Prototypal inheritance in JavaScript

#JavaScript #Prototype #Inheritance

Object is not a primitive

```
var person = {
  name: 'John Doe',
  married: false,
  age: 25,
};
```

```
console.log(person.name);  // John Doe
```

Object is not a primitive

```
var person = {
  name: 'John Doe',
  married: false,
  age: 25,
};
```

```
console.log(person.name);  // John Doe
```

```
var odd    = [ 1, 3, 5, 7, 9 ];
```

```
console.log(odd.length);      // 5
```

Object is not a primitive

```
var person = {
  name: 'John Doe',
  married: false,
  age: 25,
};
```

```
console.log(person.name);  // John Doe
```

```
var odd    = [ 1, 3, 5, 7, 9 ];
```

```
console.log(odd.length);       // 5
```

```
var regex  = /match-me/i;
```

```
console.log(regex.ignoreCase); // true
```

Object is not a primitive

```
var person = {
  name: 'John Doe',
  married: false,
  age: 25,
};
```

```
console.log(person.name);  // John Doe
```

```
var odd    = [ 1, 3, 5, 7, 9 ];
```

```
console.log(odd.length);       // 5
```

```
var regex  = /match-me/i;
```

```
console.log(regex.ignoreCase); // true
```

```
var greet  = function (name) {
  alert('Hello,' + name);
};
```

```
console.log(greet.length);     // 1
```

Object is not a primitive

```
var message = 'Welcome!';
var count   = 0;
var flag    = false;
```

```
var person = {
  name: 'John Doe',
  married: false,
  age: 25,
};
```

```
console.log(person.name);  // John Doe
```

```
var odd    = [ 1, 3, 5, 7, 9 ];
```

```
console.log(odd.length);       // 5
```

```
var regex  = /match-me/i;
```

```
console.log(regex.ignoreCase); // true
```

```
var greet  = function (name) {
  alert('Hello,' + name);
};
```

```
console.log(greet.length);     // 1
```

Object is not a primitive

```
var blank   = null;
var nothing = undefined;

var message = 'Welcome!';
var count   = 0;
var flag    = false;
```

```
var person = {
  name: 'John Doe',
  married: false,
  age: 25,
};
```

```
console.log(person.name);  // John Doe
```

```
var odd    = [ 1, 3, 5, 7, 9 ];
```

```
console.log(odd.length);       // 5
```

```
var regex  = /match-me/i;
```

```
console.log(regex.ignoreCase); // true
```

```
var greet  = function (name) {
  alert('Hello,' + name);
};
```

```
console.log(greet.length);     // 1
```

Theory of strings?!

Theory of strings?!

```
var obj = {};
obj[null] = 42;

console.log(obj.hasOwnProperty('null'));
// true

console.log(obj['null']);
// 42
```

Theory of strings?!

```
var obj = {};
obj[null] = 42;

console.log(obj.hasOwnProperty('null'));
// true

console.log(obj['null']);
// 42
```

```
var a = ['zero','one','two'];
a['3'] = 'three';

console.log(a[1]);
// one
console.log(a['1']);
// one

console.log(a[3]);
// three
console.log(a['3']);
// three

console.log(a.length);
// 4
```

Theory of strings?!

```
var obj = {};
obj[null] = 42;

console.log(obj.hasOwnProperty('null'));
// true

console.log(obj['null']);
// 42
```

```
var a = ['x','y','z'];
a['test'] = 'array is object';

console.log(a['test']);
// array is object

console.log(a.length);
```

```
var a = ['zero','one','two'];
a['3'] = 'three';

console.log(a[1]);
// one
console.log(a['1']);
// one

console.log(a[3]);
// three
console.log(a['3']);
// three

console.log(a.length);
// 4
```

null vs. undefined

```
console.log(typeof null === 'object');          // true
console.log(typeof undefined === 'undefined');  // true
```

null vs. undefined

```
console.log(typeof null === 'object');          // true
console.log(typeof undefined === 'undefined');  // true
```

```
var n;

console.log(n);
// undefined
```

null vs. undefined

```
console.log(typeof null === 'object');          // true
console.log(typeof undefined === 'undefined');  // true
```

```
var n;

console.log(n);
// undefined
```

```
var obj = {
  abc: 55
};

console.log(obj.xyz);
```

null vs. undefined

```
console.log(typeof null === 'object');          // true
console.log(typeof undefined === 'undefined');  // true
```

```
var n;

console.log(n);
// undefined
```

```
function f (arg) {
  console.log(arg);
}

f();
// undefined
```

```
var obj = {
  abc: 55
};

console.log(obj.xyz);
```

null vs. undefined

```
console.log(typeof null === 'object');          // true
console.log(typeof undefined === 'undefined');  // true
```

```
var n;

console.log(n);
// undefined
```

```
function f (arg) {
  console.log(arg);
}

f();
// undefined
```

```
function nop () {
  // return omitted
}

console.log(nop());
// undefined
```

```
var obj = {
  abc: 55
};

console.log(obj.xyz);
```

null vs. undefined

```
console.log(typeof null === 'object');          // true
console.log(typeof undefined === 'undefined');  // true
```

```
var n;

console.log(n);
// undefined
```

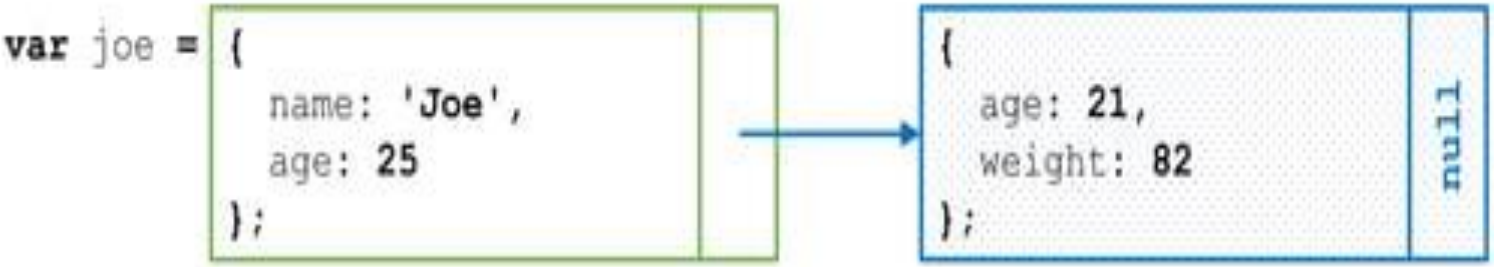
```
function f (arg) {
  console.log(arg);
}

f();
// undefined
```

```
function nop () {
  // return omitted
}

console.log(nop());
// undefined
```

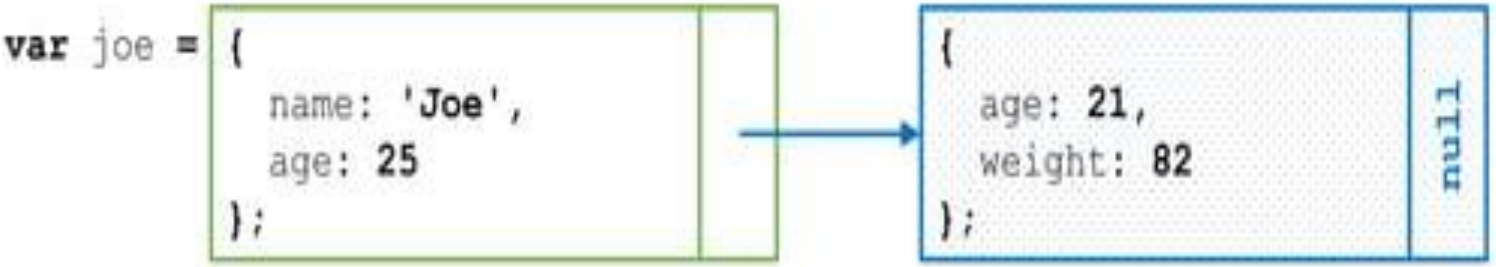
```
var obj = {
  abc: 55
};

console.log(obj.xyz);
```

```
function ret () {
  return;
}

console.log(ret());
```

Inheriting from another object

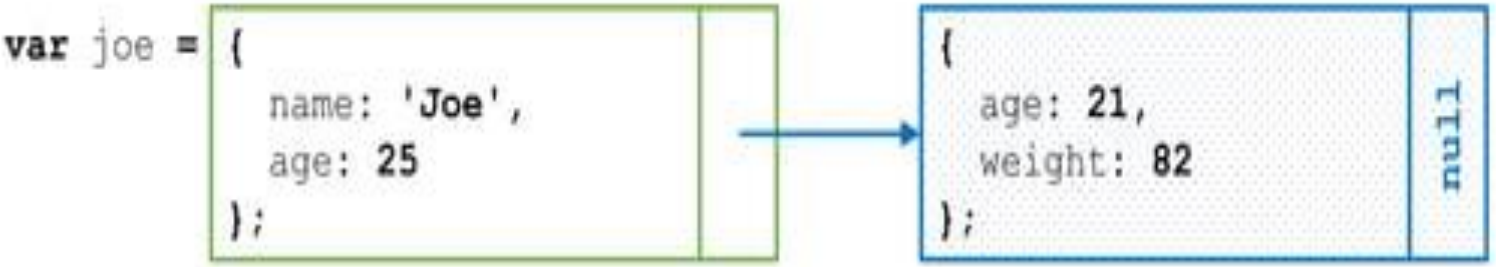


Inheriting from another object



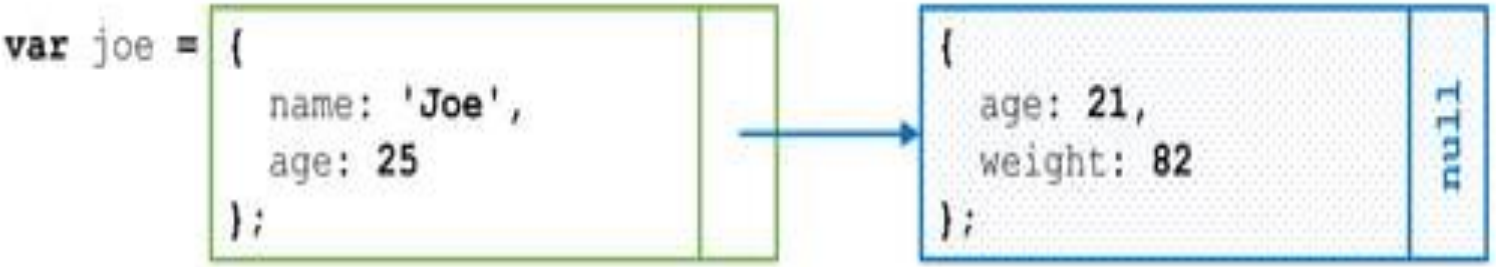

```
console.log(joe.age);
// 25
```

Inheriting from another object



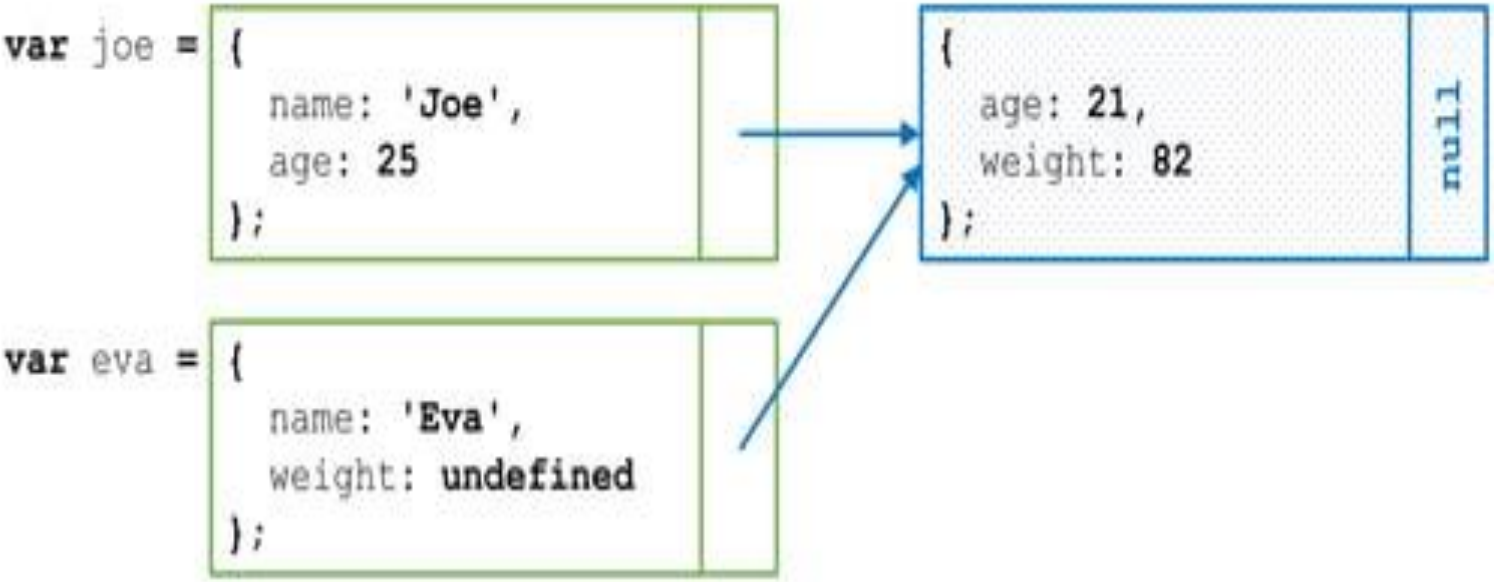

```
console.log(joe.age);
// 25
console.log(joe.weight);
// 82
```

Inheriting from another object



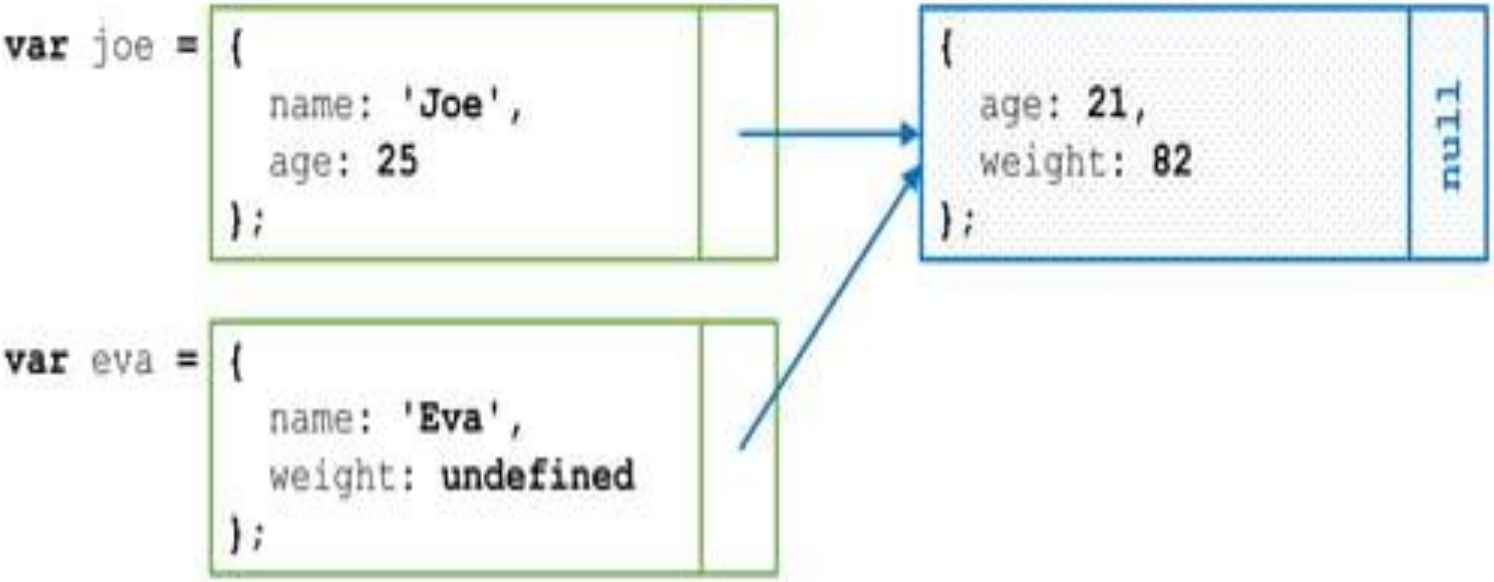

```
console.log(joe.age);
// 25
console.log(joe.weight);
// 82
```

Inheriting from another object




```
console.log(joe.age);
// 25
console.log(joe.weight);
// 82
```

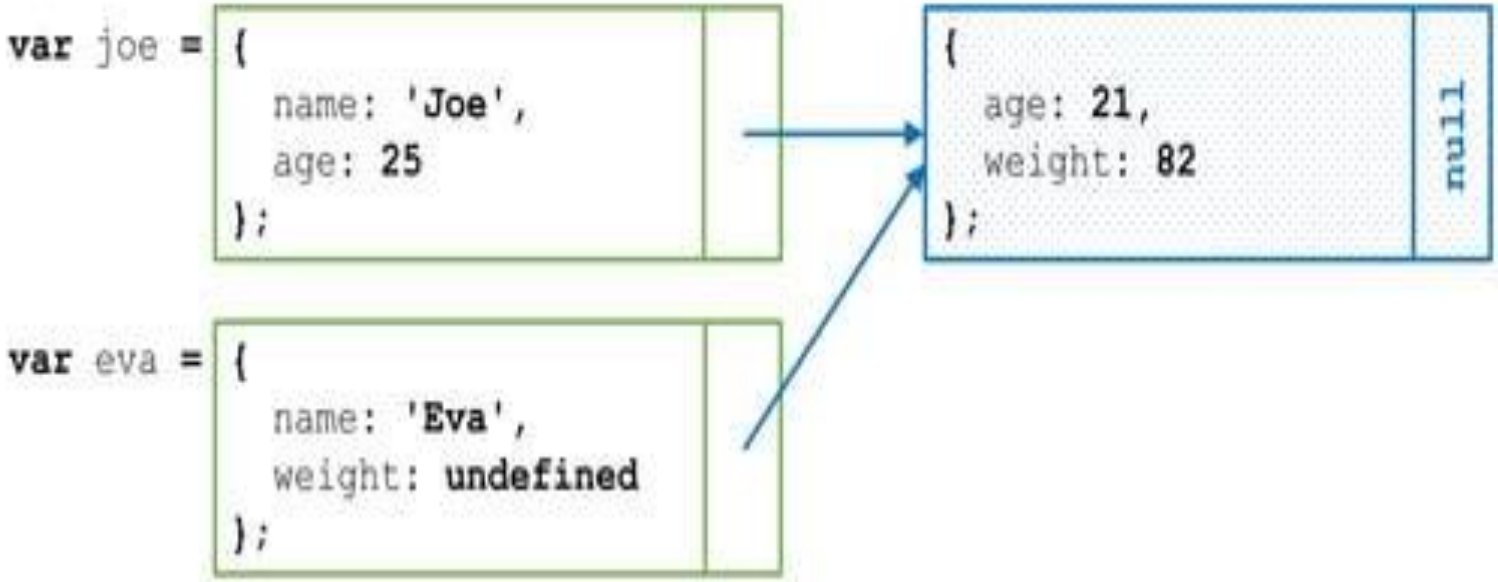
Inheriting from another object




```
console.log(joe.age);
// 25
console.log(joe.weight);
// 82
```

```
console.log(eva.age);
// 21
```

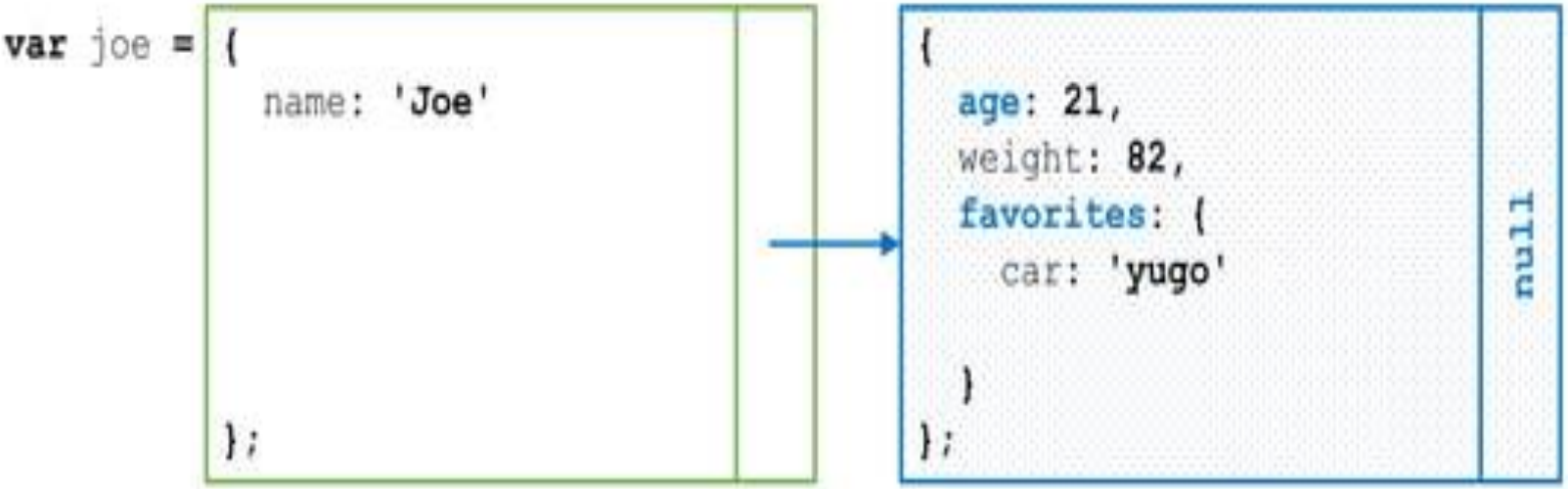
Inheriting from another object




```
console.log(joe.age);
// 25
console.log(joe.weight);
// 82
```

```
console.log(eva.age);
// 21
console.log(eva.weight);
// undefined
```

Hide and seek!



Hide and seek!

```
var joe = {
  name: 'Joe'
};
```

```
{
  age: 21,
  weight: 82,
  favorites: {
    car: 'yugo'
  }
};
```

null

```
console.log(joe.age); // 21
```

Hide and seek!

```
var joe = {
  name: 'Joe',
  age: 25

};
```

```
{
  age: 21,
  weight: 82,
  favorites: {
    car: 'yugo'

  }
};
```

null

```
console.log(joe.age); // 21
```

```
joe.age = 25;

console.log(joe.age); // 25
```

Hide and seek!

```
var joe = {
  name: 'Joe',
  age: 25

};
```

```
{
  age: 21,
  weight: 82,
  favorites: {
    car: 'yugo',
    food: 'jelly'
  }
};
```

null

```
console.log(joe.age); // 21
```

```
joe.favorites.food = 'jelly';
```

```
joe.age = 25;

console.log(joe.age); // 25
```

Hide and seek!

```
var joe = {
  name: 'Joe',
  age: 25,
  favorites: {
    food: 'plums'
  }

};
```

```
{
  age: 21,
  weight: 82,
  favorites: {
    car: 'yugo',
    food: 'jelly'
  }
};
```

null

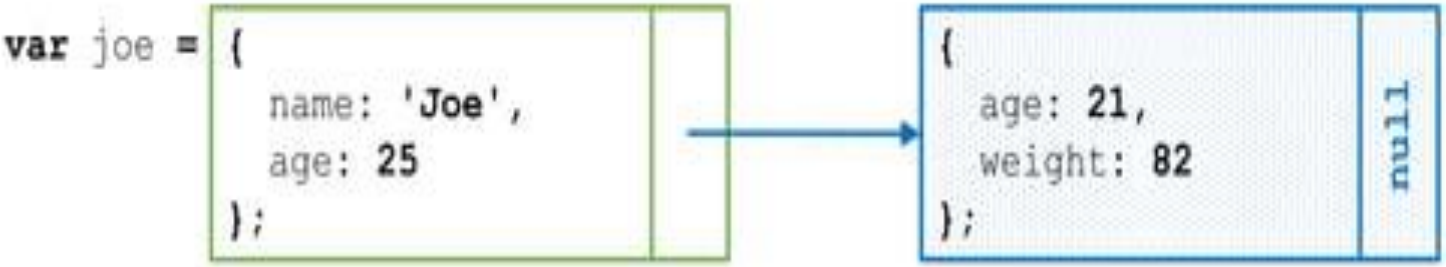
```
console.log(joe.age); // 21
```

```
joe.favorites.food = 'jelly';
```

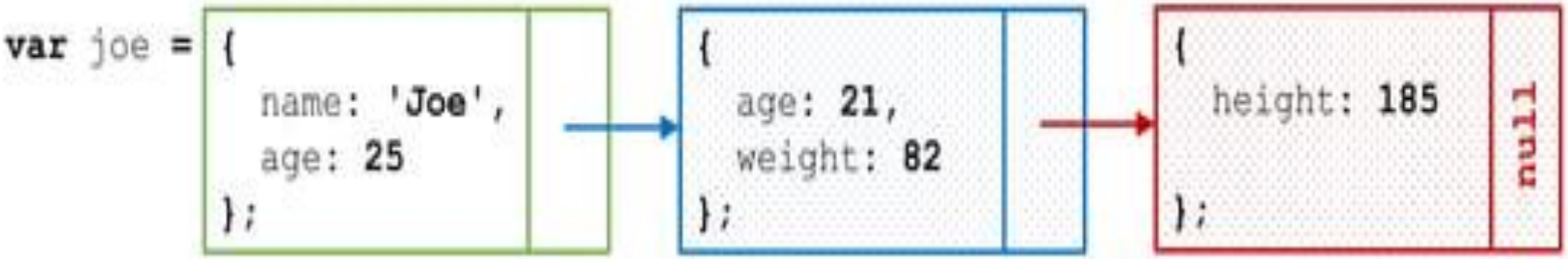
```
joe.age = 25;

console.log(joe.age); // 25
```

```
joe.favorites = {
  food: 'plums'
};
```

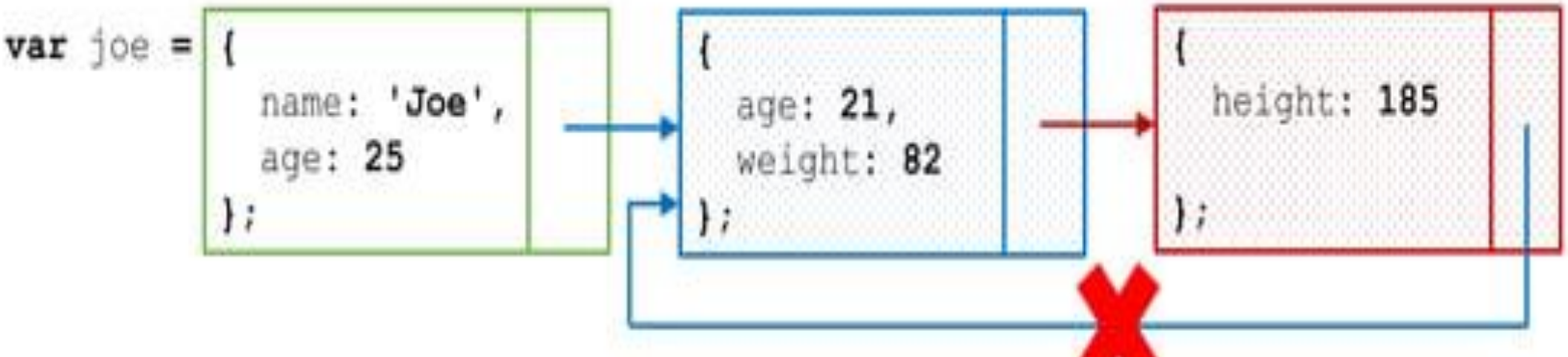
Bound in prototype chains?



Bound in prototype chains?

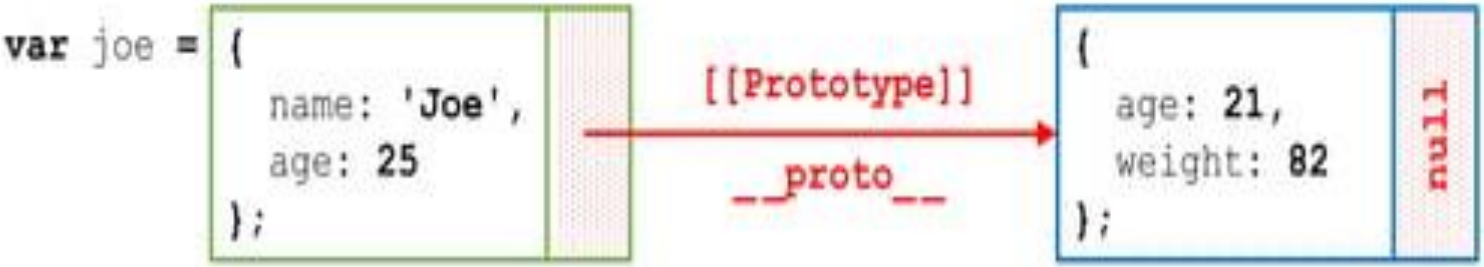


Bound in prototype chains?

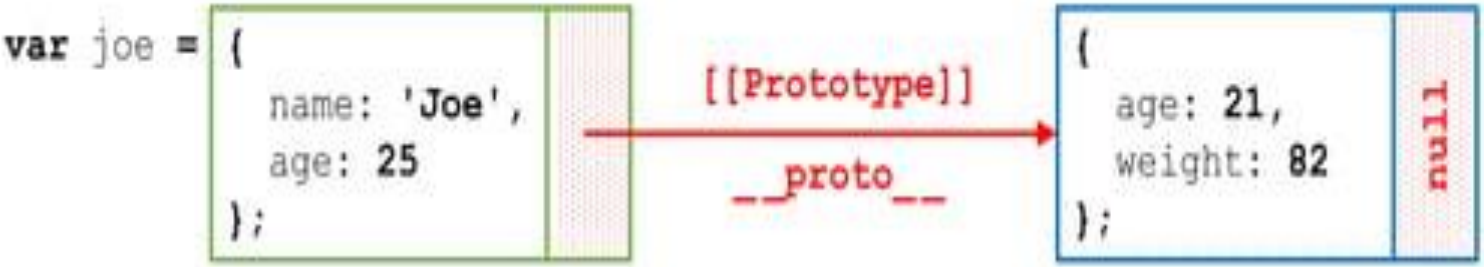


[[Prototype]] & __proto__

[[Prototype]] & __proto__

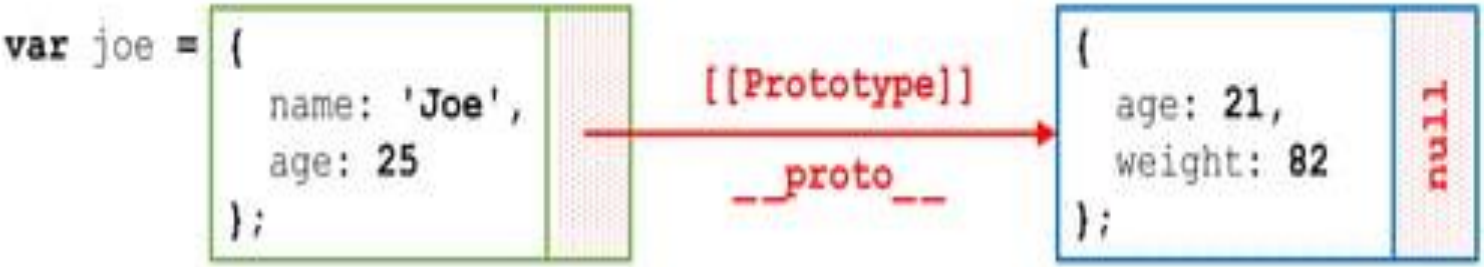


[[Prototype]] & __proto__




```
console.log(joe.age);
// 25
```

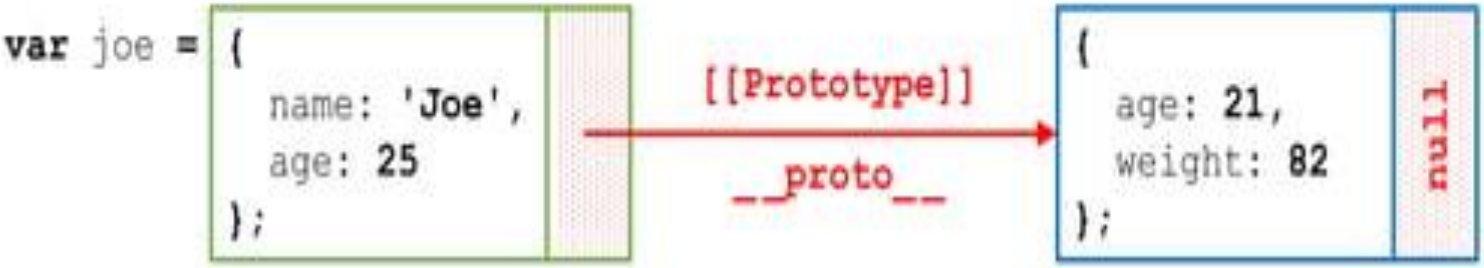
[[Prototype]] & __proto__




```
console.log(joe.age);
// 25
```

```
console.log(joe.__proto__.age);
// 21
```

[[Prototype]] & __proto__




```
console.log(joe.age);
// 25
```

```
console.log(joe.__proto__.age);
// 21
```

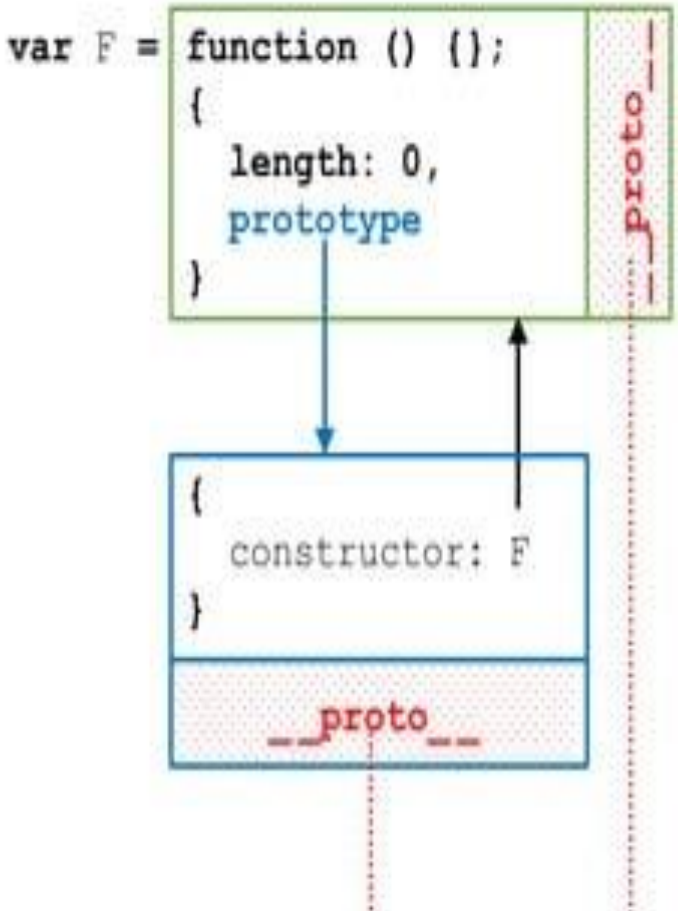
```
console.log(joe.__proto__.__proto__);
// null
```

prototype property

```
var F = function () {};
```

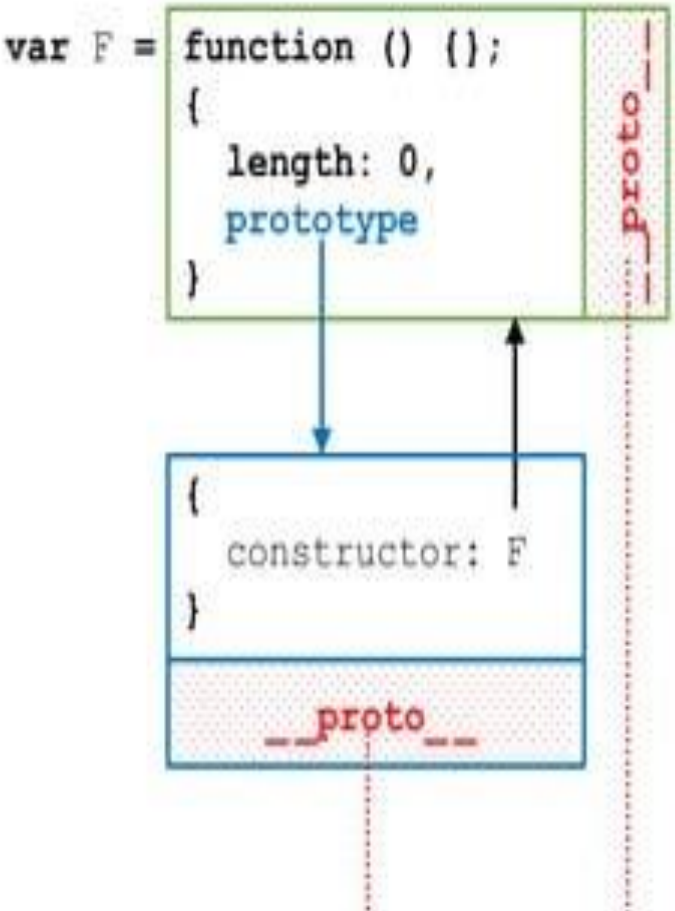
prototype property

```
var F = function () {};
```



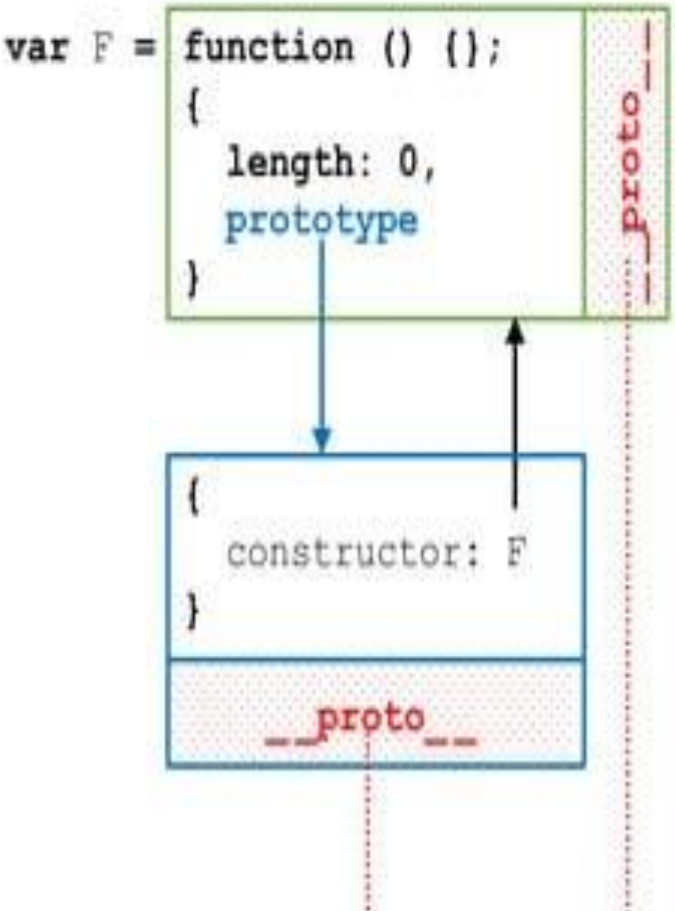
prototype property

```
var F = function () {};
console.log(typeof F);
// function
```



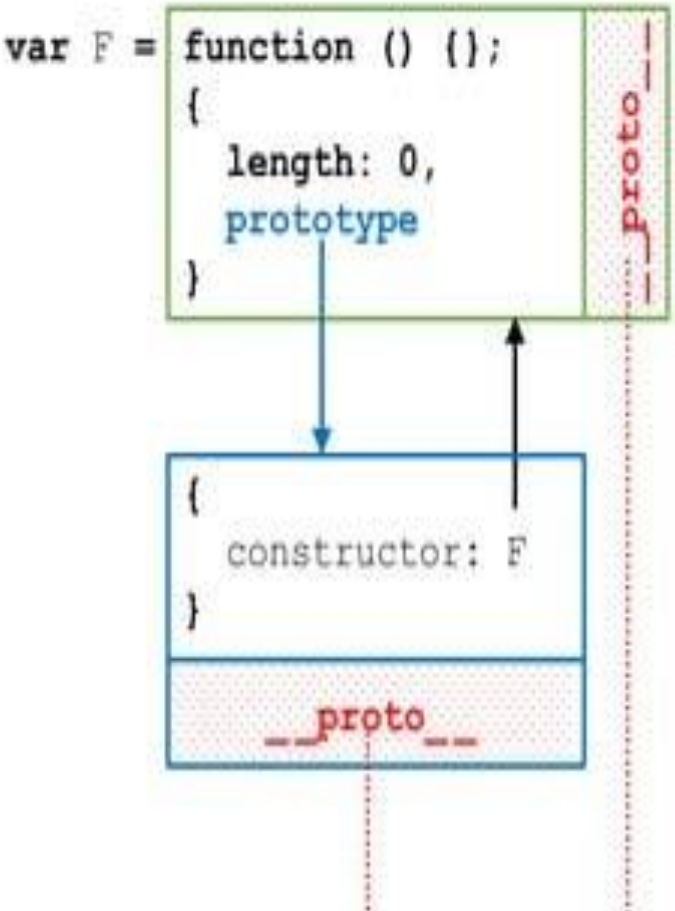
prototype property

```
var F = function () {};
console.log(typeof F);
// function
console.log(typeof F.prototype);
// object
```



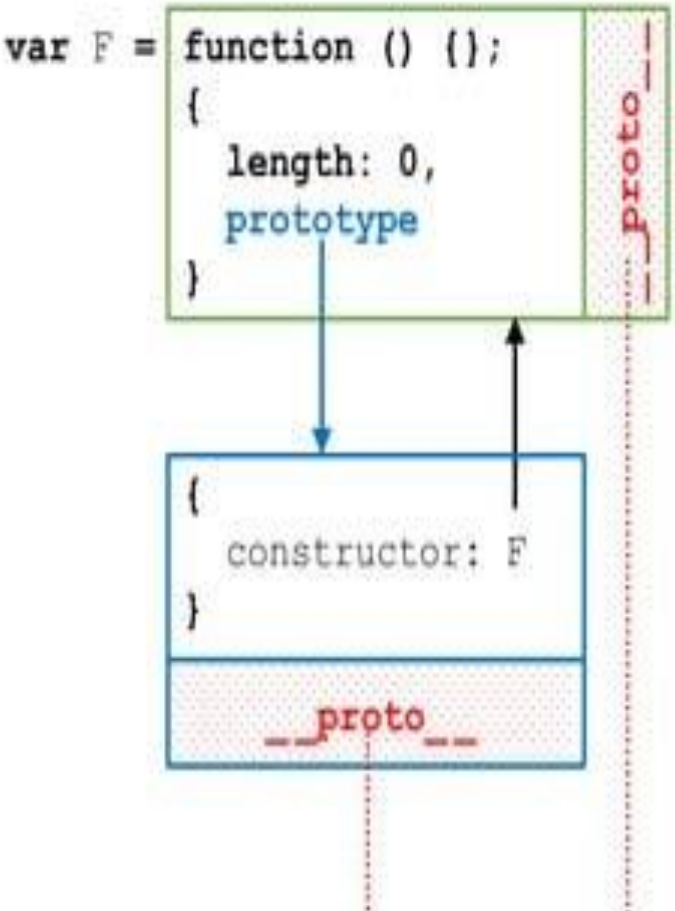
prototype property

```
var F = function () {};
console.log(typeof F);
// function
console.log(typeof F.prototype);
// object
console.log(F.prototype.constructor === F);
// true
```



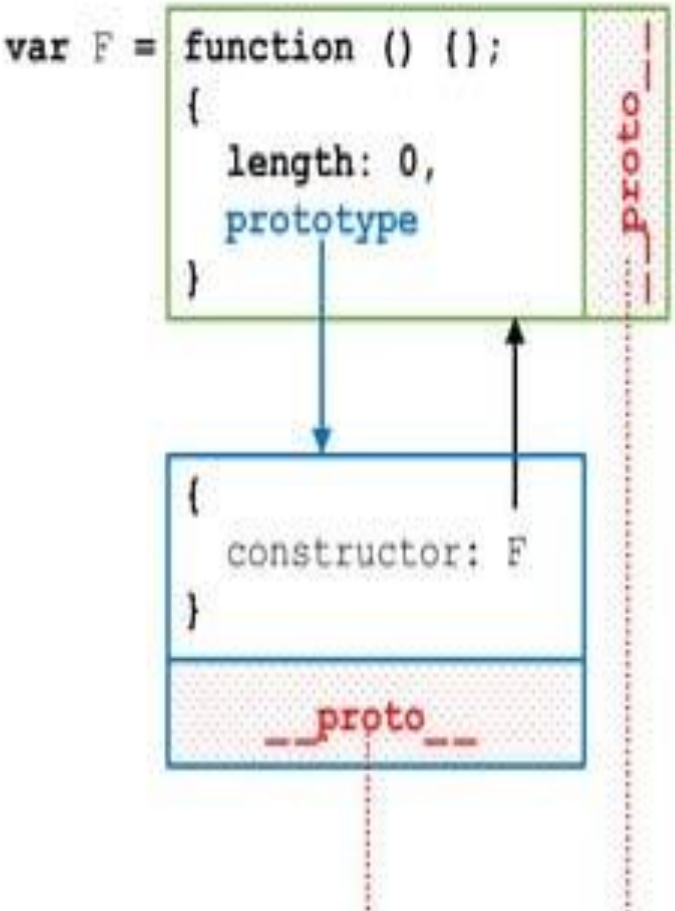
prototype property

```
var F = function () {};
console.log(typeof F);
// function
console.log(typeof F.prototype);
// object
console.log(F.prototype.constructor === F);
// true
console.log(F.prototype.prototype);
// undefined
```



prototype property

```
var F = function () {};
console.log(typeof F);
// function
console.log(typeof F.prototype);
// object
console.log(F.prototype.constructor === F);
// true
console.log(F.prototype.prototype);
// undefined
console.log(F.prototype !== F.__proto__);
// true
```

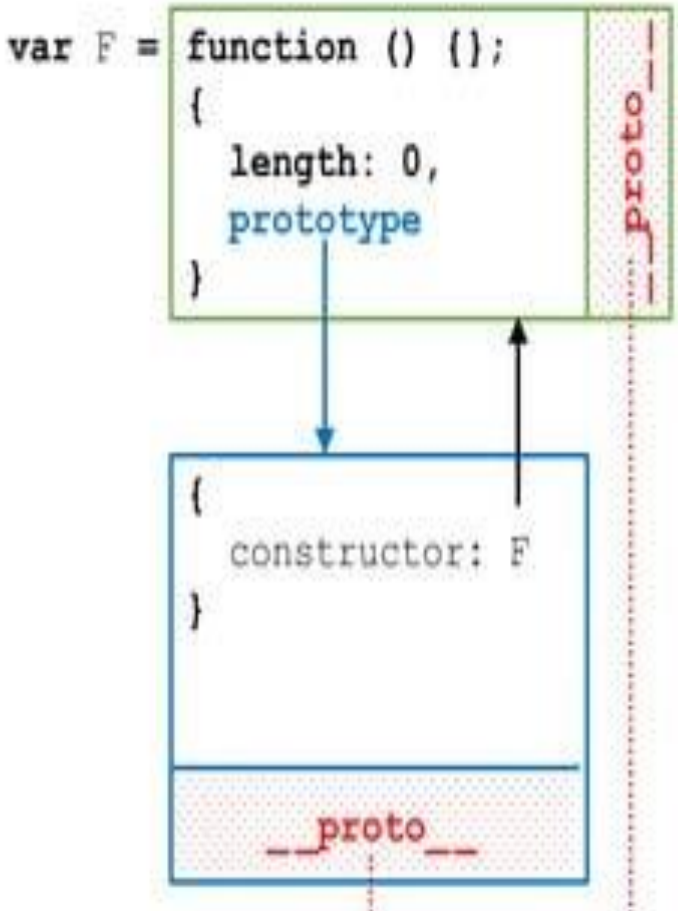


prototype property & operator new

```
var F = function () {};
```

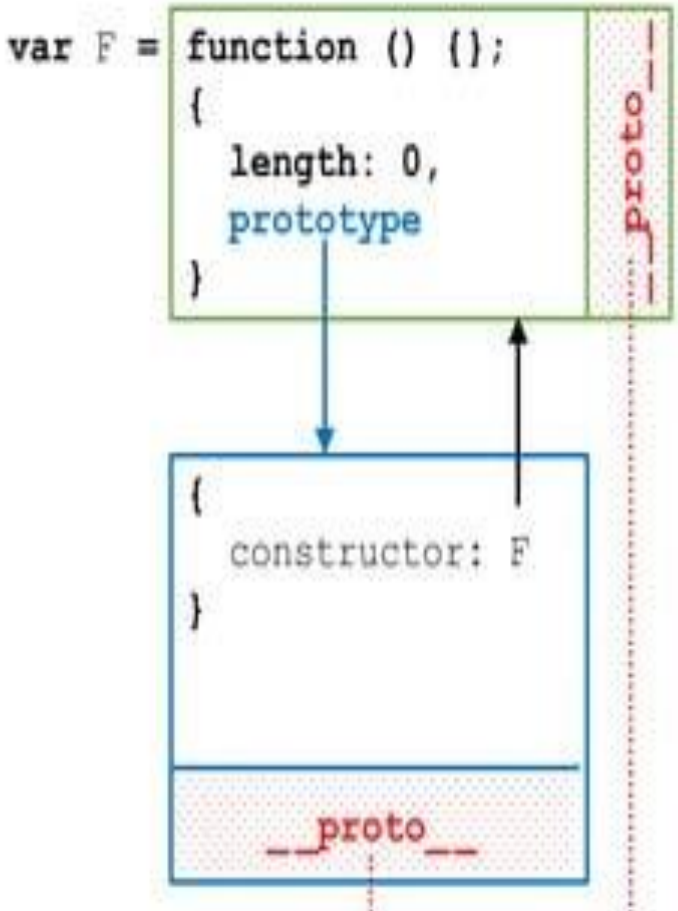
prototype property & operator new

```
var F = function () {};
```



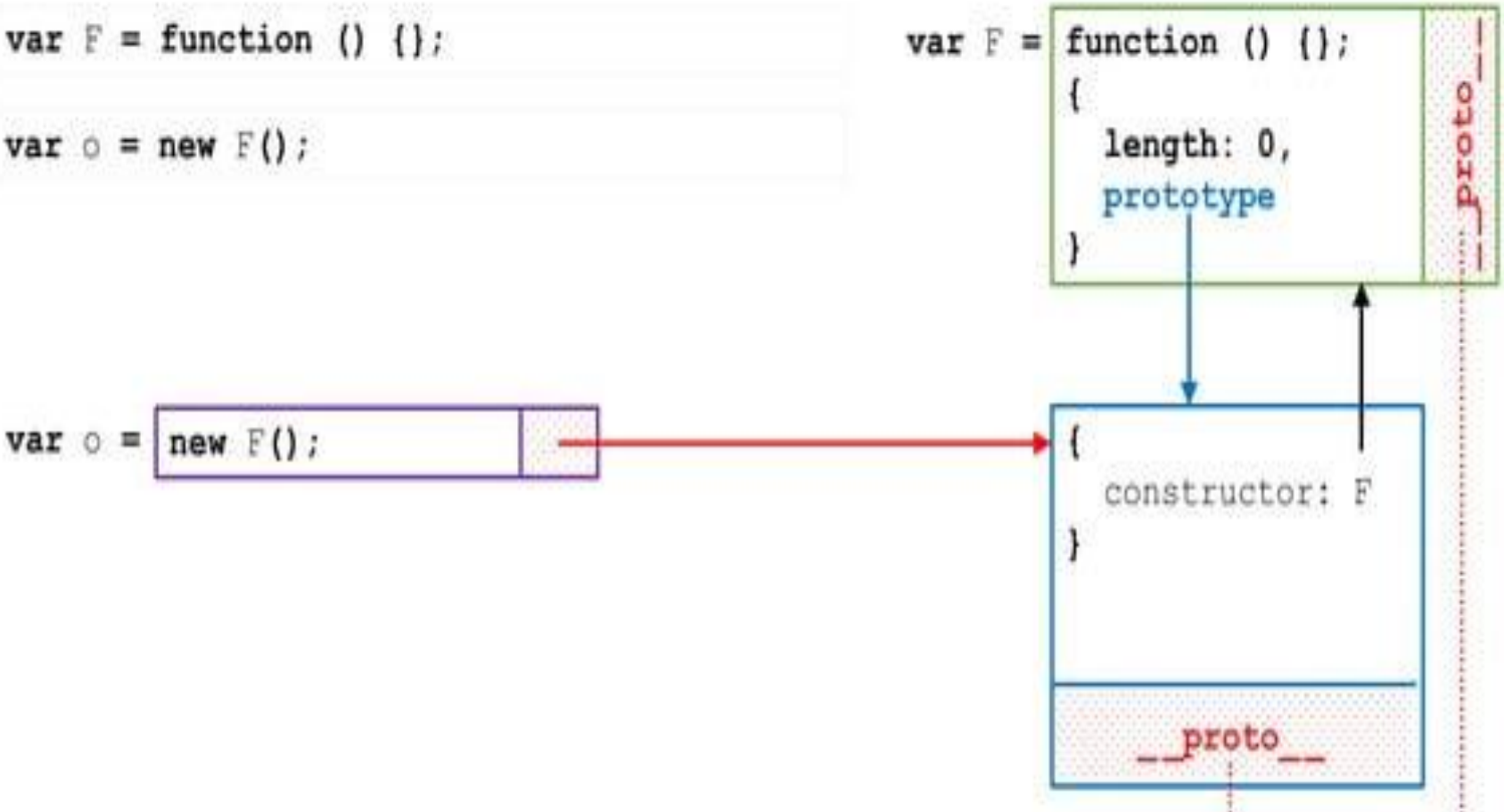
prototype property & operator new

```
var F = function () {};

var o = new F();
```



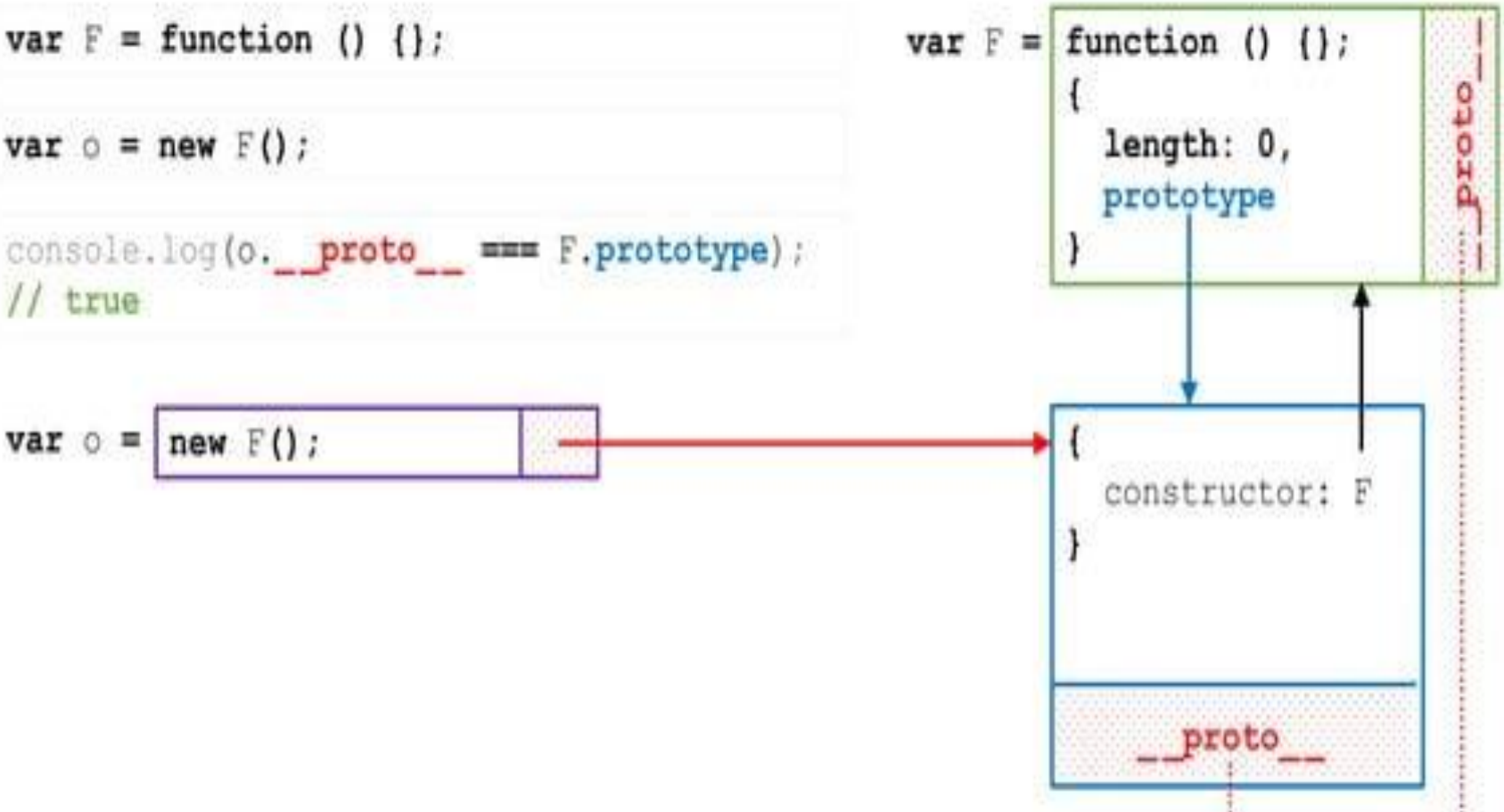
prototype property & operator new

```
var F = function () {};

var o = new F();
```



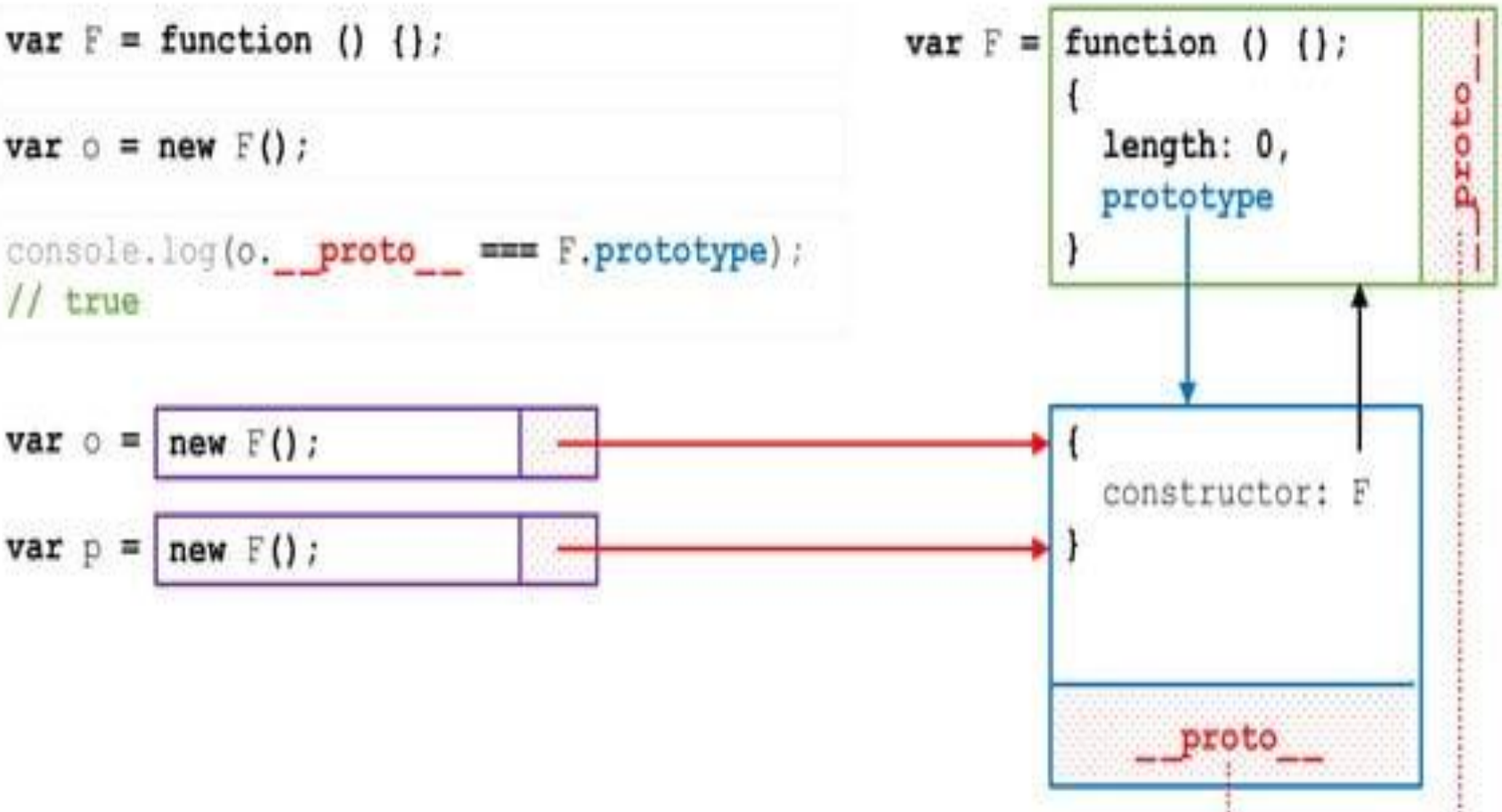
prototype property & operator new

```
var F = function () {};

var o = new F();

console.log(o.__proto__ === F.prototype);
// true
```



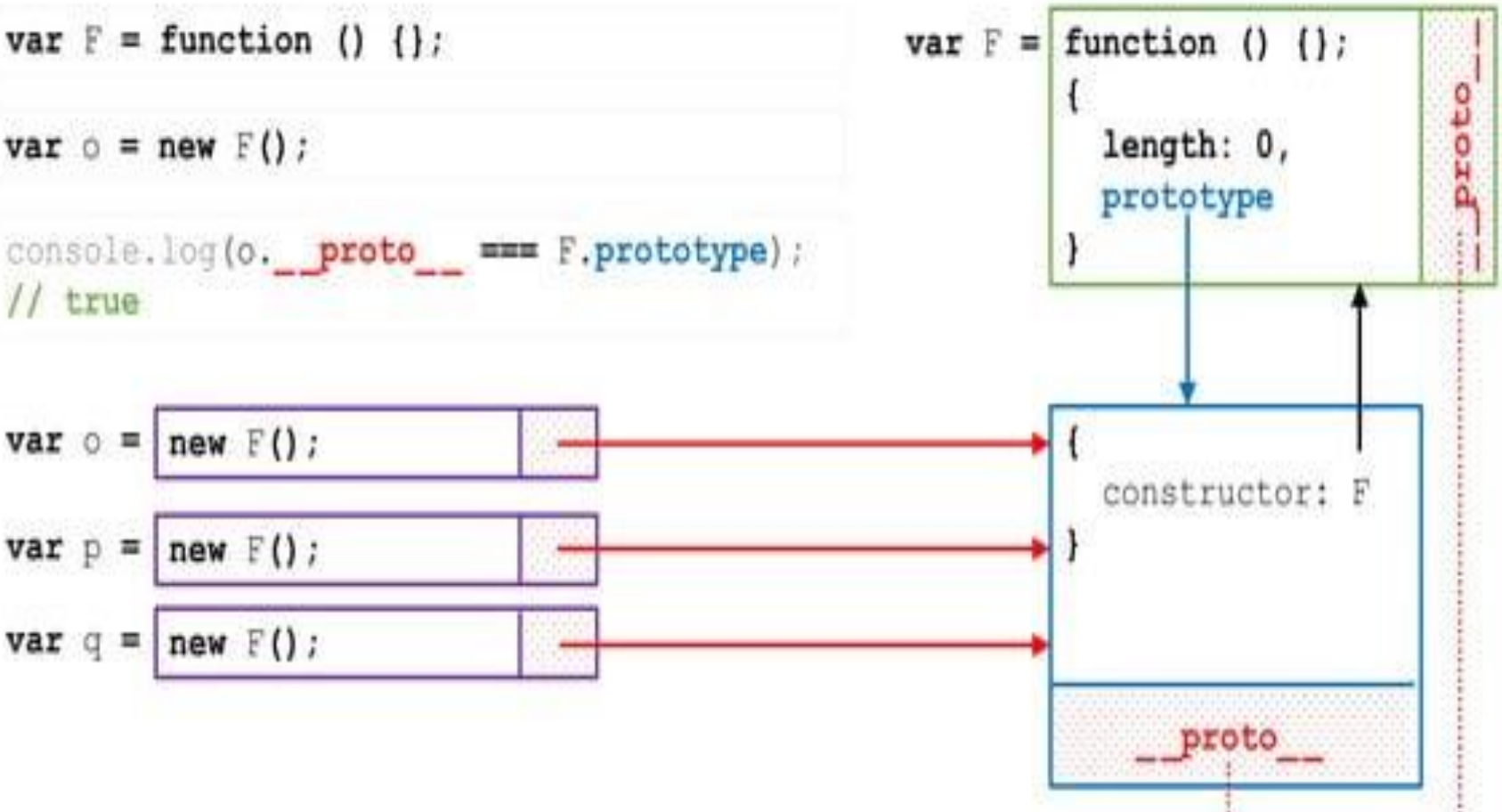
prototype property & operator new

```
var F = function () {};

var o = new F();

console.log(o.__proto__ === F.prototype);
// true
```



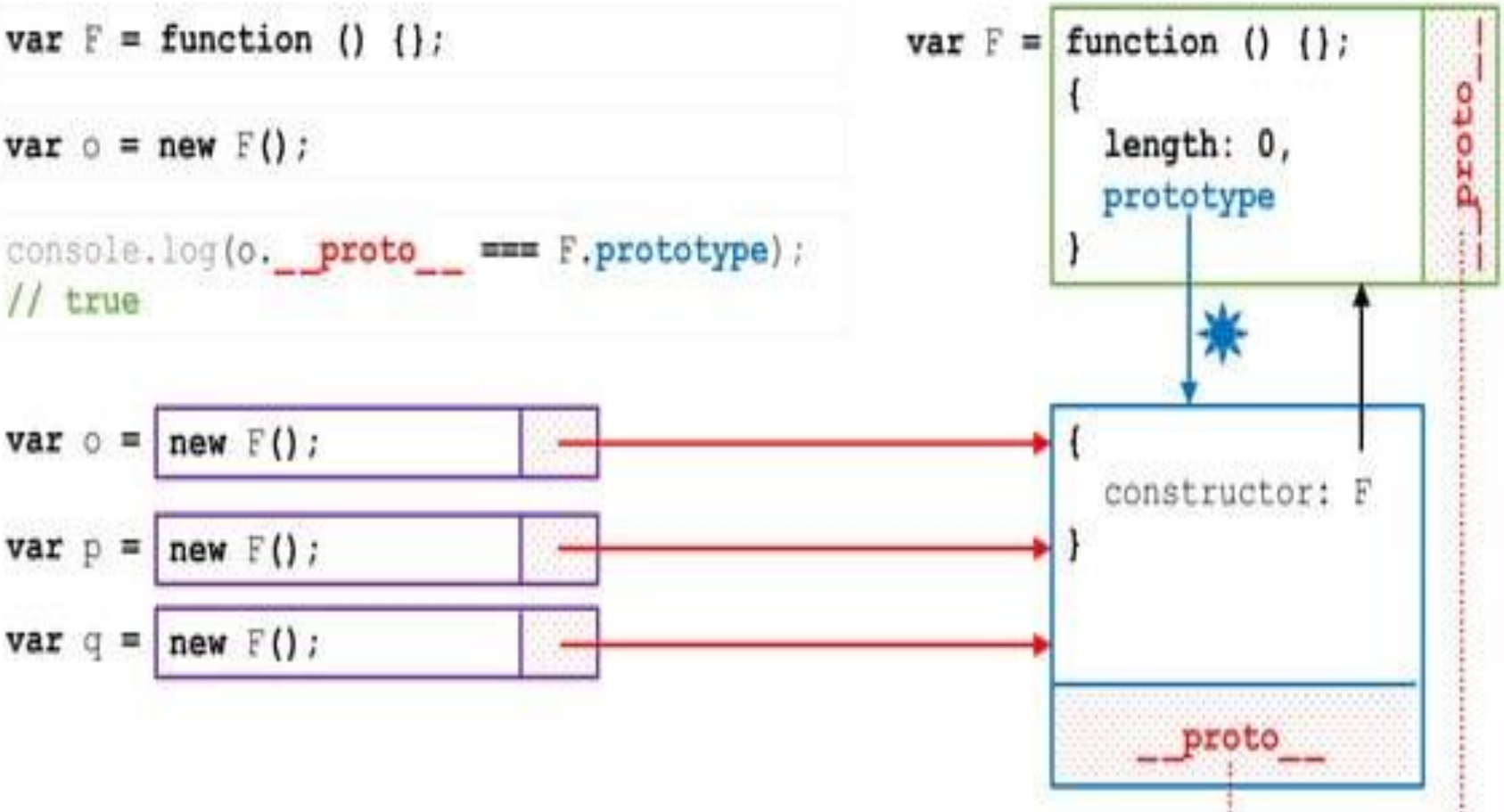
prototype property & operator new

```
var F = function () {};

var o = new F();

console.log(o.__proto__ === F.prototype);
// true
```



prototype property & operator new

```
var F = function () {};

var o = new F();

console.log(o.__proto__ === F.prototype);
// true
```

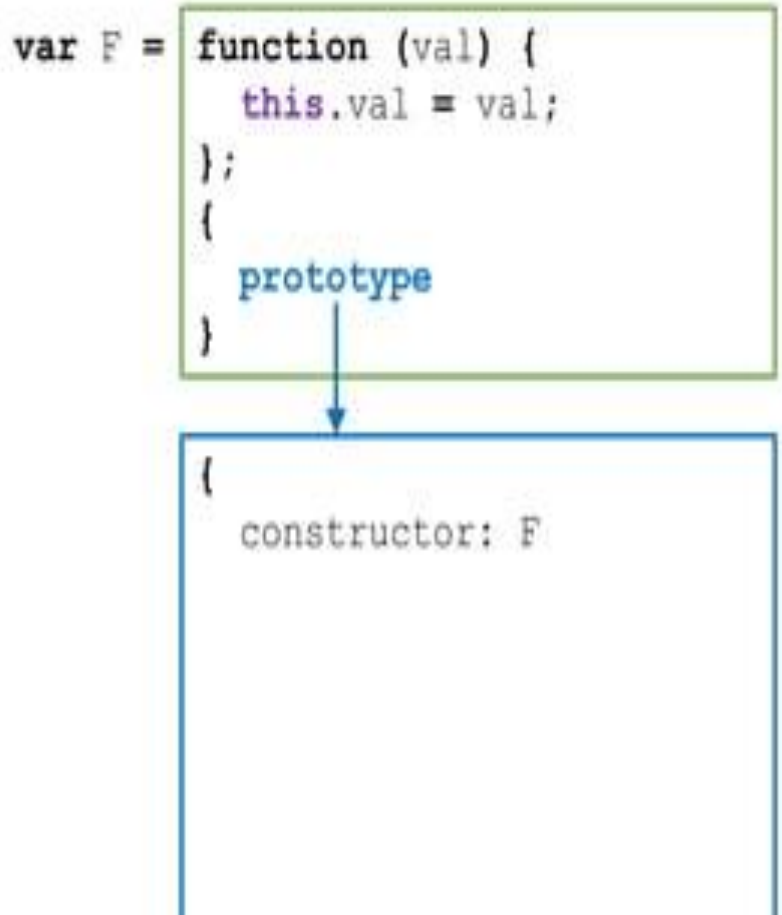


prototype property & operator new

```
var F = function (val) {
  this.val = val;
};
```

prototype property & operator new

```
var F = function (val) {
  this.val = val;
};
```



prototype property & operator new

```
var F = function (val) {
  this.val = val;
};

F.prototype.print = function () {
  console.log(this.val);
};
```

```
var F = function (val) {
  this.val = val;
};
{
  prototype
}
```

```
{
  constructor: F,

  print: function () {
    console.log(this.val);
  }
```

prototype property & operator new

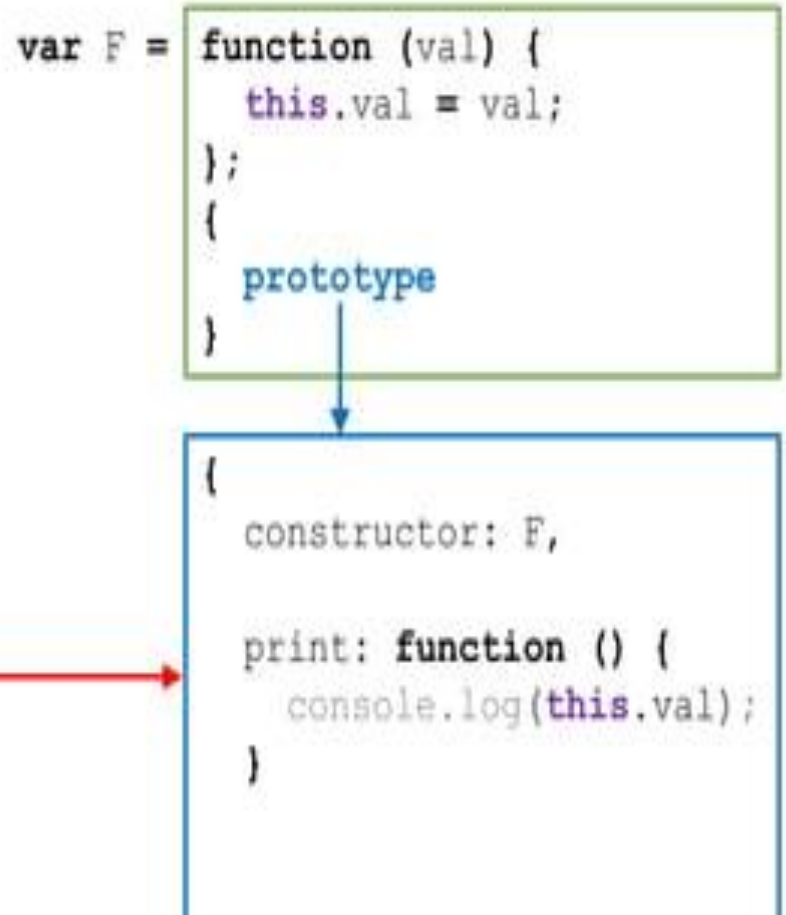
```
var F = function (val) {
  this.val = val;
};

F.prototype.print = function () {
  console.log(this.val);
};

var objX = new F(11);
```

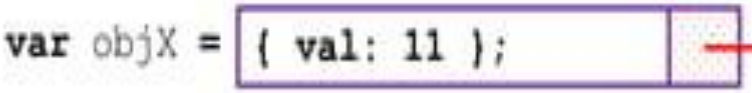
```
var F = function (val) {
  this.val = val;
};
{
  prototype
}
```

```
{
  constructor: F,

  print: function () {
    console.log(this.val);
  }
```

prototype property & operator new

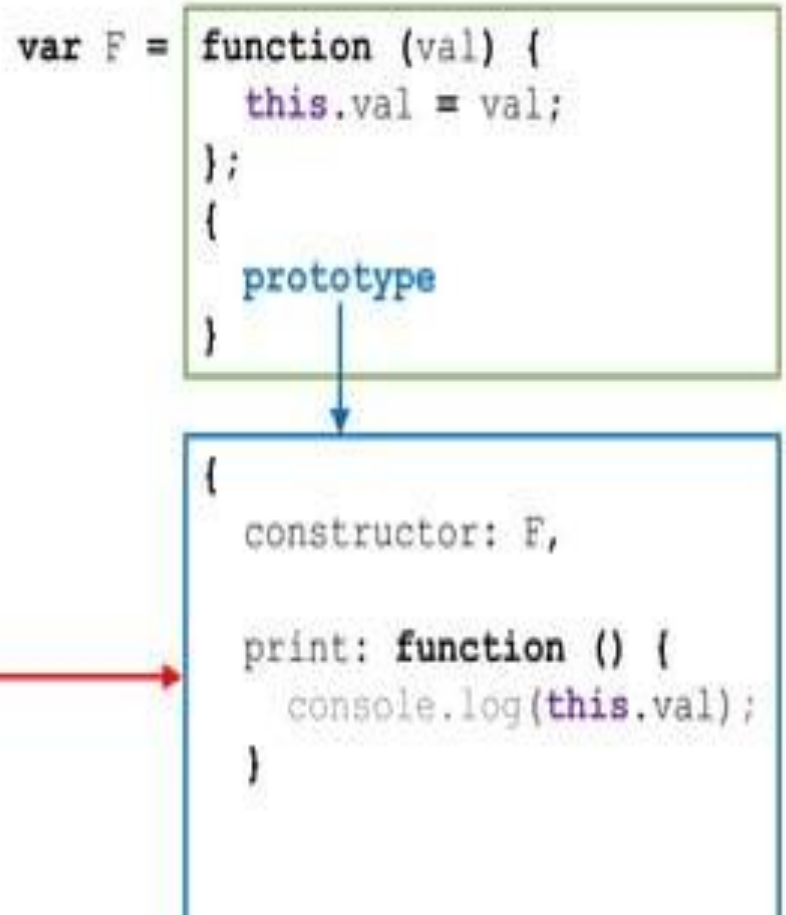
```
var F = function (val) {
  this.val = val;
};

F.prototype.print = function () {
  console.log(this.val);
};

var objX = new F(11);
```





prototype property & operator new

```
var F = function (val) {
  this.val = val;
};

F.prototype.print = function () {
  console.log(this.val);
};

var objX = new F(11);

objX.print(); // 11

var objX = { val: 11 };
```



prototype property & operator new

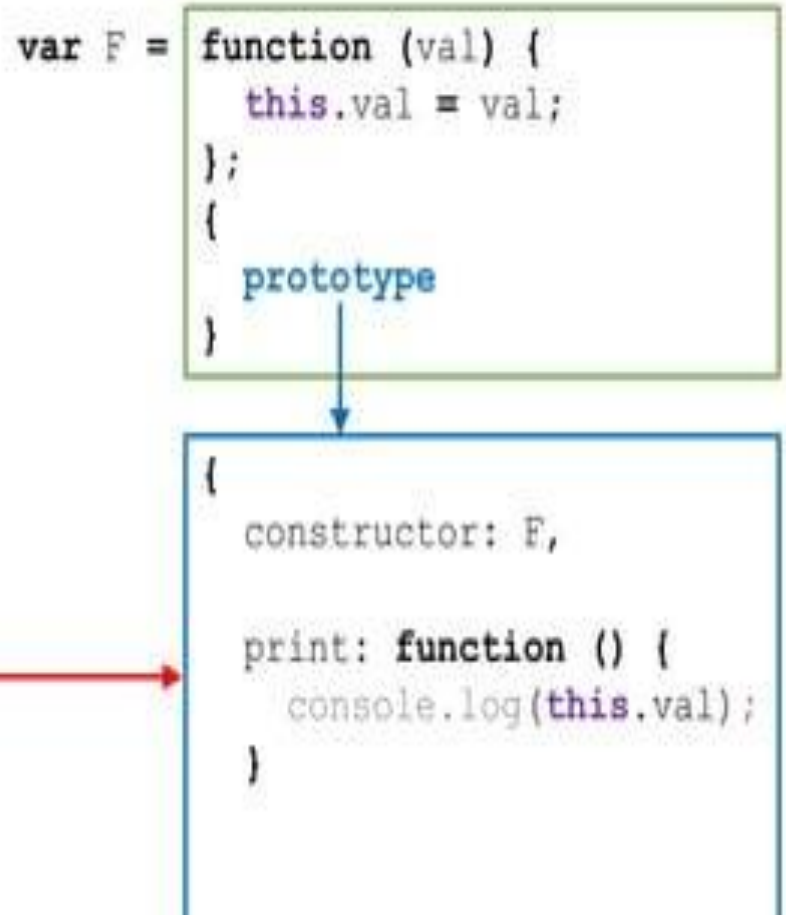
```
var F = function (val) {
  this.val = val;
};

F.prototype.print = function () {
  console.log(this.val);
};

var objX = new F(11);

objX.print(); // 11

var objX = { val: 11 };

var objY = new F(45);
```

```
var F = function (val) {
  this.val = val;
};
{
  prototype
}
```

```
{
  constructor: F,

  print: function () {
    console.log(this.val);
  }
```

prototype property & operator new

```
var F = function (val) {
  this.val = val;
};

F.prototype.print = function () {
  console.log(this.val);
};

var objX = new F(11);

objX.print(); // 11

var objX = { val: 11 };

var objY = new F(45);
```



prototype property & operator new

```
var F = function (val) {
  this.val = val;
};

F.prototype.print = function () {
  console.log(this.val);
};

var objX = new F(11);

objX.print(); // 11

var objX = { val: 11 };

var objY = new F(45);

objY.print(); // 45
```

```
var F = function (val) {
  this.val = val;
};
{
  prototype
}
```

```
{
  constructor: F,

  print: function () {
    console.log(this.val);
  }
```

this?

this?

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);                    // new object
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);          // new object

f(args);              // window
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);                  // new object

f(args);                      // window

f.call(context, args);        // context object
f.apply(context, [args]);     // context object
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);                // new object

f(args);                    // window

f.call(context, args);      // context object
f.apply(context, [args]);   // context object

f.call(null, args);         // window
f.call(undefined, args);    // window
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);                // new object

f(args);                    // window

f.call(context, args);      // context object
f.apply(context, [args]);   // context object

f.call(null, args);         // window
f.call(undefined, args);    // window

f.call(42, args);           // new Number(42)
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);                // new object

f(args);                    // window

f.call(context, args);      // context object
f.apply(context, [args]);   // context object

f.call(null, args);         // window
f.call(undefined, args);    // window

f.call(42, args);           // new Number(42)

obj.m(args);                // obj
array[0](args);             // array
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);                // new object

f(args);                    // window

f.call(context, args);      // context object
f.apply(context, [args]);   // context object

f.call(null, args);         // window
f.call(undefined, args);    // window

f.call(42, args);           // new Number(42)

obj.m(args);                // obj
array[0](args);             // array

var g = obj.m;
g(args);
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);                // new object

f(args);                    // window

f.call(context, args);      // context object
f.apply(context, [args]);   // context object

f.call(null, args);         // window
f.call(undefined, args);    // window

f.call(42, args);           // new Number(42)

obj.m(args);                // obj
array[0](args);             // array

var g = obj.m;
g(args);                    // window
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

this?

```
new f(args);                // new object

f(args);                    // window

f.call(context, args);      // context object
f.apply(context, [args]);   // context object

f.call(null, args);         // window
f.call(undefined, args);    // window

f.call(42, args);           // new Number(42)

obj.m(args);                // obj
array[0](args);             // array

var g = obj.m;
g(args);                    // window
```

```
var f = function (args) {
  console.log(this);
};

var obj = { m: f };

var array = [ f ];
```

Something in `return`?

Something in `return`?

```
var f = function () {
  return; // or w/o return
};
```

Something in `return`?

```
new f();    // new object
f();        // undefined
```

```
var f = function () {
  return; // or w/o return
};
```

Something in `return`?

```
new f();    // new object
f();        // undefined
```

```
var f = function () {
  return; // or w/o return
};
```

```
var g = function () {
  // return null, undefined,
  // or a primitive value
  return -1;
};
```

Something in `return`?

```
new f();    // new object
f();        // undefined
```

```
var f = function () {
  return; // or w/o return
};
```

```
new g();    // new object
g();        // -1 (null, undefined or primitive)
```

```
var g = function () {
  // return null, undefined,
  // or a primitive value
  return -1;
};
```

Something in `return`?

```
new f();    // new object
f();        // undefined
```

```
var f = function () {
  return; // or w/o return
};
```

```
new g();    // new object
g();        // -1 (null, undefined or primitive)
```

```
var g = function () {
  // return null, undefined,
  // or a primitive value
  return -1;
};
```

```
var h = function () {
  return { val: 42 };
};
```

Something in `return`?

```
new f();    // new object
f();        // undefined
```

```
var f = function () {
  return; // or w/o return
};
```

```
new g();    // new object
g();        // -1 (null, undefined or primitive)
```

```
var g = function () {
  // return null, undefined,
  // or a primitive value
  return -1;
};
```

```
new h();    // { val: 42 }
h();        // { val: 42 }
```

```
var h = function () {
  return { val: 42 };
};
```

Something in `return`?

```
new f();    // new object
f();        // undefined
```

```
var f = function () {
  return; // or w/o return
};
```

```
new g();    // new object
g();        // -1 (null, undefined or primitive)
```

```
var g = function () {
  // return null, undefined,
  // or a primitive value
  return -1;
};
```

```
new h();    // { val: 42 }
h();        // { val: 42 }
```

```
var h = function () {
  return { val: 42 };
};
```

```
var wtf = function () {
  return
```

Something in `return`?

```
new f();    // new object
f();        // undefined
```

```
var f = function () {
  return; // or w/o return
};
```

```
new g();    // new object
g();        // -1 (null, undefined or primitive)
```

```
var g = function () {
  // return null, undefined,
  // or a primitive value
  return -1;
};
```

```
new h();    // { val: 42 }
h();        // { val: 42 }
```

```
var h = function () {
  return { val: 42 };
};
```

```
new wtf(); // new object
wtf();     // undefined
```

```
var wtf = function () {
  return
```

Something in `return`?

```
new f();    // new object
f();        // undefined
```

```
var f = function () {
  return; // or w/o return
};
```

```
new g();    // new object
g();        // -1 (null, undefined or primitive)
```

```
var g = function () {
  // return null, undefined,
  // or a primitive value
  return -1;
};
```

```
new h();    // { val: 42 }
h();        // { val: 42 }
```

```
var h = function () {
  return { val: 42 };
};
```

```
new wtf(); // new object
wtf();     // undefined
```

```
var wtf = function () {
  return;
```

Inherit from `that`

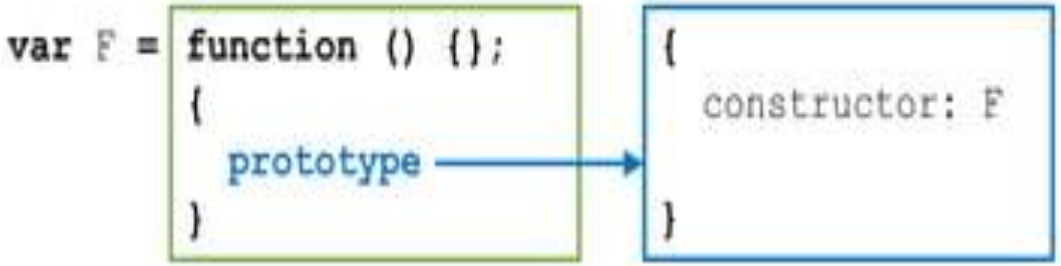
```
function create (that) {



}

var newObj = create(that);
```

Inherit from `that`

```
function create (that) {
  var F = function () {};

  return new F();
}

var newObj = create(that);
```

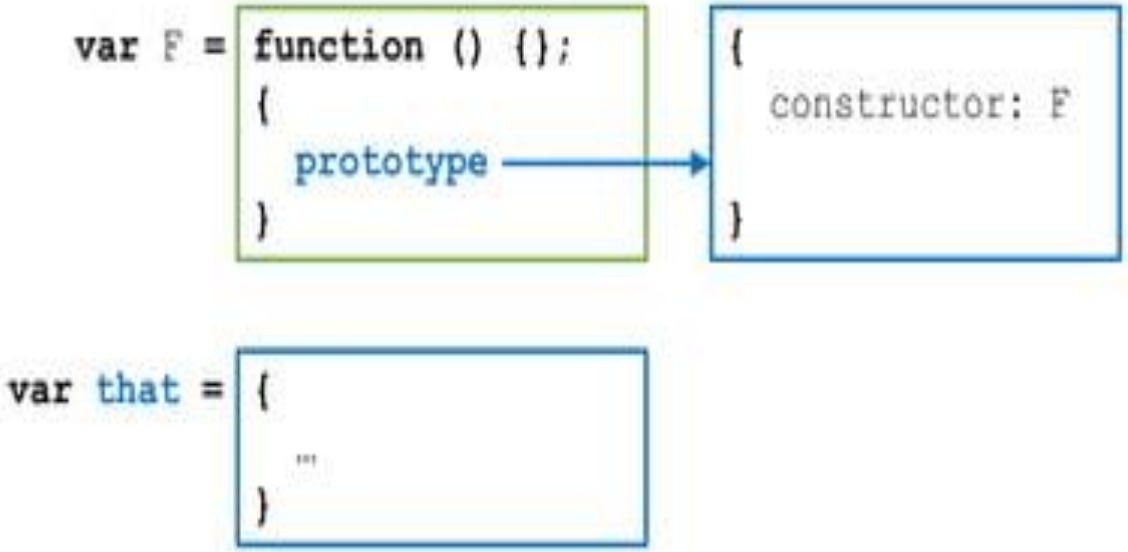
Inherit from `that`

```
function create (that) {
  var F = function () {};

  return new F();
}

var newObj = create(that);
```



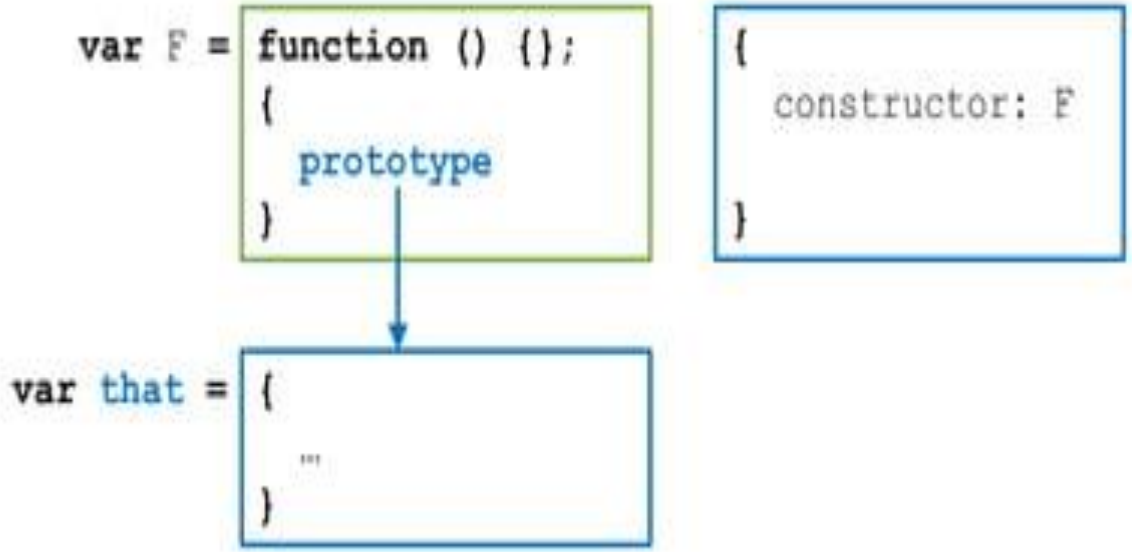
Inherit from that

```
function create (that) {
  var F = function () {};

  return new F();
}

var newObj = create(that);
```



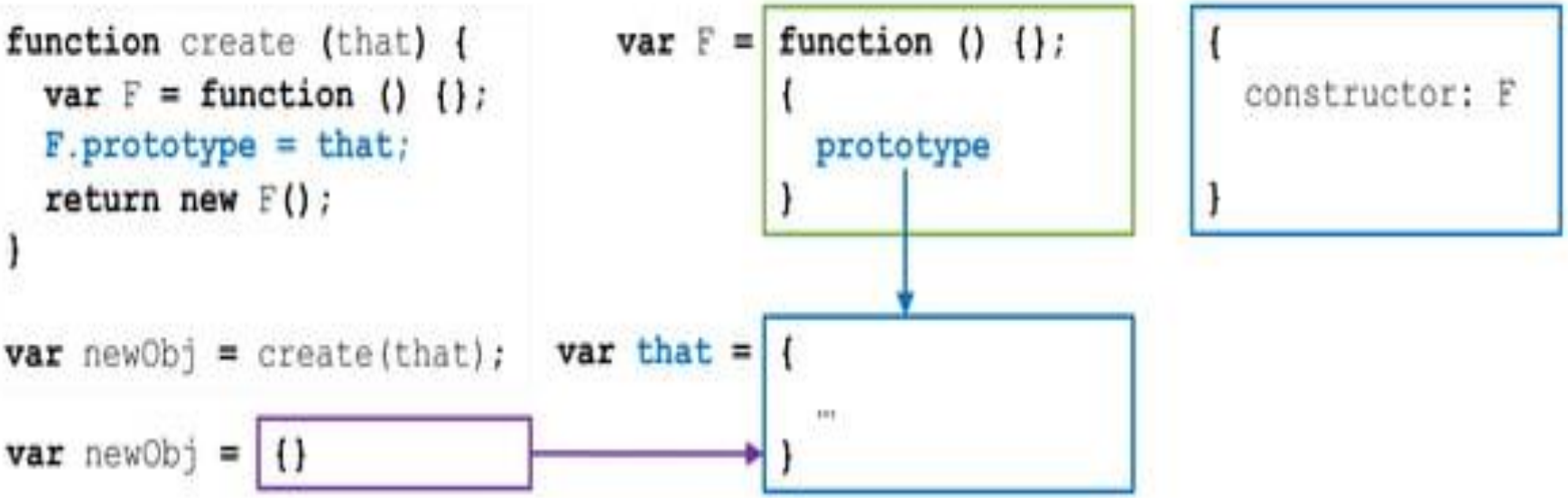
Inherit from `that`

```
function create (that) {
  var F = function () {};
  F.prototype = that;
  return new F();
}

var newObj = create(that);
```



Inherit from that

```
function create (that) {
  var F = function () {};
  F.prototype = that;
  return new F();
}

var newObj = create(that);
```



Inherit from that

```
function create (that) {
  var F = function () {};
  F.prototype = that;
  return new F();
}

var newObj = create(that);
```

```
var newObj = {}
```

```
var F = function () {};
{
  prototype
}
```

```
var that = {
  ...
}
```

```
{
  constructor: F
}
```

```
var newObj = Object.create(that);
```

Inherit from that

```
function create (that) {
  var F = function () {};
  F.prototype = that;
  return new F();
}

var newObj = create(that);
```

```
var newObj = {}
```

```
var F = function () {};
{
  prototype
}
```

```
var that = {

  ...
}
```

```
{
  constructor: F

}
```

```
var newObj = Object.create(that);
```

```
var newObj = Object.create(null);
```

instanceof

```
console.log(objX instanceof F);
```

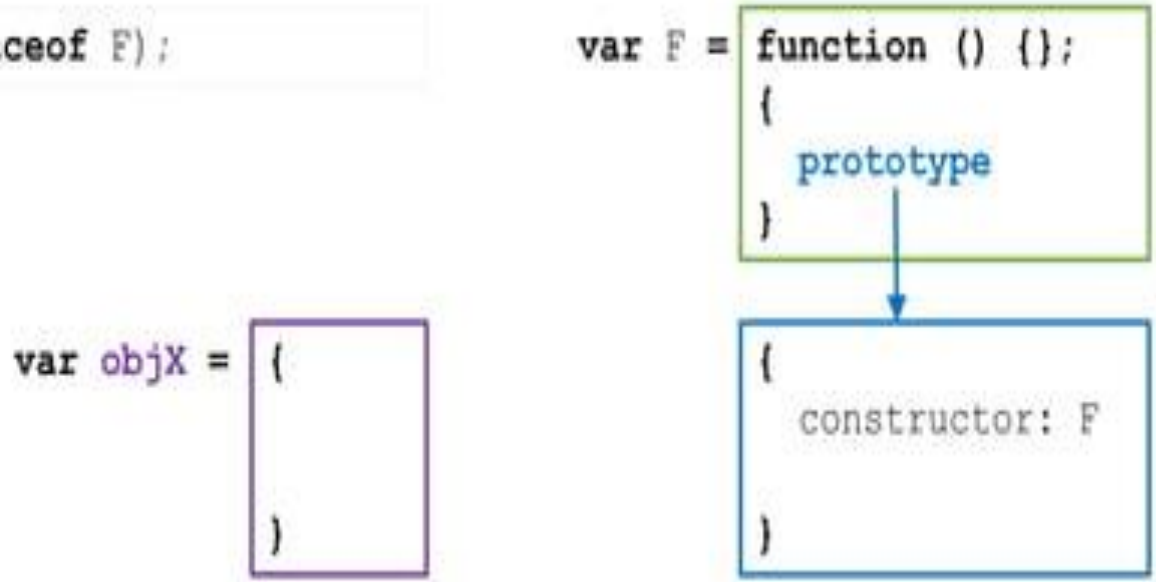
instanceof

```
console.log(objX instanceof F);
```

```
var objX = {

}
```

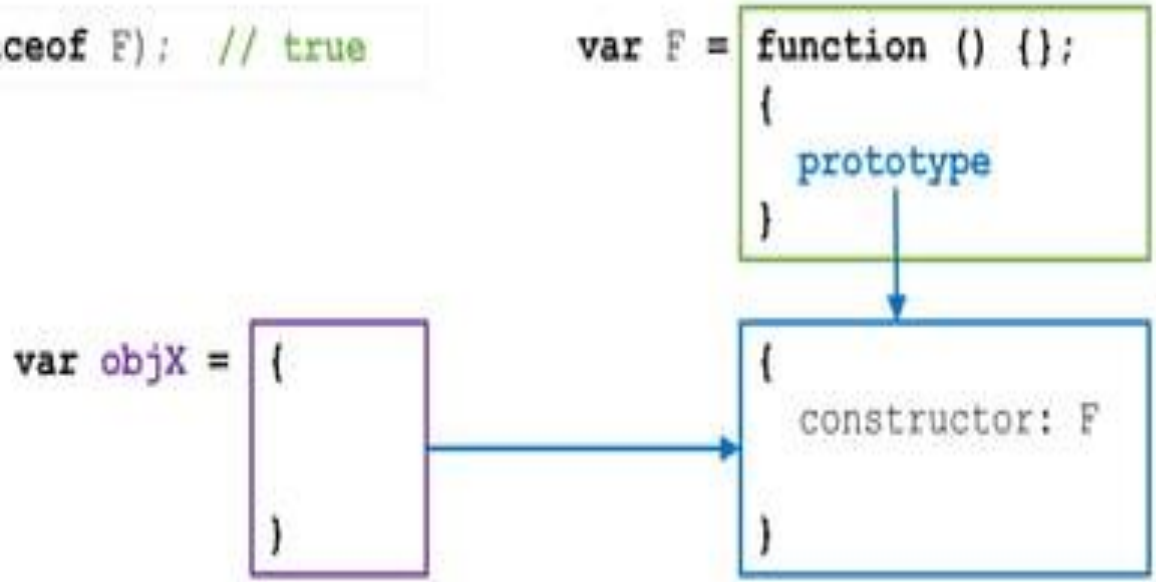
instanceof

```
console.log(objX instanceof F);
```



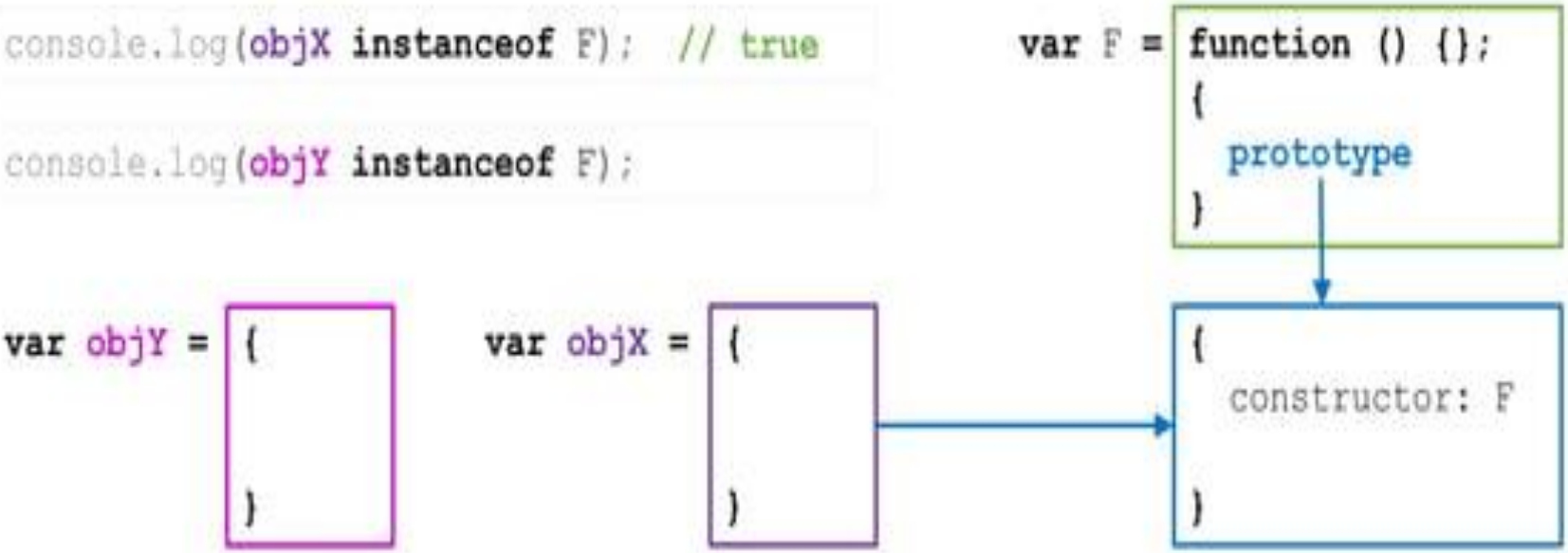
instanceof

```
console.log(objX instanceof F);  // true
```



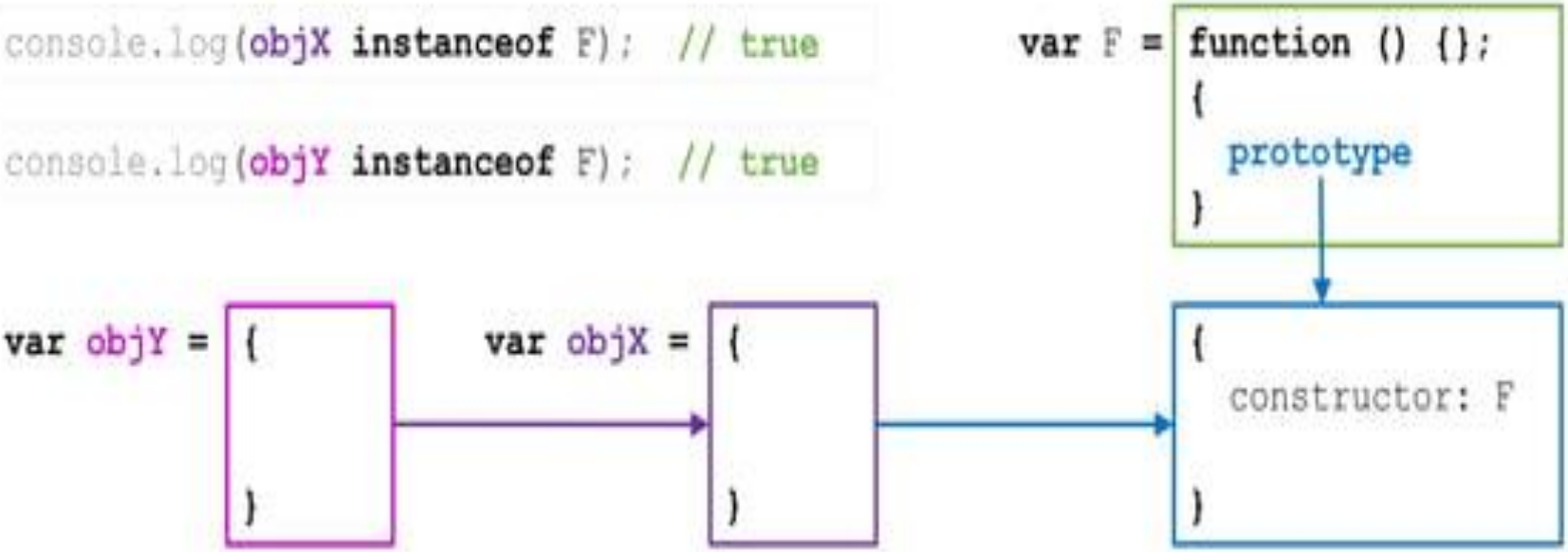
instanceof

```
console.log(objX instanceof F);  // true

console.log(objY instanceof F);
```



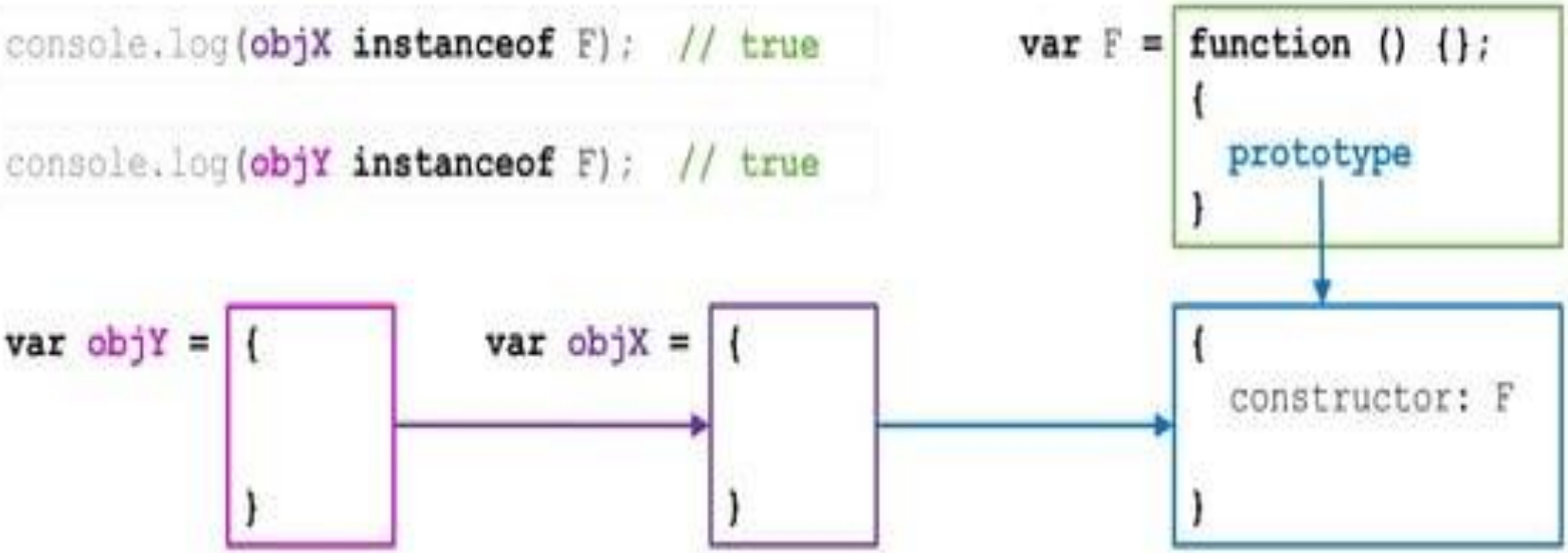
instanceof

```
console.log(objX instanceof F);  // true
console.log(objY instanceof F);  // true
```



instanceof

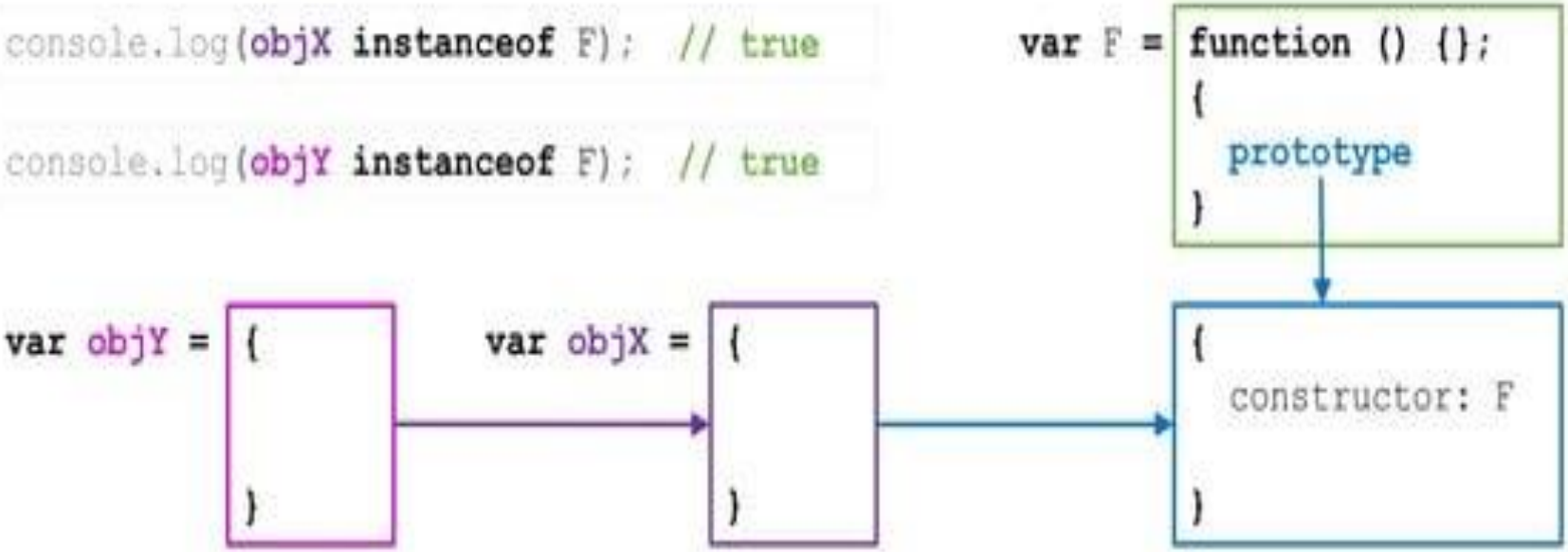
```
console.log(objX instanceof F);  // true
console.log(objY instanceof F);  // true
```




```
console.log(objX.isPrototypeOf(objY));  // true
```

instanceof

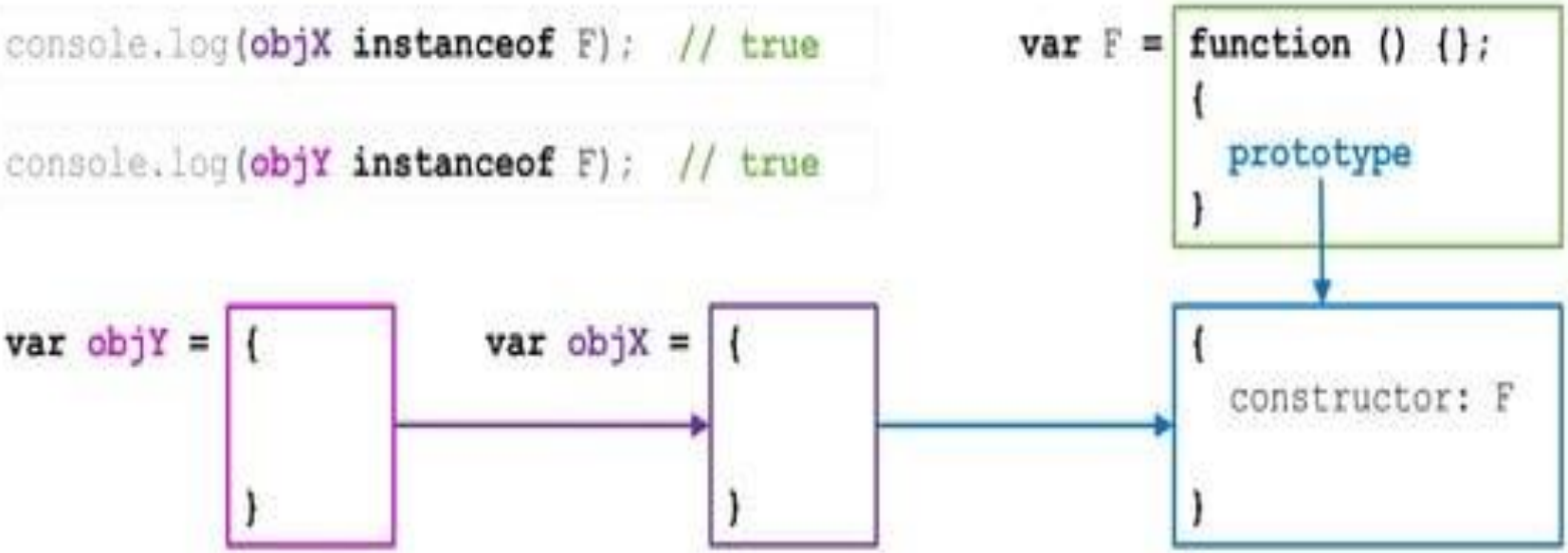
```
console.log(objX instanceof F);  // true
console.log(objY instanceof F);  // true
```




```
console.log(objX.isPrototypeOf(objY));          // true
console.log(F.prototype.isPrototypeOf(objY));   // true
```

instanceof

```
console.log(objX instanceof F);  // true
console.log(objY instanceof F);  // true
```




```
console.log(objX.isPrototypeOf(objY));          // true
console.log(F.prototype.isPrototypeOf(objY));   // true
```

Chicken or the egg?

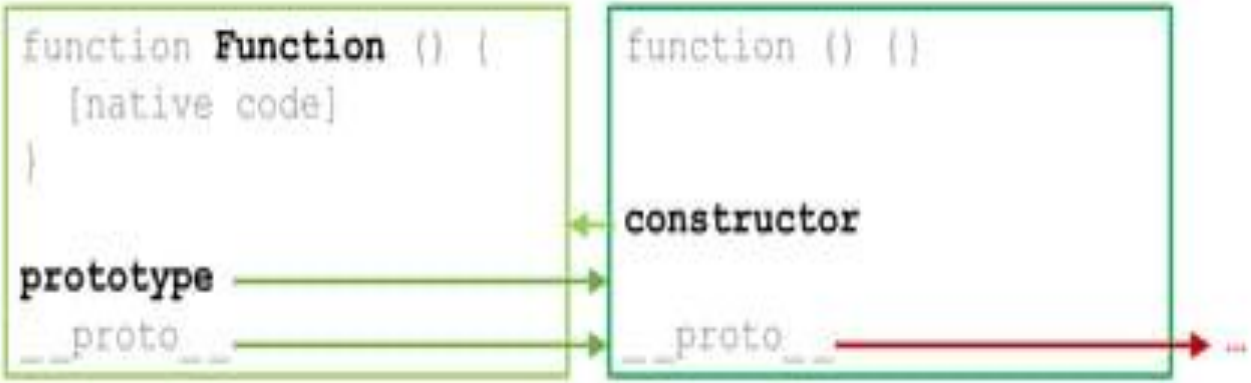
Chicken or the egg?

```
console.log(Function instanceof Object);        // true
```

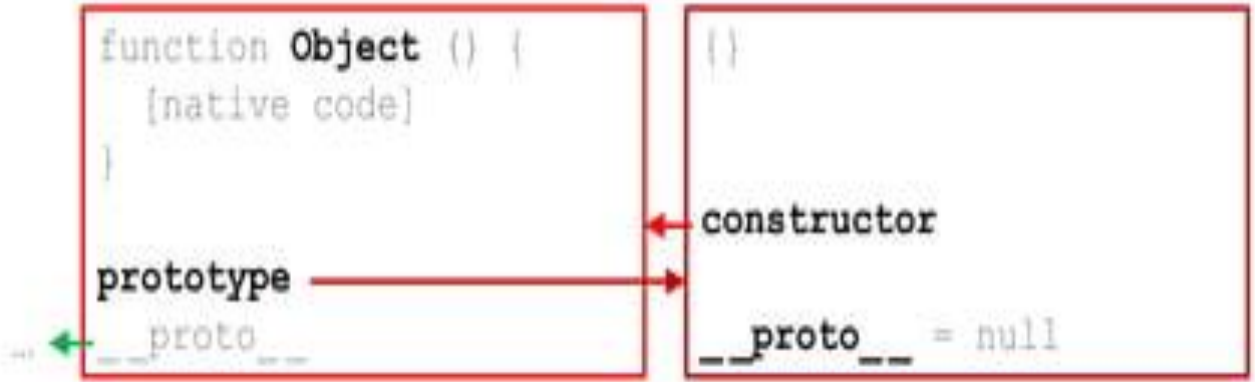
Chicken or the egg?

```
console.log(Function instanceof Object);        // true

console.log(Object instanceof Function);        // true
```

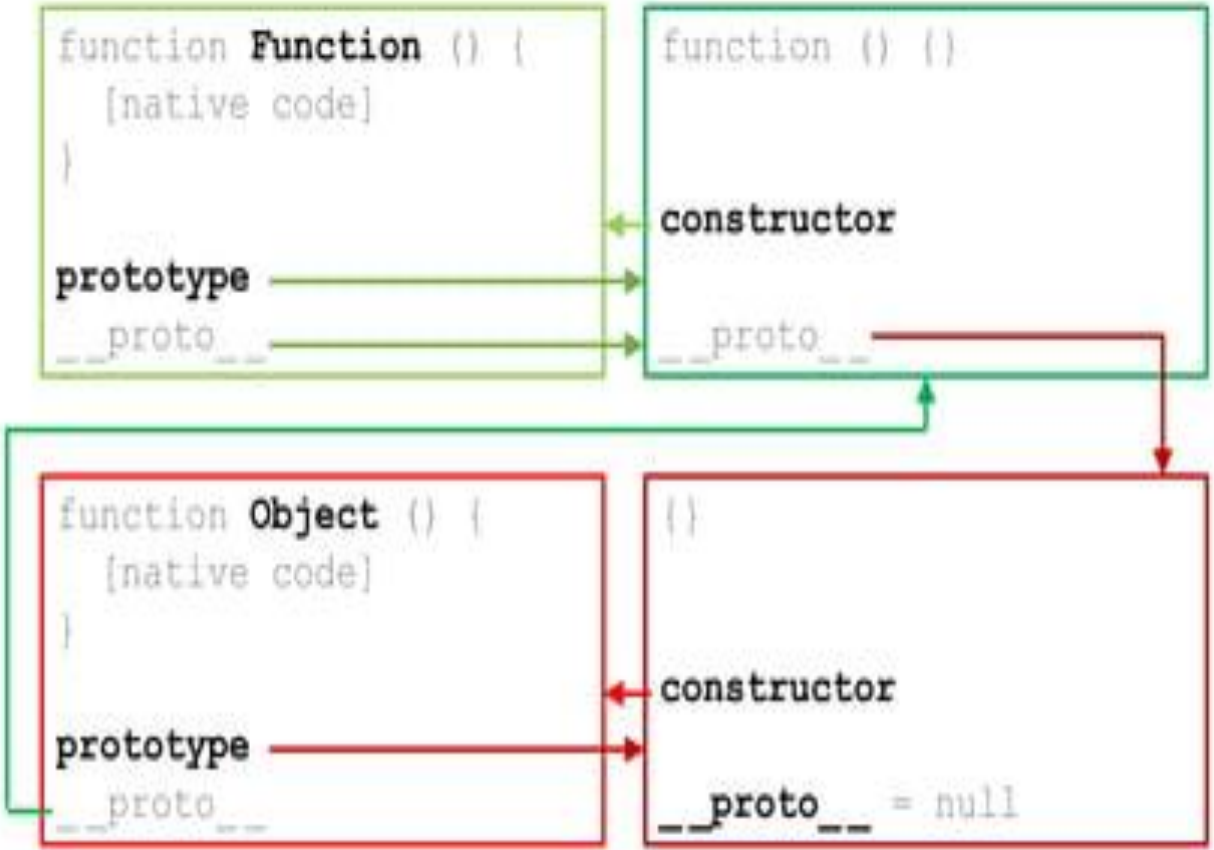
Chicken or the egg?

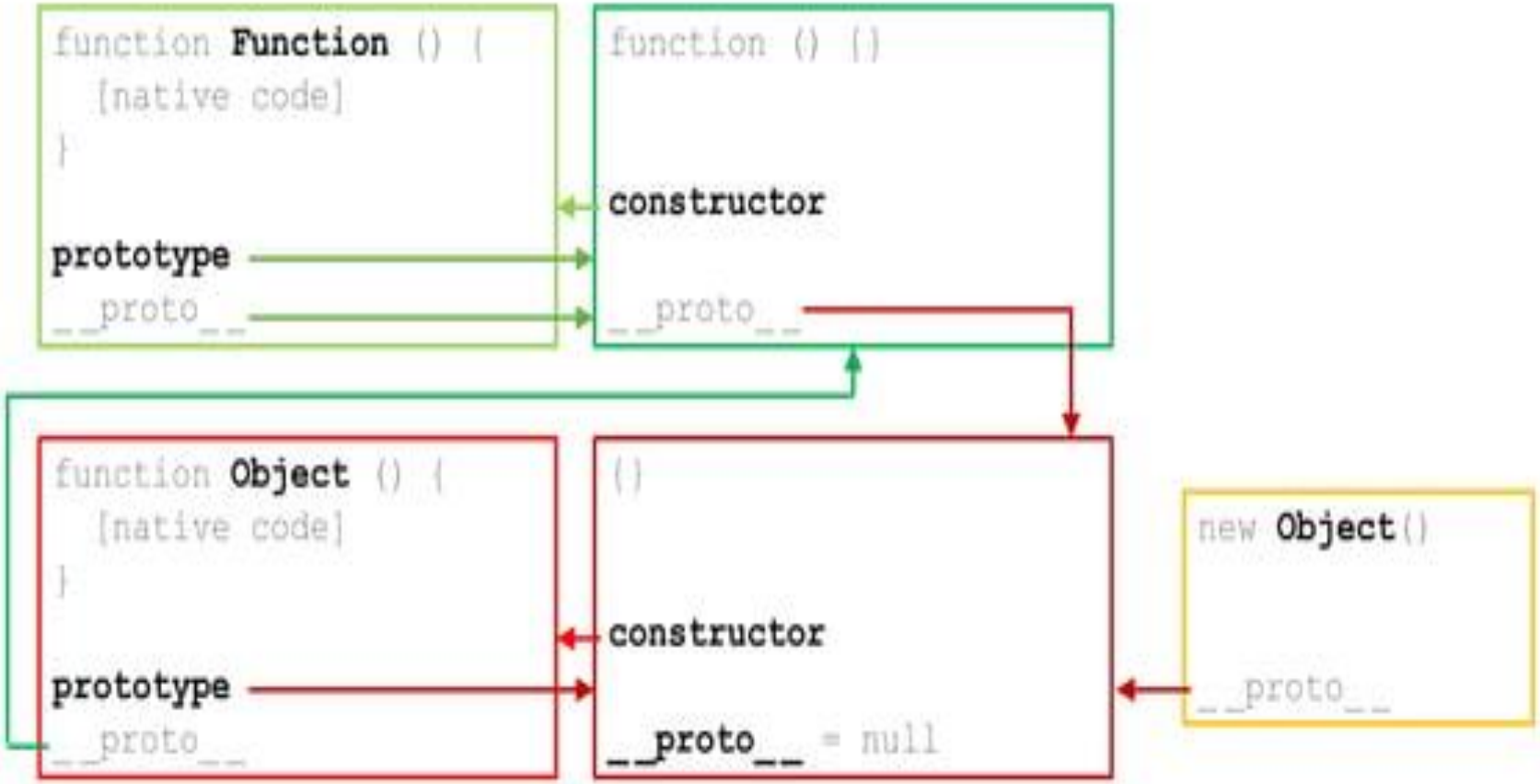


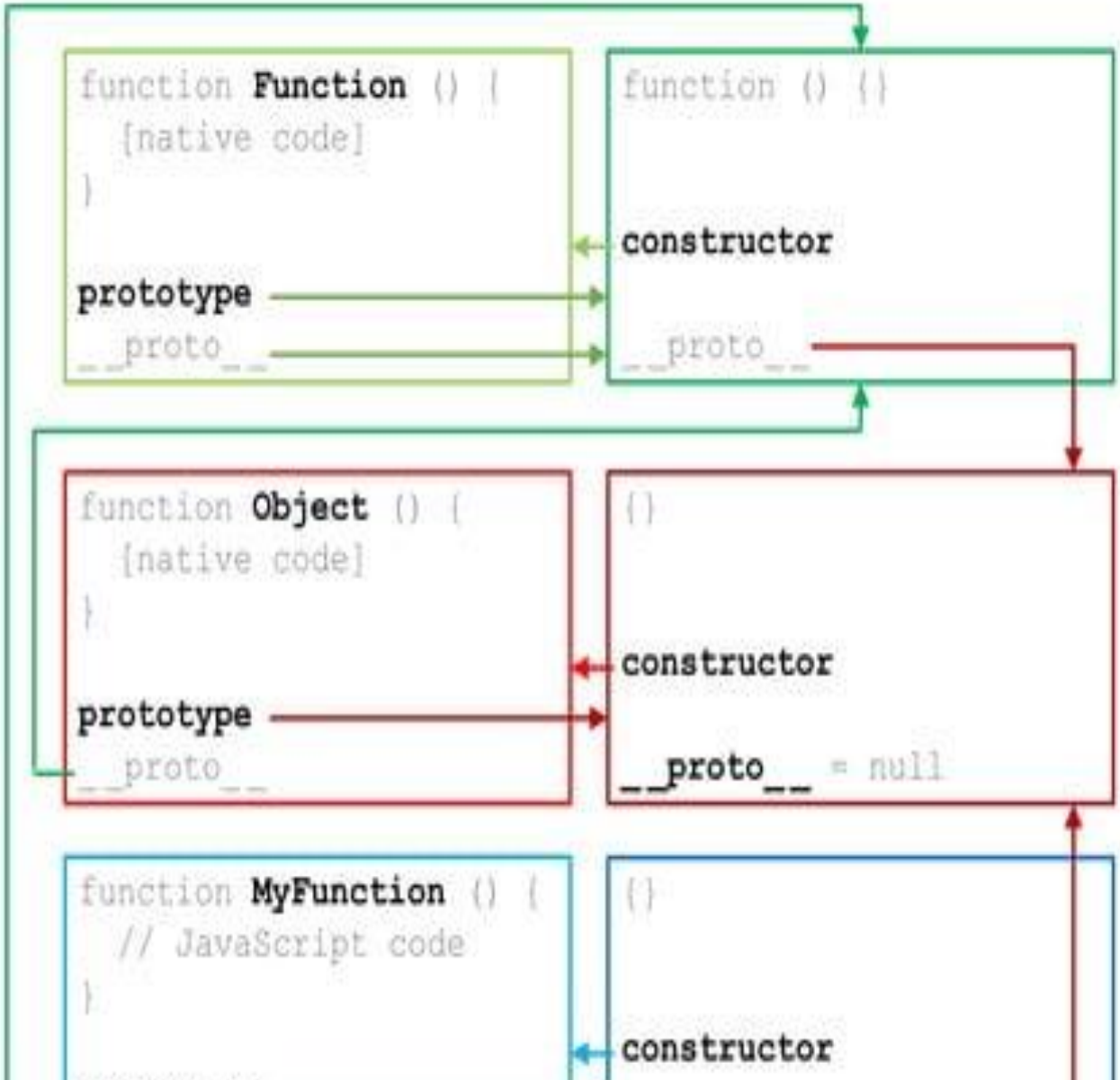
Chicken or the egg?

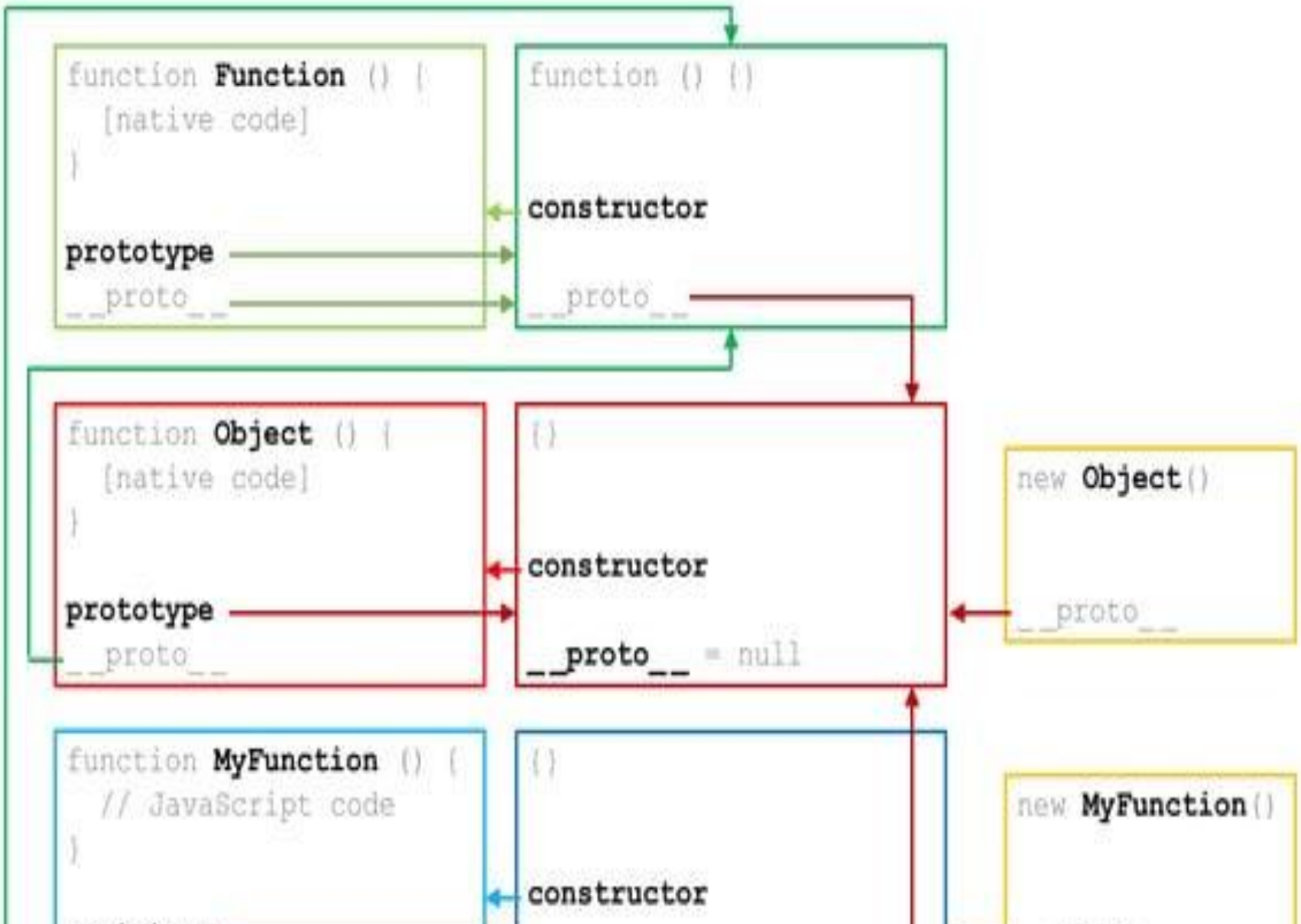


Chicken or the egg?




function Function () {
[native code]
}
prototype
__proto__
function () {}
constructor
__proto__
function Object () {
[native code]
}
prototype
__proto__
{}
constructor
__proto__ = null
new Object()
__proto__


function Function () {
[native code]
}
prototype
__proto__
function () {}
constructor
__proto__
function Object () {
[native code]
}
prototype
__proto__
{}
constructor
__proto__ = null
function MyFunction () {
// JavaScript code
}
{}
constructor


function Function () {
[native code]
}
prototype
__proto__
function () {}
constructor
__proto__
function Object () {
[native code]
}
prototype
__proto__
{}
constructor
__proto__ = null
new Object()
__proto__
function MyFunction () {
// JavaScript code
}
{}
constructor
new MyFunction()

Takeaways

- no classes – only objects and primitive values
- `__proto__` vs. `prototype`
- one function = two objects
- `this` & `return`
- use `F.prototype` to share properties
- inherit from `that`
- `instanceof` vs. `Object.isPrototypeOf()`
- the map of `Function` and `Object` inheritance
- MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript

Thank you!

Q & A

BONUS

Overview of object oriented and functional concepts, with examples

Polymorphism – Inheritance

```
// Base class providing common interface to all subclasses
var Vehicle = function (name, wheels) {
  this.name = name;
  this.wheels = wheels;
};
Vehicle.prototype.drive = function () {
  console.log('Driving the ' + this.name + ' on ' + this.wheels + ' wheels');
};

// Subclass with additional methods, sharing the common interface
var Car = function (doors) {
  this.superConstructor('car', 4);
  this.doors = doors;
};
// Inherit the base class and keep a reference to the super class constructor
Car.prototype = Object.create(Vehicle.prototype);
Car.prototype.superConstructor = Vehicle;
// Specialized subclass methods
Car.prototype.checkDoors = function () {
  console.log('This ' + this.name + ' has ' + this.doors + ' doors');
};

// Another subclass sharing the common interface
var Bicycle = function () {
  this.superConstructor('bicycle', 2);
};
```

```
// Example

var car = new Car(5);
var bicycle = new Bicycle();

car.checkDoors(); // This car has 5 doors
car.drive();      // Driving the car on 4 wheels
bicycle.drive();  // Driving the bicycle on 2 wheels
```

Polymorphism – Overloading

```
// Behavior depends on the number of supplied arguments
var argumentCounter = function() {
  if (arguments.length <= 1) {
    console.log('Processing at most one argument');
  } else {
    console.log('Processing multiple arguments');
  }
};

// Behavior depends on the argument types
var typeChecker = function(arg) {
  if (typeof arg === 'string') {
    console.log('Processing a string argument');
  } else {
    console.log('Processing a non-string argument');
  }
};
```

```
// Example

argumentCounter();        // Processing at most one argument
argumentCounter(1, 2, 3); // Processing multiple arguments

typeChecker('test');      // Processing a string argument
typeChecker(true);        // Processing a non-string argument
```

Encapsulation

```
var House = function() {
  var that = {},

  privateRoom = function() {
    console.log('in private');
  };

  that.publicRoom = function() {
    console.log('entered public');
    privateRoom();
    console.log('exiting public');
  };

  return that;

};
```

```
// Example

var house = new House();
house.publicRoom();                    // entered public
                                       // in private
                                       // exiting public

console.log('privateRoom' in house);   // false
```

Aggregation

```
var Vehicle = function (name) {
  return { name: name };
};

var Garage = function() {
  var that = {}, vehicles = [];

  that.add = function(vehicle) {
    vehicles.push(vehicle);
    console.log('Added ' + vehicle.name);
  };

  that.remove = function(vehicle) {
    var found;
    vehicles = vehicles.filter(function(v) {
      found = found || vehicle === v;
      return vehicle !== v;
    });
    console.log((found ? 'Removed ' : 'Could not remove ') + vehicle.name);
  };

  that.print = function() {
    var names = vehicles.map(function(v) {
      return v.name;
    });
    console.log('Vehicles: ' + names.join(', '));
  };
```

```
// Example

var truck = Vehicle('truck');
var car = Vehicle('car');
var bicycle = Vehicle('bicycle');

var garage = Garage();

garage.add(car);          // Added car
garage.add(bicycle);      // Added bicycle
garage.print();           // Vehicles: car, bicycle

garage.remove(bicycle);   // Removed bicycle
garage.remove(truck);     // Could not remove truck
garage.print();           // Vehicles: car
```

Higher-order functions & collections

```javascript
// Incapsulating iteration
function each(data, callback) {
  for (var i = 0; i < data.length; i += 1) {
    callback(data[i]);
  }
}

// Mapping each value to a result
function map(data, operation) {
  var result = [];
  each(data, function(value) {
    result.push(operation(value));
  });
  return result;
}

// Collecting only values satisfying the predicate
function filter(data, predicate) {
  var result = [];
  each(data, function(value) {
    if (predicate(value)) {
      result.push(value);
    }
  });
  return result;
}
```

```javascript
// Example

function increment(x) {
  return x + 1;
}

function odd(x) {
  return x % 2 === 1;
}

var data = [0, 1, 2, 3, 4];

console.log(map(data, increment));  // [1, 2, 3, 4, 5]
console.log(filter(data, odd));     // [1, 3]
```

Higher-order functions & composition

```
function compose (f, g) {
  // Receive function arguments and/or return a function as the result
  return function (x) {
    return f(g(x));
  };
}
```

```
// Example

function increment (x) {
  return x + 1;
}

function times2 (x) {
  return 2 * x;
}

var f = compose(increment, times2);

console.log(f(1)); // 3
console.log(f(2)); // 5
console.log(f(3)); // 7
```

Memoization

```javascript
// Without memoization, the function may repeatedly
// compute the result for the same arguments
var fibonacciSlow = function fib (n) {
  var result;
  if (n < 2) {
    result = n;
  } else {
    result = fib(n - 1) + fib(n - 2);
    console.log('fibSlow(' + n + ') = ' + result);
  }
  return result;
};


// Example

fibonacciSlow(5);
// fibSlow(2) = 1
// fibSlow(3) = 2
// fibSlow(2) = 1
// fibSlow(4) = 3
// fibSlow(2) = 1
// fibSlow(3) = 2
// fibSlow(5) = 5
```

```javascript
var fibonacci = (function () {
  // Function keeps previously computed values in its private closure
  // and returns the cached results when they are available
  var memo = [0, 1];
  return function fib (n) {
    var result = memo[n];
    if (typeof result !== 'number') {
      result = fib(n - 1) + fib(n - 2);
      console.log('fib(' + n + ') = ' + result);
      memo[n] = result;
    }
    return result;
  };
}());

// Example

fibonacci(5);
// fib(2) = 1
// fib(3) = 2
// fib(4) = 3
// fib(5) = 5
```

Currying and partial function application

```
function curry (f) {
  var slice = Array.prototype.slice;
  var concat = Array.prototype.concat;
  // Return the curried function f. The returned function is a "named
  // anonymous function" or more precisely a named function expression.
  // On the outside, the function is anonymous and its identifier 'curried'
  // is accessible only from the inside of the function
  return function curried () {
    if (f.length > arguments.length) {
      // If some arguments are still missing, save the supplied arguments
      // and return a new function delegating back to this function, but
      // with the additional arguments concatenated to the saved args
      var args = slice.apply(arguments);
      return function () {
        return curried.apply(null, concat.apply(args, arguments));
      };
    }
    // If the sufficient number of arguments is supplied, delegate to f
    return f.apply(null, arguments);
  };
}
```

```
// Example

function sum3 (a, b, c) {
  return a + b + c;
}

s = curry(sum3);

console.log(s(1)(2)(3));  // 6
console.log(s(1)(2,3));   // 6
console.log(s(1,2)(3));   // 6
console.log(s(1,2,3));    // 6
```