Advanced Strings and File I/O

Strings can do operations on themselves:

.lower(), .upper(), .capitalize()

```
>>> "funKY tOwn".capitalize()
'Funky town'
>>> "funky tOwn".lower()
'funky town'
```

.split([sep [,maxsplit]])

```
>>> "funKY tOwn".split()
['funKY', 'tOwn']
>>> "funKY tOwn".capitalize().split()
['Funky', 'town']
>>> [x.capitalize() for x in "funKY tOwn".split()]
['Funky', 'Town']
>>> "I want to take you to, funKY tOwn".split("u")
['I want to take yo', ' to, f', 'nKY tOwn']
>>> "I want to take you to, funKY tOwn".split("you")
['I want to take ', ' to, funKY tOwn']
```

.strip(), .join(), .replace()

```
>>> csv_string = ‘Dog,Cat,Spam,Defenestrate,1, 3.1415 \n\t’
>>> csv_string.strip()
‘Dog,Cat,Spam,Defenestrate,1, 3.1415’
>>> clean_list = [x.strip() for x in csv_string.split(",")]
>>> clean_list
[‘Dog’, ’Cat’, ’Spam’, ’Defenestrate’, ’1’, ’3.1415’]
```

.join() allows you to glue a list of strings together with a certain string

```
>>> print ",".join(clean_list)
‘Dog,Cat,Spam,Defenestrate,1,3.1415’
>>> print "\t".join(clean_list)
Dog Cat Spam Defenestrate 1  3.1415
```

.replace() strings in strings

```
>>> csv_string = ‘Dog,Cat,Spam,Defenestrate,1, 3.1415 \n\t’
>>> alt_csv = csv_string.strip().replace(‘ ’,‘’)
>>> alt_csv
‘Dog,Cat,Spam,Defenestrate,1,3.1415’
>>> print csv_string.strip().replace(‘ ’,‘’).replace(‘,’,‘\t’)
Dog Cat Spam Defenestrate 1  3.1415
```

.find() incredibly useful for searching, returning the index of the search

```
>>> s = ‘My Funny Valentine’
>>> s.find("y")
1
>>> s.find("y",2)
7
>>> s[s.find("Funny"):]
‘Funny Valentine’
>>> s.find("z")
-1
>>> ss = [s,"Argentine","American","Quarentine"]
>>> for thestring in ss:
...   if thestring.find("tine") != -1:
...     print "’" + str(thestring) + "‘ contains ‘tine’."
...
‘My Funny Valentine’ contains ‘tine’.
‘Argentine’ contains ‘tine’.
‘Quarentine’ contains ‘tine’.
>>>
```

.string() module
exposes useful variables and functions

```
>>> import string
>>> string.swapcase("fUNKY tOWN")
'Funky Town'
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.digits
'0123456789'
```

```
import string

## let's only allow .com, .edu, and .org email domains
allowed_domains = ["com","edu","org"]

## let's nix all the possible bad characters
disallowed = string.punctuation.replace(".","")

while True:
    res = raw_input("Enter your full email address: ")
    res = res.strip()   # get rid of extra spaces from a key-happy user
    if res.count("@") != 1:
        print "missing @ sign or too many @ signs"
        continue
    username,domain = res.split("@")

    ## let's look at the domain
    if domain.find(".") == -1:
        print "invalid domain name"
        continue
    if domain.split(".")[-1] not in allowed_domains:
        ## does this end as it should?
        print "invalid top-level domain...must be in " + ",".join(allowed_domains)
        continue
    goodtogo = True
    for s in domain:
        if s in disallowed:
            print "invalid character " + s
            ## cannot use continue here because then we only continue the for loop, not the while loop
            goodtogo = False


    ## if we're here then we're good on domain. Make sure that
    for s in username:
        if s in disallowed:
            print "invalid character " + s
            goodtogo = False

    if goodtogo:
        print "valid email. Thank you."
        break
```

File: checkmail.py

example: check email address

```
BootCamp> python checkemail.py
Enter your full email address: damgreen.umd.edu
missing @ sign or too many @ signs
Enter your full email address: damgreen@umdedu
invalid domain name
Enter your full email address: damgreen!@umd,.edu
invalid character ,
invalid character !
Enter your full email address: damgreen@umd.edu
valid email. Thank you.
BootCamp>
```

String Format

casting using str() is very limited Python gives access to C-line string formatting

usage: "%(format)" % (variable)

```
>>> print "My favorite integer is %i and my favorite float is %f,\n" \
" which to three decimal places is %.3f and in exponential form is %e" \
% (3,math.pi,math.pi,math.pi)
My favorite integer is 3 and my favorite float is 3.141593,
which to three decimal places is 3.142 and in exponential form is 3.141593e+00
```

common formats: f(float), i(integer), s(string), g(nicely formatting floats)

String Format

% escapes "%"

```
>>> print "I promise to give 100%% effort whenever asked of %s." % ("me")
I promise to give 100% effort whenever asked of me.
```

\+ and zero-padding

```
>>> print "%f\n%+f\n%+f\n%010f\n%10s" %
(math.pi,math.pi,-1.0*math.pi,math.pi,"pi")
3.141593
+3.141593
-3.141593
003.141593
        pi
```

String Formatting

the (new) preferred way is string.format(value0,value1,...)

```
>>> ‘on {0}, I feel {1}’.format("saturday","groovy")
‘on saturday, I feel groovy’
>>> ‘on {}, I feel {}’.format("saturday","groovy")
‘on saturday, I feel groovy’
>>> ‘on {0}, I feel {1}’.format(["saturday","groovy"])
IndexError: tuple index out of range
>>> ‘on {0}, I feel {0}’.format(["saturday","groovy"])
"on [‘saturday’, ‘groovy’], I feel [‘saturday’, ‘groovy’]"
>>> ‘on {0}, I feel {0}’.format("saturday","groovy")
‘on saturday, I feel saturday’
```

you can assign by arguments position

```
>>> ‘desire to place’.format(desire=‘Fly me’,place=‘The Moon’)
‘Fly me to The Moon’
>>> ‘desire to place or else I wont visit place.’.format(desire=‘Fly me’,place=‘The Moon’)
‘Fly me to The Moon or else I wont visit The Moon.’
>>> f = "desire": "I want to take you", "place": "funky town"
>>> ‘desire to place’.format(**f)
‘I want to take you to funky town’
```

or by name

Formatting comes after a colon (:)

```
>>> ("%03.2f" % 3.14159) == "{0:03.2f}".format(3.14159)
True
>>> "{0:03.2f}".format(3.14159,42)
'3.14'
>>> "{1:03.2f}".format(3.14159,42)
'42.00'
>>> # format also supports binary numbers
>>> "int: {0:d}; hex: {0:x}; oct: {0:o}; bin: {0:b}".format(42)
'int: 42; hex: 2a; oct: 52; bin: 101010'
```

```
format_spec ::=  [[fill]align][sign][#][0][width][,][.precision][type]
fill        ::=  <a character other than '}'>
align       ::=  "<" | ">" | "=" | "^"
sign        ::=  "+" | "-" | " "
width       ::=  integer
precision   ::=  integer
type        ::=  "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X"
```

```
>>> "{0:*^11]}".format(" meh ")
'*** meh ***'
>>> "{0:*<11]}".format(" meh ")
' meh ******'
>>> "{0:*>11]}".format(" meh ")
'****** meh '
>>> "{0:>11.2]}".format(3.1415)
'          3.1'
```

Regular Expressions

complex strings that defines search

import re

```
>>> import re
>>> emailsearch = re.compile(r"[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$
%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-
z0-9-]*[a-z0-9])?"))
>>> emailsearch.findall("damgreen@umd.edu")
['damgreen@python.org']
>>> emailsearch.findall("damgreen@umd!edu")
```

This seems horrible but it isn't really that bad. Let's do an easy example:

```
>>> import re
>>> line = "I like the number 234 and 21."
>>> print line.spit()
['I', 'like', 'the', 'number', '234', 'and', '21.']
>>> print re.split("(\d+)",line)
['I like the number ', '234', ' and ', '21', '.']
```

A much more robust version of strip, good for manipulation of desired elements of strings

File I/O (read/write)

open() and close() are built in functions

```
>>> file_stream = open(mydata.dat,"r")
<type 'file'>
>>> file_stream.close()
```

open modes: "r" (read), "w" (write), "r+" (read+update), "rb" (read as a binary,...)

Writing data: .write() or .writelines()

```
>>> f= open("test.dat","w")
>>> f.write("This is my first file I/O. Zing!")
>>> f.close()
>>> import os ; os.system("cat %s" % "test.dat")
This is my first file I/O. Zing!0
```

```
>>> f= open("test.dat","w")
>>> f.writelines(["This is my first file I/O.\n","Take that Dr. Zing!\n"])
>>> f.close() ; os.system("cat %s" % "test.dat")
This is my first file I/O.
Take that Dr. Zing!
0
```

Likewise, there is .readlines() and .read()

```
>>> f= open("test.dat","r")
>>> data = f.readlines()
>>> f.close() ; print data
This is my first file I/O.
Take that Dr. Zing!
>>>
```

```
"""
small copy program that turns a csv file into a tabbed file

  PYTHON BOOT CAMP EXAMPLE;
    created by Josh Bloom at UC Berkeley, 2010,2012 (ucbpythonclass+bootcamp@gmail.com)

"""

import os

def tabbify(infilename,outfilename,ignore_comments=True,comment_chars="#;/"):
    """
INPUT: infilename
OUTPUT: creates a file called outfilename
    """
    if not os.path.exists(infilename):
        return  # do nothing if the file isn't there
    f = open(infilename,"r")
    o = open(outfilename,"w")
    inlines = f.readlines() ; f.close()
    outlines = []
    for l in inlines:
        if ignore_comments and (l[0] in comment_chars):
            outlines.append(l)
        else:
            outlines.append(l.replace(",","\t"))
    o.writelines(outlines) ; o.close()
```

file: tabbify_my_csv.py

```
BootCamp> cat google_share_price.csv
# Date,Open,High,Low,Close,Volume,Adj Close
2008-10-14,393.53,394.50,357.00,362.71,7784800,362.71
...
BootCamp> cat google_share_price.tab
# Date,Open,High,Low,Close,Volume,Adj Close
2008-10-14 393.53 394.50 357.00 362.71 7784800 362.71
....
```

File I/O (read/write)

shutil module is preferred for copying, archiving and removing files/directories

tempfile module is used for the creation of temporary directors and files

```
>>> import tempfile
>>> tmp = tempfile.TemporaryFile() ; type(tmp)
<type 'file'>
>>> tmp = tempfile.NamedTemporaryFile(suffix=".csv",\
                                      prefix="boot",dir="/tmp",delete=False)
>>> tmp.name
'/tmp/bootG2zoE8.csv'
>>> tmp.write("# stock phrases of today's youth\nYOLO,OMG,LOL,SWAG,Python\n")
>>> tmp.close() ; import os ; os.system("cat %s" % tmp.name)
# stock phrases of today's youth
YOLO,OMG,LOL,SWAG,Python
0
```

StringIO module

handy for making file-like objects out of strings

```
>>> import StringIO
>>> myfile = StringIO.StringIO( \
"# stock phrases of today's youth\nYOLO,OMG,LOL,SWAG,Python\n")
>>> myfile.getvalue() ## get what we just wrote
"# stock phrases of today's youth\nYOLO,OMG,LOL,SWAG,Python\n"
>>> myfile.seek(0) ## go back to the beginning
>>> myfile.readlines()
["# stock phrases of today's youth\n", 'YOLO,OMG,LOL,SWAG,Python\n']
>>> myfile.close()
>>> myfile.write(''not gonna happen'')
ValueError: I/O operation on closed file
>>> myfile = StringIO.StringIO("# stock phrases of today's youth
\nYOLO,OMG,LOL,SWAG,Python\n")
>>> myfile.seek(2) ; myfile.write("silly") ; myfile.seek(0)
>>> myfile.readlines()
["# silly phrases of today's youth\n", 'YOLO,OMG,LOL,SWAG,Python\n]
```

(cStringIO is actually faster but doesn't work on some platforms)

subprocess module subprocess the preferred way to interact with other programs, as you might do on the command line

```
>>> from subprocess import *
>>> p = Popen("ls", shell=True, stdout=PIPE) # list the directory
>>> p.pid # get the process ID of the new subprocess
12121
>>> print p.stdout.readlines()
[‘Archive.zip’, ‘Day1BreakoutSolutions\n’, ‘Day1Files\n’, ‘LecturePDFs\n’,
‘Object_Oriented_I.key\n’,...]
>>> p = Popen("spamalot", shell=True, stdout=PIPE,stderr=PIPE)
>>> print p.stderr.readlines()
[‘/bin/sh: spamalot: command not found\n’]
```

it’s often advisable to wait until the subprocess has finished

```
>>> # this returns immediately
>>> p = Popen("find .. -name ‘*.py’", shell=True, stdout=PIPE,stderr=PIPE)
>>> os.waitpid(p.pid, 0) ## this will block until the search is done
[‘../py4science/examples/pyrex/trailstats/setup.py\n’,
‘../py4science/examples/qsort.py\n’,
‘../py4science/examples/quad_newton.py\n’]
```

Breakout Work

Build a command-line utility file which copies the input file to another file and:

1. reverses the ending of the file name (e.g. dave.dat is copied to dave.tad)
2. deletes every other line
3. changes every occurrence of the words: love →hate, not → is, is → not
4. sets every number to half its original value:
   - I like 3.14 and you like 2
   - I like 1.57 and you like 1
5. count the number of words “astrology” and “physics”