DOM

抓取

- 單一
  - html
  - js
- 多選
  - html
  - js
  - 多選陣列迴圈
- 練習

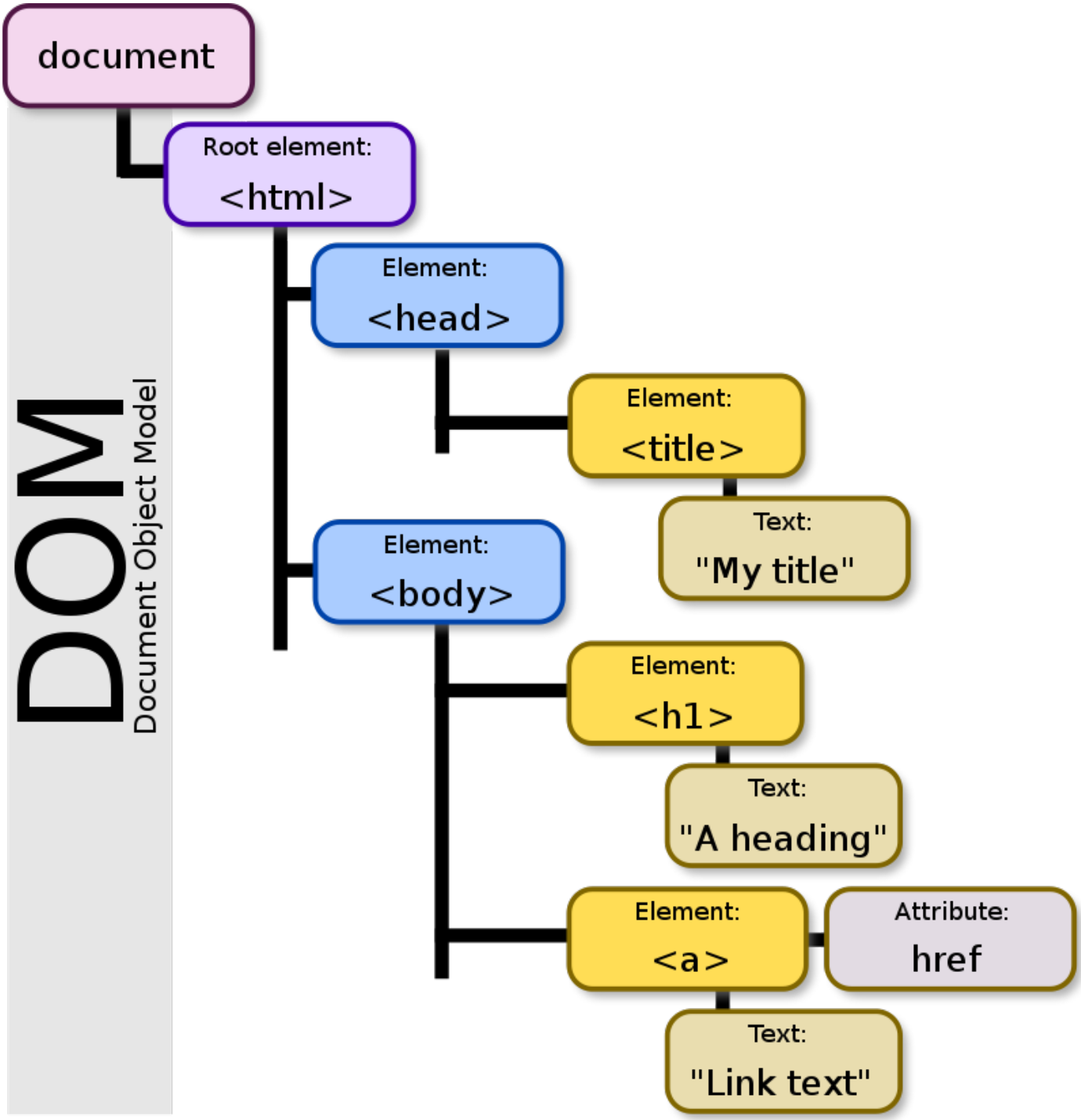
控制

- 內容
  - html
  - js
- 數值
  - html
  - js
- CSS
  - html
  - js
- class
  - 附加
    - html
    - js
  - 移除
    - html
    - js
  - 取得
    - html
    - js
  - 判斷 class 存在
    - html
    - js
- 練習
  - html
  - css
  - js
- dataset
  - html
  - js

節點

- 移除
  - html
  - js
- 父節點
  - html
  - js
- 子節點
  - html

DOM



https://zh.wikipedia.org/wiki/%E6%96%87%E6%A1%A3%E5%AF%B9%E8%B1%A1%E6%A8%A1%E5%9E%8B

萬物皆為物件(Object)。

抓取

利用 JavaScript 取得 HTML 元素。

codepen

單一

`documennt.querySeletor('seletor')`

單一抓取情況下，無論抓取的 DOM 有幾個都只會顯示第一個。

html

```html
<div id="uid">UID</div>
<div class="students">Studnet 1</div>
<div class="students">Studnet 2</div>
```

js

```js
let uid = document.querySelector('#uid');
let students = document.querySelector('.students');
let div = document.querySelector('div');

console.log(uid.innerHTML);
console.log(students.innerHTML);
console.log(div.innerHTML);
```

`dom.innerHTML` ⇒ 抓取內容

多選

`documennt.querySeletorAll('seletor')`

多選抓取情況下，無論抓取的 DOM 有幾個都會顯示陣列型態。

html

```html
<div id="uid">UID</div>
<div class="students">Studnet 1</div>
<div class="students">Studnet 2</div>
```

js

```
let uid = document.querySelectorAll('#uid');
let students = document.querySelectorAll('.students');
let div = document.querySelectorAll('div');

console.log(uid.innerHTML);
console.log(students.innerHTML);
console.log(div.innerHTML);
```

多選陣列迴圈

```
students.forEach(function(student, index) {
    console.log(index + ': ' + student.innerHTML)
})
```

練習

抓取下列架構的 `<li>3</li>`

```
<ul id="items">
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
</ul>
```

控制

抓取 `DOM` 後，進一步對此進行讀取與寫入操作。

內容

`dom.innerHTML`

html

```
<div id="uid">UID</div>
```

js

```js
let uid = document.querySelector('#uid');

// 讀取
let html = uid.innerHTML;
console.log(html);

// 寫入
uid.innerHTML = 'New UID';

// 再次讀取
console.log(uid.innerHTML);
```

數值

`dom.value`

html

```html
<input value="David Lin" id="full-name">
```

js

```js
let fullName = document.querySelector('#full-name');

// 讀取
let value = fullName.value;
console.log(value);

// 寫入
fullName.value = 'John Chen';

// 再次讀取
console.log(fullName.value);
```

CSS

`dom.style`

遇到有 `-` 相連的屬性，使用小駝峰命名法指定。

`background-color` ⇒ `backgroundColor`

html

```
<div id="uid">UID</div>
```

js

```
let uid = document.querySelector('#uid');

uid.style.color = 'red';

uid.style.backgroundColor = 'yellow';
```

class

`dom.classList`

附加

`dom.classList.add('class name')`

重複附加同樣的名稱也只會顯示一個。

html

```
<div id="uid">UID</div>
```

js

```
let uid = document.querySelector('#uid');

uid.classList.add('some');
```

移除

`dom.classList.remove('class name')`

移除不存在的名稱不會發生錯誤。

html

```
<div id="uid">UID</div>
```

js

```
let uid = document.querySelector('#uid');

uid.classList.remove('some');
```

取得

`dom.className`

字串顯示，可使用 `String.split(' ')` 拆分為陣列。

html

```
<div id="uid" class="first name full">UID</div>
```

js

```
let uid = document.querySelector('#uid');

console.log(uid.className);

let classes = uid.className.split(' ');
console.log(classes);
```

判斷 class 存在

html

```
<div id="uid" class="first name full">UID</div>
```

js

```
let uid = document.querySelector('#uid');

let classes = uid.className.split(' ');
if (classes.indexOf('first') ≥ 0) {
    console.log('first exists.');
}
```

除了透過字串拆分陣列尋找的方式外，也可以用內建的 `classList.contains('class name')` 來檢查。

```
let uid = document.querySelector('#uid');

if (uid.classList.contains('first')) {
    console.log('first exists.');
}
```

練習

1. 建立一顆球體與一顆按鈕。

2. 按鈕按鈕後，球體向左移動 80%。

3. 再按一次按鈕，球體恢復原本位置。

html

```
<button id="run">Run</button>
<div id="ball">Ball</div>
```

css

```
#ball {
    width: 60px;
    height: 60px;
    position: relative;
    display: flex;
```

```
    align-items: center;
    justify-content: center;
    background-color: #dedede;
    border-radius: 60px;
    margin-top: 30px;
    left: 0;
}

#ball.go {
    animation: go 2s ease-in-out;
    left: 80%;
}

#ball.back {
    animation: back 2s ease-in-out;
    left: 0;
}

@keyframes go {
    0% {
        left: 0;
    }

    100% {
        left: 80%;
    }
}

@keyframes back {
    0% {
        left: 80%;
    }

    100% {
        left: 0;
    }
}

```

js

```
let run = document.querySelector("#run");
let ball = document.querySelector("#ball");

run.addEventListener("click", function () {
    if (ball.classList.contains("go")) {
        ball.classList.remove("go");
        ball.classList.add("back");
    } else {
        ball.classList.remove("back");
        ball.classList.add("go");
    }
});
```

codepen

dataset

`data-*` 屬性的值。

html

```
<div id="uid" data-id="1">UID 1</div>
```

js

```
let uid = document.querySelector('#uid');

console.log(uid.dataset.id);

uid.dataset.id = 999;
```

節點

控制自己以外的上層與下層。

移除

`dom.remove()`

html

```
<div id="uid">UID</div>
```

js

```
let uid = document.querySelector('#uid');

uid.remove();
```

父節點

`dom.parentNode`

html

```html
<div id="uid-parent">
    <div id="uid">UID</div>
    <div id="full-name">David Lin</div>
</div>
```

js

```js
let uid = document.querySelector('#uid');

let parent = uid.parentNode;

console.log(parent.innerHTML)
```

子節點

`dom.childNodes` ⇒ `NodeList` 型態，可使用 `Array.forEach`，連同純文字也會列出。

`dom.children` ⇒ `HTMLCollection` 型態，不可使用 `Array.forEach`，只會列出元素。

html

```html
<div id="uid-parent">
    <div id="uid">UID</div>
    <div id="full-name">David Lin</div>
</div>
```

js

```js
let uid = document.querySelector('#uid');

let parent = uid.parentNode;

console.log(parent.childNodes);
// [object NodeList] (5)

console.log(parent.children);
// [object HTMLCollection]
```

迴圈取得

```
parent.childNodes.forEach(function (node) {
    console.log(node);
});

parent.children.forEach(function (node) {
    console.log(node);
});
```

> parent.children.forEach is not a function

解法

利用 `Array.prototype.slice.call` 將 `HTMLCollection` 轉換為 `Array`

```
let children = Array.prototype.slice.call(parent.children);

children.forEach(function (node) {
    console.log(node);
});
```

子節點異動

子節點增加、刪除、插入操作。

增加

`dom.appendChild(element)`

`element` 必須是 `document.createElement('node')` 產生的物件。

html

```
<div id="uid-parent">
    <div id="uid">UID</div>
    <div id="full-name">David Lin</div>
</div>
```

js

```
let uidParent = document.querySelector('#uid-parent');

let newP = document.createElement('p');
newP.innerHTML = 'Hi, New P element,';
uidParent.appendChild(newP)
```

移除

`dom.removeChild(elememt)`

`element` 必須是 `document.createElement('node')` 產生的物件。

js

```
uidParent.removeChild(newP)
```

插入

`dom.insertBefore(insertElement, targetElement)`

`insertElement` ⇒ 插入的元素，必須是 `document.createElement('node')` 產生的物件。

`targetElement` ⇒ 插入的節點，必須是 `element` 物件。

js

```
uidParent.insertBefore(newP, uidParent.children[1]);
```

Event

`JavaScript` 任何事情都需要事件(`Event`)來驅動。

監聽

`dom.addEventLister(event, function)`

`event` ⇒ 事件類型

`function` ⇒ 執行函數

html

```
<button id="btn">Click</button>
```

js

```
let btn = document.querySelector('#btn');

btn.addEventListener('click', function(e) {
   console.log(e);
});
```

`e` ⇒ 事件物件，當事件觸發呼叫函數時，做為參數傳入。

事件類型

事件觸發的方式，下面介紹常用的事件類型。

click

點擊觸發。

```
let btn = document.querySelector('#btn');

btn.addEventListener('click', function(e) {
   console.log(e);
});
```

change

數值變化時觸發，用於輸入類型元素(有 `value` 屬性)。

html

```
<input type="text" id="full-name">
```

js

```
let fullName = document.querySelector('#full-name');

fullName.addEventListener('change', function(e) {
    console.log(fullName.value);
})
```

blur

焦點移出時觸發(非 `focus` 時)。

html

```
<input type="text" id="full-name">
```

js

```
let fullName = document.querySelector('#full-name');

fullName.addEventListener('blur', function(e) {
    console.log(fullName.value);
})
```

> 觀察 change 與 blur 差異。

keypress

鍵盤按下時觸發。

html

```html
<input type="text" id="full-name">
```

js

```js
let fullName = document.querySelector('#full-name');

fullName.addEventListener('keypress', function(e) {
    console.log(fullName.value);
})
```

keyup

鍵盤放開時觸發。

html

```html
<input type="text" id="full-name">
```

js

```js
let fullName = document.querySelector('#full-name');

fullName.addEventListener('keyup', function(e) {
    console.log(fullName.value);
})
```

練習

分數判斷

1. 建立分數數字輸入匡與一顆計算按鈕。
2. 按下按鈕時，抓取輸入匡內容，並判斷分數。
3. 分數等級如下
    - ≥ 90 ⇒ 甲
    - ≥ 80 ⇒ 乙
    - ≥ 70 ⇒ 丙
    - ≥ 60 ⇒ 丁
    - < 60 ⇒ 不及格

```html
<div><input type="number" id="score">
    <button id="calc">Calc</button>
</div>
<div id="result"></div>
```

```javascript
let result = document.querySelector("#result");
let score = document.querySelector("#score");
let calc = document.querySelector("#calc");

calc.addEventListener("click", function (e) {
    result.innerHTML = "不及格";
    let num = score.value;
    if (num ≥ 90) {
        result.innerHTML = num + " is " + "甲";
        return;
    }
    if (num ≥ 80) {
        result.innerHTML = num + " is " + "乙";
        return;
    }
    if (num ≥ 70) {
        result.innerHTML = num + " is " + "丙";
        return;
    }
    if (num ≥ 60) {
        result.innerHTML = num + " is " + "丁";
        return;
    }
});

```

codepen

BMI 計算

1. 建立身高與體重數字輸入匡與一顆計算按鈕。
2. BMI = 體重(公斤) / 身高(公尺)平方
3. BMI < 18.5 ⇒ 過輕
4. BMI ≥ 35 ⇒ 重度肥胖
5. BMI ≥ 30 ⇒ 中度肥胖
6. BMI ≥ 27 ⇒ 輕度肥胖
7. BMI ≥ 24 ⇒ 過重
8. 18.5 ≤ BMI < 24 ⇒ 正常

```html
<div>
    <label for="">體重</label>
    <input type="number" id="weight">
</div>
<div>
    <label for="">身高</label>
    <input type="number" id="height">
</div>
<div>
    <button id="calc">Calc</button>
</div>
<div id="result"></div>
```

```javascript
let weight = document.querySelector("#weight");
let height = document.querySelector("#height");
let calc = document.querySelector("#calc");
let result = document.querySelector("#result");

calc.addEventListener("click", function (e) {
    result.innerHTML = "";
    let w = weight.value;
    let h = height.value;
    if (!w || !h) {
        return;
    }

    let bmi = w / Math.pow(h / 100, 2);
    let content = "BMI is " + bmi + ", ";

    if (bmi < 18.5) {
        result.innerHTML = content + "過輕";
        return;
    }

    if (bmi ≥ 35) {
        result.innerHTML = content + "重度肥胖";
        return;
    }

    if (bmi ≥ 30) {
        result.innerHTML = content + "中度肥胖";
        return;
    }

    if (bmi ≥ 27) {
        result.innerHTML = content + "輕度肥胖";
        return;
    }

    if (bmi ≥ 24) {
        result.innerHTML = content + "過重";
        return;
    }

    result.innerHTML = content + "正常";
```

```
});

```

codepen

Event Bubble

事件監聽很好用，目前都是針對單一節點做事件綁定，如果需要綁定大量的節點，該如何處理？

例如：

當點擊 `li` 時觸發 `click` 事件，並且顯示該 `li` 內容。

```
<ul id="nav">
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
</ul>
```

```
let lis = document.querySelectorAll("#nav li");

lis.forEach(function (li) {
    li.addEventListener("click", function (e) {
        console.log(li.innerHTML);
    });
});
```

看起來可以正常監聽每個 `li`。

問題

1. 如果 `li` 有一萬個，會發生什麼事情？
2. 如果後續再增加 `li` 事件是否會有效果？

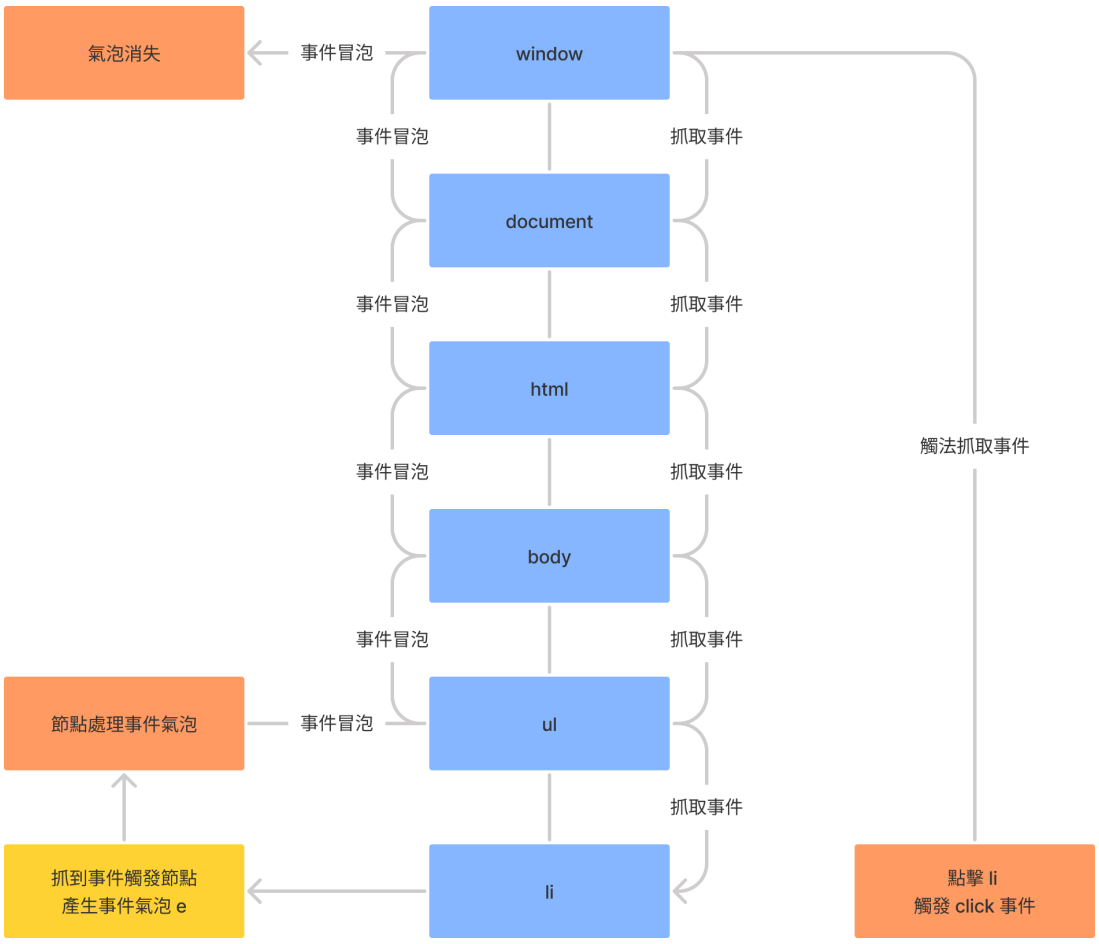
```
<ul id="nav">
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
</ul>
<button id="add-btn">add child</button>
```

```
let lis = document.querySelectorAll("#nav li");

lis.forEach(function (li) {
    li.addEventListener("click", function (e) {
        console.log(li.innerHTML);
    });
});

let addBtn = document.querySelector("#add-btn");
```

```
10
11  addBtn.addEventListener("click", function () {
12      let nav = document.querySelector("#nav");
13      let newLi = document.createElement("li");
14      newLi.innerHTML = "new li";
15      nav.appendChild(newLi);
16  });
17
```

觸發流程

先抓取，再冒泡。



事件氣泡

了解事件觸發流程後，針對氣泡特性，改善先前 `n` 個 `li` 的事件綁定問題。

```
let nav = document.querySelector("#nav");
nav.addEventListener("click", function (e) {
    let el = e.target;
    console.log(el.innerHTML);
});
```

`e.target` ⇒ 事件氣泡觸發元素物件。

問題

有沒有機會觸發到 `ul` ？

```
let nav = document.querySelector("#nav");
nav.addEventListener("click", function (e) {
    let el = e.target;
    if (e.tagName == "LI") {
        console.log(el.innerHTML);
    }
});
```

`e.tagName` ⇒ 取得觸發元素標籤，統一大寫。

中斷冒泡

事件氣泡無論節點是否有處理，都會往上冒泡。如果不想氣泡繼續往上，使用 `e.stopPropagation()` 中斷冒泡。

中斷預設行為

特定元素無需事件監聽就會進行預設行為，例如 `a` 元素會進行網頁跳轉。如想要禁止這種預設行為執行，使用 `e.preventDefault()` 中斷預設行為。

```
<a href="https://www.google.com/" id="link">Google</a>
```

```
let link = document.querySelector('#link');
link.addEventListener('click', function(e) {
    e.preventDefault();
    console.log(e.target.innerHTML);
});
```

計時器 Timer

在特定時間後，執行特定程序。

單次計時器

在特定秒數後，執行特定函數，只執行一次。

`setTimeout(function, ms)`

```
setTimeout(function() {
    console.log('Run after 1000ms');
}, 1000)
```

`1s = 1000ms`

取消計時器

`clearTimeout(timer)`

`timer` ⇒ 為 `setTimeout` 回傳計時器編號。

```
let timer = setTimeout(function() {
    console.log('Run after 4000ms');
}, 4000);

clearTimeout(timer);
```

重複計時器

在特定秒數後，執行特定函數，重複執行直到取消為止。

`setInterval(function, ms)`

```
setInterval(function() {
    console.log('Run after 1000ms');
}, 1000)
```

`1s = 1000ms`

取消計時器

`clearInterval(timer)`

`timer` ⇒ 為 `setInterval` 回傳計時器編號。

```
let timer = setInterval(function() {
    console.log('Run after 4000ms');
}, 4000);

clearInterval(timer);
```

練習

時鐘

1. 建立 UI，HH:MM:SS。
2. 每秒更新該 UI 的時間。

時間取得方法

```
let d = new Date();
let hh = d.getHours();
let mm = d.getMinutes();
let ss = d.getSeconds();
```

參考

```
<div id="clock">
    <div>0</div>
    <div>0</div>
    <div>:</div>
    <div>0</div>
    <div>0</div>
    <div>:</div>
    <div>0</div>
    <div>0</div>
</div>
```

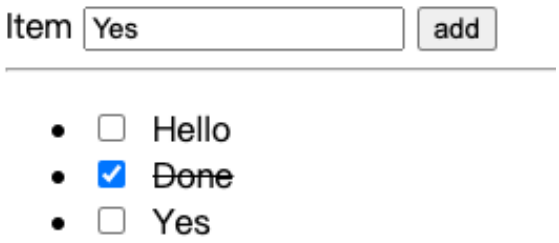
```
#clock {
    display: flex;
}

#clock > div {
    width: 60px;
    height: 60px;
    background-color: #000;
    color: #fff;
    display: flex;
    justify-content: center;
    align-items: center;
    border-left: 1px solid #cecece;
}

```

```
function update() {
    let d = new Date();
    let hh = d.getHours();
    let mm = d.getMinutes();
```

```
    let ss = d.getSeconds();

    if (hh < 10) {
        hh = "0" + hh;
    }

    if (mm < 10) {
        mm = "0" + mm;
    }

    if (ss < 10) {
        ss = "0" + ss;
    }

    let time = hh.toString() + mm.toString() + ss.toString();
    let clock = document.querySelector("#clock");

    [0, 1, 3, 4, 6, 7].map(function (num, index) {
        clock.children[num].innerHTML = time.charAt(index);
    });
}

update();

setInterval(function () {
    update();
}, 1000);

```

codepen

TODO List



1. 建立文字輸入匡，與一顆新增按鈕。
2. 建立列表區塊，每個項目前面加入 checkbox，如打勾則開項目顯示刪除線樣式。(text-decoration: line-through;)
3. 按下新增按鈕後，將目前文字輸入框內容新增至下方列表。

```
<div>
    <label for="">Item</label>
    <input type="text" id="item">
    <button id="add-btn">add</button>
</div>
<hr>
<ul id="todo-list"></ul>
```

```css
#todo-list li input:checked + span {
    text-decoration: line-through;
}

#todo-list li span {
    padding-left: 10px;
}
```

```js
let todo = document.querySelector("#todo-list");
let addBtn = document.querySelector("#add-btn");
let input = document.querySelector("#item");

addBtn.addEventListener("click", function (e) {
    let value = item.value;
    if (!value) {
        return;
    }

    let li = document.createElement("li");
    let checkbox = document.createElement("input");
    let span = document.createElement("span");

    checkbox.type = "checkbox";
    span.innerHTML = value;
    li.appendChild(checkbox);
    li.appendChild(span);

    todo.appendChild(li);
});

```

codepen