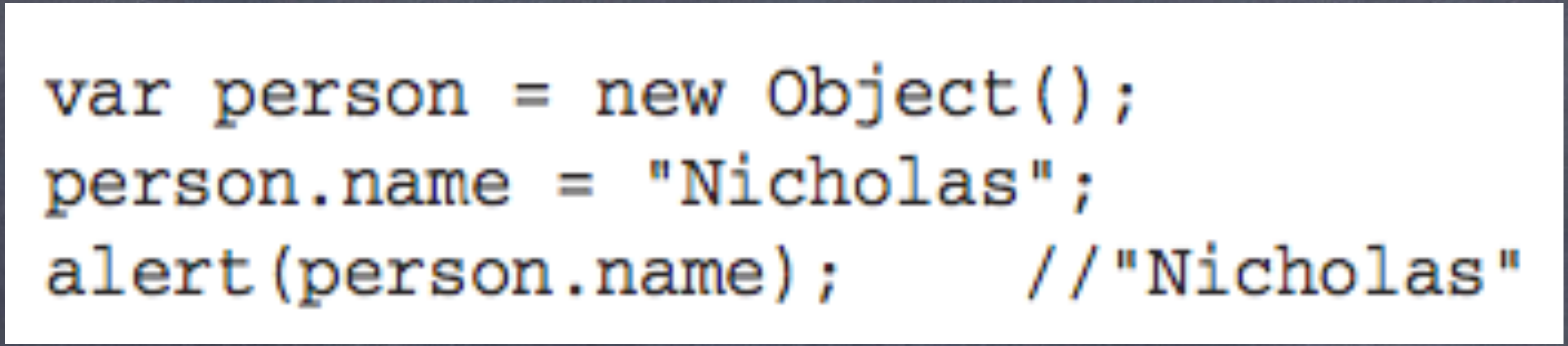
Variables, Scope and Memory

fanzhongkai@baidu.com

PRIMITIVE AND REFERENCE VALUES

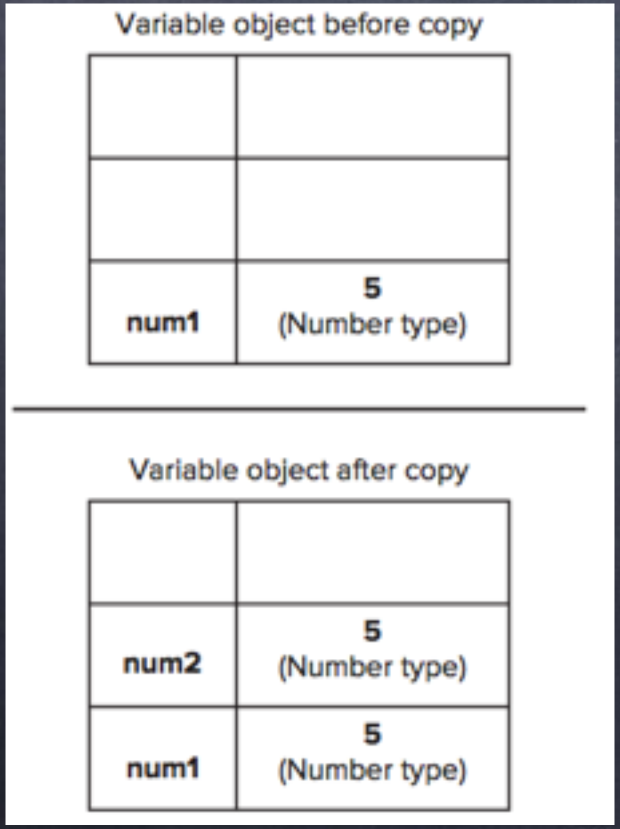
Dynamic Properties

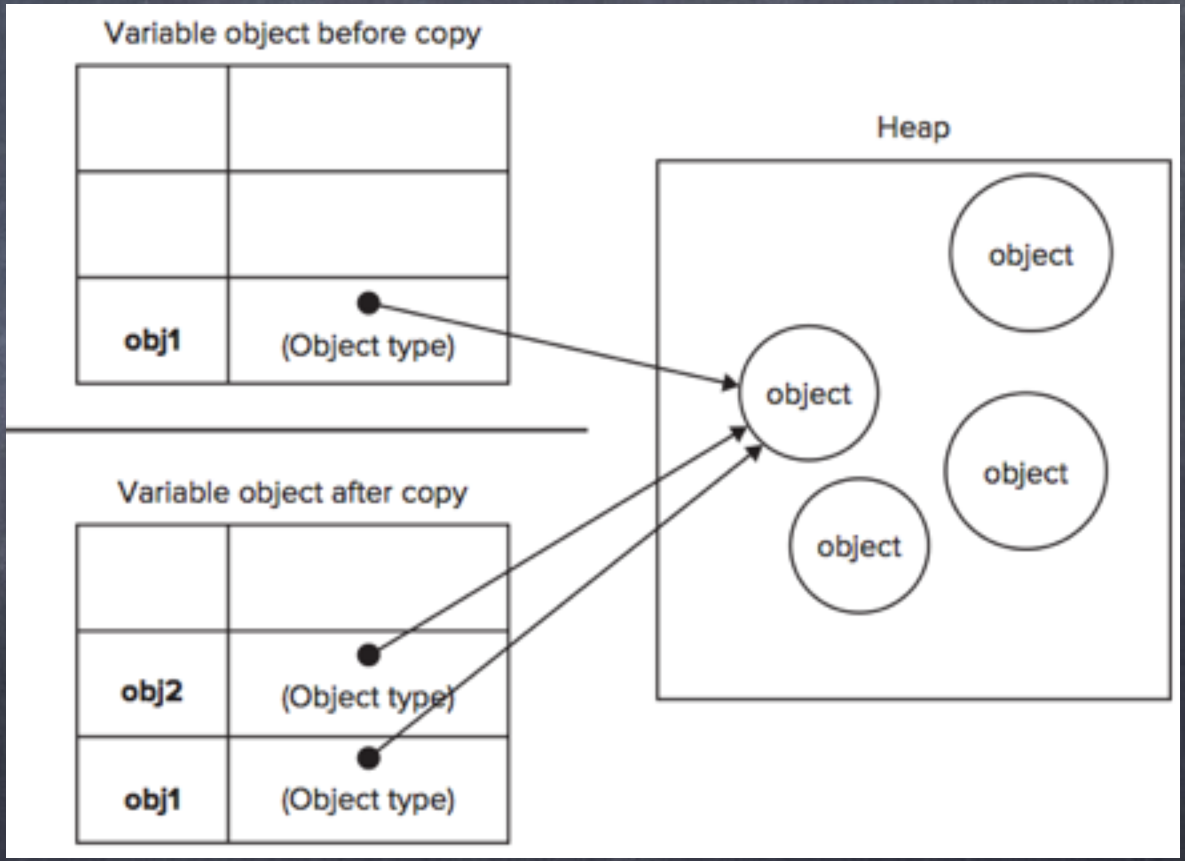


```
var person = new Object();
person.name = "Nicholas";
alert(person.name);    //"Nicholas"
```

*Copying Values

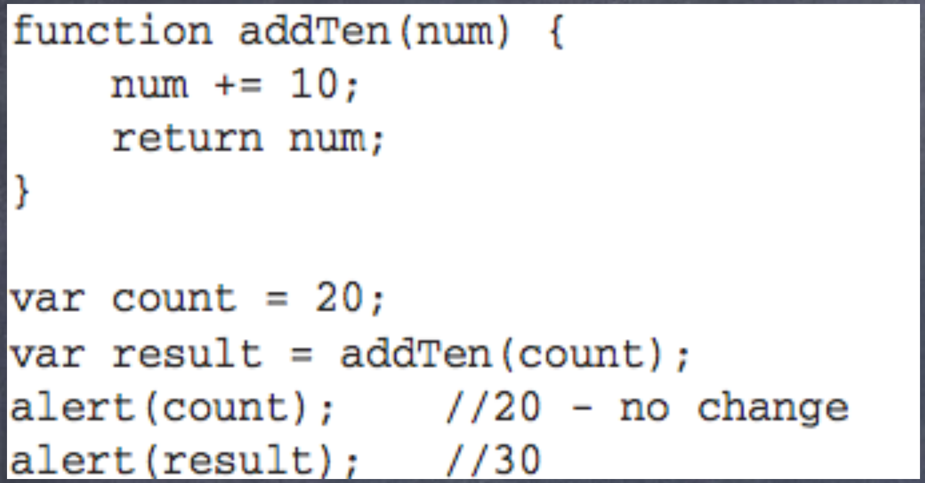
- Primitive value: copy variable
- Reference value: copy pointer



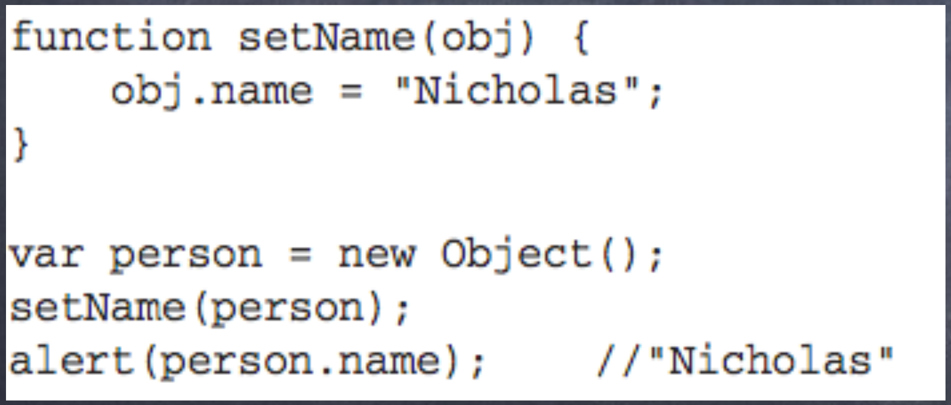


```
var obj1 = new Object();
var obj2 = obj1;
obj1.name = "Nicholas";
alert(obj2.name);     //"Nicholas"
```

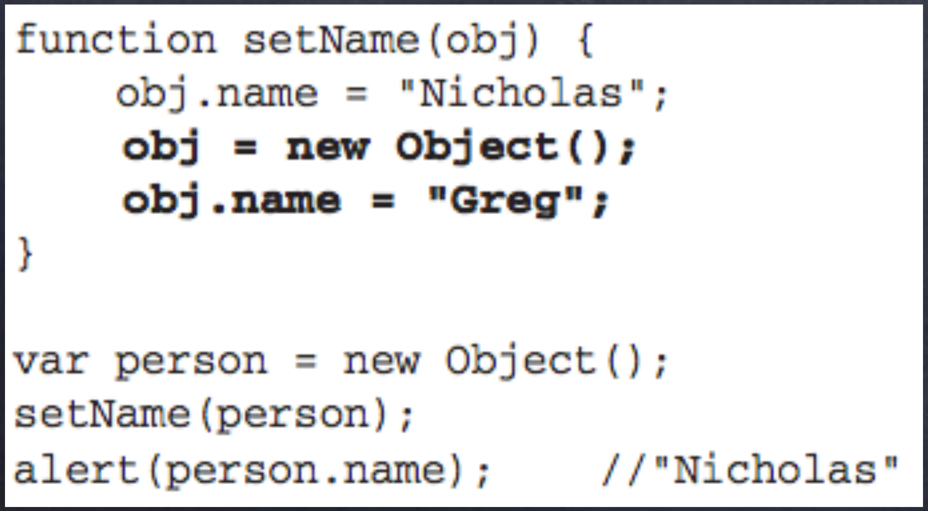
Argument Passing



```
function addTen(num) {
    num += 10;
    return num;
}

var count = 20;
var result = addTen(count);
alert(count);    //20 - no change
alert(result);   //30
```



```
function setName(obj) {
    obj.name = "Nicholas";
}

var person = new Object();
setName(person);
alert(person.name);    //"Nicholas"
```

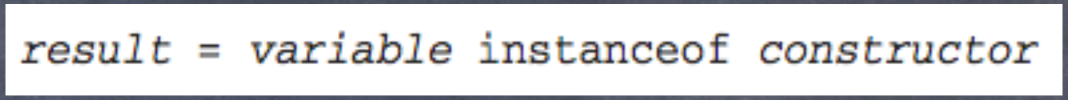


```
function setName(obj) {
    obj.name = "Nicholas";
    obj = new Object();
    obj.name = "Greg";
}

var person = new Object();
setName(person);
alert(person.name);    //"Nicholas"
```

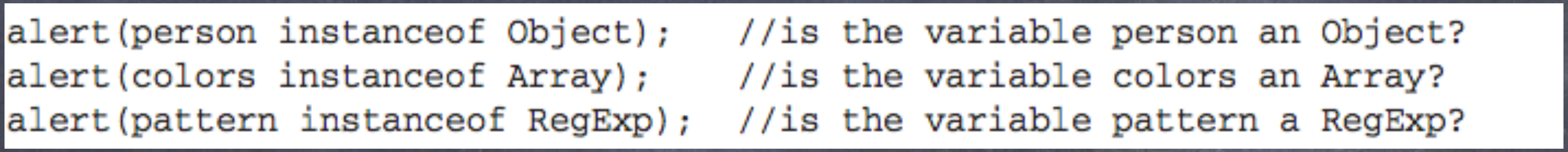
Think of function arguments in ECMAScript as nothing more than local variables.

Determining Type

- typeof vs instanceof



```
result = variable instanceof constructor
```



```
alert(person instanceof Object);   //is the variable person an Object?
alert(colors instanceof Array);    //is the variable colors an Array?
alert(pattern instanceof RegExp);  //is the variable pattern a RegExp?
```

Constructor

return primitive;

return reference;

without return;

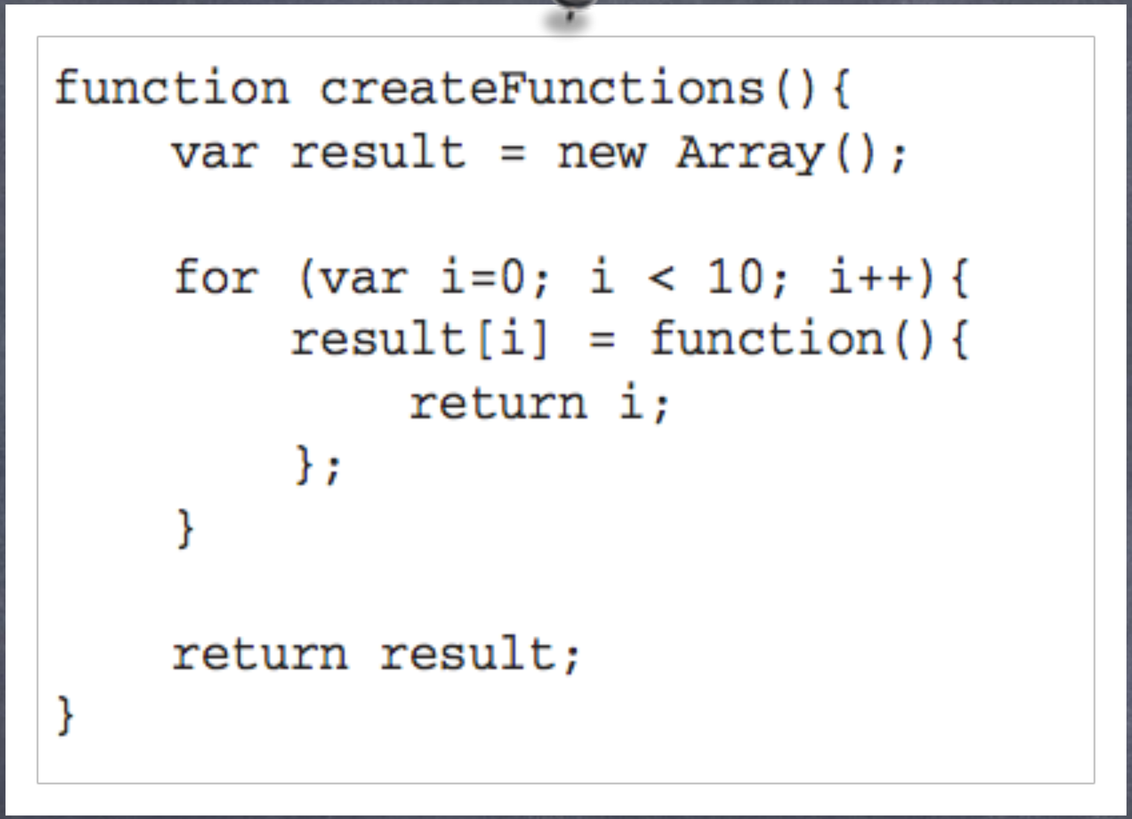
*Execute Context & Scope

```
var a = {b: 1, c: function() {console.info(this.b);}}; b = 2; console.info(this.b); a.c();
```

this

- scope chain
- scope chain augmentation (with & catch)
- No Block-Level Scopes
- Identifier Lookup

```
if (true) {
var color = “blue”;
}
alert(color); //”blue”
```

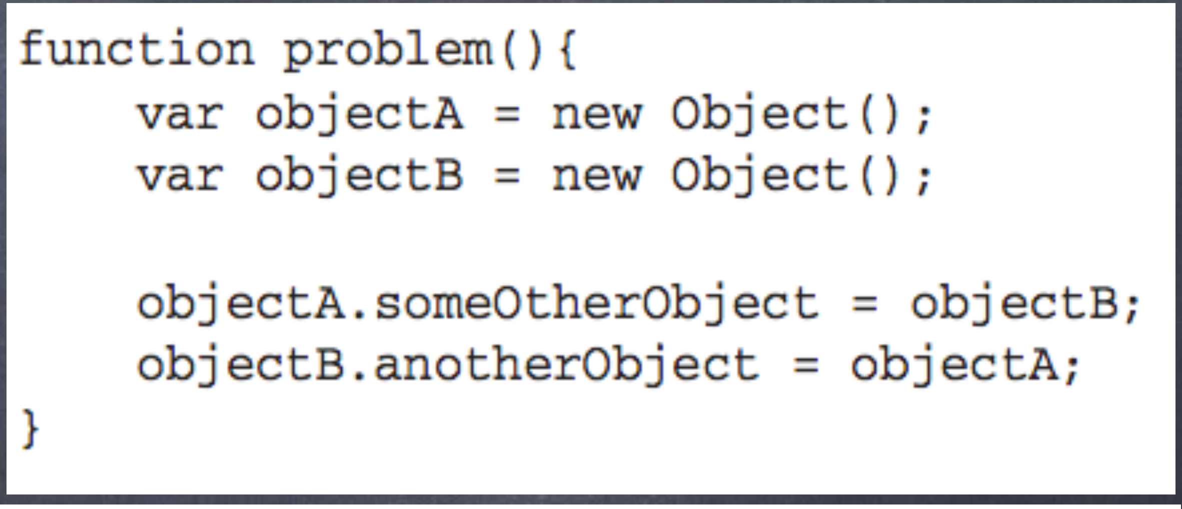


```
function createFunctions(){
    var result = new Array();

    for (var i=0; i < 10; i++){
        result[i] = function(){
            return i;
        };
    }

    return result;
}
```

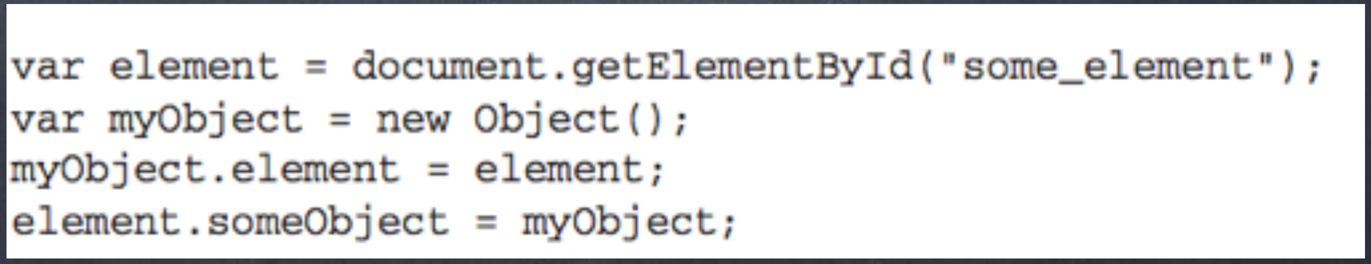
闭包 Closure

GARBAGE COLLECTION

- Mark-and-Sweep
- Reference Counting (circular reference)
- Performance
- Managing Memory (dereferencing)



```
function problem(){
    var objectA = new Object();
    var objectB = new Object();

    objectA.someOtherObject = objectB;
    objectB.anotherObject = objectA;
}
```



```
var element = document.getElementById("some_element");
var myObject = new Object();
myObject.element = element;
element.someObject = myObject;
```