Introduction to Programming for Scientists

Prof. Steven Ludtke
N420, sludtke@bcm.edu

Lecture 3:
Writing Programs

Group Learning 2

```
last=input("Max: ")
for i in range(1,last,2):
  print(i,i*i)
```

Group Learning 2

```
x=1
y=1.0
for i in range(2,30):
  x=x*i
  y=y*i
  print(i,x,y,x/y)
```

Homework 2.1

- From grade school, you will recall that a factor is any integer that another integer is divisible by. For example, the factors of 12 are 1,2,3,4,6,12. There is the extended concept of prime factors which include only the factors which are prime numbers, but that is not our concern in this problem. Write a program to ask the user for an integer, and display all of the number's factors (exclude 1 and the number itself). No cheating and using a library function to do this. You should write the code using for loops, if statements and basic math.

Homework 2.1

```
num=int(input("Enter a number:"))
for i in range(2,num):
    if num%i==0 : print(i)
```

Homework 2.1

```
num=int(input("Enter a number:"))
for i in range(2,num/2+1):
    if num%i==0 : print(i)
```

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)

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 ?
    - Data -> selection of player 1 and player 2, string?
- Break the task into small pieces
    - Ask for player 1 choice
    - Validate and standardize input
    - repeat for player 2
    - compute outcome
    - display result
- Code each of the pieces

```
import sys

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

help()

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

```
def rps(name):
 ret=""
 while ret not in ("rock","paper","scissors"):
  ret=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")
```

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

```
import sys
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!")
 sys.exit(0)

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

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

Start with the team learning exercise

```
cgccatggag accaacaccc ttcccaccgc cactccccct tcctctcagg gtccctgtcc          0
cctccagtga atcccagaag actctggaga gttctgagca gggggcggca ctctggcctc        120
tgattggtcc aaggaaggct ggggggcagg acgggaggcg aaacccctgg aatattcccg        180
acctggcagc ctcatcgagc tcggtgattg gctcagaagg gaaaaggcgg gtctccgtga        240
cgacttataa aagcccaggg gcaagcggtc cggataacgg ctagcctgag gagctgctgc        300
gacagtccac tacctttttc gagagtgact cccgttgtcc caaggcttcc cagagcgaac        360
```

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

- Read file from web
- Preprocess data (just the letters we want)
- Loop over the data 3 elements at a time
    - Translate
- Print results

Group Learning 2

```
import urllib
seq=urllib.request.urlopen("https://www.ebi.ac.uk/ena/data/view/
M11717&display=text&download=txt&filename=M11717.txt").read()
```

The sequence will have numbers, spaces and newlines in it. This is a typical DNA sequence representation. Come up with a strategy for removing all of this annotation, and produce another string containing just the sequence itself (acgt only).

Group Learning 2

Fix string issue:

```
import urllib
seq=urllib.request.urlopen("https://www.ebi.ac.uk/ena/data/view/
M11717&display=text&download=txt&filename=M11717.txt").read()
seq=str(seq,"utf-8")
```

The sequence will have numbers, spaces and newlines in it. This is a typical DNA sequence representation. Come up with a strategy for removing all of this annotation, and produce another string containing just the sequence itself (acgt only).

Group Learning 2

For machines with incorrect SSL configuration (certificate error):

```
import urllib
import ssl
context = ssl._create_unverified_context()
seq=urllib.request.urlopen("https://www.ebi.ac.uk/ena/data/view/
M11717&display=text&download=txt&filename=M11717.txt",context=conte
xt).read()
seq=str(seq,"utf-8")
```

The sequence will have numbers, spaces and newlines in it. This is a typical DNA sequence representation. Come up with a strategy for removing all of this annotation, and produce another string containing just the sequence itself (acgt only).

Group Learning 2

Fix string issue:

```
import urllib
seq=urllib.request.urlopen("https://www.ebi.ac.uk/ena/data/view/
M11717&display=text&download=txt&filename=M11717.txt").read()
seq=str(seq,"utf-8")
dna=seq[6050:]
```

The sequence will have numbers, spaces and newlines in it. This is a typical DNA sequence representation. Come up with a strategy for removing all of this annotation, and produce another string containing just the sequence itself (acgt only).

Read Data & Preprocess

```
import urllib
seq=urllib.request.urlopen("https://www.ebi.ac.uk/ena/data/
view/
M11717&display=text&download=txt&filename=M11717.txt").read()
dna=str(seq,"utf-8")

# find the sequence ?
dna=dna.split("SQ")[1]
dna=dna[dna.find("\n")+1:]
```

… but what if there is a "SQ" in the middle of the text somewhere?

Read Data & Preprocess

```
import urllib
seq=urllib.request.urlopen("https://www.ebi.ac.uk/ena/data/
view/
M11717&display=text&download=txt&filename=M11717.txt").read()
dna=str(seq,"utf-8").split("\n")

for i,ln in enumerate(dna):
    if ln[:2]=="SQ" : break

dna="".join(dna[i+1:])
```

New concepts

- enumerate(list)
  - returns [(0,item0),(1, item1),(2,item2)…]
- break
  - exits a 'for' loop prematurely

Read Data & Preprocess

```
import urllib
seq=urllib.request.urlopen("https://www.ebi.ac.uk/ena/data/
view/
M11717&display=text&download=txt&filename=M11717.txt").read()
dna=str(seq,"utf-8").split("\n")

for i,ln in enumerate(dna):
    if ln[:2]=="SQ" : break
dna=" ".join(dna[i+1:])

# 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=[]
for i in range(0,len(dna),3):
  triplet=dna[i:i+3]
  try: amino=xlate[triplet]
  except:
    print("Unknown triplet: ",triplet)
    break
  out.append(amino)

out="".join(out)
print(out)
```

Put it all together

```
import urllib
seq=urllib.request.urlopen("https://www.ebi.ac.uk/ena/data/view/
M11717&display=text&download=txt&filename=M11717.txt").read()
dna=str(seq,"utf-8").split("\n")

for i,ln in enumerate(dna):
    if ln[:2]=="SQ" : break
dna=" ".join(dna[i+1:])

dna=dna.translate(str.maketrans("CAGT","cagt","/0123456789 \t\n\r"))

xlate={"ttt":"f","ttc":"f", …}

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

out="".join(out)
print(out)
```

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. 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.

Due before class next Friday. Monday is a holiday, no class!