



Software Park Thailand  
</Code Camp>

# JAVASCRIPT

---

Nuttachai Kulthammanit

Slide Design By Patteera Banlangthammas

# หัวข้อ



Software Park Thailand  
</Code Camp>

- การเขียน Code
- หลักการเขียน Comment
- Babel
- Object

- Garbage Collection
- Methods ของ Object
- Constructor กับ New
- Methods ของ Primitive



Software Park Thailand  
</Code Camp>

# การเขียน Code

# 1. การเขียน Code

## 1.1. การเขียน Code ที่ดี

- สะอาด
- ง่ายต่อการอ่าน
- ไม่ได้เป็นกฎเกณฑ์บังคับ

```
function pow(x, n) {  
  let result = 1;  
  
  for (let i = 0; i < n; i++) {  
    result *= x;  
  }  
  
  return result;  
}  
  
let x = prompt("x?", "");  
let n = prompt("n?", "");  
  
if (n < 0) {  
  alert(`Power ${n} is not supported,  
  please enter a non-negative integer number`);  
} else {  
  alert( pow(x, n) );  
}
```

มีช่องว่างระหว่าง parameters

ไม่ต้องเว้นระหว่างข้อฟังก์ชันกับวงเล็บ รวมใส่ parameters ด้วย

ปีกกาอยู่บรรทัดเดียวกัน หลังจากเว้นช่องว่างแล้ว

เว้น 2 spaces

เว้นช่องว่างรอบ ๆ operators ด้วย

เว้นช่องหลังจาก for, if, หรือ while

ใส่ semi-colon ด้วยเสมอ

ใส่ช่องว่างระหว่าง parameters

แต่ละบรรทัดไม่ควรยาวเกินไป

เว้นบรรทัดสำหรับแต่ละ logical blocks

} else { ไม่ขึ้นบรรทัดใหม่

มีช่องว่างเมื่อมีฟังก์ชันซ้อน

# 1. การเขียน Code

## 1.2. Curly Braces ( วงเล็บปีกกา )

- ตัวอย่างที่ 2
- ไม่ควรขึ้นบรรทัดใหม่ แบบไม่มีปีกกาครอบ

```
if (n < 0)  
    alert(`Power ${n} is not supported`);
```

# 1. การเขียน Code

## 1.2. Curly Braces ( วงเล็บปีกกา )

- ตัวอย่างที่ 3
- เป็นตัวอย่างที่สามารถใช้ได้ถ้าเป็นโค้ดสั้นๆ

```
if (n < 0) alert(`Power ${n} is not supported`);
```



# 1. การเขียน Code

## 1.2. Curly Braces ( วงเล็บปีกกา )

- ตัวอย่างที่ 4
- เป็นตัวอย่างที่เหมาะสมกับการใช้งานมากที่สุด

```
if (n < 0) {  
    alert(`Power ${n} is not supported`);  
}
```



# 1. การเขียน Code

## 1.3. Line Length ( ความยาวของบรรทัด )

- ตัวอย่างที่ 1
- สามารถขึ้นบรรทัดใหม่เมื่อบรรทัดยาวเกินไป

```
let str = `
    ECMA International's TC39 is a group of JavaScript developers,
    implementers, academics, and more, collaborating with the community
    to maintain and evolve the definition of JavaScript.
`;
```



# 1. การเขียน Code

## 1.2. Curly Braces ( วงเล็บปีกกา )

- ตัวอย่างที่ 1
- ถ้า if นั้นมี บรรทัดเดียว ไม่ต้องใส่ปีกกา

```
if (n < 0) {alert(`Power ${n} is not supported`);}
```

# 1. การเขียน Code

## 1.3. Line Length ( ความยาวของบรรทัด )

- ตัวอย่างที่ 2
- ในส่วนของ if
- บรรทัดนี้ ไม่ควรเกิน 80-120 ตัวอักษร

```
if (  
    id === 123 &&  
    moonPhase === 'Waning Gibbous' &&  
    zodiacSign === 'Libra'  
) {  
    letTheSorceryBegin();  
}
```

# 1. การเขียน Code

## 1.4. Indents ( ย่อหน้า เว้นบรรทัด )

- 1.4.1 ย่อหน้าหรือเว้นวรรคระหว่างตัวอักษร 2 หรือ 4 spaces
- สามารถ กดปุ่ม Tab ได้
- โดยที่ ปุ่ม spaces จะมี ความสามารถหลากหลายกว่า Tab

```
show(parameters,  
    aligned,  
    one,  
    after,  
    another  
)
```

# 1. การเขียน Code

## 1.5. Semicolons ( อัฒภาค )

- ควรมีเครื่องหมายอัฒภาคอยู่หลังแต่ละคำสั่งแม้ว่าจะสามารถข้ามไปได้ก็ตามเพื่อ ลดปัญหา Error

# 1. การเขียน Code

## 1.6. Nesting Levels ( การเขียนซ้อน )

- พยายามหลีกเลี่ยงการเขียนโค้ดซ้อนกันมากเกินไป
- ตัวอย่างเช่นในวงวน( loop ) เราควร ใช้คำสั่ง continue เพื่อลดความซับซ้อน

# 1. การเขียน Code

## 1.6. Nesting Levels ( การเขียนซ้อน )

- การแทนที่จะเพิ่มความซับซ้อนในเงื่อนไข if

```
for (let i = 0; i < 10; i++) {  
  if (cond) {  
    ... // <- เพิ่ม nesting level  
  }  
}
```

# 1. การเขียน Code

## 1.6. Nesting Levels ( การเขียนซ้อน )

- เราสามารถเขียนได้ดังนี้

```
for (let i = 0; i < 10; i++) {  
  if (!cond) continue;  
  ... // <- ไม่มี nesting level เพิ่ม  
}
```

# 1. การเขียน Code

## 1.6. Nesting Levels ( การเขียนซ้อน ) - ตัวอย่างที่ 1

- เราสามารถใช้ได้กับ if/else และ return

```
function pow(x, n) {  
  if (n < 0) {  
    alert("Negative 'n' not supported");  
  } else {  
    let result = 1;  
    for (let i = 0; i < n; i++) {  
      result *= x;  
    }  
    return result;  
  }  
}
```



# 1. การเขียน Code

## 1.6. Nesting Levels ( การเขียนซ้อน ) - ตัวอย่างที่ 2

- ตัวอย่างนี้มีการเขียนซ้อนกันน้อยกว่า

```
function pow(x, n) {  
  if (n < 0) {  
    alert("Negative 'n' not supported");  
    return;  
  }  
  let result = 1;  
  for (let i = 0; i < n; i++) {  
    result *= x;  
  }  
  return result;  
}
```

# 1. การเขียน Code

## 1.7. Function Placement ( ตำแหน่งของฟังก์ชัน )

- หากมีการเขียนฟังก์ชันหลายฟังก์ชันและมีการเรียกใช้ฟังก์ชัน
- มี 3 วิธีที่นิยมใช้จัดการ

# 1. การเขียน Code

## 1.7. Function Placement ( ตำแหน่งของฟังก์ชัน )

### - 1.7.1. เรียกใช้ฟังก์ชัน หลังประกาศฟังก์ชัน

```
function createElement() {  
    ...  
}  
function setHandler(elem) {  
    ...  
}  
function walkAround() {  
    ...  
}  
// โค้ดที่เรียกใช้ฟังก์ชัน  
let elem = createElement();  
setHandler(elem);  
walkAround();
```

# 1. การเขียน Code

## 1.7. Function Placement ( ตำแหน่งของฟังก์ชัน ) - นิยมสุด

### - 1.7.2.การประกาศโค้ดที่เรียกใช้ฟังก์ชันไว้ด้านบน

```
// โค้ดที่เรียกใช้ฟังก์ชัน
let elem = createElement();
setHandler(elem);
walkAround();
function createElement() {
  ...
}
function setHandler(elem) {
  ...
}
function walkAround() {
  ...
}
```

# 1. การเขียน Code

## 1.7. Function Placement ( ตำแหน่งของฟังก์ชัน )

- 1.7.3.แบบผสมคือฟังก์ชันจะถูกประกาศในที่ที่โค้ดที่เรียกใช้ฟังก์ชัน

# 1. การเขียน Code

## 1.8. Style Guides ( การเขียนแบบอื่นๆ )

- [Google JavaScript Style Guide](#)
- [Airbnb JavaScript Style Guide](#)
- [Idiomatic.JS](#)
- [StandardJS](#)

# 1. การเขียน Code

## 1.9. Automated Linters

- Linter ช่วยตรวจวิธีการเขียน โค้ดเพื่อพัฒนาต่อไป
- Linter tool เพิ่มเติม (JSLint , JSHint , ESLint)

# 1. การเขียน Code

## 1.10. แบบฝึกหัด

- แก้ไขการเขียนโค้ดต่อไปนี้

```
function pow(x,n)
{
    let result=1;
    for(let i=0;i<n;i++) {result*=x;}
    return result;
}
let x=prompt("x?", ''), n=prompt("n?", '')
if (n<=0)
{
    alert(`Power ${n} is not supported, please
enter an integer number greater than zero`);
}
else
{
    alert(pow(x,n))
}
```





Software Park Thailand  
</Code Camp>



# หัวข้อ



Software Park Thailand  
</Code Camp>

- การเขียน Code
- **หลักการเขียน Comment**
- Babel
- Object

- Garbage Collection
- Methods ของ Object
- Constructor กับ New
- Methods ของ Primitive



Software Park Thailand  
</Code Camp>

# หลักการเขียน Comment

# 2. หลักการเขียน Comment

## 2.1. การเขียน Comment

- การเขียน Comment จะทำให้เข้าใจ Code ง่ายก็จริง แต่ถ้าใช้ในทางที่ผิด ก็อาจจะทำให้กลายเป็น code ที่ไม่มีคุณภาพ ได้
- โดยใช้ syntax ( // สำหรับ 1 บรรทัด และ /\*...\*/ สำหรับหลายบรรทัด )

# 2. หลักการเขียน Comment

## 2.2. การเขียน Comment ที่ ไม่ควรทำ

- ยากต่อการเข้าใจ
- Comment เฉพาะ โค้ดที่ซับซ้อนจริง ๆ
- ใส่ comment ที่ไม่จำเป็น

```
// โค้ดนี้มันจะทำแบบนี้ (... ) and แล้วก็ทำแบบนี้ต่อ (... )  
very;  
complex;  
code;
```

# 2. หลักการเขียน Comment

## 2.3. แยกออกมาเป็น ฟังก์ชัน

- สามารถใช้ชื่อของฟังก์ชันแทน Comment

```
function showPrimes(n) {  
  nextPrime:  
  for (let i = 2; i < n; i++) {  
    // check if i is a prime number  
    for (let j = 2; j < i; j++) {  
      if (i % j == 0) continue nextPrime;  
    }  
    alert(i);  
  }  
}
```

```
function showPrimes(n) {  
  
  for (let i = 2; i < n; i++) {  
    if (!isPrime(i)) continue;  
    alert(i);  
  }  
}  
function isPrime(n) {  
  for (let i = 2; i < n; i++) {  
    if (n % i == 0) return false;  
  }  
  return true;  
}
```

# 2. หลักการเขียน Comment

## 2.4. การเขียน Comment ที่แนะนำ

- ให้ใส่ไว้ข้างบนของฟังก์ชัน
- มีการระบุประเภทและคำอธิบายของ parameters
- มีการระบุประเภทและคำอธิบายของตัวที่ return

```
/**  
 * Returns x raised to the n-th power.  
 *  
 * @param {number} x The number to raise.  
 * @param {number} n The power, must be a natural number.  
 * @return {number} x raised to the n-th power.  
 */  
function pow(x, n) {  
    ...  
}
```



Software Park Thailand  
</Code Camp>





# หัวข้อ



Software Park Thailand  
</Code Camp>

- การเขียน Code
- หลักการเขียน Comment
- **Babel**
- Object

- Garbage Collection
- Methods ของ Object
- Constructor กับ New
- Methods ของ Primitive

# Babel

# 3. Babel

## 3.1. Babel คืออะไร

- การแปลง Code Version ใหม่ ให้เป็นเวอร์ชันดั้งเดิม
- เนื่องจาก ES6 ยังไม่รองรับครบทุก Platform แต่โปรแกรมเมอร์อยากเขียน Code ในรูปแบบของ ES6 จึงมี Babel เกิดขึ้นมา



# 3. Polyfills

## 3.2. Polyfills

- เนื่องจากเพิ่ม Feature ใน ES6 ไม่ได้มีการเพิ่มแค่ Syntax แต่มีการเพิ่ม Functions ที่ไม่มีในเวอร์ชันเก่าอีกด้วย แต่ตัว Polyfills จะไปสร้างฟังก์ชันนั้นให้ใน Code ที่เป็นเวอร์ชันเก่า แบบนี้เรียกว่า “Polyfills”



Software Park Thailand  
</Code Camp>



# หัวข้อ

- การเขียน Code
- หลักการเขียน Comment
- Babel
- Object

- Garbage Collection
- Methods ของ Object
- Constructor กับ New
- Methods ของ Primitive



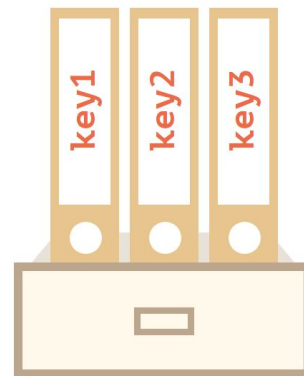
Software Park Thailand  
</Code Camp>

# Object

# 4. Object

## 4.1. Object คืออะไร - ( สำคัญที่สุดใน JavaScript )

- ใน JavaScript จะมีข้อมูลทั้งหมด 7 ประเภท
- string, boolean, bigint, number, null, undefined และ Object
- ซึ่ง 6 ประเภทแรกจะเรียกว่า primitive



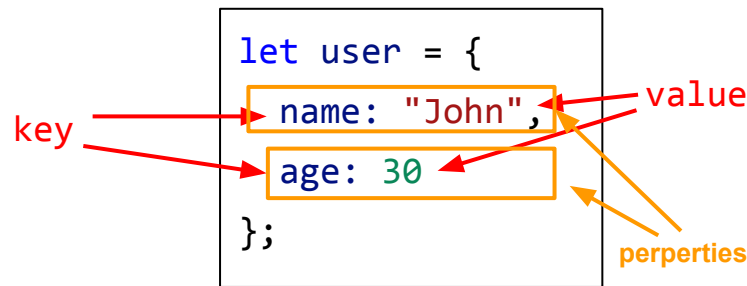
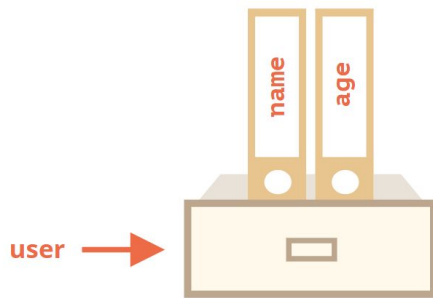
Objects



# 4. Object

## 4.1. Object คืออะไร

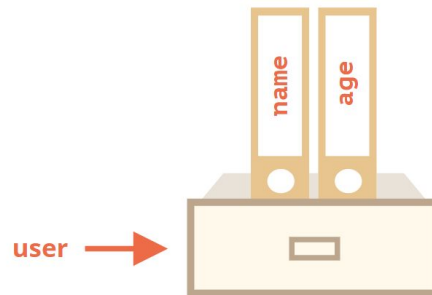
- Object จะประกอบด้วย properties
- ซึ่ง properties แต่ละอันจะประกอบด้วย key และ value



# 4. Object

## 4.1. Object คืออะไร

- ตัวอย่างที่ 1
- Key หรือเรียกอีกชื่อว่า property name
- Value หรือเรียกอีกชื่อว่า property value
- key และ value รวมกันเรียกว่า properties  
ของ Object



```
let user = {  
  name: "John",  
  age: 30  
};
```

Diagram illustrating the structure of a JavaScript object. The code defines a variable `user` pointing to an object with two properties: `name` (value: "John") and `age` (value: 30). Red arrows point from the labels 'key' and 'value' to the corresponding parts of the object literal: 'name' and 'age' are keys, and 'John' and '30' are values.

# 4. Object

## 4.1. Object คืออะไร

- Properties นั้นสามารถมีช่องว่าง (space) ได้
- แต่ต้องเขียนเป็นค่า String

```
let user = {  
  name: "John",  
  age: 30,  
  "computer skill": null  
};
```

# 4. Object

## 4.2. การดึงค่าของ Properties ออกมาใช้ - มีทั้งหมด 2 วิธี

- การใช้เครื่องหมายจุด
- `<ชื่อของ Object>.<ชื่อ Properties>`
- ตัวอย่าง

`alert( user.age ); // 30`  
`alert( user.name ); // "John"`

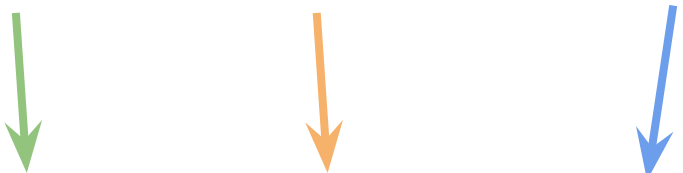
```
let user = {  
  name: "John",  
  age: 30  
};
```

# 4. Object

## 4.3. การเพิ่ม Properties ของ Object

-  $\text{<ชื่อของ Object>.\text{<ชื่อ Properties>} = \text{<ค่าที่จะใส่ไปใน property นั้น>}}$

- ตัวอย่าง



```
user.height = 176;  
user.isSingle = true;
```

# 4. Object

## 4.3. การเพิ่ม Properties ของ Object - ตัวอย่าง

- การเพิ่ม properties ของ user เพิ่ม 2 properties

โดยเพิ่ม properties ที่มี key ชื่อ height และ value เป็น number ที่มีค่า 176

และ properties ที่มี key ชื่อ isSingle และ value เป็น boolean ที่มีค่าเป็น true

```
let user = {  
  name: "John",  
  age: 30  
};
```

```
user.height = 176;  
user.isSingle = true;
```

```
let user = {  
  name: "John",  
  age: 30,  
  height: 176,  
  isSingle: true,  
};
```

# 4. Object

## 4.4. การลบ Properties ของ Object

- delete <ชื่อของ Object>.<ชื่อ Properties>
- ตัวอย่าง

delete user.height

# 4. Object

## 4.4. การลบ Properties ของ Object - ตัวอย่าง

- การลบ properties ของ user 1 properties

โดยลบ properties ที่มี key ชื่อ height

```
let user = {  
  name: "John",  
  age: 30,  
  height: 176,  
  isSingle: true,  
};
```

delete user.height;


```
let user = {  
  name: "John",  
  age: 30,  
  isSingle: true,  
};
```



# 4. Object

## 4.5. แล้ว [ ] ทำได้แบบเดียวกับจุดใหม่

- การใช้วงเล็บก้ามปู [ ]
- ใช้ในกรณีที่ key มีช่องว่างได้ด้วย
- <ชื่อของ Object>[<ชื่อ Properties>]
- ตัวอย่าง

  
alert( user["computer skill"] ); // null  
alert( user."computer skill" ); // Error  
alert( user.computer skill ); // Error

```
let user = {  
  name: "John",  
  age: 30,  
  "computer skill": null  
};
```

แบบจุดจะ Error ทั้งคู่

# 4. Object

## 4.5. แล้ว [ ] ทำได้แบบเดียวกับจุดใหม่

- key ยังสามารถเป็นตัวแปรได้ด้วย

```
let user = {  
  name: "John",  
  age: 30  
};  
  
let key = prompt("What do you want to know about the user?", "name");  
  
// access by variable  
alert( user[key] ); // John (ถ้าใส่ "name")
```

# 4. Object

## 4.5. แล้ว [ ] ทำได้แบบเดียวกับจุดใหม่

- ได้เหมือนกันทั้งเพิ่ม, ลบ และ เรียกได้เช่นเดียวกับจุดเลย

```
let user = {};  
  
// set  
user["likes birds"] = true;  
  
// get  
alert(user["likes birds"]); // true  
  
// delete  
delete user["likes birds"];
```

# 4. Object

## 4.6. การสร้าง Object

- การสร้าง Object มีทั้งหมด 2 วิธี
  1. แบบ Object constructor
  2. แบบ Object Iteral

```
let user = new Object(); // "Object constructor"  
let user = {}; // "Object Iteral"
```



# 4. Object

## 4.6. การสร้าง Object - แบบฝึกหัด

1. ให้สร้าง Object แบบ Object Iteral โดยให้กำหนดตัวแปรชื่อ human โดยมี Properties ทั้ง 5 อย่าง
  - a. ชื่อของผู้เรียน เป็น String
  - b. อายุของผู้เรียนเป็น number
  - c. บ้านของตัวเองเป็น String
  - d. โสดหรือไม่โสดเป็น boolean
  - e. คะแนนความฉลาดของตัวเองเป็น number (เต็ม 10)

# 4. Object

## 4.7. Computed Properties

- กรณีที่ชื่อของ Properties นั้นไม่แน่นอน แล้วเราอยากสร้าง Properties ที่มี key มาจากตัวแปร ให้ใช้ Computed Properties
- โดยให้ ครอบตัวแปรนั้นด้วย []

```
let propertyName = "age";

let obj = {
  propertyName: 20, // จะได้ key เป็น propertyName
}

console.log(obj);
```

```
let propertyName = "age";

let obj = {
  [propertyName]: 20, // จะได้ key เป็น age
}

console.log(obj);
```

# 4. Object

## 4.7. Computed properties

- key ที่เป็น computed properties สามารถตั้งชื่อจากตัวแปรภายนอกได้
- โดยสามารถกำหนดค่าตัวที่รับเข้ามาเป็น value ที่ตั้งไว้ก่อนได้

```
let fruit = prompt("Which fruit to buy?", "apple");
```

```
let bag = {};
```

```
// เอาค่าชื่อมาจากตัวแปร
```

```
bag[fruit] = 5;
```

# 4. Object

## 4.7. Computed Properties - แบบฝึกหัด

1. ให้เขียนโปรแกรมที่รับค่า key และ value ของ Properties ของ Object หนึ่ง จนกว่าจะเจอคำว่า stop และนำค่าเหล่านั้นมาสร้าง Object หลังจากนั้น console.log() object นั้นออกมา



# 4. Object

## 4.7. Computed Properties - แบบฝึกหัด

2. ให้เขียนโปรแกรมที่รับค่า key และ value ของ Properties ของ Object หนึ่ง โดยให้ key เป็นชื่อของผลไม้ และ value เป็นจำนวนของผลไม้ (number) โดยถ้า ผลไม้ชนิดไหนที่มีมากกว่า 1 ผล ให้เติม s ไปหลัง key นั้นด้วย

# 4. Object

## 4.8. Property value shorthand คือ

- เราสามารถใช้ ชื่อของตัวแปร ที่เราเก็บค่าไว้ เป็น key ได้โดยเขียนดังนี้

```
function makeUser(name, age) {  
  return {  
    name: name,  
    age: age, // ใส่ค่า key : value อื่น  
  };  
}  
  
let user = makeUser("John", 30); // สร้าง object ใหม่ชื่อ User  
alert(user.name); // John
```

```
function makeUser(name, age) {  
  return {  
    name,  
    age,  
  };  
}  
  
let user = makeUser("John", 30);  
alert(user.name); // John
```

# 4. Object

## 4.9. Property names limitations คือ

- ชื่อของ key สามารถเขียนได้ทั้งแบบ String และ symbol
- key
- คำต้องห้าม(Reserved word)ก็สามารถใช้ได้เช่นกัน (for, function , ...)

```
let obj = {  
  0: "test" // number 0 จะถูกแปลงเป็น String "0"  
};  
// การแสดงผล สามารถเขียนค่า key เป็น ตัวเลข หรือ เขียนตัวเลขแบบ string  
alert( obj["0"] ); // test  
alert( obj[0] ); // test ได้ค่าเหมือนกัน
```

# 4. Object

## 4.10. การทดสอบการมีอยู่ของค่า key

- ผลลัพธ์จะเป็นค่า ความจริงว่า มี เท่ากับ จริง, ไม่มี เท่ากับ ไม่จริง
- object เปล่ามีค่าเท่ากับ undefined
- `<"key"> in <object>`

```
let user = { name: "John", age: 30 };
```

`alert( "age" in user );` // มี key ชื่อ age ใน object ชื่อ user  
`alert( "blabla" in user );` // ไม่มี key ชื่อ blabla ใน object

# 4. Object

## 4.11. การใช้ in ใน for .... loop

- การใช้คือ for ( key in object )
- การใช้งานจะแตกต่างจาก for( .. ; .. ; .. )
- เป็นการดึงค่าแต่ละ key ของ object

```
for (key in object) {  
    // ค่า key ทุกค่าใน object สามารถนำมาใช้ใน for loop นี้ได้ทั้งหมด  
}  
  
let user = {  
    name: "John",  
    age: 30,  
    isAdmin: true  
};  
  
for (let key in user) {  
    // ตั้งชื่อ key เป็นตัวแทนของค่า key ทั้งหมดใน object  
    alert( key ); // key = name, age, isAdmin  
    alert( user[key] ); // value = John, 30, true  
}
```

# 4. Object

## 4.11. การใช้ in ใน for .... loop

- ตัวอย่าง

```
let user = {  
  name: "John",  
  age: 30,  
  isAdmin: true  
};
```

```
for (let key in user) {  
  alert( key );  
  alert( user[key] );  
}
```

// ตัวแปรชื่อ key จะเป็นตัวแทนของ keys ทั้งหมดใน object  
// key = name, age, isAdmin  
// value = John, 30, true

# 4. Object

## 4.12. ลำดับใน for ( key in object )

- ถ้า key เป็น string ของตัวเลขมันจะเรียงให้
- ถ้า key เป็น string ที่ไม่ใช่ตัวเลขมันก็จะแบบเดิม

```
let codes = {  
  "49": "Germany",  
  "41": "Switzerland",  
  "44": "Great Britain",  
  // ..,  
  "1": "USA"  
};  
  
for (let code in codes) {  
  alert(code); // 1, 41, 44, 49  
}
```

```
let user = {  
  name: "John",  
  age: 30,  
  isAdmin: true  
};  
  
for (let key in user) {  
  alert( key ); // name, age, isAdmin  
}
```

# 4. Object

## 4.12. ลำดับใน for ( key in object)

- โดยลำดับจะเรียงตามข้อมูลใน object ในกรณีที่เป็น string
- กรณี ตัวเลข มีค่า + และ ไม่มีค่า + จะเรียงจาก มากไปน้อย และ น้อยไปมาก

```
let codes = {  
  "49": "Germany",  
  "41": "Switzerland",  
  "44": "Great Britain",  
  // ..,  
  "1": "USA"  
};  
  
for (let code in codes) {  
  alert(code);  
  // 1, 41, 44, 49  
}
```

```
let codes = {  
  "+49": "Germany",  
  "+41": "Switzerland",  
  "+44": "Great Britain",  
  // ..,  
  "+1": "USA"  
};  
  
for (let code in codes) {  
  alert(code);  
  // 49, 44, 41, 1  
}
```

```
let user = {  
  name: "John",  
  age: 30,  
  isAdmin: true  
};  
  
for (let key in user) {  
  alert( key ); // name, age, isAdmin  
}
```



# 4. Object

## 4.13. primitive VS object

- ความแตกต่างระหว่าง primitive และ Object คือ
- primitive จะ copied by value
- object จะ copied by reference

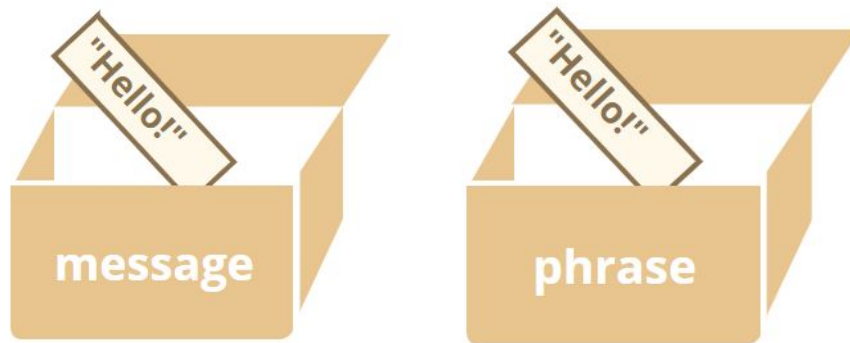
# 4. Object

## 4.13. primitive VS object

- ตัวอย่าง primitive (copied by value)
- ตัวแปรของ primitive จะเก็บ value ของ primitive

```
let message = "Hello!";
```

```
let phrase = message;
```



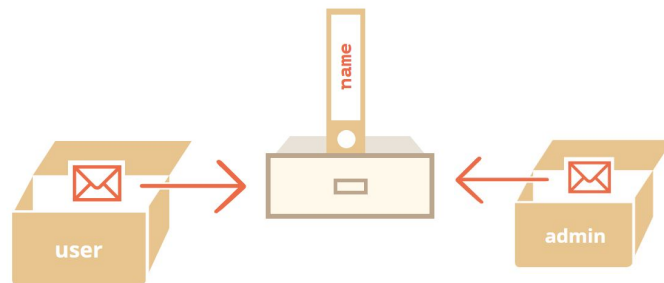
# 4. Object

## 4.13. primitive VS object

- ตัวอย่าง Object (copied by reference)
- ตัวแปรของ Object จะเก็บ Reference(ที่อยู่) ของ object

```
let user = {  
  name: "John"  
};
```

```
let admin = user; // copy the reference
```



# 4. Object

## 4.13. primitive VS object

- เมื่อมีการเปลี่ยนค่าจะส่งให้อีกตัวเปลี่ยนค่าด้วย

```
let user = { name: 'John' };

let admin = user;

admin.name = 'Pete'; // เปลี่ยน properties name ใน admin

alert(user.name); // 'Pete', properties name ใน user ก็จะถูกเปลี่ยนเช่นเดียวกัน
```

# 4. Object

## 4.13. การเปรียบเทียบ Object

- เป็นการส่งค่าของ ตัวแปรที่อ้างอิงไปให้อีกตัวแปรหนึ่ง

```
let a = {};  
let b = a; // ค่า b จากเชื่อมไปยัง a  
  
alert( a == b ); // true  
alert( a === b ); // true  
// a และ b เปรียบเสมือนตัวเดียวกัน
```

```
let a = {};  
let b = {}; // b ไม่เกี่ยวข้องกับ a  
  
alert( a == b ); // false
```

# 4. Object

## 4.14. Object ที่ประกาศตัวแปรด้วย const

- ค่า value ใน object ที่เป็น const สามารถ เปลี่ยนแปลงค่า ได้
- แต่ เปลี่ยนแปลงได้เฉพาะ value ข้างใน ไม่สามารถประกาศโครงสร้างใหม่ ทับได้

```
const user = {  
  name: "John"  
};  
  
// Error ไม่สามารถประกาศค่า object ใหม่  
// ได้  
user = {  
  name: "Pete"  
};
```

# 4. Object

## 4.14. Object ที่ประกาศตัวแปรด้วย const

- ตัวอย่าง
- แต่สามารถเพิ่ม properties เข้าไปได้

```
const user = {  
  name: "John"  
};  
  
user.age = 25;  
// เพิ่ม properties ใหม่  
  
alert(user.age); // 25
```

# 4. Object

## 4.15. Cloning and merging, Object.assign

- การ Clone จะป้องกันค่า ที่เราดึงมาจากตัวแปร แต่ต้องการ คัดลอกแบบ ตัวแปรต่าง ๆ นั้นเหมือนกันแต่ค่าไม่เชื่อมโยงกัน
- เนื่องจากใน JS ไม่มี ฟังก์ชันที่เอาไว้สำหรับ clone ค่า object เราจึงต้อง เขียนเอง



# 4. Object

## 4.15. Cloning and merging, Object.assign

### - ตัวอย่าง

```
let user = {  
  name: "John",  
  age: 30  
};  
let clone = {};  
// ใช้ for..in..loop สำหรับแทนค่า key แต่ละตัว  
for (let key in user) {  
  clone[key] = user[key];  
}  
clone.name = "Pete";    // กำหนดค่า value ในแต่ละ key  
alert( user.name );    // John in the original object
```

# 4. Object

## 4.16. Object.assign

- เป็น function การรวมกันของ Object

```
let user = { name: "John" };

let permissions1 = { canView: true };
let permissions2 = { canEdit: true };

// copies all properties from permissions1 and permissions2 into user
Object.assign(user, permissions1, permissions2);

// now user = { name: "John", canView: true, canEdit: true }
```

# 4. Object

## 4.16. Object.assign

- ตัวอย่าง

```
let user = { name: "John" };

// overwrite name, add isAdmin
Object.assign(user, { name: "Pete", isAdmin: true });

// now user = { name: "Pete", isAdmin: true }
```

# 4. Object

## 4.16. Object.assign

- นำมาประยุกต์กับการ clone Object

```
let user = {  
  name: "John",  
  age: 30  
};  
  
let clone = Object.assign({}, user);
```

# 4. Object

## 4.17. properties ที่เป็น object

- object ก็สามารถมี properties ที่เป็น object ได้เหมือนกัน(Object ซ้อน Object)

```
let user = {  
  name: "John",  
  sizes: {  
    height: 182,  
    width: 50  
  }  
};  
  
alert( user.sizes.height ); // 182
```

# 4. Object

## 4.18. แบบฝึกหัด

1. ให้ทำตามคำสั่งต่อไปนี้
  - a. สร้าง Object เปล่าขึ้นมา
  - b. เพิ่ม properties name เข้าไปและให้ value เป็น “Sonter”
  - c. เพิ่ม properties surname เข้าไปและให้ value เป็น “Pakorn”
  - d. เปลี่ยน properties name เป็น “Boy”
  - e. ลบ properties name ออกจาก Object

# 4. Object

## 4.18. แบบฝึกหัด

2. ให้เขียนฟังก์ชันชื่อ isEmpty(obj) โดยจะมี parameters เป็น obj และฟังก์ชันนี้จะเช็คว่ obj นีมี properties ไหม ถ้ามีให้คืนค่า true ถ้าไม่มีให้คืนค่า false

# 4. Object

## 4.18. แบบฝึกหัด

### 3. การเขียนข้างล่างต่อไปนี้ Error ไหม

```
const user = {  
  name: "John"  
};  
  
// does it work?  
user.name = "Pete";
```



# 4. Object

## 4.18. แบบฝึกหัด

4. จงเขียนฟังก์ชัน `sum(obj)` ที่รับ `obj` ที่เก็บ properties โดยมี key เป็นชื่อพนักงานและมี value เป็นเงินเดือน ให้ฟังก์ชันคืนค่าเป็นผลรวมของเงินเดือนพนักงานทั้งหมด

```
let salaries = {  
  John: 100,  
  Ann: 160,  
  Pete: 130  
}
```

# 4. Object

## 4.18. แบบฝึกหัด

5. จงเขียนฟังก์ชัน `multiplyNumeric(obj, times)` โดยถ้า `properties` นั้นมี `value` เป็น `number` ให้คูณ `value` นั้นด้วย `times` ถ้าข้อมูลเป็นอย่างอื่นไม่ต้องทำอะไร

```
// before the call  
let menu = {  
  width: 200,  
  height: 300,  
  title: "My menu"  
};
```

```
multiplyNumeric(menu, 2);  
  
// after the call  
menu = {  
  width: 400,  
  height: 600,  
  title: "My menu"  
};
```



Software Park Thailand  
</Code Camp>



# หัวข้อ



Software Park Thailand  
</Code Camp>

- การเขียน Code
- หลักการเขียน Comment
- Babel
- Object

- Garbage Collection
- Methods ของ Object
- Constructor กับ New
- Methods ของ Primitive



Software Park Thailand  
</Code Camp>

# Garbage Collection

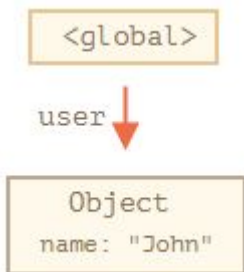
# 5. Garbage Collection

## 5.1. Garbage Collection คืออะไร

- เป็นตัวที่ช่วยกำจัดตัวแปรที่ไม่สามารถเข้าถึงได้แล้วให้อัตโนมัติ
- ตัวอย่าง

```
// user has a reference to the object
```

```
let user = {  
  name: "John"  
};
```



```
user = null;
```





Software Park Thailand  
</Code Camp>



# หัวข้อ

- การเขียน Code
- หลักการเขียน Comment
- Babel
- Object

- Garbage Collection
- **Methods ของ Object**
- Constructor กับ New
- Methods ของ Primitive





# Methods ของ Object

# 6. Methods ของ Object

## 6.1. การสร้าง Methods ใน Object

- สามารถเขียนได้เช่นเดียวกับการกำหนดค่าของ primitive

- `<ชื่อของ Object>.<ชื่อ Properties> = <function>`

```
let user = {  
  name: "John",  
  age: 30  
};
```

```
user.sayHi = function() {  
  alert("Hello!");  
};
```

```
user.sayHi(); // Hello!
```

# 6. Methods ของ Object

## 6.1. การสร้าง Methods ใน Object

- สามารถ ประกาศฟังก์ชันก่อนแล้ว Assign ค่าใส่ตัวแปรก็ได้

```
let user = {  
  // ...  
};  
// ประกาศ Methods ก่อน  
function sayHi() {  
  alert("Hello!");  
};  
  
// แล้วกำหนด Method ให้ Object  
user.sayHi = sayHi;  
  
user.sayHi(); // Hello!
```

# 6. Methods ของ Object

## 6.2. การสร้าง Methods แบบย่อ (Method Shorthand)

- สามารถเขียนใน {} ปีกกาเป็น Property ของ Object ได้
- โดยชื่อของ function จะกลายเป็น Property name
- ค่าของ function นั้นจะถูก assign ให้ property นั้นด้วย

# 6. Methods ของ Object

## 6.2. การสร้าง Methods แบบย่อ (Method Shorthand)

- ตัวอย่าง
- การเขียนทั้งสองแบบ เหมือนกัน

```
user = {  
  sayHi: function() {  
    alert("Hello");  
  }  
};  
  
user = {  
  sayHi() { // เหมือนกับ "sayHi: function()"  
    alert("Hello");  
  }  
};
```

# 6. Methods ของ Object

## 6.3. “this” คีย์เวิร์ดใน Method

- การเขียน this ใน Method จะหมายถึง Object ปัจจุบัน นั้น
- ใช้กรณีที่เราต้องการ อ้างอิง Object ที่เก็บ Method นั้น

# 6. Methods ของ Object

## 6.3. “this” ก็ยั้เฝ้าใน Method

- ตัวอย่าง

```
let user = {  
  name: "John",  
  age: 30,  
  
  sayHi() {  
    // "this" จะอ้างถึง Object ปัจจุบัน  
    alert(this.name);  
  }  
};  
  
user.sayHi(); // John
```

# 6. Methods ของ Object

## 6.4. “this” ขึ้นอยู่กับตัวที่เก็บ Object ปัจจุบัน

- ตัวอย่าง

```
let user = { name: "John" };
let admin = { name: "Admin" };

function sayHi() {
  alert( this.name );
}

// ใส่ function sayHi ไปให้ทั้งสอง Object
user.sayHi = sayHi;
admin.sayHi = sayHi;

// การเรียกฟังก์ชันทั้งสองครั้งนี้ this จะแตกต่างกัน
user.sayHi(); // John (this จะหมายถึง user)
admin.sayHi(); // Admin (this จะหมายถึง admin)

admin['sayHi'](); // Admin (dot หรือ square brackets ก็ได้)
```



# 6. Methods ของ Object

## 6.4. “this” ขึ้นอยู่กับตัวที่เก็บ Object ปัจจุบัน

- จำง่าย ๆ คือ `obj.function()` ก็หมายความว่า this คือตัวใน `function()` จะหมายถึง Object ที่ชื่อ `obj`

# 6. Methods ของ Object

## 6.4. “this” จะหมายถึง Object ที่อยู่นำจุดเสมอ

- "this" is always object **before** the dot.
- การเรียก function ที่ไม่ผ่าน Object ตัว this ก็็เวิร์ดจะกลายเป็น undefined

```
function sayHi() {  
    alert(this);  
}  
  
sayHi(); // undefined
```

# 6. Methods ของ Object

## 6.5. Arrow function ไม่มี this เป็นของตัวเอง

- เนื่องจาก arrow function ไม่มี this เป็นของตัวเอง ถ้าเราเรียกมันจาก function มันจะใช้ this keyword ของ function

```
let user = {  
  firstName: "Ilya",  
  sayHi() {  
    let arrow = () => alert(this.firstName);  
    arrow();  
  }  
};  
  
user.sayHi(); // Ilya
```

# 6. Methods ของ Object

## 6.6. แบบฝึกหัด

1. การทำงานของ code ดังกล่าวจะได้อะไรออกมา

```
let user = {  
  name: "John",  
  go: function() { alert(this.name) }  
}  
  
(user.go)()
```

# 6. Methods ของ Object

## 6.6. แบบฝึกหัด

### 2. การทำงานของ code ดังกล่าวจะได้อะไรออกมา

```
function makeUser() {  
  return {  
    name: "John",  
    ref: this  
  };  
};  
  
let user = makeUser();  
  
alert( user.ref.name ); // What's the result?
```

# 6. Methods ของ Object

## 6.6. แบบฝึกหัด

- สร้าง object calculator จาก 3 methods นี้:  
read() ใช้ prompts สำหรับรับค่ามา 2 ค่าและเก็บเป็น object properties.  
sum() คำนวณผลบวกของ 2 ค่านั้น.  
mul() คำนวณผลคูณของ 2 ค่านั้น.

```
let calculator = {  
  // ... your code ...  
};  
  
calculator.read();  
alert( calculator.sum() );  
alert( calculator.mul() );
```

# 6. Methods ของ Object

## 6.6. แบบฝึกหัด

4. ให้ Object ชื่อ ladder มี method ขึ้น และ ลง

```
let ladder = {  
  step: 0,  
  up() {  
    this.step++;  
  },  
  down() {  
    this.step--;  
  },  
  showStep: function() { // shows the current step  
    alert( this.step );  
  }  
};
```

# 6. Methods ของ Object

## 6.6. แบบฝึกหัด

4(ต่อ). Object ชื่อ ladder สามารถเรียก function แบบนี้ได้

```
ladder.up();  
ladder.up();  
ladder.down();  
ladder.showStep(); // 1
```



# 6. Methods ของ Object

## 6.6. แบบฝึกหัด

4(ต่อ). ดัดแปลง Object ชื่อ ladder สามารถเรียก function แบบนี้ได้

```
ladder.up().up().down().showStep(); // 1
```



Software Park Thailand  
</Code Camp>



# หัวข้อ



Software Park Thailand  
</Code Camp>

- การเขียน Code
- หลักการเขียน Comment
- Babel
- Object

- Garbage Collection
- Methods ของ Object
- Constructor กับ New
- Methods ของ Primitive



Software Park Thailand  
</Code Camp>

# Constructor กับ New

# 7. Constructor กับ New

## 7.1. Constructor function คืออะไร

- เป็น ฟังก์ชันที่ใช้สร้าง Object
- ใช้คู่กับ new keyword เท่านั้น
- ต้อง ขึ้นต้นด้วยตัวอักษรพิมพ์ใหญ่ เท่านั้น

# 7. Constructor กับ New

## 7.1. Constructor function คืออะไร

- เมื่อ function ถูกรันพร้อมกับ new keyword จะเกิด
  1. Object วางเปล่าถูกสร้าง และ ใส่ค่าไปให้ this
  2. คำสั่งใน constructor function ถูกรัน และ assign properties ให้ this
  3. สุดท้าย return this ออกไป

```
function User(name) {  
  this.name = name;  
  this.isAdmin = false;  
}  
  
let user = new User("Jack");  
  
alert(user.name); // Jack  
alert(user.isAdmin); // false
```

# 7. Constructor กับ New

## 7.1. Constructor function คืออะไร

- เปรียบเสมือนการทำแบบนี้

```
function User(name) {  
  this.name = name;  
  this.isAdmin = false;  
}
```

```
let user = new User("Jack");  
  
alert(user.name); // Jack  
alert(user.isAdmin); // false
```

```
function User(name) {  
  // this = {}; (ทำโดยอัตโนมัติ)  
  
  // add properties to this  
  this.name = name;  
  this.isAdmin = false;  
  
  // return this; (ทำโดยอัตโนมัติ)  
}
```

# 7. Constructor กับ New

## 7.1. Constructor function คืออะไร

- จากการใช้ constructor function จะให้ผลลัพธ์เหมือนกับการประกาศ Object เลย

```
let user = new User("Jack")
```

```
let user = {  
  name: "Jack",  
  isAdmin: false  
};
```



# 7. Constructor กับ New

## 7.1. Constructor function คืออะไร

- จุดประสงค์หลักของ Constructor function คือการ reuse Code ที่ใช้สร้าง Object

# 7. Constructor กับ New

## 7.2. การใส่ return ใน Constructor function

- ปกติถ้าไม่ใส่ return ใน constructor function ตัว constructor function จะ return this มาให้อัตโนมัติ แต่ถ้าเรา ใส่ return เอง ตัว constructor ก็จะไม่ return this แต่ return ตามค่าที่เราเขียนแทน

# 7. Constructor กับ New

## 7.2. การใส่ return ใน Constructor function

### - ตัวอย่าง

```
function BigUser() {  
  
    this.name = "John";  
  
    return { name: "Godzilla" }; // <-- returns อันนี้แทน  
}  
  
alert( new BigUser().name ); // Godzilla
```

# 7. Constructor กับ New

## 7.3. Methods ใน Constructor function

- ใน Constructor function เราสามารถ ใส่ Method ให้กับ constructor เปรียบเสมือน constructor เป็น Object ได้ เลย

# 7. Constructor กับ New

## 7.3. Methods ใน Constructor function

### - ตัวอย่าง

```
function User(name) {  
  this.name = name;  
  
  this.sayHi = function() {  
    alert( "My name is: " + this.name );  
  };  
}  
  
let john = new User("John");  
  
john.sayHi(); // My name is: John  
  
/*  
john = {  
  name: "John",  
  sayHi: function() { ... }  
}  
*/
```

# 7. Constructor กับ New

## 7.4. แบบฝึกหัด

1. สร้าง constructor function ที่ใช้สำหรับสร้าง Calculator โดยต้องมี 3 Methods นี้
  - a. read(): รับค่าจาก prompt สองตัว
  - b. sum(): ให้คืนค่าจากการบวกกันของตัวแปรสองตัว
  - c. mul(): ให้คืนค่าจากการคูณกันของตัวแปรสองตัว

# 7. Constructor กับ New

## 7.4. แบบฝึกหัด

2. สร้าง constructor function Accumulator(startingValue)

- โดยที่ Object ดังกล่าวควร เก็บผลรวมไว้ใน property ที่มี key ชื่อว่า value, ค่าเริ่มต้นของ key ชื่อ value นี้ คือ startingValue
- ฟังก์ชัน read() ควรอ่านค่าจาก prompt() และ เพิ่มค่าที่ใส่เข้ามาใน key ชื่อ value

พูดง่าย ๆ ก็คือ value คือผลรวมของ prompt โดยเริ่มจาก startingValue



Software Park Thailand  
</Code Camp>





# หัวข้อ



Software Park Thailand  
</Code Camp>

- การเขียน Code
- หลักการเขียน Comment
- Babel
- Object

- Garbage Collection
- Methods ของ Object
- Constructor กับ New
- **Methods ของ Primitive**



# Methods ของ Primitive

# 8. Methods ของ Primitive

## 8.1. Methods ของ primitive

- Object จะหนักกว่า primitive
- แต่ primitive ไม่สามารถใช้ Methods ได้

# 8. Methods ของ Primitive

## 8.1. Methods ของ primitive

- ตัวอย่าง

```
let str = "Hello";  
  
alert( str.toUpperCase() ); // HELLO
```

