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 2

Variables และ Datatypes

หลังจากที่ได้ทำความรู้จักกับ JavaScript มาในบทที่แล้ว บทนี้เราจะเริ่มเจาะลึกลงในรายละเอียดของตัวภาษา JavaScript โดยจะเริ่มต้นที่เรื่องของตัวแปร ทั้งนี้เพราะตัวแปรและชนิดของมันมีความสำคัญต่อการเรียนรู้ส่วนอื่นๆของภาษา JavaScript ในบทต่อๆไป ดังนั้นในบทนี้เราจะมาเรียนรู้หัวข้อต่างๆที่เกี่ยวข้องกับตัวแปรดังต่อไปนี้

- การประกาศตัวแปร
- ขอบเขตการใช้งาน
- ชนิดของตัวแปร
- ความแตกต่างระหว่างตัวแปรชนิด Primitive และ Object

ตัวแปร (Variable) ใช้สำหรับเก็บข้อมูล

ตัวแปรมีไว้สำหรับเก็บข้อมูลหรือเก็บค่า (Value) แต่ก่อนเราจะใช้ตัวแปรได้ เราต้องประกาศมันขึ้นมาเสียก่อน การประกาศตัวแปรและการกำหนดค่าตั้งต้นมี Syntax ดังนี้

```
var var_name = value;
```

ตัวอย่างเช่น

```
var price = 500;
var message = "Welcome";  // string value is enclosed in
                             quotes
```

คีย์เวิร์ด `var` ใช้สำหรับการประกาศตัวแปร และการประกาศตัวแปรจะทำเพียงครั้งเดียว (สำหรับชื่อเดิม) แต่จะเรียกใช้กี่ครั้งก็ได้ เช่น

```
var price = 500;
price = price * 1.05;  // price is up 5%
```

และเราสามารถประกาศตัวแปรมากกว่าหนึ่งตัวด้วยคีย์เวิร์ด var ตัวเดียวกันก็ได้ ตัวอย่างเช่น

```
var x,y;
```

```
var num1=10, num2=15;
```
นอกจากนี้คีย์เวิร์ด `var` ยังละได้ใน JavaScript อย่างไรก็ตามผมแนะนำว่าให้เขียนไว้เสมอ (เหตุผลจะกล่าวในลำดับถัดไป) ส่วนการตั้งชื่อมีกฎดังต่อไปนี้

- ให้เริ่มด้วยตัวอักษร หรือเครื่องหมาย Underscore "_" หรือเครื่องหมาย Dollar Sign "$"
- ตัวถัดไปสามารถเป็นตัวเลขได้ (สรุปคือตัวเลขห้ามใช้ขึ้นต้นชื่อนั่นเอง)
- ห้ามมีช่องว่างในชื่อเด็ดขาด
- เป็น Case-sensitive นั่นคือตัวอักษรพิมพ์เล็กหรือพิมพ์ใหญ่ถือว่าแตกต่างกัน เช่น `Price` กับ `price` ถือว่าคนละตัวกัน
- ห้ามใช้ Reserved Words หรือคำสงวนมาตั้งชื่อ โดยมีทั้งหมด 28 ตัวดังต่อไปนี้

```
break
case, catch, continue
default, delete, do
else
false, finally, for, function
if, in, instanceof
new, null
return
switch
this, throw, true, try, typeof
var, void
while, with
```

กฎการตั้งชื่อนี้นอกจากจะใช้กับตัวแปรแล้ว ยังใช้กับการตั้งชื่ออื่นๆใน JavaScript ด้วย เช่น ชื่อฟังก์ชัน เป็นต้น

เพิ่มเติมเกี่ยวกับการประกาศตัวแปร

ตัวแปรใน JavaScript เป็นประเภท Untyped กล่าวคือตัวแปรใน JavaScript สามารถเก็บข้อมูลชนิดใดก็ได้ เพราะไม่มีการระบุชนิด อีกทั้งยังสามารถเปลี่ยนแปลงชนิดของข้อมูลที่จัดเก็บได้อีกด้วย ตัวอย่างเช่น

```
var price = 500;
price = "five hundred";
```

จะเห็นได้ว่าตัวแปร `price` ถูกประกาศขึ้นมาและเก็บค่าชนิดตัวเลข (`number`) แต่ในบรรทัดถัดไปเราได้กำหนดค่าชนิดใหม่ชนิดตัวอักษร (`string`) ให้กับตัวแปร `price` ซึ่งก็

ทำได้ใน JavaScript และนี่คือเหตุผลที่ว่าทำไม JavaScript ถึงจัดได้ว่าเป็นภาษาประเภท Dynamic Typing ในภาษาประเภท Static Typing อาทิเช่น Java ทำไม่ได้

ใน JavaScript การประกาศตัวแปรชื่อซ้ำกันด้วยคีย์เวิร์ด var ถือว่า**ไม่ผิด** และถือว่าเป็นตัวเดียวกัน ดังนั้นหากมีการกำหนดค่า ค่าที่กำหนดใหม่จึงทับค่าเก่าทันที ตัวอย่างเช่น

```
var num = 15;
var num = 25;
document.write( num );  // 25
```

ดังนั้นต้องระวังให้ดีนะครับ เราอาจไม่รู้หรือลืมไปว่าได้ประกาศตัวแปรชื่อนี้ไปแล้ว (โดยเฉพาะคนที่ถนัดภาษาประเภท Static มาก่อน เพราะในภาษาประเภท Static มันจะฟ้องเราตอนคอมไฟล์)

ที่นี้มาดูเรื่องของการใช้ตัวแปรที่ประกาศแล้วแต่ยังไม่กำหนดค่า กับการใช้ตัวแปรที่ยังไม่ได้ประกาศบ้าง ถ้ามีการเรียกใช้ตัวแปรที่ประกาศแล้วแต่ยังไม่ได้มีการกำหนดค่าให้ค่าที่ได้จะเป็น `undefined` ตัวอย่างเช่น

```
var myName;
document.write( myName );  // undefined
```

แต่หากเราพยายามจะใช้งานตัวแปรที่ยังไม่มีตัวตนหรือยังไม่ได้ประกาศ จะก่อให้เกิด Error ขึ้นและโปรแกรมจะหยุดทำงานทันที ตัวอย่างเช่น

```
document.write( nonExist );
document.write("Can you see this message?");
```

ในตัวอย่างข้างต้น เราจะไม่เห็นข้อความ “Can you see . . .” เพราะโปรแกรมหยุดทำงานเสียก่อน ดังนั้น Undefined กับ Undeclare ไม่เหมือนกันนะครับ

ขอบเขตการใช้งานของตัวแปร (Variable Scopes)

Variable Scopes คือขอบเขตการใช้งานของตัวแปร ซึ่งมีอยู่ด้วยกับสอง Scopes ดังนี้

- **Global** ตัวแปร Global คือตัวแปรที่ถูกประกาศนอกฟังก์ชัน ซึ่งเราจะเรียกใช้มันที่ใดก็ได้ แม้แต่เรียกใช้ในฟังก์ชัน
- **Local** ตัวแปร Local คือตัวแปรที่ถูกประกาศภายในฟังก์ชัน ซึ่งเราจะเรียกใช้มันได้เพียงแค่ภายในฟังก์ชันที่มันถูกประกาศขึ้นเท่านั้น และหากมีการประกาศตัวแปร Global และ Local ชื่อเดียวกัน และเรียกใช้ชื่อนั้นในฟังก์ชัน จะถือว่าเรียกใช้ตัวแปร Local ตัวอย่างเช่น

```
var globalScope = "Global";
var x = 10;
function testScope(){
  var localScope = "Local";
  document.write( globalScope );  // Global
  document.write( "<br/>");
  document.write( localScope );   // Local
  var x = 15;
  document.write( x );            // 15
}
testScope();
document.write( x );              // 10
```

ในการประกาศตัวแปร Local หากเราละคีย์เวิร์ด `var` แล้วละก็ มันอาจกลายเป็นการกำหนดค่าให้กับตัวแปร Global ก็ได้ ตัวอย่างเช่น

```
var num = 100;
function set(){
  num = 0;
}
set();
document.write( num );  // 0 (Not 100)
```

ในตัวอย่างข้างต้นเราตั้งใจจะประกาศตัวแปร Local ชื่อ `num` ในฟังก์ชัน `set()` และได้ละคีย์เวิร์ด `var` ไว้ซึ่งก็ไม่ผิดกฎอะไร แต่เนื่องจากมีการประกาศตัวแปร Global ชื่อ `num` อยู่ก่อนแล้ว จึงกลายเป็นการกำหนดค่าให้กับตัวแปร Global แทน ดังนั้นผมแนะนำว่าให้ใส่คีย์เวิร์ด `var` ทุกครั้งที่ต้องการประกาศตัวแปรใหม่ เพื่อให้เกิดความเคยชิน

นอกจากนี้การประกาศตัวแปรใน JavaScript ไม่มี Scope แบบ Block-level (เช่น บล็อกของการทำเงื่อนไข, บล็อกของการวนลูป เป็นต้น) ตัวอย่างเช่น

```
function test(x){
  if(x >= 0){
    var y = "Positive integer"
  }else{
    var y = "Negative integer";
  }
  document.write( y );
}
test( 100 );  // Positive integer
```

ในตัวอย่างข้างต้นเราสามารถเรียกใช้ตัวแปร `y` ภายนอกบล็อกของ `if-else` ได้ !!!

สุดท้ายเรามาดูเรื่องของลำดับการเขียนที่เกี่ยวข้องกับตัวแปร Local กัน เรื่องนี้อาจจะยากสักหน่อย ถ้าไม่สามารถเข้าใจได้ ก็สามารถข้ามมันไปก่อนได้ ตัวแปร Local เป็น

ตัวแปรที่จะถูกเตรียมไว้ก่อนเสมอ ก่อนการรันแต่ละ Statement ในฟังก์ชัน ตัวอย่างเช่น

```
var x = 0;
function test(){
  document.write( x );  // undefined (not 0)
  var x = 10;
  document.write("<br/>");
  document.write( x ); // 10
}
test();
```

ในตัวอย่างข้างต้นตัวแปร `x` ในบรรทัดแรกของฟังก์ชัน `test()` คือตัวแปร Local ***ไม่ใช่ Global*** ทั้งนี้เพราะ JavaScript จะสร้างตัวแปร Local รอไว้ก่อน โดยการตรวจว่าฟังก์ชันนั้นมีตัวแปร Local อะไรบ้าง แต่จะมีการกำหนดค่าให้ก็ต่อเมื่อรันจนถึงบรรทัดนั้นๆ ทำให้ในตัวอย่างข้างต้นนี้เกิดการใช้งานตัวแปร `x` ก่อนที่มันจะได้รับค่า (แต่ `x` มีตัวตนแล้ว) อย่างไรก็ตามในทางปฏิบัติคงไม่ค่อยได้พบปัญหานี้ เพราะเราจะประกาศก่อนเรียกใช้งานอยู่แล้ว (ก็ในเมื่อตั้งใจจะเรียกใช้ตัวแปร Local ในตัวอย่างนี้ ก็ควรเขียนการประกาศตัวแปรก่อน)

[Note] อันที่จริงแล้วเราสามารถใช้ตัวแปร Local ภายนอกฟังก์ชันได้ ด้วยเทคนิคการทำ Closure แต่มันเกินความจำเป็นต่อการใช้งานทั่วไป จึงขอละไว้

[Note] ตัวแปร Global ที่ประกาศในแท็ก `<script>` ชุดหนึ่ง สามารถถูกเรียกใช้ในแท็ก `<script>` ชุดอื่นๆได้

ชนิดของตัวแปร (Datatypes)

ตัวแปรมีไว้สำหรับเก็บข้อมูลหรือเก็บค่า แต่ว่าค่าชนิดไหนบ้างหล่ะที่สามารถเก็บลงไปในตัวแปรได้? อันนี้ก็ต้องดูที่ Datatypes ที่ JavaScript รองรับ Datatypes ใน JavaScript แบ่งได้เป็นสองกลุ่มด้วยกันคือ

- **Primitive types** ซึ่งประกอบไปด้วย Numbers, Strings, และ Booleans
- **Reference types** ซึ่งประกอบไปด้วย Objects, Arrays, และ Functions

ตัวแปรชนิด Primitive

ตัวแปรชนิด Primitive คือตัวแปรที่เก็บค่าดังต่อไปนี้ ค่าตัวเลข (Numbers), ค่าตัวอักษร (Strings), และค่าจริงเท็จ (Booleans)

ตัวแปรที่เก็บค่าตัวเลขคือตัวแปรชนิด `number` โดยตัวเลขสามารถเป็นได้ทั้งเลขจำนวนเต็ม (Integer) และเลขที่มีจุดทศนิยม (floating-point) และจะเป็นค่าบวกหรือค่าลบก็ได้ ตัวอย่างเช่น

```
var num = 10;
var price = 589.50;
var xPosition = -5;
var yPosition = -8;
```

ตัวแปรข้างต้นทั้งหมดจะมีชนิดเป็น `number` เพราะได้รับการกำหนดค่าเป็นชนิดตัวเลข ต้องไม่ลืมว่า JavaScript เป็นภาษาประเภท Dynamic Typing จึงไม่มีการกำหนดชนิดเจาะจงให้กับตัวแปร แต่จะดูจากค่าที่ใส่ให้กับตัวแปร นอกจากนี้หากเราต้องการกำหนดค่าตัวเลขในรูปของเลขฐานแปดให้กับตัวแปร เราสามารถทำได้โดยใส่ “`0`” นำหน้าค่า และถ้าเป็นเลขฐานสิบหกก็ให้ใส่ “`0x`” หรือ “`0X`” นำหน้าค่า ตัวอย่างเช่น

```
var num1 = 031;   // 25
var num2 = 0x19;  // 25
```

ตัวแปรที่เก็บค่าจริงหรือเท็จคือตัวแปรชนิด `boolean` ในการกำหนดค่าให้กับตัวแปรชนิด `boolean` นั้น ให้ใช้ `true` หรือ `false` โดยไม่ต้องมีเครื่องหมายคำพูด (“”) หรือ (`''`) ใดๆทั้งสิ้น ตัวอย่างเช่น

```
var result = true;
```

ตัวแปรที่เก็บค่าตัวอักษรคือตัวแปรชนิด `string` ในการกำหนดค่าให้กับตัวแปรชนิด `string` จะต้องกำหนดอยู่ในเครื่องหมายคำพูด (“”) หรือ (`''`) เสมอ ตัวอย่างเช่น

```
var message = “Welcome”;
var message = 'Welcome';
```

หากการกำหนดค่าประกอบไปด้วยเครื่องหมายพิเศษที่ไม่สามารถกำหนดได้ เช่น เครื่องหมายคำพูด “ หรือ ' เพราะอาจไปขัดแย้งกับตัวเปิดปิด `string`, หรือที่ไม่มีในคีย์บอร์ด เป็นต้น เราจำเป็นต้องใช้สัญลักษณ์ Escape Sequence “`\`” เพื่อช่วยกำหนดดังแสดงต่อไปนี้

`\“`	แทนเครื่องหมายคำพูด “
`\'`	แทนเครื่องหมายคำพูด '

\\	แทนตัว Backslash \ เอง
\u*XXX*	*XXX* คือรหัส Unicode ใช้แสดงตัวอักษรที่ไม่มีในคีย์บอร์ด เช่น \u00a9 แทน ©

นอกจากนี้ตัวแปร `string` เมื่อนำมาใช้กับเครื่องหมายบวก “+” เครื่องหมายนี้จะกลายเป็นเครื่องหมาย Concatenation ทันที นั่นหมายความว่าเป็นการนำตัวอักษรมาแปะต่อท้ายกัน ตัวอย่างเช่น

```
var message = "Hello " + "World";  // Hello World
var message = "Hello" + 1;         // Hello1
var message = "Hello" + 1 + 1;     // Hello11
var message = 1 + 1 + "Hello";     // 2Hello
```

ถ้ามีข้างใดข้างหนึ่งของเครื่องหมายบวก “+” เป็นค่า `string` เครื่องหมายบวก “+” นี้จะกลายเป็นเครื่องหมาย Concatenation ทันที

สุดท้ายเราลองมาดูภาพจำลองการจัดเก็บค่า Primitives ในเมมโมรีกัน หากมีการประกาศตัวแปรดังต่อไปนี้

```
var num = 10;
```

เราสามารถจำลองรูปได้ดังนี้

อีกตัวอย่างเช่น

```
var result = true;
```

เราสามารถจำลองรูปได้ดังนี้

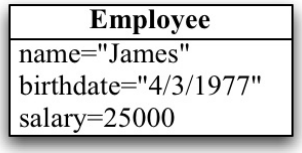
สรุปก็คือตัวแปรชนิด Primitive เก็บค่า Primitive ที่ตัวมันเองเลย (ซึ่งเดี๋ยวจะเห็นว่ามันต่างจากพวกตัวแปรชนิด Reference นะครับ)

[Note] อันที่จริงแล้วค่า Primitives จะถูกเก็บอยู่ในรูปของตัวเลข 01 หรือในรูปของเลขฐานสอง รูปข้างต้นใช้เพียงเพื่อแสดงให้ง่ายต่อความเข้าใจ

ตัวแปรชนิด Reference

ตัวแปรชนิด Reference คือตัวแปรที่เก็บ**ค่าอ้างอิง**ของ Objects, Arrays, และ Functions เราอาจเรียกตัวแปรชนิด Reference ว่าตัวแปรชนิดออบเจกต์ก็ได้ และด้วยเหตุนี้ Arrays และ Functions ใน JavaScript จึงจัดว่าเป็นออบเจกต์ชนิดหนึ่ง

แล้วออบเจกต์คืออะไร? ออบเจกต์คือการรวมกลุ่มของพร็อพเพอร์ตี้ (Property) ที่เกี่ยวข้องกัน เพื่อให้ง่ายต่อการเรียกใช้งาน เช่น ออบเจกต์ `Employee` ประกอบไปด้วยพร็อพเพอร์ตี้ `name`, `birthdate`, และ `salary` ดังแสดงในรูปข้างใต้



การใช้งานในรูปของออบเจกต์ ทำให้การเก็บค่าข้อมูลทำได้ง่ายกว่าการใช้ในรูปของตัวแปร Primitives ที่แยกขาดจากกัน ลองจินตนาการดูว่า ถ้าต้องเก็บข้อมูลพนักงานมากกว่า 50 คน โดยไม่เก็บในรูปของออบเจกต์ เราจำเป็นต้องใช้ตัวแปรจำนวนมาก เพราะเราต้องมีตัวแปรสำหรับแต่ละพร็อพเพอร์ตี้ เช่น `name1`, `name2`, `. . .`, `birthdate1`, `birthdate2`, `. . .`, `salary1`, `salary2`, `...` เป็นต้น นอกจากนี้ออบเจกต์ยังประกอบไปด้วยเมธอดได้อีกด้วย ซึ่งเมธอดก็คือฟังก์ชันที่เกี่ยวข้องกับออบเจกต์นั้นๆ เช่น ออบเจกต์ `Document` มีเมธอด `write()` อยู่, หรือสำหรับออบเจกต์ `Employee` ก็น่าจะมีเมธอด `calculateAge()` อยู่ เป็นต้น

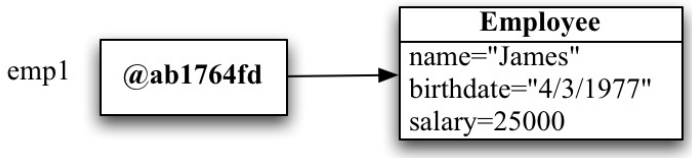
ตัวแปรชนิดออบเจกต์ใช้สำหรับเก็บตัวอ้างอิงออบเจกต์อีกทีหนึ่ง กล่าวอีกอย่างคือตัวแปรชนิดออบเจกต์ใช้สำหรับเก็บที่อยู่ในเมมโมรีของออบเจกต์ ยกตัวอย่างเช่น หากเราเขียนโค้ดดังต่อไปนี้

```
var emp1 = new Employee("James","4/3/1977",25000);
```

หรือ

```
var emp1 = {name:"James",birthdate:"4/3/1977",
            salary:25000);
```

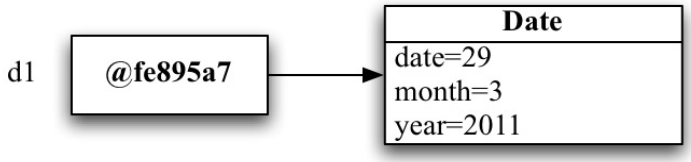
เราสามารถจำลองรูปในเมมโมรีได้ดังนี้



จะเห็นได้ว่าตัวแปร `emp1` ไม่ได้เก็บออบเจกต์ `Employee` เองเลย หากแต่เก็บที่อยู่ของออบเจกต์ `Employee` ที่ถูกโหลดด้วยคำสั่ง `new` อีกที (เราจึงพอจะจินตนาการได้ว่า `emp1` ชี้ไปยังออบเจกต์ `Employee` แต่อันที่จริงแล้ว Reference ไม่เหมือนกับ Pointer ซะทีเดียวนะครับ) ลองดูอีกซักตัวอย่างหนึ่งแล้วกัน หากเราเขียนโค้ดดังต่อไปนี้

```
var d1 = new Date(2011,3,29);
```

เราก็จะได้รูปจำลองประมาณนี้



(ฟอร์แมตของวันที่จะเป็นปี, เดือน, วัน และเดือนจะเริ่มที่ 0 ถึง 11) นอกจากนี้ออบเจกต์ยังแบ่งออกเป็นอีกสองกลุ่มหลักๆคือ

- **Built-in Objects** คือออบเจกต์สำเร็จรูปที่มากับภาษา JavaScript และ Client-side JavaScript เช่น ออบเจกต์ `Date`, `Math`, `RegExp`, `Document`, `Window` เป็นต้น
- **Custom Objects** คือออบเจกต์ที่เราสร้างขึ้นมาเอง เช่น ออบเจกต์ `Employee`, `Customer`, `Product` เป็นต้น

ส่วนรายละเอียดของออบเจกต์จะกล่าวอย่างละเอียดอีกครั้งในบทที่ 6 “Objects และ Classes” ในที่นี้ให้เข้าใจเรื่องความแตกต่างระหว่างตัวแปรชนิด Reference กับตัวแปรชนิด Primitive ก่อน

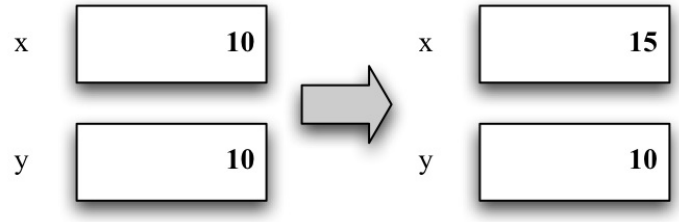
Primitives VS Objects

เพื่อให้เข้าใจความแตกต่างระหว่างตัวแปรชนิด Primitive กับตัวแปรชนิด Reference

เราลองดูตัวอย่างต่อไปนี้

```
var x = 10;
var y = x;
x = 15;
document.write( y );  // 10
```

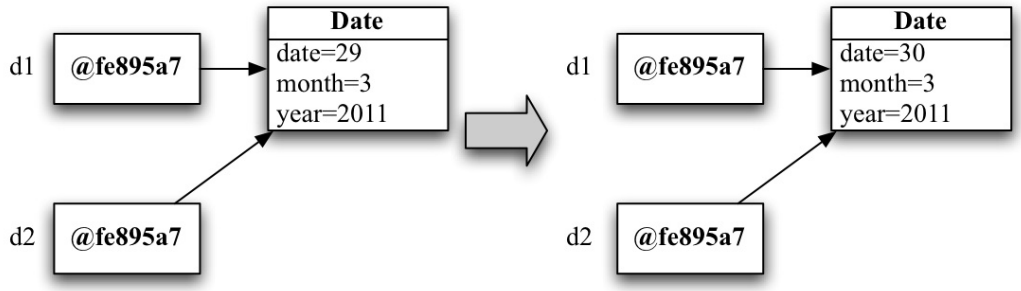
สาเหตุที่ผลลัพธ์เท่ากับ 10 เพราะ ทั้ง x และ y ต่างก็เก็บค่า 10 ของใครของมันดังแสดงในรูปข้างใต้ (y = x หมายความว่าให้ y เก็บค่าก๊อปปี้เดียวกันกับที่ x เก็บอยู่) ดังนั้นการเปลี่ยนค่าในตัวแปร x จึงไม่กระทบกระเทือนตัวแปร y



เอาหล่ะทีนี้มาดูตัวอย่างตัวแปรชนิด Reference กันบ้าง จากตัวอย่างต่อไปนี้

```
var d1 = new Date(2011,3,29);
var d2 = d1;
d1.setDate( 30 );
document.write( d2 );  // 30
```

จะเห็นได้ว่าวันที่ที่ตัวแปร d1 และ d2 อ้างอิงอยู่ เปลี่ยนแปลงเหมือนกัน ทั้งนี้เพราะเรามีออบเจกต์เพียงแค่ออบเจกต์เดียว คือ new Date() เพียงแค่ครั้งเดียว แต่มีตัวแปรอ้างอิงถึงถึงสองตัวคือ d1 และ d2 ดังแสดงในรูปข้างใต้



สรุปคือตัวแปรชนิด Primitive จะเก็บค่าที่ตัวมันเอง ในขณะที่ตัวแปรชนิด Reference จะเก็บที่อยู่ของออบเจกต์อีกที

Wrapper Objects

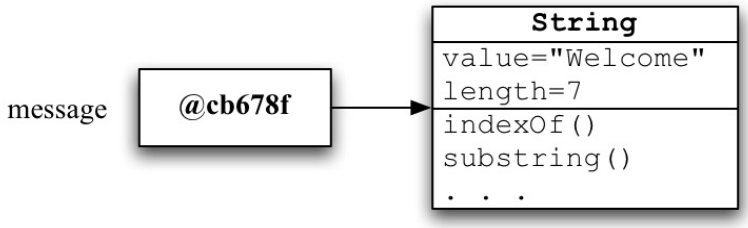
Wrapper Objects คือออบเจกต์สำเร็จรูป (Built-in) ที่เป็นตัวแทนของค่า Primitives หรือเป็นออบเจกต์ที่เก็บค่า Primitives อีกทีหนึ่ง ซึ่งก็ได้แก่ออบเจกต์ `Number`, `String`, และ `Boolean` (จะขึ้นต้นชื่อด้วยตัวอักษรพิมพ์ใหญ่) แล้วทำไมเราจะต้องใช้ Wrapper Objects ด้วย ในเมื่อมีตัวแปรชนิด Primitive อยู่แล้ว ทั้งนี้เพราะ Wrapper Objects ประกอบไปด้วยพร็อพเพอร์ตี้และเมธอดที่จำเป็น ยกตัวอย่างเช่น หากเราใช้ตัวแปร `string` ที่เป็น Primitive เราก็ได้แค่เก็บค่าตัวอักษรเท่านั้น ดังนี้

```
var message = "Welcome";
```

แต่หากเราใช้ตัวแปรที่เป็นออบเจกต์ชนิด `String` แล้ว เราสามารถตรวจสอบจำนวนตัวอักษรทั้งหมดได้ด้วยพร็อพเพอร์ตี้ `length` ดังนี้

```
var message = new String("Welcome");
document.write( message.length );  // 7
```

รูปข้างใต้แสดงการภาพจำลอง หากใช้ตัวแปรในรูปของออบเจกต์ `String`



แม้ว่า Wrapper Objects จะประกอบไปด้วย `Nubmer`, `String`, และ `Boolean` ทว่าที่เราใช้กันจริงๆก็คือ `String` ดังนั้นเรามาดูพร็อพเพอร์ตี้และเมธอดของออบเจกต์ `String` ที่น่าสนใจกัน ซึ่งมีดังต่อไปนี้

พร็อพเพอร์ตี้ `length`

ใช้สำหรับเก็บค่าจำนวนตัวอักษรทั้งหมดรวมช่องว่างด้วย ดังที่แสดงในตัวอย่างข้างต้น

เมธอด `substring(`*`from`*`:int,`*`to`*`:int): string`

รีเทิร์น `string` ใหม่ที่เกิดจากช่วงที่กำหนด คือเริ่มจาก *`from`* ไปจนถึง *`to`* (ไม่

รวมตัว *to* เอง) และ Index จะเริ่มที่ 0 เสมอ โดยตัวอักษรเก่ายังคงเดิม ยกตัวอย่างเช่น

```
var message = new String("Welcome");
var newMsg = message.substring(0,3);
document.write( newMsg );
// "Wel"
    012
```

เมธอด indexOf(*str*:string): int

รีเทิร์นค่าตำแหน่งแรกของการค้นหาคำ หากไม่พบจะรีเทิร์นค่า -1 ตัวอย่างเช่น

```
var message = "Welcome";
document.write( message.indexOf("W") );  // 0
```

เมธอด lastIndexOf(*str*:string): int

รีเทิร์นค่าตำแหน่งแรกของการค้นหาคำ โดยเป็นการค้นหาจากข้างท้าย หากไม่พบจะรีเทิร์นค่า -1 ตัวอย่างเช่น

```
var message = "Welcome";
document.write( message.lastIndexOf("e") );  // 6
```

เมธอด toLowerCase(): string

รีเทิร์น `string` ใหม่ที่ถูกเปลี่ยนให้เป็นตัวอักษรพิมพ์เล็กทั้งหมด ตัวอย่างเช่น

```
var message = "Welcome";
document.write( message.toLowerCase() );  // welcome
```

เมธอด toUpperCase(): string

รีเทิร์น string ใหม่ที่ถูกเปลี่ยนให้เป็นตัวอักษรพิมพ์ใหญ่ทั้งหมด ตัวอย่างเช่น

```
var message = "Welcome";
document.write( message.toUpperCase() );  // WELCOME
```

[Note] ใน JavaScript ไม่มี Datatype ชนิด `int` มันถูกใช้สำหรับสื่อให้เห็นว่าเป็นเลขจำนวนเต็มเท่านั้น

[Note] การค้นหาในเมธอด `indexOf()` และ `lastIndexOf()` เป็นแบบ Case-sensitive

JavaScript นั้นเป็นภาษาที่ยืดหยุ่นมาก ถ้าเราใช้ค่า Primitives ในสถานการณ์ที่ควรจะเป็นออบเจกต์ JavaScript จะทำการเปลี่ยนเป็น Wrapper Objects ให้ทันที :) ลองดูจากตัวอย่างดังต่อไปนี้นะครับ

```
var message = "Welcome";
document.write( message.length );  // 7
document.write("Welcome".length);  // 7
```

จากตัวอย่างข้างต้นจะเห็นได้ว่า เราสร้างตัวแปรในลักษณะของ Primitive แต่สามารถใช้งานมันได้เสมือนเป็นออบเจกต์ และด้วยเหตุนี้หลายคนจึงไม่สร้าง Wrapper Objects ขึ้นโดยตรง เพราะเมื่อใดอยากใช้ JavaScript ก็แปลงให้โดยอัตโนมัติ

สุดท้ายหากคุณยังสงสัยหรือไม่แน่ใจว่าตัวแปรตัวไหนเป็นชนิด Primitive ตัวไหนเป็นชนิด Reference/Object ก็ลองตรวจสอบดูได้ด้วยคำสั่ง `typeof` ตัวอย่างเช่น

```
var str1 = new String("hello");
var str2 = "hello";
document.write( typeof str1 );  // object
document.write( typeof str2 );  // string
```

จับทุกอย่างมารวมกัน

ทั้งหมดที่ผ่านมาคือเรื่องของตัวแปรและชนิด ในหัวข้อสุดท้ายนี้เรามาปิดบทด้วยตัวอย่างรวมจากสิ่งที่เราได้เรียนมากัน ตัวอย่างที่ 2-1 (Primitives_Example.html) แสดงการใช้งานตัวแปรชนิด Primitive ซึ่งผลลัพธ์การรันจะแสดงอยู่ในรูปที่ 2-1

รูปที่ 2-1: ผลลัพธ์การรัน Primitives_Example.html บนเว็บเบราว์เซอร์

ตัวอย่างที่ 2-1: Primitives_Example.html

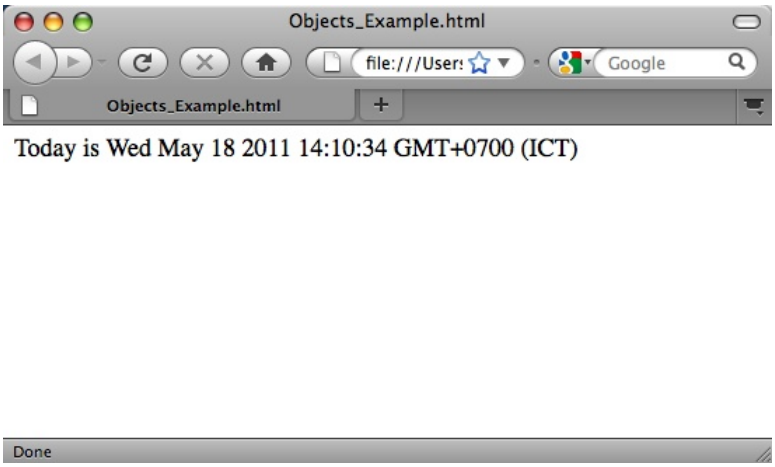
```
<html>
  <head>
    <title>Primitives_Example.html</title>
  </head>
  <body>
    <script>
      var num1 = 10;
      var num2 = 15;
      var num3 = num1 + num2;
      document.write("num3: " + num3 + "<br/>");


      var bool = true;
      document.write("bool: " + bool+ "<br/>");


      var str = "hello";
      document.write("str is a type of: " + typeof str);
    </script>
  </body>
</html>
```

ในตัวอย่างข้างต้นเราใช้ “
” เพื่อให้ขึ้นบรรทัดใหม่ เพราะนี่คือ Syntax ของ HTML และคำสั่ง typeof ใช้เพื่อตรวจสอบชนิดของตัวแปร ตัวอย่างถัดไปตัวอย่างที่ 2-2

(Objects_Example.html) แสดงการใช้งานตัวแปรชนิด Reference ซึ่งผลลัพธ์การรันจะแสดงอยู่ในรูปที่ 2-2



รูปที่ 2-2: ผลลัพธ์การรัน Objects_Example.html บนเว็บเบราว์เซอร์

ตัวอย่างที่ 2-2: Objects_Example.html

```
<html>
  <head>
    <title>Objects_Example.html</title>
  </head>
  <body>
    <script>
      var now = new Date();
      document.write("Today is " + now);
    </script>
  </body>
</html>
```