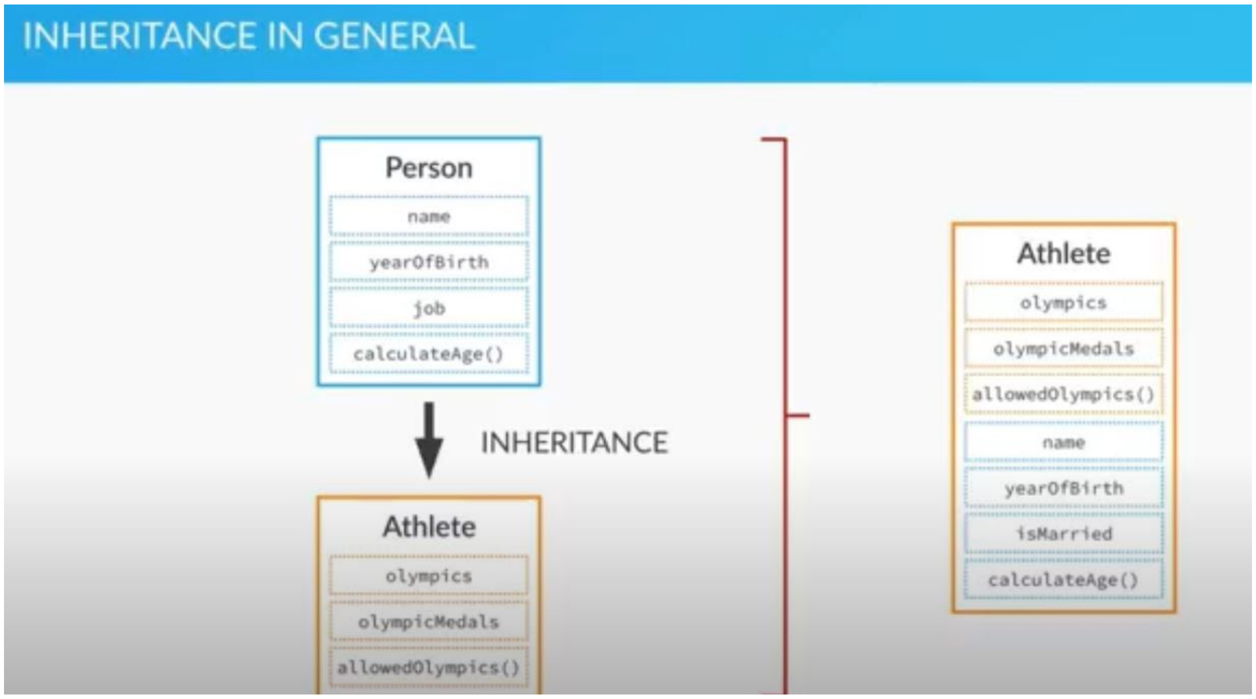
JavaScript

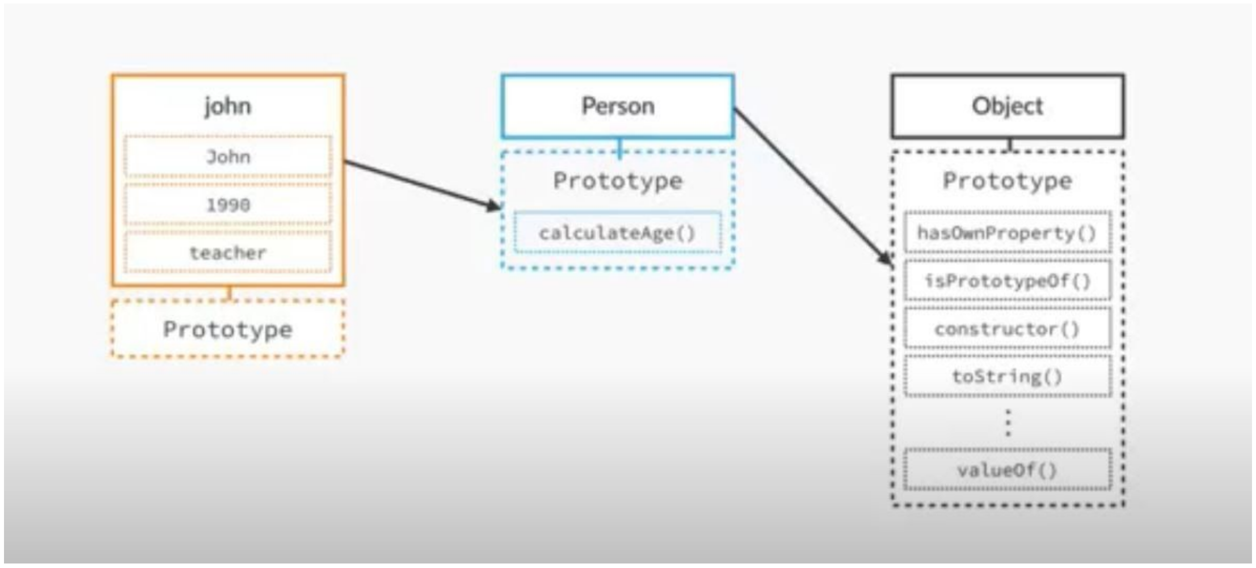
Part-5

1. OOP’s (contd.)
   Inheritance: When one object is able to inherit another object’s properties and methods.
   Ex:



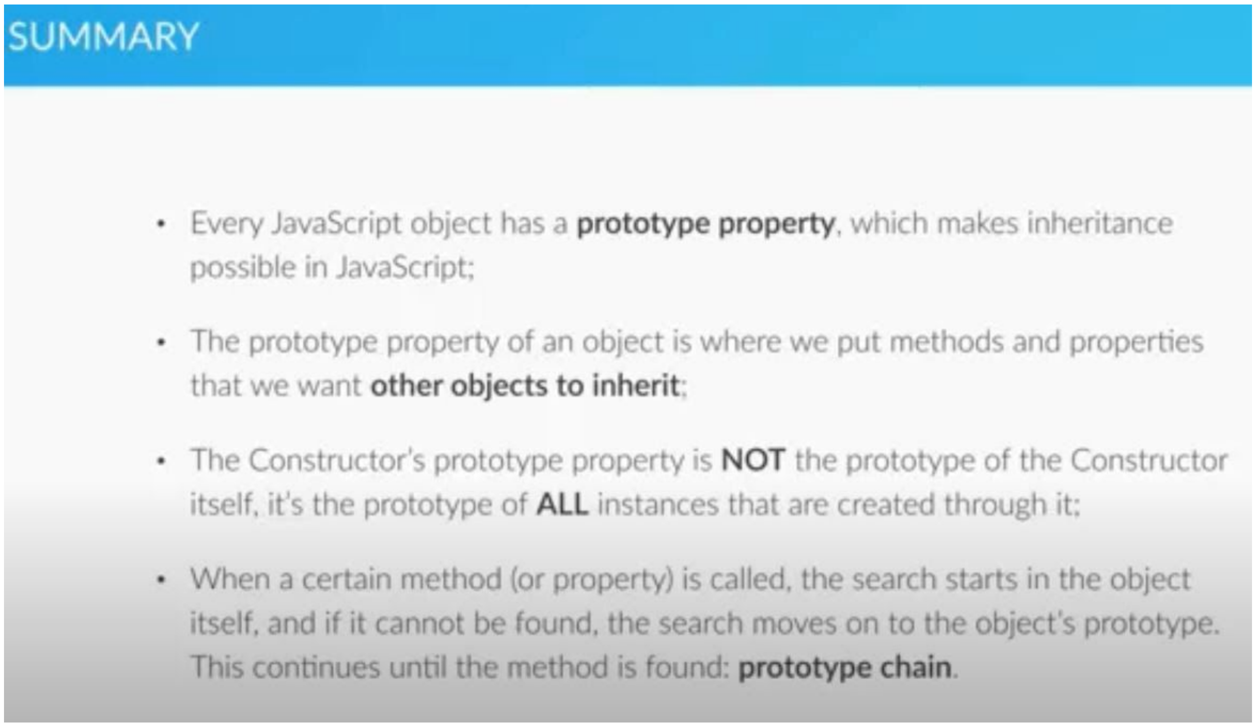
   Inheritance works by using prototype property. Each object has a prototype property which makes inheritance possible in JS.

Prototype chain:



Explanation to prototype chain: (john is assigned values using the constructor Person)
'Person' object has calculateAge() function in its prototype and this can be accessed by object 'john'. 'Person' object (constructor) is the part of a bigger constructor k/as the object-object constructor. Every object that we ever create is a part of this object-object constructor and has several predefined methods in it. And this inheritance of properties and methods from the higher object in the hierarchy is k/as

prototype chain.



SUMMARY

- Every JavaScript object has a **prototype property**, which makes inheritance possible in JavaScript;
- The prototype property of an object is where we put methods and properties that we want **other objects to inherit**;
- The Constructor's prototype property is **NOT** the prototype of the Constructor itself, it's the prototype of **ALL** instances that are created through it;
- When a certain method (or property) is called, the search starts in the object itself, and if it cannot be found, the search moves on to the object's prototype. This continues until the method is found: **prototype chain**.

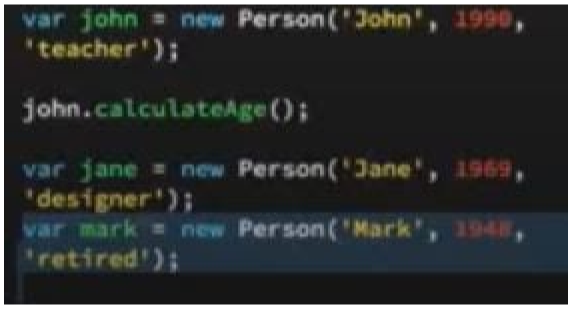
2. Function constructor:
   Used to make a blueprint(constructor) which can be used to make objects with same names and different values.
   By convention, the name of a function constructor must start with a capital letter.
   Declaration:
   var name_of_constructor = function(parameters_for_actual_object){};
   Ex:

```
var Person = function(name, yearOfBirth, job){
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.job = job;
        this.calculateAge = function(){
                console.log(2020 - this.yearOfBirth);
        }
}
```

```
var john = new Person('john', 1990, 'driver');
```

Explanation:
When we use the new function a new object is created and the function is called with parameters passed on. As we know in a general function the 'this' object points to the global execution context but in this case it refers to the object which it is part of and this leads to passing of all the parameters to the new created object.

Creating different objects:



```
var john = new Person('John', 1990,
'teacher');

john.calculateAge();

var jane = new Person('Jane', 1969,
'designer');
var mark = new Person('Mark', 1948,
'retired');
```

We have not yet added the function to the prototype. Let's do it.

```
var Person = function(name, yearOfBirth, job){
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.job = job;
        }
}

// using prototype property to use in inheritance
Person.prototype.calculateAge = function(){
                console.log(2020 - this.yearOfBirth);
        }

var john = new Person('john', 1990, 'driver');
```

Now after this declaration every object that has Person as its parent can use inheritance whenever required. Like: john.calculateAge();

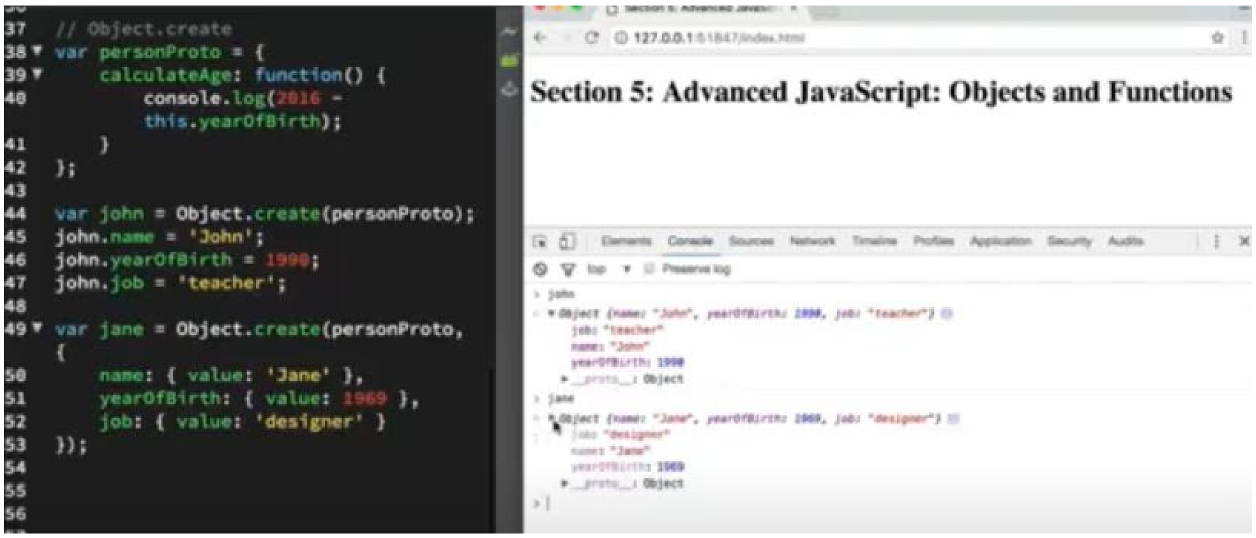
- To verify is an object has certain property or not:
  Ex:

In the previous example, ‘john’ had ‘job’ as one of the properties. So:
john.hasOwnProperty(‘job’) is evaluated to true.
However, if john inherits some property through the scope chain then on using ‘hasOwnProperty’ will evaluate to false.

- ‘instanceof’ method:
  We made use of the Person constructor to assign john as an object. So, john is an instanceof Person.
  John instanceof Person
  >> True

  Inspecting arrays:
  var x = [2,3,5];
  console.info(x);
  //gives info of the array like position of every element, length and a __proto__ (inside the prototype property all the functions)
  x.length
  >>3

3. Using object.create method for generating objects:

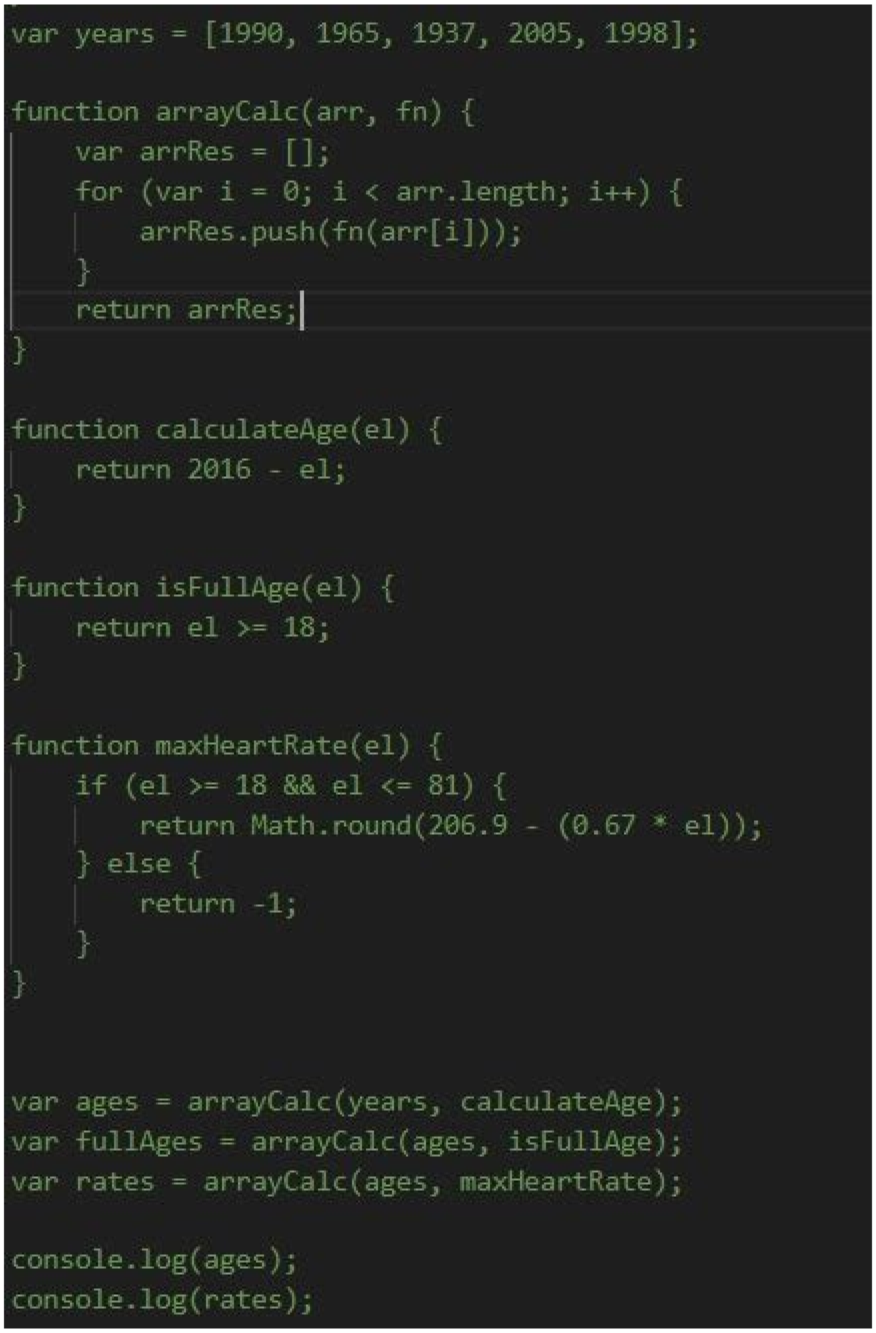


```
// Object.create
var personProto = {
    calculateAge: function() {
        console.log(2016 -
        this.yearOfBirth);
    }
};

var john = Object.create(personProto);
john.name = 'John';
john.yearOfBirth = 1990;
john.job = 'teacher';

var jane = Object.create(personProto,
{
    name: { value: 'Jane' },
    yearOfBirth: { value: 1969 },
    job: { value: 'designer' }
});
```

Section 5: Advanced JavaScript: Objects and Functions

```
> john
  ▼Object {name: "John", yearOfBirth: 1990, job: "teacher"}
      job: "teacher"
      name: "John"
      yearOfBirth: 1990
    ▶__proto__: Object
> jane
  ▼Object {name: "Jane", yearOfBirth: 1969, job: "designer"}
      job: "designer"
      name: "Jane"
      yearOfBirth: 1969
    ▶__proto__: Object
>
```

Passing function as arguments:



```
var years = [1990, 1965, 1937, 2005, 1998];

function arrayCalc(arr, fn) {
    var arrRes = [];
    for (var i = 0; i < arr.length; i++) {
        arrRes.push(fn(arr[i]));
    }
    return arrRes;
}

function calculateAge(el) {
    return 2016 - el;
}

function isFullAge(el) {
    return el >= 18;
}

function maxHeartRate(el) {
    if (el >= 18 && el <= 81) {
        return Math.round(206.9 - (0.67 * el));
    } else {
        return -1;
    }
}


var ages = arrayCalc(years, calculateAge);
var fullAges = arrayCalc(ages, isFullAge);
var rates = arrayCalc(ages, maxHeartRate);

console.log(ages);
console.log(rates);
```

4. First class functions:
   Passing one function into another.

   Eg:

```
function interviewQuestion(job) {
    if (job === 'designer') {
        return function(name) {
            console.log(name + ', can you please explain what UX
design is?');
        }
    } else if (job === 'teacher') {
        return function(name) {
            console.log('What subject do you teach, ' + name + '?');
        }
    } else {
        return function(name) {
            console.log('Hello ' + name + ', what do you do?');
        }
    }
}

var teacherQuestion = interviewQuestion('teacher');
var designerQuestion = interviewQuestion('designer');



teacherQuestion('John');
designerQuestion('John');
designerQuestion('jane');
designerQuestion('Mark');
designerQuestion('Mike');

interviewQuestion('teacher')('Mark');
```

```
// This is a valid approach because the above expression is
evaluated from left to right so first: interviewQuestion('teacher')
is evaluated
```

5. Immediately invoked function expressions(IIFE):
   - How does an IIFE look like:

     Eg:
     ```
     (function (goodLuck) {
         var score = Math.random() * 10;
         console.log(score >= 5 - goodLuck);
     })(5);
     ```

     (function (function_name) {
     //code
     }) (value is passed);

6. Closures:

```
function retirement(retirementAge) {
    var a = ' years left until retirement.';
    return function(yearOfBirth) {
        var age = 2016 - yearOfBirth;
        console.log((retirementAge - age) + a);
    }
}

var retirementUS = retirement(66);
var retirementGermany = retirement(65);
var retirementIceland = retirement(67);

retirementGermany(1990);
retirementUS(1990);
retirementIceland(1990);

//instead of individually declaring both the parameters one by one
we can declare simultaneously
retirement(66)(1990);
```

7. Bind, call and apply:
   Borrow and call:

```
var john = {
    name: 'John',
    age: 26,
    job: 'teacher',
    presentation: function(style, timeOfDay) {
        if (style === 'formal') {
            console.log('Good ' + timeOfDay + ', Ladies and
gentlemen! I\'m ' +  this.name + ', I\'m a ' + this.job + ' and I\'m
' + this.age + ' years old.');
        } else if (style === 'friendly') {
            console.log('Hey! What\'s up? I\'m ' +  this.name + ',
I\'m a ' + this.job + ' and I\'m ' + this.age + ' years old. Have a
nice ' + timeOfDay + '.');
        }
    }
};

var emily = {
    name: 'Emily',
    age: 35,
    job: 'designer'
};

john.presentation('formal', 'morning');

john.presentation.call(emily, 'friendly', 'afternoon');

//john.presentation.apply(emily, ['friendly', 'afternoon']);
var johnFriendly = john.presentation.bind(john, 'friendly');
johnFriendly('morning');
johnFriendly('night');
var emilyFormal = john.presentation.bind(emily, 'formal');
emilyFormal('afternoon');
```

```
// Another cool example
var years = [1990, 1965, 1937, 2005, 1998];

function arrayCalc(arr, fn) {
    var arrRes = [];
    for (var i = 0; i < arr.length; i++) {
        arrRes.push(fn(arr[i]));
    }
    return arrRes;
}

function calculateAge(el) {
    return 2016 - el;
}

function isFullAge(limit, el) {
    return el >= limit;
}

var ages = arrayCalc(years, calculateAge);
var fullJapan = arrayCalc(ages, isFullAge.bind(this, 20));
console.log(ages);
console.log(fullJapan);
*/
```