dnaseq.py

```
# Maps integer keys to a set of arbitrary values.
class Multidict:
  # Initializes a new multi-value dictionary, and adds any key-value
  # 2-tuples in the iterable sequence pairs to the data structure.
  def __init__(self, pairs=[]):
    raise Exception("Not implemented!")
  # Associates the value v with the key k.
  def put(self, k, v):
    raise Exception("Not implemented!")
  # Gets any values that have been associated with the key k; or, if
  # none have been, returns an empty sequence.
  def get(self, k):
    raise Exception("Not implemented!")

# Given a sequence of nucleotides, return all k-length subsequences
# and their hashes.  (What else do you need to know about each
# subsequence?)
def subsequenceHashes(seq, k):
  raise Exception("Not implemented!")

# Similar to subsequenceHashes(), but returns one k-length
    subsequence
# every m nucleotides.  (This will be useful when you try to use two
# whole data files.)
def intervalSubsequenceHashes(seq, k, m):
  raise Exception("Not implemented!")

# Searches for commonalities between sequences a and b by comparing
# subsequences of length k.  The sequences a and b should be
    iterators
# that return nucleotides.  The table is built by computing one hash
# every m nucleotides (for m >= k).
def getExactSubmatches(a, b, k, m):
  raise Exception("Not implemented!")

if __name__ == '__main__':
  if len(sys.argv) != 4:
    print 'Usage: {0} [file_a.fa] [file_b.fa] [output.png]'.format(
        sys.argv[0])
    sys.exit(1)

  # The arguments are, in order: 1) Your getExactSubmatches
  # function, 2) the filename to which the image should be written,
  # 3) a tuple giving the width and height of the image, 4) the
  # filename of sequence A, 5) the filename of sequence B, 6) k, the
  # subsequence size, and 7) m, the sampling interval for sequence
  # A.
  compareSequences(getExactSubmatches, sys.argv[3], (500,500), sys.
      argv[1], sys.argv[2], 8, 100)
```

dnaseqlib.py

```python
# Produces hash values for a rolling sequence.
class RollingHash:
    def __init__(self, s):
        self.HASH_BASE = 7
        self.seqlen = len(s)
        n = self.seqlen - 1
        h = 0
        for c in s:
            h += ord(c) * (self.HASH_BASE ** n)
            n -= 1
        self.curhash = h

    # Returns the current hash value.
    def current_hash(self):
        return self.curhash

    # Updates the hash by removing previtm and adding nextitm. Returns the updated
    # hash value.
    def slide(self, previtm, nextitm):
        self.curhash = (self.curhash * self.HASH_BASE) + ord(nextitm)
        self.curhash -= ord(previtm) * (self.HASH_BASE ** self.seqlen)
        return self.curhash
```

```python
def compareSequences(getExactSubmatches, imgfile, imgsize, afile, bfile, k, m):
    a = kfasta.FastaSequence(afile)
    b = kfasta.FastaSequence(bfile)
    matches = getExactSubmatches(a, b, k, m)
    buildComparisonImage(imgfile, imgsize[0], imgsize[1],
                         kfasta.getSequenceLength(afile),
                         kfasta.getSequenceLength(bfile), matches)
```

kfasta.py

```
# An iterator that returns the nucleotide sequence stored in the
    given FASTA file.
class FastaSequence:
    def __init__(self, filename):
        self.f = open(filename, 'r')
        self.buf = ''
        self.info = self.f.readline()
        self.pos = 0
    def __iter__(self):
        return self
    def next(self):
        while '' == self.buf:
            self.buf = self.f.readline()
            if '' == self.buf:
                self.f.close()
                raise StopIteration
            self.buf = self.buf.strip()
        nextchar = self.buf[0]
        self.buf = self.buf[1:]
        self.pos += 1
        return nextchar
```

Iterators vs Generators

```
class Reverse:
    """Iterator for looping over a sequence backwards."""
    def __init__(self, data):
        self.data = data
        self.index = len(data)
    def __iter__(self):
        return self
    def next(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]

# >>> rev = Reverse('spam')
# >>> iter(rev)
# <__main__.Reverse object at 0x00A1DB50>
# >>> for char in rev:
# ...     print char
# ...
# m
# a
# p
# s
```

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]

# >>> for char in reverse('golf'):
# ...     print char
# ...
# f
# l
# o
# g
```

```
>>> data = 'golf'
>>> list(data[i] for i in range(len(data)-1,-1,-1))
['f', 'l', 'o', 'g']

>>> sum(i*i for i in range(10))                 # sum of squares
285

>>> xvec = [10, 20, 30]
>>> yvec = [7, 5, 3]
>>> sum(x*y for x,y in zip(xvec, yvec))         # dot product
260
```