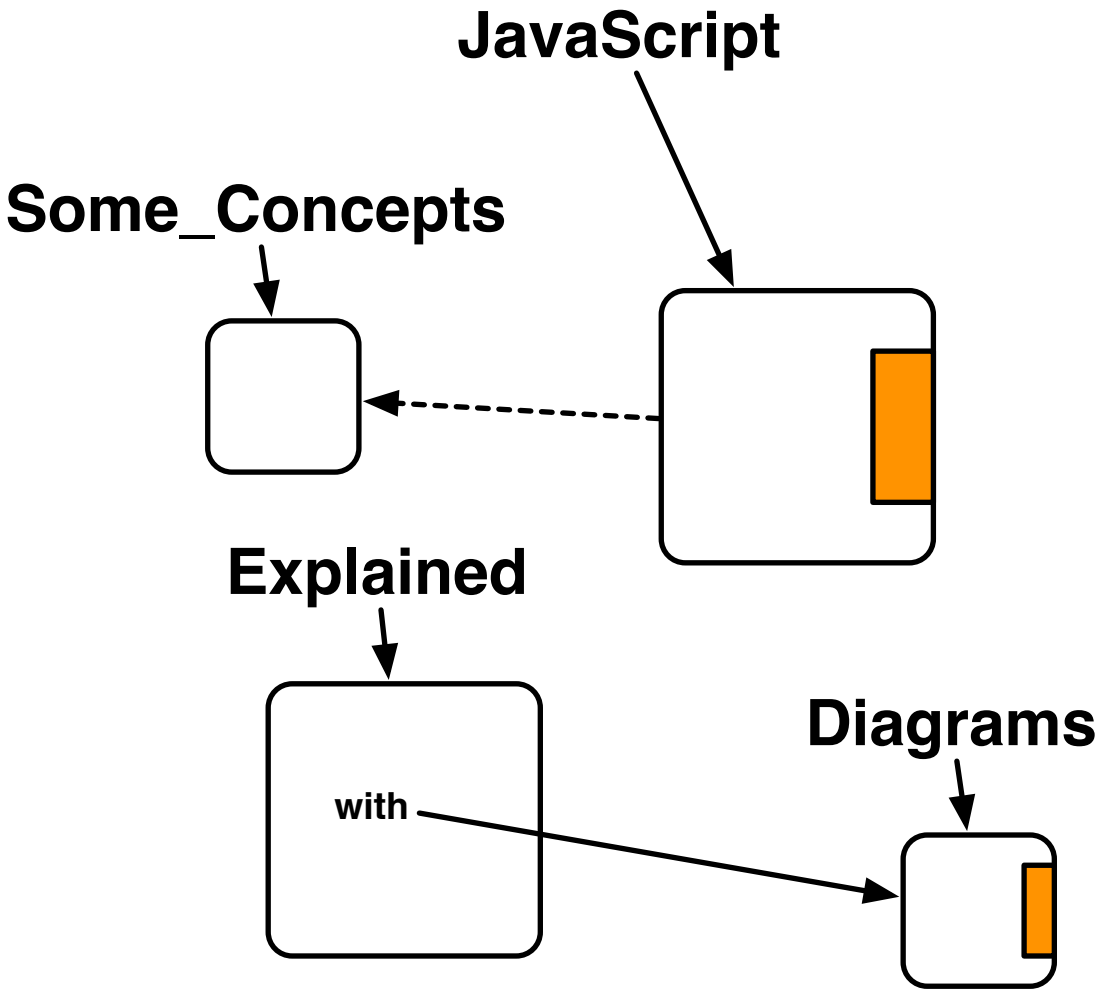


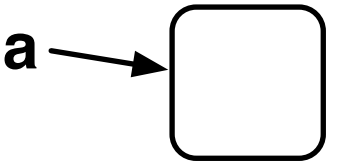

```
/* By Pierre Rioux */
/* ACE Lab Meeting */
/* September 2013  */
/* May 2017        */
```

Simple Objects

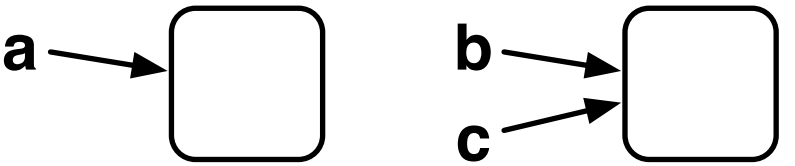
```
/* No code */
```

(nothing in memory)

```
var a = {};
```




```
var a = {};
var b = {};
var c = b;
```



Object Properties

```
var a = {};
a.hello = 1;
a['good bye'] = 2;
```


```
var b = {
    hello: 'xyz',
    'good bye': 9
    };
```

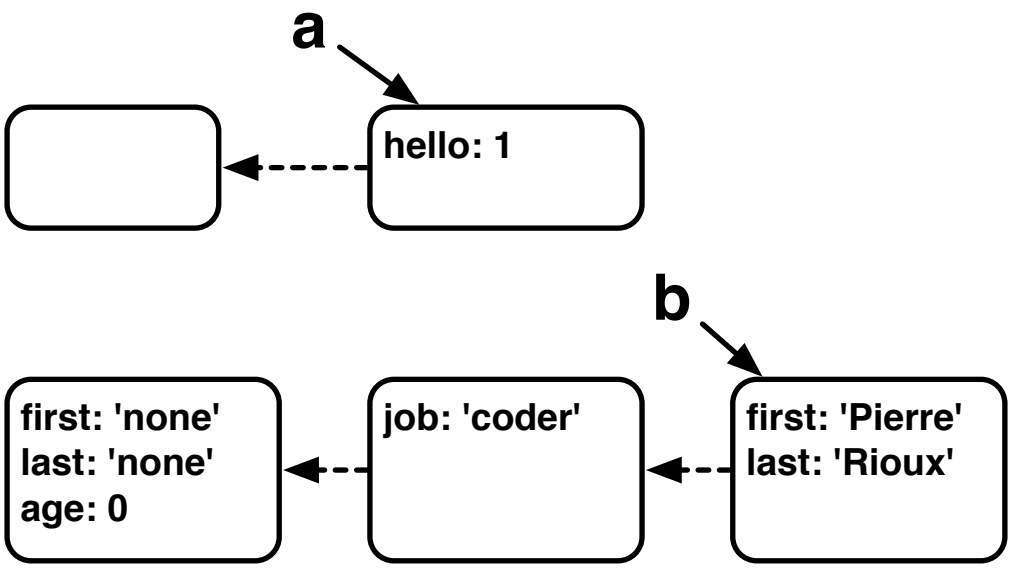

```
a.hello = b.hello;
```

Object Prototypes

```
var a = {};
a.hello = 1;
```

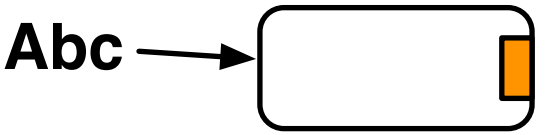
```
var b = (code not shown);
b.first = 'Pierre';
b.last  = 'Rioux';
```

```
b.first
> 'Pierre'
b.last
> 'Rioux'
b.job
> 'coder'
b.age
> 0
b.salary
> undefined
```

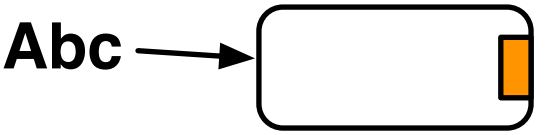


Functions (are objects with code)

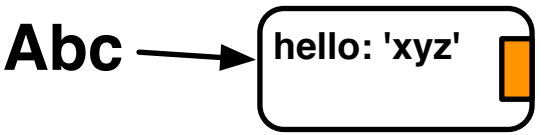
```
function Abc() {};
```




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

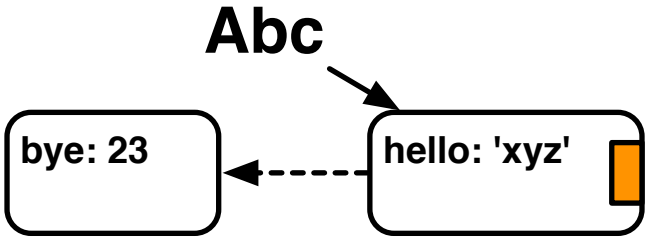



```
Abc.hello = 'xyz';
```

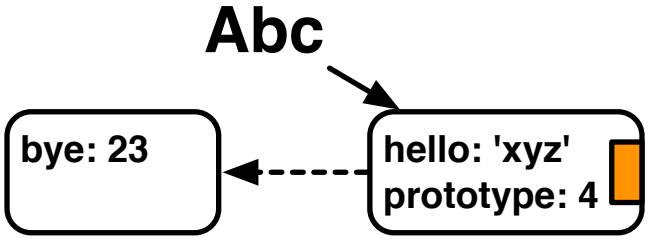


Functions have prototypes too

```
function Abc() {};
Abc.hello = 'xyz';
Abc.bye;
> 23
```




```
Abc.prototype = 4;
```



Invoking a function's code: four patterns

```
var result = Abc(2,3);
```

The function invocation pattern

```
var result = obj.abc(2,3);
```

The method invocation pattern

```
var result = new Abc(2,3);
```

The constructor invocation pattern

```
var result = Abc.apply(obj, [2,3]);
```

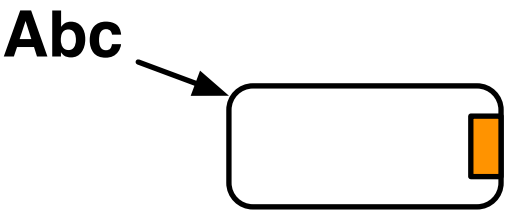
The apply invocation pattern

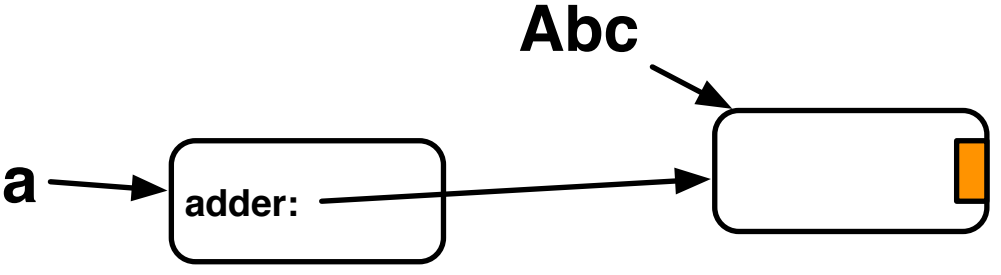
The function invokation pattern

```
var Abc = function(a,b)
{
  return a+b;
}
```

```
var x = Abc(3,5);
x;
> 8
```

```
var a = {};
a.adder = Abc;
a.adder(1211,4321);
> 5532
```



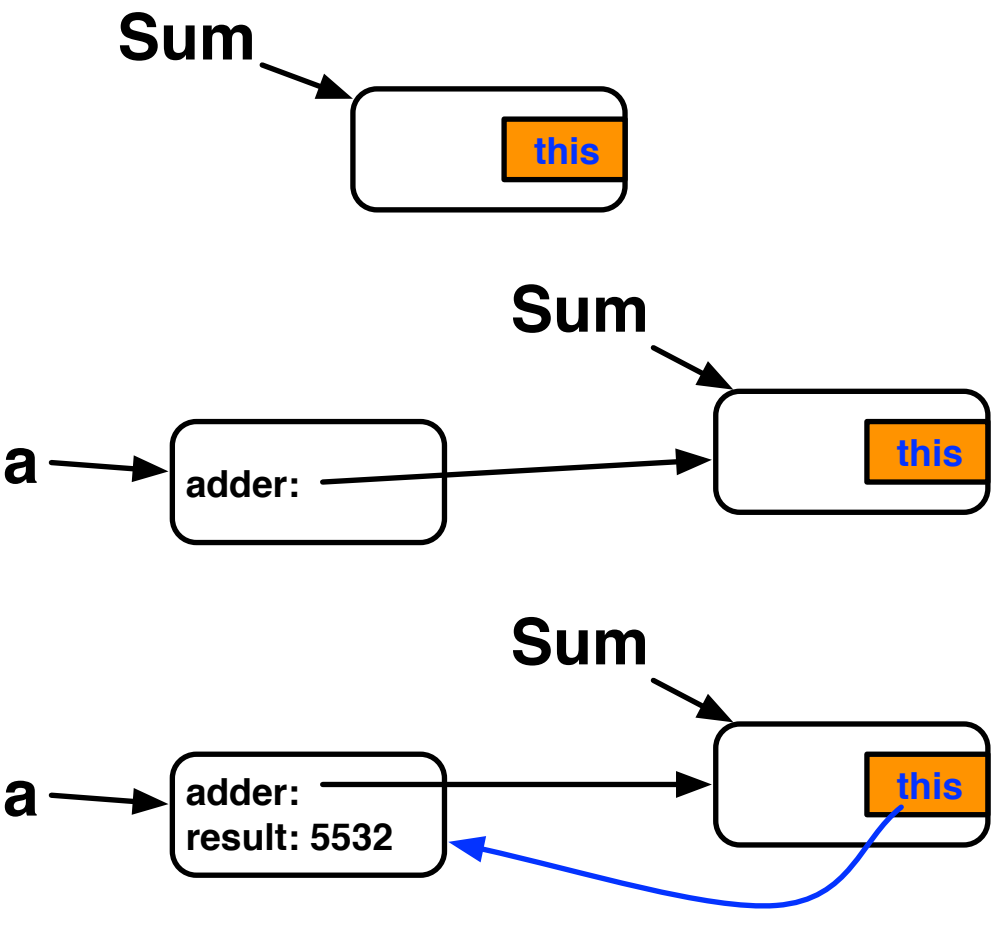


The method invocation pattern

```
var Sum = function(a,b)
{
  this.result = a+b;
  return "I'm done";
}
```

```
var a = {};
a.adder = Sum;
```

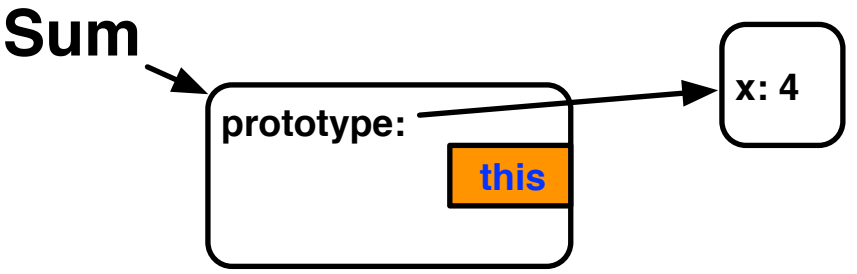
```
a.adder(1211,4321);
> "I'm done"
a.result
5532
```



The constructor invocation pattern (1/5)

```
var Sum = function(a,b) {
  this.result = a+b;
  return this;
}
Sum.prototype = { x: 4 };
```

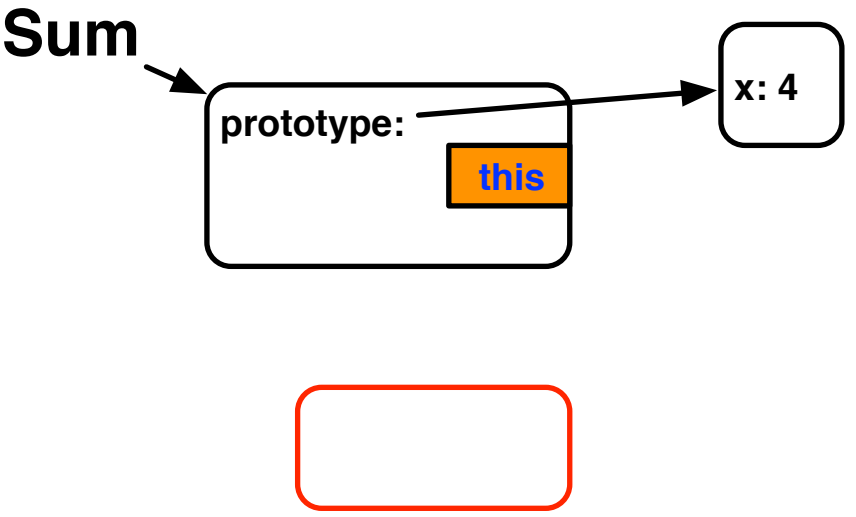
```
var m = new Sum(7,8);
```



The constructor invocation pattern (2/5)

```
var Sum = function(a,b) {
  this.result = a+b;
  return this;
}
Sum.prototype = { x: 4 };
```

```
var m = new Sum(7,8);
```

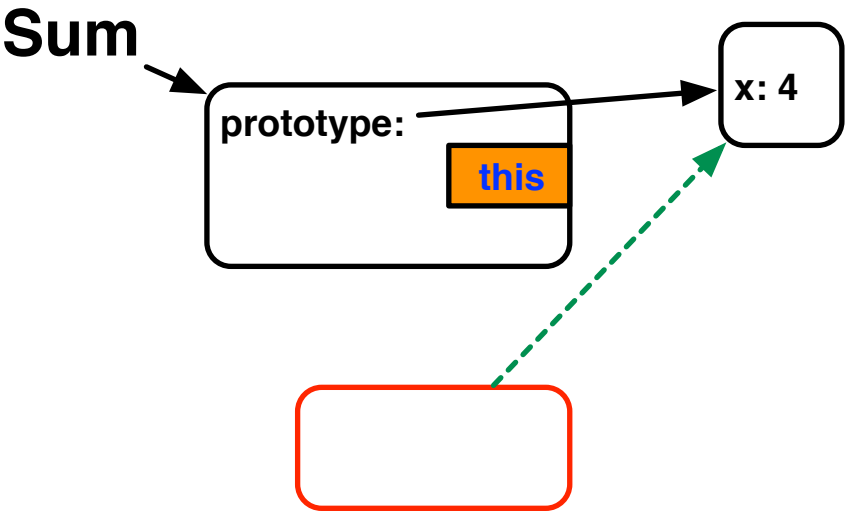


1- A brand new blank object is created by JavaScript

The constructor invocation pattern (3/5)

```
var Sum = function(a,b) {
  this.result = a+b;
  return this;
}
Sum.prototype = { x: 4 };
```

```
var m = new Sum(7,8);
```



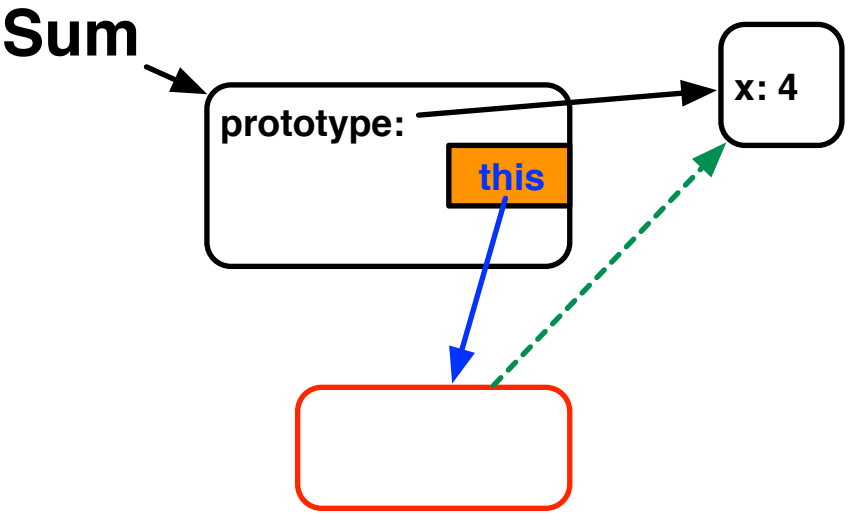
1- A brand new blank object is created by JavaScript

2- Its prototype is set to the object pointed to by the prototype property of the function

The constructor invocation pattern (4/5)

```
var Sum = function(a,b) {
  this.result = a+b;
  return this;
}
Sum.prototype = { x: 4 };
```

```
var m = new Sum(7,8);
```



1- A brand new blank object is created by JavaScript

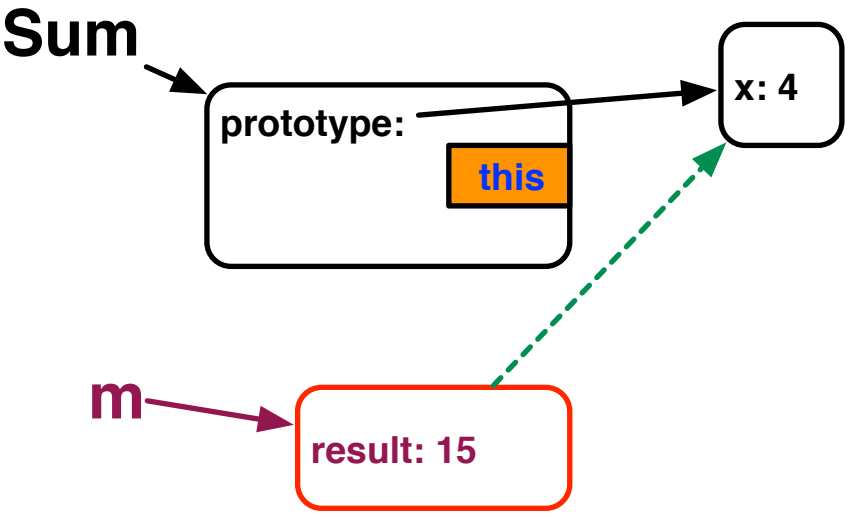
2- Its prototype is set to the object pointed to by the prototype property fo the function

3- The keyword 'this' in the function code is bound to the object

The constructor invocation pattern (5/5)

```
var Sum = function(a,b) {
  this.result = a+b;
  return this;
}
Sum.prototype = { x: 4 };
```

```
var m = new Sum(7,8);
```



1- A brand new blank object is created by JavaScript

2- Its prototype is set to the object pointed to by the prototype property fo the function

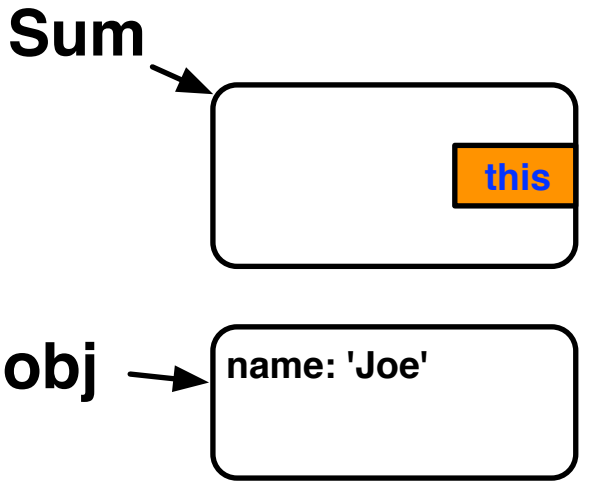
3- The keyword 'this' in the function code is bound to the object

4- The code is run, the returned value is assigned to m. (Typically, the code returns this, the new object).

```
m.result
> 15
m.x
> 4
```

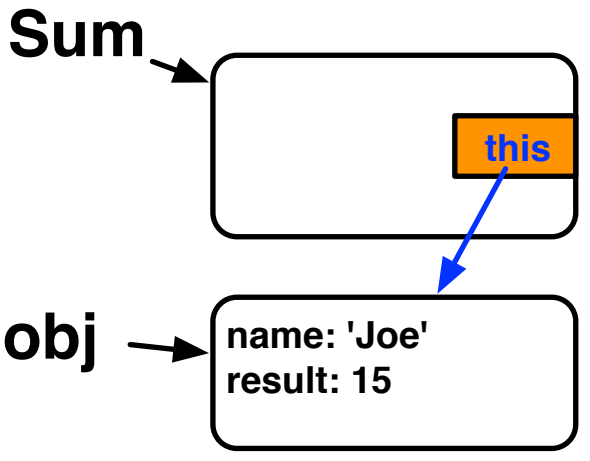
The apply invocation pattern

```
var Sum = function(a,b) {
  this.result = a+b;
  return "All OK";
}
var obj = { name: 'Joe' };
```




```
var m = Sum.apply(obj, [7,8]);

m
> "All OK"
obj.name
> "Joe"
obj.result
> 15
```



What 'this' does in the four patterns

```
var result = Abc(2,3);
```

The function invocation pattern:
this points to the global object! BAD!

```
var result = obj.abc(2,3);
```

The method invocation pattern:
this points to obj

```
var result = new Abc(2,3);
```

The constructor invocation pattern:
this points to brand new object (typically returned as result)

```
var result = Abc.apply(obj, [2,3]);
```

The apply invocation pattern:
this points to obj of your choice