OOP

by : pna & yk

Batam, 28 February 2019

KEYWORD THIS

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the **person object** that "owns" the `fullName` function.

In other words, `this.firstName` means the `firstName` property of **this object**.

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id     : 5566,
  fullName : function() {
    return this.firstName + " " +
this.lastName;
  }
};
```

EXAMPLE

```
var book1 = {
   title : "Book One ",
   author : "John Doe ",
   year : "2019",
   getSummary : function(){
       return `${this.title} was written by
${this.author} in ${this.year}`
   }
};
```

this Alone

When used alone, the **owner** is the Global object, so `this` refers to the Global object.

In a browser window the Global object is `[object Window]`:

```
var x = this;
```

CALL METHOD

The `call()` method is a predefined JavaScript method.

With `call()`, an object can use a method belonging to another object.

```
var sayHi ={
 message:function(){
   return this.namaAwal+ " " + this.namaAkhir +
"say hi"
 }
}
var badai = {
 namaAwal :"Winata",
 namaAkhir  : "beibeh",
}
var danu ={
 namaAwal : "Zakarias",
 namaAkhir  : "Danujatmiko"
}

console.log(sayHi.message.call(badai)) ;
```

The meaning / purpose of a constructor function

Let's imagine that we are tasked with building an application that requires us to create `car` objects. Each car that we create should have a make, model and year. So we get started by doing something like this:

```
var car1 = {
    make: "Honda",
    model: "Accord",
    year: 2002
}
var car2 = {
    make: "Mazda",
    model: "6",
    year: 2008
}
var car3 = {
    make: "BMW",
    model: "7 Series",
    year: 2012
}
```

Constructor

But notice how much duplication is going on! All of these objects look the same, yet we are repeating ourselves over and over again. It would be really nice to have a **blueprint** that we could work off of to reduce the amount of code that we have.

Our first constructor function

So what is a constructor function? It's written just like any other function, except that by convention we capitalize the name of the function to denote that it is a constructor. We call these functions constructors because their job is to construct objects. Here is what a constructor function to create car objects might look like. Notice the capitalization of the name of the function; this is a **best** practice when creating constructor functions so that other people know what kind of function it is.

CONSTRUCTOR

```
function Car(make, model, year){
    this.make = make;
    this.model = model;
    this.year = year;
}
```

```
var probe = new Car('Ford', 'Probe', 1993);
var cmax = new Car('Ford', 'C-Max', 2014);

probe.make;  // Returns "Ford"
cmax.year;   // Returns 2014
```

So how do constructor functions actually "construct" these objects? Through the `new` keyword that we saw before. To construct a new `Car`, use `new`:

PROTOTYPE

Every single function that is created in JavaScript has a `prototype` property.

Let's start by looking at `Object.prototype`. In the Chrome console, try typing `Object.prototype` then expand the object you get back. You can see that `Object` already has many properties on its prototype.

```
function Person(name) {
   this.name = name;
}

var tim = new Person("Tim");

Person.prototype; // Object {}
```

EXAMPLE

When you create a constructor function, that function will have it's own prototype. Let's try that out by creating a `Person` constructor function:

```
function Person(name) {
    this.name = name;
}

var tim = new Person("Tim");

Person.prototype; // Object {}
```

So far, our `Person` constructor function has a prototype and the only two properties available on the prototype should be `constructor` and `__proto__`. Let's try adding a function to the Person prototype:

```
Person.prototype.sayHello = function() {
    return "Hello, " + this.name;
};
```

LATIHAN

1. Create a constructor function for a Person, each person should have a firstName, lastName, favoriteColor and favoriteNumber.

2. Write a method called multiplyFavoriteNumber that takes in a number and returns the product of the number and the Person's favorite number

3. Refactor the following code so that there is no duplication inside the `Child` function.

```
function Parent(firstName, lastName,
favoriteColor, favoriteFood){
    this.firstName = firstName;
    this.lastName = lastName;
    this.favoriteColor = favoriteColor;
    this.favoriteFood = favoriteFood;
}
```

```
function Child(firstName, lastName,
favoriteColor, favoriteFood){
    this.firstName = firstName;
    this.lastName = lastName;
    this.favoriteColor = favoriteColor;
    this.favoriteFood = favoriteFood;
}
```

LATIHAN

berikut function yang berisikan sebuah constructor,

tambahkan parameter berupa nama orang tua yang memiliki key/properti berupa namaAyah dan ibu, dan properti hobby yang memiliki nama data type berupa array, dan buatlah sebuah fungsi di dalamnya yang bernama getSummary yang mengambil setiap parameter yang diinputkan.

```
//Skeleteon Code

function student (namaAwal, namaAkhir){
 this.namaAwal = namaAwal;
 this.namaAkhir = namaAkhir;
//////// tambahkan namaayah dan namaIbu, simpan di objek
orangTua
//tambahkan hobby simpan di dalam array
 this.getSummary= function (){
   return `` ---> gunakan concatenation
 }
}
//test case
var student1 = new
student("Winata","Beibeh","jesiiica","tony
stark","memancing","memasak")
```

```
Winata Beibeh seorang pemuda dari Surabaya, lahir dari pasangan jesiiica dan tony
stark ia memiliki hobby memancing dan memasak
> undefined
```

INHERITANCE

An important concept in object oriented programming is inheritance. The idea behind inheritance is that one or more parent / super classes can pass along functions and properties to other child / sub classes.

```
function
User(name,job,id,membership) {
    Person.call(this, name, job);
    this.id = id;
    this.membership = membership;
}

User.prototype =
Object.create(Person.prototype);

var user1 = new User('Yudi',
'otaku', 1, "premium");
console.log(user1);
```