COMP7004 - Systems Scripting

Lecture 15: Organising Files

Dr. Vincent Emeakaroha

vincent.emeakaroha@mtu.ie

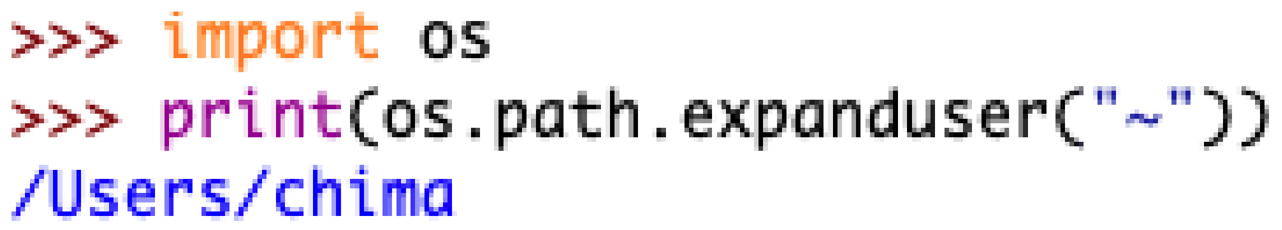
Semester 2, 2025

Objectives

- Python provides the ability to organise pre-existing files on a hard drive. This enables automating boring tasks such as:
- Making copies of all PDF files (and only the PDF files) in every sub-folder of a folder.
- Removing the leading zeros in the filename for every file in a folder of hundreds of files named as *cat001.txt*, *cat002.txt*, *cat003.txt*, and so on.
- Compressing the content of several folders into one ZIP file (which could be a simple backup system).

Accessing User Home Directory

- Python provides a function in the os module to return a user home directory.
- The function
  - os.path.exapanduser()
  - Passing tilda (~) as parameter to this function ensure interoperability across platforms.
- Example

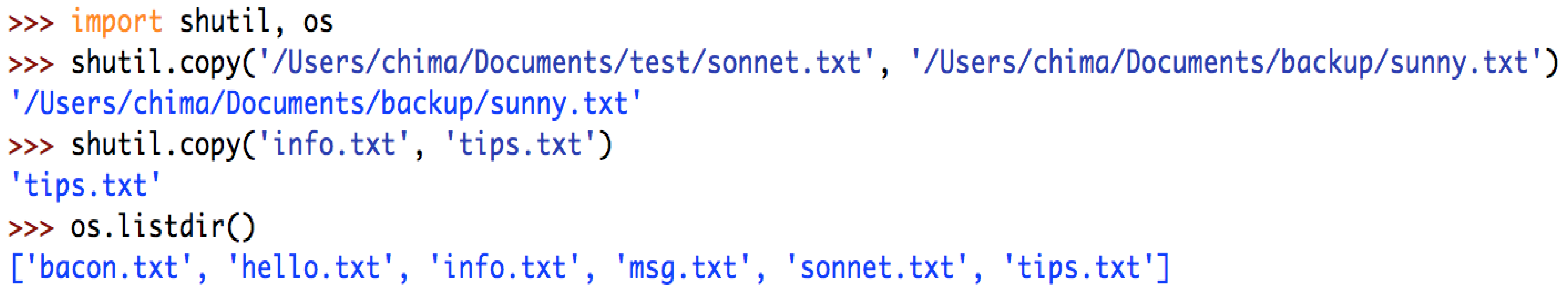


```
>>> import os
>>> print(os.path.expanduser("~"))
/Users/chima
```

Copying Files and Folders

- The shutil (or shell utilities) module provides functions for copying files, as well as entire folders.
- To copy a file, we can call shutil.copy(*source*, *destination*). This will copy the file at the path *source* to the folder at the path *destination*. Both *source* and *destination* are string data type.
- If the *destination* is a filename, it will be used as the new name of the copied file. This function returns a string of the path of the copied file.

Copying Files and Folders



```
>>> import shutil, os
>>> shutil.copy('/Users/chima/Documents/test/sonnet.txt', '/Users/chima/Documents/backup/sunny.txt')
'/Users/chima/Documents/backup/sunny.txt'
>>> shutil.copy('info.txt', 'tips.txt')
'tips.txt'
>>> os.listdir()
['bacon.txt', 'hello.txt', 'info.txt', 'msg.txt', 'sonnet.txt', 'tips.txt']
```

- While shutil.copy() will copy a single file, shutil.copytree() will copy an entire folder and every subfolders and files contained in it.
- Calling shutil.copytree(*source*, *destination*) will copy the folder at the path source, along with all of its files and subfolders, to the folder at the path destination. The source and destination parameters are both strings.
- It returns a string of the path of the copied folder.

Moving and Renaming Files and Folders

- The shutil module provides the shutil.move() function for this purpose.
- Calling shutil.move(*source*, *destination*) will move the file or folder at the path *source* to the path *destination* and will return a string of the absolute path of the new location.
- If *destination* points to a folder, the source file gets moved into destination and keeps its current filename. See following example.
- If the filename exist in the destination folder, it will be silently overwritten.

Moving and Renaming Files and Folders

```
>>> import shutil
>>> shutil.move('/Users/chima/Documents/test/tips.txt', '/Users/chima/Documents/backup')
'/Users/chima/Documents/backup/tips.txt'
```

- If the folder “backup” is not existing, the function will rename the file “tips.txt” into “backup” at the path destination. Be careful in using this function.
- The *destination* path can also specify a filename. In this case, the *source* filename will be renamed to the name specified at the *destination* path.

```
>>> import shutil
>>> shutil.move('/Users/chima/Documents/test/msg.txt', '/Users/chima/Documents/backup/userTips.txt')
'/Users/chima/Documents/backup/userTips.txt'
```

Permanently Deleting Files and Folders

- One can delete a single file or a single empty folder with functions in the os module, whereas for a non-empty folder and all of its contents, we use the shutil module.
  - Calling os.unlink(*path*) will delete a file at the specified *path*.
  - Calling os.rmdir(*path*) will delete a folder at the specified *path*. The folder must be empty of any files or folders.
  - Calling shutil.rmtree(*path*) will remove the folder at the specified *path* and all files and folders it contains will be deleted too.
- Be very careful in using these functions. It is a good idea to first print out the content intended to be deleted before calling these delete function on the content to avoid mistakenly deleting important files.

Deleting Files: Example

```
import os
for filename in os.listdir():
    if filename.endswith('.rxt')
        os.unlink(filename)
```

```
import os
for filename in os.listdir():
    if filename.endswith('.rxt'):
        #os.unlink(filename)
        print(filename)
```

- This will permanently delete all the files in the current working directory that has the ending ".rxt".
- Always print out the file names first to know exactly the files you are about to delete.

Safe Deletes with the send2trash Module

- Since Python’s built-in shutil.rmtree() function irreversibly deletes files and folders, it can be dangerous to use.
- A much better way to delete files and folders is with the third-party send2trash module.
- The send2trash module can be installed from a Terminal window by simply running python3 –m pip install send2trash.
- Using send2trash is much safer because it will send folders and files to your computer’s recycle bin or trash instead of permanently deleting them.

send2trash Module: Example

```python
import send2trash
baconFile = open('bacon.txt', 'a')
baconFile.write('Bacon is not a vegetable')
baconFile.close()
send2trash.send2trash('bacon.txt')
```

- While sending files to the recycle bin lets you recover them later, it will not free up disk space like permanently deleting them does. If you want your program to free up disk space, use the os and shutil functions to delete file and folders.
- Note that the send2trash() function can only send files to the recycle bin, it cannot pull files out of it.

Walking a Directory Tree

- Let assume you want to rename every file in some folder and also every file in every subfolder of that folder. That is, you want to walk through the directory tree touching each file as you go.
- Python provides the os.walk() function to handle this process for you. This function accepts a single value – the path of a folder.
- You can use os.walk() in a for loop statement to walk a directory tree, much like how you can use the range() function to walk over a range of numbers.

Walking a Directory Tree

- Unlike range(), the os.walk() function will return three values on each iteration through the loop:

1. A string of the current folder's name.
2. A list of strings of subfolders in the current folder
3. A list of strings of the files in the current folder

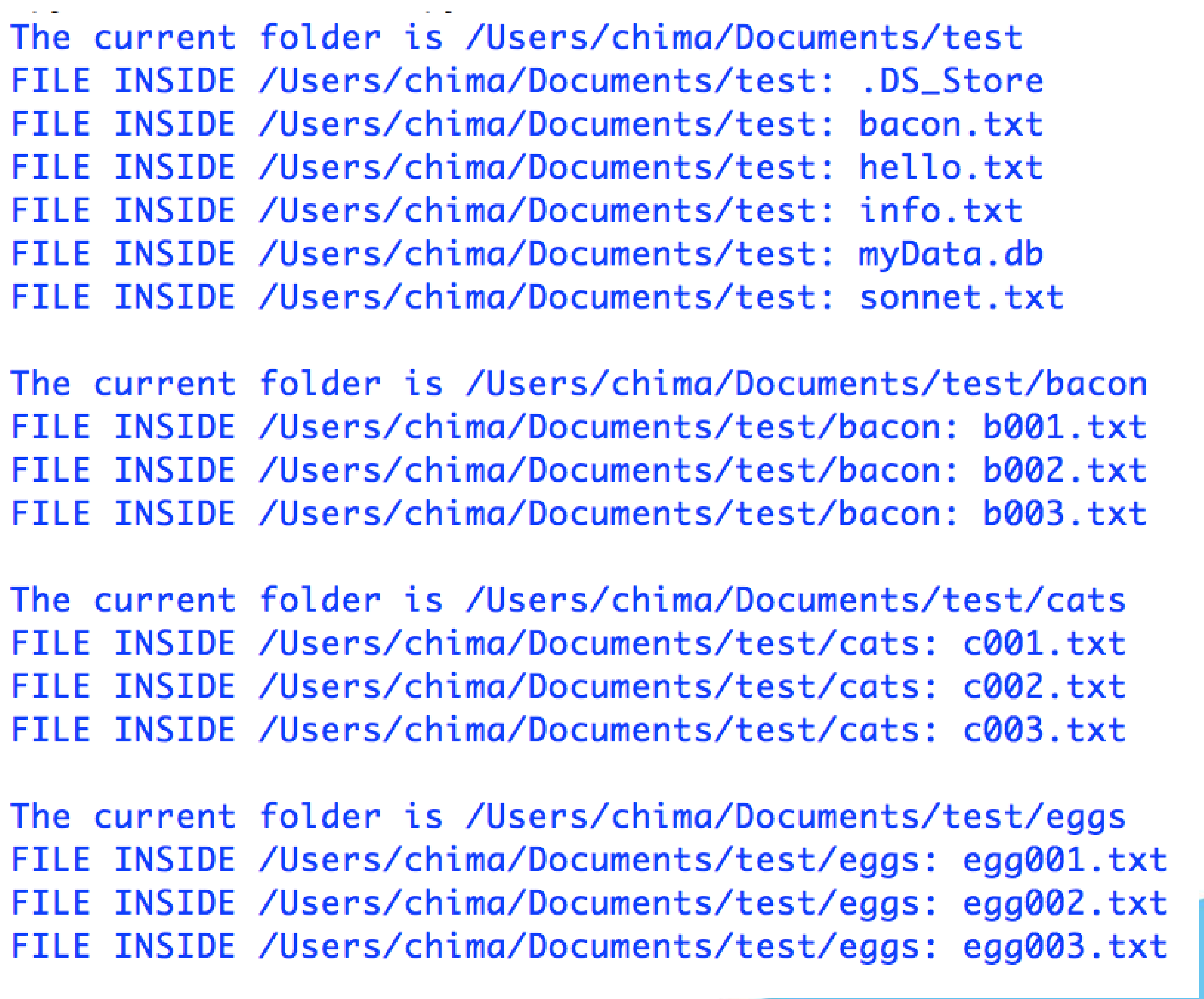
- Note that the current folder means the folder for the current iteration of the for loop. The current working directory of the program is not changed by os.walk() function.
- See the following example.

Walking a Directory Tree: Example

```python
import os

for folderName, subfolders, filenames in os.walk('/Users/chima/Documents/test'):
    print('The current folder is '+ folderName)

    for filename in filenames:
        print('FILE INSIDE ' + folderName + ': ' + filename)
    print('')
```

- Just like you can choose the variable name 'i' in a for loop, you can choose variable names for the three values.
- One can use a loop to iterate over the three variables.

Walking a Directory Tree: Example Output



```
The current folder is /Users/chima/Documents/test
FILE INSIDE /Users/chima/Documents/test: .DS_Store
FILE INSIDE /Users/chima/Documents/test: bacon.txt
FILE INSIDE /Users/chima/Documents/test: hello.txt
FILE INSIDE /Users/chima/Documents/test: info.txt
FILE INSIDE /Users/chima/Documents/test: myData.db
FILE INSIDE /Users/chima/Documents/test: sonnet.txt

The current folder is /Users/chima/Documents/test/bacon
FILE INSIDE /Users/chima/Documents/test/bacon: b001.txt
FILE INSIDE /Users/chima/Documents/test/bacon: b002.txt
FILE INSIDE /Users/chima/Documents/test/bacon: b003.txt

The current folder is /Users/chima/Documents/test/cats
FILE INSIDE /Users/chima/Documents/test/cats: c001.txt
FILE INSIDE /Users/chima/Documents/test/cats: c002.txt
FILE INSIDE /Users/chima/Documents/test/cats: c003.txt

The current folder is /Users/chima/Documents/test/eggs
FILE INSIDE /Users/chima/Documents/test/eggs: egg001.txt
FILE INSIDE /Users/chima/Documents/test/eggs: egg002.txt
FILE INSIDE /Users/chima/Documents/test/eggs: egg003.txt
```

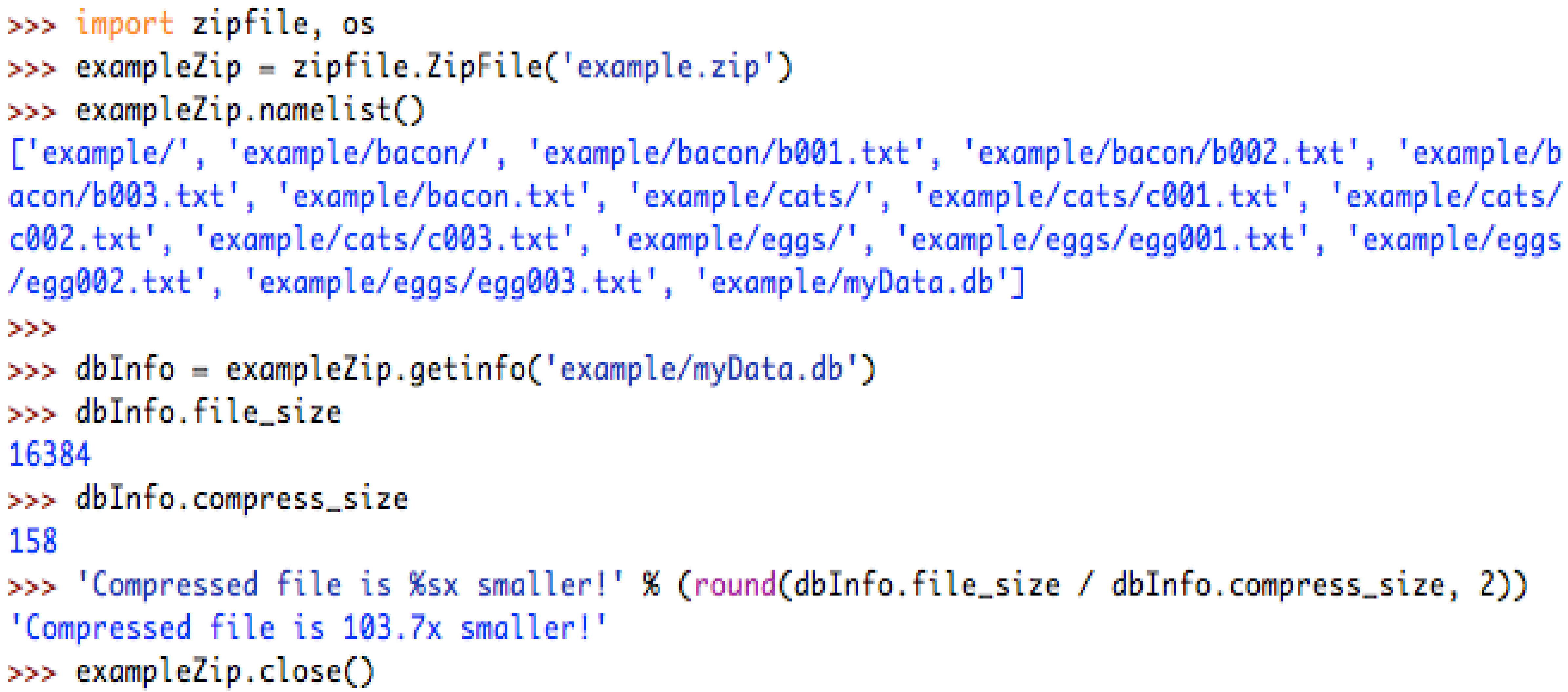
Compressing Files with the zipfile Module

- Compressing a file reduces its size, which is useful for transferring it over the Internet.
- A Zip file can contain multiple files and subfolders. So it is a handy way to package several files into one archive.
- Python can both create and open (or extract) Zip files using functions in the zipfile module.

Reading ZIP Files

- To read the content of a ZIP file, you must first create a ZipFile object (note the capital letters *Z* and *F*).
- ZipFile objects are conceptually similar to the File object we saw returned by the open() function previously. They are values through which the program interacts with the file.
- To create a ZipFile object, call zipfile.ZipFile() function, passing it a string of the .zip file’s filename.
- Note that zipfile is the name of the Python module, and ZipFile() is the name of the function.

Reading ZIP Files: Example

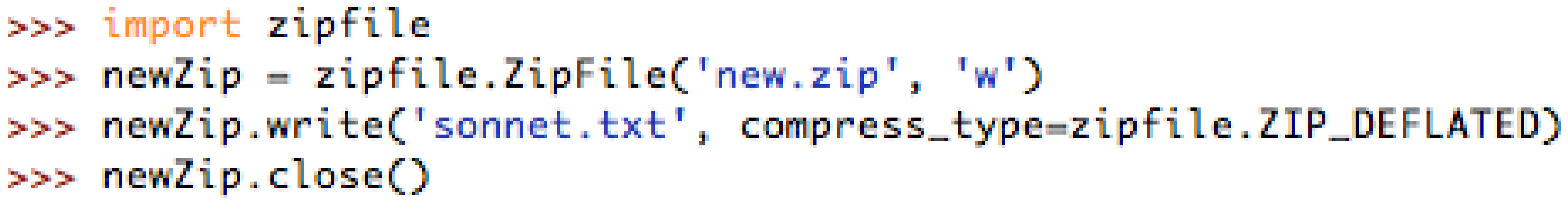


```
>>> import zipfile, os
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.namelist()
['example/', 'example/bacon/', 'example/bacon/b001.txt', 'example/bacon/b002.txt', 'example/b
acon/b003.txt', 'example/bacon.txt', 'example/cats/', 'example/cats/c001.txt', 'example/cats/
c002.txt', 'example/cats/c003.txt', 'example/eggs/', 'example/eggs/egg001.txt', 'example/eggs
/egg002.txt', 'example/eggs/egg003.txt', 'example/myData.db']
>>>
>>> dbInfo = exampleZip.getinfo('example/myData.db')
>>> dbInfo.file_size
16384
>>> dbInfo.compress_size
158
>>> 'Compressed file is %sx smaller!' % (round(dbInfo.file_size / dbInfo.compress_size, 2))
'Compressed file is 103.7x smaller!'
>>> exampleZip.close()
```

Creating and Adding to ZIP Files

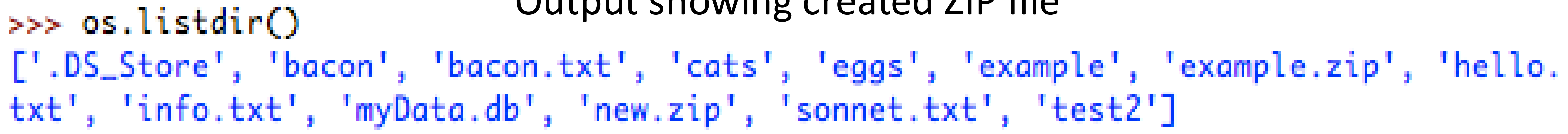
- To create your own compressed ZIP files, you must open the ZipFile object in write mode by passing 'w' as the second argument.
- When you pass a path to the write() method of a ZipFile object, Python will compress the file at that path and add it into the ZIP file.
- The write() method's first argument is a string of the filename to add. The second argument is the compression type parameter, which tell the computer what algorithm it should use to compress the file. You can just set this value to zipfile.ZIP_DEFLATED.
- See the following example:

Creating and Adding to ZIP Files



```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('sonnet.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

Output showing created ZIP file



```
>>> os.listdir()
['.DS_Store', 'bacon', 'bacon.txt', 'cats', 'eggs', 'example', 'example.zip', 'hello.
txt', 'info.txt', 'myData.db', 'new.zip', 'sonnet.txt', 'test2']
```

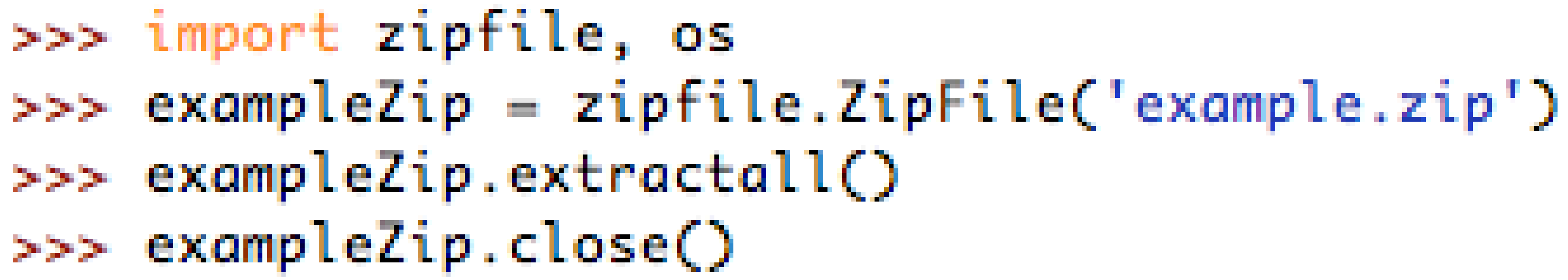
- Keep in mind that, just as with writing to files, write mode will erase all existing content of a ZIP file.
- If you want to add file to an existing ZIP file, pass 'a' as the second argument to zipfile.ZipFile() to open the ZIP file in append mode.

Succeeding Together

www.mtu.ie

Extracting from ZIP Files

- Calling the extractall() method on ZipFile objects extracts all the files and folders from a ZIP file into the current working directory.
- Example:

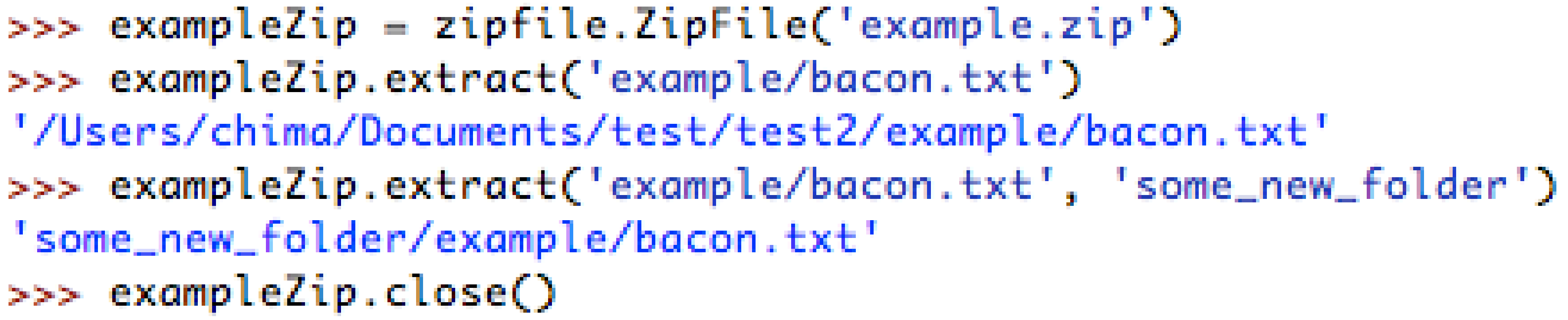


```
>>> import zipfile, os
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.extractall()
>>> exampleZip.close()
```

- Optionally, you can pass a folder name as an argument to extractall() to have it extract the files in a folder other than the current working directory. If the folder name passed does not exist, it will create it.

Extracting from ZIP Files

- The extract() method for ZipFile object will extract a single file from the ZIP file.
- Let us continue with the previous example:



```
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.extract('example/bacon.txt')
'/Users/chima/Documents/test/test2/example/bacon.txt'
>>> exampleZip.extract('example/bacon.txt', 'some_new_folder')
'some_new_folder/example/bacon.txt'
>>> exampleZip.close()
```

- The string you pass to extract() must match one of the strings in the list returned by namelist(). You can optionally, pass a second argument to extract() to extract to a folder other than the current working directory. Python will create the folder if it does not exist.

Use Case Scenario: Backing Up a Folder

- Say you are working on a project whose files you keep in a folder name /home/vin/progProject. You are worried about loosing your work, so you would like to create ZIP file snapshots of the entire folder.
- You would like to keep different versions, so you want the ZIP file's filename to increment each time it is made.
- For example:
  - progProject_1.zip, progProject_2.zip, progProject_3.zip and so on.
- You could do this by hand but it is rather annoying and you might accidentally misnumber the ZIP files' names. It would be much simpler to run a program that does this boring task for you.

Step1: Figure out the ZIP File's Name

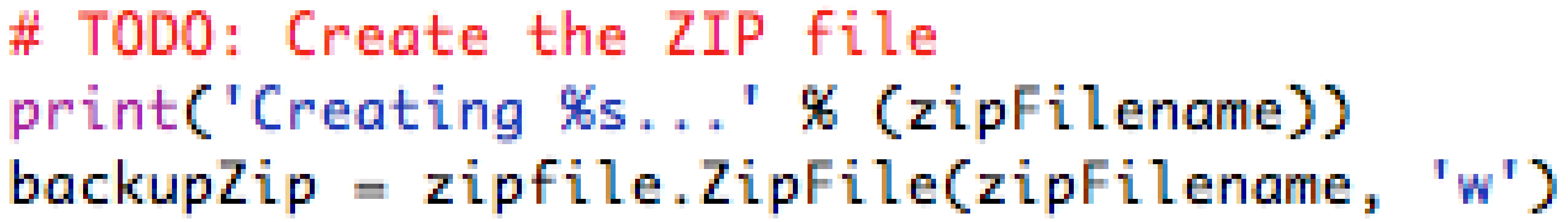
- The whole program will be written as a Python function so that it will be easy to reuse it in different Python programs.
- We will create a function that takes one parameter as argument. The parameter will represent the path to the folder name.
- To name the ZIP file, we will extract the base name from the folder path. We can then concatenate it with a running number count.
- Before a new folder with incremented count number is created, we have to first check whether it already exist using the os.path.exists() function.

Working Solution Structure

```python
#! python3
# backupToZip.py - copy an entire folder and its content into a ZIP file whose name increments

import zipfile, os
def backupToZip(folder):
    folder = os.path.abspath(folder) # make sure folder is absolute path
    # Figure out the file name
    number = 1
    while True:
        zipFilename = os.path.basename(folder) + '_' + str(number) + '.zip'
        if not os.path.exists(zipFilename):
            break
        number = number + 1

        # TODO: Create the ZIP file

        # TODO: Walk the entire folder tree and compress the files in each folder.
        print('Done')
backupToZip('/home/vin/progProject')
```

Step2: Create the New ZIP File

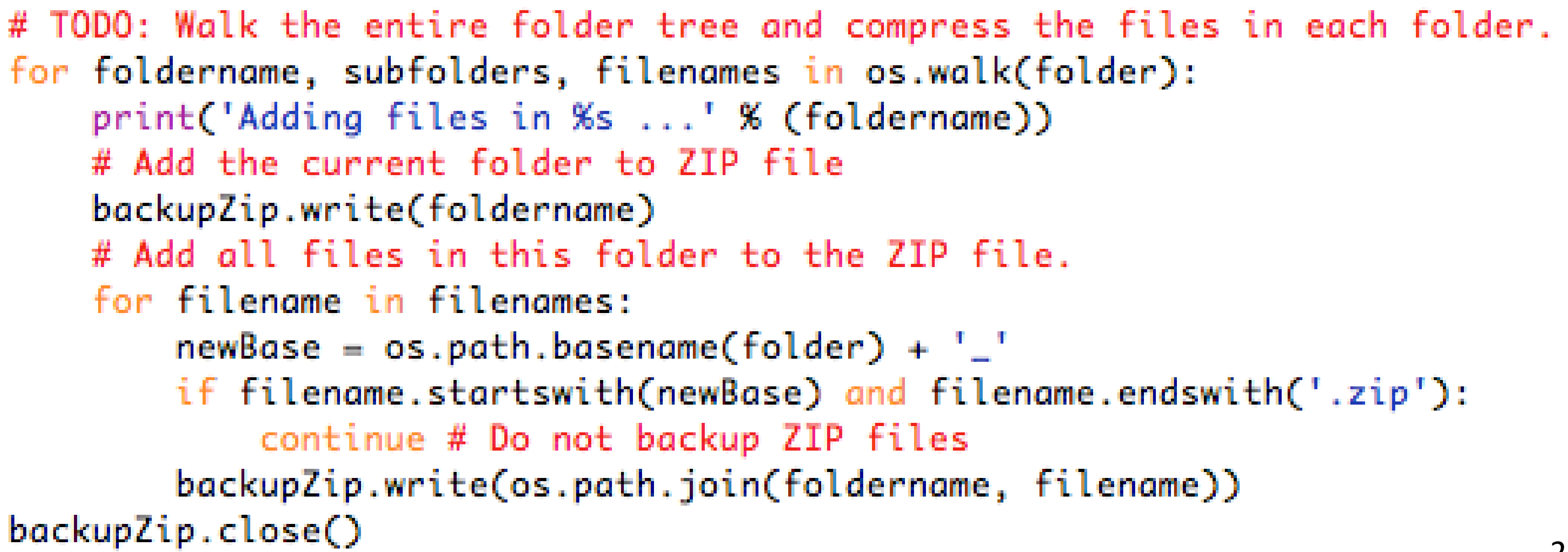
- Now that the new ZIP file's name is stored in the zipFilename variable, you can call zipfile.ZipFile() function to actually create the ZIP file.
- Be sure to pass 'w' as second argument so that the ZIP file is opened in write mode.



```
# TODO: Create the ZIP file
print('Creating %s...' % (zipFilename))
backupZip = zipfile.ZipFile(zipFilename, 'w')
```

Step3: Walk the Directory Tree and Add to ZIP File

- You can use os.walk() function in a for loop to iterate over the directories.



```
# TODO: Walk the entire folder tree and compress the files in each folder.
for foldername, subfolders, filenames in os.walk(folder):
    print('Adding files in %s ...' % (foldername))
    # Add the current folder to ZIP file
    backupZip.write(foldername)
    # Add all files in this folder to the ZIP file.
    for filename in filenames:
        newBase = os.path.basename(folder) + '_'
        if filename.startswith(newBase) and filename.endswith('.zip'):
            continue # Do not backup ZIP files
        backupZip.write(os.path.join(foldername, filename))
backupZip.close()
```

Complete Solution

```python
#! python3
# backupToZip.py - copy an entire folder and its content into a ZIP file whose name increments

import zipfile, os
def backupToZip(folder):
    folder = os.path.abspath(folder) # make sure folder is absolute path
    # Figure out the file name
    number = 1
    while True:
        zipFilename = os.path.basename(folder) + '_' + str(number) + '.zip'
        if not os.path.exists(zipFilename):
            break
        number = number + 1

    # TODO: Create the ZIP file
    print('Creating %s...' % (zipFilename))
    backupZip = zipfile.ZipFile(zipFilename, 'w')

    # TODO: Walk the entire folder tree and compress the files in each folder.
    for foldername, subfolders, filenames in os.walk(folder):
        print('Adding files in %s ...' % (foldername))
        # Add the current folder to ZIP file
        backupZip.write(foldername)
        # Add all files in this folder to the ZIP file.
        for filename in filenames:
            newBase = os.path.basename(folder) + '_'
            if filename.startswith(newBase) and filename.endswith('.zip'):
                continue # Do not backup ZIP files
            backupZip.write(os.path.join(foldername, filename))
    backupZip.close()

    print('Done')
backupToZip('/Users/chima/Documents/test/test2')
```

thank you
danke
рахмат
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
mersi
kia ora
barka
welalin
tack
spas
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
vinaka
спасибі
blagodaram
gracias
хвала
asante
manana
kiitos
dankie
dhanyavad
nanni
nandri
bedankt
maururu
köszönöm
hvala
gràcie
bayarlalaa
akun
dankon
aciú
obrigada
tenki
enkosi
djiere dieuf
tau
mochchakkeram
murakoze
chokrane
дякую
mamnun
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
sobodi
děkuji
obrigado
mèsi
sagolun
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
trugarez
didi madloba
najis tuke
ありがとう
tanemirt
grazie
rahmet
kam sah hamnida
rahmat
terima kasih
ευχαριστώ
diolch
dhanyavadagalu
shukriya
merće
мерси
তোমাকে ধন্যবাদ
감사합니다
xiexie
merci