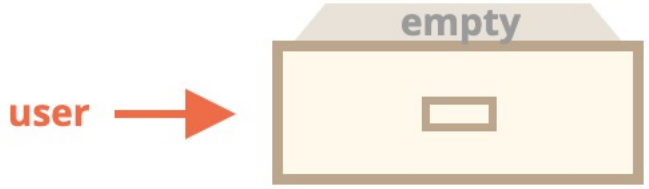
Advance Modern JavaScript

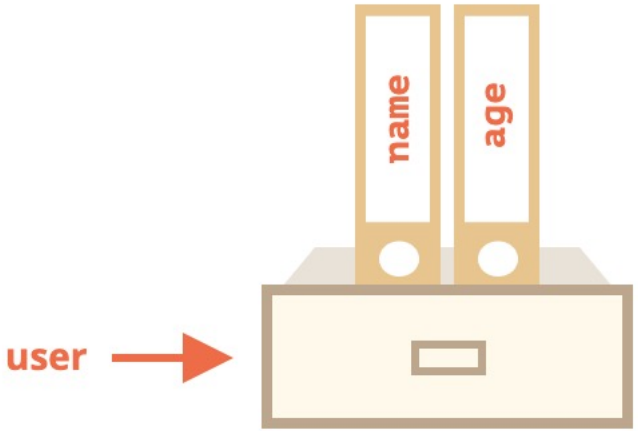
Objects

- JavaScript object คือชนิดตัวแปรประเภทหนึ่ง ซึ่งใช้ในการเก็บค่าในรูปแบบของ key-value
- Object สามารถสร้างได้ด้วยการใช้ Keyword “new” หรือเครื่องหมาย {}

```
> let user = new Object();
  let user = {};
```




```
1 let user = {
2     name: "Terry",
3     age: "30"
4 };
```

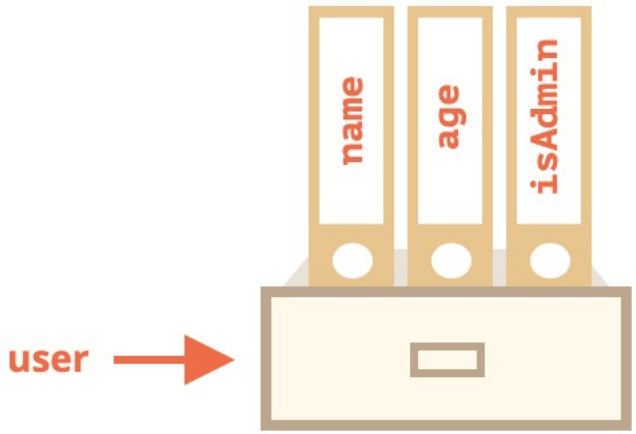


Objects (2)

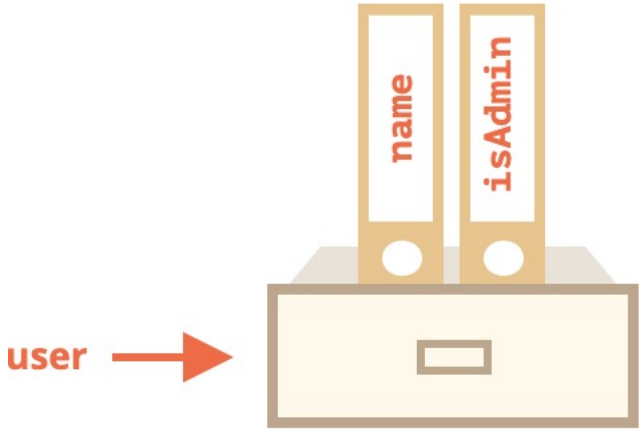
- เราสามารถเพิ่มหรือลบ attribute ได้

```
6 user.isAdmin = true;
```

```
6 user['isAdmin'] = true;
```




```
6 delete user.age;
```



Objects (3)

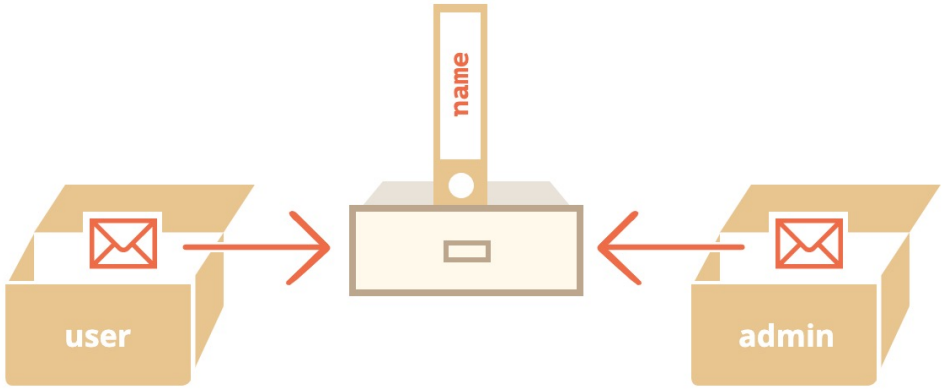
- เราสามารถ return object กลับไปจาก function ได้ทันทีโดยไม่ต้องประกาศตัวแปร
  - เรียกว่า Property value shorthand

```
1 function createUser(name, age) {
2     return {
3         name: name,
4         age: age
5     }
6 }
7
8 let user = createUser('John', 20);
9 alert(`Name ${user.name} with age ${user.age}`);
```

Object (4)

- การ Copy Object ทั้งหมดจะเป็นการเรียกแบบ by reference

```
1 let user = { name: "John" };
2 let admin = user;
3 admin.name = "Smith";
4 alert(user.name);
```



Object (5) – Const Object

- Object ที่ประกาศด้วย const สามารถแก้ไขค่าภายในได้ แต่ไม่สามารถ assign reference ใหม่ได้

```
1 const user = { name: 'John' };
2 user.name = 'Smith';
3 user.age = 30;
4 alert(`Name : ${user.name} Age : ${user.age}`);
```

```
1 const user = { name: 'John' };
2 user = { name: 'Smith', age: 30 };
3 alert(`Name : ${user.name} Age : ${user.age}`);
```

Object (6) – Object parameter list

- Parameter ของ Object สามารถวน loop เพื่อพิมพ์ออกมาได้ โดยการใช้ for...in

```
let user = {
    name: 'John',
    age: 30
}

for(param in user) {
    console.log(`Param : ${param} Value : ${user[param]}`);
}
```

- หรือใช้ Object.entries(<myObject>) จะได้ array ของ [key, value] ออกมา

```
let user = {name: 'John', age: 30}

console.log(Object.entries(user));
```

Object (7) – Object Method

- เราสามารถประกาศ method ไว้เป็น property หนึ่งของ Object ได้

```
let user = { name: "john" };
user.showMessage = function () {
    alert("Hi!");
}

user.showMessage();
```

```
let user = {
    name: "john",
    showMessage() {
        alert("Hi!")
    }
};

user.showMessage();
```

Short-hand function

Array

- Array คือกลุ่มของข้อมูล ซึ่งใน JavaScript สามารถสร้างได้ 2 วิธีคือ

```
1 let colors = new Array();
2 colors[0] = 'red';
3 colors[1] = 'blue';
4 colors.push('green');
5 alert(colors);
```

```
1 let colors = ["red", "blue", "green"];
2 colors.push('yellow');
3 alert(colors);
```

Array (2) - Looping

- การเข้า loop ข้อมูลภายใน array สามารถทำได้ 2 วิธีคือ

```
1 let colors = ["red", "blue", "green"];
2 colors.push('yellow');
3 for(let c of colors) {
4     console.log(c);
5 }
```

การใช้ for....of

```
1 let colors = ["red", "blue", "green"];
2 colors.push('yellow');
3 colors.forEach( c => {
4     console.log(c);
5 });
```

การใช้ forEach

ระวังสับสน for...of ใช้สำหรับการวน loop iterables object เช่น array, string
แต่ for...in ใช้สำหรับวน property ใน object

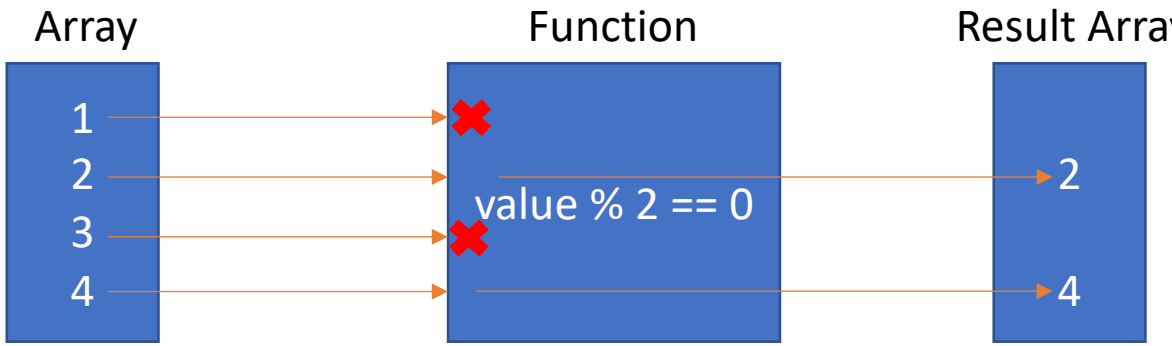
Exercise – Grader function

- ประกาศตัวแปรชนิด Array ที่เก็บคะแนนของนักเรียนไว้ เช่น 60, 82, 20, 98, 39
- เขียน Function ที่รับ Array ของคะแนน แล้ว return ค่าเป็น object ที่เก็บเกรดของนักเรียนเพื่อนับว่าแต่ละเกรดมีจำนวนกี่คน
  - เช่น เกรด A 10 คน, เกรด B 5 คน
  - การแบ่งเกรดจะแบ่งตามคะแนน โดยถ้าคะแนนมากกว่า 80 ได้ A, 70-79 ได้ B, 60-69 ได้ C, 50-59 ได้ D, น้อยกว่า 50 ได้ F

Array (3) – Built in functions

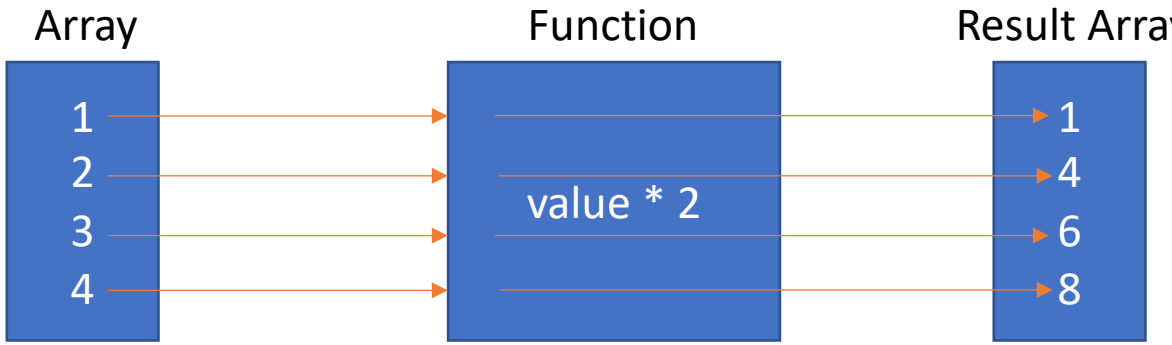
- length สำหรับนับจำนวนของ element ใน array
- concat(arr1, arr2) สำหรับเชื่อม array โดยจะ return array ตัวใหม่
- **filter**(function(value, index, arr)) สำหรับสร้าง array ใหม่กรณีผ่านการทดสอบ
- **map**(function(value, index, arr)) สร้าง array ใหม่จากผลลัพธ์ของการเรียก function นี้ในทุกๆ element ของ array
- **reduce**(function(total, currentValue, currentIndex, arr), initialValue) สำหรับนำค่าใน array มาเรียกใช้ function ไปเรื่อยๆ จนเหลือค่าสุดท้ายตัวเดียว

Filter



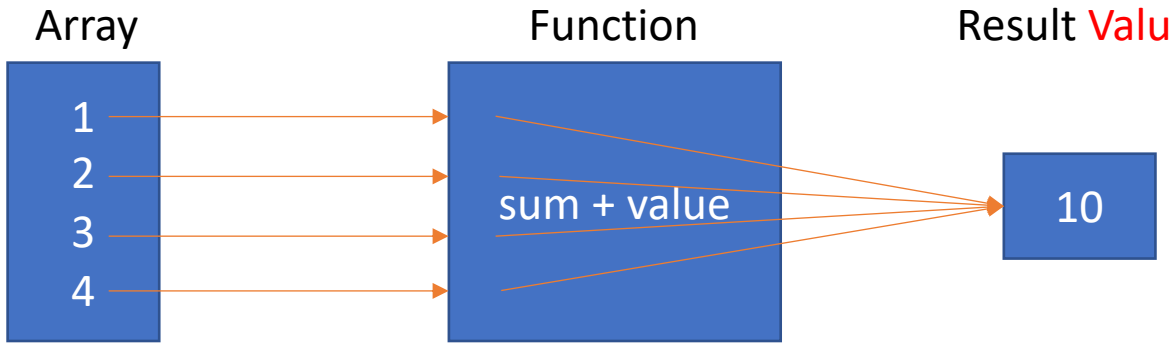

```
array.filter(value => value % 2 == 0)
```

Map




```
array.map(value => value *2)
```

Reduce




```
array.reduce((sum,value) => sum + value)
```

Exercise – Grader Average Caculator

- มี array ของ object student ประกอบไปด้วย name และ score
- students = [{"name" : "Alice", "score" : 63}, {"name" : "Bob", "score" : 80}]
- ให้ใช้ function reduce ในการหาค่าคะแนนทั้งหมด แล้วใช้ function length ในการหาค่าเฉลี่ยของคะแนน

Map

- Map คือตัวแปรที่สามารถเก็บค่าได้แบบ Key-Value
  - เหมือนกับ Object แต่ Map สามารถระบุ Key เป็นตัวแปรชนิดใด ๆ ก็ได้
- Function สำคัญของ Map คือ
  - new Map() => การสร้าง map ว่าง ๆ ขึ้นมา
  - map.set(key, value) => เก็บค่า key, value เข้าไปใน map
  - map.has(key) => return Boolean เพื่อบอกว่ามี key อยู่ใน map หรือไม่
  - map.delete(key) => ลบค่าโดยใช้ key
  - map.clear() => ลบค่าทั้งหมดใน map
  - map.size() => return integer จำนวน element ทั้งหมดใน map

Map (2) - Iteration

- การดึงข้อมูลออกจาก Map สามารถทำได้โดยคำสั่งต่อไปนี้
  - map.keys() => return list ของ key ทั้งหมดของ map
  - map.values() => return list ของ value ทั้งหมดใน map
  - map.entries() => return list ของ [key, value] สามารถใช้ได้กับ for....of

Destructuring

- Destructuring ใน JavaScript คือ syntax พิเศษสำหรับ ”unpack” array หรือ object ออกมาเป็นตัวแปรย่อย ๆ

```
let arr = ["Ed", "Sheeran"]
let [firstName, lastName] = arr;
console.log(firstName);
console.log(lastName);
```

Array Destructuring

```
let arr = ['Ed', 'Sheeran', 'singer', ]
let [firstName,,occupation] = arr;
console.log(firstName);
console.log(occupation);
```

สามารถใช้ comma ว่าง เพื่อข้าม parameter ที่ไม่ต้องการได้

```
let arr = ['Ed', 'Sheeran', 'singer', ]
let [firstName, ...rest] = arr;
console.log(firstName);
console.log(rest[0]);
console.log(rest[1]);
```

ใช้ ...rest ในการอ้างถึงค่าที่เหลือใน array ได้

Destructuring (2) - Object

- การทำ Destructuring สำหรับ object สามารถทำได้โดย
  - let {var1, var2} = Object

```
let student = {
    firstName: "John",
    lastName: "Smith",
    grade: 3.5
};
let {firstName, lastName, grade} = student;
console.log(`firstName : ${firstName} lastName : ${lastName} Grade : ${grade}`);
```

- สามารถกำหนดตัวแปรหลังจาก Destructuring ได้

```
let student = {
    firstName: "John",
    lastName: "Smith",
    grade: 3.5
};
let {firstName: f, lastName: l, grade: g} = student;
console.log(`firstName : ${f} lastName : ${l} Grade : ${g}`);
```

Destructuring (3) - Function

- สามารถใช้ Destructuring ช่วยในการประกาศ function ได้

```
function showMenu(title = "Untitled", width = 200, height = 100, items = []) {
    console.log(title);
    for( i of items) {
        console.log(i);
    }
}

showMenu('Test', undefined, undefined, ['a', 'b']);
```

```
function showMenu({title = "Untitled", width = 200, height = 100, items = []}) {
    console.log(title);
    for( i of items) {
        console.log(i);
    }
}

let params = {
    title: 'test',
    items: ['a', 'b']
}
showMenu(params);
```

JSON

- สามารถแปลง JavaScript Object เป็น Json String ได้ด้วยคำสั่ง
  - JSON.stringify(.....);
- สามารถแปลง String กลับมาเป็น JavaScript Object ด้วยคำสั่ง
  - JSON.parse(str, [reviver]);

```
let numbers = '["1", "2", "3]';
console.log(JSON.parse(numbers));
```

JSON (2) – Reviver function

- Reviver function ที่ประกาศหลัง function parse ใช้สำหรับการ parse ชนิดตัวแปรที่ต้องการเช่น date

```
let schedule = `{
  "meetups": [
    {"title":"Conference","date":"2017-11-30T12:00:00.000Z"},
    {"title":"Birthday","date":"2017-04-18T12:00:00.000Z"}
  ]
}`;

schedule = JSON.parse(schedule, (key, value) => {
  if (key == 'date') return new Date(value);
  return value;
});

console.log(schedule.meetups[0].date.getDate());
console.log(typeof(schedule.meetups[0].date));
```

Advanced Function - Rest

- Rest operator (...rest) ใช้ในการประกาศ function เพื่ออ้างอิงถึง parameter แบบไม่จำกัดจำนวนตัว

```
function showMsg(firstName, lastName, ...titles) {
    console.log(`First name : ${firstName} last name : ${lastName}`);
    for(t of titles) {
        console.log(`Title : ${t}`);
    }
}

showMsg('Ed', 'Sheeran', 'Singer', 'Actor');
```

Advanced Function - Spread

- Spread operator (...arr) ใช้ในทางตรงกันข้ามกับ Rest คือแปลง array ให้กลายเป็นตัวแปร (variable) ย่อย ๆ แต่ละตัว

Rest

```
function showMsg(firstName, lastName, ...titles) {
    console.log(`First name : ${firstName} last name : ${lastName}`);
    for(t of titles) {
        console.log(`Title : ${t}`);
    }
}

let currentTitles = ['Singer', 'Actor']
showMsg('Ed', 'Sheeran', ...currentTitles);
```

Spread

Exercise – Grader Dropdown

- มี array ของ object student ประกอบไปด้วย name และ score
- students = [
  {"id": 1, "name" : "Alice", "score" : 63},
  {"id" : 2, "name" : "Bob", "score" : 80}
  ]
- ในการแสดงผล DropDown Object จำเป็นจะต้องมี Key, Value และ Text
- จงใช้ function map ในการสร้าง array ของ data ใหม่ที่สามารถแสดงข้อมูล student ใน dropdown ได้

```
[
{id: 1, name: “Alice”, score: 63, key: 1, value: ”Alice”,
text: “Alice[63]”},
{id: 2, name: “Bob”, score: 80, key: 2, value: ”Bob”,
text: “Bob[80]”}
]
```

Exercise – ObjectMerge Function

- เขียน Function ที่รับ Object หลายๆตัว จากนั้นรวม attribute ของ object ทั้งหมด ให้กลายเป็น object ใหม่ และ return object ใหม่นั้นกลับมา

```
let obj1 = { name: ‘John’, age: 25 };
let obj2 = { hasCar: true }
let mergedObj = objectMerge(o1, o2) { …. };
// mergedObj = {name: ‘John’, age: 25, hasCar: true }
```

Promise

- พิจารณา Code ตัวอย่างการทำ Burger ต่อไปนี้

```
const getBeef = nextStep => {
    const product = 'Raw Angus';
    console.log(`Getting Beef.... got [${product}]`);
    nextStep(product);
}

const cookBeef = (beef, nextStep) => {
    const product = 'Grilled Angus';
    console.log(`Cooking Beef ${beef}.... got [${product}]`);
    nextStep(product);
}

const getBuns = nextStep => {
    const product = 'Sesame Buns';
    console.log(`Getting Buns.... got [${product}]`);
    nextStep(product);
}

const putBeefBetweenBuns = (buns, cookedBeef, nextStep) => {
    const product = 'Angus Sesame Bun Burger';
    console.log(`Putting ${cookedBeef} between ${buns}... got [${product}]`);
    nextStep(product);
}
```

Function Declaration

```
const makeBurger = doServe => {
    getBeef( beef => {
        cookBeef(beef, cookedBeef => {
            getBuns( buns => {
                putBeefBetweenBuns(buns, cookedBeef, burger => {
                    doServe(burger);
                });
            });
        });
    });
};

function serveBurger(burger) {
    console.log(`Serving.... ${burger}`);
}

makeBurger(serveBurger);
```

Calling Function

```
const getBeef = () => {
    const product = 'Raw Angus';
    console.log("Trying to get Beef.....")

    return new Promise( (resolve, reject) => {
        if(Math.random() >= 0.5) {
            console.log(`Got [${product}]`);
            resolve(product)
        }
        else {
            reject('No more beef!!');
        }
    });
}

const cookBeef = beef => {
    const product = 'Grilled Angus';
    console.log(`Cooking Beef ${beef}.... got [${product}]`);

    return new Promise( (resolve, reject) => {
        resolve(product);
    });
}

const getBuns = () => {
    const product = 'Sesame Buns';
    console.log(`Getting Buns.... got [${product}]`);

    return new Promise( (resolve, reject) => {
        resolve(product);
    });
}

const putBeefBetweenBuns = (buns, cookedBeef) => {
    const product = 'Angus Sesame Bun Burger';
    console.log(`Putting ${cookedBeef} between ${buns}... got [${product}]`);

    return new Promise( (resolve, reject) => {
        resolve(product);
    });
}
```

```
const makeBurger = doServe => {
    return getBeef()
    .then(cookBeef)
    .then(getBuns)
    .then(putBeefBetweenBuns);
};

function serveBurger(burger) {
    console.log(`Serving.... ${burger}`);
}

makeBurger()
.then(serveBurger)
.catch(err => console.log(`Error on making burger..... [${err}]`));
```

ตัวอย่างการใช้ Promise ในการแก้ไข CallBack Hell

Promise

- Promise คือ Object สำหรับจัดการ event ในการทำงานแบบ Asynchronous
  - Promise Object รับ parameter 2 ตัวคือ success function และ failure function
  - การใช้งาน Promise Object สามารถทำได้ผ่าน .then และ .catch
  - กรณี success function และ failure function มีการ return ค่า สามารถใช้เป็น .then property function เพื่อนำมาใช้งานได้ เช่น

```
1   function testPromise (isSuccess=true) {
2       return new Promise( (resolve, reject) => {
3           isSuccess ? resolve("Success") : reject("Failed");
4       })
5   }
6
7   testPromise()
8   .then( param => {
9       console.log(`Resolve Param : [${param}]`)
10  })
11  .catch( param => {
12      console.log(`Reject Param : [${param}]`)
13  });
```

Exercise – Microwave Function

- สร้างและเรียกใช้ Function ชื่อ Microwave รับ parameter เป็น ชื่ออาหาร, จำนวนเวลา โดย function นี้จะนับถอยหลัง แล้ว return ”Cooked <item>” ออกมาในรูปแบบ Promise (ถ้าไม่ใส่ parameter เวลา หรือใส่น้อยกว่า 1 จะ promise error)

```
microwave(“Meat”, 3);
Start cooking…
3
2
1
Done!
```

```
microwave(“Meat”, 0);
Promise Error...
```

```
microwave(“Meat”);
Promise Error...
```

ผลลัพธ์ที่ได้คือ “Cooked Meat”

```
let timerId = setInterval(function, millisecond);
clearInterval(timerId);
setTimeout(function, millisecond);
```

```
// Print Hello every 1 second
let timerId = setInterval( () => console.log(“Hello”) , 1000);

// Clear interval after 5 second
setTimeout( () => clearInterval(timerId), 5000 );
```

Async / Await

- เป็น Syntax พิเศษ สำหรับใช้งานกับ Promise เพื่อให้ใช้งานได้ง่ายขึ้น

```
const makeBurger = async (doServe) => {
    const beef = await getBeef();
    const cookedBeef = await cookBeef(beef);
    const buns = await getBuns();
    const burger = await putBeefBetweenBuns(buns, cookedBeef);
    doServe(burger);
};

function serveBurger(burger) {
    console.log(`Serving.... ${burger}`);
}

makeBurger()
.then(serveBurger)
.catch(err => console.log(`Error on making burger..... [${err}]`));
```

Modules (Export, Import)

- Modules คือไฟล์ JavaScript ที่แยกออกมาเป็นไฟล์ย่อย ๆ เพื่อให้ง่ายต่อการพัฒนาโปรแกรม
- ภายใน Modules จะประกอบไปด้วย class, function, หรือตัวแปร ก็ได้

microwave.js

```
const DEFAULT_TIME = 3;

function microwave(...) {
  //  ............
}
```

microwave.js

```
export const DEFAULT_TIME = 3;

export default function microwave(...) {
  //  ............
}
```

main.js

```
import { DEFAULT_TIME as TIME } from ‘./microwave.js’;
Import m from ‘./microwave.js’;
console.log(`Time : ${TIME}`);
m(‘Meat’, 3);
```