file_systems

February 15, 2018

1 Python for File Systems

- Python os module

1.1 Executing Bash like commands with Python

1.1.1 The pwd of Python's OS Module

- `os.getcwd()` : Gets the Current Working Directory.

```
In [2]: import os
        os.getcwd()
```

```
Out[2]: '/Users/lawerencelee/py_tech_degree/file_systems'
```

The cd of Python's OS Module

- `os.chdir('<path/to/file>')` : Changes directories based on the argument it is porvided.

```
In [3]: os.chdir('..')  # Jump back one directory
        os.getcwd()
```

```
Out[3]: '/Users/lawerencelee/py_tech_degree'
```

```
In [4]: os.chdir('/Users/lawerencelee')  # Go to my home directory
        os.getcwd()
```

```
Out[4]: '/Users/lawerencelee'
```

Checking if a Path is absolute or relative

- `os.path.isabs("<path/to/file>")` : Returns True or False if path is Absolute or not.

```
In [5]: os.path.isabs('/Users/lawerencelee/')
```

```
Out[5]: True
```

```
In [9]: os.path.isabs('/lawerencelee/')  # Checks syntax not if path is real or not.
```

```
Out[9]: True
```

```
In [10]: os.path.isabs('./lawerencelee/')
```

```
Out[10]: False
```

1.2 Code Challenge

- Now I need you to write a function named absolute that takes two arguments, a path string and a root string. If the path is not already absolute, return the path with the root prepended to it. For example: absolute("projects/python_basics/", "/") would return "/projects/python_basics/" while absolute("/home/kenneth/django", "C:") would return"/home/kenneth/django".

```
import os

def absolute(path, root):
    if os.path.isabs(path):
        return path
    else:
        return root + path
```

1.3 Creating Paths

Python Documentation

- Pathlib
- Pure Path
- Concrete Path

```
In [22]: CURRENT_DIR = os.getcwd()
         os.path.join(CURRENT_DIR, 'py_tech_degree')

Out[22]: '/Users/lawerencelee/py_tech_degree'

In [23]: os.path.join(CURRENT_DIR, '..', 'fake_dir')  # Go back one directory and change to fa

Out[23]: '/Users/lawerencelee/../fake_dir'

In [25]: os.path.join(CURRENT_DIR, "examples/tree", "python")

Out[25]: '/Users/lawerencelee/examples/tree/python'
```

Pathlib This module offers classes representing filesystem paths with semantics appropriate for different operating systems. Path classes are divided between pure paths, which provide purely computational operations without I/O, and concrete paths, which inherit from pure paths but also provide I/O operations.

-pathlib Docs

```
In [16]: import pathlib
         path = pathlib.PurePath(os.getcwd())  # Created an instance of the PurePath class
         path2 = path / 'examples' / 'paths.txt'  # The division operators in pathlib are used
         path2

Out[16]: PurePosixPath('/Users/lawerencelee/examples/paths.txt')
```

Tuple of each Directory in Path

```
In [17]: path2.parts
```

```
Out[17]: ('/', 'Users', 'lawerencelee', 'examples', 'paths.txt')
```

Easily find the root of the file system (helpful if dealing with both Win and Posix)

```
In [18]: path.root
```

```
Out[18]: '/'
```

Grab a part of the path

```
In [19]: path2.parents[2]
```

```
Out[19]: PurePosixPath('/Users')
```

Get the name of the file at the end of the path

```
In [20]: path2.name
```

```
Out[20]: 'paths.txt'
```

Easily grab the file type (could be used to sort items)

```
In [21]: path2.suffix
```

```
Out[21]: '.txt'
```

1.4 Checking Directory Contents

`os.listdir` : List the contents of a directory, by default its the current working directory.

```
In [26]: os.listdir()
```

```
Out[26]: ['Music',
          'python_materials',
          '.DS_Store',
          '.CFUserTextEncoding',
          'Environments',
          'Pictures',
          '.ipython',
          'Desktop',
          'Library',
          '.cups',
          '.bash_sessions',
          'Public',
          'py_tech_degree',
          'rhewLab',
```

```
'Python-Data-Science-and-Machine-Learning-Bootcamp',
'Movies',
'zachs_writings',
'.Trash',
'jupyter_tutorials',
'.jupyter',
'Documents',
'my_py_scripts',
'.vscode',
'.bash_profile',
'Downloads',
'.python_history',
'.gitconfig',
'.bash_history',
'.viminfo']
```

`list(os.scandir())` : Outputs a list of objects representing each of the child directories, and files in the specified directory (default=Current Working Directory). `os.scandir()` produces an iterable that can be consumed which is why `list` is needed.

```
In [27]: list(os.scandir())
```

```
Out[27]: [<DirEntry 'Music'>,
          <DirEntry 'python_materials'>,
          <DirEntry '.DS_Store'>,
          <DirEntry '.CFUserTextEncoding'>,
          <DirEntry 'Environments'>,
          <DirEntry 'Pictures'>,
          <DirEntry '.ipython'>,
          <DirEntry 'Desktop'>,
          <DirEntry 'Library'>,
          <DirEntry '.cups'>,
          <DirEntry '.bash_sessions'>,
          <DirEntry 'Public'>,
          <DirEntry 'py_tech_degree'>,
          <DirEntry 'rhewLab'>,
          <DirEntry 'Python-Data-Science-and-Machine-Learning-Bootcamp'>,
          <DirEntry 'Movies'>,
          <DirEntry 'zachs_writings'>,
          <DirEntry '.Trash'>,
          <DirEntry 'jupyter_tutorials'>,
          <DirEntry '.jupyter'>,
          <DirEntry 'Documents'>,
          <DirEntry 'my_py_scripts'>,
          <DirEntry '.vscode'>,
          <DirEntry '.bash_profile'>,
          <DirEntry 'Downloads'>,
          <DirEntry '.python_history'>,
```

```
 <DirEntry '.gitconfig'>,
 <DirEntry '.bash_history'>,
 <DirEntry '.viminfo'>]
```

Grab name of a file or directory

```
In [30]: files = list(os.scandir())
         files[4].name

Out[30]: 'Environments'
```

Check if an object is a file

```
In [31]: files[4].is_file()

Out[31]: False
```

Grab stats on particular files or directories

- The `st_size` attribute could be useful in flagging files over a certain byte size.

```
In [32]: files[4].stat()

Out[32]: os.stat_result(st_mode=16877, st_ino=985677, st_dev=16777220, st_nlink=3, st_uid=501,
```

Without a context manager or using `open`, it best that you close out what your working on

```
In [35]: scanner = os.scandir()  # Notice were not using a LIST here!!!
         scanner.close()
```

Another example with Scandir

```
In [ ]: with os.scandir(path) as it:
            for entry in it:
                if not entry.name.startswith('.') and entry.is_file():
                    print(entry.name)
```

1.5 Exploring a File Tree

```
In [39]: import os

         def treewalker(start):
             total_size = 0
             total_files = 0

             for root, dirs, files in os.walk(start):
                 subtotal = sum (
                     os.path.getsize(os.path.join(root, name)) for name in files
                 )
```

```
                total_size += subtotal
                file_count = len(files)
                total_files += file_count
                print(root, "consumes", end=" ")
                print(subtotal, end=" ")
                print('bytes in', file_count, 'non-directory files')
            print(start, "contains", total_files, "files with a combined size of", total_size

        treewalker('zachs_writings')
```

```
zachs_writings consumes 1245 bytes in 1 non-directory files
zachs_writings/.ipynb_checkpoints consumes 1105 bytes in 1 non-directory files
zachs_writings contains 2 files with a combined size of 2350 bytes
```

1.6 Code Challenge

- Create a function named dir_contains takes a path to a directory and a list of file names. If all of the file names exist within that directory, return True, otherwise, return False.

```
import os


def dir_contains(path, file_list):
    dir_contents = os.listdir(path)

    for item in dir_contents:
        if item not in file_list:
            return False

    return True
```

1.7 Creating Files and Directories

`os.path.exists('<name/of/file>')` : Checks if a path already exist

```
In [40]: os.path.exists('rogers')
```

```
Out[40]: False
```

```
In [41]: os.path.exists('/Users/lawerencelee')
```

```
Out[41]: True
```

To create a file NOT ON MAC OSX use: `os.mknod('<path/to/file>')`

To create a single directory: `os.mkdir('<name>')`

To create multiple directories: `os.makedirs('<dir1/dir2/dir3>')` - YES THAT'S `MAKEDIRS`

You can tack on the `exists_ok=True` argument if you'd like python to ignore the error thrown if directory/ies already exist. Python doesn't try to recreate them, but just moves on and ignores the error.

1.8 Code Challenge

- Create a function named create_daily_dir in backup.py.
  - This function should take a string which will be a date in either year-month-day (2012-12-22) or month-day-year (12-22-2012) format.
- Use that to create a directory like 2012-12-22 (year-month-day) in the financial directory (which is in the current directory).
  - This means that by calling create_daily_dir("04-22-2017"), we'd have a directory structure like financial/2017-04-22/.

```
import os
import re

def create_daily_dir(date):
    if re.findall(r'/d{2}-/d{2}-/d{4}', date):
        day = date[3:5]
        month = date[:2]
        year = date[-4:]
        date = "{}-{}-{}".format(year, month, day)
    path = os.path.join(os.getcwd(), 'financial', date)
    os.mkdir(path)
```

1.9 Renaming Files and Directories

- os.rename- Changes the name of one item.
  - `os.rename('old_file', 'new_name')` - Example: `os.rename('child_1', 'child_one')`
  - Be sure to include the preceeding directory if working with files or use `os.renames`
  - Otherwise: `os.rename('DELETED.txt', 'child_two/') –> FileNotFoundError: [Errno 2] No such file or directory: 'DELETED.txt' -> 'child_two/'`
- os.renames
  - `os.renames('child_one/', 'new_dir/child_dir')`
- `os.replace('old_file', 'new_file')`
  - This function can take care of the some of the cross-platform issues that crop up from time to time in regards to rename/s.
  - However, it does not delete the old file as rename/s does!

1.10 Code Challenge

- Finish the function named cleanup in consistency.py. This function should take a string which will be a path to a local directory. The file names in this directory are messy. I need you to clean them up so they all follow the same pattern. Examples and further explanation are in the comments in the file below.

```
import os
import re

# Filenames consist of a username (alphanumeric, 3-12 characters)
# and a date (four digit year, two digit month, two digit day),
# and an extension. They should end up in the format
# year-month-day-username.extension.

# Example: kennethlove2-2012-04-29.txt becomes 2012-04-29-kennethlove2.txt

def cleanup(path):
    for file in os.listdir(path):
        if re.search(r'\d{4}-\d{2}-\d{2}-[a-zA-Z0-9]{3,12}(.\w+)', file) is None:
            name = re.search(r'[a-zA-Z0-9]{3,12}', file).group()
            date = re.search(r'\d{4}-\d{2}-\d{2}', file).group()
            ext = re.search(r'\.\w+', file).group()
            new_name = '{}/{}-{}{}'.format(path, date, name, ext)

            os.replace(os.path.join(path, file) , new_name)
```

```
In [56]: # files = ['2012-04-29-kennethlove2.txt', 'kennethlove2-2012-04-29.txt',
         #             'philip34king-1312-12-05.zip', 'osk18ear55-2354-12-08']
         # for file in files:
         #     os.system('touch {}'.format(file))
```

1.11 Deleting Files and Directories

- **`os.remove('<path/to/file>')`** : ONLY deletes files, NOT directories.
- **`os.rmdir('<path/to/dir>')`** or **`os.rmdirs('<path/to/dir>')`** : Deletes directories if they are EMPTY

Example:

```
for thing in os.scandir('bootstrap/js'):
    if thing.is_file():
        os.remove(thing.path)
os.rmdir('bootstrap/js')
```

1.11.1 Sending to trash rather than deleting

- To install: `pip install send2trash`
- importing with python: `from send2trash import send2trash`
- To use: `send2trash('<filename>')`

1.12 Code Challenge

- Make a function named delete_by_date. It should take date string like 2015-10-31 and delete any files in the "backups" local directory that have that date in their filename. Just like the last challenge, the files will be named in the format "year-month-day-username.extension".
- Now create a second function named delete_by_user that works similarly but deletes files that have a particular username in their filename.

```
import os

def delete_by_date(date):
    for thing in os.listdir('backups/'):
        if date in thing:
            os.remove('backups/{}'.format(thing))


def delete_by_user(user):
    for thing in os.listdir('backups/'):
        if user in thing:
            os.remove('backups/{}'.format(thing))
```

1.13 Temporary Files and Directories

- tempfile

1.13.1 Temporary Directories

In the following example:

- Creating a temporary file
- Creating a text file, and writing to it.
- After we press enter, the file and directory both disappear.
- *This won't work on Windows*

```
import os
import tempfile


with tempfile.TemporaryDirectory() as tmpdirname:
    print("Created temporary directory named {}".format(tmpdirname))
    with open(os.path.join(tmpdirname, 'temp_file.txt'), 'w') as f:
        f.write('hello Nurse\n')
    input()
```

1.13.2 Temporary Files

Works as long as you don't need to know where the files are stored.

```
import os
import tempfile
```

```
with tempfile.TemporaryFile() as tmpfile:
    tmpfile.write(b"hello\n")
    tmpfile.seek(0)
    tmpfile.read()
input()
```

This would also work and suffer from the same issue as the above example.

```
>>> fp = tempfile.TemporaryFile()
>>> fp.write(b'bello\n')
6
>>> fp.close()  # When you were ready to close, then you would type this.
```

If you need to get the filename:

```
>>> fp = tempfile.NamedTemporaryFile()
>>> fp.name
/var/folders/23/5kd5460x3qj3dfnw2htwqlxc0000gn/T/tmp1260n_bg
>>> fp.close()
```

1.13.3 Flask Skeleton Builder:

```
from distutils.util import strtobool
import os
import pathlib
import re
import unicodedata
import sys

DIRS = [
    '{project_slug}/',
    '{project_slug}/static/',
    '{project_slug}/static/img/',
    '{project_slug}/static/js',
    '{project_slug}/static/css',
    '{project_slug}/templates/',
]

FILES = {
    '{project_slug}/requirements.txt': 'requirements.txt.template',
    '{project_slug}/app.py': 'app.py.template',
    '{project_slug}/static/css/{project_slug}.css': 'project.css.template',
    '{project_slug}/static/js/{project_slug}.js': 'project.js.template',
    '{project_slug}/templates/index.html': 'index.html.template'
}
```

```
def flask_template_prepair(string):
    string = re.sub(r'\{\%', '<%', string)
    string = re.sub(r'\%\}', '%>', string)
    string = re.sub(r'\{\{', '<<', string)
    string = re.sub(r'\}\}', '>>', string)
    return string


def flask_template_repair(string):
    string = re.sub(r'\<\%', '{%', string)
    string = re.sub(r'\%\>', '%}', string)
    string = re.sub(r'\<\<', '{{', string)
    string = re.sub(r'\>\>', '}}', string)
    return string


def slugify(string):
    string = unicodedata.normalize('NFKC', string)
    string = re.sub(r'[^\w\s]', '', string).strip().lower()
    return re.sub(r'[_\-\s]+', '_', string)


def get_root():
    root = pathlib.PurePath(
        input("What's the full path where you'd like the project? ")
    )
    if not root.is_absolute():
        return get_root()
    return root


def check_delete_root(root):
    if os.path.exists(root):
        print("Path already exists!")
        try:
            delete = strtobool(input("Delete existing files/directories? [y/n] ").lower())
        except ValueError:
            return check_delete_root(root)
        else:
            if delete:
                try:
                    os.removedirs(root)
                except OSError:
                    print("Couldn't remove {}. Please delete it yourself.".format(root))
                else:
                    print("Deleted {}".format(root))
```

```
def create_dirs(root, slug):
    try:
        os.makedirs(root)
    except OSError:
        print("Couldn't create root at {}.".format(root))
        sys.exit()
    else:
        for dir in DIRS:
            try:
                os.mkdir(os.path.join(root, dir.format(project_slug=slug)))
            except FileExistsError:
                pass


def create_files(root, slug, name):
    for file_name, template_name in FILES.items():
        try:
            template_file = open(os.path.join('templates', template_name))
            file_content = template_file.read()
            file_content = flask_template_prepair(file_content)
            file_content = file_content.format(project_name=name, project_slug=slug)
            file_content = flask_template_repair(file_content)

            target_file = open(os.path.join(root, file_name.format(project_slug=slug)), 'w')
            target_file.write(file_content)
        except Exception as e:
            print(e)
        except OSError:
            print("Couldn't create {}".format(file_name.format(project_slug=slug)))
        finally:
            template_file.close()
            target_file.close()


def main():
    project_root = get_root()
    check_delete_root(project_root)
    project_name = None
    while not project_name:
        project_name = input("What's the full name for the project? ")
    project_slug = slugify(project_name)

    create_dirs(project_root, project_slug)
    create_files(project_root, project_slug, project_name)

    print("Created {} in {}\n".format(project_name, project_root))
```

```
if __name__ == "__main__":
    os.system('cls' if os.name == 'nt' else 'clear')
    main()
```

1.14 Code Challenge

- I'd like you to change how get_root works. I still want to ask for a path but if the path is relative, change it into an absolute path. You can assume that the path is relative from the current working directory. The function should always return an absolute path.

```
import pathlib
import os


def get_root():
    root = pathlib.PurePath(
        input("What's the full path where you'd like the project? ")
    )
    if not root.is_absolute():
        return '{}/{}'.format(os.getcwd(), root)
    return root
```

1.15 Code Challenge

- This challenge is a bit different from others you've probably done, so try to approach it with an open, creative mind. I made a slugify function in the last video, but that is just one approach to making a slug. I want you to make your own slugify function in slug.py. Your function should accept a string to make into an acceptable file or directory name and a path. The rules? Slugs should be unique for their path (you can't have two files or directories with the same name in the same directory), slugs shouldn't have spaces in them, and slug should start with a number, letter, or underscore. Other than that, it's up to you!

```
import os
import re


def slugify(slug, path):
    dir_contents = os.listdir(path)

    slug = re.sub(r'[_\-\s]+', '_', slug)
    if not re.search(r'[\w\d_]', slug[0]):
        slug = '_{}'.format(slug)

    if slug in dir_contents:
        if '.' in slug:
            slug = slug.split('.')
            slug = '{}_copy.{}'.format(split_slug[0], split_slug[1])
        else:
```

```
            slug = '{}_copy'.format(slug)

    return path + slug
```

```
In [35]: import re


         def slugify(string):
             string = re.sub(r'[_.\s]', '-', string).lower()
             string = re.sub(r'\\', '', string)
             return re.sub('[?#$%^&!@*]', '', string)
```

```
In [37]: slugify('Testing The FIX')

Out[37]: 'testing-the-fix'
```