=> Objects in javascript are collection of key value pairs.
=> If a function is inside of an object then we will call them "Method".
=> In Javascript function are objects. For example
=> If a function is call as a Method of an Object "this" will return the object. Like person.walk();
=> If a function is called as standalone function then "this" will return global object that is window object in browser.

```
const walked = person.walk.bind(person);
walk();
```

=> Now when we call walked() function then walked() as a sandalone function, "this" will always return 'person' object.

ARROW FUNCTIONS

```
const square = function(number) {
  return number * number;
};
```

=> Equivalent to this function with Arrow function which is in ES6 is:

```
const square = number => number*number;
```

=> We can have no parenthesis for a single arguement but requires for multiple arguement.

```
const multiply = (num1, num2) => num1 * num2;
```

ARROW FUNCTIONS AND 'THIS':

When we use **this** in a normal function which is inside a class then **this** refers to the function itself not the class because in Javascript function itself is an object. So we cannot access variables inside class but outside this function by **this** keyword. To make **this** refer to it's class we need to bind **this,** so arrow function rebinds **this** and make it point to it's class. In this way we can access variable inside class outside this function by **this.**

```
const person = {
  talk() {
    setTimeout(function() {
      console.log("this", this);
    }, 1000);
  }
};

person.talk();
```

=> This will return window object in console like:

```
this Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, parent: Window, …}
```

=> This is returning window object because Timeout is standalone function in this case. But if we declare a variabe explicitly outside of the callback function for 'this' then it will return it's obect like:

```
const person = {
  talk() {
    var self = this;
    setTimeout(function() {
      console.log("this", self);
    }, 1000);
  }
};

person.talk();
```

=> But Arrow function don't rebinds 'this' keyword:

```
const person = {
  talk() {
    setTimeout(() => {
      console.log("this", this);
    }, 1000);
  }
};

person.talk();
```

ARRAY MAP in ES6:

```
const colors = ["red", "green", "blue"];
const items = colors.map(color => `<li>${color}</li>`); // `....` (BackTic Character) is template literals in ES6. Here we can define a template for our string. What we input inside ${} will be rendered dynamically.

console.log(items);
document.write(items);
```

OBJECT DESTRUCTURING

```
const address = {
  street: "Kadaghari",
  city: "Kathmandu",
  country: "Nepal"
};

const { street: st, city, country } = address;

console.log(st, city, country);
```

SPREAD OPERATOR

```
const first = [1, 2, 3];
const second = [4, 5, 6];
const third = [7, 8, 9];

const combined = [...first, 5, ...second, 8, ...third];

console.log(combined);
```

=> Using spread operator we could easily clone an array

```
const cloned = [...first];
console.log(cloned);
```

```
[1, 2, 3]
```

=> We could also apply spread operator in objects

```
const PersonName = { Name: "Surya Prasad Bhandari" };
const PersonAge = { Age: 26 };

const PersonInfo = { ...PersonName, ...PersonAge };

console.log(PersonInfo);
```

- We could also clone objects in Javascript

```
const clonedName = { ...PersonName };
console.log(clonedName);
```

Result:

```
{Name: "Surya Prasad Bhandari"}
    1.Name: "Surya Prasad Bhandari"
    2.__proto__: Object
```

CLASSES

```
class Person {
constructor(name) {
this.name = name;
}

walk() {
console.log(this.name, "can walk.");
}
}

const Surya = new Person("Surya Prasad Bhandari");
const Bran = new Person("Bran Stark");
Surya.walk();
Bran.walk();
```

Result:

Surya Prasad Bhandari can walk.

Bran Stark can walk.

INHERITANCE

```
class Person {
constructor(name) {
this.name = name;
}

walk() {
console.log(this.name, "can walk.");
}
}

class Teacher extends Person {
constructor(name, degree) {
super(name);
this.degree = degree;
}

teach() {
console.log(
this.name +
" can teach, obviously" +
" because he has an" +
this.degree +
" degree."
);
}
}

const mosh = new Teacher("Mosh Hamedami", "MSc");
mosh.walk();
mosh.teach();
```

Result:

Mosh Hamedami can walk.

Mosh Hamedami can teach, obviously because he has an MSc degree.

MODULES

- ➔ Instead of writing all code in one files we can write code in different files and these different files are call modules.
- ➔ Lets save Person class in person.js file and Teacher class in teacher.js file.
- ➔ Modules have class private so other class cannot access by default. In order to make them public we should prefix the class with *export.*

```
export class Teacher extends Person {
```

- To access this class from other files we have an import statement.

```
import { Person } from "./person"; // We dont write teacher.js. It's Javascript way
```

NAMED AND DEFAULT EXPORTS

- In *teacher.js* we will make class *Teacher* default export

```
export default class Teacher extends Person {
```

- Now in *index.js* we can call *Teacher* class as

```
import Teacher from "./teacher";
```

Default: *import ….. from "…."*

Named: *import {….} from "…."*

- Lets look at import from react modules:

```
import React, { component } from "react";
```

here "./react" is not used because we use "./" for our own modules which is part of our project but react is not part of our project but a third party library which is stored inside of the node modules folder.