**Lab 4:**

**Files:**

One of the simplest ways for programs to maintain their data is by reading and writing text files.

**Reading and writing:**

**A text file is a sequence of characters stored on a permanent medium like a hard drive, flash memory, or CD-ROM.**

Here you are provided with text file words.txt.

This file is in plain text, so you can open it with a text editor, but you can also read it from Python. The built-in function open takes the name of the file as a parameter and returns a file object you can use to read the file.

```
>>> fin = open('words.txt')
>>> print fin
<open file 'words.txt', mode 'r' at 0x000000000368D5D0>
>>>
```

Mode ' r ' indicates that this file is open for reading (as opposed to ' w ' for writing).

The file object provides several methods for reading, including readline , which reads characters from the file until it gets to a newline and returns the result as a string:

```
>>> fin.readline()
'aa\n'
>>>
```

The first word in this particular list is “aa”. The sequence \n represents a newline, that separate the word from the next.

The file object keeps track of where it is in the file, so if you call readline again, you get the next word:

```
>>> fin.readline()
'aah\n'
```

If we use the string method strip: we can remove the ‘\n’ part.

```
>>> line = fin.readline()
>>> word = line.strip()
>>> print word
aahed
```

You can also use a file object as part of a for loop. This program reads words.txt and prints each word, one per line:

```
fin = open('words.txt')
for line in fin:
    word = line.strip()
    print word
```

To write a file, you have to open it with mode ' w ' as a second parameter:

```
>>> fout = open('output.txt', 'w')
>>> print fout
<open file 'output.txt', mode 'w' at 0x00000000038FD5D0>
>>>
```

If the file already exists, opening it in write mode clears out the old data and starts fresh, so be careful! If the file doesn't exist, a new one is created.

The write method puts data into the file.

```
>>> line1 = "this is the first line. \n"
>>> fout.write(line1)
```

Again, the file object keeps track of where it is, so if you call write again, it adds the new data to the end.

```
>>> line2 = " and this is the second one. \n"
>>> fout.write(line2)
```

When you are done writing, you have to close the file.

```
>>> fout.close()
```

**Format Operator:**

Format operator **%**

When applied to integers, % is the modulus operator. But when the first operand is a string, % is the format operator.

The first operand is the format string, which contains one or more format sequences, which specify how the second operand is formatted. The result is a string.

```
>>> camels = 900913
>>> '%d' % camels
'900913'
>>>
```

In this example, the format sequence **' %d '** means that the second operand should be formatted as an integer ( d stands for "decimal"):

A format sequence can appear anywhere in the string, so you can embed a value in a sentence:

```
>>> google = 900913
>>> 'the %d is number for google' %google
'the 900913 is number for google'
>>> 'the number equivalent for google is %d' %google
'the number equivalent for google is 900913'
>>>
```

If there is more than one format sequence in the string, the second argument has to be a tuple. Each format sequence is matched with an element of the tuple, in order.

```
>>> 'In %d years I have spotted %g %s.' %(3,0.1, 'camels')
'In 3 years I have spotted 0.1 camels.'
```

The following example uses **' %d '** to format an integer, **' %g '** to format a floating-point number, and **' %s '** to format a string.

The number of elements in the tuple has to match the number of format sequences in the string. Also, the types of the elements have to match the format sequences:

```
>>> '%d %d %d' %(1,2)

Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    '%d %d %d' %(1,2)
TypeError: not enough arguments for format string
```

```
>>> '%d' % 'dolars'

Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    '%d' % 'dolars'
TypeError: %d format: a number is required, not str
```

In the first example, there aren't enough elements; in the second, the element is the wrong type.

### Filenames and Paths:

Files are organized into **directories** (also called "folders"). Every running program has a "current directory," which is the default directory for most operations. For example, when you open a file for reading, Python looks for it in the current directory.

The os module provides functions for working with files and directories ("os" stands for "operating system"). *os.getcwd* returns the name of the current directory:

```
>>> import os
>>> cwd = os.getcwd()
>>> print cwd
C:\Python27
```

cwd stands for "current working directory."

A string like cwd that identifies a file is called a path. A relative path starts from the current directory; an absolute path starts from the topmost directory in the file system.

To find the absolute path to a file, you can use *os.path.abspath.*

```
>>> os.path.abspath('words.txt')
'E:\\WRC\\11Python\\02Python\\07Files\\words.txt'
```

In this example we use os.path.abspath() to find the absolute path of file 'words.txt'.

```
>>> os.path.exists('words.txt')
True
```

*os.path.exists* checks whether a file or directory exists:

If it exists, *os.path.isdir* checks whether it's a directory:

```
>>> os.path.isdir('words.txt')
False
>>> os.path.isdir('music')
True
```

Similarly, *os.path.isfile* checks whether it's a file.

*os.listdir* returns a list of the files (and other directories) in the given directory.

```
>>> import os
>>> cwd = os.getcwd()
>>> os.listdir(cwd)
['07Files.docx', 'music', 'output.txt', 'words.py', 'words.txt', '~$7Files.doc
, '~WRL1958.tmp']
```

The following example "walks" through a directory, prints the names of all the files, and calls itself recursively on all the directories.

```
import os
def walk(dir):
    for name in os.listdir(dir):
        path = os.path.join(dir, name)

        if os.path.isfile(path):
            print path
        else:
            walk(path)
cwd = os.getcwd()
walk(cwd)
```

*os.path.join* takes a directory and a file name and joins them into a complete path.

Output in my case:

```
============= RESTART: E:/WRC/11Python/02Python/07Files/walk.py
E:\WRC\11Python\02Python\07Files\07Files.docx
E:\WRC\11Python\02Python\07Files\output.txt
E:\WRC\11Python\02Python\07Files\walk.py
E:\WRC\11Python\02Python\07Files\words.py
E:\WRC\11Python\02Python\07Files\words.txt
E:\WRC\11Python\02Python\07Files\~$7Files.docx
E:\WRC\11Python\02Python\07Files\~WRL1958.tmp
```

Ex:

**Exercise 14.1** Modify `walk` so that instead of printing the names of the files, it returns a list of names.

**Exercise 14.2** The `os` module provides a function called `walk` that is similar to this one but more versatile. Read the documentation and use it to print the names of the files in a given directory and its subdirectories.

### Catching Exceptions:

A lot of things can go wrong when you try to read and write files. If you try to open a file that doesn't exist, you get an IOError.

```
>>> fin = open('bad_file')

Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    fin = open('bad_file')
IOError: [Errno 2] No such file or directory: 'bad_file'
```

To avoid these errors, It is better to go ahead and try, and deal with problems if they happen, which is exactly what the ***try*** statement does. The syntax is similar to an ***if*** statement:

```
>>> try:
        fin = open('bad_file')
        for line in fin:
                print line
        fin.close()
except:
        print 'Something Went wrong'


Something Went wrong
```

Python starts by executing the ***try*** clause. If all goes well, it skips the ***except*** clause and proceeds. If an exception occurs, it jumps out of the ***try*** clause and executes the ***except*** clause. Handling an exception with a try statement is called catching an exception. In this example, the ***except*** clause prints an error message that is not very helpful.

In general, catching an exception gives you a chance to fix the problem, or try again, or at least end the program gracefully.

**Copying a file:**

The ***shutil*** Module: The ***shutil*** (or shell utilities) module has functions to let you copy, move, rename, and delete files in your Python programs. To use the ***shutil*** functions, you will first need to use import ***shutil.***

Copying Files and Folders:

The shutil module provides functions for copying files, as well as entire folders.

Calling *shutil.copy(source, destination)* will copy the file at the path source to the folder at the path *destination*. (Both *source* and *destination* are strings.) If *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.

```
import os
import shutil
os.chdir("E:\\")
shutil.copy("E:\\trial.txt", "E:\\folder1")
```

```
import os
import shutil
os.chdir("E:\\")
shutil.copy("trial.txt", "E:\\folder1\\trial_copy.txt")
```

While ***shutil.copy()*** will copy a single file, ***shutil.copytree()*** will copy an entire folder and every folder and file 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.

```
import os
import shutil
os.chdir("E:\\")
shutil.copytree("E:\\1", "E:\\backup")
```

Moving and Renaming Files and Folders:

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.

```
import os
import shutil
os.chdir("E:\\")
shutil.move("E:\\trial.txt", "E:\\1")
```

The ***destination*** path can also specify a filename. In the following example, the ***source*** file is moved and renamed.

```
import os
import shutil
os.chdir("E:\\")
shutil.move("E:\\trial.txt", "E:\\1\\trial2.txt")
```