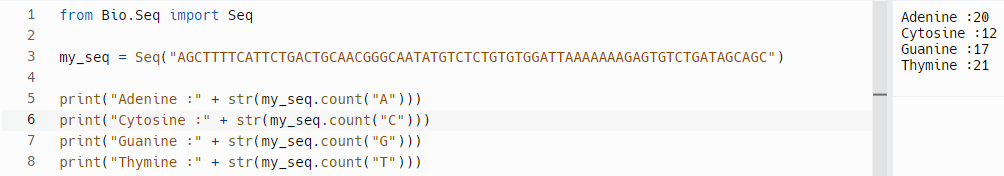
1η Εργασία βιοπληροφορικής

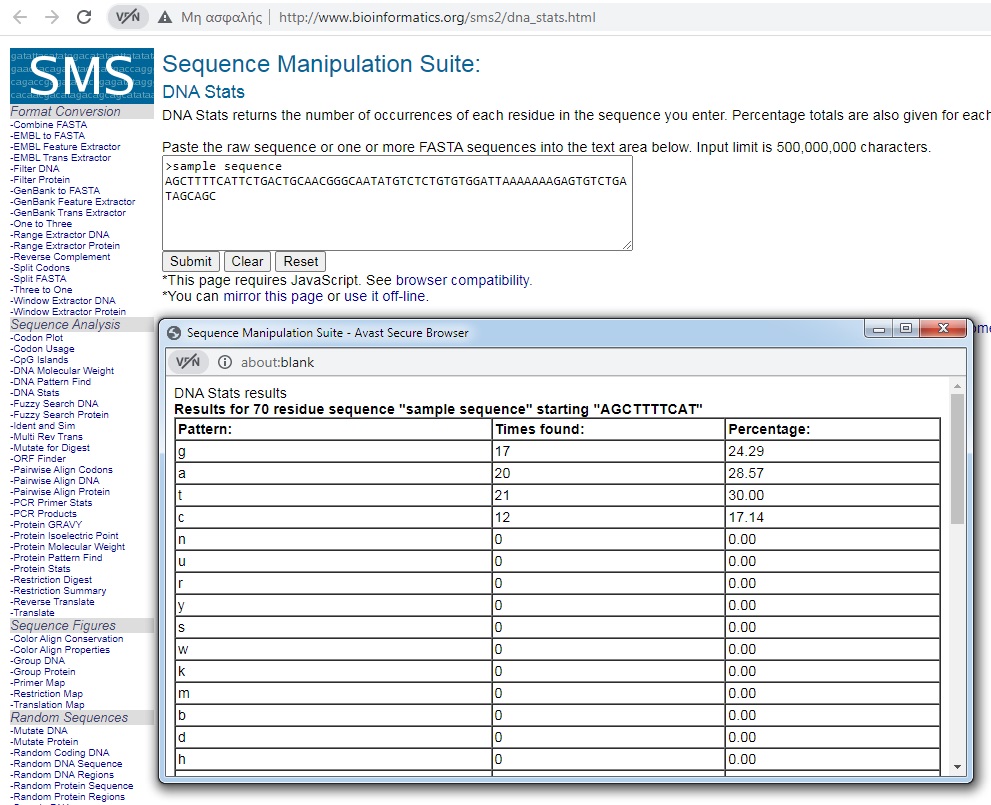
Άσκηση 1

Το πρώτο πρόβλημα είναι δεδομένης μίας αλληλουχίας DNA να μετρήσουμε τις εμφανίσεις του κάθε αμινοξέος.

Ας δούμε πρώτα με χρήση biopython

To Seq είναι μια δομή που βολεύει όταν διαχειριζόμαστε αλληλουχίες διότι μας επιτρέπει να χρησιμοποιήσουμε έτοιμες συναρτήσεις. Η str() είναι απαραίτητη διότι έχουμε δύο διαφορετικού τύπου αντικείμενα.

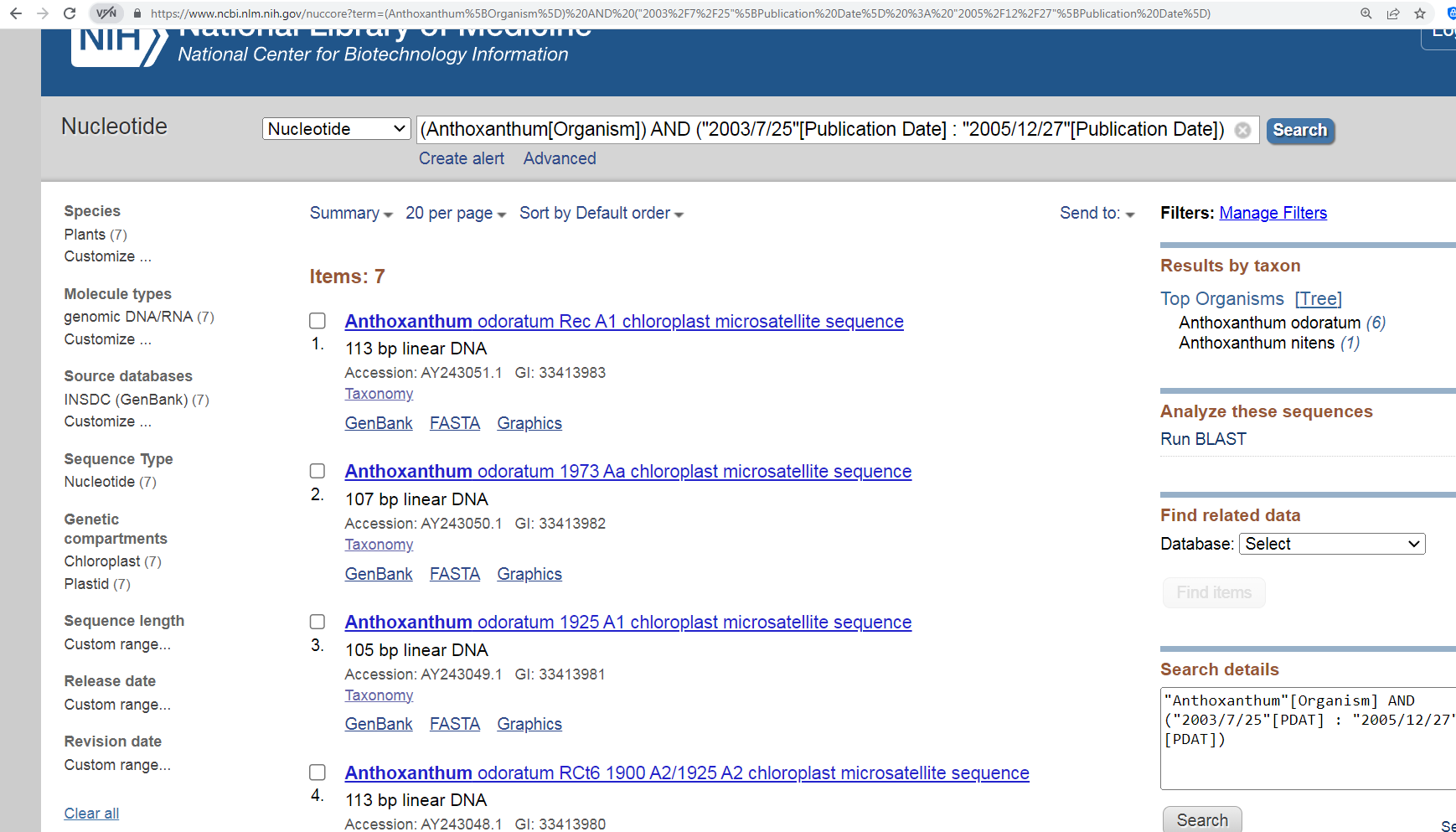
Θα δοκιμάσουμε και το Sequence Manipulation Suite ένα γνωστό εργαλείο για ανάλυση αλληλουχιών. Θα χρησιμοποιήσουμε το DNA stats που είναι online.



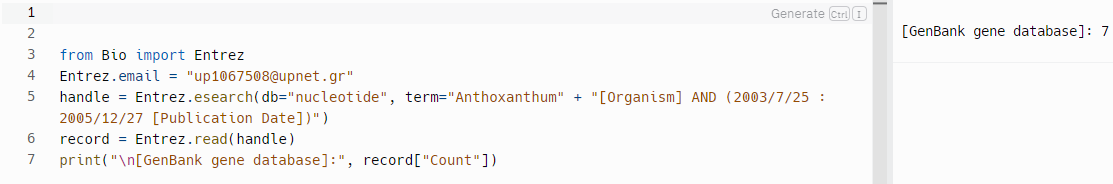
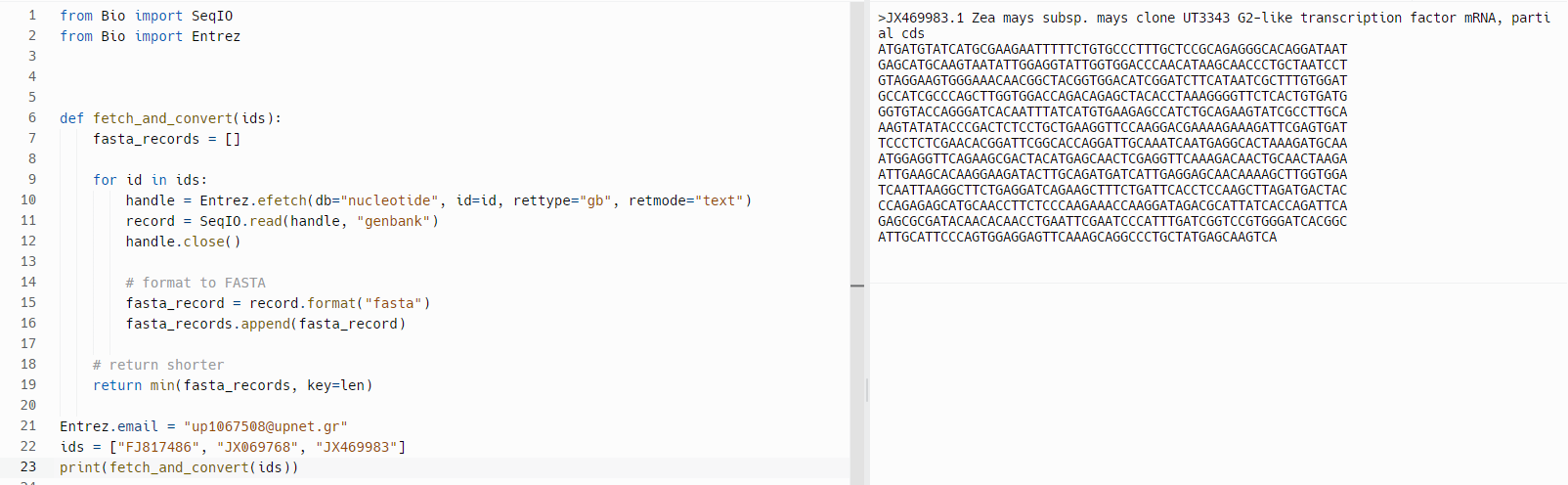
Όπως βλέπουμε αυτό το εργαλείο μας δείχνει και τα ποσοστά ύπαρξης του κάθε αμινοξέος.

Έπειτα θα αναζητήσουμε όλες τις καταχωρίσεις ενός γένους που έγιναν μεταξύ 2 ημερομηνιών, στην GenBank του NCBI.

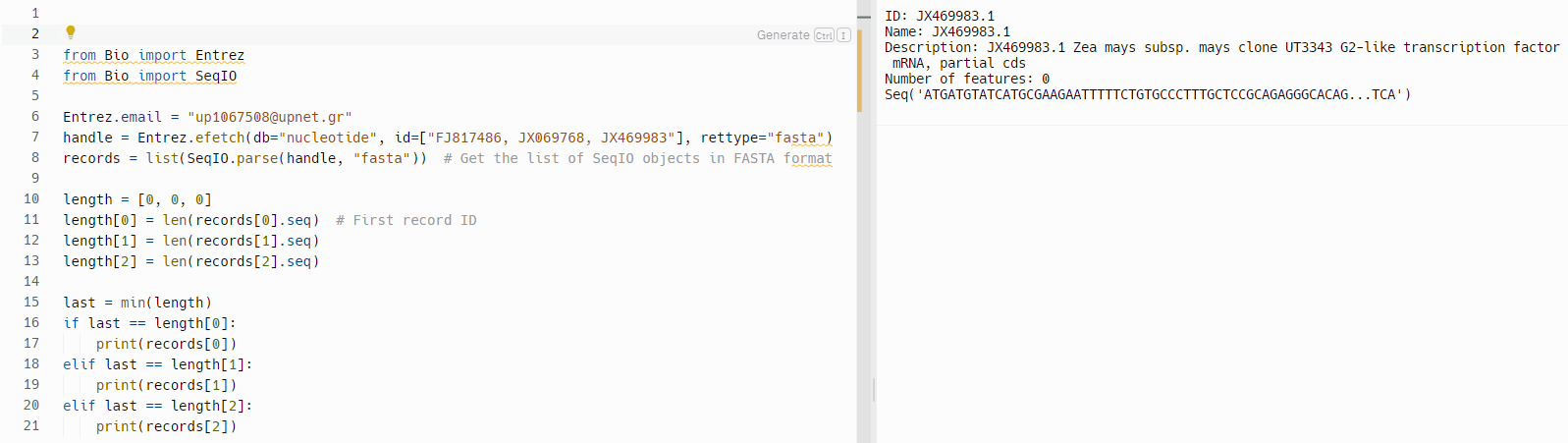
Ψάχνουμε στην online σελίδα.

Όπως βλέπουμε βάλαμε φίλτρο για αναζήτηση στην GenBank, με το “nucleotide”.

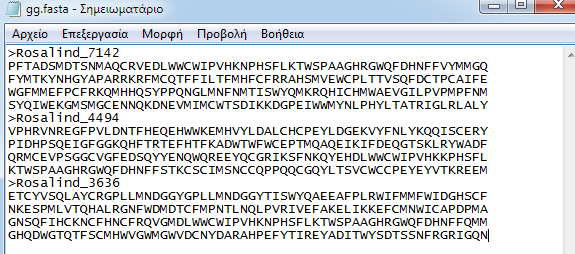
Η συνάρτηση **Bio.Entrez.esearch()** της biopython μπορεί να ψάξει όλες τις βάσεις δεδομένων του NCBI.

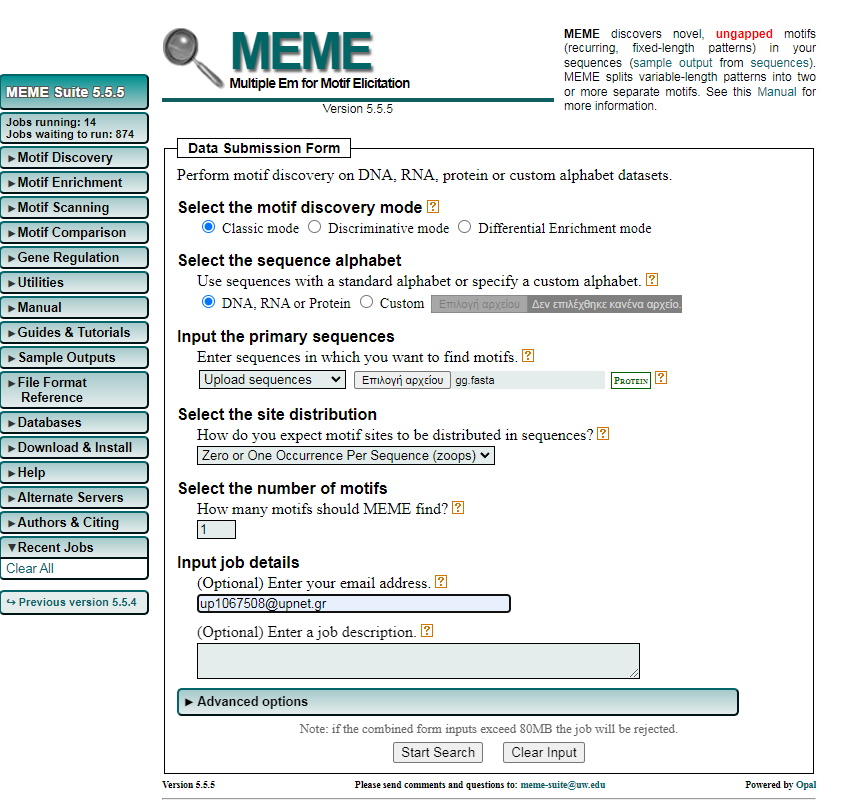
Το επόμενο πρόβλημα είναι να δώσουμε στην genBank 3 id και να μας επιστραφεί η μικρότερη αλληλουχία σε FASTA format. Θα ξεκινήσουμε με την biopython. Στην αρχή δοκιμάσαμε τον παρακάτω κώδικα ο οποίος δεν χρησιμοποιεί όλα τα εργαλεία που προτείνονται στο Rosalind. Σύντομα παρατηρήσαμε ότι παρόλο που δίνει σωστό αποτέλεσμα αυτός ο κώδικας μετράει και το μέγεθος της περιγραφής μαζί με αυτό της αλληλουχίας.

Άρα αλλάζουμε τον κώδικα σε:

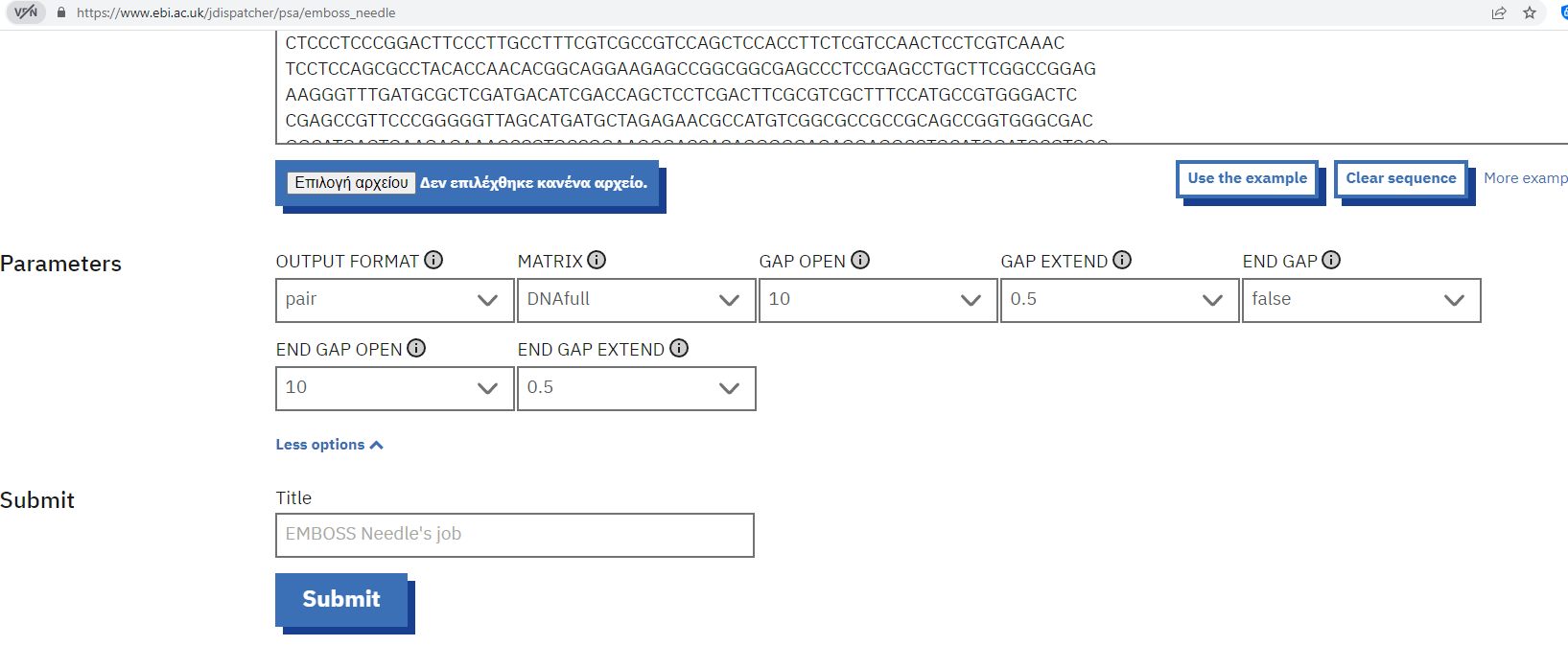
Και πλέον ο κώδικας είναι σωστός διότι μετράει μόνο το μήκος της αλληλουχίας.

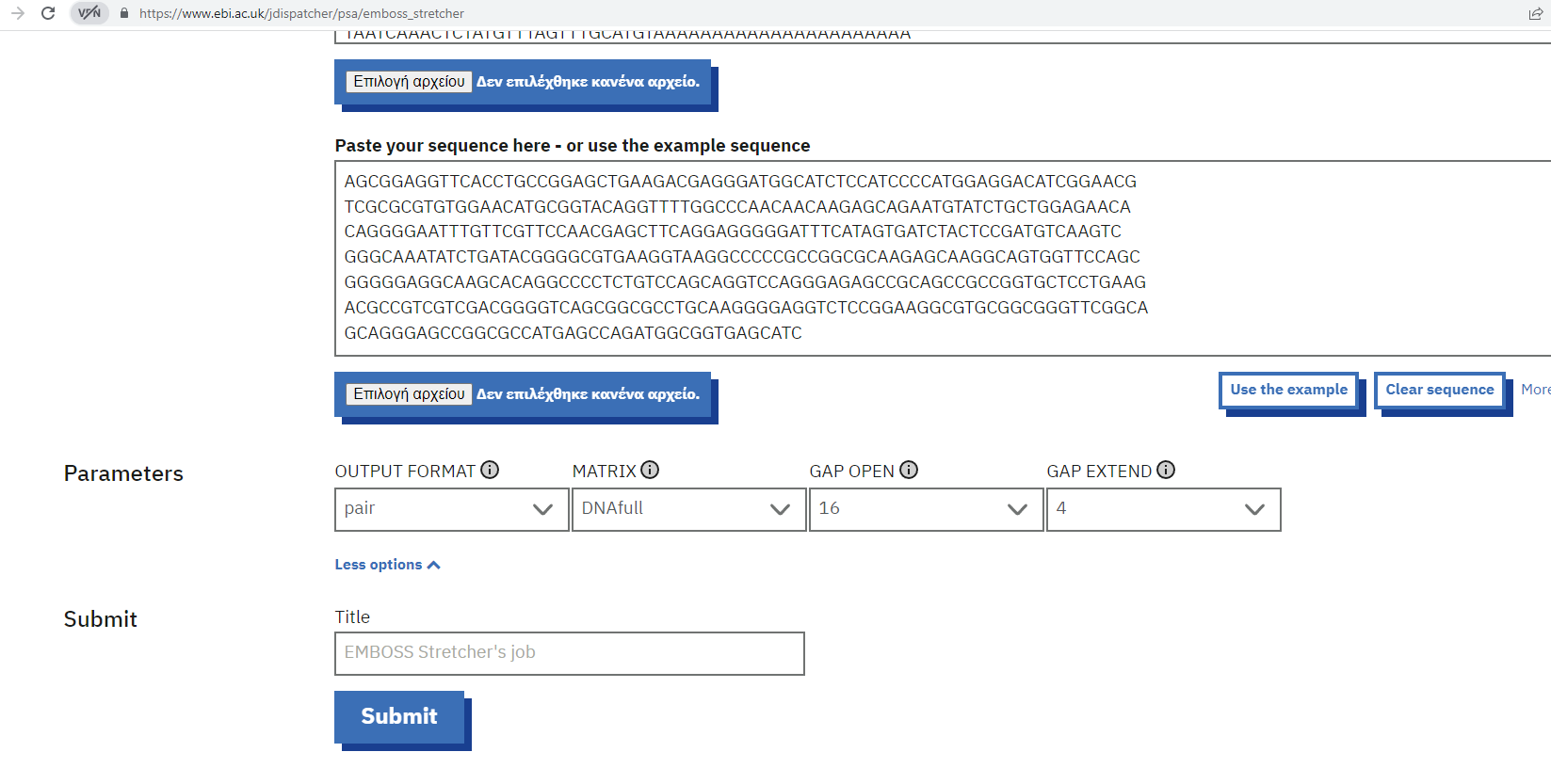
Πρώτα φτιάχνουμε ένα fasta αρχείο με τις ακολουθίες μας.



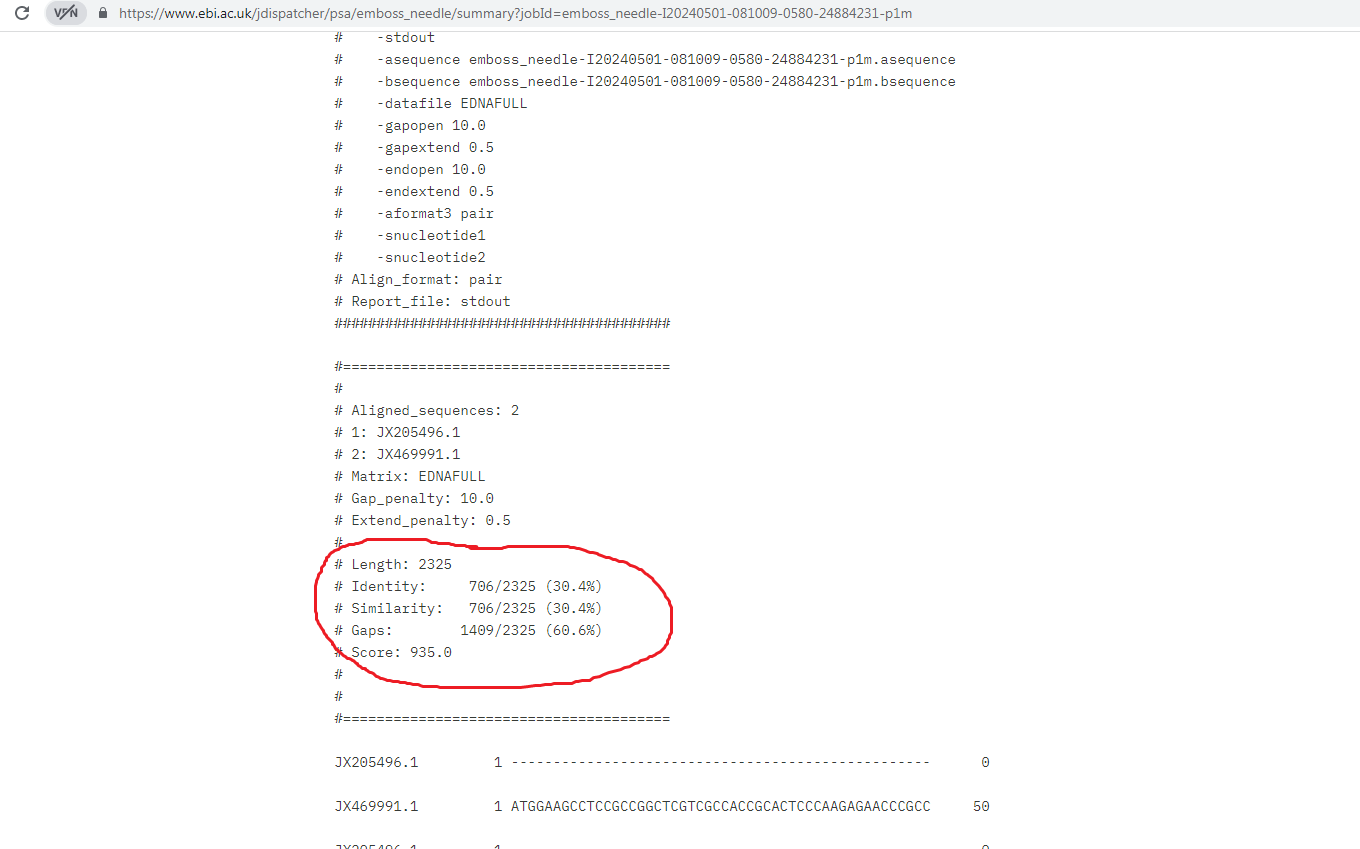
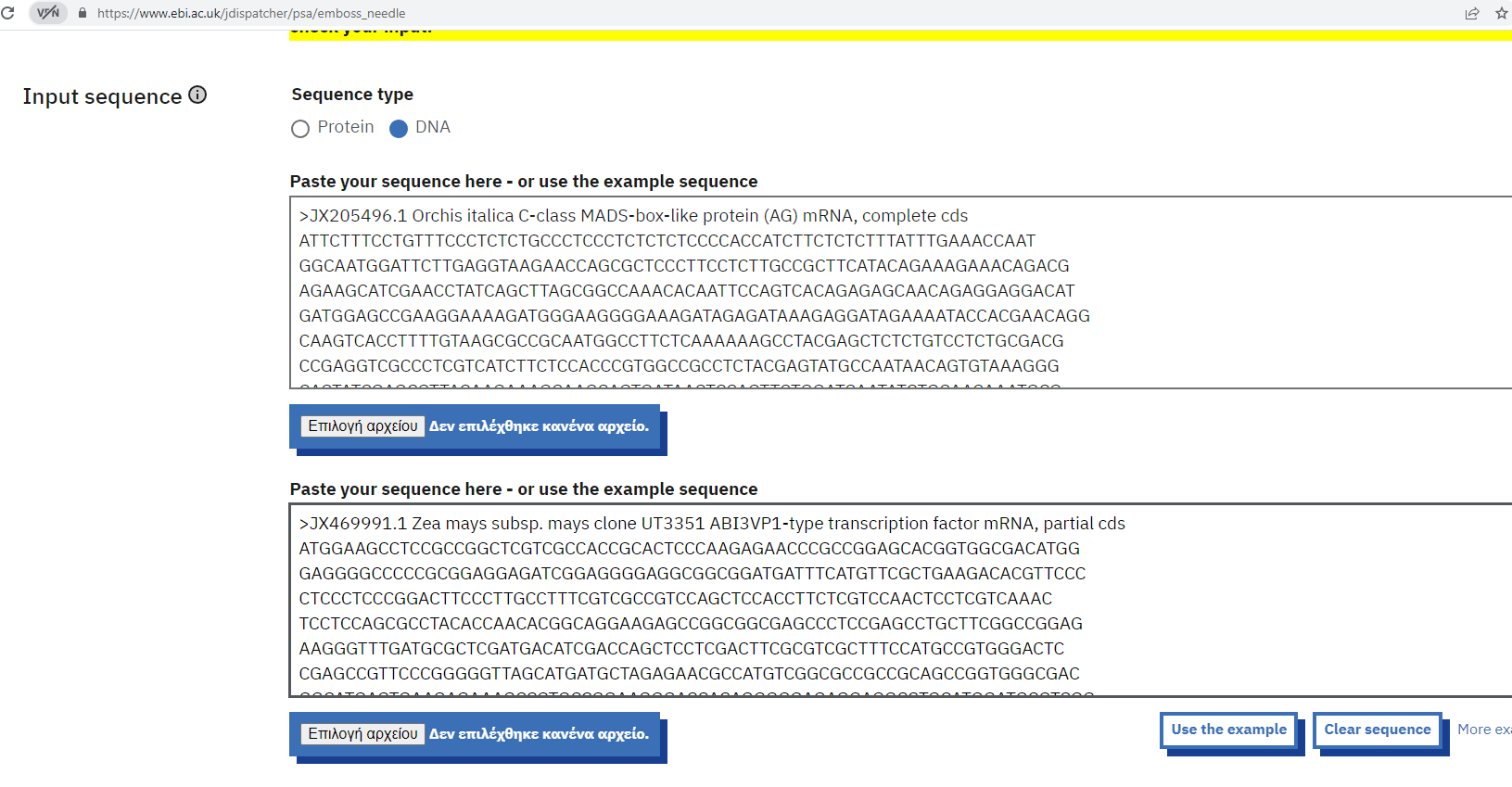
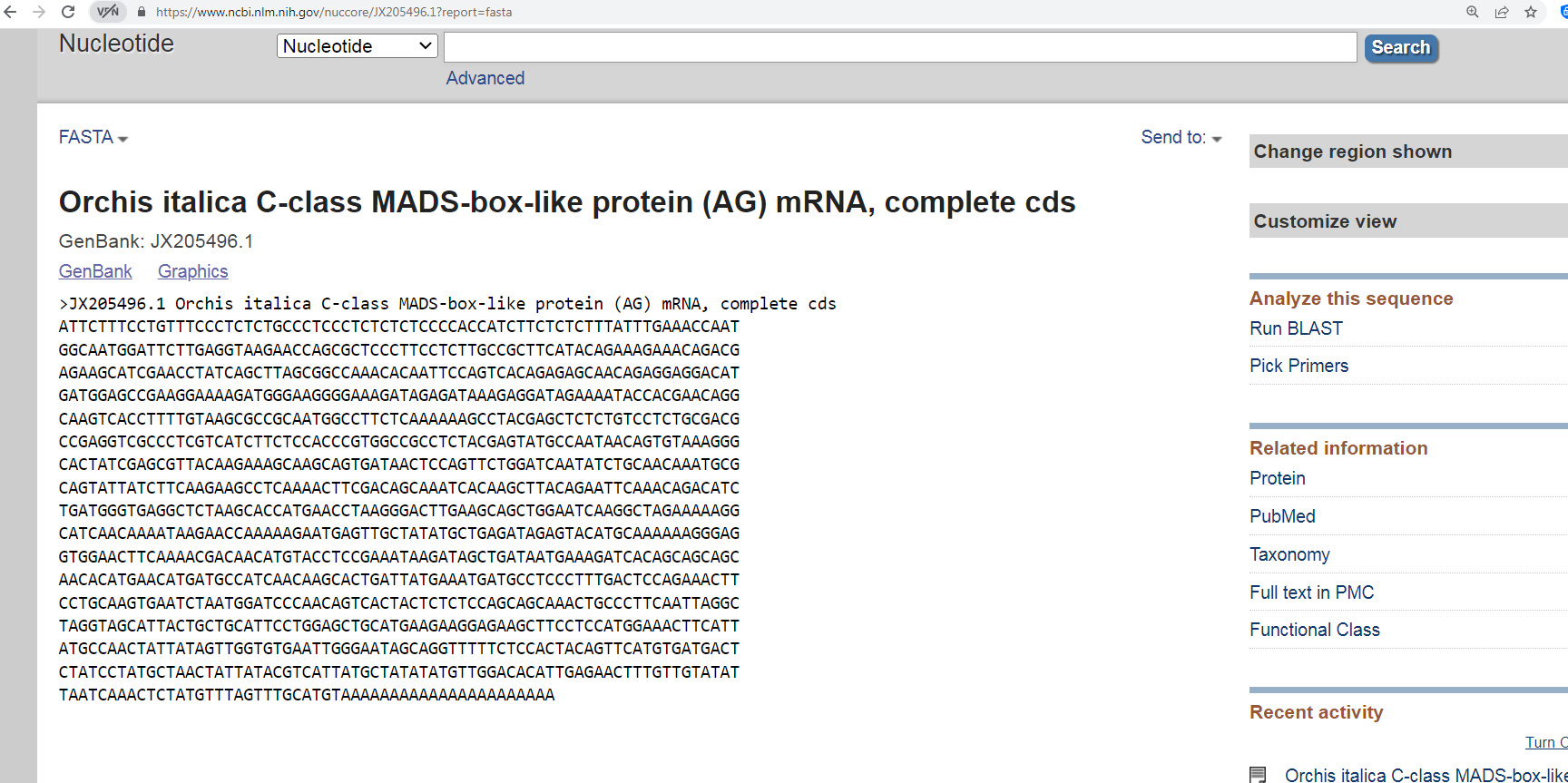
Στην επόμενη άσκηση ψάχνουμε να βρούμε αν 2 αλληλουχίες έχουν κοινούς προγόνους. Έτσι θέλουμε να κάνουμε ευθυγράμμιση και να δούμε τι σκορ λαμβάνουν.

Θα χρησιμοποιήσουμε τα εργαλεία Needle και Stretcher. Μια διαφορά τους είναι ότι το Stretcher χρησιμοποιεί διαφορετική βαθμολόγηση όταν τελειώνουν τα κενά ενώ το Needle δεν το έχει ως προεπιλογή. Επίσης στις επιλογές που μας δίνουν τα δύο εργαλεία έχουν διαφορετικά νούμερα. Πχ το Needle έχει στην αναζήτησή μας GAP EXTEND=0.5 ενώ το Stretcher έχει επιλογές 1,2,3 κτλ.

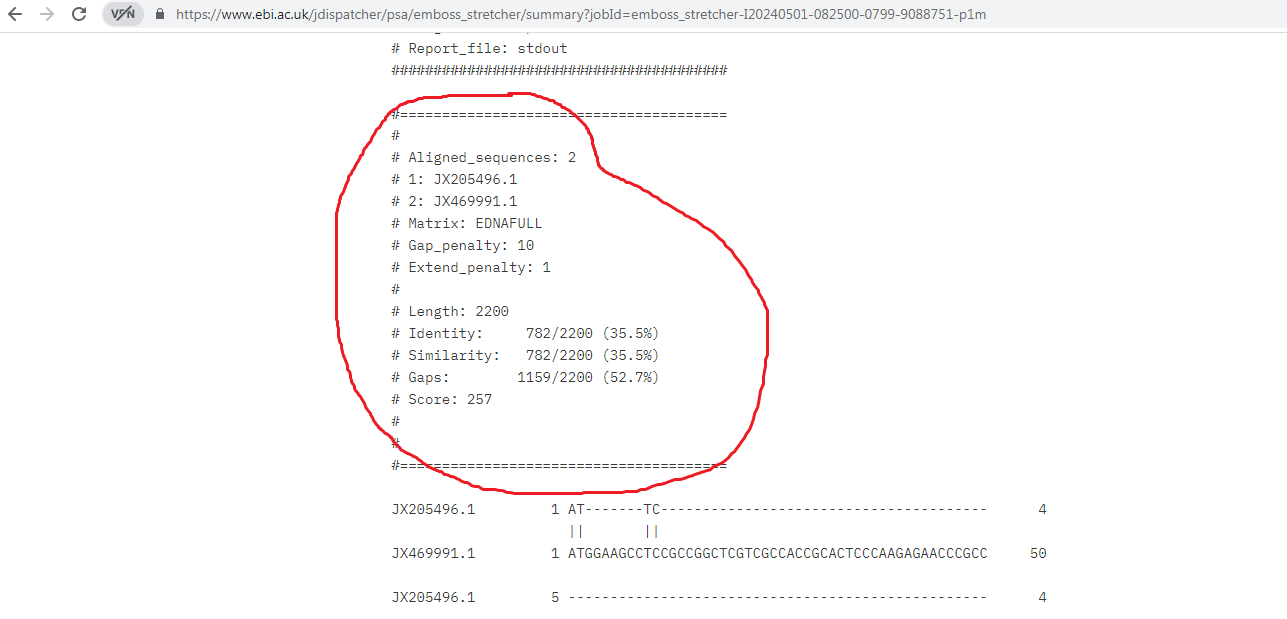




Ξεκινάμε με το Needle, βρίσκουμε το Fasta Format των ID που θέλουμε να εισάγουμε.



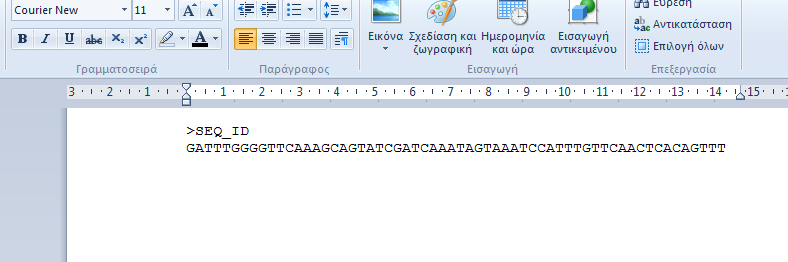
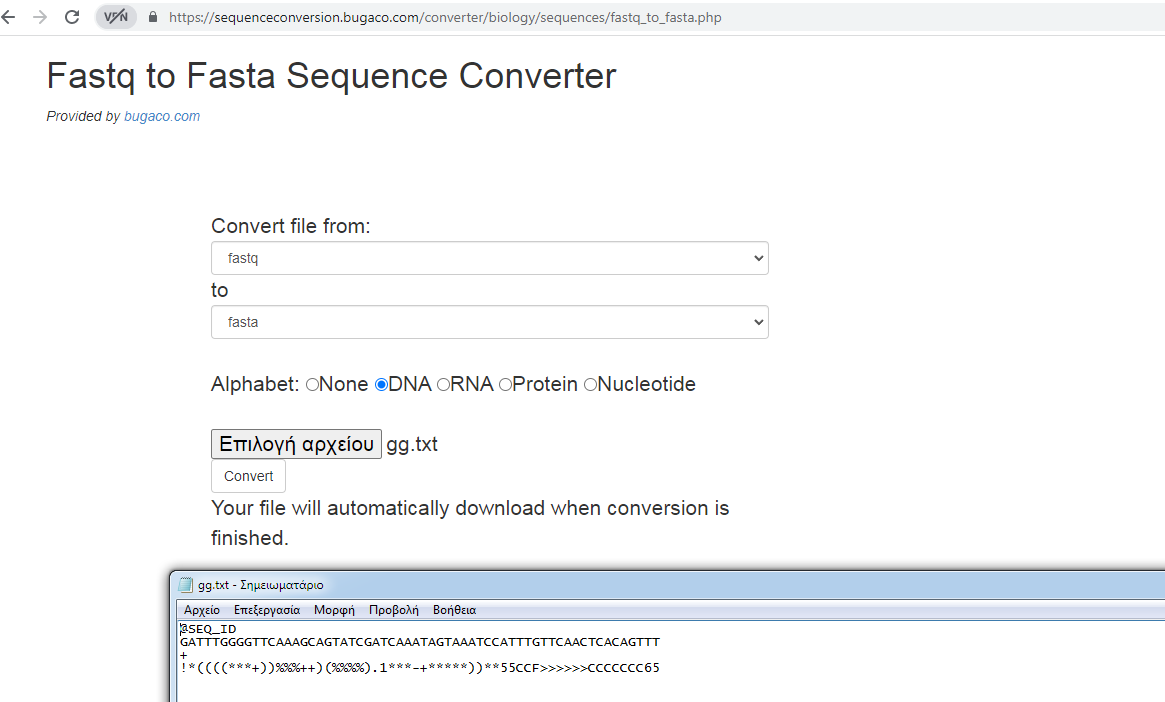
Έπειτα πάμε στο Stretcher.



Εδώ βρήκαμε 35.5% ομοιότητα. Άρα βρήκαμε μεγαλύτερη επιτυχία παρόλο που έχοντας λιγότερες παραμέτρους.

Το επόμενο ζητούμενο είναι να μετατρέψουμε ένα fastq αρχείο σε fasta μορφή.

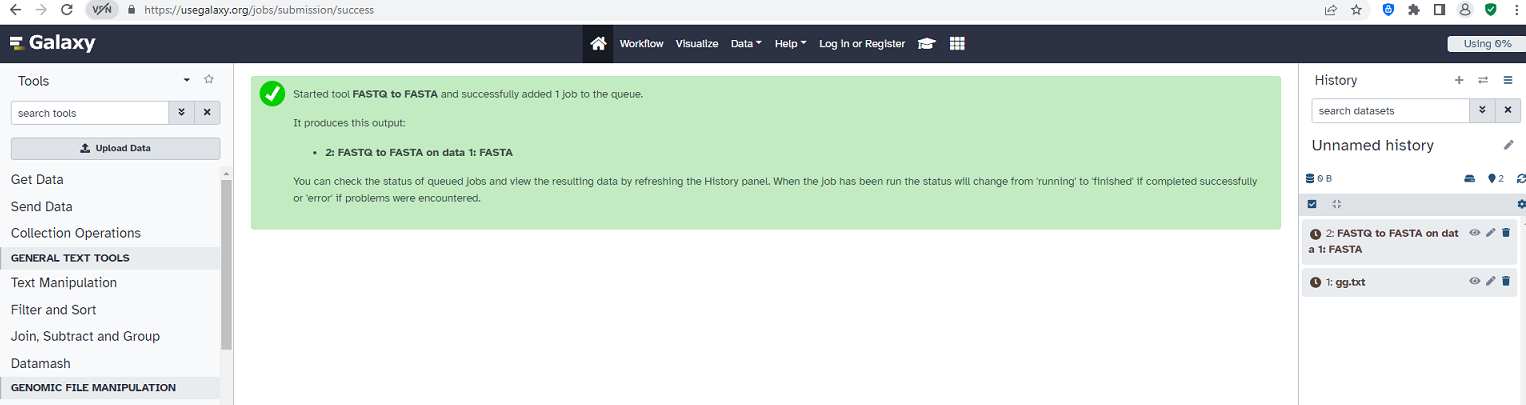
Χρησιμοποιούμε πρώτα το εργαλείο Sequence conversion website και βλέπουμε ότι μας δίνει το επιθυμητό output.

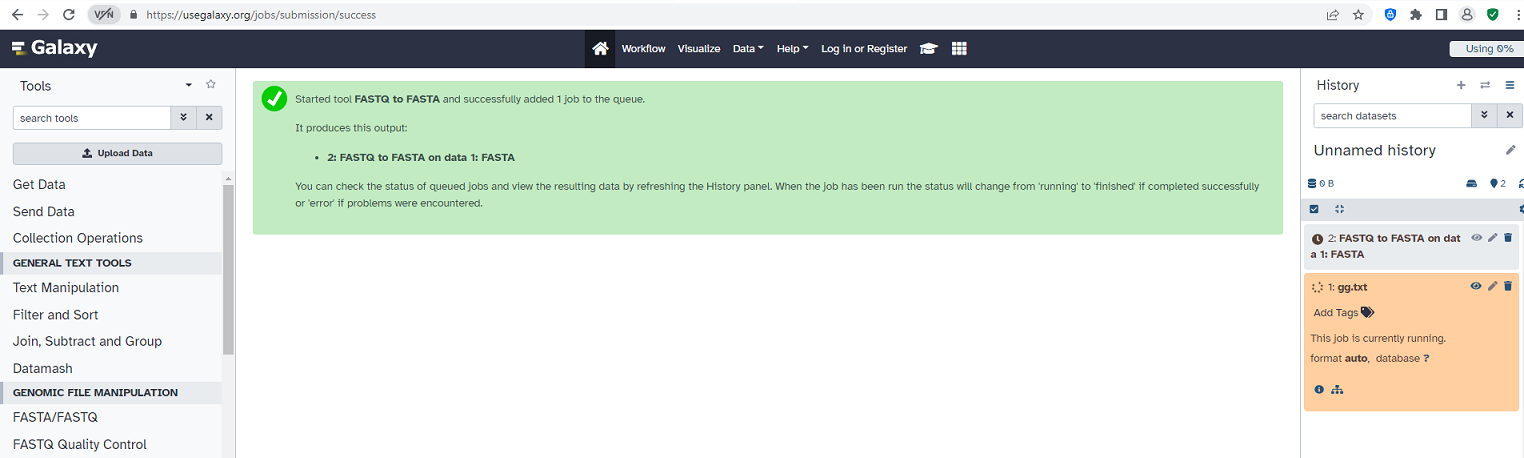


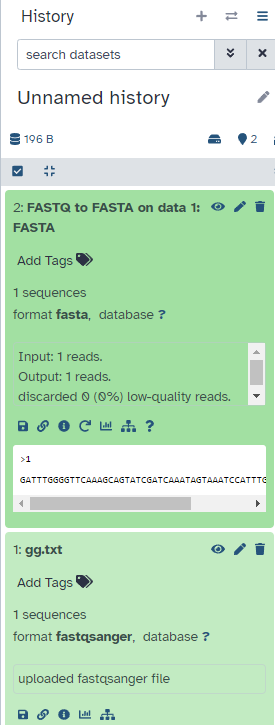
Αποφεύγουμε να χρησιμοποιήσουμε το BlastStation διότι απαιτεί λήψη και αγορά.

Θα δοκιμάσουμε όμως με galaxy.

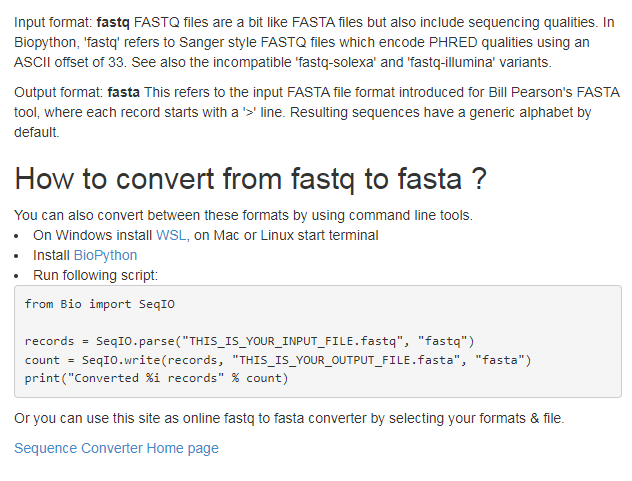
Βλέπουμε ότι δίνει το ίδιο ακριβώς αποτέλεσμα.

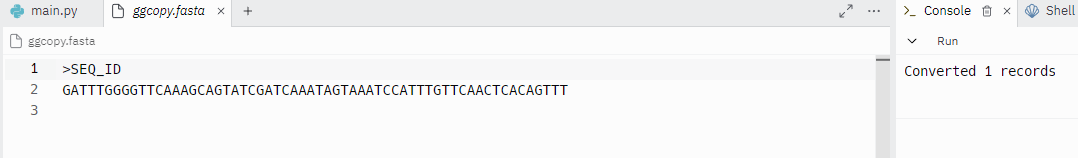
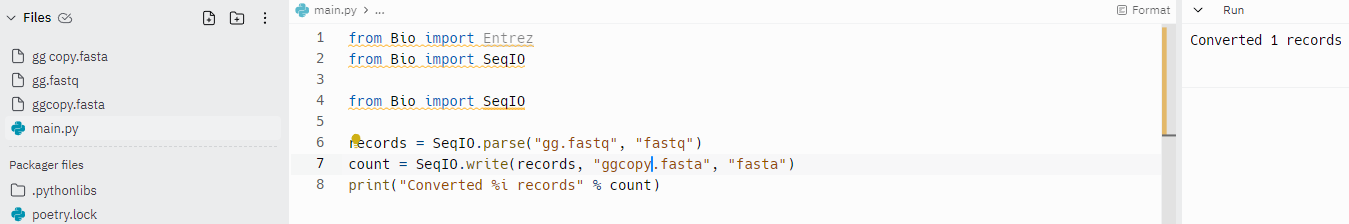




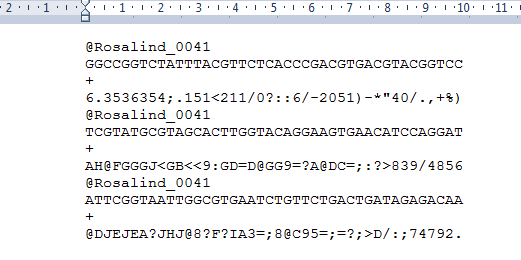


Επίσης θα δοκιμάσουμε και σε biopython.

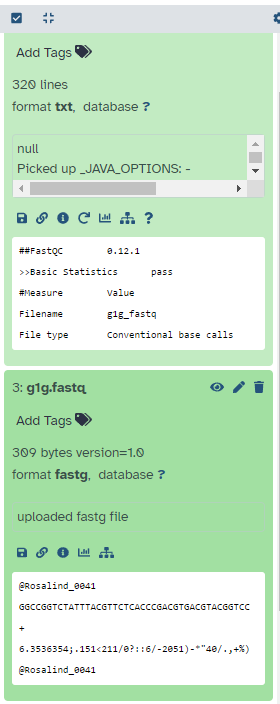




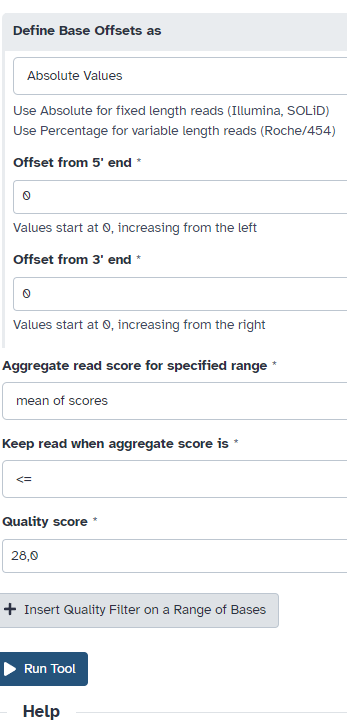
Θα χρησιμοποιήσουμε το εργαλείο fastqc που βρίσκεται online στο site galaxy. Στόχος η ανάλυση ποιότητας ενός fastq αρχείου.



Δυστυχώς με αυτό το εργαλείο δεν καταφέραμε να δούμε πολλά.



Οπότε θα ψάξουμε και στο Filter FASTQ.



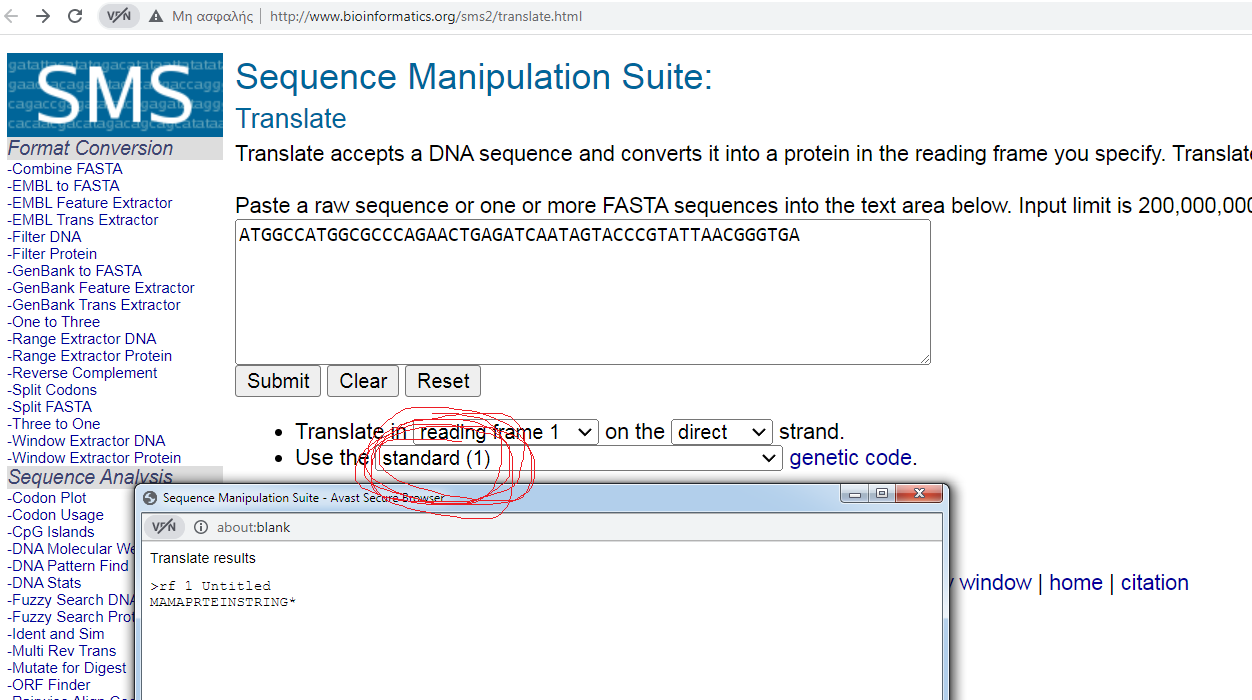


Εδώ βλέπουμε το επιθυμητό αποτέλεσμα.

Τέλος δοκιμάζουμε με biopython.



Τώρα θα μεταφερθούμε στο επόμενο πρόβλημα. Μέσω της σελίδας <http://www.bioinformatics.org/sms2/translate.html> , του εργαλείου μετάφρασης του SMS 2 . Βλέπουμε ότι για την ακολουθία dna που δώσαμε, παράγεται η ζητούμενη πρωτείνη με το γεννετικό κώδικα 1. Αυτός ο τρόπος είναι αργός καθώς χρειάζεται να κάνουμε πολλά ερωτήματα μέχρι να πετύχουμε το σωστό κώδικα.

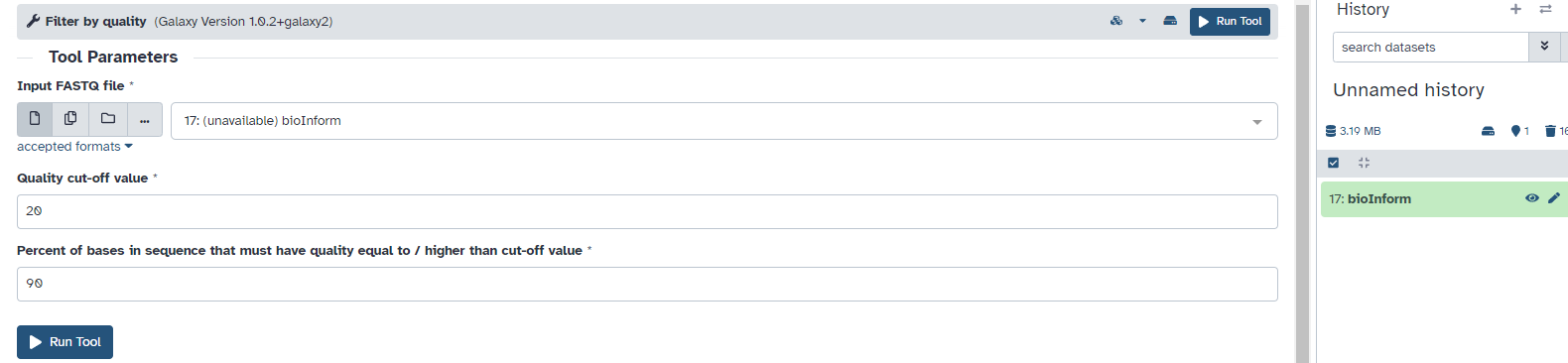


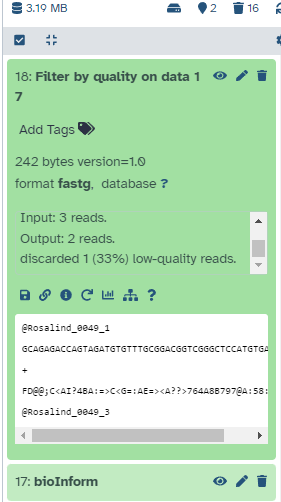
Θα δοκιμάσουμε να λύσουμε το ίδιο πρόβλημα στην biopython.



Τώρα πάμε στο επόμενο πρόβλημα, την εύρεση ποιότητας μέσω του fastg quality filter.

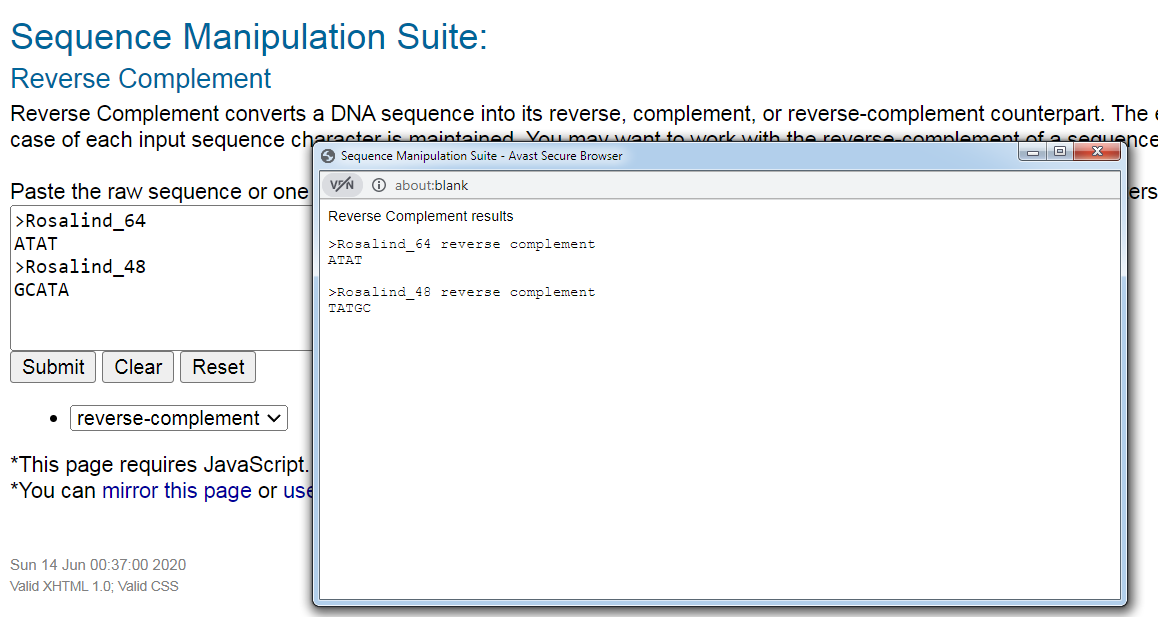
Η απάντηση είναι η αναμενόμενη, 2.





Στο επόμενο πρόβλημα κοιτάμε αν οι 2 ακολουθίες μας είναι ίδιες με τις αντίστροφες συμπληρωματικές τους. Όπως βλέπουμε μόνο η 1 είναι.

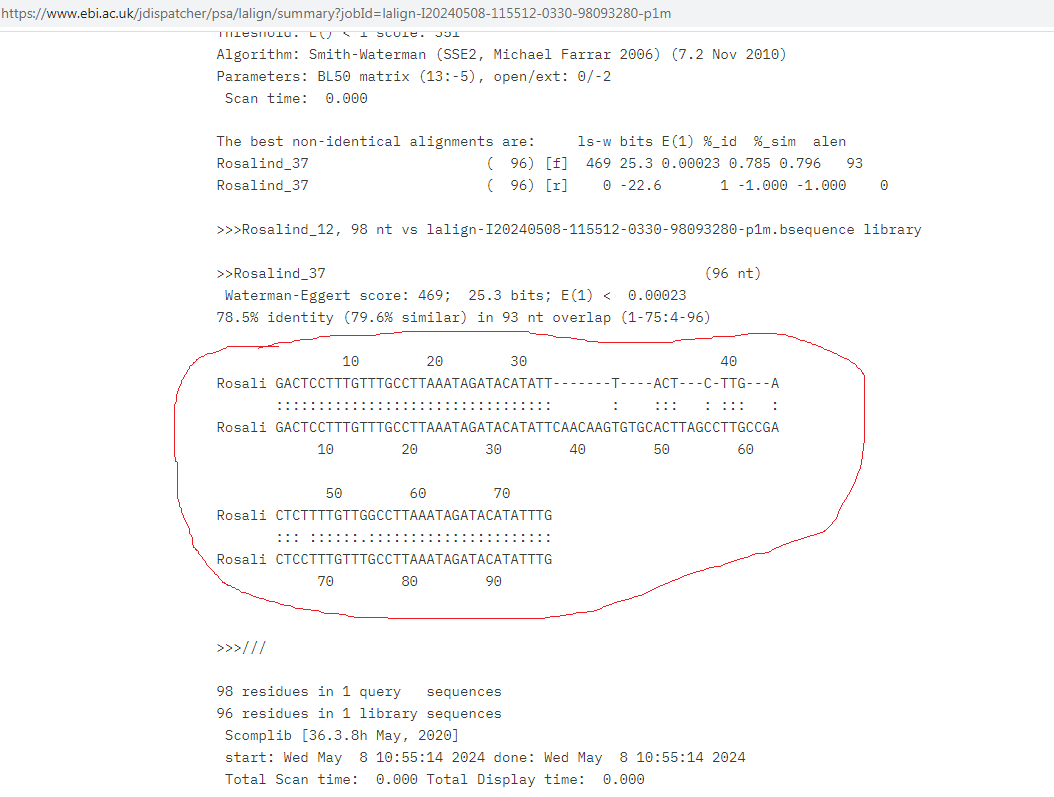
Χρησιμοποιήσαμε το Reverse Complement εργαλείο της sms2. Θα χρησιμοποιήσουμε και biopython.



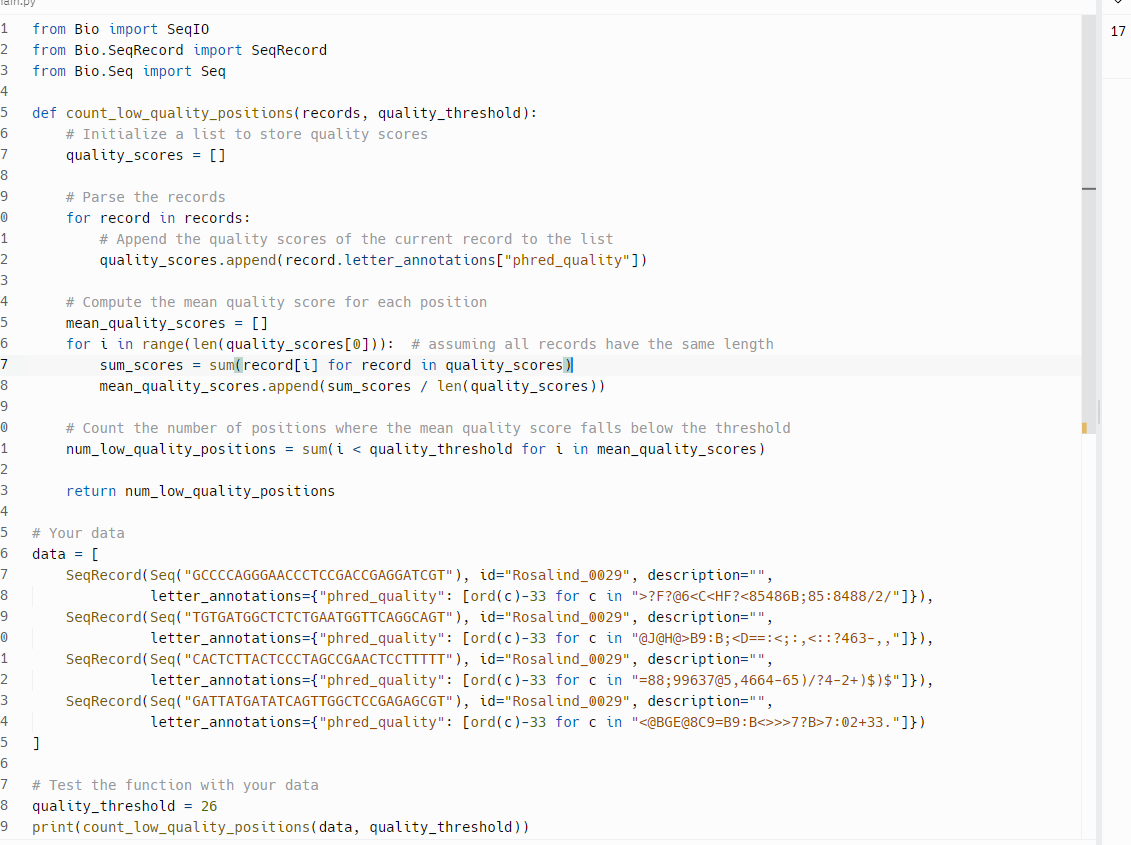


Στο επόμενο πρόβλημα θα χρησιμοποιήσουμε το Lalign εργαλείο του Ebi. Στο εργαλείο αυτό είναι δύσκολο να εντοπίσουμε αλληλουχίες που διαφέρουν μόνο 3 ζευγάρια. Στις παραμέτρους βάζουμε gap opening=0 ώστε σε περίπτωση που υπάρχει κάποια διαγραφή να μην την μετρήσει πιο δύσκολα καθώς έχουμε δικαίωμα 3 αλλαγών. Στο gap extend βάζουμε -2, ώστε να μην επεκταθεί πολύ το κενό. Στην τιμή E βάζουμε χαμηλή τιμή διότι θέλουμε μια πιο ακριβή αντιστοίχηση (δικαιούμαστε μόνο 3 λάθη).

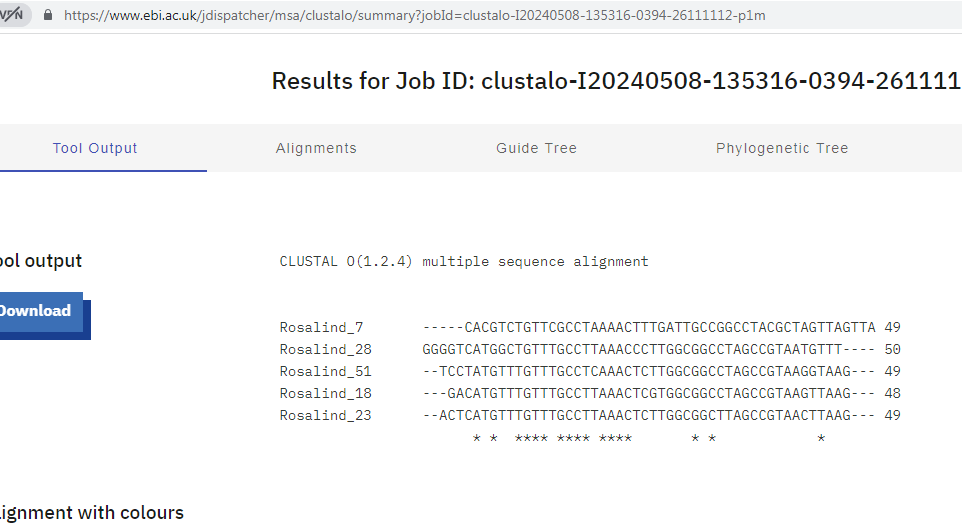




Στο επόμενο πρόβλημα μας δίνεται ένα αρχείο fastq και ένα όριο 26. Πρέπει να βρούμε τον αριθμό των θέσεων των αλληλουχιών όπου η ποιότητα βάσεων πέφτει κάτω από 26.

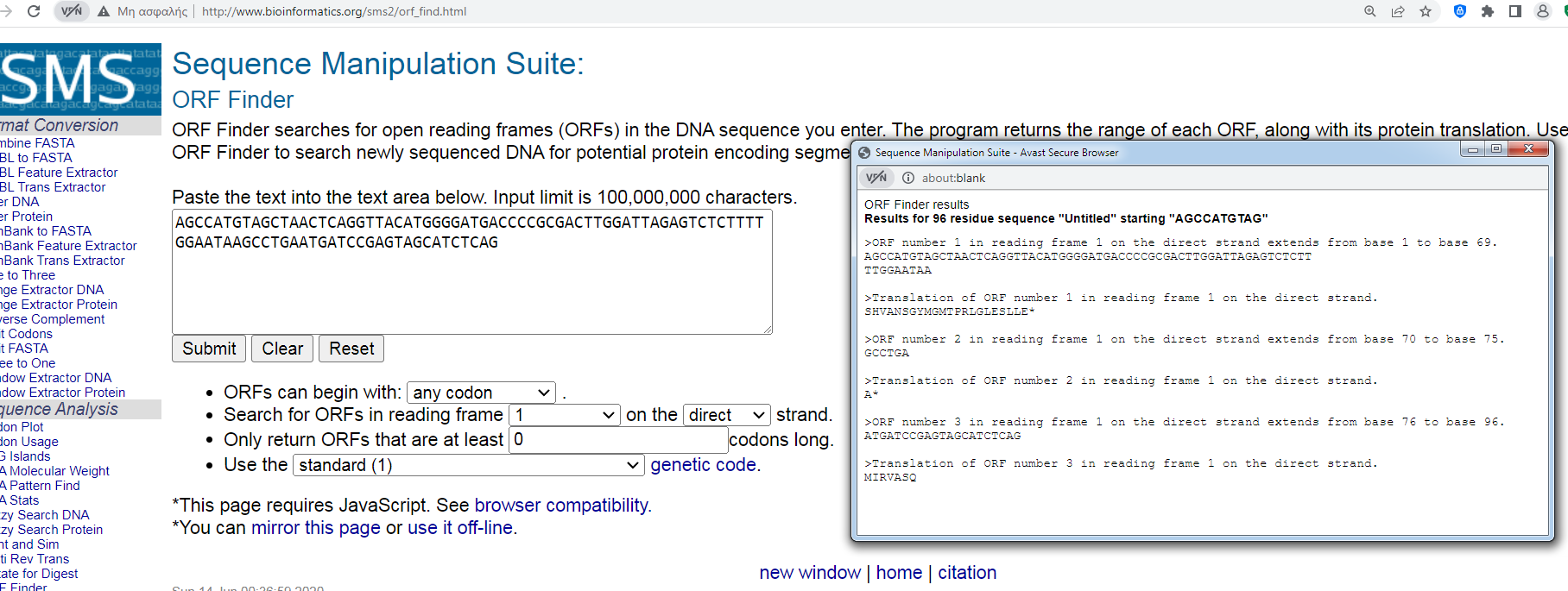


Τώρα εισάγουμε στο εργαλείο clustal τις αλληλουχίες dna και βλέπουμε οτι αυτή με τις πιο πολλές διαφορές (τα πιο πολλά κενά) είναι η πρώτη.

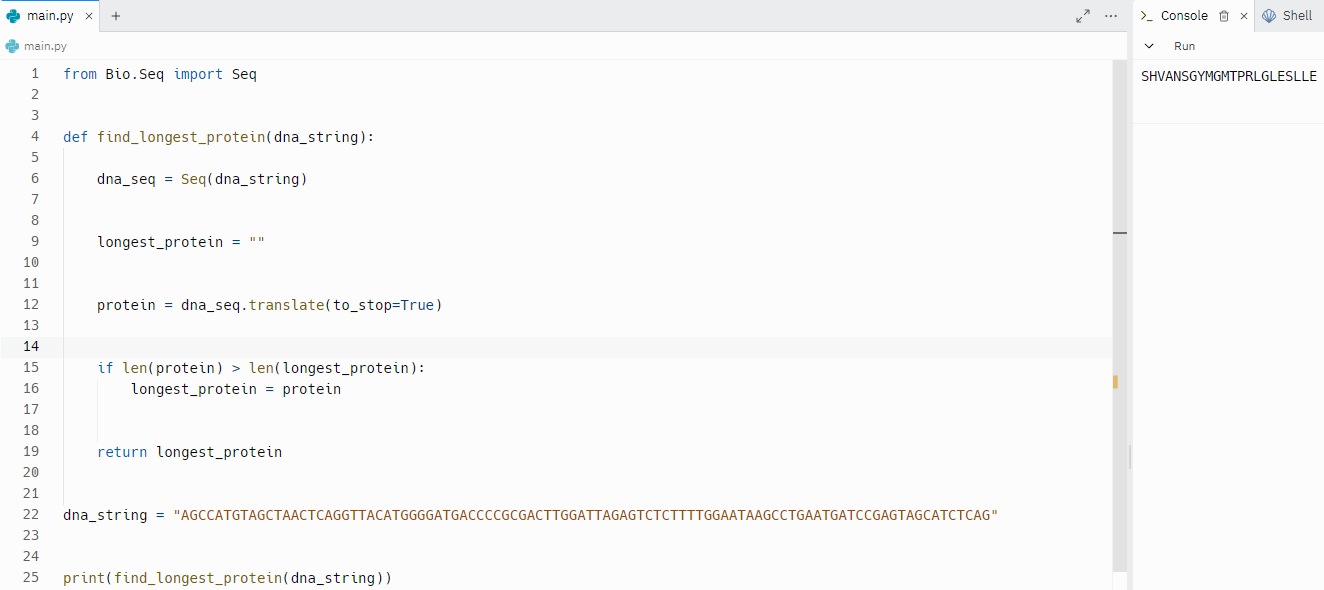


Έπειτα στο επόμενο πρόβλημα προσπαθούμε να βρούμε την μεγαλύτερη πρωτείνη που μπορεί να μεταφραστεί από ένα ORF από αυτά που υπάρχουν στην αλληλουχία μας.

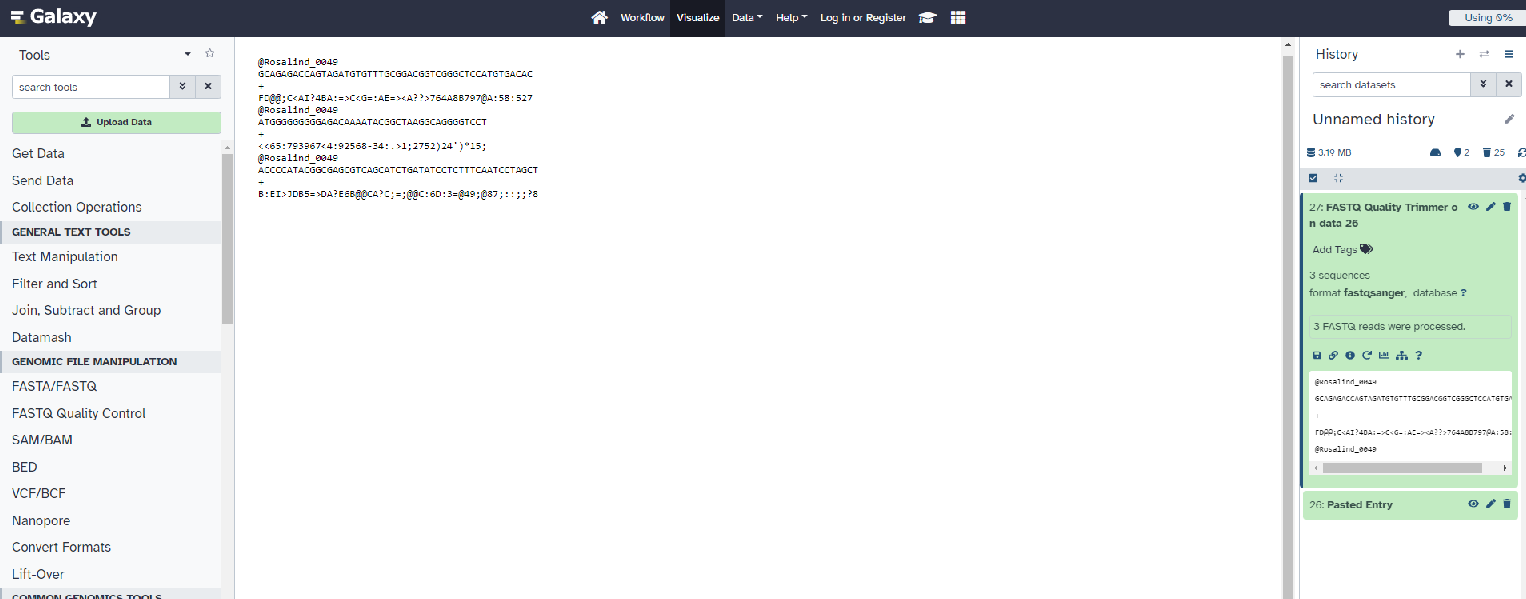
Το SMS2 δεν δίνει ίδια αποτελέσματα με το rosalind.



Δοκιμάσαμε να λύσουμε το πρόβλημα με biopython. Όμως πάλι βρήκαμε διαφορετική λύση.

Η λύσεις των biopython και sms2 είναι ίδιες.

Έπειτα χρησιμοποιούμε το εργαλείο trimmer quality του galaxy για να εντοπίσουμε τις βάσεις που έχουν κακή ποιότητα και να ελαφρύνουμε την αποθήκευσή τους χρησιμοποιώντας λιγότερο χώρο. Το πρόγραμμα μας δυσκόλεψε λίγο σε αυτό το αρχείο. Όλα διορθώθηκαν όταν αντί να επιλέγουμε μορφή αρχείου πατήσαμε autodetection.



ΚΩΔΙΚΑΣ

from Bio.Seq import Seq

my\_seq = Seq("AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC")

print("Adenine :" + str(my\_seq.count("A")))

print("Cytosine :" + str(my\_seq.count("C")))

print("Guanine :" + str(my\_seq.count("G")))

print("Thymine :" + str(my\_seq.count("T")))

---------------------------------------------------------------------------------------------------------------------------------

from Bio import Entrez

Entrez.email = "up1067508@upnet.gr"

handle = Entrez.esearch(db="nucleotide", term="Anthoxanthum" + "[Organism] AND (2003/7/25 : 2005/12/27 [Publication Date])")

record = Entrez.read(handle)

print("\n[GenBank gene database]:", record["Count"])

----------------------------------------------------------------------------------------------------------------------------------

from Bio import SeqIO

from Bio import Entrez

def fetch\_and\_convert(ids):

fasta\_records = []

for id in ids:

handle = Entrez.efetch(db="nucleotide", id=id, rettype="gb", retmode="text")

record = SeqIO.read(handle, "genbank")

handle.close()

# format to FASTA

fasta\_record = record.format("fasta")

fasta\_records.append(fasta\_record)

# return shorter

return min(fasta\_records, key=len)

Entrez.email = "up1067508@upnet.gr"

ids = ["FJ817486", "JX069768", "JX469983"]

print(fetch\_and\_convert(ids))

----------------------------------------------------------------------------------------------------------------------------------

from Bio import Entrez

from Bio import SeqIO

Entrez.email = "up1067508@upnet.gr"

handle = Entrez.efetch(db="nucleotide", id=["FJ817486, JX069768, JX469983"], rettype="fasta")

records = list(SeqIO.parse(handle, "fasta")) # Get the list of SeqIO objects in FASTA format

length = [0, 0, 0]

length[0] = len(records[0].seq) # First record ID

length[1] = len(records[1].seq)

length[2] = len(records[2].seq)

last = min(length)

if last == length[0]:

print(records[0])

elif last == length[1]:

print(records[1])

elif last == length[2]:

print(records[2])

-----------------------------------------------------------------------------------------------------------------------------------

from Bio import Entrez

from Bio import SeqIO

from Bio import SeqIO

records = SeqIO.parse("gg.fastq", "fastq")

count = SeqIO.write(records, "ggcopy.fasta", "fasta")

print("Converted %i records" % count)

-----------------------------------------------------------------------------------------------------------------------------------

from Bio import SeqIO

def count\_low\_quality\_reads(fastq\_file, quality\_threshold):

count = 0

for record in SeqIO.parse(fastq\_file, "fastq"):

avg\_quality = sum(record.letter\_annotations["phred\_quality"]) / len(record.letter\_annotations["phred\_quality"]) #letter.annotations is quality scores for every base, stored in the seqrecord

#len -> returns the num items

if avg\_quality < quality\_threshold:

count += 1

return count

# Test the function with your data

fastq\_data = """@Rosalind\_0041

GGCCGGTCTATTTACGTTCTCACCCGACGTGACGTACGGTCC

+

6.3536354;.151<211/0?::6/-2051)-\*"40/.,+%)

@Rosalind\_0041

TCGTATGCGTAGCACTTGGTACAGGAAGTGAACATCCAGGAT

+

AH@FGGGJ<GB<<9:GD=D@GG9=?A@DC=;:?>839/4856

@Rosalind\_0041

ATTCGGTAATTGGCGTGAATCTGTTCTGACTGATAGAGACAA

+

@DJEJEA?JHJ@8?F?IA3=;8@C95=;=?;>D/:;74792."""

with open("fastq\_data.fastq", "w") as f:

f.write(fastq\_data)

print(count\_low\_quality\_reads("fastq\_data.fastq", 28))

-----------------------------------------------------------------------------------------------------------------------------------

from Bio.Seq import translate

# Given DNA string

dna\_string = "ATGGCCATGGCGCCCAGAACTGAGATCAATAGTACCCGTATTAACGGGTGA"

# Loop over the genetic code tables

for gen\_table in range(1, 26): # There are 25 known genetic code tables

protein\_string = translate(dna\_string, table = gen\_table, to\_stop=True) # table -> function of translate

if protein\_string == "MAMAPRTEINSTRING":

print("The index of the genetic code variant used for translation is "+ str(gen\_table))

break

-----------------------------------------------------------------------------------------------------------------------------------

from Bio.Seq import Seq

dna\_strings = {

"Rosalind\_64": "ATAT",

"Rosalind\_48": "GCATA"

}

# Initialize a counter for the matches

matches = 0

# Iterate over the DNA strings

for name, dna\_string in dna\_strings.items():

# Create a Seq object for the DNA string

my\_seq = Seq(dna\_string)

# Get the reverse complement of the DNA string

reverse\_complement = my\_seq.reverse\_complement()

# If the DNA string matches its reverse complement

if str(my\_seq) == str(reverse\_complement):

matches += 1

print("Number of DNA strings that match their reverse complements:", matches)

-----------------------------------------------------------------------------------------------------------------------------------

from Bio import SeqIO

from Bio.SeqRecord import SeqRecord

from Bio.Seq import Seq

def count\_low\_quality\_positions(records, quality\_threshold):

# Initialize a list to store quality scores

quality\_scores = []

# Parse the records

for record in records:

# Append the quality scores of the current record to the list

quality\_scores.append(record.letter\_annotations["phred\_quality"])

# Compute the mean quality score for each position

mean\_quality\_scores = []

for i in range(len(quality\_scores[0])): # assuming all records have the same length

sum\_scores = sum(record[i] for record in quality\_scores)

mean\_quality\_scores.append(sum\_scores / len(quality\_scores))

# Count the number of positions where the mean quality score falls below the threshold

num\_low\_quality\_positions = sum(i < quality\_threshold for i in mean\_quality\_scores)

return num\_low\_quality\_positions

# Your data

data = [

SeqRecord(Seq("GCCCCAGGGAACCCTCCGACCGAGGATCGT"), id="Rosalind\_0029", description="",

letter\_annotations={"phred\_quality": [ord(c)-33 for c in ">?F?@6<C<HF?<85486B;85:8488/2/"]}),

SeqRecord(Seq("TGTGATGGCTCTCTGAATGGTTCAGGCAGT"), id="Rosalind\_0029", description="",

letter\_annotations={"phred\_quality": [ord(c)-33 for c in "@J@H@>B9:B;<D==:<;:,<::?463-,,"]}),

SeqRecord(Seq("CACTCTTACTCCCTAGCCGAACTCCTTTTT"), id="Rosalind\_0029", description="",

letter\_annotations={"phred\_quality": [ord(c)-33 for c in "=88;99637@5,4664-65)/?4-2+)$)$"]}),

SeqRecord(Seq("GATTATGATATCAGTTGGCTCCGAGAGCGT"), id="Rosalind\_0029", description="",

letter\_annotations={"phred\_quality": [ord(c)-33 for c in "<@BGE@8C9=B9:B<>>>7?B>7:02+33."]})

]

# Test the function with your data

quality\_threshold = 26

print(count\_low\_quality\_positions(data, quality\_threshold))