advanced_strings

August 20, 2016

1 Strings can do operations on themselves

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

```
In [1]: "funKY tOwn".capitalize()
```

```
Out[1]: 'Funky town'
```

```
In [2]: "funky tOwn".lower()
```

```
Out[2]: 'funky town'
```

```
In [3]: "fUNKY tOWN".swapcase()
```

```
Out[3]: 'Funky Town'
```

How you call this: > `.split([sep [,maxsplit]])`

```
In [4]: "funKY tOwn".split()
```

```
Out[4]: ['funKY', 'tOwn']
```

```
In [5]: "funKY tOwn".capitalize().split()
```

```
Out[5]: ['Funky', 'town']
```

```
In [6]: [x.capitalize() for x in "funKY tOwn".split()]
```

```
Out[6]: ['Funky', 'Town']
```

```
In [7]: "I want to take you to, funKY tOwn".split("u")
```

```
Out[7]: ['I want to take yo', ' to, f', 'nKY tOwn']
```

```
In [8]: "I want to take you to, funKY tOwn".split("you")
```

```
Out[8]: ['I want to take ', ' to, funKY tOwn']
```

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

```
In [9]: csv_string = 'Dog,Cat,Spam,Defenestrate,1, 3.1415   \n\t'
        csv_string.strip()
```

```
Out[9]: 'Dog,Cat,Spam,Defenestrate,1, 3.1415'
```

```
In [10]: clean_list = [x.strip() for x in csv_string.split(",")]
         print(clean_list)
```

```
['Dog', 'Cat', 'Spam', 'Defenestrate', '1', '3.1415']
```

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

```
In [11]: print(",".join(clean_list))
```

```
Dog,Cat,Spam,Defenestrate,1,3.1415
```

```
In [12]: print("\t".join(clean_list))
```

```
Dog         Cat         Spam        Defenestrate        1         3.1415
```

.replace() strings in strings

```
In [13]: csv_string = 'Dog,Cat,Spam,Defenestrate,1, 3.1415   \n\t'
         alt_csv = csv_string.strip().replace(' ','')
         print(alt_csv)
```

```
Dog,Cat,Spam,Defenestrate,1,3.1415
```

```
In [14]: print(csv_string.strip().replace(' ','').replace(',','\t'))
```

```
Dog         Cat         Spam        Defenestrate        1         3.1415
```

1.2 .find()

incredibly useful searching, returning the index of the search

```
In [15]: s = 'My Funny Valentine'
         s.find("y")
```

```
Out[15]: 1
```

```
In [16]: s.find("y",2)
```

```
Out[16]: 7

In [17]: s[s.find("Funny"):]

Out[17]: 'Funny Valentine'

In [18]: s.find("z")

Out[18]: -1

In [19]: 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'.
```

1.3 string module

exposes useful variables and functions

```
In [20]: import string

In [21]: string.ascii_letters

Out[21]: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

In [22]: string.digits

Out[22]: '0123456789'
```

1.4 String Formatting

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

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

```
In [23]: import math
         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 %
              % (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)
```

http://docs.python.org/release/2.7.2/library/stdtypes.html#string-formatting-operations

1.5 String Formatting

% escapes "%"

```
In [24]: 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

```
In [25]: print("%f\n%+f\n%f\n%010f\n%10s" % (math.pi,math.pi,-1.0*math.pi,math.pi,'

3.141593
+3.141593
-3.141593
003.141593
        pi
```

1.6 String Formatting

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

```
In [26]: 'on {0}, I feel {1}'.format("saturday","groovy")

Out[26]: 'on saturday, I feel groovy'

In [27]: 'on {}, I feel {}'.format("saturday","groovy")

Out[27]: 'on saturday, I feel groovy'

In [28]: 'on {0}, I feel {1}'.format(["saturday","groovy"])


        ---------------------------------------------------------------------------

        IndexError                                Traceback (most recent call last)

        <ipython-input-28-37beb7743cdb> in <module>()
    ----> 1 'on {0}, I feel {1}'.format(["saturday","groovy"])


        IndexError: tuple index out of range


In [29]: 'on {0}, I feel {0}'.format(["saturday","groovy"])
```

```
Out[29]: "on ['saturday', 'groovy'], I feel ['saturday', 'groovy']"

In [30]: 'on {0}, I feel {0}'.format("saturday","groovy")

Out[30]: 'on saturday, I feel saturday'
```

you can assign by argument position or by name

```
In [31]: '{desire} to {place}'.format(desire='Fly me',\
                                      place='The Moon')

Out[31]: 'Fly me to The Moon'

In [32]: '{desire} to {place} or else I wont visit {place}.'.format( \
                              desire='Fly me',place='The Moon')

Out[32]: 'Fly me to The Moon or else I wont visit The Moon.'

In [33]: f = {"desire": "I want to take you", "place": "funky town"}

In [34]: '{desire} to {place}'.format(**f)

Out[34]: 'I want to take you to funky town'
```

1.6.1 Formatting comes after a colon (:)

```
In [35]: ("%03.2f" % 3.14159) ==  "{:03.2f}".format(3.14159)

Out[35]: True

In [36]: "{0:03.2f}".format(3.14159,42)

Out[36]: '3.14'

In [37]: "{1:03.2f}".format(3.14159,42)

Out[37]: '42.00'

In [38]: # format also supports binary numbers
         "int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(42)

Out[38]: 'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
```

2 File I/O (read/write)

`.open()` and `.close()` are builtin functions

```
In [39]: %%file mydata.dat
         This is my zeroth file I/O. Zing!
```

```
Writing mydata.dat
```

```
In [40]: file_stream = open('mydata.dat','r') ; print(type(file_stream))
         file_stream.close()
```

```
<class '_io.TextIOWrapper'>
```

open modes: `r` (read), `w` (write), `r+` (read + update), `rb` (read as a binary stream, . . .), `rt` (read as text file)

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

```
In [41]: f= open("test.dat","w")
         f.write("This is my first file I/O. Zing!")
         f.close()
         !cat test.dat
```

```
This is my first file I/O. Zing!
```

```
In [42]: f= open("test.dat","w")
         f.writelines(["a=['This is my second file I/O.']\n","Take that Dr. Zing!\r
         f.close()
         !cat test.dat
```

```
a=['This is my second file I/O.']
Take that Dr. Zing!
```

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

```
In [43]: f= open("test.dat","r")
         data = f.readlines()
         f.close() ; print(data)
```

```
["a=['This is my second file I/O.']\n", 'Take that Dr. Zing!\n']
```

```
In [44]: %%file tabbify_my_csv.py
         """
         small copy program that turns a csv file into a tabbed file
```

```
    PYTHON BOOT CAMP EXAMPLE;
      created by Josh Bloom at UC Berkeley, 2010,2012,2013,2015 (ucbpythoncl

"""
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()
```

```
Overwriting tabbify_my_csv.py
```

```
In [45]: %run tabbify_my_csv.py
         tabbify("google_share_price.csv","google_share_price.tsv")
```

```
In [46]: !cat google_share_price.csv |head
```

```
# Date,Open,High,Low,Close,Volume,Adj Close
2008-10-14,393.53,394.50,357.00,362.71,7784800,362.71
2008-10-13,355.79,381.95,345.75,381.02,8905500,381.02
2008-10-10,313.16,341.89,310.30,332.00,10597800,332.00
2008-10-09,344.52,348.57,321.67,328.98,8075000,328.98
2008-10-08,330.16,358.99,326.11,338.11,11826400,338.11
2008-10-07,373.33,374.98,345.37,346.01,11054400,346.01
2008-10-06,373.98,375.99,357.16,371.21,11220600,371.21
2008-10-03,397.35,412.50,383.07,386.91,7992900,386.91
2008-10-02,409.79,409.98,386.00,390.49,5984900,390.49
```

```
In [47]: !cat google_share_price.tsv |head
```

```
# Date,Open,High,Low,Close,Volume,Adj Close
2008-10-14        393.53        394.50        357.00        362.71        7784800
```

```
2008-10-13        355.79        381.95        345.75        381.02        8905500
2008-10-10        313.16        341.89        310.30        332.00        10597800
2008-10-09        344.52        348.57        321.67        328.98        8075000
2008-10-08        330.16        358.99        326.11        338.11        11826400
2008-10-07        373.33        374.98        345.37        346.01        11054400
2008-10-06        373.98        375.99        357.16        371.21        11220600
2008-10-03        397.35        412.50        383.07        386.91        7992900
2008-10-02        409.79        409.98        386.00        390.49        5984900
```

3 File I/O (read/write)

`shutil` module is preferred for copying, archiving & removing files/directories
http://docs.python.org/library/shutil.html#module-shutil
`tempfile` module is used for the creation of temporary directories and files
http://www.doughellmann.com/PyMOTW/tempfile/

```
In [48]: import tempfile
         tmp = tempfile.TemporaryFile() ; type(tmp)

Out[48]: _io.BufferedRandom

In [49]: tmp = tempfile.NamedTemporaryFile(suffix=".csv",\
                                  prefix="boot",dir="/tmp",delete=False)
         print(tmp.name)

/tmp/bootuzif7_u3.csv


In [50]: tmp.write(bytes("# stock phrases of today's youth\nWassup?!,OMG,LOL,BRB,Py
         tmp.close()
         !cat $tmp.name

# stock phrases of today's youth
Wassup?!,OMG,LOL,BRB,Python


In [51]: tmp = tempfile.NamedTemporaryFile(suffix=".csv",\
                                  prefix="boot",dir="/tmp",delete=False)
         print(tmp.name)
         tmp.write(b"# stock phrases of today's youth\nWassup?!,OMG,LOL,BRB,Python\
         tmp.close()
         !cat $tmp.name

/tmp/bootmoepd5z1.csv
# stock phrases of today's youth
Wassup?!,OMG,LOL,BRB,Python
```

4 io module StringIO/BytesIO

handy for making file-like objects out of strings

```
In [52]: import io
         myfile = io.StringIO( \
            "# stock phrases of today's youth\nWassup?!,OMG,LOL,BRB,Python\n")
         myfile.getvalue()   ## get what we just wrote

Out[52]: "# stock phrases of today's youth\nWassup?!,OMG,LOL,BRB,Python\n"

In [53]: myfile.seek(0)      ## go back to the beginning
         myfile.readlines()

Out[53]: ["# stock phrases of today's youth\n", 'Wassup?!,OMG,LOL,BRB,Python\n']

In [54]: myfile.close()

In [55]: myfile.write('not gonna happen')


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last

        <ipython-input-55-87cc95864e9a> in <module>()
    ----> 1 myfile.write('not gonna happen')


        ValueError: I/O operation on closed file


In [56]: myfile = io.BytesIO(b"# stock phrases of today's youth\nWassup?!,OMG,LOL,

In [57]: myfile.seek(2)  ; myfile.write(b"silly wah wah") ; myfile.seek(0)

Out[57]: 0

In [58]: myfile.readlines()

Out[58]: [b"# silly wah wah of today's youth\n", b'Wassup?!,OMG,LOL,BRB,Python\n']
```

5 subprocess module

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

```
In [59]: from subprocess import *
         p = Popen("ls", shell=True, stdout=PIPE)  # list the directory
         p.pid  # get the process ID of the new subprocess
```

```
Out[59]: 15440

In [60]: print(p.stdout.readlines())

[b'advanced_strings.ipynb\n', b'advanced_strings.pdf\n', b'checkemail.py\n', b'goog


In [61]: p = Popen("vanRossum-Trump-2016", shell=True, stdout=PIPE,stderr=PIPE)

In [62]: print(p.stderr.readlines())

[b'/bin/sh: vanRossum-Trump-2016: command not found\n']
```

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

```
In [63]: p = Popen("find .. -name '*.py'", shell=True, stdout=PIPE,stderr=PIPE)

In [64]: os.waitpid(p.pid, 0)  ## this will block until the search is done

Out[64]: (15442, 0)
```

```
In [ ]:
```