

Node JS

Nodejs คืออะไร

NodeJS คือ Cross Platform Runtime Environment สำหรับฝั่ง Server เป็น Open Source และ Library ที่ใช้สำหรับพัฒนาเว็บแอปพลิเคชันต่าง ๆ ด้วยภาษา JavaScript เหมาะสำหรับการสร้างแอปพลิเคชันที่ต้องการใช้ข้อมูลจำนวนมาก และนิยมใช้ในการพัฒนาแอปพลิเคชันที่ใช้ข้อมูลแบบ Realtime สามารถทำงานได้ทุกระบบปฏิบัติการ โดยถูกนำมาเป็น Web Server, IoT, Webkit, TVOS, OS และอื่น ๆ เป็นต้น

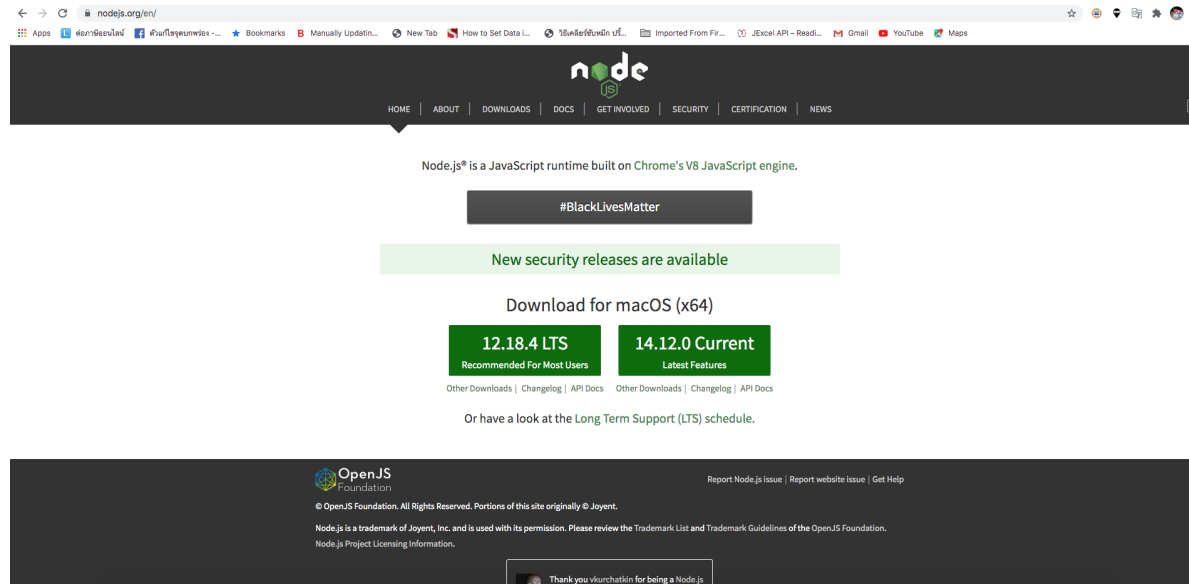
ประโยชน์ของ Nodejs

Nodejs มีประโยชน์อย่างหลากหลาย ดังนี้

1. ช่วยพัฒนาเว็บไซต์ให้ง่ายและรวดเร็วขึ้น
2. เขียนโค้ดเข้าใจง่าย ไม่ยุ่งยาก
3. มี Library ฟรีให้เลือกใช้ได้ไม่อัน !
4. ใช้ทรัพยากรน้อย ไม่เปลืองพื้นที่
5. เรียนรู้ได้เร็ว ไม่จำเป็นต้องเรียนภาษา Programming เฉพาะอื่น ๆ
6. ช่วยให้นักพัฒนา JavaScript มีโอกาสได้ทำงานหลากหลายมากขึ้น

ติดตั้ง Node.js

Download โปรแกรม Node.js จาก website ของ Node.js ได้โดยตรงที่ www.nodejs.org



หลังจากโหลดแล้วติดตั้งแล้ว เข้า commandLine พิมพ์ node-v

```
G:\c>node -v  
v21.1.0
```

NPM คือ Node package manager เป็นตัวจัดการ package เสริมต่างๆ ที่เราเอามาใช้กับ Nodejs

พิมพ์ npm -v

```
G:\c>npm -v
10.2.0
```

Variable

ในการประกาศตัวแปรใน Node.js จะมีอยู่ ขอบเขตอยู่ 3 ชนิดคือ

1. var มีขอบเขตการทำงานในระดับ function

```
> var a=10;
{
  var a = 20;
}
console.log(a);
```

20

ผลลัพธ์ คือ 20 เพราะเราประกาศค่า a=10 แล้วมาประกาศในบล็อกสโคป (block scope) อีกครั้ง a=20 ดังนั้นค่า a จึงเปลี่ยนเป็น 20 หากเรานำตัวแปร a ไปใช้งาน จะได้ค่า 20 นี้หมายถึง การประกาศตัวแปรด้วย var ค่าตัวแปรจะสามารถแก้ไขได้

2. let มีขอบเขตการทำงานในระดับ block เมื่อมีการประกาศตัวแปรใน block หรือ(ที่มีเครื่องหมายปีกกา { }) ตัวแปรจะทำงานแค่ใน block นั้นๆ

```
> var a=10;  
  {  
    let a = 20;  
    console.log(a);  
  }  
  console.log(a);
```

20

10

ผลลัพธ์ใน block คือ 20 และ นอก block คือ 10 เนื่องจาก ตัวแปร let ทำงานใน block เท่านั้น

3.const เป็นการประกาศตัวแปรแบบค่าคงที่ไม่สามารถประกาศซ้ำได้

```
> const a = 10;  
  {  
    let a = 20;  
    console.log(a);  
  }  
  {  
    var a = 30;  
    console.log(a);  
  }  
  console.log(a);
```

✖ Uncaught SyntaxError: Identifier 'a' has already been declared

เมื่อประกาศตัวแปรด้วย const แล้วมีการนำไปประกาศซ้ำ โปรแกรมจะแสดง error ออกมาว่า 'a' has already been dclared ซึ่งหมายความว่า ตัวแปร a มีการประกาศใช้งานแล้ว

ตัวดำเนินการ (Operators)

ตัวดำเนินการ (Operator) คือสัญลักษณ์ของภาษา JavaScript ที่ใช้ในคำนวณ เปรียบเทียบ หรือดำเนินการกับข้อมูลเพื่อให้ได้ผลลัพธ์ใหม่ ตัวดำเนินการจะใช้งานกับตัวแปร ซึ่งอาจมีตั้งแต่หนึ่งตัวหรือหลายตัวก็ได้ ในภาษา JavaScript นั้นมีตัวดำเนินการอยู่หลายประเภท และแต่ละประเภทมีหน้าที่การทำงานที่แตกต่างกัน ดังต่อไปนี้

1. ตัวดำเนินการกำหนดค่า

ตัวดำเนินการกำหนดค่า (Assignment operator) คือตัวดำเนินการที่ใช้สำหรับกำหนดหรืออัปเดตค่าให้กับตัวแปรหรือค่าคงที่ โดยจะใช้เครื่องหมาย = ดังตัวอย่างต่อไปนี้

```
let num = 0;  
let weight = 52.4;  
let name = "BOY";  
let isban = false;  
let x, y, z;  
  
x = 5;  
y = 10;  
z = 20;
```

2. ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic operators) คือ ตัวดำเนินการที่ใช้ในการคำนวณข้อมูลที่เป็นตัวเลข เช่น การบวก ลบ คูณ หาร

ตัวดำเนินการ	หน้าที่	ตัวอย่างการใช้งาน
+	การบวก	$a + b$
-	การลบ	$a - b$
*	การคูณ	$a * b$
/	การหาร	a / b
%	การหารเอาเศษ	$a \% b$
**	การยกกำลัง	$a ** b$

ตัวอย่างการใช้งานตัวดำเนินการในทางคณิตศาสตร์

Nodejs Code	Output
<pre>let a=5,b=3; console.log(a,b) console.log(a+b) console.log(a-b) console.log(a*b) console.log(a/b) console.log(a%b) console.log(a**b)</pre>	<pre>5 3 8 2 15 1.6666666666666667 2 125</pre>

3. ตัวดำเนินการทางตรรกศาสตร์

ตัวดำเนินการตรรกศาสตร์ (Logical operators) ใช้ดำเนินการทางตรรกศาสตร์ แล้วให้ผลลัพธ์ออกมาเป็นค่า true (จริง) หรือ false (เท็จ)

ตัวดำเนินการ	หน้าที่	ตัวอย่างการใช้งาน
&&	AND	a && b
	OR	a b
!	NOT	!a

จากตัวอย่างการใช้งาน

a && b จะได้ค่าเป็น true ถ้า a และ b มีค่าเป็น true ไม่เช่นนั้นจะได้ false

a || b จะได้ค่าเป็น true ถ้า a หรือ b มีค่าเป็น true ไม่เช่นนั้นจะได้ false

!a จะได้ค่าตรงข้ามกับ a เช่น ถ้า a มีค่าเป็น true จะได้ค่าเป็น false แต่ถ้า a มีค่าเป็น false จะได้ค่า true

4. ตัวดำเนินการเปรียบเทียบค่า

ตัวดำเนินการเปรียบเทียบ (Comparison operators) คือตัวดำเนินการที่ใช้สำหรับเปรียบเทียบระหว่างค่าสองค่า โดยจะให้ผลลัพธ์ออกมาเป็นค่า true (จริง) หรือ false (เท็จ)

ตัวดำเนินการ	หน้าที่	ตัวอย่างการใช้งาน
==	เท่ากับ	a == b
!=	ไม่เท่ากับ	a != b
<	น้อยกว่า	a < b
>	มากกว่า	a > b
<=	น้อยกว่าหรือเท่ากับ	a <= b
>=	มากกว่าหรือเท่ากับ	a >= b
===	เท่ากันทั้งค่าและประเภทข้อมูล	a === b
!==	ไม่เท่ากันทั้งค่าและประเภทข้อมูล	a !== b
?	ตรวจสอบเงื่อนไขแบบสั้น	a == b ? true : false;

5. ตัวดำเนินการกับข้อความ

ตัวดำเนินการที่ใช้กับข้อความจะมีอยู่ตัวเดียวคือ ตัวดำเนินการ + ใช้สำหรับการรวมข้อความเข้าด้วยกัน ดังตัวอย่าง

Nodejs code	Output
<pre>let str1 = "ABC" , str2 = "XYZ"; let str3 = str1 + str2; console.log('str1',str1); console.log('str2',str2); console.log('str3',str3);</pre>	<pre>str1 ABC str2 XYZ str3 ABCXYZ</pre>

6. ตัวดำเนินการกำหนดค่าแบบรวม

ตัวดำเนินการกำหนดค่าแบบรวม (Compound assignment operators) คือการใช้งานตัวดำเนินการกำหนดค่า = ร่วมกับตัวดำเนินการประเภทอื่น เช่น $a += b$ จะมีค่าเท่ากับ $a = a + b$ ตัวดำเนินการประเภทยังมีอยู่หลายตัว ดังนี้

ตัวดำเนินการ	ตัวอย่างการใช้งาน	มีค่าเท่ากับ
+=	$a += b$	$a = a + b$
-=	$a -= b$	$a = a - b$
*=	$a *= b$	$a = a * b$
/=	$a /= b$	$a = a / b$
%=	$a \% = b$	$a = a \% b$
**=	$a ** = b$	$a = a ** b$

7. ตัวดำเนินการเพิ่มค่าและลดค่า

ตัวดำเนินการประเภทนี้จะใช้กับตัวแปรที่เป็นตัวเลข มีอยู่ 2 ตัว คือ ++ สำหรับเพิ่มค่า และ -- สำหรับลดค่า ดังตัวอย่างต่อไปนี้

Nodejs Code	Output
<pre>let n=5,m=10; console.log('n =',n, 'm =',m) n++; m--; console.log('n =',n) console.log('m =',m)</pre>	<pre>n = 5 m = 10 n = 6 m = 9</pre>

การใช้ตัวดำเนินการเพิ่มค่า ลดค่า สามารถจะใส่ไว้ข้างหน้าหรือข้างหลังก็ได้ หากใส่ไว้ด้านหน้าจะเป็นการเพิ่มหรือลดค่าก่อนที่จะใช้ค่าตัวแปร แต่ถ้าใส่ไว้ข้างหลังจะเป็นการใช้ค่าตัวแปรก่อนที่จะเพิ่มหรือลดค่า ดังตัวอย่างต่อไปนี้

Nodejs Code	Output
<pre>let a=5,b=10,c=15,d=20; console.log('a =',a, 'b =',b, 'c =',c, 'd =',d); console.log('++a',++a); console.log('b++',b++); console.log('--c',--c); console.log('d--',d--); console.log('a =',a, 'b =',b, 'c =',c, 'd =',d);</pre>	<pre>a = 5 b = 10 c = 15 d = 20 ++a 6 b++ 10 --c 14 d-- 20 a = 6 b = 11 c = 14 d = 19</pre>

คำสั่ง if

เป็นคำสั่งตรวจสอบเงื่อนไข เพื่อควบคุมการทำงานของโปรแกรมให้ทำงานตามเงื่อนไขที่กำหนด

รูปแบบของคำสั่ง if

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

ตัวอย่างการใช้คำสั่ง if

nodejs Code	Output
<pre>let total = 45; if(total < 50){ console.log('total น้อยกว่า 50') }</pre>	<pre>PS G:\c> node app.js total น้อยกว่า 50</pre>

จากตัวอย่างโปรแกรมจะแสดงว่า total น้อยกว่า 50

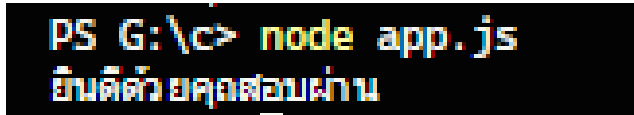
คำสั่ง if else

คำสั่ง if โปรแกรมจะทำงานเมื่อเงื่อนไขเป็นจริง และถ้าเงื่อนไขไม่เป็นจริงโปรแกรมจะข้ามการทำงานในบล็อกนั้นไป เราสามารถกำหนดบล็อกของคำสั่ง else เพื่อให้โปรแกรมทำงานในกรณีที่เงื่อนไขของคำสั่ง if ไม่เป็นจริงได้ คำสั่ง else นั้นจะต้องใช้ร่วมกับคำสั่ง if เสมอ

รูปแบบของคำสั่ง if else

```
if (condition) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```

ตัวอย่างการใช้คำสั่ง if else

nodejs Code	Output
<pre>let total = 55; if (total < 50) { console.log('คุณสอบไม่ผ่าน') } else { console.log('ยินดีด้วยคุณสอบผ่าน') }</pre>	

จากตัวอย่างนี้ โปรแกรมจะแสดงคำว่า “ยินดีด้วย คุณสอบผ่าน” เพราะ total ไม่น้อยกว่า 50


คำสั่ง else if

กรณีที่ต้องการให้โปรแกรมเลือกการทำงานได้มากกว่า 2 ทางเลือก เราสามารถใช้คำสั่ง else if เพื่อเพิ่มเงื่อนไขได้

รูปแบบคำสั่ง else if

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    // block of code to be executed if the condition1 is false and condition2 is false  
}
```

ตัวอย่างการใช้คำสั่ง else if

Nodejs Code	Output
<pre>let n = -5; if(n > 0){ console.log('ตัวแปร n มีค่าเป็นบวก') }else if (n < 0){ console.log('ตัวแปร n มีค่าติดลบ') }else{ console.log('ตัวแปร n มีค่าเป็น 0') }</pre>	 <pre>PS G:\c> node app.js ตัวแปร n มีค่าติดลบ</pre>

คำสั่ง switch case

คำสั่ง switch case เป็นคำสั่งควบคุมการทำงานที่คล้ายกับคำสั่ง if แต่จะใช้สำหรับเปรียบเทียบโดยตรงกับค่าที่กำหนดเท่านั้น ในขณะที่คำสั่ง if สามารถสร้างเงื่อนไขจากตัวดำเนินการต่างๆ ได้

รูปแบบของคำสั่ง switch case

```
switch (input) {  
    case value1:  
        // code block  
        break;  
    case value2:  
        // code block  
        break;  
    default:  
        // code block  
}
```

การทำงานของคำสั่ง switch case โปรแกรมจะตรวจสอบค่าของ input ที่กำหนด หากค่าที่กำหนดตรงกับ case ใด ก็จะเริ่มทำงานตามคำสั่งใน case นั้น ไปจนเจอคำสั่ง break; หากไม่เจอคำสั่ง break; โปรแกรมจะทำงานคำสั่งที่เหลือทั้งหมด

ตัวอย่างการใช้งานคำสั่ง switch case

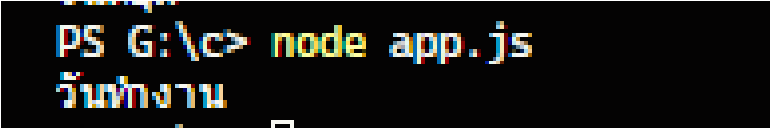
Nodejs Code	Output
<pre>let sex = "M"; switch (sex) { case "M": console.log('เพศชาย') break; case "F": console.log('เพศหญิง') break; default: console.log('เพศทางเลือก') }</pre>	<pre>PS G:\c> node app.js เพศชาย</pre>

จากตัวอย่างนี้ หากไม่ใส่ break; ในแต่ละ case จะได้ผลลัพธ์เหมือนตัวอย่างนี้

Nodejs Code	Output
<pre>let sex = "M"; switch (sex) { case "M": console.log('เพศชาย') case "F": console.log('เพศหญิง') default: console.log('เพศทางเลือก') }</pre>	<pre>PS G:\c> node app.js เพศชาย เพศหญิง เพศทางเลือก</pre>

การกำหนดเงื่อนไขแบบ OR

คำสั่ง switch case จะตรวจสอบค่าตัวแปรที่กำหนดว่าตรงกับแต่ละ case หรือไม่ ไม่สามารถกำหนดเงื่อนไขเหมือนคำสั่ง if ได้ แต่ถ้าช่วงข้อมูลไม่มากนัก และต้องการผลลัพธ์ที่เหมือนกัน สามารถกำหนดให้โปรแกรมทำหลายๆ case ได้ ดังตัวอย่างนี้

Nodejs Code	Output
<pre>1 // 2 let day = "Wen"; 3 switch (day) { 4 case "Mon": 5 case "Tue": 6 case "Wen": 7 case "Thu": 8 case "Fri": 9 console.log('วันทำงาน') 10 break; 11 case "Sat": 12 case "Sun": 13 console.log('วันหยุด') 14 break; 15 default: 16 console.log('ชื่อวันไม่ถูก') 17 }</pre>	 <pre>PS G:\c> node app.js วันทำงาน</pre>

คำสั่ง for loop

คำสั่ง for loop เป็นคำสั่งควบคุมการทำงานแบบวนซ้ำที่ใช้สำหรับควบคุมเพื่อให้โปรแกรมทำงานบางอย่างซ้ำๆ ในขณะที่เงื่อนไขเป็นจริง โดยทั่วไปแล้วเรามักใช้คำสั่ง for loop ในกรณีลูปที่จำนวนการวนรอบที่แน่นอน

รูปแบบของคำสั่ง for loop

```
for (initialize; condition; changes) {  
    // code block to be executed  
}
```

โดยที่ initialize คือการประกาศตัวแปรและค่าเริ่มต้นสำหรับใช้ภายใน Loop

condition คือเงื่อนไขในการวน Loop

changes คือการเปลี่ยนแปลงค่าตัวแปรที่ประกาศ

ตัวอย่างการใช้งานคำสั่ง for loop

Nodejs Code	Output
<pre>let rand; let min = 100, max = 0; console.log('start loop') for (i = 1; i ≤ 10; i++) { rand = Math.floor(Math.random() * 100) + 1; if (rand < min) min = rand; if (rand > max) max = rand; console.log('random :',rand); } console.log('Min = ' , min , 'Max = ' , max);</pre>	<pre>PS G:\c> node app.js start loop random : 90 random : 27 random : 72 random : 37 random : 93 random : 86 random : 65 random : 29 random : 28 random : 24 Min = 24 Max = 93</pre>

ตัวอย่างนี้เป็นการใช้คำสั่ง for loop วน loop 10 รอบ โดยประกาศตัวแปร i มีค่าเริ่มต้นเป็น 1 และเพิ่มค่า i ขึ้น รอบละ 1 ในแต่ละรอบจะสุ่มตัวเลขขึ้นมา 1 ตัว มีค่าอยู่ระหว่าง 1 - 100 แสดงค่าที่สุ่มออกมาทางหน้าจอ พร้อมทั้งหาค่าต่ำสุด และค่าสูงสุดของตัวเลขที่สุ่มได้ แสดงออกมาทางหน้าจอ หลังจากวน loop ครบแล้ว

การใช้คำสั่ง for loop ในส่วนของการกำหนดตัวแปร และส่วนของการเปลี่ยนแปลงค่า สามารถประกาศตัวแปรได้มากกว่า 1 ตัวแปร และเปลี่ยนแปลงค่าที่ละหลายค่าได้ โดยใช้เครื่องหมาย comma (,) คั่นแต่ละตัวแปร ดังตัวอย่างต่อไปนี้

Nodejs Code	Output
<pre>console.log('start loop') for (i = 1, j = 10; i < j; i++, j--) { console.log('i =', i, 'j =', j); } console.log('End');</pre>	<pre>PS G:\c> node app.js start loop i = 1 j = 10 i = 2 j = 9 i = 3 j = 8 i = 4 j = 7 i = 5 j = 6 End</pre>

การใช้คำสั่ง for ซ้อน for

เราสามารถเขียนคำสั่ง for ซ้อนอยู่ในคำสั่ง for อีกตัวได้ เรียกว่า for ซ้อน for เช่นเดียวกันคำสั่ง while loop, do while loop ก็สามารถทำได้เช่นกัน ดังตัวอย่างต่อไปนี้

Nodejs Code	Output
<pre>for(a=1;a≤6;a++){ let line = ''; for(b=1;b≤a;b++){ line += '0'; } console.log(line); }</pre>	<pre>PS G:\c> node app.js 0 00 000 0000 00000 000000</pre>

การใช้คำสั่ง break และ continue

คำสั่ง break และ continue จะใช้กับ for loop, while loop หรือ do while loop โดยที่

- break จะใช้สำหรับออกจาก loop หยุดการวน loop
- continue จะใช้สำหรับให้โปรแกรมกลับไปเริ่มต้น loop ใหม่ โดยไม่ต้องทำคำสั่งที่เหลืออยู่

ตัวอย่างการใช้คำสั่ง break

Nodejs Code	Output
<pre>for (i = 1; i ≤ 10; i++) { console.log(i); if (i == 5) { break; } } console.log('End')</pre>	<pre>PS G:\c> node app.js 1 2 3 4 5 End</pre>

ตัวอย่างนี้ กำหนดให้วน loop 10 รอบ ใน loop แต่ละรอบ มีการเช็คตัวแปร i ถ้า i เท่ากับ 5 ให้ break ออกจาก loop ทำให้โปรแกรมนี้ วน loop แค่ 5 รอบ

ฟังก์ชันคืออะไร

ฟังก์ชัน (Function) คือกลุ่มของชุดคำสั่งที่ถูกรวมเข้าด้วยกัน สำหรับการทำงานบางอย่าง ฟังก์ชันสามารถรับพารามิเตอร์เพื่อนำข้อมูลเข้ามาใช้งานและส่งค่ากลับได้ โดยปกติการทำงานบางอย่างที่ต้องใช้งานบ่อย หรือใช้หลายๆ ที่ เราจะแยกคำสั่งเหล่านั้นออกมาเป็น ฟังก์ชันไว้ เพื่อความสะดวกในการเขียนโปรแกรม ทำให้โค้ดสั้นลง ดูง่ายขึ้น

การประกาศฟังก์ชัน

ก่อนที่จะใช้งานฟังก์ชัน มันจะต้องถูกประกาศหรือสร้างขึ้นมาก่อน การประกาศฟังก์ชันจะใช้คำสั่ง `function` ตามด้วยชื่อของฟังก์ชัน `name` การตั้งชื่อของฟังก์ชันนั้นจะเหมือนกับตัวแปร

รูปแบบของการประกาศฟังก์ชัน

```
Function name(parameter1, parameter2, ...) {  
    // code to be executed  
    return value;    //Optional  
}
```

ตัวอย่างการใช้งานฟังก์ชัน

Nodejs Code	Output
<pre>function show(str){ console.log(str) } show('hello world')</pre>	<pre>hello world PS G:\c> node app.js hello world</pre>

ตัวอย่างนี้เป็นการประกาศฟังก์ชัน show() ขึ้นมา โดยมีการส่ง parameter เข้าไป 1 ตัว เพื่อให้ฟังก์ชันนี้แสดงข้อความที่กำหนด ออกมาทางหน้าจอ

การส่งค่ากลับจากฟังก์ชัน

หากต้องการส่งค่ากลับมาจากฟังก์ชัน จะใช้คำสั่ง return ในการส่ง ดังตัวอย่างต่อไปนี้

Nodejs Code	Output
<pre>function random(min, max) { let rand = Math.floor(Math.random() * (max - min)) + min; return rand; } console.log('Number :', random(1, 10))</pre>	<pre>PS G:\c> node app.js Number : 4 PS G:\c></pre>

ตัวอย่างนี้ ฟังก์ชัน random() ทำหน้าที่สุ่มตัวเลขที่มีค่าอยู่ในช่วง 1 – 10 แล้วส่งค่ากลับออกมา

การใช้งาน Array

อาร์เรย์ (Array) คือ ชุดของข้อมูลที่ถูกเรียงต่อกันเป็นลำดับ เหมือนข้อมูลที่อยู่ในตาราง เราสามารถใช้ Array เก็บข้อมูลขึ้นได้ทุกประเภท

การประกาศตัวแปร Array

```
let color = []; //ประกาศตัวแปร array เปล่า
```

```
let cars = ["BMW","VOLVO","TESLA","BYD","MG"]; //ประกาศตัวแปร Array พร้อมกำหนดข้อมูลใน Array 5 ตัว
```

การใช้ข้อมูลใน Array

ข้อมูลใน Array จะเก็บเรียงตาม Index และ Index ของ Array เริ่มต้นจาก 0 การใช้ข้อมูลใน Array จะอ้างอิงตาม Index ดังตัวอย่างต่อไปนี้

Nodejs Code	Output
<pre>let cars = ["BMW", "VOLVO", "TESLA", "BYD", "MG"]; console.log('data', cars) console.log('car[0]', cars[0]) console.log('car[1]', cars[1]) console.log('car[2]', cars[2]) console.log('car[3]', cars[3]) console.log('car[4]', cars[4])</pre>	<pre>PS G:\c> node app.js data ['BMW', 'VOLVO', 'TESLA', 'BYD', 'MG'] car[0] BMW car[1] VOLVO car[2] TESLA car[3] BYD car[4] MG</pre>

การหาขนาดของ Array

หากต้องการทราบว่า Array มีข้อมูลอยู่กี่ตัว สามารถใช้ Property length หาได้ ดังตัวอย่างนี้

Nodejs Code	Output
<pre>let cars = ["BMW", "VOLVO", "TESLA", "BYD", "MG"]; console.log('data', cars) console.log('จำนวน data', cars.length , ' ตัว')</pre>	<pre>PS G:\c> node app.js data ['BMW', 'VOLVO', 'TESLA', 'BYD', 'MG'] จำนวน data 5 ตัว</pre>

การเพิ่มข้อมูลใน Array

การเพิ่มข้อมูลใน Array มี 2 เมธอด ที่ใช้ได้ ดังนี้

- push ใช้สำหรับเพิ่มข้อมูลไปยังตำแหน่งท้ายสุดของ Array
- unshift ใช้สำหรับเพิ่มข้อมูลไปยังตำแหน่งแรกของ Array

ตัวอย่างการเพิ่มข้อมูลใน Array

Nodejs Code	Output
<pre>let cars = ["BMW", "VOLVO", "TESLA", "BYD", "MG"]; console.log('data', cars) cars.push("BENZ") console.log('data', cars) cars.push("META") console.log('data', cars)</pre>	<pre>data ['BMW', 'VOLVO', 'TESLA', 'BYD', 'MG'] data ['BMW', 'VOLVO', 'TESLA', 'BYD', 'MG', 'BENZ'] data ['BMW', 'VOLVO', 'TESLA', 'BYD', 'MG', 'BENZ', 'META']</pre>

การลบข้อมูลออกจาก Array

หากต้องการลบข้อมูลออกจาก Array จะมี 2 เมธอด ที่ทำได้ ดังนี้

- pop ใช้สำหรับลบข้อมูลใน Array ตำแหน่งสุดท้ายออก และส่งค่าที่ลบออกกลับมา
- shift ใช้สำหรับลบข้อมูลใน Array ตำแหน่งแรกออก และส่งค่าที่ลบออกกลับมา

ตัวอย่างการลบข้อมูลใน Array

Nodejs Code	Output
<pre>let cars = ["BMW", "VOLVO", "TESLA", "BYD", "MG"]; console.log('data', cars) let item = cars.pop(); console.log('method Pop', item) console.log('data', cars) item = cars.shift() console.log('method Pop', item) console.log('data', cars)</pre>	<pre>PS C:\> node app.js data ['BMW', 'VOLVO', 'TESLA', 'BYD', 'MG'] method Pop MG data ['BMW', 'VOLVO', 'TESLA', 'BYD'] method Pop BMW data ['VOLVO', 'TESLA', 'BYD']</pre>

การใช้ for loop กับ Array

หากข้อมูลใน Array มีจำนวนมาก และต้องการใช้งานทุกตัว การเขียนคำสั่งเข้าถึงทีละตัวจะไม่สะดวก ใช้คำสั่ง for loop แทนจะดีกว่า ดังตัวอย่างต่อไปนี้

Nodejs Code	Output
<pre>let number = [45, 32, 16, 25, 50]; for (i = 0; i < number.length; i++) { console.log(number[i]) }</pre>	<pre>45 32 16 25 50</pre>

การใช้ for in, for of กับ Array

คำสั่ง for in, for of เป็นคำสั่งที่วน Loop ตามจำนวนข้อมูลใน Array ที่สามารถใช้ได้เหมือน for loop แต่การใช้งานจะง่ายกว่า ดังตัวอย่างต่อไปนี้

Nodejs Code	Output
<pre>let number = [45, 32, 16, 25, 50]; for (let x in number) { console.log(number[x]) } console.log('====') for (let x of number) { console.log(x) }</pre>	<pre>45 32 16 25 50 ==== 45 32 16 25 50</pre>

คำสั่ง for of จะวนรอบสมาชิกทั้งหมดใน Array และนำค่าของการวนแต่ละรอบกำหนดไว้ในตัวแปร x

Require Function

ช่วยให้เราสามารถแบ่งโค้ดออกเป็นส่วและห่อหุ้มการทำงานในขอบเขต โค้ดที่มีลักษณะการทำงานเหมือนกันอาจอยู่ในโมดูลเดียวกัน นี่สามารถช่วยลดความซับซ้อนและทำให้โปรแกรมง่ายต่อการดูแลและการพัฒนา และสามารถนำโค้ดมาใช้ซ้ำได้ง่าย

ตัวอย่างการเรียกใช้ exports

Nodejs Code	Output
<pre>var str = {}; str.sumnumber = function (a, b) { return a + b; }; str.teststring = function (a, b) { return a + b; }; exports.data = str;</pre>	<pre>const Alldata = require('./fnc/function.js') console.log('All function', Alldata); const { data } = require('./fnc/function.js') const result = data.sumnumber(5, 10); console.log('The sum is:', result); const result2 = data.teststring('ABC', 'DEFG'); console.log('The string', result2);</pre>

Nodejs Code	Output
<pre>PS G:\c> node app.js All function { data: { sumnumber: [Function (anonymous)], teststring: [Function (anonymous)] } } The sum is: 15 The string ABCDEFG</pre>	

ตัวอย่างการเรียกใช้ module.exports

Nodejs Code	Output
<pre>var str = {}; str.sumnumber = function (a, b) { return a + b; }; str.teststring = function (a, b) { return a + b; }; module.exports = str;</pre>	<pre>const Alldata = require('./fnc/function.js') console.log('All function', Alldata); const result = Alldata.sumnumber(5, 10); console.log('The sum is:', result); const result2 = Alldata.teststring('ABC', 'DEFG'); console.log('The string', result2);</pre>
Nodejs Code	Output
<pre>All function { sumnumber: [Function (anonymous)], teststring: [Function (anonymous)] } The sum is: 15 The string ABCDEFG</pre>	

Synchronous

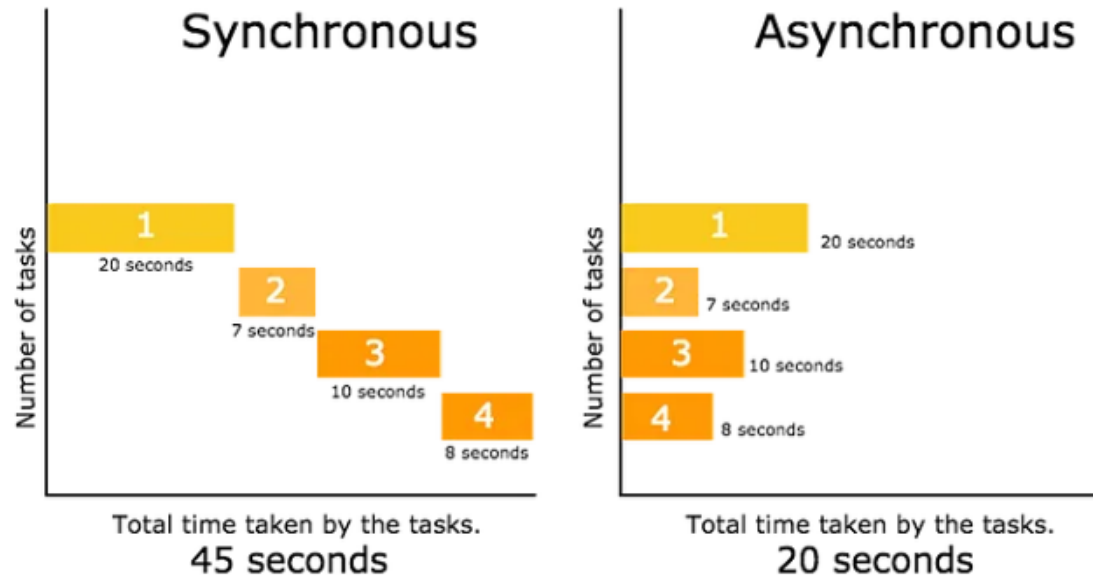


Asynchronous



Synchronous หรือ Blocking จะดำเนินการรันโปรแกรมทีละชุดคำสั่ง และจะไม่รันชุดคำสั่งต่อไปถ้ายังรันชุดคำสั่งปัจจุบันไม่จบ ตัวอย่างเช่น ถ้าโปรแกรมเรียกฟังก์ชัน A(); และ B(); ตามลำดับ โปรแกรมจะไม่รันฟังก์ชัน B(); จนกว่าฟังก์ชัน A(); จะทำงานเสร็จ

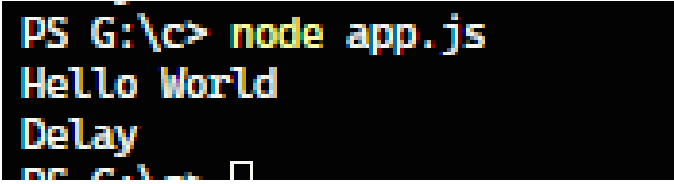
Asynchronous หรือ Non-blocking จะดำเนินการรันโปรแกรมทีละชุดคำสั่ง และจะรันชุดคำสั่งถัดไปทันทีโดยไม่จำเป็นต้องรอชุดคำสั่งก่อนหน้าทำงานเสร็จ ตัวอย่างเช่น ถ้าโปรแกรมเรียกฟังก์ชัน A(); และ B(); ตามลำดับ โปรแกรมจะรันฟังก์ชัน A(); และ B(); ตามลำดับโดยไม่สนใจว่าฟังก์ชัน A(); จะทำงานเสร็จรึยัง จะไปเรียกฟังก์ชัน B(); ต่อเลยทันที



Synchronous จากภาพจะเห็นว่าโปรแกรมจะรันชุดคำสั่งที่ 2 ได้นั้นจะต้องรันชุดคำสั่งที่ 1 เสร็จก่อน และถ้าจะรันชุดคำสั่งที่ 4 ได้นั้นจะต้องรันชุดคำสั่งที่ 1 2 3 เสร็จก่อนตามลำดับ ซึ่งหมายความว่าถ้าจะรันชุดคำสั่งที่ 4 ได้นั้นจะต้องรอตั้ง 37 วินาที (20 + 7 + 10) ทำให้ระยะเวลารวมในการรันโปรแกรมเท่ากับ 45 วินาที

Asynchronous จากภาพจะเห็นว่าโปรแกรมจะรันชุดคำสั่งที่ 1 2 3 4 ได้เลยโดยที่ไม่ต้องรอตชุดคำสั่งก่อนหน้าทำงานเสร็จก่อน ซึ่งหมายความว่าชุดคำสั่งที่ 4 จะรันได้เลยไม่ต้องรอ 37 วินาที แบบ Synchronous ในเมื่อไม่ต้องรอตชุดคำสั่งอื่นๆ ทำงานก็เปรียบเสมือนชุดคำสั่ง 1 2 3 4 ถูกเรียกทำงานพร้อมกัน ทำให้ระยะเวลารวมในการรันโปรแกรมใช้เพียง 20 วินาที (เท่ากับเวลาที่รันชุดคำสั่งที่ 1 หรือชุดคำสั่งที่รันนานที่สุดนั่นเอง)

ตัวอย่างการเรียกใช้ async / await

Nodejs Code	Output
<pre>function helloWorld(){ return "Hello World"; } function delayhelloWorld() { return new Promise((resolve, reject) => { setTimeout(()=>{ resolve("Delay"); }, 1500); }); } async function main(){ let a = helloWorld(); console.log(a); let b = await delayhelloWorld(); console.log(b); } main();</pre>	 <pre>PS G:\c> node app.js Hello World Delay PS G:\c></pre>

จากรันข้อมูลแล้ว ตัวฟังก์ชัน helloworld จะทำงานก่อน แล้วรอชุดคำสั่งต่อไป DelayhelloWorld ตั้งดีเลย์ไว้ 1.5 วิ แล้วจะโชว์

Create Server

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World!');
}).listen(3000);
```

จากตัวอย่างจะเป็นการสร้าง Server สามารถเข้าได้ที่ localhost:3000 ตามพอดที่เรากำหนด

Express framework บน Node.js

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

จากตัวอย่างจะใช้ Express

Connect Mysql

ให้ติดตั้ง lib mysql ก่อน npm install mysql

```
const express = require('express');
const mysql = require('mysql');
const app = express();
const port = 3000;

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'xxxxx',
  database: 'xxxxx'
});

app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

require mysql เข้ามา และสร้าง connect ใส่ host,user,password,database ที่สร้างไว้

```
connection.connect((err) => {
  if (err) {
    console.error('Error connecting to database');
    return;
  }
  console.log('Connected to the database');
});
```

ใช้ฟังก์ชัน Connect เพื่อ เชื่อมต่อ db

```
app.get('/', (req, res) => {  
  connection.query('SELECT * FROM account', (error, results, fields) => {  
    if (error) throw error;  
    console.log(results)  
  });  
});
```

12. Run the app

Server is running at http://localhost:3000

Connected to the database

[RowDataPacket { id: 1, userid: 'xxxx', password: 'bbbb' }]

หลังจากรันแล้วเข้า localhost:3000 จะขึ้นโชว์ตามภาพ