UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Essential Data Skills for Business Analytics

Lecture 8: Files and Modules

Decision, Operations & Information Technologies
Robert H. Smith School of Business
Spring, 2020

File processing

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- A text file can be thought of as a sequence of lines

A text file (sometimes spelled "textfile": an old alternative name is "flatfile") is a kind of computer file that is structured as a sequence of lines of electronic text.

A text file exists within a computer file system. The end of a text file is often denoted by placing one or more special characters, known as an end-of-file marker, after the last line in a text file.

However, on some popular operating systems such as Windows or Linux, text files do not contain any special EOF character.

"Text file" refers to a type of container, while plain text refers to a type of content. Text files can contain plain text, but they are not limited to such.

Open a file

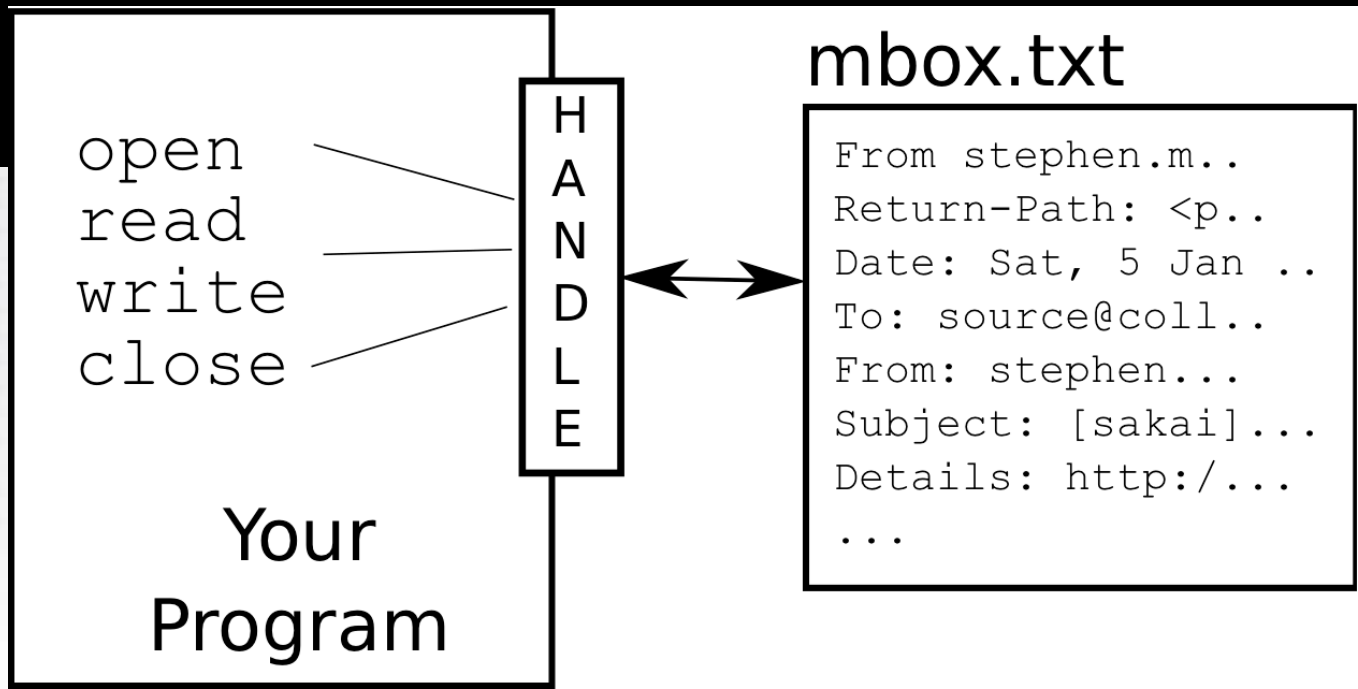
- Before we can read the contents of the file we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the open() function
- open() returns a "file handle" - a variable used to perform operations on the file
- Kind of like "File -> Open" in a Word Processor

Open()

- Syntax
  - file_handler_variable = open(filename, mode)
  - returns a handle use to manipulate the file
  - filename is a string (a string variable or a string constant)
  - mode is optional and should be 'r' if we are planning reading the file and 'w' if we are going to write to the file.

What is a handler?


```
>>> fh = open('mbox.txt','r')
>>> print fh
<open file 'test.txt', mode 'r' at 0x1004a2780>
>>>
```



When files are missing

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> fh = open('test.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'test.txt'
>>>
```

File handler as a sequence

- A file handle open for read can be treated as a sequence of strings where each line in the file is a string in the sequence
- We can use the for statement to iterate through a sequence
- Remember - a sequence is an ordered set

```
xfile = open('mbox.txt')
for line in xfile:
    print line
```

Read the 'whole' file

- We can **read** the whole file (newlines and all) into a single string.

```
>>> fh = open('mbox.txt')
>>> inp = fh.read()
>>> print (len(inp))
94626

>>> print (inp[:20])
A text file (sometim
```

Read file into a list

- We can use **readlines()** to get a list.
- Each element in the list is a line.

```
>>> fh = open('mbox.txt')
>>> lines = fh.readlines()
>>> print (len(lines))
4

>>> print (inp[:2])
['the first line', 'the second line']
```

File write

- The write() method writes any string to an open file.
- The write() method does not add a newline character ('\n') to the end of the string

```
>>> fh = open('test.txt', 'w')
>>> lines = fh.readlines()
>>> fh.write('Python is great\nI like Python')
>>> fh.close()
Python is great
I like Python
```

Other file operations

- Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.
- To use this module you need to import it first and then you can call any related functions.
  - import os
  - os.rename(current_file_name, new_file_name)
  - os.remove(file_name)
  - os.mkdir("newdir")
  - os.listdir(path)
  - …

Modules

- collection of functions and variables
- definitions can be imported
- Many build-in modules: math, random, os, etc.
- Create our own module module_example.py

```
def func1(x):
    ...
def func2(x):
```

Modules

- import module:

```
import module_example
```

- Use modules via "name space":

```
>>> module_example.func1(1000)
>>> module_example.__name__
'module_example'
```

- can give it a local name:

```
>>> fff = module_example.func1
>>> fff(500)
```

Modules

- function definition + executable statements
- executed only when module is imported
- modules have private symbol tables
- avoids name clash for global variables
- accessible as *module.globalname*
- can import into name space:

```
>>> from module_example import func1, func2
>>> func1(500)
```

- can import all names defined by module:

```
>>> from module_example import *
```

Module search path

- current directory
- list of directories specified in PYTHONPATH environment variable
- uses installation-default if not defined, e.g., .:/usr/local/lib/python
- uses sys.path

```
>>> import sys
>>> sys.path
['', 'C:\\PROGRA~1\\Python2.2', 'C:\\Program Files\\Python2.2\\DLLs',
    'C:\\Program Files\\Python2.2\\lib', 'C:\\Program Files\\Python2.2\\lib\\lib-tk',
    'C:\\Program Files\\Python2.2', 'C:\\Program Files\\Python2.2\\lib\\site-
    packages']
```

Module listing

- use dir() for each module

```
>>> dir(module_example)
['__name__', 'func1', 'func2']
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__stderr__', '__st
din__', '__stdout__', '_getframe', 'argv', 'builtin_module_names', 'byteorder',
'copyright', 'displayhook', 'dllhandle', 'exc_info', 'exc_type', 'excepthook', '
exec_prefix', 'executable', 'exit', 'getdefaultencoding', 'getrecursionlimit', '
getrefcount', 'hexversion', 'last_type', 'last_value', 'maxint', 'maxunicode', '
modules', 'path', 'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setpr
ofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout', 'version',
 'version_info', 'warnoptions', 'winver']
```