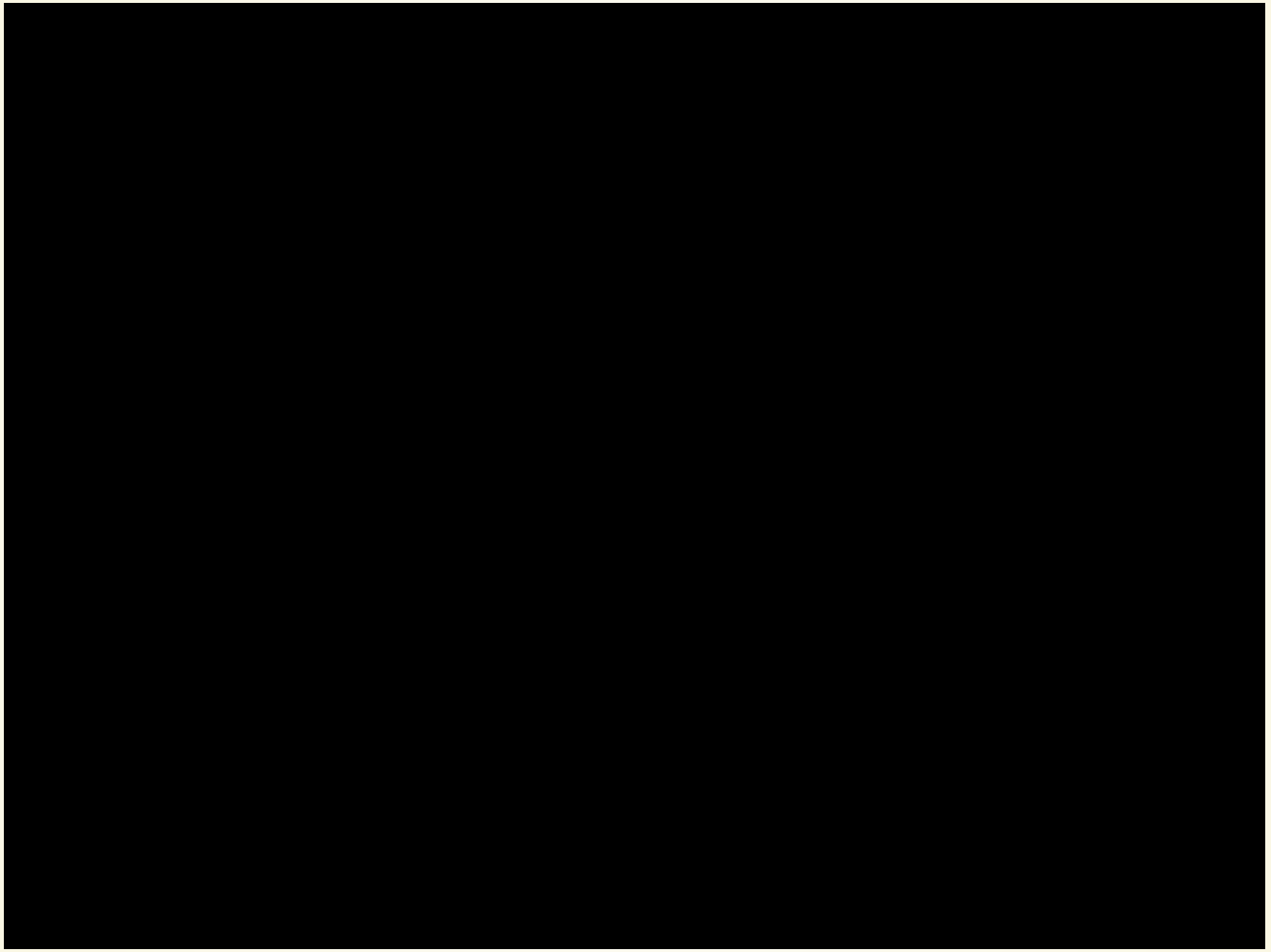
OBJECTS IN JAVASCRIPT

REFERENCING OBJECTS

John R. Williams and *Abel Sanchez*

OBJECT LAYOUT VIDEO



CREATING OBJECTS

We can create an object as shown below.

```
1 var a = 5;
2 var p = {x:3, y:4};
3 var q = p;
4
```

run code | reload original code

```
Output:
```

This object created this way is called a 'literal' object. Here 'p' is the name of the object and 'x' and 'y' are called 'properties'. We also say 'p' is a reference to the object.

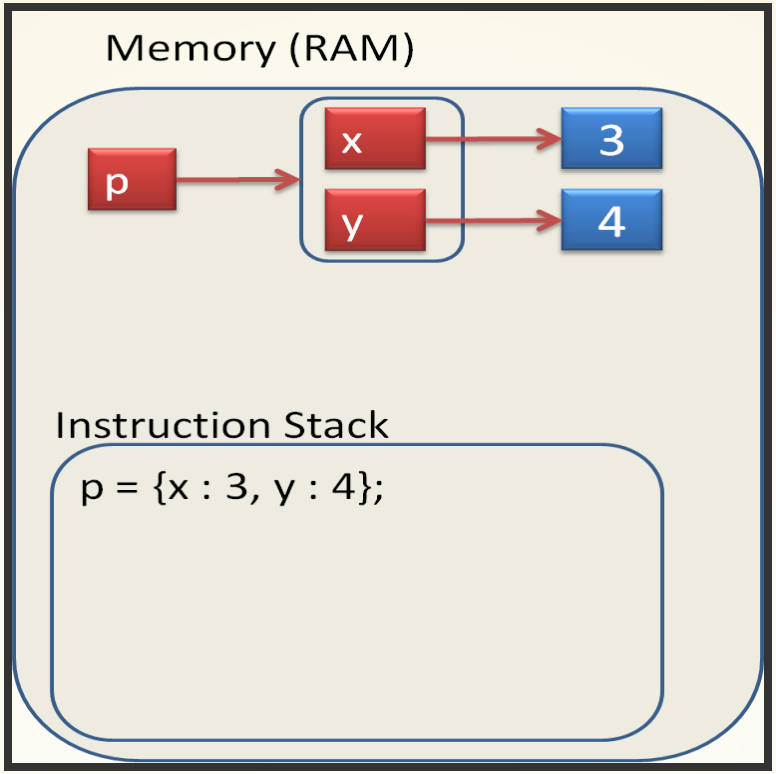
Copy the code into your Chrome Console and edit the values of the properties of 'p' and 'q'. Change the value of p.x and write out the values of p.x and p.y and q.x and q.y to convince yourself that p and q reference the same object.

Create another literal object with properties 'name' and 'age' where name is your first name and age is your age. Write out these properties then change your age to something different.

```
1 var a = 5;
2 var p = {x:3, y:4};
3 var q = p;
4
```

run code | reload original code

```
Output:
```


Memory (RAM)
p
x
y
3
4
Instruction Stack
p = {x : 3, y : 4};

Using the 'dot' notation we can reference the x and y 'properties' of the object and we can 'get' or 'set' those variables.

```
1 p = {x:3, y:4};
2 p.x = 5; // sets the value of x to 5;
3 console.log('p.x = '+ p.x);
4 var a = p.x;  // creates a value type a
5               // and sets its value to p.x ie 5
6 console.log('a = '+ a);
7
```

run code | reload original code

```
Output:
```

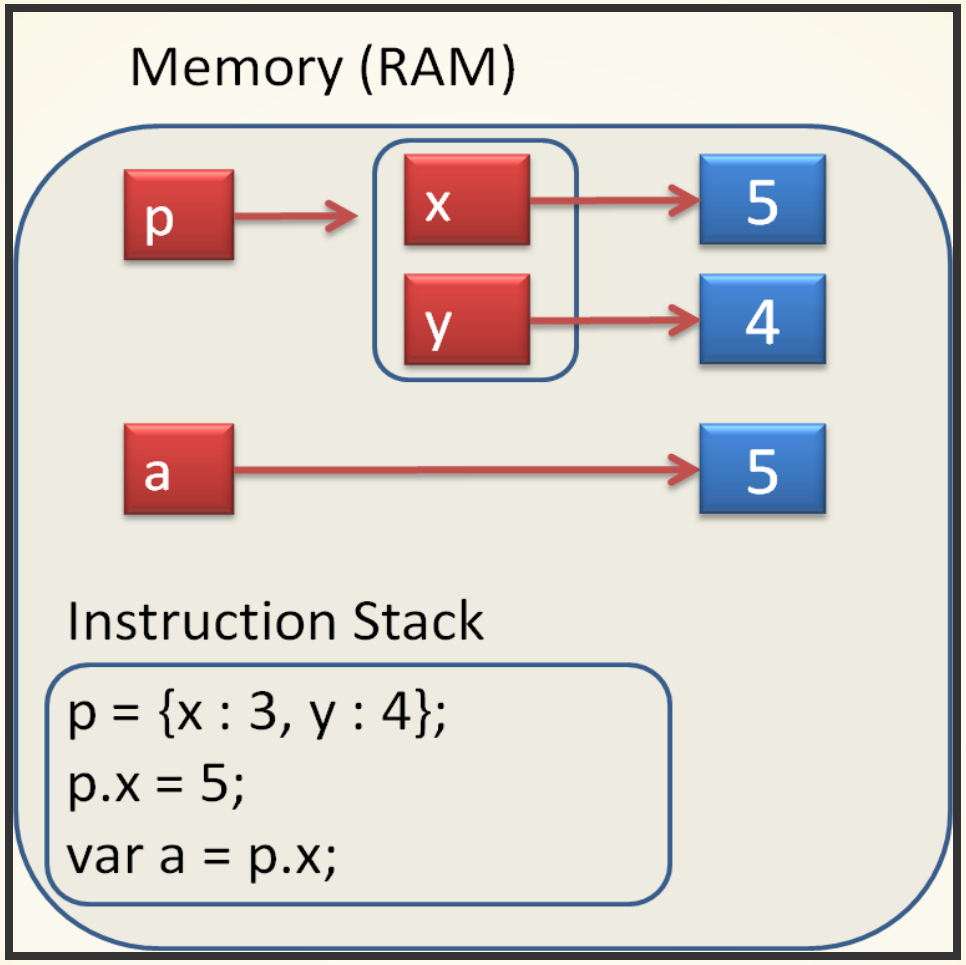
Lets convince ourselves that 'a' has its own memory slot and is independent of p.x. Can you draw out the layout in memory?

```
1 p = {x:3, y:4};
2 p.x = 5; // sets the value of p.x to 5;
3 console.log('p.x = '+ p.x);
4 var a = p.x;  // creates value type called 'a'
5 // lets check that 'a' is now independent of p.x
6 p.x = 9;
7 console.log('p.x = '+ p.x);
8 console.log('a = '+ a);
9
```

run code | reload original code

```
Output:
```

Here is the memory layout.



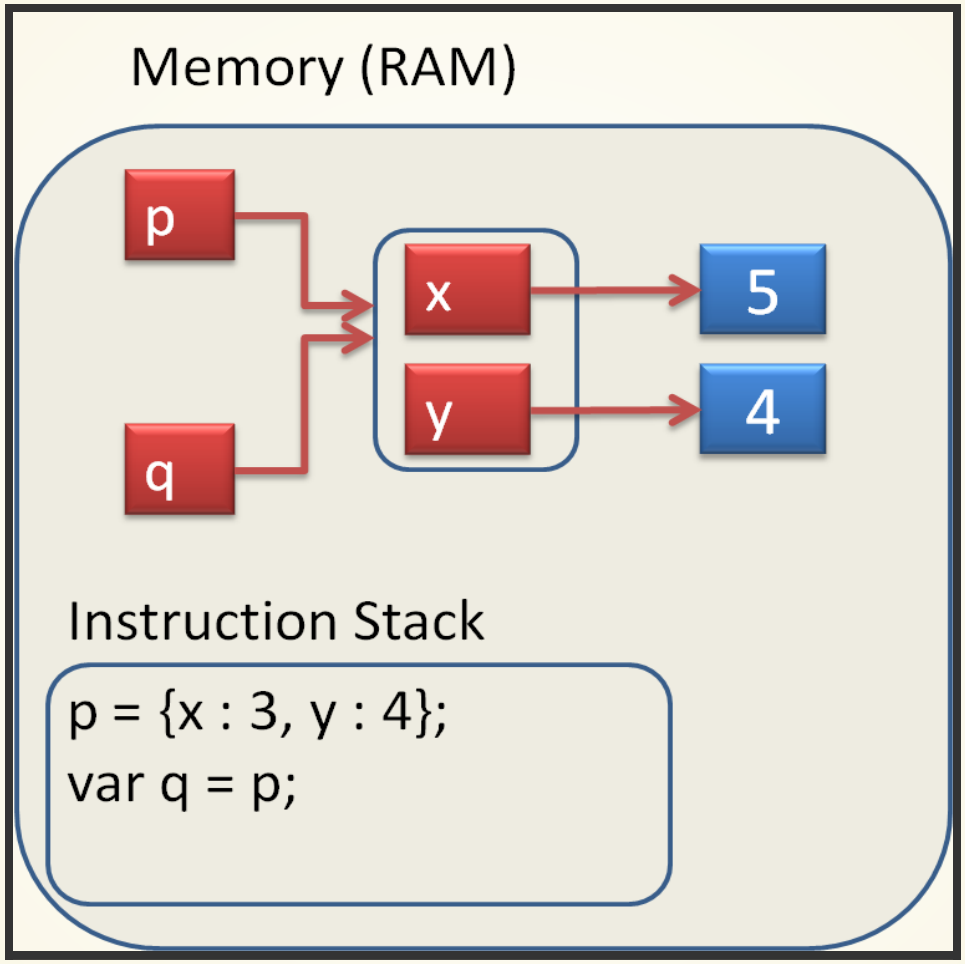
Now lets create a variable 'q' and assign 'p' to it. Why does q.x change value

```
1 p = {x:3, y:4};
2 var q = p;
3 console.log('q.x = '+ q.x);
4 p.x = 5;
5 console.log('q.x = '+ q.x);
6 // why does q.x change value ?
7
8
```

run code | reload original code

```
Output:
```

Here is the memory layout. We call 'q' a reference to the object.
Indeed, both 'p' and 'q' reference the same object.



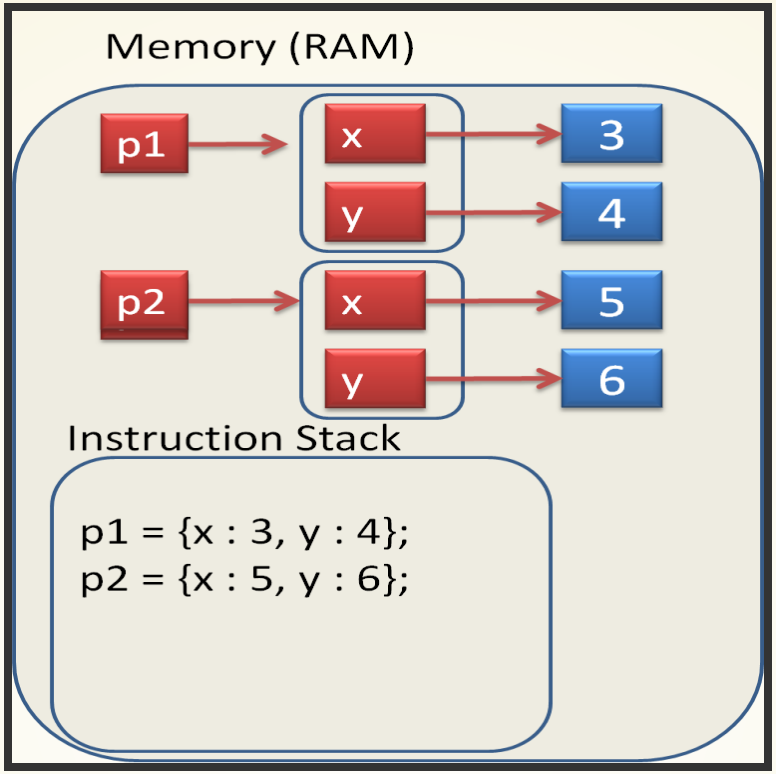
We can have many references to the same object.

Suppose we want to create several Point objects. One way is shown below. Can you create your own object? Can you imagine the memory layout diagram?

```
1 var p1 = {x:3, y:4};
2 var p2 = {x:5, y:6};
3 console.log('('+p1.x +', '+ p1.y+')');
4 console.log('('+p2.x +', '+ p2.y+')');
5
```

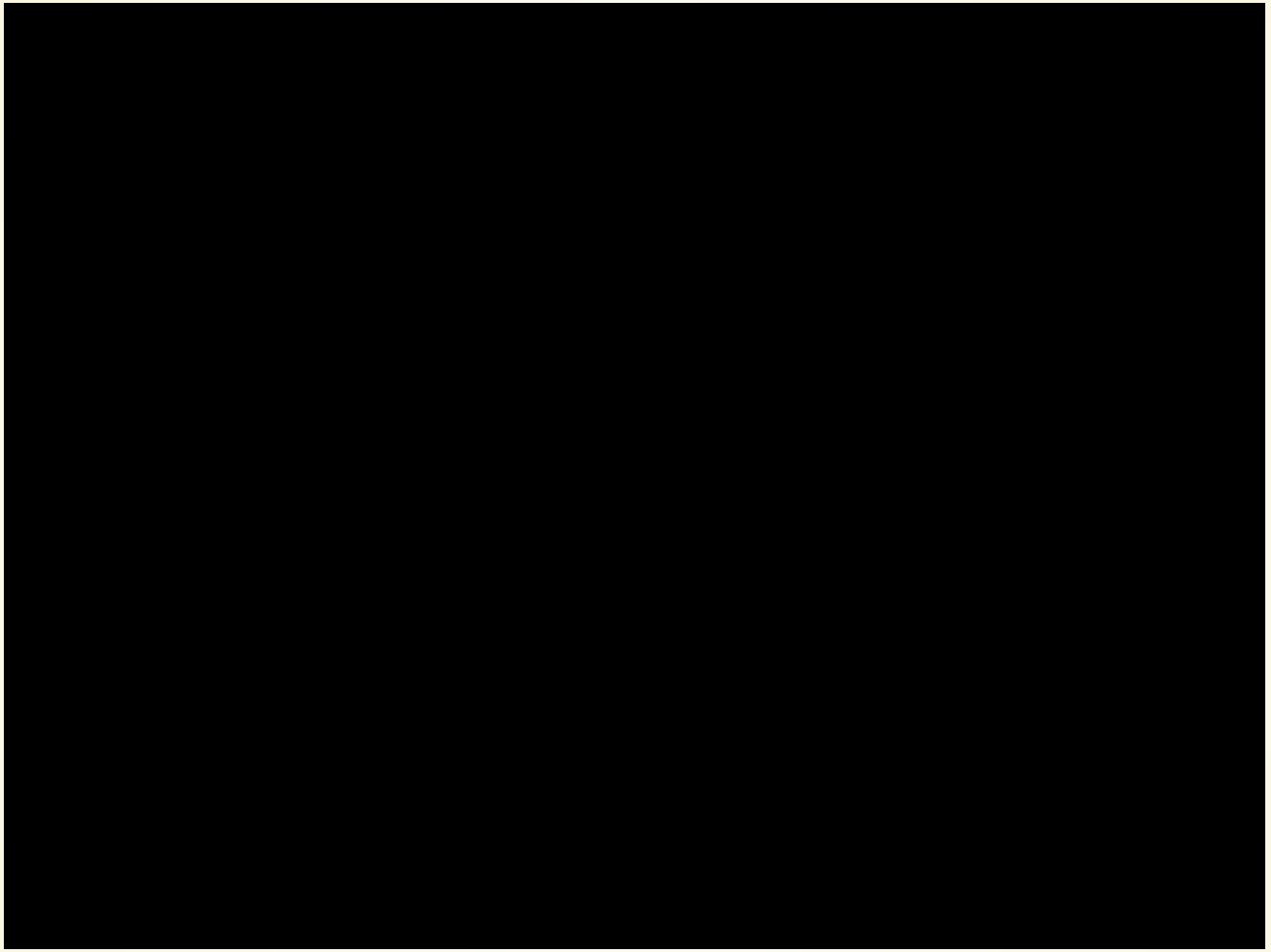
run code | reload original code

```
Output:
```


Memory (RAM)
p1
x
y
3
4
p2
x
y
5
6
Instruction Stack
p1 = {x : 3, y : 4};
p2 = {x : 5, y : 6};

Here we have seen that when we create a new 'name', such as 'var a' and follow it by an 'assignment' such as 'var a = p;', the result in Javascript depends on whether 'p' is a value, such as 5, or an object such as {x:4, y:5}. When an object is passed as an argument into a function it is the "reference" to the object that is being passed. Even though the function makes a "copy" of the reference, it still points to the same memory (the same object). This means that if we modify the contents of the object in the function we are "operating" on the actual object. There can be many "references" pointing to the same object.

POINT OBJECT CREATION



CREATING OBJECTS

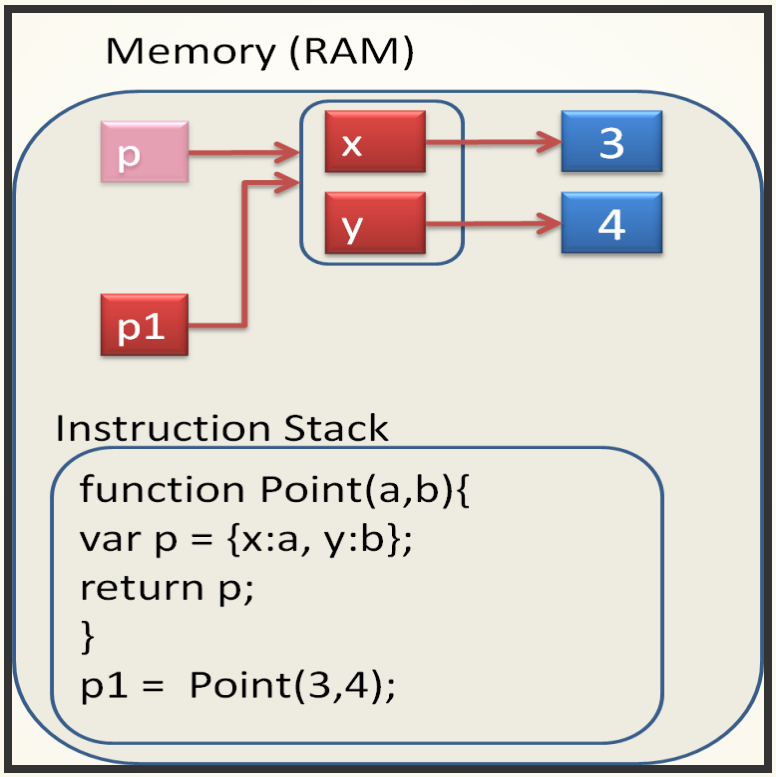
We can also create an object by creating it within a function and then returning the reference to it.

```
1 function Point(a,b){
2   var p = {x:a, y:b};
3   return p;
4 }
5 var p1 = Point(3,4);
6
7
```

run code | reload original code

```
Output:
```

Notice that the reference p created in the Point function disappears once we return from the function. However, p1 references the object and so the object continues to exist and we have a reference to it. ie p1

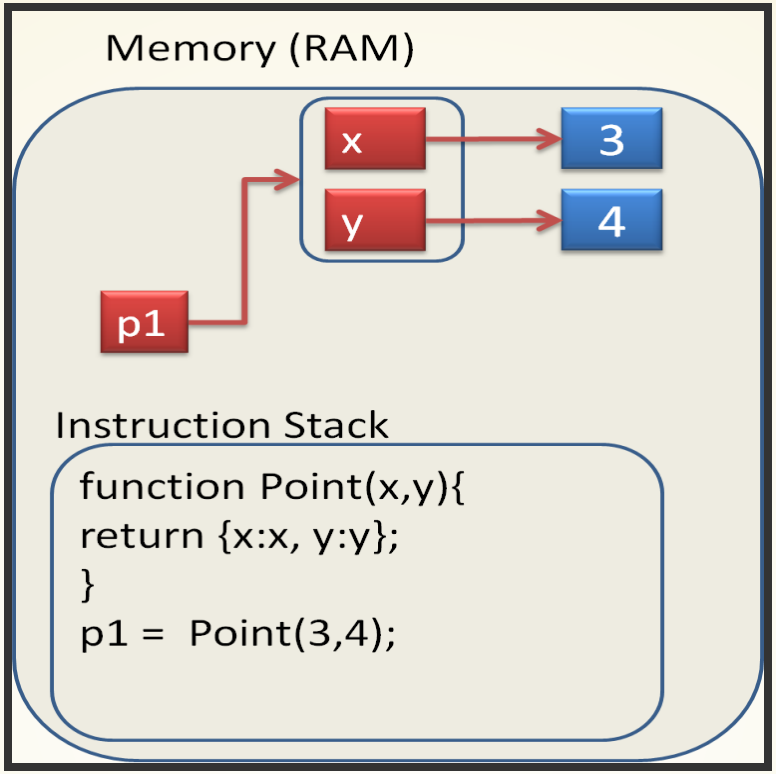


We can make the Point function more compact by having the object created in the return statement. This avoids having to create the temporary variable 'p'. Notice also that in {x:x, y:y} the first x is the **name** of the property and the second x is the **argument** that has been passed in. ie 3

```
1 function Point(x,y){
2   return {x:x, y:y};
3 }
4 var p = Point(3,4);
5
```

run code | reload original code

```
Output:
```


Memory (RAM)
x
y
3
4
p1
Instruction Stack
function Point(x,y){
return {x:x, y:y};
}
p1 = Point(3,4);

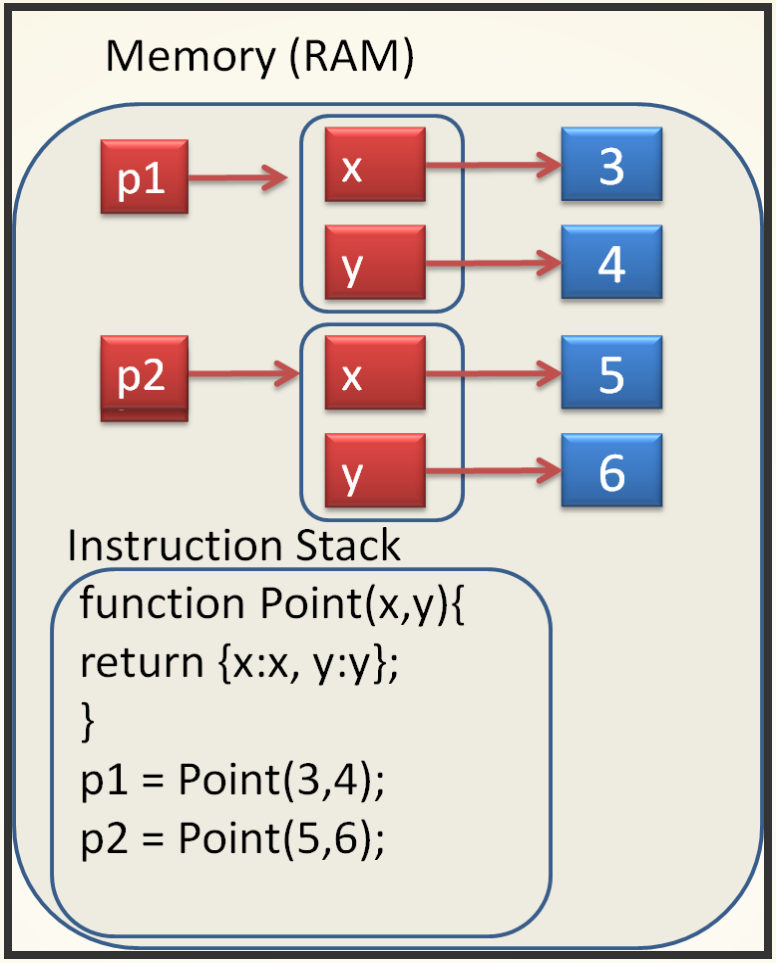
Now we can create multiple Point objects. Our function Point acts like a little "factory" that can make Point objects. Why don't you make one with the coordinates (7,9).

```
function Point(x,y){
  return {x:x, y:y};
}
var p1 = Point(3,4);
var p2 = Point(5,6);


```

run code | reload original code

Output:


Memory (RAM)
p1
x
y
3
4
p2
x
y
5
6
Instruction Stack
function Point(x,y){
return {x:x, y:y};
}
p1 = Point(3,4);
p2 = Point(5,6);


your turn now

Change the values of the properties of p1 and convince yourself that the properties of p1 are independent of the properties of p2. This means that p1 and p2 reference different Point objects.

```
1 function Point(x,y){
2   return {x:x, y:y};
3 }
4 var p1 = Point(3,4);
5 var p2 = Point(5,6);
6
7
```

run code | reload original code

```
Output:
```

ASSIGNMENT

Type this code into your Chrome Console. What is the value of 'a' and 'p1.x' after Calc() has been executed.

```
function Point(x,y){
  return {x:x, y:y};
}
var p1 = Point(3,4);
var a = 7;
var b = Calc(a,p1);
console.log(a, p1.x);
function Calc(a, p){
    a= 99;
    p.x = 24;
    return a;
}
```

run code | reload original code

Output:

THE END