File Input & Output (or File I/O)

File *Input*: Getting Data from Files

All our programs so far got data from the user or the data was written into program

This week we will learn how to write programs that get data from files

The files are called *Input Files*

File *Output*: Storing Data in Files

All our programs so far printed results on the screen, where it disappeared after that

This week we will learn how to write programs that save data into files

The files are called *Output files*

Accessing Files

Types of files

Generally, there are two types of files:

Text file (.txt) contains data that has been encoded as text, using a scheme such as ASCII or Unicode.

Binary file contains data that has not been converted to text. Data is intended only for a program to read.

Actually, technically, all files are stored in binary, but a .txt file contains contigious 8-bit fields which represent characters.

Basically, anything that you can open with a text editor and is not gibberish

Accessing Files

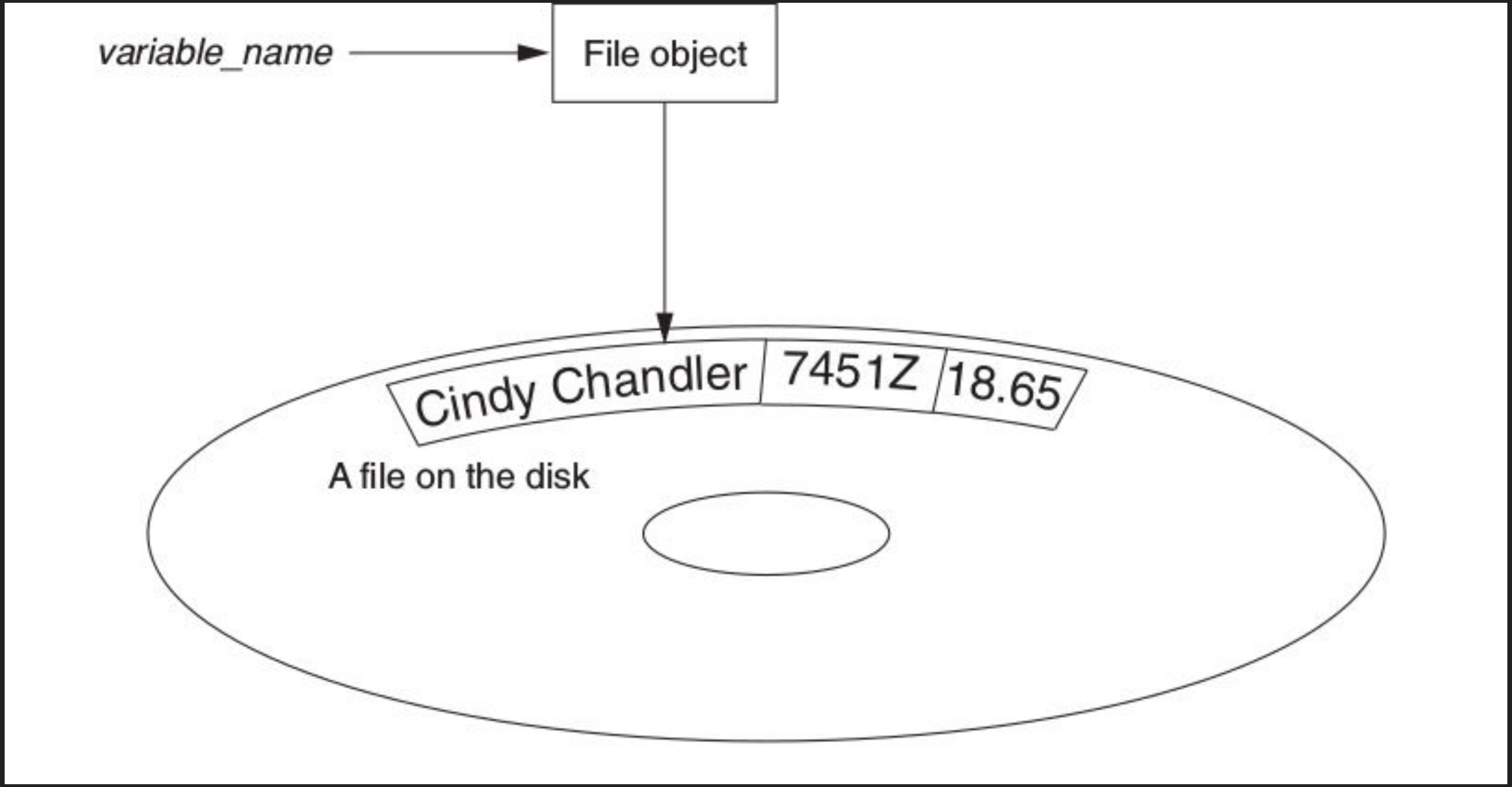
Two ways to access data in files

Sequential access: read file sequentially from beg
skip around in file (analogous to cassette tapes)

Direct access: can jump directly to any data in file (analogous to a CD)

We will focus on **sequential** access in this class *file_variable* = open (*filename, mode)*

A file object is an object that is associated with a specific file and provides a way for the program to work with that file.



Using Data Files: Three Steps

1. Open file
2. Read data from file *or* write data to file
3. Close file

Open a File

- To open a file, use the **open()** function
  - Give it the filename and desired file-access mode
  - Access modes are **r** (read), **w** (write), and **a** (append)
- Save the value **open()** returns in a variable
  - **variable = open(filename, mode)**
  - **myFile = open("datafile.dat", "r")**
- Filename can be simple or directory path
  - Simple: "myData.txt" (file must be in folder with program)
  - Directory path: "/user/alice/data/myData.txt"

Opening Files: Examples

- **inFile = open("monthlyRain.txt", "r")**
  - File must already exist and is read-only.
- **outFile = open("myData.txt", "w")**
  - If file doesn’t exist, create it. If it exists, erase it and prepare to write new data into it.
- **outFile = open("myData.txt", "a")**
  - If file doesn’t exist, create it. If it exists, prepare to add new data onto the end of the file.

Writing Data To Files

*file_variable.*write (*string)*

write is a *method* that belongs to a file object.

```
test_file = open(‘test.txt’, ‘w’)
test_file.write(‘hello world!’)
test_file.close()
```

Writing Data to Files

```
test_file = open(‘test.txt’, ‘w’)
test_file.write(‘Ada Lovelace\n’)
test_file.write(‘Grace Hopper\n’)
test_file.write(‘Katherine Johnson\n’)
test_file.close()
```

What happens if you remove the newline?

Limitations of file.write()

- It takes only one argument.
    - You **can't** give it many things to write at once, e.g., ~~ofile.write("foo", "bar")~~
    - But you **can** concatenate strings together, e.g., ofile.write("foo" + "bar")
- It takes only text strings as its argument.
    - It's not "smart" like print(). It does not convert data to strings.
    - **Your code must convert data to strings before writing it to a file.**
- It writes all data to the file as text.

Limitations of file.write()

- It takes only one argument
- It takes only text strings as its argument
- Examples:

```
a = 42
b = 37
ofile = open("results.txt", "w")
ofile.write(a, b)                    # Error!  More than 1 argument!
ofile.write(a + b)                         # Error!  Not a string!
petList = ["dog", "cat", "kangaroo", "fish"]
ofile.write(petList)                       # Error!  Not a string!
```

Limitations of file.write()

Must convert to string first

Must concatenate multiple strings into one

Examples

```
a = 42
b = 37
ofile = open("results.txt", "w")
ofile.write(str(a) + ", " + str(b))        # writes "42, 37”
ofile.write("\n”)                          # write newline (start new line in file)
ofile.write(str(a + b))                    # writes "79"
```

Reading Data from File

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

contents = myFile.read() # read entire file (as text string); stores in contents

line = myFile.readline() # read one line from file; store in line

myFile.close()           # close file
```

Reading Data from File

```
read_file = open('test.txt', 'r')

line1 = read_file.readline()
line2 = read_file.readline()
line3 = read_file.readline()

read_file.close()

print(line1)
print(line2)
print(line3)
```

There is a blank line displayed after each line in the output.

This is because each item that is read from the file ends with a newline character \n

Stripping Newline Characters

<string name>.rstrip(‘\n’)

```
read_file = open(‘test.txt’, ‘r’)

line1 = read_file.readline()
line2 = read_file.readline()
line3 = read_file.readline()

line1=line1.rstrip(‘\n’)
line2=line2.rstrip(‘\n’)
line3=line3.rstrip(‘\n’)
```

```
read_file.close()

print(line1)
print(line2)
print(line3)
```

Reading Data from File with *For* Loop

```
read_file = open('test.txt', 'r')
write_file = open('write_test.txt', 'w')

for line in read_file:
    write_file.write(line)
    print(line)

read_file.close()
write_file.close()
```

Each iteration will reference the next line in the file

No need for readline() !!!

Reading Data from File with *While* Loop

```
read_file = open('test.txt', 'r')
write_file = open('write_test.txt', 'w')

line=read_file.readline()

while line!='':
   write_file.write(line)
   print(line)
   line=read_file.readline()

read_file.close()
write_file.close()
```

In Python, the readline method returns an empty string ('') when it has attempted to read beyond the end of a file (EOF). This makes it possible to write a while loop that determines when the end of a file has been reached.