การติดตั้ง Node.js และภาษา JavaScript

ในบทนี้ คุณจะได้เรียนรู้เกี่ยวกับการติดตั้ง Node.js ซึ่งเป็นสภาพแวดล้อมที่เราจะใช้รันโปรแกรมภาษา JavaScript ในบทเรียนของเรา และการติดตั้งโปรแกรม Visual Studio Code ซึ่งเป็น Editor ที่เราอยากแนะนำสำหรับการเขียนโปรแกรมภาษา JavaScript นี่เป็นเนื้อหาของบทนี้

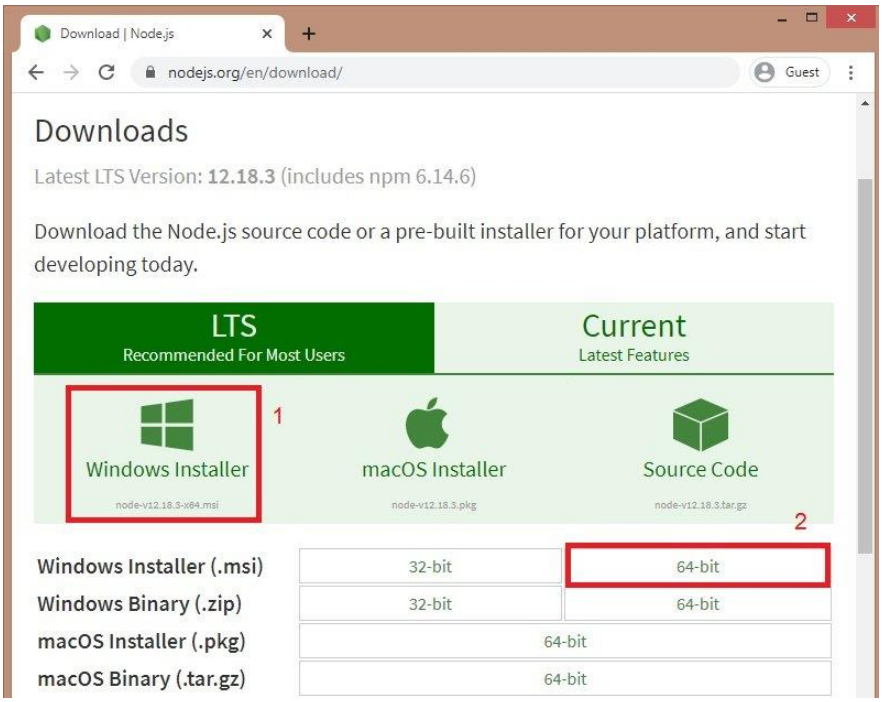
- การติดตั้ง Node.js
- การติดตั้งโปรแกรม Visual Studio Code
- ทดสอบเขียนและรันโปรแกรม

การติดตั้ง Node.js

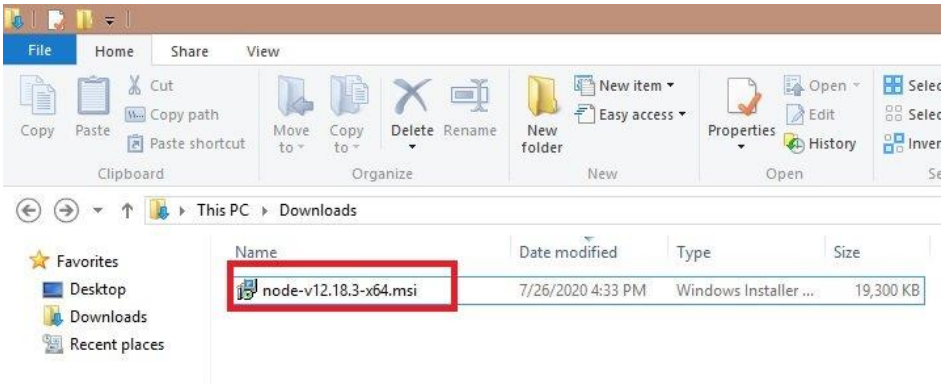
Node.js เป็นสภาพแวดล้อมในการรันโปรแกรมภาษา JavaScript นอกเว็บเบราว์เซอร์ หรือกล่าวอีกนัยหนึ่ง เราสามารถเขียนโปรแกรมภาษา JavaScript และรันบน Command line ได้; Node.js นั้นเป็นโปรเจ็คแบบโอเพ่นซอร์ส และข้ามแพลตฟอร์ม ที่มีให้ดาวน์โหลดทั้งบนระบบปฏิบัติการ Windows, Unix และ Mac

ในบทเรียนนี้ จะเป็นการติดตั้ง Node.js บน Windows 10 อย่างไรก็ตาม หากคุณใช้วินโด้เวอร์ชันอื่น เช่น Windows 11 การติดตั้งจะไม่แตกต่างกันมา ต่อไปเป็นขั้นตอนการตั้ง Node.js

1) ก่อนอื่นเราต้องดาวน์โหลดตัวติดตั้งของ Node.js มาก่อน คุณสามารถดาวน์โหลดได้ที่เว็บไซต์หลักของมันที่ https://nodejs.org/en/download/ ในหน้าดาวน์โหลด เลือกตัวเลือกการดาวน์โหลดที่แนะนำ (1) หรือเลือกตามระบบปฏิบัติการของคุณ (2)

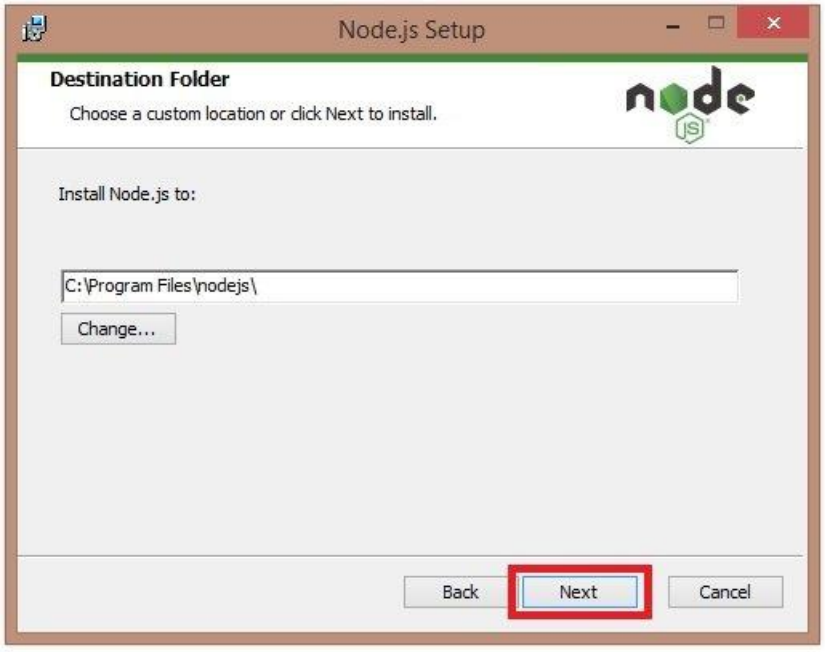


2) รอจนกว่าการดาวน์โหลดเสร็จสิ้น เปิดโฟล์เดอร์ที่เก็บไฟล์ติดตั้ง Node.js ที่ดาวน์โหลดมา หลังจากนั้นคลิกที่ไฟล์ดังกล่าวเพื่อเริ่มการติดตั้ง

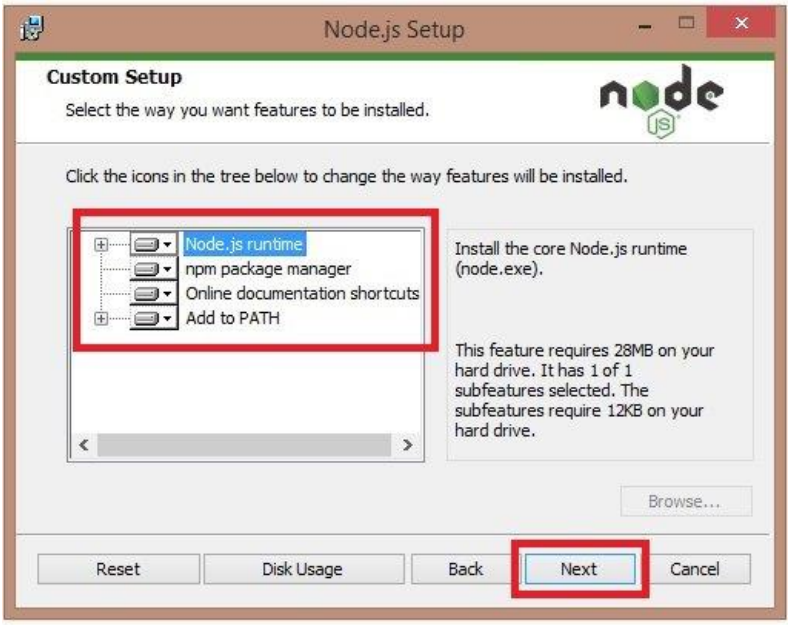


3) ในหน้าเริ่มต้นการติดตั้ง คลิก "I accept the terms and the Licence Agreement" เพื่อยอมรับเงื่อนไขการใช้งาน จากนั้นคลิก "Next" เพื่อไปยังหน้าต่อไป

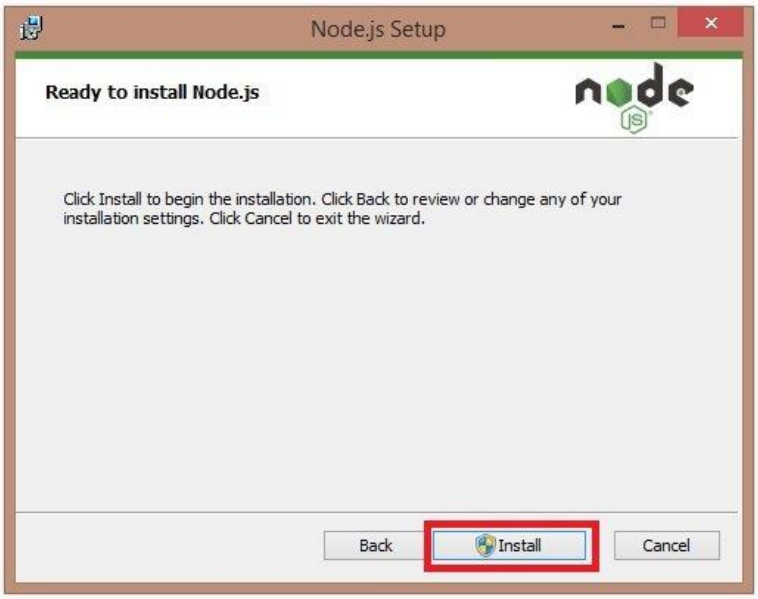
4) ในหน้ากำหนดสถานที่ติดตั้ง คุณสามารถเลือกโฟล์เดอร์สำหรับติดตั้ง Node.js ที่ต้องการได้ ในตัวอย่าง เราได้ใช้ค่าเริ่มต้น จากนั้นคลิก "Next" เพื่อไปยังหน้าต่อไป



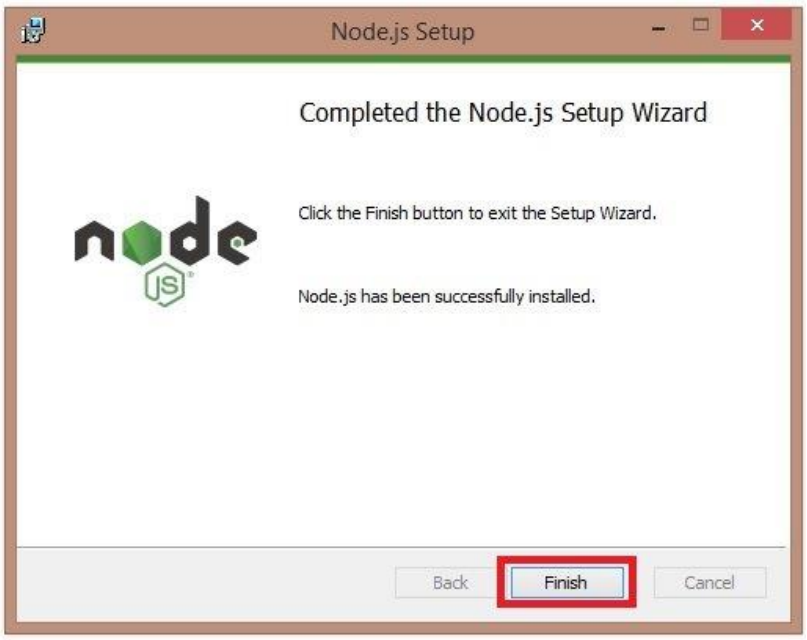
5) ในหน้าตั้งค่าการติดตั้ง เป็นการเลือกส่วนประกอบที่ต้องการติดตั้ง ในตัวอย่างเราใช้ค่าเริ่มต้นซึ่งเป็นการติดตั้งทุกอย่างที่มากับตัวติดตั้งนี้ ถ้าหากบางตัวเลือกไม่ถูกเลือก เราแนะนำให้เลือกมันทั้งหมด จากนั้นคลิก "Next" เพื่อไปยังหน้าต่อไป



6) หลังจากเราเตรียมการตั้งค่าเรียบร้อยแล้ว ตอนนี้คลิก "Install" เพื่อเริ่มต้นติดตั้ง Node.js ลงบนคอมพิวเตอร์ของคุณ



7) รอก่อนว่าการติดตั้งจะเสร็จ หลังจากนั้นคุณจะพบกับหน้าต่างนี้ คลิก "Finished" ถือว่าการติดตั้ง Node.js เสร็จเรียบร้อย



เพื่อตรวจสอบให้แน่ใจว่า Node.js ได้ติดตั้งลงบนคอมพิวเตอร์ของคุณและมันสามารถใช้งานได้ ให้เปิด Command line ขึ้นมาแล้วพิมพ์คำสั่งต่อไปนี้

```
node -v
```

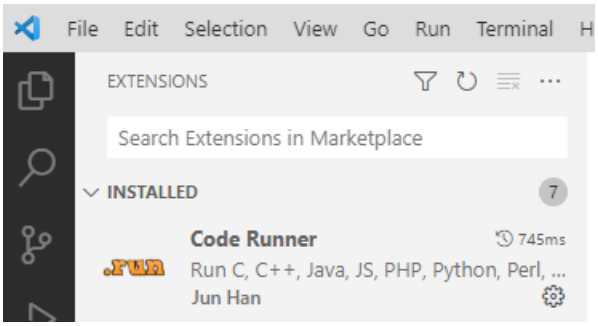
ผลลัพธ์ที่ได้ควรจะเป็นเวอร์ชันของ Node.js ที่เราเพิ่งจะติดตั้งไป

```
v18.17.0
```

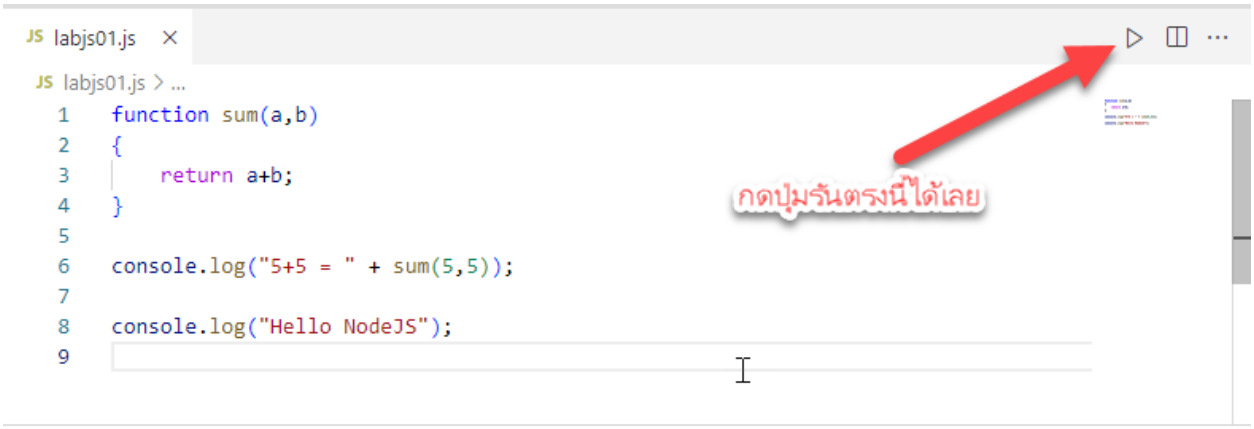
นั่นหมายความว่าในตอนนี้เราสามารถใช้งาน Node.js เพื่อรันโปรแกรมภาษา JavaScript ได้แล้ว

หมายเหตุ: เมื่อเราติดตั้ง Node.js มันจะมาพร้อมกับ JavaScript engine ที่ใช้ในการคอมไพล์และรันภาษา JavaScript; หรือกล่าวอีกนัยหนึ่ง ภาษา JavaScript จะถูกติดตั้งมาพร้อมกับ Node.js นั่นเอง

VS code ลง Code runner เพิ่มด้วยนะครับ



ทีนี้เวลาเราเขียนโปรแกรมใน VS code แล้ว ก็จะสามารถรันโปรแกรมได้ง่ายๆ เลย



JavaScript

การรับค่าและการแสดงผล ในภาษา JavaScript

ในบทนี้ คุณจะได้เรียนรู้เกี่ยวกับการรับค่าและการแสดงผลในภาษา JavaScript; ภาษา JavaScript นั้นเป็นภาษาที่รันอยู่บน Host environment และไม่มีฟังก์ชันสำหรับการรับค่าเป็นของมันเอง ในบทนี้เราจะพูดถึงการรับค่าและแสดงผลบน Command line ที่รันบน Node.js เท่านั้น เราจะไม่พูดถึงการรับค่าและแสดงผลด้วยฟังก์ชันบนเบราว์เซอร์อย่างเช่น prompt และ alert และนี่เป็นเนื้อหาในบทนี้

- การแสดงผลออกทางหน้าจอ
- การแสดงผลด้วยการแทนที่ใน String
- การแสดงผลด้วย process.stdout.write
- การรับค่าผ่านทางคีย์บอร์ด (Node.js)
- การรับค่าจาก Command line อาร์กิวเมนต์
- Escape characters

การแสดงผลออกทางหน้าจอ

ฟังก์ชัน console.log เป็นฟังก์ชันสำหรับแสดงข้อมูลทุกประเภทออกทางหน้าจอในภาษา JavaScript เมื่อเรารันของโค้ดภาษา JavaScript บน Node.js การแสดงผลจะผ่าน Command Line (Console) หรือ Terminal ในขณะที่บนเว็บเบราว์เซอร์การแสดงผลจะผ่านเว็บ Console แทน

นี่เป็นตัวอย่างการใช้งานฟังก์ชัน console.log เพื่อแสดงข้อมูลในภาษา JavaScript

output1.js

```
console.log("JavaScript");

console.log(12);

console.log(true);
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
JavaScript

12

true
```

ในตัวอย่าง เราได้ใช้ฟังก์ชัน console.log เพื่อแสดง Literal ของข้อความ ตัวเลข และ Boolean ออกทางหน้าจอตามลำดับ

สังเกตว่าค่าที่ถูกแสดงออกมาในแต่ละคำสั่งนั้นจะอยู่คนละบรรทัดกัน นั่นเป็นเพราะว่า ฟังก์ชัน console.log จะแสดงการขึ้นบรรทัดใหม่ "\n" หลังจากที่แสดงค่าของพารามิเตอร์ด้วยเสมอ

นอกจากนี้ เรายังสามารถส่งหลายพารามิเตอร์เข้าไปยังฟังก์ชันได้ ยกตัวอย่างเช่น

```
console.log("JavaScript", "TypeScript");

console.log(1, 2, 3, 4, 5);
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
JavaScript TypeScript

1 2 3 4 5
```

ในตัวอย่าง แสดงให้เห็นว่าเราสามารถส่งพารามิเตอร์มากกว่าหนึ่งเข้าไปยังฟังก์ชันได้ โดยแต่ละค่าที่แสดงออกมานั้นจะคั่นด้วยช่องว่าง

ฟังก์ชัน console.log ไม่เพียงแค่สามารถแสดงข้อมูลพื้นฐานได้เท่านั้น แต่มันยังสามารถใช้แสดงออบเจ็ค อาเรย์ ฟังก์ชัน หรือข้อมูลทุกประเภทในภาษา JavaScript

output2.js

```
let user = {
    name: "Metin",
    age: 28
};
function hello() {
    console.log("Hello World!");
}

console.log(user);
console.log([10, 20, 30, 40, 50]);
console.log(hello);
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
{ name: 'Metin', age: 28 }
[ 10, 20, 30, 40, 50 ]
[Function: hello]
```

ในตัวอย่าง แสดงให้เห็นว่าฟังก์ชัน console.log นั้นจะแสดงค่าต่างๆ ออกมาในรูปแบบ Literal ของข้อมูล ยกเว้นฟังก์ชันที่แสดงในรูปแบบของ String ที่เป็นชื่อของมัน นี่จะทำให้เราสามารถใช้มันเพื่อ Debug ค่าจากออบเจ็คประเภทต่างๆ ได้เป็นอย่างดี

การแสดงผลด้วยการแทนที่ใน String

นอกจากแสดงผลข้อมูล ตัวแปร หรือออบเจ็คแบบปกติแล้ว ฟังก์ชัน console.log ยังสนับสนุนการแสดงผลด้วยการแทนที่ใน String อีกด้วย โดยการกำหนดพารามิเตอร์แรกเป็น Format string สำหรับจัดรูปแบบการแสดงผล ซึ่ง Format string นั้นจะประกอบไปด้วยตัวกำหนดการแสดงผล (Specifier) ที่แสดงดังในตารางต่อไปนี้

ตัวกำหนดการแสดงผล	คำอธิบาย
%o หรือ %O	แสดงผลออบเจ็คในภาษา JavaScript
%d	แสดงผลตัวเลข
%i	แสดงผลตัวเลขจำนวนเต็ม
%s	แสดงผลข้อความ
%f	แสดงผลตัวเลขทศนิยม

ในการใช้งานฟังก์ชัน console.log สำหรับจัดรูปแบบการแสดงผลด้วยการแทนที่ใน String นั้น ถ้าหากพารามิเตอร์แรกเป็น String ที่มี Specifier ที่ปรากฏในตารางด้านบน ฟังก์ชันจะทำงานในรูปแบบของการแทนที่ใน String แทน

นี่เป็นตัวอย่างการแสดงผลด้วยการแทนที่ใน String ในภาษา JavaScript

string_substitution.js

```
let name = "Metin";

let height = 6.2;

console.log("%s is %f inches height", name, height);

console.log("%d + %d = %d", 3, 5, 3 + 5);
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
Metin is 6.2 inches height

3 + 5 = 8
```

ในตัวอย่าง การแสดงผลในบรรทัดแรก เนื่องจากพารามิเตอร์แรกนั้นประกอบไปด้วยตัวจัดรูปแบบการแสดงผล %s และ %f นั่นหมายความว่าการทำงานของฟังก์ชันจะเป็นแบบการแทนค่าใน String พารามิเตอร์ตัวต่อไปจะถูกนำไปแทนที่ในตัวกำหนดรูปแบบตามลำดับที่มันปรากฏใน Format string

```
console.log("%s is %f inches height", name, height);
```

ดังนั้นค่าของพารามิเตอร์ name จะถูกนำไปแทนที่ %s และถัดมาพารามิเตอร์ height จะถูกนำไปแทนที่ %f ตามลำดับ และแสดงออกมาทางหน้าจอ

```
console.log("%d + %d = %d", 3, 5, 3 + 5);
```

ในบรรทัดต่อมา ได้มีการกำหนดตัวกำหนดรูปแบบการแสดง %d สามตัวใน Format string ดังนั้นฟังก์ชันต้องการพารามิเตอร์อีกสามตัวสำหรับการทำงาน เราได้ส่งค่าตัวเลขเข้าไปยังฟังก์ชันโดยตรง

ตัวอย่างต่อมาเป็นการใช้ตัวกำหนดการแสดงผล %o เพื่อแสดงค่าของออบเจ็คพร้อมกับ String

object_output.js

```
let user = {
    name: "Metin",
    safary: 50000,
    single: true
};

console.log("Inspect object (%o)", user);

console.log("Inspect object (", user, ")");
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
Inspect object ({ name: 'Metin', safary: 50000, single: true })

Inspect object ( { name: 'Metin', safary: 50000, single: true } )
```

ในตัวอย่าง เป็นการแสดงค่าของออบเจ็ค เนื่องจากเราต้องการแสดงค่าของออบเจ็คภายในวงเล็บ (...) ดังนั้นการแทนที่ String นั้นเป็นวิธีที่ดีที่เราสามารถทำได้

```
Inspect object ( { name: 'Metin', safary: 50000, single: true } )
```

และถึงแม้ว่าเราจะสามารถใช้การส่งค่าแบบหลายพารามิเตอร์ในการแสดงผลบรรทัดที่สอง แต่ฟังก์ชัน console.log จะแสดงเว้นวรรคระหว่างแต่ละพารามิเตอร์ด้วยเสมอ ซึ่งแน่นอนว่านี่ไม่ใช่สิ่งที่เราต้องการ

ตัวอย่างต่อมาเป็นการใช้การแทนที่ String ในการแปลงข้อมูลก่อนการแสดงผล เราสามารถใช้ตัวกำหนดรูปแบบการแสดงผลใดๆ สำหรับข้อมูลประเภทใดๆ ก็ได้ โปรแกรมจะพยายามแปลงค่าจากพารามิเตอร์ให้เป็นประเภทข้อมูลที่ตรงกับตัวกำหนดรูปแบบการแสดงผลที่ระบุไว้ใน Format string นี่เป็นตัวอย่าง

type_conversion.js

```
console.log("%i", 3.14);

console.log("%s", { name: "Metin", age: 28 });

console.log("Date: %s", new Date());

console.log("Timestamp: %d", new Date());
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
3

{ name: 'Metin', age: 28 }

Date: 2020-07-29T13:37:20.942Z

Timestamp: 1596029840942
```

ในตัวอย่าง นี่เป็นการบังคับการแสดงผลโดยกำหนดตัวจัดรูปแบบการแสดงผลเป็นประเภทข้อมูลที่ไม่เหมือนกับพารามิเตอร์ของมัน

```
console.log("%i", 3.14);

console.log("%s", { name: "Metin", age: 28 });
```

เนื่องจากตัวกำหนดการแสดงผล %i ใช้สำหรับแสดงตัวเลขจำนวนเต็ม ดังนั้น 3.14 จะถูกแปลงเป็นจำนวนเต็มก่อนการแสดงผล และบรรทัดต่อมาเป็นการแสดงออบเจ็คในรูปแบบของ String ด้วย %s

```
console.log("Date: %s", new Date());

console.log("Timestamp: %d", new Date());
```

ถัดมาเป็นการแสดงออบเจ็คของวันที่ในรูปแบบของ String ค่าของออบเจ็คถูกแปลงเป็น String จากการเรียกใช้งานเมธอด toString บนออบเจ็ค และสำหรับบรรทัดต่อมา ออบเจ็คของวันที่จะถูกแปลงเป็นตัวเลข ทำให้ได้รับค่าของ Timestamp แทน

คุณอาจจะไม่เห็นความแตกต่างอย่างชัดเจนระหว่างการใช้ %s และ %o สำหรับแสดงออบเจ็ค นั่นเป็นเพราะว่าค่าในออบเจ็คนั้นเป็นประเภทข้อมูลพื้นฐาน ซึ่งมันจะถูกแสดงตาม Literal ของข้อมูลนั้นๆ ยกตัวอย่างเช่น

```
let user = {
    name: "Metin",
    age: 28
};


console.log("%s", user);  // { name: 'Metin', age: 28 }
console.log("%o", user);  // { name: 'Metin', age: 28 }
```

จากตัวอย่างด้านบน ทั้งสองคำสั่งนั้นจะให้ผลลัพธ์ที่เหมือนกัน แต่นี่จะทำให้ชัดเจนเมื่อออบเจ็คนั้นประกอบไปด้วยข้อมูลประเภทอื่นๆ ยกตัวอย่างเช่น ฟังก์ชัน

object_output2.js

```
let user = {
    name: "Metin",
    age: 28,
    sayHi: function() {
        console.log("Hi");
    },
    birthDate: {
        month: 3,
        year: 1988
    }
};
```

```
console.log("%s", user);

console.log("%o", user);
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
{
  name: 'Metin',
  age: 28,
  sayHi: [Function: sayHi],
  birthDate: [Object]
}
{
  name: 'Metin',
  age: 28,
  sayHi: [Function: sayHi] {
    [length]: 0,
    [name]: 'sayHi',
    [arguments]: null,
    [caller]: null,
    [prototype]: sayHi { [constructor]: [Circular] }
  },
  birthDate: { month: 3, year: 1988 }
```

```
}
```

และอย่างที่คุณเห็น %s จะแปลงออบเจ็คเป็น String โดยออบเจ็คย่อยภายในนั้นจะถูกแปลงในรูปแบบที่เรียบง่ายที่สุดเท่าที่จะเป็นไปได้ ในขณะที่ %o จะแสดงโครงสร้างทั้งหมดของออบเจ็ค รวมทั้ง Property ย่อยภายในออบเจ็ค

ฟังก์ชัน console.log นั้นเป็นฟังก์ชันมาตฐานสำหรับแสดงข้อมูลในภาษา JavaScript ที่สามารถใช้ได้ในทุก Host environment อย่างไรก็ตาม เนื้อหาหลังจากนี้จะเป็นการพูดถึงฟังก์ชันสำหรับรับค่าและแสดงผลที่เฉพาะบน Node.js เท่านั้น นั่นหมายความว่ามันอาจจะไม่ทำงานบน Host environment อื่นๆ เช่น เว็บเบราว์เซอร์

ประเภทข้อมูล

ประเภทข้อมูล (Data type) คือคุณลักษณะของข้อมูลที่ใช้บอกคอมไพเลอร์ว่าโปรแกรมต้องการใช้งานข้อมูลอย่างไร ประเภทข้อมูลใช้สำหรับกำหนดวิธีที่คอมพิวเตอร์จัดเก็บและจัดการกับข้อมูลในหน่วยความจำ เพื่อให้โปรแกรมทำงานได้อย่างมีประสิทธิภาพ

ในการเขียนโปรแกรม เรามักจะทำงานกับข้อมูลหลายประเภท ยกตัวอย่างเช่น ในการพัฒนาเว็บไซต์ร้านค้าออนไลน์จะมีข้อมูลเกี่ยวกับสินค้าที่ต้องเก็บ เราอาจจะเก็บราคาของสินค้าเป็นตัวเลข เพราะนั่นจะทำให้สามารถนำไปคำนวณได้ และเก็บชื่อและรายละเอียดของสินค้าเป็น String เนื่องจากว่ามันเป็นสิ่งที่ต้องอธิบายเป็นคำพูด

ในภาษา JavaScript มีประเภทข้อมูลพื้นฐาน (Primitive data type) อยู่ 6 ประเภท ซึ่งสามารถสรุปได้ดังตารางด้านล่างนี้

ประเภทข้อมูล	คำอธิบาย	ตัวอย่าง Literal
Number	ตัวเลข	1, -10, 0.5, 1.8e6
String	ข้อความ	"my string" หรือ 'hello'
Boolean	ค่าความจริง	true และ false เท่านั้น
BigInt	ตัวเลขขนาดใหญ่	1n, 9007199254740992n
Symbol	สัญลักษณ์	Symbol("name")
Undefined	ไม่ได้กำหนดค่า	undefined

คำสั่ง if

คำสั่ง if เป็นคำสั่งที่ใช้สำหรับควบคุมเพื่อให้โปรแกรมทำงานบางอย่างตามเงื่อนไขที่กำหนด มันเป็นคำสั่งตรวจสอบเงื่อนไขที่เป็นพื้นฐานและเรียบง่ายที่สุดในภาษา JavaScript นี่เป็นรูปแบบการใช้งานของคำสั่ง if

```
if (condition) {
    // Statements
}
```

คำสั่ง if else

คำสั่ง if จะทำงานเมื่อเงื่อนไขเป็นจริง และถ้าเงื่อนไขไม่เป็นจริงโปรแกรมจะข้ามการทำงานนั้นไป อย่างไรก็ตาม เราสามารถกำหนดบล็อคของคำสั่ง else เพื่อให้โปรแกรมทำงานในกรณีที่เงื่อนไขของคำสั่ง if ไม่เป็นจริงได้ คำสั่ง else นั้นจะต้องใช้ร่วมกับคำสั่ง if เสมอ นี่เป็นรูปแบบการใช้งาน

```
if (condition) {
    // Statements
```

```
} else {
   // Statements
}
```

คำสั่ง else-if

ในตัวอย่างก่อนหน้า การใช้งานคำสั่ง if else เราสามารถทำให้โปรแกรมทำงานได้แบบสองทางเลือก ในความเป็นจริงแล้ว การตัดสินใจนั้นอาจมีมากกว่าสองทางเลือกได้ ยกตัวอย่างเช่น เมื่อคุณจะไปทำงาน คุณอาจต้องตัดสินไปว่าจะเดินไป ขับรถไป หรือนั่งรถโดยสารสาธารณะไป

ในการเขียนโปรแกรมก็เช่นกัน ในกรณีที่เราต้องการให้โปรแกรมเลือกการทำงานได้มากกว่าสองทางเลือก เราสามารถใช้คำสั่ง else if สำหรับเพิ่มเงื่อนไขเพิ่มเติมได้ นี่เป็นรูปแบบการใช้งาน

```
if (condition1) {
    // Statements
} else if (condition2) {
    // Statements
} else if (condition3) {
    // Statements
} else if (conditionN) {
    // Statements
} else {
    // Statements
}
```

คำสั่ง while loop

คำสั่ง while เป็นคำสั่งวนซ้ำที่เรียบง่ายที่สุดสำหรับสร้างลูปเพื่อให้โปรแกรมทำงานซ้ำตามเงื่อนไขที่กำหนดในขณะที่เงื่อนไขเป็นจริง นี่เป็นรูปแบบการใช้งานคำสั่ง while loop ในภาษา JavaScript

```
while (condition) {
    // Statements
}
```

ในการใช้งานคำสั่ง while loop เราจะต้องกำหนดเงื่อนไข condition โปรแกรมจะทำงานในลูปในขณะที่เงื่อนไขเป็นจริง และจบการทำงานของลูปเมื่อเงื่อนไขเป็นเท็จ ดังนั้นภายในลูปเราจะต้องทำอะไรบางอย่างเพื่อทำให้เงื่อนไขเป็นเท็จเพื่อจบการทำงานของลูป

การทำงานแบบวนซ้ำเป็นหัวใจหลักของการเขียนโปรแกรม มันเป็นเรื่องพื้นฐานที่ไม่ได้มีแค่ในคอมพิวเตอร์ แต่สามารถพบได้ในชีวิตประจำวันของเรา เช่น คุณจะไปทำงานในทุกๆ วัน และทำแบบเดิมๆ ในแต่ละวัน

ต่อไปเราจะมาดูวิธีการใช้งานคำสั่ง while loop เพื่อนับและแสดงตัวเลขจาก 1 - 10 ออกทางหน้าจอ นี่เป็นตัวอย่าง

while.js

```
let i = 1;
while (i <= 10) {
    console.log(i);
    i++;
}
console.log("Loop ended");
```

นี่เป็นตัวอย่างที่ง่ายที่สุดสำหรับการใช้งานลูป เราได้แสดงตัวเลข 1 - 10 ออกทางหน้าจอด้วยคำสั่งวนซ้ำ while loop และนี่เป็นผลลัพธ์การทำงานของโปรแกรม

คำสั่ง do-while loop

คำสั่ง do-while ใช้สำหรับกำหนดให้โปรแกรมทำงานซ้ำเหมือนกับคำสั่ง while loop แต่สิ่งที่แตกต่างกันคือเงื่อนไขจะถูกตรวจสอบในตอนท้ายของลูป นั่นหมายความว่าในการใช้งานคำสั่ง do-while loop โปรแกรมจะทำงานในลูปอย่างน้อยหนึ่งรอบเสมอ นี่เป็นรูปแบบการใช้งาน

```
do {
    // Statements
} while (condition);
```

เมื่อโปรแกรมพบกับคำสั่ง do-while มันทำงานภายในลูปก่อนหนึ่งรอบทันที จากนั้นค่อยตรวจสอบเงื่อนไข หากเป็นจริงโปรแกรมทำงานลูปในรอบถัด หรือจบการทำงานของลูปหากเงื่อนไขเป็นเท็จ

นี่เป็นตัวอย่างการใช้งานคำสั่ง do-while loop สำหรับนับเลขจาก 1 - 10 เช่นเดิม

```
let i = 1;

do {
    console.log(i);
    i++;
} while (i <= 10);

console.log("Loop ended");
```

คำสั่ง for loop

คำสั่ง for loop เป็นคำสั่งควบคุมการทำงานแบบวนซ้ำที่ใช้สำหรับควบคุมเพื่อให้โปรแกรมทำงานบางอย่างซ้ำๆ ในขณะที่เงื่อนไขเป็นจริง โดยทั่วไปแล้วเรามักใช้คำสั่ง for loop ในกรณีลูปที่จำนวนการวนรอบที่แน่นอน นี่เป็นรูปแบบของการใช้งานคำสั่ง for loop ในภาษา JavaScript

```
for (initialize; condition; changes) {
    // Statements
}
```

ในการใช้งานคำสั่ง for loop นั้นเราสามารกำหนดค่าเริ่มต้น initialize เงื่อนไข condition และการเปลี่ยนแปลงในส่วนหัวของคำสั่งได้ โดยคั่นแต่ละส่วนด้วยเครื่องหมายเซมิโคลอน (;) และคำสั่ง for loop มีขั้นตอนการทำงานดังนี้

- **initialize:** คือการประกาศตัวแปรสำหรับใช้ภายในลูป ทำงานในตอนที่ลูปเริ่มต้นเท่านั้น
- **condition:** คือการกำหนดเงื่อนไขให้กับลูปเพื่อทำงาน โปรแกรมจะทำงานในลูปในขณะที่เงื่อนไขเป็นจริง
- **changes:** หลังจบการทำงานของลูปแต่ละรอบ เราสามารถเปลี่ยนแปลงค่าของตัวแปรได้ที่นี่

สำหรับตัวอย่างแรกในการใช้งานคำสั่ง for loop นั้นจะเป็นโปรแกรมนับตัวเลขจาก 1 - 5 และแสดงมันออกทางหน้าจอ มันเป็นโปรแกรมที่ง่ายที่สุดที่สามารถแสดงให้เห็นได้โดยการใช้ลูป นี่เป็นตัวอย่าง

```
for (let i = 1; i <= 5; i++) {
    console.log(i);
}
console.log("Loop ended");
```

คำสั่ง for loop กับอาเรย์

การใช้งานคำสั่ง for loop ที่พบได้บ่อยในการเขียนโปรแกรมก็คือใช้ร่วมกับอาเรย์ เนื่องจากอาเรย์นั้นมีการเก็บข้อมูลเป็นชุดลำดับและมี Index สำหรับการเข้าถึงข้อมูลเป็นตัวเลข ดังนั้นเราสามารถใช้คำสั่ง for loop เพื่อสร้าง Index สำหรับเข้าถึงข้อมูลในอาเรย์ได้ นี่เป็นตัวอย่าง

array_for.js

```
let fruits = ["Apple", "Banana", "Orange", "Grape", "Lemon"];

console.log("List of fruits");


for (let i = 0; i < fruits.length; i++) {

    console.log("%d. %s", i + 1, fruits[i]);

}


let languages = ["GO", "JavaScript", "PHP", "Python", "Ruby"];

console.log("List of programming languages (reversed)");


let count = 1;

for (let i = languages.length - 1; i >= 0; i--) {

    console.log("%d. %s", count, languages[i]);

    count++;

}
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
List of fruits

1. Apple

2. Banana

3. Orange

4. Grape

5. Lemon

List of programming languages (reversed)

1. Ruby

2. Python

3. PHP

4. JavaScript

5. GO
```

ฟังก์ชัน ในภาษา JavaScript

13 September 2020
ในบทนี้ คุณจะได้เรียนรู้เกี่ยวกับฟังก์ชันในภาษา JavaScript เราจะพูดถึงการประกาศและใช้งานฟังก์ชัน และคุณสมบัติที่สำคัญเกี่ยวกับฟังก์ชัน นี่เป็นเนื้อหาในบทนี้

- ฟังก์ชันคืออะไร
- การประกาศและใช้งานฟังก์ชัน
- การส่งค่ากลับ (return)

- Function expression
- ฟังก์ชันพารามิเตอร์กับค่าเริ่มต้น
- Function rest parameters
- ตัวอย่างการใช้งานฟังก์ชัน

ฟังก์ชันคืออะไร

ฟังก์ชัน (Function) คือกลุ่มของชุดคำสั่งที่ถูกรวมเข้าด้วยกันสำหรับการทำงานบางอย่าง ฟังก์ชันสามารถรับพารามิเตอร์เพื่อนำมาทำงานและส่งค่ากลับได้ นอกจากนี้การสร้างฟังก์ชันทำให้เราสามารถเรียกใช้มันซ้ำๆ ซึ่งเป็นการนำโค้ดกลับมาใช้ใหม่ได้ (Reusable) ซึ่งนี่เป็นแนวคิดพื้นฐานของการใช้งานฟังก์ชันในการเขียนโปรแกรม

ก่อนที่จะใช้งานฟังก์ชัน มันจะต้องถูกประกาศหรือสร้างขึ้นมาก่อน นี่เป็นรูปแบบของการประกาศฟังก์ชันในภาษา JavaScript

```
function name(param1, param2, ...) {
    // Statements
    return value; // Optional
}
```

การประกาศฟังก์ชันจะใช้คำสั่ง function ตามด้วยชื่อของฟังก์ชัน name การตั้งชื่อของฟังก์ชันนั้นจะเหมือนกับตัวแปร สามารถประกอบไปด้วยตัวอักษร ตัวเลข และสัญลักษณ์ Underscore (_) ในกรณีที่ชื่อของฟังก์ชันมีหลายคำมักจะกำหนดในรูปแบบ **camelCase**

ในวงเล็บหลังจากชื่อฟังก์ชันเป็นการกำหนดพารามิเตอร์ มันใช้สำหรับรับค่าในตอนที่ฟังก์ชันถูกเรียกใช้งาน พารามิเตอร์นั้นเป็นตัวแปรที่มีขอบเขตอยู่ภายในฟังก์ชัน และในกรณีที่ฟังก์ชันไม่มีพารามิเตอร์ วงเล็บจะถูกว่างเปล่าเอาไว้

ฟังก์ชันสามารถส่งค่ากลับไปยังที่ที่มันถูกเรียกใช้ได้ โดยการใช้คำสั่ง return และตามด้วยค่าที่ต้องการส่งกลับ ถ้าหากเราละเว้นการส่งค่ากลับ undefined จะเป็นค่าที่ส่งกลับไปแทน

การประกาศและใช้งานฟังก์ชัน

หลังจากคุณได้ทราบวิธีการประกาศฟังก์ชันแล้ว ต่อไปมาเริ่มประกาศฟังก์ชันในภาษา JavaScript เพื่อใช้งานกัน นี่เป็นตัวอย่าง

```
function currentDate() {
    let date = new Date().toDateString();
    console.log("The current date is: " + date);
}

function sayHi(name) {
    console.log("Hi " + name);
}

currentDate();
sayHi("Metin");
sayHi("JavaScript");
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
The current date is: Tue Aug 11 2020
Hi Metin
Hi JavaScript
```

เราได้ประกาศฟังก์ชันมาสองฟังก์ชันสำหรับการทำงานที่แตกต่างกัน ฟังก์ชัน currentDate ใช้สำหรับแสดงวันที่ปัจจุบันออกทางหน้าจอ ในขณะที่ฟังก์ชัน sayHi แสดงคำกล่าวทักทายจากชื่อที่ส่งเข้าไปยังฟังก์ชัน

```
currentDate();

sayHi("Metin");

sayHi("JavaScript");
```

จากนั้นเป็นการเรียกใช้งานฟังก์ชันด้วยชื่อของมัน ฟังก์ชัน currentDate เป็นฟังก์ชันที่ไม่มีพารามิเตอร์ ดังนั้นตอนเรียกใช้ไม่ต้องส่งอาร์กิวเมนต์เข้าไปยังฟังก์ชัน ส่วนฟังก์ชัน sayHi มีหนึ่งพารามิเตอร์คือ name ดังนั้นในตอนเรียกใช้ต้องส่งอาร์กิวเมนต์สำหรับพารามิเตอร์ดังกล่าวด้วย ในกรณีนี้ "Metin" และ "JavaScript" คืออาร์กิวเมนต์ที่ส่งไปยังพารามิเตอร์ของฟังก์ชัน

สังเกตว่าเราเรียกใช้งานฟังก์ชัน sayHi ถึงสองครั้งและนี่คือข้อดีของฟังก์ชัน เราสามารถเรียกใช้งานฟังก์ชันกี่ครั้งก็ได้หลังจากที่มันถูกประกาศไปแล้ว

ในบทนี้คุณอาจจะเห็นเราใช้คำว่า **พารามิเตอร์ (Parameter)** และ **อาร์กิวเมนต์ (Argument)** ควบคู่กัน พารามิเตอร์นั้นเป็นตัวแปรที่ประกาศเพื่อรับค่าในฟังก์ชัน ในขณะที่อาร์กิวเมนต์เป็นค่าที่ส่งในตอนเรียกใช้ฟังก์ชันไปยังพารามิเตอร์ของฟังก์ชัน ซึ่งจะสอดคล้องกันดังต่อไปนี้

```
// Function definition
function fnName(param1, param2, ...) {
    // Do something
}


// Calling function
```

```
fnName(arg1, arg2, ...);
```

จะกล่าวได้ว่าฟังก์ชันพารามิเตอร์และอาร์กิวเมนต์นั้นเป็นค่าหรือสิ่งเดียวกัน แต่มันเพียงเกิดขึ้นคนละที่ในโปรแกรม และเมื่อเรียกใช้ฟังก์ชันจำนวนอาร์กิวเมนต์ที่ส่งควรจะเท่ากับจำนวนของพารามิเตอร์ที่ฟังก์ชันมีเสมอ

การส่งค่ากลับ (return)

ในตัวอย่างก่อนหน้า เรายังไม่ได้ส่งค่าใดๆ กลับมาจากฟังก์ชัน เราสามารถส่งค่ากลับจากฟังก์ชันโดยการใช้คำสั่ง return ต่อไปเป็นตัวอย่างของฟังก์ชันสำหรับหาผลรวมของตัวเลขสองตัว และส่งผลลัพธ์ที่ได้กลับมา นี่เป็นตัวอย่าง

sum.js

```
function sum(a, b) {
    return a + b;
}


console.log("4 + 5 = " + sum(4, 5));
console.log("1 + -3 = " + sum(1, -3));

let doubleSum = sum(5, 5) * 2;
console.log("(5 + 5) * 2 = " + doubleSum);
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
4 + 5 = 9
1 + -3 = -2
```

```
(5 + 5) * 2 = 20
```

ในตัวอย่าง เป็นการประกาศและใช้งานฟังก์ชันสำหรับหาผลรวมของตัวเลขสองตัว ฟังก์ชัน sum มีสองพารามิเตอร์คือ a และ b ซึ่งเป็นตัวเลขที่ต้องการส่งเข้ามาหาผลรวมภายในฟังก์ชัน และฟังก์ชันส่งค่ากลับเป็นผลรวมของตัวเลขทั้งสองด้วยคำสั่ง return

ในการทำงานของฟังก์ชันนั้น คำสั่ง return เป็นการสั่งให้ฟังก์ชันจบการทำงานและส่งค่ากลับไปยังจุดที่เรียกใช้งานฟังก์ชัน เราสามารถใช้มันเพื่อจบการทำงานและข้ามการทำงานในคำสั่งที่เหลือของฟังก์ชันได้ มาดูตัวอย่างของฟังก์ชันสำหรับหาค่า Factorial ของตัวเลข และการใช้งานคำสั่ง return ในฟังก์ชัน

factorial.js

```
function factorial(n) {
    if (n == 0) {
        return 1;
    }
    let fac = 1;
    for (let i = 1; i <= n; i++) {
        fac *= i;
    }
    return fac;
}


console.log("0! = " + factorial(0));
console.log("3! = " + factorial(3));
```

```
console.log("5! = " + factorial(5));
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
0! = 1

3! = 6

5! = 120
```

ฟังก์ชัน factorial นั้นใช้สำหรับหาค่า Factorial โดยรับพารามิเตอร์เป็นตัวเลขจำนวนเต็ม n ที่ส่งเข้ามายังฟังก์ชันเพื่อนำมาคำนวณ

```
if (n == 0) {
    return 1;
}
```

ในฟังก์ชันเราได้ตรวจสอบเงื่อนไขว่าถ้าค่าในตัวแปร n ที่ส่งเข้ามามีค่าเท่ากับ 0 เราใช้คำสั่ง return ส่งค่ากลับเป็น 1 ในทันที นั่นทำให้ฟังก์ชันจบการทำงานและไม่ทำสั่งอื่นๆ ต่อ และในกรณีที่ค่าในตัวแปร n ไม่เท่ากับ 0 ฟังก์ชันทำงานปกติจนมันพบกับคำสั่ง return fac ด้านล่างเพื่อจบการทำงาน

ฟังก์ชันนั้นสามารถส่งค่ากลับซึ่งไม่ได้จำกัดเพียงประเภทข้อมูลพื้นฐาน แต่เป็นข้อมูลประเภทใดๆ ในภาษา JavaScript ยกตัวอย่างเช่น

```
function getUser() {
    return {
        id: 1,
        name: "Metin"
    };
```

```
};


function getFunction() {
    return function () {
        console.log("I'm a function");
    };
}
```

ฟังก์ชัน getUser ส่งค่ากลับเป็น Object literal ที่ประกอบไปด้วย Property name และ id ส่วนฟันก์ชัน getFunction นั้นเป็นฟังก์ชันที่ส่งค่ากลับเป็นฟังก์ชัน ซึ่งเราเรียกฟังก์ชันประเภทนี้ว่า **Closure**

Function expression

Function expression คือรูปแบบการประกาศฟังก์ชันซึ่งฟังก์ชันจะถูกสร้างขึ้นในขณะที่โปรแกรมทำงาน ในภาษา JavaScript ฟังก์ชันนั้นถือว่าเป็นค่าๆ หนึ่งที่มีประเภทข้อมูลเป็น function และสามารถกำหนดเป็นค่าของตัวแปรเหมือนกับออบเจ็คประเภทอื่นๆ ได้ นี่เป็นตัวอย่างของการประกาศ Function expression ในภาษา JavaScript

```
let sayHi = function (name) {
    console.log("Hi " + name);
};


let sum = function (a, b) {
    return a + b;
};
```

```
sayHi("Metin");

console.log(sum(2, 3));
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
Hi Metin

5
```

เราได้ประกาศสองฟังก์ชัน Function expression และกำหนดค่ามันให้กับตัวแปร sayHi และ sum ตามลำดับ เมื่อเรากำหนดค่าเป็นฟังก์ชันให้กับตัวแปร นั่นจะทำให้เราสามารถใช้ตัวแปรเหมือนกับว่ามันเป็นฟังก์ชันได้

สิ่งที่แตกต่างระหว่าง Function expression และฟังก์ชันปกติก็คือ Function expression นั้นจะถูกสร้างในตอนที่โปรแกรมทำงาน ในขณะที่ฟังก์ชันปกติจะถูกสร้างในตอนที่โปรแกรมคอมไพล์ ยกตัวอย่างเช่น

```
sayHi("Metin");     // Hi Metin


function sayHi(name) {
    console.log("Hi " + name);
}
```

ในตัวอย่างเป็นการประกาศฟังก์ชัน sayHi ในรูปแบบปกติ และโปรแกรมนี้จะสามารถทำงานได้เมื่อโปรแกรมรัน เพราะว่าฟังก์ชันในภาษา JavaScript นั้นถูกสร้างในตอนที่โปรแกรมคอมไพล์ นั่นหมายความว่าในตอนที่เราเรียกใช้ฟังก์ชัน sayHi ในบรรทัดแรก ฟังก์ชันได้ถูกสร้างมาก่อนแล้ว

ตอนนี้ลองมาเปลี่ยนเป็นประกาศฟังก์ชันในรูปแบบของ Function expression แทน

```
sayHi("Metin"); // Error


let sayHi = function (name) {
    console.log("Hi " + name);
};


// Call here will work
```

เมื่อเรารันโปรแกรมนี้จะเกิดข้อผิดพลาดขึ้น เนื่องจากในบรรทัดที่เราเรียกใช้งาน sayHi มันยังไม่ได้ถูกสร้างขึ้น ซึ่งนี่เป็นข้อผิดพลาดทั่วไปเหมือนกับการใช้งานตัวแปรที่ไม่มีอยู่ และเพื่อใช้งานฟังก์ชัน sayHi เราต้องใช้หลังจากที่มันได้ประกาศไปแล้วนั่นเอง

ฟังก์ชันพารามิเตอร์กับค่าเริ่มต้น

ฟังก์ชันในภาษา JavaScript นั้นสามารถที่จะกำหนดค่าเริ่มต้นให้กับพารามิเตอร์ของฟังก์ชันได้ เมื่อพารามิเตอร์มีค่าเริ่มต้น เราสามารถที่จะส่งหรือไม่ส่งค่าสำหรับพารามิเตอร์นั้นมาก็ได้ และถ้าหากไม่ส่งค่าเริ่มต้นจะถูกใช้งานแทน

จากตัวอย่างก่อนหน้า เราจะเขียนฟังก์ชัน sayHi ใหม่และกำหนดค่าเริ่มต้นให้กับพารามิเตอร์ name นี่เป็นตัวอย่าง

default_parameter.js

```
function sayHi(name = "Guest") {
    console.log("Hi " + name);
}
```

```
sayHi();

sayHi("Metin");
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
Hi Guest

Hi Metin
```

ในตัวอย่าง เรากำหนดค่าเริ่มต้นให้กับพารามิเตอร์ name เป็น "Guest" เมื่อพารามิเตอร์ของฟังก์ชันมีการกำหนดค่าเริ่มต้น เราสามารถเรียกใช้งานฟังก์ชันโดยไม่ส่งค่ามาได้ ซึ่งค่าเริ่มต้นจะถูกใช้แทน

```
sayHi("Metin");
```

และเช่นเดิมเรายังสามารถเรียกใช้งานโดยการส่งค่าพารามิเตอร์ในรูปแบบปกติได้ การกำหนดค่าเริ่มต้นให้กับฟังก์ชันนั้นอำนวยความสะดวกเป็นอย่างมากในการเขียนโปรแกรม และฟังก์ชันเป็นจำนวนมากในภาษา JavaScript นั้นมีค่าเริ่มต้นให้กับพารามิเตอร์

ถ้าหากฟังก์ชันนั้นมีหลายพารามิเตอร์และเราต้องการกำหนดค่าเริ่มต้นให้กับพารามิเตอร์เหล่านั้น ค่าเริ่มต้นจะต้องถูกกำหนดมาจากพารามิเตอร์ที่อยู่ทางด้านขวาสุดเสมอ ยกตัวอย่างเช่น

```
function fn1(a, b, c = 3) {
    // Do something
}


function fn2(a, b = 2, c = 3) {
    // Do something
}
```

```
function fn3(a = 1, b = 2, c = 3) {
    // Do something
}
```

ในตัวอย่าง เป็นการกำหนดค่าเริ่มต้นให้กับฟังก์ชันในกรณีที่มันมีมากกว่าหนึ่งพารามิเตอร์ โดยเราต้องกำหนดให้กับพารามิเตอร์ที่อยู่ทางด้านขวาก่อนเสมอ และในการเรียกใช้งานฟังก์ชัน จำนวนค่าที่ส่งเข้ามาต้องมีอย่างน้อยเท่ากับจำนวนของพารามิเตอร์ที่ไม่มีค่าเริ่มต้น

```
// Can be called as
// At least 2 parameters
fn1(1, 2);
fn1(1, 2, 3);

// At least 1 parameter
fn2(1);
fn2(1, 2);
fn2(1, 2, 3);

// Parameters are optional
fn3();
fn3(1);
fn3(1, 2);
fn3(1, 2, 3);
```

ในตอนเรียกใช้งานฟังก์ชันนั้นค่าอาเรย์กิวเมนต์ที่ส่งเข้ามายังฟังก์ชันจะต้องมีอย่างน้อยเท่ากับจำนวนของพารามิเตอร์ที่ไม่ได้กำหนดค่าเริ่มต้นเสมอ และเนื่องจากฟังก์ชัน fn3 มีการกำหนดค่าเริ่มต้นให้กับพารามิเตอร์ทั้งหมด ดังนั้นเราสามารถเรียกใช้งานฟังก์ชันโดยไม่ส่งอาร์กิวเมนต์มาเลยก็ได้ ซึ่งค่าเริ่มต้นทั้งหมดจะถูกใช้แทน

Function rest parameters

Rest parameter คือการกำหนดพารามิเตอร์ของฟังก์ชันโดยที่เราสามารถเรียกใช้งานฟังก์ชันโดยส่งอาร์กิวเมนต์มาเป็นจำนวนเท่าไหร่ก็ได้ โดยที่ Rest parameter จะเก็บรวบรวมเอาไว้ในตัวแปรอาเรย์เพื่อใช้งานในฟังก์ชัน ก่อนเริ่มมาดูตัวอย่างของฟังก์ชัน sum สำหรับรวมตัวเลขสองตัวเข้าด้วยกันในตัวอย่างที่แล้ว

```
function sum(a, b) {
    return a + b;
}
```

และอย่างที่คุณเห็น ฟังก์ชันนี้ทำงานกับแค่สองพารามิเตอร์เท่านั้น แล้วถ้าหากเราต้องการให้ฟังก์ชันสามารถบวกตัวเลขสามตัวหรือห้าตัวได้ในฟังก์ชันเดียว เราจะทำแบบนั้นได้อย่างไร คำตอบก็คือการกำหนด Rest parameter ให้กับฟังก์ชันดังตัวอย่างต่อไปนี้

rest_parameter.js

```
function sum(...numbers) {
    let s = 0;
    for (let n of numbers) {
        s += n;
    }
    return s;
```

```
}


console.log(sum(3, 5));
console.log(sum(3, 5, 2));
console.log(sum(3, 5, 2, 16, 19));
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม ตอนนี้ฟังก์ชัน sum ของเราสามารถรองรับการส่งอาร์กิวเมนต์เป็นจำนวนเท่าไหร่ก็ได้

```
8
10
45
```

เพื่อกำหนดพารามิเตอร์ให้เป็น Rest parameter ใส่เครื่องหมาย ... หน้าชื่อของพารามิเตอร์ นั่นจะทำให้ค่าทั้งหมดที่ส่งเข้ามาถูกรวบรวมเป็นอาเรย์และเก็บไว้ในตัวแปร numbers และเราสามารถใช้มันเหมือนอาเรย์ปกติทั่วไป

นอกจากนี้ Rest parameter ยังสามารถใช้ร่วมกับพารามิเตอร์แบบปกติได้ ในกรณีนี้ เราต้องกำหนด Rest parameter ที่ด้านหลังสุดเสมอ ยกตัวอย่างเช่น

```
function myFunction(first, ...rest) {
    console.log("first: " + first);
    console.log("rest: " + rest);
}


myFunction(1, 2, 3, 4, 5);
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม

```
first: 1

rest: 2,3,4,5
```

เมื่อเราเรียกใช้งานฟังก์ชัน พารามิเตอร์แรกจะเก็บในตัวแปร first และพารามิเตอร์ที่เหลือจะถูกรวบรวมเป็นอาเรย์และเก็บในตัวแปร rest

ตัวอย่างการใช้งานฟังก์ชัน

ในตัวอย่างก่อนหน้า คุณอาจจะเห็นว่าโค้ดภายในฟังก์ชันของเรานั้นมีไม่มากแล้วทำไมเราถึงต้องสร้างมันเป็นฟังก์ชัน คำตอบก็คือเพื่อทำให้มันนำกลับมาใช้ซ้ำได้ และมันช่วยเราให้สามารถแยกโค้ดส่วนต่างๆ ออกจากกันได้ นี่เป็นตัวอย่างโปรแกรมสำหรับค้นหาจำนวนเฉพาะระหว่าง 1 - 100 ในภาษา JavaScript

prime_numbers.js

```
// Function declaration
function isPrime(n) {
    if (n == 1) {
        return false;
    }
    for (let i = 2; i <= n; i++) {
        if (n % i == 0 && n != i) {
            return false;
        }
    }
```

```
    return true;
}


// Main program
let primeNumbers = [];
for (let i = 1; i <= 100; i++) {
    if (isPrime(i)) {
        primeNumbers.push(i);
    }
}


console.log("Prime numbers from 1 to 100:");
console.log(primeNumbers.join(", "));
console.log(primeNumbers.length + " numbers in total");
```

นี่เป็นผลลัพธ์การทำงานของโปรแกรม จำนวนเฉพาะที่มีค่าอยู่ระหว่าง 1 - 100 นั้นมีจำนวน 25 ตัว

```
Prime numbers from 1 to 100:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97
25 numbers in total
```

เราได้สร้างฟังก์ชัน isPrime สำหรับตรวจสอบว่าตัวเลขเป็นจำนวนเฉพาะหรือไม่ ฟังก์ชันรับพารามิเตอร์เป็นตัวเลขจำนวนเต็มและส่งค่ากลับเป็น Boolean เพื่อบอกว่าตัวเลขเป็นจำนวนเฉพาะหรือไม่ เราได้แยกส่วนการตรวจสอบจำเป็นเฉพาะออกไปเป็นฟังก์ชัน

```
for (let i = 1; i <= 100; i++) {
    if (isPrime(i)) {
        primeNumbers.push(i);
    }
}
```

และในส่วนของโปรแกรมหลักนั้นจะทำแค่วนตัวเลขจาก 1 - 100 และเรียกใช้งานฟังก์ชัน isPrime ในตอนนี้ โค้ดส่วนของการตรวจสอบจำนวนเฉพาะถูกแบ่งแยกออกไป และมันทำให้โค้ดอ่านเข้าใจได้ง่ายขึ้น เนื่องจากชื่อของฟังก์ชันมีความหมายในตัวมันเอง และเราไม่จำเป็นต้องเขียนโค้ดจำนวนมากไว้ในคำสั่ง for loop