Python has built-in functions that you can use to **work with files** such as **reading** and **writing** **data** **from or to a file**. A **file object** is returned when a file is called using the open() function and then you can do the operations on it such as read, write, modify and so on.

**File Handling: Opening**

Consider a book you want to write in. First, you need to open that book so that you can write in it. Isn’t it?

Same goes here, first, you need to open file so that you can write to it. So to open a file in python we use the following **syntax**

|  |  |
| --- | --- |
| 1 | object **=** open(file\_name, mode) |

The open function returns the instance of the file that you opened to work on. It takes 2 primarily arguments, file\_name and mode. There are four different modes you can open a file to:

1. “r”  = If you want to read from a file.
2. “w” = If you want to write to a file erasing completely previous data.
3. “a” = If you want to append to previously written file.
4. “x” = If you want just to create a file.

Additional used modes to specify the type of file is:

1. “t” = Text file, Default value.
2. “b” = binary file. For eg. Images.

**For example:**

|  |  |
| --- | --- |
| 1 | fp **=** open(“my\_file.png”, “rb”) |

This will open a file named my\_file.png in binary format.

**Writing in File in Python**

To write to a file first, you must open it in write mode and then you can write to it. However, it is important to note that all previously written data will be overwritten.

For this example let’s make a file name *edureka.txt* and write in it using python.

|  |  |
| --- | --- |
| 1  2  3  4 | fp **=** open(“edureka.txt”, “wt”)  **for** \_ **in** range(10):      fp.write(“Edureka **is** a platform **for** developing market based skills”)  fp.close() |

As you can see, to write to a file I have first opened a file named edureka.txt and saved its instance in variable fp. Now I ran a loop 10 times to write “Edureka is a platform for developing market-based skills” in that file 10 times. Now for good programming practice, you must close all the files that you opened.

One thing to note here is to write texts to a file, you must open it in text mode (“t”). If you are working with binary files use “b” while opening the file.

Now let us write to a binary file, first thing to remember while writing to a binary file is that data is to be converted into binary format before writing. Moreover, binary data is not human-readable hence you cannot read it by simply opening a file.

|  |  |
| --- | --- |
| 1  2  3  4 | fp **=** open(“binaryFile”, “wb”)  Data **=** [1,2,3]  fp.write(bytearray(Data))  fp.close() |

Here you can see I have first opened *binaryFile*to write my data into it. Consider I have an array of information to write to a file(in this case *Data*) then first i converted into binary data by using function *bytearray()*so that data is converted into binary format. Then, at last, I closed the file.

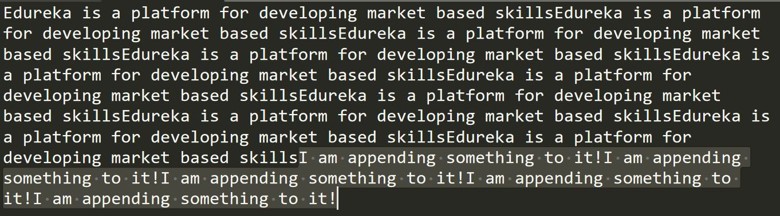
**Appending to a File**

Now, most of the times you will be writing to a file without destroying the earlier contents. To write to a file while preserving previous content is called appending to a file.

For this example let’s append to the same file that we already created. Let’s append to *edureka.txt*

|  |  |
| --- | --- |
| 1  2  3  4 | fp **=** open(“edureka,txt”, “at”)  **for** \_ **in** range(5):      fp.write(“I am appending something to it!”)  fp.close() |

Now in the above example, you can see that I have opened a file named edureka.txt using append mode. This tells python that do not overwrite data but start writing from the last line. So what it would do now is that after the ending lines it will add “I am appending something to it!” 5 times. And then we have closed that file.



So there are many reasons:

* If a file is opened to perform any operations then it’s locked to be opened by any other resource until the process itself closes it.
* Operating System keeps a check on the number of files opened by a program and thus closing files after use allows you stay within that restriction.
* Effective Resource management.
* Good programming practice.

With this, we come to an end of this File Handling in Python article. I hope you got an understanding of Opening, Reading/ Writing and Finally Closing a File in Python.

**\*args – Tuple of extra positional arguments**

python

CopyEdit

def add(\*args):

return sum(args)

add(1, 2, 3) # 6

**✅ \*\*kwargs – Dictionary of extra keyword arguments**

python

CopyEdit

def display(\*\*kwargs):

for key, value in kwargs.items():

print(f"{key} = {value}")

display(name="Alice", age=25)

**TIME COMPLEXITY**

|  |  |
| --- | --- |
| Single loop | O(n) |

|  |  |
| --- | --- |
| Nested loop | O(n²) |

|  |  |
| --- | --- |
| Binary search | O(log n) |

|  |  |
| --- | --- |
| Loop with divide (n = n//2) | O(log n) |

|  |  |
| --- | --- |
| Recursive tree | O(2ⁿ) |

**DSA Roadmap for Data Engineering**

**🔰 1. Basics (Week 1) — Foundation Layer**

These will help you understand how data is stored and moved.

* ✅ **Arrays & Strings**
  + Access, update, insert, delete
  + Slicing, string formatting
* ✅ **Lists, Tuples, Sets, Dictionaries (Python)**
  + Focus on dict as **hash map**
  + Practice set() for uniqueness

📌 *Practice:*

* Reverse a string
* Find duplicates in a list
* Count word frequencies using a dictionary

**🔁 2. Queues & Stacks (Week 2) — Job/Data Flow Logic**

Critical for job scheduling, data streaming, undo logic.

* Stack (LIFO) using list
* Queue (FIFO) using collections.deque
* Circular Queue (just conceptually)

📌 *Practice:*

* Implement a job queue
* Balanced parentheses using stack
* Reverse first K elements in a queue

**🔄 3. Hashing & Dictionaries (Week 3) — Fast Lookups**

Used heavily in real-time systems and pipelines.

* Hash maps (Python dict)
* Hash collisions (just the concept)
* Sets for uniqueness checks

📌 *Practice:*

* Find the first non-repeating character
* Two sum using hash map
* Group anagrams

**🧮 4. Sliding Window & Two Pointers (Week 4) — Streaming Efficiency**

Improves how you handle continuous/chunked data.

* Fixed window
* Variable window
* Two pointer tricks

📌 *Practice:*

* Longest substring without repeating characters
* Maximum sum of subarray of size k
* Remove duplicates from sorted array

**🛠️ 5. Heaps (Week 5) — Job Priority**

Used in job schedulers, top-K queries, resource allocation.

* Min-heap & max-heap (using heapq)
* Priority queue logic

📌 *Practice:*

* Find K largest/smallest elements
* Merge K sorted lists
* Running median

**🌱 6. Trees & Graphs (Optional for DE)**

* Very **basic binary tree** for understanding hierarchical data
* **Graphs** if working with dependency engines (e.g., Airflow DAGs)

📌 *Practice:*

* Inorder/Preorder/Postorder traversal
* Simple DFS/BFS traversal (no need for deep recursion logic)

**🧩 Additional Skills to Mix with DSA**

| **Skill** | **Why It Matters** |
| --- | --- |
| **SQL** | Every DE job expects SQL querying |
| **Pandas** | Used for data wrangling & preprocessing |
| **Airflow DAG Logic** | Involves graph-like structure |
| **Kafka/SQS Concepts** | Based on queue principles |
| **GCP/AWS Queues** | For job/data orchestration |

**🧠 Practice Sites**

* **LeetCode** → Easy tag for above topics
* **HackerRank** → Python-specific practice
* **InterviewBit** → Good for DSA roadmaps

**💡 Tips**

* Don’t overdo trees/graphs unless interviews demand it.
* Focus more on **real-world applications of DSA in DE**.
* Implement what you learn in **mini projects** like:
  + Job queue simulation
  + Stream processor
  + Simple ETL pipeline with buffering

**NUMPY**

What is numpy?

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types

Numpy Arrays Vs Python Sequences

* NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
* The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory.
* NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
* A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays.

[https://numpy.org/doc/stable/referenc...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbDNsaDktN3JFY3B0Z21kdEpfYzdhNkNLVWNFZ3xBQ3Jtc0trNGpqWDZBUU9fMWJvRllybzdNeXA4Wk42WGVJdFdLb1k5ZV8ySHl5T1VIMWhqVTJtWkh0M09aYVhkZEpfQk5DdTlyX19pUUVvZlVTdURXSWdtRDlORExhUW1TbERyU0VNbF9HVThDUkZSNXVQcVBBVQ&q=https%3A%2F%2Fnumpy.org%2Fdoc%2Fstable%2Freference%2Froutines.math.html&v=XF6DCrNTzug)

[https://numpy.org/doc/stable/referenc...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa1NyUmVFSlprUnBFSWR1Z3FaNTJpUXczVi1kd3xBQ3Jtc0tuODhUb2t5X1RYTzhWanV0X3hNVEl4bEhhN2Fwc3pVb2ZHS1FKRnR4Qkd1c0Y0N0dIN3pFNkZSTjlEdmdpcnQwNmJsUjBuTjIwbHA1VzdpZmthNlFuQW5sMXJIRjhWcWtFQmhyVkRiSnMyNWhLYVdjQQ&q=https%3A%2F%2Fnumpy.org%2Fdoc%2Fstable%2Freference%2Fufuncs.html&v=XF6DCrNTzug)

**Speed ,memory and convenience NUMPY arrays are better then lists**

**PANDAS**

What is Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

Pandas Series

A Pandas Series is like a column in a table. It is a 1-D array holding data of any type.