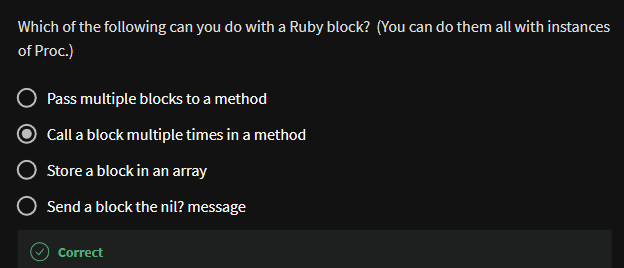
Blocks are “second-class”

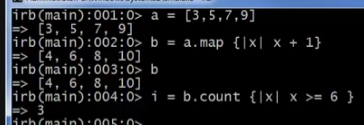
* NOT AN OBJECT
* All a method can do with a block is **yield** to it
  + Cannot return it, store it in an object (e.g., for a callback), …
  + But can also turn blocks into real closures
  + Closures are instances of class Proc
    - Called with method call
* Blocks are “second-class”
* Procs are “first-class expressions”
  + We can return it
  + We can store it in an object
  + We can call methods from it



This is Ruby, so there are several ways to make **Proc** objects

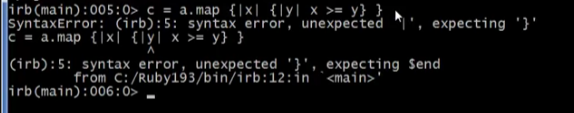
* One way: method **lambda** of Object
  + takes a block
  + returns the corresponding **Proc**

Example



* normally

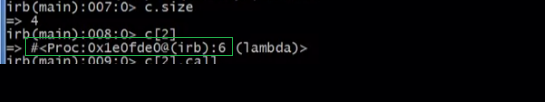
We cannot put a block inside a block:



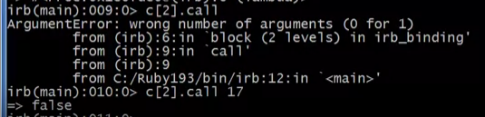
So we use **lambda** method for it



Elements inside it has a **Proc** class

****

and to invoke this, we use .call Method



* 
* This is what the call argument wants



Moral

* First-class (“can be passed/stored anywhere”) makes closure more powerful than blocks
* But blocks are (a little) more convenient and cover most uses
* This helps us understand what first-class means
* Language design question:
  + When is convenience worth making something less general and powerful?