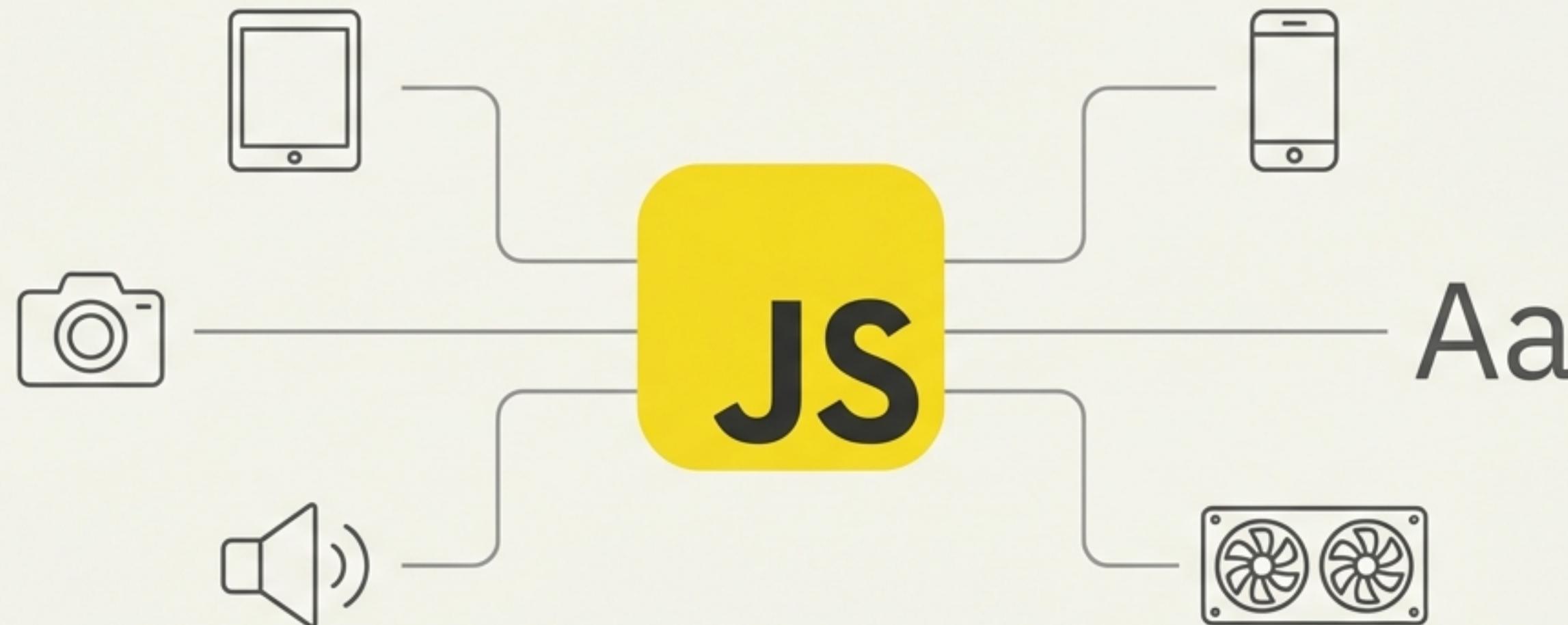
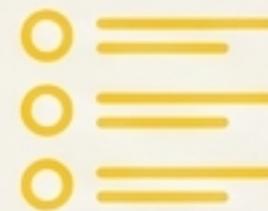


# JavaScript: เครื่องมือจัดการ ข้อมูลสำหรับนักพัฒนา



# เส้นทางสู่การเป็นผู้เชี่ยวชาญด้านข้อมูล



## การจัดเก็บข้อมูลเป็นรายการ

- `Array`

1

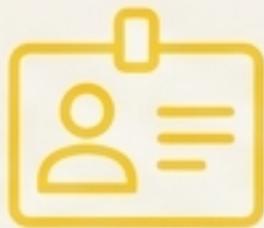


2

## การทำงานกับข้อมูลในรายการ

- `Loops` (**for, while**)  
- `Iterators` (**.forEach, .map, .reduce**)

3



## การจัดการข้อมูลที่ซับซ้อน

- `Objects`



4

## การสื่อสารข้อมูลกับโลกภายนอก

- `JSON`

# ชุดเครื่องมือจัดการข้อมูลของคุณ



Array

จัดเก็บข้อมูลเป็นรายการที่มีลำดับ



Loops & Iterators  
(.map, .reduce)

ทำงาน/แปลงรูปข้อมูลในรายการ



Object

แสดงข้อมูลที่ซับซ้อนด้วย key-value



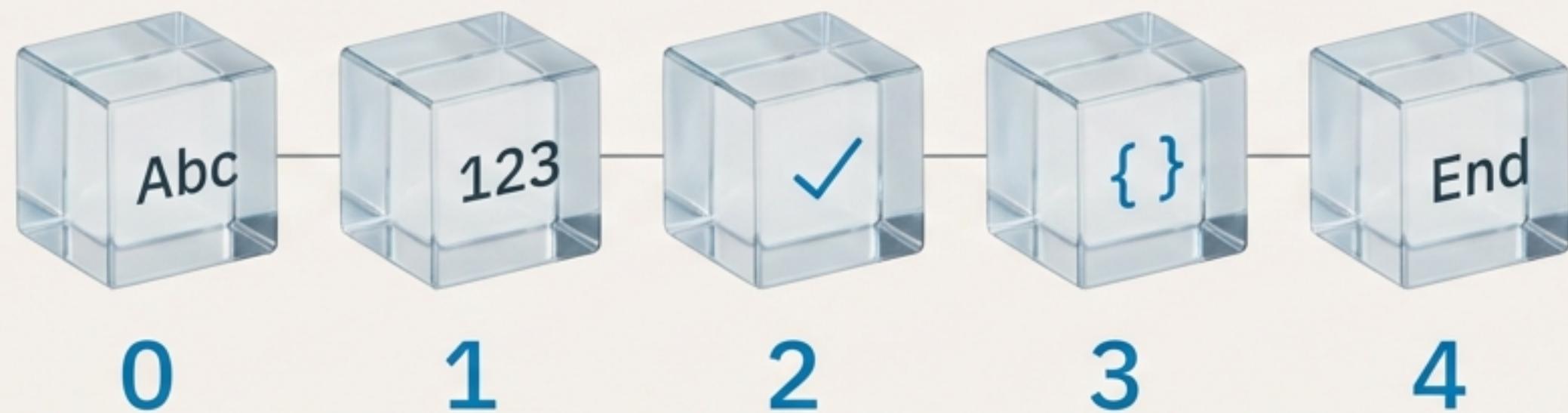
JSON

แลกเปลี่ยนข้อมูลกับระบบอื่น

ข้อมูลคือหัวใจของทุกแอปพลิเคชัน  
และตอนนี้คุณมีเครื่องมือพื้นฐานในการจัดการมันแล้ว

# Array คืออะไร?

ชุดข้อมูลที่มีลำดับ ใช้สำหรับเก็บค่าหลายๆ ค่าไว้ในตัวแปรเดียว



มีลำดับ: แต่ละไอเท็มมีตำแหน่งของตัวเอง

Index เริ่มที่ 0: การเข้าถึงข้อมูลเริ่มนับจากศูนย์

เก็บข้อมูลได้หลากหลาย: สามารถเก็บได้ทั้ง string , number , boolean , object ฯลฯ ปนกัน

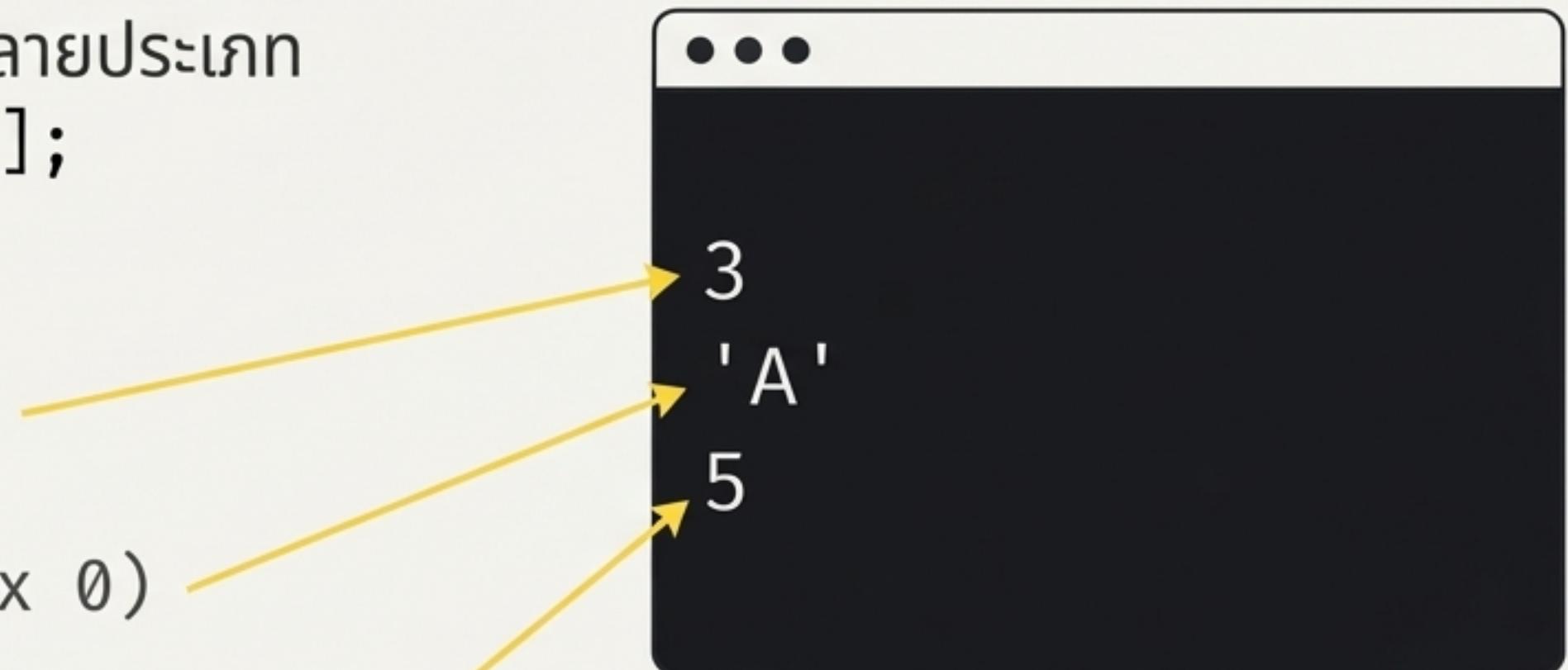
# การสร้างและเข้าถึง Array

```
// สร้าง Array ที่มีข้อมูลหลากหลายประเภท  
let arr3 = ['A', false, 5];
```

```
// ความยาวของ Array  
console.log(arr3.length);
```

```
// การเข้าถึงข้อมูลตัวแรก (index 0)  
console.log(arr3[0]);
```

```
// การเข้าถึงข้อมูลตัวสุดท้าย (index 2)  
console.log(arr3[2]);
```



# วิธีการสร้าง Array

```
// 1. แบบ Literal (The most common way)
let fruits = ['Apple', 'Banana', 'Cherry']; ←
```

```
// 2. แบบว่างเปล่า (Empty array)
let emptyArr = [];
```

```
// 3. แบบกำหนดความยาวล่วงหน้า (Pre-defined length)
let sizedArr = Array(5); // [ <5 empty items> ]
```

---

การใช้ `[]` (literal) เป็นวิธีที่นิยมและอ่านง่ายที่สุด

# การเข้าถึงข้อมูลใน Array

## การหาความยาว (Getting the Length)

```
let arr1 = ["A", true, 2];  
console.log(arr1.length);
```

Output: 3

`.length` จะบอกจำนวน  
สมาชิกทั้งหมด

## การเข้าถึงด้วย Index (Accessing by Index)

```
// เข้าถึงข้อมูลตัวสุดท้าย  
console.log(arr1[arr1.length - 1]);
```

```
// เข้าถึง Index ที่ไม่มีอยู่  
console.log(arr1[3]);
```

Output: 2

Output: undefined

จำไว้ว่า Index  
ที่ไม่มีอยู่จะคืนค่า  
'undefined'

# เครื่องมือพื้นฐาน: เพิ่มและลบข้อมูล

จัดการข้อมูลที่ก้าง Array

**.push() – เพิ่มข้อมูลหนึ่งตัว (หรือมากกว่า) ไปยังก้าง Array**

เพิ่มข้อมูลใหม่ไปยังตำแหน่งสุดท้ายของ Array

```
let arr1 = ['A', true, 2];
arr1.push("new value");
// arr1 is now ['A', true, 2, 'new value']
```



**.pop() – ลบข้อมูลตัวสุดท้ายออกจาก Array และคืนค่าที่ลบออกมา**

ลบข้อมูลตัวสุดท้ายออกจาก Array และส่งคืนค่าที่ถูกลบ

```
let removedValue = arr1.pop();
// removedValue is "new value"
// arr1 is now ['A', true, 2]
```



# เครื่องมือพื้นฐาน: เพิ่มและลบข้อมูล

จัดการข้อมูลที่หน้า Array

**.unshift()** - เพิ่มข้อมูลหนึ่งตัว  
(หรือมากกว่า) ไปยังข้างหน้าสุดของ  
Array

เพิ่มข้อมูลใหม่ไปยังตำแหน่งแรกของ Array

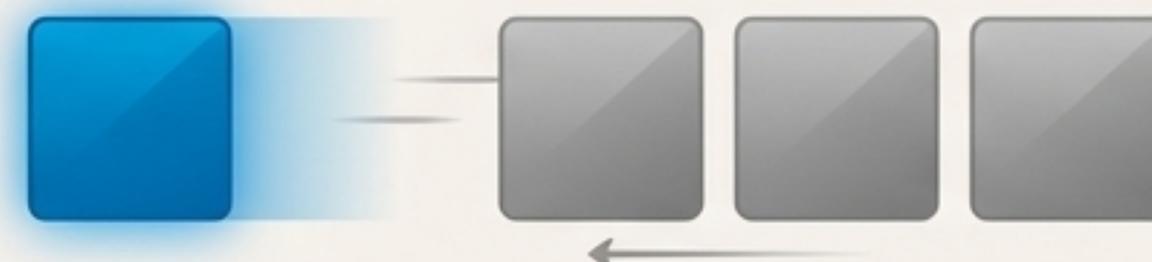
```
let arr1 = ['A', true, 2];
arr1.unshift("new value");
// arr1 is now ['new value', 'A', true, 2]
```



**.shift()** - ลบข้อมูลตัวแรกออกจาก  
Array และคืนค่าที่ลบออกมา

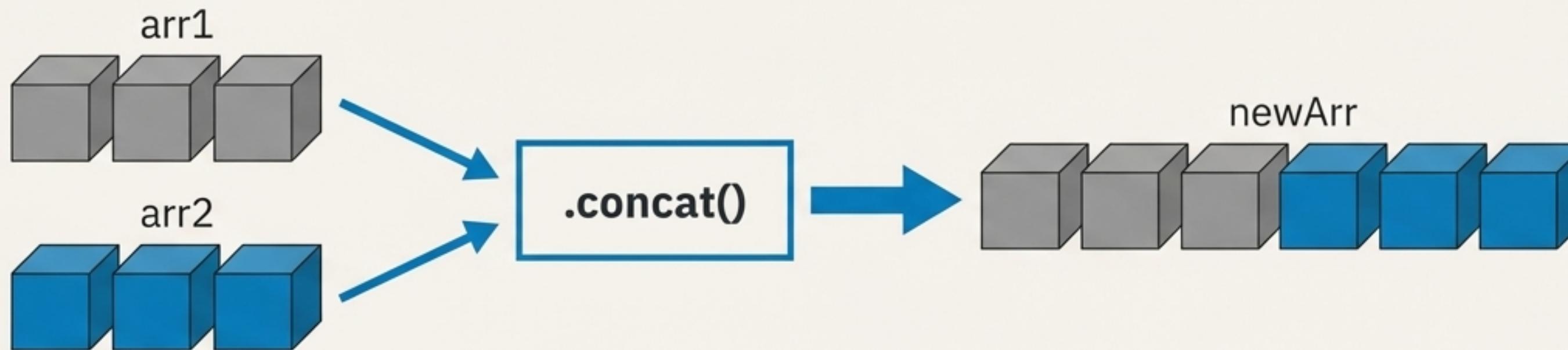
ลบข้อมูลตัวแรกออกจาก Array และส่งคืนค่าที่ถูกลบ

```
let removedValue = arr1.shift();
// removedValue is "new value"
// arr1 is now ['A', true, 2]
```



# การรวม Array

**.concat() - นำสอง Array มารวมกันเพื่อสร้างเป็น Array ใหม่**



```
let arr1 = ['A', true, 2];
let arr2 = ['B', false, 3];

let newArr = arr1.concat(arr2);
// newArr is ['A', true, 2, 'B', false, 3]
```

**Key Insight: `concat()` ไม่ได้เปลี่ยนแปลง Array เดิม แต่จะสร้าง Array ใหม่ขึ้นมา**

# โจทย์ต่อไป: จะทำงานกับทุกชิ้นในรายการได้อย่างไร?

สมมติว่าเรามีรายชื่อ 50 blog posts และต้องการแสดงผลชื่อเรื่องทั้งหมด เราจะทำซ้ำๆ ได้อย่างไร?

1. Run once at the start

2. Check before each loop

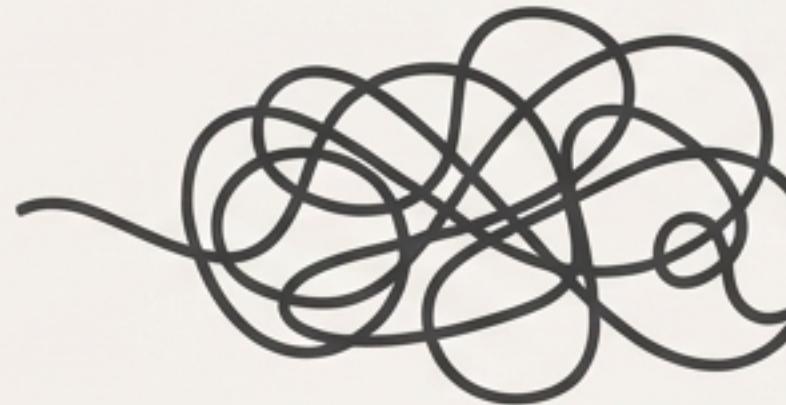
4. Run after each loop

```
for (initial-expression; condition; second-expression) {  
    ...  
    code in block  
}
```

3. Run if condition is true

```
const names = ['Justin', 'Sarah', 'Christopher'];  
  
for (let index = 0; index < names.length; index++) {  
    console.log(names[index]);  
}
```

# จาก Loop แบบเก่า... สู่ Method สมัยใหม่



## The Old Way (แบบดั้งเดิม)

```
const names = ['Justin', 'Sarah', 'Christopher'];

for (let i = 0; i < names.length; i++) {
  console.log(names[i]);
}
```

ต้องจัดการ index และเงื่อนไขเองทั้งหมด

## The Modern Way (แบบสมัยใหม่)

```
const names = ['Justin', 'Sarah', 'Christopher'];

names.forEach(name => {
  console.log(name);
});
```

โค้ดสั้นลง อ่านง่าย และสื่อความหมายชัดเจนกว่า

# Power Tool #1: .forEach()

“สำหรับข้อมูลทุกตัว... ให้ทำสิ่งนี้”

```
const numbers = [45, 4, 9, 16, 25];

// Using a named function
numbers.forEach(myFunction);

function myFunction(value) {
  console.log(value);
}

// Or as an Arrow Function (more common)
numbers.forEach(value => console.log(value));
```

## Output

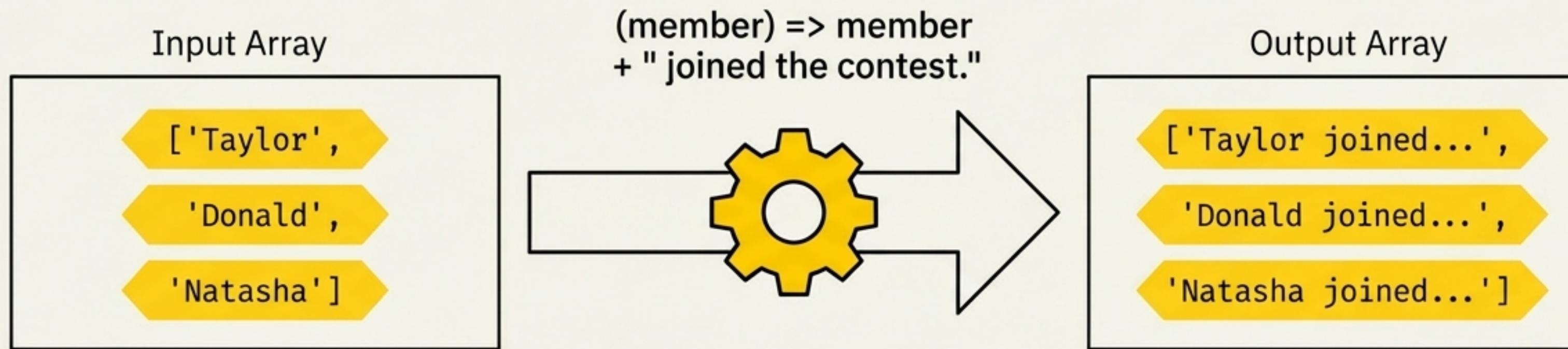
```
45
4
9
16
25
```

### Use Case:

เหมาะสำหรับการวนลูปเพื่อกำหนดงานบางอย่างกับข้อมูลแต่ละตัว โดยไม่ต้องการสร้าง Array ใหม่

# การแปลงข้อมูล: สร้าง Array ใหม่ด้วย `map()`

.map() จะสร้าง Array ใหม่ขึ้นมา โดยที่แต่ละ element คือผลลัพธ์จากการเรียกใช้ function กับ element เดิม



```
const finalParticipants = ["Taylor", "Donald", "Don", "Natasha", "Bobby"];
```

```
const announcements = finalParticipants.map((member) => {
  return member + " joined the contest.";
});
```

```
// announcements is a NEW array:
// ["Taylor joined...", "Donald joined...", ...]
```

# Power Tool #2: แปลงข้อมูลด้วย .map()

สร้าง Array ใหม่โดยนำข้อมูลแต่ละตัวใน Array เดิมมาแปลงค่า

## Example 1: Simple Transformation

[●, ●, ●] → [■, ■, ■]

```
const names = ["Taylor", "Donald", "Natasha"];
const announcements = names.map(member => {
  return member + " joined the contest.");
});
// announcements is now:
// ["Taylor joined...", "Donald joined...", ...]
```

## Example 2: Complex Transformation

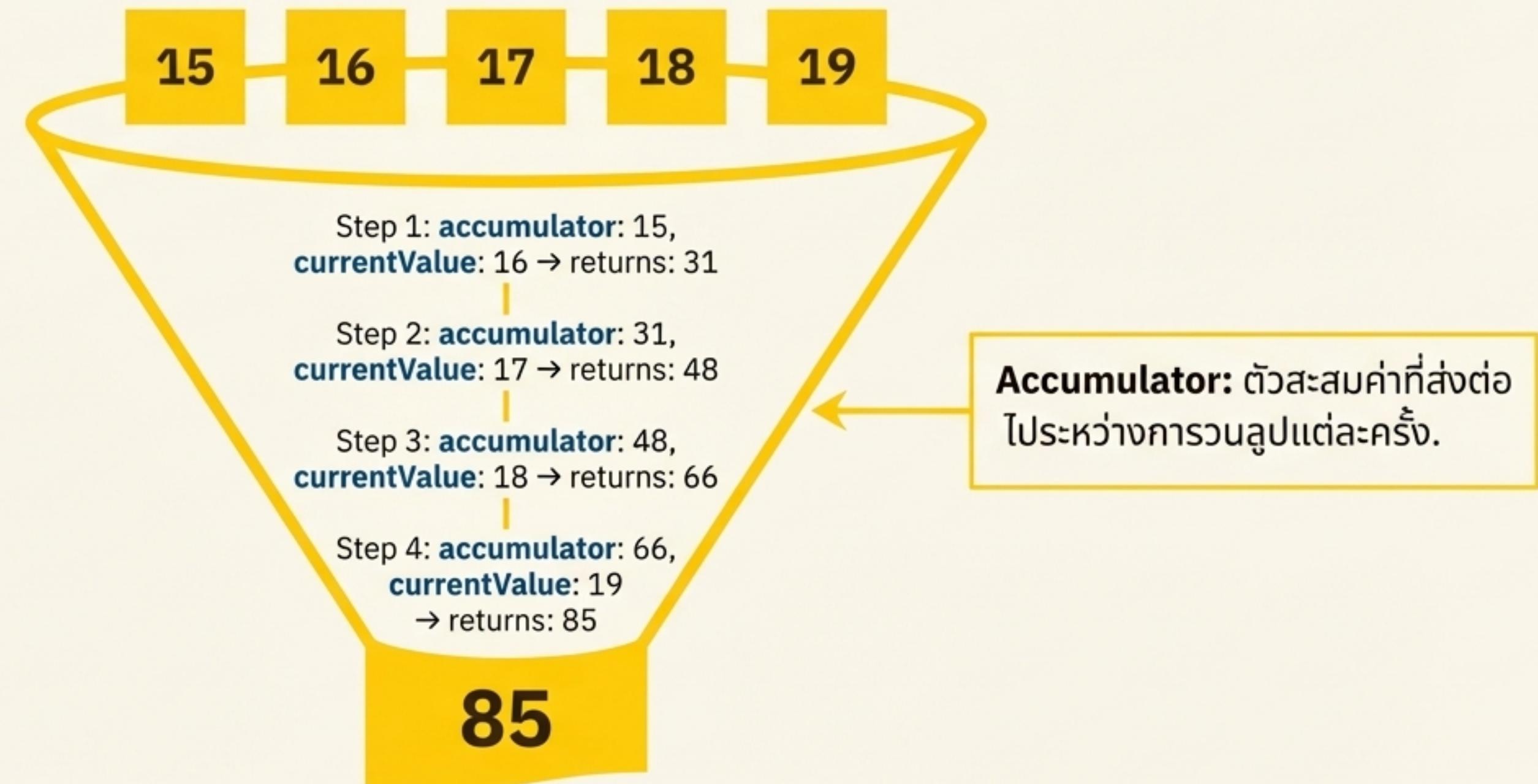
{ key: 1, value: 10 } → { '1': 10 }  
{ key: 2, value: 20 } → { '2': 20 }

```
const kvArray = [
  { key: 1, value: 10 },
  { key: 2, value: 20 }
];
const reformatted = kvArray.map(
  ({ key, value }) => ({ [key]: value })
);
// reformatted is now:
// [ { '1': 10 }, { '2': 20 } ]
```

**Key Insight:** .map() จะคืนค่าเป็น Array ใหม่เสมอ โดยที่ Array เดิมไม่เปลี่ยนแปลง

# การสรุปรวมยอด: ย่อๆ กัน Array ให้เหลือค่าเดียวด้วย `reduce()`

ใช้สำหรับ 'ลดรูป' ข้อมูลใน Array ทั้งหมดให้กลายเป็นค่าเดียว เช่น การหาผลรวม



```
const array = [15, 16, 17, 18, 19];
array.reduce(reducer);
```

# Power Tool #3: ยุบข้อมูลด้วย .reduce()

นำข้อมูลทุกตัวใน Array มาประมวลผลต่อเนื่องกันจนเหลือผลลัพธ์เพียงค่าเดียว

```
const numbers = [1, 2, 3, 4];  
  
const sum =  
  numbers.reduce((accumulator,  
    currentValue) => {  
    return accumulator +  
      currentValue;  
  });  
  
// sum is 10
```

## Visual Breakdown

How the `accumulator` works:

accumulator: 15 + currentValue: 16 → returns: 31

accumulator: 31 + currentValue: 17 → returns: 48

accumulator: 48 + currentValue: 18 → returns: 66

accumulator: 66 + currentValue: 19 → returns: 85

`accumulator` คือค่าที่สะสมไว้, `currentValue` คือค่าของสมาชิกตัวปัจจุบัน



# Workshop: ลงมือปฏิบัติ

เราได้เรียนรู้เครื่องมือต่างๆ ไปแล้ว ตอนนี้มาลองทำไปใช้แก้ไขอยีปัญหากัน

# แบบฝึกหัดที่ 1: วนลูปแสดงตัวอักษร

The Challenge: จาก Array ที่กำหนดให้, จงเขียนโค้ดเพื่อวนลูปและ `console.log` ตัวอักษรของแต่ละคำอອกมาทีละตัว

```
const furniture = ['Table', 'Chairs', 'Couch'];  
  
// Your code here...
```

**Expected Output (Hint)**

T  
a  
b  
l  
e  
c  
h  
a  
i  
r  
s  
...

# ແບບຝຶກຮັດທີ່ 2: ມາຂ້ອມຸລທີ່ຈໍາກັນ

The Challenge: ຈາກ Array ກົ່ງສາມຊຸດນີ້, ຈົນຫວ່າມີສາມາດີກຕັ້ງໃໝ່ນັບງານທີ່ປະກຸບຢູ່ໃນ **ທຸກ** Array

```
let values1 = ['Apple', 1, false];
let values2 = ['Fries', 2, true, 'Apple'];
let values3 = ['Mars', 9, 'Apple'];
```

// Your code here...

Expected Output (Hint)

```
['Apple']
```

# The JavaScript Array Journey

การเขียนโค้ด JavaScript สมัยใหม่ คือการเปลี่ยนจากการ “แก้ไข” Array โดยตรง...  
ไปสู่การ “แปลงข้อมูล” เพื่อสร้าง Array ใหม่

**Mutation**  
(การแก้ไขข้อมูลเดิม)



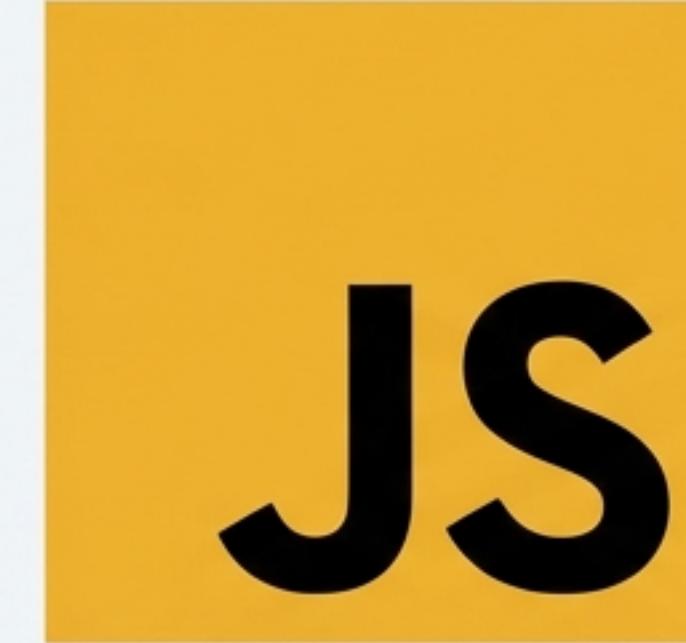
[...].`push()`  
[...].`pop()`  
[...].`shift()`

**Transformation**  
(การสร้างข้อมูลใหม่)



[...].`map()`  
[...].`reduce()`  
[...].`filter()`

**Embrace transformation for cleaner, more predictable code.**



# **Master JavaScript Loops: เขียนโค้ดช้าๆ อย่างมือโปร**

## ทำความเข้าใจ Loop ตั้งแต่ **for** ถึง **for...of** พร้อมแบบฝึกหัดลงมือทำจริง

ดัดแปลงจาก: Lecture-03 ArrayLoopIteratObj.pdf

# คุยเจว “งานช้าชา ก” แบบนี้ใหม?

ถ้าเราต้องการแสดงชื่อสมาชิกทุกคนในทีม สิ่งที่เราต้องทำคือ...

## แบบที่ไม่มี Loop (The Hard Way)

```
const members = ['Justin', 'Sarah',  
'Christopher', 'Anna'];  
  
console.log(members[0]); // Justin  
console.log(members[1]); // Sarah  
console.log(members[2]); // Christopher  
console.log(members[3]); // Anna
```

```
const members = ['Justin', 'Sarah',  
'Christopher', 'Anna'];  
  
console.log(members[0]); // Justin  
console.log(members[1]); // Sarah  
console.log(members[2]); // Christopher  
console.log(members[3]); // Anna  
// แล้วถ้ามีสมาชิก 100 คนล่ะ?
```

คำถาม: จะเกิดอะไรขึ้นถ้าเรามีข้อมูลเป็นร้อยเป็นพันรายการ?  
การเขียนโค้ดแบบนี้ช้าๆ ทั้งเหนื่อยและเสี่ยงต่อความผิดพลาด

# Loops: หัวใจของการทำงานอัตโนมัติ

Loop คือโครงสร้างคำสั่งที่ช่วยให้เราสั่งให้โปรแกรมทำงานซุดเดิมซ้ำๆ จนกว่าจะตรงตามเงื่อนไขที่กำหนดช่วยให้โค้ดของเราสั้นลง, อ่านง่ายขึ้น, และจัดการได้สะดวก

**\*\*เครื่องมือหลักในกระเปาของเรา:\*\***

ไอคอน	Loop Type	คำอธิบายสั้นๆ
	<b>for</b>	เมื่อรู้จำนวนรอบที่แน่นอน
	<b>while</b>	เมื่อต้องการทำซ้ำตราบใดที่เงื่อนไขยังเป็นจริง
	<b>do...while</b>	เหมือน `while` แต่การันตีว่าทำงานอย่างน้อย 1 ครั้ง
	<b>for...of</b>	วิธีที่ง่ายและกันสมัยสำหรับวนลูปข้อมูลใน Array หรือ String

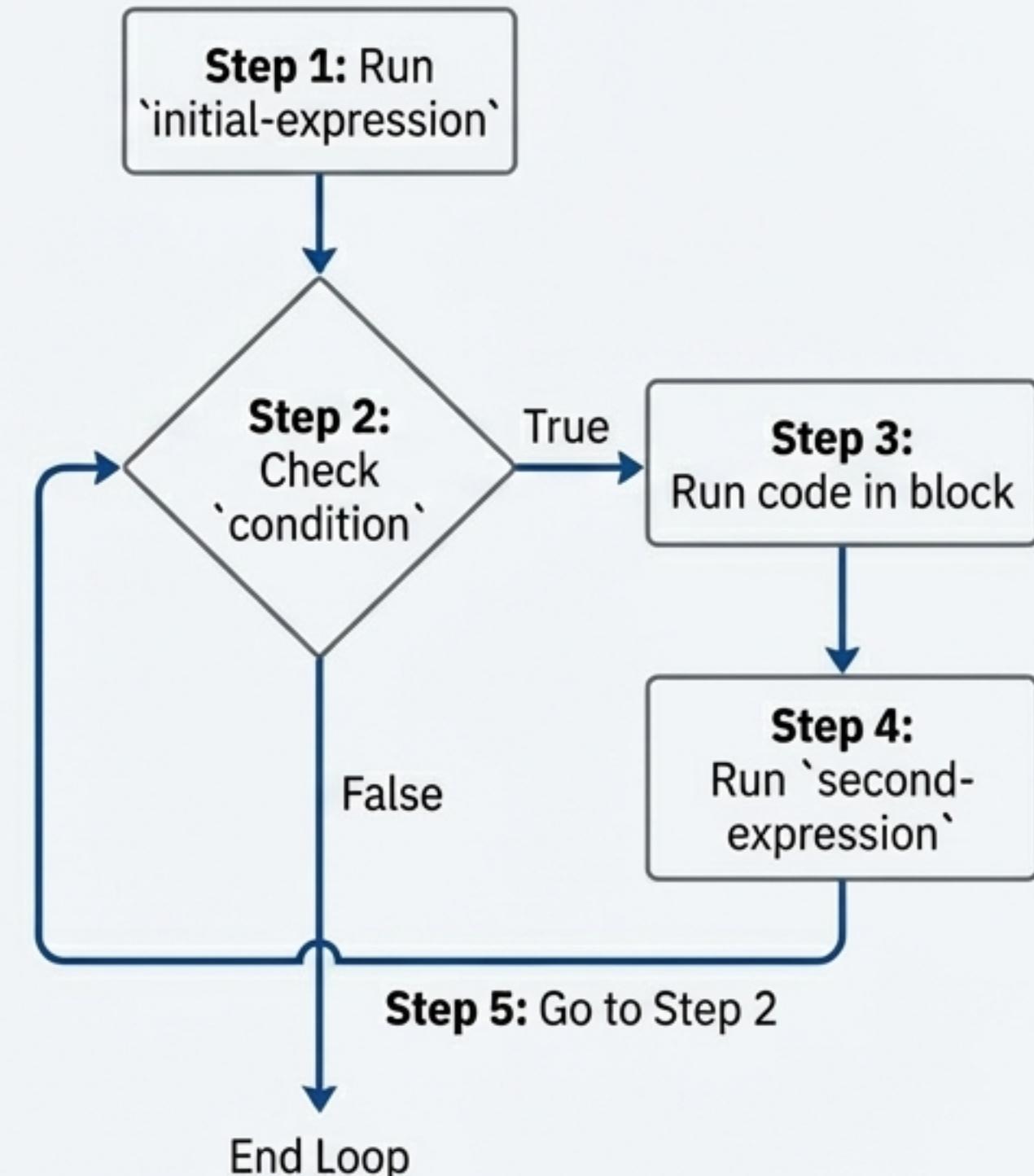
# ## `for` Loop: เมื่อรู้จำนวนรอบที่แน่นอน

เป็น Loop ที่นิยมใช้มากที่สุด เหมาะสำหรับการวนลูปตามจำนวนครั้งที่เรารู้ล่วงหน้า เช่น การวนลูปใน Array ที่เรารู้ขนาดของมัน

## โครงสร้างพื้นฐาน:

```
for (initial-expression; condition; second-expression) {
    // โค๊ดที่ต้องการให้ทำซ้ำ
}
```

- initial-expression:** กำหนดค่าเริ่มต้น (ทำงานครั้งเดียว)
- condition:** ตรวจสอบเงื่อนไขก่อนเข้า Loop (ถ้าเป็นจริงให้ทำต่อ)
- second-expression:** อัปเดตค่าหลังจบแต่ละรอบ (เช่น `i++`)



# ## \*\*`for` Loop in Action\*\*

มาดูตัวอย่างการวนลูปเพื่อแสดงรายชื่อสมาชิกทั้งหมดจาก Array

## Code Example:

```
const names = ['Justin', 'Sarah', 'Christopher'];

for (let index = 0; index < names.length; index++) {
  const name = names[index];
  console.log(name);
}
```

## Console Output:



```
Justin
Sarah
Christopher
```

**การทำงาน:** Loop จะเริ่มที่ `index = 0` และทำงานไปเรื่อยๆ ตราบใดที่ `index` ยังน้อยกว่า ความยาวของ Array (`names.length`) และในแต่ละรอบ `index` จะเพิ่มขึ้น 1

## ## \*`while` Loop: วนไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นเท็จ

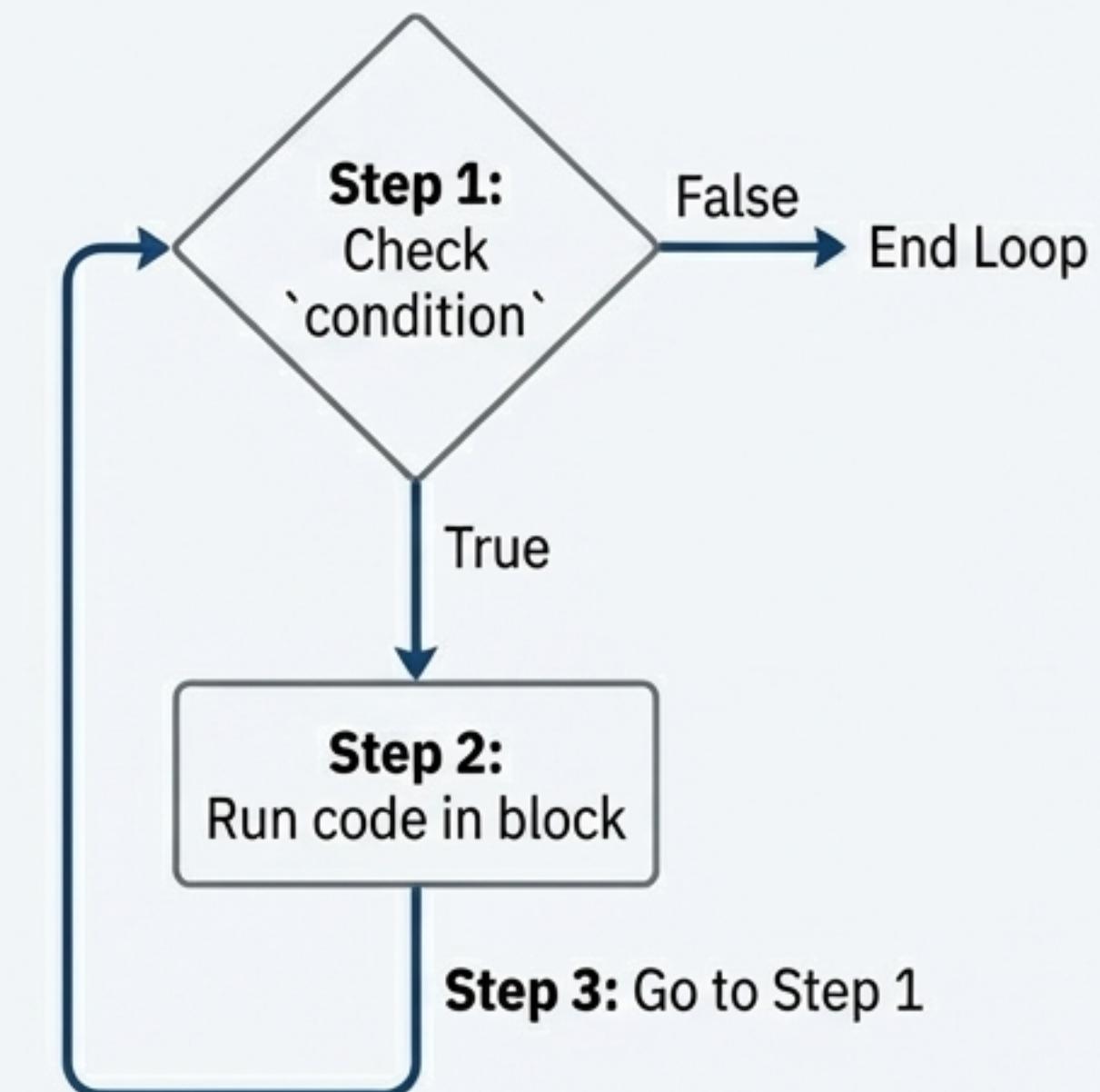
ใช้ `while` เมื่อเราไม่รู้จำนวนรอบที่แน่นอน แต่ต้องการให้ Loop ทำงานไปเรื่อยๆ ตราบใดที่เงื่อนไข (condition) ยังคงเป็นจริง

### โครงสร้างพื้นฐาน:

```
while (condition) {  
    // โค้ดที่ต้องการให้ทำซ้ำ  
    // **สำคัญ:** ต้องมีส่วนที่ทำให้ condition เปลี่ยนเป็นเท็จได้  
}
```



**ข้อควรระวัง:::** หากเงื่อนไขไม่มีวันเป็นเท็จ โปรแกรมจะทำงานไม่รู้จบ หรือที่เรียกว่า "Infinite Loop"



## ## \*`while` Loop in Action\*\*

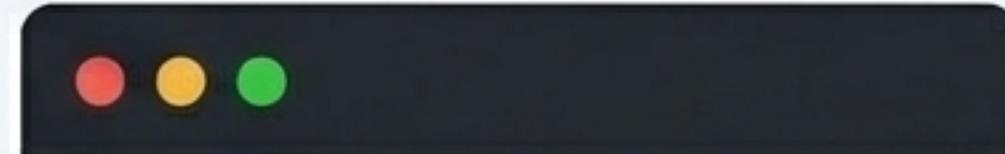
เราสามารถเขียนโค้ดจากตัวอย่างที่แล้วใหม่โดยใช้ `while` loop ได้เช่นกัน

### Code Example:

```
const names = ['Justin', 'Sarah', 'Christopher'];
let index = 0;

while (index < names.length) {
  const name = names[index];
  console.log(name);
  index++; // อัปเดตค่า index ใบແຕ່ລະຮອບ
```

### Console Output:



```
Justin
Sarah
Christopher
```

## ## `for...of`: วิธีที่ง่ายและทันสมัยในการวนลูป

เป็นวิธีที่ใหม่กว่า ถูกออกแบบมาเพื่อวนลูปบนสิ่งที่ "วนช้าได้" (Iterable) เช่น Array หรือ String โดยเฉพาะ ทำให้โค้ดสั้นและอ่านง่ายขึ้นมาก

### โครงสร้างพื้นฐาน:

```
for (const element of iterable) {  
    // ทำงานกับ 'element' ใบแต่ละรอบ  
}
```

### เปรียบเทียบกับ `for` แบบดั้งเดิม:

`for` Loop (แบบเก่า)	`for...of` Loop (แบบใหม่)
<pre>for (let i=0; i&lt;names.length; i++) {     console.log(names[i]); }</pre>	<pre>for (const name of names) {     console.log(name); }</pre>

สังเกตว่าเราไม่ต้องจัดการกับ `index` เองเลย!

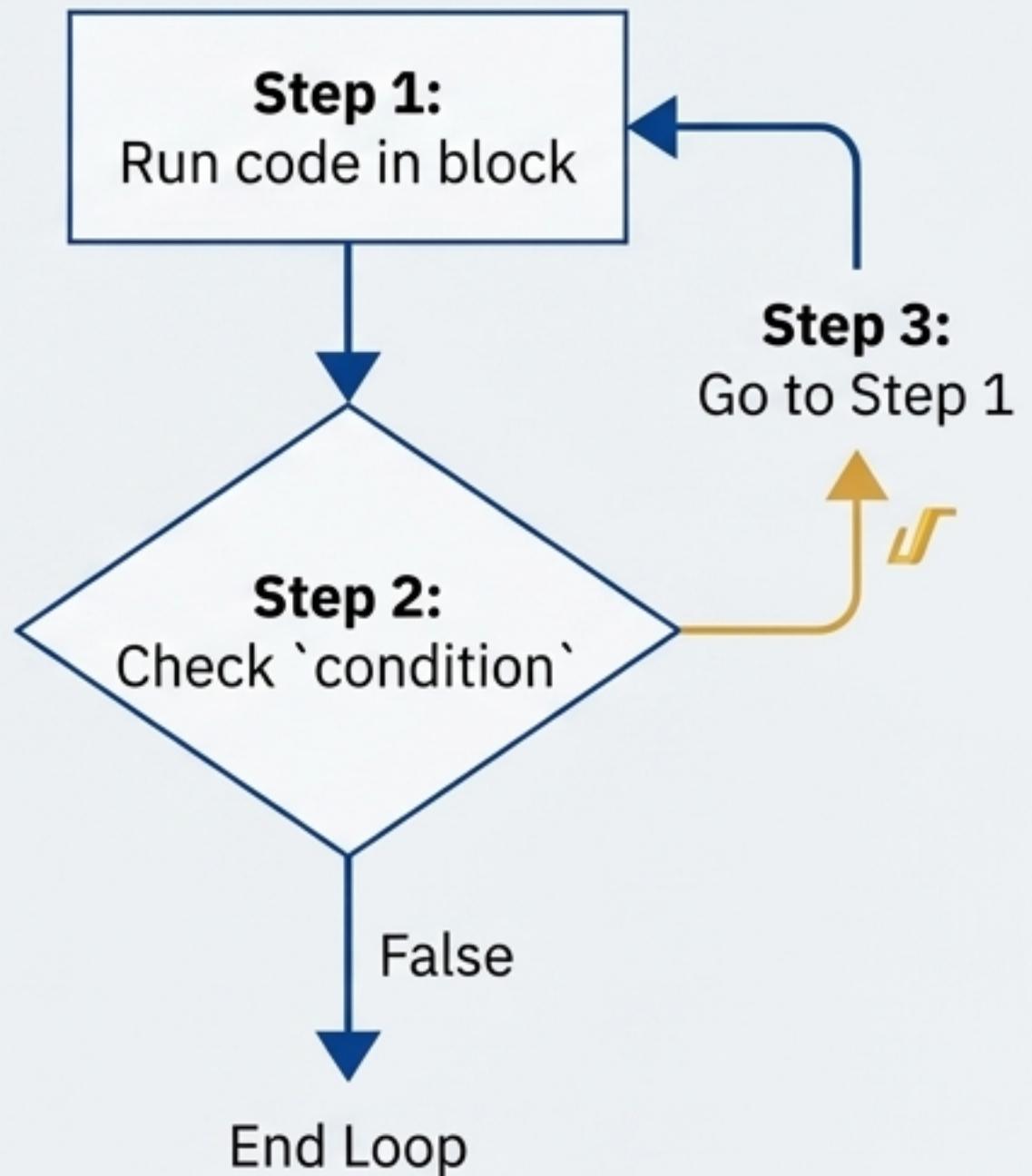
# do...while : การันตีว่าทำงานอย่างน้อย 1 ครั้ง

คล้ายกับ `while` แต่มีความแตกต่างที่สำคัญคือ `do...while` จะทำงานในบล็อกโค้ดก่อน และค่อยตรวจสอบเงื่อนไข

**โครงสร้างพื้นฐาน:**

```
do {
    // โค้ดส่วนนี้จะทำงานอย่างน้อย 1 ครั้งเสมอ
} while (condition);
```

**หมายสำคัญ:** ส่วนการวนที่เราต้องการให้มีการทำงานเกิดขึ้นก่อน 1 ครั้งเสมอ เช่น การขอ input จากผู้ใช้ครั้งแรก และค่อยตรวจสอบว่า input นั้นถูกต้องหรือไม่



# สรุป: สถานการณ์ไหน ควรใช้ Loop อะไร?

Loop Type	เหมาะสมกี่สุดเมื่อ...	ตัวอย่างสถานการณ์
`for` loop	รู้จำนวนรอบที่ต้องทำແນ้นอน	วนลูปตามความยาวของ Array, พิมพ์ตัวเลข 1 ถึง 100
`while` loop	ไม่รู้จำนวนรอบ แต่มีเงื่อนไขในการหยุด	รอรับข้อมูลจากผู้ใช้งานกว่าจะถูกต้อง, อ่านไฟล์จนกว่าจะหมด
`do...while` loop	ต้องการให้โค้ดทำงานอย่างน้อย 1 ครั้งเสมอ	แสดงเมนูให้ผู้ใช้เลือก แล้ววนลูป จนกว่าจะเลือก “ออก”
`for...of` loop	ต้องการเข้าถึงข้อมูลทุกตัวใน Array หรือ String	แสดงชื่อสินค้าทุกชิ้นในตะกร้า, นับจำนวนสระในคำ

# Workshop: ໄດ້ເວລາລົງນິ້ວທຳ!

ກຸາຍະກົມທີ່ດີທີ່ສຸດຂໍອກາຮັດນິ້ວປັບປຸງ

ເປີດ Code Editor ຂອງคຸນ ແລ້ວມາລອງແກ້ໄຈໂທຍແບບຝຶກຫັດຕ່ອໄປນີ້  
ເພື່ອກົດສອບຄວາມເຂົ້າໃຈເກື່ຽວກັບ Loop ໃນ JavaScript

ພຣ້ອມຮັບຍັງ? Let's Code!

# แบบฝึกหัดที่ 1: นับเลขและหาผลรวม



## โจทย์ที่ 1.1: นับถอยหลัง

- ใช้ `for` loop เพื่อพิมพ์ตัวเลขนับถอยหลังจาก 10 ไปถึง 1



## โจทย์ที่ 1.2: หาผลรวม

- กำหนดให้มี Array `const scores = [85, 92, 78, 95, 88];`
- ใช้ `for` loop หรือ `for...of` loop เพื่อคำนวณหา **ผลรวม** ของคะแนนทั้งหมดใน Array และแสดงผลลัพธ์ทางหน้าจอ

### Starter Code:

```
// Exercise 1.1  
// ... your code here ...
```

```
// Exercise 1.2  
const scores = [85, 92, 78, 95, 88];  
let sum = 0;  
// ... your code here ...  
console.log('Total score:', sum);
```

# แบบฝึกหัดที่ 2: วนลูปแบบมีเงื่อนไข



## โจทย์ที่ 2.1: ค้นหาชื่อ

- กำหนดให้มี Array `const guests = ['Alice', 'Bob', 'Charlie', 'David'];`
- ใช้ `while` loop เพื่อวนหาชื่อ "Charlie" ใน Array
- เมื่อเจอแล้ว ให้พิมพ์ข้อความว่า "Found Charlie!"  
แล้วหยุดการทำงานของ Loop กันที (Hint: ใช้ `break`)

(No starter code is provided on this slide  
to encourage independent thinking.)



## โจทย์ที่ 2.2: พิมพ์เฉพาะสระ

- กำหนดให้มี String `const message = "Hello World";`
- ใช้ `for...of` loop เพื่อวนลูปใน `message` และ  
พิมพ์เฉพาะตัวอักษรที่เป็นสระ (a, e, i, o, u) โดยไม่ต้อง  
สนใจตัวพิมพ์เล็ก/ใหญ่

# You've Mastered the Loops!

---

## สรุปสิ่งที่คุณได้เรียนรู้:

-  `for`: คือเพื่อนที่ดีที่สุดเมื่อคุณรู้จำนวนรอบที่แน่นอน
-  `while` & `do...while`: ยืดหยุ่นและทรงพลังเมื่อต้องทำงานกับเงื่อนไขที่ไม่แน่นอน
-  `for...of`: คือวิธีที่กันสมัยและอ่านง่ายที่สุดสำหรับการวนลูปใน Array และ String

การเข้าใจ Loop คือก้าวสำคัญในการเขียนโปรแกรมให้มีประสิทธิภาพ

## ก้าวต่อไป (Next Steps):

- ➡ ศึกษา Array Methods ที่ทำงานคล้าย Loop เช่น `.forEach()`, `.map()`, `.filter()`
- ➡ ลองนำ Loop ไปใช้แก้ปัญหาในโปรเจกต์ของคุณเอง
- ➡ ฝึกทำโจทย์เกี่ยวกับ Loop เพิ่มเติมจากเว็บไซต์ต่างๆ

**KEEP CODING & AUTOMATING!**

# JavaScript Objects: พิมพ์เขียวดิจิทัลสำหรับข้อมูล



# กายวิภาคของ Object: Properties และ Methods

## Properties (คุณสมบัติ)

ข้อมูลที่เกี่ยวข้องกับ Object

Example: book has `title`, `author`,  
`isAvailable`.

## Methods (ความสามารถ)

สิ่งที่ Object สามารถทำได้

Example: You can `checkIn` or  
`checkOut` a book.

```
const book = {  
    title: "1984",  
    isAvailable: false,  
  
    // This is a METHOD  
    checkIn: function(){  
        this.isAvailable = true;  
    },  
    checkOut: function(){  
        this.isAvailable = false;  
    }  
};
```

ภายใน Method คำว่า `this`  
จะหมายถึงตัว Object เอง

# ส่วนประกอบของ Object: Key และ Value

หัวใจของอ็อบเจกต์คือการรวมกลุ่มข้อมูลที่เกี่ยวข้องกันเป็นชุดของ "คู่ Key-Value" (Key-Value Pairs)

- **Key (คีย์):** คือชื่อหรือตัวระบุของข้อมูล (เปลี่ยนเส้นทางป้ายกำกับ) โดยจะเป็น String เสมอ
- **Value (ค่า):** คือข้อมูลจริง สามารถเป็นข้อมูลประเภทใดก็ได้: String, Number, Boolean, Array หรือแม้กระทั่ง Object อีก
- **Property (พร็อพเพอร์ตี้):** คือคู่ของ Key และ Value ที่ใช้เก็บ "ข้อมูล" หรือ "attributes" ของอ็อบเจกต์
- **Method (เมธอด):** คือ Property ที่มี Value เป็นฟังก์ชัน ใช้เพื่อแสดง "พฤติกรรม" หรือ "actions" ของอ็อบเจกต์

```
const book = {  
    // --- Properties ---  
    title: "1984",  
    author: "George Orwell",  
    isAvailable: false,  
  
    // --- Method ---  
    checkIn: function() {  
        this.isAvailable = true;  
    }  
};
```

# วิธีสร้าง Object: Object Literal

## วิธีที่ง่ายและนิยมที่สุด

Object Literal คือการสร้างอ็อบเจกต์โดยตรงด้วยเครื่องหมายวงเล็บปีกกา `{ }` เป็นวิธีที่กระชับและอ่านง่าย เหมาะสมสำหรับการสร้างอ็อบเจกต์ที่ไม่ซับซ้อนและใช้งานเฉพาะที่

### ขั้นตอน:

- ประกาศตัวแปร (แนะนำให้ใช้ `const`)
- ใช้เครื่องหมาย `{ }`
- ใส่คู่ Key-Value คันด้วยเครื่องหมายจุลภาค `(, `)`

## ตัวอย่าง: การสร้างอ็อบเจกต์หนังสือ

```
// สร้างอ็อบเจกต์ 'book' เพื่อเก็บข้อมูลเกี่ยวกับหนังสือเล่มหนึ่ง
const book = {
    title: "1984",
    author: "George Orwell",
    isAvailable: false
};

// แสดงผลอ็อบเจกต์ทั้งหมด
console.log(book);
// Output: { title: '1984', author: 'George Orwell', isAvailable: false }

// ตรวจสอบประเภทข้อมูล
console.log(typeof book);
// Output: 'object'
```

# การเข้าถึงและแก้ไขข้อมูล: Dot & Bracket Notation

เมื่อมีอ็อบเจกต์แล้ว เราสามารถเข้าถึงและจัดการข้อมูลภายในได้ 2 วิธี

## 1. Dot Notation ('.')

- รูปแบบ: object.key
- การใช้งาน: ง่ายและอ่านสะดวกที่สุด ใช้เมื่อชื่อ Key เป็นที่รู้จักและเป็นคำที่ถูกต้องตามกฎ

## 2. Bracket Notation ('[]')

- รูปแบบ: object['key']
- การใช้งาน:
  - ยืดหยุ่นกว่า ใช้เมื่อ:
    - ชื่อ Key มีการเว้นวรคหรืออักษรพิเศษ
    - ชื่อ Key มาจากค่าของตัวแปร (Dynamic Access)

## ตัวอย่างการใช้งาน

```
const user = {
  name: "Aziz Ali",
  "yearOfBirth": 1988 // Key เป็น String
};

// --- การเข้าถึงข้อมูล (Accessing Data) ---
// Dot notation
console.log(user.name); // "Aziz Ali"

// Bracket notation
console.log(user["yearOfBirth"]); // 1988

// --- การแก้ไขข้อมูล (Modifying Data) ---
user.name = "Anirach M.";
console.log(user.name); // "Anirach M."

// --- การเพิ่มข้อมูลใหม่ (Adding New Data) ---
user.email = "anirach.m@fitm.kmutnb.ac.th";
console.log(user); // แสดง object ที่มี property ใหม่
```

# ເຕີມຊື່ວົດໃຫ້ອົບເຈກຕໍດ້ວຍ Methods

ອົບເຈກຕໍໄມ້ໄດ້ມີໄວ້ແຄ່ເກີບຂ້ອມູລ  
ແຕ່ຍັງສາມາຮດ "ກະທຳ" ໄດ້ດ້ວຍ

ເມຣອດ (Method) ຄື່ອັພັງກົບໜັນທີເປັນພຣິວ  
ເພວົບຕີ້ຂອງອົບເຈກຕໍ ກຳໃຫ້ເຮົາສາມາຮດ  
ກຳນົດພຸດຕິກຣຣມທີ່ເກີ່ຍວຂ້ອງກັບຂ້ອມູລຂອງ  
ອົບເຈກຕໍນັ້ນໆ ໄດ້ ນີ້ຄື່ອສິ່ງທີ່ກຳໃຫ້  
Object ກົງພລັງ: ກາຣຣວມຂ້ອມູລ  
(Properties) ແລະພຸດຕິກຣຣມ (Methods)  
ໄວ້ໃນທີ່ເດືອກກັນ

ຕັວຢ່າງ: ເພີ່ມເມຣອດໃຫ້ `book`

```
const book = {  
    title: "1984",  
    author: "George Orwell",  
    isAvailable: true,  
  
    // Method: ສໍາຮັນກາຮືມທັນສືອ  
    checkOut: function() {  
        // 'this' refers to this book object  
        this.isAvailable = false;  
    },  
  
    // Method: ສໍາຮັນກາຮືນທັນສືອ  
    checkIn: function() {  
        // 'this' refers to this book object  
        this.isAvailable = true;  
    }  
};  
  
console.log(book.isAvailable); // true  
book.checkOut(); // ເຮືອກໄລ້ method  
console.log(book.isAvailable); // false  
book.checkIn(); // ເຮືອກໃຫ້ method  
console.log(book.isAvailable); // true
```

# คำสำคัญ 'this': บริบทคือทุกสิ่ง

## 'this' คืออะไร?

'this' เป็นคำสำคัญที่ใช้อ้างอิงถึง "บริบท" (context) ณ เวลาที่ฟังก์ชันถูกเรียกใช้

**ในกรณีของ Method:** 'this' จะซึ่ไปยังอ็อบเจกต์ที่ "เป็นเจ้าของ" และ "เรียกใช้" เมธอดนั้นๆ มันคือว่าที่เมธอดจะรู้ว่าต้องจัดการกับข้อมูลของอ็อบเจกต์ตัวไหน

**ตัวอย่าง:** เมื่อเราเรียก `book.checkIn()`, ภายในฟังก์ชัน `checkIn`, 'this' ก็คือ `book` นั่นเอง

## การทำงานของ 'this' ในเมธอด

```
const cat = {  
  name: "Pipey",  
  age: 8,  
  whatName: function() {  
    // 'this' ในที่นี่คือ object 'cat'  
    // เพราะ whatName() ถูกเรียกโดย cat  
    return this.name;  
  }  
};  
  
const dog = {  
  name: "Buddy",  
  whatName: cat.whatName // อ้มฟังก์ชันมาจาก cat  
};  
  
// บริบทที่อ 'cat', ดังนั้น this.name คือ cat.name  
console.log(cat.whatName()); // Output: Pipey  
  
// บริบทที่อ 'dog', ดังนั้น this.name คือ dog.name  
console.log(dog.whatName()); // Output: Buddy
```



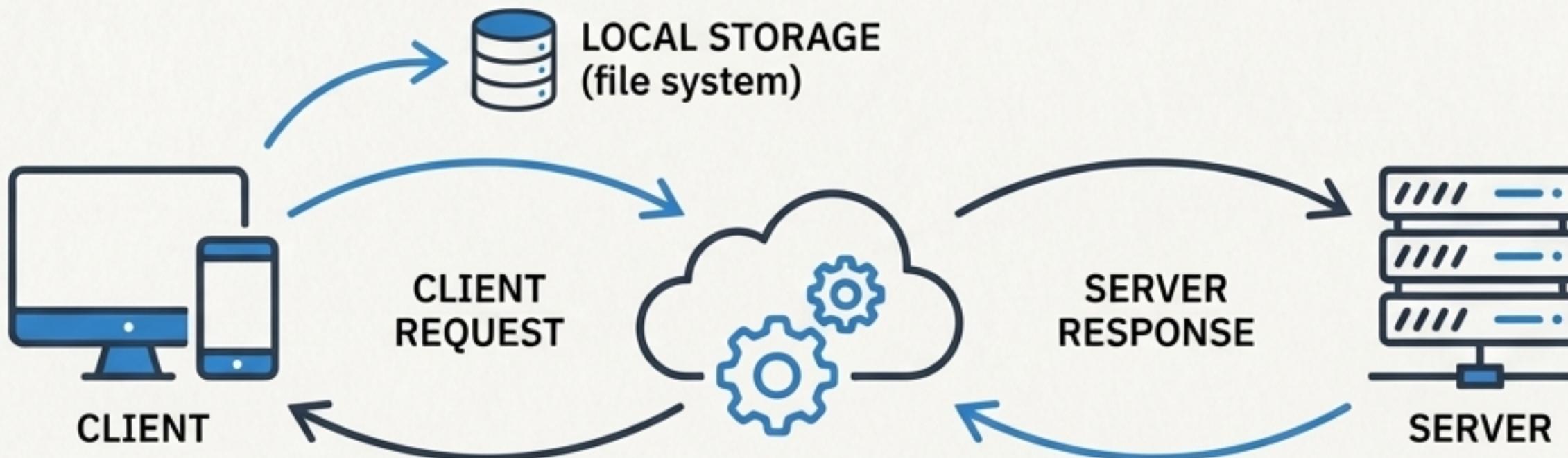
# การสื่อสารกับโลกภายนอก: JSON

เราจะส่ง Object ที่เราสร้างขึ้นผ่านเครือข่ายไปยัง Server ได้อย่างไร?  
**JSON** (JavaScript Object Notation)



- เป็นรูปแบบข้อความที่ อ่านง่าย
- ไม่ขึ้นกับภาษาโปรแกรม
- พื้นฐานมาจากโครงสร้างข้อมูล 2 อย่าง:  
คู่ของ name-value (Objects) และ  
รายการของ value (Arrays)

# Object เดินทาง: รู้จักกับ JSON



Why  
ทำไม Object ถึงสำคัญต่อเว็บ?

ในเว็บแอปพลิเคชันสมัยใหม่ Client (เบราว์เซอร์) และ Server ต้อง “พูดคุย” และเปลี่ยนข้อมูลกันผ่านเครือข่าย แต่เราไม่สามารถส่ง “JavaScript Object” ที่ซึ่งก้อนผ่านเครือข่ายได้โดยตรง

**Requirement:** Data should be transferable in a text format, easy for humans to read.

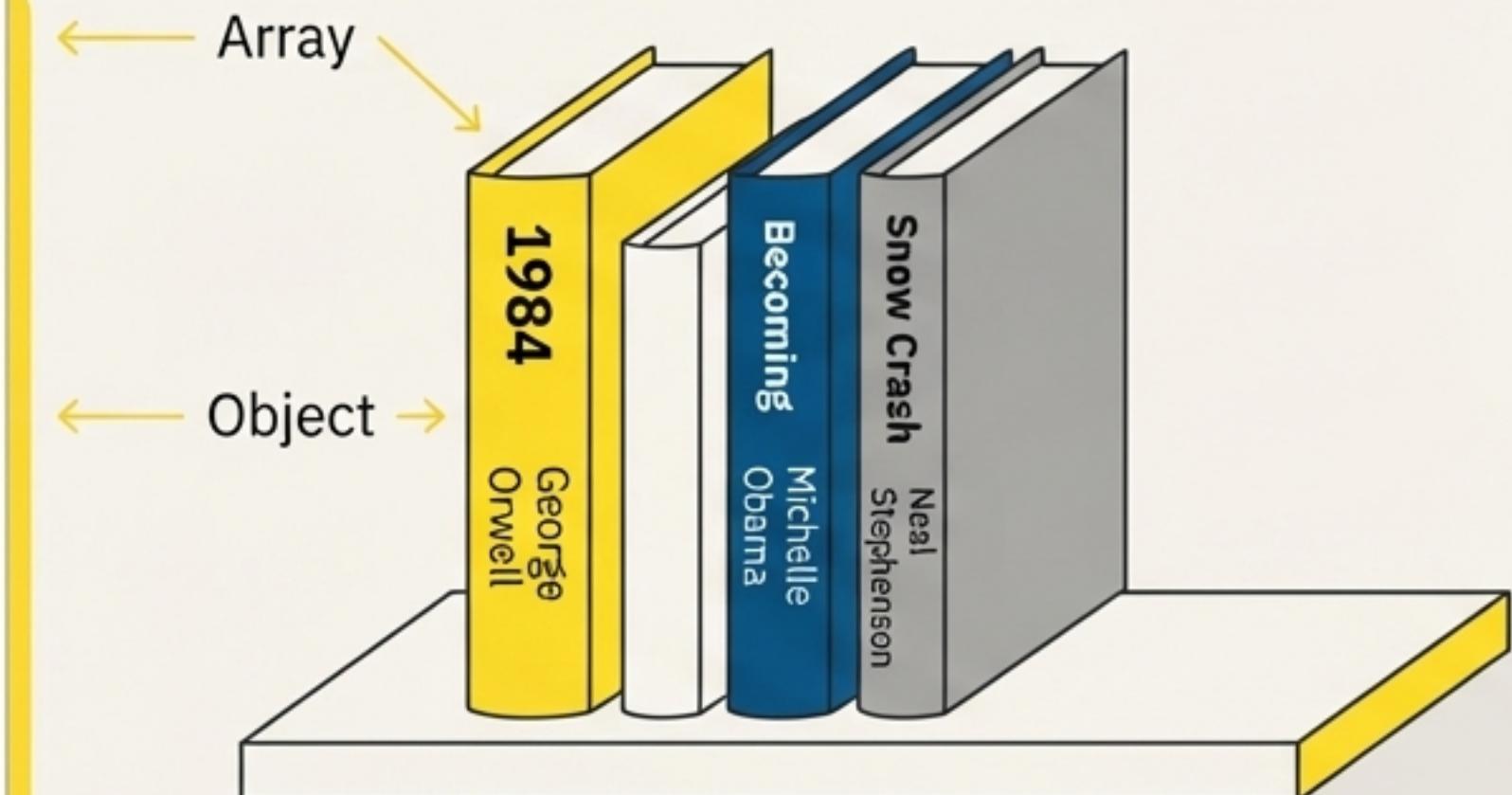
Solution  
**JSON = JavaScript Object Notation**

- เป็นรูปแบบข้อความ (text format) ที่ได้รับแรงบันดาลใจจาก syntax ของ JavaScript Object
- เป็นมาตรฐานสากลที่คอมพิวเตอร์และภาษาส่วนใหญ่เข้าใจ (Language independent)
- ง่ายต่อการอ่านเข้าใจง่าย และคอมพิวเตอร์ประมวลผลได้ง่าย

# ตัวอย่างในโลกจริง: รายการหนังสือ

ในแอปพลิเคชันจริง เราอาจจะทำงานกับ ‘รายการของ Object ที่มีโครงสร้าง’

```
// An Array of Objects
const myBooks = [
  {
    "title": "1984",
    "author": "George Orwell"
  },
  {
    "title": "Becoming",
    "author": "Michelle Obama"
  },
  {
    "title": "Snow Crash",
    "author": "Neal Stephenson"
  }
];
// This entire structure can be converted to JSON.
```

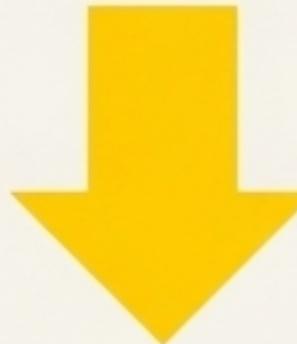


โครงสร้างข้อมูลแบบนี้สามารถแปลงเป็น JSON ได้อย่างง่ายดาย

# จาก Object สู่ JSON String ด้วย `JSON.stringify()`

## Step 1: Our JavaScript Object

```
const book = {  
    title: "1984",  
    author: "George Orwell",  
    isAvailable: false  
};
```



```
JSON.stringify(book)
```

- { } • ใช้สำหรับครอบ Object
- [ ] • ใช้สำหรับครอบ Array
- “ ” • Keys and string values must be in **double quotes**.

## Step 2: Converted to a JSON String

```
// Result: '{"title": "1984", "author": "George Orwell", "isAvailable": false}'  
'{\\"title\\":\\"1984\\", \\"author\\":\\"George Orwell\\", \\"isAvailable\\":false}'
```

# แปลงร่าง Object: `JSON.stringify()` และ `JSON.parse()`

JavaScript มีเครื่องมือในการตัวสำหรับแปลง Object และ JSON

## 1. `JSON.stringify()`

- หน้าที่: แปลง JavaScript Object ให้กลายเป็น JSON String (Serialize a JavaScript Object into an equivalent text string).
- ใช้เมื่อ: ต้องการส่งข้อมูลจาก Client ไป Server หรือต้องการบันทึก Objectลงในไฟล์/LocalStorage

## 2. `JSON.parse()`

- หน้าที่: แปลง JSON String กลับมาเป็น JavaScript Object ที่ใช้งานได้
- ใช้เมื่อ: ได้รับข้อมูลจาก Server และต้องการนำมาใช้งานในโค้ด JavaScript

## กระบวนการแปลงไป-กลับ

```
// 1. เริ่บต้นด้วย JavaScript Object  
const bookObj = {  
  title: "Becoming",  
  author: "Michelle Obama",  
  isAvailable: false  
};
```

JSON.stringify()

```
// 2. แปลงเป็น JSON String เพื่อส่งข้อมูล  
const bookJSON = JSON.stringify(bookObj);
```

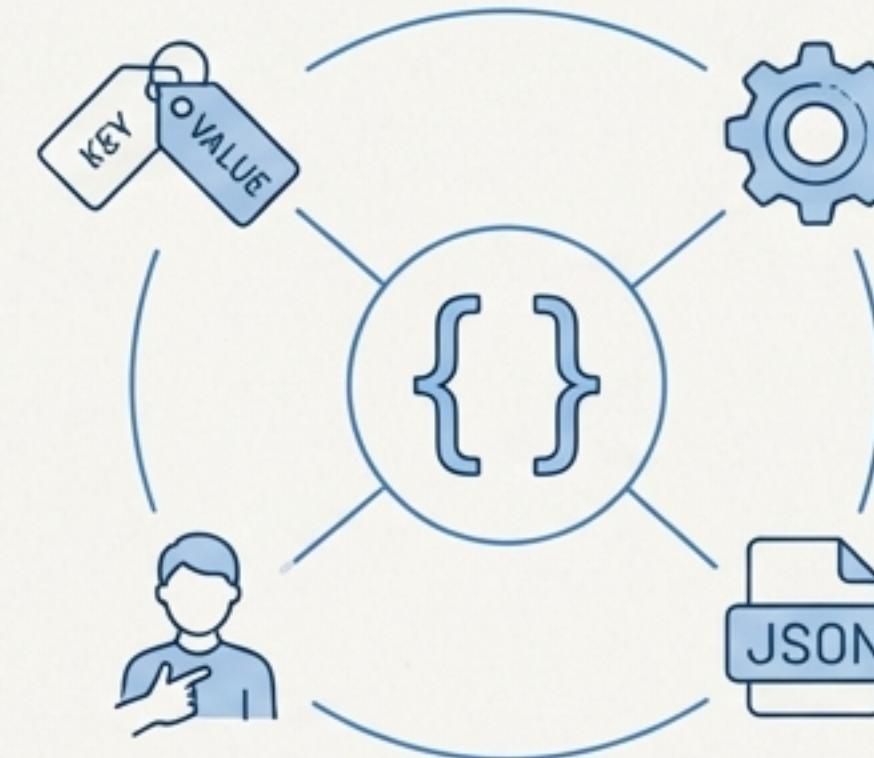
JSON.parse()

```
console.log(bookJSON);  
// Output: '{"title":"Becoming","author":"Michelle  
Obama","isAvailable":false}'  
console.log(typeof bookJSON); // "string"
```

```
// 3. เมื่อได้รับข้อมูลกลับมา แปลง JSON String กลับเป็น Object เพื่อใช้งาน  
const receivedBookObj = JSON.parse(bookJSON);
```

```
console.log(receivedBookObj.author); // "Michelle Obama"  
console.log(typeof receivedBookObj); // "object"
```

# สรุป: พลังของ JavaScript Objects



คุณได้เรียนรู้:

- **แก่นแท้:** Object คือชุดของคู่ Key-Value ที่รวมข้อมูล (Properties) และพฤติกรรม (Methods) ไว้ด้วยกัน
- **การสร้าง:** ใช้ Object Literal {} เป็นวิธีที่ง่ายและรวดเร็วที่สุด
- **การใช้งาน:** เข้าถึงและจัดการข้อมูลด้วย Dot (`.`) และ Bracket (`[]`) Notation
- **หัวใจของเมธอด:** `this` คือบริบทที่ทำให้เมธอดรู้ว่ากำลังทำงานกับอ็อบเจกต์ใด
- **การประยุกต์ใช้:** Object คือพื้นฐานของ JSON ซึ่งเป็นมาตรฐานในการแลกเปลี่ยนข้อมูลบนเว็บ

*mastery over Objects is mastery over modern JavaScript.*