同濟大學 TONGJI UNIVERSITY

生物信息学系 DEPARTMENT OF BIOINFORMATICS

Section 5

Python: Analyzing a Data Column Parsing Data Records Searching Data

Yong Zhang, Ph. D

School of Life Science and Technology
Tongji University

Apr 4, 2019

yzhang@tongji.edu.cn

Story: Dendritic lengths

- The length of neurons is in a single column file.

```
 16.38
139.90
441.46
 29.03
 40.93
202.07
142.30
346.00
300.00
```

Example Python session

```
data = []

for line in open('neuron_data.txt'):
    length = float(line.strip())
    data.append(length)

n_items = len(data)
total = sum(data)
shortest = min(data)
longest = max(data)

data.sort()

output = open("results.txt","w")
output.write("number of dendritic lengths : %4i \n"%(n_items))
output.write("total dendritic length      : %6.1f \n"%(total))
output.write("shortest dendritic length   : %7.2f \n"%(shortest))
output.write("longest dendritic length    : %7.2f \n"%(longest))
output.write("%37.2f\n%37.2f"%(data[-2], data[-3]))
output.close()
```

Example Python session

- Running results

```
number of dendritic lengths :    9
total dendritic length      : 1658.1
shortest dendritic length   :   16.38
longest dendritic length    :  441.46
                               346.00
                               300.00
```

Reading text files

```
data = []

for line in open('neuron_data.txt'):
    length = float(line.strip())
    data.append(length)

n_items = len(data)
total = sum(data)
shortest = min(data)
longest = max(data)

data.sort()

output = open("results.txt","w")
output.write("number of dendritic lengths : %4i \n"%(n_items))
output.write("total dendritic length      : %6.1f \n"%(total))
output.write("shortest dendritic length   : %7.2f \n"%(shortest))
output.write("longest dendritic length    : %7.2f \n"%(longest))
output.write("%37.2f\n%37.2f"%(data[-2], data[-3]))
output.close()
```

Reading text files

- You can read the entire body of data by three Python commands:

```
text_file = open('neuron_data.txt')
lines = text_file.readlines()
text_file.close()
```

  - 1. *Opens a text file*. You would need to add the directory path before the filename, if it isn't in the same directory as you runs Python program.
  - 2. *Reads information from the file*. The *readlines()* function simply reads everything that is in the file line by line and stores each line in a separate string. The strings are returned as a list of strings. In contrast, *read()* reads the entire file into a single string.
  - 3. *Closes the text file*.

Writing Text Files

```
data = []

for line in open('neuron_data.txt'):
    length = float(line.strip())
    data.append(length)

n_items = len(data)
total = sum(data)
shortest = min(data)
longest = max(data)

data.sort()

output = open("results.txt","w")
output.write("number of dendritic lengths : %4i \n"%(n_items))
output.write("total dendritic length      : %6.1f \n"%(total))
output.write("shortest dendritic length   : %7.2f \n"%(shortest))
output.write("longest dendritic length    : %7.2f \n"%(longest))
output.write("%37.2f\n%37.2f"%(data[-2], data[-3]))
output.close()
```

Writing Text Files

- A file is written by three Python commands:

```
output_file = open('counts.txt', 'w')
output_file.write('number of neuron lengths: 7\n')
output_file.close()
```

  – *Opens a text file for writing.* A file opened with the 'w' flag can be used only for writing.
  – *Writes a string to the file.* The *write()* function accepts only string data. As *write()* does not introduce line breaks automatically, so you have to add them explicitly if you need them. Alternatively, the *writelines()* function accepts a list of lines (each in the form of a string).
  – *Closes the file after usage.*

List data structures

```
data = []

for line in open('neuron_data.txt'):
    length = float(line.strip())
    data.append(length)

n_items = len(data)
total = sum(data)
shortest = min(data)
longest = max(data)

data.sort()

output = open("results.txt","w")
output.write("number of dendritic lengths : %4i \n"%(n_items))
output.write("total dendritic length      : %6.1f \n"%(total))
output.write("shortest dendritic length   : %7.2f \n"%(shortest))
output.write("longest dendritic length    : %7.2f \n"%(longest))
output.write("%37.2f\n%37.2f"%(data[-2], data[-3]))
output.close()
```

List data structures

- A list is an ordered mutable set of objects enclosed in square brackets.
- The elements of a list can be any kind of object (numbers, strings, tuples, other lists, dictionaries, sets, or even functions) or a blend of different objects.

```
>>> d1 = []
>>> d2 = [1, 2, 5, -9]
>>> d3 = [1, "hello", 12.1, [1, 2, "three"], "seq", (1, 2)]
```

List data structures

- Indexing and slicing.

```
>>> d3[0]
1
>>> d3[2:]
[12.1, [1, 2, 'three'], 'seq', (1, 2)]
>>> d2[-1]
-9
```

- Two or more brackets are possible.

```
>>> d3[3][2]
'three'
>>> d3[3][2][0]
't'
```

List data structures

- Lists can be modified.

```
>>> data = [0,1,2,3,4]
>>> data[0] = 'A'
>>> data
['A', 1, 2, 3, 4]
>>> data = [0,1,2,3,4]
>>> data.append(5)
>>> data
[0, 1, 2, 3, 4, 5]
```

- The method *append()* adds the item in brackets to the end of the list.
- *append()* is a method of the list object (when a function refers to a specific object, it is called a *method* of that object).

List data structures

- Creating a list of consecutive numbers

```
>>> range(3)
[0, 1, 2]
```

- Creating a list of zeroes

```
>>> data = [0.0] * 10
>>> data
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

- List comprehension

```
>>> data = [x**2 for x in range(5)]
>>> data
[0, 1, 4, 9, 16]
>>>
>>> bases = ['A', 'C', 'T', 'G']
>>> seq = 'GGACXCAGXXGATT'
>>> seqlist = [base for base in seq if base in bases]
>>> seqlist
['G', 'G', 'A', 'C', 'C', 'A', 'G', 'G', 'A', 'T', 'T']
```

Tuples

- Tuples are immutable ordered sequences of objects and are indicated with round brackets, *(a, b, c)*, or by simply listing the sequence of items separated by commas: *a, b, c*.
- Once you have defined a tuple, you cannot change or replace its elements.
- A tuple of a single item must be written either *data = (1,)* or *data = 1, .*

```
>>> my_tuple = (1,2,3)
>>> my_tuple[0]          #indexing
1
>>> my_tuple[2:]         #slicing
(3,)
>>> my_tuple[0] = 0      #reassigning (Forbidden)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Converting text to numbers

```
data = []

for line in open('neuron_data.txt'):
    length = float(line.strip())
    data.append(length)

n_items = len(data)
total = sum(data)
shortest = min(data)
longest = max(data)

data.sort()

output = open("results.txt","w")
output.write("number of dendritic lengths : %4i \n"%(n_items))
output.write("total dendritic length      : %6.1f \n"%(total))
output.write("shortest dendritic length   : %7.2f \n"%(shortest))
output.write("longest dendritic length    : %7.2f \n"%(longest))
output.write("%37.2f\n%37.2f"%(data[-2], data[-3]))
output.close()
```

Converting text to numbers

```
>>> number = float('100.12') + 100.0
>>> number
200.12
>>> number = '100.34' + '100.0'
>>> number
'100.34100.0'
>>> number = int(100.45)
>>> number
100
>>> f_number = float(number)
>>> f_number
100.0
```

Converting numbers to text

```
data = []

for line in open('neuron_data.txt'):
    length = float(line.strip())
    data.append(length)

n_items = len(data)
total = sum(data)
shortest = min(data)
longest = max(data)

data.sort()

output = open("results.txt","w")
output.write("number of dendritic lengths : %4i \n"%(n_items))
output.write("total dendritic length      : %6.1f \n"%(total))
output.write("shortest dendritic length   : %7.2f \n"%(shortest))
output.write("longest dendritic length    : %7.2f \n"%(longest))
output.write("%37.2f\n%37.2f"%(data[-2], data[-3]))
output.close()
```

Converting numbers to text

- Both integer and float numbers can be converted to a string using *str()* function for the conversion:

```
>>> text = str(number)
>>> text
'100'
```

- String formatting:

```
>>> 'Result:%3i' % (17)
'Result: 17'
```

  – The *%3i* indicates that the string should contain an integer formatted to three positions.

String formatting

- You can insert floating-point numbers into a string with *%x.yf*, where x is the number of total characters and y is the number of decimal places.

```
>>> '%8.3f' % (12.3456)
' 12.346'
```

- You can also format strings with %s:

```
>>> name = 'E.coli'
>>> 'Hello, %s' % (name)
'Hello, E.coli'
```

- You can right-justify the string by, e.g., *%10s* or left-justify it by *%-10s*:

```
>>> 'text:%25s numbers:%4i%4i%5.2f' % ('right-justified', 1, 2, 3)
'text:          right-justified numbers:   1   2 3.00'
```

Writing a data column to a text file

- If your result is a list of numbers, you can format them to a list of strings and then pass the list to the *writelines()* method of file objects:

```
data = [16.38, 139.90, 441.46, 29.03, 40.93, 202.07, 142.30, 346.00, 300.00]

out = []
for value in data:
    out.append(str(value) + '\n')
open('results.txt', 'w').writelines(out)
```

- You can format the result as a long single string:

```
out = []
for value in data:
    out.append(str(value))
out = '\n'.join(out)
open('results.txt', 'w').write(out)
```

  – The *join()* function connects all values to a single string.

Calculations on a list of numbers

```
data = []

for line in open('neuron_data.txt'):
    length = float(line.strip())
    data.append(length)

n_items = len(data)
total = sum(data)
shortest = min(data)
longest = max(data)

data.sort()

output = open("results.txt","w")
output.write("number of dendritic lengths : %4i \n"%(n_items))
output.write("total dendritic length      : %6.1f \n"%(total))
output.write("shortest dendritic length   : %7.2f \n"%(shortest))
output.write("longest dendritic length    : %7.2f \n"%(longest))
output.write("%37.2f\n%37.2f"%(data[-2], data[-3]))
output.close()
```

Calculations on a list of numbers

```
>>> data = [16.38, 139.90, 441.46, 29.03, 40.93, 202.07, 142.30, \
... 346.00, 300.00]
>>> len(data)
9
>>> max(data)
441.46
>>> min(data)
16.38
>>> sum(data)
1658.07
```

Examples

- How to calculate a mean value?
  – Calculate average from float numbers.

```
data = [3.53, 3.47, 3.51, 3.72, 3.43]
average = sum(data) / len(data)
print average
```

  – Calculate average from integer numbers.

```
data = [1, 2, 3, 4]
average = float(sum(data)) / len(data)
print average
```

Examples

- How to calculate a standard deviation?

```
import math

data = [3.53, 3.47, 3.51, 3.72, 3.43]
average = sum(data) / len(data)

total = 0.0
for value in data:
    total += (value - average) ** 2
stddev = math.sqrt(total / len(data))

print stddev
```

Examples

- How to calculate a median value?

```
data = [3.53, 3.47, 3.51, 3.72, 3.43]

data.sort()
mid = len(data) / 2
if len(data) % 2 == 0:
    median = (data[mid - 1] + data[mid]) / 2.0
else:
    median = data[mid]

print median
```

Story: Integrating mass spectrometry data into metabolic pathways

- Mass spectrometry (MS) is a technique used to determine the elemental composition of a sample of molecules. Example of MS report file:

```
P43686
P62333
```

- Example of text file containing a list of proteins participating in a pathway:

```
P62258
P61981
P62191
P17980
P43686
P35998
P62333
Q99460
O75832
```

Example Python session

```
# proteins participating in cell cycle
list_a = []
for line in open("cell_cycle_proteins.txt"):
    list_a.append(line.strip())
print list_a

# proteins expressed in a given cancer cell
list_b = []
for line in open("cancer_cell_proteins.txt"):
    list_b.append(line.strip())
print list_b

for protein in list_a:
    if protein in list_b:
        print protein, 'detected in the cancer cell'
    else:
        print protein, 'not observed'
```

Output of the script

```
['P62258', 'P61981', 'P62191', 'P17980', 'P43686', 'P35998', 'P62333',
 'Q99460', 'O75832']
['P43686', 'P62333']
P62258 not observed
P61981 not observed
P62191 not observed
P17980 not observed
P43686 detected in the cancer cell
P35998 not observed
P62333 detected in the cancer cell
Q99460 not observed
O75832 not observed
```

The *if/elif/else* statements

```
# proteins participating in cell cycle
list_a = []
for line in open("cell_cycle_proteins.txt"):
    list_a.append(line.strip())
print list_a

# proteins expressed in a given cancer cell
list_b = []
for line in open("cancer_cell_proteins.txt"):
    list_b.append(line.strip())
print list_b

for protein in list_a:
    if protein in list_b:
        print protein, 'detected in the cancer cell'
    else:
        print protein, 'not observed'
```

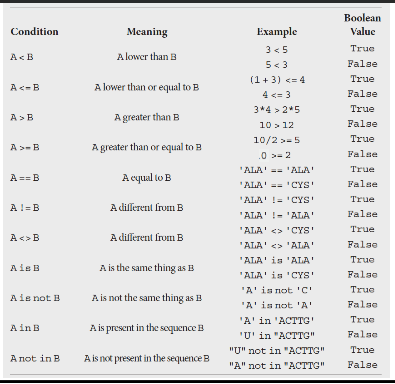
The *if/elif/else* statements

- The structure is the following:

```
if <condition 1>:
        <statements 1>
[elif <condition 2>]:
        <statements 2>]
[elif <condition 3>]:
        pass]
…
[else:
        <statements N>]
```

- The *elif* and *else* statements and the corresponding blocks of instructions are optional.

Operators used in *if* conditions



Condition	Meaning	Example	Boolean Value
A < B	A lower than B	3 < 5	True
		5 < 3	False
A <= B	A lower than or equal to B	(1 + 3) <= 4	True
		4 <= 3	False
A > B	A greater than B	3*4 > 2*5	True
		10 > 12	False
A >= B	A greater than or equal to B	10/2 >= 5	True
		0 >= 2	False
A == B	A equal to B	'ALA' == 'ALA'	True
		'ALA' == 'CYS'	False
A != B	A different from B	'ALA' != 'CYS'	True
		'ALA' != 'ALA'	False
A <> B	A different from B	'ALA' <> 'CYS'	True
		'ALA' <> 'ALA'	False
A is B	A is the same thing as B	'ALA' is 'ALA'	True
		'ALA' is 'CYS'	False
A is not B	A is not the same thing as B	'A' is not 'C'	True
		'A' is not 'A'	False
A in B	A is present in the sequence B	'A' in 'ACTTG'	True
		'U' in "ACTTG"	False
A not in B	A is not present in the sequence B	"U" not in "ACTTG"	True
		"A" not in "ACTTG"	False

Boolean operators *and*, *not*, *or*

```
seq = "MGSNKSKPKDASQRRRSLEPAENVHGAGGGAFPASQTPSKPASADGHRGPSAAFAPAAAE"

if 'GGG' in seq and 'RRR'in seq:
    print 'GGG is at position: ', seq.find('GGG')
    print 'RRR is at position: ', seq.find('RRR')

if 'WWW' in seq or 'AAA' in seq:
    print 'Either WWW or AAA occur in the sequence'

if 'AAA' in seq and not 'PPP' in seq:
    print 'AAA occurs in the sequence but not PPP'
```

The *elif* statement

```
for i in range(30):
    if i < 4:
        print "prime number:", i
    elif i % 2 == 0:
        print "multiple of two:", i
    elif i % 3 == 0:
        print "multiple of three:", i
    elif i % 5 == 0:
        print "multiple of five:", i
    else:
        print "prime number:", i
```

Examples

- Read a sequence file in FASTA format and write only the sequence header to a new file.

```
>sp|P31946|1433B_HUMAN 14-3-3 protein beta/alpha OS=Homo sapiens GN=YWHAB PE=1 SV=3
MTMDKSELVQKAKLAEQAERYDDMAAAMKAVTEQGHELSNEERNLLSVAYKNVVGARRSS
WRVISSIEQKTERNEKKQQMGKEYREKIEAELQDICNDVLELLDKYLIPNATQPESKVFY
LKMKGDYFRYLSEVASGDNKQTTVSNSQQAYQEAFEISKKEMQPTHPIRLGLALNFSVFY
YEILNSPEKACSLAKTAFDEAIAELDTLNEESYKDSTLIMQLLRDNLTLWTSENQGDEGD
AGEGEN
>sp|P62258|1433E_HUMAN 14-3-3 protein epsilon OS=Homo sapiens GN=YWHAE PE=1 SV=1
MDDREDLVYQAKLAEQAERYDEMVESMKKVAGMDVELTVEERNLLSVAYKNVIGARRASW
RIISSIEQKEENKGGEDKLKMIREYRQMVETELKLICCDILDVLDKHLIPAANTGESKVF
YYKMKGDYHRYLAEFATGNDRKEAAENSLVAYKAASDIAMTELPPTHPIRLGLALNFSVF
YYEILNSPDRACRLAKAAFDDAIAELDTLSEESYKDSTLIMQLLRDNLTLWTSDMQGDGE
EQNKEALQDVEDENQ
```

Examples

```
fasta_file = open('SwissProt.fasta','r')
out_file = open('SwissProt.header','w')

for line in fasta_file:
    if line[0:1] == '>':
        out_file.write(line)
out_file.close()
```

Examples

- How to extract a list of accession codes (AC) from a multiple sequence FASTA file?

```
>sp|P31946|1433B_HUMAN 14-3-3 protein beta/alpha OS=Homo sapiens GN=YWHAB PE=1 SV=3
MTMDKSELVQKAKLAEQAERYDDMAAAMKAVTEQGHELSNEERNLLSVAYKNVVGARRSS
WRVISSIEQKTERNEKKQQMGKEYREKIEAELQDICNDVLELLDKYLIPNATQPESKVFY
LKMKGDYFRYLSEVASGDNKQTTVSNSQQAYQEAFEISKKEMQPTHPIRLGLALNFSVFY
YEILNSPEKACSLAKTAFDEAIAELDTLNEESYKDSTLIMQLLRDNLTLWTSENQGDEGD
AGEGEN
>sp|P62258|1433E_HUMAN 14-3-3 protein epsilon OS=Homo sapiens GN=YWHAE PE=1 SV=1
MDDREDLVYQAKLAEQAERYDEMVESMKKVAGMDVELTVEERNLLSVAYKNVIGARRASW
RIISSIEQKEENKGGEDKLKMIREYRQMVETELKLICCDILDVLDKHLIPAANTGESKVF
YYKMKGDYHRYLAEFATGNDRKEAAENSLVAYKAASDIAMTELPPTHPIRLGLALNFSVF
YYEILNSPDRACRLAKAAFDDAIAELDTLSEESYKDSTLIMQLLRDNLTLWTSDMQGDGE
EQNKEALQDVEDENQ
```

Examples

```
input_file = open("SwissProtSeq.fasta","r")
ac_list = []

for line in input_file:
    if line[0] == '>':
        fields = line.split('|')
        ac_list.append(fields[1])

print ac_list
```

- The *split()* method of string objects is used to cut the string into pieces, and returns a Python list.

Examples

- How to parse GenBank sequence records?

```
LOCUS       AY810830                 705 bp    mRNA    linear   HTC 22-JUN-2006
DEFINITION  Schistosoma japonicum SJCHGC07869 protein mRNA, partial cds.
ACCESSION   AY810830
VERSION     AY810830.1  GI:60600350
KEYWORDS    HTC.
SOURCE      Schistosoma japonicum
  ORGANISM  Schistosoma japonicum
. . .

ORIGIN
        1 ctcatgttga atctgataaa gttcctgtag catctattca tgcaacattg aatggtccgg
       61 gaagtatccg tattacgtgg tctaatccag tcaaacctaa tggtttaatt atacattatt
      121 tattgcggta tagaccaagg aatcatgatc agagttatac agatagtaac cattcgtctt
      181 cagatgtgtc gctgccatgg ttgacaaaat gtatttcgat gagtcattgg tcggctgacc
      241 attctgaaca cgcattgact tcaagttcat atatagctat taatcaaaaa gaagtatcac
      301 gaagtaaacg tggttataat gctaatagta gtactactga tggcggaatc tcaattaaag
      361 atttatcacc aggtagctat gaatttcaaa ttttagccgt ttctcttgct ggtaacggag
      421 aatggagtcc aaccgtaata ttcaatattc cattctatac agaccataat ggcacaataa
      481 accgtatgtt tatagaactc ttattattta cagtttgtgt cccatgtatg ccgcatcacg
      541 tgtaatgttt tgattaagga gattcaaatt ttatacgttc tctcataagt gatctttact
      601 tttaattgtg tgctctaaga atatacgcat tttcggttca atagattcta aaacaatgca
      661 attatgagtt agatttcatt aatgcatatg taagctaatt ttcta
//
```

Examples

```
genbank_file = open("AY810830.gb")
output_file = open("AY810830.fasta","w")

flag = False
for line in genbank_file:
    if line[0:9] == 'ACCESSION':
        accession = line.split()[1].strip()
        output_file.write('>' + accession + '\n')
    if line[0:6] == 'ORIGIN':
        flag = True
    elif flag:
        fields = line.split()
        if fields != []:
            seq = ''.join(fields[1:])
            output_file.write(seq.upper() + '\n')

genbank_file.close()
output_file.close()
```

- The *strip()* method of strings erases blank spaces before and after a string of characters.

Examples

- Read a multiple sequence file in FASTA format and write records from Homo sapiens to a new file.

```
fasta_file = open('SwissProt.fasta','r')
out_file = open('SwissProtHuman.fasta','w')

seq = ''
for line in fasta_file:
    if line[0] == '>' and seq == '':
        # process the first line of the input file
        header = line
    elif line [0] != '>':
        # join the lines with sequence
        seq = seq + line
    elif line[0] == '>' and seq != '':
        # in subsequent lines starting with '>',
        # write the previous header and sequence
        # to the output file. Then re-initialize
        # the header and seq variables for the next record
        if "Homo sapiens" in header:
            out_file.write(header + seq)
        seq = ''
        header = line

# take care of the very last record of the input file
if "Homo sapiens" in header:
    out_file.write(header + seq)
out_file.close()
```

Story: Translating an RNA into protein

```
>A06662.1 Synthetic nucleotide sequence of the human GSH transferase pi gene
UGGGACCAGUCAGCAGAGGCAGCGUGUGUGCGCGUGCGUGUGCGUGUGUGUGCGUGUGUGUGUG
UACGCUUGCAUUUGUGUCGGGUGGGUAAGGAGAUAGAGAUGGGCGGGCAGUAGGCCCAGGUCCC
GAAGGCCUUGAACCCACUGGUUUGGAGUCUCCUAAGGGCAAUGGGGGCCAUUGAGAAGUCUGAA
CAGGGCUGUGUCUGAAUGUGAGGUCUAGAAGGAUCCUCCAGAGAAGCCAGCUCUAAAGCUUUUG
CAAUCAUCUGGUGAGAGAACCCAGCAAGGAUGGACAGGCAGAAUGGAAUAGAGAUGAGUUGGCA
GCUGAAGUGGACAGGAUUUGGUACUAGCCUGGUUGUGGGGAGCAAGCAGAGGAGAAUCUGGGAC
UCUGGUGGUCUGGCCUGGGGCAGACGGGGGUGUCUCAGGGGCUGGGAGGGAUGAGAGUAGGAUG
AUACAUGGUGGUGUCUGGCAGGAGGCGGGCAAGGAUGACUAUGUGAAGGCACUGCCCGGGCAAC
UGAAGCCUUUUGAGACCCUGCUGUCCCAGAACCAGGGAGGCAAGACCUUCAUUGUGGGAGACCA
GGUGAGCAUCUGGCC
```

Example Python session

```
codon_table = {
    'GCU':'A', 'GCC':'A', 'GCA':'A', 'GCG':'A', 'CGU':'R', 'CGC':'R',
    'CGA':'R', 'CGG':'R', 'AGA':'R', 'AGG':'R', 'UCU':'S', 'UCC':'S',
    'UCA':'S', 'UCG':'S', 'AGU':'S', 'AGC':'S', 'AUU':'I', 'AUC':'I',
    'AUA':'I', 'UUA':'L', 'UUG':'L', 'CUU':'L', 'CUC':'L', 'CUA':'L',
    'CUG':'L', 'GGU':'G', 'GGC':'G', 'GGA':'G', 'GGG':'G', 'GUU':'V',
    'GUC':'V', 'GUA':'V', 'GUG':'V', 'ACU':'T', 'ACC':'T', 'ACA':'T',
    'ACG':'T', 'CCU':'P', 'CCC':'P', 'CCA':'P', 'CCG':'P', 'AAU':'N',
    'AAC':'N', 'GAU':'D', 'GAC':'D', 'UGU':'C', 'UGC':'C', 'CAA':'Q',
    'CAG':'Q', 'GAA':'E', 'GAG':'E', 'CAU':'H', 'CAC':'H', 'AAA':'K',
    'AAG':'K', 'UUU':'F', 'UUC':'F', 'UAU':'Y', 'UAC':'Y', 'AUG':'M',
    'UGG':'W',
    'UAG':'STOP', 'UGA':'STOP', 'UAA':'STOP'
    }
```

Example Python session

```
# read the RNA sequence into a single string
rna = ''
for line in open('A06662-RNA.fasta'):
    if not line.startswith('>'):
        rna = rna + line.strip()

# translate one frame at a time
for frame in range(3):
    prot = ''
    print 'Reading frame ' + str(frame + 1)
    for i in range(frame, len(rna), 3):
        codon = rna[i:i + 3]
        if codon in codon_table:
            if codon_table[codon] == 'STOP':
                prot = prot + '*'
            else:
                prot = prot + codon_table[codon]
        else:
            # handle too short codons
            prot = prot + '-'

    # format to blocks of 48 columns
    i = 0
    while i < len(prot):
        print prot[i:i + 48]
        i = i + 48
```

Running output

```
Reading frame 1
WDQSAEAACVRVRVRVCACVCVRLHLCRVGKEIEMGGQ*AQVPKALNP
LVWSLLRAMGAIEKSEQGCV*M*GLEGSSREASSKAFAIIW*ENPARM
DRQNGIEMSWQLKWTGFGTSLVVGSKQRRIWDSGGLAWGRRGCLRGWE
G*E*DDTWWCLAGGGQG*LCEGTARATEAF*DPAVPEPGRQDLHCGRP
GEHLA
Reading frame 2
GTSQQRQRVCACVCVCVRVCVYACICVGWVRR*RWAGSRPRSRRP*TH
WFGVS*GQWGPLRSLNRAVSECEV*KDPPEKPALKLLQSSGERTQQGW
TGRME*R*VGS*SGQDLVLAWLWGASRGESGTLVVWPGADGGVSGAGR
DESRMIHGGVWQEAGKDDYVKALPGQLKPFETLLSQNQGGKTFIVGDQ
VSIW-
Reading frame 3
GPVSRGSVCARACACVCVCVCTLAFVSGG*GDRDGRAVGPGPEGLEPT
GLESPKGNGGH*EV*TGLCLNVRSRRILQRSQL*SFCNHLVREPSKDG
QAEWNRDELAAEVDRIWY*PGCGEQAEENLGLWWSGLGQTGVSQGLGG
MRVG*YMVVSGRRRARMTM*RHCPGN*SLLRPCCPRTREARPSLWETR
*ASG-
```

Dictionaries

```
codon_table = {
    'GCU':'A', 'GCC':'A', 'GCA':'A', 'GCG':'A', 'CGU':'R', 'CGC':'R',
    'CGA':'R', 'CGG':'R', 'AGA':'R', 'AGG':'R', 'UCU':'S', 'UCC':'S',
    'UCA':'S', 'UCG':'S', 'AGU':'S', 'AGC':'S', 'AUU':'I', 'AUC':'I',
    'AUA':'I', 'UUA':'L', 'UUG':'L', 'CUU':'L', 'CUC':'L', 'CUA':'L',
    'CUG':'L', 'GGU':'G', 'GGC':'G', 'GGA':'G', 'GGG':'G', 'GUU':'V',
    'GUC':'V', 'GUA':'V', 'GUG':'V', 'ACU':'T', 'ACC':'T', 'ACA':'T',
    'ACG':'T', 'CCU':'P', 'CCC':'P', 'CCA':'P', 'CCG':'P', 'AAU':'N',
    'AAC':'N', 'GAU':'D', 'GAC':'D', 'UGU':'C', 'UGC':'C', 'CAA':'Q',
    'CAG':'Q', 'GAA':'E', 'GAG':'E', 'CAU':'H', 'CAC':'H', 'AAA':'K',
    'AAG':'K', 'UUU':'F', 'UUC':'F', 'UAU':'Y', 'UAC':'Y', 'AUG':'M',
    'UGG':'W',
    'UAG':'STOP', 'UGA':'STOP', 'UAA':'STOP'
    }
```

Dictionaries

- Dictionaries are structures for mapping immutable objects (*keys*) to arbitrary objects (*values*).
- A key and its value are separated by a colon, and *key:value* pairs are separated by a comma.
- Dictionaries are useful for searching information quickly.

```
>>> print codon_table['GCU']
'A'
```

Dictionaries

- Keys must be unique.
  - If you try to insert two identical keys in a dictionary, the newer one will overwrite the other.
- You can also assign elements one by one.

```
>>> codon_table = {}
>>> codon_table['GCU'] = 'A'
>>> codon_table
{'GCU': 'A'}
>>> codon_table['CGA'] = 'R'
>>> codon_table
{'GCU': 'A', 'CGA': 'R'}
```

- If you want to look up the value corresponding to a given key, you can use square brackets.

```
>>> codon_table['GCU']
'A'
```

Dictionaries

- Get a list of all keys or values:

```
>>> codon_table.keys()
['CGA', 'GCU']
>>> codon_table.values()
['R', 'A']
```

- Check if a dictionary contains a given key:

```
>>> 'GCU' in codon_table
True
```

- Calculate the number of elements of the dictionary:

```
>>> len(codon_table)
2
```

- Selectively delete one key:value pair:

```
>>> del codon_table['GCU']
```

Searching in a Dictionary

```
# read the RNA sequence into a single string
rna = ''
for line in open('A06662-RNA.fasta'):
    if not line.startswith('>'):
        rna = rna + line.strip()

# translate one frame at a time
for frame in range(3):
    prot = ''
    print 'Reading frame ' + str(frame + 1)
    for i in range(frame, len(rna), 3):
        codon = rna[i:i + 3]
        if codon in codon_table:
            if codon_table[codon] == 'STOP':
                prot = prot + '*'
            else:
                prot = prot + codon_table[codon]
        else:
            # handle too short codons
            prot = prot + '-'

    # format to blocks of 48 columns
    i = 0
    while i < len(prot):
        print prot[i:i + 48]
        i = i + 48
```

The *while* Statement

```
# read the RNA sequence into a single string
rna = ''
for line in open('A06662-RNA.fasta'):
    if not line.startswith('>'):
        rna = rna + line.strip()

# translate one frame at a time
for frame in range(3):
    prot = ''
    print 'Reading frame ' + str(frame + 1)
    for i in range(frame, len(rna), 3):
        codon = rna[i:i + 3]
        if codon in codon_table:
            if codon_table[codon] == 'STOP':
                prot = prot + '*'
            else:
                prot = prot + codon_table[codon]
        else:
            # handle too short codons
            prot = prot + '-'

    # format to blocks of 48 columns
    i = 0
    while i < len(prot):
        print prot[i:i + 48]
        i = i + 48
```

The *while* Statement

- The general *while* loop syntax is:

```
while <condition>:
        <statements>
```

- The main thing to keep in mind when writing *while* loops is the exit condition.

The *while* Statement

- You need to extract a specific record from the Uniprot database.

```
swissprot = open("SwissProt.fasta")
insulin_ac = 'P61981'
result = None

while result == None:
    line = swissprot.next()
    if line.startswith('>'):
        ac = line.split('|')[1]
        if ac == insulin_ac:
            result = line.strip()

print result
```

- Running output:

```
>sp|P61981|1433G_HUMAN 14-3-3 protein gamma OS=Homo sapiens GN=YWHAG PE=1 SV=2
```

Examples

- How to fill a dictionary from a FASTA file where the Uniprot ACs are the keys and the corresponding sequences are their values?

```
sequences = {}
ac = ''
seq = ''
for line in open("SwissProt.fasta"):
    if line.startswith('>') and seq != '':
        sequences[ac] = seq
        seq = ''
    if line.startswith('>'):
        ac = line.split('|')[1]
    else:
        seq = seq + line.strip()

sequences[ac] = seq
print sequences.keys()
```

Examples

- How to extract the amino acid sequence from a PDB file?

```
HEADER    HYDROLASE (SERINE PROTEINASE)           24-JUL-89   1TLD
TITLE     CRYSTAL STRUCTURE OF BOVINE BETA-TRYPSIN AT 1.5 ANGSTROMS
TITLE    2 RESOLUTION IN A CRYSTAL FORM WITH LOW MOLECULAR PACKING
TITLE    3 DENSITY. ACTIVE SITE GEOMETRY, ION PAIRS AND SOLVENT
TITLE    4 STRUCTURE
COMPND    MOL_ID: 1;
 ... ...
SEQRES   1 A  223  ILE VAL GLY GLY TYR THR CYS GLY ALA ASN THR VAL PRO
SEQRES   2 A  223  TYR GLN VAL SER LEU ASN SER GLY TYR HIS PHE CYS GLY
SEQRES   3 A  223  GLY SER LEU ILE ASN SER GLN TRP VAL VAL SER ALA ALA
SEQRES   4 A  223  HIS CYS TYR LYS SER GLY ILE GLN VAL ARG LEU GLY GLU
SEQRES   5 A  223  ASP ASN ILE ASN VAL VAL GLU GLY ASN GLU GLN PHE ILE
```

Examples

```
aa_codes = {
     'ALA':'A', 'CYS':'C', 'ASP':'D', 'GLU':'E',
     'PHE':'F', 'GLY':'G', 'HIS':'H', 'LYS':'K',
     'ILE':'I', 'LEU':'L', 'MET':'M', 'ASN':'N',
     'PRO':'P', 'GLN':'Q', 'ARG':'R', 'SER':'S',
     'THR':'T', 'VAL':'V', 'TYR':'Y', 'TRP':'W'}

seq = ''

for line in open("1TLD.pdb"):
    if line[0:6] == "SEQRES":
        columns = line.split()
        for resname in columns[4:]:
            seq = seq + aa_codes[resname]

i = 0
print ">1TLD"
while i < len(seq):
    print seq[i:i + 64]
    i = i + 64
```

Summary

- Managing Your Biological Data with Python
  - Chapter 3. Analyzing a Data Column
  - Chapter 4. Parsing Data Records
  - Chapter 5. Searching Data
- Python codes in https://bitbucket.org/krother/python-for-biologists/src/