

[Courseware \(/courses/BerkeleyX/CS169.1x/2013_Spring/courseware\)](/courses/BerkeleyX/CS169.1x/2013_Spring/courseware/)[Course Info \(/courses/BerkeleyX/CS169.1x/2013_Spring/info\)](/courses/BerkeleyX/CS169.1x/2013_Spring/info/)[Syllabus \(/courses/BerkeleyX/CS169.1x/2013_Spring/syllabus/\)](/courses/BerkeleyX/CS169.1x/2013_Spring/syllabus/)[Textbook & VM \(/courses/BerkeleyX/CS169.1x/2013_Spring/textbook_vm/\)](/courses/BerkeleyX/CS169.1x/2013_Spring/textbook_vm/)[Tutorials & Resources \(/courses/BerkeleyX/CS169.1x/2013_Spring/tutorials_resources/\)](/courses/BerkeleyX/CS169.1x/2013_Spring/tutorials_resources/)[Discussion \(/courses/BerkeleyX/CS169.1x/2013_Spring/discussion/forum\)](/courses/BerkeleyX/CS169.1x/2013_Spring/discussion/forum/)[Wiki \(/courses/BerkeleyX/CS169.1x/2013_Spring/course_wiki\)](/courses/BerkeleyX/CS169.1x/2013_Spring/course_wiki/)[Progress \(/courses/BerkeleyX/CS169.1x/2013_Spring/progress\)](/courses/BerkeleyX/CS169.1x/2013_Spring/progress/)

HW 1-7: ITERATORS, BLOCKS, YIELD

Given two collections (of possibly different lengths), we want to get the Cartesian product of the sequences. A Cartesian product is a sequence that enumerates every possible pair from the two collections, where the pair consists of one element from each collection. For example, the Cartesian product (denoted by \times) of the sequences

`a = [:a, :b, :c]` and `b = [4, 5]` is:

```
a × b = [ [:a,4], [:a,5], [:b,4], [:b,5], [:c,4], [:c,5] ]
```

Create a constructor for the class `CartesianProduct` that takes two sequences as arguments, these values will define the behavior of your object. Define `each` as an instance method for `CartesianProduct`. Your method should return an iterator which yields the cartesian product of the two sequences used in the class' constructor. The iterator should yield the values one at a time as 2 element arrays.

It doesn't matter what order the elements are returned in. So for the above example, the ordering

`[[:a,4], [:b,4], [:c,4], [:a,5], [:b,5], [:c,5]]` would be correct as well.

It does matter that within each pair, the order of the elements matches the order in which the original sequences were provided. That is, `[:a,4]` is a member of the Cartesian product `a × b` but `[4,:a]` is not. (Instead, `[4,:a]` is a member of the Cartesian product `b × a`.)

Below is the code skeleton:

```
class CartesianProduct
  include Enumerable
  # Your code here
end
```

Example test cases:

```
c = CartesianProduct.new([:a,:b], [4,5])
c.each { |elt| puts elt.inspect }
# [:a, 4]
# [:a, 5]
# [:b, 4]
# [:b, 5]

c = CartesianProduct.new([:a,:b], [])
c.each { |elt| puts elt.inspect }
# Nothing printed since Cartesian product of anything with an empty collection is empty
```

[Choose Files](#) No file chosen

Check

Show Discussion

New Post



[Find Courses \(/courses\)](/courses) [About \(/about\)](/about) [Blog \(http://blog.edx.org/\)](http://blog.edx.org/) [Jobs \(/jobs\)](/jobs) [Contact \(/contact\)](/contact)



(<http://youtube.com/user/edxonline>)



(<https://plus.google.com/108235383044095082735>)



(<http://www.facebook.com/EdxOnline>)



(<https://twitter.com/edXOnline>)