public a = "a";  
    // 0x35353636  
    bytes public number = "5566";  
    // 0x4920616d20736164  
    bytes public song = "I am sad";  
}
```



如何得知bytes的長度?

# Solidity資料型態



## 布林 bool

- 只有二種值
- true / false

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <0.7.0;

contract Example {
    bool isItGoodToDrink = true;
    bool isClicked = false;
}
```



## 列舉 enum

- 自己定義資料型態
- 至少含有一個成員

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <0.7.0;

contract Example {
    enum Stage {Denial, Anger, Barganining, Depression, Acceptance}

    Stage currentStage = Stage.Denial;
}
```

# Solidity常用變數

- *block.timestamp*
  - 區塊此時的時間戳記(以秒為單位)
- *msg.sender*
  - 可以用來記錄此次呼叫合約的帳戶address
    - buyer = msg.sender;
  - 在合約中即會記錄成
    - buyer = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
- *msg.value*
  - 可以用來記錄此次呼叫合約投入的金額
    - money = msg.value;
  - 在合約中若此次投入金額為10 Ether,
    - money = 100000000000000000000000;

# Solidity建構子



## constructor

- 做為合約的初始化，只會執行一次
- 智慧合約不一定要有建構子
- 可帶參數也可不帶參數
- 一個合約只會有一個建構子

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <0.7.0;

contract Example {

    constructor() public {

    }

    constructor(uint8 age) public {

    }

    constructor(string memory name) public {

    }

}
```

# Solidity建構子



試試看

初始化



```
contract Example {  
    string public name;  
    uint8 public age;  
    string public phone;  
    address public account;  
    bool public isMarried;  
  
    constructor() public {  
        name = "Alice";  
        age = 28;  
        phone = "7533967";  
        account = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;  
        isMarried = true;  
    }  
}
```

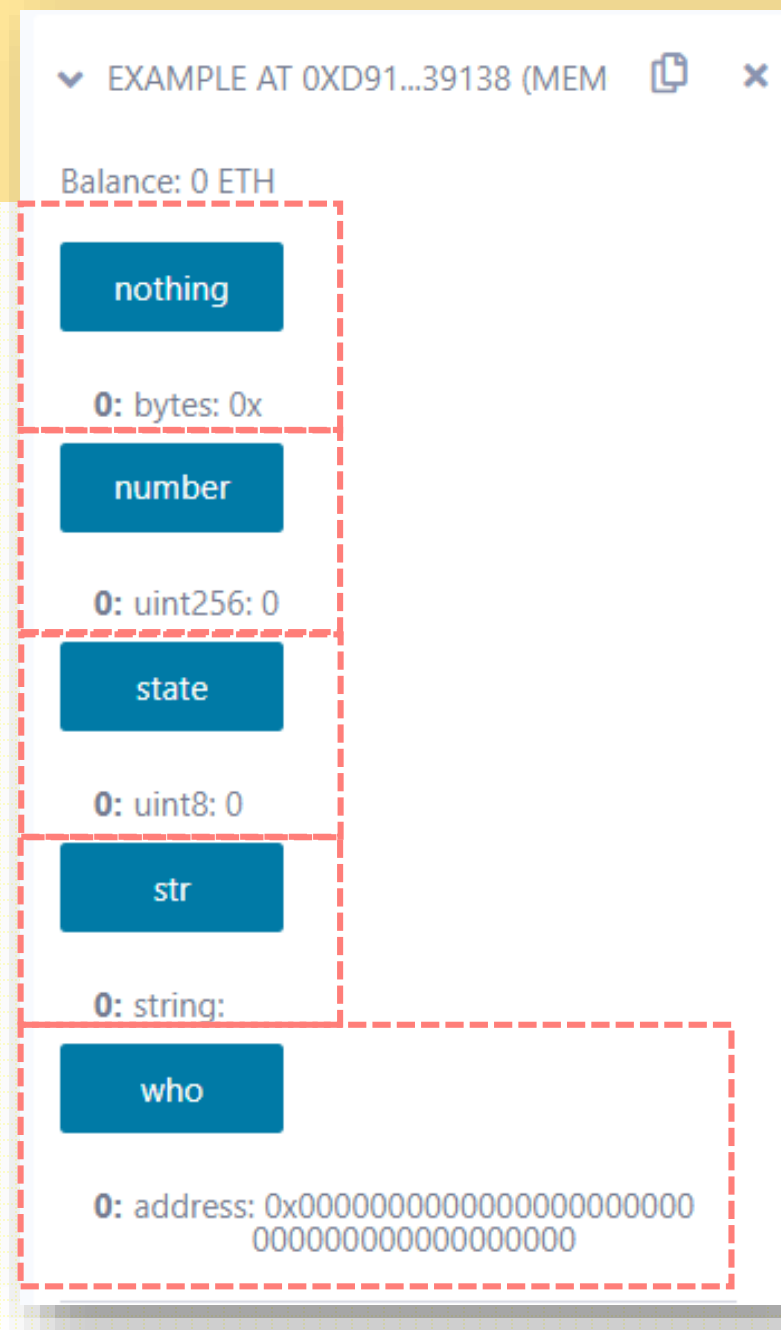


# 預設值

```
pragma solidity >=0.5.0 <0.7.0;

contract Example {
    string public str;
    uint public number;
    address public who;
    bool public haveValue;
    bytes public nothing;

    enum State {Start, End}
    State public state;
}
```



# delete用法



## delete

- 根據資料型態給與變數初始值
- 固定陣列、映射無法使用
- 動態陣列長度歸0

```
contract Example {  
    string public str = "123";  
    uint public number = 100;  
    bool public haveValue = true;  
    bytes public alphabet = "a";  
    address public who = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;  
  
    enum State {Start, Stop, Restart, End}  
    State state = State.Stop;  
  
    function del() public {  
        delete str;  
        delete number;  
        delete haveValue;  
        delete alphabet;  
        delete who;  
        delete state;  
    }  
}
```

# Solidity函式

- *function*
- 用來操作合約行為(實現邏輯), 可以透過它傳遞參數, 也可以返回資料

```
pragma solidity >=0.5.0 <0.7.0;
```

```
contract Example {
```

```
    function foo() public {
```

```
    }
```

```
    function qoo(uint8 age) public {
```

```
    }
```

```
    function sayHello() pure public returns(string memory) {
```

```
        return "Hello";
```

```
    }
```

```
}
```

能見度

參數

指定回傳型態

修飾符號

回傳值

# Solidity 函式



## overloading 多載

- 一樣的函示名稱
- 不一樣的參數

```
contract Example {  
    string public subject;  
    uint8 public score;  
  
    function setData(string memory _subject) public {  
        subject = _subject;  
    }  
  
    function setData(uint8 _score) public {  
        score = _score;  
    }  
  
    function setData(string memory _subject, uint8 _score) public {  
        subject = _subject;  
        score = _score;  
    }  
}
```

# 能見度 Visibility



## public

- 可以被其他合約或自己本身呼叫
- 用於狀態變數時會自動產生同名的getter()函式



## external

- 可以被其他合約或自己本身呼叫，自己呼叫external函式須加上this
- 變數沒有external

# 能見度 Visibility



## internal

- 只能被自己本身或繼承合約呼叫
- 狀態變數預設使用internal

## private

- 只能被自己本身使用

### 能見度使用

能見度	變數	函式	建構子
public	O	O	O
external	X	O	X
internal	O	O	O
private	O	O	X



# 函式內部呼叫

## 由合約的其他函式呼叫

```
contract Example {  
    uint count = 0;  
  
    function addByNumber(uint number) public {  
        count = count + number;  
    }  
  
    function addByOne() public {  
        addByNumber(1);  
    }  
  
    function getCount() public view returns(uint) {  
        return count;  
    }  
}
```

## 遞迴呼叫 Recursive

```
contract Example {  
    // 輾轉相除法  
    function gcd(uint a, uint b) public pure returns(uint) {  
        if (b == 0) {  
            return a;  
        } else {  
            return gcd(b, a % b);  
        }  
    }  
}
```

# 繼承 Inheritance

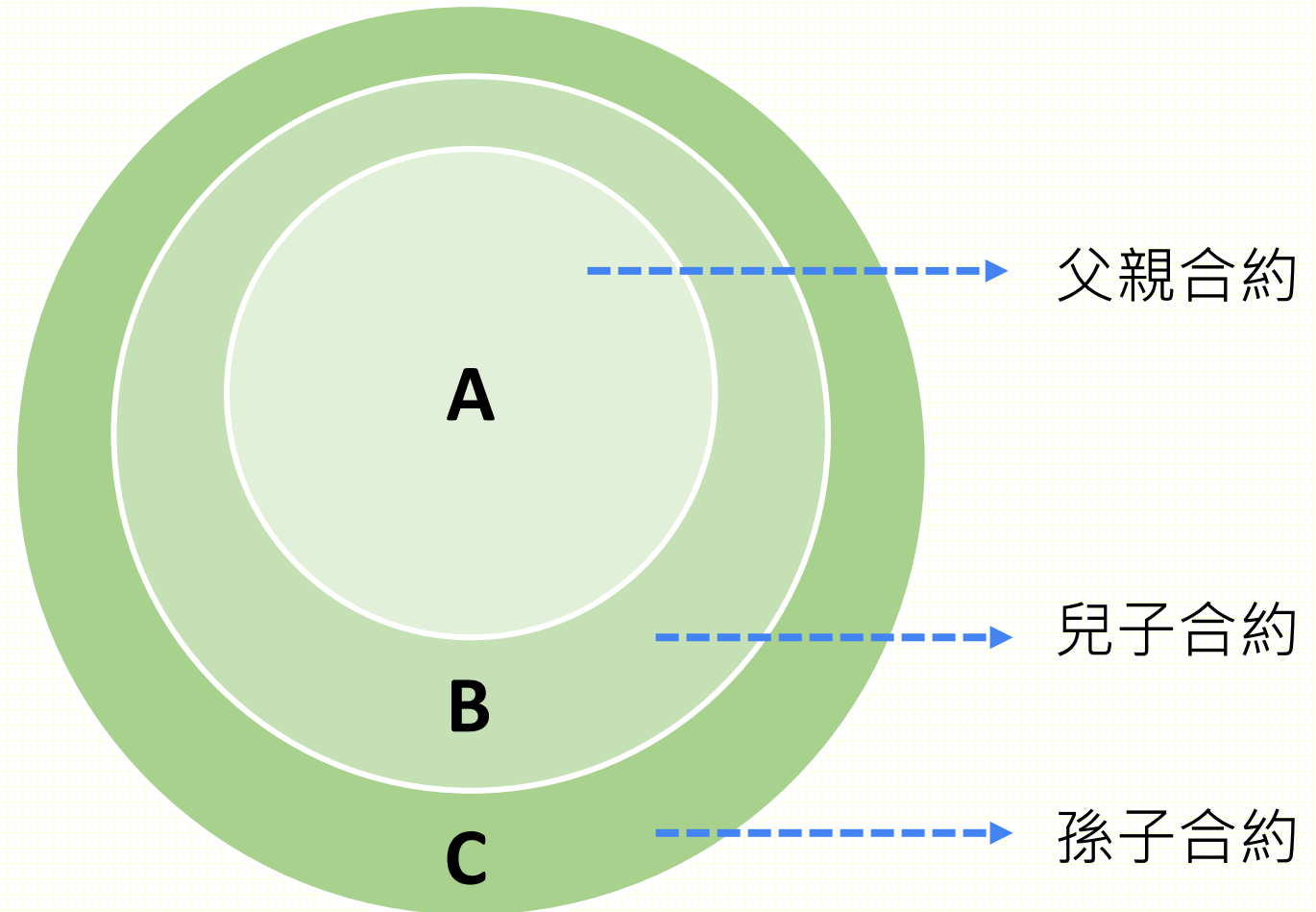


## 繼承 Inheritance

- Solidity支援多重繼承
- 使用保留字`is`

```
contract Parent {  
    // code here  
}
```

```
contract Child is Parent {  
    // code here  
}
```





# 能見度 Visibility



## 能見度調用權限

	public	external	internal	private
內部調用	○	X / ○	○	○
繼承	○	X	○	X
外部調用	○	○	X	X

# 狀態變數 & 區域變數

- 在函式裡宣告命名的，即為區域變數，作用範圍只在函式裡
- 在合約裡宣告命名的，即為狀態變數，作用範圍涵蓋整個合約

```
contract Example {  
    string name = "Alice";  
    uint8 age = 28;  
  
    function foo() public view returns(uint) {  
        uint currentTime = block.timestamp;  
  
        return currentTime;  
    }  
  
    function qoo() public view returns(uint) {  
        return currentTime;  
    }  
}
```

狀態變數

區域變數

編譯錯誤

# 變數儲存修飾



Storage

- state variable
- permanent
- function arguments



Memory

- local variables : array struct mapping
- temporary



Stack

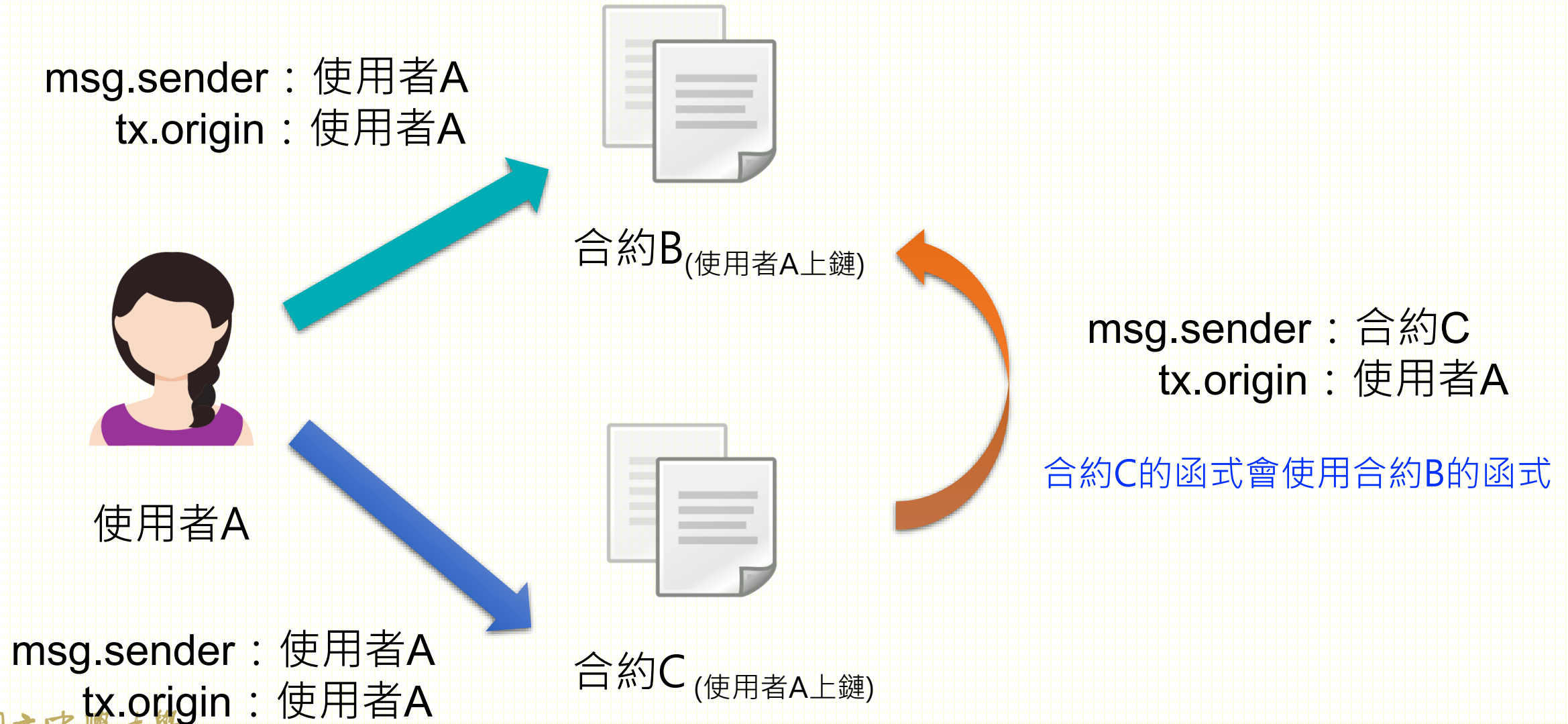
- local variables : uint bool address enum bytes
- temporary

# 常用變數

變數	回傳單位	說明
block.gaslimit	uint	目前區塊的gas limit
block.number	uint	目前區塊的編號
block.difficulty	uint	目前的區塊難度
block.timestamp / now	uint	目前區塊的時間戳記 / 目前時間
msg.sender	address	呼叫函式的帳戶地址
msg.data	bytes	完整的呼叫資料(calldata)
msg.sig	bytes4	calldata的前四個bytes
msg.value	uint	發送多少的以太幣(單位是wei)
tx.gasprice	uint	交易的gas price
tx.origin	address	交易原始發起者的地址



# 常用變數



# 常用變數與函示

變數/函式名稱	回傳單位	說明
<code>this</code>		指目前的合約
<code>gasleft()</code>	<code>uint</code>	剩餘的gas
<code>keccak256(argument)</code>	<code>bytes32</code>	計算傳入值的Ethereum-SHA-3 hash value
<code>sha256(argument)</code>	<code>bytes32</code>	計算傳入值的SHA256 hash value
<code>&lt;address&gt;.balance</code>	<code>uint</code>	該地址的餘額(單位是wei)
<code>&lt;address&gt;.transfer(uint money)</code>		轉錢給指定位址(單位是wei)
<code>address(this)</code>	<code>address</code>	合約地址
<code>selfdestruct(address recipient)</code>		銷毀目前合約，並把合約裡的錢轉到指定的位址

# 錯誤處理函式

變數/函式名稱	說明
<code>assert(bool condition)</code>	如果條件不滿足， <code>transaction</code> 會失敗，拿走所有gas
<code>require(bool condition)</code>	如果條件不符合，狀態會被還原，退回使用到的gas
<code>require(bool condition, string message)</code>	如果條件不符合，狀態會被還原(可自訂錯誤訊息)
<code>revert()</code>	執行會被終止，並回復到修改前的狀態
<code>revert(string message)</code>	執行會被終止，並回復到修改前的狀態(可自訂錯誤訊息)

# 函式回傳值

```
contract Example {
```

```
    function poo() pure public {
```

```
    }
```

沒有回傳值

```
    function foo() view public returns(uint) {
```

```
        return block.timestamp;
```

```
    }
```

一個回傳值

```
    function qoo() public view returns(address, uint) {
```

```
        return (msg.sender, 0);
```

```
    }
```

多個回傳值

```
    function boo() public view {
```

```
        uint time = foo();
```

```
        (address account, uint money) = qoo();
```

```
    }
```

多個回傳值指派

```
}
```



# 修飾符



## view

- 當函式不修改任何狀態時

## pure

- 當函式不讀取或不修改狀態時

## payable

- 讓函式可以收取以太幣
- 也會用在address資料型態

```
contract Example {
    uint count = 0;

    function add(uint number) public {
        count = count + number;
    }

    function getCount() public view returns(uint) {
        return count;
    }

    function add2(uint number) public pure returns(uint) {
        return number + 1;
    }

    function getValue() public payable returns(uint) {
        return msg.value;
    }
}
```

# 修飾符



## constant

- 用來表示變數的值是不可變動的
- 變數宣告時即需指派值

```
pragma solidity >=0.5.0 <0.7.0;
```

```
contract Example {
```

```
    string constant NAME = "Alice";
```

```
    uint constant AGE;
```

```
    constructor() public {
```

```
        AGE = 28;
```

```
    }
```

正確宣告

編譯錯誤，未指派值給變數AGE

編譯錯誤，重新指派值給變數AGE

# 運算符號



## 算術運算子

項目	符號	表示	$a = 10, b = 3$
加	+	$a = a + b;$	$a = 13$
減	-	$a = a - b;$	$a = 7$
乘	*	$a = a * b;$	$a = 30$
除	/	$a = a / b;$	$a = 3$
餘數	%	$a = a \% b;$	$a = 1$
遞增	++	$a++; ++a;$	$a = 11$
遞減	--	$a--; --a;$	$a = 9$

```
m = 10;  
m = m++;  
m = ?
```

```
n = 10;  
n = n--;  
n = ?
```

```
x = 10;  
x = ++x;  
x = ?
```

```
y = 10;  
y = --y;  
y = ?
```



# Solidity練習(一)

- 讓使用者輸入二個整數變數
- 可以針對這二個整數計算加減乘除
  - 加+ 減- 乘\* 除/
- 將計算後的結果儲存起來
- 透過函式回傳

運算符號跟函式一起練習





# Solidity練習(二)

- 讓使用者輸入考試成績
- 計算平均值
  - 需有一變數記錄總共輸入幾科考試成績
- 回傳平均值

你會需要知道總共輸入幾次成績



# 運算符號

## 比較運算子

項目	符號	表示	a = 10, b = 3
大於	>	a > b	true
小於	<	a < b	false
等於	==	a == b	false
不等於	!=	a != b	true
大於等於	>=	a >= b	true
小於等於	<=	a <= b	false

要嘛true，  
要嘛false。

AB型



# 運算符號



## 邏輯運算子

項目	符號	表示	a = true, b = false
And	&&	a && b	false
Or		a    b	true
Not	!	!a	false

## 三元運算子

項目	符號	表示	
Conditional	?:	<conditional> ? <if-true> : <if-false>	(75 > 60) ? "Pass" : "Fail";
			((6%2)==0) ? "even" : "odd";
			(isTHSR) ? 700 : 375;

# 字串操作



```
function compare(string memory s1, string memory s2) public pure returns(bool)
{
    bytes memory bytes_s1 = bytes(s1);
    bytes memory bytes_s2 = bytes(s2);

    return keccak256(bytes_s1) == keccak256(bytes_s2);
}
```



比較字串

```
function concat(string memory a, string memory b) public pure returns(string memory)
{
    return string(abi.encodePacked(a, b));
}
```



串接字串



# 流程控制



## 判斷式 if ... else ...

```
if (condition) {  
    // code be executed as condition is true  
}
```

```
if (condition) {  
    // condition is true  
} else {  
    // condition is false  
}
```

```
if (condition1) {  
    // condition1 is true  
} else if (condition2) {  
    // condition1 is true  
} else if (condition3) {  
    // condition1 is true  
} else {  
  
}
```

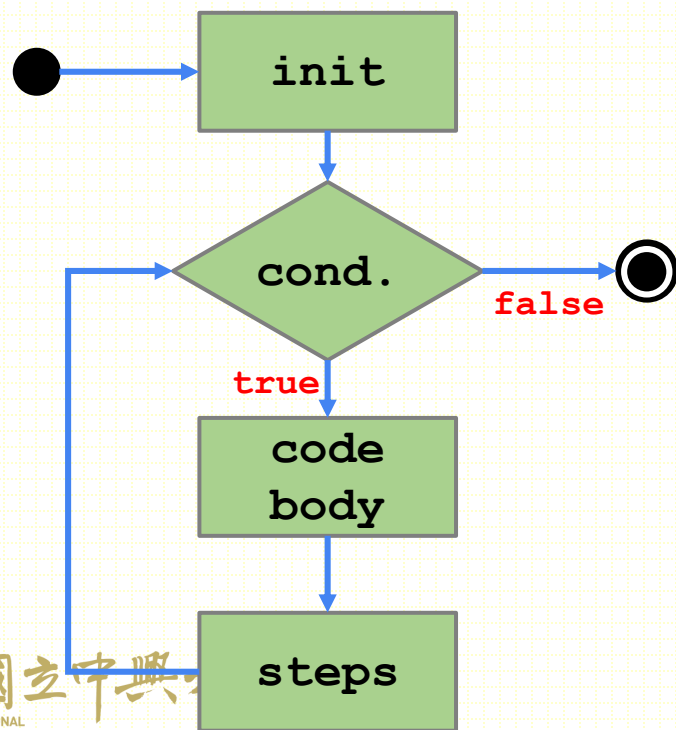
```
contract Example {  
  
    function abs(uint a, uint b) public pure returns(uint) {  
        if (a > b) {  
            return a - b;  
        } else {  
            b - a;  
        }  
    }  
  
}
```

```
contract Example {  
  
    function getResult(uint score) public pure returns(string memory) {  
        if (score >= 90) {  
            return "繼續保持";  
        } else if (score >= 75) {  
            return "還有進步空間";  
        } else if (score >= 60) {  
            return "要再努力";  
        } else {  
            return "不當你當誰";  
        }  
    }  
  
}
```

# 流程控制

## 迴圈 for loop

```
for (init; condition; steps) {  
    // code here  
}
```



```
contract Example {  
    uint public sum = 0;  
  
    function doStuff() public {  
        for (uint i = 0; i <= 10; i++) {  
            sum = sum + i;  
        }  
    }  
}
```

```
contract Example {  
    uint public sum = 0;  
  
    function doStuff() public {  
        sum = 0;  
        for (uint j = 100; j > 0; j--) {  
            if (j % 10 == 0 && j % 4 != 0) {  
                sum = sum + j;  
            }  
        }  
    }  
}
```





# Solidity練習(三)

- 閏年 Leap Year
- 判斷規則
  - 公元年份不是4的倍數，則為平年
  - 公元年份為4的倍數但不是100的倍數，則為閏年
  - 公元年份為100的倍數但不是400的倍數，則為平年
  - 公元年份為400的倍數，則為閏年

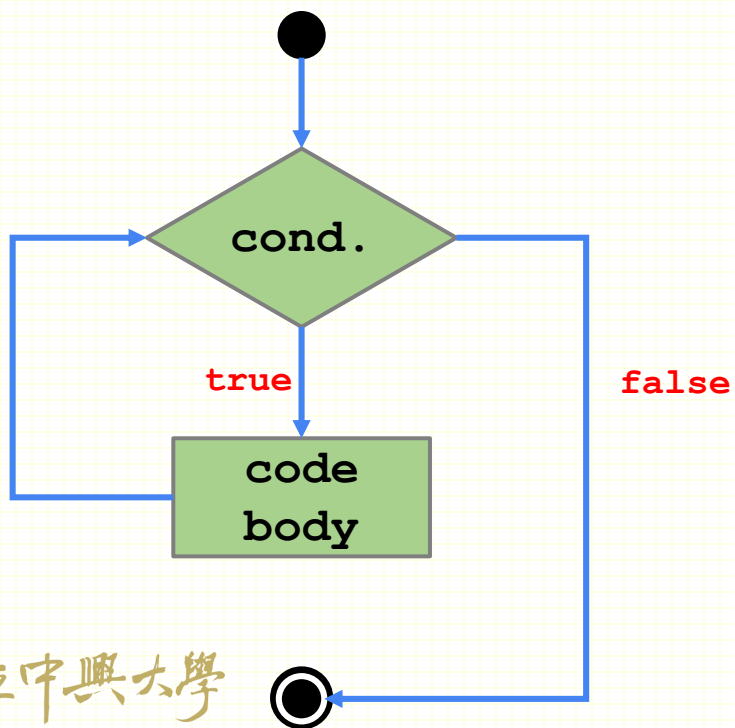
閏年          1996 2016 2020

平年          1900 2019 2100

# 流程控制

## 迴圈 while

```
while (condition) {  
    // code here  
}
```



```
contract Example {  
  
    uint public sleepHour;  
  
    function doStuff() public {  
        while (sleepHour < 8) {  
            sleepHour++;  
        }  
    }  
}
```

```
contract Example {  
    bool public sleep = true;  
    uint public hour = 0;  
  
    function doStuff() public {  
        while (sleep) {  
            hour++;  
  
            if (hour == 8) sleep = !sleep;  
        }  
    }  
}
```

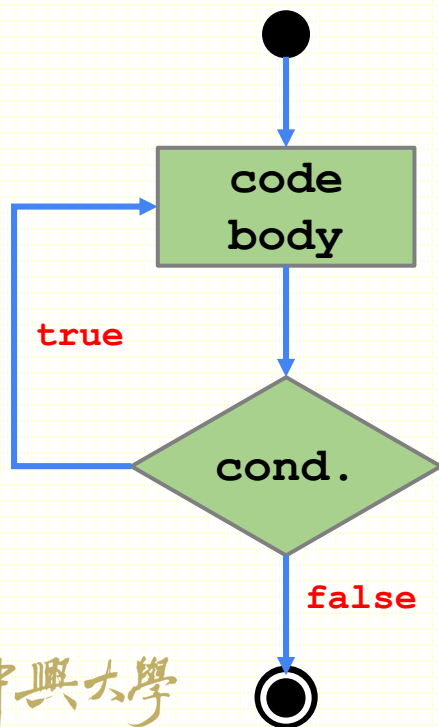


# 流程控制



## 迴圈 do while

```
do {  
    // code here  
} while (condition);
```



```
contract Example {  
  
    uint public sleepHour;  
  
    function doStuff() public {  
        do {  
            sleepHour++;  
        } while (sleepHour < 8);  
    }  
}
```

```
contract Example {  
    bool public sleep = true;  
    uint public hour = 0;  
  
    function doStuff() public {  
        do {  
            hour++;  
  
            if (hour == 8) sleep = !sleep;  
        } while (sleep);  
    }  
}
```

# Solidity資料型態



## 陣列 array



array1: 固定長度



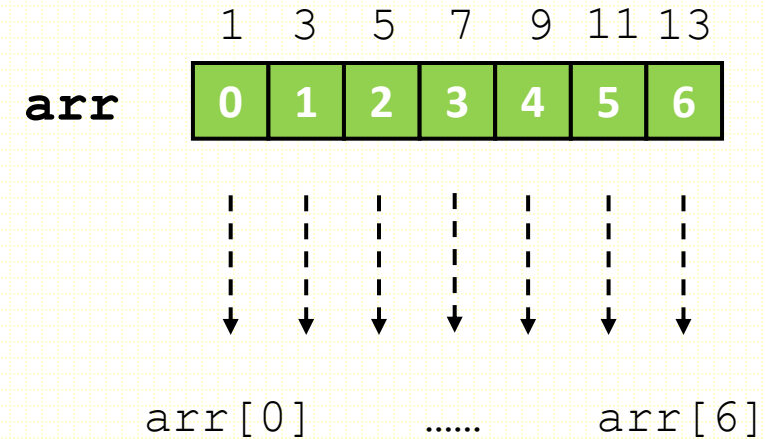
array2: 可改變長度



array3: 可改變長度

```
contract Example {  
    uint[5] array1 = [1, 2, 3, 4, 5];  
  
    uint[] array2 = [1, 2, 3, 4, 5];  
  
    uint[] array3 = new uint[](5);  
}
```

取值方式 → 陣列名稱[索引]



取得陣列的長度 `uint length = array.length;`



# Solidity練習(四)

- 費波納西數列 `fibonacci`
  - 假設第1項與第2項的值皆為1
  - 1, 1, 2, 3, 5, 8, 13, ...
- 取得第n個費波納西數列的值
  - 設計一個函數，讓使用者輸入n，回傳數列第n項的值
  - 使用三個變數
- 根據輸入的整數，判斷其在費波納西數列裡的第幾項
  - 設計一個函數，讓使用者輸入一個數字，回傳其在數列的第幾項
  - 假如該輸入的數字不在數列裡，則回傳0



# Solidity練習(五)

- 一陣列
  - `uint[10] nums = [12, 23, 58, 6, 97, 41, 85, 30, 69, 47];`
- 利用迴圈，分別找出最大值與最小值
- 找出陣列裡的奇數數字與偶數數字，並回傳其總和



# 自訂修飾符



## modifier

- 可以自行定義的函式修飾符號
- 類似if else功能，讓程式碼更容易編輯
- 易於設計與維護

## require()

- `require(condition);`
- `require(condition, message);`

```
modifier <Name> {  
    require(msg.sender == owner);  
    ;  
}
```

```
modifier <Name>(uint amount) {  
    require(msg.value == amount);  
    ;  
}
```



# 自訂修飾符

```
contract Example {  
    address owner;  
  
    function setOwner(address newOne) public {  
        owner = newOne;  
    }  
  
    function doA() public view {  
        if (msg.sender == owner) {  
            // some code ...  
        }  
    }  
  
    function doB() public view {  
        if (msg.sender == owner) {  
            // some code ...  
        }  
    }  
}
```

沒有使用modifier

```
contract Example {  
    address owner;  
  
    function setOwner(address newOne) public {  
        owner = newOne;  
    }  
  
    modifier checkOwner() {  
        require(msg.sender == owner, "you are not the owner");  
        _;  
    }  
  
    function doA() public checkOwner {  
        // some code ...  
    }  
  
    function doB() public checkOwner {  
        // some code ...  
    }  
}
```

使用modifier

# Solidity資料型態



## 結構 struct

- 字串、整型、布林、地址、映射、結構
- internal function可回傳struct型態
- member access operator (.)

```
struct <Name> {  
    type1 varName1;  
    type2 varName2;  
    type3 varName3;  
}
```

```
contract BookStore {  
    struct Book {  
        uint isbn;  
        string bookName;  
        uint bookPrice;  
    }  
  
    Book newBook;  
  
    function create() public {  
        newBook = Book(1, "演員的自我修養", 100);  
  
        newBook = Book({  
            isbn : 2,  
            bookName : "一蓮托生品",  
            bookPrice : 150  
        });  
    }  
  
    function getBookPrice() public view returns(uint) {  
        return newBook.bookPrice;  
    }  
}
```

# Solidity資料型態



```
pragma solidity >=0.5.0 <0.7.0;  
pragma experimental ABIEncoderV2;
```



要讓函式支援public回傳struct  
pragma experimental ABIEncoderV2  
(0.8以下版本)

```
contract BookStore {  
    struct Book {  
        uint isbn;  
        string bookName;  
        uint bookPrice;  
    }  
}
```

```
function get() public pure returns(Book memory) {  
    return Book(1, "演員的自我修養", 100);  
}
```



用public回傳struct

```
function get2() internal pure returns(Book memory) {  
    return Book(3, "唐詩三百首", 200);  
}
```



internal function  
才能回傳struct

# Solidity資料型態



## 映射 mapping

key	value
uint	string
1	Alice
2	Bob
3	Cindy

StudentName

mapping (keyType => valueType) <mappingName>;

資料型態	Key Type	Value Type
uint / int	0	0
string	0	0
bytes	0	0
address	0	0
bool	0	0
enum	0	0
array	X	0
struct	X	0
mapping	X	0



# struct + mapping

```
contract BookStore {  
  struct Book {  
    uint isbn;  
    string bookName;  
    uint bookPrice;  
  }  
  
  mapping (uint => Book) public bookList;  
  
  function create(uint no, string memory name, uint price) public {  
    bookList[no] = Book(no, name, price);  
  }  
  
  function findBookByISBN(uint isbn) internal view returns(Book memory) {  
    return bookList[isbn];  
  }  
  
  function findBookNameByISBN(uint isbn) public view returns(string memory) {  
    return bookList[isbn].bookName;  
  }  
}
```

自定義結構型態

自定義映射

派值方式

取值方式

# Solidity事件



## 事件 event

- 用來記錄日誌資料(Log)的方法
- 使用時須加上emit

```
contract Bank {  
  
    event Log(address who, uint money);  
  
    function deposit(uint money) public {  
        // do something ...  
        emit Log(msg.sender, money);  
    }  
}
```

宣告事件

觸發合約事件



# 函式庫Library

- 把函式打包成函式庫，讓開發者引入函式庫來使用這些函式，以減少重複的程式碼並節省gas花費
- 類似Java的靜態類別
- 特性
  - 不可以使用狀態變數
  - 不能使用event
  - 不能使用payable
  - 使用import保留字引用已部署之函式庫(OpenZeppelin)



# OpenZeppelin



OpenZeppelin | contracts

## 智慧合約

- 使用Solidity
- 開源、安全
- 降低程式中的漏洞發生率

## 安全審計

- 協助檢視智慧合約的漏洞

# OpenZeppelin

名稱	模組	說明
Ownable.sol	access	限制某些特定合約函式的操作權限
Roles.sol	access	允許合約在部署前定義多種權限的角色
AccessControl.sol	access	定義哪個帳號可以進行角色的授權與回收
SafeMath.sol	math	安全的算術運算
SafeCast.sol	utils	安全的進行型別轉換
ERCXXX.sol	token	提供 <b>ERC</b> 代幣標準基本實作
Address.sol	utils	地址辨識

# ERC-1155 Example

```
// contracts/GameItems.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";

contract GameItems is ERC1155 {
    uint256 public constant GOLD = 0;
    uint256 public constant SILVER = 1;
    uint256 public constant THORS_HAMMER = 2;
    uint256 public constant SWORD = 3;
    uint256 public constant SHIELD = 4;

    constructor() ERC1155("https://game.example/api/item/{id}.json") {
        _mint(msg.sender, GOLD, 10**18, "");
        _mint(msg.sender, SILVER, 10**27, "");
        _mint(msg.sender, THORS_HAMMER, 1, "");
        _mint(msg.sender, SWORD, 10**9, "");
        _mint(msg.sender, SHIELD, 10**9, "");
    }
}
```

引用外部函示庫

鏈接外部URI

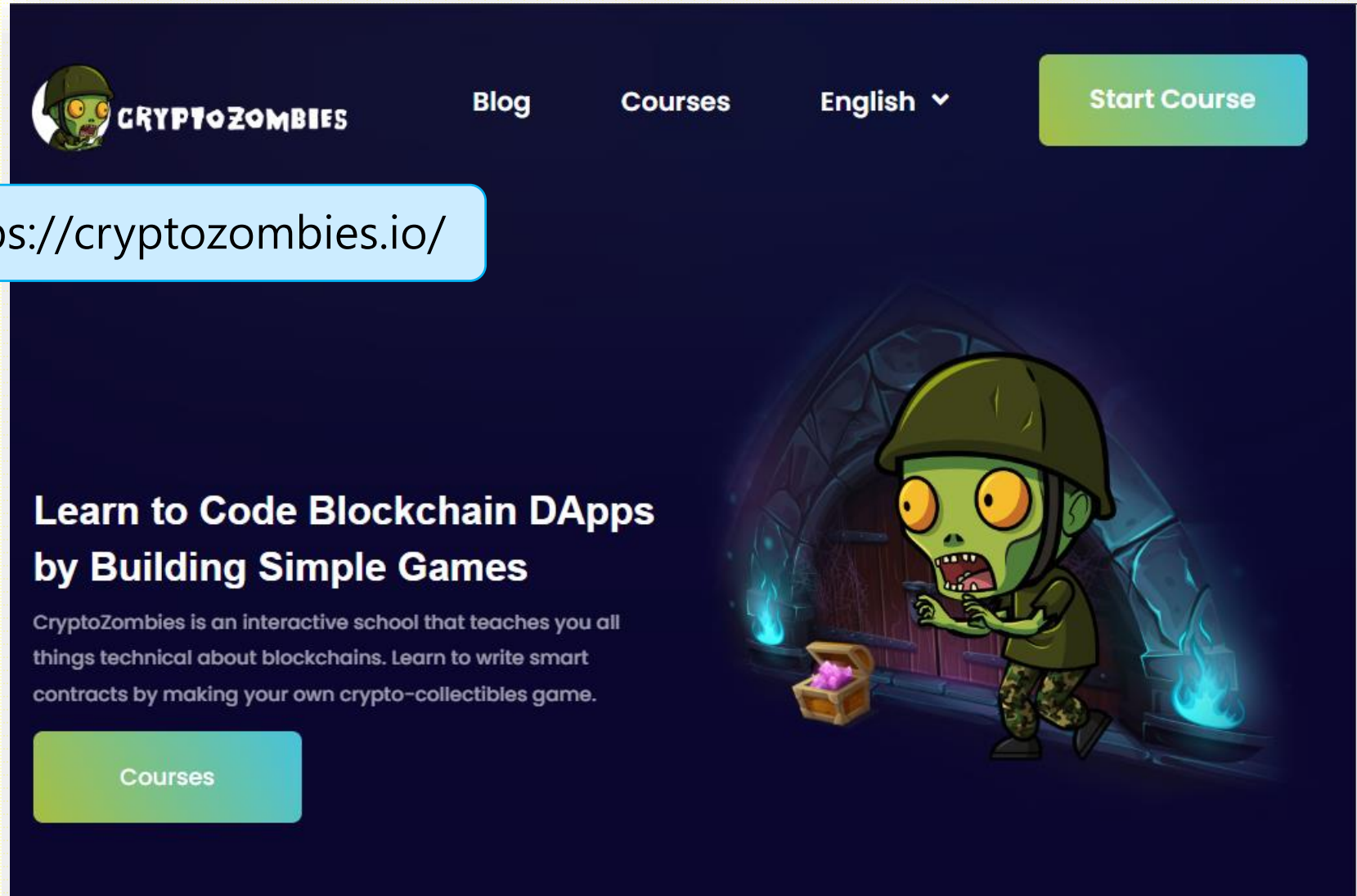
metadata

```
{
  "name": "Thor's hammer",
  "description": "Mjöltnir, the legendary hammer of the Norse god of thunder.",
  "image": "https://game.example/item-id-8u5h2m.png",
  "strength": 20
}
```

# Learn to Code



<https://cryptozombies.io/>



# Learn to Code



根據名稱  
變化不同種類殭屍

# 智慧合約範例介紹

# 合約範例練習(1)



- 開學團購課本
- 合約需顯示發起人地址、書商地址、書本名稱、書本價格及合約開始時間
- 智慧合約發行後，可以讓使用者登記買書，進行購買後，合約會記錄購買者的地址、購買數量及最新購買時間。
- 最後執行智慧合約上的發送款項功能(發起人才能執行)，將合約內之所有Ether發送給書商。

# 合約範例練習(1)



## 合約需求

variable

書商地址  
書本名稱  
書本價格  
發起人地址  
合約開始時間

struct

- 購買資訊
- 購買者地址
  - 購買數量
  - 購買時間

mapping

購買者資料表



# 合約範例練習(1)



## 合約需求

初始化

書商地址  
書本名稱  
書本價格  
發起人地址  
合約開始時間

function

登記買書  
發送款項

modifier

合約發行者才能執行

# 合約範例練習(1)

## 登記買書

- 1) 購買者輸入欲購買數量及總計費用
- 2) 透過mapping方式記錄買書者之address、購買數量、購買時間
- 3) 檢查輸入金額是否正確
- 4) 重複購買需累加書本數量

## 發送款項

- 1) 只有合約發起者可以操作此函式
- 2) 將合約內的款額發送給書商

# 合約範例練習(2)



- 慈善撲克王
- 簡易募款智慧合約，可以讓使用者進行捐贈動作。進行捐贈後，合約會記錄捐贈者之address、捐贈金額及捐贈時間。
- 最後執行智慧合約上的發送募款功能，將合約內之所有Ether發送給慈善機構。

# 合約範例練習(2)



## 合約需求

variable

慈善機構名稱  
慈善機構地址  
發起人姓名  
發起人地址  
合約開始時間

struct

捐款人

- 帳戶地址
- 捐款金額
- 捐款時間

mapping

捐款人資料表

# 合約範例練習(2)



## 合約需求

初始化

慈善機構名稱  
慈善機構地址  
發起人姓名  
發起人地址  
合約開始時間

function

捐款  
發送捐款  
查詢款項總額

modifier

合約發行者才能執行

# 合約範例練習(2)

## 捐款

- 1) 捐款者可以自行輸入欲投入之金額
- 2) 透過mapping方式記錄捐款者之address、捐款金額、捐款時間

## 發送捐款

- 1) 只有合約發起者可以操作此函式
- 2) 將合約內的款項發送到慈善機構

## 查詢款項總額

- 1) 查詢目前捐款總額

# 合約範例練習(3)



- 終極密碼
- 由合約發起人舉辦遊戲，發起人可以自訂每次猜數字所必須支付的以太幣數量，合約發起人不能參與猜數字。
- 當玩家猜對數字後會將獎金池(合約)中的九成金額轉給贏家，剩餘金額轉給遊戲發起人。

# 合約範例練習(3)



## 合約需求

variable

合約發起人  
贏家  
投入金額  
答案數字

enum

遊戲的狀態

- Start
- End

mapping

紀錄玩家猜測次數



# 合約範例練習(3)



## 合約需求

初始化

設定合約擁有者為莊家  
下注金額  
初始化遊戲  
隨機產生亂數作為答案

function

發送獎金  
猜數字  
查看合約金額

modifier

合約發行者才能執行  
檢查遊戲狀態

```
// 亂數產生數字範圍 0 ~ 99 (不安全作法)
```

```
uint256(keccak256(abi.encodePacked(block.timestamp))) % 100;
```

# 合約範例練習(3)

## 猜數字函式

- 1) 透過修飾符判斷目前的狀態是否為Start ，正確才繼續以下操作。
- 2) 合約擁有者不可參與活動
- 3) 金額是否輸入正確
- 4) 猜數字大小不可以大於100
- 5) 將每位玩家猜的次數記錄下來
- 6) 猜的數字大於正確數字，回傳”bigger”
- 7) 猜的數字小於正確數字，回傳”smaller”
- 8) 猜中正確數字回傳”You win”，並且記錄贏家，將狀態改為End

# 合約範例練習(3)

## 發送獎金函式

- 1) 只有發起人可以操作此函式
- 2) 判斷目前的狀態是否為End, 正確才繼續以下操作。
- 3) 贏家可以獲得九成獎金
- 4) 發起人可以獲得剩餘獎金

## 查看合約金額函式

- 1) 查看目前的總獎金

# 合約範例練習(4)



- 統一發票
- 設計一個紀錄統一發票的智慧合約
- 記錄每張發票的編號、消費時間、總金額、明細

# 合約範例練習(4)



## 合約需求

struct

- 發票資訊
- 發票號碼
  - 消費時間
  - 總金額
  - 記錄放了多少筆明細
  - 消費明細mapping

struct

- 發票明細
- 商品名稱
  - 商品價錢

# 合約範例練習(4)



合約需求

mapping

紀錄所有發票

function

新增發票

新增某張發票的明細  
印出某張發票的明細

# 合約範例練習(5)



- 大醫院小醫師
- 醫院為病患註冊個人基本資料，並且記錄住院明細。
- 病患出院後可以授權保險公司至醫院調閱病歷，保險公司可以自動理賠病患住院費用。

# 合約範例練習(5)



## 合約需求

variable

醫院地址

struct

病歷資料

- 病症
- 病因
- 住院日期
- 住院天數
- 住院花費
- 是否有紀錄
- 是否已繳費

struct

病人資料

- 姓名
- 居住地址
- 病歷總量
- 病歷映射
- 保險公司
- 是否有紀錄



# 合約範例練習(5)



## 合約需求

mapping

紀錄病患所有資料  
授權狀態

function

新增病患基本資料  
新增病患住院資料  
病患繳清帳單  
病患授權保險公司  
保險公司申請病患資料  
保險公司理賠

modifier

合約發行人才 able 執行

# 合約範例練習(5)

## 新增病患基本資料

- 1) 只有醫院可以操作此函式
- 2) 紀錄前須先判斷病患是否曾經註冊過
- 3) 新增病患資料至病人結構當中

# 合約範例練習(5)

## 新增病患住院資料

- 1) 只有醫院可以操作此函式
- 2) 紀錄前須先判斷病患是否存在
- 3) 累加病歷總量作為索引值
- 4) 新增病患住院資料至病歷結構中
- 5) 回傳此次住院病歷索引(uint)

# 合約範例練習(5)

## 病患繳清帳單

- 1) 紀錄前須先判斷病患是否存在
- 2) 紀錄前須先判斷病歷是否存在
- 3) 判斷是否已付款成功
- 4) 判斷金額是否正確
- 5) 付款至醫院合約當中
- 6) 付款成功後，修正病歷結構裡的是否已繳費欄位

# 合約範例練習(5)

## 病患授權保險公司

- 1) 儲存病患的保險公司地址
- 2) 修改授權狀態

# 合約範例練習(5)

## 保險公司申請病患資料

- 1) 判斷病患是否同意授權
- 2) 判斷病患是否存在
- 3) 判斷病歷是否存在
- 4) 回傳病歷資料

# 合約範例練習(5)

## 保險公司理賠

- 1) 判斷病患是否同意授權
- 2) 判斷病患是否已付款給醫院
- 3) 判斷理賠金額與病患付款金額是否相等
- 4) 保險公司轉ETH給病患