

# Node Version Control

# What is NVM?

NVM (Node Version Manager) เป็นโปรแกรมอรรถประโยชน์ บรรทัดคำสั่งที่ช่วยให้นักพัฒนาสามารถจัดการ Node.js หลายเวอร์ชันได้อย่างง่ายดาย NVM มี Interface ที่เรียบง่ายสำหรับติดตั้ง สลับ และลบเวอร์ชันของ Node.js ช่วยให้นักพัฒนาสามารถทำงานใน Project ต่างๆ ได้โดยไม่ต้องกังวลเกี่ยวกับปัญหาความเข้ากันได้ ด้วย NVM คุณสามารถรัน Node.js หลายเวอร์ชันบนเครื่องเดียวกันได้ ทำให้ NVM กลายเป็นเครื่องมือสำคัญสำหรับนักพัฒนา Node.js

```
PS C:\> nvm
Running version 1.1.7.

Usage:

  nvm arch                         : Show if node is running in 32 or 64 bit mode.
  nvm install <version> [arch]       : The version can be a node.js version or "latest" for the latest stable version.
                                         Optionally specify whether to install the 32 or 64 bit version (defaults to system arch).
                                         Set [arch] to "all" to install 32 AND 64 bit versions.
  nvm list [available]              : List the node.js installations. Type "available" at the end to see what can be installed. Aliased as ls.
  nvm on                           : Enable node.js version management.
  nvm off                          : Disable node.js version management.
  nvm proxy [url]                  : Set a proxy to use for downloads. Leave [url] blank to see the current proxy.
                                         Set [url] to "none" to remove the proxy.
  nvm node_mirror [url]            : Set the node mirror. Defaults to https://nodejs.org/dist/. Leave [url] blank to use default url.
  nvm npm_mirror [url]             : Set the npm mirror. Defaults to https://github.com/npm/cli/archive/. Leave [url] blank to default url.
  nvm uninstall <version>          : The version must be a specific version.
  nvm use [version] [arch]          : Switch to use the specified version. Optionally specify 32/64bit architecture.
                                         nvm use <arch> will continue using the selected version, but switch to 32/64 bit mode.
  nvm root [path]                 : Set the directory where nvm should store different versions of node.js.
                                         If <path> is not set, the current root will be displayed.
  nvm version                      : Displays the current running version of nvm for Windows. Aliased as v.
```

```
3. ayuth@Ayuths-MacBook-Pro: ~ (zsh)
15:22:43 in ~
→ nvm

Node Version Manager

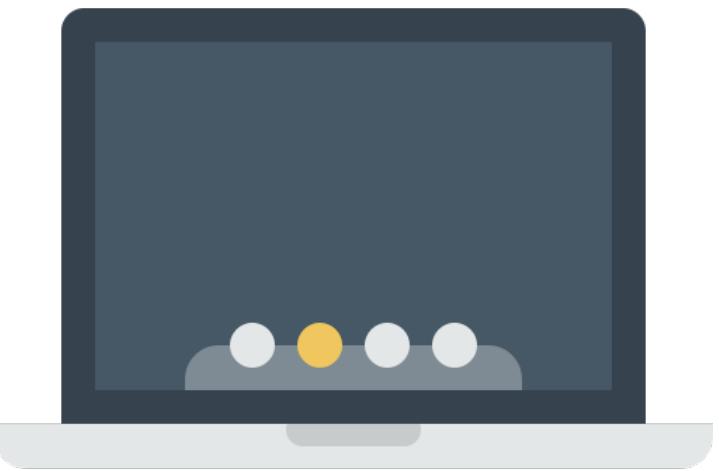
Note: <version> refers to any version-like string nvm understands. This includes:
  - full or partial version numbers, starting with an optional "v" (0.10, v0.1.2, v1)
  - default (built-in) aliases: node, stable, unstable, iojs, system
  - custom aliases you define with `nvm alias foo`

Any options that produce colorized output should respect the `--no-colors` option.

Usage:
  nvm --help                         Show this message
  nvm --version                      Print out the installed version of nvm
  nvm install [-s] <version>          Download and install a <version>, [-s] from source. Uses .nvmrc if available
  --reinstall-packages-from=<version>
  --lts                            When installing, reinstall packages installed in <node|iojs|node version number>
  --lts=<LTS name>                 When installing, only select from LTS (long-term support) versions
  --skip-default-packages           When installing, only select from versions for a specific LTS line
  --latest-npm                      When installing, skip the default-packages file if it exists
  --latest                           After installing, attempt to upgrade to the latest working npm on the given node

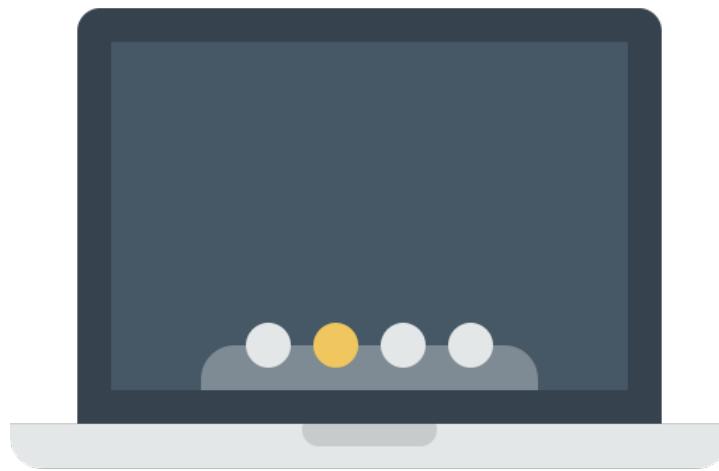
version
  --no-progress                     Disable the progress bar on any downloads
  nvm uninstall <version>           Uninstall a version
  nvm uninstall --lts               Uninstall using automatic LTS (long-term support) alias `lts/*`, if available.
  nvm uninstall --lts=<LTS name>   Uninstall using automatic alias for provided LTS line, if available.
  nvm use [--silent] <version>      Modify PATH to use <version>. Uses .nvmrc if available
  --lts                            Uses automatic LTS (long-term support) alias `lts/*`, if available.
  --lts=<LTS name>                Uses automatic alias for provided LTS line, if available.
  nvm exec [--silent] <version> [<command>] Run <command> on <version>. Uses .nvmrc if available
  --lts                            Uses automatic LTS (long-term support) alias `lts/*`, if available.
  --lts=<LTS name>                Uses automatic alias for provided LTS line, if available.
  nvm run [--silent] <version> [<args>] Run `node` on <version> with <args> as arguments. Uses .nvmrc if available
  --lts                            Uses automatic LTS (long-term support) alias `lts/*`, if available.
  --lts=<LTS name>                Uses automatic alias for provided LTS line, if available.
  nvm current                      Display currently activated version of Node
  nvm ls                           List installed versions
  nvm ls <version>                 List versions matching a given <version>
  nvm ls-remote                    List remote versions available for install
  --lts                            When listing, only show LTS (long-term support) versions
  nvm ls-remote <version>          List remote versions available for install, matching a given <version>
  --lts                            When listing, only show LTS (long-term support) versions
  --lts=<LTS name>                When listing, only show versions for a specific LTS line
  nvm version <version>            Resolve the given description to a single local version
  nvm version-remote <version>    Resolve the given description to a single remote version
  --lts                            When listing, only select from LTS (long-term support) versions
  --lts=<LTS name>                When listing, only select from versions for a specific LTS line
  nvm deactivate                   Undo effects of `nvm` on current shell
  nvm alias [<pattern>]            Show all aliases beginning with <pattern>
```

# What is NVM?



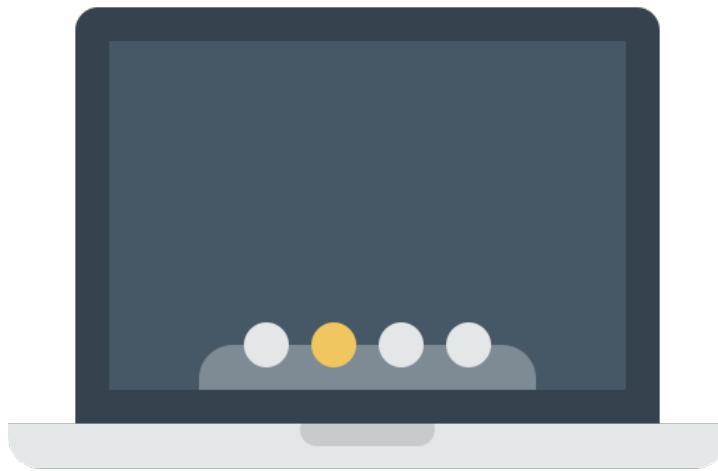
Project A

node version 10.15.0



Project B

node version 16.0.0



Project C

node version 14.15.1

# Installing NVM on Windows

เพื่อติดตั้ง nvm บน Windows คุณสามารถทำตามขั้นตอนดังต่อไปนี้:

- ดาวน์โหลดตัวติดตั้ง nvm สำหรับ Windows จากที่เก็บข้อมูลบน GitHub ของ nvm สำหรับ Windows.
- รันตัวติดตั้งและทำตามคำแนะนำที่ปรากฏบนหน้าจอ.
- เมื่อการติดตั้งเสร็จสิ้นแล้ว เปิดหน้าต่างคำสั่งใหม่หรือหน้าต่าง PowerShell.
- ทำการตรวจสอบการติดตั้งโดยการรันคำสั่งต่อไปนี้:

```
nvm --version
```

หากทุกอย่างติดตั้งและเป็นไปตามที่ควร คุณควรจะเห็นเลขเวอร์ชันของ NVM แสดงออกมานิหน้าต่างคำสั่ง และแสดงว่าการติดตั้งเสร็จสมบูรณ์และพร้อมใช้งานแล้ว.

# Installing NVM on macOS

เพื่อติดตั้ง NVM บน macOS โปรดทำตามขั้นตอนต่อไปนี้:

1. เปิดโปรแกรม Terminal ของคุณ.
2. ติดตั้ง NVM โดยการรันคำสั่งต่อไปนี้:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
```

หรือถ้าคุณใช้ cURL ไม่ได้ คุณสามารถใช้ wget แทน:

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
```

1. เมื่อการติดตั้งเสร็จสิ้น ปิดและเปิด Terminal ของคุณใหม่.
2. ทำการตรวจสอบการติดตั้งโดยการรันคำสั่งต่อไปนี้:

```
nvm --version
```

หากทุกอย่างติดตั้งและเป็นไปตามที่ควร คุณควรจะเห็นเลขเวอร์ชันของ NVM แสดงออกมานใน Terminal แสดงว่าการติดตั้งเสร็จสมบูรณ์และพร้อมใช้งานแล้ว.

# การใช้งาน Node version manager

คำสั่งแสดงเวอร์ชันของ Node.js ที่เราดาวน์โหลดลงมาในเครื่องแล้ว

```
nvm ls
```

คำสั่งนี้เป็นการแสดงเวอร์ชัน Node.js ออกมากทางหน้าจอ

```
nvm ls-remote
```

; เมื่อทำการรันบน command จะแสดงรายการเวอร์ชัน Node.js ที่คุณติดตั้งได้โดยใช้ NVM

การ Install Node version ที่เราต้องการจะลงโดยกำหนด version  
ของ Node ของเราที่ต้องการลง

```
nvm install <version>
```

; เมื่อทำการรันบน command จะแสดงรายการเวอร์ชัน Node.js ที่คุณติดตั้งได้โดยใช้ NVM

# การใช้งาน Node version manager

คำสั่งนี้เป็นการเลือกใช้ node version ที่อยู่ในเครื่อง  
(เวลาดูเวอร์ชันที่อยู่ในเครื่องของเรามาจากคำสั่ง nvm ls )

```
nvm use <node version>
```

คำสั่งการสร้าง alias คือการสร้างนามแฝงให้กับ Node ของเรา เช่นการประกาศ  
คำสั่ง alias ให้กับชื่อ ProjectA โดยใช้ node v14.15.1 เราจะสามารถใช้ nvm  
use ProjectA ก็จะใช้คำสั่ง nvm use v14.15.1 ให้กับเราทันที

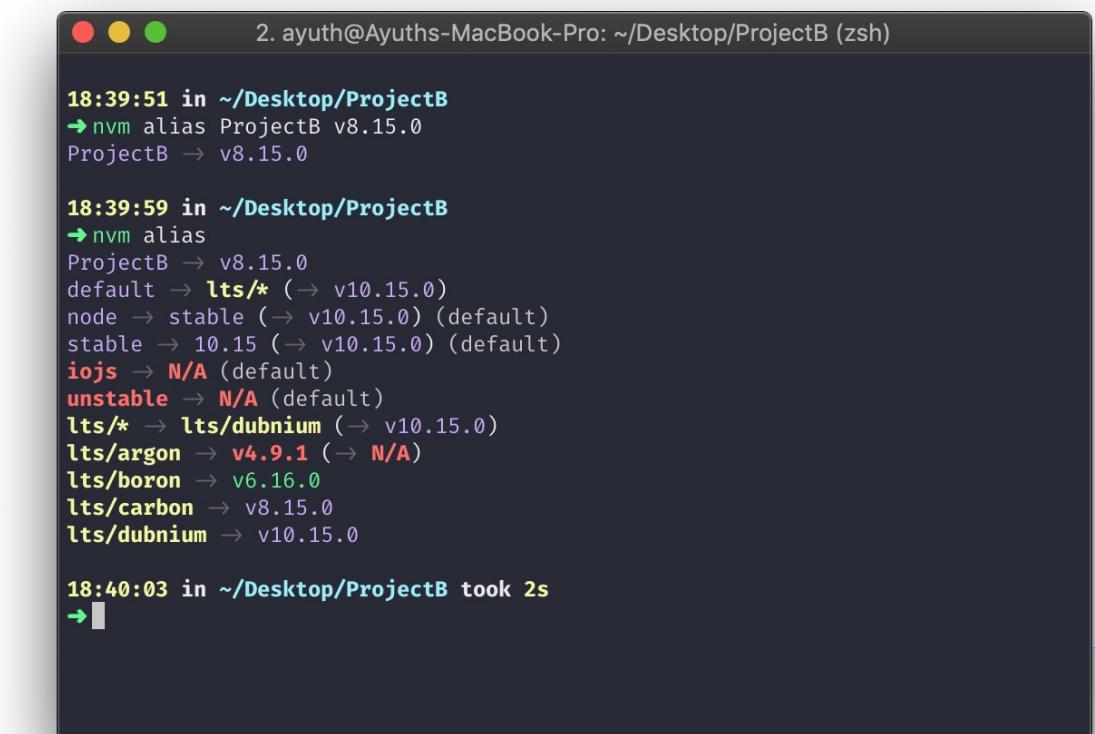
```
nvm alias <alias name> <node version>
```

; ตัวอย่างคำสั่งการสร้าง alias ให้กับ ProjectA เป็น node v14.15.1

```
nvm alias ProjectA v14.15.1
```

คำสั่งแสดงรายการ alias ทั้งหมดในเครื่องเรา

```
nvm alias
```



```
2. ayuth@Ayuths-MacBook-Pro: ~/Desktop/ProjectB (zsh)
18:39:51 in ~/Desktop/ProjectB
→ nvm alias ProjectB v8.15.0
ProjectB → v8.15.0

18:39:59 in ~/Desktop/ProjectB
→ nvm alias
ProjectB → v8.15.0
default → lts/* (→ v10.15.0)
node → stable (→ v10.15.0) (default)
stable → 10.15 (→ v10.15.0) (default)
iojs → N/A (default)
unstable → N/A (default)
lts/* → lts/dubnium (→ v10.15.0)
lts/argon → v4.9.1 (→ N/A)
lts/boron → v6.16.0
lts/carbon → v8.15.0
lts/dubnium → v10.15.0

18:40:03 in ~/Desktop/ProjectB took 2s
→ █
```

# วิธีการใช้ไฟล์ .nvmrc

ไฟล์ .nvmrc ทำให้เราทราบว่า Project นั้นใช้ node เวอร์ชันอะไรในการพัฒนาอยู่ และช่วยในการควบคุมการทำงานของ nvm ได้ง่ายเพียงพิมพ์คำสั่ง nvm use แต่ถ้าเรามีไฟล์นี้เจ้าตัว nvm ก็จะมาอ่านที่ไฟล์นี้แล้วไปใช้ node เวอร์ชันนั้น ๆ แต่ถ้าไม่มีมันก็จะบอกให้ ดาวน์โหลดมาโดยใช้คำสั่ง nvm install <version> เรา ก็ดาวน์โหลดลงและพิมพ์คำสั่ง nvm use ใหม่ก็จะใช้ได้แล้วล่ะ

## วิธีการสร้างไฟล์ .nvmrc

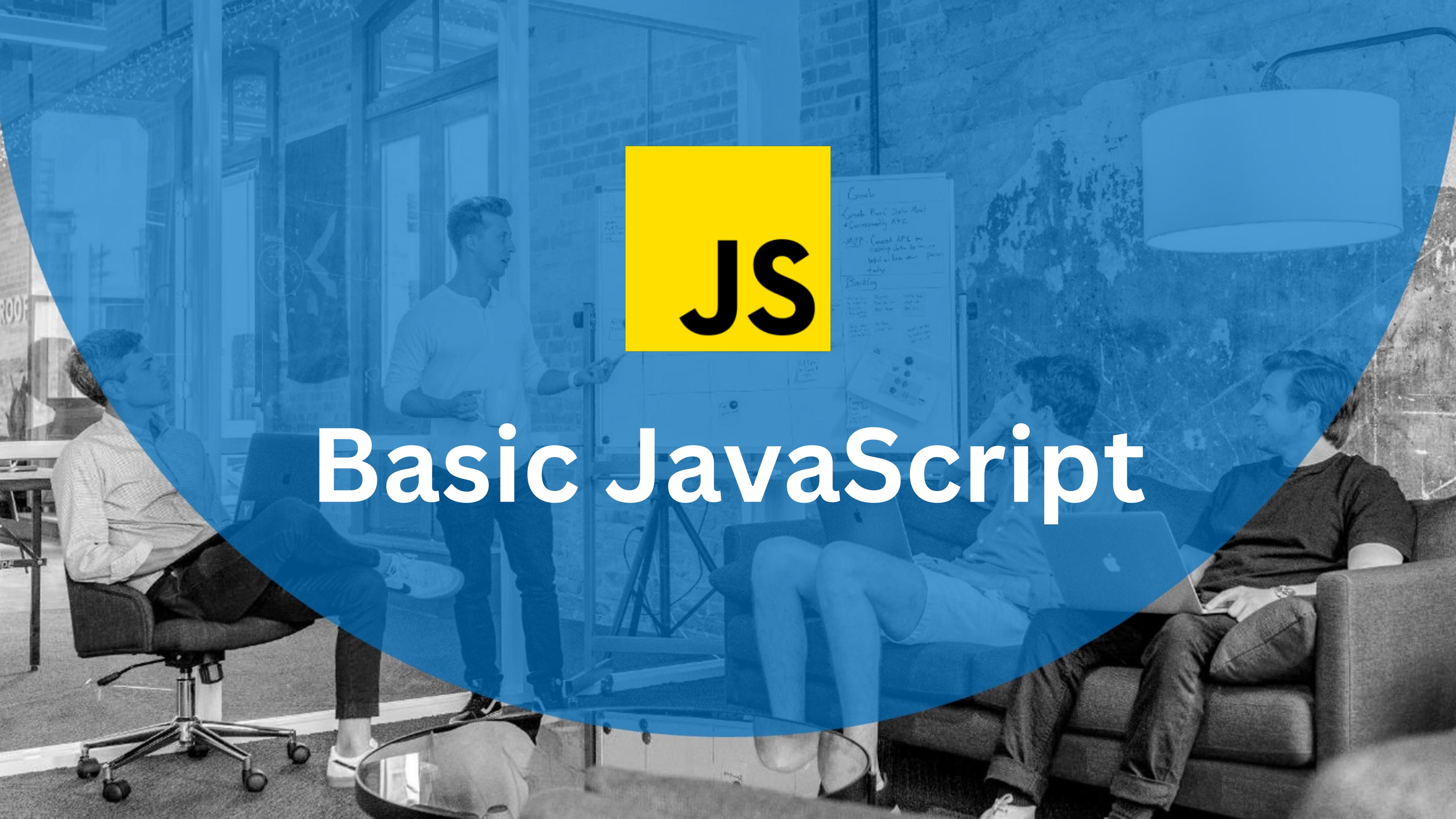
สำหรับการสร้างไฟล์ .nvmrc คือไปที่ folder นอกสุดของ project และใช้คำสั่ง  
(ถ้าเป็น windows สร้างไฟล์แล้วสามารถพิมพ์เวอร์ชันลงได้เลย)

```
node -v > .nvmrc
```

```
$ echo "5.9" > .nvmrc # ให้ใช้เวอร์ชันของ node เป็นเวอร์ชัน 5.9 เสมอ
$ echo "lts/*" > .nvmrc # ให้ใช้เวอร์ชัน lts ล่าสุดเสมอ
$ echo "node" > .nvmrc # ให้ใช้เวอร์ชันของ node เป็นเวอร์ชันล่าสุด
```

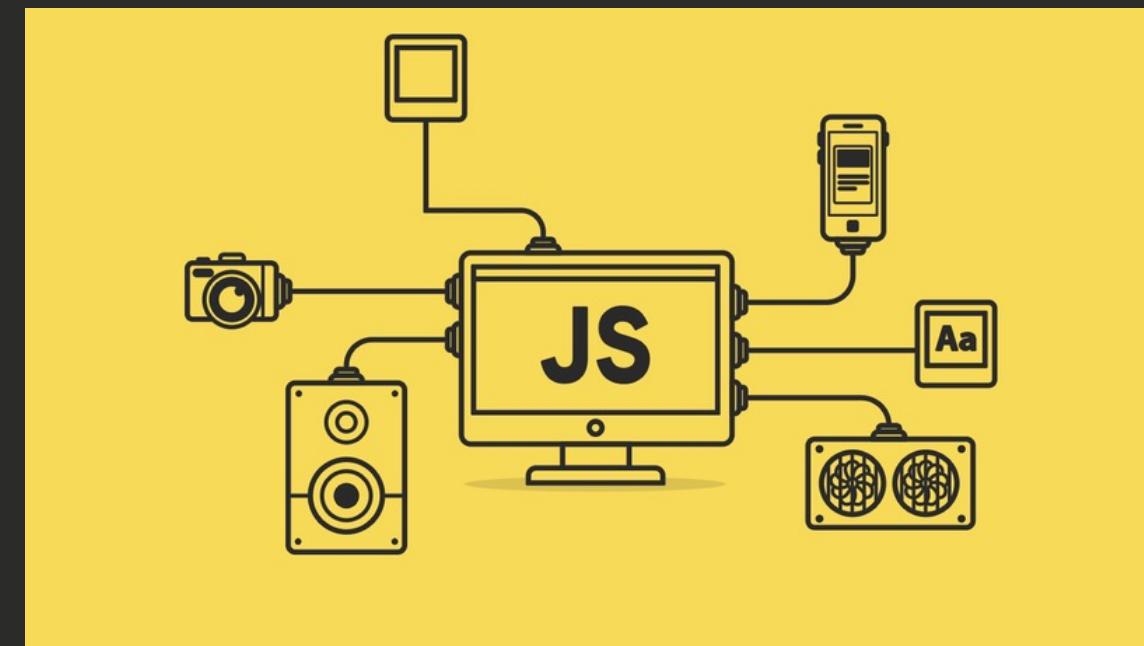
```
● ● ●
```

```
$ echo "5.9" > .nvmrc
$ echo "lts/*" > .nvmrc # to default to the latest LTS version
$ echo "node" > .nvmrc # to default to the latest version
```



JS

# Basic JavaScript



# JavaScript

ภาษาคอมพิวเตอร์ภาษาหนึ่งที่มีไว้เพื่อจัดการเอฟเฟ็ตหรือการทำงานของหน้าเว็บไซต์ นอกเหนือจาก HTML ที่ใช้จัดการเรื่องของเนื้อหาของเว็บ และ CSS ที่ใช้จัดการเรื่องของโครงสร้างและดีไซน์ของเว็บ

## Variable

ตัวแปร (Variable) คือชื่อหรือสัญลักษณ์ที่ใช้อ้างอิงถึงข้อมูลที่ถูกเก็บในหน่วยความจำ ตัวแปรใช้สำหรับเก็บข้อมูลในโปรแกรมเพื่อเข้าถึงข้อมูล โดยการใช้ชื่อของตัวแปรสำหรับอ้างถึงข้อมูลดังกล่าว เช่น การอ่านค่าในตัวแปร หรือการเปลี่ยนแปลงค่าของตัวแปรเป็นค่าใหม่



# Variables

การสร้าง variables หรือ ตัวแปร ประกอบด้วย 3 อาย่าง

Identifiers

Data type

Assignment operator

## Identifiers

- ตัวแปรต้องมีชื่อเฉพาะ ชื่อเฉพาะของตัวแปรจะเรียกว่า identifier
- ชื่อเฉพาะนิยมเขียนในลักษณะ Camel case
- ตัวแปรใน JavaScript เป็น case sensitive หมายความว่า ตัวแปรที่สะกดเหมือนกัน แต่ใช้ตัวพิมพ์ใหญ่พิมพ์เล็กต่างกัน จะถือว่า เป็นตัวแปรคละตัวกัน เช่น myName กับ myname เป็นตัวแปรคละตัวกัน
- ห้ามขึ้นต้นด้วยตัวเลขหรือสัญลักษณ์พิเศษ แต่สามารถขึ้นต้นด้วย \$(dollar sign) และ \_(underscore) ได้
- ไม่สามารถใช้ Keyword เป็นตัวแปรได้

# Identifiers

ตัวแปรต้องมีชื่อเฉพาะ ชื่อเฉพาะของตัวแปรจะเรียกว่า identifier

```
script.js ×  
1 var favariteFood = "pizza";  
2 var numOfSlice = 8  
3 console.log(favariteFood);  
4 console.log(numOfSlice);  
5 |
```

```
Console ×  
pizza  
8
```

รูปแบบการตั้งชื่อเฉพาะจะมีการใช้งานอยู่สองรูปแบบหลัก ๆ คือ Camel จะใช้กับ method + Class หรือชื่อเฉพาะ Snake case จะนิยมใช้กับ variable ธรรมชาติที่นำไป

```
script.js ×  
1 var camelCase = "Camel case";  
2 var snake_case = "Snake case";  
3 |
```

ตัวแปรใน JavaScript เป็น case sensitive คือ ตัวแปรที่สะกดเหมือนกันแต่ใช้ตัวพิมพ์ใหญ่พิมพ์เล็กต่างกัน จะถือว่า เป็นตัวแปรคนละตัวกัน เช่น myName กับ myname เป็นตัวแปรคนละตัวกัน

```
script.js ×  
1 const myName = "text 1"  
2 const myname = "text 2"  
3 console.log(myName)  
4 console.log(myname)
```

```
Console ×  
text 1  
text 2
```

# Identifiers

ห้ามขึ้นต้นด้วยตัวเลขหรือสัญลักษณ์พิเศษ แต่สามารถ  
ขึ้นต้นด้วย \$(dollar sign) และ \_(underscore) ได้

```
script.js ×          Console × ...
1 const 1myVariable = "variable";      Syntax error "m"  script.js? 1:7
2 const *myVariable = 2;
3 console.log(1myVariable)
4 console.log(*myVariable)

script.js ×          Console × ...
1 const $myVariable = "variable";
2 const _myVariable = 2;
3 console.log($myVariable)
4 console.log(_myVariable)
```

ไม่สามารถใช้ Keyword เป็นตัวแปรได้

## List of Keywords

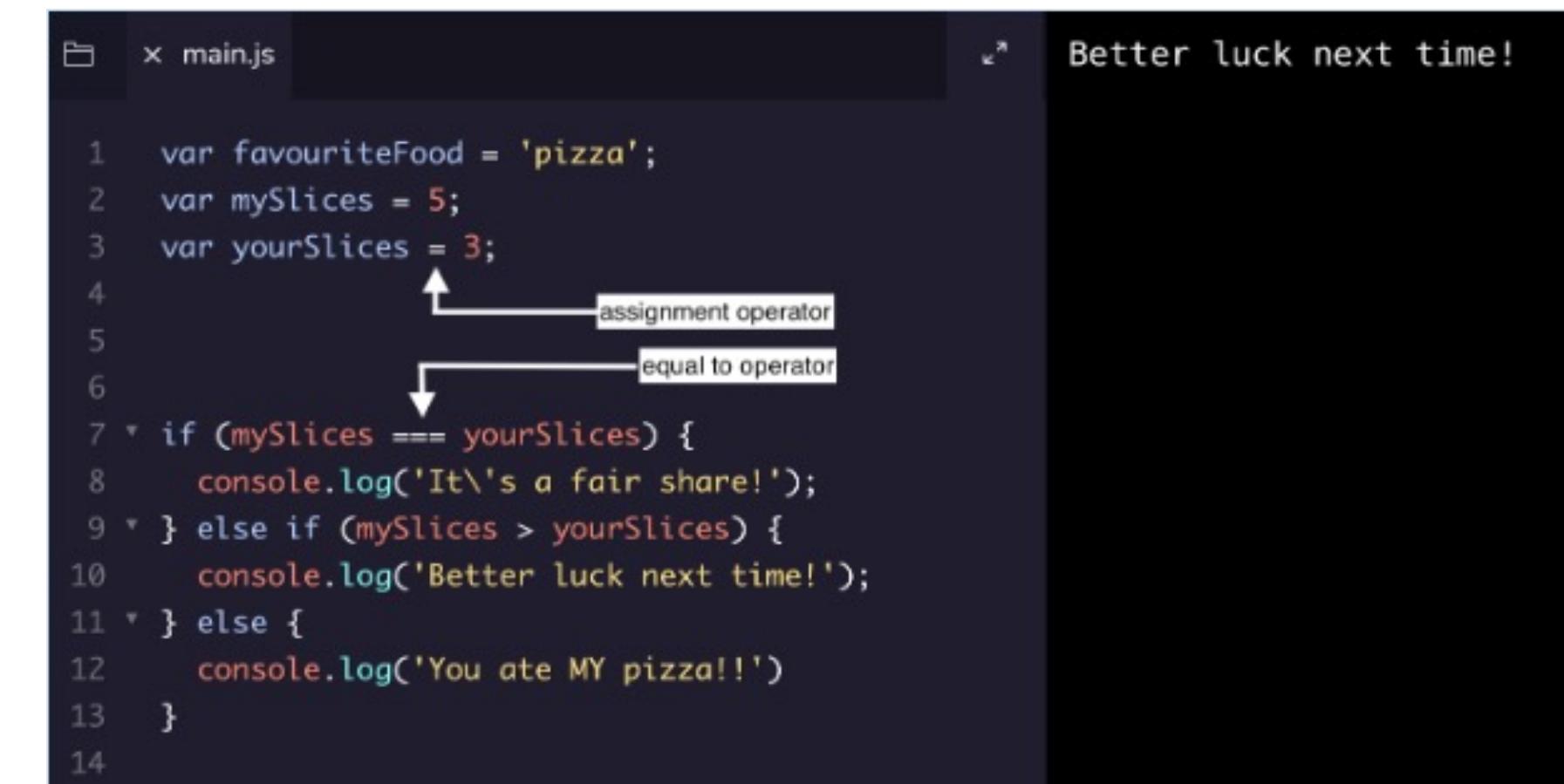
while	case	class	void	function	instanceof
throw	export	delete	catch	private	package
true	debugger	extends	default	interface	super
with	enum	if	return	switch	try
let	yield	typeof	public	for	static
new	else	finally	false	import	var
do	protected	null	in	implements	this
await	const	continue	break		

# Assignment operator

เครื่องหมาย assignment คือ เครื่องหมาย  
ที่กำหนดค่าของตัวแปร แทนด้วย =

เครื่องหมายเท่ากับหรือ equal to operator  
ใน JavaScript แทนด้วย ===

เครื่องหมาย = ไม่มีความหมายว่า  
“เท่ากับ” แต่เป็น “กำหนดให้เป็น”



```
File main.js
1 var favouriteFood = 'pizza';
2 var mySlices = 5;
3 var yourSlices = 3;
4 * if (mySlices == yourSlices) {
5     console.log('It\'s a fair share!');
6 } else if (mySlices > yourSlices) {
7     console.log('Better luck next time!');
8 } else {
9     console.log('You ate MY pizza!!')
10 }
11
12
13
14
```

Better luck next time!

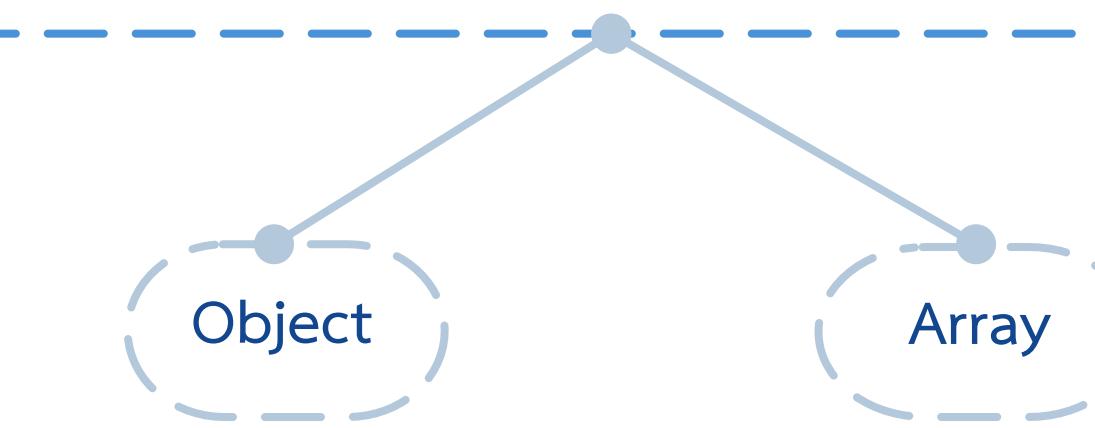
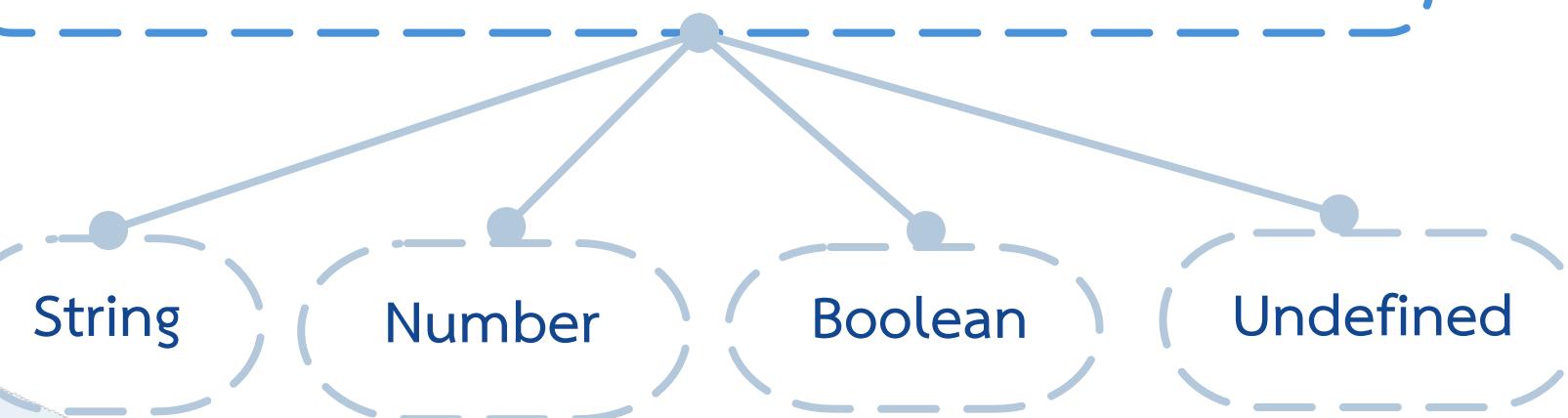
# Data Types

## Primitive Type

## Reference Types

Primitive data types คือชนิดของข้อมูลที่ไม่มีความซับซ้อน มีค่าเดียว ไม่มี properties และ/หรือ methods ชนิดของข้อมูลนี้

Reference Type คือเป็นการสร้างตัวแปรแบบอ้างอิง โดยที่เมื่อตัวแปรไหนเปลี่ยนตัวแปรอื่นก็จะเปลี่ยนตามด้วย จะใช้กับทั้ง Array และ Object



# String

ข้อมูลที่เป็นข้อความ หรือ string

ข้อความต้องอยู่ในเครื่องหมายคำพูด

สามารถใช้ได้ทั้ง 3 แบบ double quotes "", single quotes '' และ backticks ``

double quotes และ single quotes เป็น simple quotes

backticks เป็น extended functionality มีประโยชน์ในการ interpolate ตัวแปร โดยใช้ template literals

## script.js ×

```
1 let x = `พี`;
2 let y = "เค้ก";
3 let z = 'หล่อ';
```

## script.js ×

```
1 const your_name = 'อัครเดช';
2 const age = 30;
3
4 const message = `คุณ ${your_name}! คุณมีอายุ ${age} ปีแล้ว`;
5
6 console.log(message);
```

## Console ×

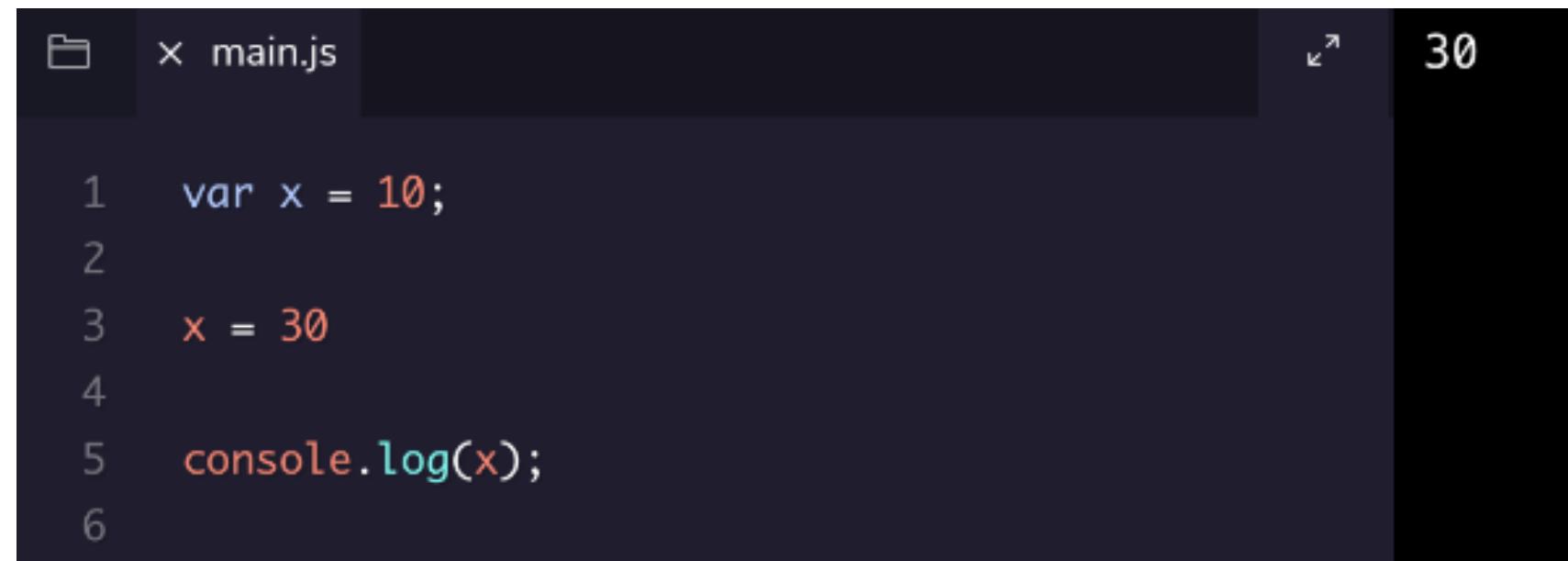
คุณ อัครเดช! คุณมีอายุ 30 ปีแล้ว

# Number

เป็นข้อมูลชนิดตัวเลข สามารถ reassing หรือ กำหนดค่าของตัวแปรใหม่ได้

ใช้ได้ทั้งกับจำนวนเต็มและทศนิยม

มี operator ที่สามารถใช้กับข้อมูลชนิดนี้ได้หลายตัว เช่น บวก ลบ คูณ หาร เป็นต้น



```
main.js
1 var x = 10;
2
3 x = 30
4
5 console.log(x);
6
```

30

# Number

## Special Numeric Values

### Infinity

```
File main.js | - Infinity  
1 console.log(Number.POSITIVE_INFINITY);  
2  
3 var y = 2/0;  
4 console.log(y);  
5  
6 var z = Infinity;  
7 console.log(z);
```

- จำนวนเต็มบวกใดๆที่หารด้วย 0 จะได้ผลลัพท์เป็น Infinity
- สามารถกำหนดค่าตัวแปรเป็น Infinity ได้ โดยการ assign ตัวแปรเป็น Infinity

### Negative Infinity

```
File main.js | -Infinity  
1 console.log(Number.NEGATIVE_INFINITY);  
2  
3 var y = -2/0;  
4 console.log(y);  
5  
6 var z = -Infinity;  
7 console.log(z);
```

- จำนวนเต็มลบใดๆที่หารด้วย 0 จะได้ผลลัพท์เป็น -Infinity
- สามารถกำหนดค่าตัวแปรเป็น -Infinity ได้ โดยการ assign ตัวแปรเป็น -Infinity

### NaN

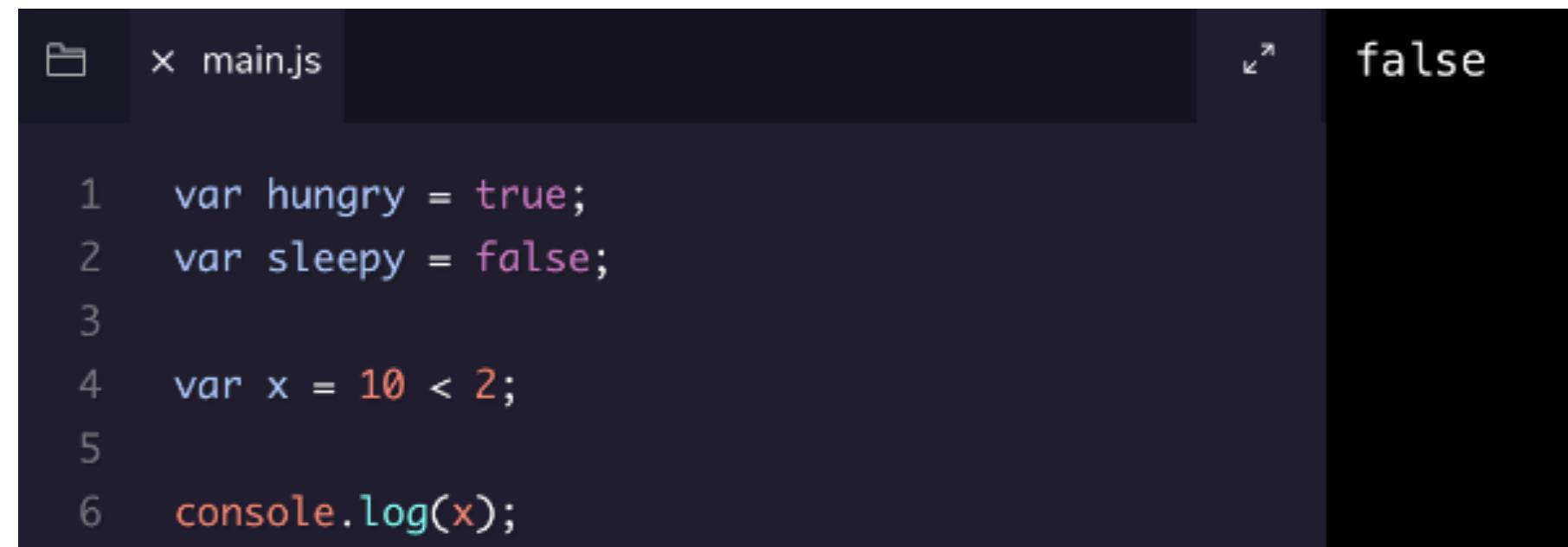
```
File main.js | NaN  
1 var x = 10;  
2  
3 x = 31.22;  
4  
5 y = 'hundred';  
6  
7 console.log(y/x);  
8
```

- NaN หรือ Not a Number หมายถึง ข้อผิดพลาดในการคำนวณ
- หากการคำนวณได้แสดงผล NaN การคำนวณอื่นที่เกี่ยวข้องจะกลับเป็น NaN ทั้งหมด

# Booleans

มีสองค่าเท่านั้น คือ true (เป็นจริง) และ false (เป็นเท็จ)

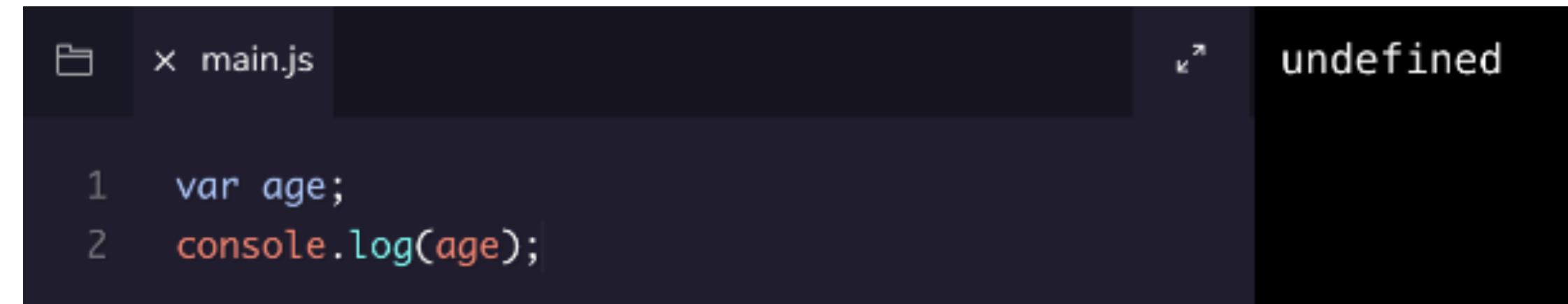
สามารถใช้ในการเปรียบเทียบเพื่อแสดงความเป็นจริงหรือเป็นเท็จได้



```
main.js
1 var hungry = true;
2 var sleepy = false;
3
4 var x = 10 < 2;
5
6 console.log(x);
```

# undefined

ค่า undefined หมายถึง มีการประกาศตัวแปร แต่ไม่ได้กำหนดค่าตัวแปร



```
main.js
1 var age;
2 console.log(age);
```

undefined

# Reference Types

Reference Types ใน JavaScript เป็นชนิดของข้อมูลที่อ้างอิงถึงค่าในหน่วยความจำ ซึ่งถูกเก็บเป็น "ตำแหน่ง" หรือ "การอ้างอิง" ไม่ใช่ค่าโดยตรง

## Object

แบบสำคัญที่สุดและมักใช้งานมากที่สุด ใช้สร้าง Object ใหม่ โดย Object นี้จะมีคุณสมบัติและเมธอดต่าง ๆ ที่สามารถเข้าถึงและแก้ไขได้

```
script.js ×  
1 var person = {  
2   name: "Jame",  
3   age: 27  
4 };
```

## Array

เป็นชนิดข้อมูลที่ใช้เก็บข้อมูลเป็นลำดับ มักใช้งานในการเก็บข้อมูลที่มีโครงสร้างเชิงลำดับ เช่น รายการสินค้า ลิสต์ของผู้ใช้ เป็นต้น

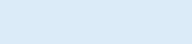
```
script.js ×  
1 var fruits = ["Apple", "Banana", "Orange"];  
2
```

## 🔍 การประกาศตัวแปร



ก่อนการสร้างตัวแปร ต้องมีการประกาศตัวแปร  
หรือที่เรียกว่า declaration  
ใน JavaScript สามารถประกาศตัวแปรได้ 3  
วิธี คือ การประกาศโดยใช้

- 1.var
- 2.let
- 3.const



## Var

- สามารถ reassign และ update ค่าของตัวแปรได้
- หากประกาศและสร้างตัวแปร แต่ไม่ได้กำหนดค่าของตัวแปร ตัวแปรจะมีค่าเป็น undefined
- ตัวแปร var เป็น function scope นั่นหมายความว่าตัวแปรนี้เมื่อถูกสร้างขึ้นใน function จะอ้างถึงได้ในเมื่อถึงใน function เท่านั้น
- หากตัวแปรไม่ได้ถูกสร้างใน function ตัวแปรนั้นนับเป็น global scope สามารถอ้างถึงได้ทั้งโปรแกรม

```
main.js
1 var x = 2;
2 x = 10;
3
4 if (true) {
5   var result = x + x;
6   console.log('The result is ' + result);
7 }
8
9 console.log(result);
```

The result is 20  
20

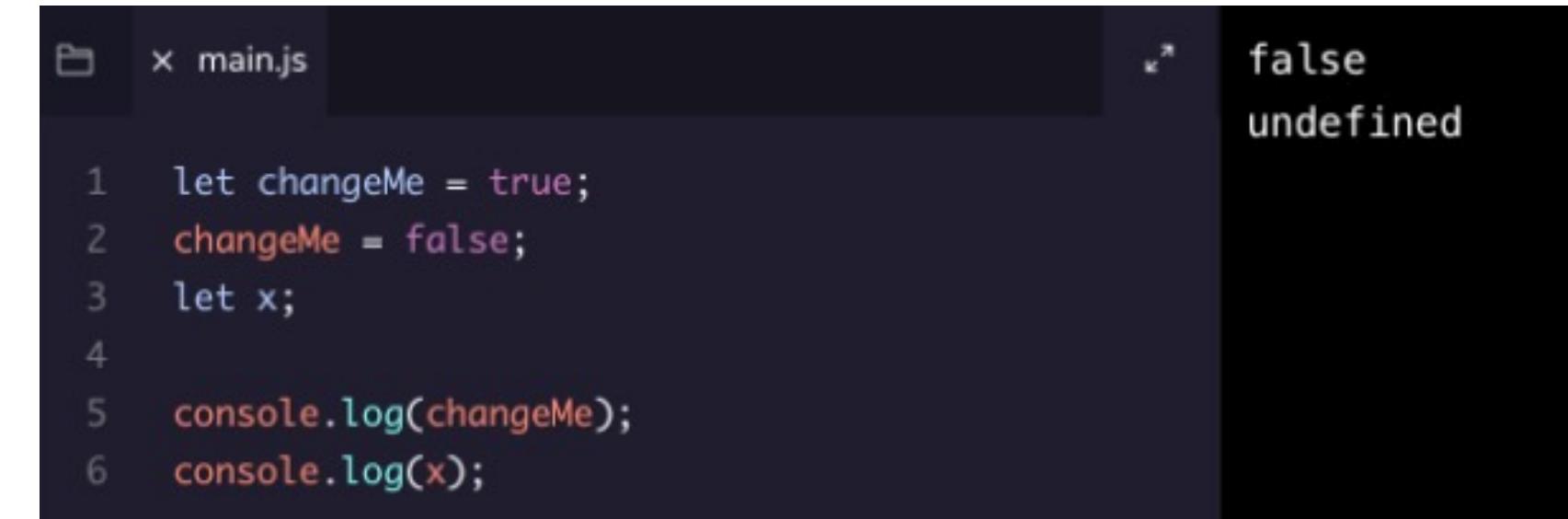
หากอ้างถึงตัวแปร var ที่ถูกสร้างใน function ข้างนอก function โปรแกรมจะแสดง ReferenceError

```
main.js
1 var x = 2;
2 x = 10;
3
4 function sum() {
5   var result = x + x;
6   console.log('The result is ' + result);
7 }
8
9 sum();
10 console.log(result);
```

The result is 20  
/home/ccuser/workspace/variables-var/main.js:10:13  
ReferenceError: result is not defined  
at Object.<anonymous>  
(/home/ccuser/workspace/variables-var/main.js:10:13)  
at Module.\_compile (module.js:571:3)  
at Object.Module.\_extensions..js  
(module.js:580:10)

## let

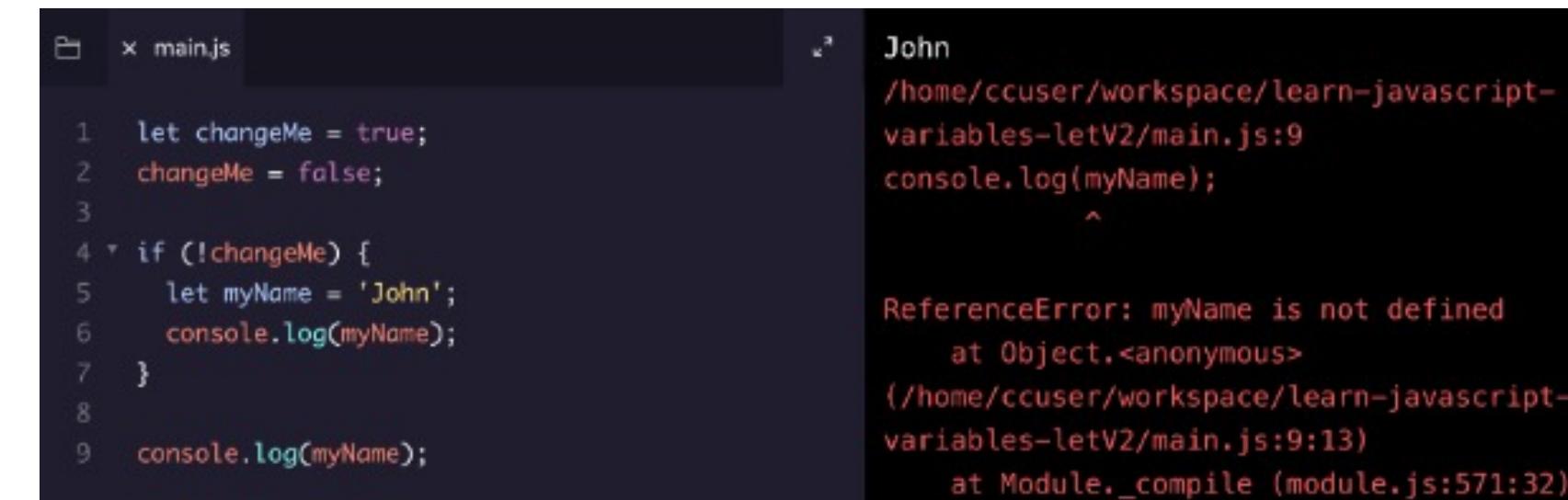
- สามารถ reassign และ update ค่าของตัวแปรได้
- หากประกาศและสร้างตัวแปร แต่ไม่ได้กำหนดค่าของตัวแปร ตัวแปรจะมีค่าเป็น undefined
- ตัวแปร let เป็นตัวแปรที่ใช้ใน block scope หมายความว่า ตัวแปรนี้สามารถถูกอ้างถึงได้ใน code block ที่สร้างตัวแปรเท่านั้น



```
main.js
1 let changeMe = true;
2 changeMe = false;
3 let x;
4
5 console.log(changeMe);
6 console.log(x);
```

false  
undefined

หากอ้างถึงตัวแปร let ที่อยู่ใน block scope นอก block scope โปรแกรมจะแสดง ReferenceError



```
main.js
1 let changeMe = true;
2 changeMe = false;
3
4 if (!changeMe) {
5   let myName = 'John';
6   console.log(myName);
7 }
8
9 console.log(myName);
```

John  
/home/ccuser/workspace/learn-javascript-variables-letV2/main.js:9  
console.log(myName);  
  
ReferenceError: myName is not defined  
at Object.<anonymous>  
(/home/ccuser/workspace/learn-javascript-variables-letV2/main.js:9:13)  
at Module.\_compile (module.js:571:32)

## const

- ตัวแปรที่ประกาศด้วยคีย์เวิร์ด const จะไม่สามารถ reassigned ค่าของตัวแปรได้
- ไม่สามารถประกาศและสร้างตัวแปร แต่ไม่กำหนดค่าตัวแปรได้ โปรแกรมจะแสดง SyntaxError

The terminal window shows a file named main.js with the following code:

```
1 const entree = 'Enchiladas';
2 console.log(entree);
3
4 entree = 'Tacos';
5 //const flowers;
```

When run, it outputs:

```
Enchiladas
/home/ccuser/workspace/learn-javascript-variables-constV2/main.js:4
entree = 'Tacos';
^

SyntaxError: Assignment to constant variable.
at Object.<anonymous>
```

ตัวแปร const เป็นตัวแปรที่ใช้ใน block scope หากอ้างถึงตัวแปร const ที่อยู่ใน block scope นอก block scope โปรแกรมจะแสดง ReferenceError

The terminal window shows a file named main.js with the following code:

```
1 const x = 10;
2 let y = 2;
3 if (true) {
4     const result = x/y;
5     console.log(result);
6 }
7
8 console.log(result);
```

When run, it outputs:

```
5
/home/ccuser/workspace/learn-javascript-variables-constV2/main.js:8
console.log(result);
^

ReferenceError: result is not defined
at Object.<anonymous>
(/home/ccuser/workspace/learn-javascript-variables-constV2/main.js:8:13)
```

The terminal window shows a file named main.js with the following code:

```
1 const flowers;
```

When run, it outputs:

```
/home/ccuser/workspace/learn-javascript-variables-constV2/main.js:1
(function (exports, require, module,
filename, dirname) { const flowers;
^^^^^
SyntaxError: Missing initializer in const declaration
```



JS

# Basic Javascript

# Function

เป็นบล็อกของโค้ดที่ถูกสร้างขึ้นเพื่อทำงานบางอย่าง เมื่อถูกเรียกใช้ ชิ้นสามารถรับค่าเข้าไปประมวลผลและส่งค่าออกมามาได้

```
function myFunction(name) {  
    return `Hello, ${name}!`;  
}  
  
myFunction("World");
```

**Parameter**

คือตัวแปรที่ใช้ในการรับค่าที่ส่งเข้ามายังฟังก์ชัน

**Argument**

คือค่าที่ส่งเข้ามายังฟังก์ชัน

# Method

## Array Method

- **map()** วนค่าทุกค่าใน array ตามพังก์ชันที่เราต้องการ และคืนค่าออกมาเป็น array ใหม่

```
const arr = [1, 2, 3];  
  
const new_arr = arr.map((item) => item + 1); // new_arr = [2, 3, 4]
```

- **filter()** วนค่าทุกค่าใน array เพื่อกรองเอาค่าตามเงื่อนไขที่กำหนด และคืนค่าออกมาเป็น array ใหม่

```
const arr = [1, 2, 3];  
  
const new_arr = arr.filter((item) => item > 2); // new_arr = [3]
```

- **sort()** ใช้ในการจัดเรียงลำดับรายการของ array โดยค่าเริ่มต้นจะเรียงจากน้อยไปมาก

```
const arr = [9, 2, 5];  
  
arr.sort(); // arr = [2, 5, 9]  
arr.sort((a, b) => b - a); // arr = [9, 5, 2]
```

- **forEach()** วนค่าทุกค่าใน array มาทำตามคำสั่งที่กำหนด แต่จะไม่คืนค่าออกมา

```
const arr = [1, 2, 3];  
  
arr.forEach((item) => console.log(item));  
// 1  
// 2  
// 3
```

- **concat()** ใช้ในการรวม array เข้าด้วยกันเป็น array ใหม่

```
const arr1 = ["a", "b"];
const arr2 = ["c", "d"];

const arr3 = arr1.concat(arr2); // arr3 = ["a", "b", "c", "d"]
```

- **includes()** ใช้ในการเช็คว่า array นั้นมีค่าที่ต้องการจะค้นหาอยู่หรือไม่ โดยจะคืนค่า  
ออกมานะ เป็น true หรือ false

```
const arr = [1, 2, 3, 4];

const result1 = arr.includes(2); // true
const result2 = arr.includes(9); // false
```

- **join()** เป็นการส่งค่า array กลับมาเป็น string คืนด้วยตัวค่านั้นที่กำหนด โดยตัวค่านั้นเริ่มต้นคือเครื่องหมาย , (comma)

```
const arr = ["a", "b", "c"];  
  
const result1 = arr.join(); // a,b,c  
const result2 = arr.join("-"); // a-b-c
```

- **reduce()** เป็นการนำค่าใน array มารวมกันให้เหลือเพียงค่าเดียว ก่อนที่จะคืนค่าออกมา

```
const arr = [1, 2, 3, 4, 5];  
  
const result = arr.reduce((acc, cur) => {  
  
    return acc + cur; // ค่ารวม + ค่าล่าสุดที่รับเข้ามา  
}, 0);  
  
// 15
```

- **find()** ใช้ในการค้นหาค่าตามเงื่อนไขที่กำหนด โดยคืนค่าเฉพาะข้อมูลแรกที่ตรงตามเงื่อนไขเท่านั้น

```
const arr = [1, 2, 3, 4, 5];  
  
const result = arr.find((item) => item > 2); // 3
```

- **findIndex()** การทำงานจะคล้ายกับ find() แต่จะคืนค่าเป็นเลข index ของ array ถ้าไม่เจอค่า จะคืนค่า -1 กลับมา

```
const arr = [1, 2, 3, 4, 5];  
  
const result1 = arr.findIndex((item) => item > 2); // 2  
const result2 = arr.findIndex((item) => item > 5); // -1
```

- **push()** เป็นเพิ่มค่าใหม่ที่ตำแหน่งสุดท้ายของ array

```
const arr = ["a", "b", "c"];  
  
arr.push("test"); // ["a", "b", "c", "test"];
```

- **slice()** เป็นการหั่นค่าจาก parameters ที่รับค่า index เริ่มต้น และ index สุดท้าย มาเป็น array ใหม่ แต่ index สุดท้ายใน array ที่เราเลือกจะไม่ถูกคืนค่าออกมาด้วย

```
const arr = [1, 2, 3, 4, 5];  
  
const result1 = arr.slice(2); // [3, 4, 5]  
const result2 = arr.slice(2, 4); // [3, 4]
```

- **splice()** เป็นการเปลี่ยนแปลงค่าใน array เดิม ซึ่งเราสามารถที่จะ เพิ่ม ลด หรือ แทนที่ค่า ใน array ได้

```
array.splice(start, deleteCount, item1, item2, ...)
```

```
const arr = ["a", "b", "c"];  
  
arr.splice(1, 1, "test"); // ["a", "test", "c"];  
  
arr.splice(2, 0, "hello"); // ["a", "test", "hello", "c"];  
  
arr.splice(0, 2); // ["hello", "c"];
```

# Number Method

- **toFixed()** กำหนดจำนวนทศนิยมในตัวเลข
- **toPrecision()** กำหนดจำนวนทศนิยมและจำนวน digit ในตัวเลข
- **toString()** แปลงตัวเลขเป็นสตริง
- **parseInt()** แปลงสตริงเป็นตัวเลขจำนวนเต็ม
- **parseFloat()** แปลงสตริงเป็นตัวเลขทศนิยม

# Math Method

- **Math.abs()** คืนค่าสมบูรณ์ของตัวเลข (Absolute value)
- **Math.ceil()** ปัดเลขขึ้นเป็นจำนวนเต็มที่ใกล้ที่สุด
- **Math.floor()** ปัดเลขลงเป็นจำนวนเต็มที่ใกล้ที่สุด
- **Math.round()** ปัดเลขให้เป็นจำนวนเต็มที่ใกล้ที่สุด
- **Math.max()** คืนค่าสูงสุดจากตัวเลขที่กำหนด
- **Math.min()** คืนค่าต่ำสุดจากตัวเลขที่กำหนด

# String Method

- **length** คืนค่าจำนวนตัวอักษร
- **charAt(index)** คืนค่าตัวอักษรที่ตำแหน่ง index ที่กำหนด
- **concat(str1, str2, ...)** รวมสตริงที่กำหนดเข้าด้วยกัน
- **toUpperCase()** แปลงสตริงเป็นตัวอักษรใหญ่ทั้งหมด
- **toLowerCase()** แปลงสตริงเป็นตัวอักษรเล็กทั้งหมด
- **trim()** ลบช่องว่างที่อยู่ด้านหน้าและด้านหลังของสตริง

- **startsWith(searchValue, startIndex)** ตรวจสอบว่าสตริงเริ่มต้นด้วยสตริงที่กำหนดหรือไม่
- **endsWith(searchValue, endIndex)** ตรวจสอบว่าสตริงลงท้ายด้วยสตริงที่กำหนดหรือไม่
- **substring(startIndex, endIndex)** คัดลอกส่วนของสตริงตามช่วง index ที่กำหนด
- **replace(searchValue, replaceValue)** แทนที่สตริงที่ค้นหาด้วยสตริงที่กำหนด
- **split(separator, limit)** แยกสตริงเป็น Array โดยใช้ตัวแยก (separator) และกำหนดจำนวนสูงสุดของสมาชิกใน Array (limit)

# Workshop

1. ให้เขียนฟังก์ชันเพื่อหาค่าสูงสุดใน Array [22, 19, 2, 89, 77] และแสดงผลลัพธ์ออกมา
2. ให้เขียนฟังก์ชันเพื่อเรียงลำดับตัวเลขเป็น Array จากค่าที่กำหนด  
ตัวอย่าง    กำหนดค่า 5  
ผลลัพธ์ -> [1, 2, 3, 4, 5]



# THANK YOU