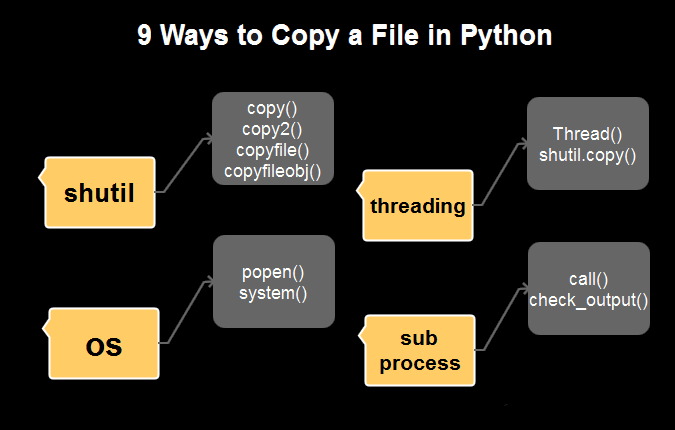
**Copy File**

****

**Shutil copyfile()**

Copies the content of the source to the destination only if the target is writable. If you don’t have the right permissions, then it will raise an **IOError**.

It works by opening the input file for reading while ignoring its file type.

copyfile(source\_file, destination\_file)

# Python Copy File - Sample Code

from shutil import copyfile

from sys import exit

source = input("Enter source file with full path: ")

target = input("Enter target file with full path: ")

# adding exception handling

try:

copyfile(source, target)

except IOError as e:

print("Unable to copy file. %s" % e)

exit(1)

except:

print("Unexpected error:", sys.exc\_info())

exit(1)

print("\nFile copy done!\n")

while True:

print("Do you like to print the file ? (y/n): ")

check = input()

if check == 'n':

break

elif check == 'y':

file = open(target, "r")

print("\nHere follows the file content:\n")

print(file.read())

file.close()

print()

break

else:

continue

**Shutil copy() method**

The**copy()** method functions like the **“cp”** command in Unix. It means if the target is a folder, then it’ll create a new file inside it with the same name (**basename**) as the source file. Also, this method will sync the permissions of the target file with the source after copying its content. It too throws the **SameFileError** if you are copying the same file.

import os

import shutil

source = 'current/test/test.py'

target = '/prod/new'

assert not os.path.isabs(source)

target = os.path.join(target, os.path.dirname(source))

# create the folders if not already exists

os.makedirs(target)

# adding exception handling

try:

shutil.copy(source, target)

except IOError as e:

print("Unable to copy file. %s" % e)

except:

print("Unexpected error:", sys.exc\_info())

**Copy() Vs Copyfile()**

1. The copy() also sets the permission bits while copying the contents whereas the copyfile() only copies the data.
2. The copy() will copy a file if the destination is a directory whereas the copyfile() will fail with error 13.
3. Interestingly, the copyfile() method utilizes the copyfileobj() method in its implementation whereas the copy() method makes use of the copyfile() and copymode() functions in turn.
4. Point-3 makes it apparent that copyfile() would be a bit faster than the copy() as the latter has an additional task (preserving the permissions) at hand.

**Shutil copy2()**

However, the **copy2()** method functions like the **copy()**. But it also gets the access and modification times added in the meta-data while copying the data. Copying the same file would result in **SameFileError**.

from shutil import \*

import os

import time

from os.path import basename

def displayFileStats(filename):

file\_stats = os.stat(basename(filename))

print('\tMode :', file\_stats.st\_mode)

print('\tCreated :', time.ctime(file\_stats.st\_ctime))

print('\tAccessed:', time.ctime(file\_stats.st\_atime))

print('\tModified:', time.ctime(file\_stats.st\_mtime))

os.mkdir('test')

print('SOURCE:')

displayFileStats(\_\_file\_\_)

copy2(\_\_file\_\_, 'testfile')

print('TARGET:')

displayFileStats(os.path.realpath(os.getcwd() + './test/testfile'))

**Os Popen() Method**

#### For Windows OS.

import os

os.popen('copy 1.txt.py 2.txt.py')

#### For Linux OS.

import os

os.popen('cp 1.txt.py 2.txt.py')

**OS System() method**

#### For Windows OS.

import os

os.system('copy 1.txt.py 2.txt.py')

#### For Linux OS.

import os

os.system('cp 1.txt.py 2.txt.py')

**Using Threading**

If you want to copy a file asynchronously, then use the below method. In this, we’ve used Python’s threading module to run the copy operation in the background.

While using this method, please make sure to employ locking to avoid deadlocks. You may face it if your application is using multiple threads reading/writing a file.

import shutil

from threading import Thread

src="1.txt.py"

dst="3.txt.py"

Thread(target=shutil.copy, args=[src, dst]).start()

**Subprocess’s Call()**

import subprocess

src="1.txt.py"

dst="2.txt.py"

cmd='copy "%s" "%s"' % (src, dst)

status = subprocess.call(cmd, shell=True)

if status != 0:

if status < 0:

print("Killed by signal", status)

else:

print("Command failed with return code - ", status)

else:

print('Execution of %s passed!\n' % cmd)

**Subprocess’s Check\_output() Method**

import os, subprocess

src=os.path.realpath(os.getcwd() + "./1.txt.py")

dst=os.path.realpath(os.getcwd() + "./2.txt.py")

cmd='copy "%s" "%s"' % (src, dst)

status = subprocess.check\_output(['copy', src, dst], shell=True)

print("status: ", status.decode('utf-8'))