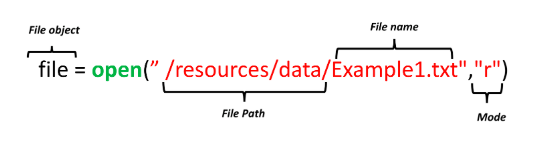
## **Reading Text Files**

One way to read or write a file in Python is to use the built-in open function. The open function provides a **File object** that contains the methods and attributes you need in order to read, save, and manipulate the file. In this notebook, we will only cover **.txt** files. The first parameter you need is the file path and the file name. An example is shown as follow:



The mode argument is optional and the default value is **r**. In this notebook we only cover two modes:

* **r** Read mode for reading files
* **w** Write mode for writing files

For the next example, we will use the text file **Example1.txt**. The file is shown as follow:

This is line 1

This is line 2

This is line 3

We read the file:

# Read the Example1.txt

example1 = "/resources/data/Example1.txt"

file1 = open(example1, "r")

We can view the attributes of the file.

The name of the file:

# Print the path of file

file1.name

The mode the file object is in:

# Print the mode of file, either 'r' or 'w'

file1.mode

We can read the file and assign it to a variable :

# Read the file

FileContent = file1.read()

FileContent

The **/n** means that there is a new line.

We can print the file:

# Print the file with '\n' as a new line

print(FileContent)

The file is of type string:

# Type of file content

type(FileContent)

We must close the file object:

# Close file after finish

file1.close()

## **A Better Way to Open a File**[**¶**](https://e-2629b3c96d.cognitiveclass.ai/lab#A-Better-Way-to-Open-a-File)

Using the with statement is better practice, it automatically closes the file even if the code encounters an exception. The code will run everything in the indent block then close the file object.

# Open file using with

with open(example1, "r") as file1:

FileContent = file1.read()

print(FileContent)

The file object is closed, you can verify it by running the following cell:

# Verify if the file is closed

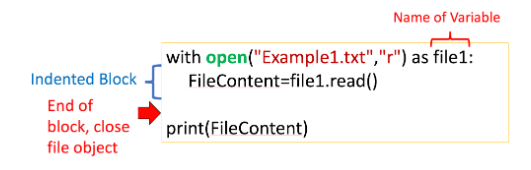
file1.closed

We can see the info in the file:

# See the content of file

print(FileContent)

The syntax is a little confusing as the file object is after the as statement. We also don’t explicitly close the file. Therefore we summarize the steps in a figure:



We don’t have to read the entire file, for example, we can read the first 4 characters by entering three as a parameter to the method **.read()**:

# Read first four characters

with open(example1, "r") as file1:

print(file1.read(4))

Once the method .read(4) is called the first 4 characters are called. If we call the method again, the next 4 characters are called. The output for the following cell will demonstrate the process for different inputs to the method read():

# Read certain amount of characters

with open(example1, "r") as file1:

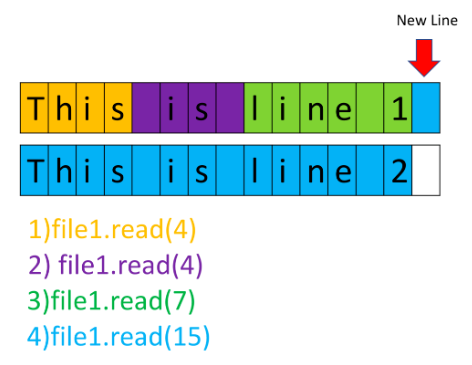
print(file1.read(4))

print(file1.read(4))

print(file1.read(7))

print(file1.read(15))

The process is illustrated in the below figure, and each color represents the part of the file read after the method read() is called:



Here is an example using the same file, but instead we read 16, 5, and then 9 characters at a time:

# Read certain amount of characters

with open(example1, "r") as file1:

print(file1.read(16))

print(file1.read(5))

print(file1.read(9))

We can also read one line of the file at a time using the method readline():

# Read one line

with open(example1, "r") as file1:

print("first line: " + file1.readline())

We can use a loop to iterate through each line:

# Iterate through the lines

with open(example1,"r") as file1:

i = 0;

for line in file1:

print("Iteration", str(i), ": ", line)

i = i + 1;

We can use the method readlines() to save the text file to a list:

# Read all lines and save as a list

with open(example1, "r") as file1:

FileasList = file1.readlines()

Each element of the list corresponds to a line of text:

# Print the first line

FileasList[0]

# Print the second line

FileasList[1]

# Print the third line

FileasList[2]