Files in Python

Learning objects

1. **Understand file types**(text, binary).
2. **Open, read, write, close files** using Python functions.
3. **Use context managers** (`with` statement) for file handling.
4. **Read file contents** using methods like `read()`, `readline()`.
5. **Write and append data** to files.
6. **Handle file-related exceptions** (`FileNotFoundError`).
7**. Work with file paths** .

Text Files in Programming

- Text files contain characters and lines, not always human-readable.
- Can be viewed in a text editor, even if not understandable.

Examples:

- FASTA files (DNA or protein sequences)
- Command-line program outputs (e.g., BLAST)
- FASTQ files (DNA sequencing reads)
- HTML files
- Word processing documents
- Python code

File Processing

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

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

http://www.py4e.com/code/mbox-short.txt
https://www.py4e.com/code3/mbox.txt

FYI: Getting these files from the terminal:

```
wget https://www.py4e.com/code3/mbox.txt
```

Opening 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

Using `open()`

```
fhand = open('mbox.txt', 'r')
```

- `handle = open(filename, mode)`
- returns a handle use to manipulate the file
- filename is a string
- mode is optional and should be 'r' if we are planning to read the file and 'w' if we are going to write to the file

https://www.py4e.com/code3/mbox.txt

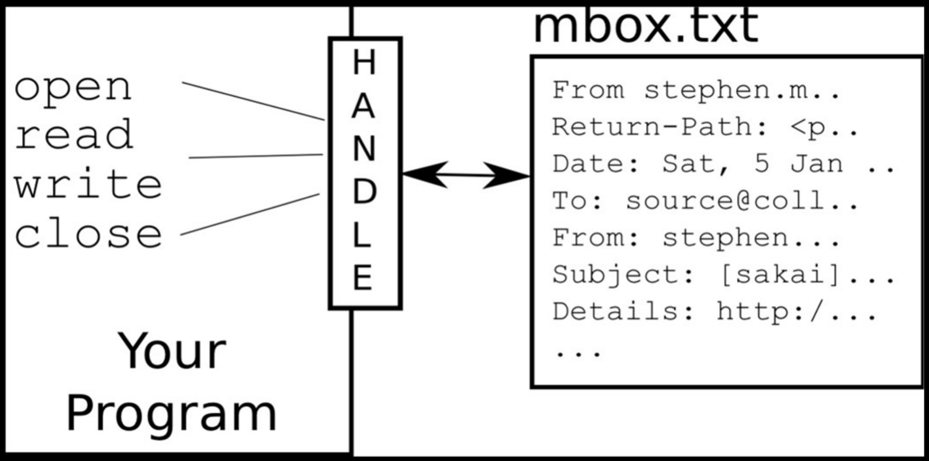
Using `open()`

```
fhand = open('mbox.txt', 'r')
```

```
'r'  Open for reading (default)

'w'  Open for writing, truncating (overwriting) the file first

'a' Open for append

'r+', for both reading and writing

'rb' or 'wb' Open in binary mode (read/write using byte data)
```

What is a Handle?

```
>>> fhand = open('mbox.txt', 'r')
>>> print(fhand)
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='UTF-8'>
```



When Files are Missing

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

File Processing

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

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/

Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

File Processing

But remember, a text file has newlines at the end of each line

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008\n
Return-Path: <postmaster@collab.sakaiproject.org>\n
Date: Sat, 5 Jan 2008 09:12:18 -0500\n
To: source@collab.sakaiproject.org\n
From: stephen.marquard@uct.ac.za\n
Subject: [sakai] svn commit: r39772 - content/branches/\n
\n
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772\n
```

File Handle 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, the file handle, can be thought of as an **iterator**
- **When opening a file this way, remember to `close()`**

```
fhand = open('mbox.txt', 'r')
for line in fhand:
    print(line)

fhand.close()
```

Closing a File

- It's your responsibility to close the file
- In most cases, upon termination of an application or script, a file will be closed eventually
- However, there is no guarantee when exactly that will happen
- This can lead to unwanted behavior including resource leaks
- It's also a best practice within Python to make sure that your code behaves in a way that is well defined and reduces any unwanted behavior
- When you're manipulating a file, there are two ways that you can use to ensure that a file is closed properly, even when encountering an error. The **first** way to close a file using the method `close()`, and do this in `try-finally()`

https://realpython.com/read-write-files-python/

CL

Counting Lines in a File

- Open a file read-only
- Use a for loop to read each line
- Count the lines and print out the number of lines

```
fhand = open('mbox.txt', 'r')
count = 0
for line in fhand:
    count = count + 1
print('Line Count:', count)

fhand.close()
```

```
$ python open.py
Line Count: 132045
```

Reading the *Whole* File

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

```
>>> fhand = open('mbox-short.txt', 'r')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

See `<file>.readlines()` - returns a list of the remaining lines in the file.

Reading Only the First N Lines

- `<file>.read()` - returns the entire remaining contents of the file as a single (possibly large, multi-line) string

- `<file>.readline()` - returns the next line of the file. This is all text up to *and including* the next newline character

- `<file>.readlines()` - returns a list of the remaining lines in the file. Each list item is a single line including the newline characters. ***Need to be careful with this if the file is very large***

Dealing with newlines

Searching Through a File

We can put an if statement in our for loop to only print lines that meet some criteria

```
fhand = open('mbox-short.txt', 'r')
for line in fhand:
    if line.startswith('From:'):
        print(line)

fhand.close()
```

OOPS!

What are all these blank lines doing here?

```
From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu
...
```

OOPS!

What are all these blank lines doing here?

- Each line from the file has a newline at the end
- The print statement adds a newline to each line
- We could update the print function `end = ''`
- But is that what we should do?

```
From: stephen.marquard@uct.ac.za\n
\n
From: louis@media.berkeley.edu\n
\n
From: zqian@umich.edu\n
\n
From: rjlowe@iupui.edu\n
\n
...
```

Searching Through a File (fixed)

- We can strip the whitespace from the right-hand side of the string using `rstrip()` from the string library
- The newline is considered "white space" and is stripped

```
fhand = open('mbox-short.txt', 'r')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:'):
        print(line)

fhand.close()
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
....
```

Using in to Select Lines

We could easily remove the `not` and the `continue`

```
fhand = open('mbox-short.txt', 'r')
for line in fhand:
    line = line.rstrip()
    if '@uct.ac.za' in line:
        print(line)

fhand.close()
```

Easier to understand imo

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f...
```

Your turn

"Create a text file named 'dna.txt' containing the sequence:

`ACTGTACGTGCACTGATC`

After entering the sequence, press Enter and save the file."

```python
# open the file
my_file = open("dna.txt")
# read the contents
my_dna = my_file.read()
# calculate the length
dna_length = len(my_dna)
# print the output
print("sequence is " + my_dna + " and length is " + str(dna_length))
```

What is a Solution?

`Desired solution` `:`sequence is ACTGTACGTGCACTGATC and length is 18

Remember: Closing a File

- It’s your responsibility to close the file
- In most cases, upon termination of an application or script, a file will be closed eventually
- However, there is no guarantee when exactly that will happen
- This can lead to unwanted behavior including resource leaks
- It’s also a best practice within Python to make sure that your code behaves in a way that is well defined and reduces any unwanted behavior
- When you’re manipulating a file, there are two ways that you can use to ensure that a file is closed properly, even when encountering an error. The second way to ensure a close on a file is to use the `with` statement

https://realpython.com/read-write-files-python/

CL

Writing a File

- Opening a file for writing prepares the file to receive data
- If you open an existing file for writing, you wipe out the file’s contents. If the named file does not exist, a new one is created

```
out_handle = open("mydata.out", "w")
print(<expressions>, file=out_handle)
out_handle.close()
```

Your Turn

Open a file and write a single line of text to it:

```
my_file = open("out.txt", "w")

# write "abcdef"
my_file.write("abc" + "def")
# write "8"
my_file.write(str(len('AGTGCTAG')))
# write "TTGC"
my_file.write("ATGC".replace('A', 'T'))
# write "atgc"
my_file.write("ATGC".lower())
# write contents of my_variable

myfile.close()
```

Missing Files

What happens if we try to read a file that doesn't exist?

```
my_file = open("nonexistent.txt")
```

We get a new type of error that we've not seen before:

```
IOError: [Errno 2] No such file or directory: 'nonexistent.txt'
```

Paths and folders

on Linux:

```
my_file = open("/home/martin/myfolder/myfile.txt")
```

on Windows:

```
my_file = open(r"c:\windows\Desktop\myfolder\myfile.txt")
```

on a Mac:

```
my_file = open("/Users/martin/Desktop/myfolder/myfile.txt")
```

The `with` Statement

- Often, it is hard to remember to `close` the file once we are done with the file
- Python offers an easy solution for this.
- We can use `with` statement in Python so that we don't have to close the file handler
- The `with` statement creates a ***context manager*** and it will automatically close the file handler for you when you are done with it
- Here is an example using `with` statement

```
with open('depth_of_coverage.txt,'r') as infh:
    for line in infh:
        print(line)
```

https://docs.python.org/3/library/stdtypes.html#typecontextmanager

The `with` Statement

- We can also use the `with` statement to open more than one file
- Here is an example of using with statement in Python **to open one file for reading and another file for writing**

```
with open(in_filename) as infh, open(out_filename, 'w') as outfh:
   for line in infh:
      ...
      ...
      print(parsed_line,  file=outfh)
```

- The `with` statement simplifies exception handling (next lecture) by encapsulating common preparation and cleanup tasks
- In addition, it will automatically close the file(s). The `with` statement provides a way for ensuring that a clean-up is always used

Always us the `with` Statement?

- When you have a single source Python program (1 file), It's always best to use the `with` statement for opening files
- When your projects grow, and you have modules that all might need to open a file, or write to a file an multiple different places - then handling exceptions each time can be redundant when using the `with` statements in those modules
- This also break the DRY principle (Don't repeat yourself)
- An IO (input/output) package can be implemented to handle all opening of file
- Where the function sends back a `filehandle`, and then the user calls `filehandle.close()`

Examples:

Let's look at the following codes:

```
read_from_file.py

readMbox.py

Read_fasta_file.py

Write_to_file.py
```