introduction_to_python_for_the_central_dogma_supervisor_copy

November 27, 2020

1 Introduction to Python for The Central Dogma of Biology

1.1 Aims and objects

The goal of this session is to let you practice working with basic biological sequence data in Python3.

It is assumed attendees have a basic grasp of Python and have likely attended the Python 101 sessions held by DoCSoc.

It is advised to use **Anaconda Python** for this session (you can access it through Imperial College AppsAnywhere service) but the main requirement is simply to have a working version of **Jupyter Notebook** with the **Python3 kernal**. You can alternatively write your script as a Python file and run it in a terminal.

We recommend saving your Jupyter Notebook in the **same** directory as the files required for the session.

By the end of the session, you should be able to: - Open and handle sequence data from a FASTA file (without using the Biopython module) - Translate a DNA into a protein sequence in all six reading frames - Search for short motifs/domains in a (protein) sequence.

1.2 Required files

If not downloaded, you can save the required files for today's session from our GitHub page.

Please save them in a **new folder/directory** along with a blank Jupyter Notebook (or add to this one) for writing your code.

You should have six files: - `sars-cov2_spike_protein_mRNA.fasta` - `a2d_mRNA.fasta` - `VWA_domain.fasta` - `VWA-like_domain.fasta` - `cache_domain.fasta`

… including this one.

1.3 Tasks

1.3.1 Parsing the SARS-COV2 Spike Protein mRNA sequence

Parsing is the action of converting one data structure into another. In this case, you will be converting the sequence stored in a text file into a Python string object.

1) Use the `open(...)` function to import the `sars-cov2_spike_protein_mRNA.fasta` file as a Python object.

2) Write a function to convert the sequence from the FASTA file into a single string.

[ ]:
```
# Part 1)

# COVID EXAMPLE
spike_protein_file = open('sars-cov2_spike_protein_mRNA.fasta')
```

[ ]:
```
# Part 2)

# Function to convert a FASTA file into a DNA sequence string object.

def FastaToSequence(fasta_file):

    # Initialise empty string
    DNA_sequence = ''

    # Iterate through lines of FASTA file
    for line in fasta_file:

        # Omit the description line
        if line[0] == '>':
            pass

        # Append all other lines (assumes they are fragments of sequence) to␣
 ↪empty string
        else:
            line = line[:-1]
            DNA_sequence += line

    # Returns FASTA's DNA sequence as a string
    return DNA_sequence
```

[ ]:
```
# Part 2)

spike_prot_mrna_seq = FastaToSequence(spike_protein_file)
```

1.3.2 Translate the mRNA sequence into its corresponding peptide

Translation is the process by which the cell synthesises a protein sequence (a polymer of amino acid residues) according to the ribonucleic acid code of a mRNA molecule. The mRNA sequence contains all the information to code for the primary structure of its protein through the degenerate assignment of amino acids to codons (three RNA base sequences).

3) Write a second function to translate this 'exon-less' gene into its corresponding peptide. You can copy and paste the the amino acid-codon dictionary below into your script.

[ ]:
```
# Codon to amino acid reference table

```

```
codon_dict = { # https://www.geeksforgeeks.org/dna-protein-python-3/
    'ATA':'I', 'ATC':'I', 'ATT':'I', 'ATG':'M',
    'ACA':'T', 'ACC':'T', 'ACG':'T', 'ACT':'T',
    'AAC':'N', 'AAT':'N', 'AAA':'K', 'AAG':'K',
    'AGC':'S', 'AGT':'S', 'AGA':'R', 'AGG':'R',
    'CTA':'L', 'CTC':'L', 'CTG':'L', 'CTT':'L',
    'CCA':'P', 'CCC':'P', 'CCG':'P', 'CCT':'P',
    'CAC':'H', 'CAT':'H', 'CAA':'Q', 'CAG':'Q',
    'CGA':'R', 'CGC':'R', 'CGG':'R', 'CGT':'R',
    'GTA':'V', 'GTC':'V', 'GTG':'V', 'GTT':'V',
    'GCA':'A', 'GCC':'A', 'GCG':'A', 'GCT':'A',
    'GAC':'D', 'GAT':'D', 'GAA':'E', 'GAG':'E',
    'GGA':'G', 'GGC':'G', 'GGG':'G', 'GGT':'G',
    'TCA':'S', 'TCC':'S', 'TCG':'S', 'TCT':'S',
    'TTC':'F', 'TTT':'F', 'TTA':'L', 'TTG':'L',
    'TAC':'Y', 'TAT':'Y', 'TAA':'_', 'TAG':'_',
    'TGC':'C', 'TGT':'C', 'TGA':'_', 'TGG':'W',
}
```

[ ]:
```
# Part 3)

# Function to translate a given DNA sequence into a peptide sequence

def Translation(DNA_sequence):     # type(argument) = string

    # Defining codon start and end indices (bounds)
    start_boundary =  0
    end_boundary = 3

    # Initialise empty string
    protein_sequence = ''

    # Iterate through the given DNA sequence until end is reached
    while end_boundary < len(DNA_sequence):

        # Current codon defined
        codon = DNA_sequence[start_boundary:end_boundary]

        # Amino acid coded for by codon retrieved
        amino_acid = codon_dict[codon]

        # Amino acid added to protein sequence
        protein_sequence += amino_acid

        # Codon bounds incremented by 3 for subsequent iteration in 'while' loop
        start_boundary += 3
        end_boundary += 3
```

```
    # Return protein sequence as string
    return protein_sequence
```

[ ]:
```
# Part 3)

spike_prot = Translation(spike_prot_mrna_seq)
```

1.3.3 Finding all six reading frames

Due to the double-stranded, triplicate-nature of DNA, a protein-coding gene can exist across six reading frames (the start to finish stretch of bases that code for the peptide). If this is unfamiliar, spend a minute to consider how this is achieved.

4) From the sequence you generated using your first function, create a new function that returns all possible peptides across the gene's six reading frames. How does reversing the sequence affect translation? You can call upon other functions from previous steps (or write additional functions to further abstract the problem) within this new function to save repeating blocks of code.

[ ]:
```
# Part 4)

# Function to generate the (reverse) complementary strand from a given DNA␣
 ↪sequence

def ComplmentaryStrand(DNA_sequence):

    # Initialising empty string
    complementary_DNA_sequence = ''

    # Iterates through bases (characters) in DNA sequence (string)
    for base in DNA_sequence[::-1]:

        if base == 'A':    # A --> T
            complementary_DNA_sequence += 'T'

        elif base == 'T':  # T --> A
            complementary_DNA_sequence += 'A'

        elif base == 'C':  # C --> G
            complementary_DNA_sequence += 'G'

        elif base == 'G':  # G --> C
            complementary_DNA_sequence += 'C'

    # Return complementary DNA sequence as string
    return complementary_DNA_sequence
```

[ ]:
```
# Part 4)

# Function to translate all reading frames (RFs) from a given DNA sequence

def TranslationAllRFs(DNA_sequence):

        # Reverse strand generated
        reverse_complementary_strand = ComplmentaryStrand(DNA_sequence)

        # Reading frames defined in list
        RFs = [0, 1, 2]

        # Empty list for forward strand reading frames to be subsequently added
        forward_RFs = []

        # Empty list for reverse complementary strand reading frames to be␣
 ↪subsequently added
        reverse_complementary_RFs = []

        # Iterate through all 6 reading frames
        for RF in RFs:

            # Forward 3 reading frames
            forward_RF = DNA_sequence[RF:]
            forward_RF_peptide = Translation(forward_RF)
            forward_RFs.append(forward_RF_peptide)

            # Reverse 3 reading frames
            reverse_RF = reverse_complementary_strand[RF:]
            reverse_RF_peptide = Translation(reverse_RF)
            reverse_complementary_RFs.append(reverse_RF_peptide)

        # All reading frames merged to return ordered list of strings
        all_RFs = forward_RFs + reverse_complementary_RFs

        # Return reading frames as list of translated sequence strings
        return all_RFs
```

[ ]:
```
# Part 4)

# Test
all_spike_prot_RFs = TranslationAllRFs(spike_prot_mrna_seq)
```

1.3.4 Searching for motifs

Motifs are short stretches of amino acids that are requisite for a protein's function. Often, the rest of the sequence is free to change (mutate) over time, providing its motif residues remain unchanged (conserved). Due to similarities in side chain chemistry of some amino acids and the 3D structure of proteins, motifs can, in reality, mutate. Profiles of position-dependent amino acid frequency can be created for motifs, capturing this tug of war between random mutation and natural selection. For the sake of simplicity, we will only be considering exact matches between query motifs and a given sequence here.

5) Write a function that takes in two arguments: a motif and a reference sequence. Both should be treated as strings. The function should search for the motif in the reference sequence and print (or return) the 'start' and 'end' positions of the motif within the reference. NB. Sequence indexes used in biology typically start at +1, whereas Python uses a starting index of 0.

6) Open the file `sars-cov2_arbitrary_motif.fasta` and parse the motif sequence as a string to a new variable. Using your motif-searching function, what reading frame of the SARS-CoV2 Spike Protein does it exist and what are its start/end positions?

[ ]:
```
# Part 5)

# Function to search a given motif for a given motif (both as strings)

def MotifSearch(protein_seq, motif):

    # Initial position index
    count = 0

    # Iterate through positions in given protein sequence until a matching␣
↪string is found ...
    while count < len(protein_seq):

        search_area = protein_seq[count:(count + len(motif))]

        if search_area == motif:

            print('Motif start position:', (count + 1))
            print('Motif end position:', (count + len(motif)))

            break    # Ends loop once match found. Assumes no further␣
↪occurences of motif
                     # ... assumption true here but can you come up with a␣
↪better solution to this problem?

        # Increment position index
        count += 1

    # ... position of match (motif in parent sequence) is printed.
```

[ ]:
```
# Part 6)

# Open motif file
SC2_arb_motif = open('sars-cov2_arbitrary_motif.fasta')

# Parse file to sequence
SC2_arb_motif = FastaToSequence(SC2_arb_motif)

# Test
RF_count = 1

for RF in all_spike_prot_RFs:

    print('Reading frame:', RF_count)

    MotifSearch(RF, SC2_arb_motif)

    RF_count += 1
```

1.3.5 Analysing the human protein alpha-2-delta-1

7) Using the functions you have written, import the mRNA sequence of the alpha-2-delta-1 protein, translate it in all six reading frames and search for VWA, VWA-like and Cache domains within each peptide sequence. You can find the domain sequences from the following files:

- `VWA_domain.fasta`
- `VWA-like_domain.fasta`
- `chache_domain.fasta`

8) With which reading frame do these domains match? What are the start/end positions of the domains in the reading frame(s)?

[ ]:
```
# Part 7)

# They should write a loop for this. It is displayed in separate cells here to␣
 ↪make the answers clear
# for the instructor.

# NEUROBIOLOGY EXAMPLE
a2d_dna = FastaToSequence(open('a2d_mRNA.fasta'))

a2d_protein = Translation(a2d_dna)

a2d_protein_allRFs = TranslationAllRFs(a2d_dna)
```

[ ]:
```
# Part 8)
```

```
# VWA domain sequence
vwa_domain = a2d_protein[140:265]

print(vwa_domain)

MotifSearch(a2d_protein, vwa_domain)
```

[ ]:
```
# Part 8)

# VWA-like domain sequence
vwa_like_domain = a2d_protein[276:456]

print(vwa_like_domain)

MotifSearch(a2d_protein, vwa_like_domain)
```

[ ]:
```
# Part 8)

# Cache domain sequence
cache_domain = a2d_protein[579:664]

print(cache_domain)

MotifSearch(a2d_protein, cache_domain)
```

All of the domains are in RF1