Web Programming Spring 2021

#7

Chi-Jen Wu

Topics

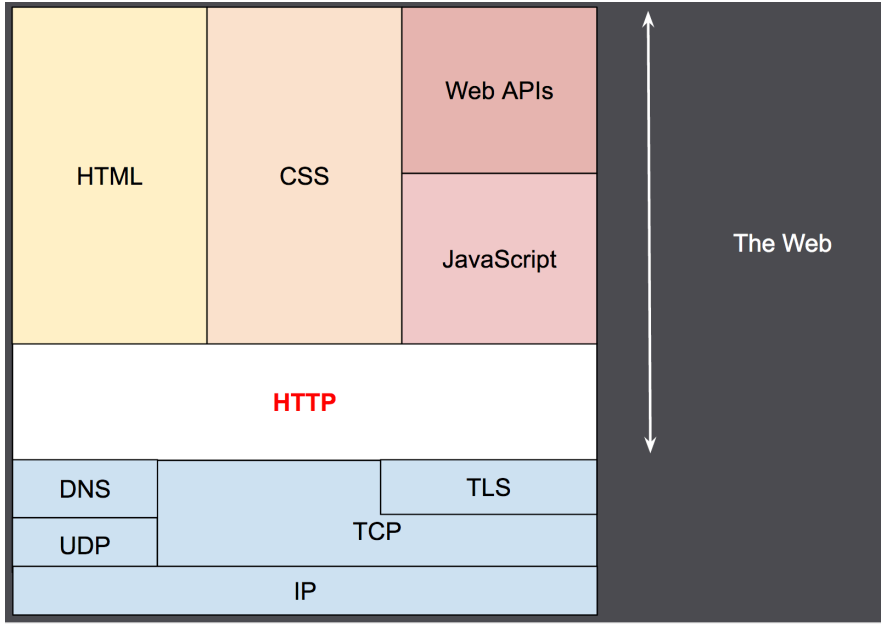
- The concepts of Web Services
- Web data protocols
  - HTTP, WebSocket, WebRTC
  - HTML, CSS
- **Web JavaScript programming**
- Cookies and sessions
- Web Frontend frameworks
- Web Backend frameworks
- RESTful API design

GitHub

Google Analytics

Web data protocols

- HTTP, HTTPS
- Web APIs
- HTML, HTML5
- CSS, CSS3
- **JavaScript**
- Conclusion



JavaScript

- Introduction
- Basics
- Document Object Model
- Browser Object Model
- JavaScript ES6
- jQuery & AJAX

JavaScript basics

- Dev Environment & Debug
- Data Types
- Variable
- Operator
- Control Structure
- Array
- JSON

JavaScript 開發環境與開發工具

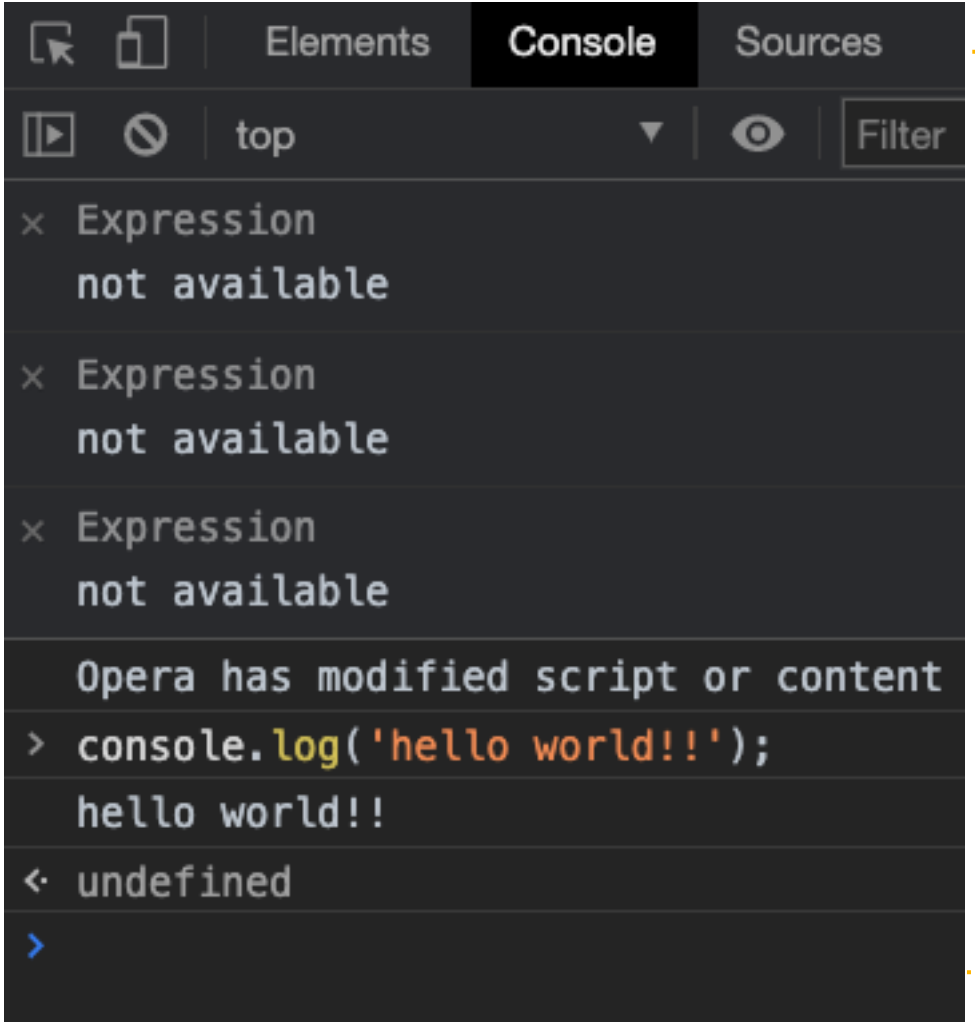
- Online JavaScript Editor
- Visual Studio Code (VS Code)
- Node.js, npm (npm install http-server -g)
- Google Chrome

- Option
  - eslint: 程式碼檢查工具
  - webpack: 模組的打包程式
  - Flow: 靜態類型檢查工具
  - Live Server, HTML CSS Support, Prettier, Code Runner

JavaScript Debug

- Visual Studio Code
- Select “Run”
- 選擇 node.js
  - ignore launch.json
- You can debug by F10

https://code.visualstudio.com/docs/nodejs/working-with-javascript



JavaScript Hello world! 最快速的方式

JavaScript 語法

- Case-sensitive（區分大小寫）
- 使用 Unicode 編碼
- 每行指令被稱為 Statements，並用分號（;）分隔
- 空格、Tab 與換行符號皆被視為空白
- 文件會從左到右進行掃描，並轉換成一系列的元素

註解（Comments）

```
// a one line comment

/* this is a longer,
   multi-line comment
 */

/* You can't, however, /* nest comments */ SyntaxError
*/
```

註解語法跟 C++ 和其他語言相同

JavaScript Data Types

- JavaScript Strings
- JavaScript Numbers
- JavaScript Booleans
- JavaScript Arrays
- JavaScript Objects
- typeof Operator
- **Undefined**

會常常看到

JavaScript typeof

```
console.log(typeof 37) //'number'
console.log(typeof NaN) //'number'
console.log(typeof '') //'string'
console.log(typeof (typeof 1)) //'string'
console.log(typeof true) //'boolean'
console.log(typeof null) //'object'
console.log(typeof function(){}) //'function'
```

宣告（Declarations）

- 三種宣告方式

`var`

宣告一個可隨意更改其內容的變數

`let`

宣告一個可隨意更改其內容的區塊區域變數

`const`

宣告一個只可讀取的不可變常數

var

宣告一個可隨意更改其內容的變數

```
var length = 16;                                    // Number
var lastName = "Johnson";                           // String
var x = {firstName:"John", lastName:"Doe"};         // Object
```

Use quotes inside a string

```
var answer1 = "It's alright";          // Single quote inside double quotes
var answer2 = "He is called 'Johnny'";  // Single quotes inside double quotes
var answer3 = 'He is called "Johnny"';  // Double quotes inside single quotes
```

```
var a = "A";
var b = a;

// Equivalent to:

var a, b = a = "A";
```

不要這樣寫

```
var x = y, y = 'A';
console.log(x + y); // undefinedA
```

宣告（Declarations）

let

宣告一個可隨意更改其內容的區塊區域變數

```
var x = 'global';
let y = 'global';
console.log(this.x); // "global"
console.log(this.y); // undefined
```

`let` 並不會在全域物件中建立變數

let

JavaScript是一個鬆散資料類型(dynamically typed)的程式語言

```
let foo = 42     // foo現在是Number資料類型
let foo = 'bar' // foo現在是String資料類型
let foo = true // foo現在是Boolean資料類型
```

let

```
if (true) {
  var x = 5;
}
console.log(x);  // x is 5
```

```
if (true) {
  let y = 5;
}
console.log(y);  // ReferenceError: y is not defined
```

let

```
function varTest() {
  var x = 1;
  {
    var x = 2;
    console.log(x);  // 2
  }
  console.log(x);  // 2
}
```

```
function letTest() {
  let x = 1;
  {
    let x = 2;
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

const

宣告一個只可讀取的不可變常數

```
// 定義一個常數 MY_FAV 並賦予它的值為7
const MY_FAV = 7;

// 這裡會發生錯誤 - Uncaught TypeError: Assignment to constant variable.
MY_FAV = 20;

// MY_FAV 是 7
console.log('我喜歡的數字是: ' + MY_FAV);
```

JavaScript 字串串接

```
//使用concat()串接
const aString = 'JavaScript'
const bString = aString.concat(' is a', ' script language')
console.log(bString)

//使用(+=)串接
let cString = 'JavaScript'
cString += ' is a'
cString += ' script language'
console.log(cString)
```

JavaScript 字串長度

```
const aString = 'Hello World!'
const bString = '你好'
const aStringLength = aString.length //12
const bStringLength = bString.length //2
```

JavaScript 子字串搜尋

```
const aString = 'Apple Mongo Banana'


console.log( aString.indexOf('Apple') ) // 0
console.log( aString.indexOf('Mongo') ) // 6
console.log( aString.indexOf('Banana') ) // 12
console.log( aString.indexOf('Honey') ) // -1
```

JavaScript
adding a number and a string

```
var x = 16 + "csie@cgu";
```

JavaScript
adding a number and a string

```
var x = "csie@cgu" + 16;
```

JavaScript
adding a number and a string

```
var x = 16 + 4 + "csie@cgu";
```

JavaScript
adding a number and a string

```
var x = “csie@cgu” + 16 + 4 ;
```

JavaScript
adding a number and a string

var x = 16 + "csie@cgu";	16csie@cgu
var x = "csie@cgu" + 16;	csie@cgu16
var x = 16 + 4 + "csie@cgu";	20csie@cgu
var x = “csie@cgu” + 16 + 4 ;	csie@cgu164

JavaScript 跳脫字元(Escape characters)

```
const aString = 'It\'s ok'
const bString = 'This is a blackslash \\'
```

JavaScript String 嵌入變數/常數

```
const firstName = 'Eddy'
console.log(`Hello ${firstName}!
Do you want some
rabbits tonight?`)



const x = 5
console.log(`5 + 3= ${x + 3}`)
```

JavaScript Boolean

```
const a = true
const b = false

console.log(typeof a) //boolean
console.log(1=='1') //true
console.log(typeof (1=='1')) //boolean
console.log(b!=a) //true
```

JavaScript 比較運算

- 值的比較(==)是相等
- 值與類型比較(===)

```
'foo' === 'foo' // true
123 === 123 // true
123 === '123' // false
```

```
null  == undefined // true
null === undefined // false
```

JavaScript 比較運算

```
let a = {}
let b = {}


a === b // false


let c = {}
let d = c


c === d // true
```

物件類型的變數儲存的其實是
「記憶體位置」，
彼此在比較相等性時，
需要指向同一個物件才會得到 `true`

const 浮點數

```
const aNumber = parseFloat("10")  //10
const bNumber = parseFloat("10.00") //10
const cNumber = parseFloat("10.33") //10.33
const dNumber = parseFloat("34 45 66") //34
const eNumber = parseFloat("40 years") //40
const fNumber = parseFloat("He was 40")  //NaN
```

宣告（Declarations）

*如果未指定數值給該變數，那麼該變數的值會是 `undefined`

<u>*函數(function) 中 var 陳述式應該盡可能地置放在接近函數(function)的頂部</u>

一行宣告
一個變數或常數

```
//不好的宣告方式
const items = getItems(),
    goSportsTeam = true,
    dragonball = 'z';

//好的宣告方式
const items = getItems()
const goSportsTeam = true
const dragonball = 'z'
```

Declarations 變數(函式、類別)命名

camelCase

```
let numberOfStudents
const numberOfLegs
function setBackgroundColor()
class Student{}
```

CamelCase

```
const NAMES_LIKE_THIS='Hello'
```

JavaScript Arrays

```
var arr = new Array(element0, element1, ..., elementN);
var arr = Array(element0, element1, ..., elementN);
var arr = [element0, element1, ..., elementN];
```

```
let arr = ["apple", "banana", "orange", "guava", "papaya"];
```

```
let arr = new Array("apple", "banana", "orange", "guava", "papaya");
```

```
let arr = new Array(5);
```

JavaScript Arrays Methods

- **toString**() converts an array to a string of (comma separated) array values.
- **join**() method also joins all array elements into a string
  - join(“, “)
- **pop**() method removes the last element from an array
- **push**() method adds a new element to an array (at the end)

- Array indexes start with **0**. [0] is the first array element, [1] is the second, [2] is the third

JavaScript Arrays 陣列走訪

```
let arr = ["apple", "banana", "orange", "guava", "papaya"];

arr.forEach(function (elem) {
    console.log(elem);
});
```

```
let arr = ["apple", "banana", "orange", "guava", "papaya"];

for (let i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

JavaScript Arrays Changing Elements

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";          // Changes the first element of fruits to "Kiwi"
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";          // Appends "Kiwi" to fruits
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];          // Changes the first element in fruits to undefined
```

JavaScript 字串與字元

```
const a = 'cat'.charAt(1)   //  'a'
const b = 'cat'[1]   // 'a'


console.log(typeof a) //string
```

JavaScript Object

- JacaScript裡頭的"物件"可以與真實生活中的物件做類比
- objectName.propertyName

```
var myCar = new Object();
myCar.make = 'Ford';
myCar.model = 'Mustang';
myCar.year = 1969;
```

```
myCar.color; // undefined
```

```
myCar['make'] = 'Ford';
myCar['model'] = 'Mustang';
myCar['year'] = 1969;
```

```
var propertyName = 'make';
myCar[propertyName] = 'Ford';

propertyName = 'model';
myCar[propertyName] = 'Mustang';
```

JavaScript Object

```
var myHonda = {
  color: 'red',
  wheels: 4,
  engine: {
   cylinders: 4,
   size: 2.2}
  };
```

JavaScript Object create method

```
// Animal properties and method encapsulation
var Animal = {
  type: 'Invertebrates',
  // Default value of properties
  displayType: function() {
    // Method which will display type of Animal
    console.log(this.type);
  }
};

// Create new animal type called animal1
var animal1 = Object.create(Animal);
animal1.displayType(); // Output:Invertebrates
```

JavaScript 運算子

- `+`：相加
- `-`：相減
- `*`：相乘
- `/`：相除
- `%`：取餘數
- `+`：取正號
- `-`：取負號
- `++`：遞增 (加 1)
- `--`：遞減(減 1)

- `===`：嚴格相等
- `!==`：嚴格不等
- `==`：相等
- `!=`：不等
- `>`：大於
- `>=`：大於等於
- `<`：小於
- `<=`：小於等於

- `&&`：且 (and)
- `||`：或 (or)
- `!`：否 (not)

邏輯運算子

&&	expr1 && expr2	Logical AND，如果 expr1 和 expr2 都是 true，就會傳回 true， 否則傳回 false
||	expr1 || expr2	Logical OR，如果 expr1 或 expr2 是 true，就會傳回 true，否則傳回 false
!	!expr	Logical NOT，如果 expr 是 true，就傳回 false，否則傳回 true

JavaScript 運算子

三元運算子

(condition) ? .. : ..

bitwise and &

bitwise or |

bitwise xor ^

bitwise not ~

left shift <<

right shift >>

zero-fill right shift >>>

+=

-=

*=

/=

%=

<<=

>>=

>>>=

&=

|=

^=

Assignment

Operators

位元運算子

&	15 & 9	1111 & 1001 = 1001	9	Bitwise AND 運算，如果兩個位元都是 1，結果就是 1，否則是 0
|	15 | 9	1111 | 1001 = 1111	15	Bitwise OR 運算，如果任何一個位元是 1，結果就是 1，否則是 0
^	15 ^ 9	1111 ^ 1001 = 0110	6	Bitwise XOR 運算，如果位元不相同，結果是 1，否則是 0
~	~15	~00000000...00001111 = 11111111...11110000	-16	Bitwise NOT 運算，將所有位元的 0 變成 1，1 變成 0

邏輯運算子

```
// foo 是 Dog
var foo = 'Cat' && 'Dog';

// foo 是 false
// 因為 && 遇到 false 的運算元，就會直接返回，不會繼續再往下判斷 (Short-circuit evaluation)
var foo = false && 'Cat';

// foo 是 Cat
// 因為 || 遇到 true 的運算元，就會直接返回，不會繼續再往下判斷 (Short-circuit evaluation)
var foo = 'Cat' || 'Dog';

// foo 是 Cat
var foo = false || 'Cat';
```

位元運算子

<<	9 << 2 = 36	左移運算 (Left shift)，將所有位元向左移 n 個位置，右邊的位元補入 0
>>	9 >> 2 = 2	右移運算 (Sign-propagating right shift)，將所有位元向右移 n 個位置，最左邊的位元 (sign bit) 補入跟原本最左位元一樣值，保持正負數一致
>>>	19 >>> 2 = 4	補零右移 (Zero-Fill Right Shift)，跟 >> 一樣，但最左邊的位元補 0

三元運算子

```
condition ? val1 : val2
```

如果 condition 是 true， 就傳回 val1 的結果，否則傳回 val2 的結果。例如：

```
// 如果 age 變數大於等於 18，則 status 就會是 adult
// 相反的，如果 age 變數小於 18，則 status 就會是 minor
var status = (age >= 18) ? 'adult' : 'minor';
```

**=	x **= y	意思跟 x = x ** y 一樣，將 x 的 y 次方值指定回 x 變數
<<=	x <<= y	意思跟 x = x << y 一樣，將 x 所有位元左移 y 位，右邊的位元補入 0 後的值指定回 x 變數
>>=	x >>= y	意思跟 x = x >> y 一樣，將 x 所有位元右移 y 位，最左邊的位元 (sign bit) 補入跟原本最左位元一樣值後指定回 x 變數
>>>=	x >>>= y	跟 >>= 一樣，但最左邊的位元補 0
&=	x &= y	意思跟 x = x & y 一樣，將 x y 做位元 AND 運算後的值指定回 x 變數
^=	x ^= y	意思跟 x = x ^ y 一樣，將 x y 做位元 XOR 運算後的值指定回 x 變數
|=	x |= y	意思跟 x = x | y 一樣，將 x y 做位元 OR 運算後的值指定回 x 變數

%	12 % 5	模數運算子 (Remainder)，以某運算式的值除以另一個運算式的值，並傳回餘數。12 % 5 等於 2
++	++10 ++x x++	遞增運算子 (Increment)，每次將變數的值加一。如果運算子在變數之前，則會在執行運算式之前修改值。如果運算子在變數之後，則會在執行運算式之後修改值。 例 1: ++10 等於 11 例 2: j = ++k 則 j 的值是 k 的原始值加一 例 3: j = k++ 則 j 的值是 k 的原始值，k 會在其值指派給 j 之後遞增

--	--10 --x x--	遞減運算子 (Decrement)，每次將變數的值減一。如果運算子在變數之前，則會在執行運算式之前修改值。如果運算子在變數之後，則會在執行運算式之後修改值。 例 1: --10 等於 9 例 2: j = --k 則 j 的值是 k 的原始值減一 例 3: j = k-- 則 j 的值是 k 的原始值，k 會在其值指派給 j 之後遞減

&	15 & 9	1111 & 1001 = 1001	9	Bitwise AND 運算，如果兩個位元都是 1，結果就是 1，否則是 0
|	15 | 9	1111 | 1001 = 1111	15	Bitwise OR 運算，如果任何一個位元是 1，結果就是 1，否則是 0
^	15 ^ 9	1111 ^ 1001 = 0110	6	Bitwise XOR 運算，如果位元不相同，結果是 1，否則是 0
~	~15	~00000000...00001111 = 11111111...11110000	-16	Bitwise NOT 運算，將所有位元的 0 變成 1，1 變成 0

&&	expr1 && expr2	Logical AND，如果 expr1 和 expr2 都是 true，就會傳回 true， 否則傳回 false
||	expr1 || expr2	Logical OR，如果 expr1 或 expr2 是 true，就會傳回 true，否則傳回 false
!	!expr	Logical NOT，如果 expr 是 true，就傳回 false，否則傳回 true

```
// foo 是 Dog
var foo = 'Cat' && 'Dog';

// foo 是 false
// 因為 && 遇到 false 的運算元，就會直接返回，不會繼續再往下判斷 (Short-circuit evaluation)
var foo = false && 'Cat';

// foo 是 Cat
// 因為 || 遇到 true 的運算元，就會直接返回，不會繼續再往下判斷 (Short-circuit evaluation)
var foo = 'Cat' || 'Dog';

// foo 是 Cat
var foo = false || 'Cat';
```

JavaScript 運算子優先權

member	. []
call / create instance	() new
negation/increment	! ~ - + ++ -- typeof void delete
multiply/divide	* / %
addition/subtraction	+ -
bitwise shift	<< >> >>>
relational	< <= > >= in instanceof
equality	== != === !==
bitwise-and	&

JavaScript 運算子優先權

bitwise-xor	^
bitwise-or	|
logical-and	&&
logical-or	||
conditional	?:
assignment	+= -= *= /= %= <<= >>= >>>= &= ^=
comma	,

JavaScript 流程控制 (Control flow)

- if...else
- switch
- for
- while
- label
- try catch finally

The else if Statement

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

switch 語句

```
switch (fruitType) {
    case 'Oranges':
        alert('Oranges');
        break;
    case 'Apples':
        alert('Apples');
        break;
    case 'Bananas':
        alert('Bananas');
        break;
    default:
        alert('沒有符合的條件');
}
```

for 迴圈語法

```
var i = 0;
var counter = 0;
for (;;) {
    counter += i;
    i += 1;
    if (i >= 5) {
        break;
    }
}
```

```
var counter = 0;
for (var i = 0;i < 5; i++) {
    if (i < 3) {
        continue;
    }
    counter += i
}
```

while 迴圈語法

```
var n = 0;
var x = 0;
while (n < 3) {
    n++;
    x += n;
}
```

```
var i = 10;
do {
    i++;
} while (i < 5)
```

label (標籤)

```
var x = 0;
var z = 0;
// 把外層的迴圈標記叫做 labelCancelLoops
labelCancelLoops:
while (true) {
    console.log('Outer loops: ' + x);
    x += 1;
    z = 1;
    while (true) {
        console.log('Inner loops: ' + z);
        z += 1;
        if (z === 3 && x === 3) {
            // 跳出 labelCancelLoops 迴圈
            break labelCancelLoops;
        } else if (z === 3) {
            // 跳出當前迴圈
            break;
        }
    }
}
```

label (標籤)

```
var i, j;

// 把外層的迴圈標記叫做 loop1
loop1:
for (i = 0; i < 3; i++) {
    // 把內層的迴圈標做 loop2
    loop2:
    for (j = 0; j < 3; j++) {
        if (i === 1 && j === 1) {
            // 跳出 loop1 迴圈
            continue loop1;
        }
        console.log('i = ' + i + ', j = ' + j);
    }
}
```

Function

```
function getScore () {
    // 局部變數 - function scope
    // 作用範圍只在函數內部
    var num1 = 2;
    var num3 = 4;
    // 如果沒加 var 宣告變數，這個變數則是一個全域變數
    num2 = 5; // 存取到全域變數 num2
    num4 = 6; // 宣告一個新的全域變數 num4
    // 函數也可以宣告在其他函數內部 (nested function) - function scope
    function add() {
        // 內部函數可以存取到外部函數的局部變數
        return name + ' scored ' + (num1 + num2 + num3);
    }
    return add();
}
```

JavaScript 允許巢狀函式（nesting of functions）

Function

```
var square = function(number) {
    return number * number;
};
```

Function 不確定數量的參數

```
function fun1(...theArgs) {
  console.log(theArgs.length);
}

fun1();  // 0
fun1(5); // 1
fun1(5, 6, 7); // 3
```

Return multiple values

```
function getValues() {
    return {
        first: getFirstValue(),
        second: getSecondValue(),
    };
}

var values = getValues();
var first = values.first;
var second = values.second;
```

Function Closure

```
const counter = (function() {
    let i = 1

    return function() {
        console.log(i++)
    }
}())

counter() //1
counter() //2
```

靜態變數

function私有變數

Function Closure

```
var pet = function(name) {
      var getName = function() {
         return name;
      }
      return getName;
};

myPet = pet("Vivie");
myPet(); // Returns "Vivie"
```

Function Closure

```
function add() {
  var counter = 0;
  function plus() {counter += 1;}
  plus();
  return counter;
}

console.log(add()); // 1
console.log(add()); // 1
console.log(add()); // 1
```

Function Closure

```
var add = (function () {
  var counter = 0;
  return function () {
    counter += 1;
    return counter;
  }
})();

console.log(add()); // 1
console.log(add()); // 2
console.log(add()); // 3
```

```
function add() {
  var counter = 0;
  function plus() {
    counter += 1;
  }
  plus();
  return counter;
}

console.log(add()); // 1
console.log(add()); // 1
console.log(add()); // 1
```

```
var add = (function () {
  var counter = 0;
  return function () {
    counter += 1;
    return counter;
  }
})();

console.log(add()); // 1
console.log(add()); // 2
console.log(add()); // 3
```

private 變數

About function

把大象放進冰箱

About function

把大象放進冰箱

1. 打開冰箱
2. 放進大象
3. 關閉冰箱

1.1 拉手把

1.2 轉冰箱門

1.3 開冰箱燈

About function

抽象化的架構

具體化的coding

First Law of Software Quality

$$e = mc^2$$

$$\text{errors} = (\text{more code})^2$$

Inside a function

不要超過100行

Inside a function

不要超過100行

Inside a function

不要超過100行

Function

“超過100行者”

分數 * 80%

Function

“超過100行者”

分數 * 80%

Function

“超過100行者”

分數 * 80%

變數命名原則

```
let numberOfStudents

const numberOfLegs

function setBackgroundColor()

class Student{}
```

變數命名原則

```
let numberOfStudents

const numberOfLegs

function setBackgroundColor()

class Student{}
```

變數命名原則

```
let numberOfStudents

const numberOfLegs

function setBackgroundColor()

class Student{}
```

Error Handling (例外處理)

```
try {
    blah('Hello world');

} catch(err) {
    alert(err.name + ': ' + err.message);

} finally {
    alert('try catch 區塊結束');
}
```

Error Handling (例外處理)

```
try {
    throw 'myException';
} catch (err) {
    // err 是字串 "myException"
    err;
}
try {
    throw 101;
} catch (err) {
    // err 是數字 101
    err;
}
```

Error Handling (例外處理) Error 物件

```
try {
    throw new Error('oops');
} catch (err) {
    // 輸出 "Error: oops"
    console.log(err.name + ': ' + err.message);
}
```

JSON: JavaScript Object Notation

- Sending Data
- Receiving Data
- Storing Data

JSON: JavaScript Object Notation

- Sending Data
- Receiving Data
- Storing Data

Example

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

Evaluates to JavaScript Objects

JSON

```
{ "name":"John" }
```

JavaScript

```
{ name:"John" }
```

JSON: Valid Data Types

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

JSON values **cannot** be one of the following data types:

- a function
- a date
- *undefined*

JSON.stringify()

```
'{ "name":"John", "age":30, "city":"New York"}'
```

```
var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');
```

JSON.parse()

```
var obj = { name: "John", age: 30, city: "New York" };
var myJSON = JSON.stringify(obj);
```

{"name":"John","age":30,"city":"New York"}

Arrays in JSON Objects

Example

```
{
"name":"John",
"age":30,
"cars":[ "Ford", "BMW", "Fiat" ]
}
```

Conclusion

- Dev Environment & Debug
- Data Types
- Variable
- Operator
- Control Structure
- Array
- JSON

Function

“超過100行者”

分數 * 80%

Function

“超過100行者”

分數 * 80%

Function

“超過100行者”

分數 * 80%

Thanks!

Open for any questions

CJ Wu

cjwu@mail.cgu.edu.tw

GO!

練習時間！！

```
"    *"
"   ***"
"  *****"
"*******"
"    *"
"    *"
"    *"
```

jsfiddle.net

```
"    *"
"   ***"
"  *****"
"*******"
"    *"
"    *"
"    *"
```

```
"    *"
"   ***"
"  *****"
"*******"
"    *"
"    *"
"    *"
```

```
"    *"
"   ***"
"  *****"
"*******"
"    *"
"    *"
"    *"
```

```
"    *"
"   ***"
"  *****"
"*******"
"    *"
"    *"
"    *"
```