Lecture 4

Homework Review
Standard Libraries

Prof. Steven Ludtke
N410.07, sludtke@bcm.edu

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

Homework Review

Tempting:

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

for i in xrange(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=raw_input("Enter a sequence: ").upper()

cmpl=""
for i in xrange(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=raw_input("Enter a sequence: ").upper()

cmpl=""
for i in xrange(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(seq))
```

Works, but unfortunately this is EXTREMELY inefficient in Python.

Homework Review

Ok, then, let's make the string into a list:

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

for i in xrange(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(raw_input("Enter a sequence: ").upper())

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

for i in xrange(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(raw_input("Enter a sequence: ").upper())

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

for i in xrange(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(raw_input("Enter a sequence: ").upper())

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

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

Homework Review

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

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

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

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

seq.reverse()
print seq
```

Homework Review

Python3 version:

```
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",seq[i],"is unknown")

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

Homework Review

Let's get back to that original approach :

Tempting:

```
seq=raw_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=raw_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 ? :

```
import string
seq=raw_input("Enter a sequence: ").upper()

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

print seq.translate(table)
```

This is the most efficient program to perform this task !

Homework Review

2. Write a program to identify the winner of a rock,paper,scissors game. Ask the user what player 1 picked (rock, paper or scissors), then ask what player 2 picked. Finally, print the winner (player 1, 2 or tie)

```
import sys

pick1=raw_input("Player 1 (rock, paper, scissors): ")
if pick1 not in ("rock","paper","scissors"):
   print "Bad input!"
   sys.exit(1)
pick1=pick1[0].lower()

pick2=raw_input("Player 2 (rock, paper, scissors): ")
if pick2 not in ("rock","paper","scissors"):
   print "Bad input!"
   sys.exit(1)
pick2=pick2[0].lower()
```

Functions

A function is used when some action needs to be completed in different parts of a program, or re-used in multiple programs. It allows code to be grouped in a self-contained block, and can also make debugging easier.

Generally it is not good practice to make functions that are called only one time strictly for organizational purposes. Use comments instead.

Examples

```
def middle(x): return int(str(x)[1:-1])


def between(lo,val,hi):
   """Checks to see if val is between lo and hi"""
   if lo<val and val<hi : return True
   else: return False


def cmp(a,b):
   """Compare the second element of list a to list b for
   use with sort(), returns -1, 0 or 1"""
   return a[1]-b[1]
```

```
def rps(name):
 ret=""
 while ret not in ("rock","paper","scissors"):
  ret=raw_input(name+" (rock,paper,scissors): ")
 return ret[0].lower()

pick1=rps("Player 1")
pick2=rps("Player 2")
```

```
pick1=rps("Player 1")
pick2=rps("Player 2")

if pick1==pick2 : print "Tie!"
elif pick1=="r" and pick2=="p": print "Player 2 wins!"
elif pick1=="p" and pick2=="r": print "Player 1 wins!"
elif pick1=="s" and pick2=="r": print "Player 2 wins!"
elif pick1=="r" and pick2=="s": print "Player 1 wins!"
elif pick1=="p" and pick2=="s": print "Player 2 wins!"
elif pick1=="s" and pick2=="p": print "Player 1 wins!"
```

```
pick1=rps("Player 1")
pick2=rps("Player 2")

table={("r","p"):2, ("p","r"):1, ("s","r"):2, … }

print "Player",table[(pick1,pick2)],"wins"
```

```
pick1=rps("Player 1")
pick2=rps("Player 2")

table={ ("p","r"):"Paper covers rock",
 ("r","s"):"Rock breaks scissors",
 ("s","p"):"Scissors cuts paper" }

if pick1==pick2 : print "Tie!"

try:
   print table[(pick1,pick2)], "Player 1 wins!"
except:
   print table[(pick2,pick1)], "Player 2 wins!"
```

Read/write files

- handle=open(<filename>,<mode>) -or- file(<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

String Formatting

- {[field][:format]}

- format ::= [[fill]align][sign][#][0][width][,][.precision][type]
- fill ::= <any character>
- align ::= "<" | ">" | "=" | "^"
- sign ::= "+" | "-" | " "
- width ::= integer
- precision ::= integer
- type ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"

A Few Standard Libraries

- sys - System-specific parameters
- os - Operating system functions
- string - String manipulation
- time - Delays, formatting time
- datetime - Manipulate dates/times
- pprint - Pretty printing
- urllib2 - Easy web access

File Manipulation

- os.getcwd() - the current working directory (folder)
- os.chdir() - change the current working directory
- os.listdir - Lists files in a particular folder
- os.stat - info about a file
- os.rename - rename (mv) a file
- os.mkdir - create a folder
- os.remove - delete a file
- os.rmdir - remove a directory
- os.system - execute a command (mostly mac/linux)

PyPi

- http://pypi.python.org
- Note that many packages also have installers available for Windows
- easy_install
  - Comes with Mac
  - May be in a package called python_setuptools on linux
- pip
  - newer alternative, but not standard on mac

Homework 2

- Start thinking about a topic for your class projects. Next homework you will need to tell me what your planned project is.

- If you take out a large loan, such as a mortgage on a house, the bank will generate an Amortization Schedule for you. This is a table which shows after each payment how much you still owe on the loan, and generally also how much of each payment is interest and principal. Write a program which asks the user for the amount of the loan, amount of the monthly payment, and the annual interest rate. Compute the interest as if it were 1/12 the annual amount, compounded monthly. Print the amortization schedule until the principle falls to zero. You don't need to deal explicitly with the final month which may have a slightly different payment.
- Lab #1: Next Monday is a holiday, so we won't be able to finish Lab 1 then.
  - If your group finished Lab 1 and turned it in, good. You have no second homework assignment for next Friday.
  - If you can get your group together again informally sometime, you are welcome to finish the lab as it was assigned and turn it in as Lab #1 before next Friday. Amanda (TA) will have the list of group assignments if you don't remember your group.
  - Otherwise, you will need to individually complete either Task #1 or, if you prefer, the combination of Task #3 and #4, and turn it in as Lab #1 before next Friday. Remember if you do 3/4 you must do it differently than the way I did it in the downloadable program.