Python code

August 13, 2018

```
In [1]: gencode = {"TTT":"F","TTC":"F","TTA":"L","TTG":"L","TCT":"S","TCC":"S",
                      "TCA":"S","TCG":"S", "TAT":"Y","TAC":"Y","TAA":"*","TAG":"*",
                      "TGT":"C","TGC":"C","TGA":"*","TGG":"W", "CTT":"L","CTC":"L",
                      "CTA":"L","CTG":"L","CCT":"P","CCC":"P","CCA":"P","CCG":"P",
                      "CAT":"H","CAC":"H|His","CAA":"Q","CAG":"Q","CGT":"R","CGC":"R",
                      "CGA":"R","CGG":"R", "ATT":"I","ATC":"I","ATA":"I","ATG":"M",
                      "ACT":"T","ACC":"T","ACA":"T","ACG":"T", "AAT":"N","AAC":"N",
                      "AAA":"K","AAG":"K","AGT":"S","AGC":"S","AGA":"R","AGG":"R",
                      "GTT":"V","GTC":"V","GTA":"V","GTG":"V","GCT":"A","GCC":"A",
                      "GCA":"A","GCG":"A", "GAT":"D","GAC":"D","GAA":"E",
                      "GAG":"E","GGT":"G","GGC":"G","GGA":"G","GGG":"G"}


        def translate_codon(codon):
            return gencode.get(codon.upper(), 'x')

        # a function to split a sequence into codons
        def split_into_codons(dna, frame):
            codons = []
            for i in range(frame - 1, len(dna)-2, 3):
                codon = dna[i:i+3]
                codons.append(codon)
            return codons

        def translate_dna_single(dna, frame=1):
            codons = split_into_codons(dna, frame)
            amino_acids = ''
            for codon in codons:
                amino_acids = amino_acids + translate_codon(codon)
            return amino_acids

In [2]: print(translate_codon("ACT"))
        print(split_into_codons("ACGTAAGGGCCCT",3))
        print(translate_dna_single("ACGTAAGGGCCCT"))
```

```
T
['GTA', 'AGG', 'GCC']
```

```
T*GP
```

```
In [3]: def is_complement(base1, base2):
            bases = {'A':'T', 'T':'A', 'G':'C', 'C':'G'}
            if bases.get(base1)==base2:
                return True
            else:
                return False

        print(is_complement('A', 'T'))
        print(is_complement('A', 'G'))
        # The previous function works also for unknown bases
        print(is_complement('N', 'T'))

        def max_palindrome(dna, center):
            if center ==0:
                maximal_palindrome=0
                return maximal_palindrome
            else:
                for i in range(min(len(dna)-center, center)):
                    if is_complement(dna[center-i-1], dna[center+i]):
                        maximal_palindrome=2*(i+1)
                    else:
                        maximal_palindrome=2*i
                        break
                return maximal_palindrome
        print(max_palindrome(list('AGGGCCT'), 4))



        def read_fasta(file):
            fasta_file = open(file, 'r')
            DNA_list = []
            for line in fasta_file:
                    line = line.strip('\n>seq1')
                    for character in line:
                        DNA_list.append(character)
            fasta_file.close()
            return DNA_list

        print(read_fasta('smallTest.fasta'))
        # as the lists are in upper case no need to use st.upper method


        def find_palindrome(dna):
            palindrome_dict={}
            for i in range(len(dna)):
                palindrome_length =max_palindrome(dna, i)
                if palindrome_length in palindrome_dict:
                    palindrome_dict[palindrome_length].append([i])
                else:
                    palindrome_dict[palindrome_length]=[[i]]
            return palindrome_dict

        print(find_palindrome(read_fasta('smallTest.fasta')))

        def print_centers(palindromes, min_length):
            i=0
            list_of_lengths = []
            while i < min_length:
                if i in palindromes:
                    list_of_lengths.append(str(i))
                i = i+1

            return '\n'.join(list_of_lengths)

        print(print_centers(find_palindrome(read_fasta('smallTest.fasta')), 9))
```

```
True
False
False
4
['A', 'C', 'G', 'T', 'A', 'C', 'G', 'T', 'A']
{0: [[0], [1], [3], [5], [7]], 4: [[2]], 8: [[4]], 6: [[6]], 2: [[8]]}
0
2
4
6
8
```

```
In [4]: def find_longest_word(words_list):
            word_len = []
            for n in words_list:
                word_len.append((len(n), n))
            word_len.sort()
            return word_len[-1][1]

        print(find_longest_word(["PHP", "Exercises", "Backend"]))
        def is_vowel(char):
           all_vowels = 'aeiou'
           return char in all_vowels
        print(is_vowel('c'))
        print(is_vowel('e'))
```

```
Exercises
```

```
False
True
```

```
In [5]: def dict_fun(keys, values):
            d=dict()
            remaining=[]
            k=len(keys)
            v=len(values)
            if k==v:
                for i in range(k):
                    d[keys[i]]=values[i]
            elif k>v:
                for i in range(v):
                    d[keys[i]]=values[i]
                for i in range(k-v):
                    d[keys[i+v]]=None
            else:
                for i in range(k):
                    d[keys[i]]=values[i]
                for i in range(k,v):
                    remaining +=values[i]
            return d, remaining
        print(dict_fun([1, 2,3,4,5,6,7],['a','b','c','d','f','k']))
```

```
({1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'f', 6: 'k', 7: None}, [])
```

```
In [6]: def palin(words):
            words=words.casefold()
            revwo=reversed(words)
            if list(words)==list(revwo):
                print('pali')
            else:
                print('no')
        palin('madam')
        palin('isr')
        palin('nurses run')
```

```
pali
no
no
```

```
In [7]: for i in range(3):
           for j in range(3):
               if i + j % 2 == 1:
                   print('-', end='')
               elif i == 0:
                   print('+', end='')
               else:
                   print('*', end='')
           print('@'.center(12), end='')
```

```
+-+     @      -*-     @      ***     @
```

```
In [8]: def issorted(list):
           if list==sorted(list):
               return 'True'
           else:
               return sorted(list)
        print(issorted([1,2,3]))
        def remove_list(list):
           list1=[]
           for x  in list:
               if x  not in list1:
                   list1.append(x)
           return list1
        print(remove_list([1,2,3,2]))
        def common_list(l1,l4,l2):
           l3=[]
           for i in l1:
               if i in l4:
                    if i in l2:
                       l3.append(i)
           return l3
        print(common_list([1,2,3,4],[1,2,4,3],[1,5,6,4,3]))

        def elemstr(list):
           element= ''
           for i in list:
               element+=str(i)
           return element
        print(elemstr([1,2,'a','b']))
        def remdib(list):
            notdub=[]
            for i in list:
                if i not in notdub:
                    notdub.append(i)
            return notdub
        print(remdib([12,24,35,24,88,120,155,88,120,155]))
```

```
True
[1, 2, 3]
[1, 3, 4]
12ab
[12, 24, 35, 88, 120, 155]
```

```
In [9]: class Student(object):
            def __init__(self, name, age=None, major=None):
                self.name = name
                self.set_age(age)
                self.set_major(major)

            def set_age(self, age):
                self.age = age

            def set_major(self, major):
                self.major = major
        class MasterStudent(Student):
            def __init__(self, name):
                super().__init__(name)
                self.internship = 'mandatory, from March to June'
        John = MasterStudent('John')
        John.set_age(24)
        print(John.name, John.age, John.internship)
```

```
John 24 mandatory, from March to June
```

```
In [10]: class Time(object):

             #--------------------------------------------------------------------------
             # Initializer

             def __init__(self, hour = 0, minute = 0, second = 0):
                 self.hour = hour
                 self.minute = minute
                 self.second = second

             #--------------------------------------------------------------------------
             # Conversion methods

             def __str__(self):
                 fmt = '{:02d}:{:02d}:{:02d}'
                 return fmt.format(self.hour, self.minute, self.second )

             def __repr__(self):
                 return '{}.{}({},{},{})'.format(
                     __name__,
                     self.__class__.__name__,
                     self.hour, self.minute, self.second)

             #--------------------------------------------------------------------------
             # Overloaded operators
```

```python
def __add__(self, other):
    if isinstance(other, Time):
        return self.add_time(other)
    else:
        return self.increment(other)

def __radd__(self, other):
    return self.__add__(other)

def __iadd__(self, other):
    self.hour   += other.hour
    self.minute += other.minute
    self.second += other.second

    if self.second >= 60:
        self.second -= 60
        self.minute += 1

    if self.minute >= 60:
        self.minute -= 60
        self.hour   += 1

    return self

#-------------------------------------------------------------------------
# static methods

@staticmethod
def int_to_time(seconds):
    """
    Converts an integer (seconds) to a Time object
    """
    (minutes, seconds) = divmod(seconds, 60)
    (hours, minutes) = divmod(minute, 60)
    return Time(hours, minutes, seconds)

#-------------------------------------------------------------------------
# properties

@property
def hour(self):
    return self.__hour

@hour.setter
def hour(self, hour):
    assert isinstance(hour, int), 'invalid value specified for hour'
    self.__hour = hour
@property
def minute(self):
    return self.__minute

@minute.setter
def minute(self, minute):
    assert isinstance(minute, int), 'invalid type specified for minute'
    assert 0 <= minute < 60, 'invalid value specified for minute'
    self.__minute = minute
@property
def second(self):
    return self.__minute

@second.setter
def second(self, second):
    assert isinstance(second, int), 'invalid type specified for second'
    assert 0 <= second < 60, 'invalid value specified for second'
    self.__second = second

#---------------------------------------------------------------------------
# Output methods

def print_time(self):
    fmt = '{:02d}:{:02d}:{:02d}'
    print(fmt.format(self.hour, self.minute, self.second ))

#---------------------------------------------------------------------------
# Conversion methods

def time_to_int(self):
    minutes = self.hour * 60 + self.minute
    seconds = minutes * 60 + self.second
    return seconds

#---------------------------------------------------------------------------
# Misc

def increment(self, seconds):
    seconds += self.time_to_int()
    return int_to_time(seconds)

def is_after(self, other):
    return self.time_to_int() > other.time_to_int()

def add_time(self, other):
    seconds = self.time_to_int() + \
              other.time_to_int()
    return int_to_time(seconds)
```

```
In [11]: def func1(s):
             if len(s) == 0:
                 return s
             else:
                 return s[0] * len(s) + func2(s)
         def func2(s):
             return func1(s[1:])
         print(func1('Israel'))
```

```
IIIIIIsssssrrrraaaeel
```

```
In [12]: import random
         def has_duplicates(list):
             list1=sorted(list)
             i=0
             while i < len(list1)-1:
                 if list1[i]==list1[i+1]:
                     return True
                 else:
                     i=i+1
             return False
         def rand_birthday(n, students):
             number_same_birthday=0
             for i in range(n):
                 birthday = []

                 for i in range(students):
                     birthday.append(random.randint(1, 365))
                 if has_duplicates(birthday):
                     number_same_birthday = number_same_birthday + 1

             return number_same_birthday / n * 100

         print(rand_birthday(1000, 23))
```

```
50.9
```

```
In [13]: from collections import defaultdict

         with open('E-coli.txt', 'r') as f:
             genome = f.read().rstrip()


         kmer = 9
         Length = 500
         min_clumpsize = 3
```

```
def get_substrings(g, k):
    """
Take the input genome window 'g', and produce a list of unique
substrings of length 'k' contained within it.
    """
    substrings = list()

    for i in range(k):
        line = g[i:]
        substrings += [line[i:i + k]
                       for i in range(0, len(line), k) if i + k <= len(line)]


    results = defaultdict(int)
    for s in substrings:
        results[s] += 1
    return results


def find_clumps(genome, kmer, Length, clumpsize):

    window = genome[0:Length]

    # Initialise our counter, because the main algorithm can't start from
    # scratch.
    patterns = get_substrings(window, kmer)

    # Using a dictionary not a list because the lookups are faster once the
    # size of the object becomes large
    relevant = {p: 1 for p in patterns if patterns[p] >= clumpsize}

    starting_string = genome[0:kmer]

    for i in range(Length, len(genome)):

        window = window[1:]
        window += genome[i]

        patterns[starting_string] -= 1
        starting_string = window[0:kmer]

        ending_string = window[-kmer:]
        patterns[ending_string] += 1

        if patterns[ending_string] >= clumpsize and ending_string not in relevant:
            relevant[ending_string] = 1
```

```
             return list(relevant)


         if __name__ == "__main__":
             clumps = find_clumps(genome, kmer, Length, min_clumpsize)
             print("Total: {}".format(len(clumps)))
```

```
Total: 1918
```

```
In [17]: import os

         def walk(dir):
             """Prints the names of all files in dirname and its subdirectories.

             dirname: string name of directory
             """
             for name in os.listdir(dir):
                 path = os.path.join(dir, name)

                 if os.path.isfile(path):
                     print(path)
                 else:
                      walk(path)
                 print(walk(os.getcwd()))
         def walk2(dirname):
             """Prints the names of all files in dirname and its subdirectories.

             dirname: string name of directory
             """
             for root, dirs, files in os.walk(dir):
                 for filename in files:
                     print(os.path.join(root, filename))


         if __name__ == '__main__':
             walk('.')
             walk2('.')
```

```
In [19]: data=open('words.txt', 'r')
         words=data.read().splitlines()
         def anagram_dict(words):
             anagrams=dict()
             for word in words:
                 sorted_word=''.join(sorted(word))
                 anagrams[sorted_word]=anagrams.get(sorted_word, [])
                 anagrams[sorted_word].append(word)
```

```
    anagrams = {sorted_word: anagrams[sorted_word] for sorted_word in anagrams if len
    return anagrams

print(anagram_dict(words))
```