Name : Mayur Purushvani

Objects In javaScript :

Everything is an object.

We have 2 values I javaScripts :

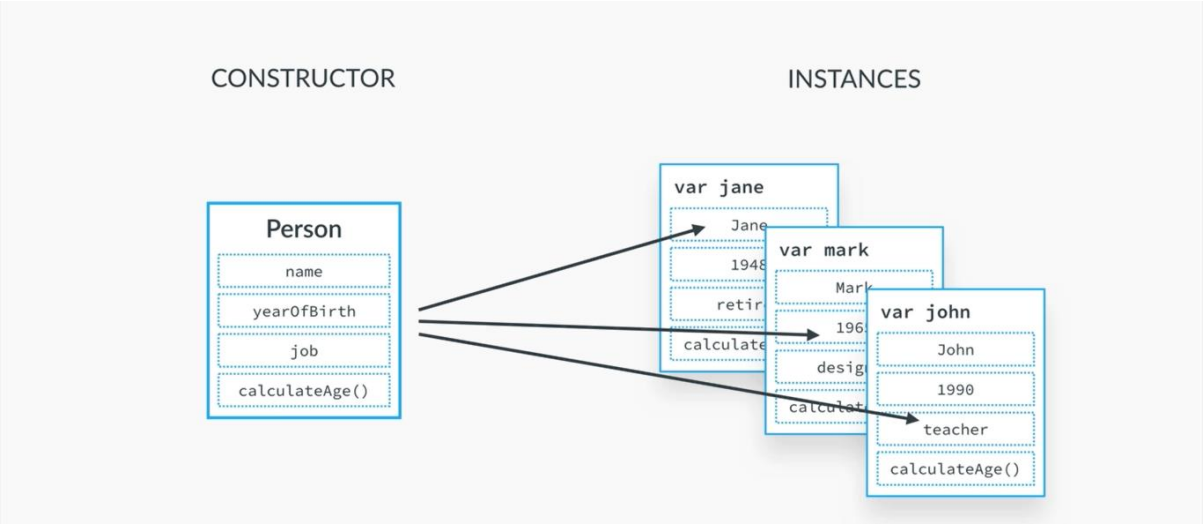
- Primitive : Numbers, String, Boolean, Undefined, Null, etc.
- Everything else : Array, Function, Objects, Dates, etc. (Is an objects).

Object-Oriented Programming :

- Objects interacting with one another through methods and properties.
- Used to store data, structure applications into modules and keeping code clean.

Constructor and instances in JS :

The constructor prototype property is not the prototype of the constructor itself, it's the prototype of all instances that are created through it.

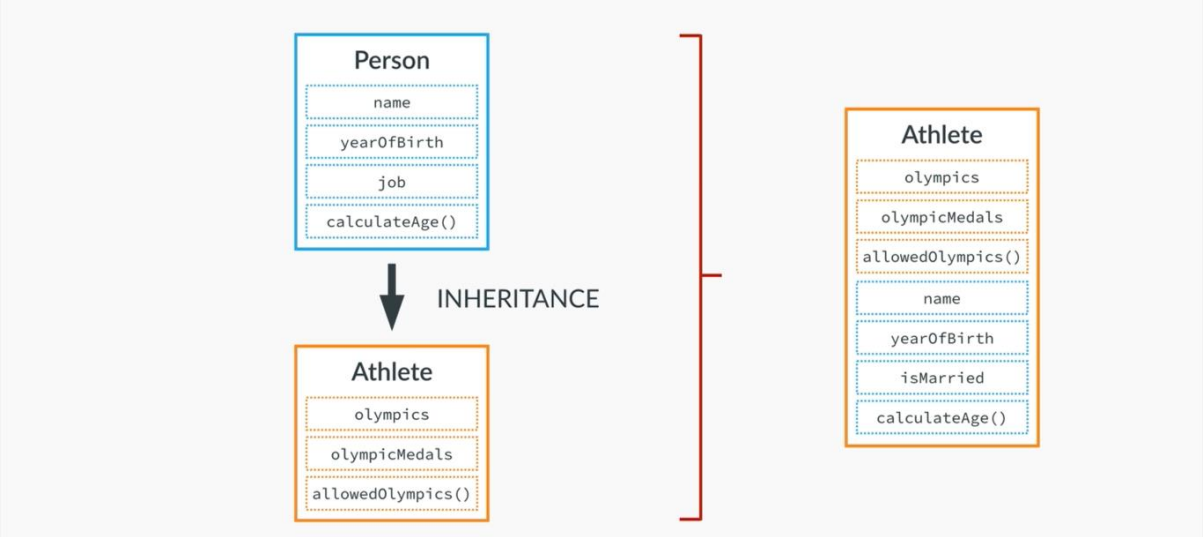


Inheritance :

It has a parent child relationship.

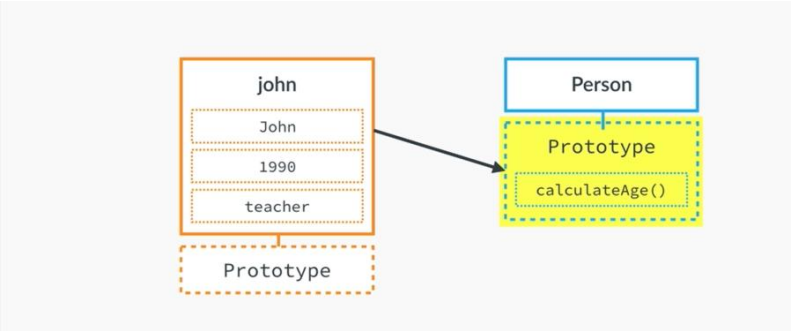
It is like a objects can access another objects methods and properties.

Look at the below example :



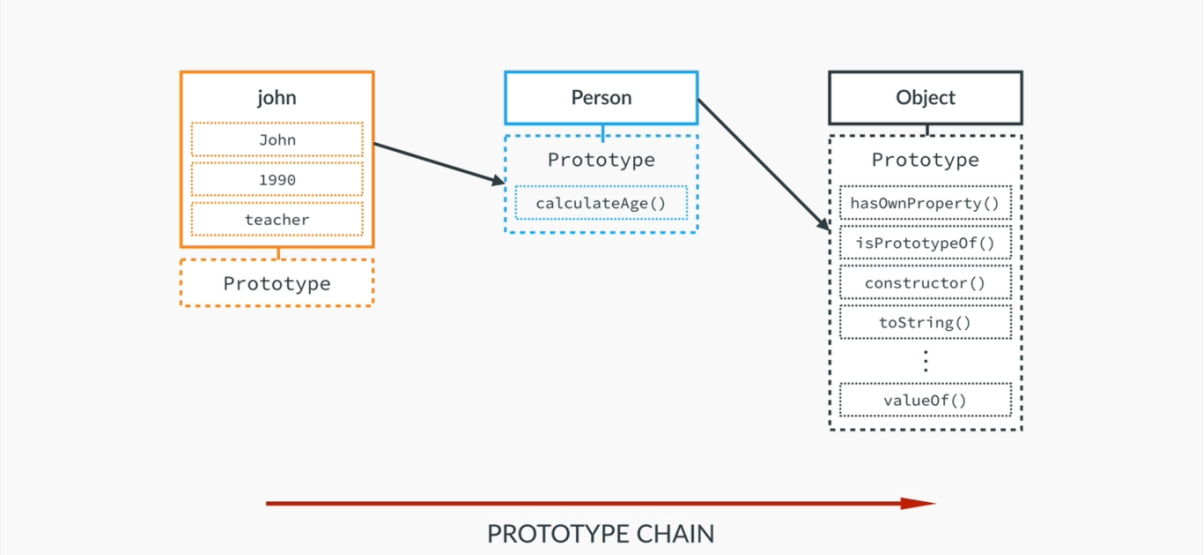
Prototypes :

- Every javascript objects have a prototype. Which makes inheritance possible in JS.
- Let's look at the below example :
- There are one person class and one jhon class, so, jhon can access all the prototypes methods of person class.



Prototype chain :

- The prototype property of a object is where we put methods and properties that we want other objects to inherits.
- It means the person class have some in-Built objects prototypes. So jhon can also access objects prototypes.
- When we call certain methods, the search starts in the objects itself. Ad if it cannot be found, the search moves on the objects prototype. This continues until the method is not found.



Function Constructor :

```
var mayur = {
    name : "mayur",
    yearOfBirth : 1999,
    job : "software engineer"
};
var person = function(name,yearOfBirth,job){
    this.name = name,
    this.yearOfBirth = yearOfBirth,
    this.job = job
    /* Simple calculateAge() method in person class! */

    /*this.calculateAge = function()
    {
        console.log(2020 - yearOfBirth);
    }*/

    person.prototype.lastName = 'purushvani';

};

/* prototype person class */
person.prototype.calculateAge = function()
{
    console.log(2020 - this.yearOfBirth);
}

var mayur = new person('mayur',1999,'software engineer');
var mike = new person('mike',1998,'designer');
var jhon = new person('jhon',1997,'designer');

mayur.calculateAge();
mike.calculateAge();
jhon.calculateAge();

console.log(jhon.lastName);
console.log(mike.lastName);
console.log(jhon.lastName);
```

Here, we have an example of prototype methods. We've a 4 different class mayur,mike ,person and jhon. And every class can access the method calculateAge() of person class.

So, I've declared one prototype method in person class and I just called that on every other method.

Here, We have to confirm that is there any prototype chain exists or not, So what we can do..!

We just simply type any object name into console. And then check it is exists or not. Then you have a detailed information about that objects. Like prototypes, objects etc. expand that things and just call the in-built methods of that objects like : mayur.hasOwnProperty('job');

You can see that if that objects have their own property, it will return true, and that object have a prototype, then that returns a false value.

Create Object with object.create() :

```
var personProto = {
    calculateAge : function (){
        console.log(2020 - this.yearOfBirth);
    }

};

var mayur =  Object.create(personProto);
mayur.name = 'mayur';
mayur.yearOfBirth = 1999;
mayur.job = 'software engineer';

var jhon = Object.create(personProto, {
    name : { value : 'jhon'},
    yearOfBirth : {value : 1998 },
    job : {value : 'designer'}
});
```

We can also create an object like I've created 'jhon'.

Primitives vs Objects :

We already know that, number, Boolean, string are primitive values and all others are objects.

Let's take an example :

```
//primitives
var a = 23;
var b = a;
a = 46;
console.log(a);
console.log(b);

//Objects
var obj = {
    name : 'mayur',
    age : 22
};

var obj2 = obj;
obj.age = 30;
console.log(obj.age);
console.log(obj2.age);

//Functions
var age = 28;
var obj3 = {
    name : 'mike',
    age : 30
};
function change(a, b) {
    a = 30;
    b.city = 'rajkot';
}

change(age,obj3);
console.log(age);
console.log(obj3.city);
```

Here we've the basic difference between objects, functions, and primitives.

First class functions passing functions as arguments :

Functions are also objects in js :

- A function is an instance of an objects.
- A function behave like any other objects.
- We can stores functions in a variables.
- We can pass a function as an argument to another function.
- We can return a function from a function.

Let's look at the example :

```
var years = [1999,1972,1937,2005,1975];

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 2020 - el;
}
function isFullAge(el)
{
    return el >= 18;
}

function maxHartBeats (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,maxHartBeats);

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

In above example, I've just created on generic function called arraycalc().

This functions have 2 arguments which have other function passes. This concept is known as first function passes the argument to other function.

Like, I've created other 3 functions named : calculateAge(), isFullAge(), MaxHeartBeats().

So I've just simply pass that argument to my generic function and print on console.