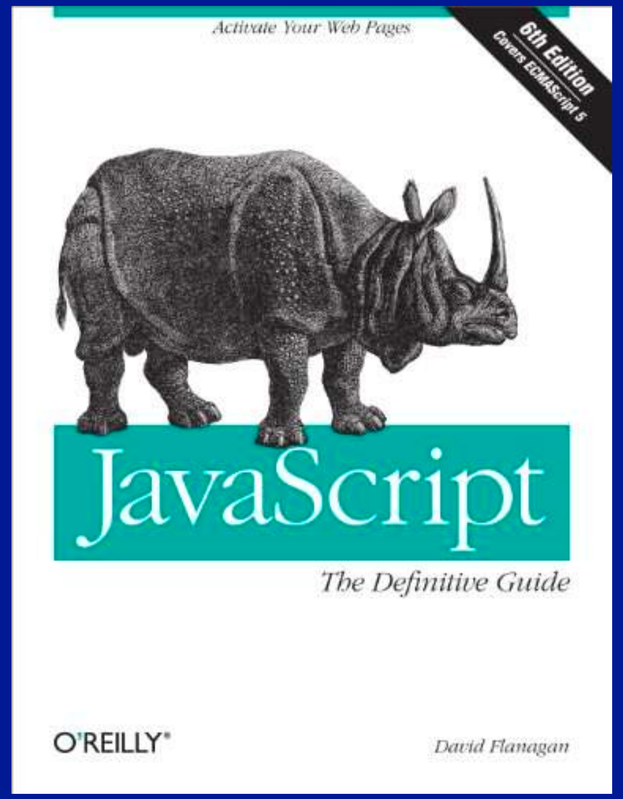
# 02. Intro to JavaScript




Ric Huang / NTUEE

(EE 3035) Web Programming

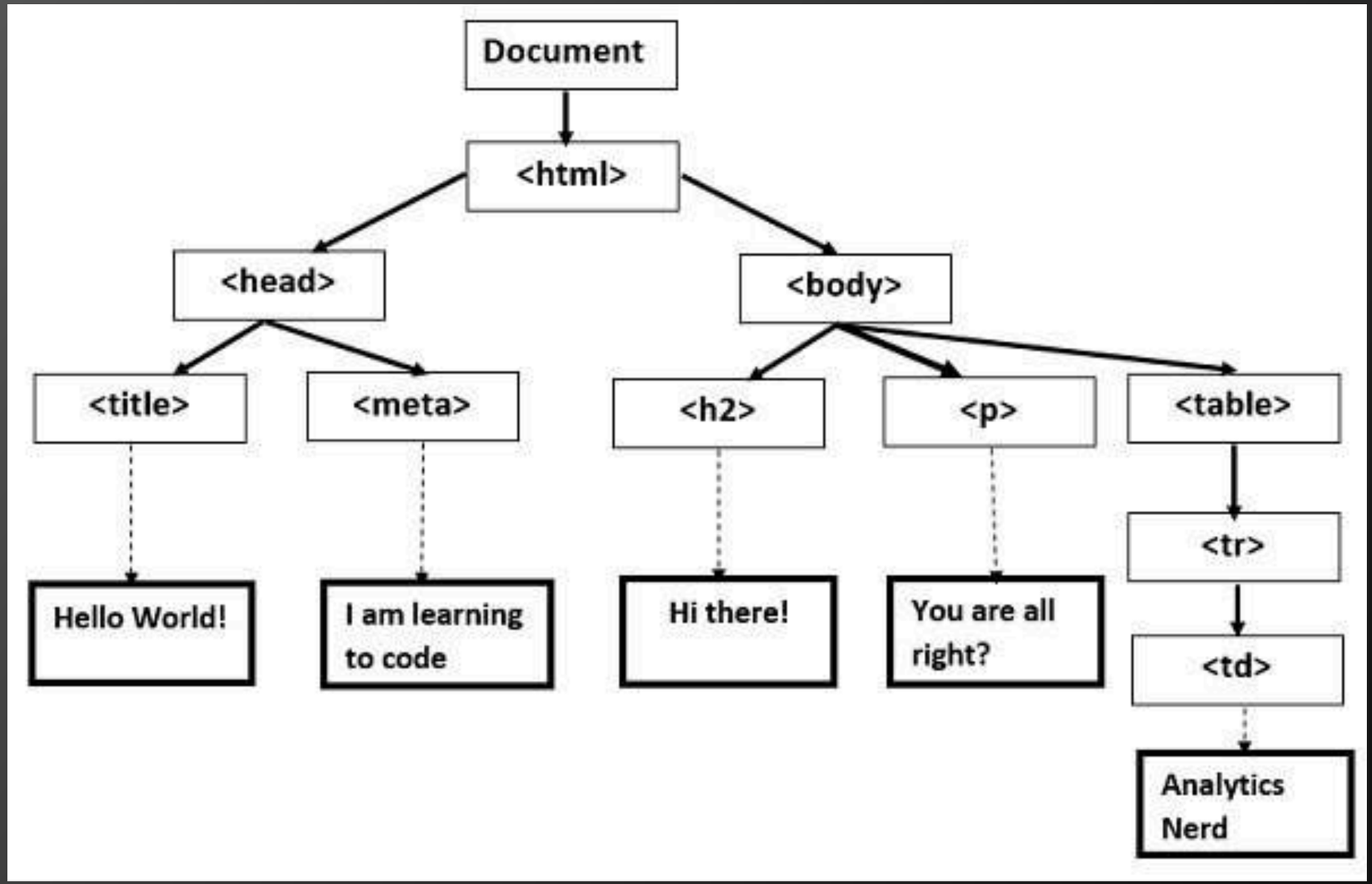
在學習完 HTML & CSS 之後，
你應該是有能力可以刻出一個靜態網頁
（雖然 CSS 的 state selector
也是可以做出一定程度的動態網頁啦！）

現代的動態網頁 (各種跟使用者 I/O 的互動)，
幾乎都是使用 JavaScript 寫的
但為了接下來理解如何
利用 JavaScript 製作出動態網頁，
我們必須先瞭解何謂
DOM (Document Object Model)

# DOM (Document Object Method)

(wiki) …is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.

# DOM Example


Document
<html>
<head>
<body>
<title>
<meta>
<h2>
<p>
<table>
<tr>
<td>
Hello World!
I am learning to code
Hi there!
You are all right?
Analytics Nerd

# Let's Hack it!

- https://www.ntu.edu.tw/administration/president.html

# Hacking 台大校長室?

- 開啟台大校長室首頁
- 打開 開發人員選項 -> 檢視元素
- 滑鼠在頁面滑動，看是否可以看得出 DOM structure?
- 滑到管校長玉照，click on its parent  element (有看到旁邊多了一個 $0 嗎？)
- 打開 “console”，輸入 $0 試試看！
- (先到網路上選一張圖片，複製連結) 輸入 $0.innerHTML = “<img src=‘你的圖片連結’ … 照著原先的打”
- (選到 “校長 管中閩博士” 的 <p>) 用 $0.innerText 改改看！

Did you really hack it??

# Not Really!!

剛剛改的是你瀏覽器(前端)收到的 HTML (i.e. local 端)，一但 reload, 從 server 端 (i.e. 台大計中) 重新載入網頁，剛剛的修改就都不見了！

不過，如果你寫了一個網頁服務程式，
放在自己的伺服器，讓使用者在他與
瀏覽器進行互動的過程中 (如：滑鼠點按、
填資料送出等)，將「動作」送至伺服器，
然後你的網頁程式「聽到」這個動作「事件」後，
送回對應的網頁更新(部分)腳本 (i.e. DOM node)，
使用者就會看到更新的網頁了！

# Additional Notes

- 你如果把圖片下載到 local disk, 然後打入：

```
$0.innerHTML = "<img src='/youFileLocaton'... >"
```

- 你會發現這樣不 work...  (why?)
  - 這跟 CORS (Cross-Origin Resource Sharing) 限制有關
    // covered later
- 此外，字串的引號 "" 與 '' 的交錯使用也要注意

在介紹JavaScript語法之前，
讓我們來了解一下 JavaScript 精彩的
黑歷史
Possibly the most hated language in the world

# A new web language was born...

Year 1995 (rise of IE; Netscape/Sun/Microsoft)

Brendan Eich created it in a few weeks

披著 Java 皮，流著 Scheme (first-class function) / Self (prototype-based)的血

(Note: Microsoft implemented their own version of JavaScript, called JScript, and included it in IE 3.0 in 1996)

# 醜小鴨變天鵝

- 短時間生產出來的畸形兒
- ECMA standardization (ES):
  1 (1997), 2 (1998)
- ES3: the first big change (1999)
- ES4: the unborn child
- ES3.1 (2008) -> ES5 (2009)

# ES6 (aka. ES 2015): The Modern JS

- let/const
- arrow function
- class
- promise (to relieve callback hells)
- generator
- binary/octal literals

# ES7 and Beyond

- ES7 (aka. ES 2016)
  - minor changes (e.g. exponential **)
- ES2017
  - async/await (to improve “promise”)
  - shared memory and atomics
- ES2018
  - asynchronous iteration
  - generator arrow functions
- ES2019
  - New functions to Array and String
  - Description method is added to Symbol

# What's next?

- Inter-platform operability
- Native support on VM
- (Already is) Full-stack language

# JavaScript 與 C++ 比較。相異之處

- 弱型別，所有的變數都是 var (or let/const)

```
var a = "38";
var b = false;
console.log(a + !b);
```

- Statement 的後面可以不用加 “;”
- 數字都是 “double“

```
0.1 + 0.2; // = 0.30000000000000004
```

- 除了 ==, 還有 ===
- 字串可以用 ' ' or " “

# JavaSCript。Prototype–Based Object Construction

不用先定義 class 即可建置物件，
且一旦一個物件被建置好，
後續的物件可以用它來當作原型
來建置類似的物件

# JavaScript。First-Class Functions

也就是說 functions 被當成
是一般的變數(物件)，
可以當成其他 function 的
參數或是回傳值，也可以被
assigned 給別的變數

# JavaScript Basics

- console.log()
- Types/Variables/Objects
- Function
- Array
- Control statements
- Variable/Function Hoist and Scope
- DOM manipulation & Event listening

# console.log()

- 是你 debug 的好朋友，會將訊息印在 “console”
- 在 Browser 打開 JS console
  - e.g. command+option+j for mac/Chrome
- Example:
  - `var a = 1;`
  - `console.log(a);`

# Types/Variables/Objects

- JavaScript 只有五種內建型別(primitive types)，其他都是物件(objects)

| 型態 | 範例 |
| --- | --- |
| Undefined (未定義) | undefined |
| Null (空值) | null |
| String (字串) | "哎呀" |
| Boolean (布林值) | true, false |
| Number (數字) | 3.1415926 |

# Any error?

- var a = 3;
  a = 4;
  var a = 5;
- b = 6;
  var b = 7;

// See “Variable Hoist” and “variable Scope” later

# In JS, almost everything is an object...

- Object in JavasScript is represented as:

  ```
  { property: value, ... }
  ```

- Example:

  ```
  var me = { name: "Ric", age: 18 }
  ```

- Note:
  - “Name” and “age” are properties, not variables. No need to define “type” or “var” for them
  - Recall: Object is defined when it is constructed

# 三種產生 object 的方法

1. Object literal
   - `var a = { name:"Ric", score:100 };`
2. “new” operator
   - `var b = new Date;`
3. Constructor function
   - ```
     function Student(name, score) {
       this.name = name;
       this.score = score;
     }
     ```
   - `var c = new Student("Ric", 100); // return this`

- What happens for “var d = Student( “Nacy", 20);” ?
  ==> The returned type of Student() is “undefined”
     ```

# 不要把五種內建型別宣告成 object

- var a = new Number(3); // 不建議
- var b = Number(3); // What's the difference?
- console.log(a === b); // false

- var a = 3; // a is a primitive
- var b = Number(3); // b is a primitive
- var c = new Number(3); // c is an object

# The keyword “Object” defines an object

- var o = new Object;

# “prototype-based” object construction

```
var o = new Object;
console.log(o);
// No need to define "name" and
"score" in advance
o.name = "Ric";
o.score = 100;
var o2 = new Object;
console.log(o);
console.log(o2);
```

# Primitive variable assignment makes a “copy”

```
• var a = 3;
• var b = a;   // {a, b} = {3, 3}
• b = 4;       // {a, b} = {3, 4}
• a = 5;       // {a, b} = {5, 4}
```

# Object variable assignment pass the “reference”

- `var i = { a:3 };`
- `var j = i;`
- `j.a = 4;  // i.a = 4`
- `i.a = 5;  // j.a = 5`

# Function Objects

# Function as an Object

- Functions in JavaScript is “first-class”
  // Function as an object (i.e. constructor function)
  - var f = function add(a, b) { return a + b; };
  - Note: Function 的 arguments 沒有必要加上 “var”
    NOT “function add(var a, var b)”
- Function can be anonymous // recommended
  - var add = function(a, b) { return a + b; };

# Function Assignment

- 如同 object assignment (i.e. pass by reference)
- // f is an reference to add
  ```
  var f = function add(a, b) { return a + b; };
  ```
- // print out the returned value of f(3,4)
  ```
  console.log(f(3,4));
  ```
- // print out f itself
  ```
  console.log(f);
  ```
- // Error (why?)
  ```
  console.log(add);
  ```

# Typename or A Function Object?

- This is wrong

```
var a = function add(x, y) { return x + y; }
a(3, 5);
add(10, 20);  // Error
```

- This is OK // a constructor function

```
function add(x, y) { return x + y; }
add(10, 20);
```

# Return of a function

- Unless explicitly state a returned value, by default, the return value of a function is "undefined"
  - var f = function(a, b) { return a + b; };
    f(3,5);
    f(f(1,2),3);
  - var g = function() { console.log(0); }
    g();  // 0; undefined
  - var h = new add(…);  // return this

# Return a function

```
• var f = function(s) {
    return s?
      function(a,b) { return a+b; }:
      function(a,b) { return a-b; }
  };
• var f1 = f(true); f1(3,5);
• var f2 = f(false); f2(3,5);
```

# Methods in Objects

- Recall: prototype-based object construction
  - `var a = { name:"Ric", score:100 };`
    `a.gender = "Male";`
- Since function is also an object, we can define "method functions" in an object as:
  - ```
    var a = {
        name:"Ric", score:100,
        report: function() {
            return this.name + " got " +
                   this.score;
        }
    };
    a.report();    // Ric got 100
    ```
    ```

# Define a method afterwards

- a.isPass = function() {
  console.log(this.score >= 60? “Yes” : “No” );
  };
  a.isPass()  // Yes

# Immediately Invokable Function Expression

- When a function is used only once, we can declare it anonymously and evoke it immediately
  - ```
    (function() {
    ... some statements
    })();
    ```
    ```

# Test yourself #1 (Any errors?)

- function f(a) { console.log(a); }
  f(0);
  var b = f;
  b;
  b(10);
  b("Hello");
  var c = f(20);
  c;  // undefined
  c(30);  // Error!

# Test yourself #2 (Any errors?)

- var f = function add(a, b) { return a + b; };
  add(10, 20); // Error!
  f(20, 30);
  var g = f;
  g(30, 40);
  var h = add; // Error!
  h(40, 50); // Error!

# Test yourself #3 (Any errors?)

- ```
  var f = function add(a, b) { return a + b; };
  var g = add; // Error
  var h = function add(a, b) { return a + b; };
  var i = function add(a, b, c) { return a + b + c; };
  var f = "Hello";  // overwriting f
  ```
  ```

# Test yourself #4 (Any errors?)

- var f = function pp() { ... }
  var g = f;
  var f = function qq() { ... }  // w/wo var
  f();  // qq
  g();  // pp

# Array Objects

# To define an array

1. Using array literal
   - var students = [ "John", "Mary", "Ric" ];

2. Using new Array // not recommended
   - var students = new Array("John", "Mary", "Ric");

# Data in an array can be of any types

```
var a = [
  "Ric",
  100,
  function() { console.log("Hello!"); },
  [ 1, 2, 3]
];
a[2]();  // Hello!
a[3][1]; // 2
```

# ”length” property and “push” method in Array

- // initialized as an empty array
  var s = [ ];
  s.push("John"); // recommended
  s[s.length] = "Mary"; // length is now 2
  s[3] = "Ric";  // OK, but create an
  // “undefined” in [2]

# Array is a special kind of object

- var students = [ "John", "Mary", "Ric" ];
  typeof students; // object
  Array.isArray(students);  // true

# Array elements MUST BE accessed by numbers

- 雖然一些 object 的行為看起來很像 array,
  但嚴格來說，他們是不一樣的
- var a = { name:"Ric", score:100 };
  a["name"]; // "Ric"
  a[0]; // undefined
  Array.isArray(a); // false

# Array or Object?

- Array is an object.
- var s = [];
  s[0] = "John";
  s["Name"] = "Ric";
  console.log(s);  // ["John", name: "ric"]
  Array.isArray(s); // true

Control statements in JavaScript are pretty much the same as in Java/C/C++

(so we skip the details here…)

# “==“ or “===“ ?

- Comparison in JavaScript uses “===” and “!==”
  - “==” 跟 “!=” 會雞婆地幫你做型別轉換
- Example:
  - “5” == 5; // = true
  - null == undefined; // = true
- 更多令人崩潰的型別轉換/比較會在下一章補充

# Variable/Function Hoist and Scope

# Variable 定義的位置會自動抬升到最前面

- 以下兩種寫法是一樣的(雖然前者不太正常)：

1. 
```
a = 1;
var a = 2;
```

2. 
```
var a = 1;
a = 2;
```

# Function Hoist

- Function declaration 定義的位置會被抬升到最前面
  - sum(3,5);  // This is OK!
    function sum(a, b) { return a + b; };
- 但用 expression 定義的 function 變數則不會被抬升
  => 如果在定義之前就被使用，就會是 error
  - sum(3,5);  // Error
    var sum = function(a, b) { return a + b; };

# Variable Scope

- 事實上，“var” 沒有寫也會過，只是會被視為全域變數 (global variable)
- 此外，“var” 如為區域變數，其範圍並非 block, 而是 function scope
  - function() {
    for (var i = 0; i < 10; i++)
    console.log(i);
    }
  - // 等同於
    function() {
    var i;
    for (i = 0; i < 0; i++)
    console.log(i);
    }

# Variable Scope

- 在同一個範圍內重複用 “var” 宣告同名字的變數，後者會被無視，而只進行 assignment
  - var a = 0;
    { var a = 10; } // 仍屬同一範圍之 'a'.
    // a = 10 now
    var b = function() { var a = 20; }
    b();
    console.log(a);
    // local 'a' 並非 global 'a'. 所以 a 仍然為 10

# “let” and “const”

2015 年發表的 ES6 版本加入了

“let” 跟 “const”

# “let” 跟 C/C++ 的變數一樣，採取 block scope

- 同一個 scope 不能宣告相同名稱的變數
- 不能在宣告變數之前就使用 (i.e. 用 let 宣告的變數不會被抬升 (hoist) 到 scope 的最前面)
- 在 global scope 用 let 宣告的變數並不是真正的 global variable

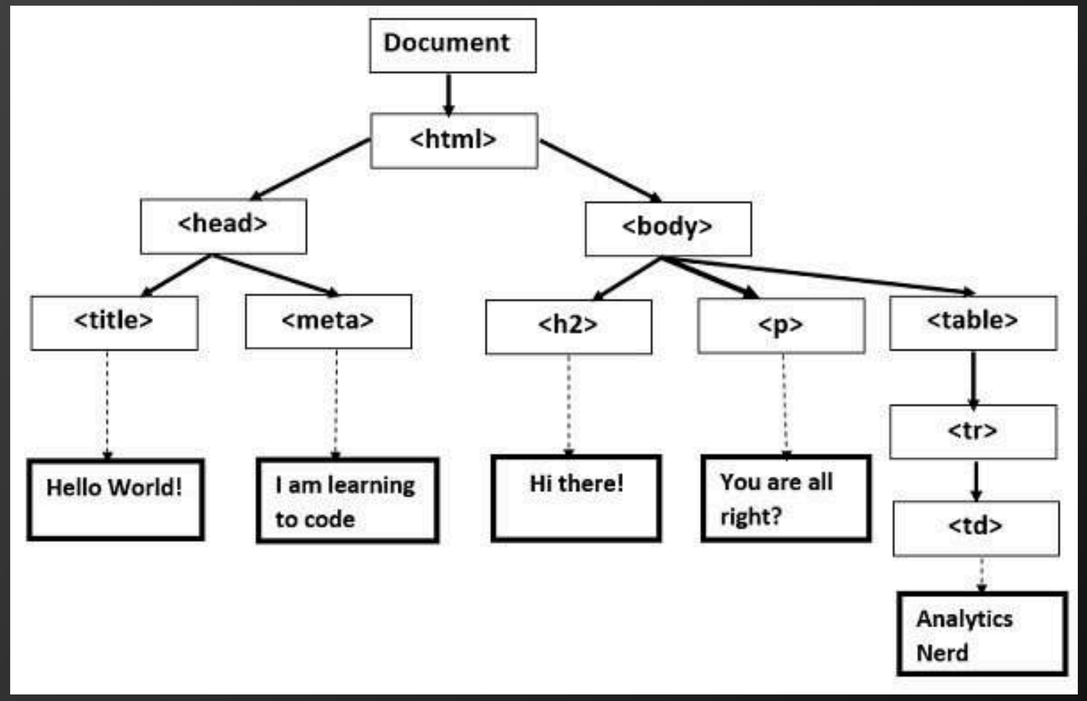
  ==> 並不會變成 “window” 這個全域物件底下的屬性，因此，像是用 module 載入的程式碼並看不到這個 global scope 的 let 變數

“const” 亦跟 C/C++ 一樣，
表示 “read-only” 的常數變數
養成習慣，多多使用 const

# DOM Manipulations

- 選擇節點
- 瀏覽節點
- 增減節點
- 修改節點內容
- 事件監聽綁定

在一個瀏覽器的頁面 “window”
是唯一的全域物件，
而 “document” 是它的一個屬性



如前所述，所謂的動態網頁
就是在 event 觸發之後，
選擇某個/某些 DOM 節點，
然後改寫其內容或是周邊的
節點 (node/element)

# 選擇節點

- 從 top level (i.e. “document”) 尋找 DOM nodes
  - 

1. 回傳 unique DOM node
   - var target = document.getElementById("target");
2. 回傳一個 array of DOM nodes
   - var arr = document.getElementsByClassName( “a-class");
   - var arr2 = document.getElementsByTagName("div");
3. 用 CSS 的一部份 selector 來選取
   - var target = document.querySelector("#target");
   - var arr = document.querySelectorAll(".a-class");

# 瀏覽節點

- 從某個 DOM node 找到下一層的節點
  (These are properties, not functions)
  - var arr = parentNode.children;
  - var element1 = parentNode.firstElementChild;
  - var element2 = parentNode.lastElementChild;
  - var count = parentNode.childElementCount;

# 增減節點

- 在 document 底下 或是 某個節點前後 增減節點
  - var newElement = document.createElement( “div” );
  - var newText = document.createTextNode( “Hello!” );
- Become the last child
  - var appendedNode = parentNode.appendChild(childNode);
- Removed from parent
  - var removedNode = parentNode.removeChild(childNode);
- Insert before refNode
  - var insertedNode = parentNode.insertBefore(newNode, refNode);
- Replace child node
  - var replacedNode = parentNode.replaceChild(newNode, oldNode);

# 修改節點內容

- 修改節點內容、屬性、樣式… 等
  - These are properties, not functions
- Get the serialized HTML code for all of its children
  - const childrenHTMLCode = thisNode.innerHTML;
- Replace all of its children using the HTML code
  - thisNode.innerHTML = childrenHTMLCode; // string
- Modify CSS style
  - thisElement.style.color = “red”;
- Modify properties
  - thisElement.className = “new-class1 new-class2”;
  - thisElement.classList.add(className); // 可用來產生動畫效果
  - thisElement.classList.remove(className);

# 事件監聽綁定（方法一）– addEventListener()

| eventType | 說明 |
| --- | --- |
| click | 點擊 |
| focus | 開始在輸入框輸入的時候 |
| blur | 離開輸入框的時候 |
| change | 輸入值改變的時候 |
| keydown | 鍵盤按下去的時候 |
| keyup | 鍵盤按下去的鍵上來的時候 |
| mouseenter | 滑鼠進到元素裡面的時候 |
| mouseleave | 滑鼠移出元素的時候 |

# 事件監聽綁定（方法一）– addEventListener()

- <button id= “target">是個按鈕</button>
- var targetElement = document.getElementById("target");
  targetElement.addEventListener(
  "click",
  function() {
  // ..做一些事，任何事，你希望按鈕按了要幹嘛？
  alert('你希望按鈕按了要幹嘛？');
  }
  );

# 事件監聽綁定（方法二）– GlobalEventHandlers [ref]

- 語法：target.onclick = functionRef;
- Click anywhere in this example.

- let log = document.getElementById('log');
  log.onclick = inputChange;
  function inputChange(e) {
    //.. some something here
  }

# 事件監聽綁定（方法三）— As a tag attribute

- List of global attributes [ref]
- <div class="myClass"
  onclick="clickHandler()">

# Practice Homework #2
# Image Viewer

(Download on Ceiba)

Deadline: 9pm, Monday, 03/15

# Image Viewer

Source: https://img.buzzfeed.com/thumbnailer-prod-us-east-1/dc23cd051d2249a5903d25faf8eeee4c/BFV36537_CC2017_2IngredintDough4Ways-FB.jpg?output-quality=60&resize=1000:*

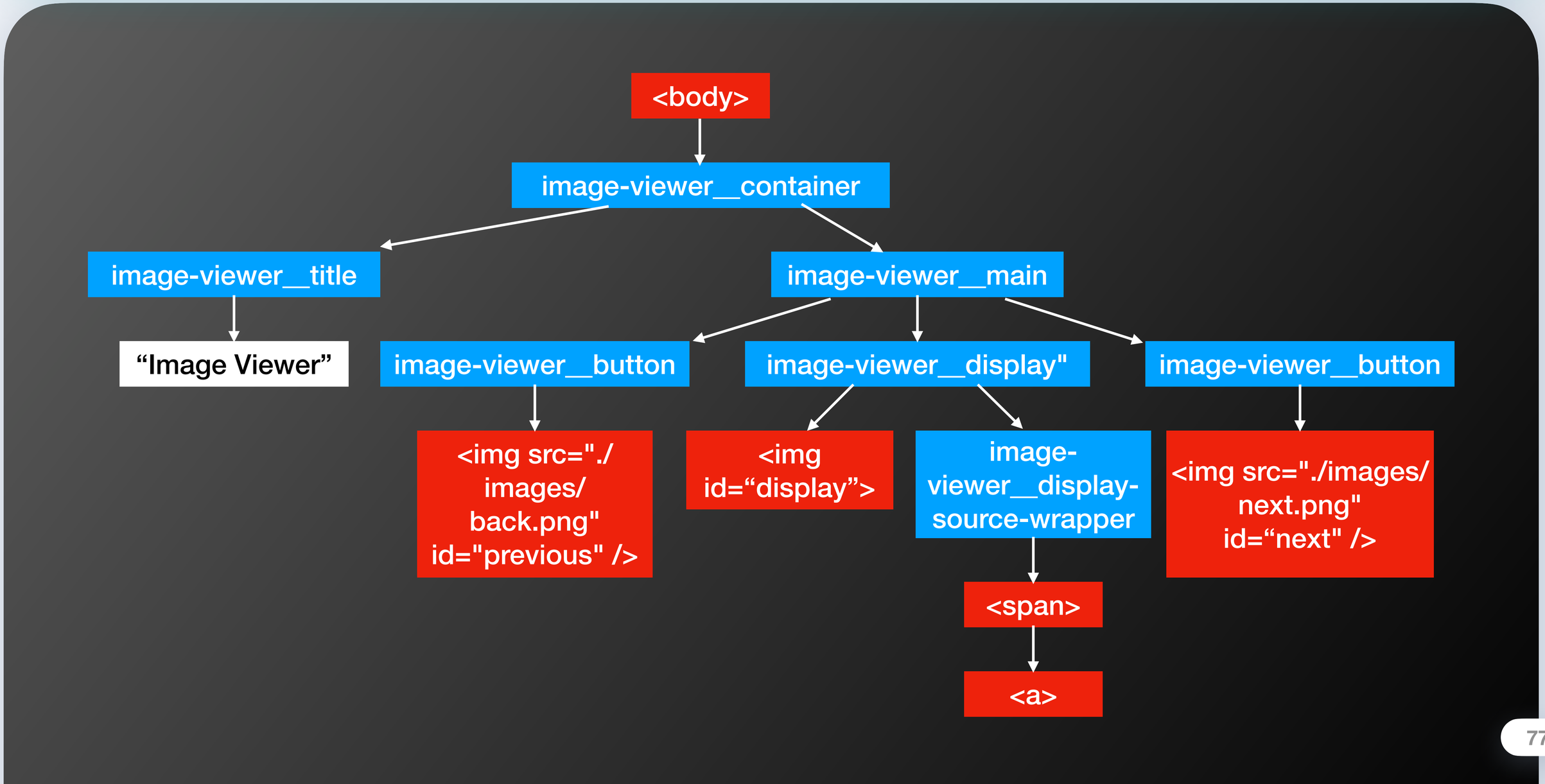
# Image Viewer。功能說明

- (Baseline) 至少完成可左右觀看不同圖片
- (Should have) 到結尾時按鈕不可以再按 (disabled)
- (Should have) 底下 show image source
- (Nice to have) Loading 時顯示 “loading.gif”

# Step 1: Create a HTML Layout

- Modify “index.html”
- Use the class names and IDs as defined in styles.css so that the outlook can be the same as the reference.

# Step 1: Create a HTML Layout



# Step 2: Write the JavaScript file

- Modify “main.js”

1. Create an array variable that stores the URLs of the images (to show)
2. Define some object variables that refer to the HTML elements participating in the image viewing

# Step 3: Create the interactions!

1. 先列出所有的互動
2. 針對每一個互動，先想好：
   - 哪一個 element 觸發？(如：某個  or <img>)
   - 什麼事件觸發？(如：onclick)
   - 觸發什麼樣的行為？(如：抽換圖片)
3. 一些細節的設計 (如：style 的改變，loading 時的貼心設計)

```
<div class="image-viewer__button" onclick="previousImage()">
```

# Step 4: Detail the JS functions

- 寫程式的 Lesson 1: No duplicated codes
  - Create functions to share!
- 畫面內容有什麼修改？
  - element.src = ‘…’;
  - element.href = ‘…’;
  - element.innerHTML = ‘…’;
- 呈現樣貌有什麼修改？
  - element.style.property = ‘…’;

# Step 5: 一些貼心的設計

- 如果需要時間載入，背景是否有個圖以免一片空白
  - Push some code before image loading
- 「計數」到達邊界時，讓按鈕的樣貌行為改變
  - 如何針對同樣的觸發事件有不同的呈現樣貌？
  - element.classList.add(someClass)
  - element.classList.remove(someClass)

# 感謝聆聽！

Ric Huang / NTUEE

(EE 3035) Web Programming