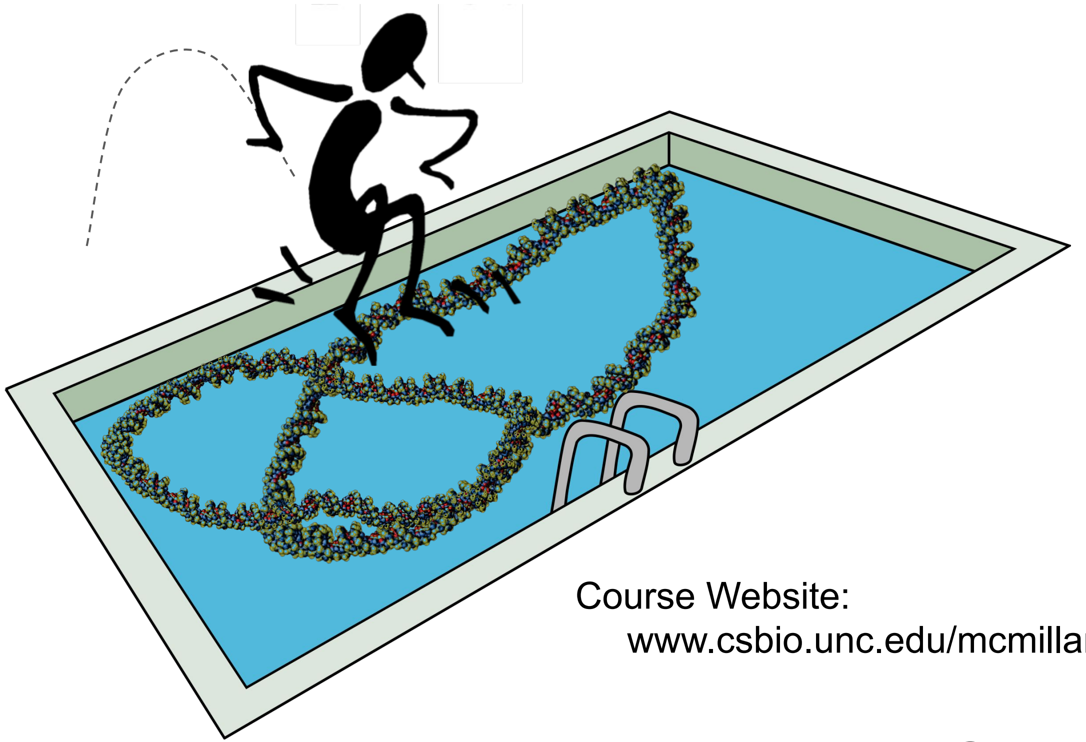
Comp 555 - BioAlgorithms - Spring 2021

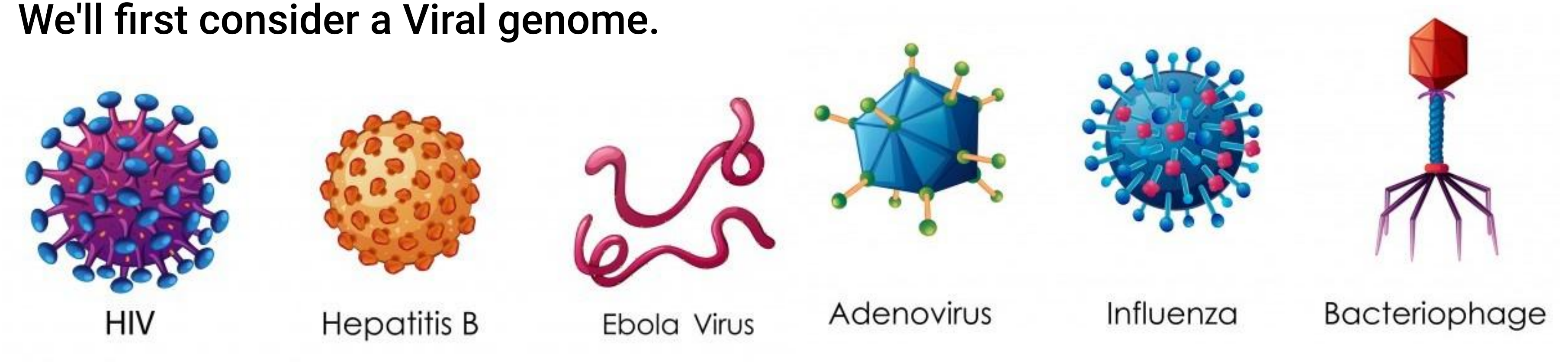


Course Website:
www.csbio.unc.edu/mcmillan/index.py?run=Courses.Comp555S21

Jumping into Genomes

A simple genome

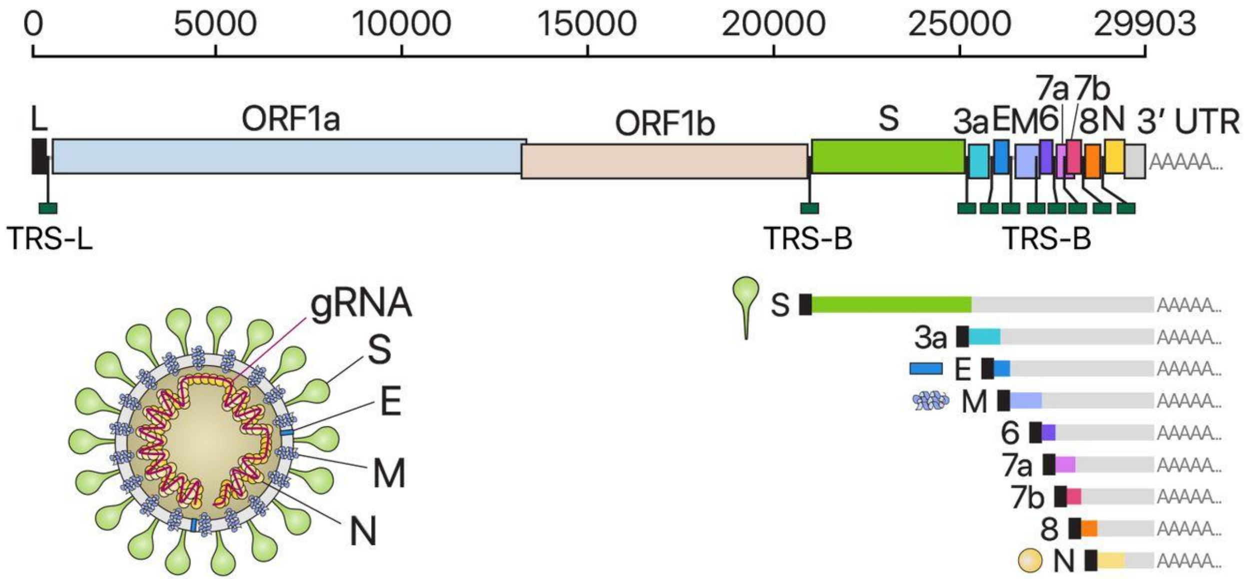
We'll first consider a Viral genome.



Characteristics of Viral genomes:

- Small, dense, and tricky
- Viral genomes code for functional proteins in order to "live", but rely on a host's machinery to perform essential functions
- Small genomes (3K - 30K bases) with a few "key" genes

Today's Virus

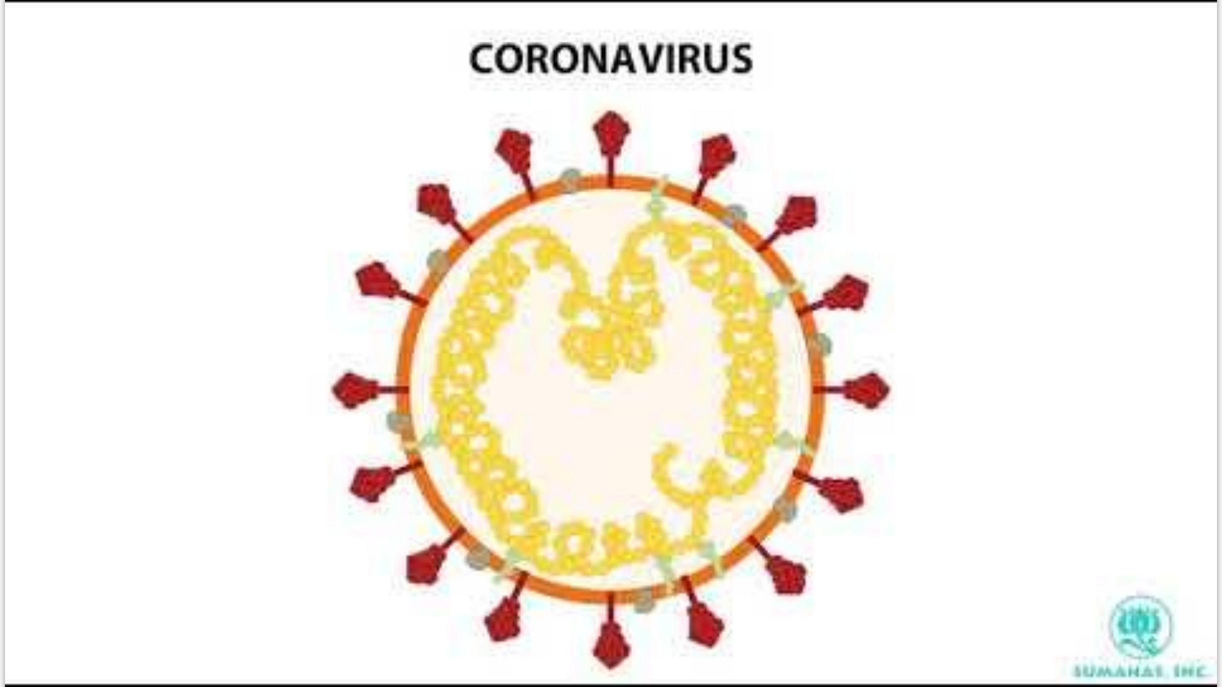


SARS-CoV-2, the virus that causes COVID-19

- 29903 bases of the original Wuhan isolate
- 10 (11?) genes, 4 structural, 2 with primary functions

How viral life works

Youtube: https://www.youtube.com/watch?v=Xv3TxtFtCNE

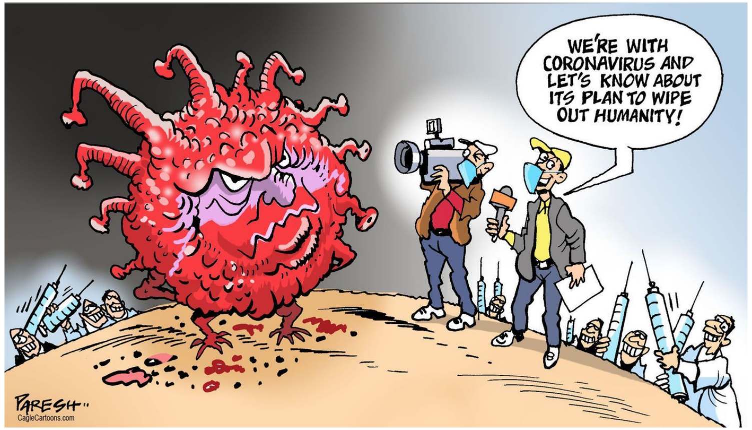


Time to get serious

- **By next Tuesday's class meeting everyone should set up a Jupyter Notebook environment**
- **Recommend using Anaconda**

  https://www.anaconda.com/products/individual
  - Includes an isolated environment, an IDE, common packages, and a package manager

- **Will need it for problem sets and exams**
- **Next Wednesday's office hours will focus on helping folks install Jupyter**
- **COMP555 accounts should be up by next Tuesday**
- **We'll start using Python and Jupyter today.**
  - You should go back through today's Notebook to verify your setup


WE'RE WITH CORONAVIRUS AND LET'S KNOW ABOUT ITS PLAN TO WIPE OUT HUMANITY!
PARESH
CagleCartoons.com

Let's look at it

FASTA is a common format for biological sequences

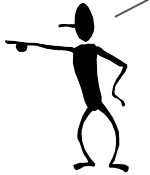
- **Each sequence is preceded by a header line that starts with ‘>’**
- **Followed by multiple lines of sequence data from a standard alphabet**
  - For DNA, alphabet = “ACGT”
  - For RNA, alphabet = “ACGU”
  - For Proteins, alphabet = “ACDEFGHIKLMNOPQRSTUVWY”
- **A sequence ends when either another header line is reached or the end-of-file**
- **Multiple sequences per file are allowed**
- **Sequences are 1-indexed rather than 0-indexed!**

An Example

```
In [123]: !head data/SARS-CoV-2.fa

>NC_045512.2 |Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome
ATTAAAGGTTTATACCTTCCCAGGTAACAAACCAACCAACTTTCGATCTCTTGTAGATCT
GTTCTCTAAACGAACTTTAAAATCTGTGTGGCTGTCACTCGGCTGCATGCTTAGTGCACT
CACGCAGTATAATTAATAACTAATTACTGTCGTTGACAGGACACGAGTAACTCGTCTATC
TTCTGCAGGCTGCTTACGGTTTCGTCCGTGTTGCAGCCGATCATCAGCACATCTAGGTTT
CGTCCGGGTGTGACCGAAAGGTAAGATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAAAC
ACACGTCCAACTCAGTTTGCCTGTTTTACAGGTTCGCGACGTGCTCGTACGTGGCTTTGG
AGACTCCGTGGAGGAGGTCTTATCAGAGGCACGTCAACATCTTAAAGATGGCACTTGTGG
CTTAGTAGAAGTTGAAAAAGGCGTTTTGCCTCAACTTGAACAGCCCTATGTGTTCATCAA
ACGTTCGGATGCTCGAACTGCACCTCATGGTCATGTTATGGTTGAGCTGGTAGCAGAACT
```

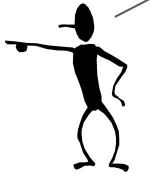
"head", by default prints the first 10 lines of a file



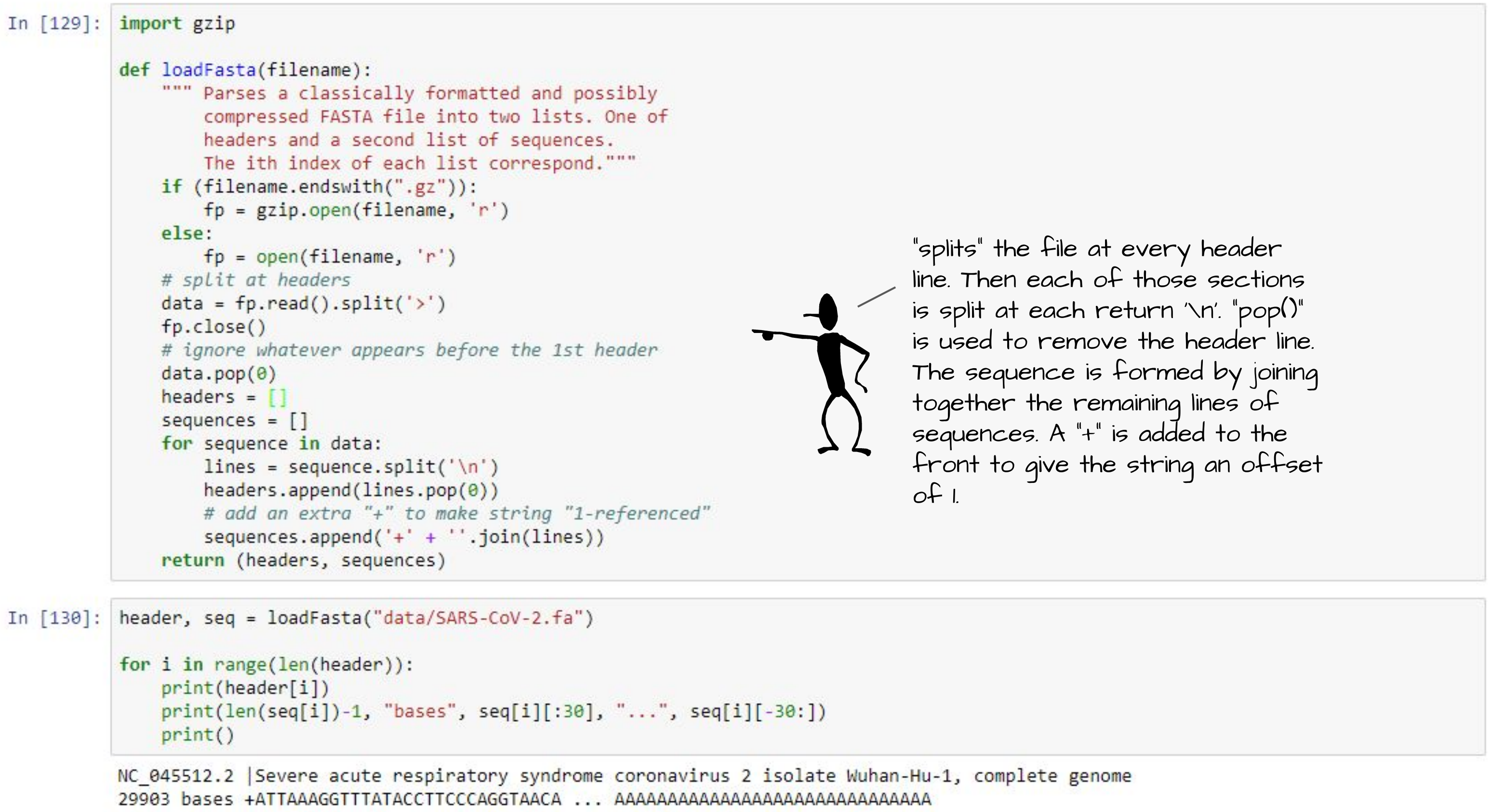
```
In [125]: !tail data/SARS-CoV-2.fa

TATTGACGCATACAAAACATTCCCACCAACAGAGCCTAAAAAGGACAAAAAGAAGAAGGC
TGATGAAACTCAAGCCTTACCGCAGAGACAGAAGAAACAGCAAACTGTGACTCTTCTTCC
TGCTGCAGATTTGGATGATTTCTCCAAACAATTGCAACAATCCATGAGCAGTGCTGACTC
AACTCAGGCCTAAACTCATGCAGACCACACAAGGCAGATGGGCTATATAAACGTTTTCGC
TTTTCCGTTTACGATATATAGTCTACTCTTGTGCAGAATGAATTCTCGTAACTACATAGC
ACAAGTAGATGTAGTTAACTTTAATCTCACATAGCAATCTTTAATCAGTGTGTAACATTA
GGGAGGACTTGAAAGAGCCACCACATTTTCACCGAGGCCACGCGGAGTACGATCGAGTGT
ACAGTGAACAATGCTAGGGAGAGCTGCCTATATGGAAGAGCCCTAATGTGTAAAATTAAT
TTTAGTAGTGCTATCCCCATGTGATTTTAATAGCTTCTTAGGAGAATGACAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAA
```

"tail", prints the last 10 lines



A little code for reading FASTA



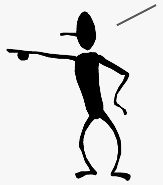
```
In [129]: import gzip

          def loadFasta(filename):
              """ Parses a classically formatted and possibly
                  compressed FASTA file into two lists. One of
                  headers and a second list of sequences.
                  The ith index of each list correspond."""
              if (filename.endswith(".gz")):
                  fp = gzip.open(filename, 'r')
              else:
                  fp = open(filename, 'r')
              # split at headers
              data = fp.read().split('>')
              fp.close()
              # ignore whatever appears before the 1st header
              data.pop(0)
              headers = []
              sequences = []
              for sequence in data:
                  lines = sequence.split('\n')
                  headers.append(lines.pop(0))
                  # add an extra "+" to make string "1-referenced"
                  sequences.append('+' + ''.join(lines))
              return (headers, sequences)
```

```
In [130]: header, seq = loadFasta("data/SARS-CoV-2.fa")

          for i in range(len(header)):
              print(header[i])
              print(len(seq[i])-1, "bases", seq[i][:30], "...", seq[i][-30:])
              print()
```

```
NC_045512.2 |Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1, complete genome
29903 bases +ATTAAAGGTTTATACCTTCCCAGGTAACA ... AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



"splits" the file at every header line. Then each of those sections is split at each return '\n'. "pop()" is used to remove the header line. The sequence is formed by joining together the remaining lines of sequences. A "+" is added to the front to give the string an offset of 1.

Let’s take a minute to explore

Genome sequences are best understood by examining subsequences

Often we examine all subsequences of length *k*, called *k-mers*.

The statistics and patterns of k-mers can shed light on a genome’s organization and local function.

Two simple rules to consider:

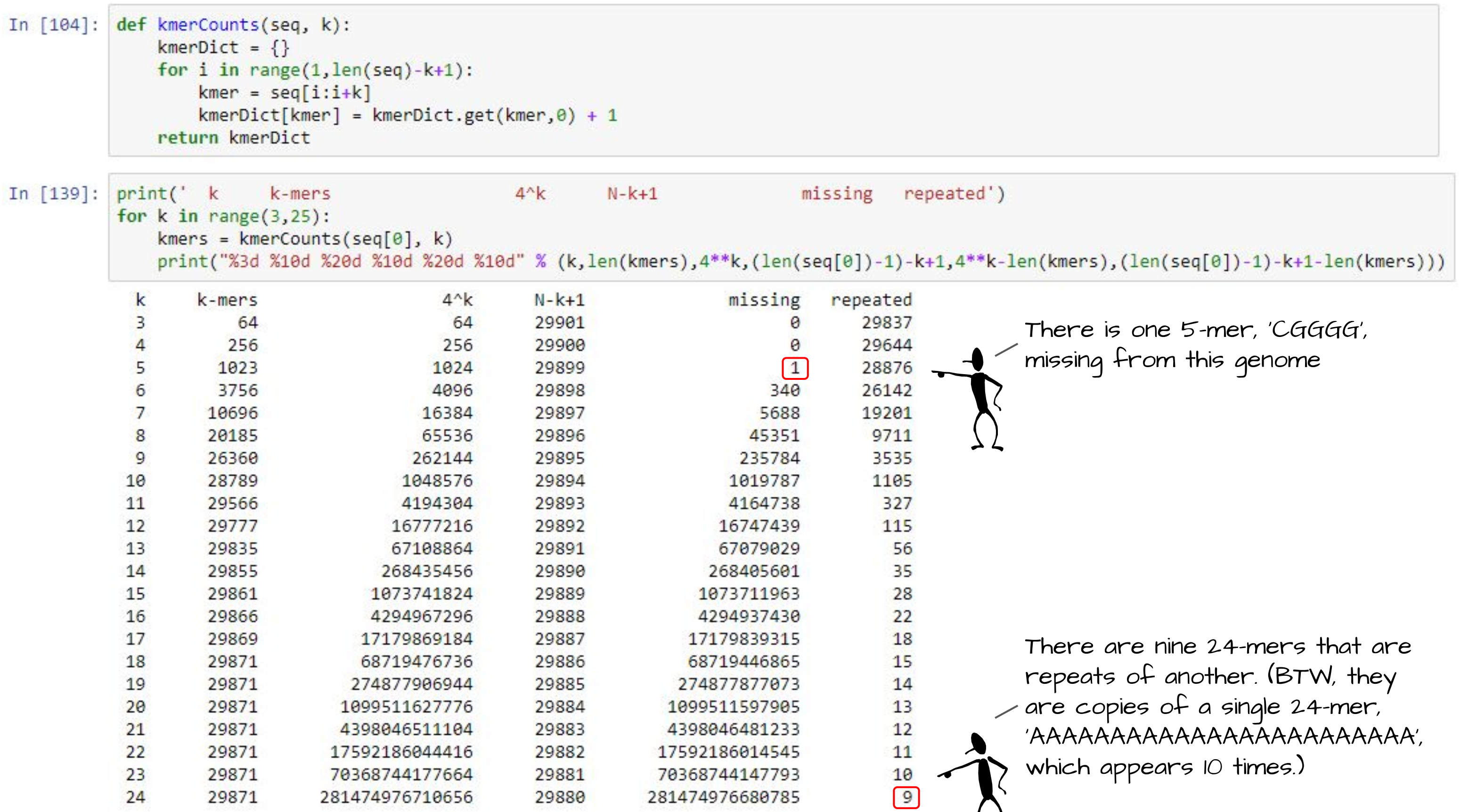
1) **There are 4^k possible DNA k-mers**
2) **A linear sequence of length N has *N - k + 1* k-mers**

```
ATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAAACA
ATGGAG  CCTTGT  CTGGTT  AACGAG
 TGGAGA  CTTGTC  TGGTTT  ACGAGA
  GGAGAG  TTGTCC  GGTTTC  CGAGAA
   GAGAGC  TGTCCC  GTTTCA  GAGAAA
    AGAGCC  GTCCCT  TTTCAA  AGAAAA
     GAGCCT  TCCCTG  TTCAAC  GAAAAC
      AGCCTT  CCCTGG  TCAACG  AAAACA
       GCCTTG  CCTGGT  CAACGA
```

A 36 base sequence has 31, 6-mers

$(36 - 6 + 1) = 31$

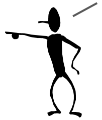
Genome "k-mer" statistics



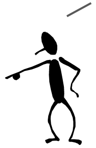
```
In [104]: def kmerCounts(seq, k):
              kmerDict = {}
              for i in range(1,len(seq)-k+1):
                  kmer = seq[i:i+k]
                  kmerDict[kmer] = kmerDict.get(kmer,0) + 1
              return kmerDict
```

```
In [139]: print('  k     k-mers                   4^k      N-k+1                missing   repeated')
          for k in range(3,25):
              kmers = kmerCounts(seq[0], k)
              print("%3d %10d %20d %10d %20d %10d" % (k,len(kmers),4**k,(len(seq[0])-1)-k+1,4**k-len(kmers),(len(seq[0])-1)-k+1-len(kmers)))
```

```
  k     k-mers                  4^k      N-k+1              missing   repeated
  3         64                   64      29901                    0      29837
  4        256                  256      29900                    0      29644
  5       1023                 1024      29899                    1      28876
  6       3756                 4096      29898                  340      26142
  7      10696                16384      29897                 5688      19201
  8      20185                65536      29896                45351       9711
  9      26360               262144      29895               235784       3535
 10      28789              1048576      29894              1019787       1105
 11      29566              4194304      29893              4164738        327
 12      29777             16777216      29892             16747439        115
 13      29835             67108864      29891             67079029         56
 14      29855            268435456      29890            268405601         35
 15      29861           1073741824      29889           1073711963         28
 16      29866           4294967296      29888           4294937430         22
 17      29869          17179869184      29887          17179839315         18
 18      29871          68719476736      29886          68719446865         15
 19      29871         274877906944      29885         274877877073         14
 20      29871        1099511627776      29884        1099511597905         13
 21      29871        4398046511104      29883        4398046481233         12
 22      29871       17592186044416      29882       17592186014545         11
 23      29871       70368744177664      29881       70368744147793         10
 24      29871      281474976710656      29880      281474976680785          9
```

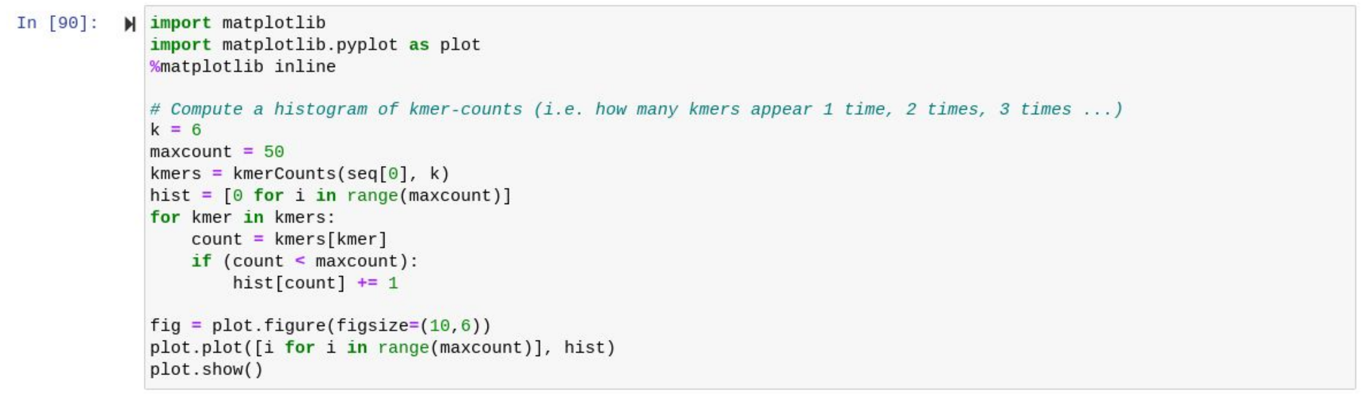


There is one 5-mer, 'CGGGG', missing from this genome



There are nine 24-mers that are repeats of another. (BTW, they are copies of a single 24-mer, 'AAAAAAAAAAAAAAAAAAAAAAAA', which appears 10 times.)

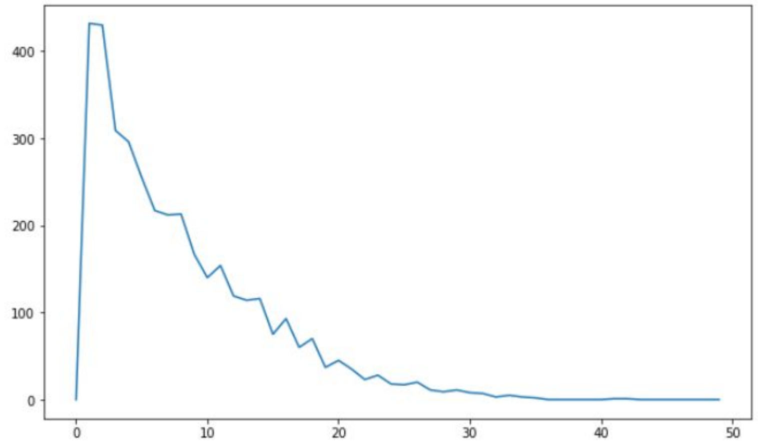
What do k-mer statistics look like?



```
In [90]:  import matplotlib
          import matplotlib.pyplot as plot
          %matplotlib inline

          # Compute a histogram of kmer-counts (i.e. how many kmers appear 1 time, 2 times, 3 times ...)
          k = 6
          maxcount = 50
          kmers = kmerCounts(seq[0], k)
          hist = [0 for i in range(maxcount)]
          for kmer in kmers:
              count = kmers[kmer]
              if (count < maxcount):
                  hist[count] += 1

          fig = plot.figure(figsize=(10,6))
          plot.plot([i for i in range(maxcount)], hist)
          plot.show()
```

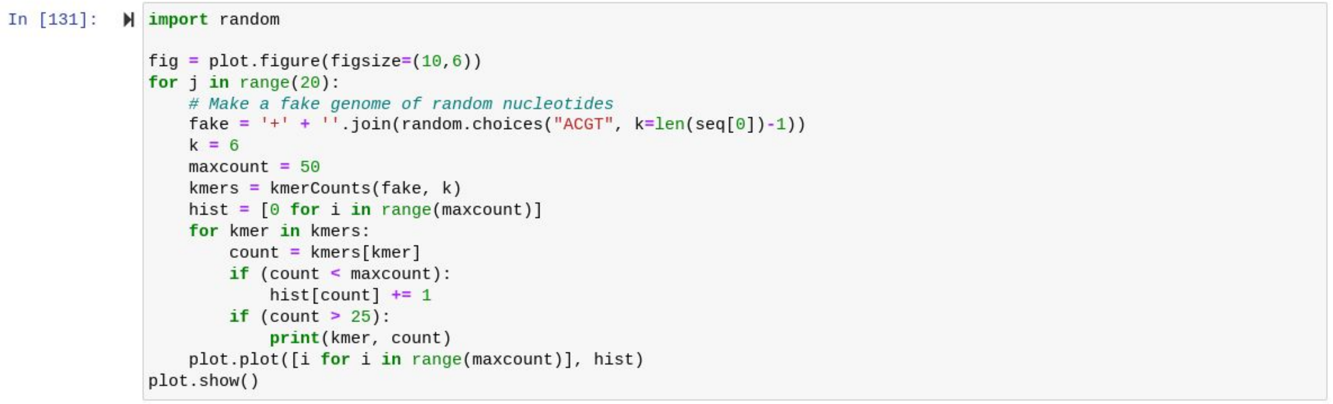


Okay, there are 432 6-mers that appear only once, 430 that are repeated twice, and the fewer and fewer are repeated 3, 4, 5, and so on.

Meanwhile there are two 6-mers that are repeated more than 40 times ("TTGTTA" 42 times, and "TGTTAA" 41 times)

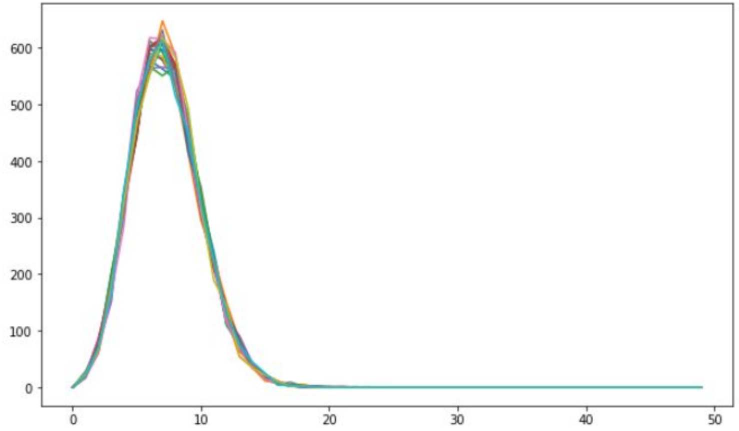
But are these counts typical?

How does it compare to a random sequence?



```
In [131]:   import random

            fig = plot.figure(figsize=(10,6))
            for j in range(20):
                # Make a fake genome of random nucleotides
                fake = '+' + ''.join(random.choices("ACGT", k=len(seq[0])-1))
                k = 6
                maxcount = 50
                kmers = kmerCounts(fake, k)
                hist = [0 for i in range(maxcount)]
                for kmer in kmers:
                    count = kmers[kmer]
                    if (count < maxcount):
                        hist[count] += 1
                    if (count > 25):
                        print(kmer, count)
                plot.plot([i for i in range(maxcount)], hist)
            plot.show()
```



In a random sequence of the same length as SARS-CoV-2, there would be far fewer unique 6-mers (typically around 20). Also, most 6-mers would appear approximately 7 times (roughly 29903/4096 = 7.3 times).

Also it would be rare for any 6-mer to be repeated more than 25 times.

Conclusion… virus sequences aren't random patterns

.

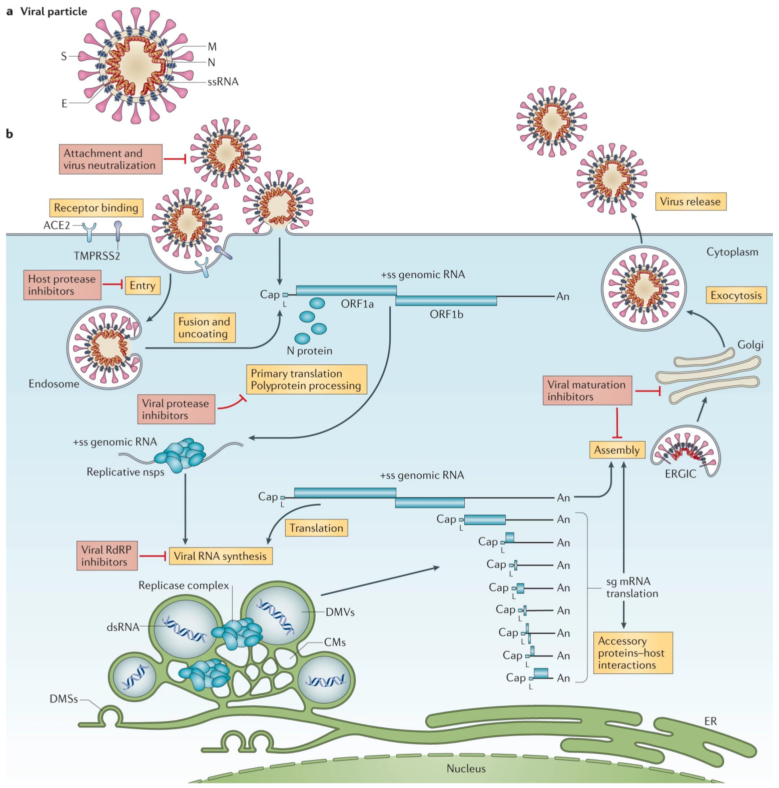
Let's look at some key genes

The "Spikes" of the viral envelope seek out the ACE2 recptors in order to infect a cell.

Eventually, an immune response is set off.

B-cells use knowlege (acquired from T-cells) about the Spike sequence to generate antibodies that target the virus to inactive it.

The key point is learning to recognize the spike sequence.



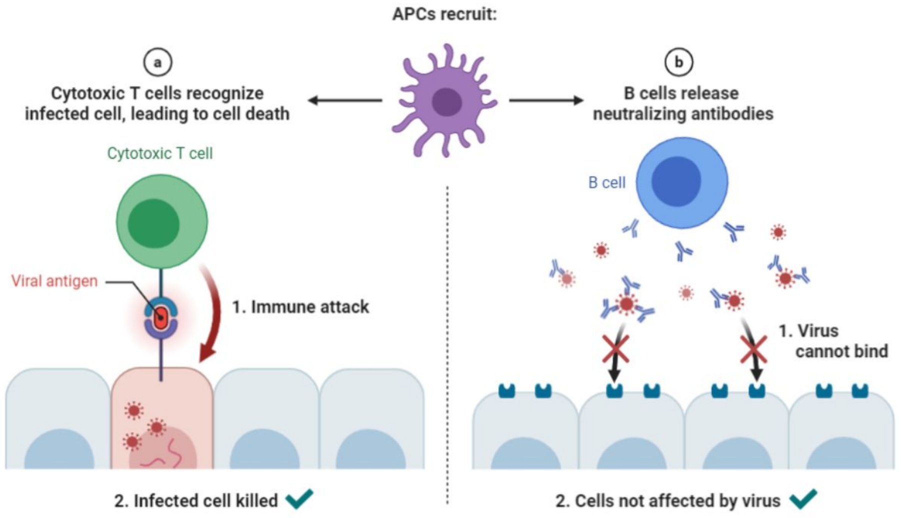
Let's look at some key genes

The "Spikes" of the viral envelope seek out the ACE2 recptors in order to infect a cell.

Eventually, an immune response is set off.

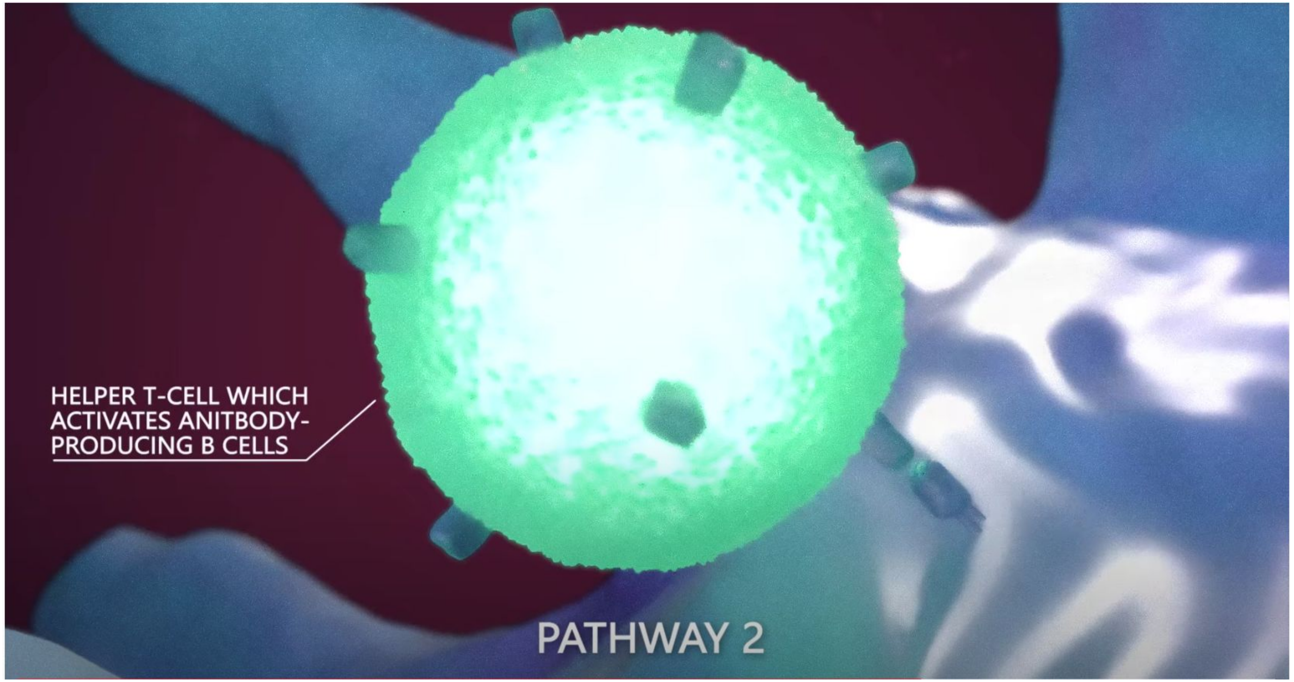
B-cells use knowlege (acquired from T-cells) about the Spike sequence to generate antibodies that target the virus to inactive it.

The key point is learning to recognize the spike sequence.


APCs recruit:
a
Cytotoxic T cells recognize infected cell, leading to cell death
b
B cells release neutralizing antibodies
Cytotoxic T cell
B cell
Viral antigen
1. Immune attack
1. Virus cannot bind
2. Infected cell killed ✓
2. Cells not affected by virus ✓

How an mRNA vaccine works

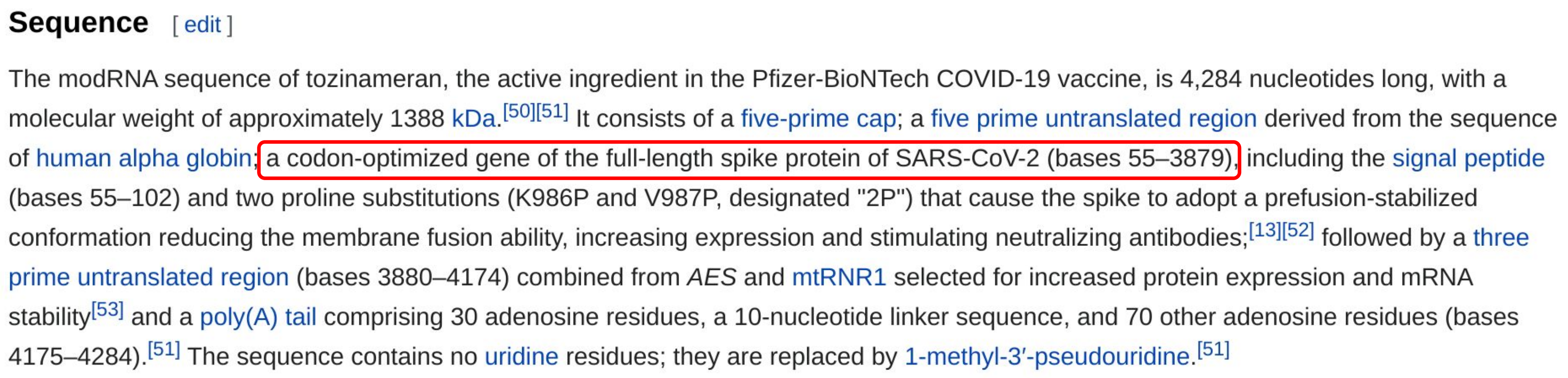
https://www.youtube.com/watch?v=LcTEmHlvY10



How a vaccine works

It we introduce a proxy that "looks" sufficently like the Spike, then we can set off the immune reaction, without having to go through the infection.

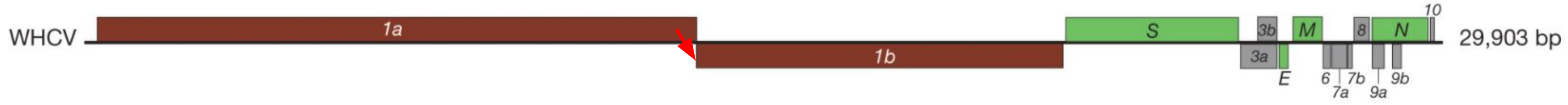
From "Pfizer-BioNTech COVID-19 vaccine" wikipedia page:

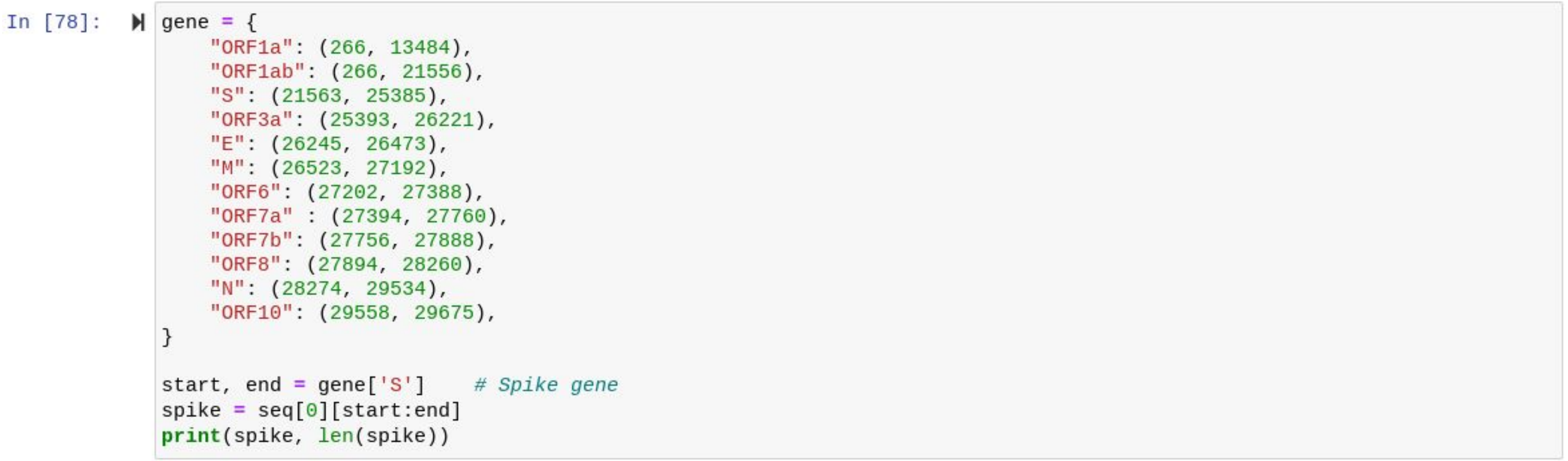


Sequence [edit]

The modRNA sequence of tozinameran, the active ingredient in the Pfizer-BioNTech COVID-19 vaccine, is 4,284 nucleotides long, with a molecular weight of approximately 1388 kDa.[50][51] It consists of a five-prime cap; a five prime untranslated region derived from the sequence of human alpha globin; a codon-optimized gene of the full-length spike protein of SARS-CoV-2 (bases 55–3879), including the signal peptide (bases 55–102) and two proline substitutions (K986P and V987P, designated "2P") that cause the spike to adopt a prefusion-stabilized conformation reducing the membrane fusion ability, increasing expression and stimulating neutralizing antibodies;[13][52] followed by a three prime untranslated region (bases 3880–4174) combined from *AES* and mtRNR1 selected for increased protein expression and mRNA stability[53] and a poly(A) tail comprising 30 adenosine residues, a 10-nucleotide linker sequence, and 70 other adenosine residues (bases 4175–4284).[51] The sequence contains no uridine residues; they are replaced by 1-methyl-3′-pseudouridine.[51]

A look at the Spike, 'S', gene sequence



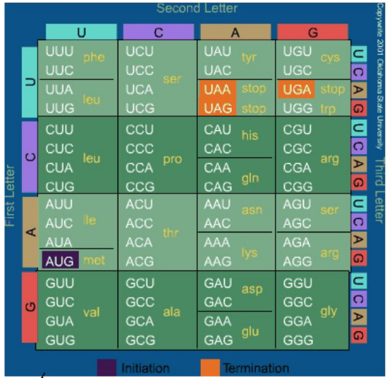


```
In [78]:  gene = {
              "ORF1a": (266, 13484),
              "ORF1ab": (266, 21556),
              "S": (21563, 25385),
              "ORF3a": (25393, 26221),
              "E": (26245, 26473),
              "M": (26523, 27192),
              "ORF6": (27202, 27388),
              "ORF7a" : (27394, 27760),
              "ORF7b": (27756, 27888),
              "ORF8": (27894, 28260),
              "N": (28274, 29534),
              "ORF10": (29558, 29675),
          }

          start, end = gene['S']     # Spike gene
          spike = seq[0][start:end]
          print(spike, len(spike))
```

A look at the Spike, 'S', gene sequence

```
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCTGCATACACTAATTCTTTCACACGTGGTGTTTATTACC
CTGACAAAGTTTTCAGATCCTCAGTTTTACATTCAACTCAGGACTTGTTCTTACCTTTCTTTTCCAATGTTACTTGGTTCCATGCTATACATGTCTCTGGGACCAATGGTACTAA
GAGGTTTGATAACCCTGTCCTACCATTTAATGATGGTGTTTATTTTGCTTCCACTGAGAAGTCTAACATAATAAGAGGCTGGATTTTTGGTACTACTTTAGATTCGAAGACCCAG
TCCCTACTTATTGTTAATAACGCTACTAATGTTGTTATTAAAGTCTGTGAATTTCAATTTTGTAATGATCCATTTTTGGGTGTTTATTACCACAAAAACAACAAAAGTTGGATGG
AAAGTGAGTTCAGAGTTTATTCTAGTGCGAATAATTGCACTTTTGAATATGTCTCTCAGCCTTTTCTTATGGACCTTGAAGGAAAACAGGGTAATTTCAAAAATCTTAGGGAATT
TGTGTTTAAGAATATTGATGGTTATTTTAAAATATATTCTAAGCACACGCCTATTAATTTAGTGCGTGATCTCCCTCAGGGTTTTTCGGCTTTAGAACCATTGGTAGATTTGCCA
ATAGGTATTAACATCACTAGGTTTCAAACTTTACTTGCTTTACATAGAAGTTATTTGACTCCTGGTGATTCTTCTTCAGGTTGGACAGCTGGTGCTGCAGCTTATTATGTGGGTT
ATCTTCAACCTAGGACTTTTCTATTAAAATATAATGAAAATGGAACCATTACAGATGCTGTAGACTGTGCACTTGACCCTCTCTCAGAAACAAAGTGTACGTTGAAATCCTTCAC
TGTAGAAAAAGGAATCTATCAAACTTCTAACTTTAGAGTCCAACCAACAGAATCTATTGTTAGATTTCCTAATATTACAAACTTGTGCCCTTTTGGTGAAGTTTTTAACGCCACC
AGATTTGCATCTGTTTATGCTTGGAACAGGAAGAGAATCAGCAACTGTGTTGCTGATTATTCTGTCCTATATAATTCCGCATCATTTTCCACTTTTAAGTGTTATGGAGTGTCTC
CTACTAAATTAAATGATCTCTGCTTTACTAATGTCTATGCAGATTCATTTGTAATTAGAGGTGATGAAGTCAGACAAATCGCTCCAGGGCAAACTGGAAAGATTGCTGATTATAA
TTATAAATTACCAGATGATTTTACAGGCTGCGTTATAGCTTGGAATTCTAACAATCTTGATTCTAAGGTTGGTGGTAATTATAATTACCTGTATAGATTGTTTAGGAAGTCTAAT
CTCAAACCTTTTGAGAGAGATATTTCAACTGAAATCTATCAGGCCGGTAGCACACCTTGTAATGGTGTTGAAGGTTTTAATTGTTACTTTCCTTTACAATCATATGGTTTCCAAC
CCACTAATGGTGTTGGTTACCAACCATACAGAGTAGTAGTACTTTCTTTTGAACTTCTACATGCACCAGCAACTGTTTGTGGACCTAAAAAGTCTACTAATTTGGTTAAAAACAA
ATGTGTCAATTTCAACTTCAATGGTTTAACAGGCACAGGTGTTCTTACTGAGTCTAACAAAAAGTTTCTGCCTTTCCAACAATTTGGCAGAGACATTGCTGACACTACTGATGCT
GTCCGTGATCCACAGACACTTGAGATTCTTGACATTACACCATGTTCTTTTGGTGGTGTCAGTGTTATAACACCAGGAACAAATACTTCTAACCAGGTTGCTGTTCTTTATCAGG
ATGTTAACTGCACAGAAGTCCCTGTTGCTATTCATGCAGATCAACTTACTCCTACTTGGCGTGTTTATTCTACAGGTTCTAATGTTTTTCAAACACGTGCAGGCTGTTTAATAGG
GGCTGAACATGTCAACAACTCATATGAGTGTGACATACCCATTGGTGCAGGTATATGCGCTAGTTATCAGACTCAGACTAATTCTCCTCGGCGGGCACGTAGTGTAGCTAGTCAA
TCCATCATTGCCTACACTATGTCACTTGGTGCAGAAAATTCAGTTGCTTACTCTAATAACTCTATTGCCATACCCACAAATTTTACTATTAGTGTTACCACAGAAATTCTACCAG
TGTCTATGACCAAGACATCAGTAGATTGTACAATGTACATTTGTGGTGATTCAACTGAATGCAGCAATCTTTTGTTGCAATATGGCAGTTTTTGTACACAATTAAACCGTGCTTT
AACTGGAATAGCTGTTGAACAAGACAAAAACACCCAAGAAGTTTTTGCACAAGTCAAACAAATTTACAAAACACCACCAATTAAAGATTTTGGTGGTTTTAATTTTTCACAAATA
TTACCAGATCCATCAAAACCAAGCAAGAGGTCATTTATTGAAGATCTACTTTTCAACAAAGTGACACTTGCAGATGCTGGCTTCATCAAACAATATGGTGATTGCCTTGGTGATA
TTGCTGCTAGAGACCTCATTTGTGCACAAAAGTTTAACGGCCTTACTGTTTTGCCACCTTTGCTCACAGATGAAATGATTGCTCAATACACTTCTGCACTGTTAGCGGGTACAAT
CACTTCTGGTTGGACCTTTGGTGCAGGTGCTGCATTACAAATACCATTTGCTATGCAAATGGCTTATAGGTTTAATGGTATTGGAGTTACACAGAATGTTCTCTATGAGAACCAA
AAATTGATTGCCAACCAATTTAATAGTGCTATTGGCAAAATTCAAGACTCACTTTCTTCCACAGCAAGTGCACTTGGAAAACTTCAAGATGTGGTCAACCAAAATGCACAAGCTT
TAAACACGCTTGTTAAACAACTTAGCTCCAATTTTGGTGCAATTTCAAGTGTTTTAAATGATATCCTTTCACGTCTTGACAAAGTTGAGGCTGAAGTGCAAATTGATAGGTTGAT
CACAGGCAGACTTCAAAGTTTGCAGACATATGTGACTCAACAATTAATTAGAGCTGCAGAAATCAGAGCTTCTGCTAATCTTGCTGCTACTAAAATGTCAGAGTGTGTACTTGGA
CAATCAAAAAGAGTTGATTTTTGTGGAAAGGGCTATCATCTTATGTCCTTCCCTCAGTCAGCACCTCATGGTGTAGTCTTCTTGCATGTGACTTATGTCCCTGCACAAGAAAAGA
ACTTCACAACTGCTCCTGCCATTTGTCATGATGGAAAAGCACACTTTCCTCGTGAAGGTGTCTTTGTTTCAAATGGCACACACTGGTTTGTAACACAAAGGAATTTTTATGAACC
ACAAATCATTACTACAGACAACACATTTGTGTCTGGTAACTGTGATGTTGTAATAGGAATTGTCAACAACACAGTTTATGATCCTTTGCAACCTGAATTAGACTCATTCAAGGAG
GAGTTAGATAAATATTTTAAGAATCATACATCACCAGATGTTGATTTAGGTGACATCTCTGGCATTAATGCTTCAGTTGTAAACATTCAAAAAGAAATTGACCGCCTCAATGAGG
TTGCCAAGAATTTAAATGAATCTCTCATCGATCTCCAAGAACTTGGAAAGTATGAGCAGTATATAAAATGGCCATGGTACATTTGGCTAGGTTTTATAGCTGGCTTGATTGCCAT
AGTAATGGTGACAATTATGCTTTGCTGTATGACCAGTTGCTGTAGTTGTCTCAAGGGCTGTTGTTCTTGTGGATCCTGCTGCAAATTTGATGAAGACGACTCTGAGCCAGTGCTC
AAAGGAGTCAAATTACATTACACATAA  3822
```



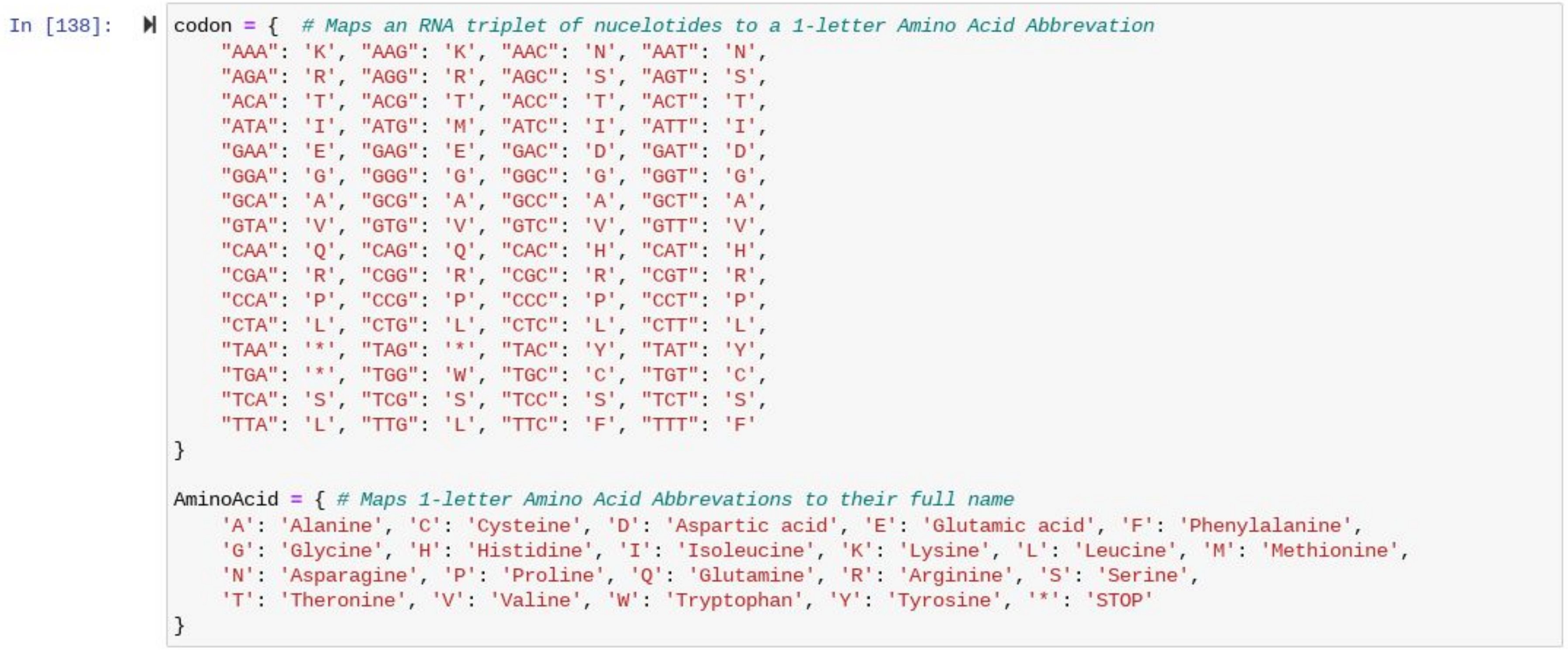
First Letter	U	C	A	G	Third Letter
U	UUU phe	UCU	UAU tyr	UGU cys	U
	UUC	UCC ser	UAC	UGC	C
	UUA leu	UCA	UAA stop	UGA stop	A
	UUG	UCG	UAG stop	UGG trp	G
C	CUU	CCU	CAU his	CGU	U
	CUC leu	CCC pro	CAC	CGC	C
	CUA	CCA	CAA	CGA arg	A
	CUG	CCG	CAG gln	CGG	G
A	AUU	ACU	AAU asn	AGU ser	U
	AUC ile	ACC	AAC	AGC	C
	AUA	ACA thr	AAA	AGA	A
	AUG met	ACG	AAG lys	AGG arg	G
G	GUU	GCU	GAU asp	GGU	U
	GUC	GCC	GAC	GGC gly	C
	GUA val	GCA ala	GAA	GGA	A
	GUG	GCG	GAG glu	GGG	G

🟪 Initiation 🟧 Termination

Why are there Us in this table?

Before a DNA sequence is translated into a protein, a copy is first made. This copy is made from RNA. In RNA, the nucleotide "Uracil" replaces "Thymine". Uracil and Thymine are both chemically and structurally very similar.

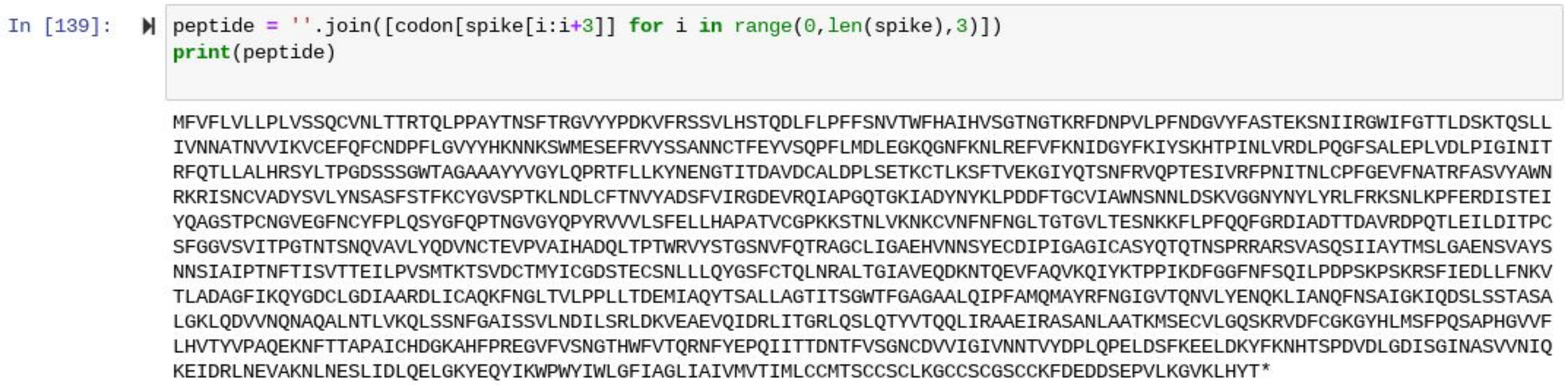
Maping to Amino Acid Residues



```
In [138]:  codon = {  # Maps an RNA triplet of nucelotides to a 1-letter Amino Acid Abbrevation
               "AAA": 'K', "AAG": 'K', "AAC": 'N', "AAT": 'N',
               "AGA": 'R', "AGG": 'R', "AGC": 'S', "AGT": 'S',
               "ACA": 'T', "ACG": 'T', "ACC": 'T', "ACT": 'T',
               "ATA": 'I', "ATG": 'M', "ATC": 'I', "ATT": 'I',
               "GAA": 'E', "GAG": 'E', "GAC": 'D', "GAT": 'D',
               "GGA": 'G', "GGG": 'G', "GGC": 'G', "GGT": 'G',
               "GCA": 'A', "GCG": 'A', "GCC": 'A', "GCT": 'A',
               "GTA": 'V', "GTG": 'V', "GTC": 'V', "GTT": 'V',
               "CAA": 'Q', "CAG": 'Q', "CAC": 'H', "CAT": 'H',
               "CGA": 'R', "CGG": 'R', "CGC": 'R', "CGT": 'R',
               "CCA": 'P', "CCG": 'P', "CCC": 'P', "CCT": 'P',
               "CTA": 'L', "CTG": 'L', "CTC": 'L', "CTT": 'L',
               "TAA": '*', "TAG": '*', "TAC": 'Y', "TAT": 'Y',
               "TGA": '*', "TGG": 'W', "TGC": 'C', "TGT": 'C',
               "TCA": 'S', "TCG": 'S', "TCC": 'S', "TCT": 'S',
               "TTA": 'L', "TTG": 'L', "TTC": 'F', "TTT": 'F'
           }

           AminoAcid = { # Maps 1-letter Amino Acid Abbrevations to their full name
               'A': 'Alanine', 'C': 'Cysteine', 'D': 'Aspartic acid', 'E': 'Glutamic acid', 'F': 'Phenylalanine',
               'G': 'Glycine', 'H': 'Histidine', 'I': 'Isoleucine', 'K': 'Lysine', 'L': 'Leucine', 'M': 'Methionine',
               'N': 'Asparagine', 'P': 'Proline', 'Q': 'Glutamine', 'R': 'Arginine', 'S': 'Serine',
               'T': 'Theronine', 'V': 'Valine', 'W': 'Tryptophan', 'Y': 'Tyrosine', '*': 'STOP'
           }
```

"Spike" as a peptide sequence



```
In [139]:   peptide = ''.join([codon[spike[i:i+3]] for i in range(0,len(spike),3)])
            print(peptide)
```

```
MFVFLVLLPLVSSQCVNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFSNVTWFHAIHVSGTNGTKRFDNPVLPFNDGVYFASTEKSNIIRGWIFGTTLDSKTQSLL
IVNNATNVVIKVCEFQFCNDPFLGVYYHKNNKSWMESEFRVYSSANNCTFEYVSQPFLMDLEGKQGNFKNLREFVFKNIDGYFKIYSKHTPINLVRDLPQGFSALEPLVDLPIGINIT
RFQTLLALHRSYLTPGDSSSGWTAGAAAYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETKCTLKSFTVEKGIYQTSNFRVQPTESIVRFPNITNLCPFGEVFNATRFASVYAWN
RKRISNCVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIADYNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLYRLFRKSNLKPFERDISTEI
YQAGSTPCNGVEGFNCYFPLQSYGFQPTNGVGYQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVNFNFNGLTGTGVLTESNKKFLPFQQFGRDIADTTDAVRDPQTLEILDITPC
SFGGVSVITPGTNTSNQVAVLYQDVNCTEVPVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEHVNNSYECDIPIGAGICASYQTQTNSPRRARSVASQSIIAYTMSLGAENSVAYS
NNSIAIPTNFTISVTTEILPVSMTKTSVDCTMYICGDSTECSNLLLQYGSFCTQLNRALTGIAVEQDKNTQEVFAQVKQIYKTPPIKDFGGFNFSQILPDPSKPSKRSFIEDLLFNKV
TLADAGFIKQYGDCLGDIAARDLICAQKFNGLTVLPPLLTDEMIAQYTSALLAGTITSGWTFGAGAALQIPFAMQMAYRFNGIGVTQNVLYENQKLIANQFNSAIGKIQDSLSSTASA
LGKLQDVVNQNAQALNTLVKQLSSNFGAISSVLNDILSRLDKVEAEVQIDRLITGRLQSLQTYVTQQLIRAAEIRASANLAATKMSECVLGQSKRVDFCGKGYHLMSFPQSAPHGVVF
LHVTYVPAQEKNFTTAPAICHDGKAHFPREGVFVSNGTHWFVTQRNFYEPQIITTDNTFVSGNCDVVIGIVNNTVYDPLQPELDSFKEELDKYFKNHTSPDVDLGDISGINASVVNIQ
KEIDRLNEVAKNLNESLIDLQELGKYEQYIKWPWYIWLGFIAGLIAIVMVTIMLCCMTSCCSCLKGCCSCGSCCKFDEDDSEPVLKGVKLHYT*
```

Next time

We'll go hunting for virus fossils.