POWERING POTENTIAL

File Handling

Module 13

DATA STORAGE AND FILE HANDLING

Contents

- File objects
- Reading files
- Writing files

New file objects

- File objects are created with the `open` function

```
FileObject = open(filename, mode='r', buffering=-1,  encoding=None, errors=None,
newline=None, closefd=True, opener=None)
```

py3

- Valid open modes :

`'r'`	**open existing file for read (default)**
`'w'`	**open file for write, create or overwrite existing file**
`'a'`	**open file for append, create if does not exist**
`'x'`	**create file and open for write, fails if file exists (3.3)**
`'r+'`	**open existing file for read/write**
`'w+'`	**create & truncate file for read/write**
`'a+'`	**create & append file for read/write**

py3

- File will be closed on exit, or may be closed manually

```
FileObject.close()
```

Reading files into Python

- **Create a file object with `open`**

```
infile = open('filename', 'r')
```

- **Read *n* characters (in text mode)**

→ May return fewer characters near end-of-file

→ If *n* is not specified, the entire file is read

```
buffer = infile.read(42)
```

- **Read a line**

```
line = infile.readline()
```

→ The line terminator "\n" is included

→ Returns an empty string (False) at end-of-file

Reading tricks

Reading the whole file into a variable

- Be careful of the file size

```
lines = open('brian.txt').read()
llines= open('brian.txt').read().splitlines()
linelist = open('brian.txt').readlines()
```

Reading a file sequentially in a loop

- Inefficient

```
for line in open('lines.txt').readlines():
    print(line, end="")
```

- Use the file object iterator

```
for line in open('lines.txt') :
    print(line, end="")
```

A safer way to open files

- **Under very rare circumstances, a file could be left open**

→ An error causing an unhandled exception

- **Some python classes are *context managers***

→ `io` is the most common - file objects are context objects

→ Used with the `with` keyword

- **Ensures files are closed on error**

→ This usually happens anyway with a for loop

→ This is rarely needed - but safer!

```
with open('gash.txt', 'r') as infile:
    for line in infile:
        print(line, end='')
```

`infile` is a file object, is a context object

Writing to files from Python

Open a file handle with `open`

- Specifying write or append

```
output = open('myfile', 'w')
append = open('logfile', 'a')
```

- Write a string
- Append "\n" to make it a line
- Returns the number of chars (text) or bytes (binary) written

```
num = output.write("Hello\n")
```

py3

- Write strings from a list
- Append "\n" to each element to make lines

```
output.writelines(list)
```

Binary mode

By default, open modes are text

- Reading and writing uses native Python strings
- Remember that Python 3 strings are multi-byte (Unicode)

Open a file as binary using 'b' with the mode

- Reading and writing uses bytes objects, not Python strings
- Convert to a Python string using *bytes*.`decode()`

```
for line in open('lines.txt', 'rb'):
    print(line.decode(), end="")
```

- Can also write a bytes object
- Convert from a Python string using *string*`.encode()`

```
fo = open('out.dat','wb')
nb = fo.write(b'Single bytes string')
s = "Native string as a line\r\n"
nb = fo.write(s.encode())
```

py3

SUMMARY

- **A file object is created by calling open**
- **Read from a file:**
- Call read, readline, or readlines methods
- Or invoke the file iterator in a for loop
- **Writing to a file:**
- Call write or writelines methods
- `print` can also be used
- Good practice to close the file as soon as possible