Project 1: Manipulating Files and folders.

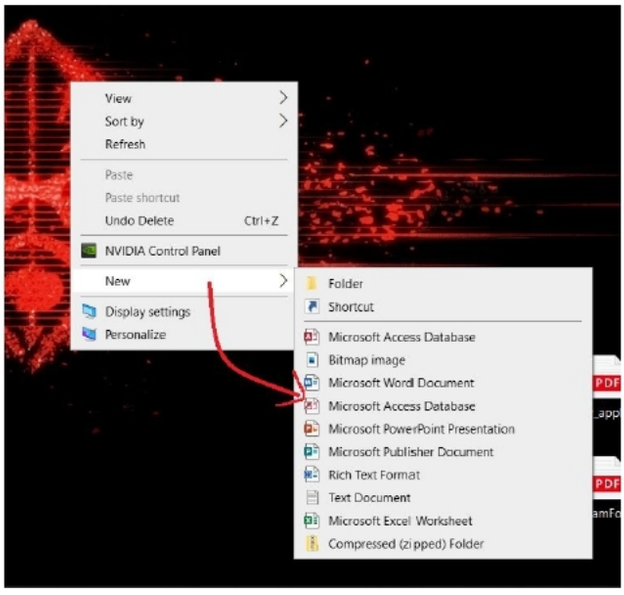
In this project, we will be having different operations on files and folders. These operations include:

- How to **create a new file**?
- How to **update a file**?
- How to **move a file**?
- How to **rename a file**?
- How to **delete a file**?

Creating a new file

To create a new file in our device, we simply do a right click, then we go to NEW and then add any folder or file to our device. If I write down the steps, it is as followed:

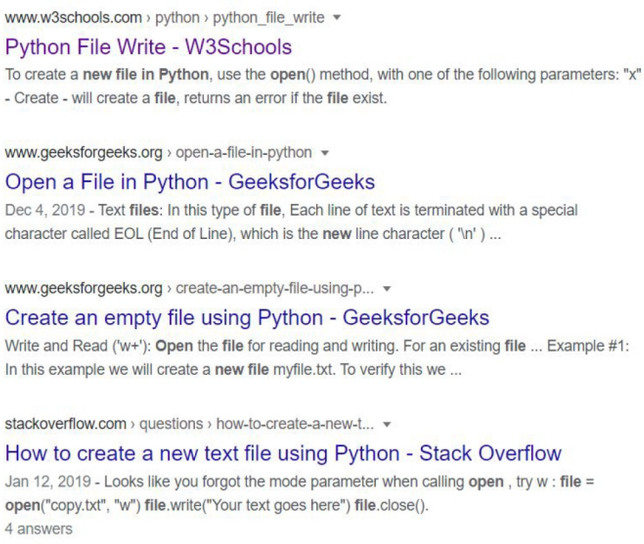
1). Open a new file



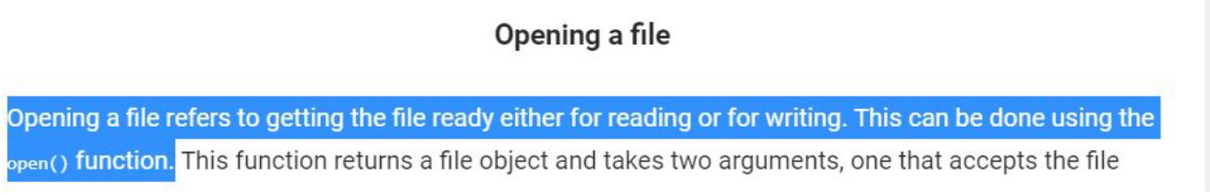
2). Give it a name.

3). Press ENTER and it will be saved.

So in 3 steps a new file will be created in our device. Now to create a new file through our program, we will be following the same 3 steps. So at first we figured out that I need to open a new file. But i don't know how to open it. In that case we can simply search for any query on Google or any other Search Engine. So if you search for *how to open a new file in python* then you will be getting various search results. Here is the snapshot of search results that i got.



Now on clicking on any one of the search results we will get to know how actually we can open a new file in python. For example,



This is what I found in one of the search results. So it says with the open() function of python, we can open a new file. But **what is a function?**

Function A function simply means a **block of code** which can be **reused.** That means whichever part of the code is repeated again and again, instead of writing them again and again, we make a function with those lines of codes and use that function again and again.

Functions are of two types.
1). User-defined Functions
2). Built-in Functions.

User-defined Functions: These are the functions created by the user. Here user means the person who is writing the code. To create a function, there are two steps.

Step1). Define a function: Defining a function simply means writing a function. Here you follow a syntax to write a function and give it a name.

So that the machine understands what the function does. The syntax to define a function is as followed.

```
def name_of_the_function(arguments):
        ……..
        ……..
        ……..
        return value
```

- Here *def* is a keyword which we use to define a function.
- At the place of *name_of_the_function*, you give any suitable name to the function
- *arguments* are the input to the function
- *value* is the output of the function which the function will return

Step2). Calling a Function: When you defined a function, you made your machine understand what this function does. But to use the same function, you will have to call it.
The syntax to call the function is as followed.
output= name_of_the_function(input)

For example: We need to create a function to find the square of a number.
Step1). Defining a function.

```
def sqr(x):
        a=x*x
        return a
```

Step 2). Calling a Function.

```
b=sqr(8)
```

Built-in Function: These are the functions which are predefined in any coding language. You don't need to define it. Whenever you want to use these functions, You simply call them.

For example:

print("any value to be printed out") is a built-in function of python, so whenever you want to use it, you simply call it and give the statement as input which you want to be printed out over the screen.

print("hello world") will print out *hello world* over the screen.

Now to create a file, we need a function called **open().**

The syntax to use this function is as followed:

Syntax:

file_object=open("file name", "mode")

where,

file name is the name with which you want to open the file.

mode is the mode in which you want to open the file.the possible values are:

r-> the file will be available for reading only

w-> the file will be available for writing only

a-> the file will be available for appending only

r+/w+/a+-> the file will be available for both reading and writing

So, this line of code will let you create a file with the name exm.txt in the D drive and returns you a file object fo.

```
fo=open("D:\\exm.txt",'w')
```

Now to write in this file, simply use file object.write and whatever you want to write in this file, give as an argument here.

So next we add this line of code

```
fo.write("Hello!!!")
```

This will write Hello!!! In your file. After running this, when you open the file exm.txt, you will find Hello!!! Written on it.

Now if you want to add multiple lines here, then you must create a list of all the sentences.
But wait. You don't know what a list is. So a question arises here, **what is a list?**

List Data structure of ordered sequence in python. That simply means more than one element and put in an order.
To create a List in python, we simply enclose the elements in square brackets([]) and separate them by comma(,).

For example:
If you want to create a list of the first five alphabets, you simply put that like this.
l=['a','b','c','d','e']
Here 'l' is the name of the list.

Now to access any element in this list, we use the "**index**". Index is the identity of any element of the list. The indexing starts with 0.
So in the above case,
the index of 'a' is 0,
the index of 'b' is 1,
the index of 'c' is 2,
the index of 'd' is 3,
the index of 'e' is 4,

Let's say you want to get c. then you will have to write l[2], i.e.
name_of_the_list[index_of_the_element]

Now coming back to creating elements, to add multiple lines here, then you must create a list of all the sentences.

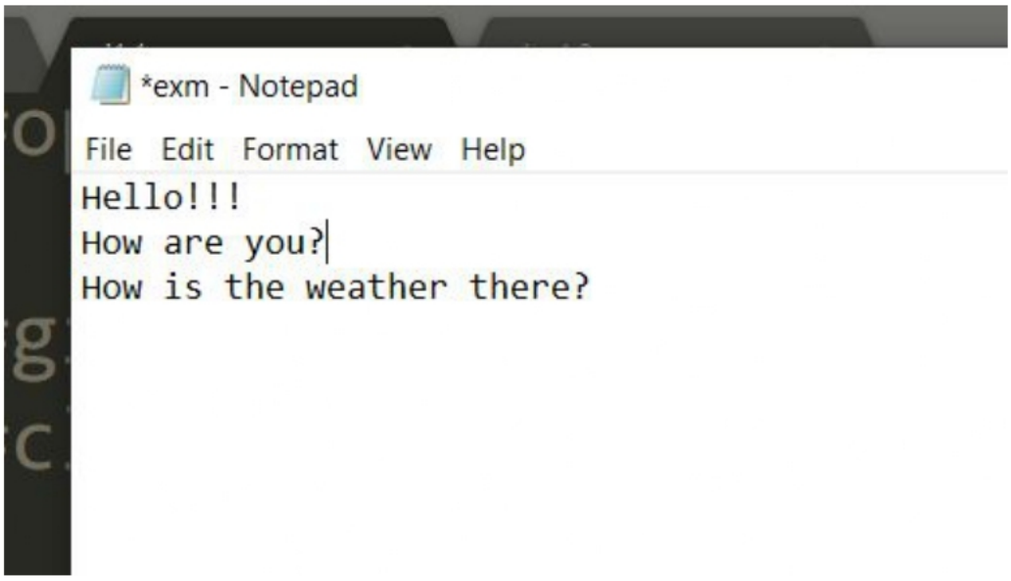
So the code to create the list is as followed:

```
line_list=["How are you? \n","How is the weather there? \n"]
```

Now to write them, you need to use writelines instead of write.
So the code to write these elements in the file whose object is fo, is as followed.

```
fo.writelines(line_list)
```

After running these lines, you file will look like this:



After you are done with working on this file, just close this file. Closing the file is important because if you don't close the file then this file can not be accessed by any other program.

To close the file, run the following line of code:

```
fo.close()
```

Updating a file

Updating a file means making some changes in the file. To Update a file, we will be following the same steps as we did in creating a file, only difference would be **instead of opening the file in writing mode, we will be opening it in append mode.** Doing this, you will be able to add new data in the file without altering the data already present in the file.

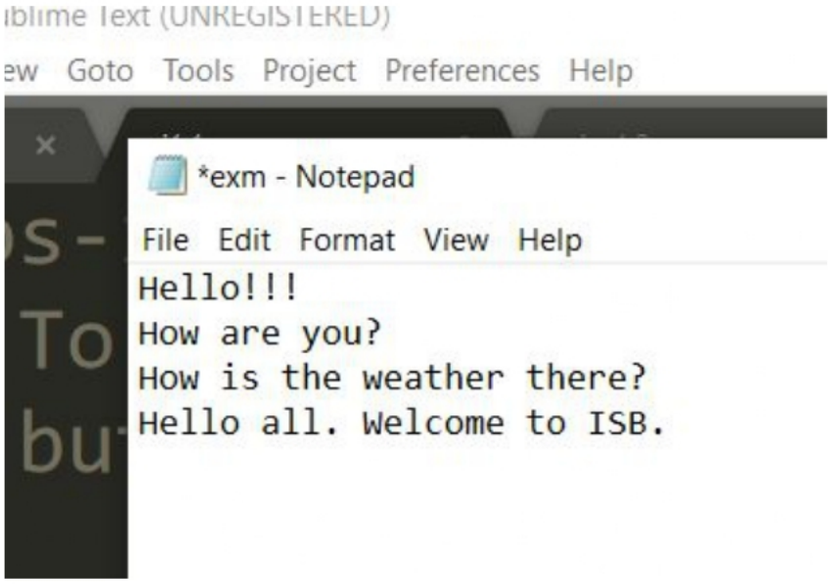
So the first step is to open the file in append mode. For this we will be running the following code.

```
fo=open("D:\\exm.txt",'a')
```

Then to update the data, we simply write with this file object like this:

```
fo.write("Hello all. Welcome to ISB.")
```

This will make the file look like this:



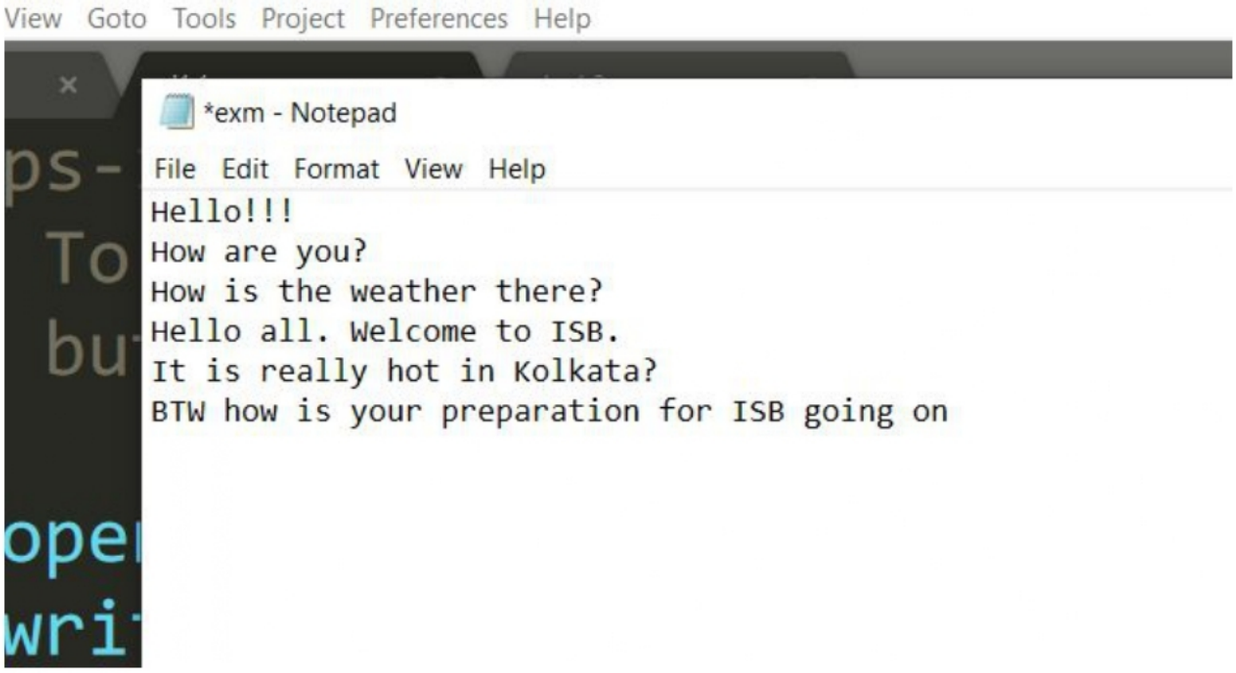
Suppose if you want to update with multiple lines, then we will use writelines instead of write.
As we did in the previous case, first all of we will create a list of lines and then call writelines.

The code for that is this:

```
line_list=["It is really hot in Kolkata.\n","BTW how is your preparation for ISB going on\n","Are you attending the warm up session?\n"]

fo.writelines(line_list)
```

This will make the file look like this.



Now let's take a use case to understand how to update the file.

Let's say,
You have one file which has data for the starting 15 days of a month and another file which has data for the next 15 days of a month. Now you want to update the data in the older file.

For that the steps which we should follow are:

1. open the first file in append mode
   To do this, we will run this code:
   ```
   ff=open("D:\\example.txt",'a')
   ```

2. open the second file in read mode
   To do this, we will run this code:
   ```
   sf=open("D:\\exm.txt",'r')
   ```

3. read data from second file
   To do this, we will run this code:
   ```
   info=sf.read()
   ```

4. append the info data in the first file
   To do this, we will run this code:
   `sf.write(info)`

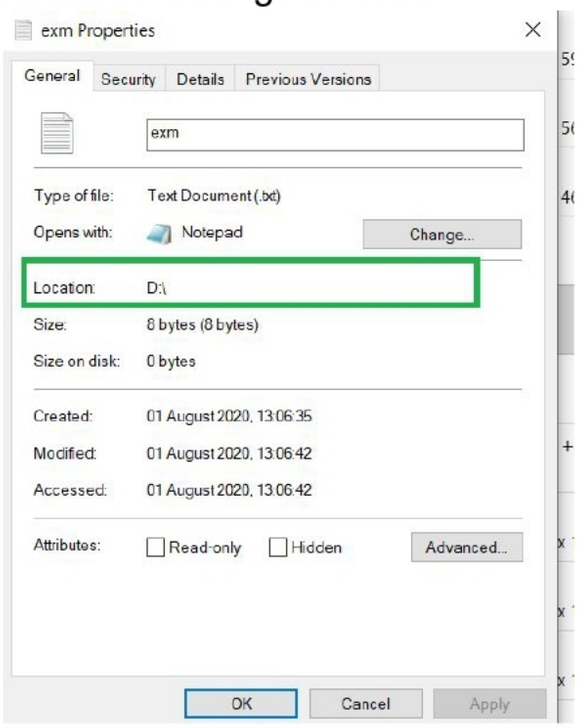
5. finally close both the files.
   To do this, we will run this code:
   `ff.close()`
   `sf.close()`

Following this step, you will be able to update the older file with data in the newer file.

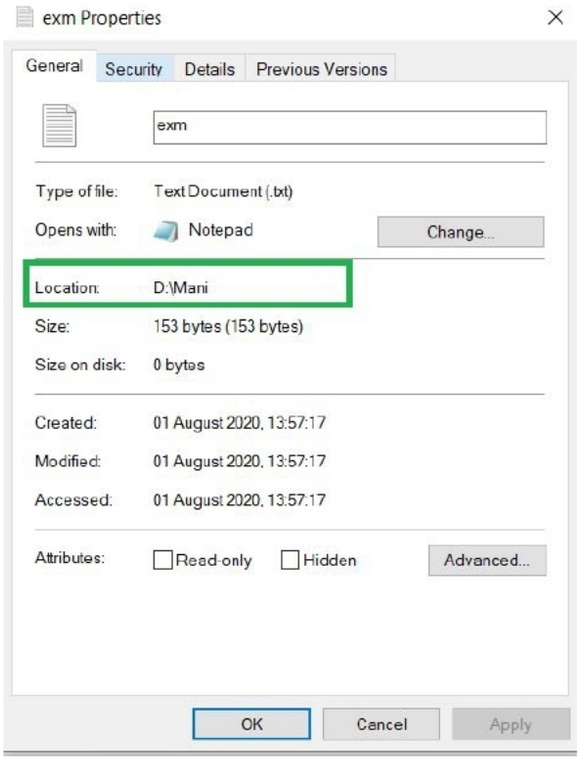
Moving a file or folder

Here we will be seeing how we can move a file or folder from one location to the other. But first let's see what happens when we try to move a file from one place to another. You will see, when you move a file, it simply **changes the path/location of the file**.

For example, right now the exm.txt file is in my D drive. If I open the properties of this file this is looking like this:



Look at the path here. And now when i move it to a folder named “Mani’ in my d drive, then it will look like this:



That simply means, if you move a file from one place to another, then the location of the file changes. Now when you have to work with paths and location of files in the device then the **module OS** is going to help you.

But before moving ahead, let’s see **what is a module?**
A **module** It is a **python file** with a set **of predefined functions, classes & variables.** This means some usable functions, classes etc are already written in some python file and these files have been made available to be used publicly. These python files are called modules.

Just like functions, modules are also of two types. One of which is built-in module.

Project 2: Playing with PDFs.

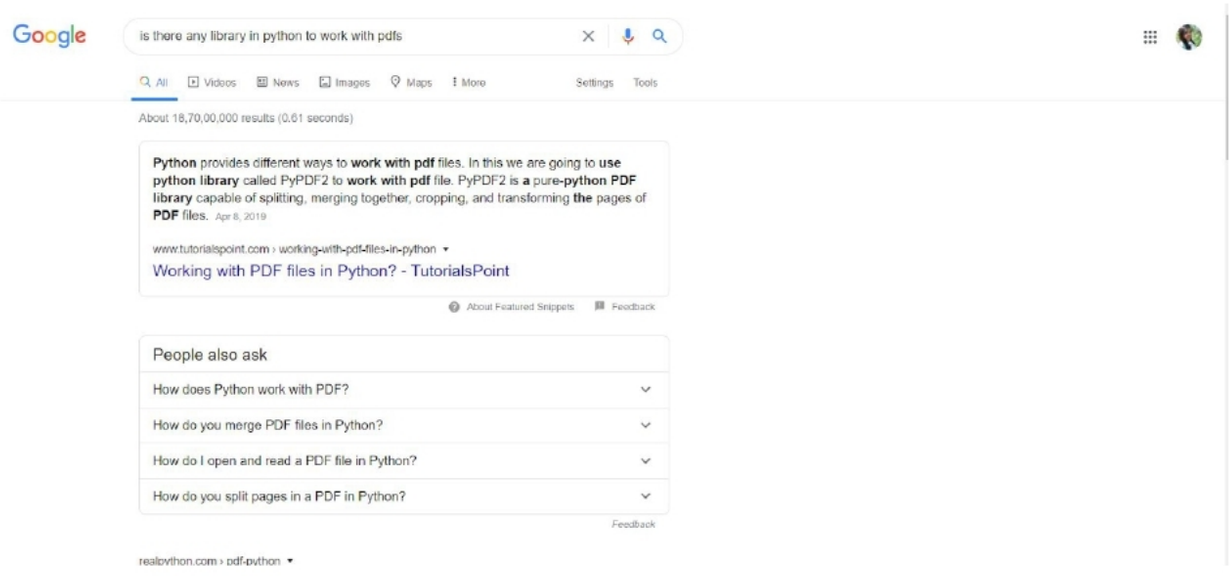
In this project we will work with pdfs and perform some operations on them like :-

1. Merging pdfs together in a single file
2. Adding watermark to the pdf
3. Encrypting the pdf

So let's begin !!

Step 1 : So the first question that arises is how are we going to work with pdfs in Python?? There should be some kind of tool that helps us to do so. Well yes we have a library for working with pdfs.

Let's Google it out to find out what it is.



Well the result shows that a library named PyPDF2 helps us in working with pdfs.

What is PyPDF2 ?

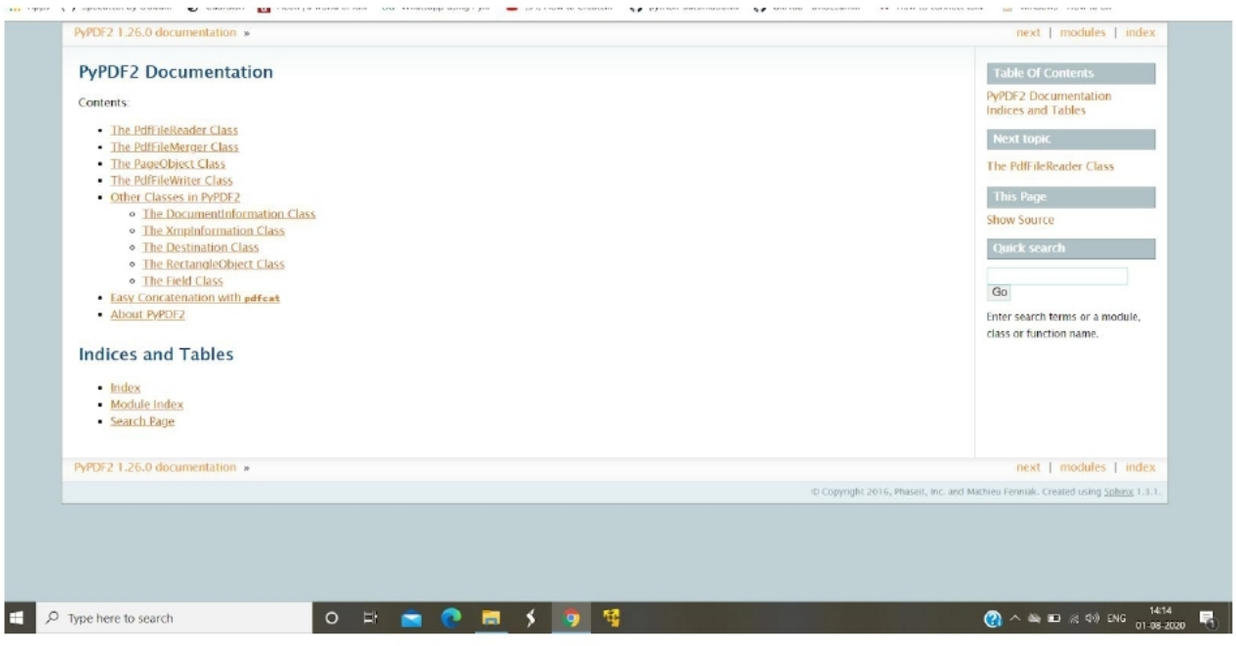
PyPDF2 is a pure-python PDF library capable of splitting, merging together, cropping, and transforming the pages of PDF files. It can also add custom data, viewing options, and passwords to PDF files. It can

retrieve text and metadata from PDFs as well as merge entire files together.

So let's install it.

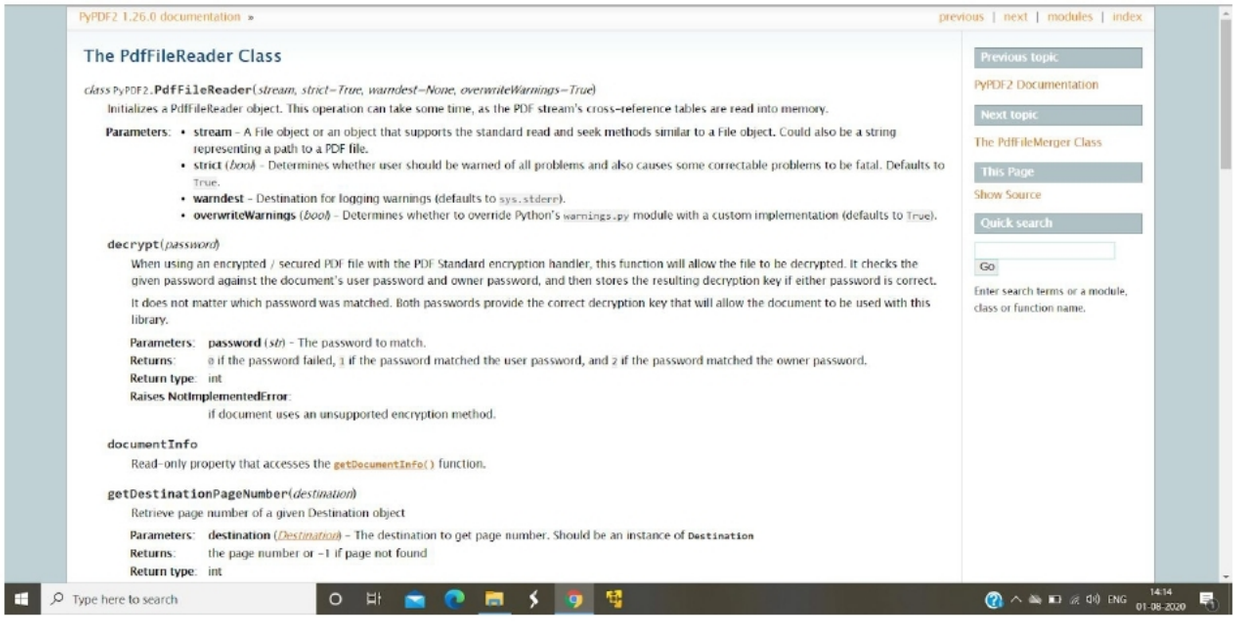
Step 2: Now once we are done with installing the library the next question that arises is how are we going to start coding. So what is the first thing that we want to do???

Of Course we would like to read the pdf file and similarly create the final output pdf file. So for that we again need some special tool. Well lets check out the documentation page of PyPDF2 to know what else we need to start our work.



Well it seems we need to import a few classes that will help us. The classes that we need are PdfFileReader Class and PdfFileWriter Class. So let's find out more about them before moving forward.

PdfFileReader Class :-



PyPDF2 1.26.0 documentation » previous | next | modules | index

The PdfFileReader Class

class PyPDF2.**PdfFileReader**(*stream, strict=True, warndest=None, overwriteWarnings=True*)

Initializes a PdfFileReader object. This operation can take some time, as the PDF stream's cross-reference tables are read into memory.

Parameters:
- **stream** – A File object or an object that supports the standard read and seek methods similar to a File object. Could also be a string representing a path to a PDF file.
- **strict** (*bool*) – Determines whether user should be warned of all problems and also causes some correctable problems to be fatal. Defaults to `True`.
- **warndest** – Destination for logging warnings (defaults to `sys.stderr`).
- **overwriteWarnings** (*bool*) – Determines whether to override Python's `warnings.py` module with a custom implementation (defaults to `True`).

decrypt(*password*)

When using an encrypted / secured PDF file with the PDF Standard encryption handler, this function will allow the file to be decrypted. It checks the given password against the document's user password and owner password, and then stores the resulting decryption key if either password is correct.

It does not matter which password was matched. Both passwords provide the correct decryption key that will allow the document to be used with this library.

Parameters: **password** (*str*) – The password to match.
Returns: `0` if the password failed, `1` if the password matched the user password, and `2` if the password matched the owner password.
Return type: int
Raises NotImplementedError: if document uses an unsupported encryption method.

documentInfo

Read-only property that accesses the `getDocumentInfo()` function.

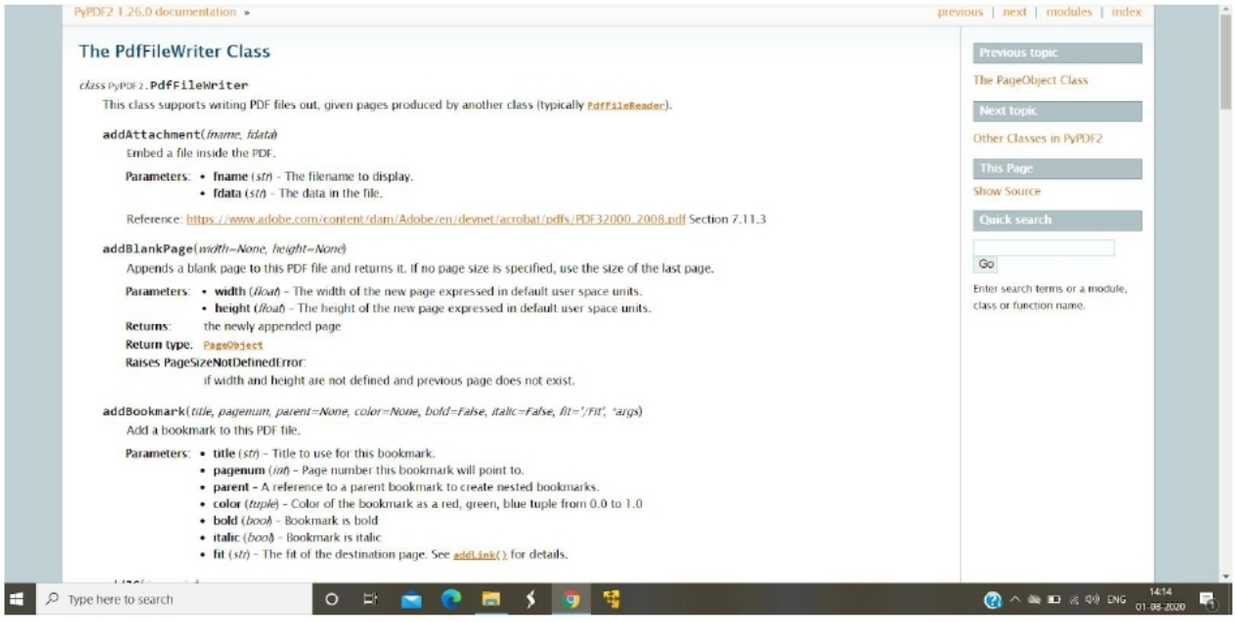
getDestinationPageNumber(*destination*)

Retrieve page number of a given Destination object

Parameters: **destination** (*Destination*) – The destination to get page number. Should be an instance of `Destination`
Returns: the page number or -1 if page not found
Return type: int

Previous topic: PyPDF2 Documentation
Next topic: The PdfFileMerger Class
This Page: Show Source
Quick search: Go — Enter search terms or a module, class or function name.

PdfFileWriter Class :



PyPDF2 1.26.0 documentation » previous | next | modules | index

The PdfFileWriter Class

class PyPDF2.**PdfFileWriter**

This class supports writing PDF files out, given pages produced by another class (typically `PdfFileReader`).

addAttachment(*fname, fdata*)

Embed a file inside the PDF.

Parameters:
- **fname** (*str*) – The filename to display.
- **fdata** (*str*) – The data in the file.

Reference: https://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/PDF32000_2008.pdf Section 7.11.3

addBlankPage(*width=None, height=None*)

Appends a blank page to this PDF file and returns it. If no page size is specified, use the size of the last page.

Parameters:
- **width** (*float*) – The width of the new page expressed in default user space units.
- **height** (*float*) – The height of the new page expressed in default user space units.

Returns: the newly appended page
Return type: `PageObject`
Raises PageSizeNotDefinedError: if width and height are not defined and previous page does not exist.

addBookmark(*title, pagenum, parent=None, color=None, bold=False, italic=False, fit='/Fit', *args*)

Add a bookmark to this PDF file.

Parameters:
- **title** (*str*) – Title to use for this bookmark.
- **pagenum** (*int*) – Page number this bookmark will point to.
- **parent** – A reference to a parent bookmark to create nested bookmarks.
- **color** (*tuple*) – Color of the bookmark as a red, green, blue tuple from 0.0 to 1.0
- **bold** (*bool*) – Bookmark is bold
- **italic** (*bool*) – Bookmark is italic
- **fit** (*str*) – The fit of the destination page. See `addLink()` for details.

Previous topic: The PageObject Class
Next topic: Other Classes in PyPDF2
This Page: Show Source
Quick search: Go — Enter search terms or a module, class or function name.

So now let's start with our first task ie to Merge pdfs together

Merging Pdfs

1. Create an object of PdfFileWriter class - We will be adding pages to this object and finally this will be written as the final pdf file.

Now what is an object.
An **object** is an element (or instance) of a **class**; **objects** have the behaviors of their **class**.

2. Then create a list containing the paths of the pdf files that we wish to merge together. Now what is a list??

   A **list** is a data structure in **Python** that is a mutable, or changeable, ordered sequence of elements. Each element or value that is inside of a **list** is called an item.They enable you to keep data together that belongs together. So we will take one item at a time from the list and operate on them.

   Example:
   Suppose we want to have a list of names

   names_list=[‘Aman’,’Biswa’,’David’,’Esha’,Gautam’]
   Here names_list is the name of the list.

   Now to access any element in this list, we use the “**index**”. Index is the identity of any element of the list. The indexing starts with 0.
   So in the above case,
   the index of ‘Aman’ is 0,
   the index of ‘Biswa’ is 1,
   the index of ‘David’ is 2,
   the index of ‘Esha’ is 3,
   the index of ‘Gautam’ is 4,

3. Create a for loop for iterating over each item in the list. So what is a loop and specifically a for loop.

   A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string)

Example :

```
birds = ["sparrow", "crow", "peacock"]

for x in birds:
        print(x)
```

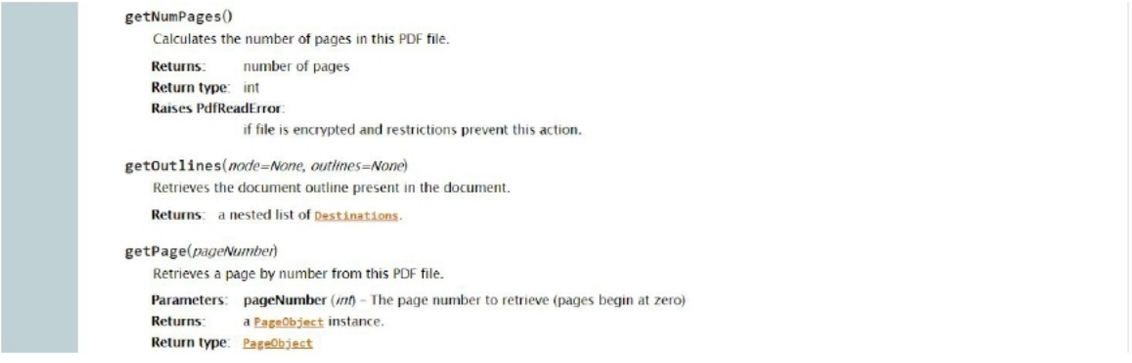
On executing the for loop the names of the birds will be displayed one by one.

4. Outer loop is used to loop over all the individual files and create an object - red_obj for each of them to read the file and using .getNumPages method we obtain the total number of pages.

5. Now the inner loop is used to loop over the pages , get them one by one using .getPage() and .addPage() is used to add pages one by one to the write object.

Well what is this .getNumPages() , .getPage() and .addPages(). Let's find out

.getNumPages() and .getPage() are methods of the PdfFileReaderClass.



`getNumPages()`
Calculates the number of pages in this PDF file.
Returns: number of pages
Return type: int
Raises PdfReadError:
if file is encrypted and restrictions prevent this action.

`getOutlines(`*node=None, outlines=None*`)`
Retrieves the document outline present in the document.
Returns: a nested list of `Destinations`.

`getPage(`*pageNumber*`)`
Retrieves a page by number from this PDF file.
Parameters: **pageNumber** (*int*) – The page number to retrieve (pages begin at zero)
Returns: a `PageObject` instance.
Return type: `PageObject`

.addPage() is a method of PdfFileWriterClass.

addPage(*page*)

Adds a page to this PDF file. The page is usually acquired from a `PdfFileReader` instance.

Parameters: **page** (*PageObject*) – The page to add to the document. Should be an instance of `PageObject`

appendPagesFromReader(*reader*, *after_page_append=None*)

Copy pages from reader to writer. Includes an optional callback parameter which is invoked after pages are appended to the writer.

Parameters: **reader** – a PdfFileReader object from which to copy page annotations to this writer object. The writer's annots will then be updated :callback after_page_append (function): Callback function that is invoked after each page is appended to the writer. Callback signature:

6. Then finally provide the path where it is to be written and then using write() finally the object is written to the pdf file.

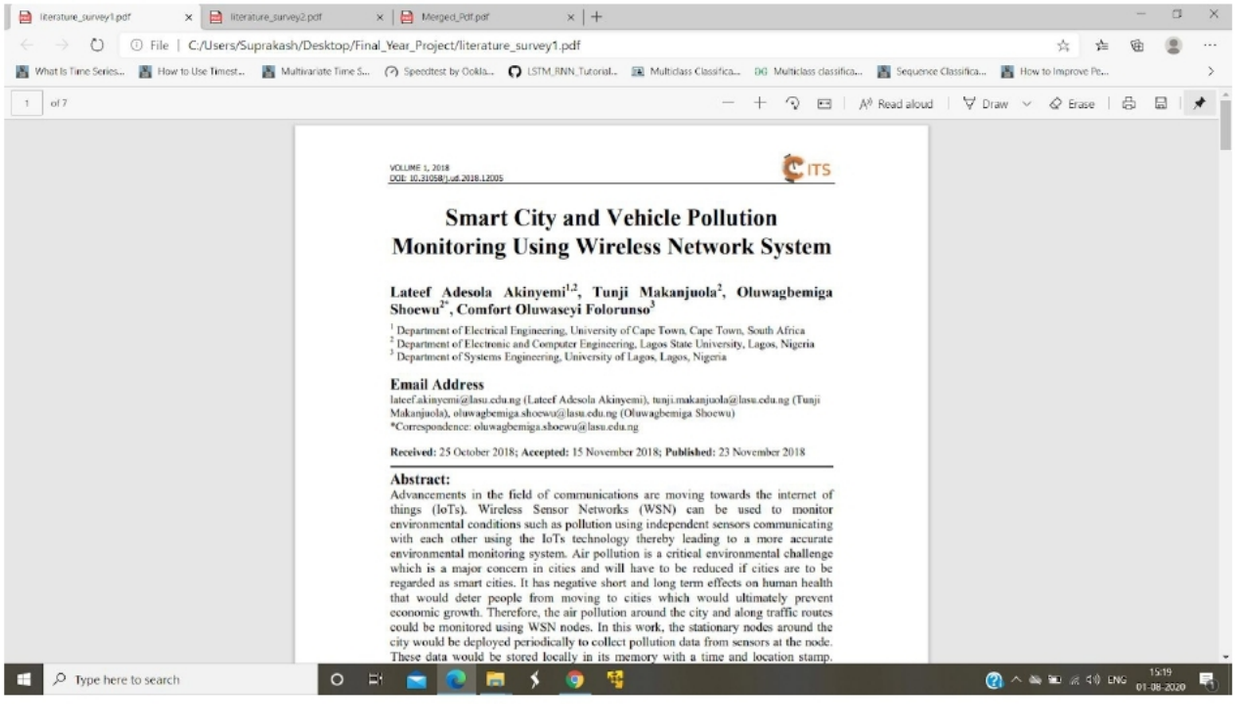
`write`(*stream*)

Writes the collection of pages added to this object out as a PDF file.

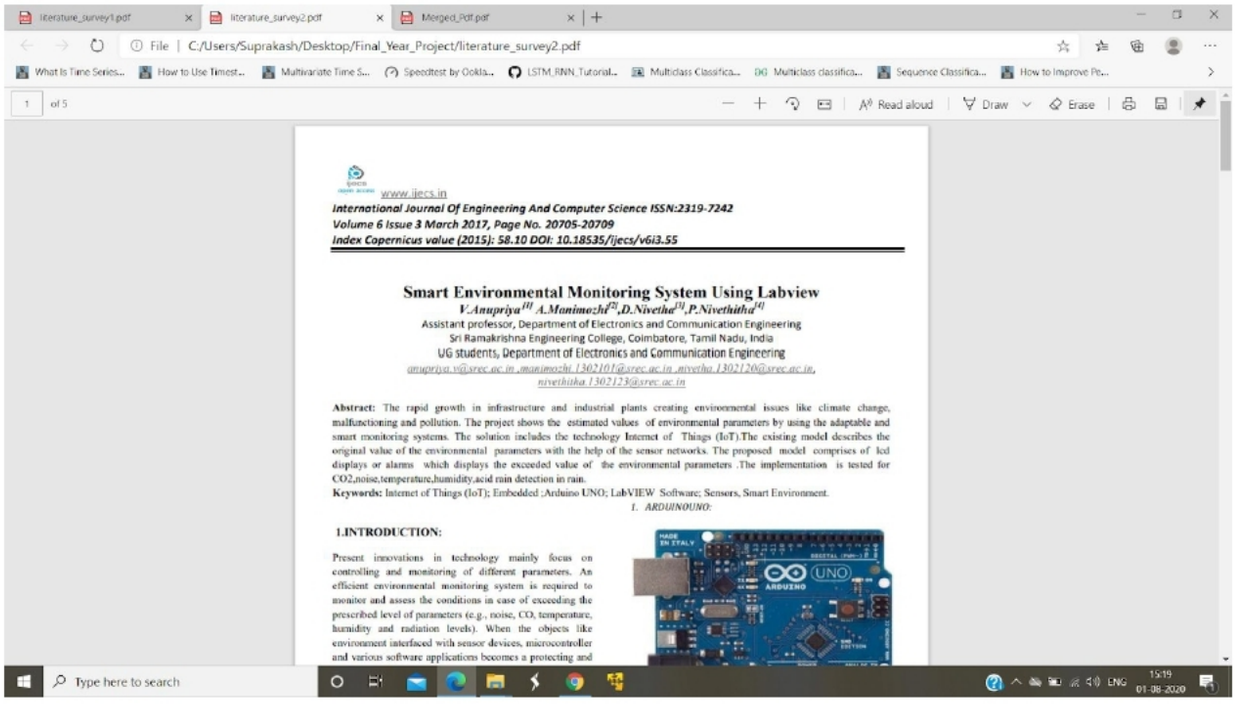
Parameters: **stream** – An object to write the file to. The object must support the write method and the tell method, similar to a file object.

Now let's have a look at the output :

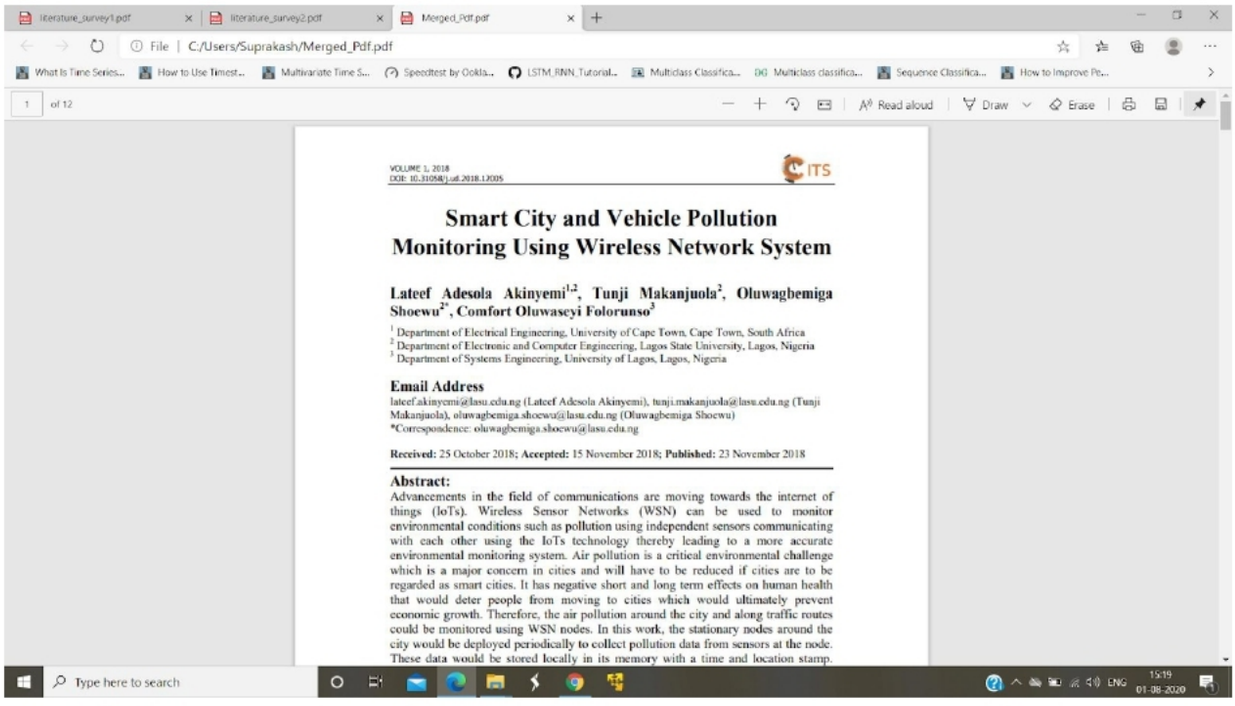
This is my first pdf -



This is my 2nd pdf -



This is my merged pdf -



You can notice that in the first pdf there are 7 pages and in the 2nd 5 pages. In the final pdf that we have merged together there are 12 pages i.e. the pdfs have been successfully merged.

Encrypting Pdf -

1. Write the code as it is till the loop.
2. The only thing that we need to add to the previous program for encryption is .encrypt()
   After adding all the pages to the write object we just need to add a line

   write_obj.encrypt(----------provide the parameters here--------)

   So let's have a look at the encrypt method and what the parameters are. This method belongs to the PdfFileWriter class.

encrypt(*user_pwd*, *owner_pwd=None*, *use_128bit=True*)

Encrypt this PDF file with the PDF Standard encryption handler.

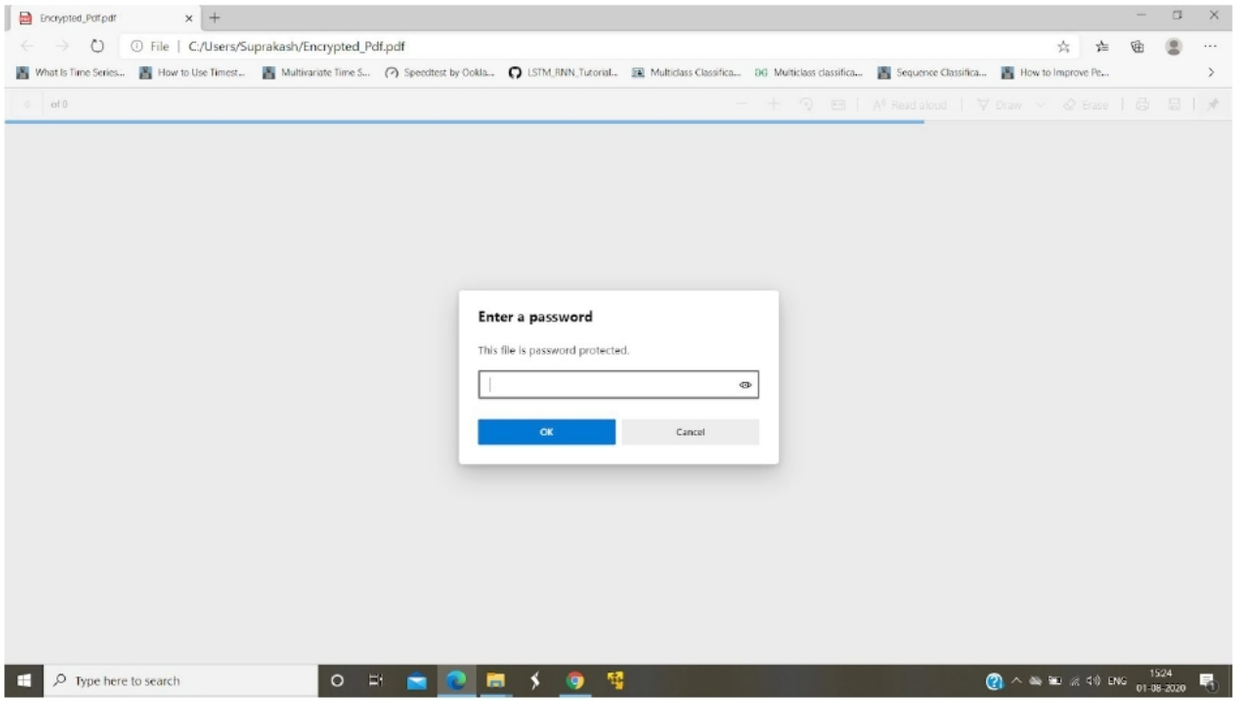
Parameters:

- **user_pwd** (*str*) – The “user password”, which allows for opening and reading the PDF file with the restrictions provided.
- **owner_pwd** (*str*) – The “owner password”, which allows for opening the PDF files without any restrictions. By default, the owner password is the same as the user password.
- **use_128bit** (*bool*) – flag as to whether to use 128bit encryption. When false, 40bit encryption will be used. By default, this flag is on

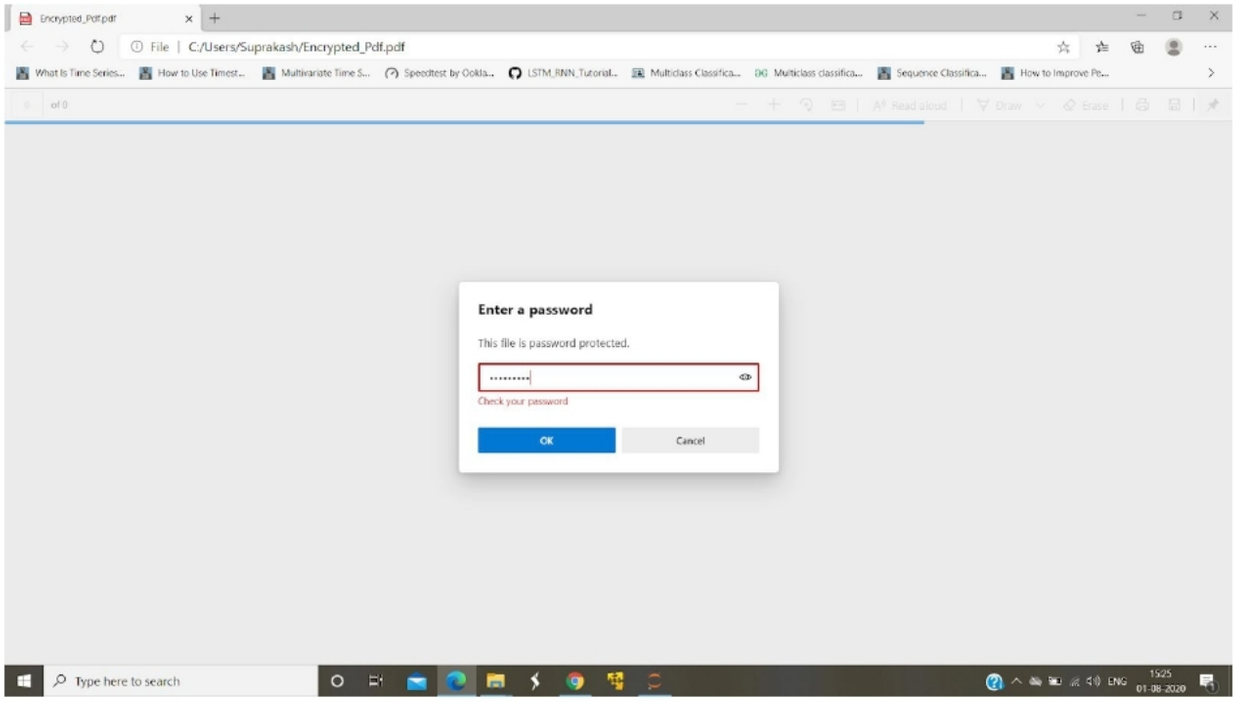
3. Provide the path where you want to save the final file and then use write() method to write the object into the file.

Let’s have a look at the output -

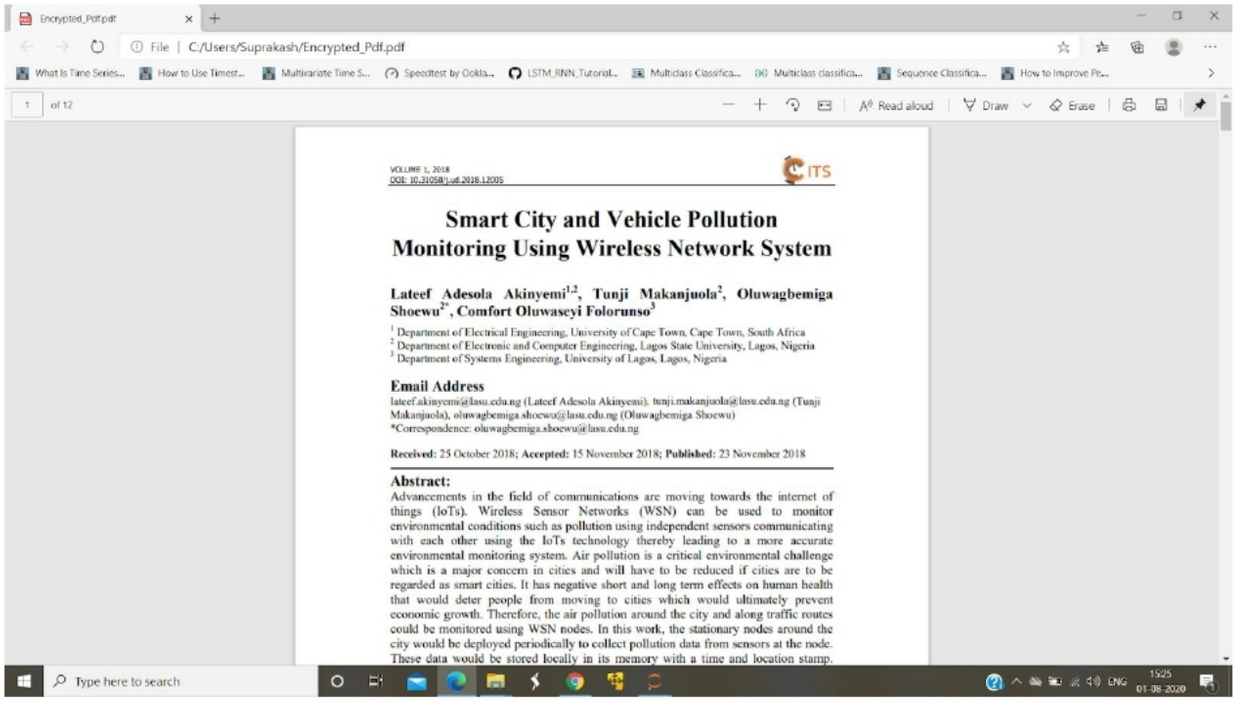
After adding an encryption when we try to open the file we get the following message -



Now if someone type a wrong password it displays the following message -



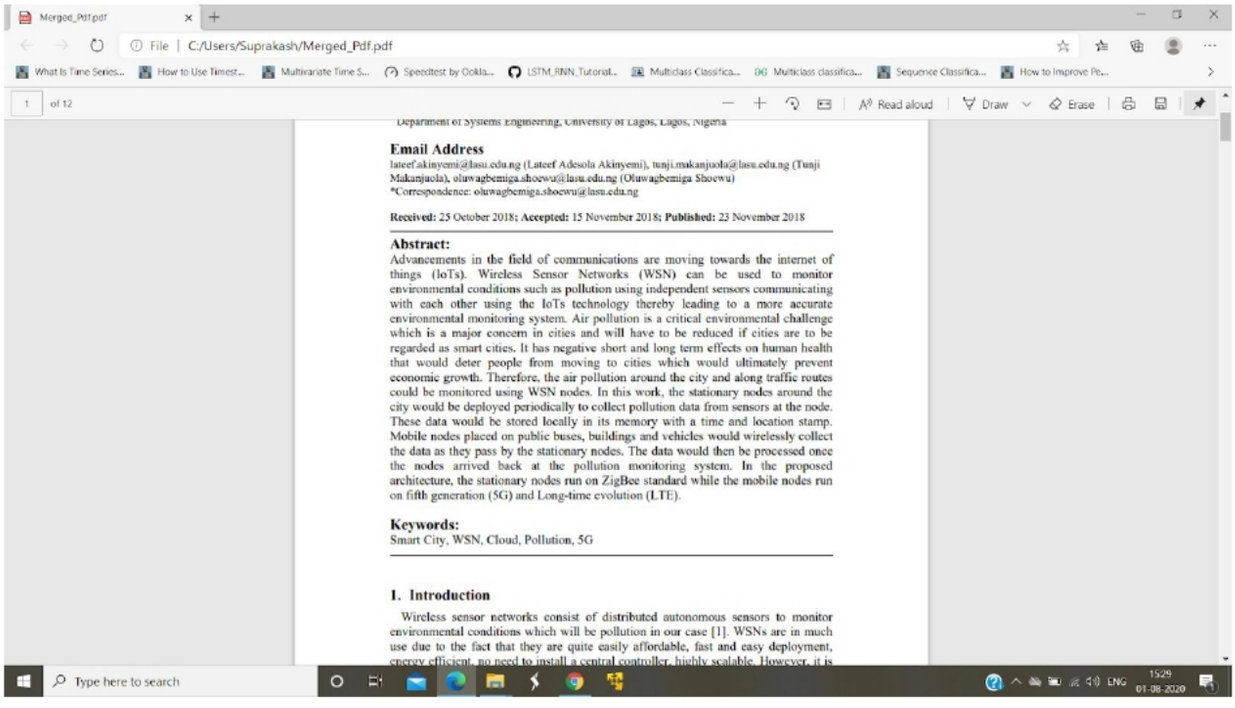
Now if you enter the correct password you will find that the file opens



Adding Watermark to a pdf -

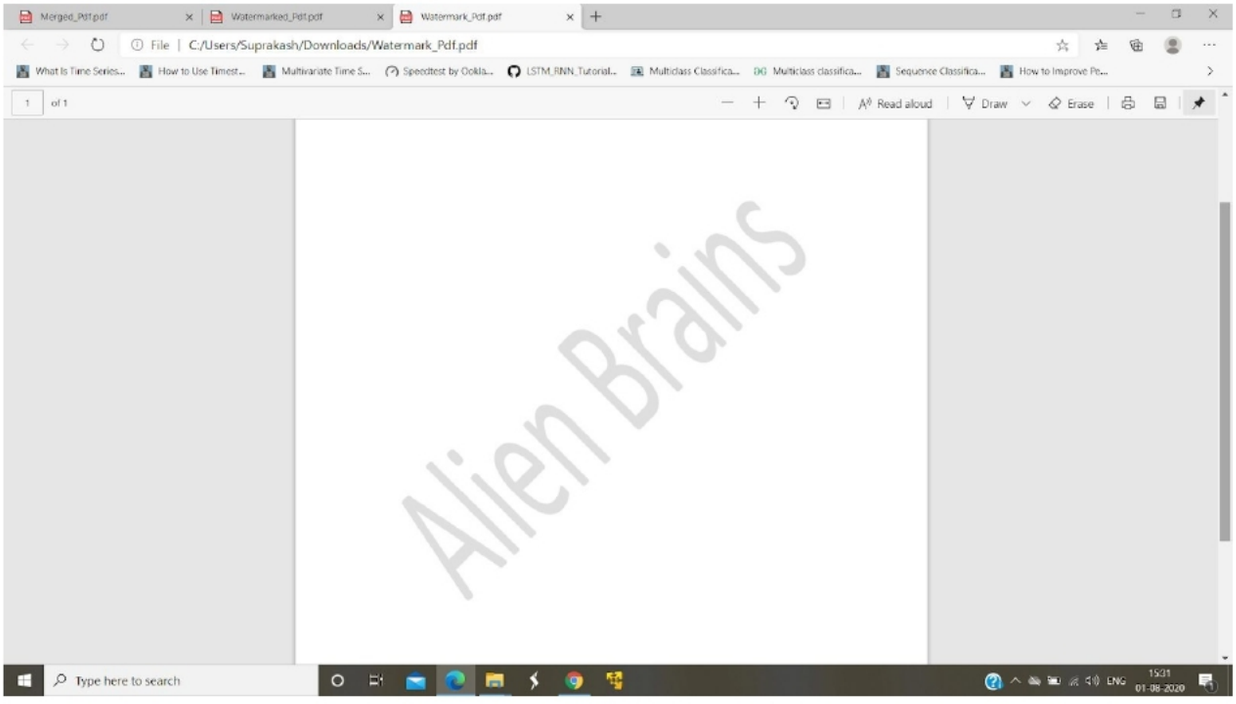
1) Again in the same way create an object of PdfFileReader class to read the input file providing the path of the input file.

The file where we wish to add the watermark is -



2) Similarly create an object of PdfFileReader class to read the file containing the watermark that we wish to add to our pdf and provide the path along with it as well.

   This the pdf containing the watermark only



3) Create the instance of the writer class
4) Get the total number of pages using .getNumPages() like we have done before
5) Using for loop iterate over the individual pages and use .getPage() to fetch them
6) merge the watermark page using .mergePage() to each individual page and then keep on adding them to create the final one

   Now what is this .mergePage(). Well it is a method of the PageObject Class.This class represents a single page within a PDF file.

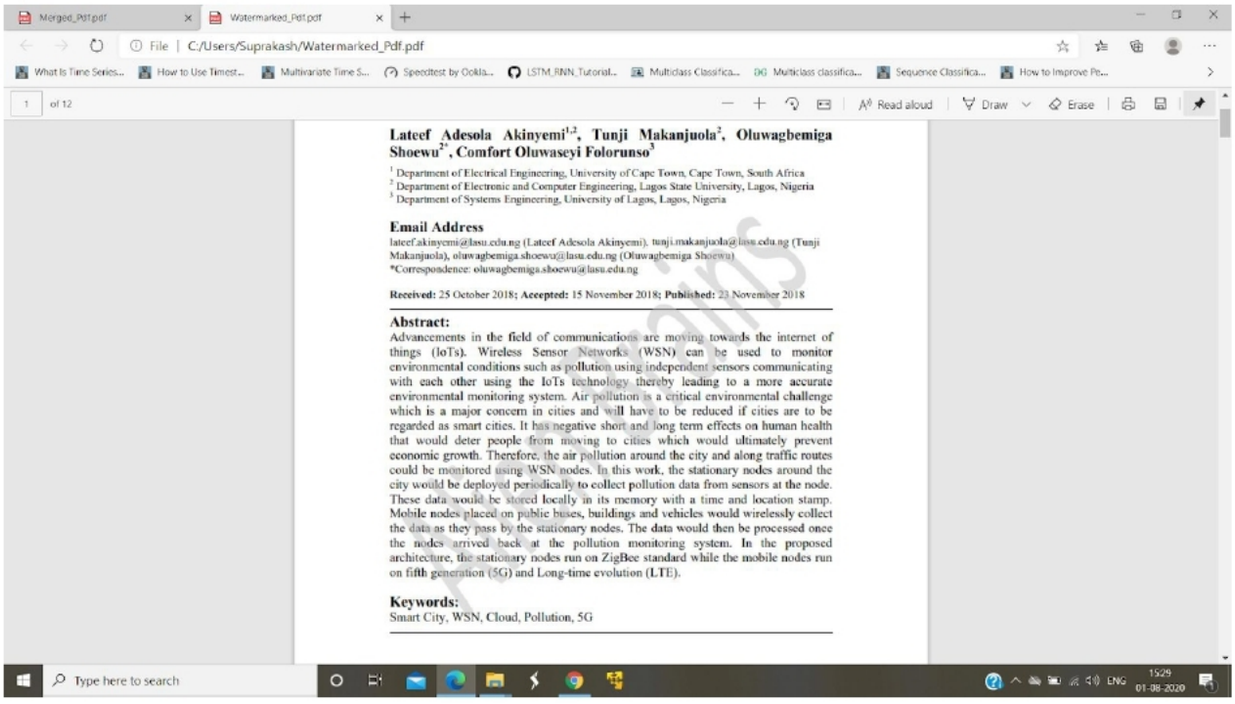
mergePage(*page2*)

Merges the content streams of two pages into one. Resource references (i.e. fonts) are maintained from both pages. The mediabox/cropbox/etc of this page are not altered. The parameter page's content stream will be added to the end of this page's content stream, meaning that it will be drawn after, or "on top" of this page.

Parameters : **page2** (*PageObject*) – The page to be merged into this one. Should be an instance of **PageObject**.

7) Mention the path of the file where the final watermarked file is to be saved and then write to that file.

Thus our final file is as follows :-



You can see that watermark has been successfully added to the pdf.