File Operations

Before we discuss creating, reading, or editing a file, there are a few functions that are commonly used to manipulate files and directories. You may not be familiar with the computer term “directory”, but you surely know the concept, perhaps as a “file folder”.

Python Directory Manipulation Functions

Some common directory manipulation functions in the Python standard library are listed in the table. To use any of the functions, you will need to `import` the module that it is in (either the `os` module or the `shutil` module). An example using the first three functions is shown in the next section.

`os.getcwd()`
: Returns a string containing the full path of the current working directory

`os.listdir(path)`
: Returns a list of the contents of a directory

`os.mkdir(name)`
: Creates a new directory.

`os.chdir(path)`
: Change the current working directory; the path may be an absolute directory path or a relative path.

`os.rmdir(path)`
: Removes an empty directory; fails if the directory is not empty.

`os.makedirs(path)`
: Creates a directory, along with any missing parent directories.

`shutil.copytree(src, dst)`
: Copies a directory and all its contents

`shutil.rmtree(path)`
: Removes a directory and all its contains.

Python Directory Manipulation Example

Examine the command line instructions, given in bold, and the resulting output.

The first commands, on lines 1, 3 and 15 are operating system shell commands. The result of the pwd command (print working directory), is the current file path. The result of the ls command is a list of the files and directories in the current file path. It is seen that there are two files in the directory. The command on line 5 starts the Python REPL.

Lines 10, 13, 15, 16, and 18 are commands given to the Python REPL. A number of operating system commands, including the ones subsequently used in the example, are available in the os module, which is imported on line 10.

The method os.getcwd returns the full file path of Python's current working directory as a Python string object. In this case, the current working directory is the same as the result of the operating system pwd command. Similarly, os.listdir returns a Python list object populated with string objects containing the names of the files and directories within the current working directory – the same as the result of the operating system ls command.

The method os.mkdir takes a string parameter and creates a new directory named using that parameter. Calling os.listdir again shows that a new directory has been created. After exiting the Python REPL by calling the exit function, line 18, the operating system ls command confirms that the directory has been created.

```
 1 % pwd
 2 /Users/nielsenedu/python
 3 % ls
 4 hello.py        pizza.py
 5  % python3
 6 >>> import os
 7 >>> os.getcwd()
 8 '/Users/christophernielsen/python'
 9 >>> os.listdir()
10 ['hello.py', 'pizza.py']
11 >>> os.mkdir("output")
12 >>> os.listdir()
13 ['hello.py', 'pizza.py', 'output']
14 >>> exit()
15 % ls
16 hello.py        output        pizza.py
```

Python File Manipulation Functions

Common file manipulation functions in the Python standard library are listed in the table. Only the `open` function will be demonstrated by example.

`open(path)`

Opens a file and returns a file object used for reading or writing the file

`os.remove(path)`

Deletes a file by path.

`os.rename(path)`

Renames or moves a file/directory.

`shutil.copy(src, dst)`

Copies a file from source to destination.

The open function does not require a module to be imported. Examine the example command line exchange given below, with user input given in bold.

```
% python3
>>> import os
>>> os.getcwd()
'/Users/christophernielsen/python'
>>> os.listdir()
['hello.py', 'pizza.py', 'output']
>>> os.chdir('output')
>>> os.listdir()
[]
>>> hello_file = open("hello.txt", "w")
>>> hello_file.write("Hello")
5
>>> hello_file.write("World!")
6
>>> hello_file.close()
>>> exit()
% ls
hello.py        output        pizza.py
% cd output
% ls
hello.txt
% cat hello.txt
HelloWorld!
%
```

The `open` function takes two parameters. The first parameter is a string containing the file path, which may be an absolute or relative path. The second parameter is the mode to open the file. The possible modes are described in the table, below.

`"r"`	read access, starting at the beginning of the file; fails if the file does not exist
`"a"`	append (write) access, creates a new file if the file does not exist; each write is appended to the end of the file
`"w"`	write access for a new file, erasing any existing file
`"x"`	write access for a new file; fails if the file already exists
`"r+"`	read/write access; starts at the beginning of the file; fails if the file does not exist
`"a+"`	read/append access; the file pointer is initially set to the start of the file (as with `"r+"`), but any write moves the file pointer to the end of the file before writing
`"w+"`	read/write access for a new file, erasing any existing file
`"x+"`	read/write access for a new file; fails if the file already exists

Each of the modes described above are for writing text files. For binary files, the letter be is included, for example to open a text file for read/append access, the mode `"a+"` is used. To open a binary file for read/append access, either mode `"a+b"` or `"ab+"` may be used (the former is more common). The “`+`” and the “`b`” must come after the “`r`”, “`a`”, “`w`”, or “`x`”.

As mentioned before, the open function returns a file object, which can be used to modify the file by calling file object methods.

File Object Methods

The table below contains the commonly used methods used with a file object.

`f.read()`

`f.read(num)`

Reads the entire file, or a specified number of characters/bytes

`f.readline()`

Reads a single line up to and including the newline character

`f.readlines()`

Reads all the lines of the file and returns them as a list of strings

`f.write(str)`

Writes the string (or bytes for binary) to the file

`f.writelines(lines)`

Given a list of strings, lines, the method writes each string to the file. Note: newline characters are not automatically added to the strings.

`f.tell()`

Returns the current file pointer position.

`f.seek()`

Moves the file pointer to a given byte offset.

`f.close()`

Closes the file, releasing system resources.