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

Properties & objects

```
var hotel = {
  // the following are key: value pairs.
    name: “Radisson”,
    rooms: 55,
    booked: 15,
    spa: false,
    roomTypes: [“single”, “twin”, “suite”],

    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}
```

Properties

Method

name, rooms, booked - are **keys** in the object
Object cannot have two keys with the same name
Value can be anything you want (string, number, boolean, function, another object)

Creating an object: literal notation

```
var hotel = {
  // the following are key: value pairs.
    name: "Radisson",
    rooms: 55,
    booked: 15,
    spa: false,
    roomTypes: ["single", "twin", "suite"],

    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}
```

Object is stored in a variable called hotel.

Each **key** is separated from its **value** by a colon :

Each property and method is separated by a comma, this keyword indicates that "rooms" and "booked" properties should be taken from this object

Accessing an object

There are two ways to access properties in an object 1) dot notation 2) square brackets

```
var hotel = {
  // the following are key: value pairs.
    name: "Radisson",
    rooms: 55,
    booked: 15,
    spa: false,
    roomTypes: ["single", "twin", "suite"],
    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}
```

Properties

Method

dot notation

```
  var hotelName = hotel.name;
  var roomsAvailable = hotel.checkAvailabilty( );
```

square brackets

```
  var hotelName = hotel["name"];
  var roomsAvailable = hotel["checkAvailability"]( );
```

Creating an object: constructor notation

```
var hotel = {
  // the following are key: value pairs.
    name: "Radisson",
    rooms: 55,
    booked: 15,
    spa: false,
    roomTypes: ["single", "twin", "suite"],

    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}
```

The new keyword creates a blank object
Then you can add properties and method using dot notation

Updating an object

```
var hotel = {
  // the following are key: value pairs.
    name: “Radisson”,
}

dot notation
    hotel.name = “Hilton”;

square brackets
    hotel[“name”] = “Hilton”;
```

Deleting property from an object

```
var hotel = {
  // the following are key: value pairs.
    name: “Radisson”,

}
```

dot notation

```
delete hotel.name;
```

Creating many objects: constructor notation

Often - you will want several objects to represent similar thing
You can use function as a template to create several objects
First we need to create template with object's properties and methods

```
var theHotel (theName, theRooms, theBooked) = {
  // the following are key: value pairs.
    this.name: theName,
    this.rooms: theRooms,
    this.booked: theBooked,

    checkAvailability: function( ) {
      return this.rooms - this.booked;
    }
}

var radissonHotel = new theHotel("Radisson", 55, 15);
var hiltonHotel = new theHotel("Hilton", 60, 60);
```

Properties

Method

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**

When do we need JavaScript Time

+ When we want to know what today (or any other day) is
+ When we want to execute something at a certain time
+ When we want to execute something repeatedly

For this we can use JavaScript Date object that has a lot of different built in methods and parameters.

All date functionality references January 1st, 1970, known as **Unix time**.

Methods you can use

+ **Date( );** - get current date
+ Date.now( ); - the number of milliseconds since January 1, 1970
+ var today = new Date(Date( )); - creates a new date object in a variable
+ today.getDate( ); - returns the day of the month (1-31)
+ today.getDay( ); - returns the day of the week (0-6, where Sunday - 0)
+ today.getFullYear( ); - returns the year
+ today.getHours ( ); - returns the hour (0-23)
+ today.getMillliseconds( ); - returns the milliseconds (0-999)
+ today.getMonth( ); - returns the month (0-10)
+ today.getSeconds( ); - returns the seconds (0-59)
+ today.getTime( ); - returns the number of milliseconds since midnight January 1, 1970, and a specified date

How to count elapsed time

```
// Get current time in milliseconds
        var theStart = Date.now( );

// Something happens for a long time…

// Get new current time in milliseconds
      var end = Date.now();
      var elapsed = end - start; // elapsed time in milliseconds
```

setTimeout, setInterval

We use setTimeout to execute function at a certain time (e.g. in 2 seconds)
We use setInterval when we want something to happen repeatedly

```
function alertMe( ) {
        alert(“It’s me!”);
}
setTimeout(alertMe, 2000);

function alertMe( ) {
        alert(“It’s me!”);
}

setInterval(alertMe, 2000);
```

Clearing timeout or interval

We assign setTimeout, setInterval to a variable
And use clearTimeout or clearInterval

```
function alertMe( ) {
        alert(“It’s me!”);
}
var alertIs = setInterval ( alertMe, 2000);

function stopAlert( ) {
        clearInterval ( alertIs );
}

setTimeout(stopAlert, 10000); // will stop after 5 alerting 5 times
```