Lesson Title

Aidan Hunt, University of Washington

Learning Objectives

After this lesson, students will be able to

- Understand and identify the unique components of a file path.
- Construct file paths using the `os` package and navigate directories.
- Utilize the `glob` package to find filenames matching a pattern.

Check-in

- Assignment cadence for the rest of the quarter
- Final assignment
- Syllabus (grading)
- Expectations

Framing

One of our overall goals as programmers is writing scalable code

- Code that works for a wide range of scenarios
- E.g. functions help us do this by eliminating redundancy and adding flexibility

So far we have worked with individual files.

- Download the file
- Hard-code the file name into your script
- Load and process

In your careers, you will likely work with batches of files.

- For example, data files collected using a particular sensor or instrument.
- Spreadsheets with a particular format.
- Image data

Goal for this week: develop techniques for working with files systematically and efficiently.

To work with files, we'll import the `os` module:

In [1]:
```
import os
```

Directories

A directory is a location on your computer where files are stored. A folder is an example of a directory.

The **current working directory** or **present working directory** is the current directory that Python is running from.

For example, for these lecture notes:

In [117…
```
# Get the current working directory
os.getcwd()
```

Out[117]:
```
'C:\\Users\\Aidan Hunt\\MREL Dropbox\\Aidan Hunt\\STEP-UP - ME 498\\Lectures\\Lecture 11 - Working with Files'
```

Two Notes:

- The `os` module returns paths as strings (`str`)
- The file separator character depends on your operating system. For me on Windows, it is backslash: `\`. `\` has special meaning in Python and is used for special text commands (for example, `\t` does not print "\t", but instead creates a tab). Therefore, to tell Python that we really want a `\` character, we must use two: `\\`. To find the file separator character for your machine:

In [118…
```
# Get the file separator
os.sep
```

Out[118]:
```
'\\'
```

Python can see all files within the current working directory without us needing to specify any additional information.

In [119…
```
# List all the files in the current working directory
os.listdir()
```

Out[119]:
```
['.ipynb_checkpoints',
 'Additional Files',
 'example1.txt',
 'expFileManager.py',
 'L11 - Working with Files.ipynb']
```

We have used this to load .txt files during lectures and assignments; if a file is located in the same directory as your Python script, then the file can be found simply by specifying the file name.

In [120…
```
# Define a quick function for reading text files
def readFile(fileName):
    '''
    Reads a text file and prints out its contents.
    '''
    with open(fileName, 'r') as file:
        data = file.read()

    print(data)


# Specify file name
fileName = 'example1.txt'
readFile(fileName)
```

```
Hello! These are the contents of an example file.
```

But what if we want to access a file that is not in the same directory as our script? For example, consider two additional text files named `example2.txt` and `example3.txt` that are located in the subfolder `Additional Files`.

In [121…
```
# Try to read directly.
fileName = 'example2.txt'

# Open file and read the text
with open(fileName, 'r') as file:
    data = file.read()

print(data)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12624\3516221924.py in <module>
      3
      4 # Open file and read the text
----> 5 with open(fileName, 'r') as file:
      6     data = file.read()
      7

FileNotFoundError: [Errno 2] No such file or directory: 'example2.txt'
```

Python cannot see the files that are located in the subdirectory `Additional Files`. We need to provide Python with additional information about the **path** to these files to be able to access these.

File Paths

A **path** is a string that represents the specific location of a directory or file on your computer. You can think of this as the directions that your computer follows to find a file.

There are two kinds of paths:

Relative paths

A relative path describes the location of a file **relative to the current working directory**. When we use relative paths, we are telling Python to find a certain file, starting from where the script is executed.

In [122…
```
# Read the file 'example.txt', which is in the current working directory
fileName = 'example1.txt'

readFile(fileName)
```

```
Hello! These are the contents of an example file.
```

In [123…
```
# The current working directory can also be represented by a single . followed by the fi
fileName = '.\\example1.txt'

readFile(fileName)
```

```
Hello! These are the contents of an example file.
```

We can specify the subfolder we want Python to look in as a relative path:

In [124…
```
# Look inside the Additional Files subdirectory, starting from the current directory
os.listdir('.\\Additional Files')
```

Out[124]:
```
['Even more files',
 'example2.txt',
 'example3.txt',
 'example4.txt',
 'exampleFive.txt']
```

In [125…
```
# Starting from the current working directory, read the file 'example2.txt' in the subfo
fileName = '.\\Additional Files\\example2.txt'

readFile(fileName)
```

```
This is a file that is located in a subfolder.
```

Absolute paths

An absolute path describes the location of a file relative to the **home directory**. When we use absolute paths, we are telling Python to find a certain file, using a **starting point that does not depend on the location of the script we are running**.

In [126…
```
# Get the home directory
os.path.expanduser('~')
```

Out[126]:
```
'C:\\Users\\Aidan Hunt'
```

In [127…
```
# Remind ourselves what the current directory is
os.getcwd()
```

Out[127]:
```
'C:\\Users\\Aidan Hunt\\MREL Dropbox\\Aidan Hunt\\STEP-UP - ME 498\\Lectures\\Lecture 11
- Working with Files'
```

In [128…
```
# Get the absolute path to `example.txt`
fullPath = os.getcwd() + '\\example1.txt'
print(fullPath)

# Read the file
readFile(fullPath)
```

```
C:\Users\Aidan Hunt\MREL Dropbox\Aidan Hunt\STEP-UP - ME 498\Lectures\Lecture 11 - Worki
ng with Files\example1.txt
Hello! These are the contents of an example file.
```

In [129…
```
# Get the absolute path to `example2.txt`
fullPath = os.getcwd() + '\\Additional Files\\example2.txt'
print(fullPath)

# Read the file
readFile(fullPath)
```

```
C:\Users\Aidan Hunt\MREL Dropbox\Aidan Hunt\STEP-UP - ME 498\Lectures\Lecture 11 - Worki
ng with Files\Additional Files\example2.txt
This is a file that is located in a subfolder.
```

Constructing file paths programmatically

The `os` module has functions for building file paths *programatically*.

- Automatically uses appropriate file separator for your operating system in string concatenation.
- Adds flexiblity: pass names of subfolders and files as parameters.

Goal: Programmatically construct a relative and absolute paths `example2.txt`.

Relative paths

Start with the current working directory, then add on the subfolder and data file:

In [130…
```
# Initialize our path to be the current working directory
fullPath = '.'
print(fullPath)

# Add subfolder
fullPath = os.path.join(fullPath, 'Additional Files')
print(fullPath)

# Add data file to the path
fullPath = os.path.join(fullPath, 'example2.txt')
print(fullPath)
```

```
.
.\Additional Files
.\Additional Files\example2.txt
```

In [131…
```
# Read the file
readFile(fullPath)
```

```
This is a file that is located in a subfolder.
```

Absolute paths

Start with the home path, then add on each subdirectory:

In [132…
```
homePath = os.path.expanduser('~')
subDirs = ['MREL Dropbox', 'Aidan Hunt', 'STEP-UP - ME 498', 'Lectures',
           'Lecture 11 - Working with Files', 'Additional Files']

fullPath = homePath # Initialize our path as the home path
for s in subDirs:
    fullPath = os.path.join(fullPath, s) # Add next subdirectory
    print(fullPath)
```

```
C:\Users\Aidan Hunt\MREL Dropbox
C:\Users\Aidan Hunt\MREL Dropbox\Aidan Hunt
C:\Users\Aidan Hunt\MREL Dropbox\Aidan Hunt\STEP-UP - ME 498
C:\Users\Aidan Hunt\MREL Dropbox\Aidan Hunt\STEP-UP - ME 498\Lectures
C:\Users\Aidan Hunt\MREL Dropbox\Aidan Hunt\STEP-UP - ME 498\Lectures\Lecture 11 - Worki
ng with Files
C:\Users\Aidan Hunt\MREL Dropbox\Aidan Hunt\STEP-UP - ME 498\Lectures\Lecture 11 - Worki
ng with Files\Additional Files
```

In [133…
```
# Read example2.txt
filePath = os.path.join(fullPath, 'example2.txt')
readFile(filePath)

# Read example3.txt
filePath = os.path.join(fullPath, 'example3.txt')
readFile(filePath)
```

```
This is a file that is located in a subfolder.
Another example file in the same subfolder.
```

Comparing path types

So, which is best? (Discuss):

Relative paths

- Pro: Quick and easy to make, especially for small numbers of files
- Con: Path is dependent on processing script location

Absolute paths

- Pro: Path is independent of processing script location
- Con: Harder to construct file paths on the fly (?)

In terms of scalability, absolute paths are the most robust because they depend on the *location of the data*, not the script that is working with the data.

Pattern-matching for filenames

Common scenario: we are working with a lot of files with similar names, stored in a similar location.

- Sequence of data files generated by a sensor/machine
- Images

Use the `glob` module for finding files based on a particular naming convention and path.

- Given a file path pattern, returns a list of the paths to all files matching that pattern.
- '*' character acts as a wildcard (can be any number of characters)

In [146…
```
# The most fun import statement that we will write all quarter
import glob
```

In [147…
```
# Find all files matching matching the template of 'example*.txt' in the current directo
glob.glob('.\\example*.txt')
```

Out[147]:
```
['.\\example1.txt']
```

In [148…
```
# Find all files matching the template of 'example*.txt' in any subdirectories of the cu
glob.glob('.\\**\\example*.txt')
```

Out[148]:
```
['.\\Additional Files\\example2.txt',
 '.\\Additional Files\\example3.txt',
 '.\\Additional Files\\example4.txt',
 '.\\Additional Files\\exampleFive.txt']
```

In [149…
```
# Find all files matching the template of 'example*.txt'
# in any subdirectories of subdirectories of the current directory
glob.glob('.\\**\\**\\example*.txt')
```

Out[149]:
```
['.\\Additional Files\\Even more files\\example6.txt',
 '.\\Additional Files\\Even more files\\example7.txt',
 '.\\Additional Files\\Even more files\\example8.txt']
```

In [150…
```
# Find all files matching the template of 'example*.txt'
# in the current directory AND any level of subdirectory
glob.glob('.\\**\\example*.txt', recursive=True)
```

Out[150]:
```
['.\\example1.txt',
 '.\\Additional Files\\example2.txt',
 '.\\Additional Files\\example3.txt',
 '.\\Additional Files\\example4.txt',
 '.\\Additional Files\\exampleFive.txt',
 '.\\Additional Files\\Even more files\\example6.txt',
```

```
 '.\\Additional Files\\Even more files\\example7.txt',
 '.\\Additional Files\\Even more files\\example8.txt']
```

Next Time

Expand on this with a batch processing routine!