Introduction to Programming for Scientists

Prof. Steven Ludtke
N410, sludtke@bcm.edu

Lecture 3:
Writing Programs

Homework Review

1. Ask the user to enter a 1-letter DNA sequence, for example “CTGGGCCACACTGGAAGAACTGTGTTGGGCCACA"

    - Count the number of each nucleotide present in the entered sequence (and print the count)

    - Print the reverse complement of the entered sequence

Programming

- How do we represent the data ?
- Break the task into small pieces
- Code each of the pieces

Programming

- How do we represent the data ?
  - string
  - list
  - bytearray
- Break the task into small pieces
- Code each of the pieces

Programming

- How do we represent the data ?
- Break the task into small pieces
  - Read the string from the user
  - Convert to upper case
  - C↔G and A↔T
  - Reverse order
- Code each of the pieces

Homework Review

Tempting:

```
seq=input("Enter a sequence: ").upper()

cmpl=seq.replace("C","G").replace("G","C").\
  replace("A","T").replace("T","A")
print("".join(reversed(cmpl)))
```

… but wrong

```
print "CAGT".replace("C","G").replace("G","C").\
  replace("A","T").replace("T","A")

'CACA'
```

Homework Review

Ok, how about:

```
seq=input("Enter a sequence: ").upper()

for i in range(len(seq)):
   if   seq[i]=="C" : seq[i]="G"
   elif seq[i]=="G" : seq[i]="C"
   elif seq[i]=="T" : seq[i]="A"
   elif seq[i]=="A" : seq[i]="T"
   else : print("ERROR with ",seq[i])

print("".join(reversed(seq)))
```

… but strings are immutable ! …grrr …now what?

Homework Review

Build a new string! :

```
seq=input("Enter a sequence: ").upper()

cmpl=""
for i in range(len(seq)):
   if   seq[i]=="C" : cmpl=cmpl+"G"
   elif seq[i]=="G" : cmpl=cmpl+"C"
   elif seq[i]=="T" : cmpl=cmpl+"A"
   elif seq[i]=="A" : cmpl=cmpl+"T"
   else :
     print("ERROR with ",seq[i]," at ",i)

print("".join(reversed(seq)))
```

Homework Review

Shortcut:

```
seq=input("Enter a sequence: ").upper()

cmpl=""
for i in range(len(seq)):
   if   seq[i]=="C" : cmpl+="G"
   elif seq[i]=="G" : cmpl+="C"
   elif seq[i]=="T" : cmpl+="A"
   elif seq[i]=="A" : cmpl+="T"
   else :
     print("ERROR with ",seq[i]," at ",i)

print("".join(reversed(cmpl)))
```

Works, but unfortunately this is EXTREMELY inefficient in Python.

Homework Review

Ok, then, let's make the string into a list (data representation):

```
seq=list(input("Enter a sequence: ").upper())

for i in range(len(seq)):
   if   seq[i]=="C" : seq[i]="G"
   elif seq[i]=="G" : seq[i]="C"
   elif seq[i]=="T" : seq[i]="A"
   elif seq[i]=="A" : seq[i]="T"
   else : print("ERROR with ",seq[i])

print("".join(reversed(seq)))
```

Further improvements ?

Homework Review

We could use a dictionary instead of all those if's :

```
seq=list(input("Enter a sequence: ").upper())

dnamap={"C":"G","G":"C","T":"A","A":"T"}

for i in range(len(seq)):
 seq[i]=dnamap[seq[i]]

print("".join(reversed(seq)))
```

If there is an error due to an illegal letter, the program crashes 😞

try, except

- A way to avoid having errors crash your program
- An alternative to lots of ‘if’ statements

- try: - try to do something
- except <exception>: - if something specific fails, do this
- except: - if anything else fails, do this

- http://docs.python.org/library/exceptions.html

Homework Review

Better error detection :

```
seq=list(input("Enter a sequence: ").upper())

dnamap={"C":"G","G":"C","T":"A","A":"T"}

for i in range(len(seq)):
   try: seq[i]=dnamap[seq[i]]
   except:
     print("Error with",seq[i],"at",i)

print("".join(reversed(seq)))
```

If there is an error due to an illegal letter, the program continues

Homework Review

A variation using the map() function :

```
seq=list(input("Enter a sequence: ").upper())

dnamap={"C":"G","G":"C","T":"A","A":"T"}
seq=list(map(dnamap.get,seq))

print("".join(reversed(seq)))
```

If there is an error due to an illegal letter, it dies again. We could use try/except on map( ), but error detection would be less precise and wouldn't continue

Homework Review

"bytearray" is a mutable string, but slightly trickier to use:

```
seq=bytearray(input("Enter a sequence: ").upper(),"utf-8")

dnamap={ord("C"):ord("G"),ord("G"):ord("C"),\
  ord("T"):ord("A"),ord("A"):ord("T")}

for i in range(len(seq)):
   try: seq[i]=dnamap[seq[i]]
   except: print("The letter",chr(seq[i]),"is unknown")

seq.reverse()
print(seq.decode())
```

Homework Review

Let's get back to that original approach :

Tempting:

```
seq=input("Enter a sequence: ").upper()

print seq.replace(“C”,"G").replace("G","C").\
  replace(“A”,"T").replace("T","A")
```

… but wrong

Maybe we could fix this…

Homework Review

It works ! :

```
seq=input("Enter a sequence: ").upper()

cmpl=seq.replace("C","g").replace("G","c").\
  replace("A","t").replace("T","a").upper()

print("".join(reversed(cmpl)))
```

Homework Review

How about this one ? :

```
seq=input("Enter a sequence: ")

table="".maketrans("ACGTacgt","TGCATGCA")

print("".join(reversed(seq.translate(table))))
```

This is the most efficient program to perform this task !

DNA → Protein

- Write a program to convert a file containing a DNA sequence to its corresponding protein sequence*.

* - ignoring post-translational modifications, splicing, and other issues, just a straight translation

Programming

- **How do we represent the data ?**
- Break the task into small pieces
- Code each of the pieces

Genbank Example

```
  1 atggcagcta aagacgtaaa attcggtaac gacgctcgtg tgaaaatgct gcgcggcgta
 61 aacgtactgg cagatgcagt gaaagttacc ctcggtccga aaggccgtaa cgtagttctg
121 gataaatctt tcggtgcacc gaccatcacc aaagatggtg tttccgttgc tcgtgaaatc
181 gaactggaag acaagttcga aaacatgggt gcgcagatgg tgaaagaagt tgcctctaaa
241 gcgaacgacg ctgcaggcga cggtaccacc actgcaaccg tactggctca ggctatcatc
301 actgaaggtc tgaaagctgt tgctgcgggc atgaacccga tggacctgaa acgtggtatc
361 gacaaagctg ttaccgctgc agttgaagaa ctgaaagcgc tgtccg
```

Data Representation

- DNA sequence
  - A string ?
  - Strip out whitespace, numbers, etc ?
  - Error checking ?
- Protein Sequence
  - A string ?
- Translation Table
  - Dictionary (?)

1st base	2nd base								3rd base
	T		C		A		G		
T	TTT	(Phe/F) Phenylalanine	TCT	(Ser/S) Serine	TAT	(Tyr/Y) Tyrosine	TGT	(Cys/C) Cysteine	T
	TTC		TCC		TAC		TGC		C
	TTA	(Leu/L) Leucine	TCA		TAA	Stop (*Ochre*)	TGA	Stop (*Opal*)	A
	TTG		TCG		TAG	Stop (*Amber*)	TGG	(Trp/W) Tryptophan	G
C	CTT		CCT	(Pro/P) Proline	CAT	(His/H) Histidine	CGT	(Arg/R) Arginine	T
	CTC		CCC		CAC		CGC		C
	CTA		CCA		CAA	(Gln/Q) Glutamine	CGA		A
	CTG		CCG		CAG		CGG		G
A	ATT	(Ile/I) Isoleucine	ACT	(Thr/T) Threonine	AAT	(Asn/N) Asparagine	AGT	(Ser/S) Serine	T
	ATC		ACC		AAC		AGC		C
	ATA		ACA		AAA	(Lys/K) Lysine	AGA	(Arg/R) Arginine	A
	ATG[A]	(Met/M) Methionine	ACG		AAG		AGG		G
G	GTT	(Val/V) Valine	GCT	(Ala/A) Alanine	GAT	(Asp/D) Aspartic acid	GGT	(Gly/G) Glycine	T
	GTC		GCC		GAC		GGC		C
	GTA		GCA		GAA	(Glu/E) Glutamic acid	GGA		A
	GTG		GCG		GAG		GGG		G

Represent as Dict

```
{0:['tag', 'taa', 'tga'], 'a':['gca', 'gcc', 'gcg', 'gct'],
'c':['tgt', 'tgc'], 'e':['gag', 'gaa'], 'd':['gat', 'gac'],
'g':['ggt', 'ggg', 'gga', 'ggc'], 'f':['ttt', 'ttc'],
'i':['atc', 'ata', 'att'], 'h':['cat', 'cac'],
'k':['aaa', 'aag'], 'm':['atg'],
'l':['tta', 'ttg', 'ctt', 'ctg', 'cta', 'ctc'],
'n':['aac', 'aat'], 'q':['cag', 'caa'],
'p':['cct', 'ccg', 'cca', 'ccc'],
's':['tct', 'tcg', 'tcc', 'tca', 'agc', 'agt'],
'r':['cgt', 'agg', 'cga', 'cgc', 'cgg', 'aga'],
't':['acc', 'act', 'aca', 'acg'], 'w':['tgg'],
'v':['gta', 'gtc', 'gtg', 'gtt'], 'y':['tat', 'tac']}
```

Represent as Dict

```
xlate={   "ttt":"f","ttc":"f","tta":"l","ttg":"l",
"ctt":"l","ctc":"l","cta":"l","ctg":"l","att":"i",
"atc":"i","ata":"i","atg":"m","gtt":"v","gtc":"v",
"gta":"v","gtg":"v","tct":"s","tcc":"s","tca":"s",
"tcg":"s","cct":"p","ccc":"p","cca":"p","ccg":"p",
"act":"t","acc":"t","aca":"t","acg":"t","gct":"a",
"gcc":"a","gca":"a","gcg":"a","tat":"y","tac":"y",
"taa":"0","tag":"0","cat":"h","cac":"h","caa":"q",
"cag":"q","aat":"n","aac":"n","aaa":"k","aag":"k",
"gat":"d","gac":"d","gaa":"e","gag":"e","tgt":"c",
"tgc":"c","tga":"0","tgg":"w","cgt":"r","cgc":"r",
"cga":"r","cgg":"r","agt":"s","agc":"s","aga":"r",
"agg":"r","ggt":"g","ggc":"g","gga":"g","ggg":"g"}
```

How does this influence the code ?

- DNA triplet -> Amino Acid
  - Dict keyed by amino acid:
    - for each key
      - for each value of that key
        - if match stop and return key
  - Dict keyed by DNA triplet:
    - Look up triplet, return value for key

Programming

- How do we represent the data ?
- **Break the task into small pieces**
- Code each of the pieces

Steps

- Get data filename
- Open file & read data
- Preprocess data (just the letters we want)
- Loop over the data 3 elements at a time
  - Translate
- Print results

Arguments

- myprogram.py file1.txt file2.txt
- python myprogram.py file1.txt file2.txt

- from sys import argv
- len(argv) -> 3
- argv[0] -> myprogram.py
- argv[1] -> file1.txt

Note: Can't do this if you start by clicking on the program

Get Filename

```
from sys import argv
fsp=argv[1]

-or-

fsp=input(“Filename:")
```

Read/write files

- handle=open(<filename>,<mode>)
- Valid modes: [r|w|a|U][+][b]
  - r - open file for reading
  - w - truncate file and open for writing
  - a - open file for appending (writing at end of file, platform dependent)
  - U - Universal text file support
  - + - in addition to basic mode, permit writing
  - b - open in binary mode (default is text mode)
- Different platforms do a newline differently:
  - Unix - '\n'
  - Old mac - '\r'
  - Windows - '/r/n'

File Methods

- string=file.read([len]) - Reads whole file (or [len] bytes)
- string=file.readline() - Read a single line of text
- stringlist=file.readlines() - Read whole file as a list of lines
- file.write(<string>) - Write <string> to file (no automatic /n)
- file.close() - Close the file (automatic when file object freed)
- file.flush() - Write output to file immediately (no buffering)
- int=file.tell() - Current location in the file (use binary mode!)
- file.seek(<loc>) - Move to a specific position in the file
- for line in file: print line - File acts as an iterator for lines

- sys.stdin, stdout, stderr - Automatic file handles

Read Data & Preprocess

```
dna=open(fsp,"r").read()     # read the entire file into ram

# This uses the 'deletechars' option of the string translate
# method to remove characters we don't want. Technically
# we could also add an upper->lower conversion
dna=dna.translate(str.maketrans("CAGT","cagt","0123456789 \t\n
\r"))
```

Loop & Translate

```
out=(fsp+".prot","w")

for i in range(0,len(dna),3):
 triplet=dna[i:i+3]
 try: amino=xlate[triplet]
 except:
  print("Unknown triplet: ",triplet)
  sys.exit(1)
 out.write(amino)

out.write("\n")
```

Put it all together

```
import sys

xlate={"ttt":"f" ... "ggg":"g"}

fsp=sys.argv[1]
dna=open(fsp,"r").read()
dna=dna.translate(str.maketrans("","","0123456789 \t\n
\r")).lower()
out=open(fsp+".prot","w")

for i in range(0,len(dna),3):
 triplet=dna[i:i+3]
 try: amino=xlate[triplet]
 except:
  print("Unknown triplet: ",triplet)
  sys.exit(1)
 out.write(amino)

out.write("\n")
```

Nested Loops

- a loop inside a loop

```
for i in range (10):
  for j in range(10):
    print i,j
```

- Continue/break - interrupting the flow of a loop

```
for i in range(20):
  if i==5 : continue
  if i>17 : break
  print i
```

While Loops

- While loop - continues as long as a condition is met

```
a=0
while a<10:
    a=a+0.1
    print a
```

Homework #3

1. Start with the simple DNA -> Protein translation program we wrote in class today (you can download it from the class site). Let's assume that we've dealt with identifying a promotor, etc, and that the sequence we're getting is within a few residues of being the start of a coding region of DNA. However, the exact frame hasn't been identified, and clearly if we start with a frame shift we'll get the wrong sequence. Modify the program to identify the correct frame by assuming the first ATG we find represents the beginning of the coding region, then translate only until a stop codon is found. example: if your program were given 'gatggcagct aaagacgtaa aatgaaaa' it should produce ‘maakdvk'

2. Write a simplified amortization program, that is, a program that keeps track of how much you still owe on a loan. We will simplify the math a bit: Assume that each month, the amount increases by the balance times 1/12 the interest rate and decreases by the amount of the fixed monthly payment. You should ask the user for the amount of the loan, the annual percentage interest rate, and the payment amount. For each month, print the payment number, interest for the month, and the remaining balance on the loan after the payment. Continue to write out new months until the loan is payed off.

   To hand in your homework: Create a ".py" file containing each program. Do not create a .zip file. You do not need to provide the output from the programs, just the programs themselves. If you need to write any comments or description of what you did, best to put them in the .py file as "#" comments. Please include your name in the filename, eg - smith_homework_3_mortgage.py