Javascript

01.01 - Intro

JavaScript is a “real” programming language
• Store variables
• Set decision points
• Loop
• Reuse code with functions

In addition
• Get data from the browser
• Manipulate the DOM that browsers use to create web pages

01.02 - DOM Review with OOP

Web Pages are built upon the DOM
• Document Object Model
• Structures documents like a tree
• Every node has one parent, and possibly many children
• Nodes have properties, methods, and events

Accessing the DOM is done with an `API – Application Programming Interface`
No matter which browser, no matter which scripting language, the API is the same

The DOM objects/elements

document – the root of the page
- document.URI, document.height, document.links, document.bgColor,….

element – a node in the tree
- Returned by a member of the API

nodeList – an array (group) of elements
- `document.getElementByTagName(‘p’)` would return a set of nodes

attribute
- A node in the DOM, though rarely used that way. Another way to manipulate/change the document

Specific APIs

• document.getElementById(id)
• document.getElementsByClassName(class)
• element.innerHTML
• element.style
• element.setAttribute(attribute, value)
• element.removeAttribute(attribute)

01.03 - JavaScript Output

What Can JavaScript do?

• Read and write HTML elements
• Reacts to events (mouse events, keyboard events, etc.)
• Validate data
• Detect the visitor's browser
• Create cookies

JavaScript Output

JavaScript doesn’t have a built-in print function

Data is displayed via
– an alert box using window.alert()

– a prompt using window.prompt()
– HTML output using document.write()
– HTML element using innerHTML()
– the browser console using console.log()

01.04 - JavaScript Variables

Declare: var name; → keyword: variable_name;

Variable Names

• Consists of letters, digits, underscores, and dollar sign($)
• Can not start with a digit
• Are case-sensitive… → name Name, naMe, NAME are all different variables
• Should be mnemonic (meaningful)

01.05 - Data Types

In JavaScript, a variable can take on many different types

01.06 - Operators and Expressions

02.01 - Functions

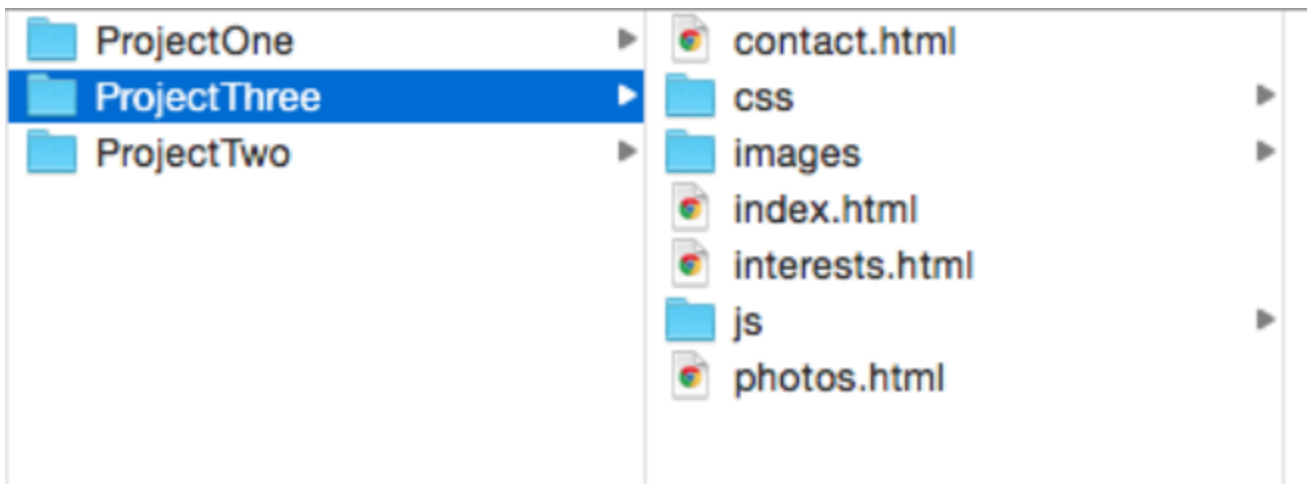
02.02 - Code placements

JavaScript code can be placed in the body, head, or in an external file

02.03 - Folder Structure

Web Developers tend to organize their code into separate parts
• HTML
• CSS
• Images
• JavaScript

Conventions



02.04 - Events

Adding the Interactivity!!

• It has been up to us to decide when the functions should execute
• It would be better if the functions were called based on special “events”
• The JavaScript API lets us add dynamic function calls!!

Events

`onclick` - User clicks on an HTML element

`onmouseover` - User moves the mouse over an HTML element

`onresize` - browser window is resized

`onload` - browser finishes loading the page

Using quotes

You can use single quotes or double quotes for the event result

Double quotes make it easier if you want to pass String parameters

```
<div onclick = "message('hi!')">
```

More Events

1. Mouse Events
   onclick, ondblclick, onmousedown, onmouseenter, onmouseleave, onmousemove, onmouseout,...
2. Keyboard Events
   onkeydown, onkeypress, onkeyup,...
3. Frame Events
   onload, onresize, onscroll, onerror,...

Comprehensive list

Event reference | MDN

Events are fired to notify code of "interesting changes" that may affect code execution. These can arise from user interactions such as using a mouse or resizing a window, changes in the state of the underlying environment (e.g. low battery or media events from the operating system), and other

https://developer.mozilla.org/en-US/docs/Web/Events

02.06 - "this"

“this” is a keyword that allows an element to reference itself

→ Every object in the DOM has an automatically generated “this”

Allows you to access an element’s info

→ Without “this” it would be difficult for the functions to know what data to use

“this” is also used outside functions

"this" example

03.01 - JavaScript Arrays

The elements in the array DON’T have to be all the same type

JavaScript Arrays are Objects

They have attributes and methods

• grades.length

• grades.sort()

• grades.push(element) → grades[grades.length] = element

03.04 - Flow Controls

Dynamic Execution

Sometimes it is helpful to put decision points into your code

• Add some variety to the program

• React to good/bad user input

• Avoid potential errors

Use if to specify a block of code to be executed, if a specified condition is true

Use else to specify a block of code to be executed, if the same condition is false

03.07 - Common Errors

Browsers try to hide your JavaScript errors, so you need to seek them out

Two classes of errors:
• Syntactic
• Logic

04.01 - Simple Forms

Forms add a new layer to the Request - Response Cycle

Forms are used to pass data

Different input elements can be used

Forms must have server with them

HTML form tags: <form>, <label>, <input>

```
<form>
 <label for='name'>Name</label>
 <input type='text' id = 'name' name = 'name'> <br/>
 <label>Email: <input type='email' name = 'email'> </label><br/>
 <label for='birth'>Birth</label>
 <input type='date' id = 'birth' value = 'birth'><br/>
 <input type = 'submit' value = 'Click Here'/>
</form>
```

Form elements have attributes
+ type
+ name
+ id

Input Types: textfield, email, password, radiobutton, checkbox, submit, number, range, color, date, url

Attribute:

+ `name` : Almost all input types should have a name attribute, The name attribute is assigned whatever value is input
+ `id` : Used for labels, by JavaScript
+ `value` :
  button: text inside the button
  textfield: provides a default value
+ `placeholder` : Provides a suggestion, Not an “official” value

04.02 - Simple Validation

How to validate

Use new HTML5 input types
• email, number, url

Use HTML5 attributes
• required, placeholder, min, max

Use JavaScript functions
• Write custom code to validate

Pattern

Works with input type = text and requires the input have a specific form
• [0-9]{5}
• [0-9]{13-16}
• [a-zA-Z]+

Limiting number

Validation with JavaScript

There isn’t always a guarantee that the browser will support the input type or pattern attributee

Code

Prompt

```
var nm;
nm = prompt("What is your name?");
document.write(nm);
console.log(nm);
alert(nm);
// ---------
var name;
name = prompt("What is your name?");
document.write(name);
document.write("<h1>" + name + "</h1>");
var date = Date();
document.write(date);
// ----------
var name;
name = prompt("What is your name?");
document.write(name);
document.write("<h1>" + name + "</h1>");
document.write(document.baseURI+"<br>");
document.write(document.title+"<br>");
// ----------
```

Current Date

```
<body onload="setDate()">
  <p> The current date is <span id = "date"></span></p>
```

```
function setDate(){
  document.getElementById('date').innerHTML=Date();
}
```

Change style

```
<h1>Changing the Style</h1>
<p>JavaScript can change the style of an HTML element.</p>

<button type="button" onclick="openMe()">Open!</button>
<button type="button" onclick="closeMe()">Close!</button>

<p id="demo">Extra details...You can open and close this paragraph using the buttons above.</p>
```

```
function closeMe(){
  // Find the element
  x=document.getElementById("demo");
  //Option 1: Change the style attribute directly
  x.style.display="none";

  //Option 2: Change the class
  // x.className="closed";
}

function openMe(){
  // Find the element
  x=document.getElementById("demo");
  //Option 1: Change the style attribute directly
  x.style.display="block";

  //Option 2: Change the class
  // x.className="open";
}
```

Add Fruit

```
<body onload="loadFruits()">
<button onclick="myFunction()">Add Your Favorite!</button>
<p id="fruits"></p>
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

function loadFruits(){
   document.getElementById("fruits").innerHTML = fruits;
}

function myFunction() {
  var fruit = prompt("What is your favorite fruit? ");
  fruits[fruits.length] = fruit;
  document.getElementById("fruits").innerHTML = fruits;
}
```

Sự khác nhau của bộ 3 Slice, Splice và Split trong Javascript

Slice

Tạo ra một **array mới** chứ không hề chỉnh sửa trên đối tượng **array** gọi đến nó.

- Không thay đổi mảng gốc
- Trả về kết quả là 1 mảng mới
- Có thể sử dụng cho cả mảng và chuỗi (string)

Syntax

```
array.slice(from, until)
> array
(6) [1, 2, 3, "hello world", 4.12, true]

> newArray
(3) [2, 3, "hello world"]
```

Splice

hàm **splice()** sẽ thay đổi trực tiếp trên đối tượng gọi đến nó.

- Dùng cho việc thêm, xóa phần tử trong mảng
- Trả về mảng là những phần tử bị xóa (trong trường hợp xóa phần tử trong mảng)
- Thay đổi trực mảng gốc
- Chỉ dùng cho mảng
- Cú pháp thêm phần tử: **array.splice (index, number of elements, element)**
- Cú pháp xóa phần tử: **array.splice (index, number of elements)**

```
array.splice(index, number of elements);
> array = [1, 2, 3, "hello world", 4.12, true];
> array.splice(3)
< (3) ["hello world", 4.12, true]
// lúc này array = [1, 2, 3]


array.splice(index, howmany, item1, ....., itemX)
let months = [1, 2, 3, 4]
months.splice(0, 1, 'january')
> months
(4) ["january", 2, 3, 4]
```

Split

chia một chuỗi **string** thành các **substrings** và trả về nó dưới dạng một mảng.

- Chia 1 chuỗi thành các chuỗi con
- Trả về là 1 mảng
- Không làm thay đổi chuỗi gốc
- Chỉ dùng cho String
- Lấy vào 2 đối số (có thể không bắt buộc) : **string.split(separator, limit)**

```
string.split(separator, limit);
"How are you doing today?".split(" ", 3)
(3) ["How", "are", "you"]
```

Array methods: `map`, `filter`, `reduce`

`map`, `filter`, `reduce` là những array method bắt nguồn từ 1 mẫu hình lập trình có tên gọi `functional programming`.

- `Array.prototype.map()` xử lí gì đó các phần tử, trả về mảng các phần tử đã được xử lí.
- `Array.prototype.filter()` kiểm tra và quyết định xem có giữ lại từng phần tử của mảng hay không, trả về là mảng chỉ bao gồm những phần tử được giữ lại.
- `Array.prototype.reduce()` nhận vào 1 mảng và gộp các phần tử của mảng thành 1 giá trị duy nhất rồi trả về giá trị đó.

Với 3 method này chúng ta có thể tránh việc sử dụng vòng lặp `for` hoặc `forEach` trong hầu hết các trường hợp.

Ví dụ

```
const numbers = [0, 1, 2, 3, 4, 5, 6];
const doubledNumbers = numbers.map(n => n * 2); // [0, 2, 4, 6, 8, 10, 12]
const evenNumbers = numbers.filter(n => n % 2 === 0); // [0, 2, 4, 6]
const sum = numbers.reduce((prev, next) => prev + next, 0); // 21
```

Ví dụ khác: tính tổng điểm của những học sinh có điểm trên 10 bằng cách kết hợp `map`, `filter` và `reduce`

```
const students = [
  { name: "Nick", grade: 10 },
  { name: "John", grade: 15 },
  { name: "Julia", grade: 19 },
  { name: "Nathalie", grade: 9 },
];

const aboveTenSum = students
  .map(student => student.grade) // map students thành mảng chứa điểm của hs
  .filter(grade => grade >= 10) // filter mảng chứa điểm và giữ lại những điểm trên 10
  .reduce((prev, next) => prev + next, 0); // tính tổng của những điểm trên 10

console.log(aboveTenSum) // 44 -- 10 (Nick) + 15 (John) + 19 (Julia), Nathalie có điểm dưới 10 nên không tính
```

Giải thích

Chúng ta sẽ sử dụng mảng sau trong các các ví dụ dưới đây

```
const numbers = [0, 1, 2, 3, 4, 5, 6];
```

Array.prototype.map()

```
const doubledNumbers = numbers.map(function(n) {
  return n * 2;
});
const doubledNumbers = numbers.map(n => n * 2);
console.log(doubledNumbers); // [0, 2, 4, 6, 8, 10, 12]
```

Array.prototype.filter()

```
const evenNumbers = numbers.filter(function(n) {
  return n % 2 === 0;
});
const evenNumbers = numbers.filter(n => n % 2 === 0);
console.log(evenNumbers); // [0, 2, 4, 6]
```

Array.prototype.reduce()

Mục đích của method `reduce` là gộp các phần tử của mảng mà nó duyệt qua thành 1 giá trị duy nhất. Gộp như thế nào thì phụ thuộc vào cách chúng ta quy định.

```
const sum = numbers.reduce(
  function(acc, n) {
    return acc + n;
  },
  0 // giá trị của biến tích lũy acc ở lần lặp đầu tiên
);

console.log(sum) //21
```

Giống như method `map` và `filter`, `reduce` được sử dụng với 1 mảng và nhận vào tham số thứ nhất là 1 hàm. Tuy nhiên có 1 điểm khác biệt đó là `reduce` nhận vào 2 tham số

- Tham số đầu tiên là 1 hàm sẽ được gọi ở mỗi lần lặp,
- Tham số thứ 2 là giá trị của biến tích lũy (`acc` trong ví dụ trên) ở lần lặp đầu tiên.

Hàm được truyền vào `reduce` cũng nhận vào 2 tham số. Tham số đầu tiên (`acc`) là biến tích lũy, tham số thứ 2 là phần tử hiện tại (`n`). Giá trị của biến tích lũy bằng với giá trị trả về của hàm ở lần lặp trước đó. Ở lần lặp đầu tiên `acc` bằng với giá trị của tham số thứ 2 được truyền vào method `reduce`.