confused javascript

wants to be a prototyping language
› no classes, prototype chain

wants to be a standard OO language
› instead of cloning operator, has pseudo constructor

consequence
› some strange rules
› easy to mess up
this is dynamically scoped
› in evaluating e.m(), this is bound to value of e inside m
› but reverts to global environment in calls that m makes

```javascript
var counter = {
  val: 0,
  inc: function() {
    this.val += 1; return this.val;
  }
}
counter.inc(); // 1
counter.inc(); // 2
```
where *this* fails

```javascript
Array.prototype.map = function (f) {
    var map_from = function (i) {
        if (i === 0) return [];
        return map_from(i-1).concat(f(this[i-1]));
    };
    return map_from(this.length);
}
```

a workaround:

```javascript
Array.prototype.map = function (f) {
    var that = this;
    var map_from = function (i) {
        if (i === 0) return [];
        return map_from(i-1).concat(f(that[i-1]));
    };
    return map_from(this.length);
}
```
forgetting to use “new”

```javascript
var Point = function (x, y) {
    this.x = function () {return x;}
    this.y = function () {return y;}
}
```

```javascript
> x = 3
3
> var p = Point(1,2)
undefined
> p.x()
TypeError
> x
function () {return x;}
> p
undefined
```

a remedy:

```javascript
var Point = function (x, y) {
    if (!(this instanceof Point)) return new Point(x, y);
    this.x = function () {return x;}
    this.y = function () {return y;}
}
```
so this or that?

how to make an ADT
› with *this* & *new*
› with closures alone

<table>
<thead>
<tr>
<th></th>
<th><em>this &amp; new</em></th>
<th>closures</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>instanceof</em></td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><em>extend prototype</em></td>
<td>yes, but can’t see rep</td>
<td>no</td>
</tr>
<tr>
<td><em>avoid nasties</em></td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>