JavaScript @ AstroCamp

有趣的本質

Recap

- 三種基本型別： `123`, `"abc"`, `false`
- 分支： `if...else`, `switch...case`
- 迴圈： `for...of` 及 `for i`
- 兩種集合： Array `[]` 及 Object `{}`

再多熟悉一下集合的語法

單層 Array

```
let ary = [1, 2, 3, 4, "test", true]
```

Array 的取值與設值

```
array[0];

let i = 1;
array[i];

let j = 2;
array[i] = j;

let number = array.pop();
array.push(100);
```

單層 Object

```
let hsh = {
 key: "my value",
 anotherKey: 100,
 "special key here": ture
}
```

Object 的取值與設值

```
hsh.key


let k = 'anotherKey';
hsh[k];


Object.keys(hsh);
Object.values(hsh);
Object.entries(hsh);
```

多加一層: 含有 Array 的 Object

```
let user = {
  name: "John",
  favorite: ["Music", "Travel", "Alcohol"]
}
```

再多一層呢？

```
let users = [
  {
    name: "John",
    favorite: [
      "Music",
      "Travel",
      "Alcohol"
    ]
  },
  {name: "Amy", favorite: ["kpop", "cuisine"]},
  {name: "T", favorite: ["math", "coffee"]},
]
```

其它集合類型

- Set
- Weak Map

https://caniuse.com/

萬事萬物的設計指南 - 1

最少意外原則

Part 1: 變數作用域 variable scope

作用域：什麼地方存取的到這個東西

先來看兩個正常的例子好了

```
var x = 1;

console.log(x);
```

```
function foo() {
  var x = 1;
}
foo();
console.log(x);
```

沒處理好的話，會有許多麻煩的情況

```
var x = 1;

function foo() {
  console.log(x);
}
```

shadowing

```
var x = 1;

function foo() {
  var x = 100;
  console.log(x);
}

console.log(x);
```

WTF moment: 不加 var 會怎樣？

```
function foo() {
  x = 200
}

console.log(x)
```

這次有呼叫了

```
function foo() {
  x = 200
}
foo()
console.log(x)
```

一定要記得的事:

ES6 之前，JavaScript 只有全域變數以及函式內變數兩種 scope。

函式內如果沒有遮蔽變數名稱，可以存取到外層的變數。

一定要記得的事:

不加 `var` 的話，會到處亂噴。

N個會印什麼出來的小測驗

```
console.log(x);
```

```
var x = 1;

console.log(x);
```

for 迴圈呢？

```
for (var i = 0; i < 10; i ++) {
}

console.log(i);
```

ES6 的 let 與 const：block scope variable

```
for (let i = 0; i < 10; i ++) {
}

console.log(i)
```

» 區塊作用域

變數放在後面讀得到嗎？

```
console.log(x);
```

```
var x = 1;
```

其實會變成

```
var x;

console.log(x)

x = 1;
```

這個概念叫 hoisting

let 變數放在後面讀得到嗎？

```
console.log(i)

let i = 1
```

如果是函式呢？

```
f()

function f() {
  console.log('hello!')
}
```

具名函式會連 body 一起 hoisting

寫程式的好習慣：變數宣告的地方愈接近使用的地方愈好

Ruby

```ruby
def foo do
  # do something
  # many
  # lines
  a = 100
  a += 200
end
```

JavaScript 會習慣把 var 宣告都放在該 scope 的最上面

```
var a = 100, b, c;
// do something
b = 200

function bar() {
  var x, y, z;
  // do something

  x = 10;
}
```

如果用了 let 就不需要這麼做

```
function baz() {
  // do something

  let x = 10
  callOtherFunction(x)
}
```

萬事萬物的設計指南 - 2

Something is elegant if it is two things at once: unusually simple and surprisingly powerful.” – Matthew E. May

兼備異常簡單，意外強大者，可謂之優雅。

Part 2: 函式是一等公民

Function as first class citizen

回想一下我們可以拿三種基本型別加上兩種集合型別做什麼？

1. 指派給變數

```
let a = 1
```

2. 當做集合的值

```
['a', 'b']
```

3. 當做呼叫函式用的參數

```
Math.max(1, 5, 2)
```

4. 當做函式的回傳值

```
function foo() {
  return {a: 1, b: 2}
}
```

那函式呢？

Narrtive:

在 Java、Ruby 這些 OO 語言中，函式被稱做方法，是物件的附庸

The kingdom of Object

在 Ruby 裡，方法不帶括號也是呼叫

```ruby
# Ruby
def f
  puts '1'
end
f # 這個是呼叫

#### 換到這個好處
response.should respond_with_content_type(:json)
```

在 JavaScript 裡，函式是一等公民，可以獨立存在。

```
// JavaScript
function foo() {}

console.log(foo)
```

1. 把函式指派給變數 -> 匿名函式

```
let addTwenty = function(x) {
  return x + 20
}

addTwenty(1)
```

匿名函式沒有 name、而且本體不會 hoisting

```
addTwenty(1)
```

```
let addTwenty = function(x) {
  return x + 20
}
```

2. 把函式當做集合的值

```
let Math = {
  abs: function(i) {
    return i > 0 ? i : -i
  }
}

Math.abs(-100)
```

3. 把函式當做(另一個函式的)參數

高階函式: 接收函式的函式

map、reduce、filter

console.log 太長？

```
console.log(console.log)
console.log(console)
let p = console.log
```

你的語言可以做這件事嗎？

```
function cook(ig, sauce, cookMethod) {
  cookMethod(ig)
  cookMethod(sauce)
  console.log(`${sauce} ${ig} is ready`)
}

cook('chicken', 'coconut', boomboom)
```

Bonus: 什麼是 pure function

作業

1. 金魚都能懂的網頁設計入門： jQuery 相關 (26, 27, 28)
2. 看一下 JavaScript 裡，Array.prototype 的函式中，有哪些是高階函式
3. RNA transscription

```
'G' -> 'C'
'C' -> 'G'
'A' -> 'U'
'T' -> 'A'
'ACGTGGTCTTAA' -> 'UGCACCAGAAUU'

Rna.transcript('GC') // => 'CG'
```

Note: JavaScript 的字串是個很像陣列，但不是陣列的東西。

進階：終止子

轉出以下三個就結束轉譯，並輸出含有終止子以前的結果

UAG、UAA、UGA

```
Rna.transcript('ACGTATTCCCCC') // => 'UGCAUAA'
```

Happy hacking!