

Aula 2: OOJS

Orientação a Objetos

Object Oriented JavaScript

Aula 2 | Etapa 1:

Protótipos

Orientação a Objetos



Protótipos



Todos os objetos Javascript herdam propriedades e métodos de um prototype.
O objeto Object.prototype está no topo desta cadeia.

```
> const objeto = {}  
< undefined  
  
> objeto  
< {}  
  ▾ __proto__:  
    ▶ constructor: f Object()  
    ▶ hasOwnProperty: f hasOwnProperty()  
    ▶ isPrototypeOf: f isPrototypeOf()  
    ▶ propertyIsEnumerable: f propertyIsEnumerable()  
    ▶ toLocaleString: f toLocaleString()  
    ▶ toString: f toString()  
    ▶ valueOf: f valueOf()  
    ▶ __defineGetter__: f __defineGetter__()  
    ▶ __defineSetter__: f __defineSetter__()  
    ▶ __lookupGetter__: f __lookupGetter__()  
    ▶ __lookupSetter__: f __lookupSetter__()  
    ▶ get __proto__: f __proto__()  
    ▶ set __proto__: f __proto__()
```

```
> array  
< []  
  ▾ __proto__: Array(0)  
    ▶ concat: f concat()  
    ▶ constructor: f Array()  
    ▶ copyWithin: f copyWithin()  
    ▶ entries: f entries()  
    ▶ every: f every()  
    ▶ fill: f fill()  
    ▶ filter: f filter()  
    ▶ find: f find()  
    ▶ findIndex: f findIndex()  
    ▶ flat: f flat()  
    ▶ flatMap: f flatMap()  
    ▶ forEach: f forEach()  
    ▶ includes: f includes()  
    ▶ indexOf: f indexOf()  
    ▶ join: f join()  
    ▶ keys: f keys()  
    ▶ lastIndexOf: f lastIndexOf()  
    length: 0
```

Os protótipos do Javascript são o esqueleto dos objetos, então todos os objetos do javascript vão herdar objeto do prototype.

Exemplo: objeto

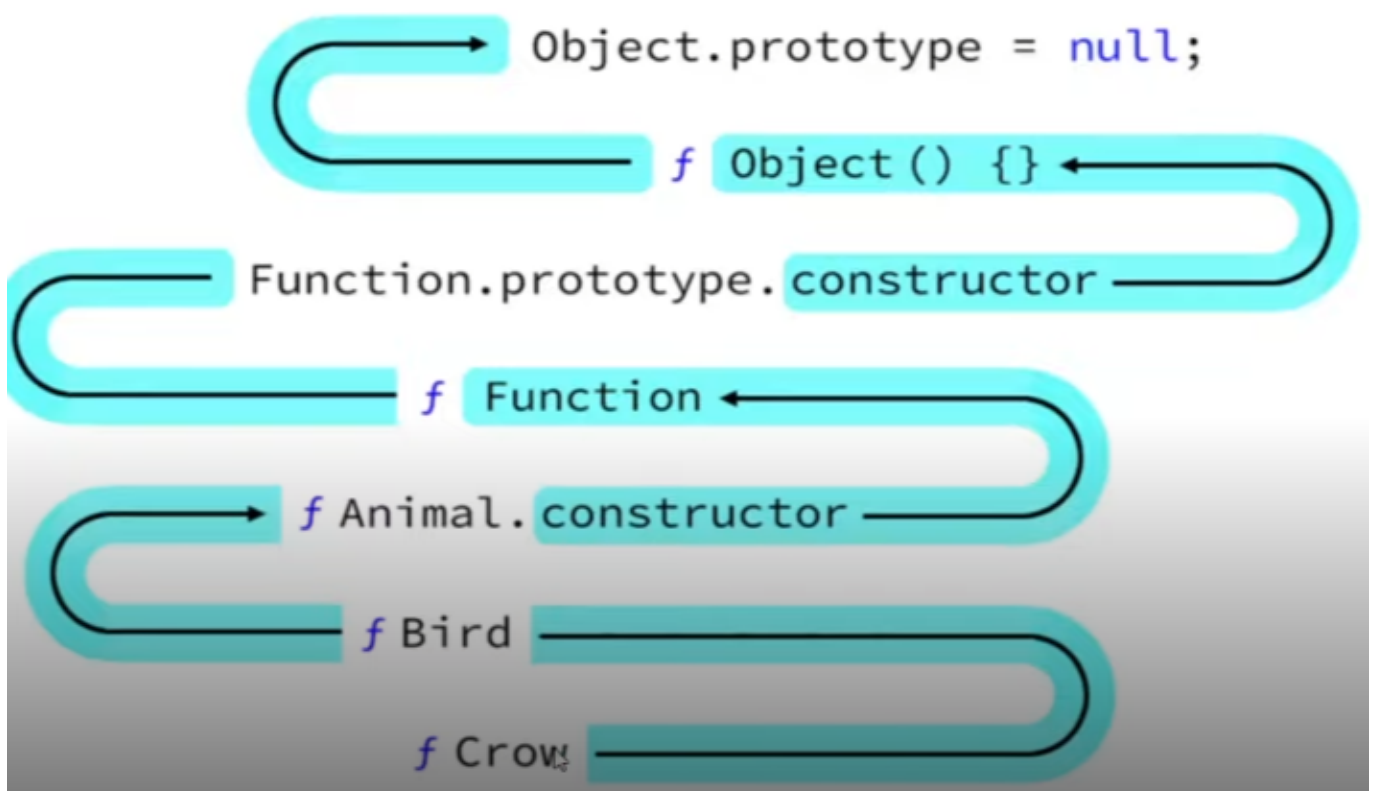
Sempre que temos uma constante do tipo não primitivo(complexo) ele vai ter a propriedade `_proto_` que vai ter uma serie de métodos e propriedades, então podemos utilizar o objeto `hasOwnProperty` para ver se o objeto tem alguma chave com aquele nome por conta do prototype, pois ele está na classe pai dele, podemos converter um objeto para `toString` pois a classe pai (prototype) tem essa função, então quando fazemos a chamada de um objeto usamos `.toString` se caso não existir ele seguiria a procura até o resultado fosse null

Exemplo array

No `_proto_` do array existem todos os métodos do array pois é o protótipo que define os arrays no javascript.

Protótipos

Cadeia de protótipos (prototype chain)



O exemplo a seguir seria o pássaro que herdaria de animal que herdaria de function.prototype que herdaria do object.prototype assim a cadeia iria procurar sempre até chegar no null.

Aula 2 | Etapa 2:

Classes

Orientação a Objetos

DIGITAL
NOVATION
NE

Classes

Syntatic sugar: uma sintaxe feita para facilitar a escrita

```
var Meal = function(food) {  
  this.food = food  
}
```

```
Meal.prototype.eat = function() {  
  return '😋'  
}
```

✗ OLD

```
class Meal {  
  constructor (food) {  
    this.food = food  
  }
```

```
  eat() {  
    return '😋'  
  }  
}
```

✓ NEW

🐦 samantha_ming

samanthaming.com

@samanthaming

As classes no Javascript não existem nativamente são açúcar sintético ou syntatic sugar, que é uma sintaxe para facilitar a escrita mas, o que acontece é usamos sempre objetos que têm protótipos mas com o javascript podemos fazer a sintaxe de classe mais parecida com outras linguagens que são feitas para acomodar o paradigma de orientação a objetos, não é o que está acontecendo e sim objetos e qualquer tipo de herança é feito por protótipos



Classes

Javascript não possui classes nativamente. Todas as classes são objetos e a herança se dá por protótipos.

```
1 class Animal {
2   constructor(type = 'animal') {
3     this.type = type
4   }
5
6   get type() {
7     return this._type
8   }
9
10  set type(val) {
11    this._type = val.toUpperCase()
12  }
13
14  makeSound() {
15    console.log('Making animal sound')
16  }
17 }
18
19 let a = new Animal()
20 console.log(a.type) //ANIMAL
```

```
1 class Cat extends Animal {
2   constructor(){
3     super('cat')
4   }
5
6   makeSound() {
7     super.makeSound()
8     console.log('Meow!')
9   }
10 }
11
12 let b= new Cat()
13 console.log(b.type) //CAT
14
```

construtor (points to line 2 of Animal)

getter e setter (points to lines 6-12 of Animal)

super() (points to line 3 of Cat)

método (points to line 6 of Cat)

Classe animal que tem a classe filha cat com a anatomia de uma classe em javascript que tem sempre um construtor que se for passado parametros ele atribui a certos valores dentro dessa classe.

O construtor é o animal, então se não for passado nenhuma informação o tipo vai ser animal se for passado outra coisa será passado de default.

Tem também os getters e setters para termos acesso aos parâmetros e propriedade do objeto utilizamos uma sintaxe para poder settar esse objetos, então o get type que é o tipo do animal vai retornar o tipo e o set para determinar o qual tipo tem outro valor então são métodos que podemos utilizar.

Também temos a classe filha do cat que tem um método chamado super que vai mandar para a função pai os parâmetros que estão ali, então quando digo supercat será construído o tipo 'animal' mas mandando cat então o tipo sera cat O super serve para utilizar as propriedades do construtor que existem na classe pai.

O metodo makeSound é para sobrescrever métodos se quisermos que a classe filha tenha um metodo diferente podemos sobrescrever.