#2019.7.2 開始一更 參考codecademy
#2019.7.6 完成基礎的一更，剩下進階課程待補
#2019.7.15 開始二更 參考modern JS tutorial
更新前言
主要希望更新part 2 應用web 部分，但希望讀完part1 順便補充一些點(part 1 參考javascipt_inroduce)
#2019.7.20 為防止篇幅過長，將Part II 另開一篇 JS 網頁，算中斷二更
同時二更目標希望讀完part 1 或是 advance topic

目錄

前言:

.JS 簡介:

*JS 一開始就只是為了"網頁端"為目的的script 語言，所以本質上很"安全"(不會碰記憶體等核心)
，JS 遵從ECMAScript 標準

*browser 內建javascript 的虛擬引擎

- V8 – in Chrome and Opera.
- SpiderMonkey – in Firefox.
- ...There are other codenames like "Trident" and "Chakra" for different versions of IE, "ChakraCore" for Microsoft Edge, "Nitro" and "SquirrelFish" for Safari, etc.

*引擎大概的工作
1.parse the script

2.convert("compile") it to machine code
3.run the machine code
(optimalize)

*in-browser JS "can" do
與使用者、伺服端互動溝通
比如:
*改動HTML tag、CSS style
*回應使用者 如:滑鼠點擊
*要求伺服端下載資料
*取得、設定 cookie，依使用者決定呈現的data
*local (client) storage
on the other hand
JS 因為 **安全** 而必須限制其功能
比如
*不得操作OS 等核心
*除非使用者同意，不得使用硬體設備
*Same origin policy (同源政策) 限制不同web 間正常來說要獨立 (資安保護)
*只能從一個server 接收資料 (否則須權限)

*JS 的特別之處
1.與HTML、CSS 整合
2.簡單
3.被大多數瀏覽器支持

*Javescript 的延伸
針對每個人不同的需求，JS 也應運而生不同變體
如CoffeeScript、Dart、Flow、Typescript....

*因預設bug 不會給web user 看，所以必須使用工具
F12 ==> element 如果是看HTML & CSS
F12 ==> console、source 就是看JS 的
*console:
可在> 後輸入JS code，Enter 執行、Shift + Enter 換行

A.Js 基礎
1.Console,a keyword,refer to a object
keyword:能被程式辨認且有特殊意義的字
object:資料與行為的集合
*console.log 不會顯示給user
*alert() 、comfirm(...)、prompt()則會跳出視窗顯示訊息

console.log(5); 印出5 ;做斷句
console.log('this is a book'); 印出string

2.註解方式:just like C++
(//、/*...*/)

3.Data Type:
primitive data
-1.數字 (include 整數、小數、Infinity、NaN)
-2.字串(' ' or " ")
-3.布林□
-4.NULL : null, 代表無□
-5.undefined :類似null
-6.Symbol (variable)
complex *data*
-7.object (data & action)

*Data type convertion
由於JS 是動態弱型語言, 有很多隱轉, 比如
字串串接
數學加法 "6"/"3" =2
....
*如果數字轉型失敗會出現NaN

4.基本運算元

1. Add: `+`
2. Subtract: `-`
3. Multiply: `*`
4. Divide: `/`
5. Remainder: `%`

5.字串串接:
‘A’ + ‘B’= ‘AB’

6.性質(重要):
所有 data 都是 data type 的實例(instance)，因此都擁有 Data type 的 property
例如所有字串都有 length 的性質
'hello'.length 就是 5

7.method:
類似性質，data type 可使用的函式
console.log()的 log()就是如此

```
console.log('hello'.toUpperCase()); // Prints 'HELLO'
console.log('Hey'.startsWith('H')); // Prints true
```

8.內建 object
善用此網站□ 詢所要函式
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects
使用 Math 物件中的 random()

```
console.log(Math.random()); // Prints a random number between 0 and 1
```

取 0-50

```
Math.floor(Math.random() * 50);
```

注意

```
Number.isInteger(2017) //vaild
2017.isInteger()  //invalid
```

9.coding skill in JS
(1)建議加;
(2)最好採用先主 code 後 help function 的方式 (讓人不一定需要讀 function code)
(3)好的 funciton、variable 名字可以代替註解
(4)善用 continue、return 避免過多層的 nest
(5)comment 只用在重要解法、high level view 、function 用法

B.JS 變數:
(a)概述
變數是儲存資料於記憶體中

變數三操作:創造、儲□、取□
變數=/=數□ :變數只是代表一個□ (box and things)
變數優點:易讀、少錯

(b)Create variables:
recall:JS 是直譯、動態(runtime 才檢□)、弱型別(隱轉)語言
var、let、const 來創造變數
1.var 例子:(ES6 以前的版本)

```
var myName = 'Arya';
console.log(myName);
```

//變數採 camel casing (thisIsAnApple)
2.let 例子:
用法跟 var 一樣

*注意:若只宣告變數而不初始化, JS 會視為 undefine

3.const 例子:
同 C++中的常數, 不可以改□ ,□ 初始化未賦□ 視為錯誤 !

(c)變數運算
1.和 C++類似有+=.. ++...等等操作
2.變數也能字串串接
3.字串插□ (重要)
template literals: 用``(反斜線刮住的字串)
可以使用${變數}來代□
eg:

```
const myPet = 'armadillo';
console.log(`I own a pet ${myPet}.`);
```

優點:易讀
4. typeof variable :回傳 variable 型態
eg:
let my = 65
typeof my retrun number

補充:ES6
https://zh.wikipedia.org/wiki/ECMAScript
Script language:
Scripting language 是為了縮短傳統的「編寫、編譯、連結、執行」(edit-compile-link-run)過程而建立的
https://zh.wikipedia.org/wiki/%E8%84%9A%E6%9C%AC%E8%AF%AD%E8%A8%80
Js 1995 年被創造用於寫網頁

隔年被改良為scripting language
ECMA-262 規範了JS 的語法規則
ES6 是整個規範最大的改動，例如加入OOP 的概念

C.條件判斷
(a)跟C++類似(if-else if -else)

```
if (true) {
  console.log('This message will print!');
}
// Prints "This message will print!"
```

(b)比較運算子

Less than: `<`

Greater than: `>`

Less than or equal to: `<=`

Greater than or equal to: `>=`

Is equal to: `===`

Is NOT equal to: `!==`

*note:字串採用字典比較
*null、undefined(NaN)在大小於比較一定會return false，故應特例處理

(c)邏輯運算子

the *and* operator (`&&`)

the *or* operator (`||`)

the *not* operator, otherwise known as the *bang* operator (`!`)

(d)資料型態真假
以下恆假

`0`

Empty strings like `""` or `''`

`null` which represent when there is no value at all

`undefined` which represent when a declared variable lacks a value

`NaN`, or Not a Number

除此之外都為真

*補充:優雅程式寫法

```
let defaultName;
if (username) {
  defaultName = username;
} else {
  defaultName = 'Stranger';
}
```

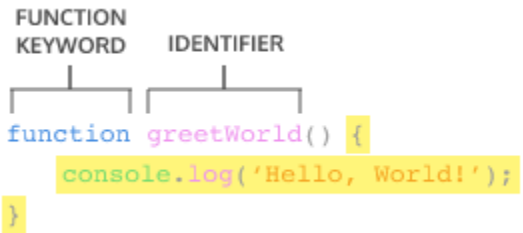
確認username 是否存在

```
let defaultName = username || 'Stranger';
```

Because `||` or statements check the left-hand condition first, the variable `defaultName` will be assigned the actual value of `username` if is truthy, and it will be assigned the value of `'Stranger'` if `username` is falsy. This concept is also referred to as *short-circuit evaluation*.

也有 ?: 三元運算子、switch

D.函式
函式是一段可重複利用的程式碼
(a) .宣告



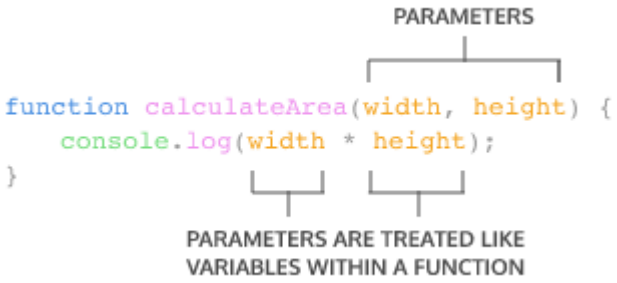
*注意function 關鍵字

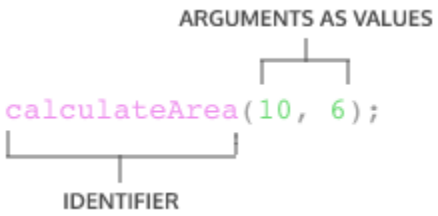
*如沒有return □□ □　return undefine

*Hoisting(提升) : JS 可以允許先使用再定義函式!

```
console.log(greetWorld()); // Output: Hello, World!

function greetWorld() {
  console.log('Hello, World!');
}
```

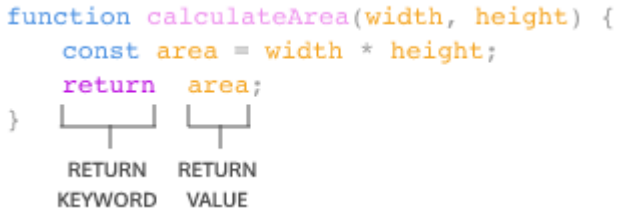
(b)呼叫函式
函式只有在被呼叫(called)時才會執行其內容
方式與C++同
*note : func()是執行函數 func 本身則是"□", 可以指派給變數
所以let exe=func
exe() // run the same thing with func()
(c) parameter & argument
傳入參數:argument(傳給參數時使用)
使用參數:parameter(定義函式時使用)





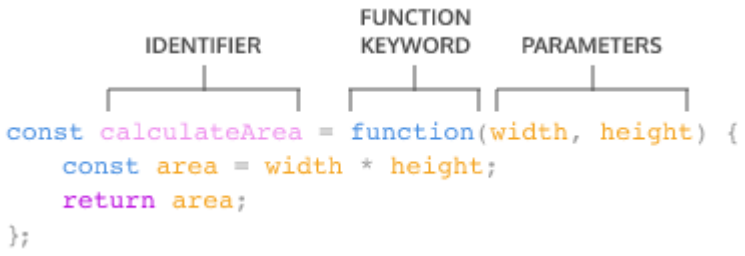
注意, JS 不需要寫parameter 的type(recall:weak type)
*Default parameter:和C++一樣可以使用default argument
(當函式沒有收到該argument 或是argument 為undefine)

(d)回傳□
未回傳即為undefine



(e) funcion expression :*anonymous function* 的概念

1.可省略funcion name




```
variableName(argument1, argument2)
```

(通常會使用const 宣告匿名函數的變數)

(使用方式相當於把變數當成函數，在JS 中函數也是□)

(省略函數名)

*匿名函數使用例子:

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
  "Do you agree?",
  function() { alert("You agreed."); },
  function() { alert("You canceled the execution."); }
);
```

*function expression VS function declaration

(1).語法上不同

(2).exp. 通常使用一次性 (使用時機不同)

(3) declaration 可以在定義前使用 (Hoisting)

(4) 在嚴格模式下 (user strict) declaration 只能在block 內可見

(比如想在if 內宣告且在外層可用，就要用exp.)

使用大原則: 盡量使用宣告 (易讀、處處可用)，特殊原因才使用exp.

(條件式宣告函數、不希望Hoisting....)

2.可以再簡化(arrow function)

連function 都不用寫

```
const rectangleArea = (width, height) => {
  let area = width * height;
  return area;
};
```

3.更多語法糖(concise (簡潔)
單一變數省略()

ZERO PARAMETERS

```
const functionName = () => {};
```

ONE PARAMETER

```
const functionName = paramOne => {};
```

單一行省略{} 與return

SINGLE-LINE BLOCK

```
amOne, paramTwo) => {};
const sumNumbers = number => number + number;
```

MULTI-LINE BLOCK

```
const sumNumbers = number => {
    const sum = number + number;
    return sum;
};
```

RETURN STATEMENT

E Scope:
基本同C++
global:不在任何一個block 內，整個檔案都能使用此區域的變數
block:在某一對 {} 中，裡面宣告的變數只有此block 能使用，使用未宣告的變數會有 `ReferenceError` (而非undefine!)
Scope pollution:過多變數在全域，可能會引發未定義行為(unexpected behavior)
Scope 好處:安全、易讀、好維護、省記憶體

Scope is the idea in programming that some variables are accessible/inaccessible from other parts of the program.

Blocks are statements that exist within curly braces `{}`.

Global scope refers to the context within which variables are accessible to every part of the program.

Global variables are variables that exist within global scope.

Block scope refers to the context within which variables that are accessible only within the block they are defined.

Local variables are variables that exist within block scope.

Global namespace is the space in our code that contains globally scoped information.

Scope pollution is when too many variables exist in a namespace or variable names are reused.

F.array & loop:
(a).Create
array 可以用
let myArray=['string',100,true]去創造，同時因為JS 是weak type，array 可存任意東西!
(可想像array 就是一堆變數)
(array 名代表整個array!(不是位址))
(b).access element
myarray[n],第n+1 個element (從0 開始)
(字串一樣可以用此方式取字元)
(取用超出arrray bound 是undefine!!!)
當然也能update 元素

*注意:用const 宣告的array 還是能修改array element，但不行重新assign 該array

eg:
const myArray=[1,2,3]
myArray[0]=2 //OK
myArray=2 //invalid!
(但是myArray 是用let 就沒問題，記住JS 是weak type)

(c)array 操作
length 可以看有多少元素 (remember 'string'.length)
*小技巧: A[A.length-1] 就能取到最後一個元素
push() 可以在array 最後面放入元素
pop()可以取出最後面的元素 (刪除兼回傳)
更多可參考 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
shift() 刪除第一個元素
unshift() 在array 前面放入元素

slice()

```
var animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));
// expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// expected output: Array ["camel", "duck"]

console.log(animals.slice(1, 5));
// expected output: Array ["bison", "camel", "duck", "elephant"]
```

indexOf()

```
const pastaIndex=groceryList.indexOf('pasta');
```

*Array 是*pass-by-reference* 在函式內更改也會影響外面的array

(d). 巢狀array (array 中當然也可以有array)

```
const nestedArr = [[1], [2, 3]];

console.log(nestedArr[1]); // Output: [2, 3]
console.log(nestedArr[1][0]); // Output: 2
```

*Loop
(a)for 與C++同
*提醒可以使用 for(let i=0;i<myArray.length;i++) 去traversal 所有元素in myArray
(b)while 與C++ 同, 也有do...while

G.High Order function & Iterator
(A).High Order function
(a) 簡介
High Order function==>將funciton 作為I/O 的function (function as data!)
Recall:函式表達式
我們將函式參考於變數中
在JS 中, 函式是first object(有property and method)
參考:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function
例如.name 可知其原本函式名

*注意 const temp=myFunciotn(); //是錯誤的, 變成接收myFunciotn 的return
const temp=myFunciotn; // 才是正確的
*注意 temp.name 而不是 temp().name
(b) function as parameter

Recall:weak type，所以函數參數也只要一variable 就□ 了

```
const timeFuncRuntime = funcParameter => {
   let t1 = Date.now();
   funcParameter();
   let t2 = Date.now();
   return t2 - t1;
}

const addOneToOne = () => 1 + 1;

timeFuncRuntime(addOneToOne);
```

```
timeFuncRuntime(() => {
  for (let i = 10; i>0; i--){
    console.log(i);
  }
});
```

*也可以接受匿名函數

(B)Iterator

Iterators are methods called on arrays to manipulate elements and return values (使之比loop 方便)

`.forEach()`

`.map()`

`.filter()`

(a) forEach()

`.forEach()` 可以使每一個element 做一樣的Code

return □□□　　undefine

```
groceries.forEach(groceryItem => console.log(groceryItem));
```

(匿名函數 input=>function body)

```
function printGrocery(element){
  console.log(element);
}

groceries.forEach(printGrocery);
```

forEach() 可以把每個element 做為input

*再次注意 function 是input 所以沒有()　(function name)
　　例如: [1,2].forEach(alert);
(b) map()
會將element 做為input 輸入函式，並回傳新Array!

```
const secretMessage=animals.map(element=>element[0]);
```

*注意:一樣要自己寫匿名函數的input (是給匿名函數看的)
*index 也算function 喔
(C).filter() 類似haskell
可將為真者篩選出來(透過驗真假函式)，　一樣回傳新陣列

```
const words = ['chair', 'music', 'pillow', 'brick', 'pen', 'door'];

const shortWords = words.filter(word => {
  return word.length < 6;
});
```

(d)findIndex()
一樣接收驗真假函式，然後吐出第一個為真者
(e) reduce()

The .reduce() Method

Another widely used iteration method is `.reduce()`. The `.reduce()` method returns a single value after iterating through the elements of an array, thereby *reducing* the array. Take a look at the example below:

```
const numbers = [1, 2, 4, 10];

const summedNums = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue
})

console.log(summedNums) // Output: 17
```

Here are the values of `accumulator` and `currentValue` as we iterate through the `numbers` array:

Iteration	accumulator	currentValue	return value
First	1	2	3
Second	3	4	7
Third	7	10	17

The `.reduce()` method can also take an optional second parameter to set an initial value for `accumulator` (remember, the first argument is the callback function!). For instance:

```
const numbers = [1, 2, 4, 10];

const summedNums = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue
}, 100)  // <- Second argument for .reduce()

console.log(summedNums); // Output: 117
```

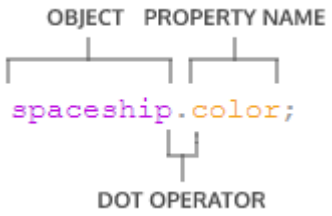
*記住arrow function 格式為
()=>{}　　等價於 f(){}
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Iteration_methods
(f) every() and some()
確認所有 或 存在 為真的元素(透過驗真假函式)

補充:
*注意 forEach 與 map 都是作用事情於陣列
但map 會回傳新陣列
*注意 string 也有 split(‘ ‘)等函式
*includes() 會檢□ 是否有特定元素

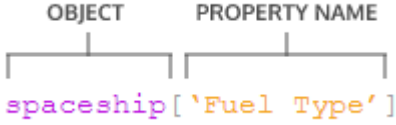
H.Object (JSON 相關)
Object 會收集相關Data 與function，就像字典一樣
(a)建立javascript object
let variable = {}
object 是用key-value pair 的形式製造

```
let spaceship = {

    'Fuel Type': 'diesel',
    color: 'silver'

};
```

*注意，key 一定是字串，但JS 允許omit(省略) ‘ ‘
*注意，鍵□ 中間是用　　: 隔開(同python 字典)
*注意，分隔是使用逗號，而且結束不用加分號
*注意，鍵就像變數名，而□ 一樣可以是任意型態　　(even 陣列、函式)
(b)物件取用(access)
法1 dot operator
*注意，取用未定義的物件變數視為undefined



法2 Bracket Notation

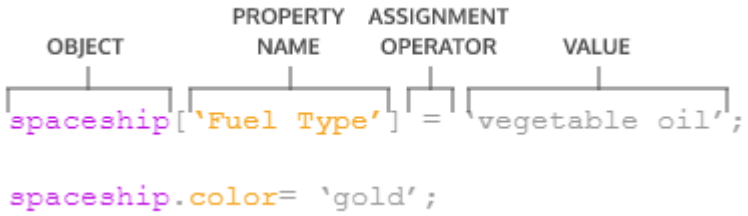


與python 類似
*注意，當鍵□ 有奇怪符號　　(如空白隔開)，應使用此

```
let returnAnyProp = (objectName, propName) => objectName[propName];

returnAnyProp(spaceship, 'homePlanet'); // Returns 'Earth'
```

If we tried to write our `returnAnyProp()` function with dot notation (`objectName.propName`) the computer would look for a key of `'propName'` on our object and not the value of the `propName` parameter.

結論:使用時沒有特殊情況用dot (dot 後面只允許接合法變數)
編寫時使用[ ] ([]中間放入key 的字串)
(c)修改object property



*注意:如果property 原本不存在object 中, 則會創造新的property
*注意:可以使用delete 去刪除property
例如 delete spaceship.color;

*補充 :[]法
我們可以使用 [變數名] = XXX 的方式增加object 的property
先看個例子:

```
let fruit = prompt("Which fruit to buy?", "apple");
let bag = {};

// take property name from the fruit variable
bag[fruit] = 5;
```

可知道變數fruit 由使用者輸入, 然後成為key 加入bag 的property 中
fruit 可以是任意字串

*可以用 "in" 檢□ 是否存在某關鍵字,□ 比如 "apple" in bag (回傳 true or false)

(d)放入method
可使用
key: function(//parameter){//function body}
或省略為(省略: 與function)
key (//parameter) {//function body}
(e)其他主題
(1) nested object(重要, 也就是常見的JSON 形式)
也就是object 中有object, 也是使用上最常見的方式
當然取用時就要層層取用(注意選用. Or [])
(2)pass by "address": (物件名類似指標)
傳給函式時也會修改到原本的object 的內容!
但同樣的只是只到那個位置, 函式無法修改原本的object 本身
(3) loop in object (like python)

```
for (var property1 in object1) {
  string1 += object1[property1];
}
```

*注:也可使用let

*注意:此法中property1 只是object1 中的第一層key name!!!
所以property1.innerproperty 是錯誤的寫法
應objext1['property1'].innerproperty

(f) advanced topic

how to use the `this` keyword.

conveying privacy in JavaScript methods.

defining getters and setters in objects.

creating factory functions.

using destructuring techniques.

這裡挑重點紀錄
(1)this:用於使用需要用到object 本身的property
例如: print:(){console.log(this.paint)} //: 而非= !!!
print:(){console.log(paint)} //invaild:what is paint?
print:(){console.log(obj.paint)} //invalid:what is obj? (obj 還未製造就使用內容物!)
*注意arrow function ()=>{} 不可使用this !
(2)privacy
JS 沒有privacy 的機制, 習慣上會在privacy 變數前加上_

例如 _myVariable

(3)setter & getter

*注意:getter 跟setter 可以同名, 但不能跟variable 同名!

getter: *使用get keyword

*條件式取得property (封裝!)

*使用時不用加() !(不是省略, 是不能加!)

```
  get fullName() {
    if (this._firstName && this._lastName){
      return `${this._firstName} ${this._lastName}`;
    } else {
      return 'Missing a first name or a last name.';
    }
  }
}

// To call the getter method:
person.fullName; // 'John Doe'
```

setter:類似setter, 但理所當然有input *輸入input 可以直接用等號

```
person.age = 40;
```

```
if(typeof num ==='number')
```

//檢□ 型別

(4) factory function

用於製造object 實例的function

```
const monsterFactory = (name, age, energySource, catchPhrase) => {
  return {
    name: name,
    age: age,
    energySource: energySource,
    scare() {
      console.log(catchPhrase);
    }
  }
};
```

整理:

一個arrow function

回傳object

```
const monsterFactory = (name, age) => {
  return {
    name,
    age
  }
};
```

ES6 語法糖(省略重複打□)

(5)destructured assignment

(類似Haskell list 的映對

[a,b]=[10,5]
則a==10,b==5)　　==>能□ 只取結構中特定部分
回到object
const a=object.property 可省略為
const {property} = object
(用於Nest object 可增加易讀性)
build-in method in object:https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object#Methods
舉例
Object.keys(myObj);　//找出所有key name
//recall: Object 是JS 八大型別之一(也是物件)
Object.entries(myObj); //回傳鍵-□ array
Object.assign(T,S) ; //將obj S {和T 一樣的property=> 取代;比T 多的property => 新增}
*注意:不要慣性思維
Object.assign(T,S) 可以放變數
當然也可以直接放obj
Object.assign(T,{'a':2})
newObj=Object.assign({'a':2},S) 也行(用在不想動S!!!)

***專案作業**

```
let dish={
  name:courseName,
  price:dishPrice
}
this._course[courseName].push(dish)//push 就被array複製一份了
```

*注意1:不用swith(courseName).....去幹
*注意2:同註解，雖然dish 這個local 變數最後會消失，但push 會複製一份了
不熟的點
一，object 的格式，中間是:，然後,常常忘記加
二，語法糖型式的函式寫法還是會怪怪的
三，this 使用時機還不熟
四，會被nested object 搞混

心得
JS object 真的像電話簿一樣，如果要大量產生相同的要用factory function 製造，而不是像C++先定義再產生實例!(JS 寫好就是存在一個變數中了(實例))

(更新，也可以定義class，所以有點像可自由產生object(也是一type))

補充:
*global object
global object 可以在任何地方使用，通常被建在environment 或程式語言本身，不同環境有不同名稱(browser 是window、Nodejs 是global)
global object 的property 可以直接取用
如alert()等價於window.alert()

*專案object 分析
有一個/menu/ 主object
內有一/套餐/object 用三個陣列(陣列紀錄/餐點/ objext)變數紀錄三種餐(前、主、甜點)
然後主object 有加入套餐的函式
餐點有價錢與名稱，然後輸入餐的類型決定放入的陣列
主object 有隨機點餐的函式
*使用技巧(取0-陣列長的數)

```
const randomIndex = Math.floor(Math.random() * dishes.length);
```

即可回傳被選中的餐
然後再用一函式接收隨機存取的三種餐，算價錢，回傳字串即可!

I.class
(a)簡介:
JS 是OOP 語言
class =>用於快速製造object，object 的模板
以下將介紹
*constructor*繼承(inheritance) *static method
Outline: class VS object

```
class Dog {
  constructor(name) {
    this._name = name;
    this._behavior = 0;
  }

  get name() {
    return this._name;
  }
  get behavior() {
    return this._behavior;
  }

  incrementBehavior() {
    this._behavior ++;
  }
}
```

```
let halley = {
  _name: 'Halley',
  _behavior: 0,

  get name() {
    return this._name;
  },

  get behavior() {
    return this._behavior;
  },

  incrementBehavior() {
    this._behavior++;
  }
}
```

(b)Constructor:直接用此名做函式

```
constructor(name) {
  this.name = name;
  this.behavior = 0;
}
```

*注意:由於object 可直接用assign 製造新property, 不需要在class 中先宣告property(ex:name 與behavior)

(c) create instance

(1)可以這樣理解instance:與class 有一樣property 與method 的object 但每個instance 都有不同的value

(圖章與圖案!)

(2)語法:

```
const halley = new Dog('Halley');
```

*重點一:使用new 關鍵字 *重點二:直接用名字當作constructor

(d)method

加入method 的方式和object 類似
惟不能用逗號隔開!

```
Class dog{
…
get age(){
    return this.age;
}

bark(){
    return “wo”;
}
…
}
```

```
let myDog={
    …
get age(){
    return this.age;
}

bark(){
    return “wo”;
}
...
}
```

使用method 的方式和object 就一模一樣了
(recall:函式用.取，getter 不用加())

(e)inheritance
基本概念:為了節省程式碼，需要`繼承`，parent class(super class) & child class(subclass)

語法

```
class Cat extends Animal {
  constructor(name, usesLitter) {
    super(name);
    this._usesLitter = usesLitter;
  }
}
```

*重點一:使用extend 關鍵字
*重點二:使用super 呼叫parent class 的建構子
*重點三:可自由增加額外的property 於下面
*重點四:繼承後可以使用parent 的函式與porperty
*解惑:為甚麼要用getter 的原因之一:
除了篩選後再輸出
還有雖然名字可以一樣(getter 不用加())但無法修該

(f)static method

```
static generateName() {
  const names = ['Angel', 'Spike', 'Buffy', 'Willow', 'Tara'];
  const randomNumber = Math.floor(Math.random()*5);
  return names[randomNumber];
}
```

共用的函式，只能className.function()，不能objectName.function()
例如animal.cry() //OK
 const dog= new animal()
        dog.cry()   //Nope TypeError
這也是為甚麼有Math.random()   Math.floor()
C++也是如此
注意分別靜態函數與靜態變數

PART II Advance topic

A.try...catch
(a)基本:
類似C++格式

```
try {

  // code...

} catch (err) {

  // error handling

}
```

*執行try 內的code，如果沒error 會無視catch 以下
*err 是一個error object
*try..catch 只能處理run-time error，也無法處理非同步的code
(setTimeout、setInterval)

(b)Error Object
有幾個poroperty
name:錯誤名稱
message:錯誤訊息　　stack:可視為詳細錯誤訊息

(c)throw
就像C++，也有throw 的功能，但是throw 只能丟error object

```
throw new SyntaxError("Incomplete data: no name"); // (*)
```

(d)
finally 有點像整合最後的結果如果有錯誤try-catch-finally
如果沒有錯誤try-finally

```
try {
   ... try to execute the code ...
} catch(e) {
   ... handle errors ...
} finally {
   ... execute always ...
}
```