# Context Managers - Efficiently Managing Resources

# using a class to show how these context managers actually work

class OpenFile:

def \_\_init\_\_(self, filename, mode):

self.filename = filename

self.mode = mode

There’re two methods that a class need to support in order to be used with with statement and they are \_\_enter\_\_() and \_\_exit\_\_() methods.

def \_\_enter\_\_(self):

self.file = open(self.filename, self.mode)

return self.file

def \_\_exit\_\_(self, exc\_type, exc\_val, traceback):

self.file.close()

with OpenFile('spam.txt', 'w') as f:

f.write('New Data')

print(f.closed) # True

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

from contextlib import contextmanager

This context manager is equivalent to our class code.

Everything before yield statement is equivalent to \_\_enter\_\_ method of our class code.

Everything after yield statement is equivalent to \_\_close\_\_ method of our class code.

@contextmanager

def open\_file(file, mode):

try:

f = open(file, mode)

yield f

finally:

f.close()

with open\_file('spam.txt', 'w') as f:

f.write('Lorem Ipsum Dolor Set Amet!')

print(f.closed) # True

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import os

from contextlib import contextmanager

# cwd = os.getcwd()

# os.chdir('Sample-Dir-One')

Since both examples involve grabbing the current working directory then changing directory; which is the set up or **\_\_enter\_\_** part and finally switching back to first directory **os.chdir(cwd)** which is the tear down or **\_\_close\_\_()** part so this code is a good candidate for context manager.

# print(os.listdir())

# os.chdir(cwd)

# cwd = os.getcwd()

# os.chdir('Sample-Dir-Two')

# print(os.listdir())

# os.chdir(cwd)

@contextmanager

def change\_dir(destination):

try:

cwd = os.getcwd()

os.chdir(destination)

yield

finally:

os.chdir(cwd)

with change\_dir('Sample-Dir-One'):

print(os.listdir())

with change\_dir('Sample-Dir-Two'):

print(os.listdir())

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The **with** statement is useful for simplifying some of the common resource management patterns.

by resources are meant:

* files
* locks
* network connections
* . . .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import sqlite3

class DataConn:

def \_\_init\_\_(self, db\_name):

self.db\_name = db\_name

def \_\_enter\_\_(self):

self.conn = sqlite3.connect(self.db\_name)

return self.conn

def \_\_exit\_\_(self, exc\_type, exc\_val, ext\_tb):

self.conn.close()

db = 'lagman2014.sqlite'

with DataConn(db) as conn:

cursor = conn.cursor()

query = '''SELECT firstname, lastname, weights

FROM master ORDER BY weight DESC LIMIT 5'''

cursor.execute(query)

results = cursor.fetchall()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_