Files and the Operating System

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels as sm
```

To open a file for reading or writing, use the built-in **open** function with either a relative or absolute file path:

In [ ]:
```
path = 'examples/segismundo.txt'
```

In [ ]:
```
f = open(path)
```

By default, the file is opened in read-only mode 'r'. We can then treat the file handle f like a list and iterate over the lines like so:

The lines come out of the file with the end-of-line (EOL) markers intact, so you'll often see code to get an EOL-free list of lines in a file like:

In [ ]:
```
lines = [x.rstrip() for x in open(path)]
```

In [ ]:
```
lines
```

In [ ]:
```
It is important to explicitly close the file when you are finished with it:
```

In [ ]:
```
f.close()
```

One of the ways to make it easier to clean up open files is to use the **with** statement, This will automatically close the file f when exiting the **with** block.:

In [ ]:
```
with open(path) as f:
    lines = [x.rstrip() for x in f]
```

If we had typed f = open(path, 'w'), a new file at examples/segismundo.txt would have been created (be careful!), overwriting any one in its place. There is also the 'x' file mode, which creates a writable file but fails if the file path already exists.

![alt text](images/file modes.png "Python file modes")

read returns a certain number of characters from the file. What constitutes a “character” is determined by the file’s encoding (e.g., UTF-8) or simply raw bytes if the file is opened in binary mode:

```
In [ ]: f = open(path)
```

```
In [ ]: f.read(10)
```

```
In [ ]: f2 = open(path, 'rb') # Binary mode
```

```
In [ ]: f2.read(10)
```

The read method advances the file handle’s position by the number of bytes read. **tell** gives you the current position:

```
In [ ]: f.tell()
```

```
In [ ]: f2.tell()
```

Even though we read 10 characters from the file, the position is 11 because it took that many bytes to decode 10 characters using the default encoding. You can check the default encoding in the **sys** module:

```
In [ ]: import sys
```

```
In [ ]: sys.getdefaultencoding()
```

seek changes the file position to the indicated byte in the file:

```
In [ ]: f.seek(3)
```

```
In [ ]: f.read(1)
```

```
In [ ]: f.close()
```

```
In [ ]: f2.close()
```

To write text to a file, you can use the file’s **write* or **writelines** methods. For example, we could create a version of prof_mod.py with no blank lines like so:

In [ ]:
```
with open('tmp.txt', 'w') as handle:
    handle.writelines(x for x in open(path) if len(x) > 1)
with open('tmp.txt') as f:
    lines = f.readlines()
```

In [ ]:
```
lines
```

![alt text](images/file methods.png "Python file methods")

In [ ]:
```
```