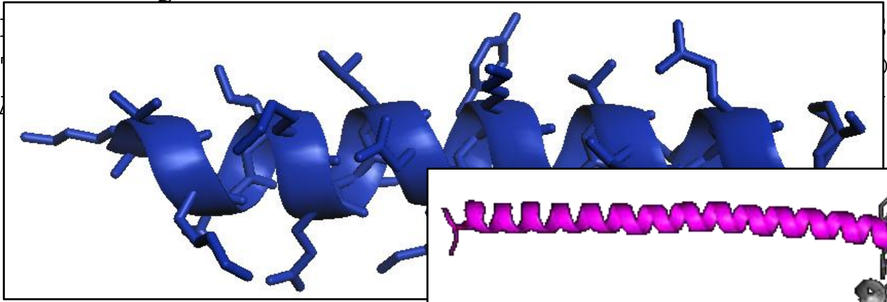
Predicting Protein Secondary Structure

Chou-Fasman Method

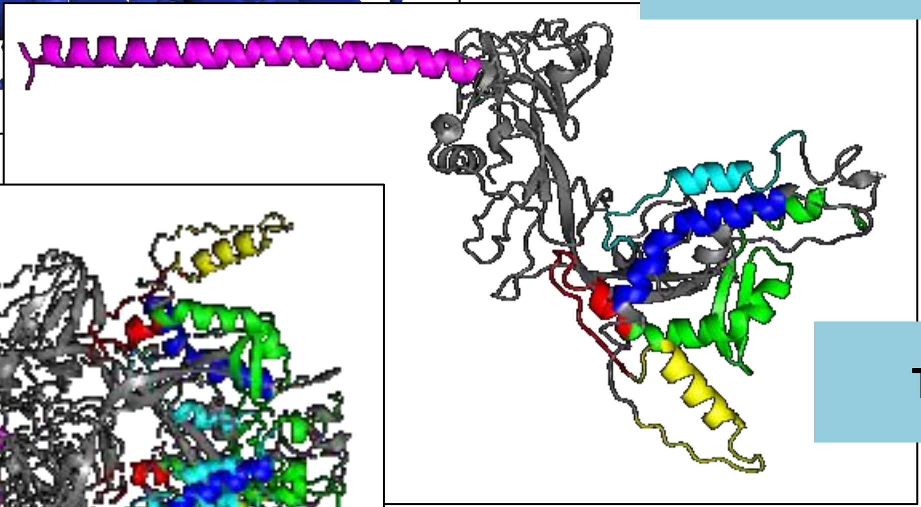
Protein structure

```
>gi|511774874|gb|AGN92848.1| fusion protein [Human respiratory syncytial virus]
MELPILKTNAITTILAAVTLCFASSQNITEEFYQSTCSAVSKGYLSALRTGWYTSVITIELSNIKENKCN
GTDAKVKLIKQELDKYKNAVTELQLLMQSTPAANSRARRELPRFMNYTINNTKNTNVTLSKKRKRRFLGF
LLGVGSAIASGIAVSKVLHLEGEVNKIKSALLSTNKAVVSLSNGVSVLTSKVLDLKNYIDKQLLPIVNKQ
SCSISNIETVIEFQQKNNRLLEITREFSVNAGVTTPVSTYMLTNSELLSLINDMPITNDQKKLMSSNVQI
VRQQSYSIMSIIKEEVLAYVVQLPLYGVIDTPCWKLHTSPLCTTNTKEGSNICLTRTDRGWYCDNAGSVS
FFPQAETCKVQSN                                             LGAIVSCYGKT
KCTASNKNRGIIK                                             PLLFPSDEF
SISQVNEKINQSL                                             CKARSTPVT
```

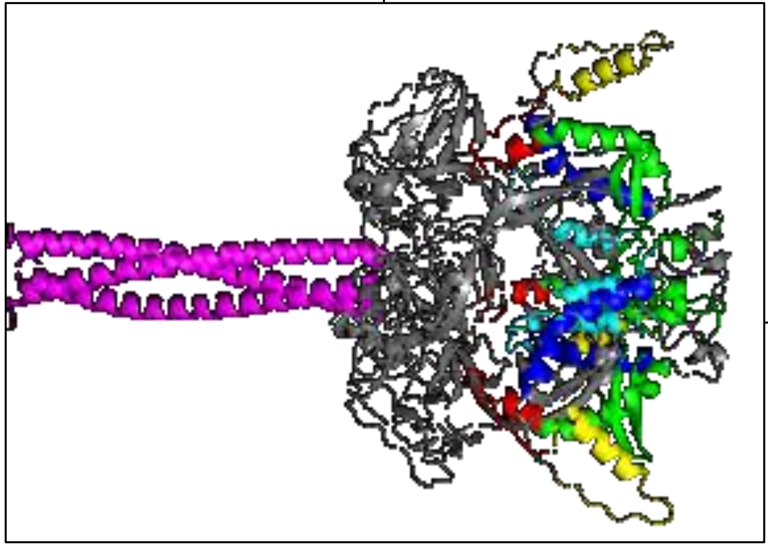
Primary



Secondary



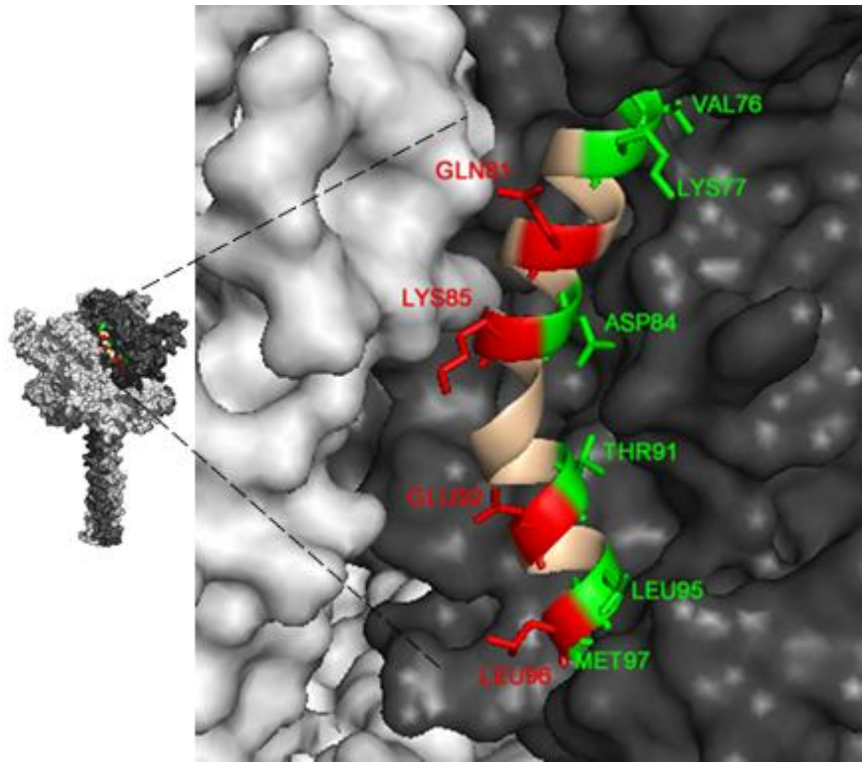
Tertiary



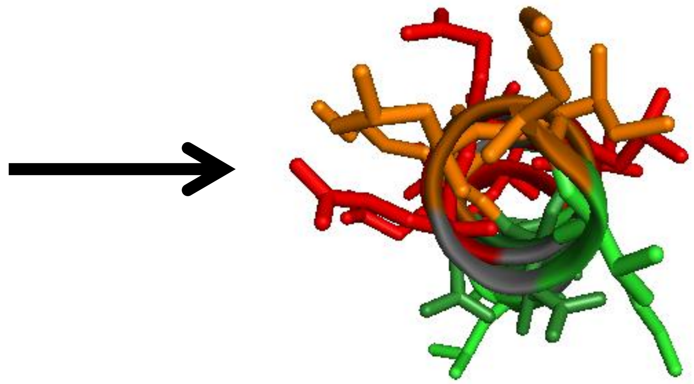
Quaternary

Viral fusion protein

- Involved in virus entry and exit of human cells through fusion of human cell and virus envelope
- Required for pathogenesis



Mutagenesis study of F
-mutate amino acids
-does F still function?

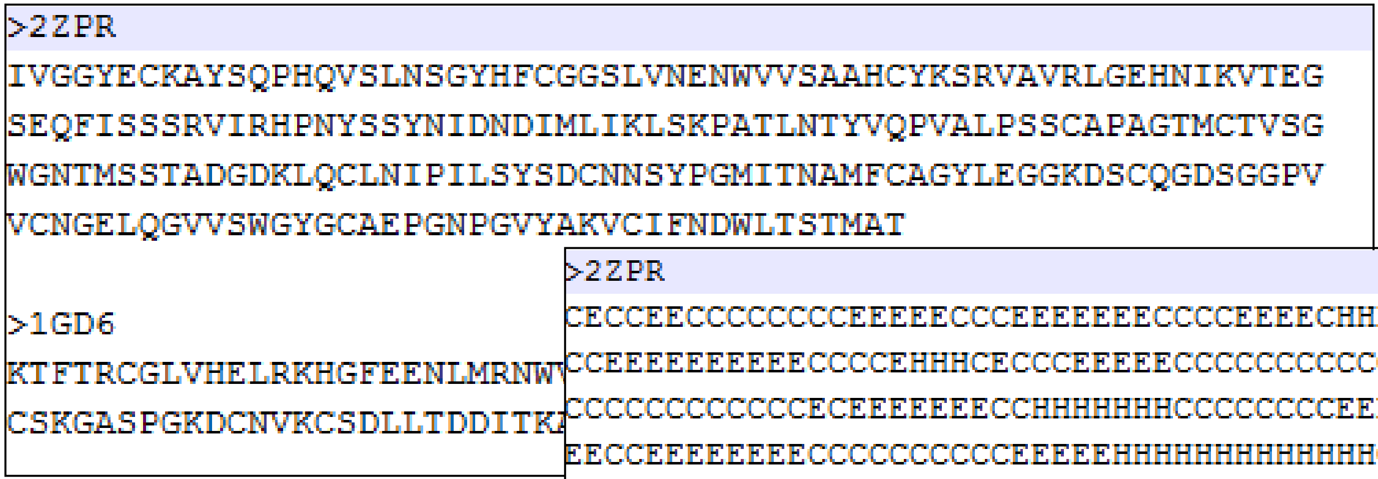


Increased fusion (more infective)
Decreased fusion (less infective)

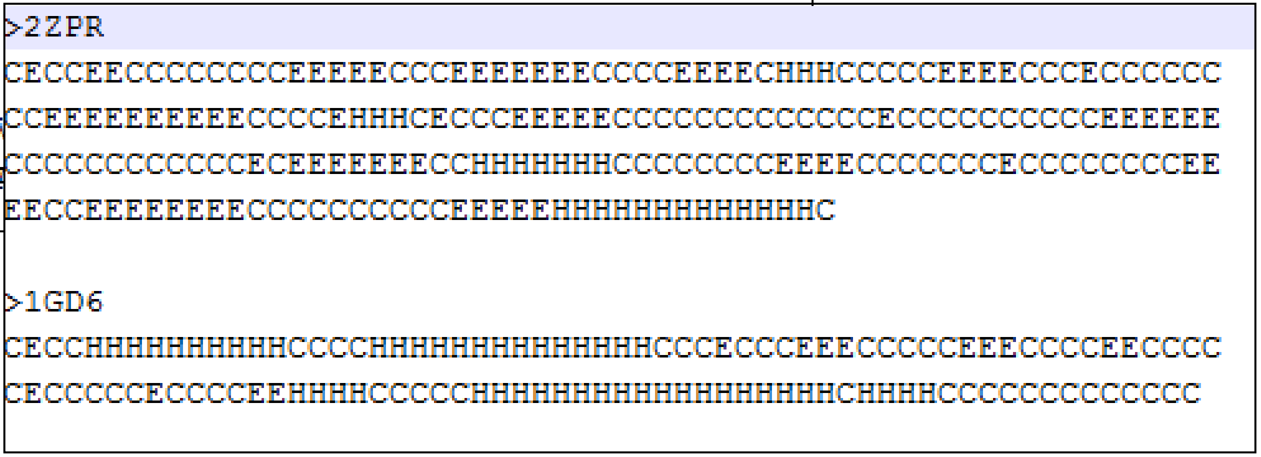
Imogen Bermingham

Secondary structure

- α-helix 'H'
- β-sheet 'E'
- Random coil/turn 'C'



```
>2ZPR
IVGGYECKAYSQPHQVSLNSGYHFCGGSLVNENWVVSAAHCYKSRVAVRLGEHNIKVTEG
SEQFISSSRVIRHPNYSSYNIDNDIMLIKLSKPATLNTYVQPVALPSSCAPAGTMCTVSG
WGNTMSSTADGDKLQCLNIPILSYSDCNNSYPGMITNAMFCAGYLEGGKDSCQGDSGGPV
VCNGELQGVVSWGYGCAEPGNPGVYAKVCIFNDWLTSTMAT

>1GD6
KTFTRCGLVHELRKHGFEENLMRNWV
CSKGASPGKDCNVKCSDLLTDDITKA
```



```
>2ZPR
CECCEECCCCCCCCEEEEECCCEEEEEEECCCCEEEECHHHCCCCCEEEECCCECCCCCC
CCEEEEEEEEEECCCCEHHHCECCCEEEEEECCCCCCCCCCCCCECCCCCCCCCCCEEEEEE
CCCCCCCCCCCCECEEEEEEEECCHHHHHHHCCCCCCCCEEEECCCCCCCECCCCCCCCCEE
EECCEEEEEEEEECCCCCCCCCCEEEEEHHHHHHHHHHHHHC

>1GD6
CECCHHHHHHHHHHCCCCHHHHHHHHHHHHHHHCCCECCCEEECCCCCEEECCCCEECCCC
CECCCCCECCCCEEHHHHCCCCCHHHHHHHHHHHHHHHHHHHCHHHHCCCCCCCCCCCCC
```

Chou-Fasman method

0. Import `sequence.py` and `sstruct.py`

   Read amino acid sequence into python

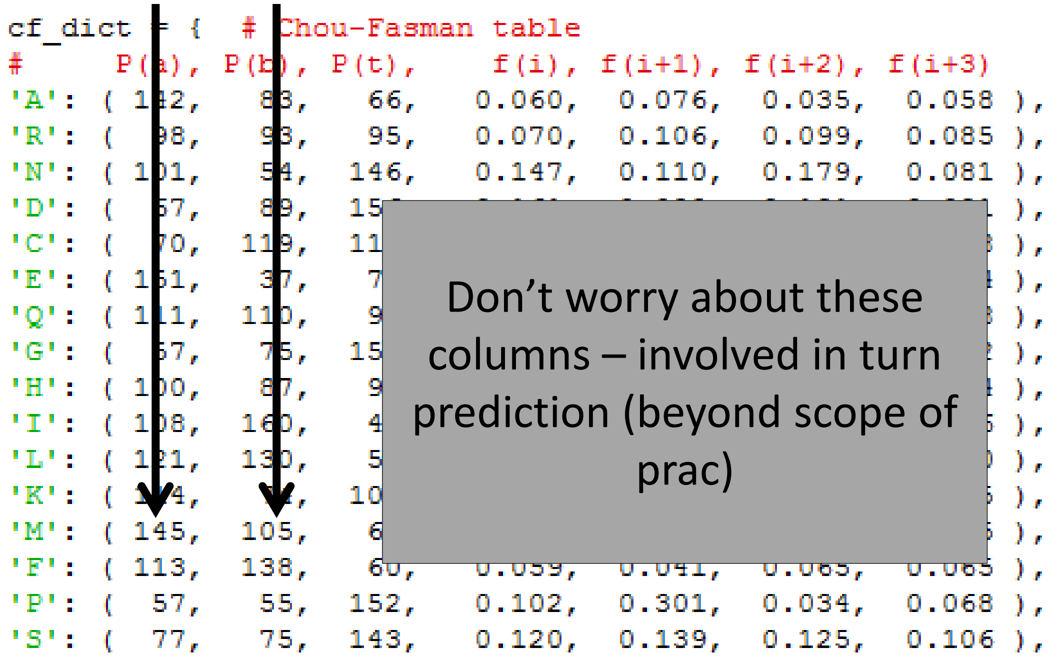
```
prot = Sequence(‘amino acid string', \
symbol.Protein_Alphabet, ‘name')
```

Original article: http://pubs.acs.org/doi/abs/10.1021/bi00699a002

1. Assign probability parameters

- Relative frequencies of 2^{o} structure observation in each amino acid

P(helix) P(β-sheet)



```
cf_dict = {   # Chou-Fasman table
#      P(a), P(b), P(t),    f(i), f(i+1), f(i+2), f(i+3)
'A': ( 142,   83,   66,   0.060,  0.076,  0.035,  0.058 ),
'R': (  98,   93,   95,   0.070,  0.106,  0.099,  0.085 ),
'N': ( 101,   54,  146,   0.147,  0.110,  0.179,  0.081 ),
'D': (  67,   89,  15                                  1 ),
'C': (  70,  119,  11                                  3 ),
'E': ( 151,   37,   7                                  4 ),
'Q': ( 111,  110,   9                                  3 ),
'G': (  57,   75,  15                                  2 ),
'H': ( 100,   87,   9                                  4 ),
'I': ( 108,  160,   4                                  5 ),
'L': ( 121,  130,   5                                  0 ),
'K': ( 114,   74,  10                                  5 ),
'M': ( 145,  105,   6                                  5 ),
'F': ( 113,  138,   60,   0.059,  0.041,  0.065,  0.065 ),
'P': (  57,   55,  152,   0.102,  0.301,  0.034,  0.068 ),
'S': (  77,   75,  143,   0.120,  0.139,  0.125,  0.106 ),
```

Don't worry about these columns – involved in turn prediction (beyond scope of prac)

```
#get scores for each residue
alpha = getScores(prot, 0)
beta  = getScores(prot, 1)
```

Helix prediction

2. Identify regions were 4/6 residues have P(α) >100

```
#find possible alpha helix regions
calls_a1 = markCountAbove(alpha, width = 6, call_cnt = 4)
```

MELPILKTNAITTILAAVTL

sum(P(α))= 300
not helix

Helix prediction

2. Identify regions were 4/6 residues have P(α) >100

```
#find possible alpha helix regions
calls_a1 = markCountAbove(alpha, width = 6, call_cnt = 4)
```

MELPILKTNAITTILAAVTL

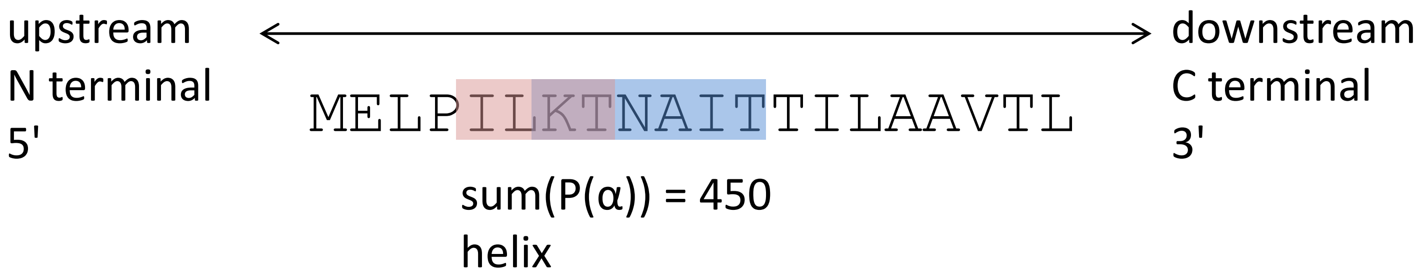
sum(P(α)) = 500
helix

Helix prediction

2. Identify regions were 4/6 residues have P(α) >100

- Extend region (both directions) until 4 residues have P(α) <100

```
#extend helix regions in both directions
calls_a2 = extendDownstream(alpha, calls_a1, width = 4)
calls_a3 = extendUpstream(alpha, calls_a2, width = 4)
```



Helix prediction

2. Identify regions were 4/6 residues have P(α) >100

- Extend region (both directions) until 4 residues have P(α) <100
  - Move to next region and repeat

```
MELPILKTNAITTILAAVTL
```

sum(P(α)) = 350
not helix

Beta-sheet prediction

3. Identify regions were 3/5 residues have P(β) >100

- Extend region (both directions) until 4 residues have P(β) <100
  - Move to next region and repeat

```
# find possible beta-sheets regions, extend in both directions
calls_b1 = markCountAbove(beta, width = 5, call_cnt = 3)
calls_b2 = extendDownstream(beta, calls_b1, width = 4)
calls_b3 = extendUpstream(beta, calls_b2, width = 4)
```

Overlap resolution

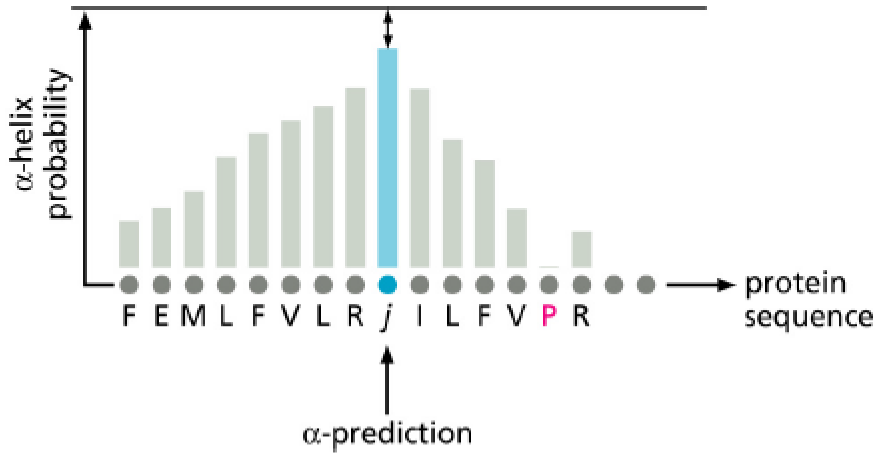
4. if average(P(α)) > average(P(β)) → helix

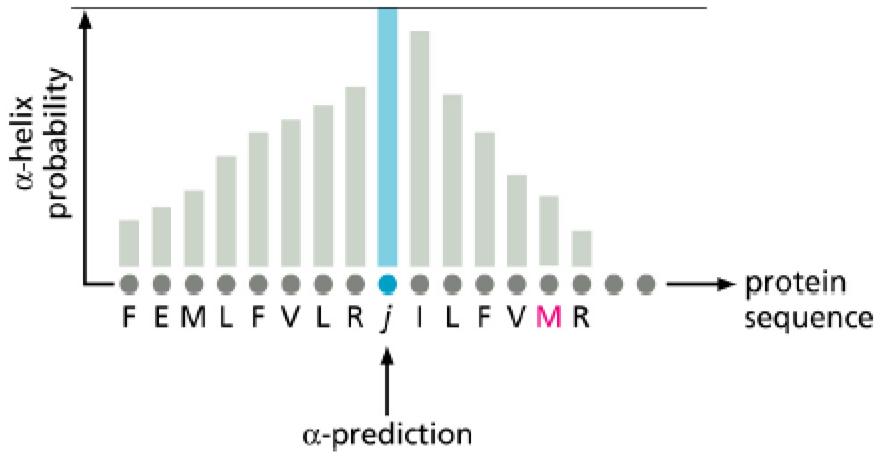
   if average(P(β))) > average(P(α)) → beta-sheet

```
#get average for helix and beta-sheet regions
avg_a = calcRegionAverage(alpha, calls_a3)
avg_b = calcRegionAverage(beta, calls_b3)

#find differences in averages between helix and beta-sheet
diff_a = [avg_a[i] - avg_b[i] for i in range(len(avg_a))]
diff_b = [avg_b[i] - avg_a[i] for i in range(len(avg_a))]

#if diff_a is >0, diff_b must be <0
#therefore i is helix (and vice versa)
calls_a4 = checkSupport(calls_a3, diff_a)
calls_b4 = checkSupport(calls_b3, diff_b)
```

Proline: helix breaker

- Lowers probability of helix formation 5 residues away (**upstream**)
- Often found in N-terminal of helices (i.e. **downstream**)





Q5)

PDB: http://www.rcsb.org/pdb/home/home.do

- Protein ID (e.g. 1EVH) and name
- Amino acid sequence; snip/grab of 3D structure
- Predicted secondary structure using `sstruct.py`
- Describe correlation between prediction and actual
- % accuracy of prediction

No. of positions correctly predicted / No. of positions in sequence

Accuracy calculations

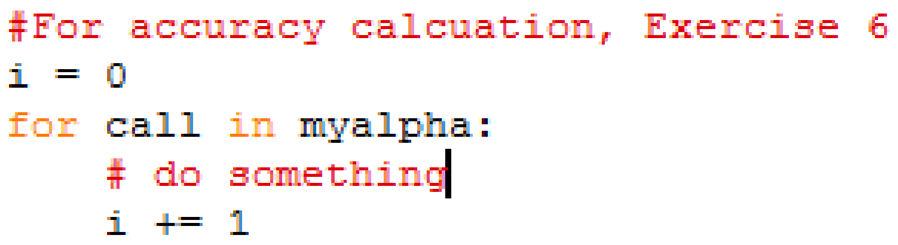
	Actually a helix	Actually not a helix
Predicted helix	True positive	False positive
Predicted not a helix	False negative	True negative

```
# Read some protein sequence data
prot = sequence.readFastaFile('prot2.fa', symbol.Protein_Alphabet)
# read the secondary structure data for the proteins above (indices should agree)
sstr = sequence.readFastaFile('sstr3.fa', symbol.DSSP3_Alphabet)

tp = 0 # number of true positives (correctly identified calls)
tn = 0 # number of true negatives (correctly missed no-calls)
fp = 0 # number of false positives (incorrectly identified no-calls)
fn = 0 # number of false negatives (incorrectly missed calls)

for index in range(len(prot)):

    myprot = prot[index]
    mysstr = sstr[index]
    myalpha = [sym == 'H' for sym in sstr[index]]
    mybeta = [sym == 'E' for sym in sstr[index]]
```

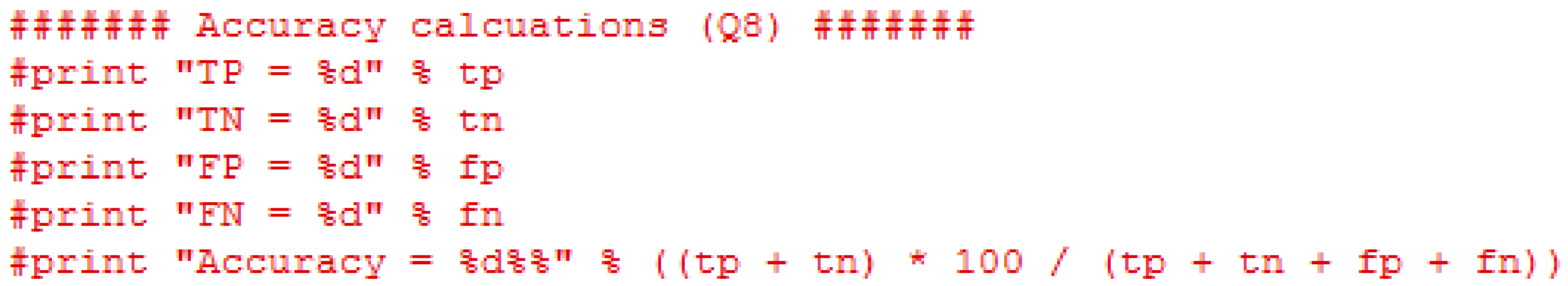
Actual structure at each position in each sequence

Accuracy calculations

- In `sstruct.py`:



```
#For accuracy calcuation, Exercise 6
i = 0
for call in myalpha:
    # do something
    i += 1
```



```
####### Accuracy calcuations (Q8) #######
#print "TP = %d" % tp
#print "TN = %d" % tn
#print "FP = %d" % fp
#print "FN = %d" % fn
#print "Accuracy = %d%%" % ((tp + tn) * 100 / (tp + tn + fp + fn))
```

Accuracy calculations

```
#For accuracy calculation, Exercise 6
i = 0
for call in myalpha:
    if call == True:  #actually helix
        if calls_a4[i] == True:   #predicted helix
            #do something
        else:                     #not predicted helix
            #do something
    else:
        if calls_a4[i] == False:  #actually not a helix
        #do something
        #do something
        #do something
    i += 1

 #repeat for beta-sheet
```

Q7)

- Base-line accuracies for randomly guessing a ‘H’, ‘E’ or ‘C’
- e.g. if you had a bag of letters with 10 H’s, 10 E’s and 10 C’s, what’s the chance you pull out a H? Or a E? Or a C?

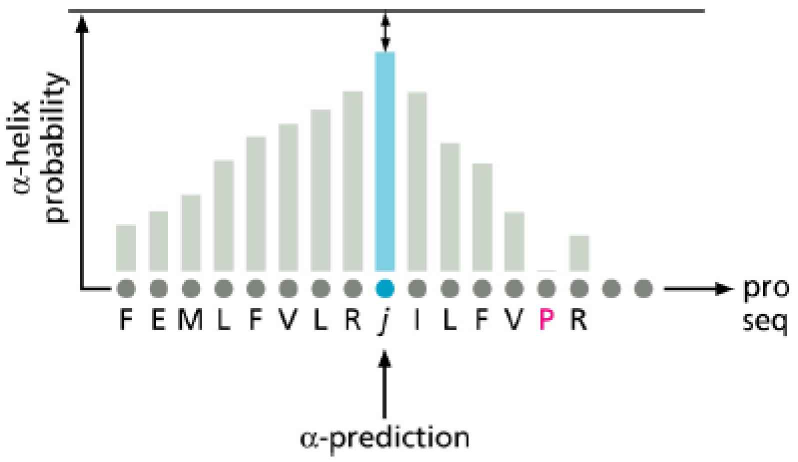
Q8)

- Give accuracy assessment
  - combined α-helices and β-sheets for all sequences in prot2.fa
- I.e. give tp, tn, fp, fn and % accuracy
- Include code for accuracy calculations

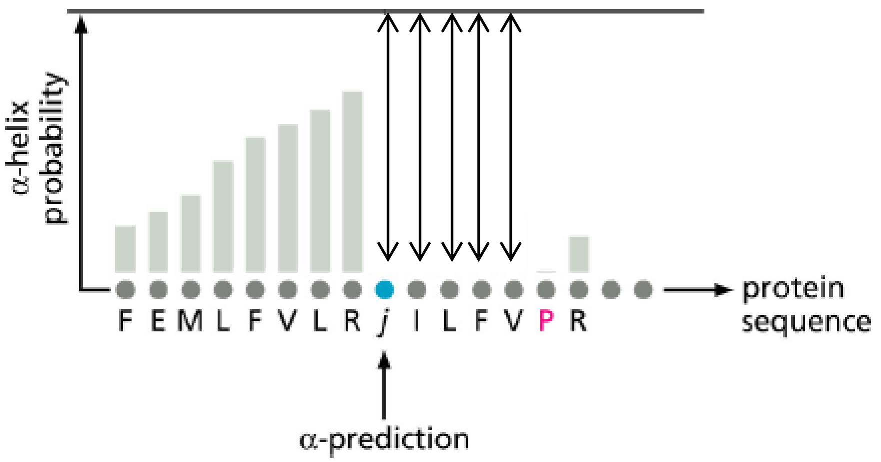
Proline: helix breaker

- When extending helix downstream, check 5 residues ahead for Proline
- Make assumption that Proline does not lower P(α) of that residue, but actually makes it NOT a helix at all – and changes every residue between original and Proline not a helix

In the real world



Assumption for prac8



Proline breaker implementation

- Make list of proline positions

  Hint: modify `myalpha = [sym == 'H' for sym in sstr[index]]` to find ‘P’ in prot

Proline breaker implementation

- Edit `sstruct.py extendDownstream():`
- Create input parameters:
  - flag specifying if we are doing helix or beta-sheet calculation
  - list specifying location of prolines in sequence

Proline breaker implementation

- For each position i
  - check helix calculation
  - check if position i+5 contains a proline
- If both conditions are met:
  - calls[i] = False #make position i not a helix
  - calls[i+1] = False; calls[i+2]=False...etc calls[i+5]=False

    #make position i through i+5 False to break helix
  - cnt = 0; sum = 0.0

    #reset count window and sum of window, and move on to next helix region

Q9)

- Describe proline breaker strategy (and assumptions)
- Provide code implementing proline breaker
- Re-run 2^o structure prediction and accuracy calculations from Q8
- Discuss difference between Q8 accuracy and Q9 accuracy