CIS 41B
Advanced Python Programming

System Modules

De Anza College

Instructor: Clare Nguyen

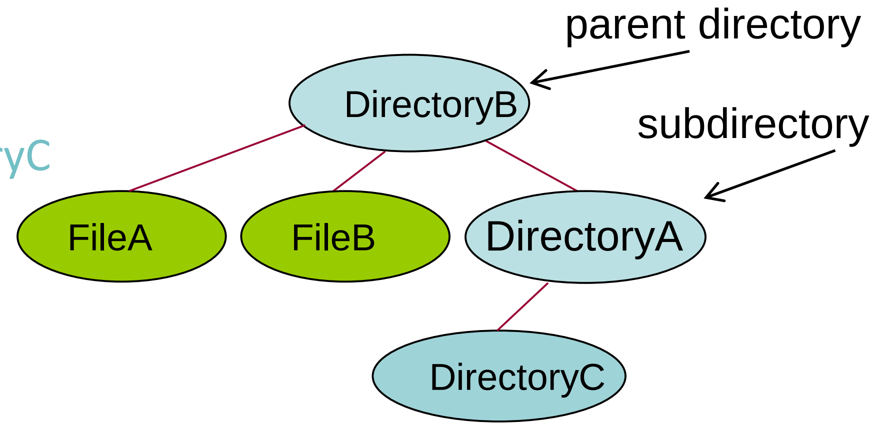
Introduction

- The system modules in Python provide us the interface to the underlying operating system that Python is running on: Windows, MacOS or Linux.
- Calling the functions of the system modules is similar to using the command window (Windows) or terminal window (Mac/Linux) to issue commands directly to the OS.
- For example, on the Mac or Linux, we can use the command `ls` at the terminal window to list all files in the current directory, and on Windows we can use the command `dir` at the command window to do the same task.
- The advantage of using the system modules is that the commands are platform independent. We don't have to think about whether to use `ls` or `dir`, we always use listdir and it will be translated to the correct command for the OS that's currently running.
- There are several system modules that we'll cover, and they all need to be imported.

File System

- The files in a system are organized into folders or directories.
- Directories themselves can be put under other directories. When a directoryA is under a directoryB, then directoryA is a subdirectory and directoryB is a parent directory.
- A path shows the location of a file or directory. A path is made up of names of directories leading to the file, separated by a slash.
  - Traditionally on Windows: \\ or \ but currently it's possible to use /
  - On Mac and Linux: /

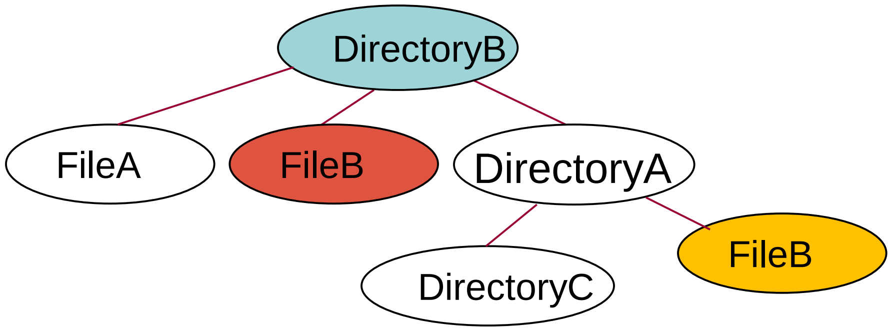
Example path:
DirectoryB/DirectoryA/DirectoryC
or
DirectoryB/FileA



- The entire file system is logically organized in a tree hierarchy, where:
  - Each directory can contain many files and subdirectories.
  - All files are terminating nodes.
  - Each file or directory has one parent directory.

Accessing Files

- When we log in to a computer system, we log in to one particular directory in the hierarchy. This directory is called our home directory.
- We can change location in the hierarchy and be at a different directory than the home directory. The directory where we are is called our current directory .
- If we refer to a file by using its name only, then by default we refer to a file in our current directory.
- To refer to files in a different directory than the current directory, then we must provide a path to the file**.**
- Example:



If our current directory is DirectoryB, then: FileB means the file immediately under DirectoryB

And to refer to the other fileB we need to use: DirectoryA/FileB

OS Module: System Information

- The following functions are related to the OS and are in the os module.
- To find the OS running on the current system:

```
os.name
```

  - Return: ‘posix’ for Mac / Linux
  - Return: ‘nt’ for Windows

- To issue a command to the OS directly, as if we’re at a command window or terminal window:

```
os.system( ‘command line’ )
```

  Python sends the ‘command line’ directly to the OS, so ‘command line’ is OS specific, it is not a Python command line.

- The information from os.name above groups the MacOS and Linux together because they are both derived from the same Unix ancestor.
  To differentiate between all 3 OS’s, we use the platform module:

```
import  platform
platform.system( )
```

Return: ‘Darwin’, ‘Linux’, or ‘Windows’

OS Module: Current Directory

- To find our current location in the directory hierarchy:

  `os.getcwd( )` cwd for current working directory
  return the path to the current directory as a string

- To change to another location in the file hierarchy

  `os.chdir( 'path' )` 'path' is the path to the new directory, as a string

- Name for the parent directory: '..' (2 periods)
- Name for the current directory: '.' (1 period)
- Name for the path separator:

  `os.sep` return '\\' (on Windows) or '/' (on Mac, Linux)
  which can be used in a path.

- Example of using chdir to change our current location up to the parent directory:

```
print(os.getcwd())      # output:  '/home/staff/cnguyen'    (on Linux)
os.chdir('..')          #  go up to parent directory
print(os.getcwd())      # output:  '/home/staff'
```

OS Module: Directories

- To list all files in a directory:

`os.listdir( )` Without input argument, return a list of files and subdirectories of current directory.

`os.listdir( 'path' )` Return a list of file and directory names in 'path' where 'path' is the location of a directory, as a string

- To list all files in a directory by using wildcards, import the glob module/

```
import  glob
glob.glob( 'path with wildcards' )
```

Wildcards:

*	0 or more characters
[ ]	1 character in the set
?	any 1 character

- To create a directory

`os.mkdir( 'name' )` Create subdirectory 'name' in the current directory

`os.makedirs( 'name1/name2' )` Create a hierarchy of as many directories as listed in the input string

  - The input directory must not already exist
  - If using a separator (slash) make sure it's the correct separator type for the current OS.

- To remove a directory `os.rmdir( 'path' )` The directory must be empty before it can be removed.

OS Module: Directory Tree Traversal

- To recursively traverse down subdirectories and list all directories and files:

```
os.walk( 'path' )
```

- The os.walk will:
  - Start at 'path' and recursively go down all subdirectories
  - At each directory, return a tuple of 3 elements:
    1. Path that leads to the directory, as a string
    2. List of names of subdirectories in the directory
    3. List of names of files in the directory
- Example of printing each directory and file at a particular path p:

```
for (path, dirList, fileList) in os.walk(p) :
    for d in dirList :
        print(os.path.join(path, d))     # print location of each directory
    for f in fileList :
        print(os.path.join(path, f))     # print location of each file
```

OS Module: Files

- To create a new, empty file or to update an existing file timestamp (not overwriting the existing file), there is no Python function that we can use. But we can take advantage of the Mac / Linux OS utility called ‘touch’:

```
os.system(‘touch  path/file’)
```

  - Only on Mac or Linux

  - For Windows there is no equivalent utility, therefore we need to open a file in append mode and close it, which will either create a new, empty file or update the file timestamp.

- To delete a file:

```
os.remove( ‘path/file’ )
```

- To rename a file or a directory:

```
os.rename( ‘source_path/file’ , ‘destination_path/file’ )
```

  - If ‘destination_path/file’ already exists and is a file, it will be overwritten on Mac / Linux. On Windows, OSError exception will be raised.
  - If ‘destination_path/file’ already exists and is a directory, OSError exception will be raised.

OS Path Module

- The following commonly used functions are in the os.path module.
- To check if a directory or a file exists

  `os.path.isdir( ‘path/file’ )` or `os.path.isfile( ‘path/file’ )` return: True/False

- To get the file size

  `os.path.getsize( ‘path/file’ )` return: size of the file (number of bytes)

- To get the absolute path of a file (path starting from root or the letter drive)

  `os.path.realpath( ‘path/file’ )` return: absolute path as a string

- To split the name of a path into a directory path and a basename:

  `os.path.split( ‘path/file’ )` return: tuple of directory path, name

- To join names together into a path, with the correct separator for the current OS:

  `os.path.join( ‘dir1’, ‘dir2’, ‘fileA’ )` return: ‘dir1/dir2/fileA’ or ‘dir1\\dir2\\fileA’

- To get to the home directory path

  `os.path.expanduser(‘~’)` return the path to the home directory

Shell Utility Module

- The following commonly used functions are in the shutil module.
- To copy a file

```
shutil.copy( 'source_path/file' , 'destination_path/file' )
```

  - If destination is an existing file, it will be overwritten
  - If destination is a directory, the source file will be copied into the directory
- To copy a directory, which means copying all its subdirectories and files

```
shutil.copytree( 'source_path' , 'destination_path' )
```

  - If destination already exists, it will cause an exception.
- To delete a directory, which means deleting all its subdirectories and files

```
shutil.rmtree( 'path' )
```

  - Use this with extreme caution!
- To move a file or directory to a new location in the file hierarchy

```
shutil.move ( 'source_path/file' , 'destination_path/file' )
```

  - If destination is an existing file, the result is the same as with os.rename
  - If the destination is an existing directory, the source is copied under the destination directory.

System Module

- The following useful system information are found in the sys module
- The version number for Python: `sys.version`
- Command line arguments are “words” that are typed on a command line to run a Python program from a terminal window:

```
python   sample.py   fileA   10        # the arguments are ‘fileA’ and ’10’
```

  In the above command line, we call on python to run the file sample.py and pass to it 2 command line arguments: ‘fileA’ and ’10’

- To access the command line in our program, use: `sys.argv`
  Return: a list of words typed on the command line

```
for n,arg in enumerate(sys.argv) :
     print(n,":",arg)
```

```
C:\Users\Clare\Desktop> python sample.py  fileA  10
0 : sample.py
1 : fileA
2 : 10
```

- The search paths that Python uses to find modules and files: `sys.path`
- The current system platform `sys.platform`

  Return: ‘darwin’, ‘linux’, ‘win32’

Tkinter File Dialog Module

- Tkinter's filedialog module provides a complete GUI window for file and directory selection

```
import  tkinter.filedialog
```

- The filedialog window allows the user to select a file or a directory in the same way as Windows' Window Explorer or Mac's Finder
- To ask for a directory:
  From an existing GUI main window or top level window:

```
directory = tk.filedialog.askdirectory(initialdir= 'path')
```

  - initialdir is the path of the starting directory that will be shown
  - directory is the path of the directory that the user selects
- To ask for a file:
  From an existing GUI main window or top level window:

```
file = tk.filedialog.askopenfilename (initialdir= 'path')
```

  - initialdir is the path of the starting directory that will be shown
  - file is the path of the file that the user selects

Going further…

- The system modules have many other functions and variables that we can use in our Python program to interact with or fetch data from the OS.
- Some of the functions / data in these system modules are platform dependent because the 3 OS's have different behavior. Some functions are only available on Windows, others are only available for Linux, etc.
- See os, os.path, shutil, and sys for more detail.

Up next: Network