Python Object-Oriented Program with Libraries

Unit 2: I/O, File System and Exceptions

CHAPTER 2: FILE SYSTEM AND PATH

DR. ERIC CHOU

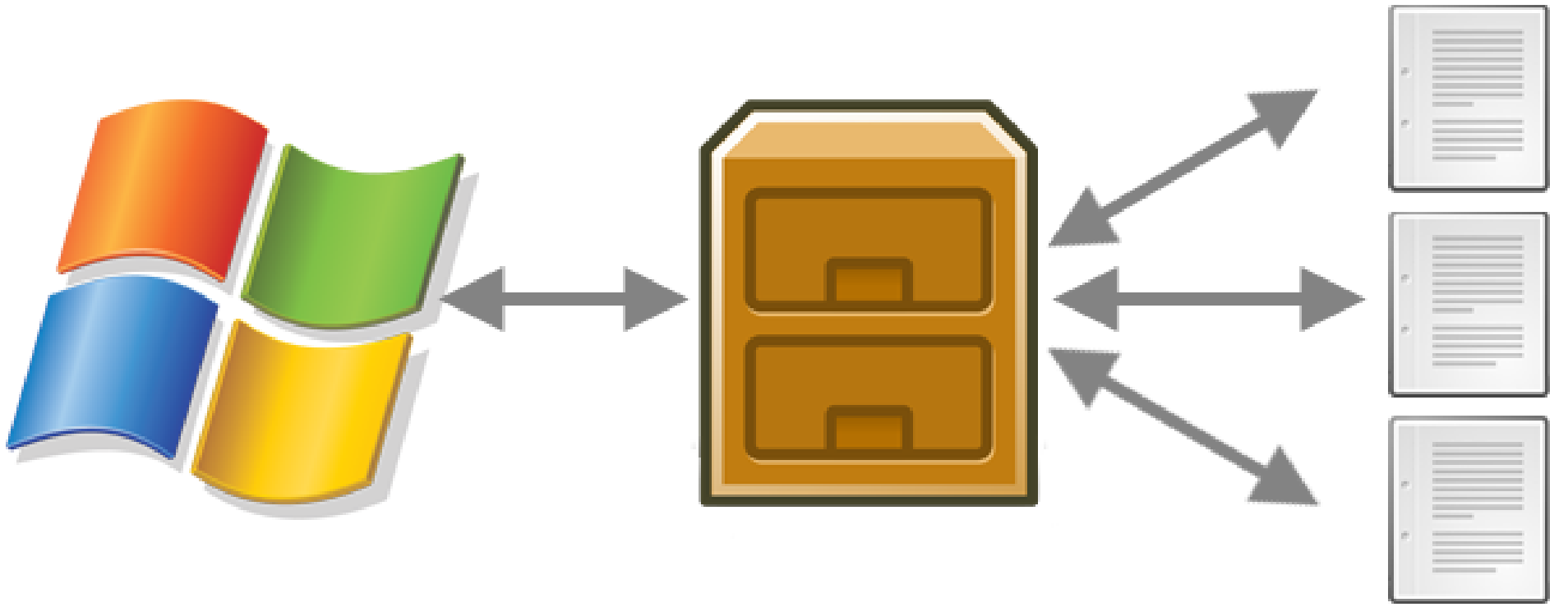
IEEE SENIOR MEMBER

Objectives

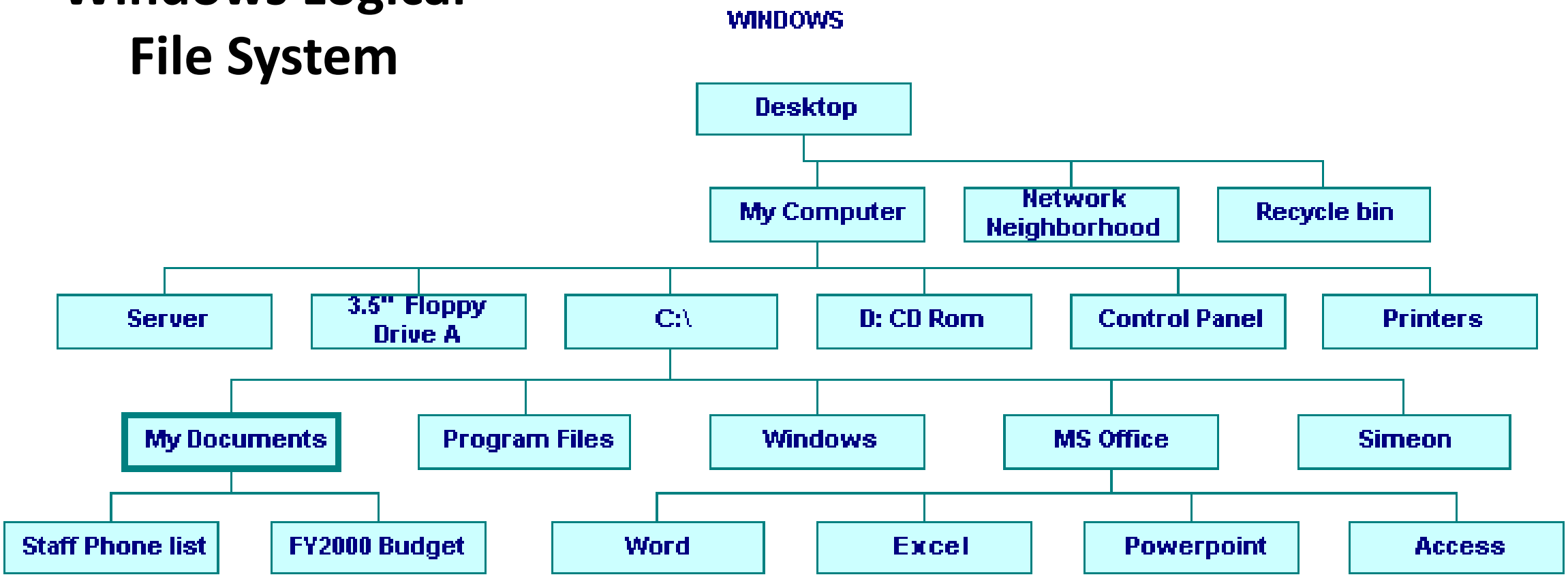
- Study about File System
- File access rights
- File reading
- File processing
- File operations
- Path and pathlib
- os and sys modules

File Systems

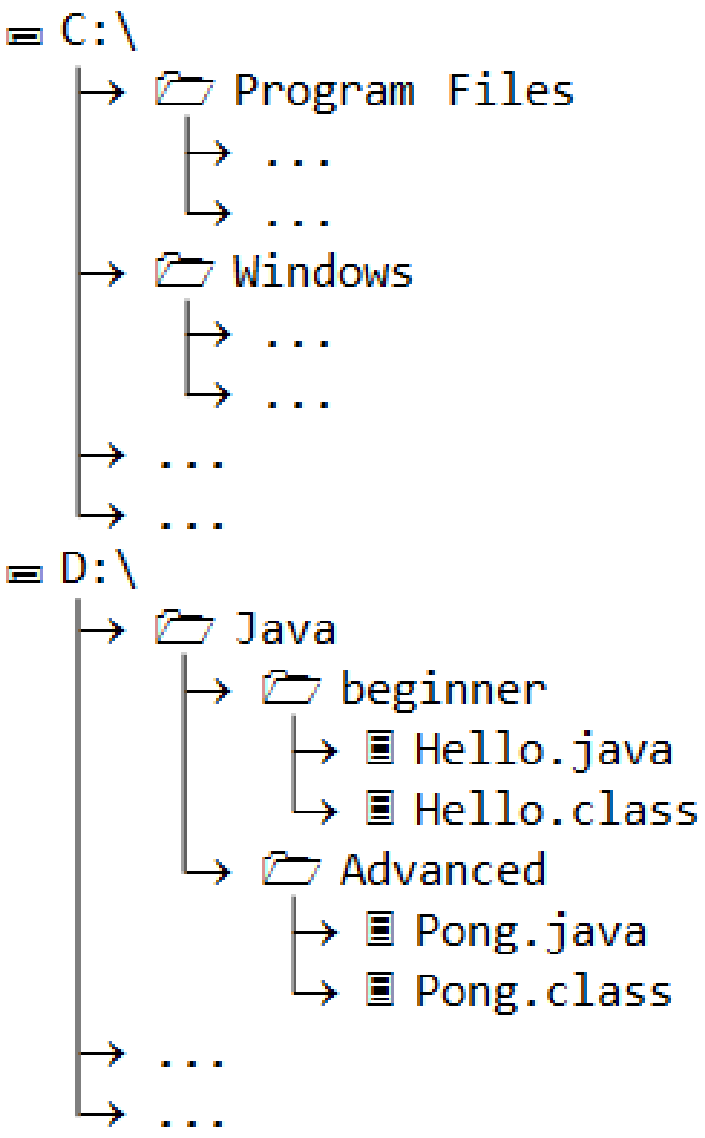
LECTURE 1


OPERATING SYSTEM
FILE SYSTEM
FILES

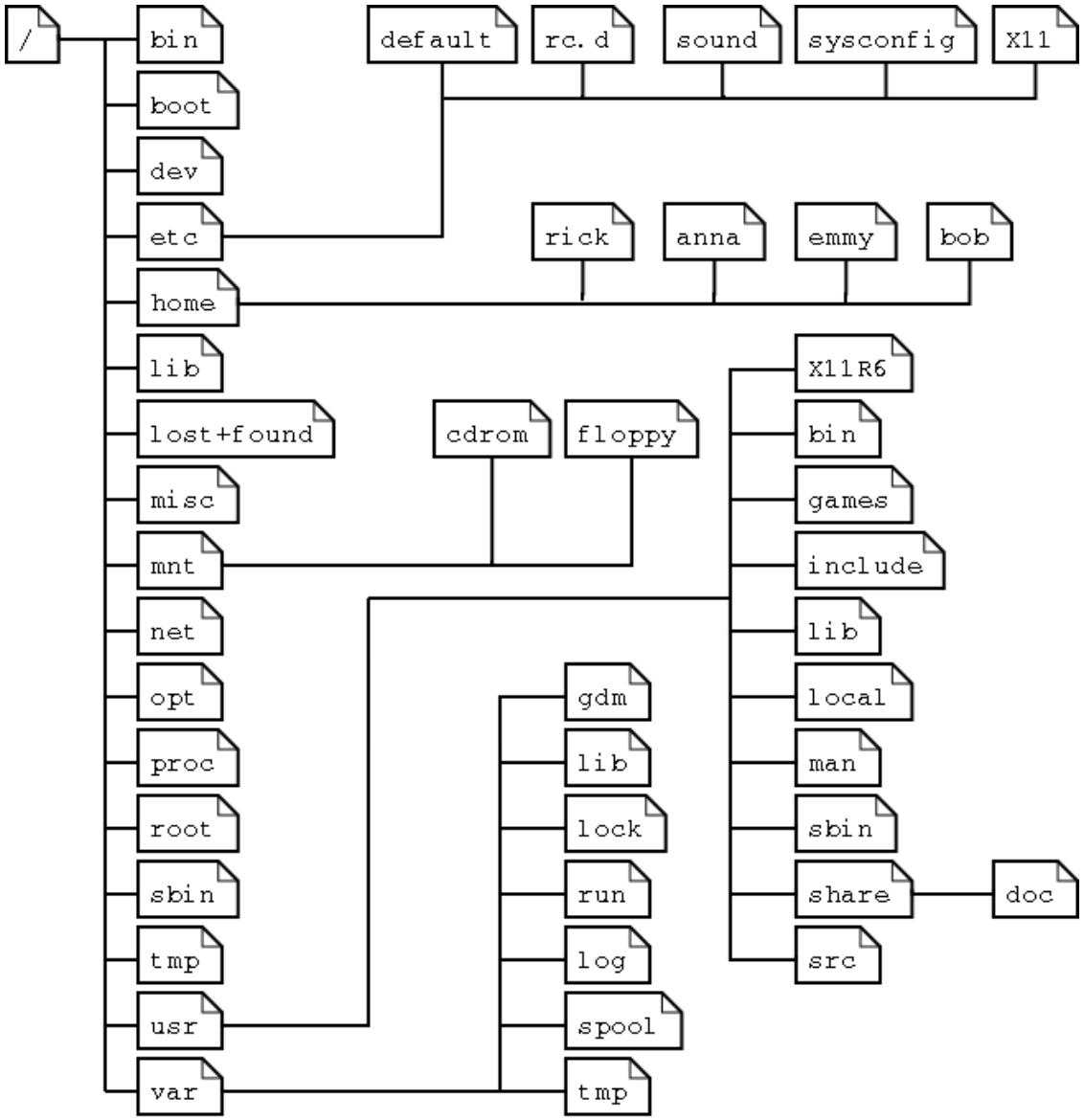
Windows Logical File System



Windows Physical File System



LINUX File System

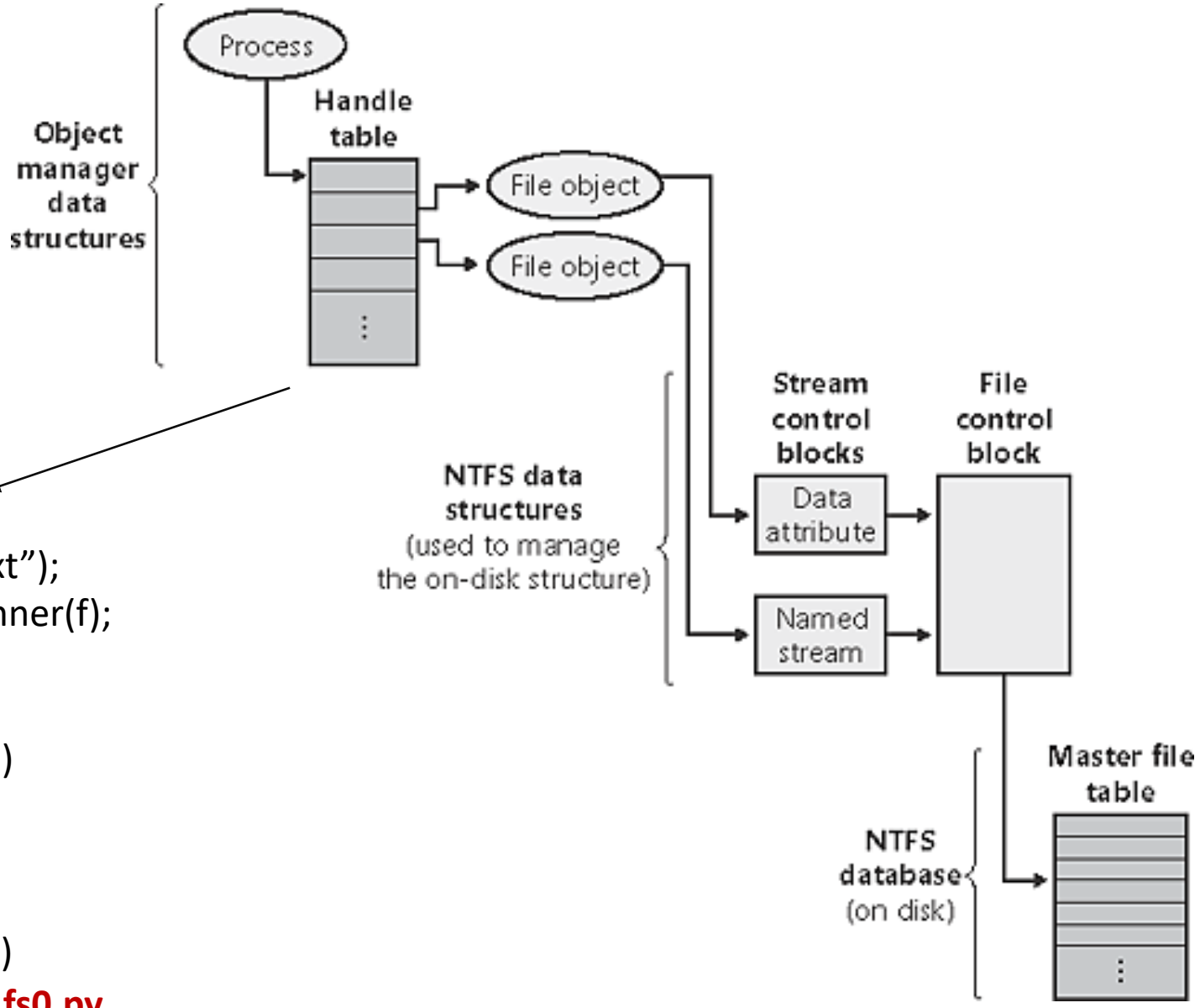


How NTFS Works

NTFS (New Technology File System)

- A **file system** is a required part of the operating system that determines how files are named, stored, and organized on a volume. A file system manages **files** and **folders**, and the information needed to locate and access these items by local and remote users.
- Microsoft Windows Server 2003 supports the **NTFS** file system on basic and dynamic disks. (Replaced the FAT file system) Basic disks and volumes are the storage types most often used with Windows operating systems. Dynamic disks offer greater flexibility for volume management because they use a database to track information about dynamic volumes on the disk and about other dynamic disks in the computer.
- During the format of a volume you can choose the type of file system for the volume. When you choose the NTFS file system, the formatting process places the key NTFS file data structures on the volume, regardless of whether it is a basic or dynamic volume.

Note: f is a file handler which has a pointer pointing to the handle table.



Java:

```
File f = new File("aa.txt");
Scanner in = new Scanner(f);
```

Python:

```
f = open("aa.txt", "r")
line = f.readline()
while line:
    print(line)
    line = f.readline()
f.close()
```

fs0.py

Demo Program: fs0.py

File handler

Read Until '\n' met ('\n' included)

File Access Mode

```
f = open("aa.txt", "r")
line = f.readline()
while line:
    print(line, end="")
    line = f.readline()
f.close()
```

NULL (0) when eof met

File closed

```
Run fs0
C:\Python\Python36\python.exe
alpha
beta
gamma
delta
epsilon
```

```
File f = new File("aa.txt", "r");
Scanner in = new Scanner(f);
while (in.hasNext()){
    line = in.nextLine();
    System.out.print(line);
}
f.close()
```


Microsoft
ReFS
Robust File System
Replacement for NTFS in Microsoft Windows Server

File Access

LECTURE 2

File in Python

- File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).
- Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.
- When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.
- Hence, in **Python**, a file operation takes place in the following order.
  1. Open a file
  2. Read or write (perform operation)
  3. Close the file

Python File **Open** Operation

```
f = open("test.txt")
f = open("test.txt", “r”)
f = open("test.txt", mode = 'r’, encoding = 'utf-8')
```

file name

file access mode

file text encoding

Python File Access Mode	
Modes	**Description**
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

Modes	Description
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

File Reading

LECTURE 3

Data File to be Read in

alpha\n

beta\n

gamma\n

delta\n

epsilon\n

null (0, or EOF)

Read File Line by Line: readline()

Demo Program: fs0.py

```
f = open("aa.txt", "r")
line = f.readline()
while line:
    print(line, end="")
    line = f.readline()
f.close()
```

```
Run fs0
C:\Python\Python36\python.exe
alpha
beta
gamma
delta
epsilon
```

Read File as a whole: readlines()

Demo Program: fs1.py

```
f = open("aa.txt", "r")
lines = f.readlines()     # lines is a list of line strings
print("Print file in list format: ")
print(lines)

all_lines = ""
for line in lines:
    all_lines += line

print("Print file as a long string: ")
print(all_lines)
f.close()
```

Output:

```
Print file in list format:
['alpha\n', 'beta\n', 'gamma\n', 'delta\n', 'epsilon\n', '\n', '\n']
Print file as a long string:
alpha
beta
gamma
delta
epsilon
```

Read File Token by Token: readlines()

Demo Program: fs2.py

```
f = open("aa.txt", "r")
lines = f.readlines()
for line in lines:
    line = line.strip()
    print(line, end=" ")
f.close()
```

Run fs2

```
C:\Python\Python36\python.exe
alpha beta gamma delta epsilon
```

Equivalent to .trim() in Java
Take out all of the white space characters (\n, \t, \f, space)

Read File Token by Token: read().split()

Demo Program: fs3.py

```
f = open("aa.txt", "r")
tokens=f.read().split()
for token in tokens:
    print("Token", token)
f.close()
```

Used to identify it is a token.

```
Run fs3
C:\Python\Python36\python.exe
Token alpha
Token beta
Token gamma
Token delta
Token epsilon
```

Note:

read(): read the whole file into a string.

read().split(): read the whole file into a string. Then split the string into a list of tokens (string)

readlines(): read the whole file into a list of lines.

Read File Character by Character: read(1)

Demo Program: fs4.py

```
f = open("aa.txt", "r")
ch=f.read(1)
while ch:
    print(ch, end=" ")
    ch=f.read(1)
f.close()
```

```
Run fs4
C:\Python\Python36\python.exe
a l p h a
 b e t a
 g a m m a
 d e l t a
 e p s i l o n
```

Use space as separator so that we know the characters are read in one by one.

File Access Pattern

File Access Pattern Algorithm:

```
Open_File
Read data from file to a buffer
while checking the buffer is valid:
      working on the buffer
      Read data from file to a buffer
```

Terminology:

File f: file handler

Buffer: ch, token(string), line, lines(list)

Read Functions: read(1), read(), readline(), readlines()

File Access Code in Python and Java

Python Pseudo Code:

```
f = open("filename.txt", "r")
buffer=f.read_function()
while buffer:
    processing(buffer)
    buffer=f.read_function()

# null return from the
# read_function is used to
# check the end_of_file
# condition
```

Java Pseudo Code:

```
File f = new File("filename.txt" );
Scanner in = new Scanner();
while (in.hasNext()){
    buffer=in.nextReading();
    processing(buffer);
}
```

File Processing

LECTURE 4

Python File Methods	
close()	readlines(size)
flush()	seek(offset)
fileno()	tell()
isatty()	truncate(size)
next()	write(string)
read(size)	writelines(list)
readline(size)	

Number of Characters in a File

Demo Program: character.py

```
f = open("usdeclar.txt", "r")
ch=f.read(1)
count = 0
while ch:
    if ch.islower() or ch.isupper():
        count += 1
    print(ch, end="")
    ch=f.read(1)
```

Run character

```
that they are absolved from all allegiance to the British Crown, and that
all political connection between them and the state of Great Britain, is and
ought to be totally dissolved; and that as free and independent states, they
have full power to levey war, conclude peace, contract alliances, establish
commerce, and to do all other acts and things which independent states may
of right do. And for the support of this declaration, with a firm reliance
on the protection of Divine Providence, we mutually pledge to each other our
lives, our fortunes and our sacred honor.

Total letter count for  usdeclar.txt  is  6631
```

Reading Numbers from File

Demo Program: numbers.py

```
f = open("num.txt", "r")
tokens=f.read().split()
for token in tokens:
    token = token.strip()
    print(token, end=": ")
    try:
        if len(token)!=0:
            num = int(token)
            print(num)
    except:
        print("Error Input Format!!!")
f.close()
```

num.txt:

```
39 49 59 50 93
34
94
59 28 29
78
```

Output:

```
Run numbers
C:\Python\
39: 39
49: 49
59: 59
50: 50
93: 93
34: 34
94: 94
59: 59
28: 28
29: 29
78: 78
```

Note:

strip() takes out whitespace characters

len(token) == 0, takes out empty strings

try-except structure: takes out non-int data

Reading Words from File and Count Them

Demo Program: words.py

```python
f = open("usdeclar.txt", "r")
tokens=f.read().split()
count = 0
for token in tokens:
    token = token.strip()
    try:
        if len(token)!=0:
            count += 1
            if count % 20 != 0: print(token, end=" ")
            else: print(token)
    except:
        print("Error Input Format!!!")
print("usdeclar.txt has ", count, " words.")
f.close()
```

Output:

```
Run words
of a free people. Nor have we been wanti
time to time of attempts by their legisl
the circumstances of our emigration and
conjured them by the ties of our common
correspondence. They too have been deaf
necessity, which denounces our separatio
friends. We, therefore, the representati
of the world for the rectitude of our in
people of these colonies, solemnly publi
and independent states; that they are ab
them and the state of Great Britain, is
states, they have full power to levey wa
and things which independent states may
on the protection of Divine Providence,
usdeclar.txt has  1340  words.
```

Transcopy: Copy from one file to the other and adding line numbers

Demo Program: Transcopy.py

```
f = open("usdeclar.txt", "r")
g = open("undeclarcopy.txt", "w")
lines = f.readlines()     # lines is a list of line strings
count = 0
for line in lines:
    count += 1
    g.write("Line "+str(count)+": "+line)

f.close()
g.close()
```

Output

```
Line 1:                          Declaration of Independence
Line 2:
Line 3:                       [Adopted in Congress 4 July 1776]
Line 4:
Line 5:
Line 6:
Line 7:     The Unanimous Declaration of the Thirteen United States of America
Line 8:
Line 9: When, in the course of human events, it becomes necessary for one people to
Line 10: dissolve the political bands which have connected them with another, and to
Line 11: assume among the powers of the earth, the separate and equal station to
Line 12: which the laws of nature and of nature's God entitle them, a decent respect
Line 13: to the opinions of mankind requires that they should declare the causes
Line 14: which impel them to the separation.
Line 15:
Line 16: We hold these truths to be self-evident, that all men are created equal,
Line 17: that they are endowed by their Creator with certain unalienable rights, that
Line 18: among these are life, liberty and the pursuit of happiness. That to secure
Line 19: these rights, governments are instituted among men, deriving their just
Line 20: powers from the consent of the governed. That whenever any form of
```

Discussion

- The same program structure for numbers.py can be used to read in data of different format using

  float(token)

- And, other data processing method

auto-mpg.txt (From auto-mpg.data from UCI)

Data Set for Auto Miles per Gallon

Attribute Information:

1. mpg: continuous
2. cylinders: multi-valued discrete
3. displacement: continuous
4. horsepower: continuous
5. weight: continuous
6. acceleration: continuous
7. model year: multi-valued discrete
8. origin: multi-valued discrete
9. car name: string (unique for each instance)

auto-mpg.txt

18.0	8	307.0	130.0	3504.	12.0	70	1	"chevrolet chevelle malibu"
15.0	8	350.0	165.0	3693.	11.5	70	1	"buick skylark 320"
18.0	8	318.0	150.0	3436.	11.0	70	1	"plymouth satellite"
16.0	8	304.0	150.0	3433.	12.0	70	1	"amc rebel sst"
17.0	8	302.0	140.0	3449.	10.5	70	1	"ford torino"
15.0	8	429.0	198.0	4341.	10.0	70	1	"ford galaxie 500"
14.0	8	454.0	220.0	4354.	9.0	70	1	"chevrolet impala"
14.0	8	440.0	215.0	4312.	8.5	70	1	"plymouth fury iii"
14.0	8	455.0	225.0	4425.	10.0	70	1	"pontiac catalina"
15.0	8	390.0	190.0	3850.	8.5	70	1	"amc ambassador dpl"
15.0	8	383.0	170.0	3563.	10.0	70	1	"dodge challenger se"
14.0	8	340.0	160.0	3609.	8.0	70	1	"plymouth 'cuda 340"
15.0	8	400.0	150.0	3761.	9.5	70	1	"chevrolet monte carlo"
14.0	8	455.0	225.0	3086.	10.0	70	1	"buick estate wagon (sw)"
24.0	4	113.0	95.00	2372.	15.0	70	3	"toyota corona mark ii"
22.0	6	198.0	95.00	2833.	15.5	70	1	"plymouth duster"
18.0	6	199.0	97.00	2774.	15.5	70	1	"amc hornet"
21.0	6	200.0	85.00	2587.	16.0	70	1	"ford maverick"
27.0	4	97.00	88.00	2130.	14.5	70	3	"datsun pl510"

Reading in List of Lists with Heterogeneous Data

Demo Program: auto.py

- Read the miles per gallon information for each car model into a list.
- Then, put all of the model lists into a higher level cars list.

```python
f = open("auto-mpg.txt", "r")
lines = f.readlines()     # lines is a list of line strings
cars = []
for line in lines:
    fields = line.split()
    for i in range(9, len(fields)):    # merging the whole string in "" together
        fields[8] += " " + fields[i]
    model = []
    ct = 0
    for data in fields:                # strip of whitespaces for each data filed
        fields[ct] = data.strip()
        ct += 1
    try:
        ct = 0
        model.append(float(fields[ct]))
        ct += 1
        model.append(int(fields[ct]))
        ct += 1
        model.append(float(fields[ct]))
        ct += 1
        model.append(float(fields[ct]))
        ct += 1
        model.append(float(fields[ct]))
        ct += 1
        model.append(float(fields[ct]))
        ct += 1
        model.append(int(fields[ct]))
        ct += 1
        model.append(int(fields[ct]))
        ct += 1
        fields[ct] = fields[ct][1:len(fields[ct])-1] # takes out the "  " marks
        model.append(fields[ct])
    except:
        pass
    finally:
        print(model)
        cars.append(model)
#print(cars)
f.close()
```

auto.py

Partial Results for Mixed Data Field auto-mpg data to be read into lists.
They will be further grouped into cars list.

Run auto

```
[34.0, 4, 108.0, 70.0, 2245.0, 16.9, 82, 3, 'toyota corolla']
[38.0, 4, 91.0, 67.0, 1965.0, 15.0, 82, 3, 'honda civic']
[32.0, 4, 91.0, 67.0, 1965.0, 15.7, 82, 3, 'honda civic (auto)']
[38.0, 4, 91.0, 67.0, 1995.0, 16.2, 82, 3, 'datsun 310 gx']
[25.0, 6, 181.0, 110.0, 2945.0, 16.4, 82, 1, 'buick century limited']
[38.0, 6, 262.0, 85.0, 3015.0, 17.0, 82, 1, 'oldsmobile cutlass ciera (diesel)']
[26.0, 4, 156.0, 92.0, 2585.0, 14.5, 82, 1, 'chrysler lebaron medallion']
[22.0, 6, 232.0, 112.0, 2835.0, 14.7, 82, 1, 'ford granada l']
[32.0, 4, 144.0, 96.0, 2665.0, 13.9, 82, 3, 'toyota celica gt']
[36.0, 4, 135.0, 84.0, 2370.0, 13.0, 82, 1, 'dodge charger 2.2']
[27.0, 4, 151.0, 90.0, 2950.0, 17.3, 82, 1, 'chevrolet camaro']
[27.0, 4, 140.0, 86.0, 2790.0, 15.6, 82, 1, 'ford mustang gl']
[44.0, 4, 97.0, 52.0, 2130.0, 24.6, 82, 2, 'vw pickup']
[32.0, 4, 135.0, 84.0, 2295.0, 11.6, 82, 1, 'dodge rampage']
[28.0, 4, 120.0, 79.0, 2625.0, 18.6, 82, 1, 'ford ranger']
[31.0, 4, 119.0, 82.0, 2720.0, 19.4, 82, 1, 'chevy s-10']
```

File Operations

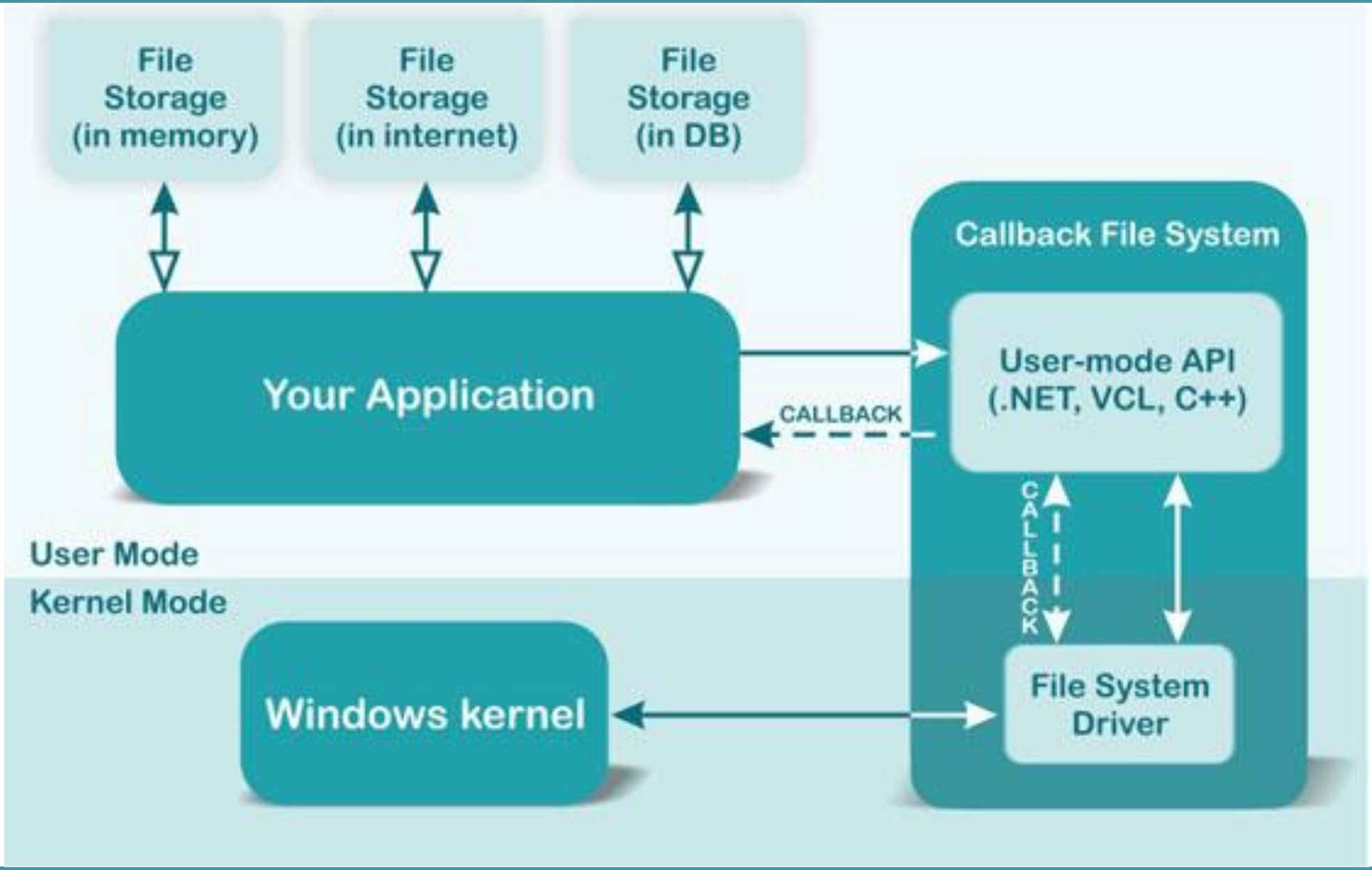
LECTURE 5

Method	Description
close()	Close an open file. It has no effect if the file is already closed.
detach()	Separate the underlying binary buffer from the TextIOBase and return it.
fileno()	Return an integer number (file descriptor) of the file.
flush()	Flush the write buffer of the file stream.
isatty()	Return True if the file stream is interactive.
read(n)	Read atmost n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.

Method	Description
seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resize the file stream to size bytes. If size is not specified, resize to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Write string s to the file and return the number of characters written.
writelines(lines)	Write a list of lines to the file.

Path

LECTURE 6


File Storage (in memory)
File Storage (in internet)
File Storage (in DB)
Callback File System
Your Application
User-mode API (.NET, VCL, C++)
CALLBACK
CALLBACK
User Mode
Kernel Mode
Windows kernel
File System Driver

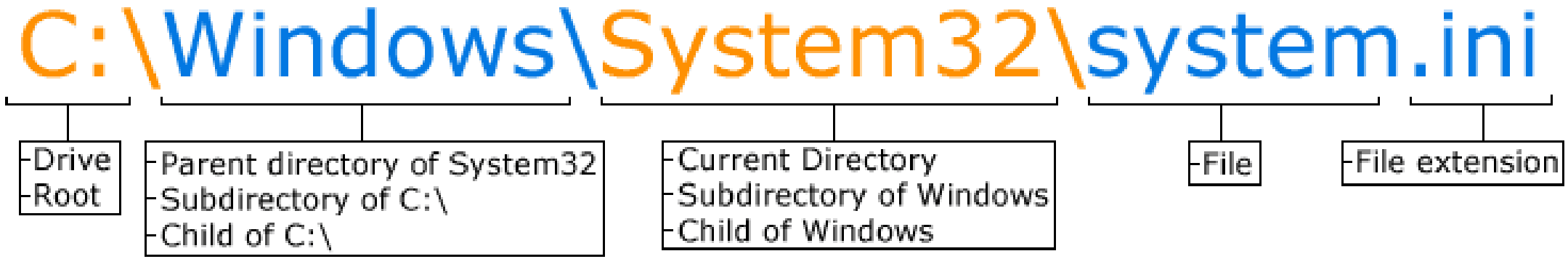
Path

A path may refer to any of the following:

1. Alternatively referred to as the pathname, the current path or path is the complete location or name of where a computer file, web page, or other object is located. Below are some basic examples of different paths you will encounter while working on a computer.

Windows MS-DOS Path

- The following example shows an MS-DOS path or file path for system.ini file.
- When working with an MS-DOS, Windows, or Windows command line path, the drives, directories, and files are all separated by a backslash.



ComputerHope.com

Linux and Apple path

In Linux, or an Apple shell using the **pwd** command, your path may look like the following example. When working with this type of path, the drives, directories, and files are all separated by a forward slash.

```
/home/hope/public_html/
```

Network and Internet Path

Network path

A **network path** is the path to a share. In the example below, "help" is the share on the "hope" computer.

```
\\hope\help
```

Internet path

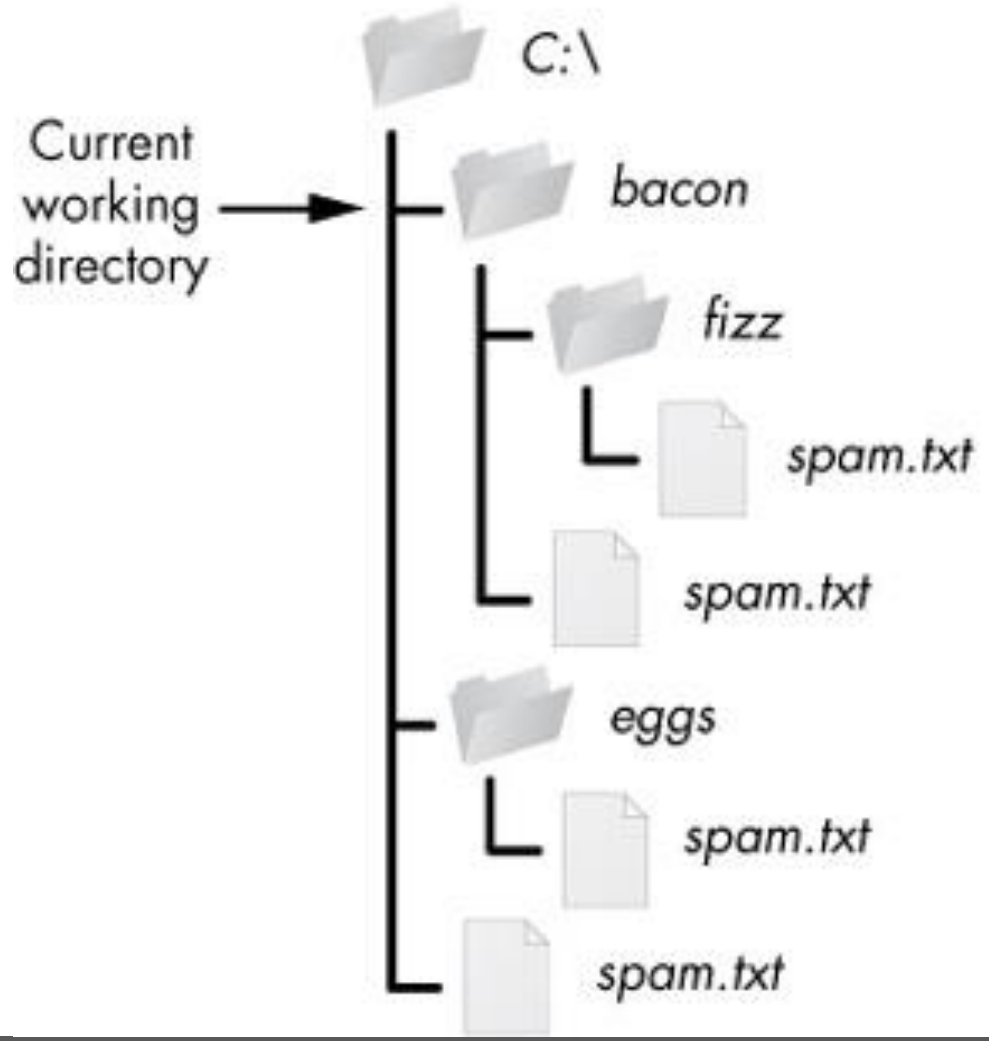
The following example shows the path (URL) to this web page on the Internet.

```
https://www.computerhope.com/jargon/p/path.htm
```

Relative Path

. Current Directory (single dot)

.. Parent Directory (double dots)

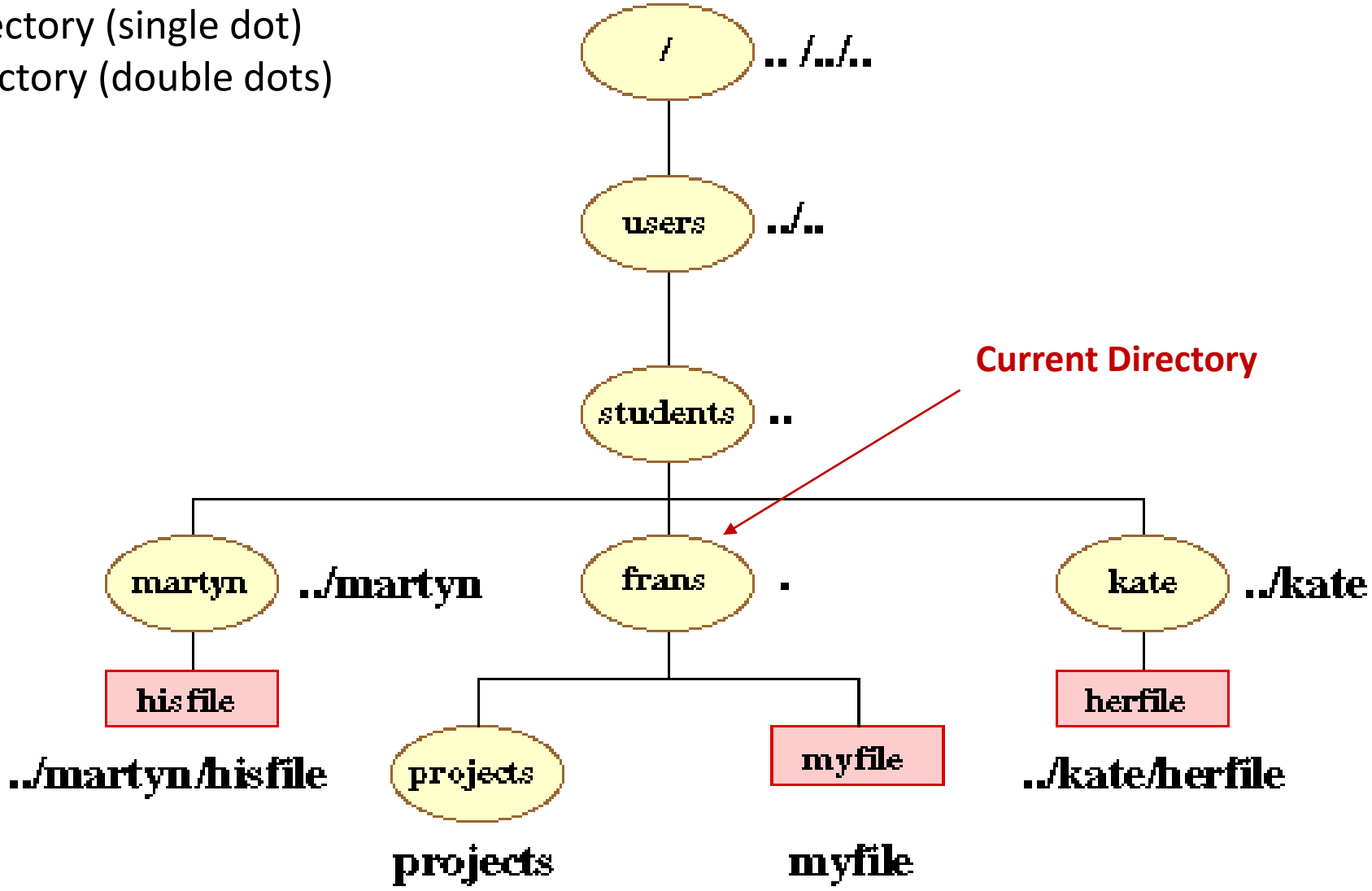


Relative Paths	Absolute Paths
..\	C:\
.\	C:\bacon
.\fizz	C:\bacon\fizz
.\fizz\spam.txt	C:\bacon\fizz\spam.txt
.\spam.txt	C:\bacon\spam.txt
..\eggs	C:\eggs
..\eggs\spam.txt	C:\eggs\spam.txt
..\spam.txt	C:\spam.txt

Relative Path

. Current Directory (single dot)

.. Parent Directory (double dots)



Path Data Representation

- Typical data representation is string.
- May contains space
- Directory levels separated by forwards slash (/, linux or Mac) or backward slash (\, windows)

Print Local File:
Demo Program: print_local.py

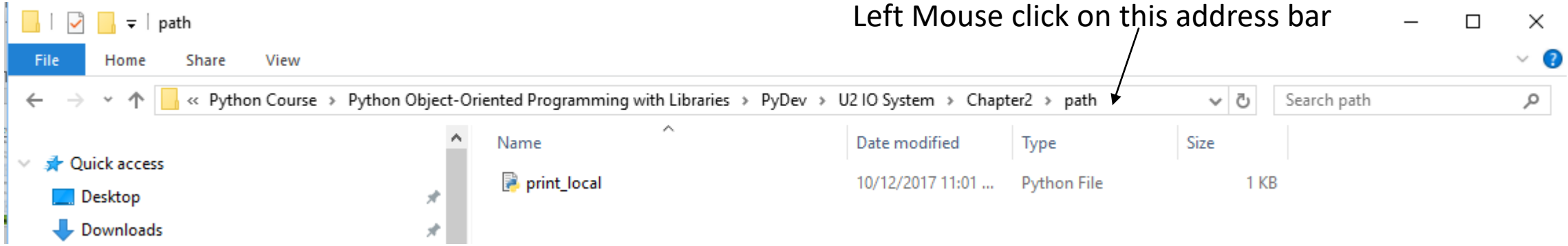
```
f = open("print_local.py", "r")
file_string = f.read()
print("Print file: ")
print(file_string)
f.close()
```

Run print_local

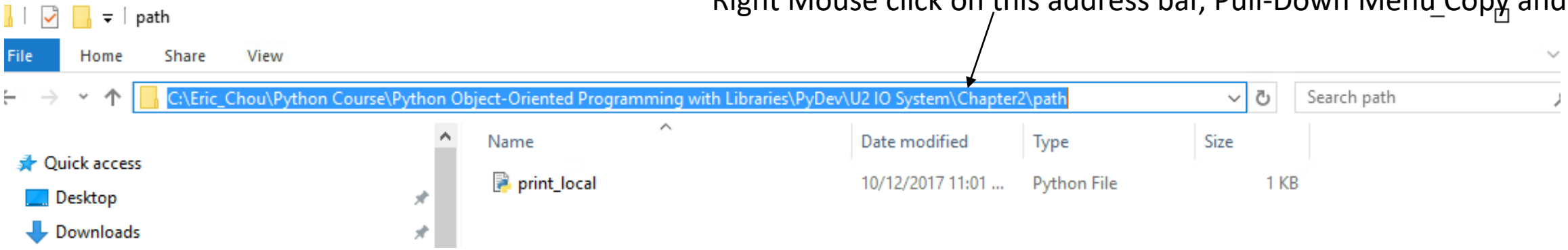
```
C:\Python\Python36\python.exe "
Print file:
f = open("print_local.py", "r")
file_string = f.read()
print("Print file: ")
print(file_string)
f.close()
```

Finding Path in Windows

Left Mouse click on this address bar



Right Mouse click on this address bar, Pull-Down Menu Copy and Paste



Print Local File:
Demo Program: print_local2.py

Copy from the window for this directory path and duplicate the \ symbol to \\ (escape symbol)

```
path = "C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System\\Chapter2\\path"
file_name = "print_local2.py"
filename = path+"\\"+file_name

f = open(filename, "r")
file_string = f.read()              # lines is a list of line strings
print("Print file: ")
print(file_string)
f.close()
```

Run print_local2

```
C:\Python\Python36\python.exe "C:/Eric_Chou/Python Course/Python Object-Oriented Programming with Libraries/PyDev/U2 IO System/Chapter2/path/print_local2.py"
Print file:
path = "C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System\\Chapter2\\path"
file_name = "print_local2.py"
filename = path+"\\"+file_name

f = open(filename, "r")
file_string = f.read()              # lines is a list of line strings
print("Print file: ")
print(file_string)
f.close()
```

Print Local File:
Demo Program: print_local3.py

- Print the same local file with absolute path, local file path, and relative path.

Go PyCharm!!!

```python
path = "C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System\\Chapter2\\path"
file_name = "print_local2.py"
filename = path+"\\"+file_name

print("Absolute Path:", filename)
f = open(filename, "r")
file_string = f.read()
print("Print file: ")
print(file_string)
f.close()

names = filename.split("\\")
fname = names[len(names)-1]    # local file name
print("Local Path:", fname)
f = open(fname, "r")
file_string = f.read()
print("Print file: ")
print(file_string)
f.close()

fname2 = ".\\" + fname
print("Relative Path: ", fname2)
f = open(fname, "r")
file_string = f.read()
print("Print file: ")
print(file_string)
f.close()
```

print_local3.py

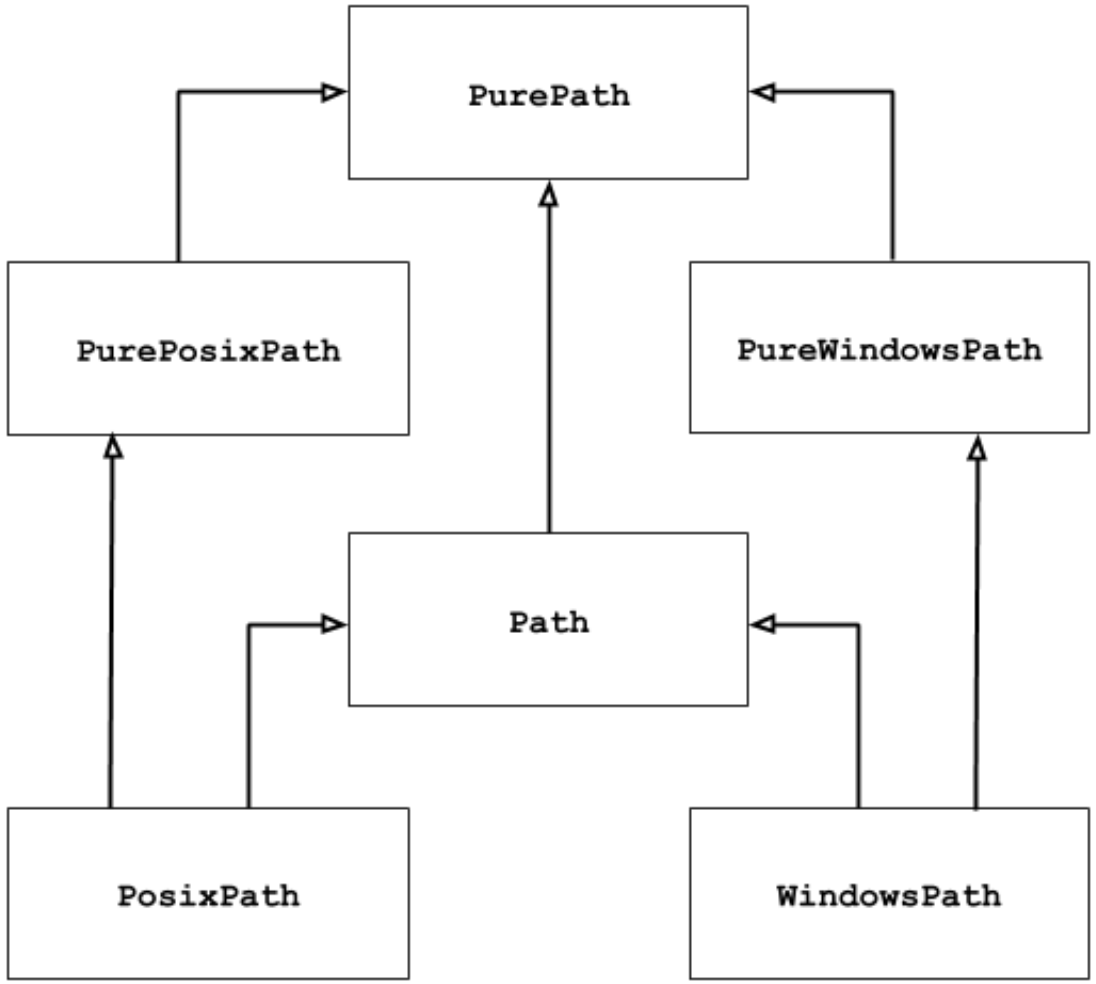
Run print_local3

```
C:\Python\Python36\python.exe "C:/Eric_Chou/Python Course/Python Object-Oriented Programming with Libraries/PyDev/U2 IO System/Chapter2/path/print_local3.py"
Absolute Path: C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter2\path\print_local2.py
Print file:
path = "C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System\\Chapter2\\path"
file_name = "print_local2.py"
filename = path+"\\"+file_name

f = open(filename, "r")
file_string = f.read()              # lines is a list of line strings
print("Print file: ")
print(file_string)
f.close()


Local Path: print_local2.py
Print file:
path = "C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System\\Chapter2\\path"
file_name = "print_local2.py"
filename = path+"\\"+file_name

f = open(filename, "r")
file_string = f.read()              # lines is a list of line strings
print("Print file: ")
print(file_string)
f.close()


Relative Path:  .\print_local2.py
Print file:
path = "C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System\\Chapter2\\path"
file_name = "print_local2.py"
filename = path+"\\"+file_name

f = open(filename, "r")
file_string = f.read()              # lines is a list of line strings
print("Print file: ")
print(file_string)
f.close()
```

Output with Highlights

pathlib

LECTURE 7



Inheritance Hierarchy of Path Classes

pathlib – Object-Oriented Filesystem Paths

pathlib

- This module offers classes representing filesystem paths with semantics appropriate for different operating systems. Path classes are divided between **pure** paths, which provide purely computational operations without I/O, and **concrete** paths, which inherit from pure paths but also provide I/O operations.
- If you've never used this module before or just aren't sure which class is right for your task, **Path** is most likely what you need. It instantiates a concrete path for the platform the code is running on.
- Pure paths are useful in some special cases; for example:
  - If you want to manipulate Windows paths on a Unix machine (or vice versa). You cannot instantiate a WindowsPath when running on Unix, but you can instantiate PureWindowsPath.
  - You want to make sure that your code only manipulates paths without actually accessing the OS. In this case, instantiating one of the pure classes may be useful since those simply don't have any OS-accessing operations.

Path Class

Demo Program: path0.py

This example is also an example for the **print_local** series. As you can see, it is way much easier.

filepath to be used to open files

Path object to hold current directory

```
from pathlib import Path

directory = Path(".")
filepath = directory / "path0.py"

print("File Path: ", filepath)
print()
print()

if filepath.exists():
    with open(filepath, "r") as f:
        fstring = f.read()
        print(fstring)
        f.close()
```

Delimiter for the path and file

Local file name

With statement:
Another way to write
f = open(filepath, “r”)

Print out the file like all print_locals did.

Output

```
File Path:  path0.py


from pathlib import Path

directory = Path(".")
filepath = directory / "path0.py"

print("File Path: ", filepath)
print()
print()

if filepath.exists():
    with open(filepath, "r") as f:
        fstring = f.read()
        print(fstring)
        f.close()
```

Check if path exists and a path is for directory (exists() and is_dir()

A Path added with a / and file name is still a path:

```
directory = Path(".")
filepath = directory / "path1.py"
```

Check if a directory or file exists or not and check if they are directory or not:

```
print("Check if the directory exists: ", directory.exists())
print("Check if Current directory is a directory: ", directory.is_dir())
```

Local File name can also be used to build a path object:

```
p = Path("path1.py")
```

```python
from pathlib import Path

directory = Path(".")
filepath = directory / "path1.py"

print("Check if the directory exists: ", directory.exists())
print("Check if Current directory is a directory: ", directory.is_dir())
print("Check if the filepath exists: ", filepath.exists())
print("Check if Current filepath is a directory: ", filepath.is_dir())

p = Path("path1.py")
print("Check if path1.py exists: ", p.exists())
print("Check if path1.py is a directory: ", directory.is_dir())

if filepath.exists():
    with open(filepath, "r") as f:
        fstring = f.read()
        print(fstring)
        f.close()
```

```
Check if the directory exists:  True
Check if Current directory is a directory:  True
Check if the filepath exists:  True
Check if Current filepath is a directory:  False
Check if path1.py exists:  True
Check if path1.py is a directory:  True
from pathlib import Path

directory = Path(".")
filepath = directory / "path1.py"

print("Check if the directory exists: ", directory.exists())
print("Check if Current directory is a directory: ", directory.is_dir())
print("Check if the filepath exists: ", filepath.exists())
print("Check if Current filepath is a directory: ", filepath.is_dir())

p = Path("path1.py")
print("Check if path1.py exists: ", p.exists())
print("Check if path1.py is a directory: ", directory.is_dir())

if filepath.exists():
    with open(filepath, "r") as f:
        fstring = f.read()
        print(fstring)
        f.close()
```

File System Traversal: (1) Find all sub-directory for a directory

Demo Program: path2.py

```
# list all directory under a directory
from pathlib import Path
root = Path("C:\\")
sub_directory = [x for x in root.iterdir() if x.is_dir()]

for d in sub_directory:
    print(d)
```

Make a list with all x which are directory under root

iterdir(): Iterate through all the directories under root path (C:\\ which is top level C: drive)

```
C:\$Recycle.Bin
C:\Bridge
C:\Config.Msi
C:\Documents and Settings
C:\Eric_Chou
C:\GNAT
C:\GNUWin
C:\hp
C:\inetpub
C:\Intel
C:\OneDriveTemp
C:\PerfLogs
C:\Photo
C:\Program Files
C:\Program Files (Night)
C:\Program Files (x86)
C:\ProgramData
C:\Python
C:\Recovery
C:\Software Downloaded
C:\SWSetup
C:\System Volume Information
C:\SYSTEM.SAV
C:\Users
C:\Windows
```

File System Traversal: (2) Recursively Searching for Subdirectory

Demo Program: path3.py

```
from pathlib import Path
def sub_dir(d):
    s = [x for x in d.iterdir() if x.is_dir()]
    for dd in s:
        print(dd)
        sub_dir(dd)

# put a correct directory in root
root = Path("C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System")
sub_dir(root)
```

Print the subdirectory and recursively search for sub-sub-directories

Make a list of subdirectories

All sub-directories under

Path("C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System")

```
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\.idea
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Custom
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Overview
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter2
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter2\.idea
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter2\File Reading
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter2\File System
C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter2\path
```

File System Traversal: (3) Recursively Searching for all Files and Sub-directories

Demo Program: path4.py

```
from pathlib import Path
def sub_dir(d):
    s = [x for x in d.iterdir() if x.is_dir()]
    files = [x for x in d.iterdir() if x.is_file()]
    for f in files:
        print("File: ", f)
    for dd in s:
        print("Dir: ", dd)
        sub_dir(dd)

# put a correct directory in root
root = Path("C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System\\Chapter1")
sub_dir(root)
```

Find all the sub-directory under a directory

Find all the files under a directory

Files just print out the file path

Recursively Search for the sub-directories

Root Set only to Chapter 1

Output:

Files and Directory Listing for Chapter 1 Only

```
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\.idea
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\.idea\Chapter1.iml
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\.idea\misc.xml
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\.idea\modules.xml
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\.idea\workspace.xml
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\data.txt
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\doWhileInput.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\eof.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\eof2.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\eof3.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\line_count.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\nestedexcept.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\nestedfinally.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\nestexcept2.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced\usdeclar.txt
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Custom
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Custom\custom1.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Custom\custom2.py
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\data.txt
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\exception01.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\exception02.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\exception03.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\exception04.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\exception05.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\exception06.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\exception07.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\exception08.py
File:  C:\Eric Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception\foo.py
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType\functionException.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType\functionException2.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType\functionException3.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType\functionException4.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType\iovalue.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType\iovalue2.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType\valueerror.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType\zerodivision.py
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Overview
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Overview\fristexception.py
File:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Overview\oops.py
```

File System Traversal: (4) Recursively Searching for all Files and Sub-directories

Demo Program: path5.py

Note:

1. files is a list of Path object. To convert a Path object to string. Use the toString method in Python str(f).
2. Then, use split("\\") to convert the string into array of directory name and files names
3. Get the last file name part. Then, print the file name.

```
from pathlib import Path
def sub_dir(d):
    s = [x for x in d.iterdir() if x.is_dir()]
    files = [x for x in d.iterdir() if x.is_file()]
    for f in files:
        fstring = str(f)                # use str to perform toString() function to covnert the path to string
        fn = fstring.split("\\")
        fname = fn[len(fn)-1]           # locate the file name part
        print(fname, end=" ")
    print()
    for dd in s:
        print("Dir: ", dd)
        sub_dir(dd)

# put a correct directory in root
root = Path("C:\\Eric_Chou\\Python Course\\Python Object-Oriented Programming with Libraries\\PyDev\\U2 IO System\\Chapter1")
sub_dir(root)
```

Output:

```
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\.idea
Chapter1.iml misc.xml modules.xml workspace.xml
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\advanced
data.txt doWhileInput.py eof.py eof2.py eof3.py line_count.py nestedexcept.py nestedfinally.py nestexcept2.py usdeclar.txt
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Custom
custom1.py custom2.py
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Exception
data.txt exception01.py exception02.py exception03.py exception04.py exception05.py exception06.py exception07.py exception08.py foo.py
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\ExceptionType
functionException.py functionException2.py functionException3.py functionException4.py iovalue.py iovalue2.py valueerror.py zerodivision.py
Dir:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter1\Overview
fristexception.py oops.py
```

os Module

LECTURE 8

Python **os** Module

- 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 methods.
- The **rename()** method takes two arguments, the current filename and the new filename

Syntax:

```
os.rename(current_file_name, new_file_name)
```

Following is the example to rename an existing file **test1.txt**:

```
import os

os.rename(“test1.txt”, “test2.txt)
```

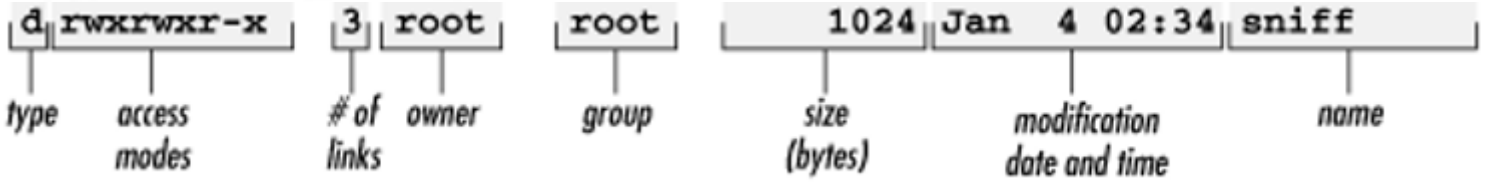
Python os Module

os module provides some os file and directory methods …

- **os.delete(file_name)** : delete a file
- **os.mkdir("newdir")**: make a new directory
- **os.chdir("newdir")**: change current working directory
- **os.getcwd()**: get current working directory in string
- **os.rmdir("dirname")**: remove a directory
- **os.listdir([dir=,])**: get the list of files in this directory.

os.path module (from os import path)

```
os.path.join(s1,s2,...)  # Join pathname parts together
os.path.getsize(path)    # Get file size of path
os.path.getmtime(path)   # Get modify time of path
os.path.getatime(path)   # Get access time of path
os.path.getctime(path)   # Get creation time of path
os.path.exists(path)     # Check if path exists
os.path.isfile(path)     # Check if regular file
os.path.isdir(path)      # Check if directory
os.path.islink(path)     # Check if symbolic link
os.path.basename(path)   # Return file part of path
os.path.dirname(path)    # Return dir part of
os.path.abspath(path)    # Get absolute path
```



Python os Variables	
altsep	Alternative sep
curdir	Current dir string
defpath	Default search path
devnull	Path of null device
extsep	Extension separator
linesep	Line separator
name	Name of OS
pardir	Parent dir string
pathsep	Patch separator
sep	Path separator
Registered OS names: "posix", "nt", "mac", "os2", "ce", "java", "riscos"	

os.listdir() - python 3
Demo Program: os1.py

- Import os module may complete similar tasks for Python programs just like pathlib.
- pathlib is available after 3.6.x

```
import os
currentDir = os.getcwd()
print("Current working direcotry: ", currentDir)
arr = os.listdir()
for f in arr:
    print("File: ", arr)
```

Output:

```
Current working direcotry:  C:\Eric_Chou\Python Course\Python Object-Oriented Programming with Libraries\PyDev\U2 IO System\Chapter2\os
File:  ['os1.py']
```

List file names end with .py

Demo Program: os2.py

```
import os
files = [x for x in os.listdir() if x.endswith(".py")]
for f in files:
    print(f)
```

Run os2

```
C:\Python\Python36\python.exe
os1.py
os2.py
```

os.path.join()

Creates a fully-expanded pathname

```
dirname = '/foo/bar'     filename = 'name'

os.path.join(dirname,filename)

'/foo/bar/name'
```

Aware of platform differences ('/' vs. '\')

```
for name in cachefiles:
    data = open(os.path.join(cachedir,name),"rb").read()
    index = 0
    while True:
        m = request_pat.search(data,index)
        if not m: break
        print m.group(1)
        index = m.end()
```

sys Module

LECTURE 9

Python sys Module

sys module provides a number of functions and variables that can be used to manipulate different parts of Python runtime environment.

Python sys Variables	
argv	Command line args
builtin_module_names	Linked C modules
byteorder	Native byte order
check_interval	Signal check frequency
exec_prefix	Root directory
executable	Name of executable
exitfunc	Exit function name
modules	Loaded modules
path	Search path
platform	Current platform
stdin, stdout, stderr	File objects for I/O
version_info	Python version info
winver	Version number

Python sys.argv	
sys.argv[0]	foo.py
sys.argv[1]	bar
sys.argv[2]	-c
sys.argv[3]	qux
sys.argv[4]	--h

sys.argv for the command:

$ python foo.py bar -c qux --h

command-line arguments

The **argv** list contains the arguments passed to the script, when the interpreter was started. The first item contains the name of the script itself.

Demo Program: sys_ex1.py

```
import sys
print "script name: %s" % sys.argv[0]
print len(sys.argv)

$ python sys_ex1.py

Output:
Script name: sys_ex1.py
1
```

Python Sys Module

- The sys module is a quick way to pull in arguments given at the command line into your script

- The sys arguments are passed in as a list starting with 0 is the script name and 1 is the next argument, 2 is the next and so on

```
import sys

script = sys.argv[0]
ip = sys.argv[1]
port = sys.argv[2]

print "[+] The script name is: "+script
print "[+] The IP is: "+ip+" and the port is: "+port
```

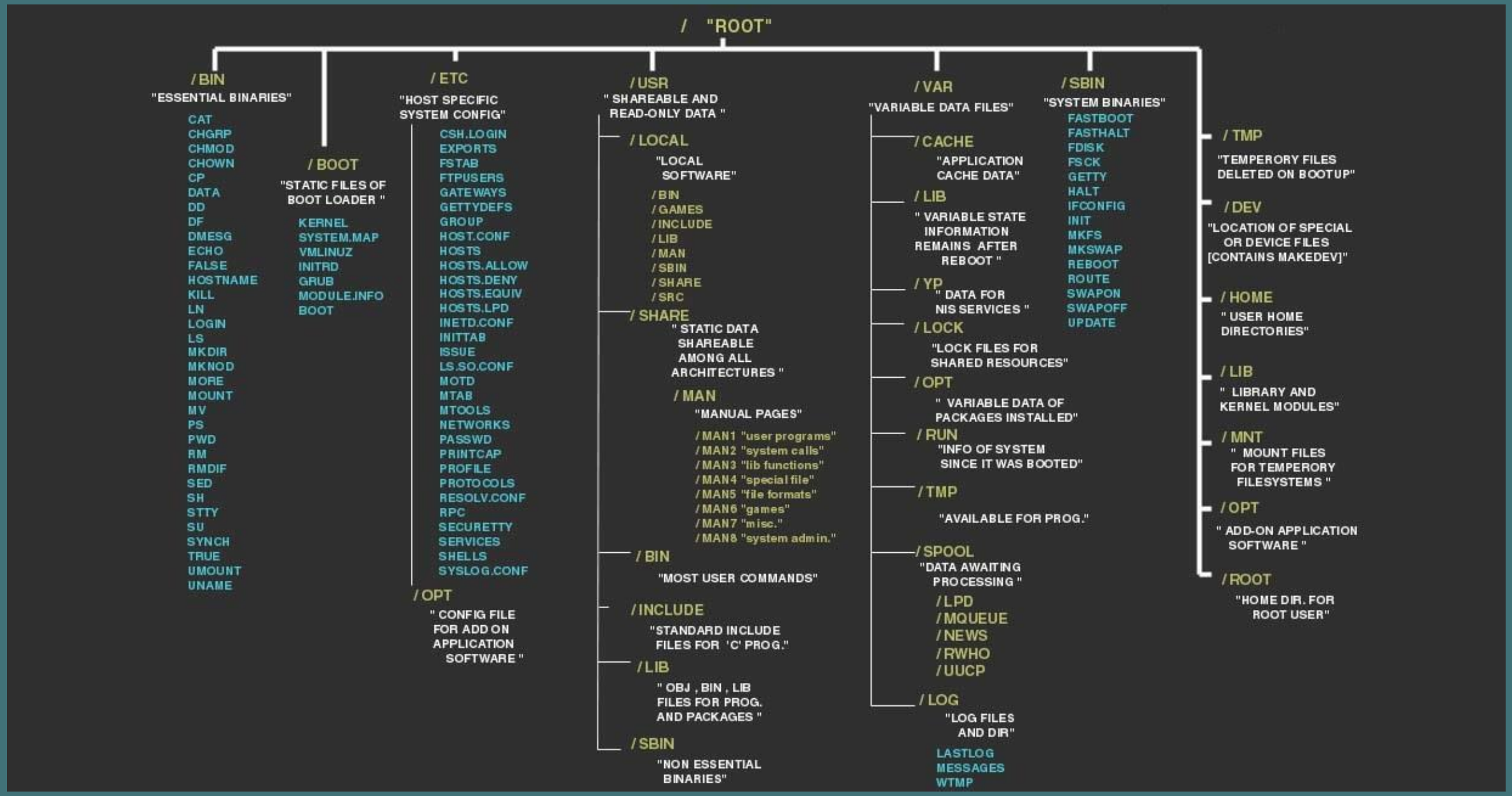
```
~$ python sys.py 8.8.8.8 53
[+] The script name is: sys.py
[+] The IP is: 8.8.8.8 and the port is: 53
```

Exit the Program: exit()

- When you reach the end of the main program, the interpreter automatically terminated.
- If you need to exit in midflight, you can call the **sys.exit** function instead
- This function takes an optional integer value, which is returned to the calling program.
- If it is an integer, **zero** is considered “successful termination” and any **nonzero** value is considered “abnormal termination”.

Linux File System

LECTURE 10


/ "ROOT"
/ BIN
"ESSENTIAL BINARIES"
CAT
CHGRP
CHMOD
CHOWN
CP
DATA
DD
DF
DMESG
ECHO
FALSE
HOSTNAME
KILL
LN
LOGIN
LS
MKDIR
MKNOD
MORE
MOUNT
MV
PS
PWD
RM
RMDIF
SED
SH
STTY
SU
SYNCH
TRUE
UMOUNT
UNAME
/ BOOT
"STATIC FILES OF BOOT LOADER "
KERNEL
SYSTEM.MAP
VMLINUZ
INITRD
GRUB
MODULE.INFO
BOOT
/ ETC
"HOST SPECIFIC SYSTEM CONFIG"
CSH.LOGIN
EXPORTS
FSTAB
FTPUSERS
GATEWAYS
GETTYDEFS
GROUP
HOST.CONF
HOSTS
HOSTS.ALLOW
HOSTS.DENY
HOSTS.EQUIV
HOSTS.LPD
INETD.CONF
INITTAB
ISSUE
LS.SO.CONF
MOTD
MTAB
MTOOLS
NETWORKS
PASSWD
PRINTCAP
PROFILE
PROTOCOLS
RESOLV.CONF
RPC
SECURETTY
SERVICES
SHELLS
SYSLOG.CONF
/ OPT
" CONFIG FILE FOR ADD ON APPLICATION SOFTWARE "
/ USR
" SHAREABLE AND READ-ONLY DATA "
/ LOCAL
"LOCAL SOFTWARE"
/ BIN
/ GAMES
/ INCLUDE
/ LIB
/ MAN
/ SBIN
/ SHARE
/ SRC
/ SHARE
" STATIC DATA SHAREABLE AMONG ALL ARCHITECTURES "
/ MAN
"MANUAL PAGES"
/ MAN1 "user programs"
/ MAN2 "system calls"
/ MAN3 "lib functions"
/ MAN4 "special file"
/ MAN5 "file formats"
/ MAN6 "games"
/ MAN7 "misc."
/ MAN8 "system admin."
/ BIN
"MOST USER COMMANDS"
/ INCLUDE
"STANDARD INCLUDE FILES FOR 'C' PROG."
/ LIB
" OBJ , BIN , LIB FILES FOR PROG. AND PACKAGES "
/ SBIN
"NON ESSENTIAL BINARIES"
/ VAR
"VARIABLE DATA FILES"
/ CACHE
"APPLICATION CACHE DATA"
/ LIB
" VARIABLE STATE INFORMATION REMAINS AFTER REBOOT "
/ YP
" DATA FOR NIS SERVICES "
/ LOCK
"LOCK FILES FOR SHARED RESOURCES"
/ OPT
" VARIABLE DATA OF PACKAGES INSTALLED"
/ RUN
"INFO OF SYSTEM SINCE IT WAS BOOTED"
/ TMP
"AVAILABLE FOR PROG."
/ SPOOL
"DATA AWAITING PROCESSING "
/ LPD
/ MQUEUE
/ NEWS
/ RWHO
/ UUCP
/ LOG
"LOG FILES AND DIR"
LASTLOG
MESSAGES
WTMP
/ SBIN
"SYSTEM BINARIES"
FASTBOOT
FASTHALT
FDISK
FSCK
GETTY
HALT
IFCONFIG
INIT
MKFS
MKSWAP
REBOOT
ROUTE
SWAPON
SWAPOFF
UPDATE
/ TMP
"TEMPERORY FILES DELETED ON BOOTUP"
/ DEV
"LOCATION OF SPECIAL OR DEVICE FILES [CONTAINS MAKEDEV]"
/ HOME
" USER HOME DIRECTORIES"
/ LIB
" LIBRARY AND KERNEL MODULES"
/ MNT
" MOUNT FILES FOR TEMPERORY FILESYSTEMS "
/ OPT
" ADD-ON APPLICATION SOFTWARE "
/ ROOT
"HOME DIR. FOR ROOT USER"

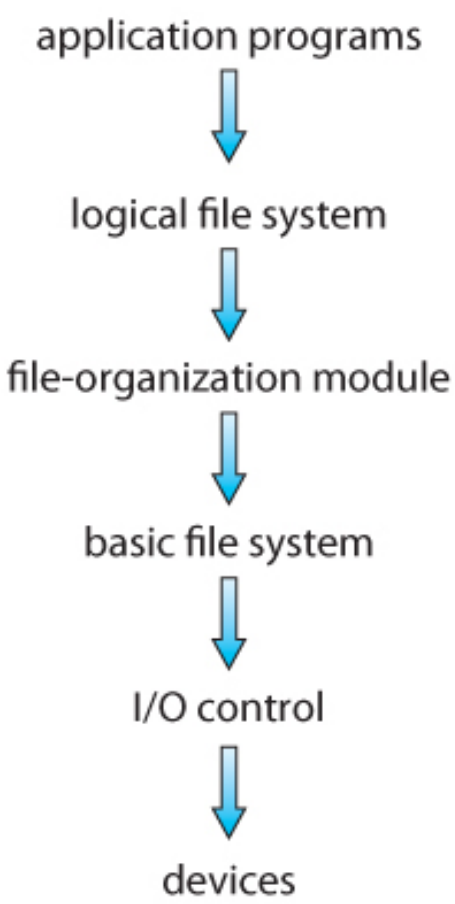
```
[root@desktop /root] # ls – l
total 558414
d rwxr-xr-x   5  root    root        1024 Dec 23 13:48 GNUstep
- rw-r--r--   1  root    root         331 Feb 11 10:19 Xrootenv.0
- rw-rw-r--   1  root    root         490 Jan  6 15:07 audio.cddb
- rw-r--r--   1  root    root    45254876 Jan  6 15:08 audio.wav
d rwxr-xr-x   2  root    root        1024 Feb 20 16:41 axhome
- rw-r--r--   1  root    root         900 Jan 18 20:15 conf
d rwxr-xr-x   2  root    root        1024 Dec 25 10:03 corel
- rw-r--r--   1  root    root         915 Jan 18 20:57 firewall
d rwxrwxr-x   2  root    root        1024 Jan  6 15:42 linux
d rwx------   2  root    root        1024 Jan  4 02:19 mail
d rwxr-xr-x   3  root    root        1024 Jan  4 01:49 mirror
- rwxr--r--   1  root    root          29 Dec 27 15:07 openn
d rwxr-xr-x   3  root    root        1024 Dec 26 13:24 scan
d rwxrwxr-x   3  root    root        1024 Jan  4 02:34 sniff
```

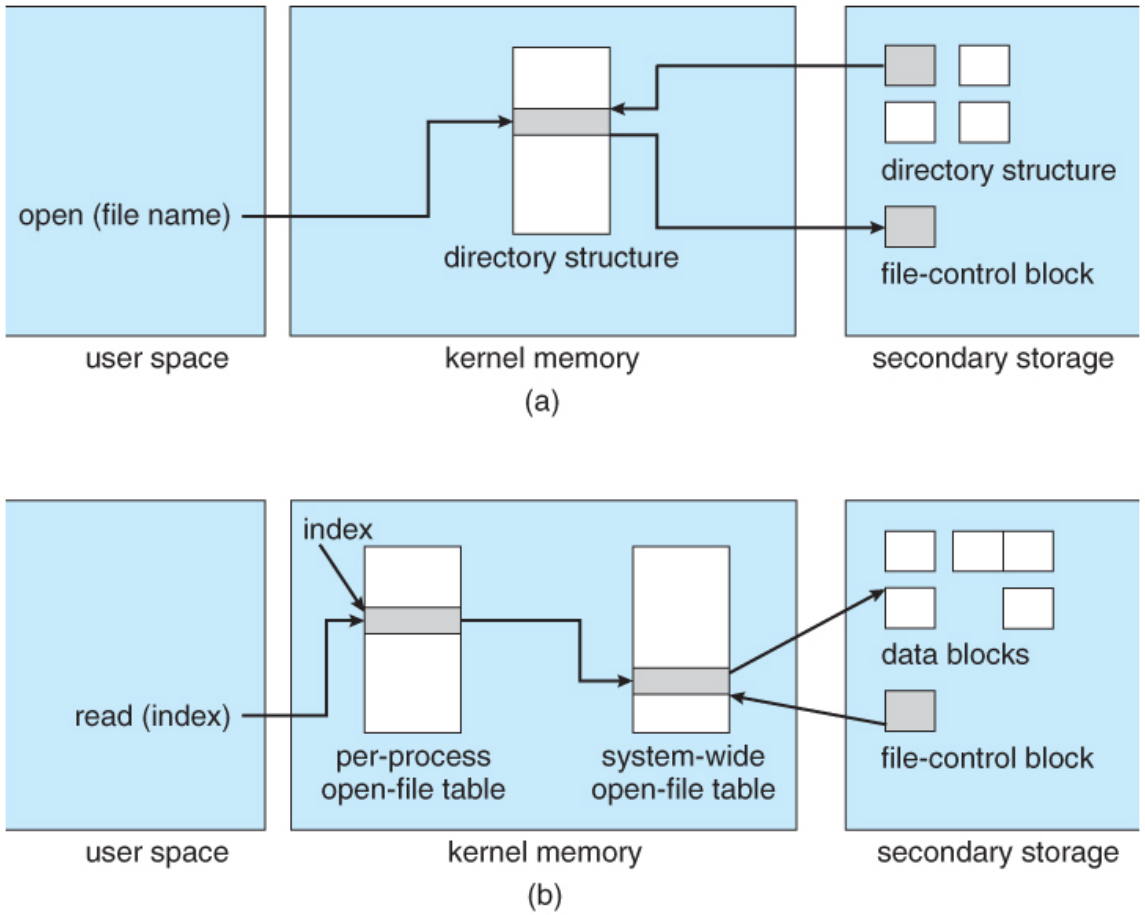
type | access modes | # of links | owner | group | size (bytes) | modification date and time | name

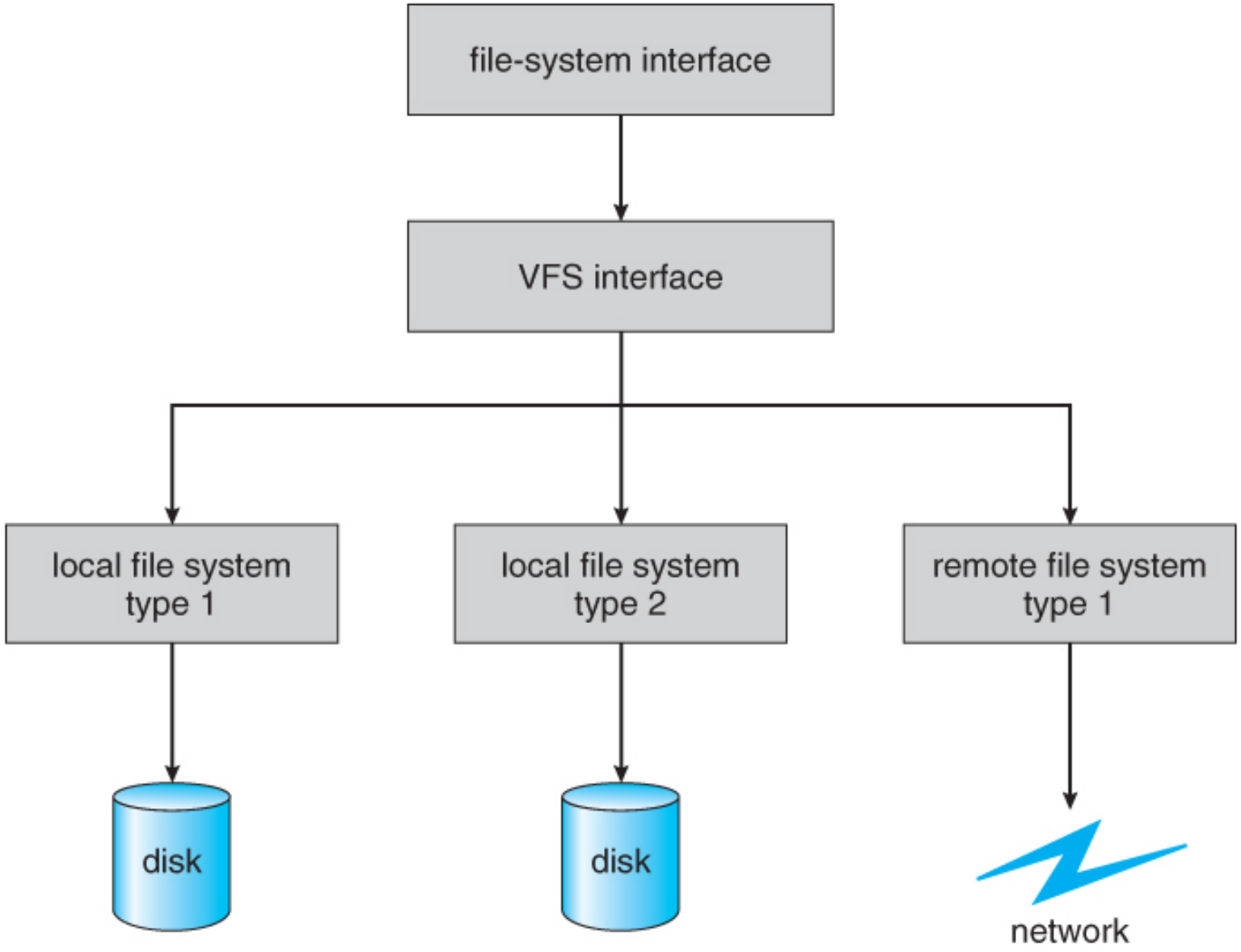
File Handler

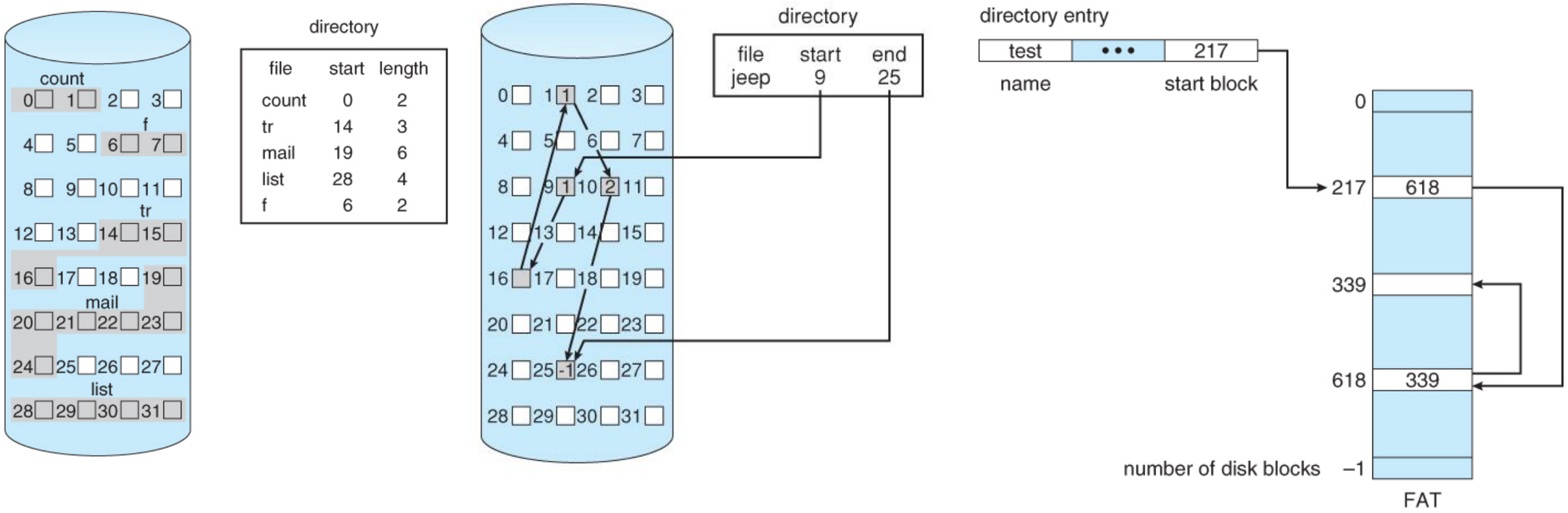
- file permissions
- file dates (create, access, write)
- file owner, group, ACL
- file size
- file data blocks or pointers to file data blocks

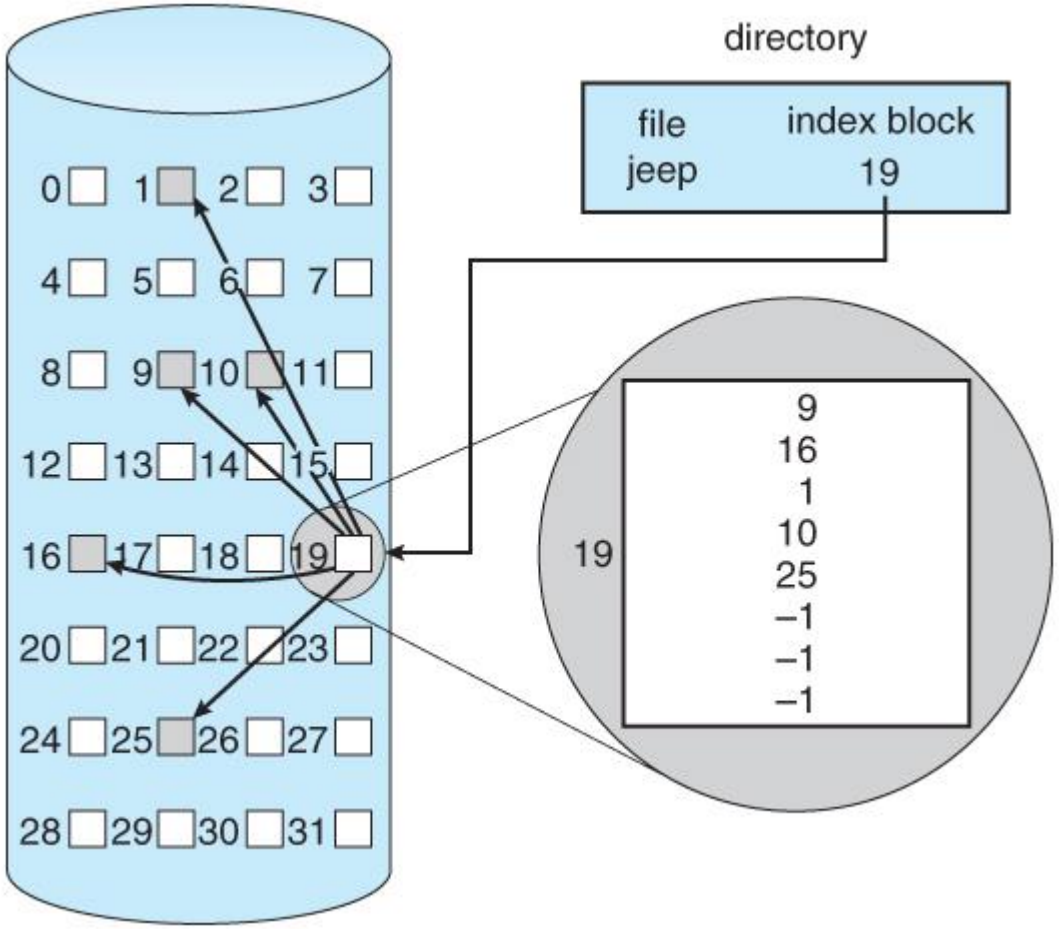
Linux File System






file-system interface
VFS interface
local file system type 1
local file system type 2
remote file system type 1
disk
disk
network





Unix/Linux inode System