* Filter
  + Removes elements from a sequence which don’t meet some criteria
  + Applies a predicate function to each element
  + Produces its results lazily
  + Only accepts a single input sequence, and the function must accept only one argument
  + filter(function, sequence)
    - return iterable object of type filter
    - iterable where function returns true
  + ex)
  + positives = filter(lambda x: x > 0, [1, -5, 0]
  + Passing None as the first argument to filter() will filter out input elements which evaluate to False
  + Map() and filter() behave differently in python 2 and python 3
  + In python 3, they are lazily evaluated
  + In python 2, they are eager and return lists
* Reduce
  + Repeatedly applies a two-argument function to an accumulated value and the next element from a sequence
  + The initial value can be the first element in the input sequence or an optional argument
  + The final accumulated - or reduced - value is returned
  + Same as reduce() and Aggregate() in other functional languages
  + Ex)
    - from functools import reduce
    - import operator
    - reduce(operator.add, [1,2,3,4,5]) #return 15
  + if you pass in empty sequence to reduce it will return a TypeError
  + if you pass in a sequence with only one element, that element is return will out calling reduce
  + reduce() accepts an optional initial value
    - conceptually added to the start of the sequence
    - serves as the first accumulator value
  + ex)
    - values = [1, 2, 3]
    - reduce(operator.add, values, 0) #return 6
    - values = []
    - reduce(operator.add, value, 0) #return 0
    - values = [1, 2, 3]
    - reduce(operator.mul, value, 1) #returns 6
* Combining the Tools
  + Map-reduce
    - map() and reduce() are related to map-reduce
    - they are the core concepts in the algorithm
  + map() applies a callable to each element in a sequence
  + map() produces it results lazily
  + map() can accept multiple input iterables
  + filter() applies a predicate to the elements of an iterable
  + it produces an iterable containing the input elements for which the predicate returned True
  + functools.reduce()
    - repeatedly applies a two-argument callable to accumulate the elements in an iterable
    - raises an exception on empty input iterables
    - you can provide an initial value to avoid this issue
    - selecting the right initial value is crucial
  + combining map() and reduce() to make map-reduce
* Multi-input Comprehensions