Course	Python 101
Term	
Week	
Date	
Chapter. Topic	6. Files and Exceptions

File I/O (Input/Output)

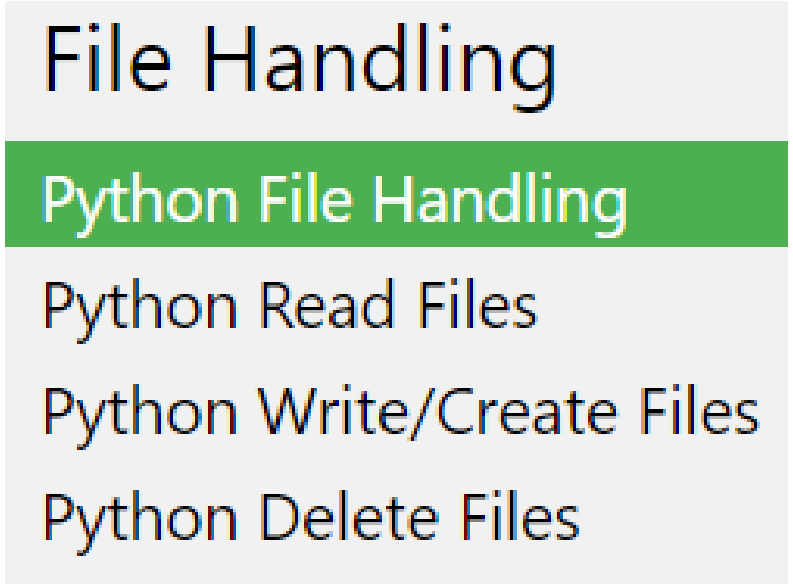
Siva R Jasthi

Computer Science and Cybersecurity

Metropolitan State University

Outline

- File Handling
- Opening the files
- Reading the files
- Writing to the files
- Closing the files
- Deleting the files



Outline

Text vs Binary Files

Relative Path vs Absolute Path

Without with

With with

Exceptions (try catch else finally raise)

Input/Output

Input: input( )

- The *input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline):

```
str = input("Enter your name: ")
print ("Your name is : ", str)
```

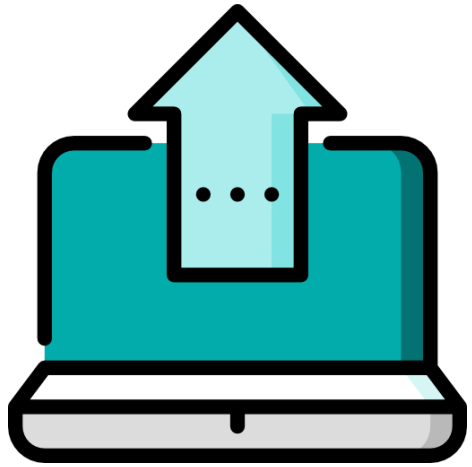
- This would prompt you to enter any string and it would display same string on the screen. When I typed "Hello Python!", it output is like this:

```
Enter your name: Python
Your name is : Hello Python
```

Output: Printing to the screen

- The simplest way to produce output is using the *print* statement where you can pass zero or more expressions, separated by commas.

```
print ("Sum of 2 + 4 is ", 2+4)
```



- Output for the above statement: would produce following result on your standard screen:

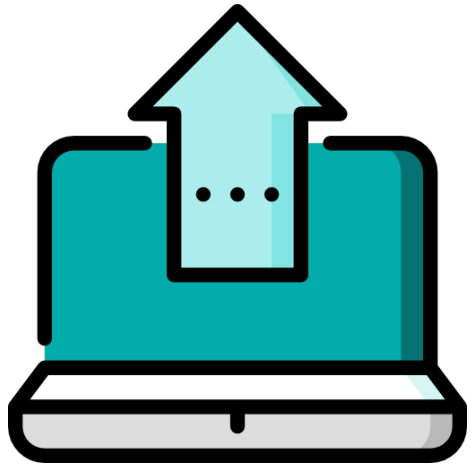
```
Sum of 2 + 4 is 6
```

Input and Output: from Files

- Input() reads from console
- Output() prints to the console

- In the same way, we can read from files and write to the files.

- Open() command – built-in function comes in handy.



File Handling: File I/O

- You can also read from files and write to the files.
- For writing or reading, you need to first OPEN the file.

- You use python's built-in function **open(file_name)** to open the files.

Built-in Functions

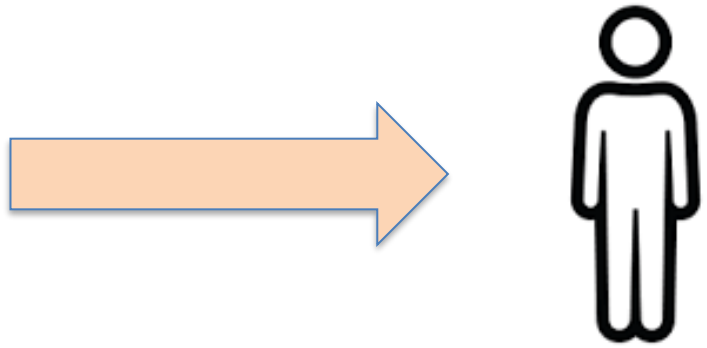
The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

File Name vs File Object

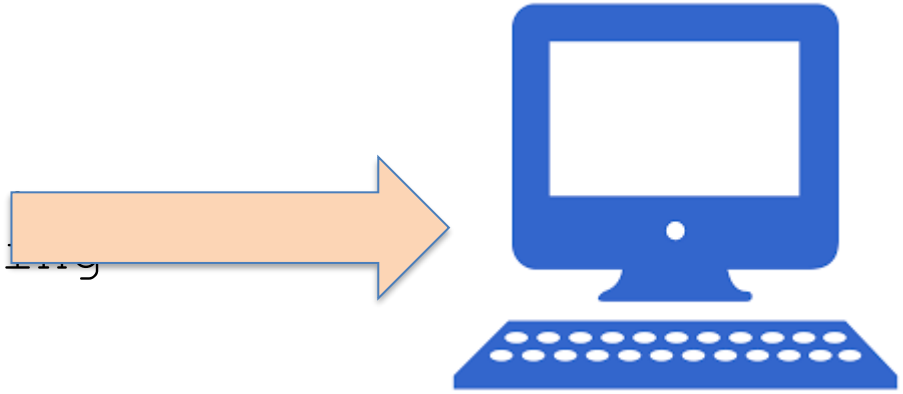
File Names (Strings):

- `C:\data\students_list.txt`
- `marks.txt`



File Objects (objects):

- `file_name = 'marks.txt'`
- `file_obj= open('marks.txt')`
- When you open the file for reading or writing, python creates a "file object"
- And all the operations are performed using this file_object



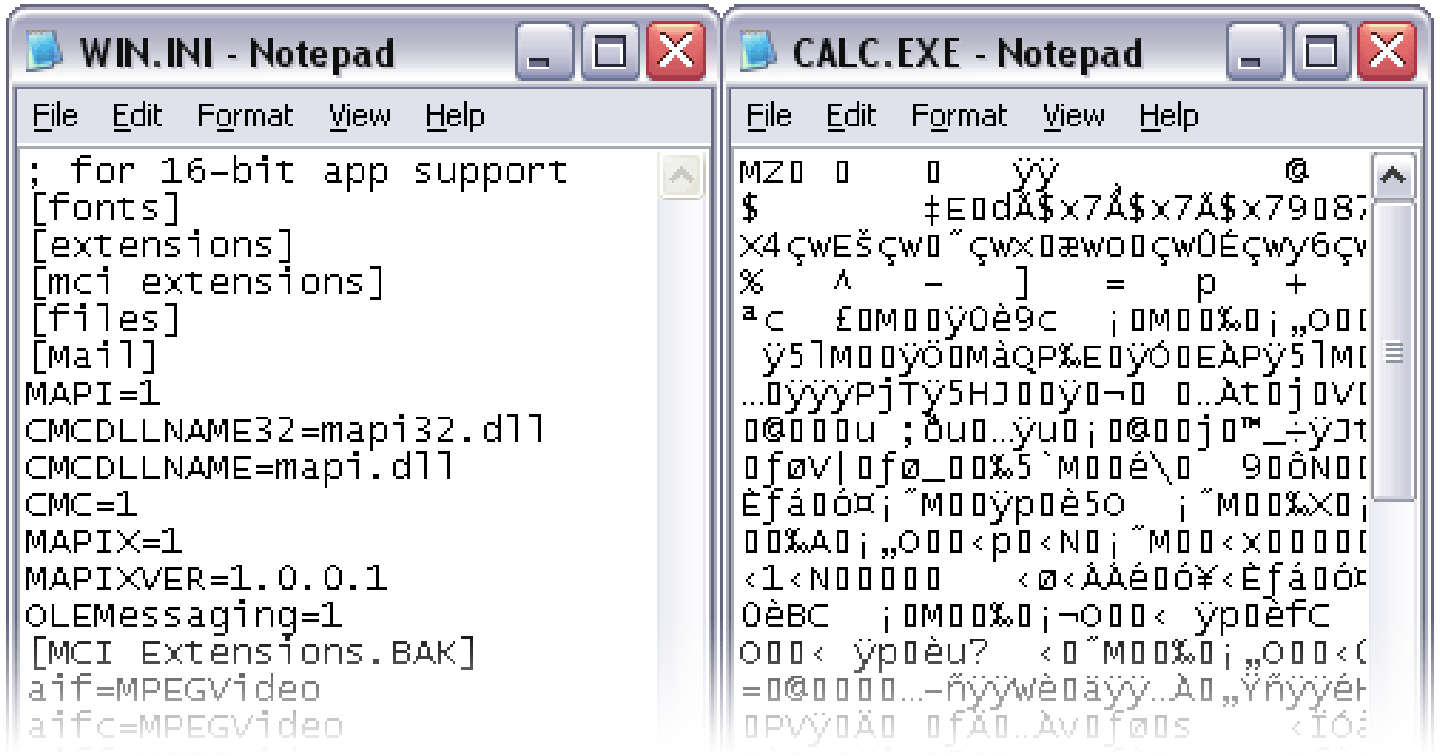
What type of file is it?

- We handle two types of files.

[1] Text file

[2] Binary File

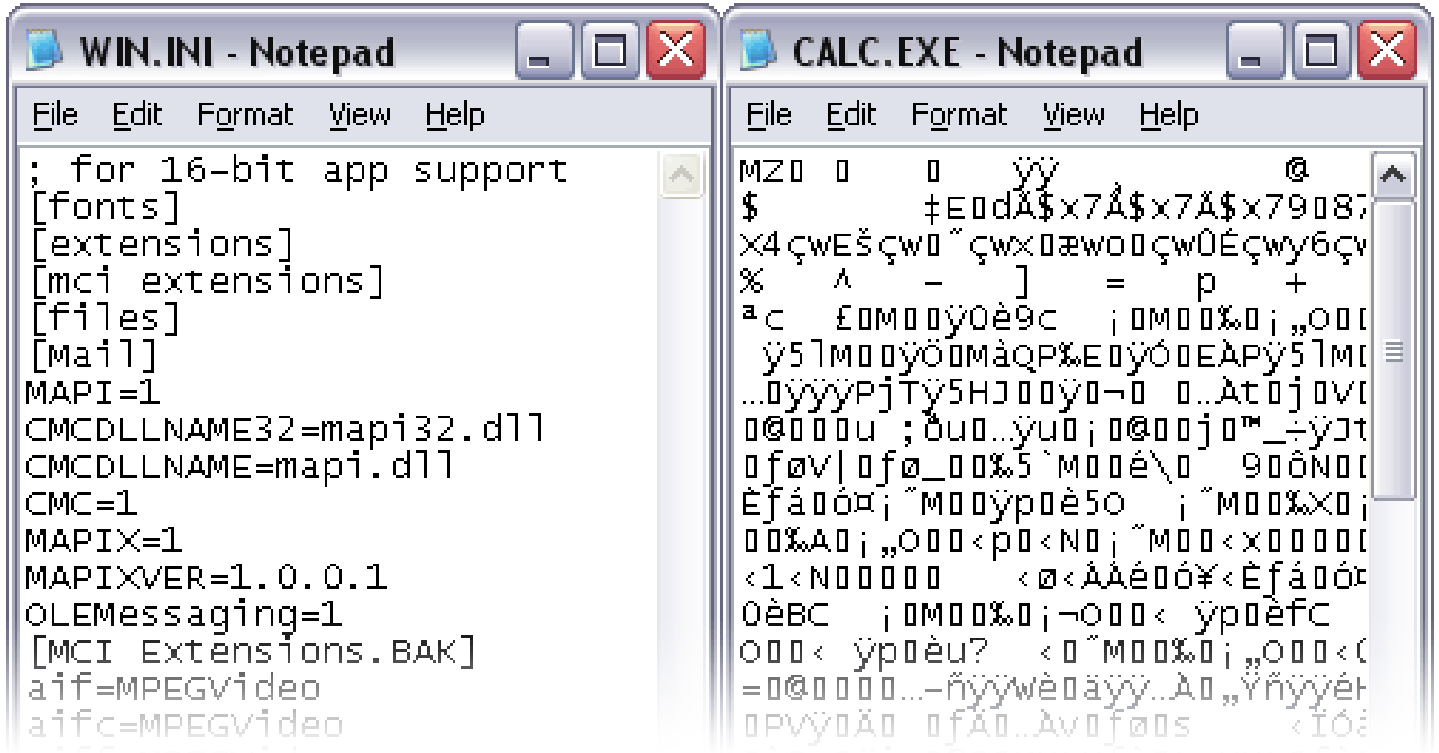
- If you can read the contents in Notepad or Visual Code or SublimeText or any of your favorite editors, it is a “Text File”. (eg: Python source code)
- However, if you open a file and all that you see is gibberish, it is a binary file (eg: JPEG or MP3 files)



Inform Python about the file type

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

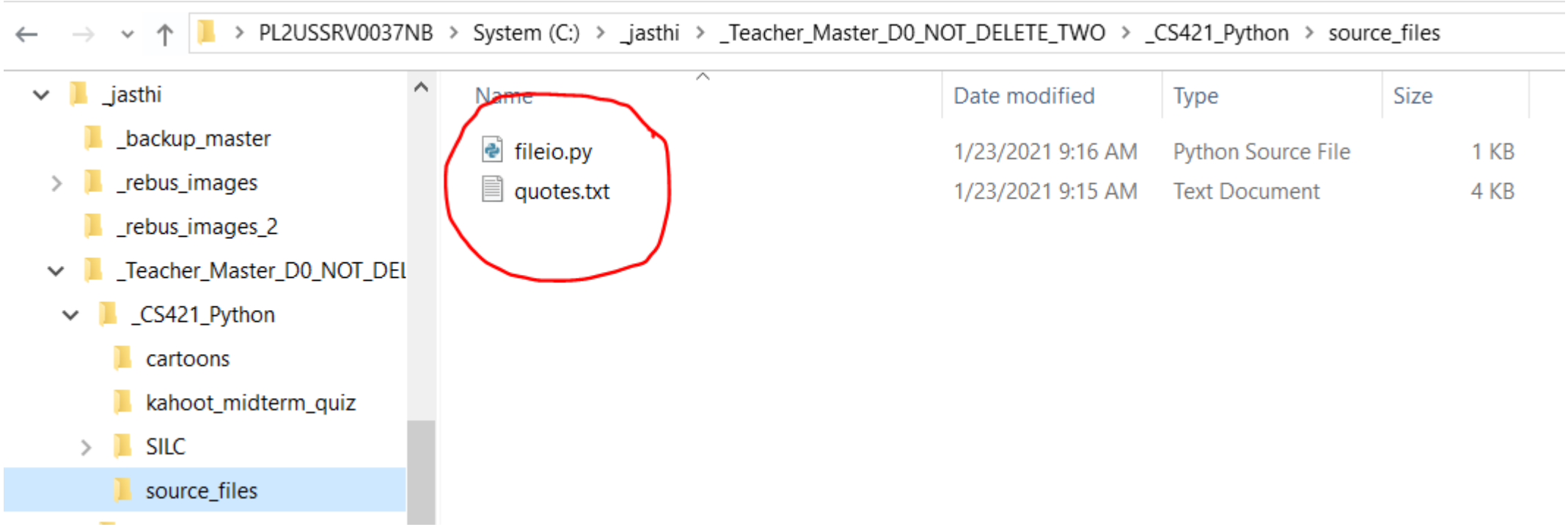


Where would Python look for the file?

file_obj = open("quotes.txt")

quotes.txt can be anywhere on your desktop.

Default Location: The source file (python program) looks for the file at the same location.



If the file is at a different location?

`file = open("quotes.txt")` will not work

We need to specify the full path (absolute path)

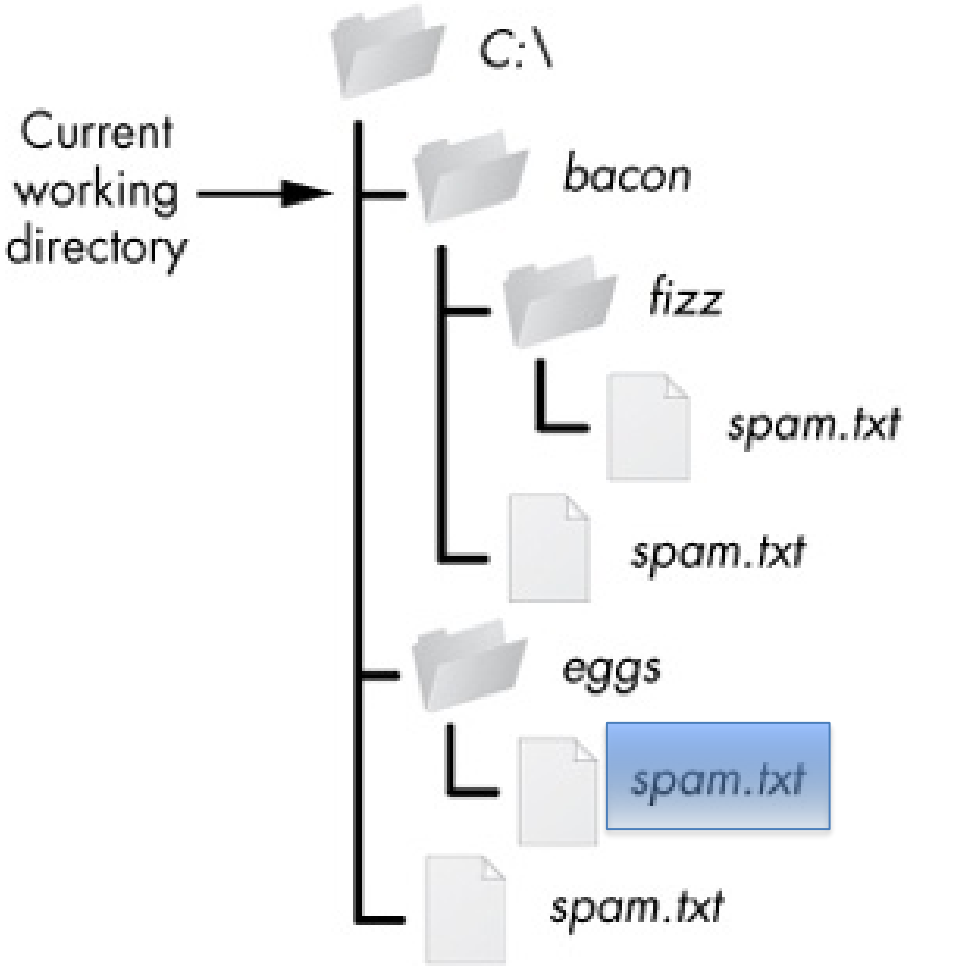
```
file =
open("C:\_jasthi\_Teacher_Master_D0_NOT_DELETE_TWO\_CS421_Python\sou
rce_files\quotes_dir\quotes_dummy.txt")
```

Oh boy! The path is so long…

It is also possible to specify the “relative” path. You will specify the path relative to the current directory

```
# relative path; Relative to the source code location
file = open("quotes_dir\quotes_dummy.txt")
print(file.read())

# relative path can be anywhere
# ./ = current directory
# ../ up one level
file = open(".\quotes.txt")
print(file.read())


# relative path can be anywhere
# ../ up one level
file = open("..\quotes.txt")
print(file.read())

# relative path can be anywhere
# ../ up two levels
file = open("..\..\quotes_dummy.txt")
print(file.read())
```

Play with relative and absolute paths



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

Open file: Then what?

- Why are we opening the file?
- Once the file is opened, we need to tell our intentions to the python.

- Open('marks.txt', 'x')
- This is called "mode".

"r" - **Read** - Default value. Opens a file for reading, error if the file does not exist

"a" - **Append** - Opens a file for appending, creates the file if it does not exist

"w" - **Write** - Opens a file for writing, creates the file if it does not exist

"x" - **Create** - Creates the specified file, returns an error if the file exists

opening files: syntax


```
file_obj = open("quotes.txt", "rt")
```

What is my intention here?

"r" - **Read** - Default

"a" - **Append**

"w" - **Write**

"x" - **Create**

"t" - **Text** – Default

"b" - **Binary**

opening files: syntax


```
open("photo.jpg", "rb")
```

What is my intention here?

"r" - **Read** - Default

"a" - **Append** (add at the end)

"w" - **Write** (overwrite)

"x" - **Create**

"t" - **Text** – Default

"b" - **Binary**

opening files: syntax


```
x = open("quotes.txt")
```

What is my intention here?

"r" - **Read** - Default

"a" - **Append**

"w" - **Write**

"x" - **Create**

"t" - **Text** – Default

"b" - **Binary**

Back to reading the files..

There are several methods to read the files.

```
file_obj = open("quotes.txt")
```

Method	Purpose
file_obj.read( )	Read the whole content in one go;
file_obj.read(n)	Read n characters from the file
file_obj.readline( )	Read one line at a time; This method reads the line at the current position (current file pointer) (analogy: where is the bookmark?)
file_obj.readlines( )	Read all the lines and returns the list of lines;
for x in file_obj: print(x)	Loop through the file contents

And it is a good idea to close the file once you are done with it.

```
file_obj.close()
```

Method	Purpose
file_obj.close( )	For closing the file

Keeping track of the status

There are several methods to read the files.

```
quotes_file = open("quotes.txt")
```

Method	What is read?	Return datatype?	Where is bookmark?
quotes_file.read( )	Read the whole content in one go;	String	Bookmark goes to the end of the file
quotes_file.read(n)	Read n characters from the file	String	Bookmark advances by n characters
quotes_file.readline( )	Read one line at a a time	String	Bookmark advances to the next line
quotes_file.readlines( )	Read all the lines	List of Strings (new line char is included)	Bookmark goes to the end of the file
for line in quotes_file: print(x)	Loop through the file contents	NA	With each iteration, the bookmark advances to the next line

And it is required to close the file once you are done with it.

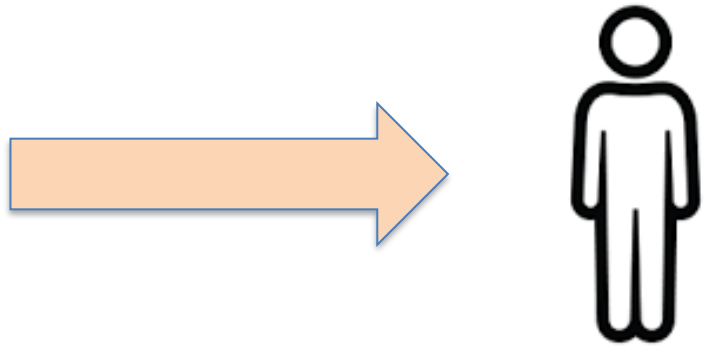
```
quotes_file.close()
```

Method	Purpose
quotes_file.close( )	For closing the file

File Name vs File Object

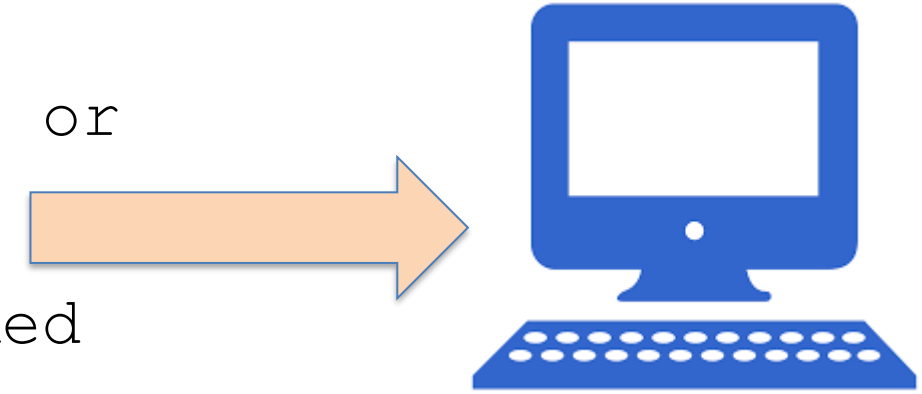
File Names (Strings):

- `C:\data\students_list.txt`
- `marks.txt`



File Objects (objects):

- `File_obj = open('marks.txt')`
- When you open the file for reading or writing, python creates a "file object".
- And all the operations are performed using this file_object

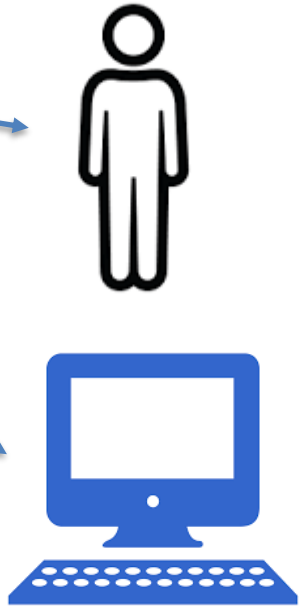


Reading the whole file as one string

Reading the whole file as one single string

```
1 #@title Reading the whole file as one single string
2
3 with open('quotes.txt') as quotes_file_obj:
4     all_text = quotes_file_obj.read()
5
6 print('read() method output: ', type(all_text))
7 print(all_text)
```

```
read() method output:  <class 'str'>


Love For All, Hatred For None. - Khalifatul Masih III
Change the world by being yourself. - Amy Poehler
Every moment is a fresh beginning. - T.S Eliot
Never regret anything that made you smile. - Mark Twain
Die with memories, not dreams. - Unknown
Aspire to inspire before we expire. - Unknown
Everything you can imagine is real. - Pablo Picasso
Simplicity is the ultimate sophistication. - Leonardo da Vinci
Whatever you do, do it well. - Walt Disney
What we think, we become. - Buddha
```

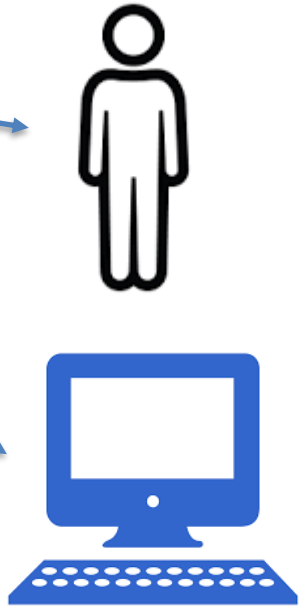


Reading the file one line at a time

Reading the file one line at a time

```
#@title Reading the file one line at a time
with open('quotes.txt') as quotes_file_obj:
  for each_line in quotes_file_obj:
    print(type(each_line))
    print(each_line)

```

```
<class 'str'>
Love For All, Hatred For None. - Khalifatul Masih III

<class 'str'>
Change the world by being yourself. - Amy Poehler

<class 'str'>
Every moment is a fresh beginning. - T.S Eliot
```



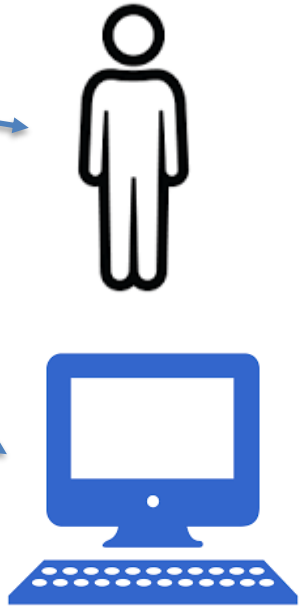
Reading all lines as a List

Reading the whole file as a list of strings

```
1 #@title Reading the whole file as a list of strings
2 with open('quotes.txt') as quotes_file_obj:
3     all_text = quotes_file_obj.readlines()
4
5 print('readlines() method output: ', type(all_text))
6
7 print("\nTraversing the list: ")
8 for each_line in all_text:
9   print(each_line)
```

wi

```
readlines() method output:  <class 'list'>

Traversing the list:
Love For All, Hatred For None. - Khalifatul Masih III

Change the world by being yourself. - Amy Poehler

Every moment is a fresh beginning. - T.S Eliot

Never regret anything that made you smile. - Mark Twain

Die with memories, not dreams. - Unknown
```



Watch out for the new line character


```
quotes_file = open("quotes.txt")
```

Let us open the ‘quotes.txt’ file and inspect it for a moment!

How does the file know to keep each quote on a separate line?

wi

At the end of each line, we have a special character (new line character) – represented as ‘\n’.

It is a hidden character. You can’t read it. But it is there.

Sometimes, you may need to strip the last character to get the exact string comparisons.

```
each_line = each_line[0:-1]
```

You can query the meta data


```
quotes_file = open("quotes.txt")
```

Once a file is opened, you have one *file* object (remember our “remote” in Class-Object),
You can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.

Writing to the files

Open the file for “appending”. Adding text at the end of the file.

```
quotes_file = open("quotes.txt", "a")
```

Open the file for “writing”. It will overwrite the entire content with the new text.

```
quotes_file = open("quotes.txt", "w")
```

Method	Purpose
quotes_file.write(input_str)	For writing the text (input_str) to the file. Based on the mode (either a or w), the text may be appended or the existing text is replaced with the next text.

And it is a good idea to close the file once you are done with it.

```
quotes_file.close()
```

Method	Purpose
quotes_file.close( )	For closing the file

Creating the files

Open the file for “writing”. If the file doesn’t exist, it will create one.

```
quotes_file = open("quotes.txt", "w")
```

You can also explicitly create a file.

```
quotes_file = open("quotes.txt", "x")
```

Deleting the files

To delete a file, you need to import os module. And call remove( ) method.

Module = a python file with a bunch of related functions

```
import os
os.remove("quotes.txt")
```

Deleting a directory

To delete a directory, you need to import os module. And call rmdir( ) method.

Note that you can only delete an empty directory.

Module = a python file with a bunch of related functions

```
import os
os.rmdir("dummy_dir")
```

Checking whether a file exists

To avoid getting an error, you might want to check if the file exists before opening or writing or deleting.

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Colab: Loading (Mounting) files from github

- `#@title Mount github repo to the colab`
- `!git clone https://github.com/sjasthi/python_input_files.git`

Colab: interactively loading files (GUI)

- `#@title Interactive upload of files`
- `from google.colab import files`
- `uploaded = files.upload()`

File Handling: Summary

Python provides simple and elegant ways to handle files.

Any operation you can do on files and directories can be performed through python.

Through File IO and String processing, many routine tasks can be automated and many useful reports can be generated.

Thank You.