Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #45 & #46

Introduction to Files

File operations: Working with Text files(read and write)

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Verified by,
PCPS Team - 2023

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M(Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

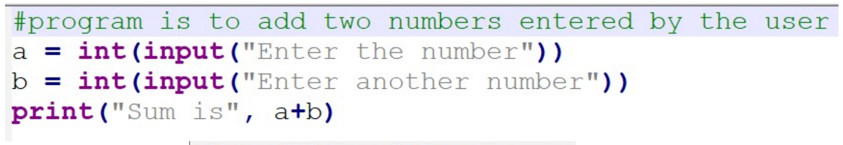
Introduction:

Various ways of giving data to the program are listed below:

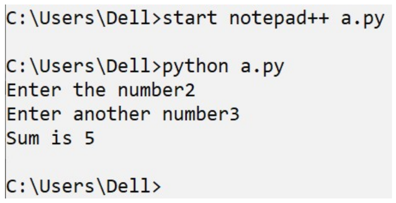
1. Interactive mode [Dialog between the program and the user]
2. Command Line Arguments processing
3. Usage of Files

1. Interactive mode: There are two ways in which you can take input from the user.

-> Using the input() function which helps to get the cursor and allows the user to types in. Input always returns str type.



```python
#program is to add two numbers entered by the user
a = int(input("Enter the number"))
b = int(input("Enter another number"))
print("Sum is", a+b)
```

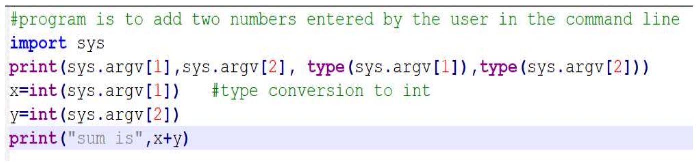


```
C:\Users\Dell>start notepad++ a.py

C:\Users\Dell>python a.py
Enter the number2
Enter another number3
Sum is 5

C:\Users\Dell>
```

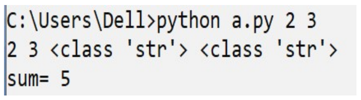
-> Methods available in sys.stdin object

2. Command Line Arguments:

While running the program itself, if we want to supply some (specific and small amount of data) data to the program, that can be done in command line itself. The arguments passed in the command line are nothing but command line arguments. These can be accessed in the program using sys.argv. Command line arguments are always of **str** type.



```python
#program is to add two numbers entered by the user in the command line
import sys
print(sys.argv[1],sys.argv[2], type(sys.argv[1]),type(sys.argv[2]))
x=int(sys.argv[1])   #type conversion to int
y=int(sys.argv[2])
print("sum is",x+y)
```



```
C:\Users\Dell>python a.py 2 3
2 3 <class 'str'> <class 'str'>
sum= 5
```

In both the cases above, the input given is very minimal. We may have to work with the application where the amount of data required may be large. In this case, it is always a good option to make use of files.

3. Usage of files

The variables we create in our program remain in memory until the end of the program. They are lost when the program terminates. We require files to make the information permanent. We call this feature persistence. The files persist not only the program termination, but also rebooting or restarting a computer.

The files belong to the operating system that runs the computer. The **naming convention depends on the operating system.**

In Microsoft Windows, the total file name (called the path) has the drive name, then ":", then \directory name – this could repeat number of times – then the filename.

In Linux, we have no drive name. The file name (path) has /(root directory) and then directory name – repeats number of times – then the filename.

To use a file, whose name depends on the operating system, we make an association of an entity called **file object** with the file. This **file object refers to some data structures outside of our program which in turn refers the file.** We call this entity a resource. **We get the resource when we want using a function called open(). We should release this whenever we do not require any more using a function called close()**

---> open() returns a file pointer or file descriptor or marker or handle associated with that file. It is of type TextIOWrapper. This reference is used to perform any operation on the file further. A file opening may not always succeed. If it fails, it throws an exception.

---> close() flushes any unwritten information and closes the file object, after which no more writing can be done. Python automatically closes a file when the reference is reassigned to another file.

Advantages of Files:

- Data is persistent even after the termination of the program.
- The data set can be much larger.
- The data can be input much more quickly and with less chance of error.

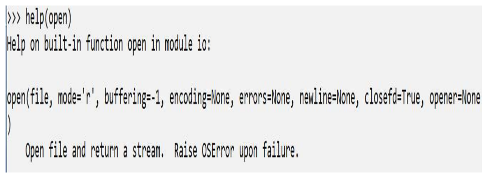
A file is some information or data which stays in the computer storage devices. Can be divided into two categories:

Text file: Stores data in the form of alphabets, digits and other special symbols by storing their ASCII values and are in a **human-readable format.**

Binary file: Contains a sequence or a collection of bytes which are **not in human readable format.**

In python, a file operation takes place in the following order: **Open a file - > Perform read or write ->Close the file.** There are many operations(read(), write()) which can be performed on a file**.** So the file should be **opened in appropriate modes(r – read only, w – write only, a – append only)**

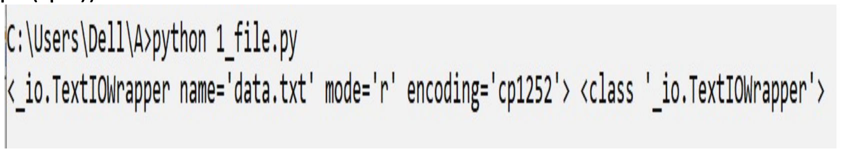
<u>open():</u> If the file to be opened in 'r' mode is not existing in that path specified in the first argument of open() function, returns an exception.



```
>>> help(open)
Help on built-in function open in module io:

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None
)
    Open file and return a stream.  Raise OSError upon failure.
```

Opening an existing file for reading

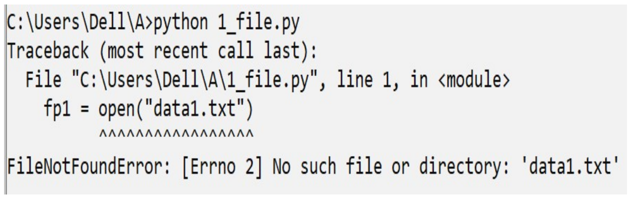
```
fp1 = open("data.txt")  # by default, the mode is read
print(fp1, type(fp1))
```



```
C:\Users\Dell\A>python 1_file.py
<_io.TextIOWrapper name='data.txt' mode='r' encoding='cp1252'> <class '_io.TextIOWrapper'>
```

Opening a non- existing file for reading

fp1 = open("data1.txt") # by default, the mode is read
print(fp1, type(fp1))



```
C:\Users\Dell\A>python 1_file.py
Traceback (most recent call last):
  File "C:\Users\Dell\A\1_file.py", line 1, in <module>
    fp1 = open("data1.txt")
          ^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'data1.txt'
```

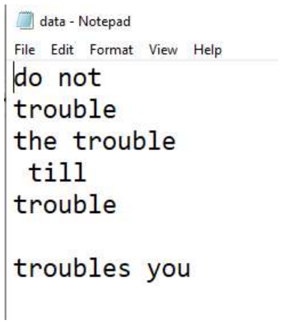
There are different ways to read a file:

read(): Returns the specified number of bytes from the file. Default is -1 which means the whole file. This function returns a str type.

readline(): Returns a line at a time from the file. This function returns a str type. Must be used inside a loop to read the whole file line by line.

readlines(): Reads the whole file into a list, where each element of the list is an individual line of the file. This function returns a list type.

Let us create a data.txt with the below data and save it manually.

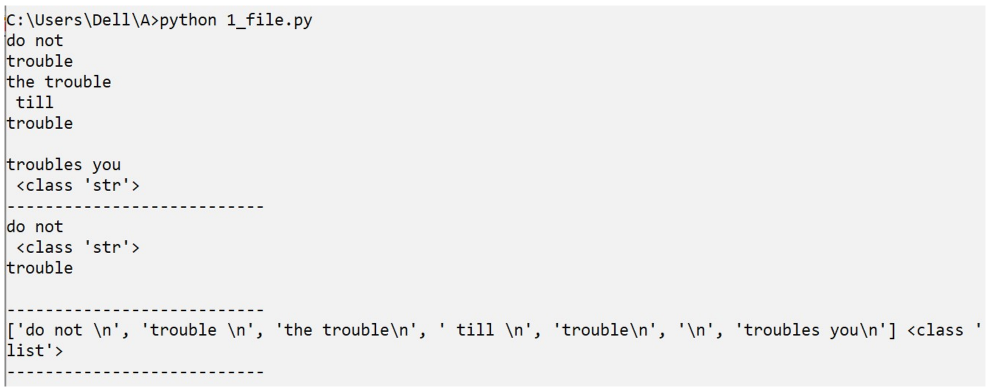


```
do not
trouble
the trouble
 till
trouble

troubles you
```

First, let's read the above file using all three functions above. Observe the outputs.

fp1 = open("data.txt")
all_data = fp1.read()
print(all_data, type(all_data))
fp1.close()
print("---------------------------")

```
fp2 = open("data.txt","r")  #optional r mode
line = fp2.readline()
print(line, type(line))  #prints only first line and its type
line = fp2.readline()  #reads again the next line
print(line)
fp2.close()
print("---------------------------")

fp3 = open("data.txt", "r")
lines = fp3.readlines()
print(lines,type(lines))
print("---------------------------")
fp3.close()
```



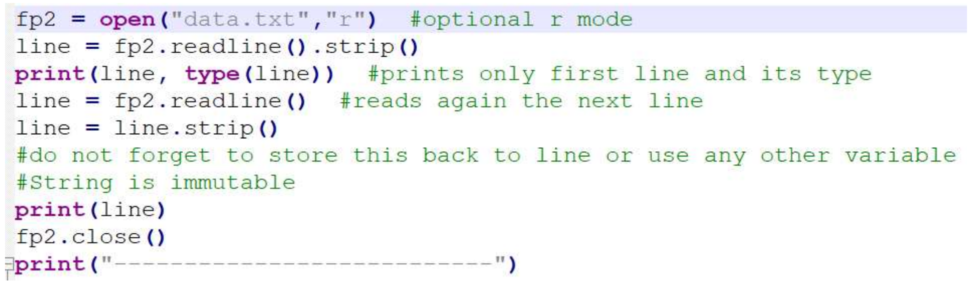
```
C:\Users\Dell\A>python 1_file.py
do not
trouble
the trouble
 till
trouble

troubles you
 <class 'str'>
---------------------------
do not
 <class 'str'>
trouble

---------------------------
['do not \n', 'trouble \n', 'the trouble\n', ' till \n', 'trouble\n', '\n', 'troubles you\n'] <class '
list'>
---------------------------
```

Few Observations and clarifications:

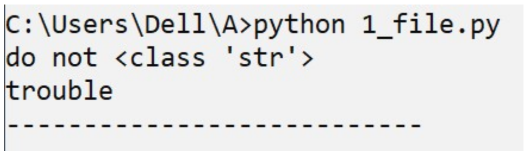
- You have read the file using readline function and stored it in a variable. Printing this variable and type(line) is done in the print function. Why the output is in two lines?
  Clarification: readline() reads the entire line along with a newline(\n) character and this whole string is stored in a variable. So output will have \n and then it prints the type as <class 'str'>. Remember the default value for each print is \n. So at the end of print function, one new line is mandatory unless you have changed the value for end keyword argument.

- How do you avoid this in every line?

  Clarification: Use strip()



```
fp2 = open("data.txt","r")  #optional r mode
line = fp2.readline().strip()
print(line, type(line))  #prints only first line and its type
line = fp2.readline()  #reads again the next line
line = line.strip()
#do not forget to store this back to line or use any other variable
#String is immutable
print(line)
fp2.close()
print("---------------------------")
```

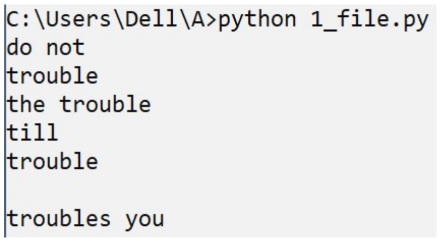


```
C:\Users\Dell\A>python 1_file.py
do not <class 'str'>
trouble
----------------------------
```

- How do you print each line of file on the terminal?

  Clarification: Use the iterable lines variable in a for loop

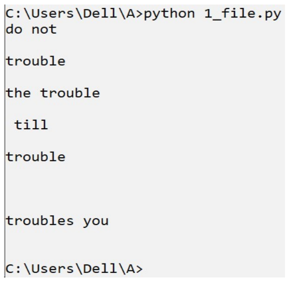
```
fp3 = open("data.txt", "r")
lines = fp3.readlines() #lines is list
for line in lines:  #iterable
   print(line.strip())
fp3.close()
```



```
C:\Users\Dell\A>python 1_file.py
do not
trouble
the trouble
till
trouble

troubles you
```

- If you replace line.strip() with line in the above code, what happens? Reason?

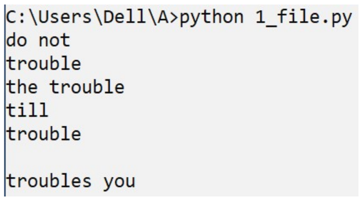
  Output is shown below.



```
C:\Users\Dell\A>python 1_file.py
do not

trouble

the trouble

 till

trouble


troubles you

C:\Users\Dell\A>
```

- Can you use the file object/file reference/file object directly in the for loop?

```
fp1 = open("data.txt", "r")
for i in fp1:   #fp1 is iterable by default
        print(i.strip())
fp1.close()
```

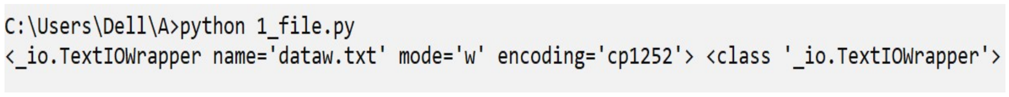


```
C:\Users\Dell\A>python 1_file.py
do not
trouble
the trouble
till
trouble

troubles you
```

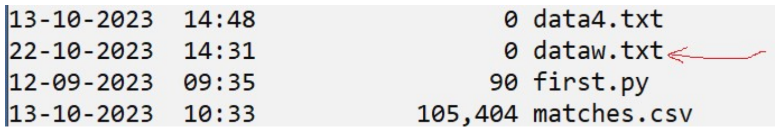
<u>open():</u> If the file to be opened in 'w' mode is not existing in that path specified in the first argument of open() function, file will be created in that path. If the file is already existing, contents are over written.

Opening a non-existing file for writing

```
fp1 = open("dataw.txt","w")
print(fp1, type(fp1))
```



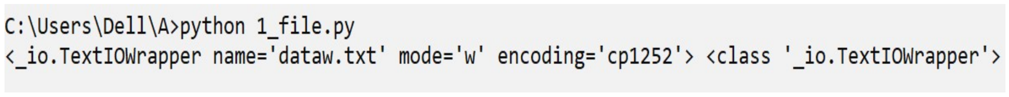
```
C:\Users\Dell\A>python 1_file.py
<_io.TextIOWrapper name='dataw.txt' mode='w' encoding='cp1252'> <class '_io.TextIOWrapper'>
```



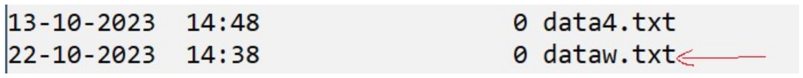
```
13-10-2023  14:48                 0 data4.txt
22-10-2023  14:31                 0 dataw.txt
12-09-2023  09:35                90 first.py
13-10-2023  10:33           105,404 matches.csv
```

Opening an existing file for writing

```
#open the file created above and add some contents to it and then run below code.
#check the time and 0 bytes to understand the w mode.
fp1 = open("dataw.txt","w")
print(fp1, type(fp1))
```



```
C:\Users\Dell\A>python 1_file.py
<_io.TextIOWrapper name='dataw.txt' mode='w' encoding='cp1252'> <class '_io.TextIOWrapper'>
```



```
13-10-2023  14:48                 0 data4.txt
22-10-2023  14:38                 0 dataw.txt
```

There are different ways to write to a file:

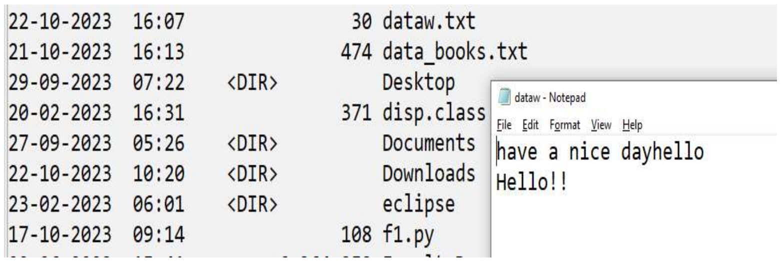
write(st): Writes a string st into an open file. It returns nothing.

print(arguments, file = file_object): Writes arguments to file_object rather than to the terminal

Let us create a file called dataw.txt and write to that file.

```
fp=open("dataw.txt","w")            #write only mode: old contents are lost
print("file opened")
fp.write("have a nice day"  )       #no implicit \n
```

```
print("hello",file = fp)     # hello will be written to the file along with \n due to end = "\n"
fp.write("Hello!! ")
#d=fp.read()                           #UnsupportedOperation
fp.close()
```



```
22-10-2023  16:07              30 dataw.txt
21-10-2023  16:13             474 data_books.txt
29-09-2023  07:22    <DIR>        Desktop
20-02-2023  16:31             371 disp.class
27-09-2023  05:26    <DIR>        Documents
22-10-2023  10:20    <DIR>        Downloads
23-02-2023  06:01    <DIR>        eclipse
17-10-2023  09:14             108 f1.py
```

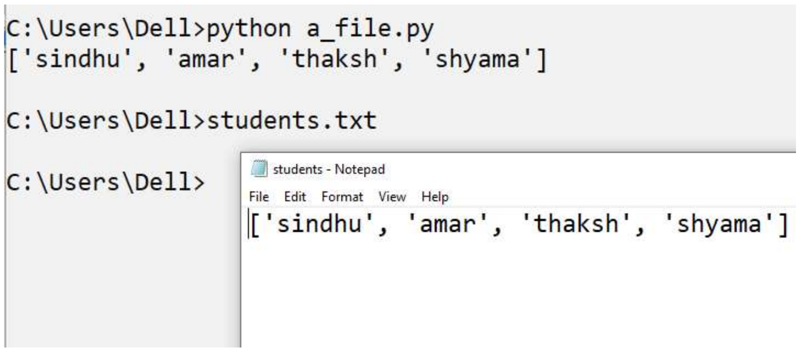
```
have a nice dayhello
Hello!!
```

Few observations and clarifications

- If you want to write a list having student names to students.txt file, how do you write it and how do you read it?

  Clarification:

  ```
  fp = open("students.txt","w")
  names = ["sindhu", "amar", "thaksh","shyama"]
  #fp.write(names)        #Returns an exception          . write expects only str type as an argument
  fp.write(str(names))
  #print(names, file = fp)  #valid as print can print anything
  fp.close()
  fp = open("students.txt")
  print(fp.read())
  fp.close()
  ```



```
C:\Users\Dell>python a_file.py
['sindhu', 'amar', 'thaksh', 'shyama']

C:\Users\Dell>students.txt

C:\Users\Dell>
```

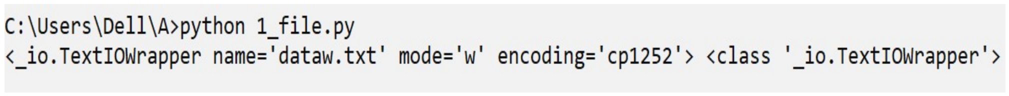
```
['sindhu', 'amar', 'thaksh', 'shyama']
```

- Can you use writelines() function in python to write few strings to a file line by line?

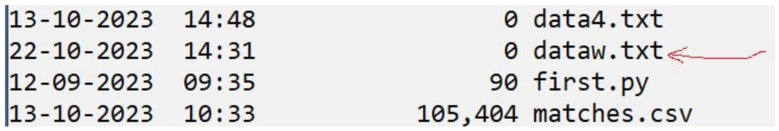
<u>open():</u> If the file to be opened in 'a' mode is not existing in that path specified in the first argument of open() function, file will be created in that path. If the file is already existing, contents are retained if any.

Opening a non-existing file for appending data

```
fp1 = open("dataw.txt","a")
print(fp1, type(fp1))
```



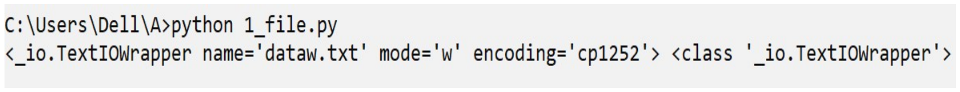
```
C:\Users\Dell\A>python 1_file.py
<_io.TextIOWrapper name='dataw.txt' mode='w' encoding='cp1252'> <class '_io.TextIOWrapper'>
```



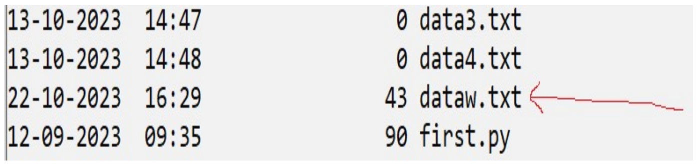
```
13-10-2023  14:48              0 data4.txt
22-10-2023  14:31              0 dataw.txt
12-09-2023  09:35             90 first.py
13-10-2023  10:33        105,404 matches.csv
```

Opening an existing file for appending data

```
#open the file created above and add some contents to it and then run below code.
#check the time and 43 bytes to understand the 'a' mode.
fp1 = open("dataw.txt","a" )
print(fp1, type(fp1))
```



```
C:\Users\Dell\A>python 1_file.py
<_io.TextIOWrapper name='dataw.txt' mode='w' encoding='cp1252'> <class '_io.TextIOWrapper'>
```

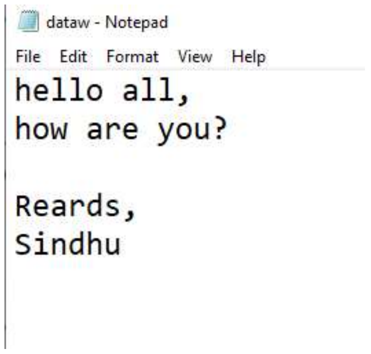


```
13-10-2023  14:47              0 data3.txt
13-10-2023  14:48              0 data4.txt
22-10-2023  16:29             43 dataw.txt
12-09-2023  09:35             90 first.py
```

Now let us try to perform write operation on the file opened in append mode.

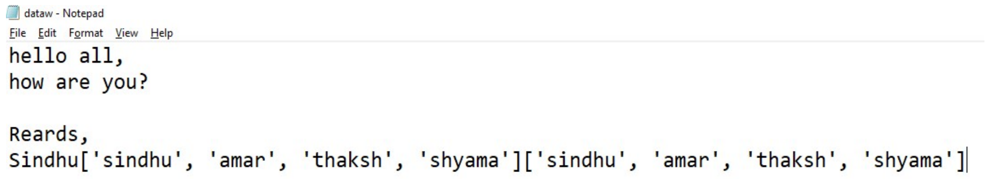
```
fp = open("dataw.txt","a")
names = ["sindhu", "amar", "thaksh","shyama"]
#fp.write(names)      #Returns an exception. write expects only str type as an argument
fp.write(str(names))
#print(names, file = fp)  #valid as print can print anything
fp.close()
fp = open("dataw.txt")
print(fp.read())
 fp.close()
```

Current content of dataw.txt is as below.

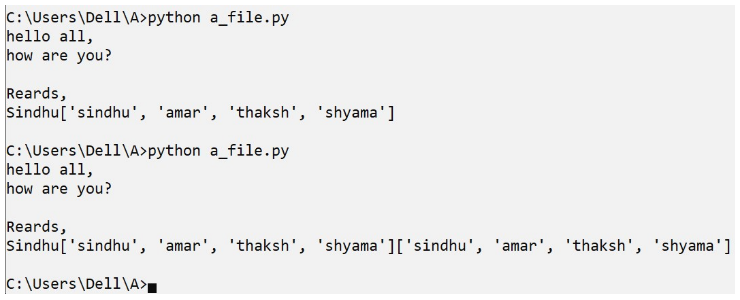


```
hello all,
how are you?

Reards,
Sindhu
```

After running the code twice, observe the content of dataw.txt and observe the output.



```
hello all,
how are you?

Reards,
Sindhu['sindhu', 'amar', 'thaksh', 'shyama']['sindhu', 'amar', 'thaksh', 'shyama']
```



```
C:\Users\Dell\A>python a_file.py
hello all,
how are you?

Reards,
Sindhu['sindhu', 'amar', 'thaksh', 'shyama']

C:\Users\Dell\A>python a_file.py
hello all,
how are you?

Reards,
Sindhu['sindhu', 'amar', 'thaksh', 'shyama']['sindhu', 'amar', 'thaksh', 'shyama']

C:\Users\Dell\A>
```

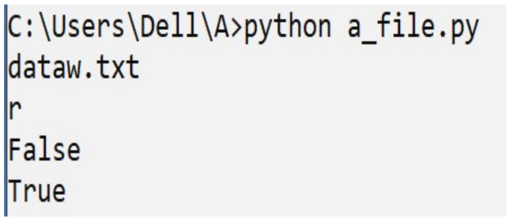
Point to think!

- **If print statement is uncommented and write function is commented, what would have been the output of above code?**

File Attributes

Properties of a file that provides additional information about the file. Examples of file attributes include file name and the mode in which the file is opened. Also includes the boolean attribute - closed to check whether file is closed or not.

```
fp = open("dataw.txt")
print(fp.name)
print(fp.mode)
print(fp.closed)
fp.close()
print(fp.closed)
```



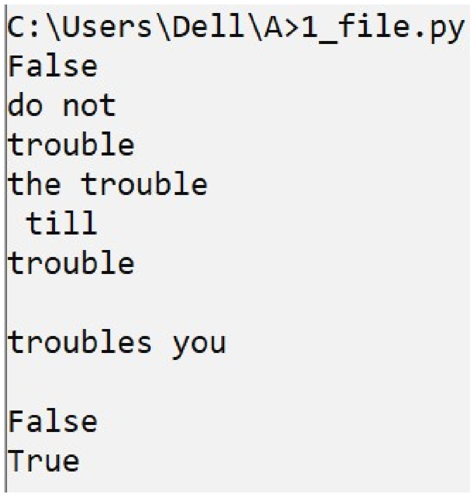
```
C:\Users\Dell\A>python a_file.py
dataw.txt
r
False
True
```

Usage of with keyword

The with statement replaces a try-catch block with a concise shorthand. More importantly, it **ensures closing resources right after processing them**.

A common example of using the with statement is reading or writing to a file and closing the file automatically.

```
with open("data.txt") as f1:
    all_data = f1.read()
    print(f1.closed)
    print(all_data)
    print(f1.closed)
print(f1.closed)
```



```
C:\Users\Dell\A>1_file.py
False
do not
trouble
the trouble
 till
trouble

troubles you

False
True
```

- END -