fIntroduction to Python

Feature Summary, Session 8

COMMAND LINE

- **sys.argv** list of strings: ***access command line arguments***
  This list of strings "springs into being" at program start. It contains the program name in the first item, and any text args entered at the command line when the program was executed in subsequent items.

```
import sys
print(sys.argv)
     # ['prog.py', 'arg1']
     # arg1 is command-
     # line arg
```

- **sys.stdout.write()** or **print()**: ***write to STDOUT: send text data output to operating system's STDOUT stream***
  (by default, STDOUT writes to the screen unless redirected)

```
print('hi')
     # writes to STDOUT
     # (with '\n' added)

import sys
sys.stdout.write('hi')
     # writes to STDOUT
     # (no '\n' added)
```

- redirect STDOUT to write to a file at command line

```
python doit.py > newfile.txt
     (at command line)
```

- redirect STDOUT to another program's input at command line

```
python this.py | wc
     (at command line:
      this.py output
      passed to wc input)

ls | python numlines.py
     (at command line:
      ls output passed to
      numlines.py input)
```

- read from STDIN: ***read the STDIN input stream***

```
import sys
text = sys.stdin.read()
```

FILES AND DIRECTORIES

- writing to a file

```
fh = open('out.txt', 'w')
fh.write('this line\n')
fh.write('that line\n')
fh.close()
```

- appending to a file

```
fh = open('out.txt', 'a')
fh.write('this line\n')
fh.write('that line\n')
fh.close()
```

- **os.listdir()**: ***list entries (files and dirs) in a directory***

```
import os
d = 'mydir/'
for fname in os.listdir(d):
    print(fname)
```

- **os.path.isfile()**, **os.path.isdir()**: ***determine whether entry name or path is a dir or a file***

```
fname = 'myfile.txt'
if os.path.isfile(fname):
    print('a file!')
elif os.path.isdir(fname):
    print('a directory!')
```

- **os.path.exists()**: ***determine whether a filename or path exists***

```
fname = 'myfile.txt'
if os.path.exists(fname):
    print('exists!')
```

- **os.path.getsize()**: ***return size of a file or directory***

```
fname = 'myfile.txt'
bts = os.path.getsize(fname)
```

- **os.walk()**: ***traverse a directory tree***

```
for base, files, dirs in
    os.walk('basedir/'):
    print(base)
    print(files)
    print(dirs)
    print('===============')
```

MULTIPROCESSING

- **subprocess.call()**: ***execute an external process***

```
import subprocess
cmd = 'sendmail'
subprocess.call([cmd,'arg'])
```

- **subprocess.check_output()**: ***execute an external process, output is returned as a string***

```
c = 'ls'
subprocess.check_output([c,
                         'arg'])
```

- **multiprocessing.Process()**: ***simultaneously execute a duplicate process***

```
import multiprocessing as mp
def f():
    print("doin' stuff!")
p = mp.Process(target=f)
p.start()
print('continuing on...')
p.join()
```

EXCEPTION HANDLING

- **try/except**: handle specified exception and run routine

```
try:
    arg = sys.argv[1]
except IndexError:
    exit('enter an arg')
```

- **raise** statement: raise an exception
  this inspires the same error condition as that generated by Python

```
raise IndexError('oops')
```

SETS

- **.difference()** method: ***return a set of items in this set not found in another container***
  argument: any container (does not need to be a set)
  return value: set

```
x = {'a', 'b', 'c', 'd'}
y = ['c', 'd', 'e']
z = x.difference(y)
                 # {'a', 'b'}
```

- **.intersection()** method: ***return a set of items in this set in common with values in other container***
  argument: any container (does not need to be a set)
  return value: set

```
x = {'a', 'b', 'c'}
y = {'b', 'c', 'd'}
z = x.intersection(y)
      # {'b', 'c'}
```

- **.union()** method: ***return a set of items in this set combined with values in other container***
  argument: any container (does not need to be a set)
  return value: set

```
x = {'a', 'b', 'c'}
y = {'b', 'c', 'd'}
z = x.union(y)
      # {'a', 'b', 'c', 'd'}
```

LIST COMPREHENSIONS

- "filtering" list comprehension: ***return each value from the source list that passes a test***
  "argument": any container of objects, with test (if) expression
  return value: a list of the same item values, filtered by expression

```
x = [-2, -1, 0, 1, 2]
y = [i for i in x if i > 0]
                 # [1, 2]
```

- "transforming" list comprehension: ***apply statement to each value in the source list***
  "argument": any container of objects, with "transforming" expression
  return value: a list of values, result of passing values through expression

```
x = [1, 2, 3]
y = [ j * 2 for j in x ]
                 # [2, 4, 6]
```

- "filtering" and "transforming" list comprehension: ***combination of filtering and transforming***
  argument: any container, filtering and transforming expressions
  return value: a list of values, result of filtering/transforming expressions

```
x = [-2, -1, 0, 1, 2]
y =[j*2 for j in x if j > 0]
                 # [2, 4]
```

- list comprehension operating on a file: ***operate and/or filter each line from a file***

  "argument": file object, filtering and/or transforming expressions
  return value: a list of lines from file, filtered/transformed

```
x = [ x.rstrip()
  for x in open('file.txt')]
```

LAMBDAS

- replacing a simple sort function

```
def by_lower(arg):
    return arg.lower()
s = sorted(seq, by_lower)

s = sorted(seq, lambda arg:
                arg.lower())
```