Lecture 4

Standard Libraries
More Nested Loops

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

Homework Review

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'

Homework Review

```
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=(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")
```

Homework Review

```
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()
dna=dna[dna.find(“atg”):]
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")
```

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"}
```

Homework Review

```
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()
dna=dna[dna.find(“atg”):]
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)

    if amino==“0” : break
    out.write(amino)

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

Homework Review

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.

Homework Review

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

Homework Review

- How do we represent the data?
  - balance - Balance at the end of each month
  - rate - monthly fractional interest rate
  - payment - amount of monthly payment
- Break the code into small pieces:
  - ask user for values
  - convert rate to monthly fraction (/1200)
  - if balance*rate > payment then raise error
  - loop until balance <=0
    - compute interest = balance*rate
    - balance = balance + interest - payment
    - print values

Amortization

```
import sys
balance=float(input("Amount of loan:"))
rate=float(input("Rate as a %:"))/1200.0
payment=float(input("Monthly payment:"))

if rate*balance>payment :
   print("Insufficient payment !")
   sys.exit(1)

month=1
while (balance>0):
   print(month,")",balance,”+",rate*balance,"-",payment,"=",
balance+rate*\ balance-payment)
   balance+=rate*balance-payment
   month+=1
```

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" | "%"

Simple Examples

```
# 3 columns with automatic formatting of an integer, a floating point number and a string
"{}\t{}\t{}".format(int1,float1,string1)

# named keywords, which can be reused
"The tree is {tree} ft tall and the house is {house} ft high, the tree is taller at {tree}
ft".format(tree=60,house=30)

# formatting numbers
"{num:0.5f}, {num:0.3f} and {num:0.5e} are all the same value, but represented in
different ways".format(num=123.45678)

# fixed width
"{str1:8s}{str2:8s}{int1:8.2f}".format("abc","testing",98.2)
"{str1:>8s}{str2:8s}{int1:8.2f}".format("abc","testing",98.2)
```

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]
```

help()

- help(object) - gives help on the object. Returns the string immediately following the object definition.

List of Lists

```
X=[["a","b","c","d"],["e","f","g","h"],
["i","j","k","l"],["m","n","o","p"]]

print(X[1])
["e","f","g","h"]

print(X[1][2])
g
```

	0	1	2	3
0	a	b	c	d
1	e	f	g	h
2	i	j	k	l
3	m	n	o	p

Row-major Ordering

List of Lists

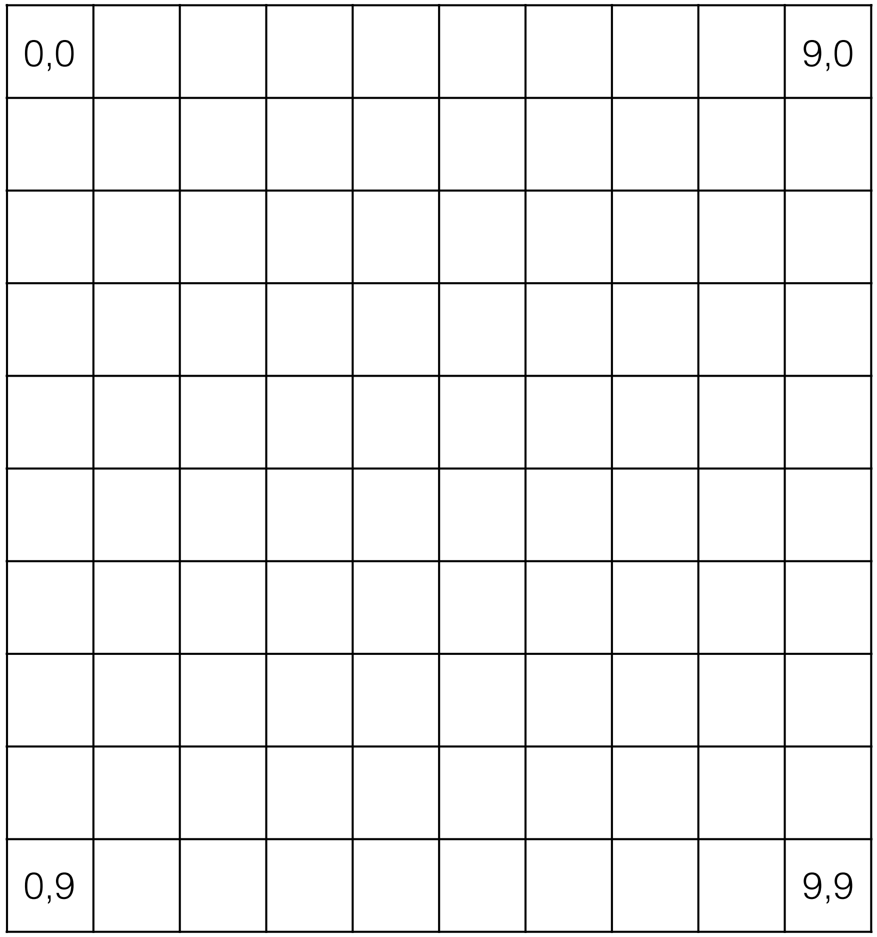
```
X=[["a","e","i","m"],["b","f","j","n"],
["c","g","k","o"],["d","h","l","p"]]

print(X[1])
["a","e","i","m"]

print(X[1][2])
j
```

	0	1	2	3
0	a	b	c	d
1	e	f	g	h
2	i	j	k	l
3	m	n	o	p

Column-major Ordering

How would you iterate over 2-dimensional data?

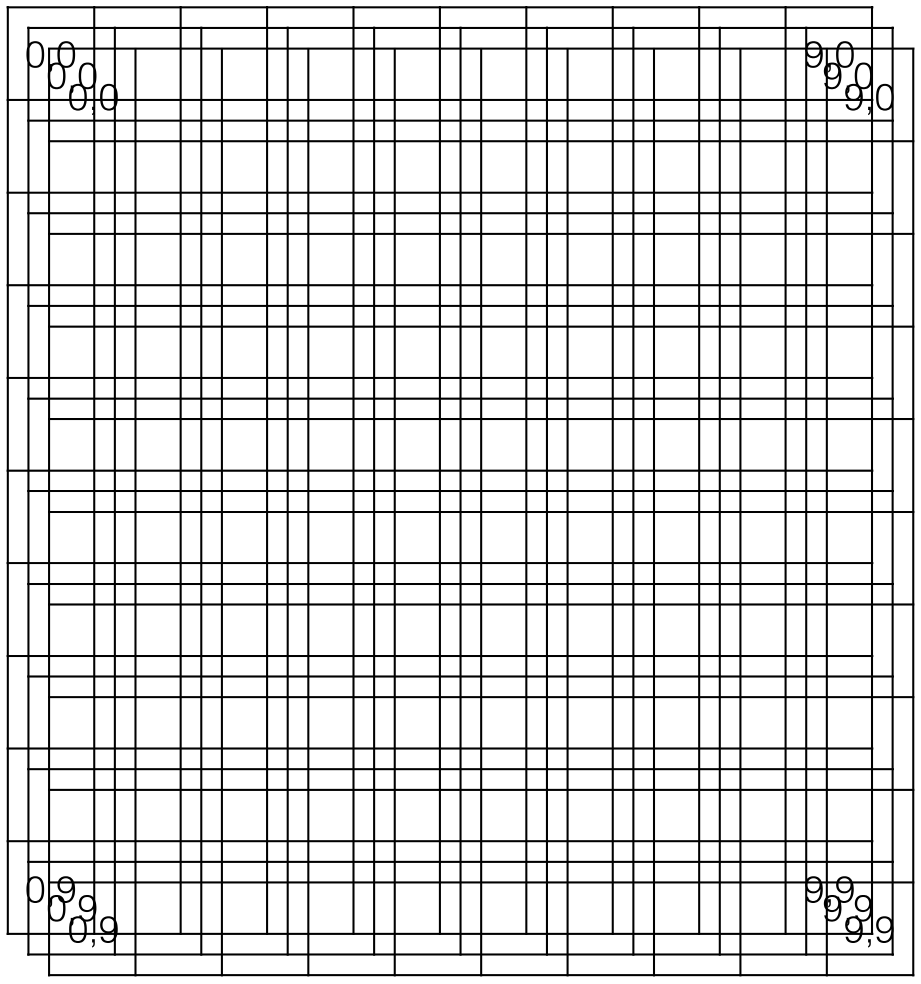
```
for a in range (100):
    y=a//10
    x=a%10
    print(array[y][x])
```



How would you iterate over n-dimensional data?

```
for a in range (1000):
    z=a//100
    y=(a%100)//10
    x=(a%100)%10
    print(array[z][y][x])
```

… (x10)

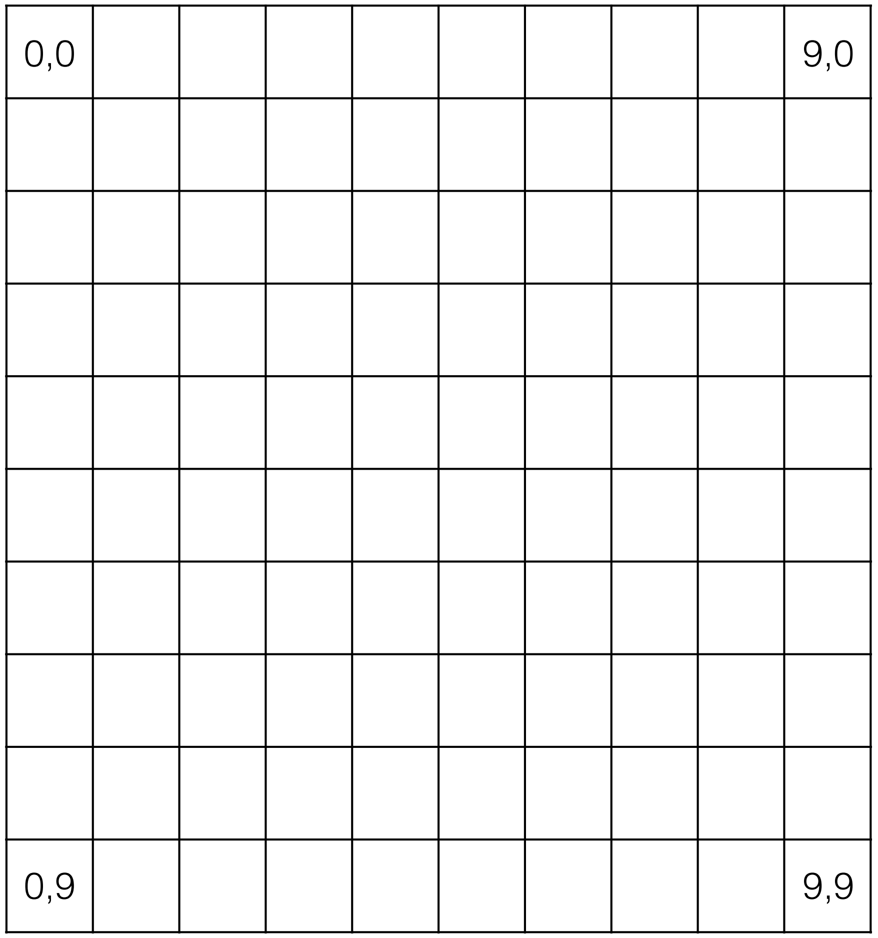


Nested Loops

(the right answer)

- a loop inside a loop

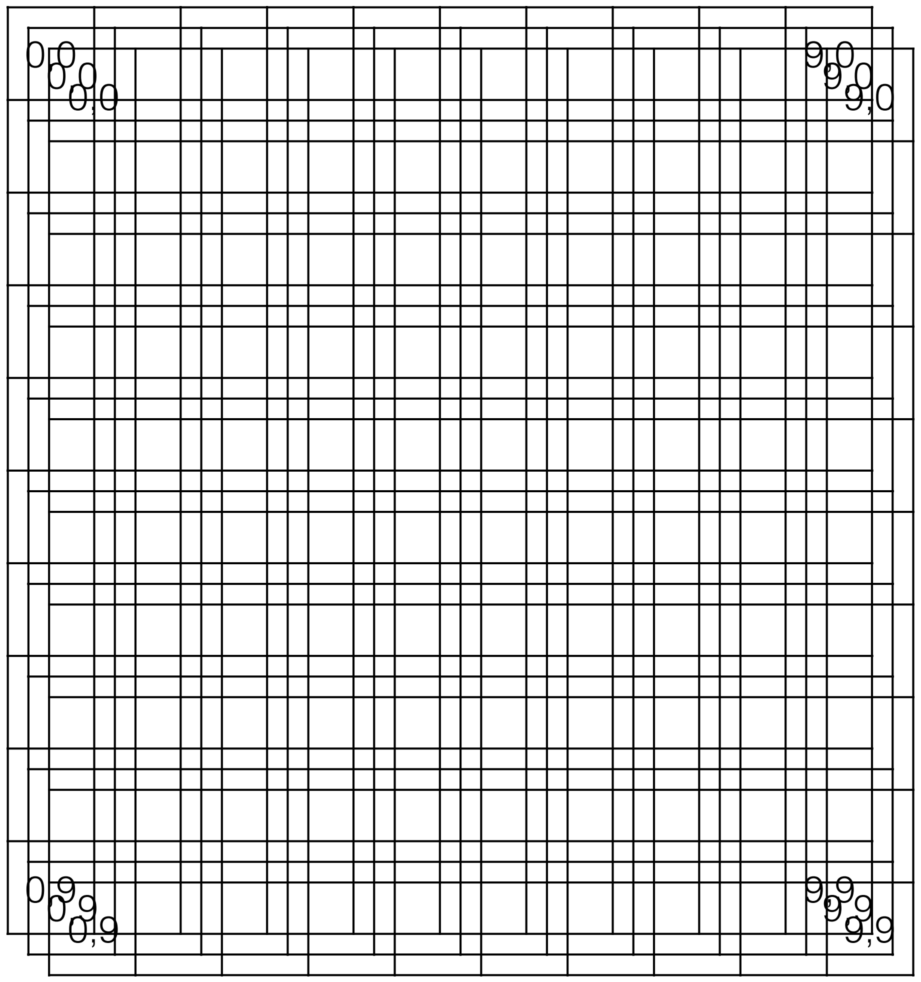
```
for y in range(10):
  for x in range(10):
    print(array[y][x])
```



How would you iterate over n-dimensional data?

```
for z in range(10):
 for y in range(10):
  for x in range(10):
   print(array[z][y][x])
```

… (x10)



Loop flow

- Continue/break/else - flow of a loop

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

- continue - skip the rest of the current iteration
- break - immediately exit the loop entirely
- else - only if the loop finishes without a break

import

- import module
- from module import name,name2,name3
- from module import *
- import module as othername

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
- urllib - Easy web access

urllib

Web Scraping:

```
from urllib.request import urlopen

conn=urlopen("http://blake.bcm.edu/dl/test.html")

for line in conn: print(line.decode("utf-8"),end="")
```

HTML

- http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html
- Declarative language
- HTML is a type of XML, XHTML obeys XML rules more completely
- Python HTMLParser module
- ‘commands’ in HTML are denoted by
  <command option=value option=value>text</command>

HTML

```
<HTML>
<HEAD><TITLE>My Page</TITLE></HEAD>
<BODY>
<H3>Hi Everyone</H3>
<P>This is really just some test text to demonstrate how
HTML works. I can do interesting things like
<i>italicize</i> make text <b>bold</b>, or even
<b><i>both together</i></b>. ta da
</BODY>
</HTML>
```

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
- pip
  - included as part of Python 2.7.9 and later
  - should be set up in Anaconda
- If using Anaconda, may consider "conda" instead of pip. Any packages available with conda will be easier to install.

Homework 4

- Install BioPython (http://biopython.org) on your computer, and make sure you can import it successfully before lab next Friday. If you have Anaconda set up as you should, you should just be able to type:

  ```
  conda install biopython
  ```

  If it is installed properly, you should be able to type this at the python3 prompt and not get an error:

  ```
  from Bio import SeqIO
  ```

- Start thinking about what you might want to do for a class project, I will be asking for your plan soon.

Homework 4

1) Write a program to print a logarithm table, with nice formatting and labels on the first row. The columns should be the different log bases. Use 2, e and 10 for bases. Rows will be the values being taken the log of. Use integers from 1 to 32.

2) Write a program that can read 2-column text files containing numbers. Assume the 2 numbers on each line are X and Y coordinates. Compute and print the minimum and maximum x and y value. You should be able to read files containing numbers on each line separated by any whitespace (any number of spaces or tabs).

eg-

```
1    2
2    3.2
2.9  3.9
4.1  5.0
…
```