Objects group together a set of variables and functions to create a model of something you would recognize from the real world.

Object properties + methods

In an object:

1. **Variables** become known as **properties**
2. **Functions** become known as **methods**

Properties tell us about the object, such as the height of a chair and how many chairs there are

Methods represent tasks that are associated with the object, e.g. count the height of all chairs by adding all heights together

Maps through objects

Every JavaScript object is a collection of property-value pairs. (We'll talk about this more later.)

Therefore you can define maps by creating Objects:

```
// Create an empty object
const thierAges = { };

const them = {
  'steve': 56,
  'mario': 30,
  'luigi': 91
};

console.log(them['mario']);
```

Maps through objects

string keys do not need quotes around them. Without the quotes, the keys are still of type string.

This is the same as the previous slide.

```
// Create an empty object
const thierAges = { };
```

```
const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

console.log(them['mario']);
```

Maps through objects

There are two ways to access the value of a property:

1. objectName[property]
2. objectName.property

(2 only works for string keys.)

```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

console.log(them['mario']);
console.log(them.mario);
```

Maps through objects

There are two ways to access the value of a property:

1. objectName[property]
2. objectName.property

(2 only works for string keys.)

Generally prefer style (2), unless the property is stored in a variable, or if the property is not a string.

```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

console.log(them['mario']);
console.log(them.mario);
```

Maps through objects

To add a property to an object, name the property and give it a value:

```
4
5   // Create an empty object
6   const thierAges = { };
7
8   const them = {
9     steve: 56,
0     mario: 30,
1     luigi: 91
2   };
3
4   them.mary = 42;
5
6   let newName = 'michelle';
7   them[newName] = 21;
8
9   console.log(them);
0
```

```
▶ {steve: 56, mario: 30, luigi: 91, mary: 42, michelle: 21}
```

Maps through objects

To remove a property to an object, use **delete**:

```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

them.mary = 42;

let newName = 'michelle';
them[newName] = 21;

delete them.mario;

console.log(them);
```

```
▶ {steve: 56, luigi: 91, mary: 42, michelle: 21}
```

Iterating through Map

Iterate through a map using a for...in loop (mdn): (intuition: for each key in the object) :

```
for (key in object) {
    // ... do something with object[key]
}
```

- You can't use for...in on lists; only on object types
- You can't use for...of on objects; only on list types

Iterating through Map

```
// Create an empty object
const thierAges = { };

const them = {
  steve: 56,
  mario: 30,
  luigi: 91
};

them.mary = 42;

let newName = 'michelle';
them[newName] = 21;

for (let name in them) {
console.log(name + ' is ' + them[name]);
}
```

```
steve is 56                    theSketch.js:30
mario is 30                    theSketch.js:30
luigi is 91                    theSketch.js:30
mary is 42                     theSketch.js:30
michelle is 21                 theSketch.js:30
>
```

- You can't use **for...in** on lists; only on **object types**
- You can't use **for...of** on objects; only on **list types**

Adding + Removing Classes

You can can control classes applied to an HTML element via **classList.add** and **classList.remove**:

```
const theImage = document.querySelector('img');

// Adds a CSS class called "active".
theImage.classList.add('active');
// Removes a CSS class called "hidden".
theImage.classList.remove('hidden');
```

finding the element twice...

```
function whatHappens() {
  const myImage = document.querySelector('img');
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Emoji_
  myImage.removeEventListener('click', whatHappens);
}

const myImage = document.querySelector('img');
myImage.addEventListener('click', whatHappens);
```

This repetition is inelegant.

finding the element twice...

```
function whatHappens() {
  const myImage = document.querySelector('img');
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Emoji_
  myImage.removeEventListener('click', whatHappens);
}

const myImage = document.querySelector('img');
myImage.addEventListener('click', whatHappens);
```

This repetition is inelegant.

Q: is there a way to fix?

```
function whatHappens(theEvent) {
  // const myImage = document.querySelector('img');
  const myImage = theEvent.currentTarget;
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb
  myImage.removeEventListener('click', whatHappens);
}

const myImage = document.querySelector('img');
myImage.addEventListener('click', whatHappens);
```

An **Event** element is passed to the listener as a parameter:

Event.currentTarget

An **Event** element is passed to the listener as a parameter:

```
function whatHappens(theEvent) {
  // const myImage = document.querySelector('img');
  const myImage = theEvent.currentTarget;
  myImage.src = 'https://upload.wikimedia.org/wikipedia/commons/thumb
  myImage.removeEventListener('click', whatHappens);
}

const myImage = document.querySelector('img');
myImage.addEventListener('click', whatHappens);
```

The event's **currentTarget** property is a reference to the object that we attached to the event, in this case the <img>'s **Element** to which we added the listener.

Not to be confused with Event.target

Note: Event has both:

theEvent.**target**:
the element that was clicked / "dispatched the event" (might be a child of the target)

theEvent.**currentTarget**:
the element that the original event handler was attached to)

Some properties of Element objects

Property	Description
id	The value of the id attribute of the element, as a string
innerHTML	The raw HTML between the starting and ending tags of an element, as a string
textContent	The text content of a node and its descendants. (This property is inherited from Node)
classList	An object containing the classes applied to the element