```
#! /usr/bin/python
from __future__ import division
import math,re,time,sys,os,random
from Bio import Seq as seq
from Bio import SeqIO as seqio
import urllib
import re

from Bio import SeqUtils

fl = sys.stdout.flush

class ORF():

    def __init__(self,name,c1,c2,strand):
        self.name = name
        c1 = int(c1)
        c2 = int(c2)
        if strand == '+':
            self.start = min([c1,c2])
            self.end = max([c1,c2])
        elif strand == '-':
            self.start = max([c1,c2])
            self.end = min([c1,c2])
        else: raise ValueError('Strand neither + nor -')
        self.strand = strand
        self.regions = []
        self.five_prime_UTR = []
        self.three_prime_UTR = []

    def __hash__(self):
        return int(self.name[2:])

    def coords(self):
        c = [self.start,self.end]
        c.sort()
        return c

    def start_end_list(self,c1c2):
        return self.start_end(c1c2[0],c1c2[1])

    def start_end(self,c1,c2):
        if self.strand == '+':
            start = min([c1,c2])
            end = max([c1,c2])
        elif self.strand == '-':
            start = max([c1,c2])
            end = min([c1,c2])
        else: raise ValueError('Strand neither + nor -')
        return [start,end]
```

```
    def add_region(self,typeof,c1,c2):
        c1 = int(c1)
        c2 = int(c2)
        self.regions.append([typeof,self.start_end(c1,c2)])

    def add_UTR(self,typeof,c1,c2):
        c1 = int(c1)
        c2 = int(c2)
        if typeof == 'three_prime_UTR':
            self.three_prime_UTR = self.start_end(c1,c2)
        elif typeof == 'five_prime_UTR':
            self.five_prime_UTR = self.start_end(c1,c2)

    def ORF_region(self):
        return [self.start,self.end]

    def get_full_length(self):
        return self.start_end_list(self.five_prime_UTR +
self.three_prime_UTR + self.ORF_region())

    def is_in_range(self,r,coord):
        r.sort()
        if coord < r[0] or coord > r[1]:
            return False
        else:
            return True

    def is_exon(self,coord):
        for item in self.regions:
            if self.is_in_range(item[1],coord) and item[0] == 'CDS':
                return item[1]
        return []

    def is_intron(self,coord):
        for item in self.regions:
            if self.is_in_range(item[1],coord) and item[0] ==
'intron':
                return item
        return False

    def is_UTR(self,coord):
        if self.three_prime_UTR:
            if self.is_in_range(self.three_prime_UTR,coord):
                return 'three_prime_UTR'
        if self.five_prime_UTR:
            if self.is_in_range(self.five_prime_UTR,coord):
                return 'five_prime_UTR'
        return ''
```

```
    def get_coding_sequence(self,chr,debug = False):
        final = seq.Seq('')
        # slices of seq object are done this way: AACAAC [1:4] gives
ACA
        self.regions.sort(key = lambda x: x[1][0])

        for item in self.regions:
            if item[0] == 'CDS':
                final = final + chr.seq[item[1][0] - 1: (item[1]
[1] ) ] # subtracted 1 here ; test!

        if self.strand == '-':
            final = final.reverse_complement()

        if debug:
            # test whether it is of the right length
            length = 0
            for item in self.regions:
                if item[0] == 'CDS':
                    length += abs(item[1][1] - item[1][0] + 1)
            if length - len(final) != 0:
                print 'expected - returned SHOULD BE ZERO:', (length-
len(final)),
            print ',',
            fl()

        if str(final) == '':
            print 'BREAK BREAK BREAK'
        return final

    def get_transcript_position(self,coord):
        dist = 0 # 1-indexed
        if self.strand == '+':
            self.regions.sort(key = lambda x: x[1][0])
        else:
            self.regions.sort(key = lambda x: -x[1][0])
        for item in self.regions:
            if item[0] == 'CDS':
                if self.is_in_range(item[1],coord):
                    if self.strand == '-':
                        dist += abs(coord - item[1][1]) + 1
                    else:
                        dist += abs(coord - item[1][0]) + 1
                    break
                else:
                    if self.strand == '-':
                        pass
                    dist += (1 + abs(item[1][0] - item[1][1]))
        return dist
```

```
    def test_transcript_position(self,coord,chr):
        internal_pos = self.get_transcript_position(coord)
        internal_seq = self.get_coding_sequence(chr)
        used =  str(internal_seq[internal_pos-5:internal_pos+5].seq)
        external_seq = chr[coord-4 : coord+6]
        if self.strand == '-':
            external_seq = external_seq.reverse_complement()
        if str(external_seq.seq) != used:
            print 'not equal'
            print used
            print str(external_seq.seq)
            print coord
            print internal_pos
            j = []
            start = 10000000000
            print self.regions
            for item in self.regions:
                for this in item[1]:
                    if int(this) < start:
                        start = int(this)
            for item in self.regions:
                j.append([])
                j[-1].append(int(item[1][0]) - start)
                j[-1].append(int(item[1][1]) - start)
            for item in self.regions:
                for this in item[1]:
                    if abs(int(this)-coord)<10:
                        print abs(int(this)-coord), "YES"
            print j
            print ''

def make_index_from_gff(gff_loc):
    '''Argument is file handle to gff file. Returns dict-of-lists:

DICT 1: chromosome, LIST

LIST: ORFs # [start, end] , [ region 1 : start,end,kind ] , [ region
2 : start,end,kind ] ... '''

    feature_dict = {}
    no_ORF_yet = [] # store lines in case we have to do categorization
a second time

    # separate the file by lines and tabs
    gff_loc = open(gff_loc,'r')
    file_contents = [ line.split('\t') for line in gff_loc.readlines()
if not line.startswith('#') ]
    for item in range(0,len(file_contents)):
        file_contents[item][0] = sensibleChroms(file_contents[item]
[0])
```

```
    # line[2] has feature type: chromosome / contig [not helpful], ORF
['parent'], three_prime_UTR, five_prime_UTR, CDS, intron
    # catch each alternative
    count = 0

    for line in file_contents:

        # update on progress
        count += 1
        if count % 10000 == 0:
            print count
            fl()

        #do actual work
        if line[2] in ['chromosome','contig']:
            # this makes it work ONLY BECAUSE chroms/contigs are at
start of file. if this changes, need to change following code.
            if sensibleChroms(line[0]) not in feature_dict:
                feature_dict[sensibleChroms(line[0])] = []
        elif line[2] == 'ORF':
            this_ORF =
ORF(getTokenFromInformation(line[8],'ID'),int(line[3]),int(line[4]),li
ne[6],getTokenFromInformation(line[8],'Name'),getTokenFromInformation(
line[8],'Note')) # name, coord1, coord3, strand
            if sensibleChroms(line[0]) in feature_dict:
                feature_dict[sensibleChroms(line[0])].append(this_ORF)
            else:
                feature_dict[sensibleChroms(line[0])] = [ this_ORF ]

        elif line[2] in ['CDS','intron']:
            for item in
range(len(feature_dict[sensibleChroms(line[0])])-1,-1,-1):
                if ( getTokenFromInformation(line[8],'Parent') ==
feature_dict[sensibleChroms(line[0])][item].name ):
                    feature_dict[sensibleChroms(line[0])]
[item].add_region(line[2],line[4],line[3])
                    break
        elif line[2][-3:] == 'UTR':
            coords = [ int(c) for c in line[3:4] ]
            for item in
range(len(feature_dict[sensibleChroms(line[0])])-1,-1,-1):
                span = feature_dict[sensibleChroms(line[0])]
[item].coords()
                if max(coords) + 1 == span[0]:
                        feature_dict[sensibleChroms(line[0])]
[item].add_UTR(line[2],line[4],line[3])
                        break
                elif min(coords) - 1 == span[1]:
                        feature_dict[sensibleChroms(line[0])]
[item].add_UTR(line[2],line[4],line[3])
```

```
                        break
    return feature_dict

def get_chroms(file):
    '''load chromosomes from fasta file called file
Returns list of SeqRecords'''
    chroms = []
    fasta = seqio.parse(file,'fasta')
    while True:
        try:
            chroms.append(fasta.next())
        except StopIteration:
            break
    for chrom in range(0,len(chroms)):
        chroms[chrom].name = sensibleChroms(chroms[chrom].name)
        chroms[chrom].description =
sensibleChroms(chroms[chrom].description)
        chroms[chrom].id = sensibleChroms(chroms[chrom].id)

    return chroms

def get_particular_chrom(chroms,id):
    ''' finds the chromosome with (sanitized) name ID, returns it'''
    for chr in chroms:
        if sensibleChroms(chr.id) == str(id):
            return chr

def findref(codon,ref,bloc=0):
    loc = str(codon).find(ref)
    if loc != -1:
        print loc+bloc
        fl()
        findref(codon[loc+1:],ref,bloc=loc)
    else:
        return

def is_synonymous(coord,chrom,ref,alt,orf_dictionary,debug=False):
    '''arguments: coordinate, chromosome, reference,
            alternate nucleotide.
       Returns [ref_aa,alt_aa,nonsynonymous/synonymous/splice/intron/
3-UTR/5-UTR,name,annotation].
'''
    # get chromosome of interest
    chrom.id = sensibleChroms(chrom.id,show=True)

    for item in orf_dictionary[chrom.id]: # lookup by chromosome
        coords = item.get_full_length()
        coords.sort()
        if coord > coords[0] and coord < coords[1]:
            # test it
```

```
            if debug:
                if item.strand == '-':
                    ref = str(seq.Seq(ref).complement())
                    alt = str(seq.Seq(alt).complement())
            if item.is_exon(coord):
                #item.test_transcript_position(coord,chrom)
                trans = item.get_coding_sequence(chrom)
                pos = item.get_transcript_position(coord)
                # pos % 3 == 1 -- first?
                loc = pos - 1 # was -2, see also 'subtracted 1 here '
comment
                if item.strand == '-': # test
                    loc = loc
                codon = trans[loc - (loc%3): loc + 3-loc%3]
                adjcodon = codon.tomutable()
                try:
                    adjcodon[loc%3] = alt
                    adjcodon = adjcodon.toseq()
                except:
                    sys.stdout.flush()
                    time.sleep(1)
                if str(codon)[loc%3] != ref:
                    print 'STRAND ', item.strand
                    print 'CODON', str(codon)
                    print 'REF', ref, 'ALT', alt
                    print 'LOC%3',loc%3
                    print 'REGIONS',item.regions
                    print 'COORD' ,coord
                    print 'CHROM ID',chrom.id
                    for location in range(0,len(trans[loc-15:loc
+15])):
                        if str(trans[location]) == ref:
                            print location, 'YES',
str(trans[location])
                        else:
                            print location, 'NO', str(trans[location])
                #check synonymous
                refAA = str(codon.translate())
                altAA = str(adjcodon.translate())
                [refAA,altAA] =
[SeqUtils.seq3(refAA),SeqUtils.seq3(altAA)]
                if refAA == altAA:
                    return [refAA,altAA,'SILENT',item.name]
                else:
                     return [refAA,altAA,'NONSYN',item.name]
            # it's in it and coding
            elif item.is_intron(coord):
                intron = item.is_intron(coord)
                if abs(coord - intron[1][0]) < 60 or abs(coord -
intron[1][1]) < 60:
```

```
                    return ['-','-', 'SPLICE',item.name]
                else:
                    return ['-','-', 'INTRON',item.name]
            elif item.is_UTR(coord):
                return ['-','-',item.is_UTR(coord),item.name]

    return None

def run_diagnostic_test(chroms,orfs):
    random.seed()
    # choose random positions, and check if transcripts have the same
bases in those positions
    for ii in range(0,10000):
        randchrom = random.choice(chroms)
        randpos = random.randint(1,len(randchrom)) # acting like GFF
file
        # handle as if in file
        j = is_synonymous(randpos,randchrom,randchrom[randpos -
1],'G',orfs,debug=True)
    print 'random tests complete'

    print is_synonymous(2367969,get_particular_chrom(chroms,
7),'C','A',orfs,debug=True)
    print is_synonymous(3089658,get_particular_chrom(chroms,
5),'G','T',orfs)
    print is_synonymous(3089631,get_particular_chrom(chroms,
5),'G','C',orfs)
    print is_synonymous(3092196,get_particular_chrom(chroms,
5),'G','T',orfs)
    print is_synonymous(4541552,get_particular_chrom(chroms,
8),'G','A',orfs)
    print is_synonymous(4544119,get_particular_chrom(chroms,
8),'G','T',orfs)
    print is_synonymous(4546694,get_particular_chrom(chroms,
8),'G','T',orfs)


def sensibleChroms(chrom,show=False):

    RN = {1:'i',2:'ii',3:'iii',4:'iv',5:'v',6:'vi',7:'vii',8:'viii'}
    Roman_Numerals = dict([(k,v) for (v,k) in RN.iteritems()])

    chrom = str(chrom).lower()
    chr_pos = chrom.find('chr')
    if chr_pos == -1 and not re.search('mt[0-9]+',chrom):
        chr = chrom
    elif re.search('mt[0-9]+',chrom):
        chr = re.search('mt[0-9]+',chrom).group()
        #print chr
    elif re.search('chr([xvi]+)',chrom):
```

```
        chr = str( Roman_Numerals[ re.search('(?<=chr)[xvi]
+',chrom).group() ] )
    else:
        c =re.search('(?<=chr)[0-9]+',chrom)
        if chrom.find('mt') != -1:
            return re.search('mt[0-9]+',chrom).group()
        chr = str(c.group())
    return chr

class Range():
    def __init__(self,strand,chr,lower,upper):
        self.lower = min([lower,upper])
        self.upper = max([upper,lower])
        self.chr = chr
        self.strand = strand
    def __hash__(self):
        return (self.chr,self.lower,self.upper).__hash__()
    def __cmp__(self,other):
        if not isinstance(other,(int,float,long)):
            if other.chr == self.chr and other.lower >= self.lower and
other.upper <= self.upper: return 1
            elif other.chr == self.chr and other.upper >= self.upper
and other.lower <= self.lower: return 0
            elif other.chr > self.chr or other.upper > self.upper:
return -1
            elif other.chr < self.chr or other.lower < self.lower:
return 1
        else:
            if other <= self.upper and other >= self.lower: return 0
            elif other < self.lower: return 1
            else: return -1

class Bound():
    def __init__(self,strand,chr,bound,id,typeof):
        self.chr = chr
        self.bound = bound
        self.id = id
        self.type = typeof
        self.other = 0
        self.strand = strand
    def __cmp__(self,other):
#        if other.chr == self.chr and other.bound == self.bound and
other.id == self.id and self.type == other.type: return 0
        if other.chr == self.chr and other.id == self.id and self.type
== other.type: return 0
        elif other.chr < self.chr or (other.chr == self.chr and
other.bound < self.bound): return 1
        else: return -1
    def __hash__(self):
        chr_id_type = ''.join((str(ord(x)) for x in
```

```
''.join((str(self.chr),self.id,self.type))))
        return long(chr_id_type)
    def __repr__(self):
        return ''.join(("'Bound' object: ",str(self.bound),",
chromosome ", str(self.chr),", ",self.type," ",self.id))
    def __str__(self):
        return self.__repr__()

def getTokenFromInformation(info,token):
    try:
        return urllib.unquote(re.search(''.join(('(?<=',token,'=)[^;]
+')),info).group())
    except AttributeError:
        return ''

class SNP():
    def __init__(self,disp):
        self.chr = disp[4]
        self.coord = int(disp[5])
        self.id = disp[-2]
        self.disp = disp
        self.covr = int(self.disp[-4]) + int(self.disp[-3])
        self.qual = self.disp[0]
        self.gtype_alt = self.disp[7]
        self.gtype_ref = self.disp[6]
    def __eq__(self, other):
        if (self.chr == other.chr) and (self.coord == other.coord) and
(self.id == other.id) and (self.gtype_alt == other.gtype_alt): return
True
        else: return False
    def __neq__(self,other):
        return not self.__eq__(self,other)
    def __hash__(self):
        ret = ''
        let_dict = {'A':'0','C':'1','G':'2','T':'3','N':'4'}
        try:
            ret = str(int(self.id[2:]))
        except:
            ret = ''.join(str(ord(char)-40) for char in self.id)
        # if ID is the same, don't need to worry about chromosome!
        ret = ''.join((let_dict[self.gtype_alt],ret.zfill(16)))
        return int(''.join((ret,str(self.coord))))
    def __repr__(self):
        return '\t'.join(str(item) for item in self.disp)
    def __str__(self):
        return self.__repr__()
```