One-minute responses

- I think more problems would be OK as long as we could finish two in class time, and the others are for practice on our own
- It was nice that you pointed out that peculiarity with sort(). *More of those today!*
- Only improvement I could think of would be to somehow integrate the topics from the non-Python portion of lecture into the practice problems.
- I feel like I miss a lot in class, but that I can figure it out with more time at home.

One-minute responses

- I think I can understand the idea of dictionaries, but probably today unearthed some confusion in regards to lists and strings. I couldn't figure out how to make Python "read" the text file so that word 1 (and not character 1) is my dictionary key, and that word 2 is my item **for** word 1 (key). Any suggestions? *The easiest way to do this is to use* `split` *to turn your string into a list of words. You could examine each letter yourself, piling them up until you reach a space which would signal the end of a word, but* `split` *does this for you automatically. Writing your own version of* `split` *can be a valuable exercise.*

What is a function?

- Reusable piece of code
  - Write and test once, use many times
- Takes defined inputs and may return a defined output
- Helps organize your program

Parts of a function

- `def myname(myarg1, myarg2) :`
- The `def` statements creates a function
- The function name allows us to call it
- The argument list tells us what arguments it will receive
- The names in the argument list will be variables in the function

Parts of a function

```
return myanswer
```

- The `return` statement defines the value that the function returns
- If no `return` is executed, the function returns `None`
- It's legal to have more than one `return`:

```
if value <= 1.0 :
  return value
else :
  return 1.0
```

Jukes-Cantor distance correction

```
import sys
import math
rawdist = float(sys.argv[1])
if rawdist < 0.75 and rawdist > 0.0 :
  newdist = (-3.0/4.0) * math.log(1.0 - (4.0/3.0)* rawdist)
  print newdist
elif rawdist >= 0.75 :
  print 1000.0
else :
  print 0.0
```

Jukes-Cantor function step 1

```
import sys
import math
# add a function definition
def jc_dist(rawdist) :
  rawdist = float(sys.argv[1])
  if rawdist < 0.75 and rawdist > 0.0 :
    newdist = (-3.0/4.0) * math.log(1.0 - (4.0/3.0)* rawdist)
    print newdist
  elif rawdist >= 0.75 :
    print 1000.0
  else :
    print 0.0
```

Jukes-Cantor function step 2

```
import sys
import math
# add a function definition
def jc_dist(rawdist) :
# use the function argument instead of argv
  if rawdist < 0.75 and rawdist > 0.0 :
    newdist = (-3.0/4.0) * math.log(1.0 - (4.0/3.0)* rawdist)
    print newdist
  elif rawdist >= 0.75 :
    print 1000.0
  else :
    print 0.0
```

Jukes-Cantor function step 3

```
import sys
import math
# add a function definition
def jc_dist(rawdist) :
# use the function argument instead of argv
  if rawdist < 0.75 and rawdist > 0.0 :
    newdist = (-3.0/4.0) * math.log(1.0 - (4.0/3.0)* rawdist)
# return the value rather than printing it
    return newdist
  elif rawdist >= 0.75 :
# return the value rather than printing it
    return 1000.0
  else :
# return the value rather than printing it
    return 0.0
```

Jukes-Cantor function: final version

```
import sys
import math
def jc_dist(rawdist) :
  if rawdist < 0.75 and rawdist > 0.0 :
    newdist = (-3.0/4.0) * math.log(1.0 - (4.0/3.0)* rawdist)
    return newdist
  elif rawdist >= 0.75 :
    return 1000.0
  else :
    return 0.0
```

Using the function

```
>>> raw = 0.23
>>> corrected = jc_dist(raw)
>>> print corrected
0.274683296216
```

Using the function

```
mydata = [0.2, 0.22, 0.34, 0.18]
for index in range(0,len(mydata)) :
  mydata[index] = jc_dist(mydata[index])
print mydata
[0.2326, 0.2604, 0.4529, 0.2058]

# or a different approach
newdata = []
for entry in mydata :
  newdata.append(jc_dist(entry))
print newdata
```

We have seen several functions already

- `log()`
- `readline()`, `readlines()`, `read()`
- `sort()`
- `split()`, `replace()`, `lower()`

Most of these are attached to objects rather than stand-alone functions; this will be covered in an upcoming lecture.

Practice problem 1

Write a function which:

- Takes a DNA sequence (a string) as input
- Makes a new string in which all T or t have been replaced by U or u (DNA to RNA)
- Returns the new string
- In the same file, create a DNA sequence and call this function on it
- Print the value that the function returns

Solution and discussion

```
def dna_to_rna(seq) :
  seq = seq.replace("T","U")
  seq = seq.replace("t","u")
  return seq

myDNA = "ATCGTCGATCG"
print dna_to_rna(myDNA)
AUCGUCGAUCG
```

Why doesn’t this work?

```
# warning:  bad program!
def dna_to_rna(seq) :
  seq.replace("T","U")
  seq.replace("t","u")
  return seq

myDNA = "ATCGTCGATCG"
print dna_to_rna(myDNA)
ATCGTCGATCG
```

Why doesn’t this work?

```
# warning:  bad program!
def dna_to_rna(seq) :
  seq.replace("T","U")
  seq.replace("t","u")
  return seq

myDNA = "ATCGTCGATCG"
print dna_to_rna(myDNA)
ATCGTCGATCG
```

- String functions never change the string they are called on (strings are immutable, so they can’t)
- `seq.replace("T","U")` does not change `seq`
- Strings and lists seem similar, but this is a major difference

- `mylist.append(myDNA)` DOES change mylist

Another failed attempt

```
# warning:  bad program!
def dna_to_rna(seq) :
  seq = seq.replace("T","U")
  seq = seq.replace("t","u")

myDNA = "ATCGTCGATCG"
print dna_to_rna(myDNA)
None
```

Why doesn’t this work?

```
def dna_to_rna(seq) :
  seq = seq.replace("T","U")
  seq = seq.replace("t","u")

myDNA = "ATCGTCGATCG"
print dna_to_rna(myDNA)
None
```

- The string argument is a copy of the one in the main program
- Changes in the function do not change the original

Watch out for lists!

```
# warning: surprising program!

def dna_to_rna(seq) :
  for index in range(0,len(seq)) :
    if seq[index] == "T" :
      seq[index] = "U"
    if seq[index] == "t" :
      seq[index] = "u"

myDNAlist = ["A", "C", "T", "T", "T", "C", "G"]
dna_to_rna(myDNAlist)
print myDNAlist
['A','C','U','U','U','C','G']
```

Why did that happen??

- Immutable objects:
  - string
  - tuple
  - number
- When immutables are passed to a function, the function cannot change them (it can only assign a new object to its local name)
- Mutable objects:
  - list
  - dictionary
- When mutables are passed to a function, the function can change the internal parts

One more difficulty!

```
# warning: bad program!
mydata = [0.2, 0.22, 0.34, 0.18]
for entry in mydata :
  entry = jc_dist(entry)
print mydata
[0.2, 0.22, 0.34, 0.18]
```

Why not??

```
# warning: bad program!
mydata = [0.2, 0.22, 0.34, 0.18]
for entry in mydata :
  entry = jc_dist(entry)
print mydata
[0.2, 0.22, 0.34, 0.18]
```

- The problem is that “entry” is a copy of the item in the list
- We re-assign the copy, but that doesn’t change the list

Why not??

```
# wrong way
mydata = [0.2, 0.22, 0.34, 0.18]
for entry in mydata :
  entry = jc_dist(entry)
print mydata
[0.2, 0.22, 0.34, 0.18]

# right way
for index in range(0,len(mydata)) :
  mydata[index] = jc_dist(entry)
print mydata
[0.2326, 0.2604, 0.4529, 0.2058]
```

Summary

- Functions allow a section of code to be re-used
- The `def` statement creates a function
- The `return` statement causes it to return a value
- If there is no `return` the function returns `None`
- A function cannot change a passed-in immutable
- It can change the internal elements of a mutable (list or dictionary)

Summary

Things to beware of:

- To change items in a list, use an index, not `for element in list`
- Because strings are immutable, string functions do not change their strings
- Because lists are mutable, many list functions do change their list
- Such functions often return `None`

Modules

- Most Python programs are one main file and several modules
- Modules are additional files containing things your program can use
- We have already used the `sys` module

sys

- `import sys`
- `progname = sys.argv[0]`
- `firstarg = sys.argv[1]`

import

- `import` allows your program to use a module
- names in the module can be referred to as modulename.variablename
- module `sys` has a variable named `argv`
- when you import it, this becomes `sys.argv`

Practice problem 2

- Write a function which reads a string and either returns the string unchanged, or if it is "Jan" returns "January"
- Write a program which applies this function to every word in a file
- Print out the changed text
- Test it on a short file which contains the words "Jan", "Trojan", and "Janet" as well as some other words

Solution

```
def jan_expand(word) :
  if word == "Jan" :
    return "January"
  else :
    return word

import sys
filename = sys.argv[1]
filehandle = open(filename,"r")
linelist = filehandle.readlines()
for line in linelist :
  wordlist = line.split()
  for word in wordlist :
    print jan_expand(word),
  print "\n",
```

Practice problem 3

- Put your ”january” function in a separate file
- Import it into your main program and use it there
- The import command does NOT use the ”.py” part of the file name
- `import mymodule` not `import mymodule.py`
- Don’t forget to add the module name to the function name when you call it
- Recommendation: Give the module a different name than the function

Problem 3 solution

```
# in file "month_routines.py"
def jan_expand(word) :
  if word == "Jan" :
    return "January"
  else :
    return word
```

Problem 3 solution

```
# in file "expander.py"
import sys
import month_routines
filename = sys.argv[1]
filehandle = open(filename,"r")
linelist = filehandle.readlines()
for line in linelist :
  wordlist = line.split()
  for word in wordlist :
    print month_routines.jan_expand(word),
  print "\n",
```

Summary

Command	Meaning
def	define a function
return	return a value from a function
import	make functions and variables in a module available for use