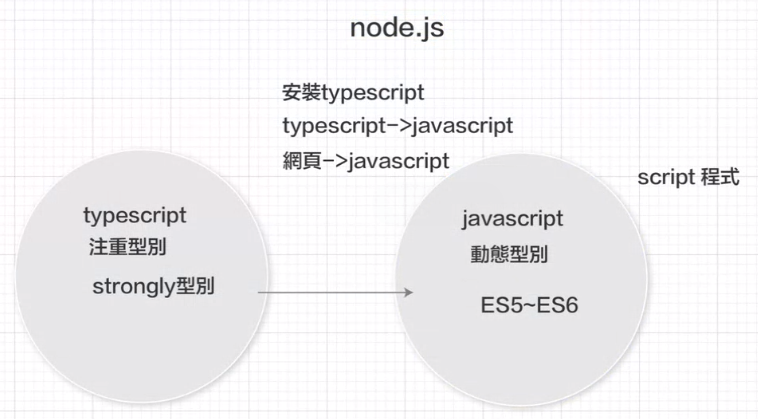
# Javascript



2023-05-08

## 1.安裝 node.js

<https://nodejs.org/zh-tw/download>

node --version 查詢是否安裝成功



npm 套件管理程式

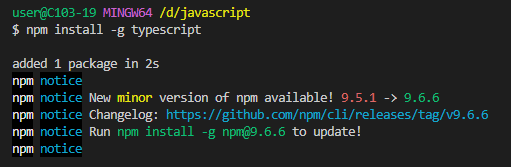
npm --version



## 2.透過npm安裝typescript

https://www.npmjs.com/package/typescript

npm install -g typescript



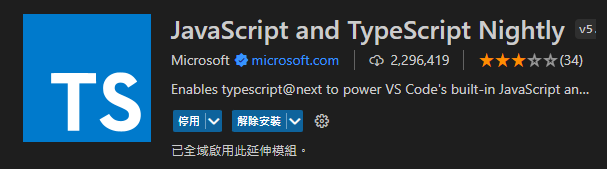
安裝完會有 tsc 命令=> typescript 指令

查詢版本

tsc --version



VS安裝套件 TypeScript



## 設定 tsconfig.json

tsc --init

產生 tsconfig.json

修改 ts 及輸出目錄

  "rootDir": "./c01/ts",

  "outDir": "./c01/js",

存檔後, 執行 tsc -w 偵測並自動產生 js

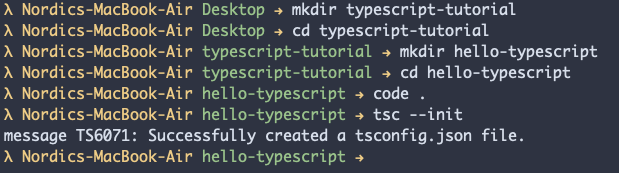
tsc -w

首先，沒有安裝過 TypeScript 的話，必須先安裝它的指令工具，記得用 -g，因為我們要讓該指令工具可以在任何地方使用喔：

$ npm install -g typescript

如果已經下載完成的話，我們可以下達：

$ tsc --init

  
圖八：終端機下達基本指令

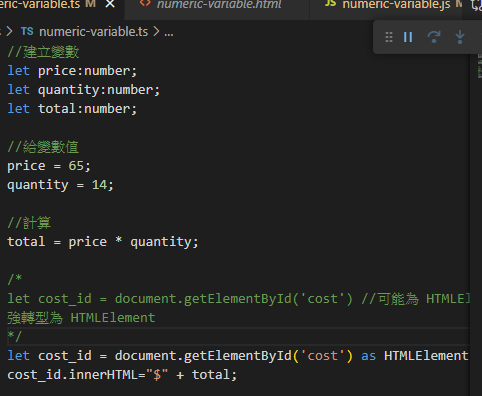
此時你的專案會出現一個名為 tsconfig.json 的檔案，這就是 TypeScript 編譯器的設定檔

js 會出現 使用嚴格形別

"use strict";

**DOM** 全名為 **Document Object Model** 中文翻譯為 **文件物件模型**，看起來很抽象但其實就是把一份 **HTML** 文件內的各個標籤，包括文字、圖片等等都定義成物件，而這些物件最終會形成一個[樹狀結構](https://zh.wikipedia.org/wiki/%E6%A8%B9%E7%8B%80%E7%B5%90%E6%A7%8B)，下面有一張示意圖可以參考。  


document.getnElementById(‘idname’)



let user\_name:string;

let message:string;

user\_name="Maple"

message="hello"

let user\_span = document.getElementById('name') as HTMLElement;

let note\_span = document.getElementById('note') as HTMLElement;

/\*

 innerHTML 是 W3C 規定的標準寫法，

 而 innerText 則是除了可以用來取得 HTML 元素之外，

   還會把元素的 HTML 標籤去除掉，

   但 innerText 並非 W3C 所規定的標準寫法，

   而且僅適用於 IE 瀏覽器

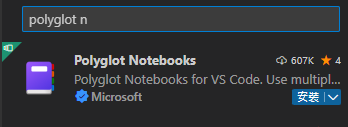
 \*/

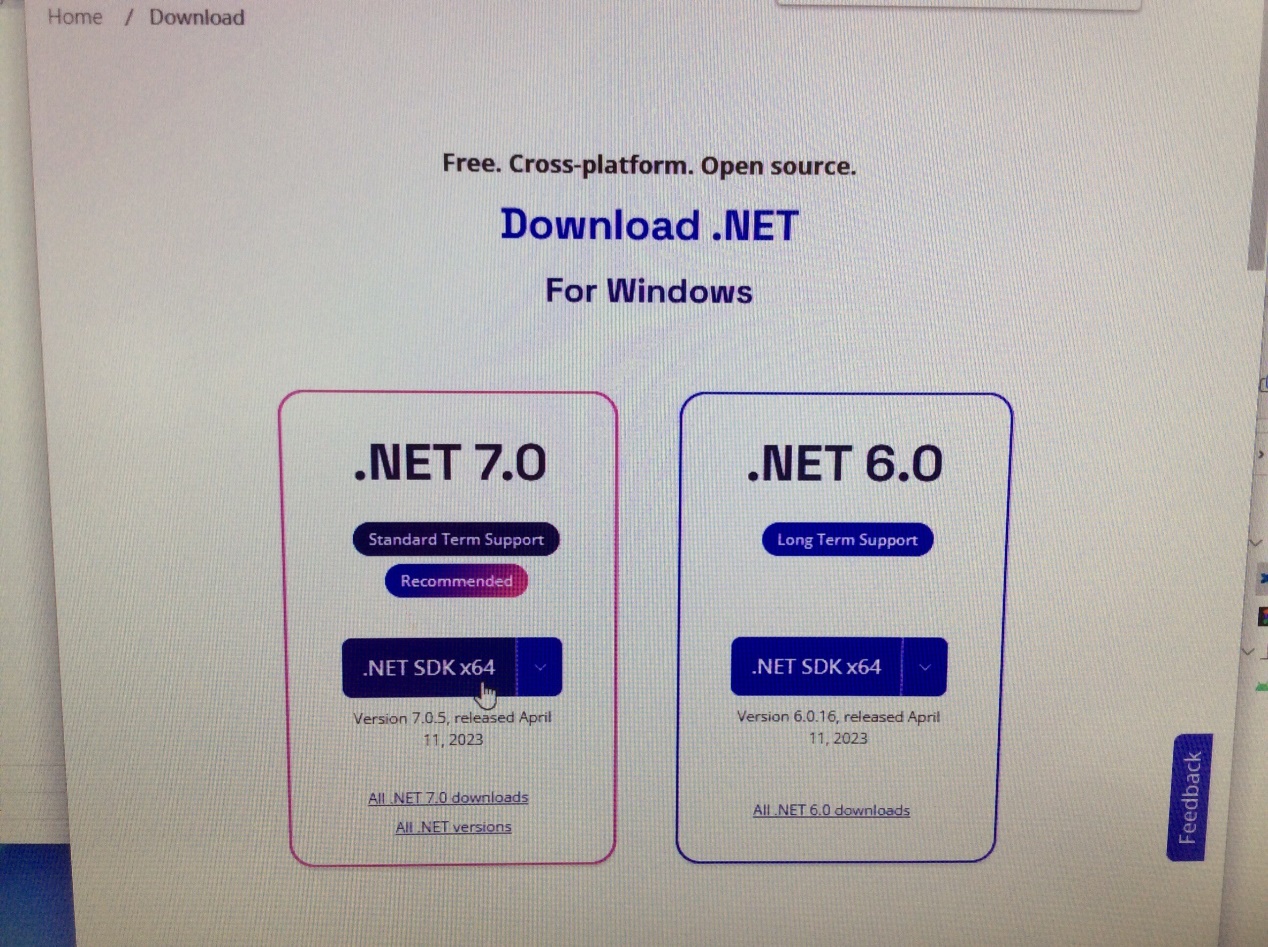
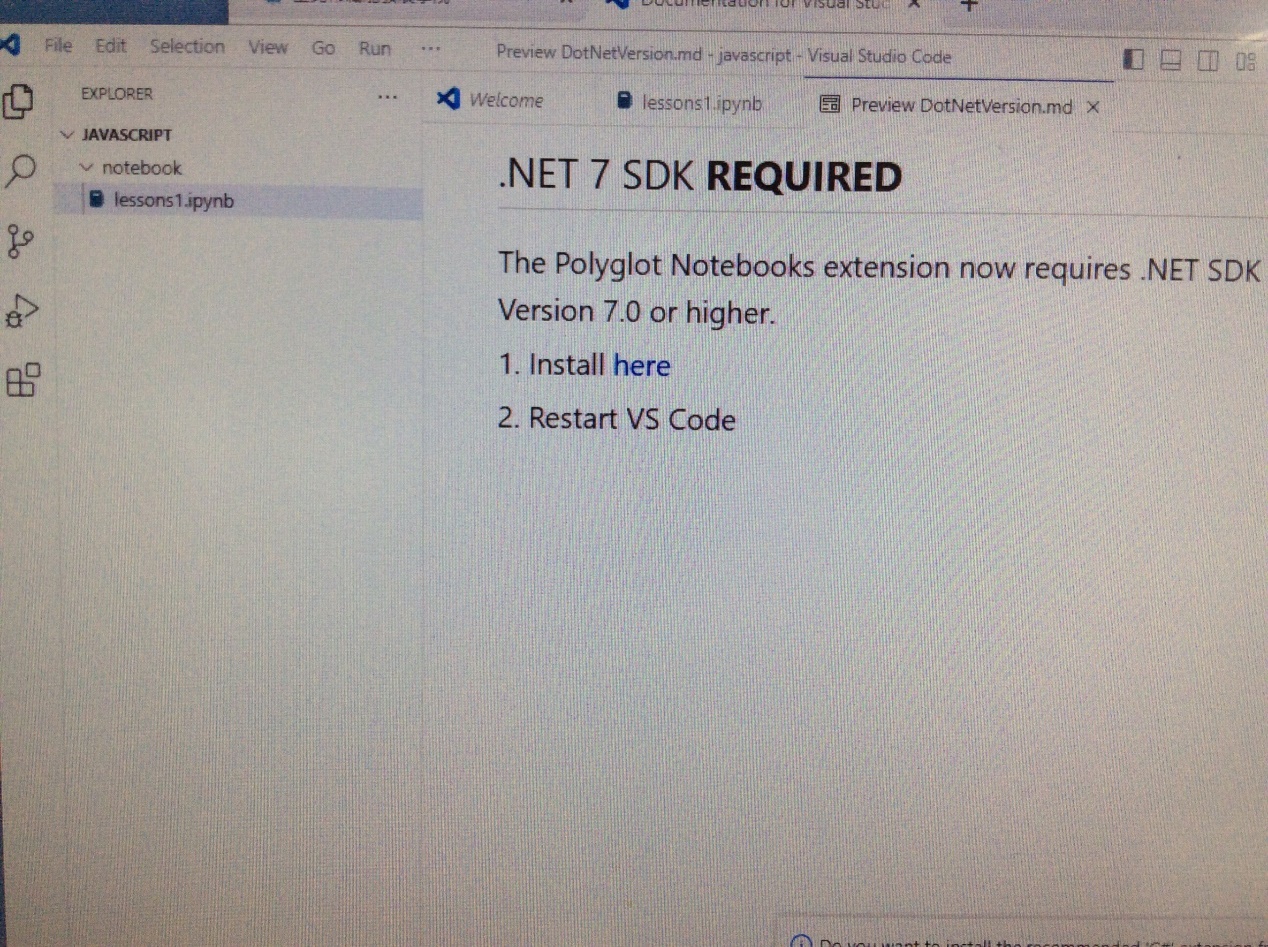
user\_span.innerHTML = user\_name;

note\_span.innerText = message;

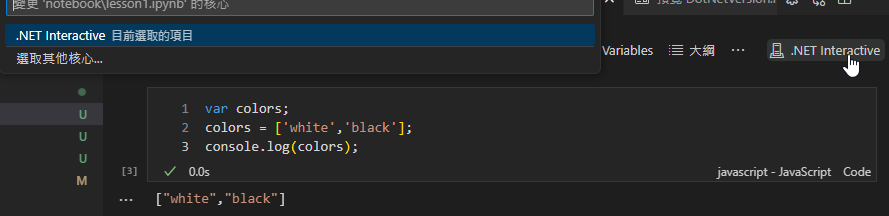


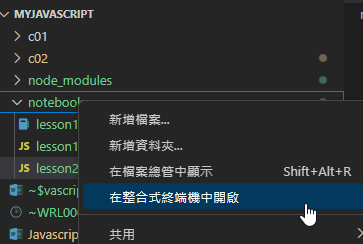
vsc安裝 Polyglot Notebooks





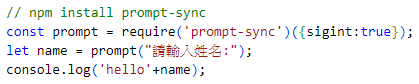
下載安裝後, 重啟vsc







3種javascript 開發環境

1. 網頁環境
2. node.js環境 : 測試輸入 require prompt
3. 筆記本環境 (.ipynb) : 測試

2023-05-09

## 數學運算

console.log(Math.round(.6))//4捨5入

console.log(Math.floor(3.9)) //無條件捨去

console.log(Math.ceil(.95)) //四捨五入並返回大於或等於給定數字的較小整數

console.log(Math.pow(2,2))  //次方

console.log(Math.log(10))  //對數

console.log(Math.pow(Math.E,2.302585092994046))

console.log(Math.sqrt(100)) //平方根

console.log(Math.trunc(-10.123)); //刪除小數

console.log('取亂數:'+Math.floor(Math.random() \* 100));//取亂數

console.log(Math.max(1, 3, 2))//取最大值

console.log(Math.abs(-22)) //取絕對值

//建立array

console.log([1,2,3,4,5])

//建立二維array

console.log([1,2,5],[3,4],[5,6])

let a=[[1,2,5],[3,4],[5,6]]

console.log(a[2][1])

//object

//物件

let p = {x:2.3, y:-1.2,abc:333}

console.log(p['x'])

console.log(p.x)

console.log(p['y'])

console.log(p.y)

console.log(p['abc'])

//object 用大括號建立

var book={

    'main title':"javascript",

    'sub-title':"The Definitive Guide",

    for:"all audiences",

    author:{

        firstname:"Davie",

        surname:"Flanagan"

    }

}

console.log(book['main title'],book['sub-title'])

console.log(book.for,book.author.firstname)

//Object 用new方法建立

var book = new Object()

book['main title'] = "javascript"

book.author = new Object()

book.author.firstname="David"

book.author.Lastname="Bob"

console.log(book['main title'],book.author.firstname,book.author.Lastname)

console.log(book)

結果

javascript

David

Bob

{"main title":"javascript","author":{"firstname":"David","Lastname":"Bob"}}

## 動態取得瀏覽器的寬及高

//windows 代表整個瀏覽器視窗

//Document 代表物件框

//window.document.body.innerHTML = "hello world";

//{$window.innerHeight} 瀏覽器inner高度

//{$window.innerWidth} 瀏覽器inner寬度

let msg = `<h2>browser window</h2>`;

msg += `<p>brower\_width: ${window.innerWidth}</p>`;

msg += `<p>brower\_height: ${window.innerHeight}</p>`;

msg += `<p>使用者的螢幕寬度: ${window.screen.width}</p>`;

msg += `<p>使用者的螢幕高度: ${window.screen.height}</p>`;

let info\_element = document.getElementById('info');

info\_element.innerHTML = msg;

msg += `<p>brower\_width: ${window.innerWidth}</p>`;

msg += `<p>brower\_height: ${window.innerHeight}</p>`;

let info\_element = document.getElementById('info');

info\_element.innerHTML = msg;

## 跳出自定視窗說明

html

<dialog id="favDialog">

      <p>Greetings, one and all!</p>

      <form method="dialog">

        <button>OK</button>

      </form>

</dialog>

js

let favDialog\_element=document.getElementById('favDialog');

favDialog\_element.showModal();

//favDialog\_element.show();

## 新增querySelector 與 forEach 及 for迴圈

// dinamic insert li

let ul = document.querySelector('ul');

let li = document.createElement('li');

li.innerText = 'new li';

li.className='cool';

ul.appendChild(li);

let first\_hot\_element = document.querySelector('li.hot'); //只會抓第一個

first\_hot\_element.className = 'cool';

//一次取出多個

let all\_elements = document.querySelectorAll('li.hot');

console.log(all\_elements.length);

//querySelectorAll foreach

## /\* forEach 方法一

all\_elements.forEach(function(element){

    element.className = 'complete';

});

\*/

/\*forEach 方法二

all\_elements.forEach(

    // 箭頭表 anonymous function

    (currentElement) => {

        currentElement.className = 'complete';

    });

\*/

## //for 方法

for (let i = 0; i < all\_elements.length; i++) {

    all\_elements[i].className = 'complete';

}

## //匿名function 寫法

1.

function(element) {

        element.className = 'complete';

    }

2.

 currentElement => {

        currentElement.className = 'complete';

    }

## 取得鄰居的元件(Sibling)兄弟姐妹

let two\_element = document.getElementById('two');

// 取得鄰居的元件

var previous\_element=two\_element.previousElementSibling;

var next\_element=two\_element.nextElementSibling;

previous\_element.className='complete';

next\_element.className='cool';

let ul\_element=document.getElementsByTagName('ul')[0];

// get ul first child li element

let firstchild = ul\_element.firstElementChild;

let lastchlid = ul\_element.lastElementChild;

firstchild.className='cool';

lastchlid.className='complete';

/\*

ul\_element.firstElementChild.className='cool';

ul\_element.lastElementChild.className='complete';

\*/

20230515

## 抓取節點

<html>

  <head>

    <title>JavaScript &amp; jQuery - Chapter 5: Document Object Model - Node Value</title>

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="css/c05.css">

  </head>

  <body>

    <div id="page">

      <h1 id="header">List</h1>

      <h2>Buy groceries</h2>

      <ul>

        <li id="one" class="hot"><em>fresh</em> figs</li>

        <li id="two" class="hot">pine nuts</li>

        <li id="three" class="hot">honey</li>

        <li id="four">balsamic vinegar</li>

      </ul>

    </div>

    <script src="js/node-value.js"></script>

  </body>

</html>

js

let elementTwo = document.getElementById('two');

let nodeTwo = elementTwo.firstChild;

console.log(nodeTwo.nodeValue);

console.log(nodeTwo.nextSibling);

console.log(nodeTwo.nextSibling.firstChild.nodeName);

console.log(nodeTwo.nextSibling.firstChild.nodeType);

console.log(nodeTwo.nextSibling.firstChild.nodeValue);

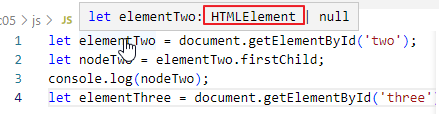
console.log(elementTwo.innerHTML);

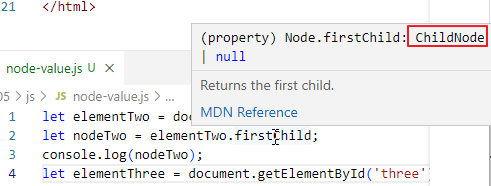
console.log(elementTwo.innerText);

nodeTwo.nodeValue=nodeTwo.nodeValue.replace('pine nuts','aaaa');

log顯示結果







## 取得node的文字內容

let one=document.getElementById("one");

console.log(one.textContent)

let firstchild=one.firstChild;

let nest=firstchild.nextSibling;

console.log(firstchild.firstChild.nodeValue);

console.log(nest.nodeValue);

## innerText 與 textContent 的差別

節錄 MDN 對於兩個 API 的描述：

*Node.innerText 是一個代表節點及其後代之「已渲染」（rendered）文字內容的屬性。*

*Node.textContent 屬性表示了節點或其後代的文字內容。*

已渲染指的是什麼？簡單地說，innerText 回傳「實際所見的內容」，我們針對 h1 加入一點點的 CSS：

h1 {  
 text-transform: uppercase;  
}

在網頁中我們會看到被 CSS 改為大寫的 HELLO WORLD，接著到終端機：



從這邊就能很清楚地了解，innerText 取得的是被 CSS 調整過樣式後渲染的文字；textContent 則是實際取得節點中的文字內容。

再看一個例子，拿掉 CSS，這次我們在 h1 中加入一個 span 元素，並將其設定為 display:none：

<h1>Hello world<span style="display:none;">!!!</span></h1>

終端機：



由於 span 在畫面上並不會出現，innerText 自然不會取到它的值。另外，當我們使用到例如換行的 br 元素時：

<h1>Hello world, <br/>I love Kanao</h1>

innerText 回傳的會是換行的結果；textContent 則會忽略掉它。

let newLiElement = document.createElement('li');

//newLiElement.textContent = '新項目';

let newTextNode = document.createTextNode('新項目');

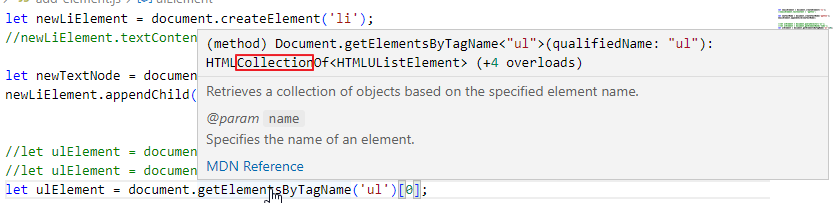
newLiElement.appendChild(newTextNode);

//let ulElement = document.querySelector('ul');

//let ulElement = document.getElementById('todo');

let ulElement = document.getElementsByTagName('ul')[0];

ulElement.appendChild(newLiElement);



## removeChild

let removeElement=document.getElementsByTagName('li')[3];

console.log(removeElement);

let ulelement = document.getElementsByTagName('ul')[0];

ulelement.removeChild(removeElement);

## setAttribute 設定屬性

let oneElement = document.getElementById('one');

oneElement.className = 'complete';

//var fourthElement = document.getElementsByTagName('li')[3];

var fourthElement = document.getElementsByTagName('li').item(3);

fourthElement.setAttribute('class', 'cool');

## removeAttribute 移除屬性

let oneElement = document.getElementById('one')

if (oneElement.hasAttribute('class')){

    oneElement.removeAttribute('class')

}

/\*

let ul\_element=document.querySelector("ul");

for(let i=0;i<ul\_element.children.length;i++){

    ul\_element.children[i].removeAttribute("class");

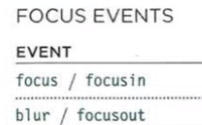
}

\*/



| **p5.js事件名稱** | **DOM的事件名稱(type)** |
| --- | --- |
| keyPressed(e) | keydown |
| keyReleased(e) | keyup |
| keyTyped(e) | keypress |







Event handling

1. 取得 Element
2. 註冊事件 用 bind()
3. 要做什麼事

# 事件觸發操作[鍵盤 KeyboardEvent]

今天來整理p5.js事件觸發操作

在網頁中的事件觸發是關係到整個操作流程及流暢度的關鍵。  
提供完整的事件項目，對於精準掌握要回應的內容以及穩定度，  
具有很重要的角色。

一般來說，網頁的主體是網頁的元件標籤，  
因此事件的種類與事件的來源，事件本物提供的資訊內容，  
就是需要研究的重點了。

在p5.js的事件類型分成4種  
Keyboard 鍵盤  
Mouse 滑鼠  
Touch 面板觸擊  
Acceleration 裝置加速度偵測

### Keyboard 鍵盤

事件在p5.js的用法  
原本的寫法是

function keyPressed() {

console.log("key: "+key);

}

但是為了可以獲取更多的事件資訊  
建議改以下面的寫法  
其中 keyPressed(e) 的 e 代表回傳的事件物件

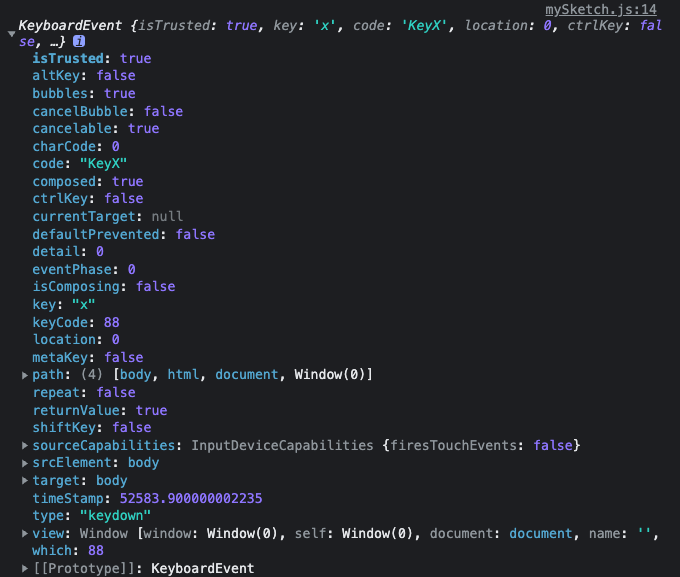
function keyPressed(e) {

console.log("key: "+key);

console.log(e);

}

產生的事件物件為 KeyboardEvent

這是從鍵盤事件物件中，可以讀取的事件屬性  


主要會需要讀取的事件屬性有

| **事件屬性** | **說明** |
| --- | --- |
| key: "x" | 按鍵名稱 |
| keyCode: 88 | 按鍵 ASCII |
| type: "keydown" | 事件動作類型 |
| target: body | 事件觸發元件對象 |

這3個按鍵事件如果連續觸發，只會觸發一次，直到其他事件觸發，才會再次觸發。

| **p5.js事件名稱** | **DOM的事件名稱(type)** |
| --- | --- |
| keyPressed(e) | keydown |
| keyReleased(e) | keyup |
| keyTyped(e) | keypress |

以下這2個功能，是用在draw()區段中，可以連續偵測按鍵事件

keyIsPressed 是否按下鍵盤  
keyIsDown(keyCode) 是否按下 LEFT\_ARROW 向左鍵  
keyCode為按鍵 ASCII

複合按鍵偵測 為 if(keyIsDown(CONTROL) && keyIsDown(65)){ }

function draw() {

if (keyIsDown(LEFT\_ARROW)) { //-- LEFT\_ARROW

console.log(key+", "+keyCode);

}

if (keyIsDown(CONTROL) && keyIsDown(65)){ //-- CTRL+A

console.log("CTRL+A "+key+", "+keyCode);

}

if (keyIsDown(32)) { //-- space key

console.log(key+", "+keyCode);

}

if (keyIsDown(91) && keyIsDown(65)){

console.log("META+A "+key+", "+keyCode); //-- META+A

}

if(keyIsPressed){

console.log(key+", "+keyCode);

}

}

至於要查詢按鍵的ASCII，可以利用鍵盤事件來查詢

function keyPressed(e) {

console.log("keyPressed: " + key + ", " + keyCode); *// p5.js*

console.log("keyPressed: " + e.key + ", " + e.keyCode); *// DOM*

console.log(e);

}

有些按鍵的keyCode, 在p5js可以直接用按鍵名稱來表示  
BACKSPACE(32), DELETE(46), ENTER(13), RETURN, TAB(9), ESCAPE(27), SHIFT(16), CONTROL(17), OPTION(18), ALT, , UP\_ARROW(38), DOWN\_ARROW(40), LEFT\_ARROW(37), RIGHT\_ARROW(39), META(91)

keyIsPressed 與 keyIsDown(keyCode) 是特有的功能，主要用在需要連續偵測的狀況。  
另外，英文字母大小寫有時keyCode會時相同的，且會以大寫的keyCode為主  
'a' 的ascii是 97， 'A'的ascii是 65  
但是接下A鍵時 keyCode是65，要特別注意，以免誤以為條件設定錯誤。

最後是 在 keyPressed(e) 程式區段 最後一行加上

e.preventDefault();  
return false;

可避免觸發原本系統預設的按鍵功能。  
function keyPressed(e) {  
console.log("keyPressed: " + key + ", " + keyCode);  
console.log(e);  
e.preventDefault();  
return false; //-- 可避免觸發原本系統預設的按鍵功能  
}

像是 command+A 原本是全選的功能，加上return false; 後  
就只會偵測是否按下 command+A，而不會觸發全選的動作。

以下這些按鍵，都是可以形成組合鍵的主鍵，若是長按也只會觸發一次。  
可以用以判斷是否為組合鍵。  
shift, option(alt), control, meta(command), caps

KeyboardEvent {key: 'CapsLock', code: 'CapsLock' }  
KeyboardEvent {key: 'Shift', code: 'ShiftLeft' }  
KeyboardEvent {key: 'Control', code: 'ControlLeft' }  
KeyboardEvent {key: 'Alt', code: 'AltLeft' }  
KeyboardEvent {key: 'Meta', code: 'MetaLeft' }  
KeyboardEvent {key: 'Meta', code: 'MetaRight' }  
KeyboardEvent {key: 'Control', code: 'ControlRight' }  
KeyboardEvent {key: 'Shift', code: 'ShiftRight' }

以下是利用HTML DOM的方法來實現偵測組合鍵。

const map = [];

function setup() {

document.onkeydown = document.onkeyup = function (e) {

map[e.key] = e.type == 'keydown';

console.log(map);

if (map['Meta'] && map['a']) {

console.log('Meta+A');

}

if (map['Control'] && map['a']) {

console.log('Control+A');

}

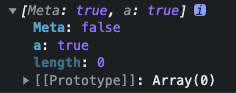
e.preventDefault();

return false;

};

}

利用map陣列來記錄在按下與放開之間，按了哪些按鍵，這些按鍵在按下時狀態是true，放開時是false，  
最後就可以檢查是否同時組合鍵的狀態是true，就可以確認按下組合鍵。  
假設按下meta(command)鍵時，  
map[e.key] = (e.type == 'keydown');  
--> map['Meta'] = true;  
if (map['Meta'] && map['a']){}



一般設定鍵盤事件的對象都是以整個網頁為對象，  
比較沒有針對 個別元件在設定鍵盤事件，  
所以在setup()的區段設定鍵盤事件就可以了

document.onkeydown = document.onkeyup = function(e) { };  
document.onkeydown = function(e) { };  
document.onkeyup = function(e) { };

也可以使用 arrow function 箭頭函式的方式表示  
document.onkeydown = document.onkeyup = (e) => { };  
document.onkeydown = (e) => { };  
document.onkeyup = (e) => { };

document.onkeydown = document.onkeyup = (e) => {

map[e.key] = e.type == 'keydown';

console.log(map);

if (map['Meta'] && map['a']) {

console.log('Meta+A');

}

if (map['Control'] && map['a']) {

console.log('Control+A');

}

e.preventDefault();

return false;

};

以上是先以鍵盤按鍵事件為例，其他類型的事件操作，大致就是依循這個原則來理解。

# 事件觸發操作[滑鼠 MouseEvent]

今天來整理滑鼠的事件操作

在p5.js中滑鼠的事件，常用的有  
mouseClicked()，單擊  
doubleClicked()，雙擊

mouseMoved()，移動(沒有按下的時候)

mousePressed()，按下  
mouseDragged()，拖動(按下的時候)  
mouseReleased()，放開  
(簡稱：點，拖，放)

mouseWheel()，滾動

按鍵的種類有 左鍵，中鍵(滾輪)，右鍵

先來看看事件的幾種程式寫法  
onclick = function(e){ console.log(e); };  
onclick = (e) => { console.log(e); };  
產生的事件物件類別為 PointerEvent

//-- 這是以id為d1的div作為觸發的對象

//-- 將事件要處理的事情，另用 function click1(e) 編寫

<div id="d1" onclick="click1">Click Me!</div>

<script>

function click1(e) {

console.log(e);

}

</script>

//-- 以選取id為d1的div，設定事件onclick，並以箭頭函式的方式表示

<div id="d1">Click Me!</div>

<script>

document.querySelector("#d1").onclick = (e) => { console.log(e); };

</script>

//-- 這是以整個body作為觸發的範圍

<script>

function mouseClicked(e) {

console.log(e);

return false;

}

</script>

一般事件的觸發，需要知道的資訊有  
1.觸發的對象  
2.觸發事件的種類  
3.觸發的座標

let can;

function setup() {

can = createCanvas(200, 200);

can.id("can1");

can.class("c1");

can.position(100, 100);

can.style("padding", "20px");

can.style("border", "solid");

can.parent("p1");

background(100);

document.querySelector("#d1").onclick = (e) => { console.log(e.path[0]); };

can.onclick = (e) => { console.log(e.target+" click!"); };

}

function showEventInfo(e) {

console.log(e);

console.log(e.target.localName); *//-- 觸發對象*

console.log(e.type); *//-- 動作種類*

console.log(e.button); *//-- 按鍵編號*

console.log(e.offsetX + ", " + e.offsetY); *//-- 觸發座標*

console.log(mouseX + ", " + mouseY); *//-- 觸發座標*

}

function mouseMoved(e) {

showEventInfo(e); *//-- MouseEvent, mousemove*

return false;

}

function mouseDragged(e) {

showEventInfo(e); *//-- MouseEvent, mousemove*

return false;

}

function mousePressed(e) {

showEventInfo(e); *//-- MouseEvent, mousedown*

return false;

}

function mouseReleased(e) {

showEventInfo(e); *//-- MouseEvent, mouseup*

return false;

}

function mouseClicked(e) {

showEventInfo(e); *//-- PointerEvent, click*

return false;

}

function doubleClicked(e) {

showEventInfo(e); *//-- MouseEvent, dblclick*

return false;

}

function mouseWheel(e) {

showEventInfo(e); *//-- WheelEvent, wheel*

return false;

}

滑鼠左鍵編號：0  
滑鼠中鍵(滾輪鍵)編號：1  
滑鼠右鍵編號：2

# 網頁元件的事件操作

今天來整理一下在網頁中常見的事件操作的功能

一般除了鍵盤，滑鼠(觸擊)之外，網頁的元件都有提供相對應的事件，讓網頁的功能可以更適時地產生回應的動作，甚至是驅動整個網頁流程的運作。

常見的有 window，document，div，video，audio，canvas，iframe，a(hyper-link)，input，button，table，img，這些是主要的網頁元件，雖然元件的種類很多，但是大致上分成2類，第1類是跟使用者操作有關的，像是鍵盤，滑鼠(觸擊，點擊)，選擇，輸入等，第2類是跟元件本身的運作有關，像是載入，播放，完成動作，連線等。  
這些我們都要以一種系統性地來理解事件運作的原理，這樣當查閱一個元件有哪些事件可以使用，就更可以掌握使用的時機。

首先，先來了解一下，在HTML DOM當中，事件的類型有哪些，當事件產生的時候，可以透過 console.log(event)來查閱是什麼事件，以及提供的事件屬性資料有哪些。  
常見的有

| **事件物件種類** | **事件產生場合** |
| --- | --- |
| AnimationEvent | For CSS animations CSS 動畫事件 |
| ClipboardEvent | For modification of the clipboard 剪貼簿事件 |
| DragEvent | For drag and drop interaction 拖拉事件 |
| FocusEvent | For focus-related events 聚焦事件 |
| HashChangeEvent | For changes in the anchor part of the URL 錨點事件 |
| InputEvent | For user input 輸入事件 |
| KeyboardEvent | For keyboard interaction 鍵盤事件 |
| MouseEvent | For mouse interaction 滑鼠事件 |
| PageTransitionEvent | For navigating to, and away from, web pages 網頁切換事件 |
| PopStateEvent | For changes in the history entry 網頁造訪歷史事件 |
| ProgressEvent | For the progress of loading external resources 外部載入進度事件 |
| StorageEvent | For changes in the window's storage area. 儲存資料事件 |
| TouchEvent | For touch interaction 觸控事件 |
| TransitionEvent | For CSS transitions CSS 轉場事件 |
| UiEvent | For user interface interaction UI操事件 |
| WheelEvent | For mousewheel interaction 滾輪事件 |

若是以元件種類來分類的話，有以下類別  
Window Event -> body  
Form Events -> input  
Keyboard Events ->  
Mouse Events  
Drag Events  
Clipboard Events  
Media Events  
Misc Events

### 網頁載入事件

由以上可看出，在DOM JS中，document本身並沒有提供事件，

console.log(document);



基本上，如果要在網頁載入時觸發的事件，寫法如下  
第1種寫法

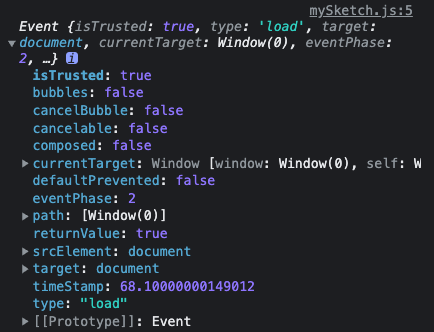
window.onload = (e) => {

console.log(e);

e.preventDefault();

return false;

};



event object: Event  
target: document  
type: "load"

第2種寫法

<body onload="ready(event)"></body>

<script>

function ready(e){

console.log(e);

console.log(e.target);

e.stopPropagation();

}

</script>

這個onload寫在 body標籤中，若要將事件傳到函式中，  
可以用event元件作為參數傳入

<body onload="ready(event, this)"></body>

<script>

function ready(e, a){

console.log(e); *//-- Event*

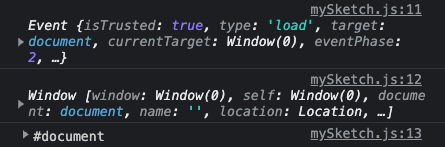
console.log(a); *//-- Window*

console.log(e.target);

e.stopPropagation();

}

</script>



### 觸發的連動與預防

有時事件觸發時只想觸發自訂的動作，不想引發原本存在於網頁的動作時，可以使用 preventDefault() 指令，  
event.preventDefault(); 停止預設事件動作  
return false; 也有停止預設事件動作的效果

有時事件觸發時只想觸發作用在本身的元件上，不想連動到上層的元件時，可以使用 stopPropagation() 指令，  
event.stopPropagation(); 停止事件向外層傳遞

<div id="d1">

<div id="d2">

<div id="d3">Click Me!</div>

</div>

</div>

<script>

document.querySelector("#d1").onclick = (e) => {

console.log(e);

console.log(e.target);

e.stopPropagation();

};

document.querySelector("#d2").onclick = (e) => {

console.log(e);

console.log(e.target);

e.stopPropagation();

};

document.querySelector("#d3").onclick = (e) => {

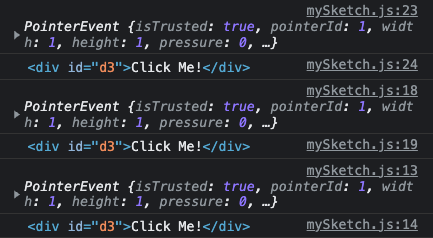
console.log(e);

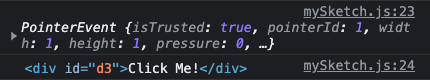
console.log(e.target);

e.stopPropagation();

};

</script>

沒有加 e.stopPropagation(); 時，當click div#d3時，觸發3次事件。  


加上 e.stopPropagation(); 後，當click div#d3時，就只會觸發1次事件。  


### 停止滑鼠右鍵觸發Context Menu

window.oncontextmenu = (e) => {

e.preventDefault();

console.log(e);

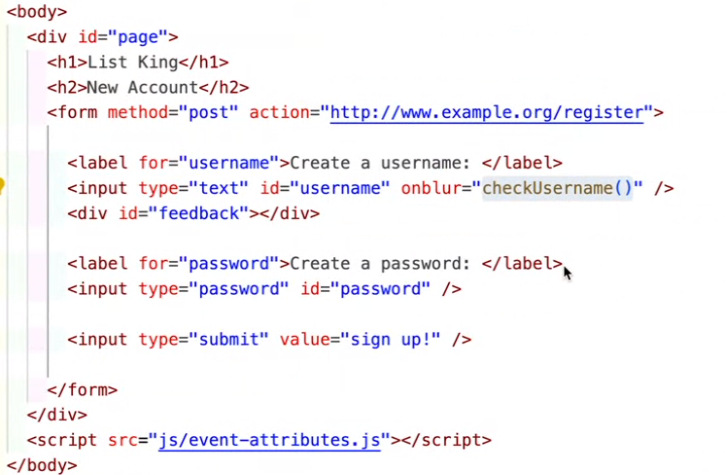
return false;

};

event object: PointerEvent  
target: html  
type: "contextmenu"

## onblur 離開 (not focus)

Html even (此方法已不用了, 改用直接寫在 js 的方法)



js

function checkUsername() {

    let feedbackElement = document.getElementById("feedback");

    let usernameElement = document.getElementById("username");

    let username = usernameElement.value;

    if (username.length < 5) {

        feedbackElement.textContent = "使用者名稱必須大於5個字元"

    }

    else {

        feedbackElement.textContent = ""

    }

}

## 方法一 直接寫在 js 裡面 用 .onblur

function checkUsername() {

    let feedbackElement = document.getElementById("feedback");

    //let username = usernameElement.value;

    let username = this.value;

    if (username.length < 5) {

        feedbackElement.textContent = "使用者名稱必須大於5個字元"

    }

    else {

        feedbackElement.textContent = ""

    }

}

//html handler

let usernameElement = document.getElementById("username");

usernameElement.onblur = checkUsername;

## 方法二 直接寫在 js 裡面, 用 addEventListener

//event listener

document.getElementById('username').addEventListener('blur', checkUsername,false);

function checkUsername() {

    let feedbackElement = document.getElementById("feedback");

    //let username = usernameElement.value;

    let username = this.value;

    if (username.length < 5) {

        feedbackElement.textContent = "使用者名稱必須大於5個字元"

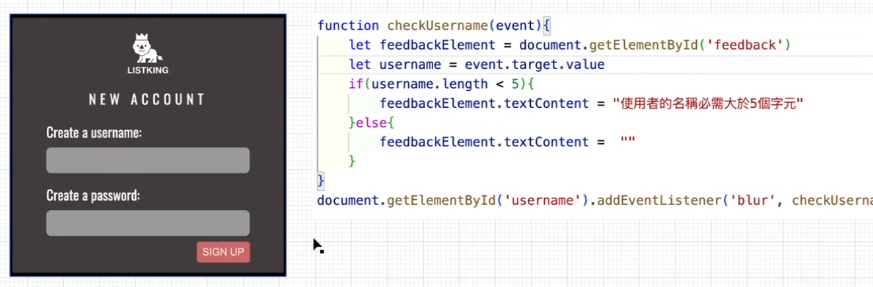
    }

    else {

        feedbackElement.textContent = ""

    }

}



<https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>

window load 事件

//() =>  匿名function

//宣導式function

// 註冊

window.addEventListener('load', (event) => {

console.log('網頁已載入');

});

//() =>  匿名function arrow function

//宣導式function

// 註冊

window.addEventListener('load', (event) => {

console.log('網頁已載入');

let usernameElement=document.getElementById('username');

usernameElement.focus();

});

## ()=> 的裡面使用 this 的時候, 指的是 windows 要注意!!

document.getElementById('username').addEventListener('focus',(event)=>{

let feedbackElement = document.getElementById('feedback');

feedbackElement.className = 'tip';

feedbackElement.innerHTML = 'username 最少要5個元';

});

//用匿名, this 是指 windows , 要小心

document.getElementById('username').addEventListener('blur',(event)=>{

let feedbackElement = document.getElementById('feedback');

console.log(this,this.value);

console.log(event.target,event.currentTarget);

if(event.currentTarget.value.length<5) //event.target == event.currentTarget.rentTarget

{feedbackElement.className = 'warning';

feedbackElement.innerHTML = '注意:username 最少要5個元';  }

else{

    feedbackElement.innerHTML = "";

}

});

## mouse 事件

let bodyElement = document.getElementById("body");

let sx = document.getElementById("sx");

let sy = document.getElementById("sy");

let px = document.getElementById("px");

let py = document.getElementById("py");

let cx = document.getElementById("cx");

let cy = document.getElementById("cy");

bodyElement.addEventListener("mousemove",(event) =>{

    sx.value = event.screenX;

    sy.value = event.screenY;

    px.value = event.pageX;

    py.value = event.pageY;

    cx.value = event.clientX;

    cy.value = event.clientY;

});

## 鍵盤事件

* 鍵盤相關事件有下列三種，在大多數情況下會將鍵盤事件註冊在 input 的輸入框上。
  + keydown 事件: 「壓下」鍵盤按鍵時會觸發 keydown 事件。
  + keypress 事件: 除了 Shift, Fn, CapsLock 這三種按鍵外按住時會觸發，若按著不放則會連續觸發。
  + keyup 事件: 「放開」鍵盤按鍵時觸發。

如果我們針對同個元素同時綁定了這三個鍵盤事件，那麼這三個事件執行的順序會是：

"keydown"

"keypress"

"keyup"

若此時想要知道使用者按下的按鍵，則可以透過 event.keyCode 屬性來查詢。  
註： keyCode 的對應表可以到這裡查看：<https://gist.github.com/tylerbuchea/8011573> 。

例如，今天你想要當使用者在 input 輸入框按下 「enter」 時，發動 submit，就可以這樣做：

textBox.addEventListener('keydown', function(e){

*// enter 的 keyCode 是 13*

if( e.keyCode === 13 ){

formSubmit();

}

}, false);

像這樣，透過 e.keyCode 就可以判斷使用者目前按下的是哪個按鈕。

let messageElement = document.getElementById('message');

let lastKey = document.getElementById('lastKey');

let charDisplay = document.getElementById('charactersLeft');

messageElement.addEventListener('keyup',(event) => {

lastKey.innerHTML = `最後一個<br>key: ${event.key} <br>keycode:${event.keyCode}<br>code:${event.code}`

let textEntered = messageElement.value;

let counter = 180 - textEntered.length;

charDisplay.textContent = `還剩下可輸入字數: ${counter}`;

} )

### 表單相關事件

* + input 事件： 當 input、 textarea 以及帶有 contenteditable 的元素內容被改變時，就會觸發 input 事件。
  + change 事件： 當 input、select、textarea、radio、checkbox 等表單元素被改變時觸發。 但與 input 事件不同的是，input 事件會在輸入框輸入內容的當下觸發，而 change 事件則是在目前焦點離開輸入框後才觸發。
  + submit 事件：當表單被送出時觸發，通常表單驗證都會在這一步處理，若驗證未通過則 return false;。
  + focus 事件：當元素被聚焦時觸發。
  + blur 事件：當元素失去焦點時觸發。

## Validation 驗證

let formElement = document.getElementById('formSignup');

let termsHint = document.getElementById('termsHint');

let terms = document.getElementById('terms');

let packageElement = document.getElementById('package');

let packageHint = document.getElementById('packageHint');

let packagselect = packageElement.options[packageElement.selectedIndex].value;

console.log(packagselect);

packageElement.addEventListener('change', (event) => {

    // selectElement options

    /\*

    for(const option of packageElement.options){

    console.log(`value:${option.value}, \ntext:${option.text}, \nselected:${option.selected ? 'yes':'no'}`);

    }

    \*/

    //selectedIndex 取對應的index

    if (packageElement.value == "monthly") {

        packageHint.innerText = "如果選擇1年,可省 $10 美金";

    }

    else { packageHint.innerText = "Good Job正確的選擇"; }

});

formElement.addEventListener('submit', (event) => {

    console.log('form submitted');

//event.preventDefault(); //關掉預設的動作

    if (terms.checked) {

        termsHint.innerText = '感謝你的訂購';

        termsHint.style.backgroundImage = "url(images/hint.png)";

    } else {

        event.preventDefault(); //關掉預設的動作

        termsHint.innerText = '你必須打勾同意選項';

        termsHint.className = 'warning';

    }

})

## mutation變種功能DOM 变动观察器（Mutation observer）

//let classbuttonelement = document.getElementsByClassName('button')[0];

//let classbuttonelement = document.querySelector('.button');

let classbuttonelement = document.querySelector('a.add');

//let ulelement = document.getElementsByTagName('ul')[0];

let ulelement = document.querySelector('ul');

let countelement = document.querySelector('#counter');

classbuttonelement.addEventListener('click', (event) => {

    event.preventDefault();

    console.log('Button Clicked');

    let linum = ulelement.getElementsByTagName('li').length;

    let createaddlist = document.createElement('li');

//    createaddlist.innerText = 'New Item';

let textNode = document.createTextNode('New Item'+linum);

    createaddlist.appendChild(textNode);

    ulelement.appendChild(createaddlist);

})

const observer = new MutationObserver(function (mutations) {

    let listItems = ulelement.getElementsByTagName('li').length;

    countelement.innerHTML=listItems;

console.log('mutations');

  });

observer.observe(ulelement, {

    childList: true,

    attributes: true,

    characterData: true,

  });

### 特殊事件

* + Composition Event (組成事件):

**Composition Event** 其實指的是 compositionstart 、 compositionend ，以及 compositionupdate 這三個事件。

介紹 Composition Events 之前先來談談 DOM API 過去對輸入框偵測變化的幾個方式。

前面介紹過，常見的表單輸入框如: <input type="text"> 如果要動態監聽輸入框的文字變化時， 大多會透過監聽 keydown、keypress、keyup 等鍵盤事件來判斷 value 是否變動，但如果是透過「**複製貼上**」之類的操作，就無法透過鍵盤事件來判斷。

而即使是 change 事件則是要在使用者改變內容，且**焦點離開輸入框**的前一刻才會被觸發。

所以後來有了 input 事件， input 事件會在輸入框的**內容被改變時即時觸發**，確實也解決了過去在 onChange 以及鍵盤相關事件功能不足所產生的問題。

但是，新的問題來了！

通常像這樣的搜尋框，我們會用類似 autocomplete (自動完成) 的方式給使用者搜尋建議 (以 google 為例)：  


如上圖，在輸入中文的時候，通常會需要透過注音之類的輸入法來做拼字。

但是在大部分的情況下，針對「注音符號」或是「拼音文字」去給搜尋建議是沒有太大意義的。

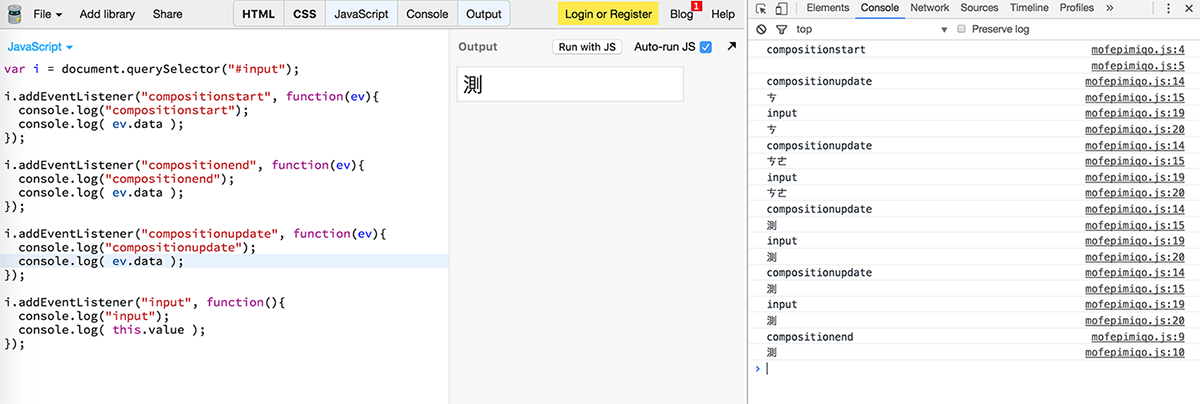
這個時候就需要透過 Composition Events 來為輸入框做增強。

透過 Composition Events 我們可以觀察使用者在輸入框內開啟輸入法 (Input Method Editor, IME) 時，組字或選字的狀態。

Composition Events 提供三個事件給開發者監聽：分別是 compositionstart 、 compositionend ，以及 compositionupdate。

* + compositionstart: 輸入框內開啟輸入法，且**正在拼字時**觸發。
  + compositionupdate: 輸入框內開啟輸入法，且**正在拼字**或**選字時**更改了內容時觸發。
  + compositionend: 輸入框內開啟輸入法，拼字或選字**完成**，正要送出至輸入框時觸發。

執行的時候像這樣：



可以看到，如果要確認使用者輸入完成並送出文字時，就可以透過 compositionend 來做最後確認。

## 自訂事件

自訂事件可以用 Event constructor 建立，同樣透過 addEventListener 去監聽，由 dispatchEvent 決定觸發的時機。

var event = new Event('build');

*// 監聽事件*

elem.addEventListener('build', function (e) { ... }, false);

*// 觸發事件*

elem.dispatchEvent(event);

若是想要在自訂事件內增加更多資料，則可以改用 CustomEvent：

var event = new CustomEvent('build', { 'detail': elem.dataset.time });

那麼在 Event Handler 就可以透過 event 來接收：

function eventHandler(e) {

log('The time is: ' + e.detail);

}

當然，瀏覽器提供的事件相當多，今天分享的部分主要是比較常見，以及工作實務上需要特別注意的部分。

其他的多數事件你都可以在 MDN 的 [Event reference](https://developer.mozilla.org/en-US/docs/Web/Events) 找到。