CORSO INTEGRATO DI INFORMATICA E BIOINFORMATICA per il CLT in BIOLOGIA MOLECOLARE

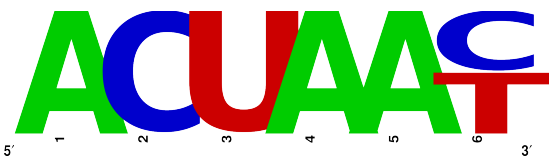
IV Esercitazione di Bioinformatica

Reperimento di sequenze di trascritti e ricerca di pattern usando le espressioni regolari: *un'applicazione alla predizione di interazioni tra RNA e proteine su scala genomica*

Introduzione

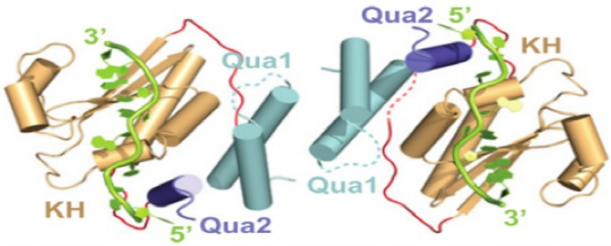
Le **RNA-binding proteins** (**RBPs**) sono proteine che legano l'RNA partecipando alla formazione di complessi ribonucleoproteici, i quali hanno moltissime funzioni diverse.
Generalmente le RBP sono dotate di domini proteici specializzati che legano agli RNA. Il **riconoscimento tra la proteina e l'RNA e' sequenza-specifico** cioè mediato dalla presenza, nell'RNA, di sequenze ben definite (RBP response elements).
Una certa RBP non lega però solo una sequenza precisa (come ACUAAT) ma anche sequenze simili (ad es. ACUAAC). E' possibile quindi comparare le sequenze riconosciute dalla medesima RBP in diversi RNA e definire delle **sequenze consenso** o dei motivi di sequenza (ACUAA[CT]) che vengono riconosciuti e legati. Questi possono essere rappresentati mediante sequence logo e definiti mediante **espressioni regolari** (Figura a fianco).

Sequence logo:

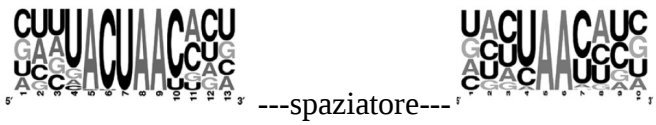


Espressione regolare:
ACUAA[CT]

Quacking (QKI) è una RBP multifunzionale che regola lo splicing alternativo nel nucleo, ma anche il processamento di altri RNA, il trasporto dal nucleo al citoplasma (Figura a fianco).
QKI sotto forma di dimero (Figura a fianco) riconosce negli RNA bersaglio una **sequenza consenso bipartita**, composta cioè da due parti di sequenza definita separate da uno spaziatore (qualsiasi sequenza) mai più lungo di 20 basi.



Regione Core ---spaziatore--- **Regione H**



---spaziatore---

Conoscendo la sequenza consenso per QKI e la sequenza di un RNA è possibile predire la loro interazione verificando se l'RNA contiene una **sequenza legante QKI** (**ACUAAY-$N_{1–20}$-UAAY,** dove Y sta per pirimidina), codificabile nell'espressione regolare **ACUAA[CU][ACGU]{1,20}UAA[CU]**.

Obiettivo 1: recuperare i trascritti umani prodotti da geni localizzati sul cromosoma 21

Nelle esercitazioni precedenti si è visto come accedere ai database di sequenze biologiche (NCBI Genome, UCSC Genome Browser). Un'altra risorsa simile è il database di Ensembl (http://www.ensembl.org/), con cui possiamo visualizzare geni e altre annotazioni biologiche sul genoma. Ad esempio diamo uno sguardo al genoma umano cliccando su "Human GRCh38.p5" nel riquadro "Browse a genome"; cliccando su "More information and statistics" ci vengono date alcune statistiche sulle annotazioni del genoma umano.
Torniamo alla pagina precedente e clicchiamo su "View karyotype": ci verrà mostrato il cariotipo umano e alcune statistiche (come prima) sulle annotazioni. Clicchiamo sul cromosoma 21 e "jump to region view". Ci verrà mostrata in dettaglio la regione su cui abbiamo cliccato e un browser a vari ingrandimenti con cui possiamo "navigare" sul cromosoma.
Questo approccio è utile se ci si concentra su una piccola regione o se si lavora su uno o pochi geni alla volta, ma come facciamo se vogliamo condurre delle analisi e ottenere informazioni su centinaia o migliaia di geni?

Ensembl fornisce un'interfaccia per selezionare e scaricare le informazioni presenti nel suo database in maniera massiva, che si chiama **BioMart (http://www.ensembl.org/biomart)**.
Usiamo Ensembl BioMart per ottenere tutti i trascritti umani annotati nel cromosoma 21.
A) Selezioniamo il database e la specie, poi applichiamo un filtro per restringere i risultati al cromosoma di interesse, e selezioniamo gli attributi che ci serviranno per identificare le sequenze

```
choose database → Ensembl Genes 84
choose dataset → Homo sapiens genes (GRCh38.p5)
filters: REGION →  v chromosome → 21
attributes: Features → GENE → Ensembl Gene ID + Ensembl Transcript ID +
 Associated Gene Name + Description
```

Clicchiamo "Results" in alto a sinistra. Ci verrà presentata una tabella con un'anteprima dei risultati, e la possibilità di scaricare/visualizzare i dati in vari formati (HTML, o tabulare)
N.B: ogni gene può generare più di un trascritto, lo si nota dal fatto che gene ID e gene names sono ripetuti più volte per transcript ID diversi.
Per scaricare i risultati si seleziona il formato preferito nel menù a tendina di " Export all results to" e si clicca "Go".

B) Quello che faremo ora però sarà **scaricare le sequenze nucleotidiche dei trascritti annotati nel cromosoma 1 umano**. Procediamo in modo simile a quanto appena fatto, solo che cambiamo gli attributi, selezionando le sequenze, e scaricheremo i risultati in un file di testo in formato FASTA, compresso (per risparmiare sul traffico di rete e spazio su disco):

```
choose database → Ensembl Genes 84
choose dataset → Homo sapiens genes (GRCh38.p5)
filters: REGION →  v chromosome → 21
attributes: Sequences → SEQUENCES: cDNA sequences (*)vedi sotto la struttura di un trascritto
Header Information → Gene Information: Ensembl Gene ID + Associated Gene Name +
Chromosome Name; Transcript Information: Ensembl Transcript ID + Transcript Start
(bp) + Transcript End (bp) + Strand + CDS start (within cDNA) + CDS end (within
cDNA)
```

Click Results
`Export all results to “compressed file (.gz)” + FASTA + unique results only → Go`

Salvare i risultati in una directory, aprire un terminale e navigare nella directory (comando '`cd percorso/della/directory`' per entrare in una directory) in cui abbiamo salvato il file.

Visionare il contenuto del file senza decomprimerlo: usiamo `zcat`, il pipe (|) e `less`:
`zcat mart_export.txt.gz | less`
oppure `zless`, la versione per file compressi di `less`
`zless mart_export.txt.gz`

(*) Struttura della sequenza di un trascritto

5'UTR (UnTranslated Region)
Codone start ATG 129
CDS (in grassetto, comprende anche i codoni di inizio e fine)
Codone stop TGA 752
3'UTR

```
>ENSG00000010072|SPRTN|ENST00000008440|1|231338256|231352521|1|450;350;129|752;449;349
GACGGGCCGTCTCGAGAGCCGGCATCTCCTAGGAGCTAGTCCTGGTCCTCGGCTAGGCGG
CTTGGGGTCGCGGCGTAACTGGGGAGCCAGCCTGACGCCGGCGGACCCCGCCTGTGATCC
TGGCAACGATGGATGATGACTTGATGTTGGCACTGCGGCTTCAGGAGGAGTGGAACTTGC
AGGAGGCGGAGCGCGATCATGCCCAGGAGTCCCTGTCGCTAGTGGACGCGTCGTGGGAGT
TGGTGGACCCCACACCGGACTTGCAGGCACTGTTTGTTCAGTTTAACGACCAATTCTTCT
GGGGCCAGCTGGAGGCCGTCGAGGTGAAGTGGAGCGTGCGAATGACCCTGTGTGCTGGGA
TATGCAGCTATGAAGGGAAGGGTGGAATGTGTTCCATCCGTCTCAGCGAACCCCTTTTGA
AGTTGAGGCCAAGAAAGGATCTTGTAGAGGTATACCATACTTTTCACGATGAGGTGGATG
AGTATCGGCGACACTGGTGGCGCTGCAATGGGCCGTGCCAGCACAGGCCACCGTATTACG
GCTATGTCAAACGAGCTACTAACAGGGAACCCTCTGCTCATGACTATTGGTGGGCTGAGC
ACCAGAAAACCTGTGGAGGCACTTACATAAAAATCAAGGAACCAGAGAATTACTCAAAAA
AAGGCAAAGGAAAGGCAAAACTAGGAAAGGAACCAGTATTGGCCGCAGAGAATAAAGGTA
CCTTCGTGTATATTCTTCTGATTTTTATGTGACCATAGCTATGATGTAAAGACAATACTG
TCCTTCAGAGAACTGGTATTAAGATAAACTTAAGGATCGTTTCTGGTGTAGAAGTCTTCA
AGTGTAGACTTAAGGAAAAAATCCCACTGTCCATGAAATGATGGTAGGAAAACAGACTTT
GCTCTGTACAGAAGTAAGTAAAAGTAGGAATAGTTTCCATGGATATTTTTATTTTTATTA
ACTTTTTTCAGTTTCTTTTTATTCAAAGAAACAAAATTCAATCTCTGATAATATTTGAGG
TAAAGTTCCTTTCCCTATCTTGACTCACTGAGTTATTAGGAAACAGAAGGCAAAAAGATT
GTCAAAATAAAAACAATAATTCAAGTAACAATGCCCGGAATATACGTCCTAACTACACCC
CTTCCTATCAGCTGGATTCTATCCAAGTGACTCTATTGATGTATGTATGTTCATTCAAAG
AATGGGAAAAGGATATGACATATATTTGCCAGTACTTCATCTTCAAGATTTACCCTTTTC
CTGTGAAGTTCAGAGTTACTGAAGATGCTTCTTCCCTTGGGAAGTTGTTGACCCAAGAAC
ATAGGTTATATTTCCCAAATCTTTAATTATTGAGTGAAAGAGCTATAGATGAATTGATAT
GGAAAGACCGTATCTTCATTTTCGTGAGTAGAAGGAAAGATAAGAATGAGGCAGCAGATT
TTCCCTCCTGGAATTACACATAAAGGACACTAAGCAATTTTCAAGGTAAATGTTGCCTTG
TTGTTGGTCTTTGGCATGATAAGATTCTTTATTTAAATATGAGAGAATTTTTTTTTATCC
TTTATATTCTCTCAATATCAGAACTCCTGAATTCTGAAGATTGCCCTCCTCCCATTAATA
GGATTGTATGGATGTAAGATGGAATAAAATACTAGTTCTTCATTTTG
```

Obiettivo 2: identificare gli RNA che possono legare la proteina Quaking (QKI) attraverso un sito bersaglio riconosciuto in maniera sequenza-specifica

Trova i matching del pattern di binding (vedere Introduzione)
ACUAAY-$N_{1,20}$-UAAY
dove Y sta per una pirimidina (C, T o U);
$N_{1,20}$ sta per “da 1 fino a 20” (qualsiasi tipo di) basi.

NB: abbiamo scaricato cDNA, quindi abbiamo timina (T) al posto di uracile (U)!

Useremo le espressioni regolari (regular expressions o *regex*) per cercare un pattern in una sequenza.

Regular expression

Le espressioni regolari sono delle sequenze di (meta)caratteri con cui possiamo rappresentare pattern e motivi di stringhe.

Regex metacharacters (some examples)

PATTERN	MATCHES	CHARACTER	MEANING
CGC	CGCAGCCGC CAC AGCGC	most characters	themselves
^CGC	CGCAGCCGC CAC AGCGC	^	start of line
CGC$	CGCAGCCGC CAC AGCGC	$	end of line
CA*T	CAAT CAAAC CAT CAAAT CT	x*	zero or more x
C.T	CAT CAAT CGT CT CCT CTC	.	any character
C[AG]T	CAT CCT CGT CAAT CT CTT	[…]	any one character of ...
C[^AG]T	CAT CCT CGT CAAT CT CTT	[^...]	any one character not of ...
C[A]{1,3}T	CAT CCT CAAT CAAAT CAAAAT	[...]{n,m}	n to m times characters of ...
C\>	CAC CC AAA ACA CA C	\< \>	start and end of word

Regex combinations

PATTERN	MATCHES
[AT][CG]	ATTCAGTCGCATGCTACGTCGTAGCGTA
A[CG]*T	ATTCAGTCGCATGCTACGTCGTAGCGTA
A[CG][CG]*T	ATTCAGTCGCATGCTACGTCGTAGCGTA
\<A[CGT]*\>	AT CAG TC GC ATG CTAC GTC GTA GC GTA
\<[CG][^]*[^A]\>	AT CAG TC GC ATG CTAC GTC GTA GC GTA

Le regex possono risultare difficili da comporre e da leggere ed è molto facile sbagliarle. Per questo fare delle prove è molto importante!

Ad esempio:

PATTERN	MATCHES	BETTER PATTERNS
chr1	chr1 chr2 chr11 chr21	“chr1 ”, chr1\>, chr1\s
<.*>	<aa> cc <bb>	<[^>]*>

Definire il pattern ACUAAY-$N_{1,20}$-UAAY con le regex

Ritorniamo alla nostra domanda iniziale: identificare il pattern ACUAAY-$N_{1,20}$-UAAY. Per comodità, consideriamo U come T, quindi avremo ACTAAY-$N_{1,20}$-TAAY
Definiamo l'espressione regolare corrispondente, componendola per passi successivi:

1. c'è una parte “fissa”: **ACTAA**Y-$N_{1,20}$-TAAY
proviamo a contare quante righe contengono la parte fissa del pattern

```
zgrep -c 'ACTAA' mart_export.txt.gz
```

per vedere i match evidenziati con un colore diverso:

```
zgrep --color=always 'ACTAA' mart_export.txt.gz | less -R
```

2. poi c'è un carattere variabile che può essere o C o T: ACTAA**Y**-$N_{1,20}$-TAAY; in questo caso il pattern viene scritto come [CT] e matcherà sia ACTAA**C** che ACTAA**T**
Y = [CT]

```
zgrep -c 'ACTAA[CT]' mart_export.txt.gz
```

(il numero di righe è inferiore al precedente perché la sequenza che cerchiamo è più lunga e quindi più specifica)

```
zgrep --color=always 'ACTAA[CT]' mart_export.txt.gz | less -R
```

3. poi abbiamo una parte variabile di caratteri, da 1 fino a 20: ACTAAY-**$N_{1,20}$**-TAAY
N = [ACTG] = una base qualsiasi
$N_{1,20}$= [ACTG]{1,20} = una qualsiasi “stringa di basi” lunga da 1 a 20

il **comando grep** interpreta il carattere `{` come un carattere qualsiasi, ma noi gli vogliamo dire che lo deve interpretare come un metacarattere, quindi usiamo la sequenza di escape che è il carattere backslash `\` ; quindi la combinazione `\{` identifica la parentesi graffa come un metacarattere. Stessa cosa per `}`.
Un semplice esempio per vedere questo tipo di pattern: vediamo le righe che contengono delle sequenze di T lunghe 18, 19 o 20 basi

```
zgrep --color=always 'T\{18,20\}' mart_export.txt.gz | less -R
```

e quelle che contengono sequenze di T e C lunghe 18, 19 o 20 basi

```
zgrep --color=always '[CT]\{18,20\}' mart_export.txt.gz | less -R
```

aggiungiamo la parte variabile al pattern che stiamo costruendo: ACTAA[CT][ACGT]{1,20}

```
zgrep --color=always 'ACTAA[CT][ACGT]\{1,20\}' mart_export.txt.gz | less -R
```

4. la parte restante del pattern è una parte “fissa” e una pirimidina ACTAAY-$N_{1,20}$-**TAAY**. Componiamo come prima TAA[CT] e aggiungiamo al resto del pattern ottenendo il pattern voluto:
ACTAA[CT][AGCT]\{1,20\}TAA[CT]

```
zgrep --color=always 'ACTAA[CT][ACGT]\{1,20\}TAA[CT]' mart_export.txt.gz | less -R
```

Problemi ancora da risolvere:

- Da quali trascritti provengono i matches?
- In quali posizioni dei trascritti troviamo il pattern?
- Se una sequenza è spezzata in più righe (il formato FASTA usa solitamente max 60 nt per riga) non consideriamo possibili pattern a cavallo delle righe (**)
- Quanto lungo è lo spaziatore $N_{1,20}$ nei match?

Rispondere a queste domande usando solo gli strumenti della shell può risultare poco pratico.

(**) Esempio di un trascritto in cui abbiamo trovato la sequenza a cavallo di due righe nel FASTA
QRE pattern trovato: ***ACTAATCTAAT*** in posizione 479-490 del trascritto ENST00000256652

```
>ENSG00000134256|CD101|ENST00000256652|1|117001750|117034250|1|
2497;2893;2086;493;112;1681;69;910;1297|2892;3134;2496;909;492;2085;111;1296;1680
AGCATTTGTCACTCAACCTCTGAATGTTAGTGACACTATTGGGACGAAAAAGGACTGTGC
TGGCCCAAATGGCAGGCATCTCATATGTGGCATCTTTCTTTCTCCTTCTGACTAAGCTCA
GCATTGGCCAGAGAGAAGTAACAGTTCAGAAAGGACCACTGTTTAGAGCTGAAGGTTACC
CAGTCAGCATTGGCTGCAATGTAACTGGCCACCAGGGACCTTCTGAGCAGCATTTCCAGT
GGTCTGTTTACCTGCCGACAAACCCGACCCAGGAAGTCCAGATCATTAGCACCAAGGATG
CTGCCTTCTCTTACGCAGTATATACGCAGCGGGTGCGAAGCGGAGACGTCTACGTGGAGA
GGGTCCAGGGCAACTCAGTCTTGTTGCACATCTCAAAACTCCAGATGAAGGATGCTGGCG
AGTATGAGTGTCACACACCAAACACTGATGAGAAATACTATGGAAGTTACAGTGCAAAGA
CTAATCTAATTGTTATTCCAGATACCCTCTCTGCCACCATGAGTTCTCAGACTCTCGGTA
AGGAGGAAGGTGAGCCATTAGCCCTCACCTGTGAGGCATCCAAAGCCACAGCCCAACATA
CTCACCTCTCTGTCACCTGGTACCTAACACAGGATGGAGGAGGAAGCCAAGCCACTGAGA
TTATTTCTCTCTCCAAAG[....................]CCAGAGAGCAAGCTAAAAG
TGAATTCAAGGAGTCAAGTCCAAGAGCTCTCCATCAACTCCAACACTGATATAGAATGTA
GCATCTTGTCCCGGTCCAATGGAAACCTTCAGTTAGCCATTATTTGGTATTTTTCTCCTG
TTTCCACTAATGCCTCTTGGCTAAAGATCCTGGAGATGGACCAAACCAATGTTATAAAAA
CTGGGGATGAGTTTCACACCCCACAGAGAAAACAAAAATTTCATACTGAGAAGGTTTCCC
AAGACTTATTTCAGCTGCACATTCTGAATGTGGAAGACAGCGATCGGGGCAAATATCACT
GTGCTGTGGAGGAATGGCTCCTGTCTACAAATGGCACTTGGCACAAGCTTGGAGAAAAGA
AGTCAGGACTAACAGAATTGAAACTCAAGCCCACAGGAAGTAAGGTACGTGTCTCCAAAG
TGTACTGGACCGAAAATGTGACTGAGCACAGAGAAGTGGCCATCCGCTGCAGCCTGGAGA
GTGTAGGCAGCTCAGCCACTCTGTACTCTGTGATGTGGTACTGGAACAGAGAAAACTCTG
GAAGTAAATTGCTGGTGCACTTGCAACATGATGGCTTGCTGGAGTATGGGGAAGAGGGGC
TCAGGAGGCACCTGCACTGTTACCGTTCATCCTCTACAGACTTTGTCCTGAAGCTTCATC
AGGTGGAGATGGAGGATGCAGGAATGTACTGGTGTAGGGTGGCAGAGTGGCAGCTCCATG
GACACCCAAGCAAGTGGATTAATCAAGCATCCGATGAGTCACAGCGGATGGTGCTCACGG
TGCTGCCTTCAGAGCCCACGCTTCCTTCCAGGATCTGCTCCTCGGCCCCTTTACTCTATT
TCCTGTTCATCTGTCCCTTCGTCCTGCTCCTCCTTCTGCTCATCTCCCTCCTCTGCTTAT
ACTGGAAGGCCAGGAAGTTGTCAACACTGCGTTCCAACACACGGAAAGAAAAAGCTCTCT
GGGTGGACTTGAAAGAGGCTGGAGGTGTGACCACAAATAGGAGGGAAGACGAGGAGGAAG
ATGAAGGCAACTGAATCCCAAGAGGCACCTGCAGCCAGGAAGGAAAGGTGGGGGCTTTTT
TGGGCAAGTTACCTAGGAATCAGAGGGAGCATTCACTGAGTGC
```

Obiettivo 3: Creare uno script Python che ci consenta di maneggiare più agevolmente le informazioni che ci interessano

Useremo nuovamente le librerie Biopython per leggere il FASTA file; useremo anche la libreria 're' di Python che implementa le funzioni alle espressioni regolari.
Di seguito il codice dello script, che potete scaricare da http://compgen.bio.unipd.it/~enrico/QRE_motif_finder.py
Lanceremo il nostro programma da linea di comando.
`python QRE_motif_finder.py -h`

```
#!/usr/bin/env python

'''
Given a FASTA file with (multiple) sequences returns the transcript ids
in which QRE pattern (1) is found.
1. de Bruin, R. G. et al. Quaking promotes monocyte differentiation into
pro-atherogenic macrophages by controlling pre-mRNA splicing and gene
expression. Nat Commun 7, 10846 (2016).
'''

import re, argparse, gzip
from Bio import SeqIO

'''Return the indexes and the stretch of the QRE pattern matches '''
'''in a string readable format [start-end:stretch, start-end:stretch, ...]'''
def find_qre_pattern(sequence):
    # store the QRE pattern for subsequent use
    qre_pattern = re.compile('ACTAA[CT][AGCT]{1,20}TAA[CT]')
    # use finditer in order to get also the positions of the match
    matches = qre_pattern.finditer(sequence)
    result = []
    # the pattern may be found multiple times within the same sequence
    for m in matches:
        # three element list
        matched_pos = [m.start(), m.end(), m.group(0)]
        result.append(matched_pos)
    # the result will be a list of three-element lists
    return result

if __name__ == '__main__':

    # just handle input parameters
    parser = argparse.ArgumentParser(description = \
        '''Given a FASTA file with (multiple) sequences returns the transcript ids '''
        '''in which QRE pattern (1) is found. '''
        '''[1]: de Bruin, R. G. et al. Quaking promotes monocyte differentiation into '''
        '''pro-atherogenic macrophages by controlling pre-mRNA splicing and gene '''
        '''expression. Nat Commun 7, 10846 (2016).''')
    parser.add_argument('infile', type = str,
                help = 'A gzipped FASTA file')
    args = parser.parse_args()

    # if compressed file use appropriate function
    if args.infile.endswith('.gz'):
        open = gzip.open
```

```
# open file
with open(args.infile, 'r') as f:
    # load FASTA entries
    transcripts = SeqIO.parse(f, "fasta")
    # cicle through the entries
    for trs in transcripts:
        # get sequence
        seq = str(trs.seq)
        # check if pattern is in current sequence
        matches = find_qre_pattern(seq)
        # if pattern matches one or more times in sequence
        if matches:
            # split FASTA header into separate fields
            trs_fields = trs.id.split('|')
            # trnascript ID
            trID = trs_fields[2]
            # gene symbol
            geneName = trs_fields[1]
            # format matches as a string and put them into a list
            matched_pos = []
            for m in matches:
                qreStart = m[0]
                qreEnd   = m[1]
                qreString= m[2]
                matched_pos_str = str(qreStart)+ '-' +\
                          str(qreEnd) + ':' +\
                          str(qreString)
                matched_pos.append(matched_pos_str)

            # print results in TAB separated format
            print '\t'.join([trID, geneName, str(matched_pos)])
```

Possiamo eseguire lo script direttamente sul file compresso

```
python QRE_motif_finder.py mart_export.txt.gz | less -S
```

Reindirizziamo l'output ad altri programmi della shell, tramite il pipe, per fare qualche statistica (es. numero trascritti, geni, etc.)

Quanti trascritti contengono il pattern? 132 (verifica questo numero)

Obiettivo 4: quante volte troviamo il QRE per ogni trascritto?

Modificare lo script QRE_motif_finder.py in modo che restituisca solo il numero di pattern trovati per trascritto

Obiettivo 5: c'è (almeno) un QRE pattern nella regione codificante (CDS) del trascritto?

Scrivi uno script che trovi e restituisca le sequenze QRE comprese in regioni codificanti dei trascritti. Puoi partire dallo script QRE_motif_finder.py e modificarlo ove necessario.
Suggerimento:
usare split(';') per ottenere i vari CDS start/end e convertirli in numero, poi trovare il minimo di start e il massimo di end per avere la regione codificante
ATTENZIONE:
alcuni trascritti non sono codificanti e quindi non avranno i campi CDS start e CDS end nell'header! Vi consiglio di verificare la lunghezza della lista ritornata da header.split('|').

Confronta la tua soluzione con questa
http://compgen.bio.unipd.it/~enrico/QRE_motif_finder_in_CDS.py