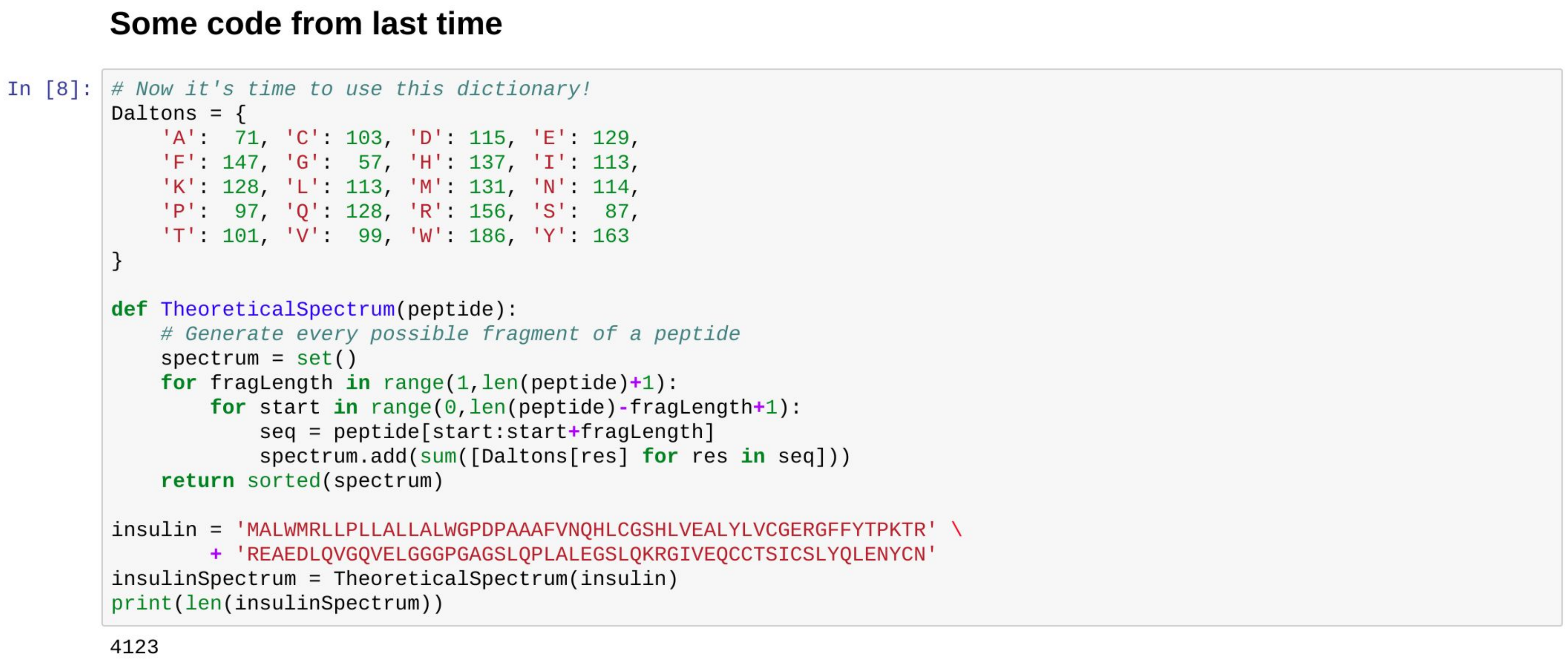
Comp 555 - BioAlgorithms - Spring 2020

- How well do our methods of mapping spectrums to sequences scale?
- How can we determine a peptide's sequence in the presence of errors or impurities?

PROBLEM SET #4 IS DUE NEXT TUESDAY

Scaling Up Peptide Sequencing

Some code from last time

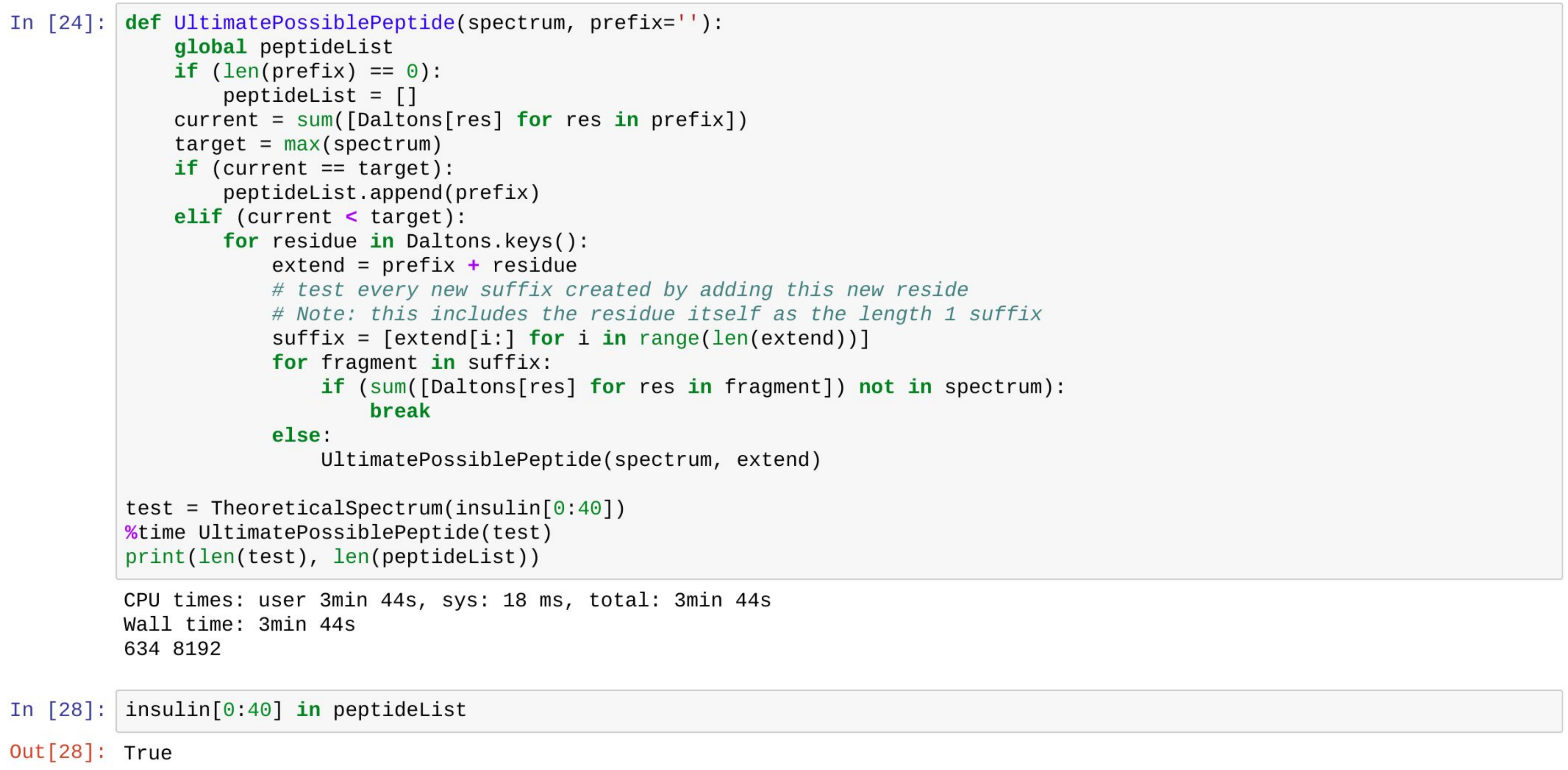


Some code from last time

In [8]:
```
# Now it's time to use this dictionary!
Daltons = {
    'A':  71, 'C': 103, 'D': 115, 'E': 129,
    'F': 147, 'G':  57, 'H': 137, 'I': 113,
    'K': 128, 'L': 113, 'M': 131, 'N': 114,
    'P':  97, 'Q': 128, 'R': 156, 'S':  87,
    'T': 101, 'V':  99, 'W': 186, 'Y': 163
}

def TheoreticalSpectrum(peptide):
    # Generate every possible fragment of a peptide
    spectrum = set()
    for fragLength in range(1,len(peptide)+1):
        for start in range(0,len(peptide)-fragLength+1):
            seq = peptide[start:start+fragLength]
            spectrum.add(sum([Daltons[res] for res in seq]))
    return sorted(spectrum)

insulin = 'MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTR' \
        + 'REAEDLQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN'
insulinSpectrum = TheoreticalSpectrum(insulin)
print(len(insulinSpectrum))
```

```
4123
```

Reminder where we left off



```
In [24]: def UltimatePossiblePeptide(spectrum, prefix=''):
             global peptideList
             if (len(prefix) == 0):
                 peptideList = []
             current = sum([Daltons[res] for res in prefix])
             target = max(spectrum)
             if (current == target):
                 peptideList.append(prefix)
             elif (current < target):
                 for residue in Daltons.keys():
                     extend = prefix + residue
                     # test every new suffix created by adding this new reside
                     # Note: this includes the residue itself as the length 1 suffix
                     suffix = [extend[i:] for i in range(len(extend))]
                     for fragment in suffix:
                         if (sum([Daltons[res] for res in fragment]) not in spectrum):
                             break
                     else:
                         UltimatePossiblePeptide(spectrum, extend)

         test = TheoreticalSpectrum(insulin[0:40])
         %time UltimatePossiblePeptide(test)
         print(len(test), len(peptideList))

         CPU times: user 3min 44s, sys: 18 ms, total: 3min 44s
         Wall time: 3min 44s
         634 8192

In [28]: insulin[0:40] in peptideList

Out[28]: True
```

Our assumptions have been a little Naïve

In reality, Mass Spectometers don't report the Theoretical Spectrum of a peptide

- Instead they report a measured or Experimental Spectrum
- This spectrum might miss some fragments
- It might also report false fragments
- From Contaminants
- New peptides formed by unintended reactions between fragments
- The result is that some of the masses that appear may be misleading, and some that we want might be missing
- We need to develop algorithms for reporting candidate protein sequences that are robust to noise

Example experimental spectrum for Tyrocidine B1

97, 99, 113, **114**, 128, 147, 163,
186, **200**, 227, 241, 242, 244, 260,
261, 283, 291, 333, 340, 357, **388**,
389, 405, 430, 447, **457**, **485**, 487,
543, 544, 552, 575, 577, 584, **659**,
671, 672, 690, 691, **731**, 738, 770,
804, 818, 819, 835, **906**, 917, 932,
982, 1031, **1060**, 1095, 1159, **1223**, 1322

False Masses: present in the experimental spectrum, but not in the theoretical spectrum

Missing Masses: present in the theoretical spectrum, but not in the experimental spectrum

Example experimental spectrum for Tyrocidine B1

97,	99,	113,		128,	147,	163,
186,	**200**,	227,	241,	242,	244,	260,
261,	283,	291,	333,	340,	357,	
	405,	430,	447,	**457**,		487,
543,	544,	552,	575,	577,	584,	**659**,
671,	672,	690,	691,	**731**,	738,	770,
804,	818,	819,	835,	**906**,	917,	932,
982,	1031,		1095,	1159,		1322

False Masses: We don't which these are

Missing Masses: And these values don't even appear

An aside: Faking an Experimental Spectrum

In [26]:
```python
# generate a synthetic experimental spectrum with 10% Error
import itertools
import random
random.seed(1961)

spectrum = TheoreticalSpectrum(TyrocidineB1)

# Pick around ~10% at random to remove
missingMass = random.sample(spectrum[:-1], 6)  # keep largest mass
print("Missing Masses = ", missingMass)

# Add back another ~10% of false, but actual, peptide masses
falseMass = []
for i in range(5):
    fragment = ''.join(random.sample(Daltons.keys(), random.randint(2,len(TyrocidineB1)-2)))
    weight = sum([Daltons[residue] for residue in fragment])
    falseMass.append(weight)
print("False Masses = ", falseMass)

experimentalSpectrum = sorted(set([mass for mass in spectrum if mass not in missingMass] + falseMass))
```

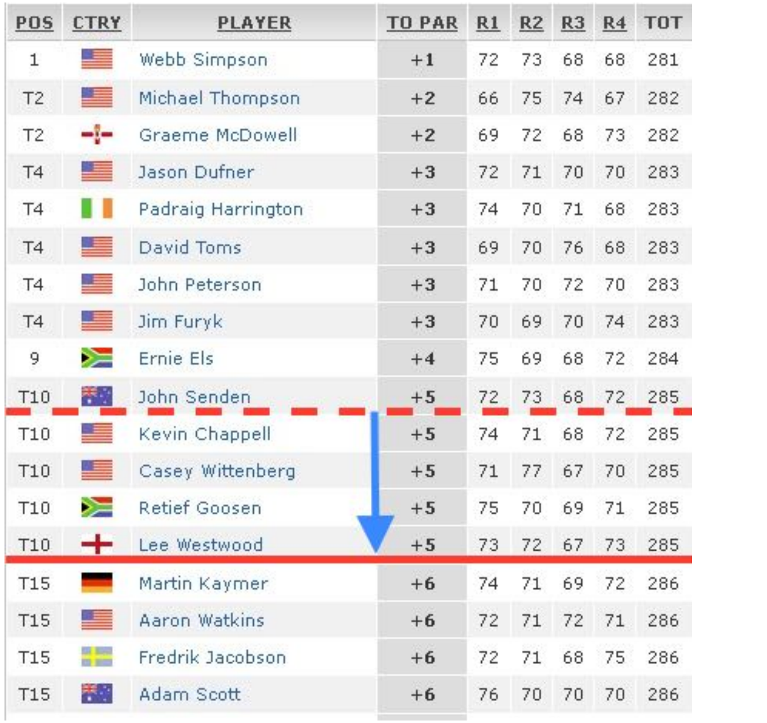
```
Missing Masses =  [917, 114, 244, 405, 241, 99]
False Masses =  [211, 652, 691, 359, 354]
```

In [27]:
```python
print(experimentalSpectrum)
```

```
[97, 113, 128, 147, 163, 186, 211, 227, 242, 260, 261, 283, 291, 333, 340, 354, 357, 359, 388, 389, 430, 447, 485, 48
7, 543, 544, 552, 575, 577, 584, 652, 671, 672, 690, 691, 738, 770, 804, 818, 819, 835, 932, 982, 1031, 1060, 1095, 1
159, 1223, 1322]
```

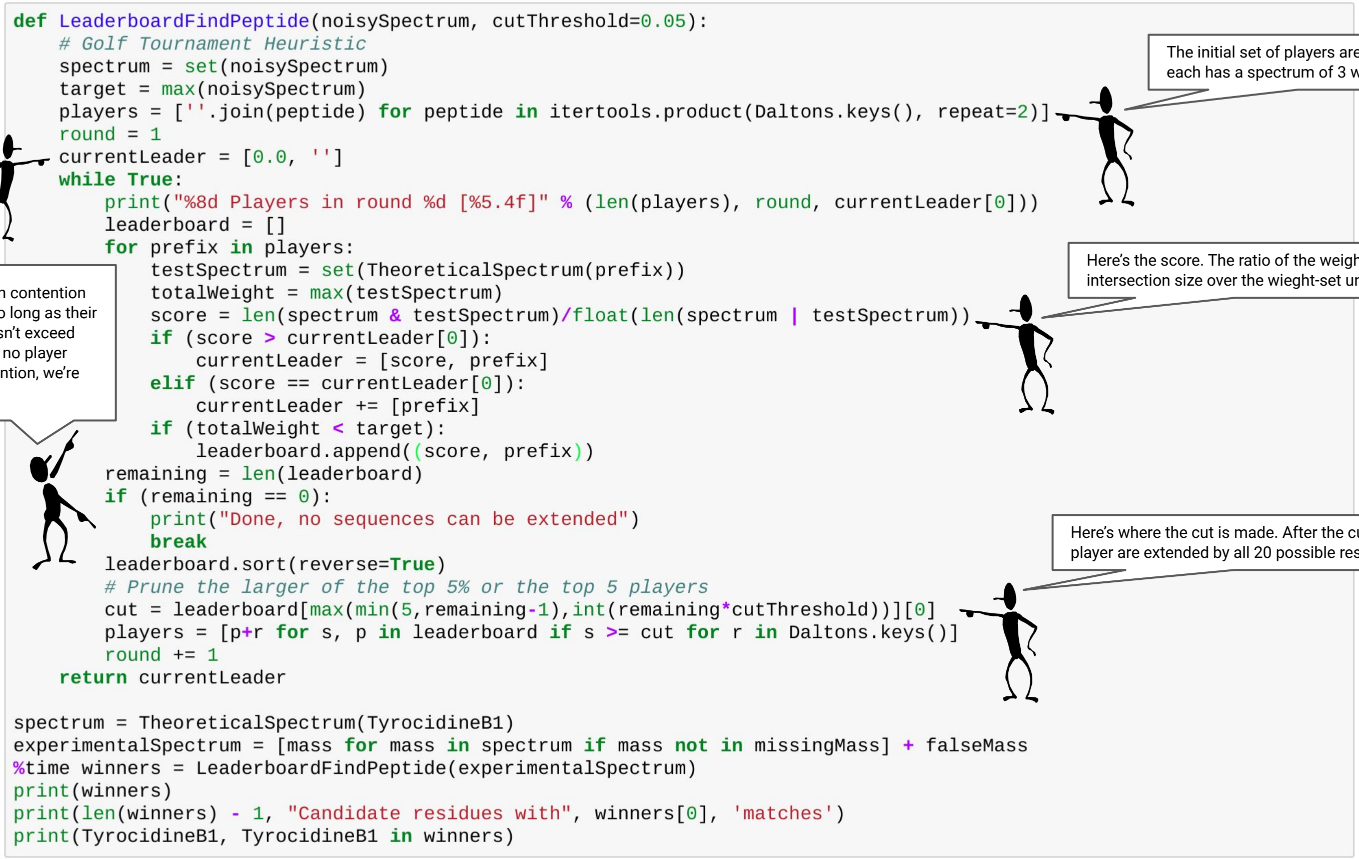
A Golf Tournament Analogy

- After the first couple of rounds of a major golf tournament a *cut* is made of all golfers who are so far back from the leader that it is deemed they are unlikely to ever finish in the money
- These *cut* golfers are removed from further consideration
- This choice is *heuristic*
  - It is possible that a player just below the cut could have two exceptional rounds, but that is considered unlikely
- What is the equivalent of a score in our peptide finding problem?
  - The number of matching masses in the candidate peptide's Theoretical Spectrum and the Experimental Spectrum
  - Normalized score, why?
  - len(intersection of candidate and experimental spectrums) / len(union of candidate and experimental spectrums)
  - ***Jaccard Index*** for sets
- In our peptide *golf game* a round will be considered a one peptide extension of a active set of *player* peptides
- We will do cuts on every round, keeping to top 5% of finishers or the top 5 players, which ever is more
- Why 5%? It is arbitrary, but on each round we will extend the current set of players by one of 20 amino acids, thus increasing the number of peptides by a factor of 20, so reducing by 5% leaves the poolsize relatively stable.



POS	CTRY	PLAYER	TO PAR	R1	R2	R3	R4	TOT
1		Webb Simpson	+1	72	73	68	68	281
T2		Michael Thompson	+2	66	75	74	67	282
T2		Graeme McDowell	+2	69	72	68	73	282
T4		Jason Dufner	+3	72	71	70	70	283
T4		Padraig Harrington	+3	74	70	71	68	283
T4		David Toms	+3	69	70	76	68	283
T4		John Peterson	+3	71	70	72	70	283
T4		Jim Furyk	+3	70	69	70	74	283
9		Ernie Els	+4	75	69	68	72	284
T10		John Senden	+5	72	73	68	72	285
T10		Kevin Chappell	+5	74	71	68	72	285
T10		Casey Wittenberg	+5	71	77	67	70	285
T10		Retief Goosen	+5	75	70	69	71	285
T10		Lee Westwood	+5	73	72	67	73	285
T15		Martin Kaymer	+6	74	71	69	72	286
T15		Aaron Watkins	+6	72	71	72	71	286
T15		Fredrik Jacobson	+6	72	71	68	75	286
T15		Adam Scott	+6	76	70	70	70	286

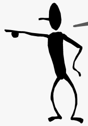
An Implementation



```
In [33]: def LeaderboardFindPeptide(noisySpectrum, cutThreshold=0.05):
             # Golf Tournament Heuristic
             spectrum = set(noisySpectrum)
             target = max(noisySpectrum)
             players = [''.join(peptide) for peptide in itertools.product(Daltons.keys(), repeat=2)]
             round = 1
             currentLeader = [0.0, '']
             while True:
                 print("%8d Players in round %d [%5.4f]" % (len(players), round, currentLeader[0]))
                 leaderboard = []
                 for prefix in players:
                     testSpectrum = set(TheoreticalSpectrum(prefix))
                     totalWeight = max(testSpectrum)
                     score = len(spectrum & testSpectrum)/float(len(spectrum | testSpectrum))
                     if (score > currentLeader[0]):
                         currentLeader = [score, prefix]
                     elif (score == currentLeader[0]):
                         currentLeader += [prefix]
                     if (totalWeight < target):
                         leaderboard.append((score, prefix))
                 remaining = len(leaderboard)
                 if (remaining == 0):
                     print("Done, no sequences can be extended")
                     break
                 leaderboard.sort(reverse=True)
                 # Prune the larger of the top 5% or the top 5 players
                 cut = leaderboard[max(min(5,remaining-1),int(remaining*cutThreshold))][0]
                 players = [p+r for s, p in leaderboard if s >= cut for r in Daltons.keys()]
                 round += 1
             return currentLeader

         spectrum = TheoreticalSpectrum(TyrocidineB1)
         experimentalSpectrum = [mass for mass in spectrum if mass not in missingMass] + falseMass
         %time winners = LeaderboardFindPeptide(experimentalSpectrum)
         print(winners)
         print(len(winners) - 1, "Candidate residues with", winners[0], 'matches')
         print(TyrocidineB1, TyrocidineB1 in winners)
```

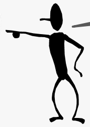
Our answer is a list, where the first element is the best score follwed by all players that achieved it.

The initial set of players are all residue pairs, each has a spectrum of 3 weights

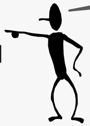


Player's remain in contention during a round so long as their total weight doesn't exceed the target. When no player remains in contention, we're finished.

Here's the score. The ratio of the weight-set intersection size over the wieght-set union size



Here's where the cut is made. After the cut all remaining player are extended by all 20 possible residues

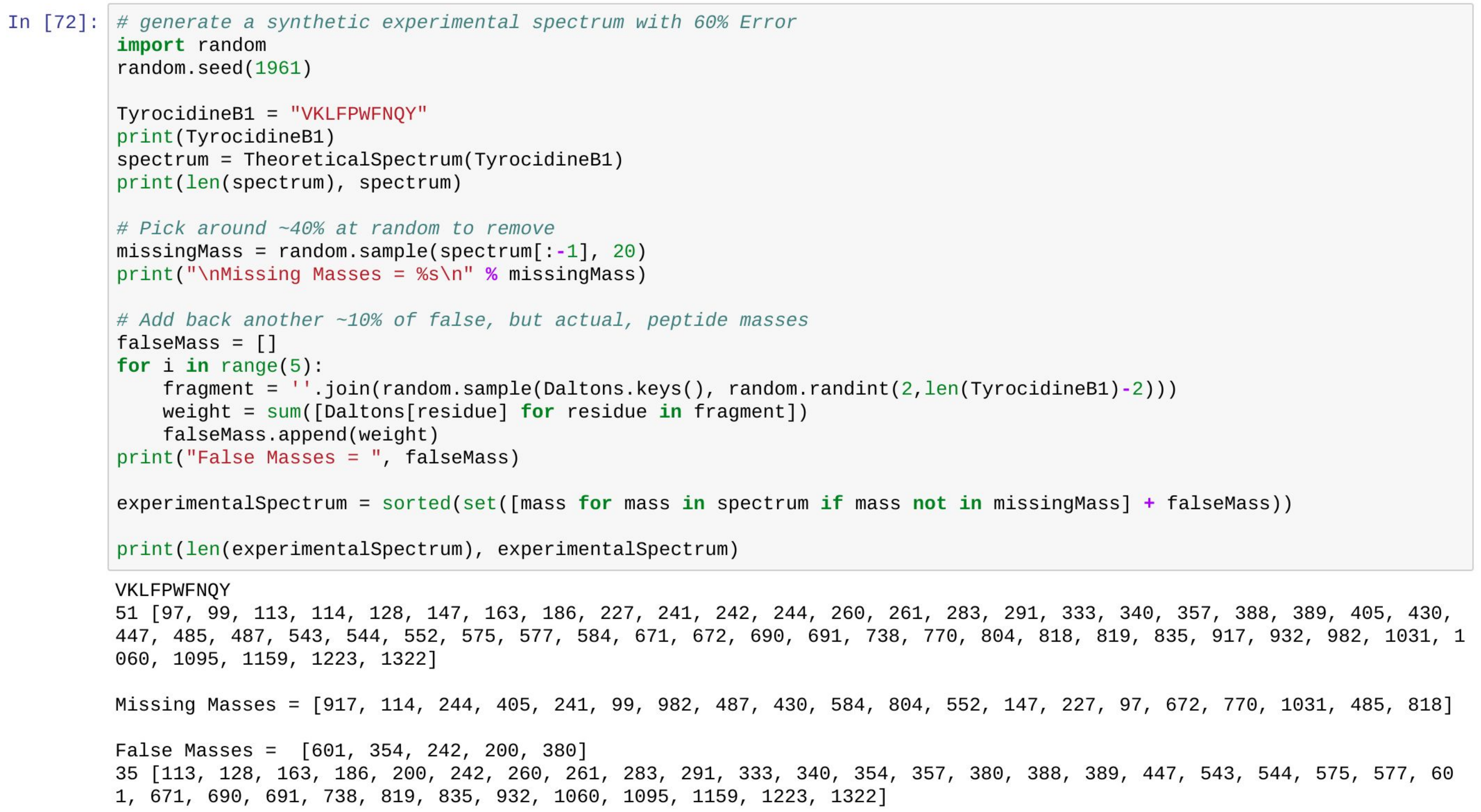


Now for a tournament

```
     400 Players in round 1 [0.0000]
    1440 Players in round 2 [0.0612]
    4960 Players in round 3 [0.1224]
    6400 Players in round 4 [0.1800]
    9380 Players in round 5 [0.2800]
   10000 Players in round 6 [0.3725]
   11820 Players in round 7 [0.4706]
   12800 Players in round 8 [0.5962]
   12880 Players in round 9 [0.6981]
    7520 Players in round 10 [0.8182]
     640 Players in round 11 [0.8182]
Done, no sequences can be extended
CPU times: user 5.54 s, sys: 27 ms, total: 5.57 s
Wall time: 5.58 s
[0.8181818181818182, 'YQNFWPFLQV', 'YQNFWPFLKV', 'YQNFWPFIQV', 'YQNFWPFIKV', 'YKNFWPFLQV', 'YKNFWPFLKV', 'YKNFWPFIQ
V', 'YKNFWPFIKV', 'VQLFPWFNQY', 'VQLFPWFNKY', 'VQIFPWFNQY', 'VQIFPWFNKY', 'VKLFPWFNQY', 'VKLFPWFNKY', 'VKIFPWFNQY',
'VKIFPWFNKY']
16 Candidate residues with 0.8181818181818182 matches
VKLFPWFNQY True
```

Not too slow! And it found our answer!

Let’s try a Nosier Spectrum



```
In [72]: # generate a synthetic experimental spectrum with 60% Error
         import random
         random.seed(1961)

         TyrocidineB1 = "VKLFPWFNQY"
         print(TyrocidineB1)
         spectrum = TheoreticalSpectrum(TyrocidineB1)
         print(len(spectrum), spectrum)

         # Pick around ~40% at random to remove
         missingMass = random.sample(spectrum[:-1], 20)
         print("\nMissing Masses = %s\n" % missingMass)

         # Add back another ~10% of false, but actual, peptide masses
         falseMass = []
         for i in range(5):
             fragment = ''.join(random.sample(Daltons.keys(), random.randint(2,len(TyrocidineB1)-2)))
             weight = sum([Daltons[residue] for residue in fragment])
             falseMass.append(weight)
         print("False Masses = ", falseMass)

         experimentalSpectrum = sorted(set([mass for mass in spectrum if mass not in missingMass] + falseMass))

         print(len(experimentalSpectrum), experimentalSpectrum)
```

```
VKLFPWFNQY
51 [97, 99, 113, 114, 128, 147, 163, 186, 227, 241, 242, 244, 260, 261, 283, 291, 333, 340, 357, 388, 389, 405, 430,
447, 485, 487, 543, 544, 552, 575, 577, 584, 671, 672, 690, 691, 738, 770, 804, 818, 819, 835, 917, 932, 982, 1031, 1
060, 1095, 1159, 1223, 1322]

Missing Masses = [917, 114, 244, 405, 241, 99, 982, 487, 430, 584, 804, 552, 147, 227, 97, 672, 770, 1031, 485, 818]

False Masses =  [601, 354, 242, 200, 380]
35 [113, 128, 163, 186, 200, 242, 260, 261, 283, 291, 333, 340, 354, 357, 380, 388, 389, 447, 543, 544, 575, 577, 60
1, 671, 690, 691, 738, 819, 835, 932, 1060, 1095, 1159, 1223, 1322]
```

Find peptides via the leaderboard approach

```
In [73]: spectrum = TheoreticalSpectrum(TyrocidineB1)
         experimentalSpectrum = [mass for mass in spectrum if mass not in missingMass] + falseMass
         %time winners = LeaderboardFindPeptide(experimentalSpectrum)
         print(winners)
         print(len(winners) - 1, "Candidate residues with", winners[0], 'matches')
         print(TyrocidineB1, TyrocidineB1 in winners)
```

```
     400 Players in round 1 [0.0000]
     960 Players in round 2 [0.0857]
    1300 Players in round 3 [0.1389]
    1740 Players in round 4 [0.2162]
    4280 Players in round 5 [0.2895]
    5600 Players in round 6 [0.3333]
    5800 Players in round 7 [0.4524]
    5960 Players in round 8 [0.5333]
    6120 Players in round 9 [0.5833]
    2480 Players in round 10 [0.5833]
     240 Players in round 11 [0.5833]
Done, no sequences can be extended
CPU times: user 2.4 s, sys: 10 ms, total: 2.41 s
Wall time: 2.4 s
[0.5833333333333334, 'YQNFWPFLK', 'YQNFWPFLQ', 'YQNFWPFIK', 'YQNFWPFIQ', 'YKNFWPFLK', 'YKNFWPFLQ', 'YKNFWPFIK', 'YKNF
WPFIQ']
8 Candidate residues with 0.5833333333333334 matches
VKLFPWFNQY False
```

A New Idea

- Maybe we are still not using our spectrum to its fullest extent
- Is there some information about missing masses that we can extract?

Information in the Mass Differences

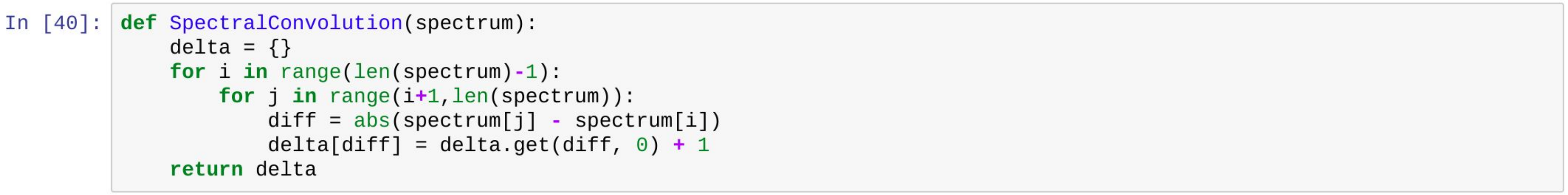
- Recall the theoretical spectrum of "PLAY" is [71, 97, 113, 163, 184, 210, 234, 281, 347, 444]
- Suppose we remove masses 71 and 163, can we get them back?
- Let's generate a table of all pair-wise differences between the observed peaks
- Notice that interesting numbers, (71, 97, 113, 137, 163, 234) are repeated in the table

	97	**113**	**184**	**210**	**234**	**281**	**347**	**444**
97		16	87	113	137	184	250	347
113			71	97	121	168	234	331
184				26	50	97	163	260
210					24	71	137	234
234						47	113	210
281							66	163
347								97

- Why does this work?
- This table of differences is called a ***Spectral Convolution***

Spectral Convolution

- Spectral Convolution recovers some missing masses
- Given a noisy experimental spectrum
  - Compute its spectral convolution
  - Add frequent masses above some threshold to the spectrum
  - Infer the peptide sequence



```
In [40]: def SpectralConvolution(spectrum):
             delta = {}
             for i in range(len(spectrum)-1):
                 for j in range(i+1,len(spectrum)):
                     diff = abs(spectrum[j] - spectrum[i])
                     delta[diff] = delta.get(diff, 0) + 1
             return delta
```

Spiking with Spectral Convolution

```
In [75]: spectrum = TheoreticalSpectrum(TyrocidineB1)
         print(sorted(missingMass), len(missingMass))
         experimentalSpectrum = sorted(set([mass for mass in spectrum if mass not in missingMass] + falseMass))
         specConv = SpectralConvolution(sorted(experimentalSpectrum))
         N = 0
         for delta, count in sorted(specConv.items()):
             if (count >= 2) and (delta not in experimentalSpectrum) and (delta > min(Daltons.values())):
                 print("%3d appears %1d times%s\t" % (delta, count,  '*' if delta in missingMass else ' '), end='')
                 experimentalSpectrum.append(delta)
                 N += 1
                 if (N % 4 == 0):
                     print()
         print()
```

```
[97, 99, 114, 147, 227, 241, 244, 405, 430, 485, 487, 552, 584, 672, 770, 804, 818, 917, 982, 1031] 20
 58 appears 3 times      64 appears 2 times      67 appears 2 times      72 appears 2 times
 73 appears 2 times      74 appears 2 times      79 appears 2 times      89 appears 2 times
 90 appears 2 times      91 appears 2 times      93 appears 2 times      94 appears 2 times
 96 appears 3 times      97 appears 8 times*     98 appears 3 times      99 appears 2 times*
105 appears 2 times     114 appears 3 times*    115 appears 2 times     120 appears 2 times
127 appears 2 times     129 appears 3 times     133 appears 2 times     146 appears 2 times
147 appears 5 times*    148 appears 3 times     154 appears 4 times     155 appears 3 times
156 appears 2 times     164 appears 3 times     170 appears 2 times     187 appears 3 times
188 appears 2 times     189 appears 3 times     194 appears 4 times     195 appears 2 times
203 appears 2 times     205 appears 2 times     212 appears 2 times     218 appears 2 times
220 appears 2 times     221 appears 2 times     225 appears 2 times     226 appears 2 times
227 appears 3 times*    241 appears 3 times*    244 appears 5 times*    247 appears 2 times
252 appears 2 times     275 appears 2 times     276 appears 3 times     282 appears 2 times
284 appears 3 times     292 appears 2 times     301 appears 2 times     302 appears 3 times
310 appears 2 times     314 appears 2 times     317 appears 2 times     331 appears 2 times
334 appears 2 times     350 appears 2 times     358 appears 3 times     381 appears 2 times
404 appears 2 times     405 appears 2 times*    415 appears 2 times     429 appears 2 times
430 appears 4 times*    431 appears 3 times     449 appears 2 times     455 appears 2 times
462 appears 2 times     478 appears 2 times     485 appears 4 times*    488 appears 2 times
528 appears 2 times     552 appears 5 times*    558 appears 3 times     578 appears 2 times
584 appears 2 times*    648 appears 2 times     649 appears 2 times     672 appears 3 times*
680 appears 2 times     706 appears 3 times     707 appears 2 times     769 appears 2 times
779 appears 2 times     804 appears 2 times*    834 appears 2 times     982 appears 2 times*
1031 appears 2 times*
```

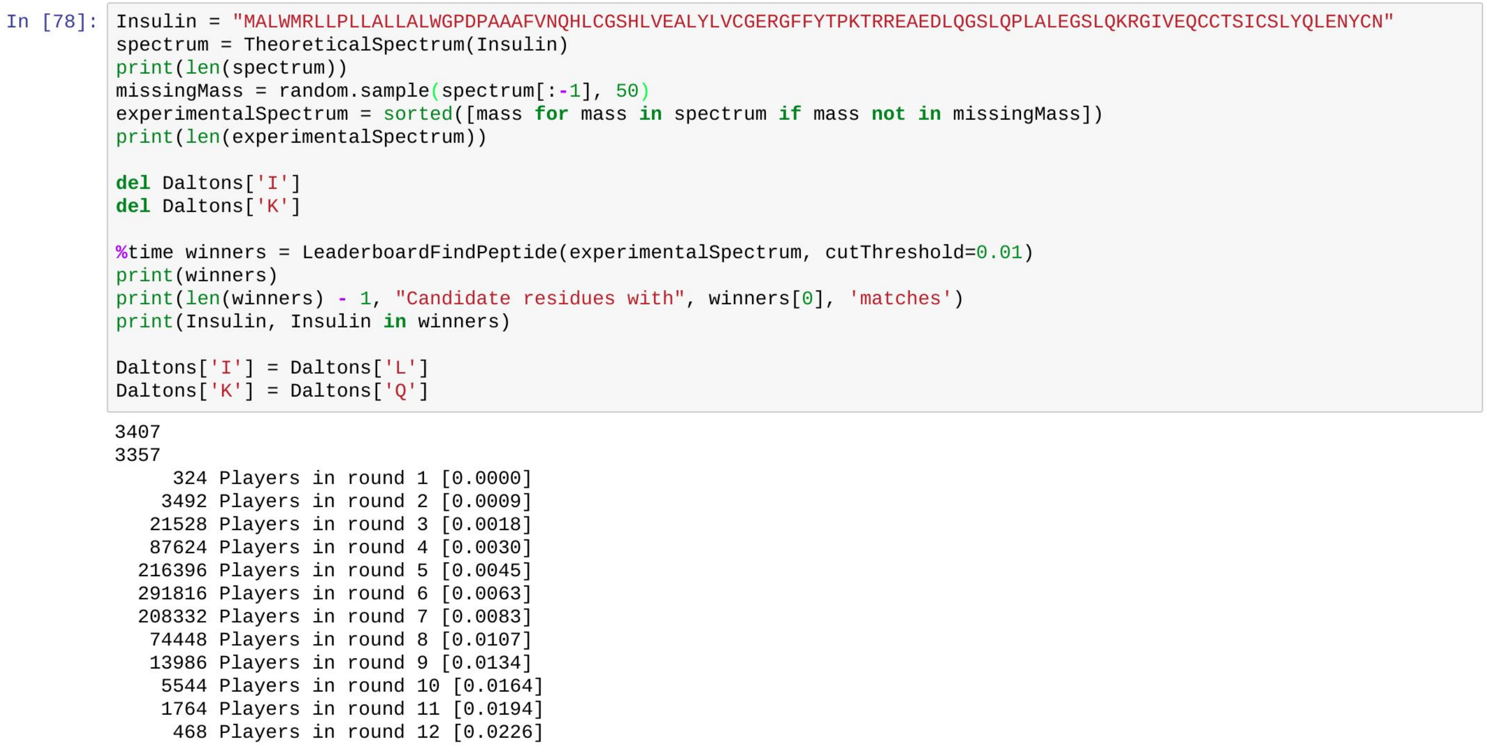
Now we try again

```
In [76]: %time winners = LeaderboardFindPeptide(experimentalSpectrum)
         print(winners)
         print(len(winners) - 1, "Candidate residues with", winners[0], 'matches')
         print(TyrocidineB1, TyrocidineB1 in winners)
```

```
     400 Players in round 1 [0.0000]
    1600 Players in round 2 [0.0234]
    3600 Players in round 3 [0.0469]
    8220 Players in round 4 [0.0781]
    8460 Players in round 5 [0.1172]
   14260 Players in round 6 [0.1641]
   18880 Players in round 7 [0.2031]
   19140 Players in round 8 [0.2656]
   19240 Players in round 9 [0.3101]
    8560 Players in round 10 [0.3561]
    2160 Players in round 11 [0.3561]
     160 Players in round 12 [0.3561]
Done, no sequences can be extended
CPU times: user 8.55 s, sys: 9 ms, total: 8.56 s
Wall time: 8.55 s
[0.3560606060606061, 'YQNFWPFLQV', 'YQNFWPFLKV', 'YQNFWPFIQV', 'YQNFWPFIKV', 'YKNFWPFLQV', 'YKNFWPFLKV', 'YKNFWPFIQ
V', 'YKNFWPFIKV', 'VQLFPWFNQY', 'VQLFPWFNKY', 'VQIFPWFNQY', 'VQIFPWFNKY', 'VKLFPWFNQY', 'VKLFPWFNKY', 'VKIFPWFNQY',
'VKIFPWFNKY']
16 Candidate residues with 0.3560606060606061 matches
VKLFPWFNQY True
```

A more *Realistic* Example

For long sequences the underlying exponential growth becomes more evident



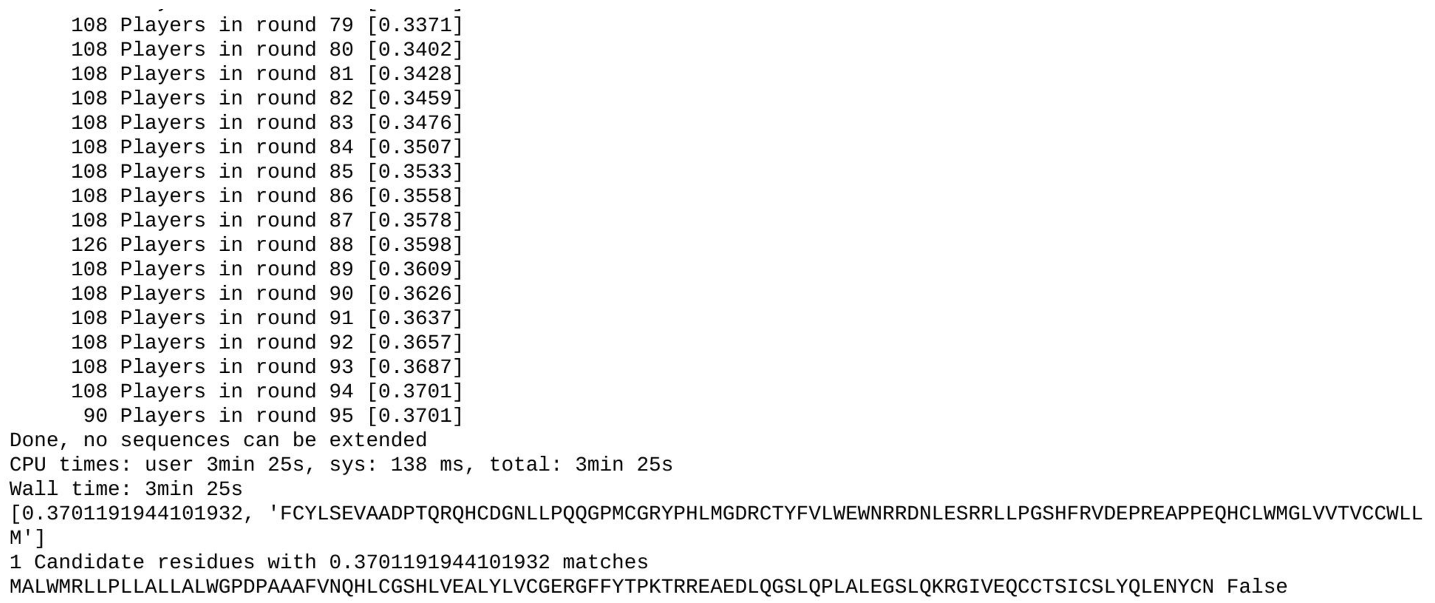
```
In [78]: Insulin = "MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN"
         spectrum = TheoreticalSpectrum(Insulin)
         print(len(spectrum))
         missingMass = random.sample(spectrum[:-1], 50)
         experimentalSpectrum = sorted([mass for mass in spectrum if mass not in missingMass])
         print(len(experimentalSpectrum))

         del Daltons['I']
         del Daltons['K']

         %time winners = LeaderboardFindPeptide(experimentalSpectrum, cutThreshold=0.01)
         print(winners)
         print(len(winners) - 1, "Candidate residues with", winners[0], 'matches')
         print(Insulin, Insulin in winners)

         Daltons['I'] = Daltons['L']
         Daltons['K'] = Daltons['Q']
```

```
3407
3357
     324 Players in round 1 [0.0000]
    3492 Players in round 2 [0.0009]
   21528 Players in round 3 [0.0018]
   87624 Players in round 4 [0.0030]
  216396 Players in round 5 [0.0045]
  291816 Players in round 6 [0.0063]
  208332 Players in round 7 [0.0083]
   74448 Players in round 8 [0.0107]
   13986 Players in round 9 [0.0134]
    5544 Players in round 10 [0.0164]
    1764 Players in round 11 [0.0194]
     468 Players in round 12 [0.0226]
```

A more *Realistic* Example

For long sequences the underlying exponential growth becomes more evident

:



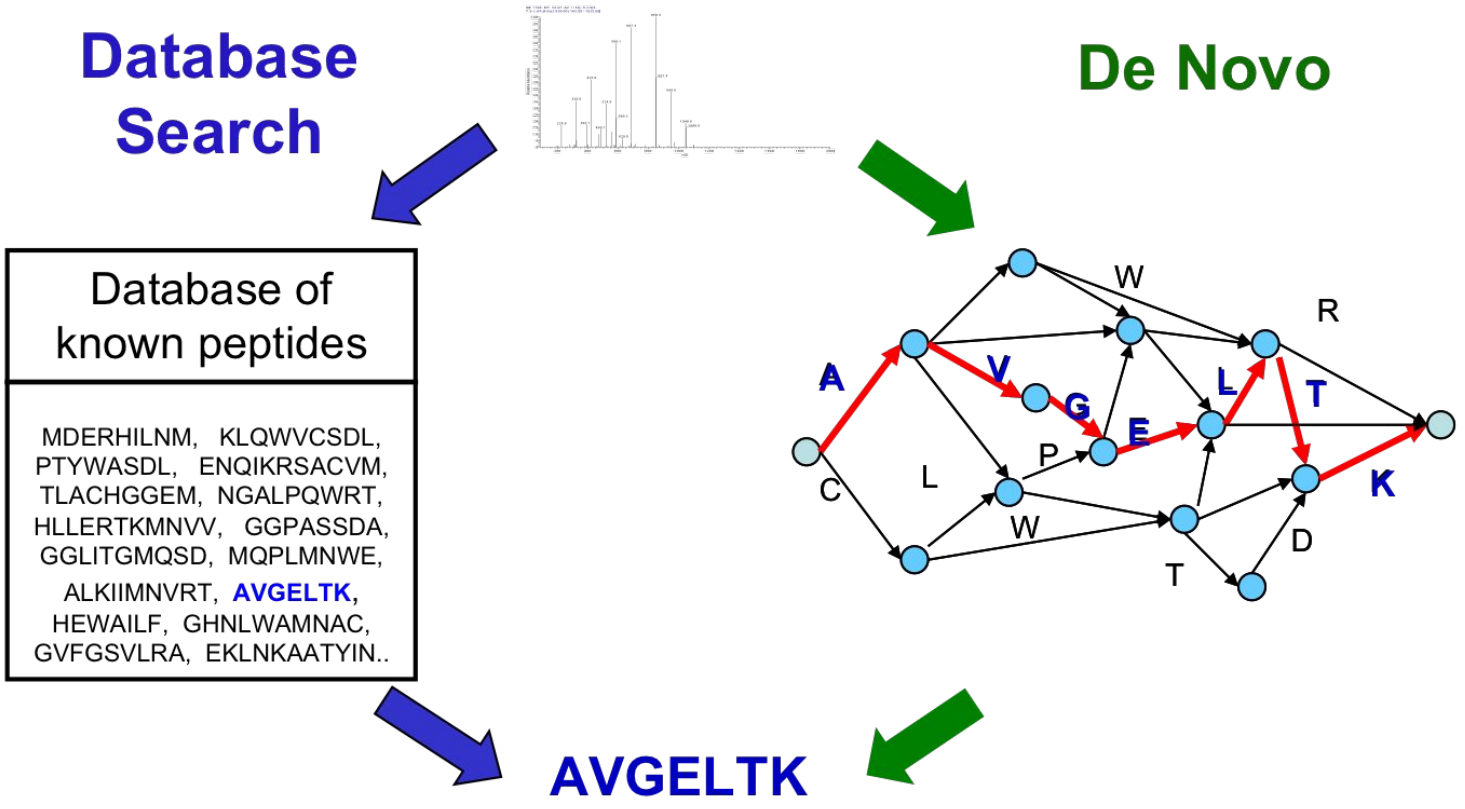
```
     108 Players in round 79 [0.3371]
     108 Players in round 80 [0.3402]
     108 Players in round 81 [0.3428]
     108 Players in round 82 [0.3459]
     108 Players in round 83 [0.3476]
     108 Players in round 84 [0.3507]
     108 Players in round 85 [0.3533]
     108 Players in round 86 [0.3558]
     108 Players in round 87 [0.3578]
     126 Players in round 88 [0.3598]
     108 Players in round 89 [0.3609]
     108 Players in round 90 [0.3626]
     108 Players in round 91 [0.3637]
     108 Players in round 92 [0.3657]
     108 Players in round 93 [0.3687]
     108 Players in round 94 [0.3701]
      90 Players in round 95 [0.3701]
Done, no sequences can be extended
CPU times: user 3min 25s, sys: 138 ms, total: 3min 25s
Wall time: 3min 25s
[0.3701191944101932, 'FCYLSEVAADPTQRQHCDGNLLPQQGPMCGRYPHLMGDRCTYFVLWEWNRRDNLESRRLLPGSHFRVDEPREAPPEQHCLWMGLVVTVCCWLL
M']
1 Candidate residues with 0.3701191944101932 matches
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQGSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN False
```

Why things blow up

1. The search space got large fast
2. There must be a LOT of ties
3. Algorithm tends to keep all (N-k+1) subpeptides as k approaches the sequence's size (k is related to our round)
4. The I/L and K/Q ambiguities lead to exponential number of ties, hence the "hack"
5. Reversed sequences are doubling our leaderboard size

There are bandaids to fix problems 3 and 4, but the problem remains

Other methods for assembling peptide sequences



Database of known peptides

MDERHILNM, KLQWVCSDL, PTYWASDL, ENQIKRSACVM, TLACHGGEM, NGALPQWRT, HLLERTKMNVV, GGPASSDA, GGLITGMQSD, MQPLMNWE, ALKIIMNVRT, **AVGELTK**, HEWAILF, GHNLWAMNAC, GVFGSVLRA, EKLNKAATYIN..

Peptide Identification Problem

Goal: Find a peptide from a database that best matchs the experimental spectrum.

Input:

- S: experimental spectrum
- database of peptides
- Δ: set of possible ion types
- *m*: parent mass

Output:

- A peptide of mass *m* from the database whose theoretical spectrum best matches the experimental spectrum S

Mass Spec Database Searches

How do you get a database?

1. Compute theoretical spectrums for all peptides from length *N* to *M*
2. More commonly, store theoretical spectrums for known peptide sequences

- Database searches are very effective in identfying *known* or *closely related* proteins.
- Experimental spectrums are compared with spectra of database peptides to find the best fit (ex. SEQUEST, Yates et al., 1995)
- But reliable algorithms for identification of new proteins is a more difficult problem.

Essence of the Database Search

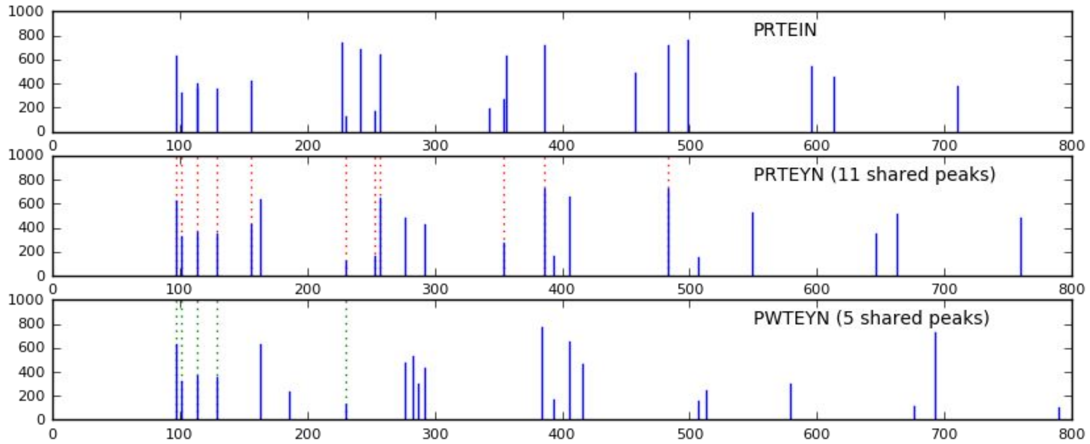
- We need a notion of *spectral similarity* that correlates well with the sequence similarity.
- If peptides are a few mutations/modifications apart, the spectral similarity between their spectra should be high.
- **Simplest measure:** *Shared Peak Counts (SPC)*
    - Very similar to the scoring function used in our *De novo* approach.

SPC Diminishes Quickly

Comparing ‘PRTEIN’ to ‘PRTEYN’ (1 difference) and ‘PWTEYN’ (2 differences)

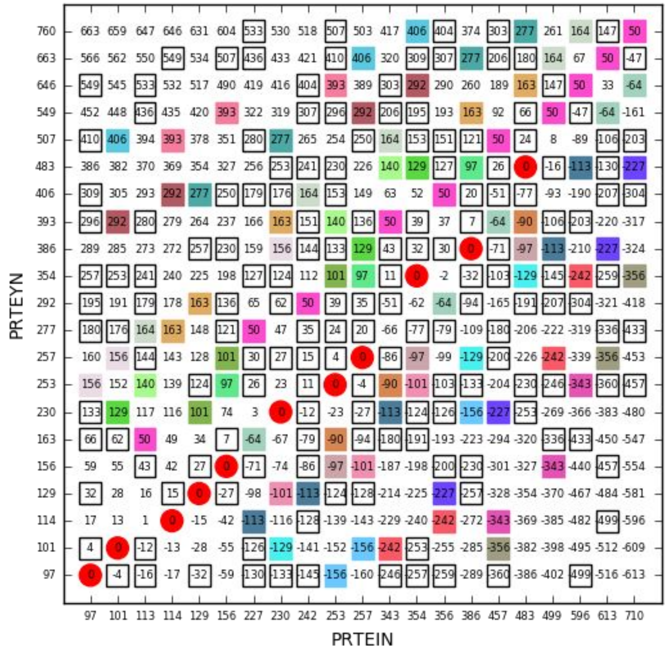
```
In [80]: print(TheoreticalSpectrum('PRTEIN'))
         print(TheoreticalSpectrum('PRTEYN'))
         print(TheoreticalSpectrum('PWTEYN'))

         print(set(TheoreticalSpectrum('PRTEIN')) & set(TheoreticalSpectrum('PRTEYN')))
         print(set(TheoreticalSpectrum('PRTEIN')) & set(TheoreticalSpectrum('PWTEYN')))
```

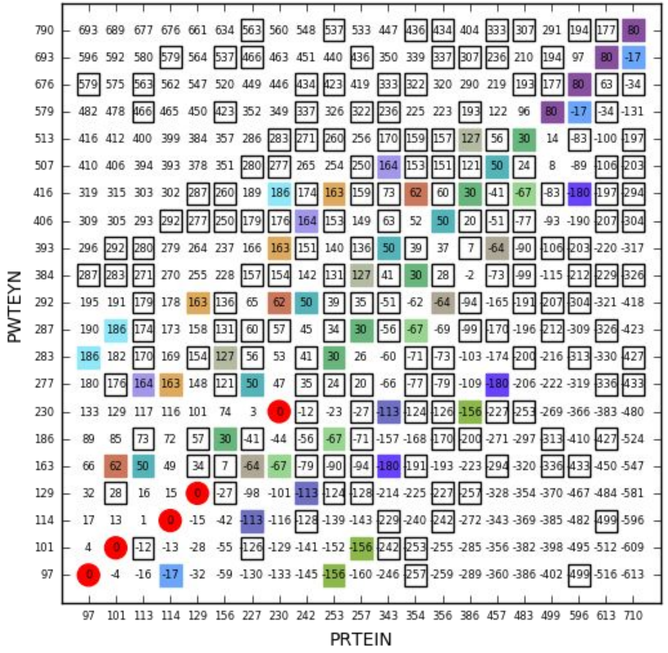
```
[97, 101, 113, 114, 129, 156, 227, 230, 242, 253, 257, 343, 354, 356, 386, 457, 483, 499, 596, 613, 710]
[97, 101, 114, 129, 156, 163, 230, 253, 257, 277, 292, 354, 386, 393, 406, 483, 507, 549, 646, 663, 760]
[97, 101, 114, 129, 163, 186, 230, 277, 283, 287, 292, 384, 393, 406, 416, 507, 513, 579, 676, 693, 790]
{129, 386, 257, 97, 354, 483, 101, 230, 114, 156, 253}
{129, 97, 101, 230, 114}
```



Spectral Convolution to the Rescue!

Difference matrix of spectrums. The elements with multiplicity > 2 are shown in colored boxes. The black outlined boxes enclose elements with multiplicity = 2. The SPC only accounts for the zero entries shown as red circles.

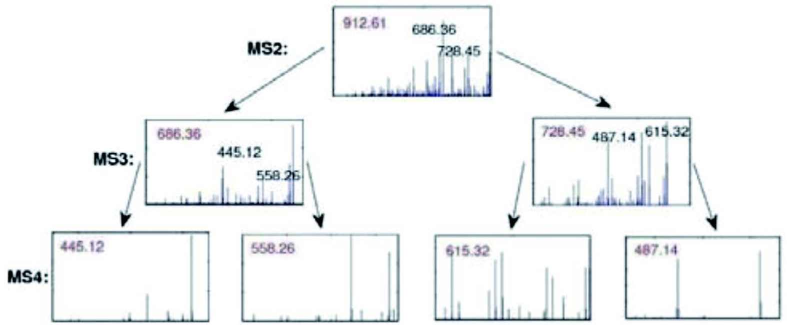


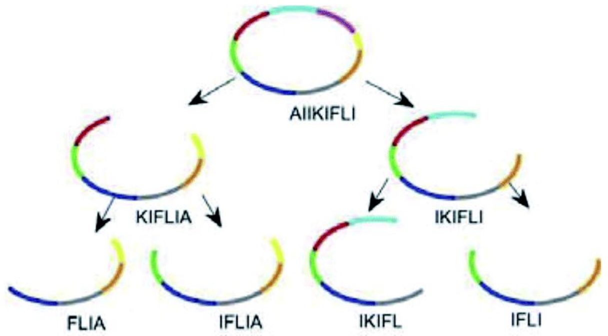


Summary

How do protein structures actually get resolved?

Database searches for protein Mass Specs is generally where most techniques begin. This works paricularly well when it agrees with an already known or very similar protein. However, one can also look for tale-tale fingerprints of peaks from known sub-peptides. For example it is fairly easy to build a library of all 20^6 = 64 million peptides of length 6 and look for eaches 15 associated peaks. Once several hexapeptides are found you can assemble from there. There are also larger subpeptides 10 to 20 in length that appear frequently.





Another common method is to, rather than brake a protein into every possible subpeptide, use an enzyme to cleave it between particular residue pairs. For example, Trypsin will cleave peptide chains immediately after the amino acids lysine and arginine, except when either is followed by proline. This leads to several large fragments, whose mass can be accurately measured using a Mass Spec. This technique is called Peptide Mass Fingerprinting (PMF).