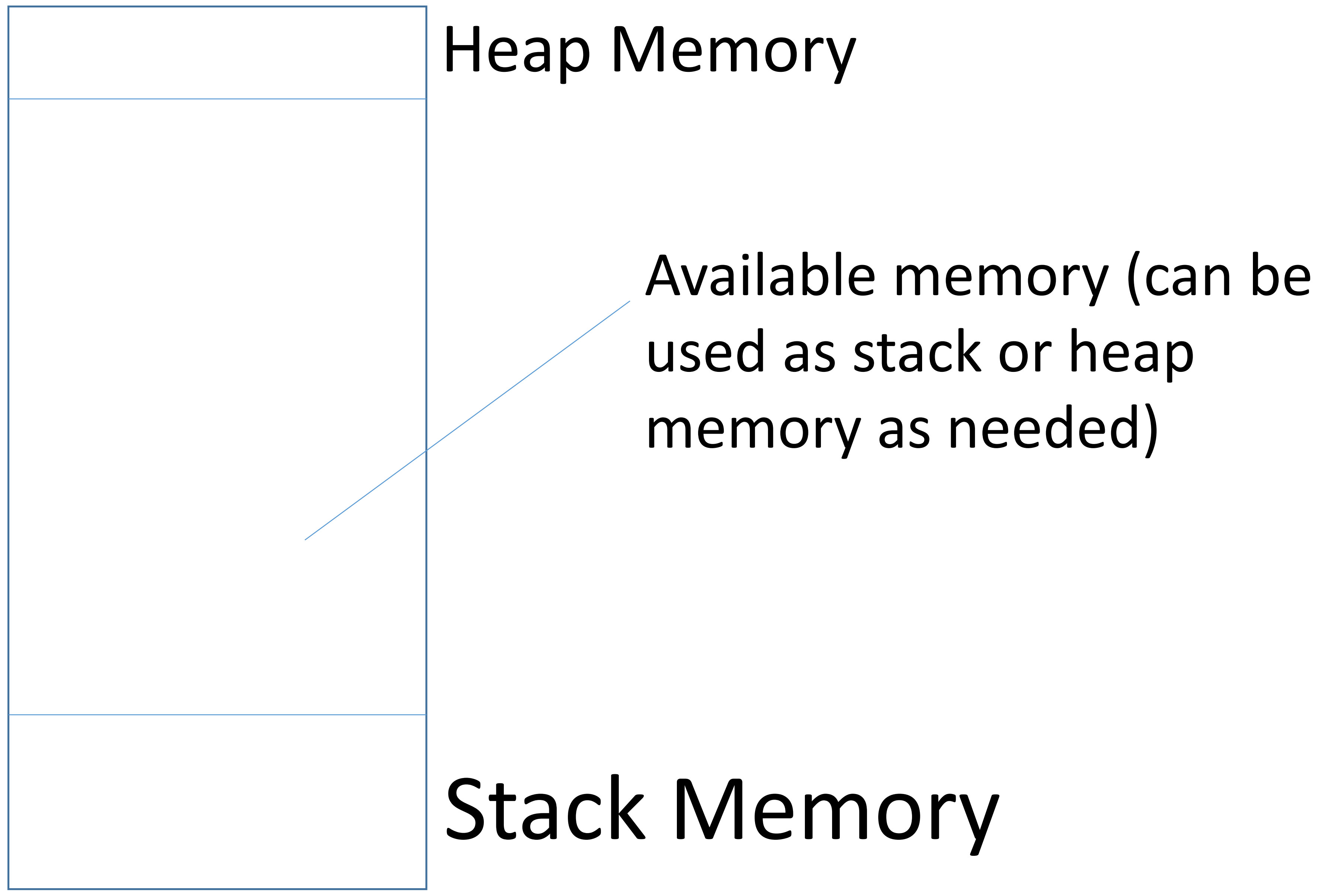
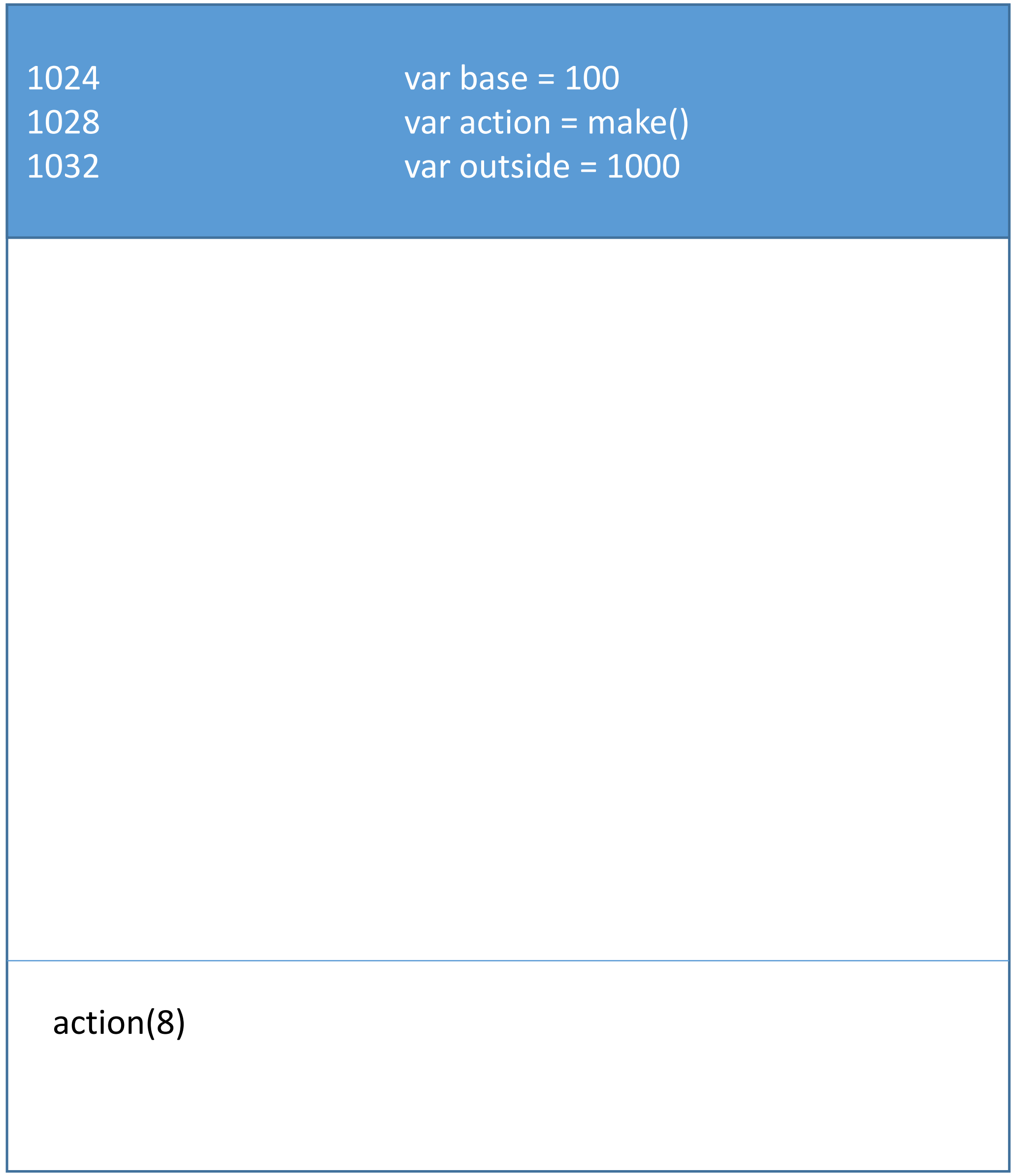
Javascript Closures

By John Diyala

*pseudo-code was used for a simpler explanation


Heap Memory
Available memory (can be used as stack or heap memory as needed)
Stack Memory



1024	var base = 100
1028	var action = make()
1032	var outside = 1000

action(8)

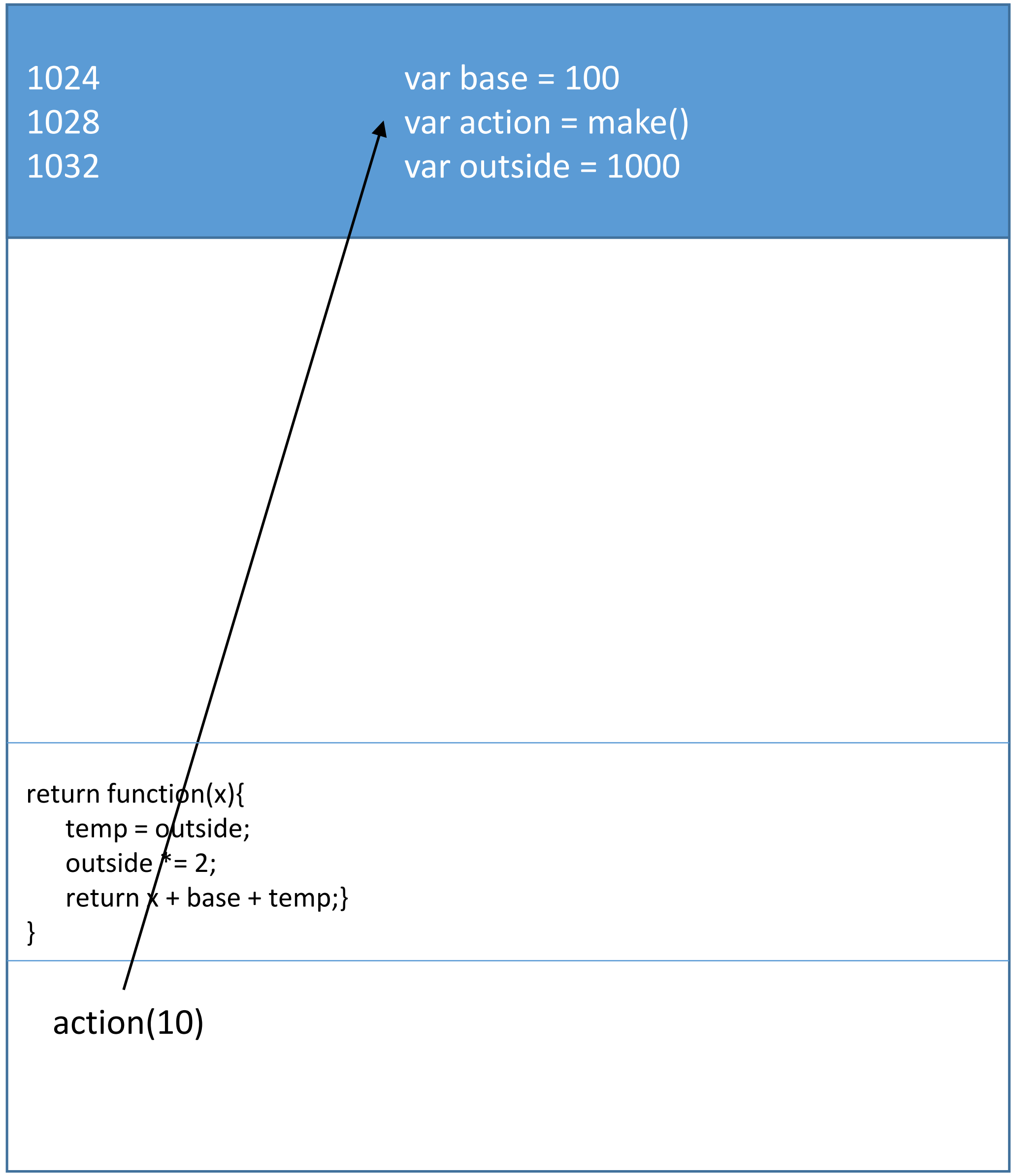
Since a new variable is created “var action = make”, this allows for make() to be Stored in heap memory

```
var base = 100;   //line 1
function make(){
  var outside = 1000;
  return function(x){
    temp = outside;
    outside *= 2;
    return x + base + temp;}
}

var action = make();

console.log(action(8));
console.log(action(8));
```

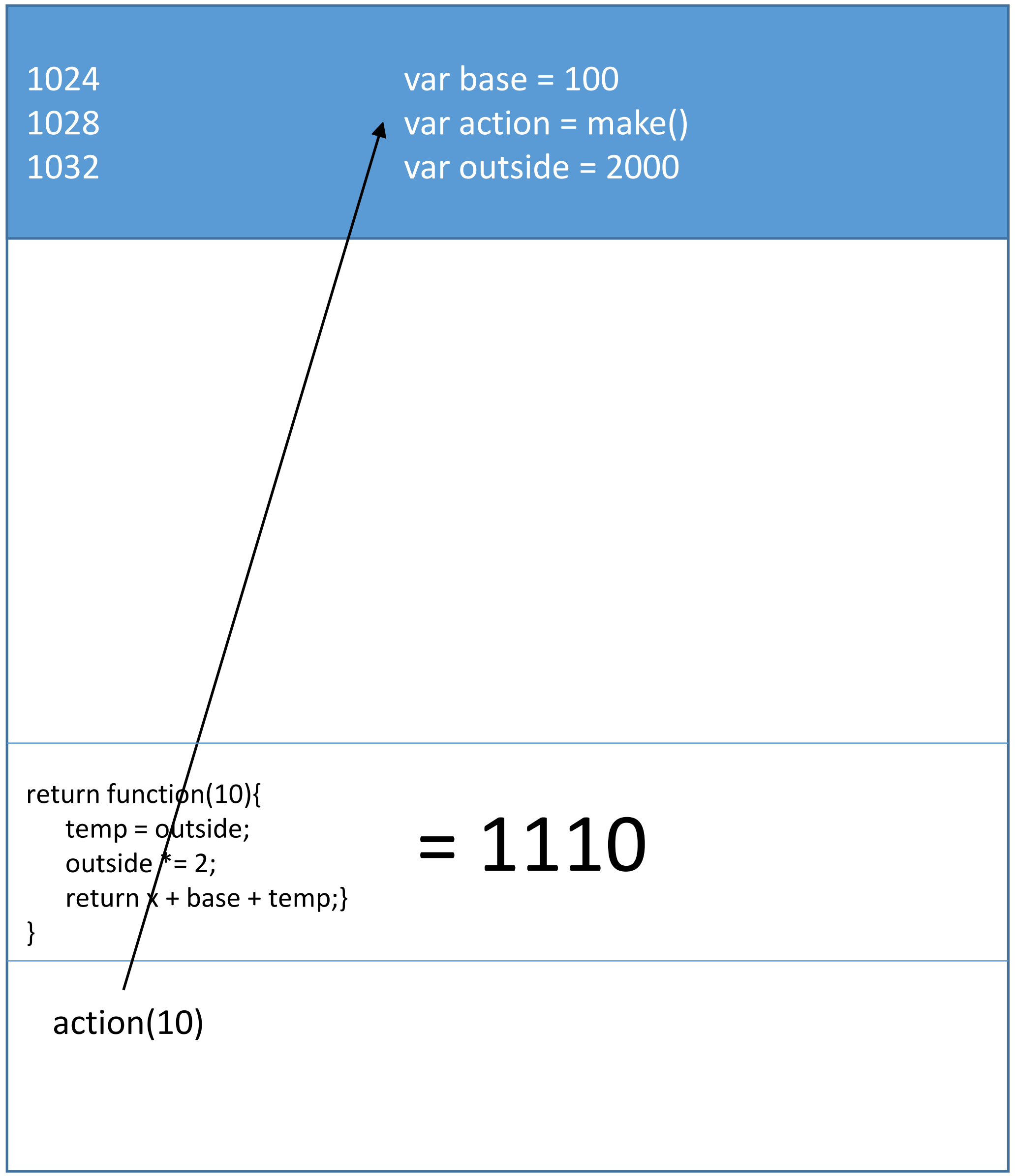
action(8) is called, so it is put on the call stack



When the self-invoking function:

```
return function(x){
    temp = outside;
    outside *= 2;
    return x + base + temp;}
}
```

Is reached, it is put on the stack and has the ability to access and edit the variables outside its scope since make() is stored in heap memory.



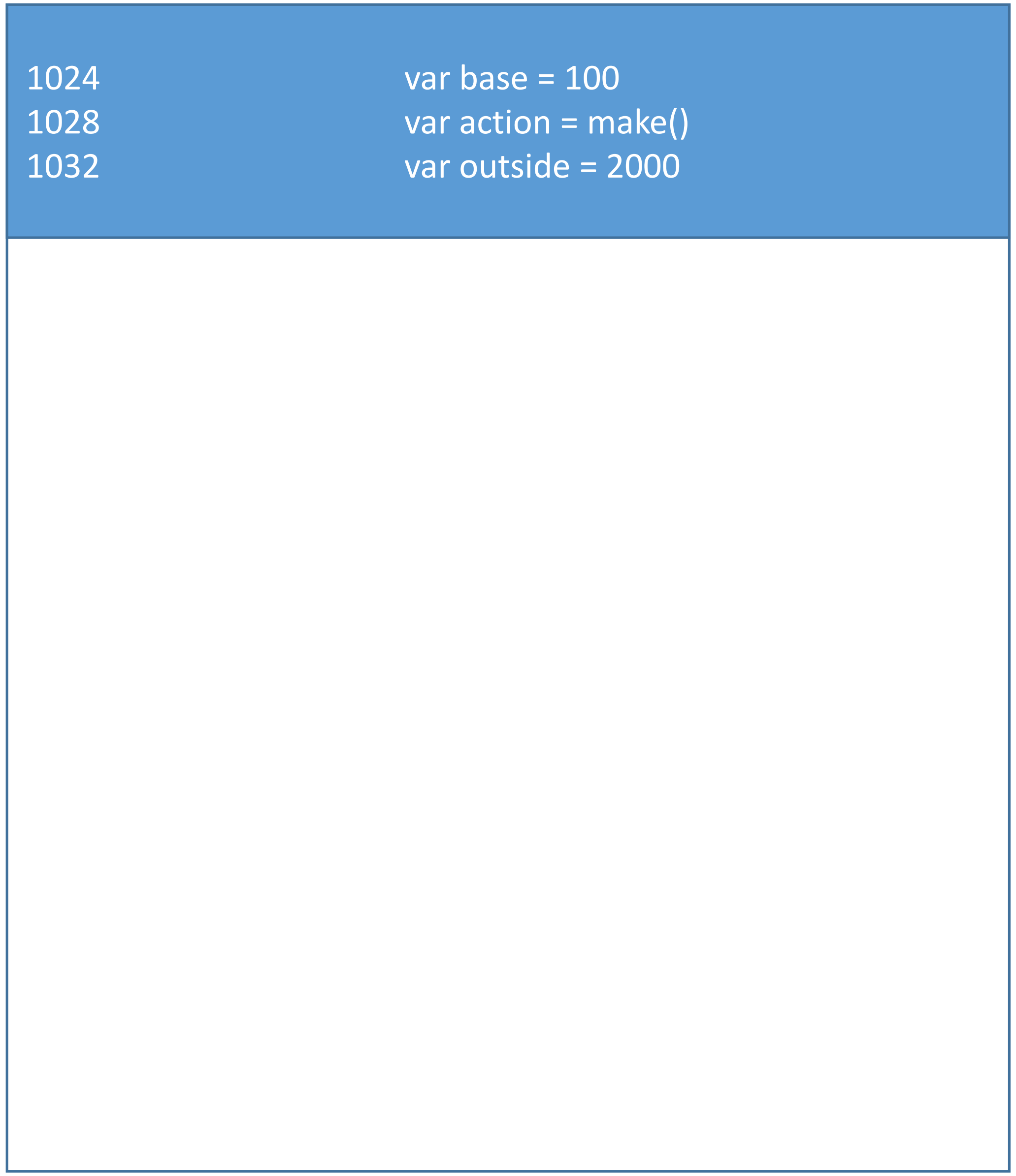
```
1024        var base = 100
1028        var action = make()
1032        var outside = 2000
```

```
return function(10){
    temp = outside;
    outside *= 2;
    return x + base + temp;}
}
```

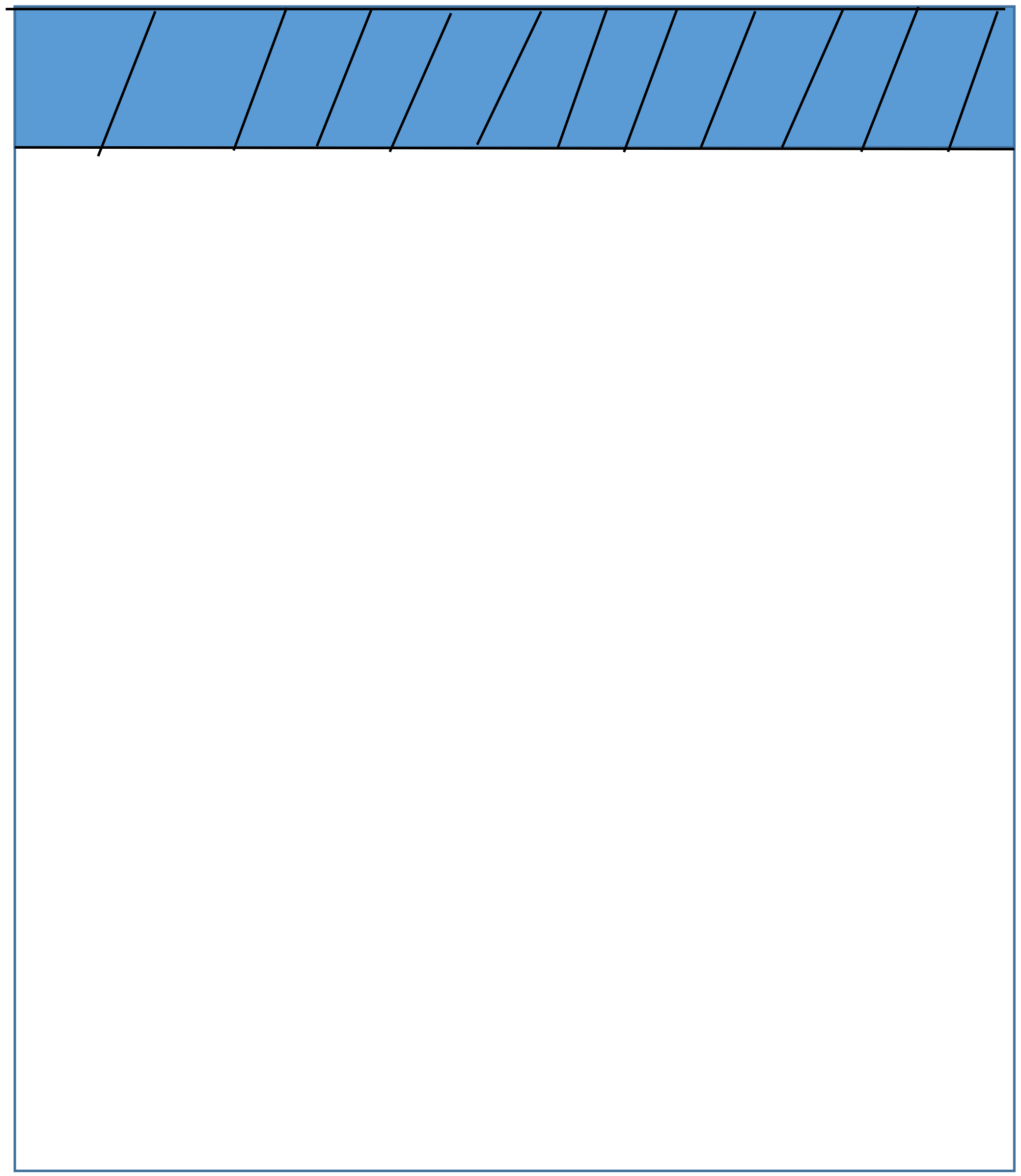
= 1110

```
action(10)
```

function(10) returns 1110, but doubles the value of var outside in the meantime, so the second call of this function would return 2110


```
1024                var base = 100
1028                var action = make()
1032                var outside = 2000
```

Function(x) and action(x) are popped off the stack since they are done executing.



Since there are no references to the variables on the heap anymore, they are also freed by the garbage collector.

R3.3)

- Describe what modifications to the stack and heap model provided, if any, are necessary to convey how recursion works with javascript functions. That is, can you illustrate your recursion using the model provided or does it fall short and require modifications or new features?

- In order to be able to access variables outside a function's scope from within that function, I had to modify the stack/heap to store variables and the function make() in the heap. This way, any other function within make() is able to edit and access the variables defined outside its local scope . This wouldn't be possible if the parent function was stored in the stack, because then we would have to wait until the child function finished executing in order to reach the parent's variables – which is a dilemma because the child function needs to reach the parent's variables to finish executing.