

# The sys, shutil, glob, and os modules

An introduction to common  
Python modules

1

This video will introduce the sys, shutil, glob, and os modules.

# The sys module

```
import sys
```

- To get run-time parameters...

```
sys.argv[x]
```

← use for ArcScripts, more  
on this in a few weeks

- To get file path name of script...

```
sys.argv[0]
```

- To force script to quit (either for successful completion or encountered error):

```
sys.exit()
```

2

The **sys** module contains a few tools that are often useful and will be used in this course. The first tool is the **argv** function which allows the script to request a parameter when the script is run. We will use this function in a few weeks when importing scripts into ArcGIS.

If a zero is specified for the argv function, then it will return the location of the script on the computer. This can be helpful if the script needs to look for a supplemental file, such as a template, that is located in the same folder as the script.

The **exit** function stops the script when it reach the exit statement. The function can be used to stop a script when a certain condition occurs – such as when a user specifies an invalid parameter. This allows the script to stop gracefully rather than crashing. The exit function is also useful when troubleshooting a script.

# The shutil module

```
import shutil
```

- Tools for copying files and folders.

- Copy file to new location...

```
shutil.copy(original_file, copy_file)
```

- Copy entire folder to new location...

```
shutil.copytree(original_folder, copy_folder)
```

- Delete entire folder including contents...

```
shutil.rmtree(folder) Note: folder + contents gone  
for good (skips recycling bin)
```

3

The **shutil** module contains tools for moving and deleting files.

The **copy** tool makes a copy of a file in a new location.

The **copytree** tool makes a copy of an entire folder in a new location.

The **rmtree** tool deletes an entire folder. Note that this tool will not send items to the recycling bin and it will not ask for confirmation. The tool will fail if it encounters any file locks in the folder.

# The os module

```
import os
```

- Miscellaneous operating system functions.
- Create folder (will create all necessary intermediate folders)...  
`os.makedirs(folder)`
- List basenames of items in folder...  
`os.listdir(folder)`
- Delete folder (must be empty)...  
`os.rmdir(folder)`
- Delete file...  
`os.remove(file)`
- Rename file...  
`os.rename(old, new)`

The **os** module contains operating system functions – many of which are often useful in a script.

The **makedirs** tool will create a new folder including any intermediate folders.

The **listdir** tool will list the basenames of the files located in a folder.

The **rmdir** tool will delete an empty folder.

The **remove** tool will delete a file

And the **rename** tool will change the name and location of the original file to the new name and location that is specified.

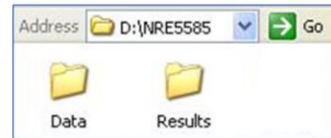
## Example creating / removing folders

- Have script create workspace for temp data.  
Delete temp workspace when finished.

```
tempWS = r"D:\NRE5585\Temp"
```

```
os.makedirs(tempWS) →
```

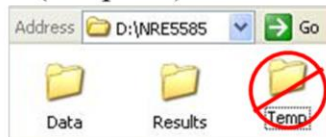
```
arcpy.env.workspace = tempWS
```



# Delete tempWS when finished with it...

```
shutil.rmtree(tempWS)
```

← deletes folder  
contents



5

Let's consider a few applications of the tools that we've discussed so far. In this example, we'll have the script create a temporary workspace where it will store intermediate files. At the end of the script, the entire temp workspace will be deleted.

This code defines a temp workspace location

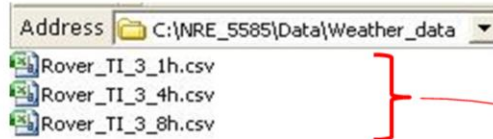
and creates it using the **os.makedirs** tool.

Arcpy's default workspace is set to the temp workspace.

At the end of the script, the **shutil.rmtree** tool is used to delete the temp workspace.

## Example: processing files in a workspace

- When multiple files, in same workspace, need to be processed in same way...



```
inWksp = r"C:\NRE_5585\Data\Weather_data"
```

```
fileList = os.listdir(inWksp)
```

```
>>> fileList
```

```
['Rover_TI_3_1h.csv', 'Rover_TI_3_4h.csv', 'Rover_TI_3_8h.csv']
```

6

In this example, the script will process a set of files contained in the same workspace.

The **os.listdir** tool gets a list of the files located in the workspace.


## Example: processing files in a workspace continued

```
>>> fileList
['Rover_TI_3_1h.csv', 'Rover_TI_3_4h.csv', 'Rover_TI_3_8h.csv']
```

- Process `fileList` with loop...

```
for fileName in fileList:
    # combine inWksp and fileName
    File = os.path.join(inWksp, fileName)
    # process file as needed...
```

```
>>> File
'C:\\NRE_5585\\Data\\Weather_data\\Rover_TI_3_1h.csv'
```



7

Now that we have a list of file names,

we can process those files using a **for loop**.

In the loop, we need to join the file basename to its workspace so that the script can find the file.

The **os.path.join** tool is convenient for building a full file pathname from a workspace and basename.

The loop can then process the file as needed

## Glob module – listing files in a workspace

- Equivalent to `os.listdir` but allows the use of a wildcard so only specific file types are included.

```
import glob
wksp = r"C:\NRE_5585\Data\*.shp"
fileLst = glob.glob(wksp)

>>> fileLst
['C:\NRE_5585\Data\APAWELL.shp',
 'C:\NRE_5585\Data\Colchester.shp'...]
```

wildcard character

8


The **glob** module will get a list of files in a workspace. It is similar to the `os.listdir` tool except that `glob` allows the use of a wild card.

The asterisk is a wild card character that can be used in place of any part of the file name – the `glob` tool returns any file names that satisfy the portion of the file name that was specified.



# The os.path module

```
from os.path import *
```

- Tools for working with file pathnames.
  - Check if file or folder exists:  
exists(path)
  - Get file basename...  
basename(file)
  - Get file directory...  
dirname(file)
  - Get file size (bytes)...  
getsize(file)
  - Join file components...  
join(path, basename)
- “C:\NRE\_5585”      “fc.shp”
- 

9

The **os.path** module contains tools for working with file path names. Note that here I've explicitly imported all tools in the os.path module so you will not see the module name appear in the statements below.

The **exists** tool returns a true or false value depending on whether the script is able to find the specified file or workspace.

The **basename** tool will return the basename of a file path name.

The **dirname** tool will return the workspace in a file path name.

The **getsize** tool will return the file size in bytes.

The **join** tool will join a workspace and a basename to create file path name.

## The os.path module continued

- Split extension from pathname (list output)...

`splitext(file)` → ["C:\NRE\_5585\fc", ".shp"]

- Split path from pathname (list output)...

`split(file)` → ["C:\NRE\_5585", "fc.shp"]

- Get time file was last accessed...

`getatime(file)`

- Get time file was last modified...

`getmtime(file)`

- Get file creation time...

`getctime(file)`

Time since epoch.  
See time module  
(`ctime` or `strptime`)  
to convert to date-  
time.

10

The **splitext** tool will return a list in which the file extension is separated from the file pathname.

The **split** tool will return a list containing both the file workspace and the basename.

There are also tools that will get the times that the file was last accessed, last modified, and created.

These times are in seconds since a reference time called an epoch. We'll learn how to convert these times to a more meaningful format when we discuss the time module in the next video.

## Setting python's working directory

- Equivalent to setting the arcpy workspace
  - works for non-arcpy operations

```
import os, os.path
wksp = r"C:\NRE_5585\Data"
os.chdir(wksp)

>>> os.path.exists("TOWNS.shp")
True
```

11

The `os` module's **`chdir`** command allows you to change Python's default workspace (a.k.a working directory). This is equivalent to setting the arcpy workspace – however, Python's default workspace only applies to Python tools. Recall that arcpy's workspace setting only applies to arcpy tools.

## Example script: os.path module

Parse workspace and basename from file for use in ArcTool.

```
import os.path, arcpy
outputFC = r"C:\NRE_5585\Results\newPolygons.shp"
outFC_dir = os.path.dirname(outputFC)
outFC_name = os.path.basename(outputFC)
arcpy.CreateFeatureclass_management(
    outFC_dir, outFC_name, "POLYGON")
```

12

Let's look at an example of working with the os.path module. In this example, the workspace and basename were parsed (i.e. extracted) from a file in order to provide the required input parameters for the CreateFeatureclass ArcTool. The **dirname** tool was used to extract the workspace and the **basename** tool was used to extract the basename of the file. Note that the same task could have been accomplished using the os.path's **split** tool – there are often a number of ways to accomplish a given task in a script. Having the script parse information from a file name can help reduce the number of parameters that a user must specify when running the script.