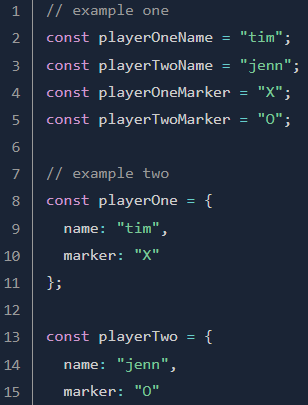
**OBJECT & OBJECT CONSTRUCTOR**

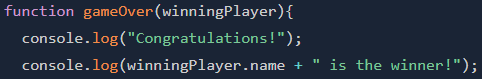
1. **Object As a Design Pattern**

One of the simplest ways you can begin to organize your code is by grouping things into objects. Take these examples from a ‘tic tac toe’ game:

The benefit of the example two is huge:

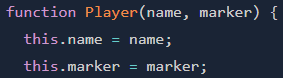
1. you dont need to remember var name (like example one)

example one:

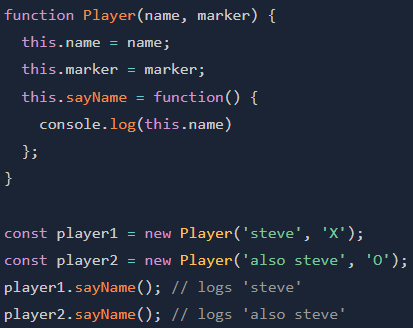
1. you don’t need to know which player’s name you want to print
2. In more complicated (such as an online shopping site with a large inventory) using objects (to keep track of an item’s name, price, description and other things) is the only way to go.

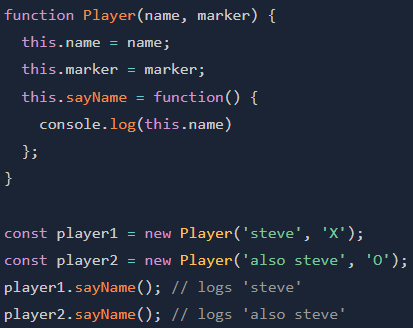
Unfortunately, in that (complicated) type of situation, manually typing out the contents of our objects is not feasible either. We need a cleaner way to create our objects, which brings us to OBJECT CONSTRUCTOR.

1. **Object Constructor**

When you have a specific type of object that you need to duplicate like our player or inventory items, a better way to create them is using an object constructor, which is a function that looks like this:

and which you use by calling the function with the keyword ' new '

Just like with objects created using the Object Literal method, you can add functions to the object:

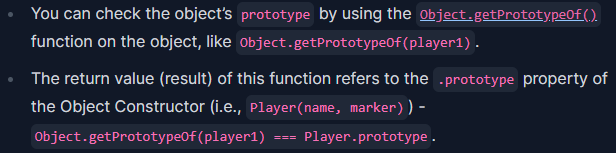
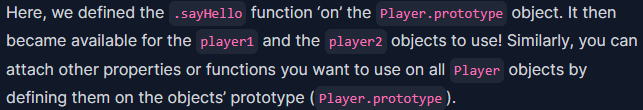
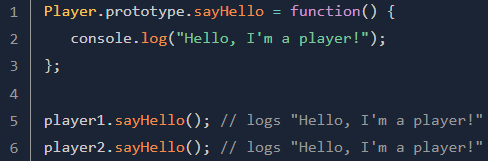


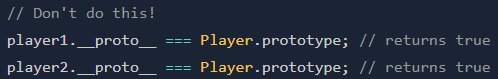
1. **The Prototype**

All objects in JavaScript have a 'prototype'. The prototype is another object that the original object inherits from, which is to say, the original object has access to all of its prototype’s methods and properties.

Semua objeck memiliki prototype. Prototype adalah objek lain yang diwarisi oleh objek asli, artinya objek asli memiliki akses ke semua metode dan properti prototype nya.

Misal: player1 atau player2 adalah objek aslinya, maka player1 atau player 2 bisa mengakses property/function (object)prototypenya. Sebagai contoh jika fungsi .sayHello() ada di (object)prototype nya maka player1 atau player2 bisa mengakses .sayHello().

* Accessing an object's prototype
* Benefit of ‘prototype’
* Object.getPrototypeOf() vs. .\_\_proto\_\_ vs. [[Prototype]]
* .\_\_proto\_\_

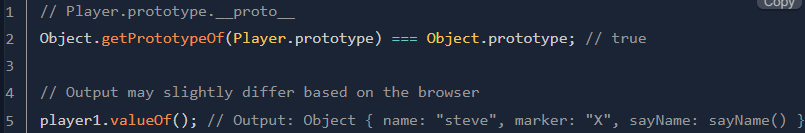
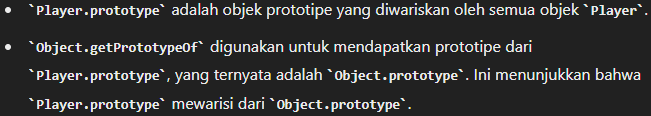
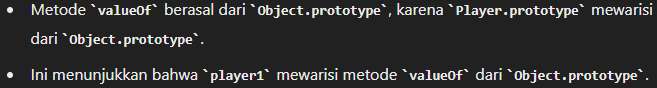
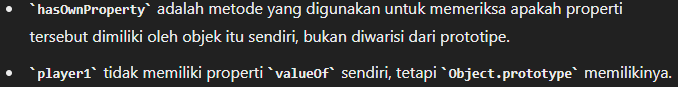
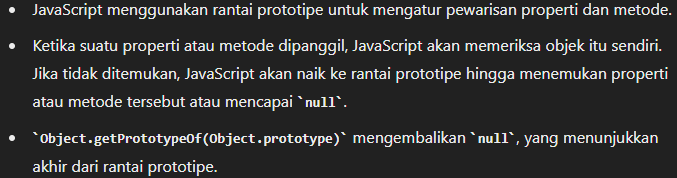
This is a non-standard way of doing so, and **deprecated**.

* [[Prototype]]

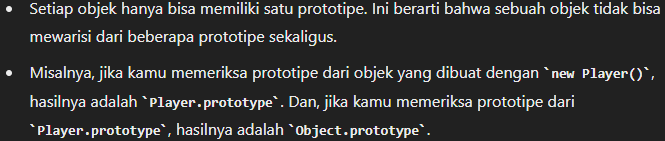
In some places, like **legacy code**, you might also come across [[Prototype]], like player1.[[Prototype]].

* The purpose of defining properties and functions on the prototype

1. We can define properties and functions common among all objects on the ‘prototype’ to save memory.
2. **Prototypal Inheritance**, which (we’ve referred earlier) is the original object has access to all of its prototype’s methods and properties.

* Perhatikan berikut ini ( utk memahami Object.prototype & valueOf() )
* Perhatikan ( utk memahami hasOwnProperty )
* **Prototype Chain (rantai properti)**
* **Note**

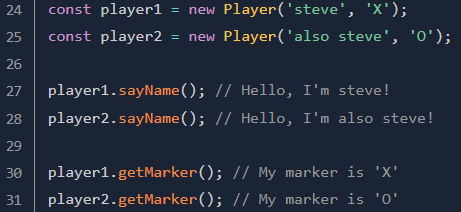
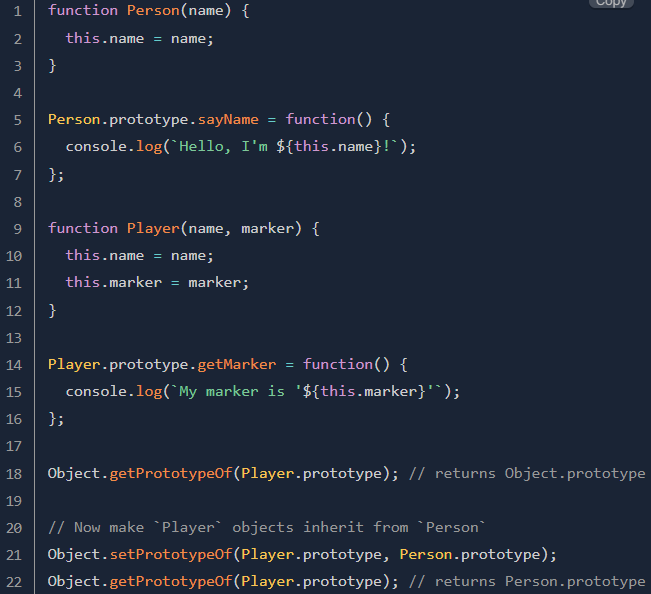
1. Every ‘prototype’ object inherits from **Object.prototype** by default.
2. Nilai 'Object.getPrototypeOf()' dari Sebuah Objek hanya satu.



1. **Recommended method for Prototypal Inheritance**

* ****
* ****

Let’s see how it works by adding a ‘Person’ Object Constructor to the ‘Player’ example, and making ‘Player’ inherit from Person:

 Note: Using setPrototypeOf() after objects(player1 & player2) have already been created can result in performance issues.