# JavaScript – Classes and Objects

# Define a class

```
class Rectangle {

    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
}
```

# Methods

```
class Rectangle {
        constructor(height, width) {
        this.height = height;
        this.width = width;
   }

// Getter
get area() {
        return this.calcArea();
  }
// Method
calcArea() {
        return this.height * this.width;
  }
}
const square = new Rectangle(10, 10);
console.log(square.area); // 100
console.log(square.calcArea()); // 100
```

# Another example

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

# Extends

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

  speak() {
    console.log(this.name + " makes a noise.");
  }
}

class Dog extends Animal {
  constructor(name) {
    super(name); // call the super class constructor and pass in the name
parameter
  }

  speak() {
    console.log(this.name + " barks.");
  }
}

const d = new Dog('Mitzie');
d.speak(); // Mitzie barks.
```

# With getters and setters

```
class Person {
  constructor(name) {
    this.name = name;
  }
  get name() {
    return this.name;
  }
  set name(newName) {
    newName = newName.trim();
    if (newName === "") {
      throw "The name cannot be empty";
    }
    this._name = newName;
  }
}
```

# Private members

```
class Person {
  #_name = ""; // # to define private members
  constructor(name) {
    this.name = name;
  }
  get name() {
    return this.#_name;
  }
  set name(newName) {
    newName = newName.trim();
    if (newName === "") {
      throw "The name cannot be empty";
    }
    this.#_name = newName;
  }
}

let p = new Person("Ali");
console.log(p.name);  // Displays Ali
console.log(p._name); // Undefined (not an error!)
```

# As a constructor funtion

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}
```

# Object Creation and Modification

- Creation

```
var myObject = new Object();
```

- The new object has no properties
  - a blank object
- Properties can be added to an object, any time

```
var myCar = new Object();
myCar.make = "Ford";
myCar.model = "Focus";
```

- Properties can be accessed by dot notation or in array notation, as in

```
var property1 = myCar["model"];
delete myCar.model;
```

# As a constructor function

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

const person1 = new Person("Peter", “John", 20, "black");
const person2 = new Person("Sally", "Rally", 48,
"green");

Person.nationality = "Canada"; // Does not work
person1.nationality = "Canada"; // Adds it to person1
only

// You can use the prototype property, see later
```

# Object Creation and Modification (2)

- An Abbreviated way

```
var myCar = {make:"ford", model: "Contour SVT"};
```

- also called the JSON way

# Visit the properties in an object

- Is the property in an object?

```
var myCar = {make:"ford", model: "Contour SVT"};
const x = 'make' in myCar;
```

- Traverse all the properties

```
for (var prop in myCar)
  document.write(myCar[prop] + "<br />");
```

# instanceof

- Is object instanceof?

```
var myCar = {make:"ford", model: "Contour SVT"};
myCar instanceof myCar;
myCar instanceof Object;
myCar instanceof Date;
```

false

# Another example

```
function Plane(newMake, newModel, newYear){
   this.make = newMake;
   this.model = newModel;
   this.year = newYear;
 }

 myPlane = new Plane("Cessna",
                     "Centurnian",
                     "1970");
```

# Another example...

```
function Plane(newMake, newModel, newYear){
   this.make = newMake;
   this.model = newModel;
   this.year = newYear;
 }

 myPlane = new Plane("Cessna",
                     "Centurnian",
                     "1970");

// We cannot add
Plane.owner = "John";
```

# A function to display the properties

```
function displayPlane() {
   document.write("Make: ", this.make,
                  "<br />");
   document.write("Model: ", this.model,
                  "<br />");
   document.write("Year: ", this.year,
                  "<br />");
 }
```

- How to call it?

```
this.display = displayPlane; // Add to function Plane
…
var myPlane = new Plane("Cessna","Centurnian", "1970");
myPlane.display();
```

# Object.prototype

- Use it to change the template of the object
- It affects all the objects of the same type
- You can put the functions into the prototype
- Use prototype to build longer chain of inheritance

# Object.prototype (2)

```
function Person(first, last, age, eyeColor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyeColor;
}

Person.prototype.nationality = "Canada";
```

# __proto__

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

name() {
    return this.name;
  }
}

const a = new A("Montreal");

const b = new Object();
b.__proto__ = a;

console.log(b.name);
```

# Class expression - unnamed

```
let Rectangle = class {
   constructor(height, width) {
   this.height = height;
   this.width = width;
   }
};
```