File Management

Lesson Objectives

At the end of this lesson, you should be able to:

- Explain the concept of files
- Use the Python programming language to interact with files

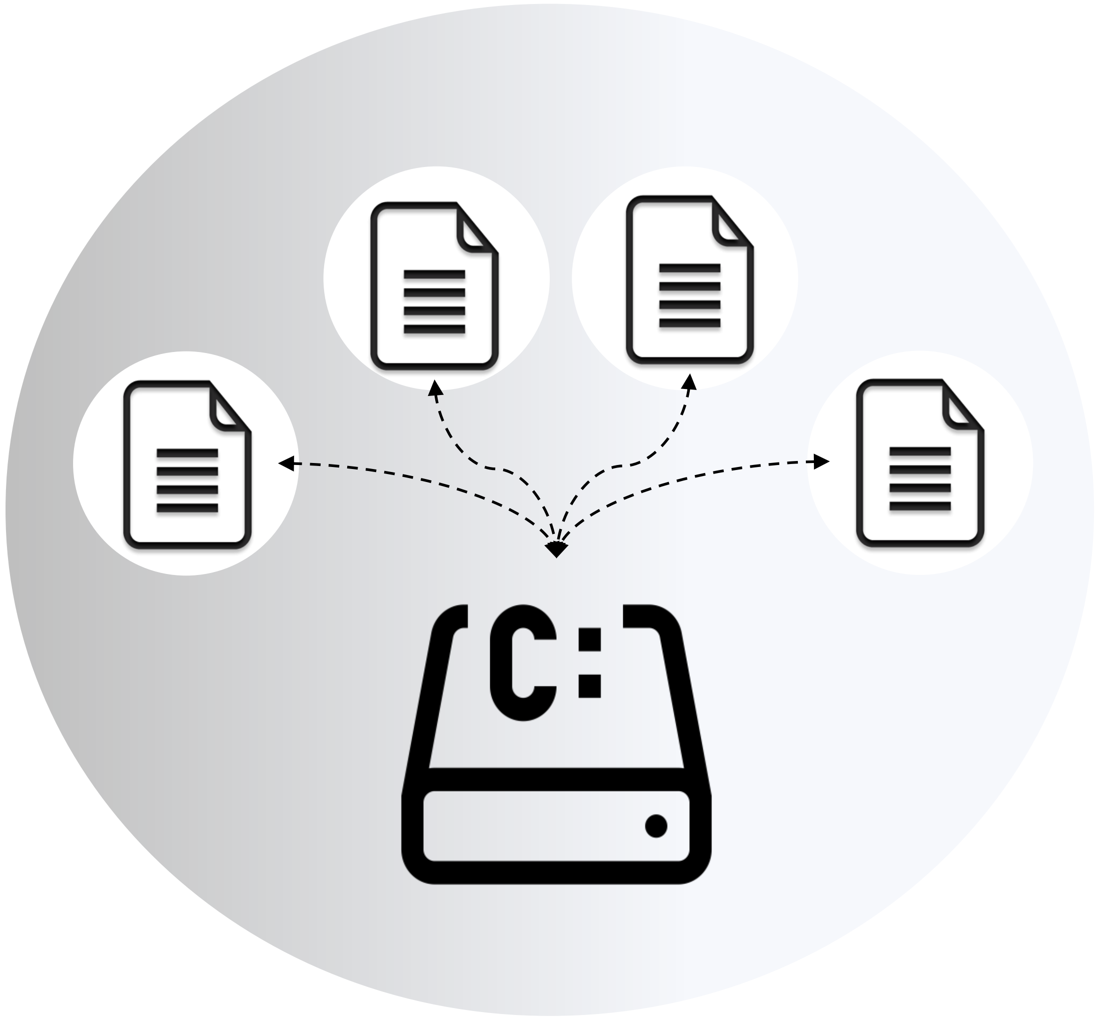
Topic Outline

File Basics

File

- A collection of data that is stored on secondary storage, e.g. a disk

- Accessing a file means establishing a connection between the file and the program and moving data between the two.



Two Types of Files

Files come in two general types:

1. Text files
   - Organized as ACSII or Unicode characters
   - Generally human readable
   - Main focus of this course

2. Binary files
   - All the information is based on specific encodings
   - Not human readable and contains non-readable information

Binary vs. Plain Text

Plain Text

+ Human readable, useful for certain file types

- Inefficient storage (each character requires ? bytes) :

  ASCII: 8 bits → 1byte

  Unicode: 32 bits → 4 bytes

Binary

+ More efficient storage, custom format

- Not human readable

Example

Assume there are 500 students in a class. If all the ages of the students were stored in both types of files, how many bytes would each entry be?

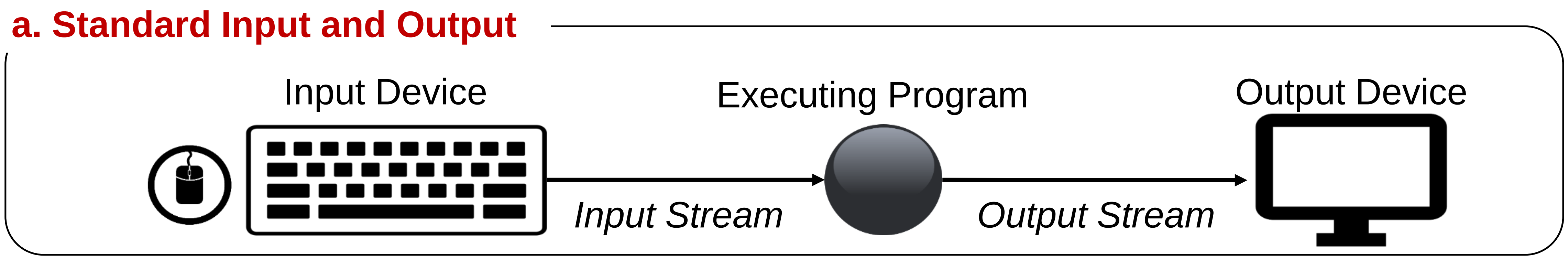
ASCII	Binary
'20'	20
'18'	18
'21'	21
'19'	19

ASCII = 2 bytes (two characters)
2 x 500 = 1000 bytes

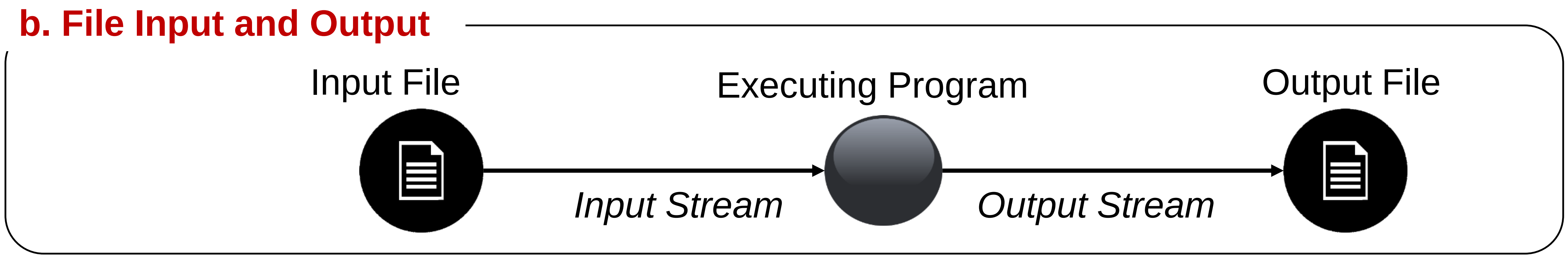
Binary= 1 byte
1 byte is 0-255 (enough for age)
1 x 500 = 500 bytes

File Interaction in Python

File Object or Stream

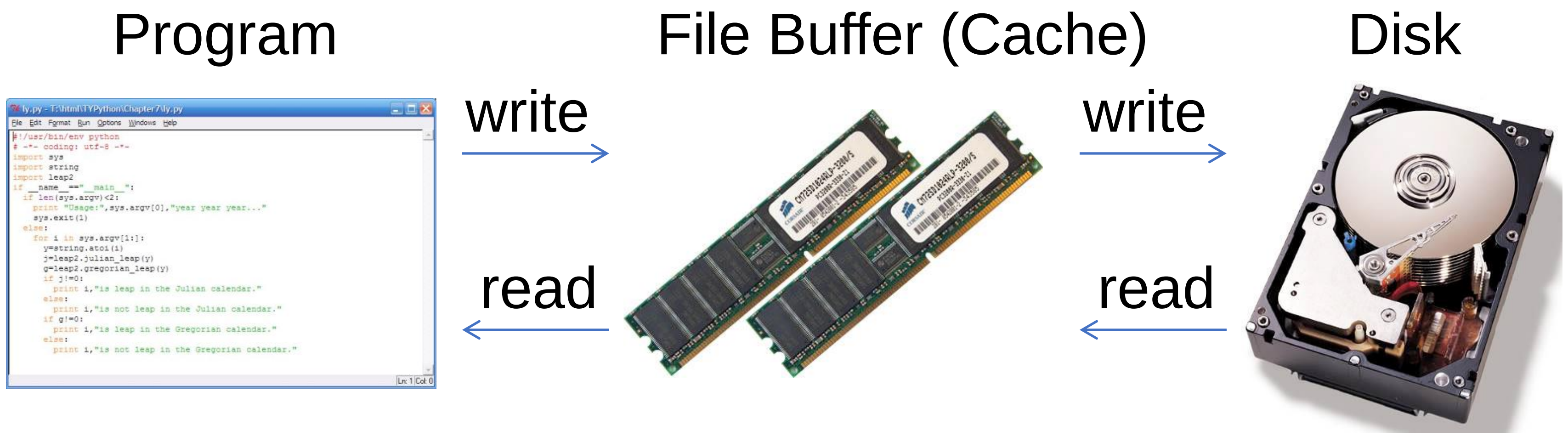


When opening a file, you create a file object or file stream that is a connection between the file and the program.



Buffering

- Reading from (writing into) a disk is very slow.
- A computer tries to read a lot of data from a file first
  - if you need the data, it will be “*buffered*” in the file object.
- The file object contains a copy of information from the file called a cache (pronounced “cash”).



Program
File Buffer (Cache)
Disk
write
read
write
read
ly.py - T:\html\TYPython\Chapter7\ly.py
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import string
import leap2
if __name__=="__main__":
  if len(sys.argv)<2:
    print "Usage:",sys.argv[0],"year year year..."
    sys.exit(1)
  else:
    for i in sys.argv[1:]:
      y=string.atoi(i)
      j=leap2.julian_leap(y)
      g=leap2.gregorian_leap(y)
      if j!=0:
        print i,"is leap in the Julian calendar."
      else:
        print i,"is not leap in the Julian calendar."
      if g!=0:
        print i,"is leap in the Gregorian calendar."
      else:
        print i,"is not leap in the Gregorian calendar."
```

Where Does the “Buffer” Reside?

- The file buffer contains the information from the file and provides the information to the program.

- Located in the file object

Creating a File Object

```
myFile = open("myFile.txt", "r")
```

- `myFile` is the file object.
- It contains the *buffer* of information.
- The first quoted string is the file name on disk, the second is the mode to open it (here, "r" means to read).

File Location

When opened, the name of the file can come in one of two forms:

- "file.txt" assumes the file name is file.txt, and it is located in the current program directory.

- "c:\bill\file.txt" is the fully qualified file name and includes the directory information.

File Modes

Mode	How Opened	File Exists	File Does Not Exist
‘r’	Read-only	Opens that file	Error
‘w’	Write-only	Clears the file contents	Creates and opens a new file
‘a’	Write-only	File contents left intact and new data appended at file’s end	Creates and opens a new file
‘r+’	Read and Write	Reads and overwrites from the file’s beginning	Error
‘w+’	Read and Write	Clears the file contents	Creates and opens a new file
‘a+’	Read and Write	File contents left intact and read and write at file’s end	Creates and opens a new file

File Encodings

- Text files
  - Default: UTF-8 (a variable-length encoding for Unicode)

- Binary files
  - Different files, depending on language/OS, will have different encodings!
  - Specify the encoding explicitly

```
open(“table.csv”, “r”, encoding=“windows-1252”)
```

- More about encodings
  - http://getpython3.com/diveintopython3/strings.html#boring-stuff

Strings or Bytes?

- If you interact with text files, remember:
  - files store things as characters (character encoding)
    - Default encoding: UTF-8

  - All access to text files is via strings
    - Read strings from a text file
    - Write strings to a text file

- How about binary files?
  - All access to binary files is via bytes (encoding)
  - More details
    - http://getpython3.com/diveintopython3/strings.html#boring-stuff

Reading a File

```
>>> aFile = open("temp.txt", "r")

>>> for line_str in aFile:
        print(line_str, end='')
```

```
First Line
Second Line
Third Line
```

temp.txt

```
First Line
Second Line
Third Line
```

Other Methods to Read Files

- `fileObject.readline()`
  - return the next line as a string.
- `fileObject.readlines()`
  - Return a list of all the lines from the file.
- `fileObject.read(N)`
  - Read N characters and returned a single string
  - If N is omitted, the entire file is read and returned as a single string

The `readline()` method

```
>>> aFile = open(“temp.txt”, “r”)

>>> first_str = aFile.readline()

>>> first_str
‘First Line\n’

>>> second_str = aFile.readline()

>>> second_str
‘Second Line\n’
```

temp.txt

```
First Line
Second Line
Third Line
```

The `readlines()` method

```
>>> aFile = open("temp.txt", "r")

>>> file_contents = aFile.readlines()

>>> file_contents

['First Line\n', 'Second Line']
```

temp.txt

```
First Line
Second Line
```

The `read()` method

```
>>> aFile = open("temp.txt", "r")

>>> aFile.read(1)
'F'

>>> aFile.read(2)
'ir'

>>> aFile.read()
'st Line\nSecond Line'

>>> aFile.read(1)
''
```

temp.txt

```
First Line
Second Line
```

Writing a Text File

```
>>> aFile = open(“temp.txt”, “w”)
>>> print(“first line”, file=aFile)
>>> print(“second line_”, file=aFile, end=‘’)
>>> print(“third line”, file=aFile)
>>> aFile.close()
```

What if you don’t specify this?

temp.txt

```
first line
second line_third line
```

temp.txt

```
```

Close the door behind you!

- Closing the file is important
    - the information in the fileObject buffer is “flushed” out of the buffer and into the file on disk

```
fileObject.close()
```

Automatic Closing

- Python 3:

```
with open("fileToRead.txt") as myFile:
    for line in myFile:
        print(line)
```

- File is automatically opened
  - Default file mode is read & file type is text.

- File is automatically closed at the end of the for loop.

Other methods to write Text Files

- `fileObject.write(str)`
  - write the string str to the file

- `fileObject.writelines(list)`
  - write a list of strings to the file

The `write()` method

```
>>> aFile = open("temp.txt", "w")
>>> aFile.write("First Line\n")
>>> aFile.write("Second Line\n")
>>> aFile.close()
```

temp.txt

```
First Line
Second Line
```

The `writelines()` method

```
>>> aFile = open(“temp.txt”, “w”)
>>> line_list = [“First Line\n”, “Second Line\n”]
>>> aFile.writelines(line_list)
>>> aFile.close()
```

temp.txt

```
First Line
Second Line
```

Errors?

- What if the file does not exist?

- Your program should behave gracefully if the file cannot be opened.
  - Exception handling
  - `try/except` block

File Facts

Current File Position

- Every file maintains a current file position.

- It is the current position in the file and indicates what will be read next.

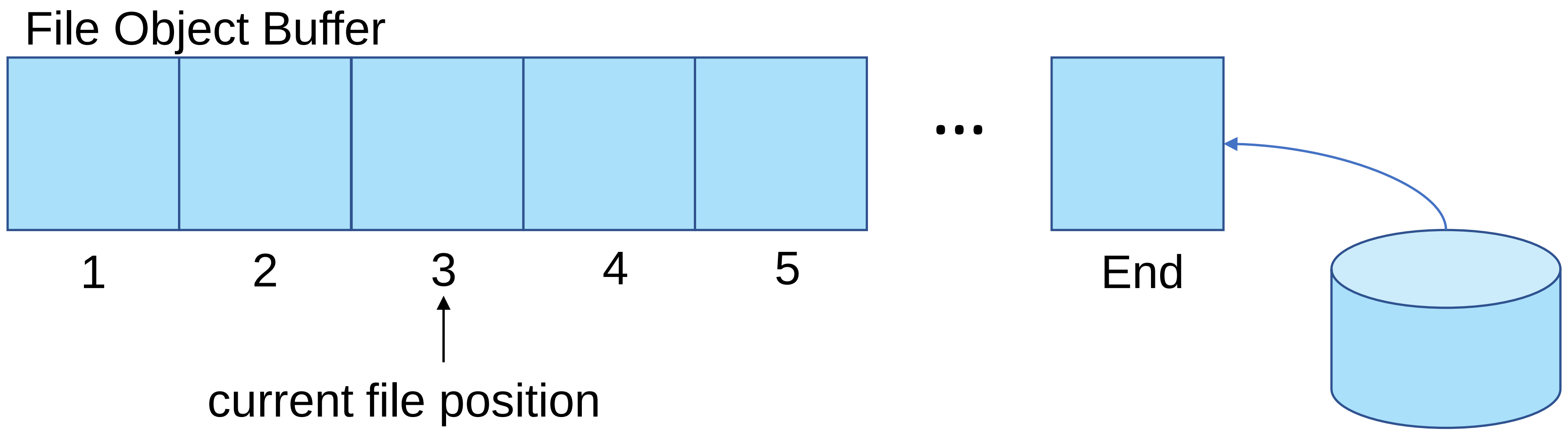
- It is set by the file mode.

File Modes

Mode	How Opened	File Exists	File Does Not Exist
‘r’	Read-only	Opens that file	Error
‘w’	Write-only	Clears the file contents	Creates and opens a new file
‘a’	Write-only	File contents left intact and new data appended at file’s end	Creates and opens a new file
‘r+’	Read and Write	Reads and overwrites from the file’s beginning	Error
‘w+’	Read and Write	Clears the file contents	Creates and opens a new file
‘a+’	Read and Write	File contents left intact and read and write at file’s end	Creates and opens a new file

File Buffer

- When the disk file is opened, the contents of the file are copied into the buffer of the file object.
- Think of the file object as a very big list
- The current file position is the present index to access the list.



The `tell()` method

- The `tell()` method tells you the current file position.
- The positions are in bytes from the beginning of the file:

  `fileObject.tell()` => 42

- This is not necessarily the same as the number of characters
  - depends on encoding, some characters take multiple bytes.
- Notice that, `read()` operates in characters

The tell() method - Example

```
>>> aFile = open("temp.txt", "r")
>>> aFile.tell()
0
>>> aFile.read(16)
'Dive into Python'
>>> aFile.read(1)
' '
>>> aFile.tell()
17
>>> aFile.read(1)
'是'
>>> aFile.tell()
20
```

Requires three bytes

temp.txt (UTF-8)

```
Dive into Python 是一本好書!
```

The `seek()` method

The `seek()` method updates the current file position to where you like (offset in bytes from the beginning of the file):

- `fo.seek(0)     # to the beginning of the file`
- `fo.seek(100) # to 100 bytes from beginning`

The `seek()` method

- Counting offset in bytes is a pain.
- `seek()` has a optional second argument:
  - 0: count from the beginning
  - 1: count from the current position in the file
  - 2: count from the end (usually paired with negative offset)

e.g. `fo.seek(-100,2)`
100 bytes before end of file

- In text files, only seeks relative to the beginning of the file are allowed
  - except `fo.seek(0, 2)` $\longrightarrow$ to the end of the file

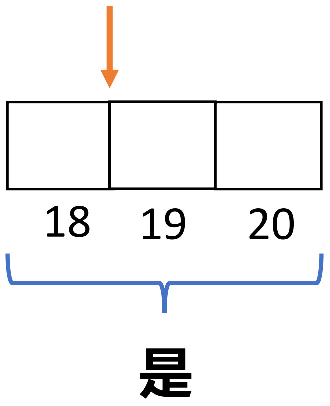
The `seek()` method - Example

```
>>> aFile = open("temp.txt", "r")
>>> aFile.read(17)
'Dive into Python '
>>> aFile.tell()
17
>>> aFile.read(1)
'是'
>>> aFile.tell()
20
>>> aFile.seek(18)
>>> aFile.read(1)
```

UnicodeDecodeError!!!!

temp.txt (UTF-8)

```
Dive into Python 是一本好書!
```



Reading Forward

- `read()`, `readline()`, `readlines()` move the current position forward.

- When you hit the end of the file, every read will just yield "" (empty string)
    - since you are at the end.

- You need to `seek` to the beginning to start again (or close and open, `seek()` is easier).

Summary

In this lesson, we have learned:

- The concept of files
- How to interact with files in Python