BIG DADDY'S PYTHON TOOLBOX For 3.5

...plus a LOT LOT more at: www.wikipython.com

Data on Disk (generally useful tools)

In this document: The symbol ↳ means "yields", "results in", or "stands for". **string** ↳ a series of characters. **file** ↳ your file being written or read. **path** ↳ the path to your **file**. **file_path** ↳ **path** + **file**. ex: ↳ example

built-ins

For reading and writting data on a physical or virtual disk Python has:
2 commands: **open** ex: fileVariable = open(file_path, mode)
(opens a new or an existing file) **with** [syntax: "with open(…"] **closes file automatically*
ex : with **open(file_path, mode)** as fileVariable
5 combinable **modes**: **r, w, a, b, +** -> see table
8 file methods: file.**write**(string), write to a new file, ***add "\n"***, returns the number of characters written
file.**read**(size), get all or some data, ↳ a string only
file.**close**(), close the file, if not using **with** structure
file.**tell**(), gives index of location in file
file.**seek**(offset, *from_what*) positions file pointer,
file.**readline**(), gets a single line that ends in a newline ("\n"), retains newline except for last file line
file.**readlines()** - same as **list**(file) - reads all lines in a file to a list

modes: 'r' : read only, 'r+' : read or write
'w' : write only, 'w+' : write or read,
'a' : append, 'a+' : append or read
+ : allow read or write
b (binary): 'rb', 'rb+', 'wb', 'wb+', 'ab', 'ab+'
binary files are not addressed in this document

from_what values (optional, defaults to 0)
text files: only allows **0** - beginning of the file
binaryfiles: 0 - beginning of the file
1 - use current position
Note: offset can be negative

Path, Directory and File Tools: (*os automatically loads os.path)

module: **os & os.path** os + below
.**access**('path', **os.F_OK**) - path exists
.**getcwd**() - current working directory
.**chdir**('path') - change cwd to 'path'
.**fspath**('path') - file sys representation
.path.**abspath**(path) ↳ absolute pathname
.path.**isfile**('path') ->True/False
.path.**isdir**('path') -> True/False
.path.**join**('path', *more paths) - smart
.path.**realpath**(__file__) active script
.path. split('path') - split path head/tail
Unix and Windows only:
.**listdir**('path') - entries in path directory
.**mkdir**('path'[,mode]) - create directory
.**rename**(src, dst) rename file/directory
module: **os.path and sys**
os.path.**basename**(sys.argv[0]) - get active script name

OR:
multiple module utility consolidated in 3.5'S pathlib module

module: **shutil**
.**copyfile**(src, dst) - copy contents to..
.**copy2**(src, dst) - copy file to file
module: **sys**
.**path** - list of search path strings, path[0] is dir that called Python
module: **inspect** as in
in.**stack**()[0][1] *path+script*
in.**getfile**(in.**currentframe**())
module: **glob** as g
g.**glob**(full_path + pattern + 'recursive = False [True]')
Ex: path = "D:\\test\\"
print(g.glob(path + "**", recursive=False)
↳ files/folders in D:\test\

pathlib - this module is new in version 3.5 -
methods in "concrete paths" - selected methods
import pathlib (or, from pathlib import Path)
pp = pathlib.**P**ath **| ppp** = pp(full directory path)
pp(pathsegments) ↳ correct path structure for currently active system
pp.**cwd**() - ↳ current directory
pp.**home**() - ↳ user's home directory
pp.**chmod**(mode) - ↳ change mode\permissions
pp.**exists**(str) - ↳ Boolean - found directory
pp.**expanduser**() - expanded ~/~user constructs
ppp.**glob**(pattern) - yield all matching files
"**" ↳ all directories and subdirs
"*.*" ↳ all files in path directory
"**/*" ↳ all directories and their files
pp.**is_dir**() - ↳ Boolean - True if dir found
pp.**is_file**() - ↳ Boolean - True if file found
pp.**is_symlink**() - ↳ Boolean - path to sym link
pp.**iterdir**() - ↳ iterates path object of directories
pp.**mkdir**(mode=0o777, parents=False, exist_ok=False)
create new dir - FileExistsError is it already exists
pp.**open**(*mode='r', buffering=-1, encoding=None, errors=None, newline=None*)
pp.**read_bytes**(str) - read binary data
pp.**read_text**() - read character data
pp.**rename**(target) - rename
pp.**resolve**(strict=False) - make absolute
pp.**rglob**(pattern) - like ** in front of **.glob**
pp.**rmdir**() - remove empty directory
pp.**write_bytes**(data)- open, write, close binary
pp.**write_text**(data, *encoding=None*, *errors=None*)

Reading a file: 7 techniques
loop : step through the lines
for line in file: *(print adds extra "\n" by default)*
.readline: gets an individual line **and adds** "\n"
getaline = file.**readline()**
.read : gets all or some of the file in a single string
test = file.**read()** <- get whole file in a string **retaining newlines**
test = file.**read().splitlines**() <- puts lines as items **in a list** and **removes newlines**
txtstr = **file. read(13)** <- gets **1st** 13 **characters** (\n counts as 1 character)
list(file): read all the lines of a file **into a list**
mylist = **list(file)** <-**retains newlines** in list items
.readlines: read all the lines of a file **into a list**
mylist = **file.readlines()** <-**retains newlines** in list items - same as list(file)

line == comparison must end with "\n" for compare to succeed

Example: (with structure auto closes the file)
with open(file_path, 'r') as file_ref_variable:
list_variable = list(file_ref_variable) ↳ *whole file*

module: **JSON** (JavaScript Object Notation) - a popular **cross-platform** data interchange format: it deals with **lists** and **dictionaries**. **import json**
methods:
.**dump**(obj, fp, *many opts*) - serialize obj to file-like obj
.**dumps**(obj,*,*many opts*)
.**load**(fp,*,*many opts*)
.**loads**(str [bytes],*,*many opts*)
.**JSONDecoder**(*,*many opts*)
.**JSONEncoder**(*,*many opts*)

Decode		Encode	
object	dict	dict	object
array	list	list, tuple	array
string	str	str	string
#(int)	int	#'s	#'s
#(real)	float	True	true
true	True	False	false
false	False	None	null
null	None		

BIG DADDY'S PYTHON TOOLBOX For 3.5

www.wikipython.com

Data on Disk - 2 (generally useful tools)

the filedialog "module" in tkinter: from tkinter import filedialog
.**askdirectory** (*parent, initial directory, title*)
.**askopenfilename** (*parent, initial directory, title, filetypes*)
.**askopenfilenames** (*parent, initial directory, title, filetypes*)
.**asksaveasfilename** (*parent, initial directory, title, filetypes*)
Example:
my_types = [("jpeg files", "*.jpg"), ("all files","*.*")]
to_open = filedialog.askopenfilename (parent = top1, initialdir = os.getcwd(), title="Please select a file", filetypes=my_types)

module: **pickle** - python specific, many object types to/from binary serilization, not human readable. **Basic** pickle uses standard "with open" structure - must be opened for **binary** operations. **import pickle**
To .**dump** (save) an object/file:
pickle.dump(object-to-pickle, save-to-file, protocol=3, fix_imports=True)
EX: pickle.dump (someObj, myFile)
To .**load** (retrieve) an object/file:
pickle.load(file-to-read [, fix_imports = True][, encoding="ASCII"][, errors= "strict"])
EX: myList = pickle.load(myFile)
Create bytes object instead of writing a file **.dumps** . Read a pickled object from a bytes object with .**loads**
***lambda** functions cannot be pickled. pickle offers much more control with many additional methods.

What can be pickled: None, True, False, integers, floating point numbers, complex numbers, strings, bytes, bytearrays, tuples, lists, sets, and dictionaries containing only picklable objects, functions defined at the top level of a module (using def, not lambda) built-in functions defined at the top level of a module, classes that are defined at the top level of a module, instances of such classes whose __dict__ or result of calling __getstate__ is picklable.

module: **shelve** - **import shelve** - A "shelf" is a persistent, dictionary-like object. The shelve module provides a simple interface to **pickle** / **unpickle** objects on DBM-style database files. Not secure.
shelve.open(filename, flag='c', protocol=None, writeback=False)
Always call **Shelf.close()** explicity. (note caps and Shelf not shelve)
If writeback=True, **Shelf.sync()** writes back entries, empties cashe, syncs with object on disk. Automatic with **Shelf.close()**.
with shelve.open('spam') as db:
db['eggs'] = 'eggs'
<-see https://docs.python.org/3.6/library/shelve.html#module-shelve

module: **sqlite3** - **import sqlite3** Create **connection** object:
sq3con = sqlite3.connect ('mysqlFile.db' [,detect_types]) **or:**
sq3con = sqlite3.connect (":memory:") - to create database in RAM
A **few** key **connection object** methods: .cursor(*see below*), .close(), .iterdump(), .commit(), .rollback(),
Create **cursor** object: CurObj = sq3con.cursor() **Methods and attributes:** .fetchone(), .fetchmany(size), .fetchall(), .close(), .rowcount, .lastrowid, arraysize, description, .executemany("sql [,parameters]"), and .execute("sql [,parameters]")
EX: Curobj.execute('''CREATE TABLE table_name (col_name data_type,...)''')
Notes: sql statements are case insensitive. Multiple statements are separated by semicolons (;). SQL ignores white space. Parameters are separated by commas but a comma after the last parameter causes a error.
Create **database:** Connection creates it if it does not exist.
A few SQL commands to **.execute :** CREATE TABLE, DROP TABLE, INSERT INTO table_name VALUE(vals), ALTER TABLE, REPLACE search_str, sub_str, rep_with, UPDATE table_name, SET col_name = new_value WHERE limiting conditions, DELETE FROM *col_name* WHERE..., SELECT col_name FROM table WHERE...,
Data types (**Python**:SQL)
None:NULL **int**:INTEGER **float**:REAL **str**:TEXT **bytes**:BLOB

module: **C S V** - **c**omma **s**eparated **v**alues
import csv - use standard built-in **open**, then create a csv.reader or csv.writer object
If csvfile is a file object, open with newline="".
.reader(csvfile [,dialect='excel'] [,**fmtparams])
QUOTE_NONNUMERIC format converts unquoted fields to float values
.writer(csvfile [,dialect='excel'] [,**fmtparams])
None is written as "". Other data written as strings.
.DictReader(f, fieldnames=None, restkey=None, restval=None, dialect='excel', *args, **kwds)
.DictWriter(f, *fieldnames*, restval='', extrasaction='raise', dialect='excel', *args, **kwds)
Note: fieldnames is NOT optional.
writer constants are: QUOTE_ALL, QUOTE-NONE, QUOTE_MINIMAL,QUOTE-NONNUMERIC
cvsreader object methods are:
.__next__() usually call as **next(reader)**
.dialect read only value of dialect in use
.line_num number of lines (not records) read
.fieldnames if not passed, initialized on 1st access
cvswriter object methods are:
.writerow(row) write the row
.writerows(rows) write all rows
.dialect read only value of dialect in use
.writeheader() write a row with field names per the constructor **Basic Examples:**

```
with open('some.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(someiterable)
with open('some.csv', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
```

module: **difflib** - not useful enough to justify space here - see notes on www.wikipython.com

module:**filecmp** - compare files & directories
import filecmp as fc
fc.cmp(f1, f2, shallow=True) ↳ Boolean
fc.cmpfiles(dir1, dir2, common, shallow=True) ↳ three lists: match, mismatch, errors
compare directories: fc.dircmp(a,b,ignore=, hide=) .report() - 1 of many methods/attributes

module: **fileinput** **import fileinput** - **creates a recursive iterator for multiple files**
fileinput.input(files=None, inplace=False, backup='', bufsize=0, mode='r', openhook=None)

```
for line in fileinput.input(files):
    process(lines)
    & then repeat for each file
```

Methods also available after 1st line is read:
.filename() .fileno() .lineno() .filelineno()
.isfirstline() .isstdin() .nextfile() .close()

modules: **tarfile, zipfile, zipapp, zipimport, zlib, gzip:** these modules provide extensive support for compression and decompression of files. **tarfile** and **zipfile** could have a whole toolbox and it would not begin to address all of their options.
tarfile.open(name=None, mode='r', fileobj=None, bufsize=10240, **kwargs) *handles gzip, bz2, lzma*
ZipFile.open name mode='r', pwd=None, *, force_zip64=False) *- context manager - use the with statement (new in 3.2) will do bzip2 and lzma*