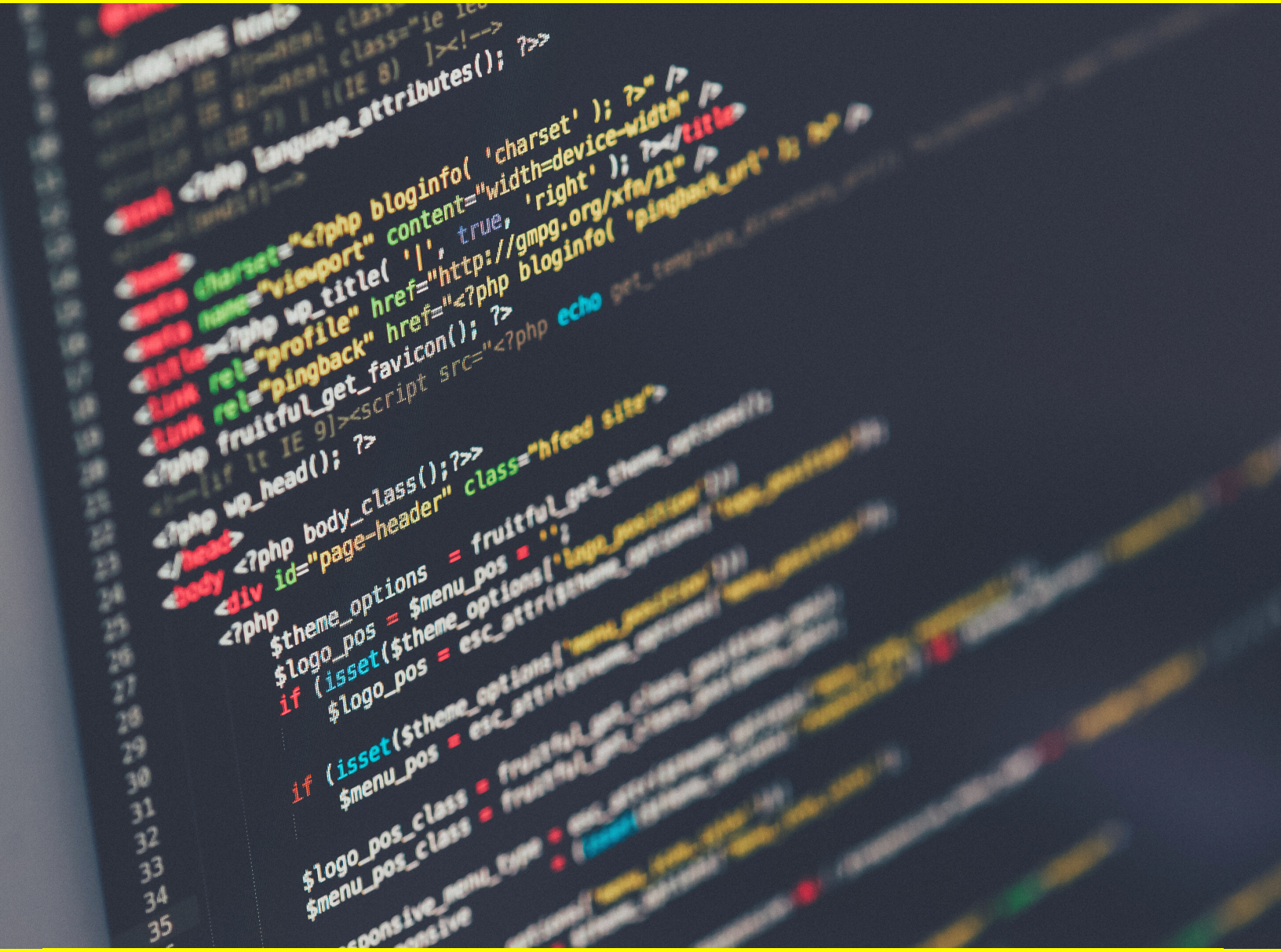
the fast lane to python

PART 5



BY:

Arshadul Shaikh

Ref taken from Coursera + Udemy

WOA WOA WOA!!!!!!
Look who is here, the person who is sleeping, eating, dreaming Python all day.

I congratulate you again, that you are consistent and made it till 5th part, which means as I have said in the earlier books you are dedicated to become one **"in demand python developers"** companies are searching for. Though we still have a long way to go, stay with me and we shall accomplish it together :)

If you have directly started with part 5, i would recommend to go through part 1 to part 4 as well, won't take much of your time to get you on board from where we are going to start.

Lot said, let's roll:

Working with Data Files

if you have reached here after reading all the previous versions, you would know that we have either coded right into the program or used data fed by the user, which is the learning steps of entering into the real world of Python

The **images** we work in our daily life be it on social media, documents or your pictures directory, they are ultimately **existing** on **some files** on your hard drive. Web page or some server from where you are retrieving the data.

Word processing documents, and music are also some examples of data that live in files.

The Python programs that you write are stored as text files.

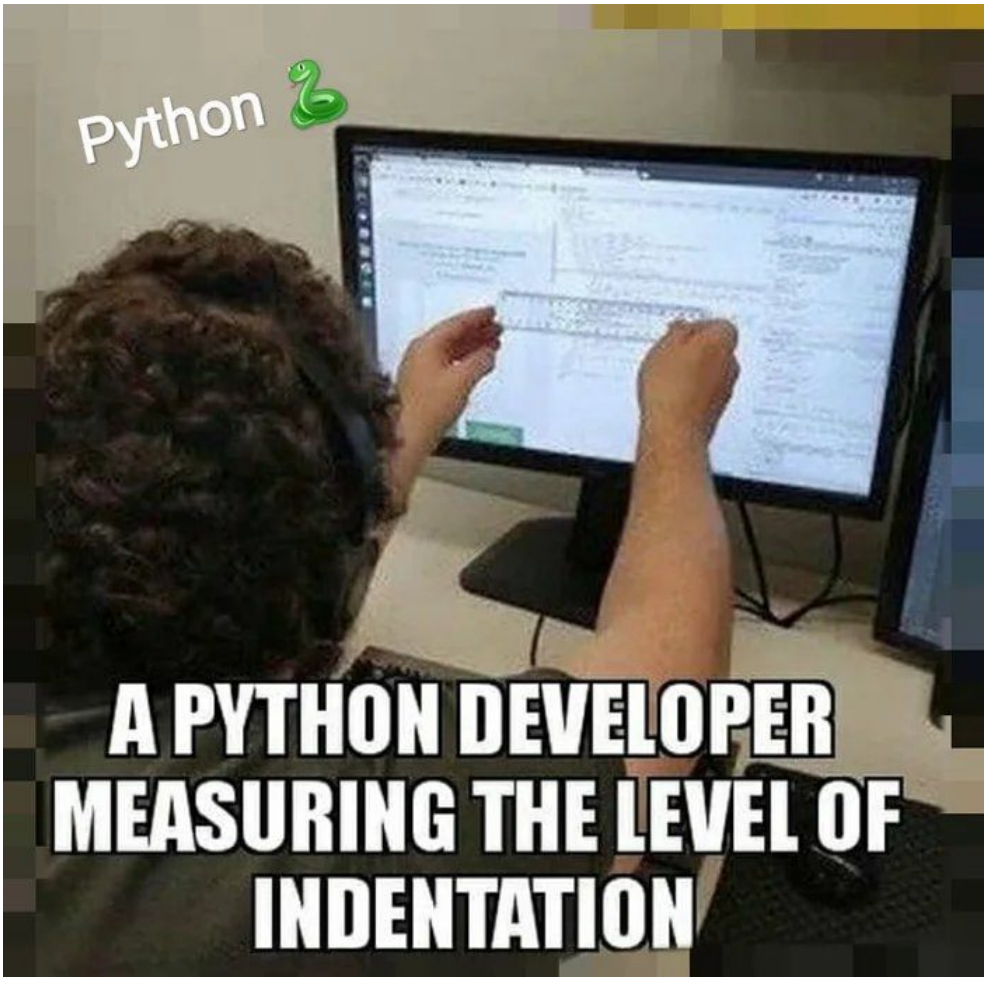
We can create files in so many ways, For example, we could use a text editor to type in and save the data. We could download the data from a website and then save it in a file.

Regardless of how the file is created, Python will allow us to manipulate the contents.

Question is hoW?

- open the file by Python built in methods.
- once a file is opened it becomes a Python object just like all other data.
- We can write/append/delete same as what we did so far with other data (obviously the commands/methods would be different) in python.
- And then close them when we are done

Method Name	Use	Explanation
`open`	`open(filename,'r')`	Open a file called filename and use it for reading. This will return a reference to a file object.
`open`	`open(filename,'w')`	Open a file called filename and use it for writing. This will also return a reference to a file object.
`close`	`filevariable.close()`	File use is complete.



As we start writing big programs indentation will be a word you will see as a monster in your nightmares :D

Reading a file

What is a file?

Files is just a collection of data saved on a hard disk or other storage that persists over time. A file has a name, and files can be organized into folders or directories. We'll be working with text files as opposed to images, or sounds, or videos.

let's try to see how can we open a file:

```
1  file_name=open("C:\Japan_olympics_2020.txt","r")
2  #we will be writing more code here when we try to understand
3  #more methods/modules/topics relate to file
4  file_name.close()
5
```

- we have used the open function to open the file.
- The variable, **file_name**, now **holds a reference** to the **file object** returned by open.
- When we are finished with the file, we can close it by using the close method as you see above.
- After the file is closed any further attempts to use file_name will result in an error.

Now as we know opening a file is that easy , You definitely would want to jump in and start manipulating the files.

Let's try to learn more file reading Methods.

File Reading Methods

We saw above how the moment we open the file, the variable became a reference to the Object of the file. Once you have it, Python provides three methods to read data from that object.

read()
The read() method returns the entire contents of the file as a single string

(or just some characters if you provide a number as an input parameter.

readlines
The readlines method returns the entire contents of the entire file as a list of strings, where each item in the list is one line of the file.

readline
The readline method reads one line from the file and returns it as a string.

*****KEY POINT:**
The strings returned by readlines or readline will contain the newline character at the end.

Method Name	Use	Explanation
`write`	`filevar.write(astring)`	Add a string to the end of the file. `filevar` must refer to a file that has been opened for writing.
`read(n)`	`filevar.read()`	Read and return a string of `n` characters, or the entire file as a single string if `n` is not provided.
`readline(n)`	`filevar.readline()`	Read and return the next line of the file with all text up to and including the newline character. If `n` is provided as a parameter, then only `n` characters will be returned if the line is longer than `n`. **Note** the parameter `n` is not supported in the browser version of Python, and in fact is rarely used in practice, you can safely ignore it.
`readlines(n)`	`filevar.readlines()`	Returns a list of strings, each representing a single line of the file. If `n` is not provided then all lines of the file are returned. If `n` is provided then `n` characters are read but `n` is rounded up so that an entire line is returned. **Note** Like `readline` `readlines` ignores the parameter `n` in the browser.

- we will generally either iterate through the lines returned by readlines() with a for loop, or use read() to get all of the contents as a single string.

- If you are coming from some other programming language background, the convenient method for going through the lines of a file one by one, **while is the favorite one.**

- Fortunately, in python we don't need to learn while for now, though we will get to it later

*****KEY POINT:**
Common error that novice programmers make is not realizing that all these ways of reading the file contents, "use up" the file.
After you call readlines() once, if you call it again you'll get an empty list.

Let's practice with some examples:

Sample File(create this file if you want as a sample):

This summer I will be travelling.
I will go to...
Italy: Rome
Greece: Athens
England: London, Manchester
France: Paris, Nice, Lyon
Spain: Madrid, Barcelona, Granada
Austria: Vienna
I will probably not even want to come back!
However, I wonder how I will get by with all the different languages.
I only know English!

Question 1:

Using the above saved file or any random file, find the number of characters in the file and assign that value to the variable num_char.

Question 2:

Find the number of lines in the file using same file, and assign it to a variable

Question 3:

Create a string called first_forty that is comprised of the first 40 characters from the file

Answers:

```
file_n=open("C:\Japan_olympics_2020.txt","r")
num_char=len(file_n.read())
print(num_char)
file_n.close()
```

```
file_n=open("C:\Japan_olympics_2020.txt","r")
num_lines=len(file_n.readlines())
print(num_lines)
file_n.close()
```

```
file_n=open("C:\Japan_olympics_2020.txt","r")
first_forty=file_n.read(40)
print(first_forty)
file_n.close()
```

Iterating over lines in a file

We are still fresh with lists like ["python", "java", "cpp", "javascript"] wherein in each example we tried to prove how python stood best :D

Well now it's time to do some simple yet complex operations on file, where we will take file as input in a program that will do some data processing.
In the program, we will examine each line of the file and print it with some additional text. Because readlines() returns a list of lines of text, we can use the for loop to iterate through each line of the file.

- A **line of a file** is defined to be a **sequence of characters up to** and including a special character **called the newline character (\n).**
- If you evaluate a string that contains a newline character you will see the character represented as \n.

- people coming from any C/C++ , have seen sometimes problems with this new line character **\n ,** When **you're reading or writing text mode files**, or to **stdin/stdout**, you must be using **\n**, and the interpreted handled the translation for us.
    - eg:

```
#include <stdio.h>
// function main begins program execution
int main( void )
    {
        printf( "Welcome to C! \n" );
    } // end function main
```

O/P: Welcome to C!

- Handling of newline is comparatively easy and we don't have to bother about \n while typing but If **you print a string** that **contains** a **newline** you will **not see the \n**, you will **just see its effects** (a carriage return).
- As the for **loop iterates** through each line of the file the **loop variable** will contain the **current line** of the file as a string of characters. The general pattern for processing each line of a text file is as follows:

```
for line in myFile.readlines():
    statement1
    statement2
    ...
```

(remember indentation is the key)

try to create a dummy file with below data for future example ref:

```
Name,Sex,Age,Team,Event,Medal
A Dijiang,M,24,China,Basketball,NA
A Lamusi,M,23,China,Judo,NA
Gunnar Nielsen Aaby,M,24,Denmark,Football,NA
Edgar Lindenau Aabye,M,34,Denmark/Sweden,Tug-Of-War,Gold
Christine Jacoba Aaftink,F,21,Netherlands,Speed Skating,NA
Christine Jacoba Aaftink,F,21,Netherlands,Speed Skating,NA
Christine Jacoba Aaftink,F,25,Netherlands,Speed Skating,NA
Christine Jacoba Aaftink,F,25,Netherlands,Speed Skating,NA
Christine Jacoba Aaftink,F,27,Netherlands,Speed Skating,NA
Christine Jacoba Aaftink,F,27,Netherlands,Speed Skating,NA
Per Knut Aaland,M,31,United States,Cross Country Skiing,NA
Per Knut Aaland,M,31,United States,Cross Country Skiing,NA
Per Knut Aaland,M,31,United States,Cross Country Skiing,NA
Per Knut Aaland,M,31,United States,Cross Country Skiing,NA
Per Knut Aaland,M,33,United States,Cross Country Skiing,NA
Per Knut Aaland,M,33,United States,Cross Country Skiing,NA
Per Knut Aaland,M,33,United States,Cross Country Skiing,NA
Per Knut Aaland,M,33,United States,Cross Country Skiing,NA
John Aalberg,M,31,United States,Cross Country Skiing,NA
John Aalberg,M,31,United States,Cross Country Skiing,NA
John Aalberg,M,31,United States,Cross Country Skiing,NA
John Aalberg,M,31,United States,Cross Country Skiing,NA
```

```
olypmicsfile = open("C:\Users\Japan_olympics_2020.txt", "r") #remember if you have a bigger path
                                    # like C:\Users\Desktop\Japan_olympics_2020.txt
                                    #then make sure you are putting double back slash
                                    #for the directories
                                    #or you will get an error

```

```
  File "<ipython-input-18-80455c209ed3>", line 1
    olypmicsfile = open("C:\Users\Japan_olympics_2020.txt", "r") #remember if you have a bigger path
                        ^
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX escape
```

```
file_name = open("C:\Japan_olympics_2020.txt", "r") #remember if you have a bigger path
                                        # like C:\Users\Desktop\Japan_olympics_2020.txt
                                        #then make sure you are putting double back slash
                                        #for the directories
                                        #or you will get an error
for lines in file_name.readlines():  #we are using readlines cause we have to perform certain
                                    #actions over file which is easy if we can convert the
                                    # file into list as we are master of list/tuple/strings now

    print(lines)                    #this will print each line one after the other
                                     #but also with a blank line , why?

    values = lines.split(",")       #this will convert the string into list
    print(values)
    print(values[0], "is from", values[3], "and is on the roster for", values[4])

olypmicsfile.close()
```

```
Name,Sex,Age,Team,Event,Medal

['Name', 'Sex', 'Age', 'Team', 'Event', 'Medal\n']
Name is from Team and is on the roster for Event
A Dijiang,M,24,China,Basketball,NA

['A Dijiang', 'M', '24', 'China', 'Basketball', 'NA\n']
A Dijiang is from China and is on the roster for Basketball
A Lamusi,M,23,China,Judo,NA
```

- To make our code simpler, and to allow for more efficient processing, we will come across a built-in way to iterate through the contents of a file one line at a time, without first reading it all into a list.

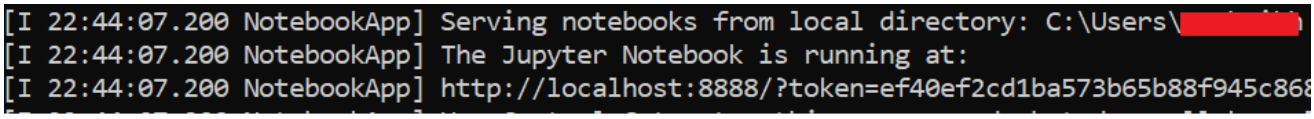
- But sometimes it's confusing , so we would keep it for future once we are well aware about all the terminologies of file related methods/modules

Finding File in your Filesystem

We have given in above examples entire path where the file is located and then read the file. I could also open and read the file directly without using path like below:

```
filename=open("Japan_olympics_2020.txt",'r')
filename.close()
```

Provided the file was present from where i am running the jupyter notebook which is C:\Users\username_directory



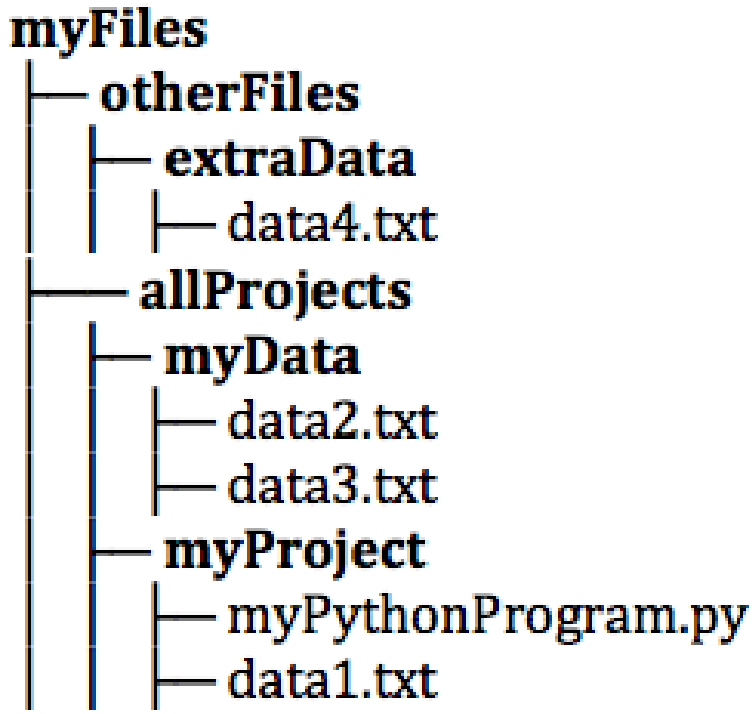
Let's try to understand how file reading and writing operations work in a better way.

Computer operating systems like Windows , Mac OS organize files into a hierarchy of folders, with some folders containing other folders.

For Linux distributions same is done in the form of directories.

If your file and your Python program are in different directories, we know we have to specify the path. You can think of the filename as the short name for a file, and the path as the full name. Typically, you will specify a relative file path, which says where to find the file to open, relative to the directory where the code is running from.

People who have worked on any version of **Unix/linux** OS knows the relevance of **"../" changing** the **directory**.



For example, consider the above image as the structure of your file system the python code file **myPythonProgram.py** could contain a line of code **open('../myData/data2.txt', 'r')**

The **../** means to go up one level in the directory structure, to the containing folder (allProjects); **myData/** says to **descend** into the **myData subfolder**. There is also an option to use an absolute file path.

For example, suppose the file structure in the figure is stored on a computer in the user's home directory, **/Users/user_name/myFiles**. Then code in any Python program running from any file folder could open **data2.txt** via **open('/Users/user_name/myFiles/allProjects/myData/data2.txt', 'r').**

You can tell an absolute file path because it begins with a **"/"**. If you will ever move your programs and data to another computer (e.g., to share them with someone else), it will be much more convenient if your use relative file paths rather than absolute. That way, if you preserve the folder structure when moving everything, you won't need to change your code.

If you use **absolute paths**, then the person you are sharing with probably not have the same home directory name, /Users/user_name/. Note that Python path_names follow the UNIX conventions (Mac OS is a UNIX variant), rather than the Windows file pathnames that use **: and **.

The Python interpreter will translate to Windows path_names when running on a Windows machine; you should be able to share your Python program between a Windows machine and a MAC without having to rewrite the file open commands.

*****SURPRISE*****
Advanced Topic Bonus

As we have been learning so good about all python concepts, let's try to get a pinch of one of the widely used advanced command along with file operations.

We have solved many problems where we have to follow 3 steps:

- Opening of file
  file_n=open("C:\Japan_olympics_2020.txt","r")
- operations to be performed on file
  num_char=len(file_n.read())
  print(num_char)
- close
  file_n.close()

But while we write code, as per my experience we remember to open the file but forget to close the file many a times, which leads to unexpected leakage of data.

And when leakage happens, while you are writing a program that may run for days or weeks at a time and does a lot of file reading and writing you may surely run into some big troubles.

So how do you get rid of this problem?

Python has the notion of a context manager that automates the process of doing common operations:
https://docs.python.org/3/library/contextlib.html

This module provides utilities for common tasks involving
the with statement. For reading and writing a file, the normal operation is to open the file and assign it to a variable. At the end of working with a file the common operation is to make sure that file is closed.

We can import the mode either :
from contextlib import contextmanager
or
import contextmanager
or
from contextlib import *

The Python with statement makes using context managers easy. The general form of a with statement is:

```
with <create some object that understands context> as <some name>:
   do some stuff with the object
   ...
```

```
import contextlib
with open('Japan_olympics_2020.txt','r') as with_context:
   for line in with_context:
      print(line)
```

- The first line of the with statement opens the file and assigns it to the variable with_context.
- Then we can iterate over the file in any of the usual ways. When we are done we simply stop indenting and let Python take care of closing the file and cleaning up.
- The final line print(with_context)

This is equivalent to code that specifically closes the file at the end, but neatly marks the set of code that can make use of the open file as an indented block, and ensures that the programmer won't forget to include the .close() invocation.

The default way of writing the same code is:

```
without_context = open('Japan_olympics_2020.txt', 'r')
for line in without_context:
   print(line)
without_context.close()
```

So to recall all the concepts we have learned so far regarding files:

1. Open the file using with and open.

2. Use .readlines() to get a list of the lines of text in the file.

3. Use a for loop to iterate through the strings in the list, each being one line from the file. On each iteration, process that line of text

4. When you are done extracting data from the file, continue writing your code outside of the indentation. Using with will automatically close the file once the program exits the with block.

Write on files the python way

I hope you were not planning to use python file operations just to read the files like we read a novel before sleep .

What is the most commonly performed operation over a file?

- read data from a file
- manipulate it in some way
- write the resulting data out to a new data file to be used for other purposes later.
- close the file.

To accomplish these, we know for read we have open function with argument 'r' but what to do when you need to write in that file?
For writing we use argument 'w' flag instead of the 'r' flag as the second parameter.
When we open a file for writing, a new, empty file with that name is created and made ready to accept our data. If an existing file has the same name, its contents are overwritten. As before, the function returns a reference to the new file object.

`write`	`filevar.write(astring)`	Add a string to the end of the file. `filevar` must refer to a file that has been opened for writing.

We have already seen the above snippet before how we can write a string to a file. The write method allows us to add data to a text file. Recall that text files contain sequences of characters. We usually think of these character sequences as being the lines of the file where each line ends with the newline \n character.

***KEY POINT**

Be very careful to notice that the write method takes one parameter, a string same as we practiced using input() function.
When invoked, the characters of the string will be added to the end of the file. This means that it is the programmer's job to include the newline characters as part of the string if desired.

choose any program you want and try to write the output in a file.

I am choosing a simple program of finding prime numbers between 2 limits:

```
start =input('what is the starting number you want?')
start=int(start)
end =input('what is the ending point of number?')
end=int(end)

for i in range(start,end):
    for j in range(2,i):
        if(i % j==0):
           # print(i)
            break
    else:
        print(i)
```

```
what is the starting number you want?11
what is the ending point of number?23
11
13
17
19
```

You all would understand the above small program, we are running it and seeing the expected output on the screen.

But how do we do the same thing but get all the output in a file which maybe would be used for next program?

- To start, we need to open a new output file by calling the open function, outfile = open("prime_number.txt",'w'), using the 'w' flag. (remember it will create a new file prime_number.txt if does not exist or will overwrite if already present on the path from where you are running your python IDE

- Once the file has been created, we just need to call the write method passing the string that we wish to add to the file. In this case, the string is already being printed so we will just change the print into a call to the write method. However, there is an **additional step** to take, **since the write method can only accept a string as input**.

- We'll need to convert the number to a string. Then, we just need to add one extra character to the string. The **newline character** needs to be **concatenated** to the end of the line.

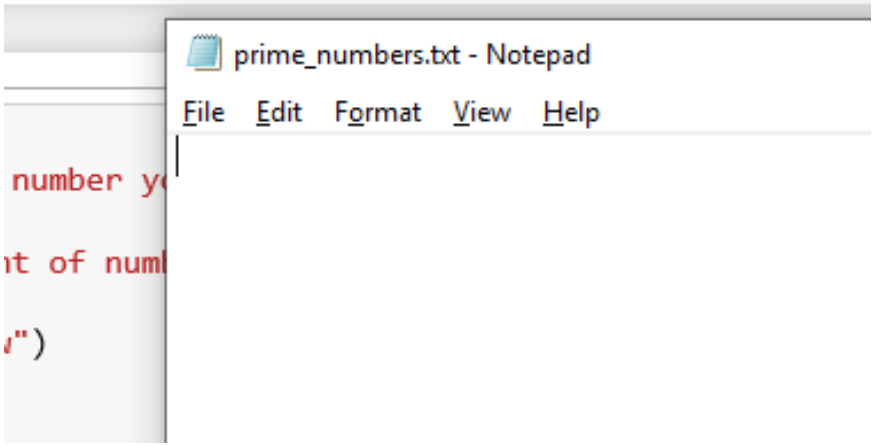
- The entire line now becomes outfile.write(str(i)+ '\n'). The print statement automatically outputs a newline character after whatever text it outputs, but the write method does not do that automatically.

- We also need to close the file when we are done. We cannot use with because we are not reading the file but writing data into it

Lets see how it can be done

```
1
2  start =input('what is the starting number you want?')
3  start=int(start)
4  end =input('what is the ending point of number?')
5  end=int(end)
6  outfile=open("prime_numbers.txt","w")
7
8  for i in range(start,end):
9      for j in range(2,i):
10         if(i % j==0):
11            # print(i)
12             break
13     else:
14         outfile.write(str(i) + "\n")
```

```
what is the starting number you want?11
what is the ending point of number?32
```



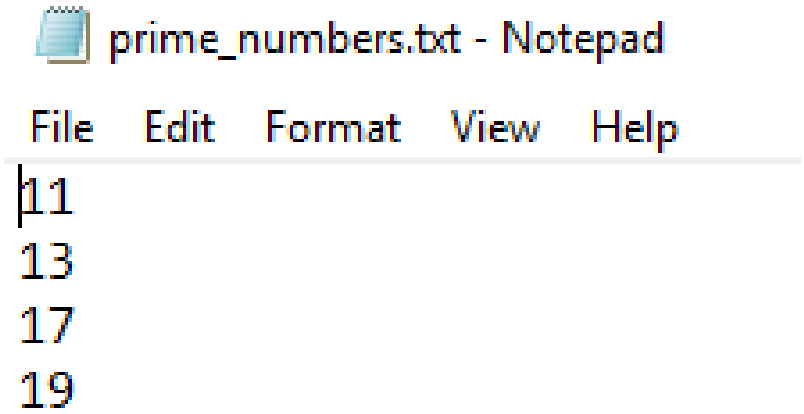
There is something wrong, even after successfully running the program, we see only file created but nothing was redirected in the file?

revisit the program and try to find out what would have went wrong!!!

```
1
2  start =input('what is the starting number you want?')
3  start=int(start)
4  end =input('what is the ending point of number?')
5  end=int(end)
6  outfile=open("prime_numbers.txt","w")
7
8  for i in range(start,end):
9      for j in range(2,i):
10         if(i % j==0):
11            # print(i)
12             break
13     else:
14         outfile.write(str(i) + "\n")
15 outfile.close()
```

```
what is the starting number you want?11
what is the ending point of number?21
```

Fixed program.
Always remember to close the file or if you find this way bogus try to use with feature by importing contextlib, make your life easy.



The Format on which Industry works-CSV

A comma-separated values (CSV) file as most of you might be knowing is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

If you print out tabular data in CSV format, it can be easily imported into other programs like Excel, Google spreadsheets, or a statistics package (R, stata, SPSS, etc.).

For example, we can make a file with the following contents. If you save it as a filename grades.csv, then you could import it into one of those programs.

- The first line gives the column names
- the later lines each give the data for one row.

Name,score,grade
Mohit,98,A+
Vijay,87,B+
Vishal,99,A+
Arun ,94,A

All file methods that we have learned so far like - **read, readline, and readlines,** and simply **iterating over the file object** itself - will work on CSV files.

In our examples, we will iterate over the lines. Because the values on each line are separated with commas, **we can use the .split()** method to parse each line into a collection of separate value.

```
olympics=open("C:\Japan_olympics_2020.txt", 'r') #we know this format of reading a file
lines=olympics.readlines()  #readlines turn all lines into a list of lines
header=lines[0] #this will print the first element of converted list
field_names=header.strip().split(',') #this will strip first meaning remove all whitespace
                                      #before and after the line
                                      #after which will splite list[0] into a list of words
                                      #of the line separated by comma

print(field_names)
for row in lines[1:]:      #we are using here lines which is a list
                           #but we are using from 1 because list[0] is header
                            #which we have already printed

    vals=row.strip().split(',')  #this will again strip and split lines one after the other
                                 #one by one
    if vals[5] != "NA":          #compare if the 6th element meaning value of 5th index is not NA
        print("{}, {}, {} ".format(vals[0],vals[4],vals[5])) #uses format value to make our life easy
        print(vals[0],",",vals[4],",",vals[5])
        print(vals[0]+", " + vals[4] + ", " +vals[5])


fileconnection.close()
```

```
['Name', 'Sex', 'Age', 'Team', 'Event', 'Medal']
Edgar Lindenau Aabye, Tug-Of-War, Gold
Edgar Lindenau Aabye , Tug-Of-War , Gold
Edgar Lindenau Aabye, Tug-Of-War, Gold
```

- You can understand what the program is doing with all the comments mentioned.

Note that the trick of splitting the text for each row based on the presence of commas only works because commas are not used in any of the field values. Suppose that some of our events were more specific, and used commas. For example, “Swimming, 100M Freestyle”.

How will a program processing a .csv file know when a comma is separating columns, and when it is just part of the text string giving a value within a column?

The CSV format is actually a little more general than we have described and has a couple of solutions for that problem. One alternative format uses a different column separator, such as | or a tab (t). Sometimes, when a tab is used, the format is called tsv, for tab-separated values).

If you get a file using a different separator, you can just call the .split('|') or .split('\\t').

"Christine Jacoba Aaftink","F","21","Netherlands","**Speed Skating, 1500M**","NA"

Writing data to a CSV file

Very basic way of writing data to files as you have seen earlier would be:

1. Have the expression which would be used to extract data for the file
2. Open the file in which you want to write
3. Create a command line which creates the header for the file cause you are creating a CSV which will have the header fields related values separated by comma
4. write an iterator loop via which you will be extracting data using the above 1. expression
5. close the file.

Let's try to convert this theoretical knowledge into an program:

```
1  #a sample list which contains details of all the participants(genuiene ones)
2  #who were supposed to  light up the Tokyo 2020 Olympics
3  #inside list every element is a tuple which we need to put in a file
4
5  light_up=[('Dina Asher-Smith',24,'Athletics','Great Britain'),
6            ('Kelsey-Lee Barber',28,'Athletics','Australia'),
7            ('Simone Biles',22,'Gymnastics','USA'),
8            ('Sky Brown',12,'Skate Boarding','Great Britain')]
9
10 #the file in which we will write the data. #syntax we all know open('file_name' , 'w')
11 out_file=open('light_up_participants.csv','w')
12
13 #now we need to create the header title
14 out_file.write('Name,age,sport,country' + '\n')
15
16 #time to start writing data in the file
17 for participants in light_up:
18
19     #first way to write the files inside the file we know we have to create a string and write
20     # into the file using file_name.write(string) using format
21     insert_row='{},{},{},{}'.format(participants[0],participants[1],participants[2],participants[3])
22     out_file.write(insert_row + '\n')
23
24     #using normal string concatenation
25     insert_row=participants[0] + ',' + str(participants[1]) + ',' + participants[2] + ',' + participants[3]
26     out_file.write(insert_row + '\n')
27
28      #using join this way would cause it contains integer value while join expects all strings
29     insert_row=','.join(participants)
30     out_file.write(insert_row + '\n')
31
32     #using the same join method, but this would be when we are not sure what kind of
33     #data does the interator contains
34     insert_row=','.join(participants[0],participants[1],participants[2],participants[3])
35     out_file.write(insert_row + '\n')
36
37 out_file.close()
38
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-90-849715085864> in <module>
     31
     32      #using join this way would cause it contains integer value while join expects all strings
---> 33     insert_row=','.join(participants)
     34     out_file.write(insert_row + '\n')
     35

TypeError: sequence item 1: expected str instance, int found
```

it failed on line 29 because join is expecting all values to be string but the sub element of list that is tuple[1] is an int which is not liked by interpreter and it fails the entire program so for now we would have to find another way of join which is below it

```python
#a sample list which contains details of all the participants(genuiene ones)
#who were supposed to  light up the Tokyo 2020 Olympics
#inside list every element is a tuple which we need to put in a file

light_up=[('Dina Asher-Smith',24,'Athletics','Great Britain'),
          ('Kelsey-Lee Barber',28,'Athletics','Australia'),
          ('Simone Biles',22,'Gymnastics','USA'),
          ('Sky Brown',12,'Skate Boarding','Great Britain')]

#the file in which we will write the data. #syntax we all know open('file_name' , 'w')
out_file=open('light_up_participants.csv','w')

#now we need to create the header title
out_file.write('Name,age,sport,country' + '\n')

#time to start writing data in the file
for participants in light_up:

    #first way to write the files inside the file we know we have to create a string and write
    # into the file using file_name.write(string) using format
    insert_row='{},{},{},{}'.format(participants[0],participants[1],participants[2],participants[3])
    out_file.write(insert_row + '\n')

    #using normal string concatenation
    insert_row=participants[0] + ',' + str(participants[1]) + ',' + participants[2] + ',' + participants[3]
    out_file.write(insert_row + '\n')

     #using join this way would cause it contains integer value while join expects all strings
    #insert_row=','.join(participants)
    #out_file.write(insert_row + '\n')

    #using the same join method, but this would be when we are not sure what kind of
    #data does the interator contains
    insert_row=','.join(participants[0],participants[1],participants[2],participants[3])
    out_file.write(insert_row + '\n')

out_file.close()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-91-6696e498ae57> in <module>
     32     #using the same join method, but this would be when we are not sure what kind of
     33     #data does the interator contains
---> 34     insert_row=','.join(participants[0],participants[1],participants[2],participants[3])
     35     out_file.write(insert_row + '\n')
     36

TypeError: join() takes exactly one argument (4 given)
```

The other join is failing too, saying that it is expecting one argument but we gave 4, hmmmm python interpreter is not gelling with us good. what can we do? let's see

```python
#a sample list which contains details of all the participants(genuiene ones)
#who were supposed to  light up the Tokyo 2020 Olympics
#inside list every element is a tuple which we need to put in a file

light_up=[('Dina Asher-Smith',24,'Athletics','Great Britain'),
          ('Kelsey-Lee Barber',28,'Athletics','Australia'),
          ('Simone Biles',22,'Gymnastics','USA'),
          ('Sky Brown',12,'Skate Boarding','Great Britain')]

#the file in which we will write the data. #syntax we all know open('file_name' , 'w')
out_file=open('light_up_participants.csv','w')

#now we need to create the header title
out_file.write('Name,age,sport,country' + '\n')

#time to start writing data in the file
for participants in light_up:

    #first way to write the files inside the file we know we have to create a string and write
    # into the file using file_name.write(string) using format
    insert_row='{},{},{},{}'.format(participants[0],participants[1],participants[2],participants[3])
    out_file.write(insert_row + '\n')

    #using normal string concatenation
    insert_row=participants[0] + ',' + str(participants[1]) + ',' + participants[2] + ',' + participants[3]
    out_file.write(insert_row + '\n')

     #using join this way would cause it contains integer value while join expects all strings
    #insert_row=','.join(participants)
    #out_file.write(insert_row + '\n')

    #using the same join method, but this would be when we are not sure what kind of
    #data does the interator contains
    insert_row=','.join([participants[0],participants[1],participants[2],participants[3]])
    out_file.write(insert_row + '\n')

out_file.close()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-92-eb8dde417afd> in <module>
     32     #using the same join method, but this would be when we are not sure what kind of
     33     #data does the interator contains
---> 34     insert_row=','.join([participants[0],participants[1],participants[2],participants[3]])
     35     out_file.write(insert_row + '\n')
     36

TypeError: sequence item 1: expected str instance, int found
```

This is one of the best feature of python, you want 1 argument, convert the list of arguments into a list on the fly but it is still failing . Cause we know the tuple contains an integer and join wants all strings. Final fix would be to convert the integer value to string

```python
#a sample list which contains details of all the participants(genuiene ones)
#who were supposed to  light up the Tokyo 2020 Olympics
#inside list every element is a tuple which we need to put in a file

light_up=[('Dina Asher-Smith',24,'Athletics','Great Britain'),
          ('Kelsey-Lee Barber',28,'Athletics','Australia'),
          ('Simone Biles',22,'Gymnastics','USA'),
          ('Sky Brown',12,'Skate Boarding','Great Britain')]

#the file in which we will write the data. #syntax we all know open('file_name' , 'w')
out_file=open('light_up_participants.csv','w')

#now we need to create the header title
out_file.write('Name,age,sport,country' + '\n')

#time to start writing data in the file
for participants in light_up:

    #first way to write the files inside the file we know we have to create a string and write
    # into the file using file_name.write(string) using format
    insert_row='{},{},{},{}'.format(participants[0],participants[1],participants[2],participants[3])
    out_file.write(insert_row + '\n')

    #using normal string concatenation
    insert_row=participants[0] + ',' + str(participants[1]) + ',' + participants[2] + ',' + participants[3]
    out_file.write(insert_row + '\n')

     #using join this way would cause it contains integer value while join expects all strings
    #insert_row=','.join(participants)
    #out_file.write(insert_row + '\n')

    #using the same join method, but this would be when we are not sure what kind of
    #data does the interator contains
    insert_row=','.join([participants[0],str(participants[1]),participants[2],participants[3]])
    out_file.write(insert_row + '\n')

out_file.close()

```

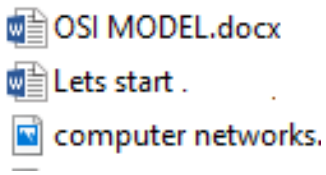
Finally, we have the output

Name	age	sport	country
Dina Asher-Smith	24	Athletics	Great Britain
Dina Asher-Smith	24	Athletics	Great Britain
Dina Asher-Smith	24	Athletics	Great Britain
Kelsey-Lee Barber	28	Athletics	Australia
Kelsey-Lee Barber	28	Athletics	Australia
Kelsey-Lee Barber	28	Athletics	Australia
Simone Biles	22	Gymnastics	USA
Simone Biles	22	Gymnastics	USA
Simone Biles	22	Gymnastics	USA
Sky Brown	12	Skate Boarding	Great Britain
Sky Brown	12	Skate Boarding	Great Britain
Sky Brown	12	Skate Boarding	Great Britain

One more thing to be kept in mind, is how you name the files. Different OS allows different ways to name your file or directories/folders.

But we need to follow some convention which is supported by all OS which is we need to make sure we are not using :

- Space - While creating files/folders in windows OS i too keep file names with spaces, don't do that



- Using forward and backward slash - we know slash denotes change of directory in cmd/linux, don't use them in your file names

- Don't use multiple extensions - txt.bkp.csv , that's really bad way to keep files

- don't capitalize the extension - textfile.EXE

Glossary

read: Will read the entire contents of a file as a string. This is often used in an assignment statement so that a variable can reference the contents of the file.

readline : Will read a single line from the file, up to and including the first instance of the newline character.

readlines: Will read the entire contents of a file into a list where each line of the file is a string and is an element in the list.

PART 6