Some Key Std Lib Modules

Overview

1. os
2. glob
3. sys

The os package

The os package

- "provides a portable way of using operating system dependent functionality"
- not to be confused with the sys module
- functions to deal with permissions, users, directories, files…
  - some functions not actually portable
  - many functions deprecated
- see https://docs.python.org/2/library/os.html for a full listing

The os package: some useful functions

- First of all, access with `import os`
- `os.chdir(path)` -- changes the working directory to `path`
- `os.getcwd()` -- get current working directory
- `os.chmod(path, mode)` -- for changing the permission of `path` to `mode`
- `os.listdir(path)` -- list the files and folders in the directory specified by `path`

The os package: more useful functions

- `os.mkdir(path[, mode])` -- create dir named `path` with permissions given by mode
    - for temp directoies see the tempfile module
- `os.makedirs(path[, mode])` -- same as mkdir, but creates intermediate dirs if missing
- `os.remove(path)` -- removes a file given by `path`
- `os.rmdir(path)` -- remove a dir at the given `path` *if it is empty*
- `os.removedirs(path)` -- removes dirs in `path` argument *if empty* (**careful with this one**)
- `os.rename(src, dst)` -- rename `src` file or dir to `dst`; essentially copy (**careful on unix**)

The os package: one more useful function

`os.walk(top, topdown=True, onerror=None, followlinks=False)`

- "Generate the file names in a directory tree by walking the tree either top-down or bottom-up. For each directory in the tree rooted at directory top (including top itself), it yields a 3-tuple (dirpath, dirnames, filenames)."
- used with a for loop to iterate through the results:

```
>>> for root, dirs, files in os.walk(os.getcwd()):
...     for dir in dirs:
...         for f in files:
...             print os.path.join(root, dir, f)

# the full path of all the files in the directory tree will print
```

The os.path module

The os.path module

- Very comprehensive file path manipulation functions
- **Do not** try to preform file path manipulation on your own: use `os.path`

- *Note:* Windows uses `\` for path separators
  - `\` causes problems in strings, i.e., `\n`, `\t`, etc.
  - `r"s\t\ring"` is the literal string `s\t\ring`
  - can also escape `\` characters by using `\\`
  - python can also use paths separated by `/`, even on Windows

The os.path module: some key functions

- `os.path.abspath(path)` -- returns normalized absolute path of `path`
- `os.path.basename(path)` -- returns the filename with extension of the file at `path`
- `os.path.dirname(path)` -- returns the directory name of the file at `path`
- `os.path.split(path)` -- returns a tuple of the dir name and filename, e.g., `(dir, filename)`
- `os.path.splitext(path)` -- returns a tuple of `path` split into head and extension, e.g. `(head, ext)`

The os.path module: more key functions

- `os.path.exists(path)` -- returns True if `path` exists, False if not
- `os.path.isfile(path)` -- returns True if `path` is file and exists
- `os.path.isdir(path)` -- returns True if `path` is existing directory
- `os.path.join(path, *paths)` -- joins `paths` to path; best way to concatenate path components

glob

glob

- glob is a simple but powerful tool for finding all files in a directory with names matching a given unix-style pattern
- based on the fnmatch module operations:
  - * is a wildcard character for multiple characters
  - ? is a wildcard character for single characters
  - sequences go between []: [abcdefg] will match a, b, c, d, e, f, and g
  - ranges also go between []: [0-9] is all numbers, [a-zA-Z] is all letters
  - ! can be used in [] for not: [!a-z] is not any lowercase letters

glob example

```
import glob

# find all files in working directory starting with a letter
fileslist = glob.glob('[a-zA-Z]*.*')
# returns a list of file names, as directory was not part of query

# find all files not starting with a letter
noletterfiles = glob.glob('[!a-zA-Z]*.*')

# find all files in a directory with .shp extension
shpfiles = glob.glob('*.shp')

# find all files with 5 letter names
fiveletterfiles = glob.glob('?????.*')

# find all files and directories
everything = glob.glob('*')

# find all .shp files in a different directory beginning with Utah, ignoring case
utahfiles = glob.glob(os.path.join(directory, '[uU][tT][aA][hH]*.shp'))
# returns list of file paths, as directory was part of query
```

sys module

sys module

The sys module has loads of functions and constants for getting system parameters. Most are irrelevant for many applications.

Three stand out:

- `sys.argv`
- `sys.path`
- `sys.exit`

sys.argv

- Used to get user arguments from the command line
- Often used with other modules like argparse
- See docs for more information

sys.path

- The list of all places python will look for modules when importing
- Based on the PYTHONPATH environment variable from the OS
- we can add to it to allow importing .py files not installed to the system:

```
>>> import sys

>>> sys.path
# list of all dirs on the path

# as it is just a list, we can append another location containing myfile.py
>>> sys.path.append(PATH_TO_DIR)

# now we can import a_function from myfile.py
>>> from myfile import a_function
```

Note: python will import modules from the same directory as a script without appending path

sys.exit

- exits python
- optional argument to return exit code
  - return 0 for successful execution
  - return any other value as an error code

```
>>>import sys

>>> sys.exit(0)
```

Please see the docs for more information about these and other modules and functions.