homework03

October 23, 2024

1 Homework03 (10 points): The genetic code as a python `dict`

Any code that's already here in the notebook or in the lectures is fair game. I've provided outline code for some of the problems; you are welcome to use that but you don't have to. There are also some hints/examples at the very end of this notebook.

Please post questions to the tfcb_2022 slack channel. Phil is also available via email (pbradley@fredhutch.org), and he loves answering Python questions. Seriously. Don't hesitate to reach out if you are feeling stuck. Direct messaging him in slack also works great.

Here's a python dictionary that stores the genetic code mapping. The keys are codons and the values are the amino acids that those codons code for. '*' represents STOP.

[18]:
```
genetic_code = {
    'AAA': 'K', 'AAC': 'N', 'AAG': 'K', 'AAT': 'N',
    'ACA': 'T', 'ACC': 'T', 'ACG': 'T', 'ACT': 'T',
    'AGA': 'R', 'AGC': 'S', 'AGG': 'R', 'AGT': 'S',
    'ATA': 'I', 'ATC': 'I', 'ATG': 'M', 'ATT': 'I',
    'CAA': 'Q', 'CAC': 'H', 'CAG': 'Q', 'CAT': 'H',
    'CCA': 'P', 'CCC': 'P', 'CCG': 'P', 'CCT': 'P',
    'CGA': 'R', 'CGC': 'R', 'CGG': 'R', 'CGT': 'R',
    'CTA': 'L', 'CTC': 'L', 'CTG': 'L', 'CTT': 'L',
    'GAA': 'E', 'GAC': 'D', 'GAG': 'E', 'GAT': 'D',
    'GCA': 'A', 'GCC': 'A', 'GCG': 'A', 'GCT': 'A',
    'GGA': 'G', 'GGC': 'G', 'GGG': 'G', 'GGT': 'G',
    'GTA': 'V', 'GTC': 'V', 'GTG': 'V', 'GTT': 'V',
    'TAA': '*', 'TAC': 'Y', 'TAG': '*', 'TAT': 'Y',
    'TCA': 'S', 'TCC': 'S', 'TCG': 'S', 'TCT': 'S',
    'TGA': '*', 'TGC': 'C', 'TGG': 'W', 'TGT': 'C',
    'TTA': 'L', 'TTC': 'F', 'TTG': 'L', 'TTT': 'F'
}
genetic_code ['AAA']
```

[18]:
```
'K'
```

2 Q1 (1 point). What does `TGC` code for?

Create a string called codon that is equal to `'TGC'` and use the genetic_code dictionary to look up its translation.

Recall that you look up a value in a dictionary using the syntax `dictionary[key]`, where `dictionary` is the name of the dictionary and `key` is the name of the key.

[19]:
```
# what does TGC code for?
codon = genetic_code['TGC']
```

3 Q2 (1 point). How many codons code for leucine ('L')?

Do this by looping through the keys (ie, codons) in the `genetic_code` dictionary and counting how many code for leucine.

[20]:
```
# initialize the count to 0
num_leucine_codons = 0

# for-loops over dictionaries give us the keys (ie, codons)
for codon in genetic_code:
    if genetic_code[codon] == 'L':
        num_leucine_codons += 1

print(num_leucine_codons)
```

```
6
```

4 Q3 (2 points). How many codons code for each amino acid?

Do this with a `for` loop over the amino acids (plus stop aka `'*'`), inside of which you figure out how many codons code for that amino acid and print out the amino acid and the number of codons. You should get an output like this:

```
* 3
A 4
C 2
D 2
```

etc.

We need a list of all the amino acids (plus the stop codon). You could hard code those, with something like:

```
amino_acids_plus_stop = ['*','A','C','D'] # etc
```

But it's easier to build the list directly from the dictionary. For starters, here's how you get all the values in a dictionary:

[21]:
```
genetic_code.values()
```

[21]:
```
dict_values(['K', 'N', 'K', 'N', 'T', 'T', 'T', 'T', 'R', 'S', 'R', 'S', 'I',
'I', 'M', 'I', 'Q', 'H', 'Q', 'H', 'P', 'P', 'P', 'P', 'R', 'R', 'R', 'R', 'L',
'L', 'L', 'L', 'E', 'D', 'E', 'D', 'A', 'A', 'A', 'A', 'G', 'G', 'G', 'G', 'V',
'V', 'V', 'V', '*', 'Y', '*', 'Y', 'S', 'S', 'S', 'S', '*', 'C', 'W', 'C', 'L',
 'F', 'L', 'F'])
```

That list-like thing contains all the amino acids, but there are duplicates, and they aren't in sorted order. An easy way to eliminate duplicates in python is to turn the list into a `set`, which is another built-in type that stores unique items. (`set`s are optimized for fast membership checking and operations like intersection and unions, but that's not relevant for this HW.)

```
[22]: # set is another builtin python type, like list or dict. It's a nice way of␣
       ↪removing duplicates.
      # Here we build a set from all the values (amino acids) in the genetic code␣
       ↪dictionary.
      # Then we use the "sorted" built-in python function to turn the set into a␣
       ↪sorted
      # list.
      amino_acids_plus_stop = sorted(set(genetic_code.values()))
      print(amino_acids_plus_stop)
```

```
['*', 'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q', 'R',
'S', 'T', 'V', 'W', 'Y']
```

Now the idea is to loop over this list of amino acids and count, for each one, how many codons code for it. For that you could use your solution to Question 2.

```
[23]: # how many codons for each amino acid?
      for aa in amino_acids_plus_stop:
          count = 0
          for codon in genetic_code:
              # do something inside here...
              if genetic_code[codon] == aa:
                  count += 1
          # now finally write out the answer
          print(aa, count)
```

```
* 3
A 4
C 2
D 2
E 2
F 2
G 4
H 2
I 3
K 2
L 6
M 1
N 2
P 4
Q 2
R 6
```

```
S 6
T 4
V 4
W 1
Y 2
```

5 Q4 (3 points). Write a translation function that takes in a DNA sequence and returns a protein sequence.

5.0.1 ACK– we had so many wonderful questions during class that I did not get to the material on defining functions! I'm sorry about that. Luckily, defining new functions in Python is very straightforward. Check out the examples in lecture06 ("Defining new functions") and at the end of this notebook, and the answer template in the next code block. If you don't want to try it, you can instead just write the code to translate `dna_seq` as a stand-alone code block (see the second code block below)

Extra nucleotides at the end should be ignored.

Use it to translate the sequence `CCTCATATTTTGTGAATTTCTTGAGCTTGAGGTCGTGAGGCTACTTGAACTGAGGCTTGTCATGAGCGT`

If you are so inclined, see how short you can make your function. Can you do it in one line? See the examples at the end of this notebook if you are unsure of the syntax for defining functions and for hints on translating.

[24]:
```python
# translation function
def translate(dna_seq):
    # here's how we count the number of complete codons.
    num_codons = len(dna_seq)//3
    # start with an empty string
    translation = ''
    # now build up the translation one amino acid at a time...
    nucleotide = 0
    for _ in range(num_codons):
        codon = dna_seq[nucleotide:nucleotide+3]
        translation += genetic_code[codon]
        nucleotide += 3
    return translation
translate('CCTCATATTTTGTGAATTTCTTGAGCTTGAGGTCGTGAGGCTACTTGAACTGAGGCTTGTCATGAGCGTT')
```

[24]:
```
'PHIL*IS*A*GREAT*TEACHER'
```

6 OR, without using a function:

[25]:
```python
# write code to translate dna_seq into an amino acid string
dna_seq =␣
 ↪'CCTCATATTTTGTGAATTTCTTGAGCTTGAGGTCGTGAGGCTACTTGAACTGAGGCTTGTCATGAGCGTT'
```

```
# here's how we count the number of complete codons.
num_codons = len(dna_seq)//3

# start with an empty string
translation = ''
# now build up the translation one amino acid at a time...

# ... (put stuff here)

print('the translation is:', translation)
```

```
the translation is:
```

7 Q5 (3 points). Translate a SARS COV2 genome sequence; what is the longest open reading frame?

Here by open reading frame I just mean a stretch of codons that doesn't contain any stop codons.

You can answer either in terms of the length of the reading frame at the nucleotide or protein level, just be clear.

This little snippet will read the SARS COV2 genome DNA sequence into the string called `genome`:

[26]:
```
#
filename = 'data/sars_cov2_genome_sequence.txt'
# lines is a list of strings, one containing each line of the file
lines = open(filename,'r').readlines()
# the first line is a header, get rid of that one using list slicing (ie,␣
  ↪lines[1:])
# then join them together using the string join method (ie, ''.join(...))
# each line ends with a newline character, remove those with the string replace␣
  ↪method
# see how we can do lots of things in one line (first slicing, then joining,␣
  ↪then replacing):
# can you see why they happen in that order?
genome = ''.join(lines[1:]).replace('\n','')
print(len(genome))
```

```
29903
```

Now translate genome into a protein sequence using the `translate` function from the previous question, OR by cutting and pasting the translation block of code you wrote up above. (Functions are handy because we don't have to duplicate code).

[27]:
```
prot_seq = translate(genome)
```

Now you want to figure out the longest stretch of letters (amino acids) in `prot_seq` that doesn't contain any `*` characters. You can do this however you like, but as a hint, consider using the string.split() function, where you split on '*'s.

[28]:
```python
l = prot_seq.split('*')

# now do something with l...
long_short = sorted(l, key=len, reverse=True)
longest_orf = long_short[0]
length = len(longest_orf)
print('The longest open reading frame in amino acids in the SARS COV2 genome␣
 ↪is', longest_orf)
print('It is', length, 'amino acids long.')
```

```
The longest open reading frame in amino acids in the SARS COV2 genome is CTIVFKR
VCGVSAARLTPCGTGTSTDVVYRAFDIYNDKVAGFAKFLKTNCCRFQEKDEDDNLIDSYFVVKRHTFSNYQHEETIYNLL
KDCPAVAKHDFFKFRIDGDMVPHISRQRLTKYTMADLVYALRHFDEGNCDTLKEILVTYNCCDDDYFNKKDWYDFVENPD
ILRVYANLGERVRQALLKTVQFCDAMRNAGIVGVLTLDNQDLNGNWYDFGDFIQTTPGSGVPVVDSYYSLLMPILTLTRA
LTAESHVDTDLTKPYIKWDLLKYDFTEERLKLFDRYFKYWDQTYHPNCVNCLDDRCILHCANFNVLFSTVFPPTSFGPLV
RKIFVDGVPFVVSTGYHFRELGVVHNQDVNLHSSRLSFKELLVYAADPAMHAASGNLLLDKRTTCFSVAALTNNVAFQTV
KPGNFNKDFYDFAVSKGFFKEGSSVELKHFFFAQDGNAAISDYDYYRYNLPTMCDIRQLLFVVEVVDKYFDCYDGGCINA
NQVIVNNLDKSAGFPFNKWGKARLYYDSMSYEDQDALFAYTKRNVIPTITQMNLKYAISAKNRARTVAGVSICSTMTNRQ
FHQKLLKSIAATRGATVVIGTSKFYGGWHNMLKTVYSDVENPHLMGWDYPKCDRAMPNMLRIMASLVLARKHTTCCSLSH
RFYRLANECAQVLSEMVMCGGSLYVKPGGTSSGDATTAYANSVFNICQAVTANVNALLSTDGNKIADKYVRNLQHRLYEC
LYRNRDVDTDFVNEFYAYLRKHFSMMILSDDAVVCFNSTYASQGLVASIKNFKSVLYYQNNVFMSEAKCWTETDLTKGPH
EFCSQHTMLVKQGDDYVYLPYPDPSRILGAGCFVDDIVKTDGTLMIERFVSLAIDAYPLTKHPNQEYADVFHLYLQYIRK
LHDELTGHMLDMYSVMLTNDNTSRYWEPEFYEAMYTPHTVLQAVGACVLCNSQTSLRCGACIRRPFLCCKCCYDHVISTS
HKLVLSVNPYVCNAPGCDVTDVTQLYLGGMSYYCKSHKPPISFPLCANGQVFGLYKNTCVGSDNVTDFNAIATCDWTNAG
DYILANTCTERLKLFAAETLKATEETFKLSYGIATVREVLSDRELHLSWEVGKPRPPLNRNYVFTGYRVTKNSKVQIGEY
TFEKGDYGDAVVYRGTTTYKLNVGDYFVLTSHTVMPLSAPTLVPQEHYVRITGLYPTLNISDEFSSNVANYQKVGMQKYS
TLQGPPGTGKSHFAIGLALYYPSARIVYTACSHAAVDALCEKALKYLPIDKCSRIIPARARVECFDKFKVNSTLEQYVFC
TVNALPETTADIVVFDEISMATNYDLSVVNARLRAKHYVYIGDPAQLPAPRTLLTKGTLEPEYFNSVCRLMKTIGPDMFL
GTCRRCPAEIVDTVSALVYDNKLKAHKDKSAQCFKMFYKGVITHDVSSAINRPQIGVVREFLTRNPAWRKAVFISPYNSQ
NAVASKILGLPTQTVDSSQGSEYDYVIFTQTTETAHSCNVNRFNVAITRAKVGILCIMSDRDLYDKLQFTSLEIPRRNVA
TLQAENVTGLFKDCSKVITGLHPTQAPTHLSVDTKFKTEGLCVDIPGIPKDMTYRRLISMMGFKMNYQVNGYPNMFITRE
EAIRHVRAWIGFDVEGCHATREAVGTNLPLQLGFSTGVNLVAVPTGYVDTPNNTDFSRVSAKPPPGDQFKHLIPLMYKGL
PWNVVRIKIVQMLSDTLKNLSDRVVFVLWAHGFELTSMKYFVKIGPERTCCLCDRRATCFSTASDTYACWHHSIGFDYVY
NPFMIDVQQWGFTGNLQSNHDLYCQVHGNAHVASCDAIMTRCLAVHECFVKRVDWTIEYPIIGDELKINAACRKVQHMVV
KAALLADKFPVLHDIGNPKAIKCVPQADVEWKFYDAQPCSDKAYKIEELFYSYATHSDKFTDGVCLFWNCNVDRYPANSI
VCRFDTRVLSNLNLPGCDGGSLYVNKHAFHTPAFDKSAFVNLKQLPFFYYSDSPCESHGKQVVSDIDYVPLKSATCITRC
NLGGAVCRHHANEYRLYLDAYNMMISAGFSLWVYKQFDTYNLWNTFTRLQSLENVAFNVVNKGHFDGQQGEVPVSIINNT
VYTKVDGVDVELFENKTTLPVNVAFELWAKRNIKPVPEVKILNNLGVDIAANTVIWDYKRDAPAHISTIGVCSMTDIAKK
PTETICAPLTVFFDGRVDGQVDLFRNARNGVLITEGSVKGLQPSVGPKQASLNGVTLIGEAVKTQFNYYKKVDGVVQQLP
ETYFTQSRNLQEFKPRSQMEIDFLELAMDEFIERYKLEGYAFEHIVYGDFSHSQLGGLHLLIGLAKRFKESPFELEDFIP
MDSTVKNYFITDAQTGSSKCVCSVIDLLLDDFVEIIKSQDLSVVSKVVKVTIDYTEISFMLWCKDGHVETFYPKLQSSQA
WQPGVAMPNLYKMQRMLLEKCDLQNYGDSATLPKGIMMNVAKYTQLCQYLNTLTLAVPYNMRVIHFGAGSDKGVAPGTAV
LRQWLPTGTLLVDSDLNDFVSDADSTLIGDCATVHTANKWDLIISDMYDPKTKNVTKENDSKEGFFTYICGFIQQKLALG
GSVAIKITEHSWNADLYKLMGHFAWWTAFVTNVNASSSEAFLIGCNYLGKPREQIDGYVMHANYIFWRNTNPIQLSSYSL
FDMSKFPLKLRGTAVMSLKEGQINDMILSLLSKGRLIIRENNRVVISSDVLVNN
It is 2701 amino acids long.
```

[ ]:

```

```

8 Extra credit (2 points, maybe): What is the expected length of the longest open reading frame?

Given that we have ~30,000 nucleotides of a given sequence composition, how long would we expect the longest reading frame to be if they were ordered randomly?

Do this by randomly shuffling the genome sequence some number of times (say 100) and finding the longest orf in all its translations. Add those lengths up and divide by the number of times you shuffled.

[29]:

```
# random is a very useful python library aka module
import random

numlist = [1,2,3,4,5,6]

# we can call functions from within modules by using this syntax: module.
 ↪function
random.shuffle(numlist)

print('shuffled numlist:', numlist)

# random.shuffle doesn't work on immutable objects like strings, but we can␣
 ↪turn a string into a list...
seq = 'ACGTACGTACGT'

seqlist = list(seq)

random.shuffle(seqlist)

seq = ''.join(seqlist)

print('shuffled seq:', seq)
```

```
shuffled numlist: [6, 5, 3, 1, 4, 2]
shuffled seq: CACATGTGCGAT
```

[30]:

```
import random

num_shuffles = 100

# also create a variable to hold the total length of all the longest orfs...
total_length = 0

for repeat in range(num_shuffles):
    # shuffle the protien sequence by turning it into a list, shuffling, and␣
↪then converting back to a string
    list_prot_seq = list(prot_seq)
    random.shuffle(list_prot_seq)
    string_prot_seq = ' '.join(list_prot_seq)

    # find the longest open reading frame
    split_seq = string_prot_seq.split('*')
    long_short = sorted(split_seq, key=len, reverse=True)
    longest_orf = long_short[0]

    # add the length of the longest orf to total_length
    length = len(longest_orf)
    total_length += length

# divide total_length by the number of shuffles to get the average
mean_longest_orf = total_length/num_shuffles
print(mean_longest_orf)
```

```
183.46
```

9 Potentially helpful examples

9.1 More details on defining functions

Defining a function that takes in an argument (in this case, a number) and returns a new value that depends on the argument.

[31]:
```
# here we define a function that doubles a number
#
# Here the function is called "double" and the argument that gets passed in is
# called "num". We could pick a different name for "num", as long as we use
# that other name inside the body of the function (the body of the function is
# the indendted part that comes after the "def ...:" line)
def double(num):
    return 2*num

print('2 doubled is', double(2))
print('3 doubled is', double(3))

a = 6
print(a, 'doubled is', double(a))
```

```
2 doubled is 4
3 doubled is 6
6 doubled is 12
```

[32]:
```
# this gives exactly the same function: it doesn't matter what we call the
# argument (num or i) as long as we are consistent.
def double(i):
    return 2*i
```

9.2 Hints for translating DNA to protein

[33]:
```
# let's translate a short dna sequence by hand
dna_seq = 'CCTCATATT'

translation = '' # start with an empty string

first_codon = dna_seq[0:3]
first_amino_acid = genetic_code[first_codon]
translation = translation + first_amino_acid # add the first aa to the end

second_codon = dna_seq[3:6]
second_amino_acid = genetic_code[second_codon]
translation = translation + second_amino_acid # add the second aa to the end

third_codon = dna_seq[6:9]
third_amino_acid = genetic_code[third_codon]
translation = translation + third_amino_acid # add the third aa to the end

print(dna_seq, 'codes for', translation)
```

```
CCTCATATT codes for PHI
```

[ ]:
```

```

[ ]:
```

```