Python III: File operation

Chaochen Wang

ZJU-UoE Institute

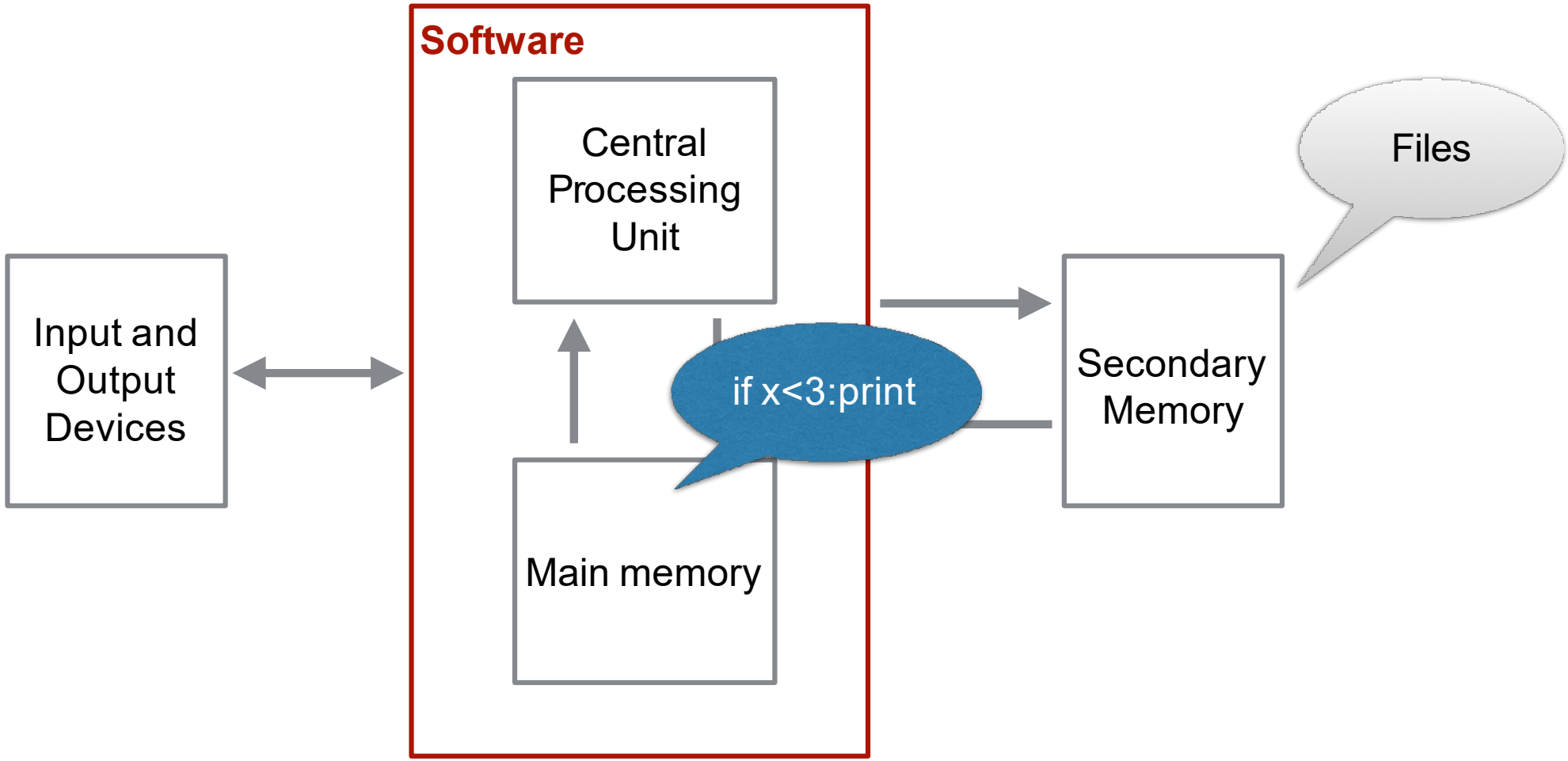
wangchaochen@intl.zju.edu.cn

Mar 30, 2020

Learning Objects

- Understand stdin and stdout
- Read and write files

Input and Output

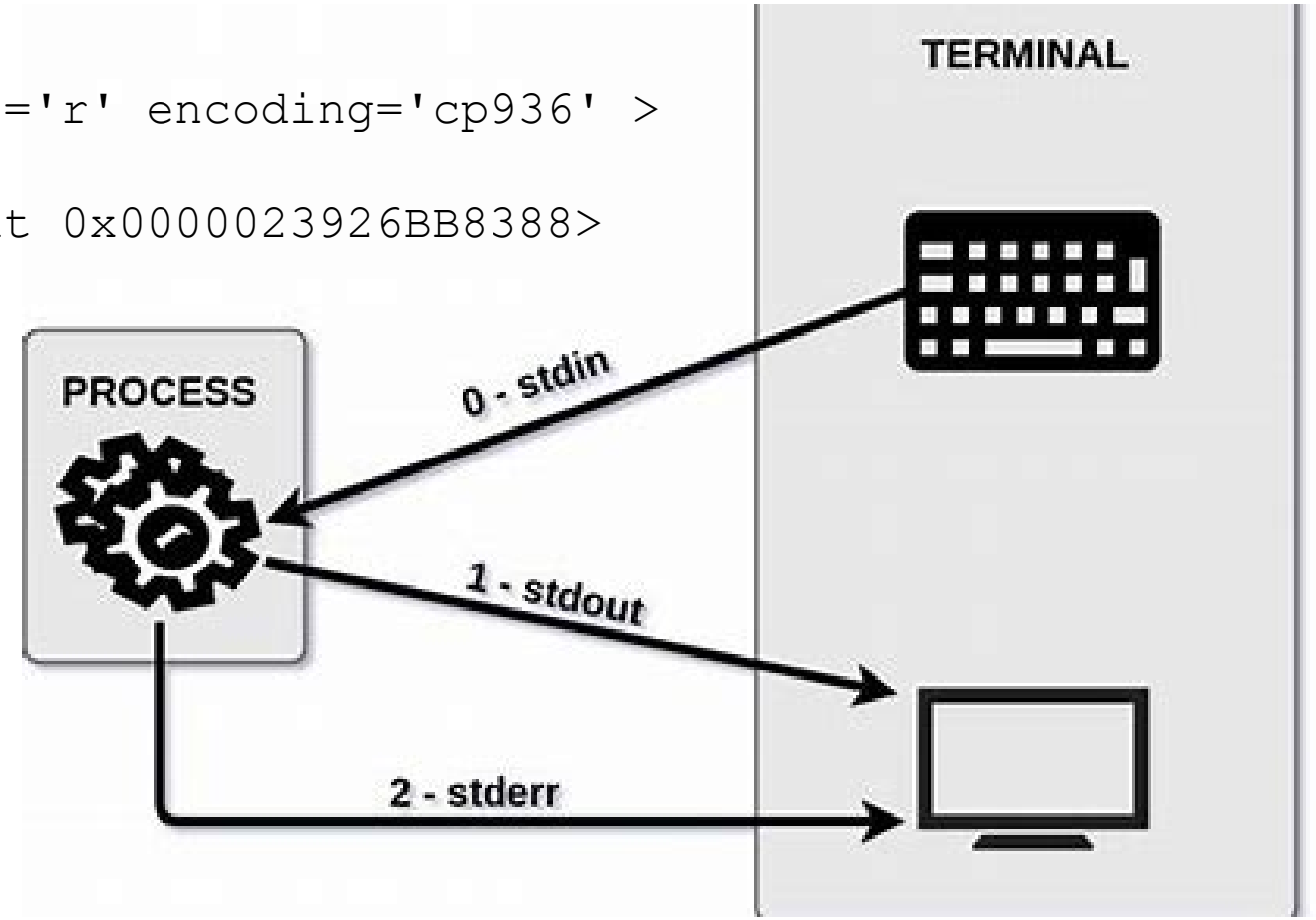


stdin and stdout

stdin is a **stream that represents input into a program**

stdout is **where all your output goes**.

```
>>>import sys
>>>print(sys.stdin)
<_io.TextIOWrapper name='<stdin>' mode='r' encoding='cp936' >
>>>Print(sys.stdout)
<ipykernel.iostream.OutStream object at 0x0000023926BB8388>
```



Read from stdin and output to stdout

input() waits and reads input from stdin, and it reads everything as string
print() prints output to stdout

```
>>>a=input()
I like python class!
>>>print(a)
I like python class!

>>>b=input()
123
>>>type(b)
str
```

This is what you input from your keyboard

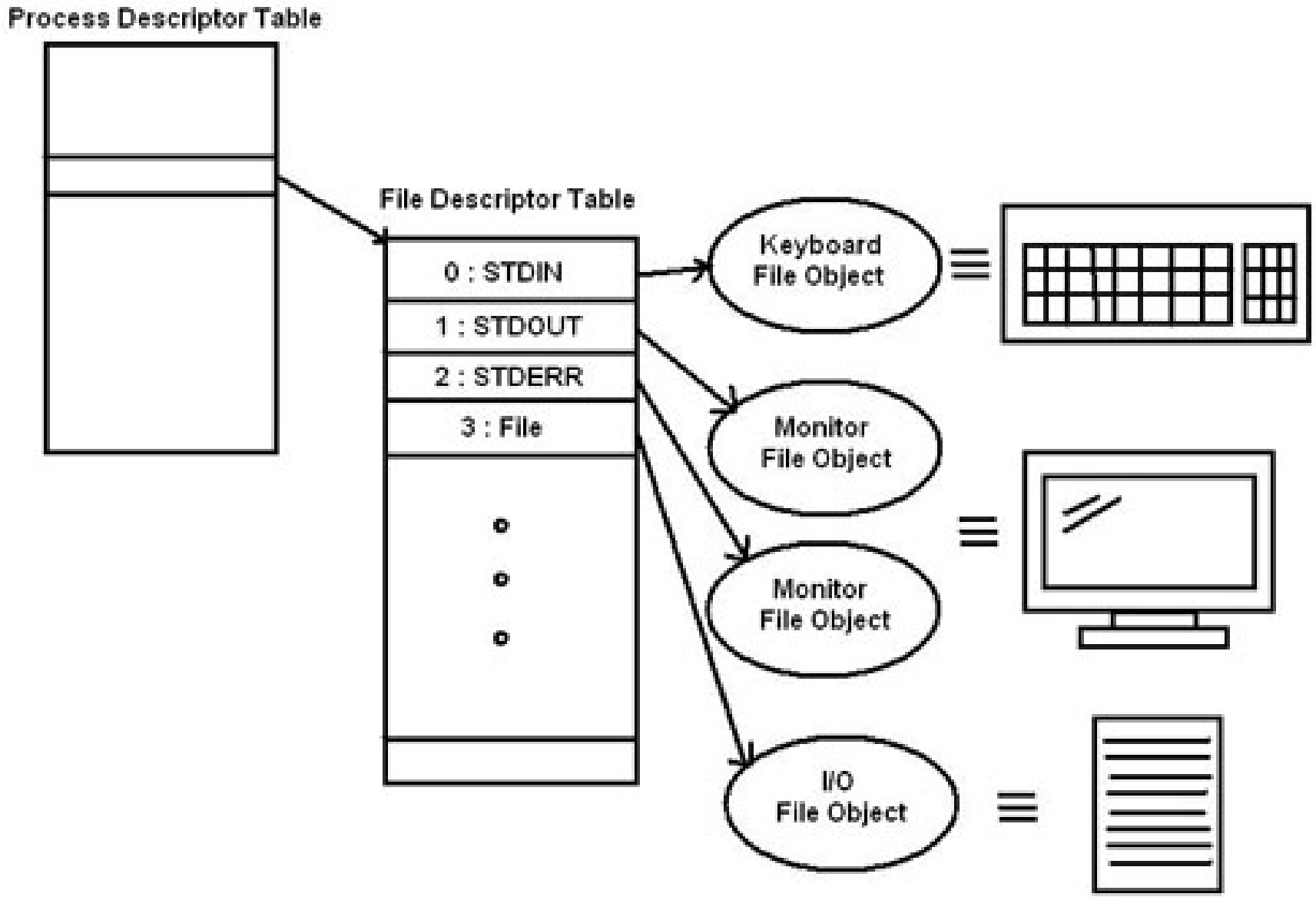
String Format

Formatted string literals (also called f-strings for short) let you include the value of Python expressions inside a string by prefixing the string with f or F and writing expressions as {expression}.
An optional format specifier can follow the expression. This allows greater control over how the value is formatted.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
        print(f'{name:10} ==> {phone:10d}')

Sjoerd     ==>       4127
Jack       ==>       4098
Dcab       ==>       7678
```

Files


Process Descriptor Table
File Descriptor Table
0 : STDIN
1 : STDOUT
2 : STDERR
3 : File
Keyboard
File Object
Monitor
File Object
Monitor
File Object
I/O
File Object

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

Using open()

handle = open(filename, mode)

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

Notice:

- a handle used to manipulate the file
- filename is a string
- mode is optional, shall be 'r' if we are planning to read the file and 'w' if we are going to write to the file

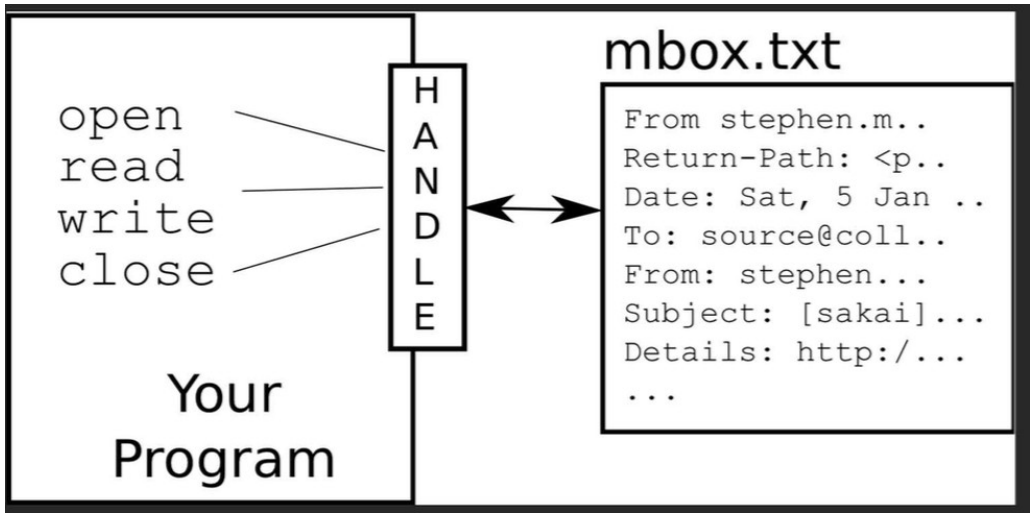
Mode

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open a disk file for updating (reading and writing)

Notice:The default mode is 'r' (open for reading text, synonym of 'rt').

What is a handle?

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



The New Line Character

We use a special character called the “**newline**” to indicate when a line ends

Represented as **\n** in strings

Newline is still one character - not two

```
>>>stuff = 'Hello\nWorld! '
>>>print (stuff)
Hello
World!
>>>stuff = 'X\nY'
>>>print (stuff)
X
Y
>>>print(len(stuff))
3
```

File Progressing

A text file has **newlines** at the end of each line

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\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 - a **sequence** is an ordered set

```
>>>xfile = open('mbox.txt')
>>>for line in xfile:
        print(line)              #returns all lines in 'mbox.txt'
```

You can also use while loop and f.readline() to do the job. Explore by yourself.

Counting Lines

Open a **file** read-only

Use a **for** loop to read each line

Count the lines and print out the number of lines

```
>>>xfile = open('mbox.txt')  count = 0
>>>for line in xfile:
      count = count + 1
>>>print('Line Count:',count)
Line Count: 1909
```

Reading the Whole File

- We can read the whole file (newlines and all) into a single **string.**
- **f.read**(size), which reads some quantity of data and returns it as a string
- *size* is an optional numeric argument. When *size* is omitted or negative, the entire contents of the file will be read and returned

```
>>>xfile = open('mbox.txt')
>>>inp = xfile.read()
>>>print(len(inp))
94625
>>>print(inp[:20])
From stephen.marquar
>>>inp20 = xfile.read(20)
>>>Print(inp20)
From stephen.marquar
```

Notice: f.read() will read the entire file into the memory. If your file is extremely large, then it will cause problem.
Use iteration is more memory efficient.

Searching

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

```
>>>xfile = open('mbox.txt')
>>>for line in xfile:
        if line.startswith('From:'):
                print(line)
```

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

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu

From: cwen@iupui.edu

From: cwen@iupui.edu

From: gsilver@umich.edu

From: gsilver@umich.edu

From: zqian@umich.edu
```

The print statement adds a newline to each line!

Searching(fixed)

We can strip the whitespace from the right-hand side of the string using **rstrip()**

The newline is considered “white space” and is **stripped**

```
>>>xfile = open('mbox.txt')
>>>count = 0
>>>for line in xfile:
        line = line.rstrip()
                if line.startswith('From:'):
                print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: gsilver@umich.edu
From: gsilver@umich.edu
From: zqian@umich.edu
From: gsilver@umich.edu
From: wagnermr@iupui.edu
From: zqian@umich.edu
From: antranig@caret.cam.ac.uk
From: gopal.ramasammycook@gmail.com
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
```

Return a copy of the string with trailing characters removed

Searching(fixed)

We could also have avoided printing that by writing

print **line[:-1]**

```
>>>xfile = open('mbox.txt')
>>>count = 0
>>>for line in xfile:
        if line.startswith('From:'):
                print(line[:-1])
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: gsilver@umich.edu
From: gsilver@umich.edu
From: zqian@umich.edu
From: gsilver@umich.edu
From: wagnermr@iupui.edu
From: zqian@umich.edu
From: antranig@caret.cam.ac.uk
From: gopal.ramasammycook@gmail.com
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: stephen.marquard@uct.ac.za
```

Skipping

We can conveniently skip a line by using the **continue** statement

```
>>>xfile = open('mbox.txt')
>>>for line in xfile:
        line = line.rstrip()
              if not line.startswith('From:'):
                    continue
                    print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: gsilver@umich.edu
From: gsilver@umich.edu
From: zqian@umich.edu
From: gsilver@umich.edu
From: wagnermr@iupui.edu
From: zqian@umich.edu
From: antranig@caret.cam.ac.uk
From: gopal.ramasammycook@gmail.com
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: stephen.marquard@uct.ac.za
```

Selecting Lines

We can look for a string anywhere **in** a **line** as our selection criteria

```
>>>xfile = open('mbox.txt')
>>>for line in xfile:
        line = line.rstrip()
                if not '@uct.ac.za' in line:
                continue
                print(line)
```

select lines with '@uct.ac.za'

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: nakamura.uits.iupui.edu: apache 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: nakamura.uits.iupui.edu: apache set sender to
david.horwitz@uct.ac.za using -f
From: david.horwitz@uct.ac.za
Author: david.horwitz@uct.ac.za
r39753 | david.horwitz@uct.ac.za | 2008-01-04 13:05:51 +0200 (Fri, 04
Jan 2008) | 1 line
From david.horwitz@uct.ac.za Fri Jan  4 06:08:27 2008
```

Prompt for File Name

```
>>>fname = input('Enter the file name: ')
Enter the file name: mbox.txt
>>>xfile = open(fname)
>>>count = 0
>>>for line in xfile:
        line = line.rstrip()
              if line.startswith('Subject:'):
                    count = count +1
>>>print('There were',count,'subject lines in',fname)
There were 27 subject lines in mbox.txt
```

You input the file name here

Writing Files

To **write** a file, you have to open it with mode "**w**" as a second parameter:

```
>>>fout = open('myfile.txt','w')
>>>print(fout)
    <_io.TextIOWrapper name='myfile.txt' mode='w' encoding='UTF-8'>
```

Notice : If the file already **exists**, opening it in write mode **clears out** the old data and starts fresh, so be careful! If the file doesn't exist, a new one is created.

If we don't want to do that we can open the file for **appending** (instead of writing) by using the argument '**a**'.

```
fout = open('myfile.txt','a')
```

Writing Files

The write method of the file handle object puts data into the file, returning the **number** of characters written.

The default write mode is text for writng (and reading) **strings**.

```
>>>fout = open('myfile.txt','w')
>>>line1 = 'Lecture 8.2\n'
>>>line2 = 'File Operation\n’
>>>fout.write(line1)
>>>fout.write(line2)
>>>fout.close()
>>>xfile = open('myfile.txt')
>>>for line in xfile:
      print(line[:-1])
Lecture 8.2
File Operation
```

Closing File

When you are done reading/writing, you have to **close** the file to make sure that the last bit of data is physically written to the disk so it will not be lost if the power goes off.

```
try:
        xfile = open('mbox.txt', 'r')
        print(xfile.read())
finally:
        if xfile:
                xfile.close()
```

Alternatives:

```
with open('/mbox.txt', 'r') as xfile:
    print(xfile.read())
```

PANDAS

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language

```
import pandas as pd
```

More information:

http://pandas.pydata.org/pandas-docs/stable/reference/io.html

Reading CSV Files

Read a comma-separated values (**csv**) file into DataFrame.

data.txt

```
a,b,c,d,name
1,2,3,4,python
5,6,7,8,java
9,10,11,12,c++
```

```
>>>import pandas as pd
>>>data = pd.read_csv("data.txt")
>>>print(data)
>>>type(data)
```

```
   a   b   c   d     name
1  1   2   3   4   python
2  5   6   7   8     java
2  9  10  11  12      c++
pandas.core.frame.DataFrame
```

Reading Excel Files

Similarly, to read an excel file, use `pandas.read_excel()`

```
>>>import pandas as pd
>>>fpath = '' #Your file path
>>>fname='PCA.xlsx'
>>>file=fpath+'/'+fname
>>>df=pd.read_excel(file)
>>>print(df)
>>>PCA1=df['PCA1'].values
>>>PCA2=df['PCA2'].values
>>>X=df[['PCA1','PCA2']].values
>>>y=df['Group'].values
```

	Sample	Group	PCA1	PCA2
0	N.1	3	97.033298	19.992441
1	N.2	3	69.778300	41.603534
2	N.3	3	78.046376	-39.509308
3	N.4	3	65.435442	9.192919
4	N.5	3	69.859828	-58.749766
5	pre.1	1	-32.996653	97.173964
6	pre.2	1	-58.970954	-27.013138
7	pre.3	1	-53.551257	-11.509871
8	pre.4	1	-48.435232	-9.056897
9	pre.5	1	-52.075736	-4.787465
10	prog.1	2	-14.136827	48.289060
11	prog.3	2	-50.677588	-21.996517
12	prog.4	2	-26.645156	-15.049542
13	prog.5	2	-42.663840	-28.579412

Summary

Operation	Interpretation
output = open(r'C:\spam', 'w')	Create output file ('w' means write)
input = open('data', 'r')	Create input file ('r' means read)
input = open('data')	Same as prior line ('r' is the default)
aString = input.read()	Read en>re file into a single string
aString = input.read(N)	Read up to next N characters (or bytes) into a string
aString = input.readline()	Read next line (including \n newline) into a string
aList = input.readlines()	Read en>re file into list of line strings (with \n)
output.write(aString)	Write a string of characters (or bytes) into file
output.writelines(aList)	Write all line strings in a list into file
output.close()	Manual close (done for you when file is collected)
output.flush()	Flush output buffer to disk without closing
anyFile.seek(N)	Change file posi>on to offset N for next opera>on
for line in open('data'):use line	File iterators read line by line
open('f.txt', encoding='latin-1')	Python 3.0 Unicode text files (str strings)
open('f.bin', 'rb')	Python 3.0 binary bytes files (bytes strings)

Most materials were prepared by Prof. Xin Chen in ZJU