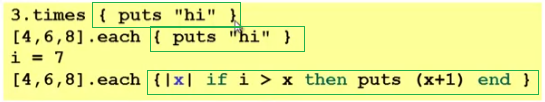
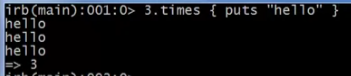
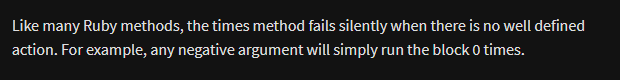
Blocks

* Strangest feature
* But almost just closures
  + Normal:
    - Easy way to pass anonymous functions to methods for all the usual reasons
  + Normal:
    - Blocks can take 0 or more arguments
  + Normal:
    - Blocks use lexical scope: block body uses environment where block was defined

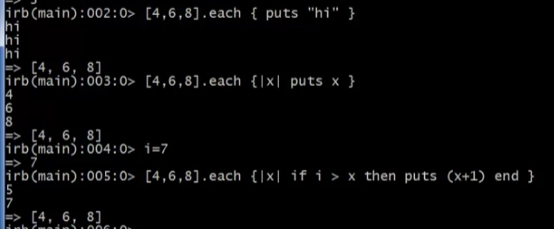
Examples:







.each Method with block



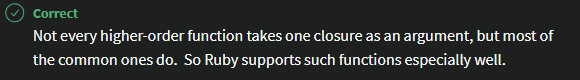
* Some what like the body of the .forEach method

Array.forEach(x => console.log(x))

Some strange things

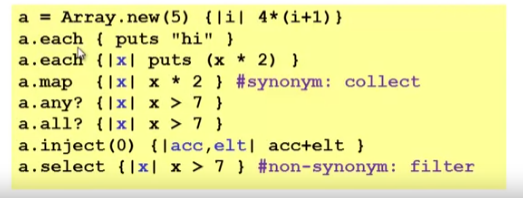
* Can pass 0 or 1 block with any message
  + Callee might ignore it
  + Callee might give an error if you do not send one
  + Callee might do different things if you do/don’t send one
    - Also, number-of-block-arguments can matter
* Just put the block “next to” the “other” arguments (if any)
  + Syntax:
    - {e}, {|x| e}, {|x,y| e}, etc. (plus variations)
  + Can also replace { and } with do and end
    - Often preferred for blocks > 1 line



* Map
* Filter
* Reduce
* Foldl
* 

Blocks everywhere

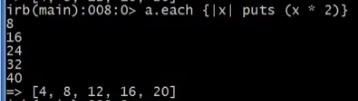
* Rampant use of great block-taking methods in standard library
* Ruby has loops but very rarely used
  + Can write (0 .. i).each { |j| e }, but often better options
* Examples



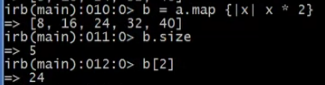
* + Creation of new array with initialization of values
    - *Array*.new(*size*) { *initialization code* }
    - Return an array



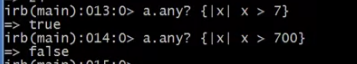
* + Printing each element of an array
    - *Array*.each { *what to do per element* }
    - Return what you do per element



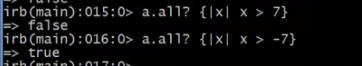
* + Mapping
    - *Array*.map { *what to map per element* }
    - Return a mapped array



* + - Sometimes *Array*.collect
  + Check is some/any are true
    - *Array*.any? { *condition per element* }
    - Return boolean



* + Check if all are true
    - *Array*.all? { *condition per element* }
    - Return Boolean



If you pass without a block



* + - Empty arrays are true!

Only false and nil return false



* + Reducing arrays
    - *Array*.inject(*initial value of accumulator*) { *|acc,elt| acc+elt* }
    - Return type of initial value



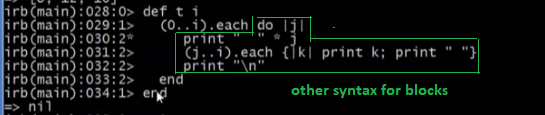
* + - If initial value of accumulator is not defined, it will use the first element’s type/class

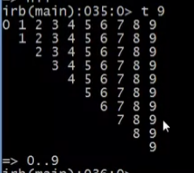


* + Filtering arrays
    - *Array*.select { *condition per element* }
    - Return a filtered array



* + Avoiding use of loops





* + - Usage of ranges
      * (first element…last element)