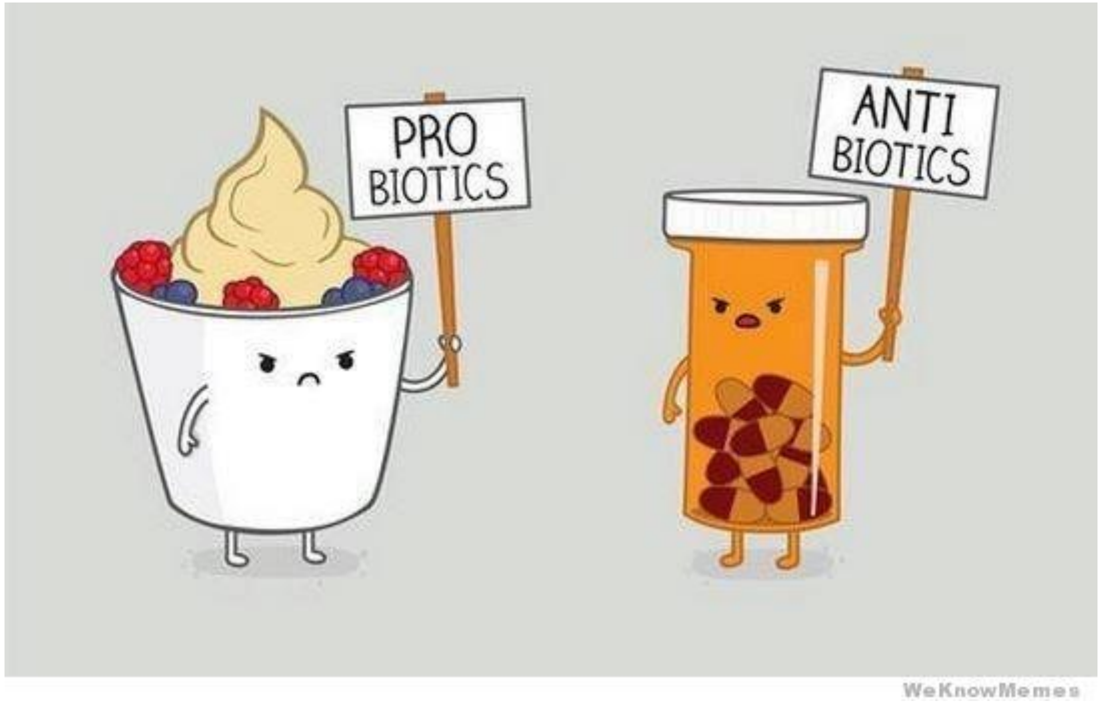
Comp 555 - BioAlgorithms - Spring 2020



- PROBLEM SET #4 IS ONLINE

- Alexander Fleming discovered Antibotics in 1928
- Small peptide sequences that target and kill bacteria
- Generated naturally by fungi and other bacteria
- Today's questions; Where do they come from?
- How can one infer their sequence?

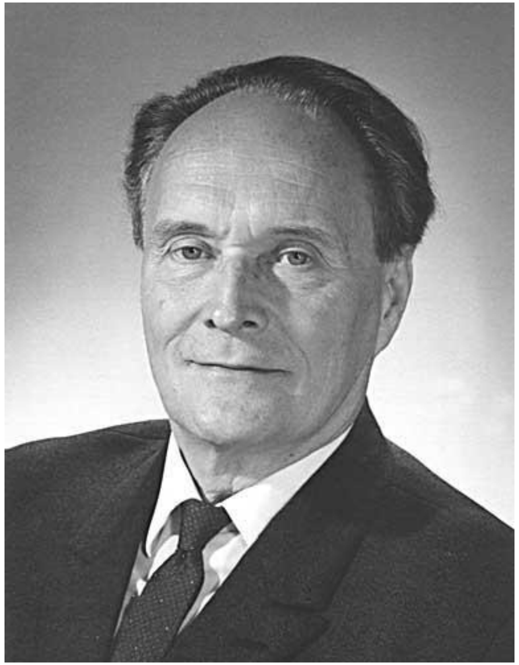
Protein Sequences and Antibotics

Discovery of Penicillin (1928)

- Upon returning from holiday in 1928 a british biologist and pharmacolgist, Alexander Fleming, discovered a strange pattern of cell death in a stack of *staphylococci* cultures he had on a bench in a corner of his laboratory.
- Fleming noticed that one culture was contaminated with a fungus, *Penicillium*, and that the colonies of staphylococci immediately surrounding the fungus had been destroyed, whereas other staphylococci colonies farther away were normal, and famously remarked "That's funny".
- He attempted to isolate the agent that killed the bacteria, and even hypothesized that such an agent might be suitable for treating infections known to be caused by *staphylococci*. But, he had little luck.
- It proved hard to collect enough Penicillin to make it viable drug throughout the 1930s.
- Eventually, a mouldy cantalope found in Illinois in 1943, would provide a mold capable of generating high quantities of Penicillin and greatly reduce the World War II battlefield infection called *Sepsis*.
- The chemical structure of Penicillin was not determined until 1945 by Dorothy Hodgkin, another british X-ray crystallographer like Rosalind Franklin.

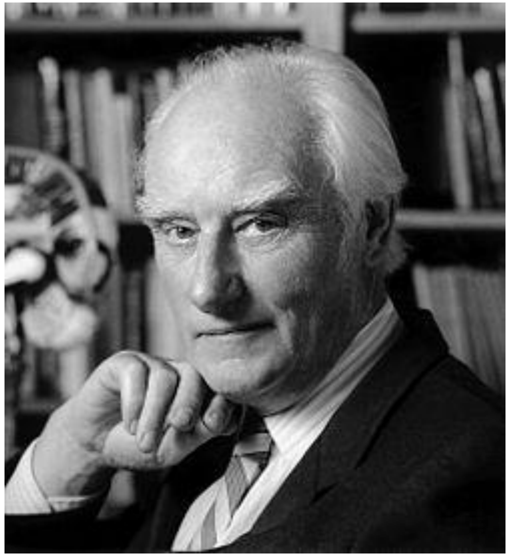
Other Antibotics

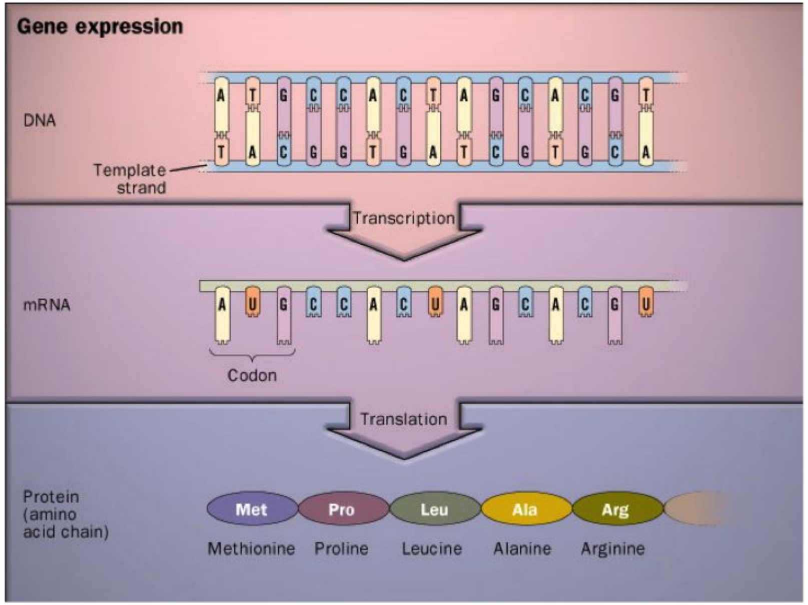
- Meanwhile, a second important antibotic, *Gramicidin S* or *Gramicidin Soviet*, was discovered by Russian microbiologist Georgyi Gause and his wife Maria Brazhnikova in 1942.
- This antibotic was discovered in another strain of bacteria, *Bacillus brevis*, which had the tendancy to kill *staphylococci* nearby it.
- It was actually easier to mass-produce than Penicillin, and it saved the lives of many russians in the second world war, as well as the lives of its inventors.
- *Gramicidin S* is largly composed of a *cyclodecapeptide*, a protein-like peptide chain composed of 10 amino-acids joined head-to-tail to form a ring, called *Tyrocidine B1*.
- The species was reclassified into the genus *Brevibacillus* in 1996.
- The ***Central Dogma of Molecular Biology*** suggests that templates for such peptide chains are encoded in **DNA**.
- Our first task today is to see if we can find a genomic template where *Gramicidin S* is encoded.



Recall the Central Dogma from Lecture 1

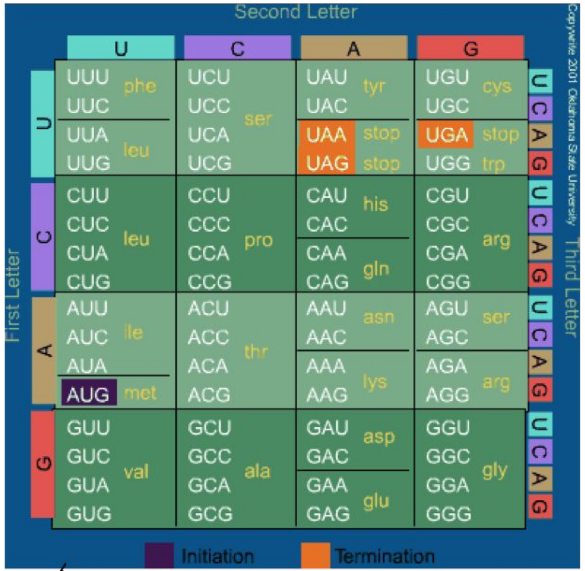
Francis Crick, one of the discoverers of DNA structure, originally coined the term "Central Dogma" in 1958. That choice of words was particularly provocative.


Francis Crick



DNA Transcription

- Triplets of nucleotide bases determine one of 20 amino acids in a protein's peptide sequence
- This mapping is called the *Genetic Code*
- Special STOP codes halt the translation process
- However, the DNA sequence that we see may not be a perfect indicator of the DNA sequence that produced it for two reasons.
    - Post-transcriptional modifications change the mRNA pattern. For example, in eucaryotes sections of code (Introns) are removed from the transcribed sequence.
    - Post-translational modifications modify the produced amino acid chain. For example when a protein is circularized
- So our goal is to find a partial pattern in the DNA which might have created Gramicidin S
- Recall the codon codings

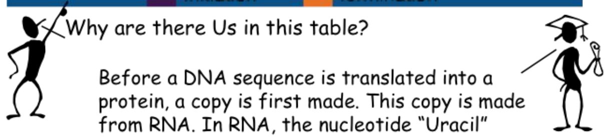


Second Letter

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

🟪 Initiation 🟧 Termination

Copywrite 2001 Oklahoma State University

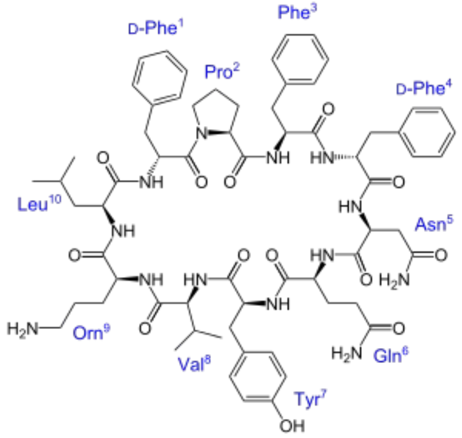


Why are there Us in this table?

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.

Tyrocidine B1's Peptide Sequence

Tyrocidine B1 is a small peptide composed of only 10 amino acids.

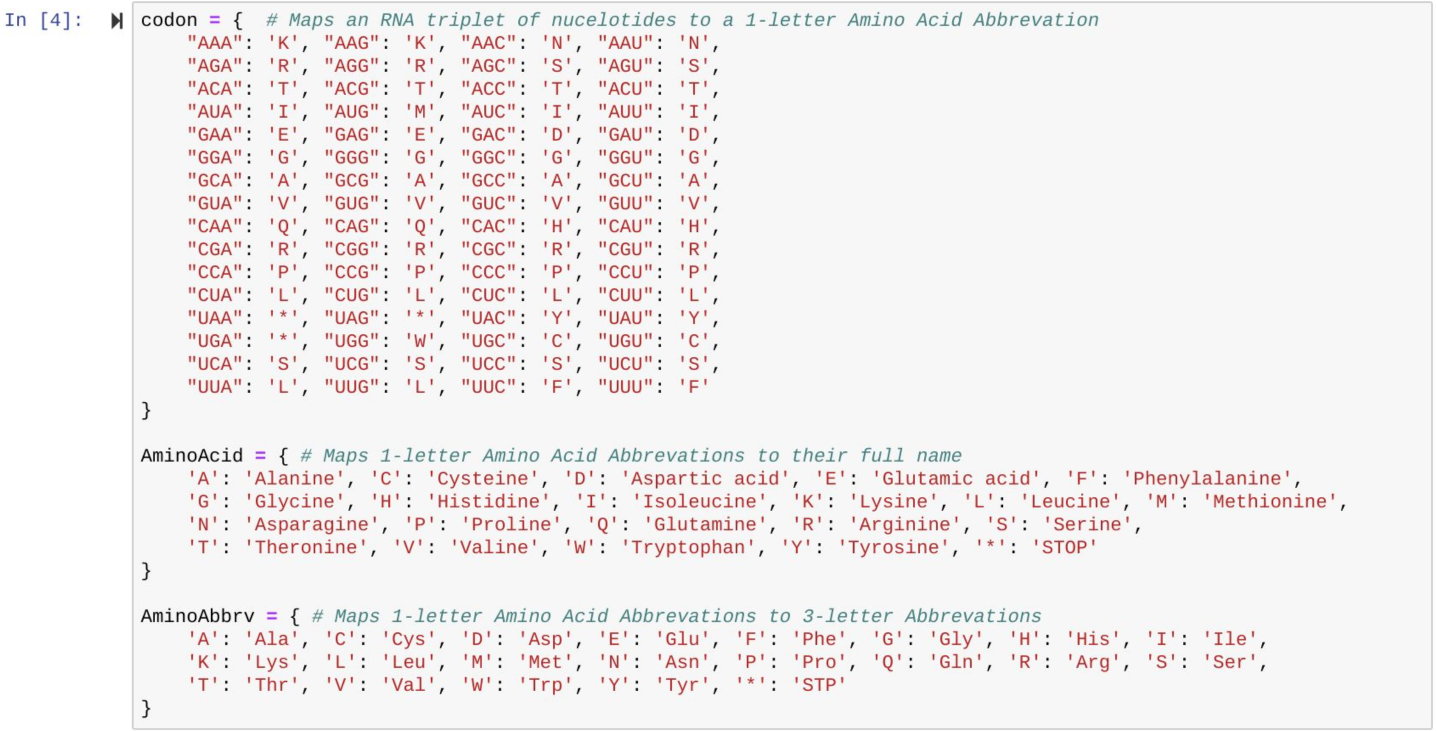


```
Amino Acid:     Valine     Leucine     Proline  Phenylalanine Glutamine
3-letter Abbr:  Val-Lys-Leu-Phe-Pro-Trp-Phe-Asn-Gln-Tyr
1-letter Abbr:   V   K   L   F   P   W   F   N   Q   Y
Amino Acid:          Lysine  Phenylalanine Tryptophan  Asparagine  Tyrosine
```

Also note that the Lysine is modified during the circularization into another non-proteinogenic amino acid, Ornithine.

A toolkit of Python Dictionaries

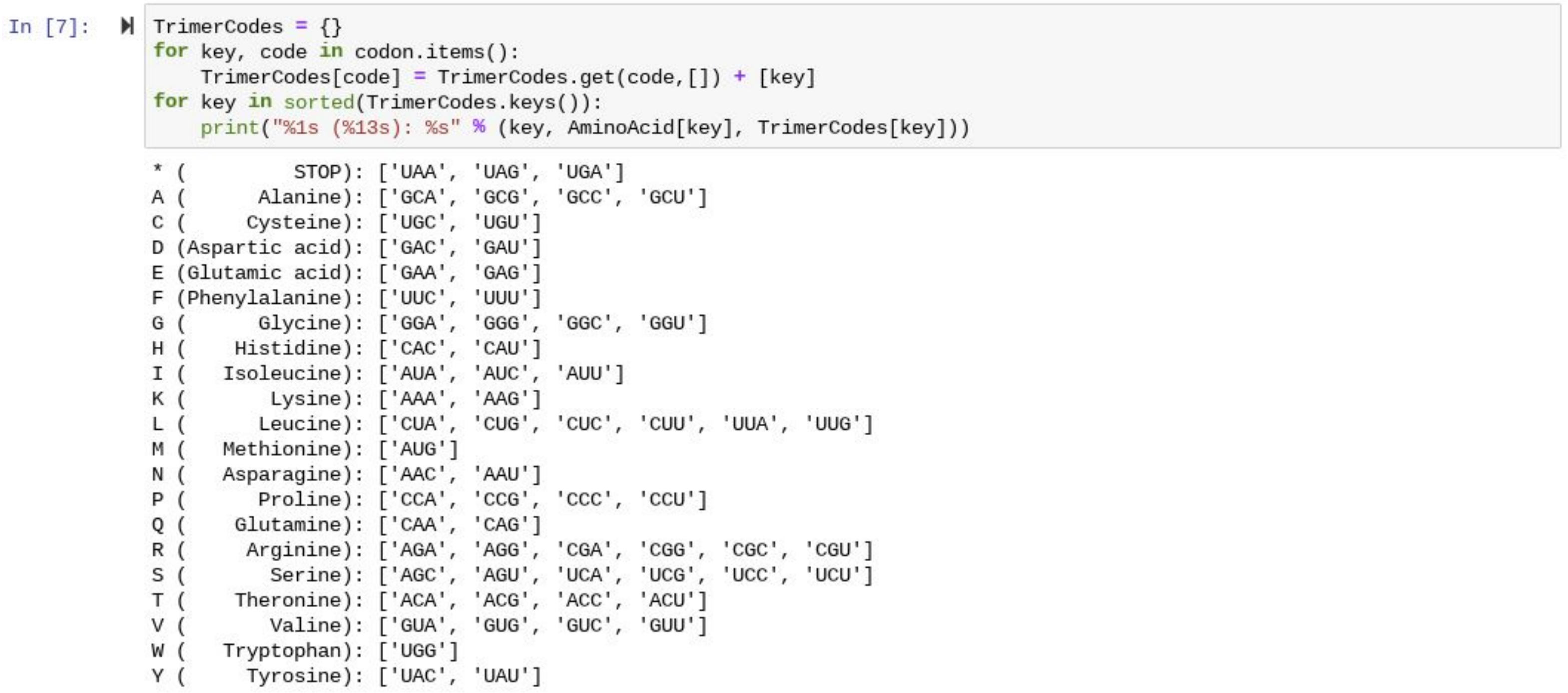
The following Python Dictionaries will be used to aid our search



```
In [4]:  codon = {  # Maps an RNA triplet of nucelotides to a 1-letter Amino Acid Abbrevation
             "AAA": 'K', "AAG": 'K', "AAC": 'N', "AAU": 'N',
             "AGA": 'R', "AGG": 'R', "AGC": 'S', "AGU": 'S',
             "ACA": 'T', "ACG": 'T', "ACC": 'T', "ACU": 'T',
             "AUA": 'I', "AUG": 'M', "AUC": 'I', "AUU": 'I',
             "GAA": 'E', "GAG": 'E', "GAC": 'D', "GAU": 'D',
             "GGA": 'G', "GGG": 'G', "GGC": 'G', "GGU": 'G',
             "GCA": 'A', "GCG": 'A', "GCC": 'A', "GCU": 'A',
             "GUA": 'V', "GUG": 'V', "GUC": 'V', "GUU": 'V',
             "CAA": 'Q', "CAG": 'Q', "CAC": 'H', "CAU": 'H',
             "CGA": 'R', "CGG": 'R', "CGC": 'R', "CGU": 'R',
             "CCA": 'P', "CCG": 'P', "CCC": 'P', "CCU": 'P',
             "CUA": 'L', "CUG": 'L', "CUC": 'L', "CUU": 'L',
             "UAA": '*', "UAG": '*', "UAC": 'Y', "UAU": 'Y',
             "UGA": '*', "UGG": 'W', "UGC": 'C', "UGU": 'C',
             "UCA": 'S', "UCG": 'S', "UCC": 'S', "UCU": 'S',
             "UUA": 'L', "UUG": 'L', "UUC": 'F', "UUU": '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'
         }

         AminoAbbrv = { # Maps 1-letter Amino Acid Abbrevations to 3-letter Abbrevations
             'A': 'Ala', 'C': 'Cys', 'D': 'Asp', 'E': 'Glu', 'F': 'Phe', 'G': 'Gly', 'H': 'His', 'I': 'Ile',
             'K': 'Lys', 'L': 'Leu', 'M': 'Met', 'N': 'Asn', 'P': 'Pro', 'Q': 'Gln', 'R': 'Arg', 'S': 'Ser',
             'T': 'Thr', 'V': 'Val', 'W': 'Trp', 'Y': 'Tyr', '*': 'STP'
         }
```

Now our Peptide sequence

We'll use the 1-letter Amino Acid abbreviations to represent our sequence, and create a dictionary that provides for every peptide in our chain the set of codons that could encode it.

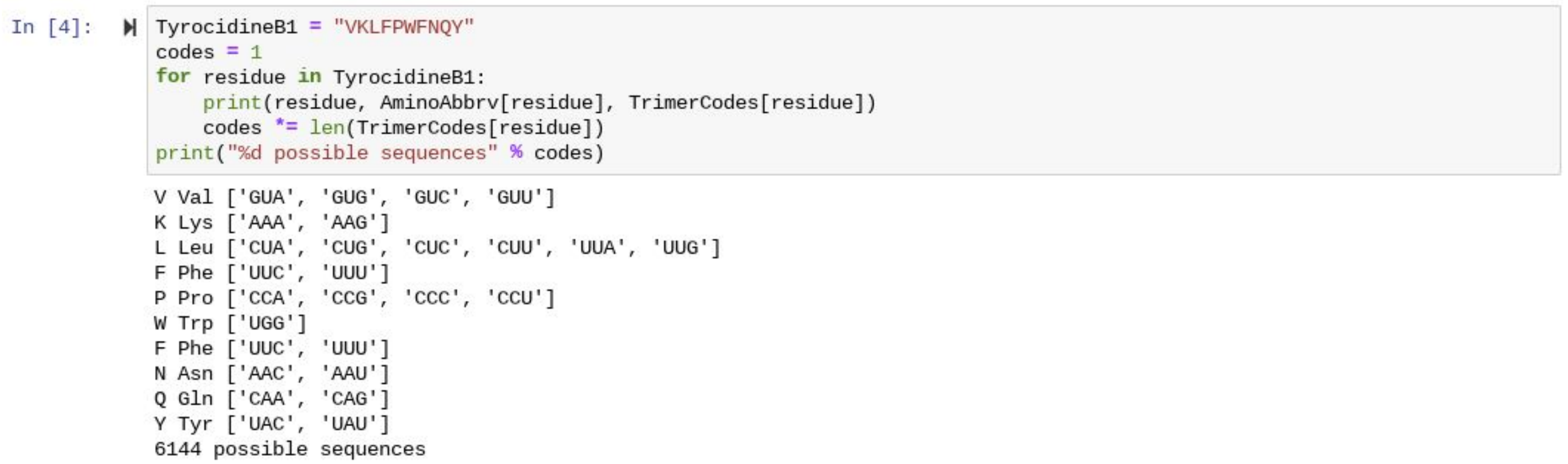


```
In [7]:  TrimerCodes = {}
         for key, code in codon.items():
             TrimerCodes[code] = TrimerCodes.get(code,[]) + [key]
         for key in sorted(TrimerCodes.keys()):
             print("%1s (%13s): %s" % (key, AminoAcid[key], TrimerCodes[key]))
```

```
* (         STOP): ['UAA', 'UAG', 'UGA']
A (      Alanine): ['GCA', 'GCG', 'GCC', 'GCU']
C (     Cysteine): ['UGC', 'UGU']
D (Aspartic acid): ['GAC', 'GAU']
E (Glutamic acid): ['GAA', 'GAG']
F (Phenylalanine): ['UUC', 'UUU']
G (      Glycine): ['GGA', 'GGG', 'GGC', 'GGU']
H (    Histidine): ['CAC', 'CAU']
I (   Isoleucine): ['AUA', 'AUC', 'AUU']
K (       Lysine): ['AAA', 'AAG']
L (      Leucine): ['CUA', 'CUG', 'CUC', 'CUU', 'UUA', 'UUG']
M (   Methionine): ['AUG']
N (   Asparagine): ['AAC', 'AAU']
P (      Proline): ['CCA', 'CCG', 'CCC', 'CCU']
Q (    Glutamine): ['CAA', 'CAG']
R (     Arginine): ['AGA', 'AGG', 'CGA', 'CGG', 'CGC', 'CGU']
S (       Serine): ['AGC', 'AGU', 'UCA', 'UCG', 'UCC', 'UCU']
T (    Theronine): ['ACA', 'ACG', 'ACC', 'ACU']
V (       Valine): ['GUA', 'GUG', 'GUC', 'GUU']
W (   Tryptophan): ['UGG']
Y (     Tyrosine): ['UAC', 'UAU']
```

How many ways to code Tyrocidine B1?

- Since most Amino Acids have multiple codon encodings, most peptide sequences can be codes in multiple ways.
- Let's figure out exactly how many ways there are to encode our little peptide.



In [4]:

```python
TyrocidineB1 = "VKLFPWFNQY"
codes = 1
for residue in TyrocidineB1:
    print(residue, AminoAbbrv[residue], TrimerCodes[residue])
    codes *= len(TrimerCodes[residue])
print("%d possible sequences" % codes)
```

```
V Val ['GUA', 'GUG', 'GUC', 'GUU']
K Lys ['AAA', 'AAG']
L Leu ['CUA', 'CUG', 'CUC', 'CUU', 'UUA', 'UUG']
F Phe ['UUC', 'UUU']
P Pro ['CCA', 'CCG', 'CCC', 'CCU']
W Trp ['UGG']
F Phe ['UUC', 'UUU']
N Asn ['AAC', 'AAU']
Q Gln ['CAA', 'CAG']
Y Tyr ['UAC', 'UAU']
6144 possible sequences
```

Where do we start?

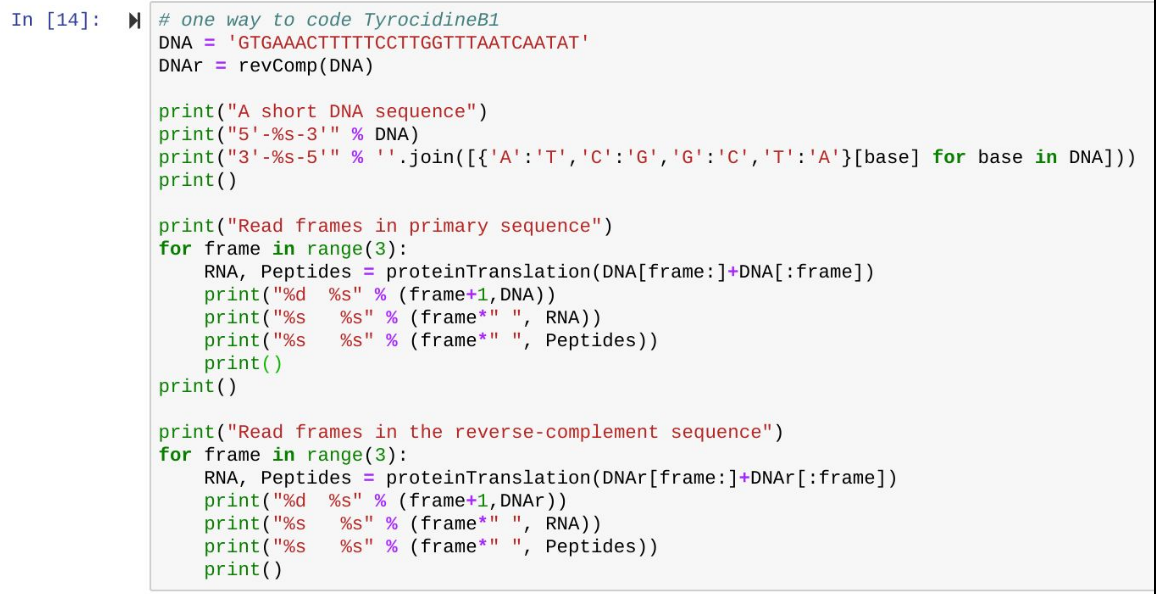
- Consider that translation could begin at any point in the genome.
- Genes can be encoded on either strand

```
<-C-A-A-T-T-T-G-A-A-A-A-A-G-G-G-A-C-C-A-A-A-T-T-G-G-T-C-A-T-A-
  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
 -G-T-T-A-A-A-C-T-T-T-T-T-C-C-C-T-G-G-T-T-T-A-A-C-C-A-G-T-A-T->
```

- There are 3 possible ***open-read frames*** (mod-3 starting positions) on each strand.
- Meanwhile, our peptide has 10 possible starting positions, since it is circular we can't be sure which codon appears first.
- Let's start with something simple
  - Converting a DNA sequence into a protein sequence.
  - In other words, the computational version of translation

Some Helper Functions

- Reverse-complement a sequence
- Translate a DNA sequence to a Protein sequence



```
In [14]:  # one way to code TyrocidineB1
          DNA = 'GTGAAACTTTTTCCTTGGTTTAATCAATAT'
          DNAr = revComp(DNA)

          print("A short DNA sequence")
          print("5'-%s-3'" % DNA)
          print("3'-%s-5'" % ''.join([{'A':'T','C':'G','G':'C','T':'A'}[base] for base in DNA]))
          print()

          print("Read frames in primary sequence")
          for frame in range(3):
              RNA, Peptides = proteinTranslation(DNA[frame:]+DNA[:frame])
              print("%d  %s" % (frame+1,DNA))
              print("%s   %s" % (frame*" ", RNA))
              print("%s   %s" % (frame*" ", Peptides))
              print()
          print()

          print("Read frames in the reverse-complement sequence")
          for frame in range(3):
              RNA, Peptides = proteinTranslation(DNAr[frame:]+DNAr[:frame])
              print("%d  %s" % (frame+1,DNAr))
              print("%s   %s" % (frame*" ", RNA))
              print("%s   %s" % (frame*" ", Peptides))
              print()
```

```
A short DNA sequence
5'-GTGAAACTTTTTCCTTGGTTTAATCAATAT-3'
3'-CACTTTGAAAAAGGAACCAAATTAGTTATA-5'

Read frames in primary sequence
1  GTGAAACTTTTTCCTTGGTTTAATCAATAT
   GUGAAACUUUUUCCUUGGUUUAAUCAAUAU
   ValLysLeuPheProTrpPheAsnGlnTyr

2  GTGAAACTTTTTCCTTGGTTTAATCAATAT
    UGAAACUUUUUCCUUGGUUUAAUCAAUAUG
    STPAsnPhePheLeuGlyLeuIleAsnMet

3  GTGAAACTTTTTCCTTGGTTTAATCAATAT
     GAAACUUUUUCCUUGGUUUAAUCAAUAUGU
     GluThrPheSerLeuValSTPSerIleCys

Read frames in the reverse-complement sequence
1  ATATTGATTAAACCAAGGAAAAAGTTTCAC
   AUAUUGAUUAAACCAAGGAAAAAGUUUCAC
   IleLeuIleLysProArgLysLysPheHis

2  ATATTGATTAAACCAAGGAAAAAGTTTCAC
    UAUUGAUUAAACCAAGGAAAAAGUUUCACA
    TyrSTPLeuAsnGlnGlyLysSerPheThr

3  ATATTGATTAAACCAAGGAAAAAGTTTCAC
     AUUGAUUAAACCAAGGAAAAAGUUUCACAU
     IleAspSTPThrLysGluLysValSerHis
```

Now let's try Something Real

```
In [18]: def loadFasta(filename):
             """ Parses a classically formatted and possibly
                 compressed FASTA file into a list of headers
                 and fragment sequences for each sequence contained"""
             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)

         header, seq = loadFasta("data/BacillusBrevis.fa")
         for i in range(len(header)):
             print(header[i])
             print(len(seq[i])-1, "bases", seq[i][:30], "...", seq[i][-30:])
             print()
```

```
Chromosome dna:chromosome chromosome:GCA_000010165.1:Chromosome:1:6296436:1
6296436 bases +GGGTCTGTGGATATCATTTTATCCACAAA ... AAGGCAAATATCCCCATAAAACTATTTCCC
```

Some simple "Computational" experiments

Just like a biologist, it is ill-advised to jump into a problem without first getting some sense of what works and what doesn't.

- Let's first take a look at our data
- One thing you might have noticed about the possible encodings of Tyrocidine B1 is that there is a single *Tryptophan* and only one encoding for this amino acid
- Let's try to anchor our search around that seed and grow the full sequence from there.
- Where and how many *Tryptophan* encodings

```
In [19]: def CodonSeqCount(genome, codonSeq):
             N = 0
             start = 0
             while True:
                 pos = genome.find(codonSeq, start)
                 if (pos < 0):
                     break
                 N += 1
                 start = pos + 1
             return N

         genome = seq[0]
         tryptophanCode = TrimerCodes['W'][0].replace('U','T')
         print("Searching for", tryptophanCode, "Found it", CodonSeqCount(genome,tryptophanCode), "times")
         revCompCode = revComp(tryptophanCode)
         print("Searching for", revCompCode, "Found it", CodonSeqCount(genome,revCompCode), "times")
```

```
Searching for TGG Found it 106555 times
Searching for CCA Found it 107112 times
```

Is this about what you expected? $\frac{6.3\times10^6}{64} \approx 100,000$. We've narrowed our search some, but there's still alot.

Narrowing things down more

Rather than searching for every possible string, let's examine the codons around this initial anchor.

```
In [21]: def AminoAcidSeqCount(genome, peptideSeq):    # Note: Only checks primary strand
             readframe = []
             anchor = peptideSeq[0]
             for rnaSeq in TrimerCodes[anchor]:
                 dnaSeq = rnaSeq.replace('U','T')
                 start = 0
                 while True:
                     pos = genome.find(dnaSeq, start)
                     if (pos < 0):
                         break
                     i = 1
                     while (i < len(peptideSeq) and (pos+3*(i+1) <= len(genome))):
                         if (codon[genome[pos+3*i:pos+3*(i+1)].replace('T','U')] != peptideSeq[i]):
                             break
                         i += 1
                     else:
                         readframe.append(pos)
                     start = pos + 1
                 return readframe

         peptide = "WFNQY"
         for i in range(1,len(peptide)+1):
             ORFs = AminoAcidSeqCount(genome,peptide[:i])
             print("Searching for", peptide[:i], "Found it", len(ORFs), "times")
```

```
Searching for W Found it 106555 times
Searching for WF Found it 3922 times
Searching for WFN Found it 111 times
Searching for WFNQ Found it 5 times
Searching for WFNQY Found it 1 times
```

Could we have been Lucky this time?

Let’s take a look.

```
In [23]: for pos in ORFs:
             start = pos
             end = pos + (3*len(peptide))
             print("%9d %s %s" % (start, genome[start:end], proteinTranslation(genome[start:end])[1]))

           2789614 TGGTTCAACCAATAT TrpPheAsnGlnTyr
```

We didn’t find what we were expecting.

What Went Wrong?

1. We only searched in one direction from our anchor
2. We didn't search the reverse-complement sequence
3. We didn't consider that the cycle could have been broken somewhere in between "W-F-N-Q-Y"
   - One of the earlier 5, 111, 3922, or 106555 candidates might be the solution
4. The best approach might be to call AminoAcidSeqCount() will all 10 circular permutations of "VKLFPWFNQY" (assuming that we also fix the reverse-complement sequence problem)

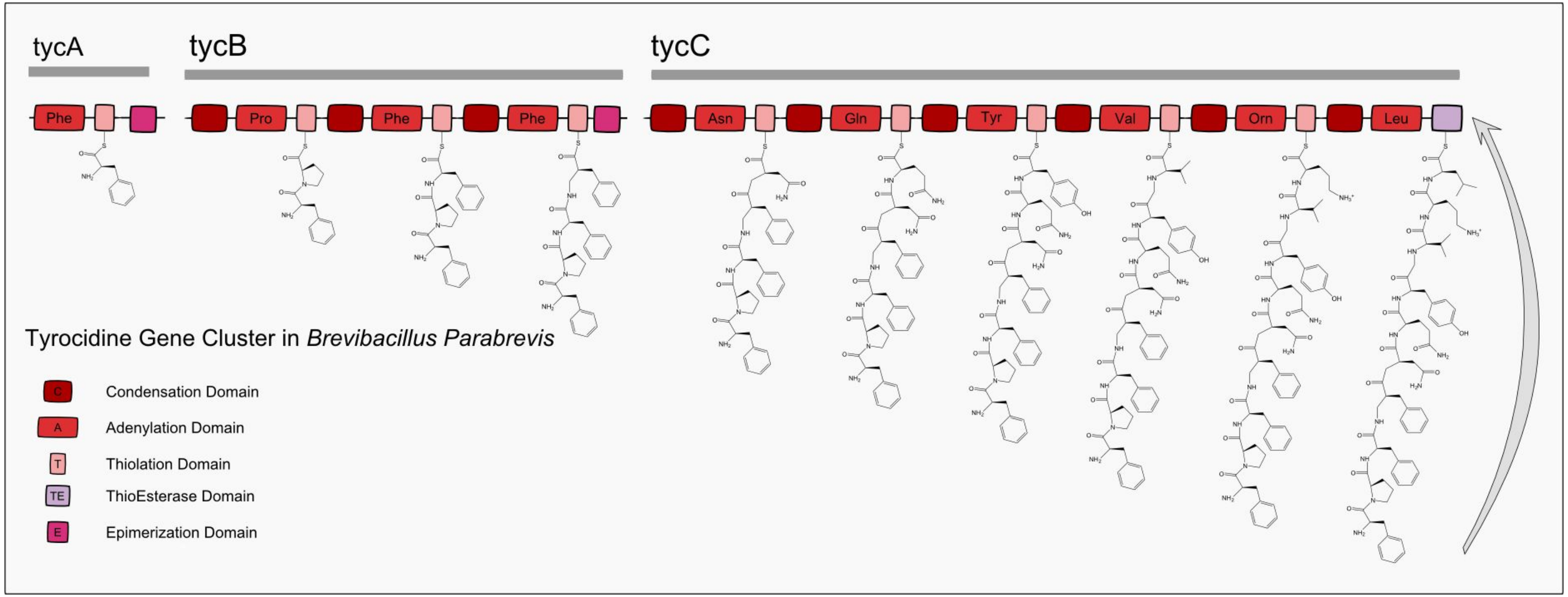
```
In [24]:  [TyrocidineB1[i:]+TyrocidineB1[:i] for i in range(len(TyrocidineB1))]

Out[24]: ['VKLFPWFNQY',
           'KLFPWFNQYV',
           'LFPWFNQYVK',
           'FPWFNQYVKL',
           'PWFNQYVKLF',
           'WFNQYVKLFP',
           'FNQYVKLFPW',
           'NQYVKLFPWF',
           'QYVKLFPWFN',
           'YVKLFPWFNQ']
```

But even if you tried alll these fixes, you would still not find the code for Tyrocidine B1 in *Brevibacillus brevis*.

Proteins that make Peptide Chains

The Central Dogma is not the only way to make proteins!

Bacteria and fungi use large multifunctional enzymes called *NonRibosomal Peptide Synthetases* (NRPSs) to produce peptide chains that are *not encoded* in DNA. Nonribosomal peptides are synthesized by nonribosomal peptide synthetases, which, unlike the ribosomes, are independent of messenger RNA. Each nonribosomal peptide synthetase can synthesize only one type of peptide. They are often toxins, siderophores, or pigments. Nonribosomal peptide antibiotics, cytostatics, and immunosuppressants are widely used as drugs.



So we are left with a new Question

- How do you figure out a peptide's amino acid sequence?
  We just saw it may not appear in the genome.
- Next time we will delve deeper into laboratory methods to start a discussion about peptide sequence inference