

# File Handling

- File Handling in Python
  - 2.7.1 Introduction to File Operations
  - 2.7.2 Opening and closing files
  - 2.7.3 File modes
  - 2.7.4 Reading from Files Reading entire files, line-by-line reading
  - 2.7.5 Using with statement for file operations
  - 2.7.6 Writing to Files: Writing data to files, Appending data to existing files
- 2.8 Working with CSV and JSON Files
  - 2.8.1 Reading and writing CSV files using csv module
  - 2.8.2 Handling JSON files using json module

# File Handling in Python

- File handling is an important activity in every web app. The types of activities that you can perform on the opened file are controlled by Access Modes. These describe how the file will be used after it has been opened.
- These modes also specify where the file handle should be located within the file. Similar to a pointer, a file handle indicates where data should be read or put into the file.

# File Modes

- In Python, there are six methods or access modes, which are:
- **Read Only ('r'):** This mode opens the text files for reading only. The start of the file is where the handle is located. It raises the I/O error if the file does not exist. This is the default mode for opening files as well.
- **Read and Write ('r+'):** This method opens the file for both reading and writing. The start of the file is where the handle is located. If the file does not exist, an I/O error gets raised.
- **Write Only ('w'):** This mode opens the file for writing only. The data in existing files are modified and overwritten. The start of the file is where the handle is located. If the file does not already exist in the folder, a new one gets created.
- **Write and Read ('w+'):** This mode opens the file for both reading and writing. The text is overwritten and deleted from an existing file. The start of the file is where the handle is located.

- **Append Only ('a'):** This mode allows the file to be opened for writing. If the file doesn't yet exist, a new one gets created. The handle is set at the end of the file. The newly written data will be added at the end, following the previously written data.
- **Append and Read ('a+')**: Using this method, you can read and write in the file. If the file doesn't already exist, one gets created. The handle is set at the end of the file. The newly written text will be added at the end, following the previously written data.
- Below is the code required to create, write to, and read text files using the Python file handling methods or access modes.

- In addition you can specify if the file should be handled as binary or text mode
- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)
- '+' Open a file for updating (reading and writing)

# Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

- **How to Create Files in Python**

- In Python, you use the `open()` function with one of the following options – "x" or "w" – to create a new file:
- **"x" – Create:** this command will create a new file if and only if there is no file already in existence with that name or else it will return an error.
- Example of creating a file in Python using the "x" command:
- `#creating a text file with the command function "x"`
- 
- `f = open("myfile.txt", "x")`



- **"w" – Write:** this command will create a new text file whether or not there is a file in the memory with the new specified name. It does not return an error if it finds an existing file with the same name – instead it will overwrite the existing file.
- Example of how to create a file with the "w" command:
- #creating a text file with the command function "w"
- 
- `f = open("myfile.txt", "w")`

- **How to Write to a File in Python**
- There are two methods of writing to a file in Python, which are:
- **The write() method:**
- This function inserts the string into the text file on a single line.
- Based on the file we have created above, the below line of code will insert the string into the created text file, which is "myfile.txt."
- 
- `file.write("Hello There\n")`

- **The writelines() method:**
- This function inserts multiple strings at the same time. A list of string elements is created, and each string is then added to the text file.
- Using the previously created file above, the below line of code will insert the string into the created text file, which is "myfile.txt."
- `f.writelines(["Hello World ", "You are welcome to Fcc\n"])`

- Example:
- `#This program shows how to write data in a text file.`
- 
- `file = open("myfile.txt","w")`
- `L = ["This is Lagos \n","This is Python \n","This is Fcc \n"]`
- 
- `# i assigned ["This is Lagos \n","This is Python \n","This is Fcc \n"] to`  
`#variable L, you can use any letter or word of your choice.`
- `# Variable are containers in which values can be stored.`
- `# The \n is placed to indicate the end of the line.`
- 
- `file.write("Hello There \n")`
- `file.writelines(L)`
- `file.close()`
-

- **How to Read From a Text File in Python**
- There are three methods of reading data from a text file in Python. They are:
- **The read() method:**
- This function returns the bytes read as a string. If no n is specified, it then reads the entire file.
- Example:
- `f = open("myfiles.txt", "r")`
- `#('r')` opens the text files for reading only
- `print(f.read())`
- `#The "f.read" prints out the data in the text file in the shell when run.`

- **The readline() method:**

- This function reads a line from a file and returns it as a string. It reads at most n bytes for the specified n. But even if n is greater than the length of the line, it does not read more than one line.

- `f = open("myfiles.txt", "r")`

- `print(f.readline())`

- **The readlines() method:**

- This function reads all of the lines and returns them as string elements in a list, one for each line.

- You can read the first two lines by calling readline() twice, reading the first two lines of the file:

- `f = open("myfiles.txt", "r")`

- `print(f.readline())`

- `print(f.readline())`

- **How to Close a Text File in Python**

- It is good practice to always close the file when you are done with it.

- **Example of closing a text file:**

- This function closes the text file when you are done modifying it:
- `f = open("myfiles.txt", "r")`
- `print(f.readline())`
- `f.close()`
- The `close()` function at the end of the code tells Python that well, I am done with this section of either creating or reading – it is just like saying End.

- **Example:**
- The program below shows more examples of ways to read and write data in a text file. Each line of code has comments to help you understand what's going on:



- `# Program to show various ways to read and`
- `# write data in a text file.`
- 
- `file = open("myfile.txt", "w")`
- `L = ["This is Lagos \n", "This is Python \n", "This is Fcc \n"]`
- 
- `#i assigned ["This is Lagos \n", "This is Python \n", "This is Fcc \n"]`
- `#to variable L`
- 
- `#The \n is placed to indicate End of Line`
- 
- `file.write("Hello There \n")`
- `file.writelines(L)`
- `file.close()`

- `# use the close() to change file access modes`
- `file = open("myfile.txt","r+")`
- `print("Output of the Read function is ")`
- `print(file.read())`
- `print()`
- 
- `# The seek(n) takes the file handle to the nth`
- `# byte from the start.`
- `file.seek(o)`
- 
- `print( "The output of the Readline function is ")`
- `print(file.readline())`
- `print()`
- 
- `file.seek(o)`

- # To show difference between read and readline
- 
- print("Output of Read(12) function is ")
- print(file.read(12))
- print()
- 
- file.seek(0)
- 
- print("Output of Readline(8) function is ")
- print(file.readline(8))
- 
- file.seek(0)
- # readlines function
- print("Output of Readlines function is ")
- print(file.readlines())
- print()
- file.close()

- While reading or writing to a file, access mode governs the type of operations possible in the opened file. It refers to how the file will be used once it's opened. These modes also define the location of the File Handle in the file. The definition of these access modes is as follows:
- **Append Only ('a'):** Open the file for writing.
- **Append and Read ('a+'):** Open the file for reading and writing

```
file1 = open("myfile.txt", "w")
L = ["This is Delhi \n", "This is Paris \n", "This is London"]
file1.writelines(L)
file1.close()
```

```
# Append-adds at last
```

```
file1 = open("myfile.txt", "a") # append mode
file1.write("Today \n")
file1.close()
```

```
file1 = open("myfile.txt", "r")
print("Output of Readlines after appending")
print(file1.read())
print()
file1.close()
```

```
# Write-Overwrites
```

```
file1 = open("myfile.txt", "w") # write mode
file1.write("Tomorrow \n")
file1.close()
```

```
file1 = open("myfile.txt", "r")
print("Output of Readlines after writing")
print(file1.read())
print()
file1.close()
```

---

### Output:

```
Output of Readlines after appending  
This is Delhi  
This is Paris  
This is LondonToday
```

```
Output of Readlines after writing  
Tomorrow
```

- **Example 2: Append data from a new line**
- In the above example of file handling, it can be seen that the data is not appended from the new line. This can be done by writing the newline '\n' character to the file.

```
file1 = open("myfile.txt", "w")
L = ["This is Delhi \n", "This is Paris \n", "This is London"]
file1.writelines(L)
file1.close()

# Append-adds at last
# append mode
file1 = open("myfile.txt", "a")

# writing newline character
file1.write("\n")
file1.write("Today")

# without newline character
file1.write("Tomorrow")

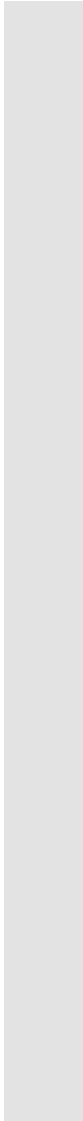
file1 = open("myfile.txt", "r")
print("Output of Readlines after appending")
print(file1.read())
print()
file1.close()
```





### Output:

```
Output of Readlines after appending  
This is Delhi  
This is Paris  
This is London  
TodayTomorrow
```



- **Example 3: Using With statement in Python**
- **with statement** is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams. Unlike the above implementations, there is no need to call `file.close()` when using with statement. The with statement itself ensures proper acquisition and release of resources.

```
L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]
```

```
# Writing to file
```

```
with open("myfile.txt", "w") as file1:
```

```
    # Writing data to a file
```

```
    file1.write("Hello \n")
```

```
    file1.writelines(L)
```

```
# Appending to file
```

```
with open("myfile.txt", 'a') as file1:
```

```
    file1.write("Today")
```

```
# Reading from file
```

```
with open("myfile.txt", "r+") as file1:
```

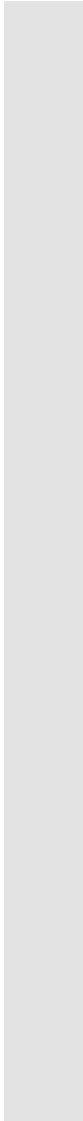
```
    # Reading form a file
```

```
    print(file1.read())
```



## Output:

```
Hello  
This is Delhi  
This is Paris  
This is London  
Today
```



- **Using the shutil module:**
- This approach uses the `shutil.copyfileobj()` method to append the contents of another file (`source_file`) to `'file.txt'`. This can be useful if you want to append the contents of one file to another without having to read the contents into memory first.

```
import shutil
```

```
with open('file.txt', 'a') as f:  
    shutil.copyfileobj(source_file, f)
```

# CSV and JSON

A CSV file is a **file that contains data in the form of characters separated by commas**. CSV files are often used to **store and exchange data between different applications that can handle tabular data**, such as spreadsheets, databases, contact managers, etc

The **csv module** is a **built-in module** that provides **functions and classes for reading and writing data in CSV format**. CSV stands for comma-separated values, which is a common format for storing and exchanging tabular data, such as spreadsheets and databases.

To use the csv module, we need to import it first:

```
import csv
```



## Opening a CSV file

To open a CSV file in Python, we use the same **open()** function that we use for text files. However, **we need to specify the newline parameter as an empty string to avoid any extra newlines in the data .**

For example, to open a CSV file named “people.csv” in read mode, we can write:

```
f = open("people.csv", "r", newline="")
```

## Closing a CSV file

To close a CSV file in Python, we use the same **close() method** that we use for text files. The **close()** method releases the resources associated with the file and ensures that any changes made to the file are saved. It is important to close a file after using it to **avoid errors or data loss**.

For example, to close the file object `f`, we can write:

```
f.close()
```

## Writing to a CSV file

To write data into a CSV file in Python, we use the **csv.writer()** function to create a writer object that **can write data to the file**. The writer object has methods like **writerow()** and **writerows()** that can **write one or more rows of data to the file**.

For example, to write some data to a CSV file named “people.csv” in write mode, we can write:

```
import csv

# open a CSV file in write mode
f = open("people.csv", "w", newline="")

# create a writer object
writer = csv.writer(f)

# write the header row
writer.writerow(["name", "id", "age", "gender"])

# write some data rows
writer.writerow(["Alice", "001", 25, "F"])
writer.writerow(["Bob", "002", 30, "M"])
writer.writerow(["Charlie", "003", 35, "M"])

# write multiple data rows at once
data = [
    ["David", "004", 40, "M"],
```

```
    ["Eve", "005", 45, "F"],
    ["Frank", "006", 50, "M"]
]

writer.writerows(data)

# close the file
f.close()
```

## Reading from a CSV file

To read data from a CSV file in Python, we use the **csv.reader()** function to create a **reader object that can iterate over the lines in the file** . The reader object returns each line as a list of strings.

For example, to read some data from a CSV file named “people.csv” in read mode, we can write:

```
import csv

# open a CSV file in read mode

f = open("people.csv", "r", newline="")

# create a reader object

reader = csv.reader(f)

# loop over each line in the file

for row in reader:

    # print the row as a list of strings

    print(row)

# close the file

f.close()
```

- See the following table

Name	Class	Section
Amit	X	A
Sumit	XI	B
Ashish	XII	C

### Python CSV File

The above table will be stored in CSV format as follows:

```
Name ,Class,Section  
Amit,X,A  
Sumit,XI,B  
Ashish,XII,C
```

If the values in the table contain comma(,) like below in column Address

Name	Class	Address
Amit	X	Fr. Agnel School, Noida
Sumit	XI	Fr. Agnel School, Noida

Then in CSV it will be stored like below (Values containing comma will be enclosed in double quotes)

```
Name,Class ,Address  
Amit,X,"Fr. Agnel School, Noida"  
Sumit,XI,"Fr. Agnel School, Noida"
```





---

# Python Provides CSV module to work with csv file:

Main Functions are:

1. **reader()**
2. **writer()**
3. **DictReader()**
4. **DictWriter()**

1. **reader() function** : This function help us to read the csv file. This function takes a file object and returns a `_csv.reader` object that can be used to iterate over the contents of a CSV file.
- 
- 

# How to read entire data from data.csv file?

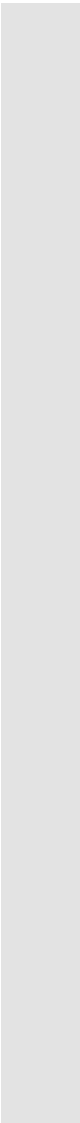
Let we try to read the following file ie "data.csv"

data.csv

CODING	OUTPUT
<pre>import csv f = open("data.csv", 'r') row = csv.reader(f) for i in row :     print(i)</pre>	<pre>['Name' , 'Class' , 'Subject'] ['Amit' , 'XII' , ' CS'] ['Sumit' , 'X' , 'IP']</pre> <p>Notice that each line in the CSV file is returned as a list of strings</p>

# How to read specific column from data.csv?

CODING	OUTPUT	Actual Data of File
<pre>import csv f = open("data.csv", 'r') row = csv.reader(f) for i in row :     print(i[0] , i[2])</pre> <p>The above code is reading first and third column from "data.csv" file</p>	<p>Name Subject</p> <p>Amit CS</p> <p>Sumit IP</p>	<p>['Name' , 'Class' , 'Subject']</p> <p>['Amit' , 'XII' , ' CS']</p> <p>['Sumit' , 'X' , 'IP']</p>



### If you want to skip the First Row (header of the table)

CODING	OUTPUT	Actual Data of File
<pre>import csv f = open("data.csv", 'r') row = csv.reader(f) next(row) for i in row :     print(i[0] , i[2])  next () function will jump to the next row.</pre>	<pre>[ 'Amit' , 'CS' ] [ 'Sumit' , 'IP' ]</pre>	<pre>[ 'Name' , 'Class' , 'Subject' ] [ 'Amit' , 'XII' , ' CS' ] [ 'Sumit' , 'X' , 'IP' ]</pre>
<pre>import csv f = open("data.csv", 'r') row = csv.reader(f) next(row) next(row) for i in row :     print(i[0] , i[2])</pre>	<pre>[ 'Sumit' , 'IP' ]</pre>	<pre>[ 'Name' , 'Class' , 'Subject' ] [ 'Amit' , 'XII' , 'CS' ] [ 'Sumit' , 'X' , 'IP' ]</pre>

If there is any other delimiter other than comma like in the following file

```
id|name|email|salary
1|Amit|amit@mail.com|25000
2|Sumit|sumit@mail.com|30000
3|Ashu|ashu@mail.com|40000
```

CODING	OUTPUT
<pre>import csv f = open("data.csv", 'r') row = csv.reader(f, delimiter = ' ') for i in row :     print(i)</pre>	<pre>['id' , 'name' , 'email' , 'salary'] ['1' , 'Amit' , 'amit@mail.com' , '25000'] ['2' , 'Sumit' , 'sumit@mail.com' , '30000'] ['3' , 'Ashu' , 'ashu@mail.com' , '40000']</pre>

2. **writer() function** : This function help us to write data in a csv file. It accepts the same argument as the reader() function but returns a writer object (i.e \_csv.writer)

There are two main methods used in writing in csv file

1. **writerow()**
2. **writerows()**

Example using writerow()

CODING	DATA.CSV
<pre>import csv f = open("data.csv", 'w') wr = csv.writer(f) wr.writerow(['Name' , 'Class']) wr.writerow(['Amit' , 'XII']) f.close( )</pre>	<pre>Name , Class 'Amit' , 'XII'</pre>

## Example using writerows()

CODING	DATA.CSV
<pre>import csv fields = ['Name' , 'Class' , 'Subject'] rows = [["Amit" , 'XII' , 'CS'] , ['Sumit' , 'X' , 'IP'] , ['Ashu' , 'XI' , 'CS']] f = open("data.csv" , 'w' , newline = ' ') wr = csv.writer(f) wr.writerow(fields) wr.writerows(rows) f.close</pre>	<p><b>Name, Class, Subject</b> <b>Amit, XII, CS</b> <b>Sumit, X, IP</b> <b>Ashu, XI, CS</b></p>

## Reading a CSV file with DictReader:

This function(DictReader) is working similar to reader(). This function return lines as a dictionary instead of list.

CODING	OUTPUT	data.csv
<pre>import csv f = open("data.csv" , 'r') row = csv.DictReader(f) for i in row: print(i)</pre>	<pre>{'Name' : 'Amit, 'Class' : 'XII'} {'Name' : 'Sumit, 'Class' : 'X'} {'Name' : 'Ashu, 'Class' : 'XI'}</pre>	<pre>Name, Class Amit, XII Sumit, X Ashu, XI</pre>



This CSV file has no header. So we have to provide field names via the fieldnames parameter

### **CODING**

```
import csv
f = open("data.csv" , 'r')
row = csv.DictReader(f, fieldnames = ['Name' , 'Class' , 'Subject'])
for i in row:
    print(i)
```

### **OUTPUT**

```
{'Name' : 'Amit', 'Class' : 'XII' , 'Subject' : 'CS'}
{'Name' : 'Sumit', 'Class' : 'X' , 'Subject' : 'IP'}
{'Name' : 'Ashu', 'Class' : 'XI' , 'Subject' : 'CS'}
```

---

**data.csv**

---

**Amit, XII, CS**

**Sumit, X, IP**

**Ashu, XI, CS**

# Binary File in Python

## What is Binary File ?

Binary Files are not in human readable format. It can be read by using some special tool or program.  
Binary files may be any of the image format like jpeg or gif.

## Difference between Binary File and Text File

### **Text File**

**These files can be read by human directly**

**Each line is terminated by a special character ,  
known as EOL**

**Processing of file is slower than binary file**

### **Binary File**

**These files can not be read by humans directly.**

**No delimiter for a line**

**Processing of file is faster than text file**

## Why Binary File in Python?

When ever we want to write a sequence like List or Dictionary to a file then we required binary file in python.

### Steps to work with Binary File in Python

1. **import pickle module.**
2. **Open File in required mode (read, write or append).**
3. **Write statements to do operations like reading, writing or appending data.**
4. **Close the binary file**

## How to write data in Binary File?

Python provides a module named pickle which help us to read and write binary file in python.

Remember : Before writing to binary file the structure (list or dictionary) needs to be converted in binary format. This process of conversion is called Pickling. Reverse process Unpickling happen during reading binary file which converts the binary format back to readable form.

**Q1. Write a function `bwrite()` to write list of five numbers in a binary file?**

test.py - C:/Users/user/AppData/Local/Programs/Python/Python38-32/test.py (3.8.5)

File Edit Format Run Options Window Help

```
def bwrite():  
    import pickle  
    f = open("data.dat", 'wb')  
    d = [1,2,3,4,5]  
    pickle.dump(d, f)  
    f.close()
```

bwrite()

### Output of above program



\*data - Notepad

File Edit Format View Help

€• ]”(K K K K K e.



**Q5. Write a function empadd() which will add a record of employee in a binary file "data.dat" using list. Data to be add include employee name, employee number. Also write a function empread() which will read all the records from the file.**

```
def empadd(): # This function is for adding record to a file
```

```
    f = open("data.dat","wb")
```

```
    import pickle
```

```
    d = [ ]
```

```
    ename=input("Enter employee name")
```

```
    en = int(input("Enter employee number"))
```

```
    temp=[ename, en]
```

```
    d.append(temp)
```

```
    d1 = pickle.dump(d,f) # This line is actually writing content to file
```

```
f.close() #This line breaks the connection of file handle with file
```

```
def empread(): #This function is for reading data from file
```

```
    f=open("data.txt","rb")
```

```
    import pickle
```

```
    d = pickle.load(f)
```

```
    for i in d:
```

```
        print(d)
```

```
    f.close()
```

```
empadd() # This statement is calling a function to add new record
```

```
empread() # This statement is calling a function to read all the records
```

JSON

# What is JSON

- The full form of JSON is Javascript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data. Python supports JSON through a built-in package called JSON. To use this feature, we import the JSON package in Python script. The text in JSON is done through quoted-string which contains the value in key-value mapping within { }. It is similar to the dictionary in Python.

- **Writing JSON to a file in Python**
- Serializing JSON refers to the transformation of data into a series of bytes (hence serial) to be stored or transmitted across a network. To handle the data flow in a file, the JSON library in Python uses dump() or dumps() function to convert the Python objects into their respective JSON object, so it makes it easy to write data to files. See the following table given below.

PYTHON OBJECT	JSON OBJECT
Dict	object
list, tuple	array
str	string
int, long, float	numbers
True	true
False	false
None	null

- **Method 1: Writing JSON to a file in Python using json.dumps()**
- The JSON package in Python has a function called json.dumps() that helps in converting a dictionary to a JSON object. It takes two parameters:
- **dictionary** – the name of a dictionary which should be converted to a JSON object.
- **indent** – defines the number of units for indentation
- After converting the dictionary to a JSON object, simply write it to a file using the “write” function.

```
import json

# Data to be written
dictionary = {
    "name": "sathiyajith",
    "rollno": 56,
    "cgpa": 8.6,
    "phonenumber": "9976770500"
}

# Serializing json
json_object = json.dumps(dictionary, indent=4)

# Writing to sample.json
with open("sample.json", "w") as outfile:
    outfile.write(json_object)
```



Output:

```
{  
  "name": "sathiyajith",  
  "rollno": 56,  
  "cgpa": 8.6,  
  "phonenum": "9976770500"  
}
```

- **Method 2: Writing JSON to a file in Python using json.dump()**
- Another way of writing JSON to a file is by using json.dump() method The JSON package has the “dump” function which directly writes the dictionary to a file in the form of JSON, without needing to convert it into an actual JSON object. It takes 2 parameters:
  - **dictionary** – the name of a dictionary which should be converted to a JSON object.
  - **file pointer** – pointer of the file opened in write or append mode.

```
# to a file
```

```
import json
```

```
# Data to be written
```

```
dictionary = {  
    "name": "sathiyajith",  
    "rollno": 56,  
    "cgpa": 8.6,  
    "phonenumber": "9976770500"  
}
```

```
with open("sample.json", "w") as outfile:  
    json.dump(dictionary, outfile)
```

Output:

```
1 {"name": "sathiyajith", "rollno": 56, "cgpa": 8.6, "phonenumber": "9976770500"}
```

- 
- Reading JSON from a file using Python
- Deserialization is the opposite of Serialization, i.e. conversion of JSON objects into their respective Python objects. The load() method is used for it. If you have used JSON data from another program or obtained it as a string format of JSON, then it can easily be deserialized with load(), which is usually used to load from a string, otherwise, the root object is in a list or Dict.
- **Reading JSON from a file using json.load()**
- The JSON package has json.load() function that loads the JSON content from a JSON file into a dictionary. It takes one parameter:
- **File pointer:** A file pointer that points to a JSON file.

```
import json

# Opening JSON file
with open('sample.json', 'r') as openfile:

    # Reading from json file
    json_object = json.load(openfile)

print(json_object)
print(type(json_object))
```

### Output:

```
E:\MATERIALS\content_writing\JSON>python reading_JSON.py  
{'name': 'sathiyajith', 'rollno': 56, 'cgpa': 8.6, 'phonenumner': '9976770500'}  
<class 'dict'>
```

