**Object creation variants.**

**Variant 1** : '**new Object()**' -> Object constructor without arguments.

var p1 = new Object(); // 'new Object()' create and return empty object -> {}

var p2 = new Object(); // 'new Object()' create and return empty object -> {}

console.log(p1); // empty object -> {}

console.log(p2); // empty object -> {}

// p1 and p2 are pointers to different objects

console.log(p1 === p2); // false

console.log(p1.prototype); // undefined

// empty object which is in fact Object.prototype

console.log(p1.\_\_proto\_\_); // {}

// empty object to which p1.\_\_proto\_\_ points

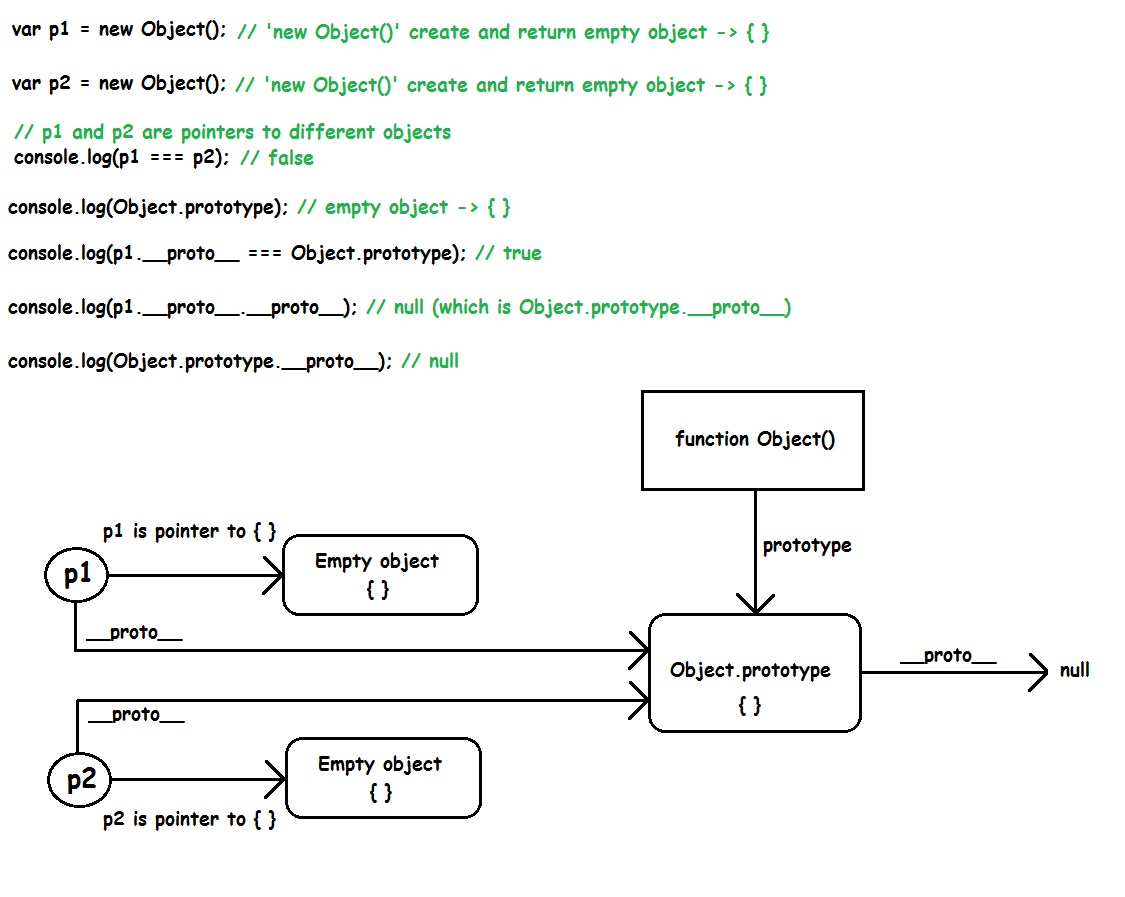
console.log(Object.prototype); // {}

console.log(p1.\_\_proto\_\_ === Object.prototype); // true

// null, which is in fact Object.prototype.\_\_proto\_\_

console.log(p1.\_\_proto\_\_.\_\_proto\_\_); // null

console.log(Object.prototype.\_\_proto\_\_); // null

[](https://i.stack.imgur.com/zcyc8.jpg)

**Variant 2** : '**new Object(person)**' -> Object constructor with argument.

const person = {

name: 'no name',

lastName: 'no lastName',

age: -1

}

// 'new Object(person)' return 'person', which is pointer to the object ->

// -> { name: 'no name', lastName: 'no lastName', age: -1 }

var p1 = new Object(person);

// 'new Object(person)' return 'person', which is pointer to the object ->

// -> { name: 'no name', lastName: 'no lastName', age: -1 }

var p2 = new Object(person);

// person, p1 and p2 are pointers to the same object

console.log(p1 === p2); // true

console.log(p1 === person); // true

console.log(p2 === person); // true

p1.name = 'John'; // change 'name' by 'p1'

p2.lastName = 'Doe'; // change 'lastName' by 'p2'

person.age = 25; // change 'age' by 'person'

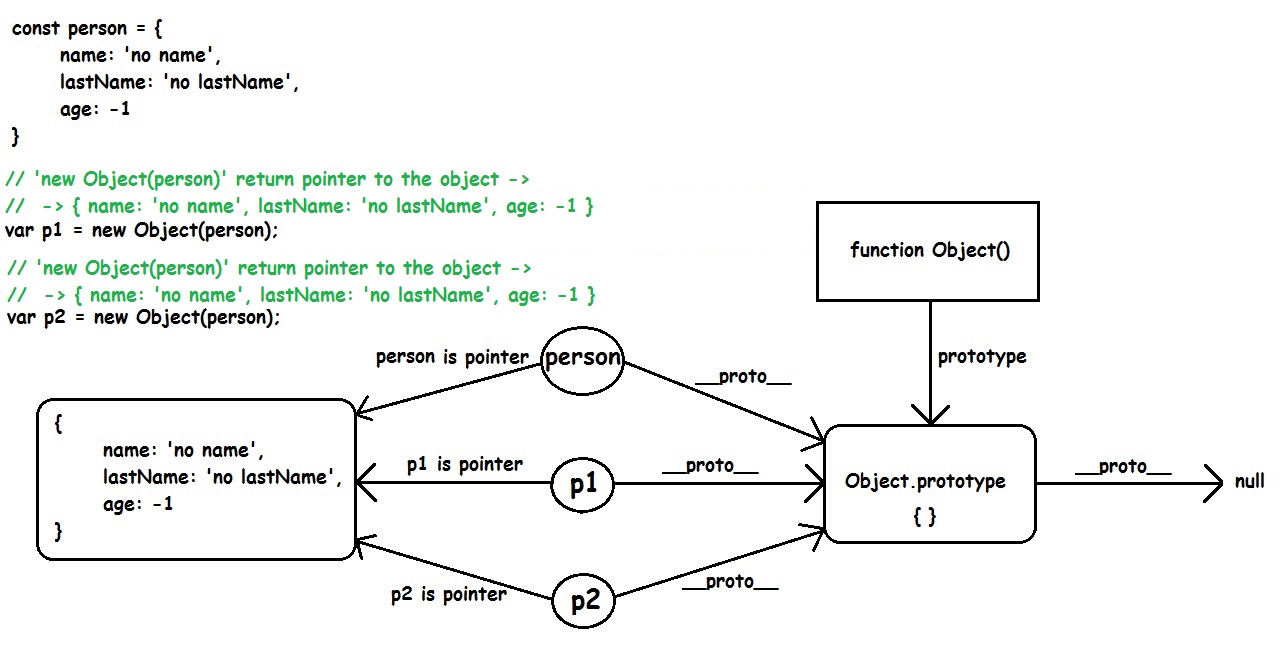
// when print 'p1', 'p2' and 'person', it's the same result,

// because the object they points is the same

console.log(p1); // { name: 'John', lastName: 'Doe', age: 25 }

console.log(p2); // { name: 'John', lastName: 'Doe', age: 25 }

console.log(person); // { name: 'John', lastName: 'Doe', age: 25 }

[](https://i.stack.imgur.com/UMtro.jpg)

**Variant 3.1** : '**Object.create(person)**'. Use Object.create with simple object 'person'. 'Object.create(person)' will create(and return) new empty object and add property '\_\_proto\_\_' to the same new empty object. This property '\_\_proto\_\_' will point to the object 'person'.

const person = {

name: 'no name',

lastName: 'no lastName',

age: -1,

getInfo: function getName() {

return `${this.name} ${this.lastName}, ${this.age}!`;

}

}

var p1 = Object.create(person);

var p2 = Object.create(person);

// 'p1.\_\_proto\_\_' and 'p2.\_\_proto\_\_' points to

// the same object -> 'person'

// { name: 'no name', lastName: 'no lastName', age: -1, getInfo: [Function: getName] }

console.log(p1.\_\_proto\_\_);

console.log(p2.\_\_proto\_\_);

console.log(p1.\_\_proto\_\_ === p2.\_\_proto\_\_); // true

console.log(person.\_\_proto\_\_); // {}(which is the Object.prototype)

// 'person', 'p1' and 'p2' are different

console.log(p1 === person); // false

console.log(p1 === p2); // false

console.log(p2 === person); // false

// { name: 'no name', lastName: 'no lastName', age: -1, getInfo: [Function: getName] }

console.log(person);

console.log(p1); // empty object - {}

console.log(p2); // empty object - {}

// add properties to object 'p1'

// (properties with the same names like in object 'person')

p1.name = 'John';

p1.lastName = 'Doe';

p1.age = 25;

// add properties to object 'p2'

// (properties with the same names like in object 'person')

p2.name = 'Tom';

p2.lastName = 'Harrison';

p2.age = 38;

// { name: 'no name', lastName: 'no lastName', age: -1, getInfo: [Function: getName] }

console.log(person);

// { name: 'John', lastName: 'Doe', age: 25 }

console.log(p1);

// { name: 'Tom', lastName: 'Harrison', age: 38 }

console.log(p2);

// use by '\_\_proto\_\_'(link from 'p1' to 'person'),

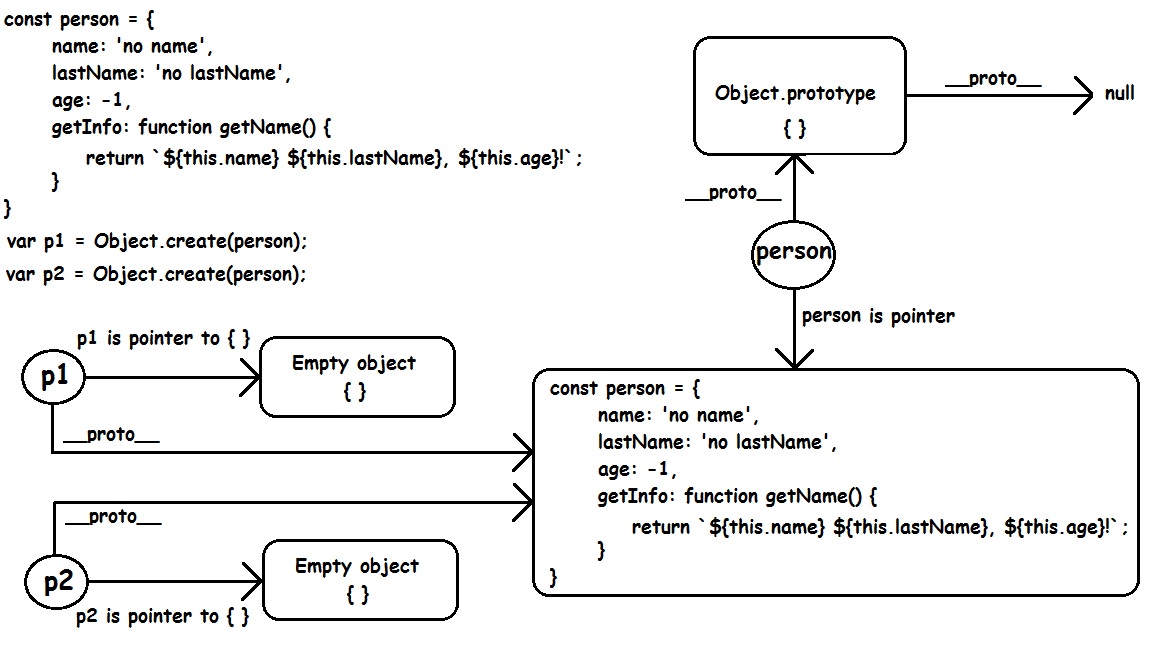
// person's function 'getInfo'

console.log(p1.getInfo()); // John Doe, 25!

// use by '\_\_proto\_\_'(link from 'p2' to 'person'),

// person's function 'getInfo'

console.log(p2.getInfo()); // Tom Harrison, 38!

[](https://i.stack.imgur.com/JJzaT.jpg)

**Variant 3.2** : '**Object.create(Object.prototype)**'. Use Object.create with built-in object -> 'Object.prototype'. 'Object.create(Object.prototype)' will create(and return) new empty object and add property '\_\_proto\_\_' to the same new empty object. This property '\_\_proto\_\_' will point to the object 'Object.prototype'.

// 'Object.create(Object.prototype)' :

// 1. create and return empty object -> {}.

// 2. add to 'p1' property '\_\_proto\_\_', which is link to 'Object.prototype'

var p1 = Object.create(Object.prototype);

// 'Object.create(Object.prototype)' :

// 1. create and return empty object -> {}.

// 2. add to 'p2' property '\_\_proto\_\_', which is link to 'Object.prototype'

var p2 = Object.create(Object.prototype);

console.log(p1); // {}

console.log(p2); // {}

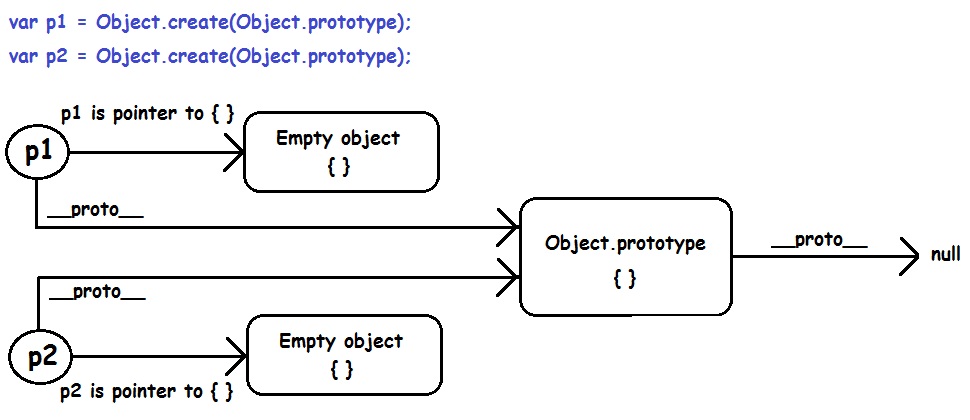
console.log(p1 === p2); // false

console.log(p1.prototype); // undefined

console.log(p2.prototype); // undefined

console.log(p1.\_\_proto\_\_ === Object.prototype); // true

console.log(p2.\_\_proto\_\_ === Object.prototype); // true

[](https://i.stack.imgur.com/glM5q.jpg)

**Variant 4** : '**new SomeFunction()**'

// 'this' in constructor-function 'Person'

// represents a new instace,

// that will be created by 'new Person(...)'

// and returned implicitly

function Person(name, lastName, age) {

this.name = name;

this.lastName = lastName;

this.age = age;

//-----------------------------------------------------------------

// !--- only for demonstration ---

// if add function 'getInfo' into

// constructor-function 'Person',

// then all instances will have a copy of the function 'getInfo'!

//

// this.getInfo: function getInfo() {

// return this.name + " " + this.lastName + ", " + this.age + "!";

// }

//-----------------------------------------------------------------

}

// 'Person.prototype' is an empty object

// (before add function 'getInfo')

console.log(Person.prototype); // Person {}

// With 'getInfo' added to 'Person.prototype',

// instances by their properties '\_\_proto\_\_',

// will have access to the function 'getInfo'.

// With this approach, instances not need

// a copy of the function 'getInfo' for every instance.

Person.prototype.getInfo = function getInfo() {

return this.name + " " + this.lastName + ", " + this.age + "!";

}

// after function 'getInfo' is added to 'Person.prototype'

console.log(Person.prototype); // Person { getInfo: [Function: getInfo] }

// create instance 'p1'

var p1 = new Person('John', 'Doe', 25);

// create instance 'p2'

var p2 = new Person('Tom', 'Harrison', 38);

// Person { name: 'John', lastName: 'Doe', age: 25 }

console.log(p1);

// Person { name: 'Tom', lastName: 'Harrison', age: 38 }

console.log(p2);

// 'p1.\_\_proto\_\_' points to 'Person.prototype'

console.log(p1.\_\_proto\_\_); // Person { getInfo: [Function: getInfo] }

// 'p2.\_\_proto\_\_' points to 'Person.prototype'

console.log(p2.\_\_proto\_\_); // Person { getInfo: [Function: getInfo] }

console.log(p1.\_\_proto\_\_ === p2.\_\_proto\_\_); // true

// 'p1' and 'p2' points to different objects(instaces of 'Person')

console.log(p1 === p2); // false

// 'p1' by its property '\_\_proto\_\_' reaches 'Person.prototype.getInfo'

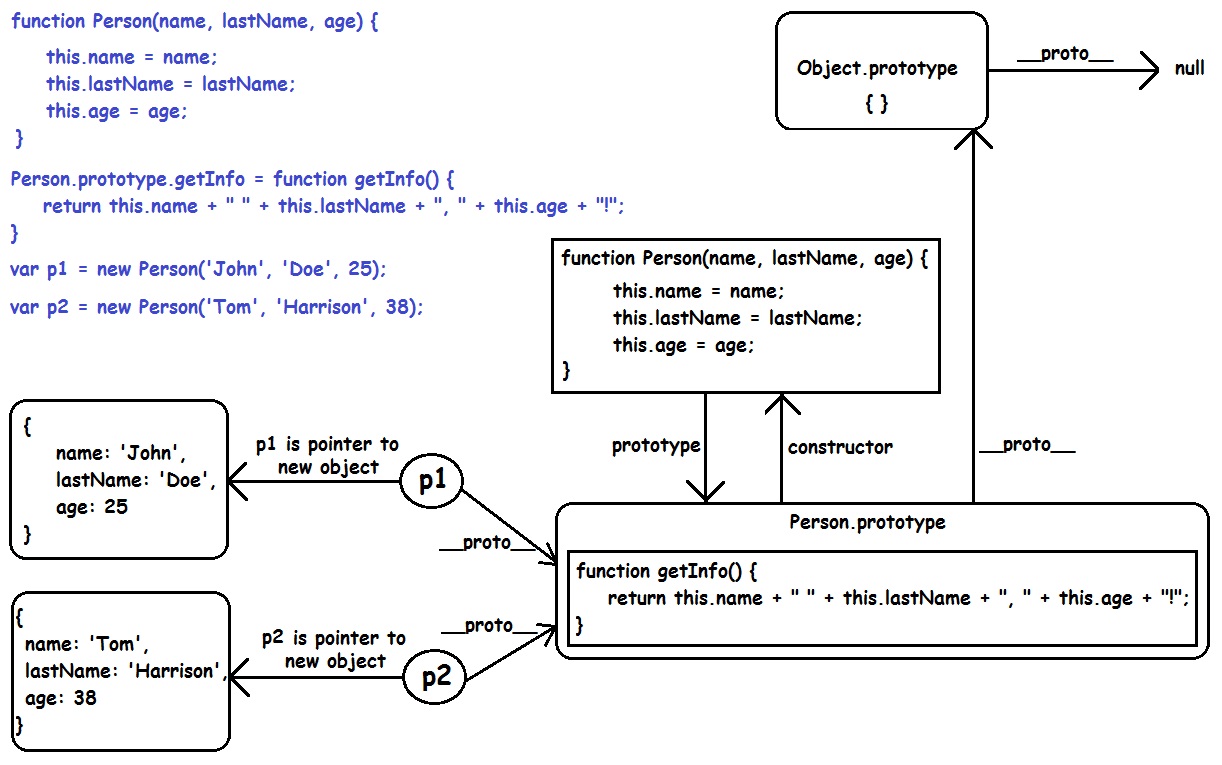
// and use 'getInfo' with 'p1'-instance's data

console.log(p1.getInfo()); // John Doe, 25!

// 'p2' by its property '\_\_proto\_\_' reaches 'Person.prototype.getInfo'

// and use 'getInfo' with 'p2'-instance's data

console.log(p2.getInfo()); // Tom Harrison, 38!

[](https://i.stack.imgur.com/y21Xj.jpg)

