JavaScript

Programming Guide

สมบูรณ์ พัฒน์ธีรพงศ์

Kontentblue Publishing

www.kontentbluepub.com

JavaScript Programming Guide

โดย สมบูรณ์ พัฒน์ธีรพงศ์

ราคา : 250 บาท

พิมพ์ครั้งที่ 1 : ธันวาคม 2554

ข้อมูลทางบรรณานุกรมของสำนักหอสมุดแห่งชาติ

สมบูรณ์ พัฒน์ธีรพงศ์

JavaScript Programming Guide.--กรุงเทพฯ : คอนเทนต์บลู, 2554.

280 หน้า.

1. จาวาสคริปต์ (ภาษาคอมพิวเตอร์). I. ชื่อเรื่อง.

005.133

ISBN 978-616-91030-0-4

จัดทำโดย : คอนเทนต์บลู พับลิชชิ่ง
เลขที่ 608/53-55 ถนนสาธุประดิษฐ์ แขวงบางโพงพาง
เขตยานนาวา กรุงเทพฯ 10120
โทรศัพท์ 0-2294-1663
http://www.kontentbluepub.com

พิมพ์ที่ : บริษัท เอ็น. วาย. ฟิล์ม จำกัด
เลขที่ 3 ซอยราชพฤกษ์ 4 ถนนราชพฤกษ์ แขวงบางจาก
เขตภาษีเจริญ กรุงเทพฯ 10160
โทรศัพท์ 0-2457-2596-7 โทรสาร 0-2868-6062
http://www.nydigitalprint.net

จัดจำหน่ายโดย : บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)
อาคารเนชั่นทาวเวอร์ ชั้นที่ 19 เลขที่ 1858/87-90
ถนนบางนา-ตราด แขวงบางนา เขตบางนา กรุงเทพฯ 10260
โทรศัพท์ 0-2739-8222, 0-2739-8000 โทรสาร 0-2739-8356-9
http://www.se-ed.com

บทที่ 5

Functions

บ่อยครั้งที่เรามีโค้ดที่เขียนซ้ำๆกันในหลายๆจุดของโปรแกรม หากเราต้องการเปลี่ยนแปลงแก้ไขโค้ดเหล่านั้น เราต้องตามไล่แก้มันทั้งหมดในทุกจุดของโปรแกรม ซึ่งเป็นเรื่องที่ยุ่งยากและเสียเวลาพอสมควร และหากเราแก้ไขไม่ครบทั้งหมด โปรแกรมของเราก็จะเกิดบั๊ก (Bug) ขึ้น เราสามารถหลีกเลี่ยงปัญหานี้ได้โดยการสร้างเป็นฟังก์ชัน ฟังก์ชัน (Function) คือการรวมกลุ่มของโค้ดเพื่อไว้เรียกใช้งานภายหลังผ่านชื่อของฟังก์ชัน ดังนั้นหากเรามีโค้ดที่เขียนซ้ำๆกันในหลายๆจุดของโปรแกรม เราจึงควรเขียนมันไว้ในฟังก์ชันและเรียกใช้ฟังก์ชันอีกทีหนึ่ง หากในอนาคตเราจำเป็นต้องเปลี่ยนแปลงแก้ไขโค้ด เราก็แก้ที่ฟังก์ชันเพียงที่เดียวเท่านั้น ที่ผ่านมาเราได้มีการเรียกใช้ฟังก์ชันไปบ้างแล้ว ไม่ว่าจะเป็น `document.write()`,`prompt()`,`array.join()` แต่นั่นเป็นฟังก์ชันสำเร็จรูปที่มาพร้อมกับ JavaScript หรือ Client-side JavaScript ในบทนี้เราจะมาเรียนรู้วิธีการสร้างและเรียกใช้ฟังก์ชันของเราเอง รวมทั้งหัวข้อต่างๆดังต่อไปนี้

- ฟังก์ชันกับตัวแปร Local
- ฟังก์ชันกับ Optional Arguments และตัวแปร Implicit ชื่อ “`arguments`”
- ฟังก์ชันกับ “`typeof`” Operator
- การสร้างฟังก์ชันด้วย Function Literal
- สร้างฟังก์ชันที่ไหนดี?

เมื่อคุณรู้เรื่องฟังก์ชันและรู้วิธีการเรียกใช้โค้ดจากไฟล์อื่นๆจากบทที่ 8 “Modules และ Namespaces” แล้ว คุณสามารถสร้างไลบรารี JavaScript ของคุณเองได้

การสร้างและการเรียกใช้ฟังก์ชัน

ในหัวข้อนี้เรามาเริ่มต้นด้วยการสร้างและการเรียกใช้ฟังก์ชันกัน การสร้างฟังก์ชันมี Syntax พื้นฐานดังนี้

```
function functionName( [arg1,arg2, . .]){
  // statements
  [return expression;]
}
```

การสร้างฟังก์ชันเริ่มต้นด้วยคีย์เวิร์ด `function` เสมอ และตามด้วยชื่อฟังก์ชัน ส่วนกฎการตั้งชื่อใช้กฎเดียวกันกับการตั้งชื่อตัวแปร *arg1*, *arg2*, . . คือตัวแปรที่รับค่าพารามิเตอร์ที่ส่งเข้ามายังฟังก์ชัน ซึ่งจะมีหรือไม่มีและจะมีกี่ตัวก็ได้ ถ้ามีมากกว่าหนึ่งตัวให้ใช้เครื่องหมาย Comma “,” คั่น นอกจากนี้ฟังก์ชันยังสามารถรีเทิร์นค่ากลับออกไปยังตัวเรียกได้ ถ้าต้องมีการรีเทิร์นค่า ให้ใส่คีย์เวิร์ด `return` ไว้เป็นคำสั่งสุดท้าย ตัวอย่างเช่น

```
function writeLine(message){
  document.write(message + "<br/>");
}


function add(num1,num2){
  return (num1 + num2);
}
```

เมื่อสร้างฟังก์ชันแล้ว เราสามารถเรียกใช้ฟังก์ชันด้วยชื่อฟังก์ชันและเครื่องหมาย “`()`” ตัวอย่างเช่น

```
writeLine("Welcome to JavaScript world");
writeLine("Today is " + new Date() );
var result = add(25,30);
writeLine( result );
```

ฟังก์ชันกับตัวแปร Local

ตัวแปร Local คือตัวแปรที่ถูกประกาศขึ้นภายในฟังก์ชัน ดังนั้นเราจึงเรียกใช้ตัวแปร Local ได้เพียงแค่ภายในฟังก์ชันที่มันถูกประกาศเท่านั้น ตัวอย่างเช่น

```
function multiply(num1,num2){
  var result = num1 * num2;
  return result;
}
```

ตัวอย่างข้างต้นตัวแปร `num1`, `num2`, และ `result` เป็นตัวแปร Local ทั้งหมด

ฟังก์ชันกับ Optional Arguments

Argument หรือเรียกได้อีกอย่างว่าพารามิเตอร์ คือค่าที่ใช้ส่งผ่านเข้าไปยังฟังก์ชันเวลาเรียกใช้ ใน JavaScript เราสามารถส่งผ่านค่าเข้าไปจำนวนเท่าใดก็ได้ ไม่จำเป็นต้องตรงกับจำนวนพารามิเตอร์ที่ประกาศไว้ในฟังก์ชัน !!! หากจำนวนค่าที่ส่งผ่านเข้าไปมีจำนวนน้อยกว่า พารามิเตอร์ที่เหลือจะมีค่าเป็น `undefined` ดังนั้นเราควรตรวจสอบว่าพารามิเตอร์มีค่าหรือไม่ก่อนใช้เสมอ ตัวอย่างเช่น

```
function writeLine(message){
  if(message != null){
    document.write(message);
  }
  document.write("<br/>");
}
```

ตัวอย่างข้างต้นหากเราเรียกฟังก์ชัน `writeLine()` โดยไม่ส่งผ่านค่าใดๆทั้งสิ้น ฟังก์ชันนี้จะทำหน้าที่เสมือนเป็นฟังก์ชันขึ้นบรรทัดใหม่

[Note] ในทางเทคนิคแล้ว พารามิเตอร์คือตัวแปรที่เราประกาศในฟังก์ชัน ส่วน Argument คือค่าที่ส่งเข้าไป ตัวอย่างเช่น

```
function fn(x){ . . . }
fn(5);
```

`x` คือพารามิเตอร์ ส่วน `5` คือ Argument อย่างไรก็ตามหนังสือเล่มนี้จะใช้คำว่าพารามิเตอร์เพื่อสื่อถึงทั้งสองอย่าง

ฟังก์ชันกับตัวแปร Implicit ชื่อ "`arguments`"

`arguments` เป็นตัวแปรพิเศษที่มีให้เรียกใช้ในทุกๆฟังก์ชันโดยอัตโนมัติ มันทำหน้าที่เสมือนตัวแทนของค่าทั้งหมดที่ถูกส่งผ่านมายังฟังก์ชัน ทั้งนี้เพื่อให้เราสามารถเข้าถึงค่าต่างๆที่ถูกส่งมาได้ โดยธรรมชาติแล้วตัวแปร `arguments` เป็นตัวแปรชนิด Array ดังนั้นเราสามารถใช้พร็อพเพอร์ตี้ `length` ที่มีใน Array เพื่อตรวจสอบจำนวนค่าที่ถูกส่งมายังฟังก์ชันได้ ตัวอย่างเช่น

```
function f(x,y,z){
  if(arguments.length != 3){
    document.write("Expect 3 arguments, but you passed
                    only: " + arguments.length);
    document.write("<br/>");
  }
  document.write(arguments[0] + "<br/>");
  document.write(arguments[1] + "<br/>");
  document.write(arguments[2] + "<br/>");
}
```

จากตัวอย่างข้างต้นหากเราเรียก `f(10,15)` เราจะได้ผลลัพธ์ดังนี้

```
Expect 3 arguments, but you passed only: 2
10
15
undefined
```

ดังนั้นด้วยตัวแปร `arguments` เราจึงสามารถสร้างฟังก์ชันชนิด Variable-length Argument ได้ หรือที่มักเรียกกันในภาษาอื่นว่า Varargs กล่าวคือ ฟังก์ชันที่รับค่าจำนวนเท่าใดก็ได้ โดยไม่ต้องระบุจำนวนพารามิเตอร์ที่ตายตัว ตัวอย่างเช่น

```
function average(){
  var sum = 0;
  for(i=0; i<arguments.length; i++){
    sum = sum + arguments[i];
  }
  return (sum/argments.length);
}


var avg = average(125,35,99,80);
document.write( avg );  // 84.75
```

จากตัวอย่างข้างต้นจะเห็นว่า เราไม่ได้กำหนดว่าต้องส่งค่าเข้ามาจำนวนสี่ค่า แต่จะส่งเข้ามาจำนวนเท่าใดก็ได้ มิเช่นนั้นเราจะคิดค่าเฉลี่ยกับตัวเลขจำนวนสิบค่าได้อย่างไร? โดยที่ไม่ต้องสร้างฟังก์ชันใหม่ สุดท้าย `arguments` ไม่ใช่คีย์เวิร์ด ดังนั้นหากเราสร้างตัวแปร Local ชื่อ "`arguments`" ขึ้นมา มันจะไปทับของเดิมทันที ยังจำได้ใช่ไหมครับว่า เราสามารถประกาศตัวแปรซ้ำของเดิมได้ โดยมันจะกลายเป็นการกำหนดค่าไป (เนื้อหาอยู่ในบทที่ 2 "Variables และ Datatypes")

ฟังก์ชันกับ "`typeof`" Operator

JavaScript เป็นภาษาประเภท Dynamic Typing ดังนั้นเราจึงไม่สามารถและไม่ต้อง

ประกาศชนิดของตัวแปร ถ้าเช่นนั้นหากฟังก์ชันของเราต้องการค่าตัวแปรชนิดหนึ่ง แต่ตัวเรียกฟังก์ชันส่งค่าเข้ามาอีกชนิดหนึ่ง จึงอาจทำให้เกิดปัญหาขึ้นได้ ดังนั้นเราจึงควรตรวจสอบชนิดของค่าที่ถูกส่งเข้ามาเสียก่อน โดยใช้เครื่องหมาย `typeof` เครื่องหมายนี้จะให้ผลการตรวจสอบดังต่อไปนี้

- ถ้าเป็นค่าชนิด Primitives ผลที่ได้จะเป็น `number`, `boolean`, หรือ `string`
- ถ้าเป็นค่าชนิด Objects หรือ Arrays ผลที่ได้จะเป็น `object`
- ถ้าเป็น Functions ผลที่ได้จะเป็น `function`

ตัวอย่างเช่น

```
function f(x){
  document.write( typeof x);
  document.write("<br/>");
}


f("hello");          // string
f( 123 );            // number
f( true );           // boolean
f( new Date() );     // object
f( [1,2,3] );        // object
f( function(){} );   // function
```

ค่าฟังก์ชัน (Function Literal)

ฟังก์ชันใน JavaScript เปรียบเสมือนค่า (Literal) ค่าหนึ่ง ดังนั้นฟังก์ชันที่เราสร้างขึ้น จึงสามารถใช้เป็นค่าของตัวแปรได้ ลองดูตัวอย่างก่อนนะครับ

```
var add = function(num1,num2){
            return num1 + num2;
          };
var multiply = function(num1,num2){
                 return num1 * num2;
               };
```

จากตัวอย่างข้างต้นเราประกาศตัวแปร `add` และ `multiply` และกำหนดค่าให้กับตัวแปรด้วยค่าฟังก์ชัน ลองเปรียบเทียบกับตัวอย่างข้างล่างเพิ่มเติมนะครับ

```
var x = 10;                          // number literal
var s = "Hello";                     // string literal
var f = function(x){ return x; };    // function literal
```

จะเห็นว่าส่วนที่ขีดเส้นใต้ก็คือค่าที่กำหนดให้กับตัวแปรนั่นเอง ดังนั้น Function Literal จึง

เป็นอีกวิธีการหนึ่งในการสร้างฟังก์ชัน และโปรดสังเกตุว่าการสร้างฟังก์ชันด้วยวิธีนี้ **เราไม่จำเป็นต้องตั้งชื่อฟังก์ชัน เพราะชื่อตัวแปรก็คือชื่อฟังก์ชันนั่นเอง**

ฟีเจอร์ทางภาษานี้มีประโยชน์มาก เพราะนั่นหมายถึงว่า **เราสามารถใช้ฟังก์ชันเป็นค่าที่ส่งผ่านให้กับพารามิเตอร์ได้ !!!** ผมขอแสดงตัวอย่างจากบทเรียนที่แล้ว "Arrays" อีกสักครั้งหนึ่ง

```
var array3 = [49,70,8,111];
array3.sort( function(first,second){
               return first - second;
             } );
document.write( array3.join() );  // 8,49,70,111
```

ตัวอย่างข้างต้นนี้เราส่งผ่านฟังก์ชันเป็นค่าพารามิเตอร์ให้กับเมธอด `sort()` แต่ถ้าฟังก์ชันมีความซับซ้อนมาก โค้ดก็จะยาวและทำให้อ่านยาก เราจึงสามารถเขียนขึ้นใหม่ในอีกรูปแบบหนึ่งได้ดังนี้

```
var comp1 = function(first,second){
              return first-second;
            };
var array3 = [49,70,8,111];
array3.sort( comp1 );
document.write( array3.join() ); // 8,49,70,111
```

จากตัวอย่างข้างต้นเราสร้างฟังก์ชันและกำหนดค่าให้กับตัวแปร `comp1` แล้วจึงค่อยส่ง `comp1` ไปเป็นพารามิเตอร์ให้กับเมธอด `sort()` อีกที ด้วยรูปแบบนี้โค้ดของเราก็อ่านง่ายขึ้น และยังสามารถเรียกใช้ภายหลังได้อีกหากลอจิกการเปรียบเทียบยังเหมือนเดิม ปิดท้ายหัวข้อนี้เรามาลองดูตัวอย่างที่เรายังสามารถประยุกต์ใช้ได้อีกดังนี้

```
var f1 = function(x){
           document.write(x);
         };
var f2 = f1;
f2("hello");  // hello
```

จากตัวอย่างข้างต้นจะเห็นได้ว่า เราสามารถส่งฟังก์ชันเป็นค่าได้ แล้วเราก็เรียก `f2` ได้เสมือนเป็นฟังก์ชัน นั่นเพราะตัวแปร `f2` คือตัวแปรชนิดฟังก์ชันนั่นเอง (ก็ในเมื่อมันเก็บค่าเป็นฟังก์ชัน)

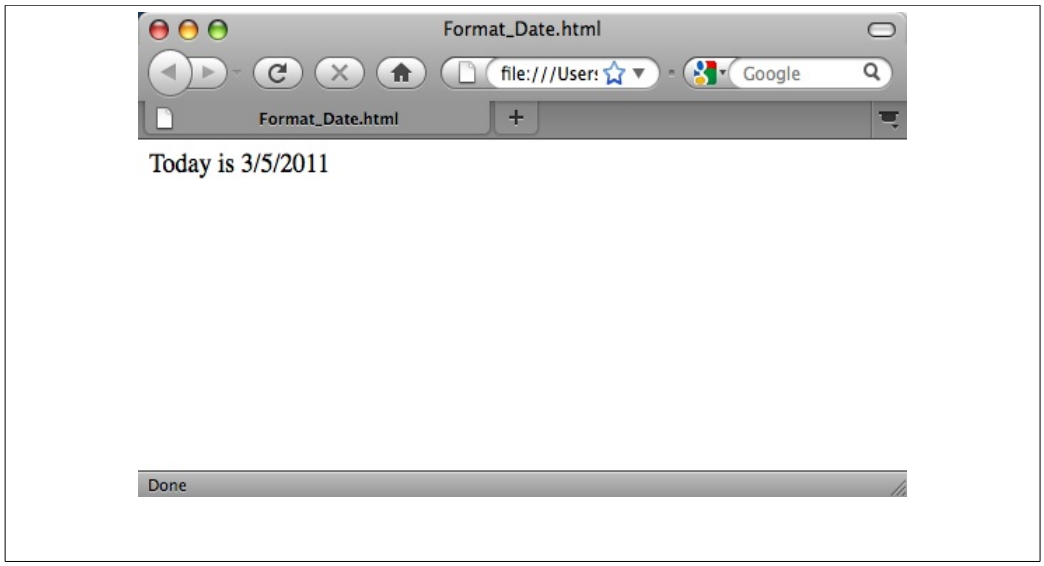
[Note] บทหน้าจะพูดถึงความแตกต่างระหว่างฟังก์ชันกับเมธอด

สร้างฟังก์ชันที่ไหนดี?

เราสามารถเขียนโค้ด JavaScript ที่ไหนก็ได้ในไฟล์ HTML อย่างไรก็ตามเรานิยมสร้างฟังก์ชันไว้ภายในแท็ก `<head>` แล้วจึงเรียกใช้มันภายในแท็ก `<body>` หรือที่ใดก็ตาม (ในกรณีที่ต้องการเรียกใช้ฟังก์ชันในไฟล์ HTML อื่นๆ เราสามารถแยกฟังก์ชันไปไว้ที่ไฟล์แยกต่างหากได้ แล้วจึงทำการอิมพอร์ตมาใช้ รายละเอียดจะกล่าวในบทที่ 8 “Modules และ Namespaces”)

จับทุกอย่างมารวมกัน

ในหัวข้อนี้เรามาปิดบทด้วยตัวอย่างรวมกัน ตัวอย่างที่ 5-1 (Format_Date.html) แสดงการสร้างฟังก์ชันจัดรูปแบบวันที่ โดยสามารถส่งผ่านเครื่องหมายที่ใช้คั่นระหว่างวัน เดือนปี เช่น จะเป็นเครื่องหมาย “/” หรือ “–” เป็นต้น ดังแสดงในรูปที่ 5-1 ในตัวอย่างนี้เราต้องบวกค่าเดือนเข้าไปอีกหนึ่ง เพราะเดือนใน JavaScript เริ่มต้นที่ 0 และปีใน JavaScript จะนับตั้งแต่ปี 1900 เป็นต้นมา ดังนั้นเราจึงต้องบวกเข้าไปอีก 1900 (IE ในบางเวอร์ชันจะมีบั๊กเรื่องตัวเลข 1900 อยู่ ซึ่งไม่เป็นไปตามสเปคของ JavaScript)



รูปที่ 5-1: ผลลัพธ์การรัน Format_Date.html บนเว็บเบราว์เซอร์

ตัวอย่างที่ 5-1: Format_Date.html

```
<html>
  <head>
    <title>Format_Date.html</title>
    <script>
      function formatDate(date,separator){
        var fDate = "";
        var d = date.getDay();
        var m = date.getMonth() + 1;
        var y = date.getYear() + 1900;
        fDate = d + separator + m + separator + y;


        return fDate;
      }
    </script>
  </head>
  <body>
    <script>
      var today = new Date();
      document.write("Today is ");
      document.write( formatDate(today, "/") );
    </script>
  </body>
</html>
```