

19-10-2024

Training Day – 21

File Handling: To store the data permanently.

We have already discussed about input and print function, in both cases data travels in string format. "Welcome " data will be travelled in streams format

```
# #New Program
# class Customer:
#     pass
# obj=Customer()
# print(type(obj))

class Customer:
    pass
obj1=Customer()    #obj1 of Customer Class
def func1():
    return Customer()
obj2=func1()
```

For file handling: Firstly we need to create an object of **TextIOWrapper** Class

TextIOWrapper Class is made inside io module.

TextIOWrapper in general terms is called file class

```
"""New Program"""
# f=open("D://temp/file1.txt","r")
# print(type(f))    #f is an object of TextIOWrapper Class

# #New Program
# f=open("D://temp/file1.txt","r")    #s="CETPA"
# data=f.read()    #data=s.lower()
# print(data)    #print(data)

# #New Program
# f=open("D://Temp/file1.txt","r")
# data=f.read(5)
# print(data)
# data=f.read(10)
# print(data)
# f.close()
```

Modes of file operations:

Text Modes: String Modes: Character Modes: Here data must be in string format only

r : Read Mode: In Read Mode, if file exists then data will be read from starting of file and if file doesn't exist then error.

w: Write Mode: In write Mode, if file exists then firstly data will be truncated and then data will be written at starting of the file and if file doesn't exist then new file will be created.

a: Append Mode: In append Mode, if file exists then data will be written at the end of the file and if file doesn't exist then new file will be created.

r+: Read and Write Mode

w+: Write and Read Mode

a+: Append and Read mode

Binary Modes: Data must be in binary format

rb: Read Mode Binary: if file exists then data will be read from starting of file and if file doesn't exist then error.

wb: Write Mode Binary: if file exists then data will be written at starting of the file and if file doesn't exist then new file will be created.

ab: Append Mode Binary: if file exists then data will be written at the end of the file and if file doesn't exist then new file will be created.

rb+ : Read and Write Mode Binary

wb+ : Write and Read Mode Binary

ab+ : Append and Read mode Binary

"""

#New Program

s=b"CETPA"

print(type(s))

#New Program

s="CE\nT\P\tA"

print(s)

#New Program

s="CE\\nT\\P\\tA"

print(s)

#New Program

f=open(r"D:\Temp\file1.txt","w")

print(f.read()) #Error

f.close()

#New Program

f=open(r"D:\Temp\file2.txt","rb")

data=f.read()

print(data)

f.close()

os library

#New Program

import os

os.startfile("D:/temp/01 Kal Ho Naa Ho - Sonu Nigam.mp3")

#New Program

import os

os.startfile("D:/temp/Cetpa Video.mp4")

20-10-2024

Training Day – 22

Libraries In Python where the name of library given at installation time is different and while importing, the library name is different. Sometimes new versions of libraries comes with new names.

```
import speech_recognition
```

For installation library name is: SpeechRecognition

```
import speech_recognition
```

DATABASE:

Types of databases: multiple types but two are majorly used:

1. Relational Database
2. Non-relational Database

1. Relational Database: data is stored in the form of tables and tables are connected through some keys.

Examples: MySQL, Oracle, MS-SQL Server, MS-Access, PostGre-SQL

2. Non-relational database: No-sql databases. Data is available in multiple structures

Examples: MongoDB, Cassandra Server...

Relational databases: MySQL Database

For installation: CETPA Video

There are two options to works on database:

1. Directly install the database and start writing the queries.
2. We can install LAMP, WAMP or XAMP Server on our machine and then use database: Here you can work in

Database Connectivity: ie we run the query in python and it should be executed in database

For Database Connectivity: We need to have

1. IP Address: If database is installed in our machine:

Local IP Address of each machine: 127.0.0.1

and this address is called 'localhost'

2. User Id: ie user name

```
user=root
```

3. Password:

```
pwd=root123
```

4. Database name: db1 (Can be given through queries)

5. Port: 3306 (Optional)

```
"""
```

21-10-2024

Training Day – 23

GUI PROGRAMMING

- Pack
- Place
- Grid geometry won't work together.

Grid is the best geometry to use

.....New Program.....

```
# import tkinter as tk
# def leftClick():
#     print("I am Clicked")
# root=tk.Tk()
# root.geometry("300x400")
# btn1=tk.Button(root,text="ABCD",font=1,bg="Red",fg="Yellow",command=leftClick)
# btn1.grid(row=0,column=0)
# root.mainloop()
```

print function in python is used to print the data on console window.

Label widget in tkinter is used to display the data on GUI screen

.....New Program.....

```
# import tkinter as tk
# root=tk.Tk()
# root.geometry("300x400")
# lbl_id=tk.Label(root,text="Enter Cust ID:",font=1)
# lbl_id.grid(row=0,column=0)
# lbl_name=tk.Label(root,text="Enter Cust Name:",font=1)
# lbl_name.grid(row=1,column=0)
# root.mainloop()
```

Entry widget: To take input from GUI screen in single line.

Like input function, we take the data in some variable, similarly in Entry widget we will input the data in some variable. Till now in Python we have discussed, that variables are created in python by assigning the value. Variables created through input functions will always be of type

string. But in GUI tkinter programming, the variables are not automatically created by assigning the value. Here also like in C Lang or Java, we need to firstly define the variable type.

```
# L=list() #Empty List
.....New Program.....
import tkinter as tk
def data_capture():
    id=var_id.get()
    print(id)
    var_id.set("")
root=tk.Tk()

root.geometry("400x500")
lbl_id=tk.Label(root,text="Enter Cust Id:",font=1)
lbl_id.grid(row=0,column=0)
var_id=tk.StringVar()
entry_id=tk.Entry(root,textvariable=var_id,font=1)
entry_id.grid(row=0,column=1)
btn_submit=tk.Button(root,text="Submit",font=1,command=data_capture)
btn_submit.grid(row=1,column=1)
root.mainloop()
```

22-10-2024

Training Day – 23

Daily Diary on Data Analysis Topics

WHAT IS DATA ANALYTICS?

sets to extract meaningful insights and support decision-making. This process helps businesses and individuals identify patterns, trends, and actionable conclusions from raw data.

- **Mathematics and Statistics:** Basics of probability, linear algebra, and hypothesis testing.
- **Programming:** Learn languages like Python which are widely used for data analysis.
- **Data Manipulation and Visualization:** Master libraries like:
 - *Python: Pandas, NumPy, Matplotlib, Seaborn*
 - *R: ggplot2,*

Work on Real-Life Projects

- Start small, such as analyzing public datasets on Kaggle or Google Dataset Search.
- Gradually tackle more complex datasets, like financial records or social media metrics

Tools for Data Analytics

1. **Data Processing and Analysis**
 - **Excel:** Good for small-scale analysis.
 - **Python:** Libraries like Pandas, NumPy, and Scikit-learn.
2. **Data Visualization**
 - **Tableau:** Easy-to-use for creating interactive dashboards.
 - **Power BI:** Microsoft's tool for creating visual reports.
 - **Matplotlib and Seaborn:** Python-based libraries for visualization.

***Topic: Introduction to NumPy Variables**

- Learned about `numpy.ndarray`, its creation, and basic properties.
- Example: Created arrays using `np.array()` and explored their dimensions, shapes, and data types learned Numpy (`np`) library and some in-built functions of numpy.
Functions like -> `astype`, `size`, `ndim`, `dtype`, `shape`, `type()`
- And also practiced about indexing and slicing of arrays in numpy.

```
21oct24.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[4] import numpy as np

[ ] di=[10,20,30]
    ti=[2,4,5]

[ ] a=np.array(di)
    b=np.array(ti)

[ ] sp=a/b
    sp

array([5., 5., 6.])

a=np.array([10,20,30])
print(a,type(a))
print(a.ndim)
print(a.shape)
```

```
21oct24.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[ ] b=np.array([[10,20,30],[40,50,60]])
print(b,type(b))
print(b.ndim)
print(b.shape)

[[10 20 30]
 [40 50 60]] <class 'numpy.ndarray'>
2
(2, 3)

[ ] c=np.array([[10,20,30],[40,50,60],[70,80,90]])
print(c,type(c))
print(c.ndim)
print(c.shape)

[[10 20 30]
 [40 50 60]
 [70 80 90]] <class 'numpy.ndarray'>
3
(3, 3)
```


23-10-2024

Training Day – 24

Topic:- NumPy Manipulation

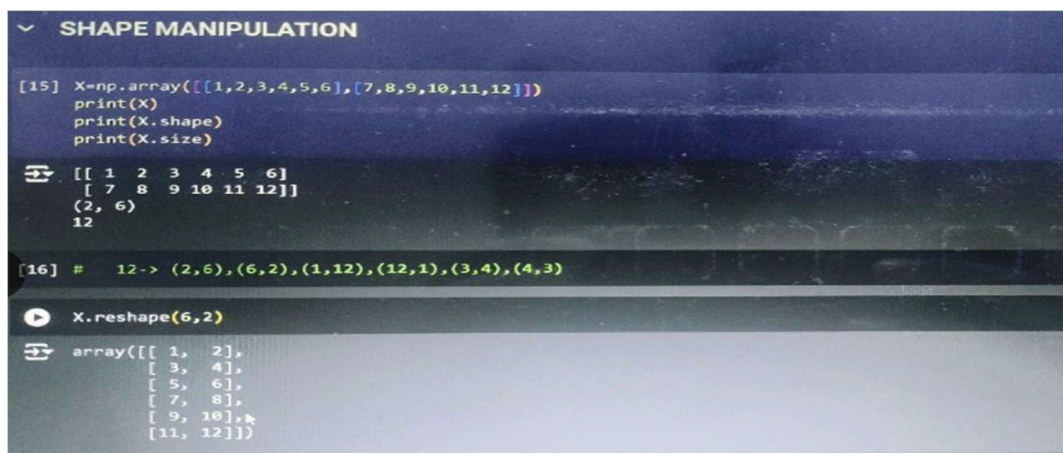
- Performed reshaping, slicing, and broadcasting operations.
- Example: Reshaped a 1D array into a 2D array and performed element-wise multiplication.

Day-2

- Today I practiced some example based on slicing in multi-dimensional array.
- Some new topics also covered in today's session as like -
 - *Shape Manipulation.
 - *Conditions In Array.
- And also getting a short knowledge of "Broadcasting" using numpy's library. NumPy arrays can be reshaped, sliced, and indexed to suit various needs.

EXAMPLE

```
array = np.arange(10)
reshaped = array.reshape(2, 5)
sliced = reshaped[:, 1:4]
broadcasted = reshaped + 10
print("Original Array:\n", array)
print("Reshaped Array:\n", reshaped)
print("Sliced Array:\n", sliced)
print("Broadcasted Array:\n", broadcasted)
```



The screenshot shows a Jupyter Notebook interface with a dark theme. The title bar at the top says 'SHAPE MANIPULATION'. The code cell [15] contains the following code:

```
X=np.array([[1,2,3,4,5,6],[7,8,9,10,11,12]])
print(X)
print(X.shape)
print(X.size)
```

The output of the code is displayed below the code cell:

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
(2, 6)
12
```

The next code cell [16] contains the following code:

```
# 12 -> (2,6), (6,2), (1,12), (12,1), (3,4), (4,3)
```

The output of the code is displayed below the code cell:

```
X.reshape(6,2)
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])
```