JS JavaScript ES6 cheat sheet

borntoDev

ติดตามอัพเดท JavaScript Cheat Sheet เวอร์ชันใหม่ได้ที่นี่ >

Variable

การประกาศตัวแปร

```
var a          การประกาศตัวแปรปกติใน JS แต่
               มีปัญหาคือทะลุ block scope
let x          การประกาศตัวแปรใหม่ใน ES6 ที่
               เป็นตัวแปรที่ไม่ทะลุ block scope แล้ว
const PI       ประกาศตัวแปรแบบ ค่าคงที่
```

String

Backtick : เครื่องหมายขีดเฉียงที่ปุ่มตัวหนอน `

```
ประกาศ string หลายบรรทัดแบบไม่ต้องแยกตัวมาบวกกัน
let myString = `Hello
               from the
               other side`

จัดรูปแบบ string
let name = 'Pompam'
console.log(`Hello ${name}`)
```

includes() : เช็คว่ามีคำใน string หรือไม่

```
console.log('My name is Pompam'.includes('Pom'))
console.log('My name is Pompam'.includes('Noo')
```

startsWith() : เช็คว่า string ขึ้นต้นด้วยคำที่ต้องการหรือไม่

```
console.log('My name is Pompam'.startsWith('name'))
console.log('My name is Pompam'.startsWith('My'))
```

repeat() - รีเทิร์น string ซ้ำตามจำนวนที่กำหนด

```
console.log('My'.repeat(5))
```

Function argument

```
Default arguments สร้างค่าตั้งต้นให้กับ function
หากไม่ใส่ Argument ลงไปในการเรียก function ค่าของ
Argument ที่ใส่เข้าไปจะเป็นค่าที่เราประกาศไว้
```

```
function nameCat (name = 'George') {
return `My ${name} is so cute!`
}
console.log(nameCat())

Rest arguments
function countNum(...num) {
  return`Count: ${num.length}`
}
console.log(countNum(5,6,7,8,1,2,3,4))

Spread
console.log(countNum(...[5,6,7,8,1,2,3,4]))
//มีค่าเหมือน console.log(countNum(5,6,7,8,1,2,3,4))
```

Arrow function

แบบมาตรฐาน

```
แบบก่อนธรรมดา
function myFunc(a, b, c) {
  return a + b + c
}
console.log(myFunc(1, 2, 3))

แบบ Arrow
var myFunc = (a, b, c) => {
  return a + b + c
}
console.log(myFunc(1, 2, 3))
```

แบบมี Parameter ตัวเดียว

```
ถ้ามีตัวเดียวจะใส่หรือไม่ใส่ ( ) ครอบก็ได้
var myFunc = a => {
  return a * 100
}
console.log(myFunc(5))
```

แบบ Return ทันที

```
ถ้าไม่ใส่ { } จะ return ค่าที่ได้โดยอัตโนมัติ
var myFunc = (a, b, c) => a + b + c
console.log(myFunc(5, 6, 7))
```

Destructuring

การประกาศค่าผ่าน Array

```
const [name, age] = ['James', '3']
console.log(`name: ${name}`)
console.log(`age: ${age}`)
```

การประกาศค่าผ่าน Object (ชื่อ property ต้องตรงกับชื่อตัวแปร)

```
let {name, age} = {name:'James', age:'3'}
console.log(`name: ${name}`)
console.log(`age: ${age}`)
```

การวน Property ใน Object ใน Array

```
let cats = [{name:'James', age:'3'},
{name:'Sali', age:'1'}]
for (let {name, age} of cats) {
   console.log(`name: ${name}`)
   console.log(`age: ${age}`)
   console.log(`==============`)
}
//output
name: James
age: 3
==============
...
```

Promise

สร้าง Promise

```
let result = 'success'
// สร้าง Promise
let myTask = new Promise((resolve, reject) => {
   // โค้ดที่ทำงานแบบ Asynchronous
   if (result === 'success') {
      // กรณีทำงานเสร็จส่งผลลัพธ์ด้วย resolve()
      resolve('ok')
   }
   else {
      // กรณี error ส่งด้วย reject()
      reject('error')
  }
})
```

ใช้งาน Promise

```
myTask.then(() => {

  // ถ้า resolve (รันแล้วสามารถทำงานได้ตามเงื่อนไข)
  จะทำงานใน then
  console.log('success yeah!')

}).catch(() => {

  // ถ้า reject (รันแล้วไม่สามารถทำงานได้ตามเงื่อนไข)
  จะทำงานใน catch
  console.log('error T^T')

}).finally(() => {

  // finally จะทำงานหลังสุดทุกกรณี ไม่ว่าจะทำ then หรือ
  catch มาก่อนก็ตาม
  console.log('bye bye.')

})
```

Promise.all()

```
สร้าง Promise เพื่อจัดการ Promise หลายๆตัวมารวม
เป็นตัวเดียว

let myTask1 = new Promise((resolve, reject) => {...})
let myTask2 = new Promise((resolve, reject) => {...})
let myTask3 = new Promise((resolve, reject) => {...})
let myTask4 = new Promise((resolve, reject) => {...})
let myTask5 = new Promise((resolve, reject) => {...})
   .
   .
let myTask(n) = new Promise((resolve, reject) => {...})

Promise.all([myTask1, myTask2, myTask3, myTask4,
   myTask5, ..., myTask(n)]).then(() =>

  // Promise ทุกตัว resolve
  console.log('All task have been completed.')

).catch(() =>

  // มีตัวที่ reject
  console.log('Some task failed.')
)

**หากตัวใดตัว reject จะไม่เข้าเงื่อนไข then ทุกกรณี**
```

JS JavaScript ES6 cheat sheet (Easy print version)

borntoDev

ติดตามอัพเดท JavaScript Cheat Sheet เวอร์ชันใหม่ได้ที่นี่ >

Variable

การประกาศตัวแปร

```
var a          การประกาศตัวแปรปกติใน JS แต่
               มีปัญหาคือทะลุ block scope
let x          การประกาศตัวแปรใหม่ใน ES6 ที่
               เป็นตัวแปรที่ไม่ทะลุ block scope แล้ว
const PI       ประกาศตัวแปรแบบ ค่าคงที่
```

String

Backtick : เครื่องหมายขีดเฉียงที่ปุ่มตัวหนอน `

```
ประกาศ string หลายบรรทัดแบบไม่ต้องแยกตัวมาบวกกัน
let myString = `Hello
               from the
               other side`

จัดรูปแบบ string
let name = 'Pompam'
console.log(`Hello ${name}`)
```

includes() : เช็คว่ามีคำใน string หรือไม่

```
console.log('My name is Pompam'.includes('Pom'))
console.log('My name is Pompam'.includes('Noo')
```

startsWith() : เช็คว่า string ขึ้นต้นด้วยคำที่ต้องการหรือไม่

```
console.log('My name is Pompam'.startsWith('name'))
console.log('My name is Pompam'.startsWith('My'))
```

repeat() - รีเทิร์น string ซ้ำตามจำนวนที่กำหนด

```
console.log('My'.repeat(5))
```

Function argument

```
Default arguments สร้างค่าตั้งต้นให้กับ function
หากไม่ใส่ Argument ลงไปในการเรียก function ค่าของ
Argument ที่ใส่เข้าไปจะเป็นค่าที่เราประกาศไว้
```

```
function nameCat (name = 'George') {
return `My ${name} is so cute!`
}
console.log(nameCat())

Rest arguments
function countNum(...num) {
  return`Count: ${num.length}`
}
console.log(countNum(5,6,7,8,1,2,3,4))

Spread
console.log(countNum(...[5,6,7,8,1,2,3,4]))
//มีค่าเหมือน console.log(countNum(5,6,7,8,1,2,3,4))
```

Arrow function

แบบมาตรฐาน

```
แบบก่อนธรรมดา
function myFunc(a, b, c) {
  return a + b + c
}
console.log(myFunc(1, 2, 3))

แบบ Arrow
var myFunc = (a, b, c) => {
  return a + b + c
}
console.log(myFunc(1, 2, 3))
```

แบบมี Parameter ตัวเดียว

```
ถ้ามีตัวเดียวจะใส่หรือไม่ใส่ ( ) ครอบก็ได้
var myFunc = a => {
  return a * 100
}
console.log(myFunc(5))
```

แบบ Return ทันที

```
ถ้าไม่ใส่ { } จะ return ค่าที่ได้โดยอัตโนมัติ
var myFunc = (a, b, c) => a + b + c
console.log(myFunc(5, 6, 7))
```

Destructuring

การประกาศค่าผ่าน Array

```
const [name, age] = ['James', '3']
console.log(`name: ${name}`)
console.log(`age: ${age}`)
```

การประกาศค่าผ่าน Object (ชื่อ property ต้องตรงกับชื่อตัวแปร)

```
let {name, age} = {name:'James', age:'3'}
console.log(`name: ${name}`)
console.log(`age: ${age}`)
```

การวน Property ใน Object ใน Array

```
let cats = [{name:'James', age:'3'},
{name:'Sali', age:'1'}]
for (let {name, age} of cats) {
    console.log(`name: ${name}`)
    console.log(`age: ${age}`)
    console.log(`==============`)
}
//output
name: James
age: 3
==============
...
```

Promise

สร้าง Promise

```
let result = 'success'
// สร้าง Promise
let myTask = new Promise((resolve, reject) => {
    // โค้ดที่ทำงานแบบ Asynchronous
    if (result === 'success') {
        // กรณีทำงานเสร็จส่งผลลัพธ์ด้วย resolve()
        resolve('ok')
    }
    else {
        // กรณี error ส่งด้วย reject()
        reject('error')
  }
})
```

ใช้งาน Promise

```
myTask.then(() => {

  // ถ้า resolve (รันแล้วสามารถทำงานได้ตามเงื่อนไข)
  จะทำงานใน then
  console.log('success yeah!')

}).catch(() => {

  // ถ้า reject (รันแล้วไม่สามารถทำงานได้ตามเงื่อนไข)
  จะทำงานใน catch
  console.log('error T^T')

}).finally(() => {

  // finally จะทำงานหลังสุดทุกกรณี ไม่ว่าจะทำ then หรือ
  catch มาก่อนก็ตาม
  console.log('bye bye.')

})
```

Promise.all()

```
สร้าง Promise เพื่อจัดการ Promise หลายๆตัวมารวม
เป็นตัวเดียว

let myTask1 = new Promise((resolve, reject) => {...})
let myTask2 = new Promise((resolve, reject) => {...})
let myTask3 = new Promise((resolve, reject) => {...})
let myTask4 = new Promise((resolve, reject) => {...})
let myTask5 = new Promise((resolve, reject) => {...})
    .
    .
let myTask(n) = new Promise((resolve, reject) => {...})

Promise.all([myTask1, myTask2, myTask3, myTask4,
    myTask5, ..., myTask(n)]).then(() =>

  // Promise ทุกตัว resolve
  console.log('All task have been completed.')

).catch(() =>

  // มีตัวที่ reject
  console.log('Some task failed.')
)

**หากตัวใดตัว reject จะไม่เข้าเงื่อนไข then ทุกกรณี**
```