**OXFORD ENGLISH SCHOOL**

**CHIDAMBARAM, THILLAINAYAGAPURAM-608102**

****

**CLASS XII**

**DEPARTMENT OF COMPUTER SCIENCE**

**PROJECT NAME:**

**COVID DATA VISUALISER**

**ACADEMIC YEAR**

**2021-2022**

**OXFORD ENGLISH SCHOOL**

**(Affiliated to Central Board of Secondary Education,New Delhi)**

**(Chidambaram, Thillainayagapuram-608102)**

****

**BONAFIDE CERTIFICATE**

Certified to be the Bonafide Project work in “**Computer Science”** done by ­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_of class XII the school during the academic year 2021-22

**Dated: HOD**

**Register No.**

**Submitted for All India Senior Secondary Practical Examination in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ held on\_\_\_\_\_\_\_\_**

**at Oxford English School(CBSE), Chidambaram-608001**

**Principal Internal Examiner External Examiner**

**ACKNOWLEDGEMENT**

*I owe my sincere thank to the Chairman, Vice – Chairman and the Principal for their encouragement to work on this project, which helped us to enhance our knowledge in the topic chosen. I would like to thank our Computer Science teacher for the guidance and immense support in completing this project.*

Signature of the Student

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| S.No | Particulars | Page No. |
| **1** | Python-Introduction |  |
| **2** | Features of Python |  |
| **3** | Various Applications of Python |  |
| **4** | Python Tokens |  |
| **5** | Python Functions |  |
| **6** | File Handling in Python |  |
| **7** | Covid Data Visualiser |  |
| **8** | Conclusion |  |
| **9** | Bibliography |  |

**PYTHON PROGRAM LANGUGE-INTRODUCTION**

Python is a widely used general purpose, high level programming language. It was created by **“Guido van Rossum”** in 1991 and further developed by the Python Software Foundation.

It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions: **Python 2 and Python 3**. Both are quite different.

**Beginning with Python programming:**

1. **Finding an Interpreter:**

Before we start python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like

[**https://ide.geeksforgeeks.org/,http://ideone.com/**](https://ide.geeksforgeeks.org/,http://ideone.com/) **or** [**http://codepad.org/**](http://codepad.org/) that can be used to run Python programs without installing an interpreter.

**Windows:** There are many interpreters available freely to run Python scripts like IDLE(Integrated Development Environment) that comes bundled with the Python software downloaded from [**http://python.org/**](http://python.org/)**.**

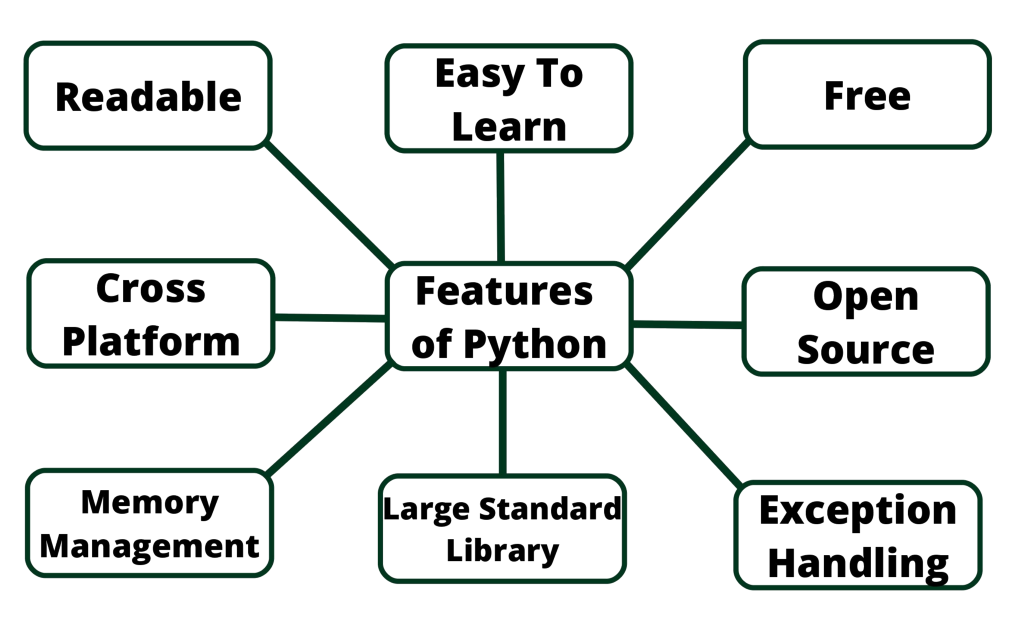
**Linux:** Python comes preinstalled with popular Linux distros such as Ubuntu and Fedora. To check with version of Python you’re running, type “python” in the terminal emulator. The interpreter should start and print the version number.

**macOS:** Generally, Python 2.7 comes bundled with macOS. You’ll have to manually install Python 3 from [**https://python.org/**](https://python.org/)**.**

**Interesting Fact:**

Python is named after the comedy television show Monty Python’s Flying Circus. It is not named after the Python snake.

**Features of Python Programming Language:**



**1. Readable**: Python is a very readable languge.

**2. Easy to Learn:** Learning pyton is easy asthis is a expressive and high level programming language, which means it is easy to understand the language and thus eay to learn.

**3. Cross Platform:** Python is available and can run on various operating systems such as Mac, Windows, Linux, Unix etc. This makes it a cross platform and portable language.

**4. Open Source**: Python is a open source programming language.

**5. Large standard library:** Python comes with a large standard library that has some handy codes and functions which we can use while writing code in python.

**6. Free:** Python is free to download and use. This means you can download it for free and use it in your application. **See:** Open Sorce Python License. Python is an example of a FLOSS(Free/Libre Open Source Software),which means you can freely distribute copies of this software, read its source code and modify it.

**7. Supports exception handling:** If you are new, you may wonder what is an exception? An exception is an event that can occur during program exception and can disrupt the normal flow of program. Python supports exception handling which means we can write less error prone code and can test various scenarios that can cause an exception later on.

**8. Advanced features:** Supports generators and list comprehensions. We will cover these features later.

**9. Automatic memory management:** Python supports automatic memory management which means the memory is cleared and freed automatically. You do not have to bother clearing the memory.

**What Can You Do with Python?**

You may be wondering what all are the applications of Python. There are so many applications of python. Here are some of them.

**1. Web development** - Web framework like Django and Flask are based on Python. They help you write server side code which helps you manage database, write backend programming logic, mapping urls etc.

**2. Machine learning** - There are many machine learning applications written in Python. Machine learning is a way to write a logic so that a machine can learn and solve a particular problem on its own. For example, products recommendation in websites like Amazon, Flipkart, eBay etc. is a machine learning algorithm that recognizes user's interest. Face recognition and Voice recognition in your phone is another example of machine learning.

**3. Data Analysis** - Data analysis and data visualization in form of charts can also be developed using Python.

**4. Scripting** - Scripting is writing small programs to automate simple tasks such as sending automated response emails etc. Such type of applications can also be written in Python programming language.

**5. Game development** - You can develop games using Python.

**6.** You can develop Embedded applications in Python.

**7. Desktop applications** - You can develop desktop application in Python using library like TKinter or QT.

**Python Tokens**

Tokens a very basic component of the source code. Characters are classified by four major categories:

**1. Keyword**

**2. Identifier**

**3. Literal**

**4. Operator**

**Keyword:**

There are some Reserved Keywords available for Python. It is not used for the Python variable and value to be assigned to the variable cannot be a keyword name.

|  |  |  |  |
| --- | --- | --- | --- |
| and | def | False | import |
| not | True | as | del |
| finally | in | or | try |
| assert | elif | for | is |
| pass | while | break | else |
| from | lambda | print | with |
| class | except | global | None |
| raise | yield | continue | If |
| nonlocal | return | exce |  |

**IDENTIFIERS:**

Identifiers are names that you give to a variable, Class, or Function. There are certain rules for naming identifiers similar to the variable declaration rules, such as:

*No special character except \_ (underscore is used as an identifier)*

Identifiers are names that you give to a variable, Class, or Function. There are certain rules for naming identifiers similar to the variable declaration rules, such as:

*No special character except \_ (underscore is used as an identifier)*

* Keywords are not used as identifiers.
* The first character of an identifier should be underscored or a character, but a number is not valid for identifiers.

• As we discussed, python is case sensitive. Therefore, if we declared Python and python, both are different identifier names.

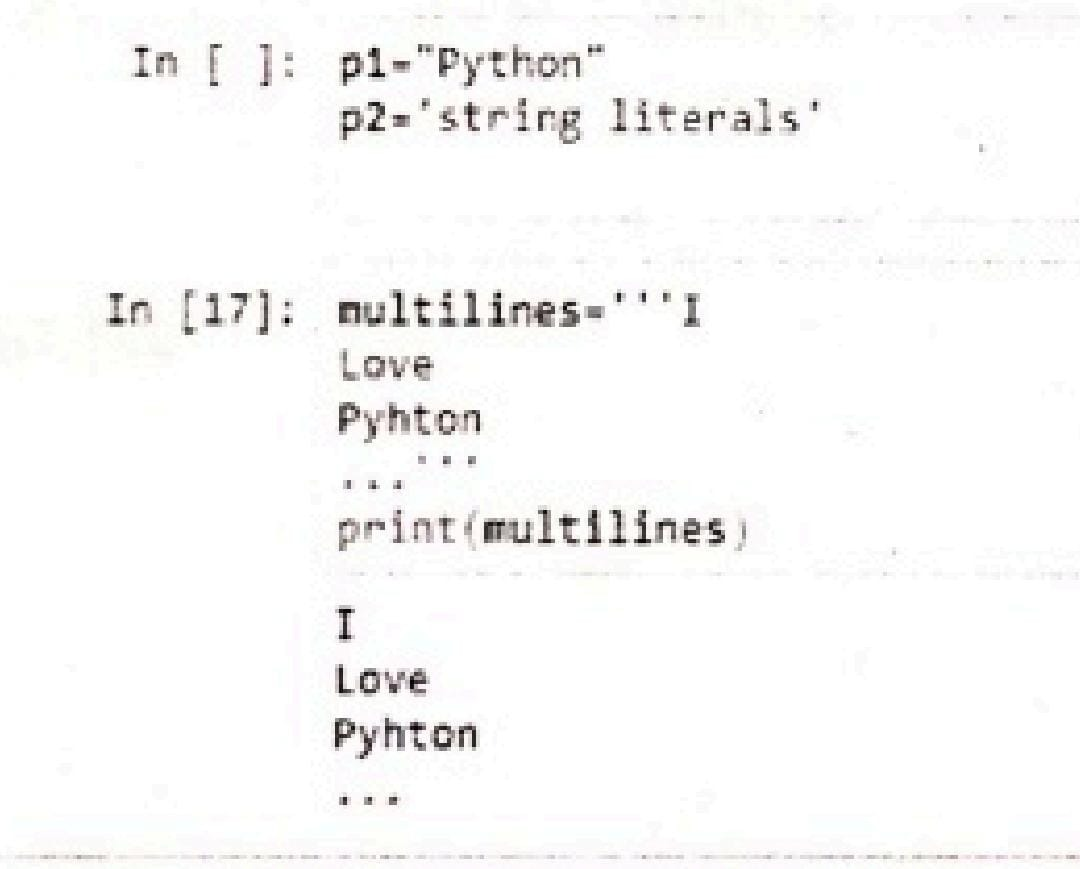
**LITERALS:**

Literals are the data given in a variable or a constant. There are four basic types of literals available in Python:

**String Literals:**

These literals are formed by enclosing

quotes are used.



In the first line, p1 and p2 are declared in quotation marks with both single and double quotes respectively.

On the other hand, if we want to print something in pre-text format, then use triple quotes:’’’. Multiline identifiers print the text written between triple quotations.

**Numeric Literals**

Int, Long, Float, and Complex are different ways of storing numbers in Python. There is no special arrangement for storing large numbers in Python. There is no restricted bit for storing Integer number value; it can be stored up to the memory limit in Python.

**Boolean Literals**

Boolean literals have just two values: True or False. Remember that you cannot use them as identifiers.

**Special Literals**

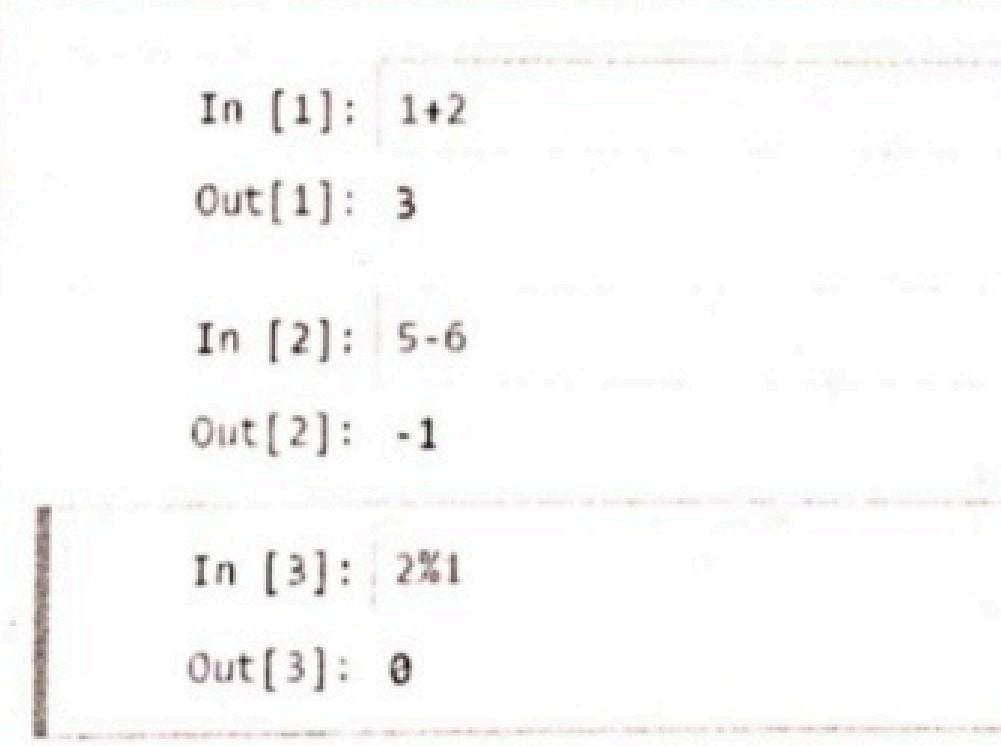
Python has just a single special literal which is NONE, which specifies that a field is not created.

**OPERATORS:**

These are specific characters which need to perform some specific task according to the functions. There are 7 types of operators available for Python.

**Arithmetic Operator**

There are two operands to perform the arithmetic operation For example 1+2,1 and 2 are operands and + is an arithmetic operator. There are different arithmetic operators used, such as +,-, /,\*,%

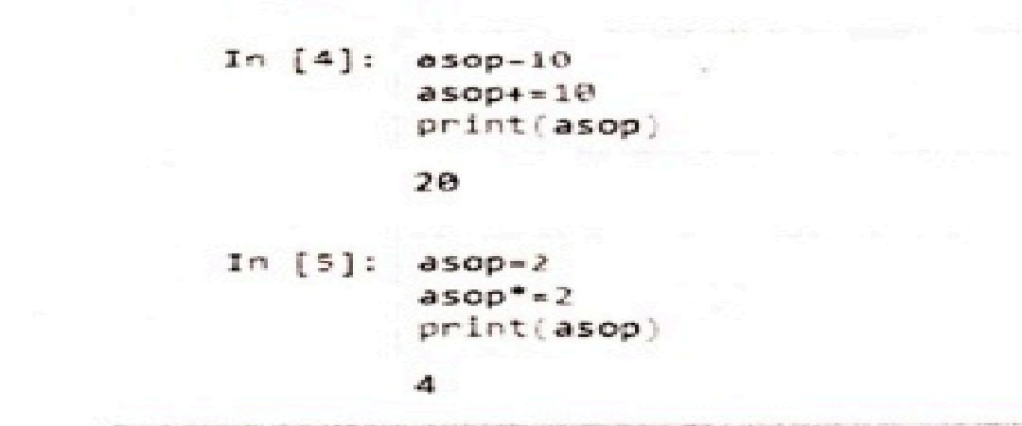
**Example **

**Assignment Operator**

This is used to assign a value to a variable. The different assignment operators are,

=,+= ,-= ,\*=

**Example**



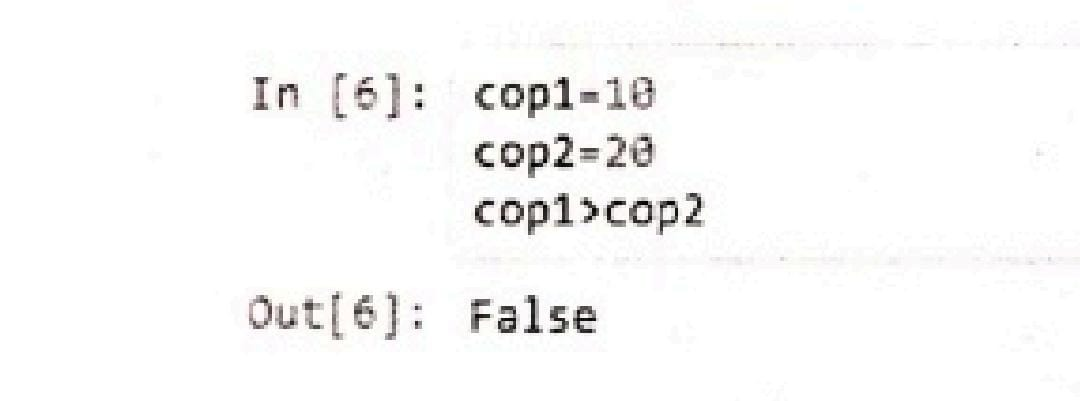
Here, asop is a variable assigned by 10. Then we write asop+= 10; which means asop=asop+10; which is 10+10=20. So, we get an output of 20. The same for the next

one, asop=asop\*2;282=4. So, the output is 4 while we use \*= this assignment operator.

**Comparison Operator**

The comparison operator is used to compare two values and return output in the form of TRUE or FALSE. There are different comparison operators, such as < ,>,<=, >= ,!=

**Example**



**Logical Operator**

Logical operators perform logical operations and return the values TRUE or FALSE.

Logical operators include these three keywords: **and, or, not.**

**Example**

****

**Bitwise Operator**

These are the bitwise operators:&,l,^,<<,>>,~.

**Example**

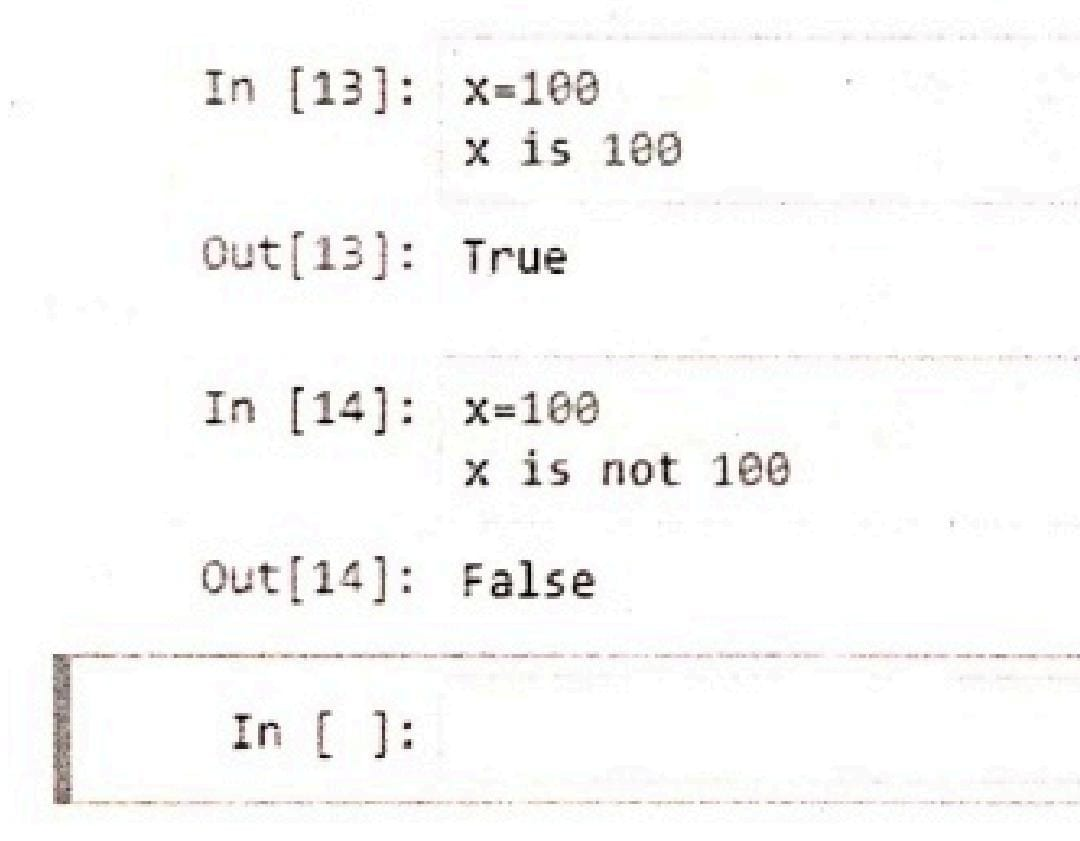


Here, all the conversions are done on the bases of Binary to Decimal bit conversions. You can check the answers through the scientific calculators.

**Identity Operator**

It is used to share identity to test two operands. There are two operators used: is, is not.

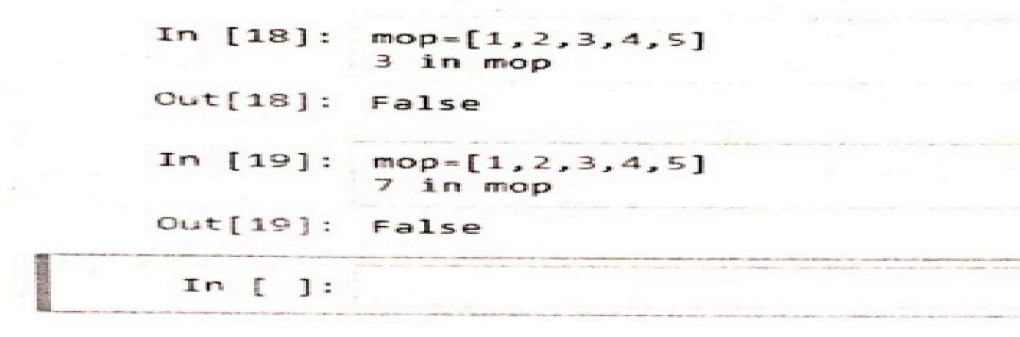
**Example**



It's just used to check if operands are working properly or not.

**Membership Operator**

This operator is used to check if the member of the value is a sequence or not. It means sequence is list, almost the same as an array, tuple, or a string. These operators include: in, in not.

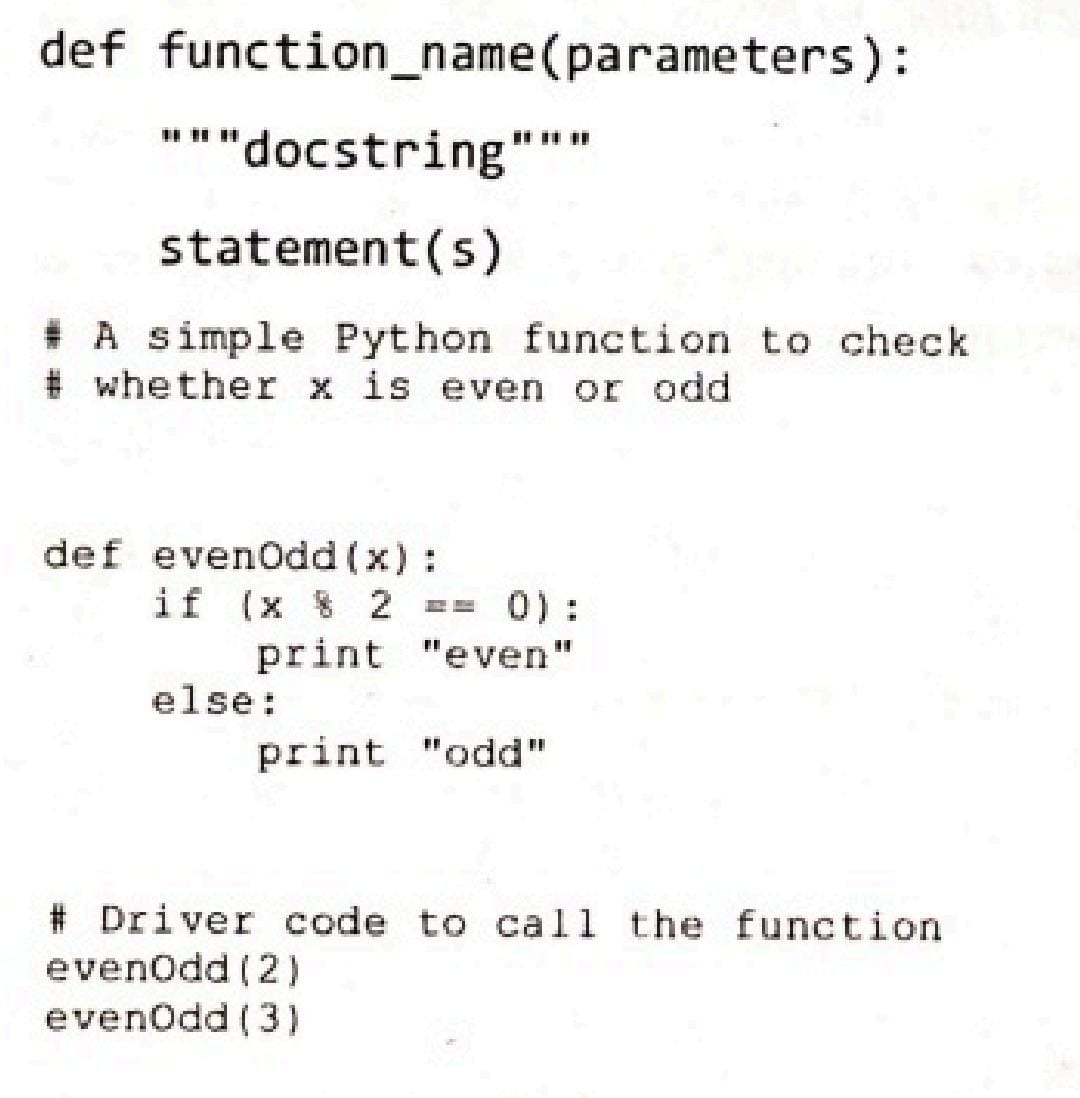
**Example**

**FUNCTIONS:**

A function in Python is an aggregation of related statements designed to perform a computational, logical, or evaluative task. The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call the function to reuse code contained in it over and over again.

Functions can be both built-in or user-defined. It helps the program to be concise, non-repetitive, and organized.

**Syntax:**

****

**OUTPUT:**

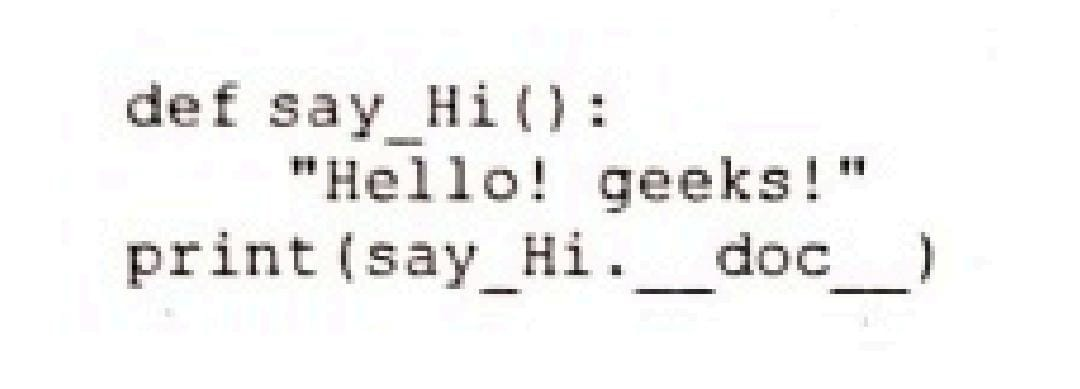
even

odd

**Docstring :**

The first string after the function is called the Document string or Docstring in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice. The below syntax can be used to print out the docstring of a function:

***Syntax:*** *print(function\_name.\_\_doc\_\_)*



**OUTPUT:**

Hello! geeks!

**The return statement**

The return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.

***Syntax:*** *return [expression\_list]*

The return statement can consist of a variable, an expression, or a constant which is returned to the end of the function execution. If none of the above is present with the return statement a None object is returned.

**Example:**



**OUTPUT:**

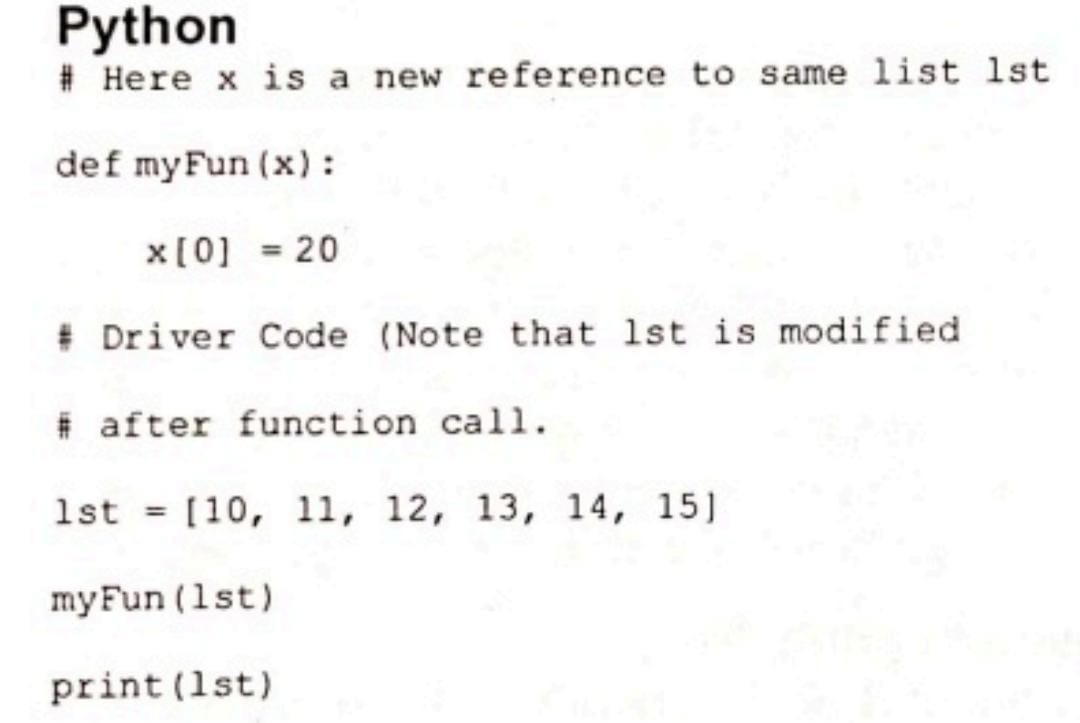
4

16

**Pass by Reference or pass by value?**

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.

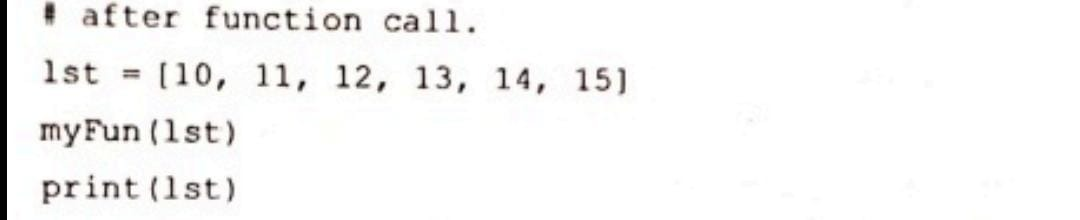
**Example:**



**OUTPUT:**

[20,11,12,13,14,15]

When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken. For example, consider the below program.



**OUTPUT:**

[10,11,12,13,14,15]

**FILE HANDLING IN PYTHON:**

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short. Python treats file differently, as text or binary and this is important. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let's start with Reading and Writing files.

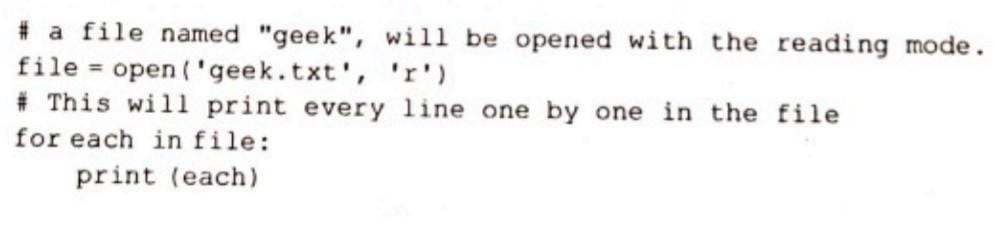
**Working of open() function**

We use **open()** function in Python to open a file in read or write mode. As explained above, **open()** will return a file object. To return a file object we use **open()** function along with two arguments, that accepts file name and the mode, whether to read or write.

*So, the syntax being:* ***open(filename, mode****).* There are three kinds of mode, that Python provides and how files can be opened:

* **"r”**, for reading.
* **"w"**, for writing.
* **"a",** for appending.
* **"r+"** , for both reading and writing

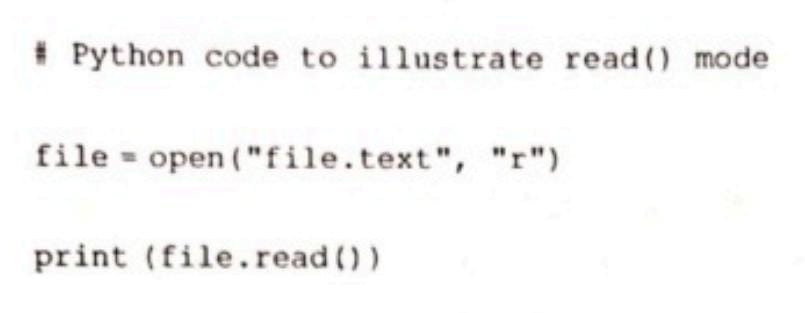
One must keep in mind that the mode argument is not mandatory. If not passed,

then Python will assume it to be "r" by default. Let's look at this program and try to analyze how the read mode works:The open command will open the file in the read mode and the for loop will print each line present in the file.

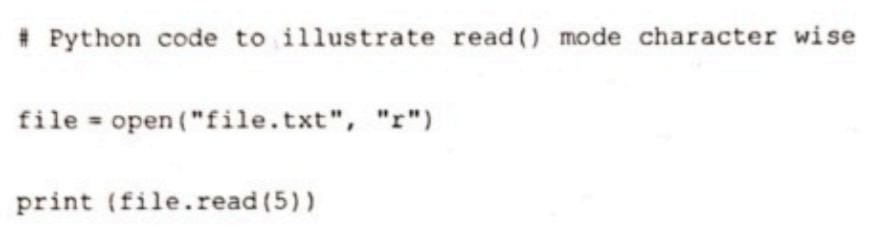
**Working of read() mode**

There is more than one way to read a file in Python. If you need to extract a string that contains all characters in the file then we can use file.read(). The full code would work

like this:

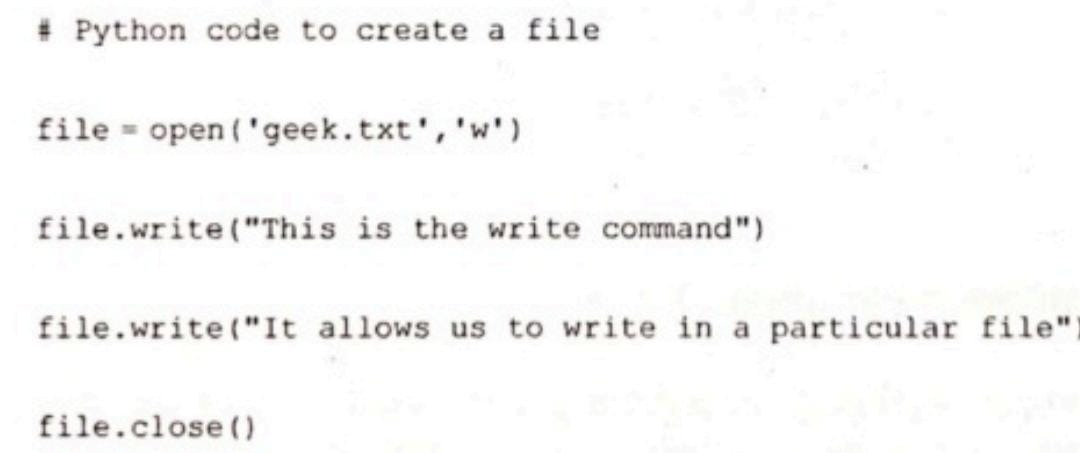
Another way to read a file is to call a certain number of characters like in the following

code the interpreter will read the first five characters of stored data and return it as a

string:

**Creating a file using write() mode**

Let's see how to create file and how write mode works. To manipulate the file, write the following in your Python environment:



The close() command terminates all the resources in use and frees the system of this particular program.

**Working of append() mode**

Let's see how the append mode works:

# Python code to illustrate append () mode

file = open(‘geek.txt’ ,’a’)

file . write(“This will add this line”)

file . close()

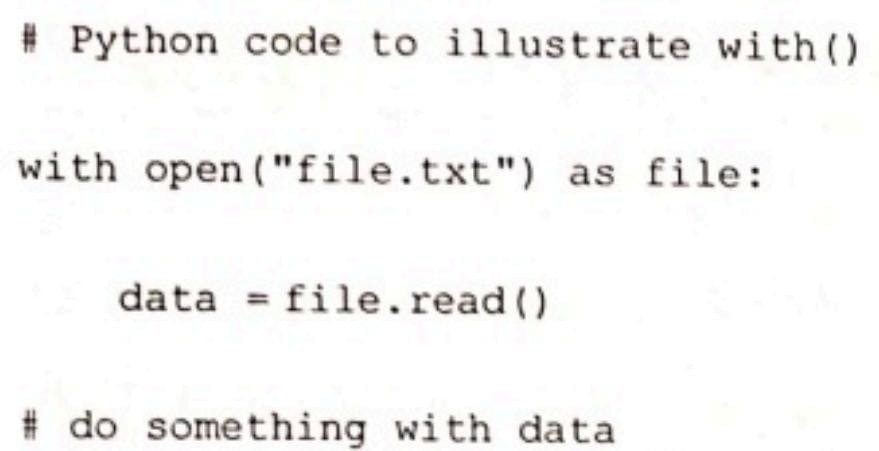
There are also various other commands in file handling that is used to handle various tasks like:

**rstrip():** This function strips each line of a file off spaces from the right-hand side.

**Istrip():** This function strips each line of a file off spaces from the left-hand side.

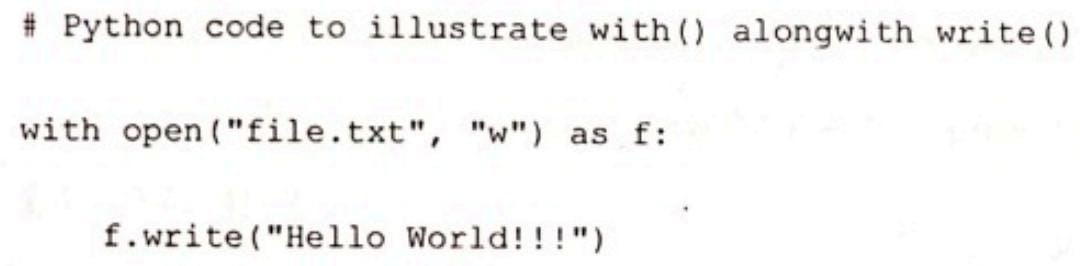
It is designed to provide much cleaner syntax and exceptions handling when you are working with code. That explains why it's good practice to use them with a statement where applicable. This is helpful because using this method any files opened will be closed automatically after is done, so auto-cleanup.

**Example:**



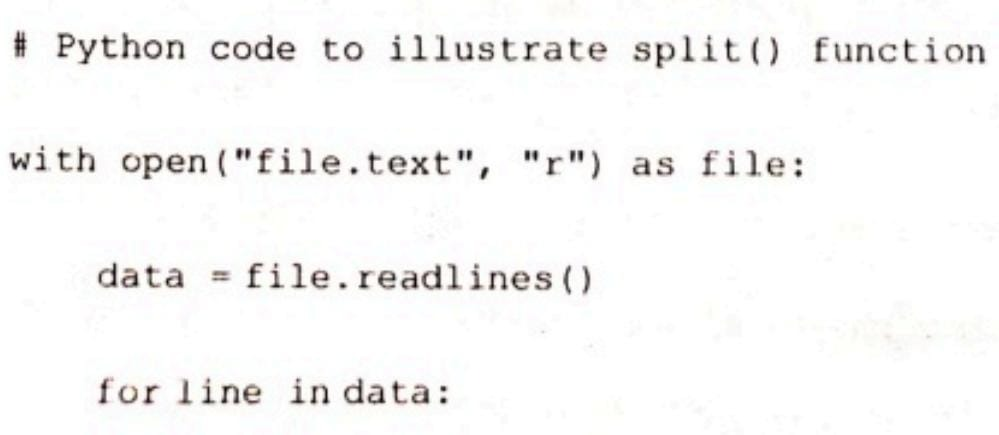
**Using write along with with() function**

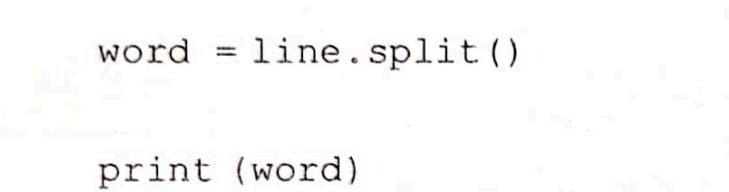
We can also use write function along with with() function:



**split() using file handling**

We can also split lines using file handling in Python. This splits the variable when space is encountered. You can also split using any characters as we wish. Here is the code:





There are also various other functions that help to manipulate the files and its contents. One can explore various other functions in Python Docs.

Python can also handle csv(comma separated files). Though this can be done through normal file handling functions, it is much easier to handle them with the inbuilt csv module of python.

To get better control of the data, modules like pandas can be used to handle csv files. This module is used in the project and serves as a powerful tool in extracting data from csv files

PROJECT

Covid data visualiser

The pandemic has had a massive effect on our quality of life and have brought out many flaws in our method of handling contagious diseases. Analysis of data from this period is crucial to prevent these mishaps and to avoid pandemics of this dimension in the future

This program has been written with this fact in the mind and visualises covid related statistics from January 2020 to December 2021. This program plots 10 graphs with data that has been imported from a csv file.

The graphs have been plotted using the matplotlib library of python. The data is obtained from the csv files using the pandas library.

The program has 10 functions with each function plotting a graph and displaying it when called. The user is presented with a list of graphs with their corresponding types and numbers. The user is asked to enter the number of the graph they wish to see.

The graph is then displayed in a separate window and runs till the user manually exits it. When the new graph window is closed, the program automatically runs again and displays the options.

The user is asked to press e/E to exit the program . This has been made possible by the sys module of python. If e/E is pressed, a thank you message is displayed and the program is closed.

All the error cases have been covered at the end of the program. If the user enters a number not present in the list or enters a string other than e/E, they are asked to enter a valid input and the program runs again.

The following graphs can be plotted by the program:

1. Covid cases by countries [Bar Graph]
2. Covid cases by countries per year [Double Bar Graph]
3. Covid deaths by countries [Bar Graph]
4. Percentage of global covid cases by countries [Pie Graph]
5. Covid cases by states [Bar Graph]
6. Covid deaths by states [Bar Graph]
7. Percentage of covid cases in India by states [Pie Graph]
8. Covid death rate by states [Horizontal Bar Graph]
9. Covid cases and discharges by states [Double Bar Graph]
10. Covid cases vs population by states [Scatter Plot]

The following libraries were used:

* Matplotlib: Used to plot graphs from data
* Pandas: Used to derive data from csv files
* Numpy: Used to create index lists to be plotted
* Random: Used to shuffle colours of the graph
* Sys: Used to implement the exit functionality of the program

The data and the csv files were obtained from the following sources:

* Datasets from Kaggle
* Data from worldometer.com
* Data from ourworldindata.com

CODE

from matplotlib import pyplot as plt  
import numpy as np  
import pandas as pd  
import random  
import sys  
  
print("Covid data visualiser")  
print("Note: This program uses data obtained in Decmeber 2021")  
  
def casecountries():  
 *# covid cases by countries as of dec 2021 top  
 # source kaggle dataset  
  
 #declaring the style and a colour list for later use* plt.style.use("seaborn")

colours = ["#bc5090","#bc5090","#bc5090","#ff6361","#ff6361","#ff6361","#ff6361","#ffa600","#ffa600","#ffa600"]  
  
 *#reading specific coloumns from a csv file* data = pd.read\_csv("data6.csv")  
 x = list(data["Country"])  
 y = list(data["Total Cases"])  
  
 *#converting the lists to a dictionary and sorting them by value* d = dict(zip(x,y))  
 d = sorted(d.items(), key = lambda x:x[1], reverse=True)  
 d = d[0:10]  
 data\_x = []  
 data\_y = []  
 for t in d:  
 data\_x.append(t[0])  
 data\_y.append(t[1])  
  
 *#using numpy to create a list of indexes to be plotted* indexes\_x = np.arange(len(data\_x))  
  
 *#plotting a bar graph* plt.bar(indexes\_x,data\_y, width=0.5, color=random.choice(colours),label= "Covid cases")  
  
 *#specifying some properties of the graph* plt.xticks(ticks=indexes\_x, labels=data\_x)  
 plt.title("Countries with the highest covid cases as of December 2021")  
 plt.xlabel("Countries")  
 plt.ylabel("Number of cases in crores")  
  
  
 *#adding credit* plt.text(7,47000000,"\*Data obtained from kaggle dataset", fontdict={"family":"serif","size":8})  
  
 *#using tight layout when encountering formatiing issues  
 #plt.tight\_layout()* plt.legend()  
 plt.show()  
  
  
  
def casecountries\_year():  
 *#covid cases by countries seperated by year  
 #data inputted from ourworldindata.com manually  
  
  
 #declaring the style and a colour list for later use* plt.style.use("seaborn")

colours = ["#bc5090","#bc5090","#bc5090","#ff6361","#ff6361","#ff6361","#ff6361","#ffa600","#ffa600","#ffa600"]  
  
 *#manually declaring data from ourworldindata.org* data\_x = ["United states", "India", "Brazil", "UK", "Russia", "Turkey", "France","Germany", "Iran", "Spain"]  
 data\_2020 = [20160000,10290000,7680000,2490000,3130000,2210000,2660000,1750000,1230000,1930000]  
 data\_2021 = [32840000,24500000,14560000,9820000,7100000,7150000,6690000,5300000,4960000,4100000]  
  
 *#using numpy to create a list of indexes to be plotted* index\_x = np.arange(len(data\_x))  
  
 *#specifying a width to plot multiple bars in the same x point* width = 0.25  
  
 *#plotting the bar graphs  
 #one of them is displaced by a specified width value* plt.bar(index\_x,data\_2020,width=width,label="2020",color="#003f5c")  
 plt.bar(index\_x+width,data\_2021,width=width,label="2021",color="#bc5090")  
  
 *#adding credit*

plt.text(7.8,30000000,"\*Data obtained from ourworldindata.org", fontdict={"family":"serif","size":8})  
  
 *#specifying properties* plt.xticks(ticks=index\_x, labels=data\_x)  
 plt.ylabel("Number of cases in crores")  
 plt.xlabel("Countries")  
 plt.title("Number of covid cases by year")  
  
  
  
 plt.legend()  
 plt.show()  
  
  
  
def casestates():  
  
 *#covid cases by state  
 #data from kaggle dataset  
  
 #declaring the style* plt.style.use("seaborn")  
  
 *#reading specific coloumns from a csv file* data = pd.read\_csv("data4.csv")  
 x = list(data["State/UTs"])  
 y = list(data["Total Cases"])  
  
 *#converting the lists to a dictionary and sorting them by value* d = dict(zip(x,y))  
 d = sorted(d.items(),key = lambda x:x[1], reverse=True)  
 d = d[0:10]  
  
 l1 = []  
 l2 = []  
  
 for t in d:  
 l1.append(t[0])  
 l2.append(t[1])  
  
 *#using numpy to create a list of indexes to be plotted* index\_x = np.arange(len(l1))  
  
 *#plotting the graph* plt.bar(index\_x,l2,width=0.5, label = "Covid cases", color="#bb0055")  
  
  
 *#specifying some properties* plt.xticks(ticks=index\_x, labels=l1)  
 plt.title("Covid cases by states")  
 plt.ylabel("Covid cases in lakhs")  
 plt.xlabel("States")  
  
 *#adding credit* plt.text(7.3,6000000,"\*Data obtained from kaggle dataset", fontdict={"family":"serif","size":8})  
  
 plt.legend()  
 plt.show()  
  
  
  
  
def deathcountries():  
  
 *#covid deaths by countries  
 #data from kaggle dataset  
  
 #declaring the style* plt.style.use("seaborn")  
  
 *#reading specific columns from a scv file* x = []  
 y = []  
 data = pd.read\_csv("data6.csv")  
 x = list(data["Country"])  
 y = list(data["Total Deaths"])  
  
  
 *#converting the lists to dictionary and sorting them by values* d = dict(zip(x, y))  
 d = sorted(d.items(), key=lambda x: x[1], reverse=True)  
 d = d[0:10]  
 data\_x = []  
 data\_y = []  
 for t in d:  
 data\_x.append(t[0])  
 data\_y.append(t[1])  
  
  
 *#plotting the graph* plt.bar(np.arange(len(data\_x)), data\_y, label="Deaths", color="#ffa600")  
  
 *#specifying some properties* plt.xticks(ticks=np.arange(len(data\_x)), labels=data\_x)  
  
 plt.ylabel("Deaths")  
 plt.title("Covid deaths by countries")  
 plt.xlabel("Countries")  
  
  
 *#adding credits* plt.text(8, 800000, "\*Data obtained from Worldometer.org", fontdict={"family": "serif", "size": 8})  
  
 plt.legend()  
 plt.show()  
  
  
def piecasescountries():  
  
 *#pie chart on covid cases by countries  
 #data inputted manually from worldometer.com  
  
  
 #manually inputting data* data\_x = ["United states", "India", "Brazil", "UK", "Russia", "Turkey", "France","Germany", "Iran", "Spain"]

data\_y = [53222424,34793333,22239436,11891292,10415230,9307124,9116068,7009634,6184762,5718007]  
  
  
 *#converting raw numbers to percentages* total\_cases = 285000000  
 percent = []  
 for i in data\_y:  
 a = (i/total\_cases)\*100  
 percent.append(a)  
  
  
 *#declaring a colour set*

colours =["#003F5C","#58508D","#BC5090","#FF6361","#FFA600","#6050DC","#D52DB7","#FF2E7E","#FF6B45","#FFAB05"]  
 random.shuffle(colours)  
  
 *#declaring a explode list to highlight a specific part of the graph* explode = [0,0.09,0,0,0,0,0,0,0,0]  
  
 *#plotting the pie chart* plt.pie(percent,labels=data\_x,wedgeprops={"edgecolor": "black"}, autopct="%1.2f%%",explode = explode,shadow=True,colors=colours)

plt.title("Covid cases by countries")  
  
  
 *#plt.legend()* plt.show()  
  
  
  
def deathstates():  
  
 *#covid deaths by states  
 #data from kaggle dataset  
  
  
  
 #declaring the style* plt.style.use("seaborn")  
  
  
 *#reading specific columns from a csv file* data = pd.read\_csv("data4.csv")  
 x = data["State/UTs"]  
 y = data["Deaths"]  
 states = list(x)  
 deaths = list(y)  
  
 *#converting the lists to a dictionary and sorting them* d = {}  
  
 for i,j in zip(states,deaths):  
 d[i] = j  
  
 f = sorted(d.items(),key=lambda x:x[1],reverse=True)

mydata = f[0:10]  
 data\_x = []  
 data\_y=[]  
  
 for t in mydata:  
 data\_x.append(t[0])  
 data\_y.append(t[1])  
  
  
 *#declaring a index list to be plotted* index\_x = np.arange(len(data\_x))  
  
 *#plotting the graph* plt.bar(index\_x,data\_y, label="Deaths", color = "#58508D")  
  
  
 *#specifying some properties* plt.xticks(ticks=index\_x,labels=data\_x)  
 plt.title("Covid death by states")  
 plt.ylabel("Deaths")  
 plt.xlabel("States/UTs")  
  
 plt.legend()  
 plt.show()  
  
  
  
  
def piecasesstates():  
 *#pie chart on covid cases by states  
 #data from kaggle dataset  
  
  
 #declaring the style  
 #plt.style.use("seaborn")  
  
 #reading specific columns from a csv file* data = pd.read\_csv("data4.csv")  
 x = data["State/UTs"]  
 y = data["Total Cases"]  
  
 *#converting the dictionary* d = dict(zip(x,y))

d = sorted(d.items(),key = lambda x:x[1], reverse=True)  
 d = d[0:10]  
  
 states=[]  
 cases=[]  
 for t in d:  
 states.append(t[0])  
 cases.append(t[1])  
  
  
 *#converting raw data to percentages* total\_cases = 34800000  
  
 percent = []  
 for i in cases:  
 a = (i/total\_cases)\*100  
 percent.append(a)  
  
  
 *#declaring a colour list* colours =["#003F5C","#58508D","#BC5090","#FF6361","#FFA600","#6050DC","#D52DB7","#FF2E7E","#FF6B45","#FFAB05"]

random.shuffle(colours)  
  
  
 *#plotting a pie chart*

plt.pie(percent, wedgeprops={"edgecolor":"black"}, autopct="%1.2f%%",colors=colours,shadow=True,labels=states)  
  
 plt.title("Covid cases by states")  
  
 plt.show()  
  
  
  
def deathratebystates():  
  
 *#barh death rate by state  
 #data from kaggle dataset  
  
  
  
 #declaring the style* plt.style.use("seaborn")  
  
 *#reading specific columns from a csv file* data = pd.read\_csv("data4.csv")  
 x = list(data["State/UTs"])  
 y = list(data["Death Ratio"])  
  
  
 *#converting the lists into dictionary and sorting them by value* dic = dict(zip(x,y))  
  
 dic = sorted(dic.items(), key = lambda x:x[1], reverse=True)  
 dic = dic[0:10]  
 dic.reverse()  
  
 states = []  
 deathratio = []  
  
 for t in dic:  
 states.append(t[0])  
 deathratio.append(t[1])  
  
  
 *#declaring an index list to plot*

indexes = np.arange(len(states))  
  
  
 *#plotting a horizontal bar graph*

plt.barh(indexes,deathratio,color= "#6050DC",label = "Death rate")  
  
 *#specifying some properties* plt.yticks(ticks=indexes,labels=states)  
 plt.xlabel("Death rate")  
 plt.title("Death rate by states")  
  
 plt.text(2.2, 8, "\*Data obtained from Worldometer.org", fontdict={"family": "serif", "size": 8})  
  
  
 plt.legend()  
 plt.show()  
  
  
def casesdischargedstates():  
  
 *#graph on cases/discharged  
 #data from kaggle dataset  
  
 #declaring the style* plt.style.use("seaborn")  
  
  
 *#reading sepcific columns from a csv file* data = pd.read\_csv("data4.csv")  
 states = data["State/UTs"]  
 cases = data["Total Cases"]  
 dis = data["Discharged"]  
  
  
 *#converting them into a nested list(tuple) and sorting them* x = list(zip(cases,states,dis))  
 y = sorted(x,reverse=True)  
  
 data1 = []  
 data2 = []  
 data3 = []  
  
  
 for i in y:  
 data1.append(i[0])  
 data2.append(i[1])  
 data3.append(i[2])  
  
 data1 = data1[0:9]  
 data2 = data2[0:9]  
 data3 = data3[0:9]  
  
  
 *#creating a index list for plotting* indexes = np.arange(len(data2))  
  
 *#plotting the bar graphs with one slightly displaced* plt.bar(indexes,data1,width=0.25,label = "Total cases",color="#BC5090")  
 plt.bar(indexes+0.25,data3,width=0.25,label="Discharged",color="#FFAB05")  
  
 *#specifying some properties* plt.legend()  
 plt.xticks(ticks=indexes,labels=data2)  
 plt.ylabel("Cases/Discharged in lakhs")  
 plt.xlabel("States")  
 plt.title("Total cases and discharges by state")  
  
 plt.text(6, 5000000, "\*Data obtained from Worldometer.org", fontdict={"family": "serif", "size": 8})  
  
  
 plt.show()  
  
  
  
def casespopulationstates():  
  
 *#graph on cases vs population(scatter plot)  
  
  
  
 #declaring the style* plt.style.use("seaborn")  
  
 *#reading specific columns from a csv file* data = pd.read\_csv("data4.csv")  
 states = data["State/UTs"]  
 cases = data["Total Cases"]  
 pop = data["Population"]  
  
  
 *#converting the data to nested list(tuple) and sorting them* x = list(zip(cases,states,pop))  
 y = sorted(x,reverse=True)  
  
 data1 = []  
 data2 = []  
 data3 = []  
  
  
 for i in y:  
 data1.append(i[0])  
 data2.append(i[1])  
 data3.append(i[2])  
  
 data1 = data1[0:20]  
 data2 = data2[0:20]  
 data3 = data3[0:20]  
  
 *#plotting the graph* plt.scatter(data3,data1,edgecolors="black",linewidths=1,alpha=0.75,color="red")  
  
  
 *#naming each point in a graph* for i,label in enumerate(data2):  
 plt.annotate(label,(int(data3[i]-10000000),int(data1[i])))  
  
  
  
 *#specifying some properties* plt.xlabel("Population")  
 plt.ylabel("Covid cases")  
 plt.title("Population vs Covid cases")  
  
  
 plt.show()  
  
while True:

print("\n")  
 print("Graphs available")  
 print("\n")  
 print("1. Covid cases by countries | Graph type: BAR")

print("2. Covid cases by countries per year | Graph type: DOUBLE BAR ")

print("3. Covid deaths by countries | Graph type: BAR")

print("4. Percentage of global covid cases by countries | Graph type: PIE")

print("5. Covid cases by states | Graph type: BAR")

print("6. Covid deaths by states | Graph type: BAR")

print("7. Percentage of covid cases in India by states | Graph type: PIE")

print("8. Covid death rate by states | Graph type: HORIZONDAL BAR")

print("9. Covid cases and discharges by states | Graph type: DOUBLE BAR")

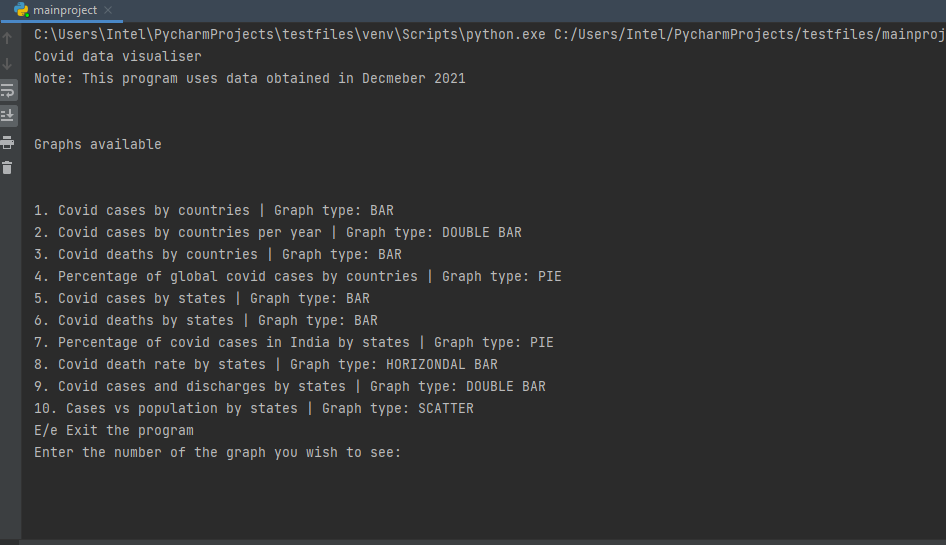
print("10. Cases vs population by states | Graph type: SCATTER")

print("E/e Exit the program")  
  
 x = input("Enter the number of the graph you wish to see: ")  
  
 if x == "e" or x == "E":  
 sys.exit("Thanks for using the program")  
  
 try:  
  
 if int(x) == 1:  
 casecountries()  
  
 elif int(x) == 2:  
 casecountries\_year()  
  
 elif int(x) == 3:  
 deathcountries()  
  
 elif int(x) == 4:  
 piecasescountries()  
  
 elif int(x) == 5:  
 casestates()  
  
 elif int(x) == 6:  
 deathstates()  
  
 elif int(x) == 7:  
 piecasesstates()  
  
 elif int(x) == 8:  
 deathratebystates()  
  
 elif int(x) == 9:  
 casesdischargedstates()  
  
 elif int(x) == 10:  
 casespopulationstates()  
  
 else:  
 print("Please enter a valid input")  
  
 except ValueError:  
 print("Please enter a valid input")  
 continue

OUTPUT

NOTE: The program was created in the pycharm IDE and these are the outputs of the program from the pycharm IDE terminal

Options displayed when the program is ran

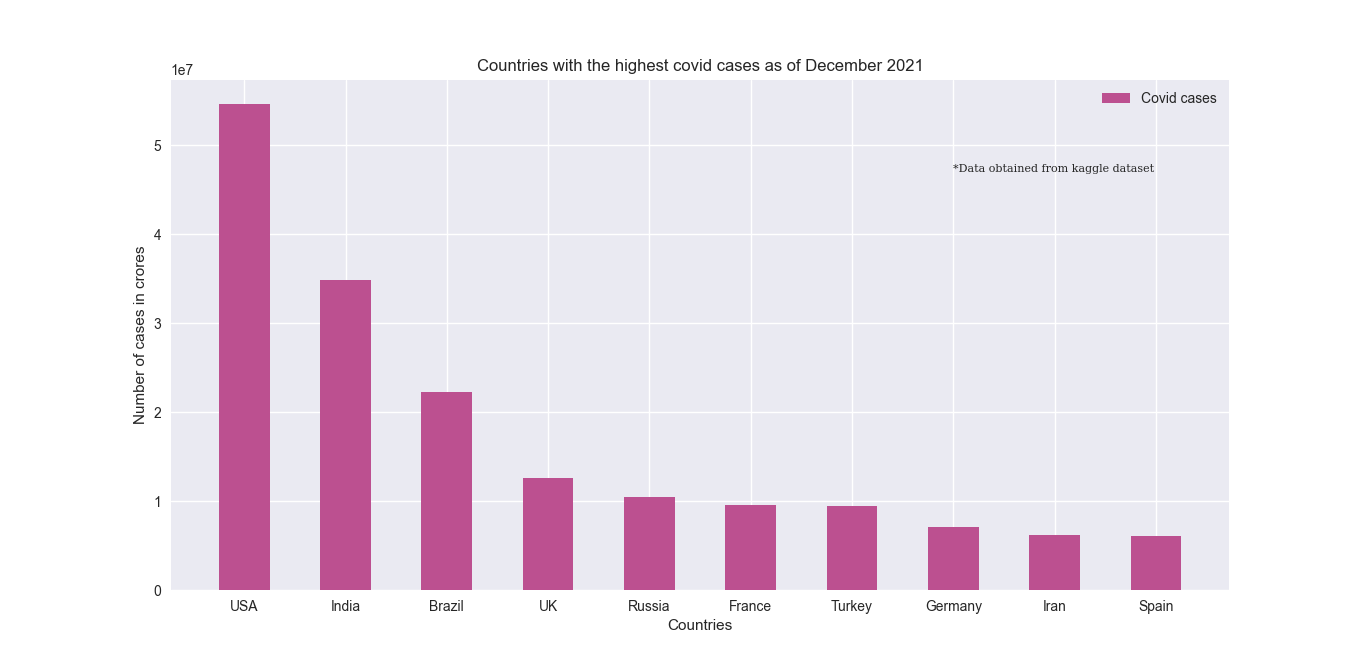


The user is presented with the above screen. It shows the available graphs with their number and type

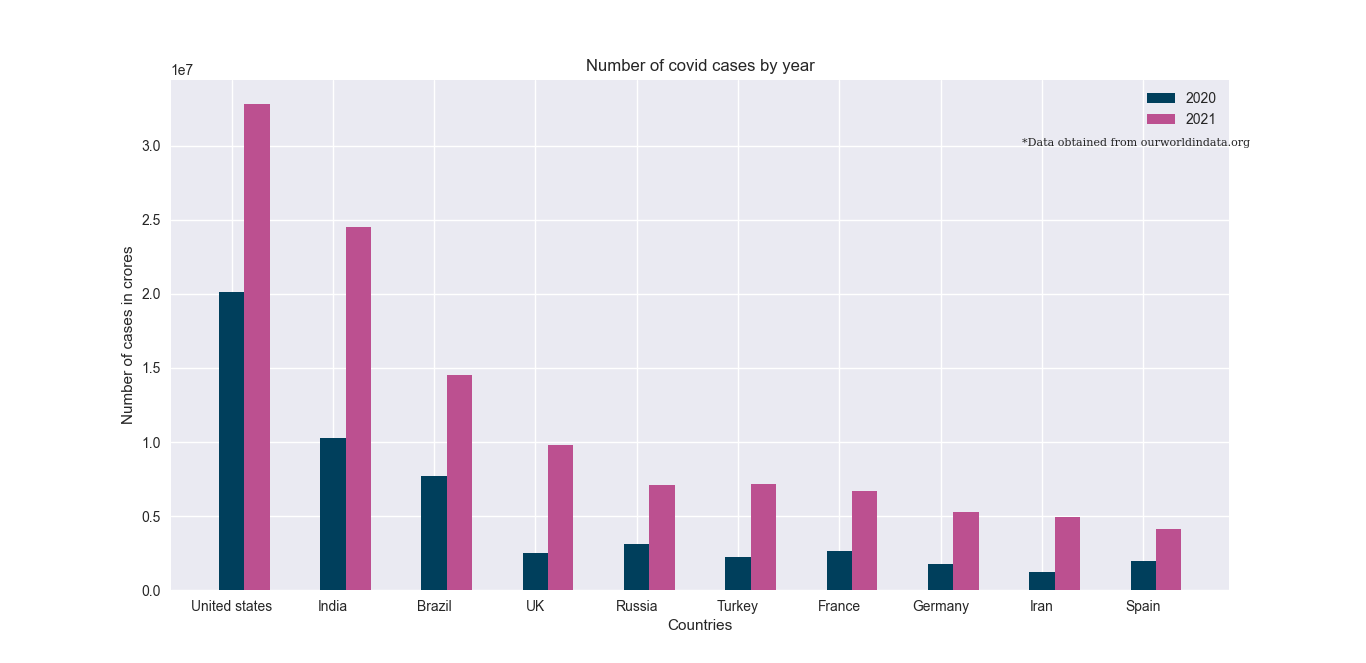
The user is requested to enter the number of the graph they wish to see or enter E/e to exit

Graphs obtained from the program

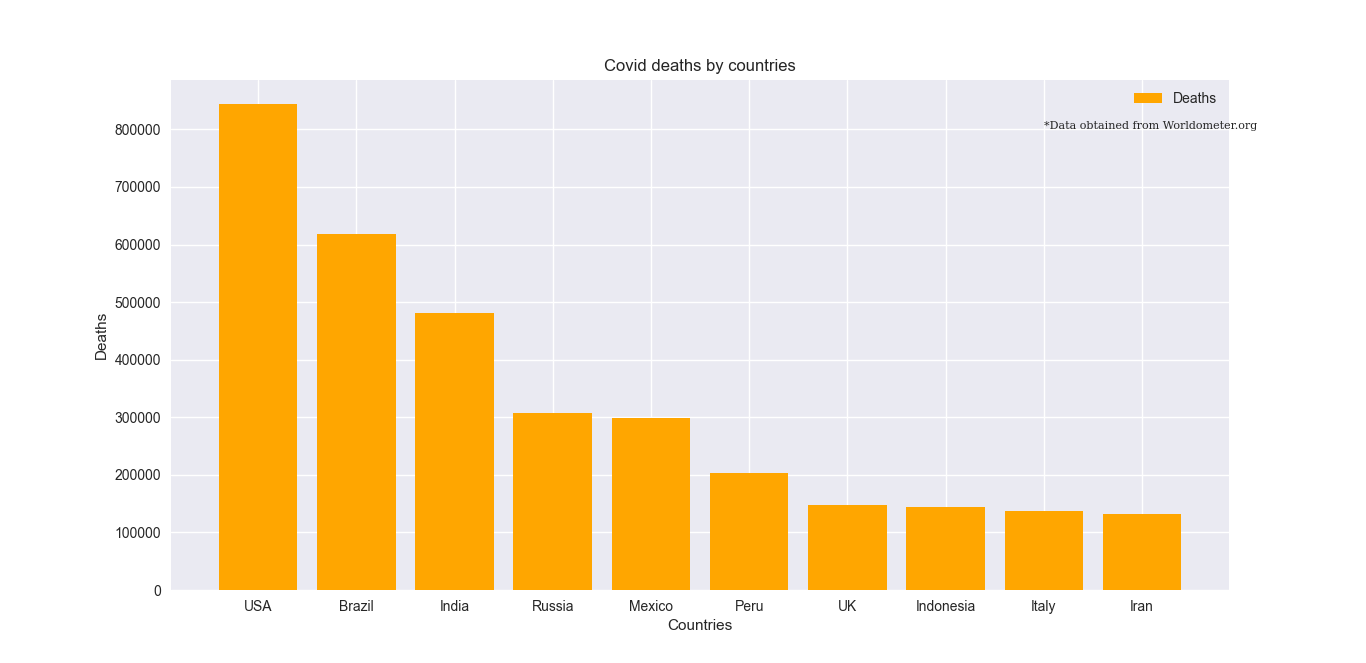
Graph 1



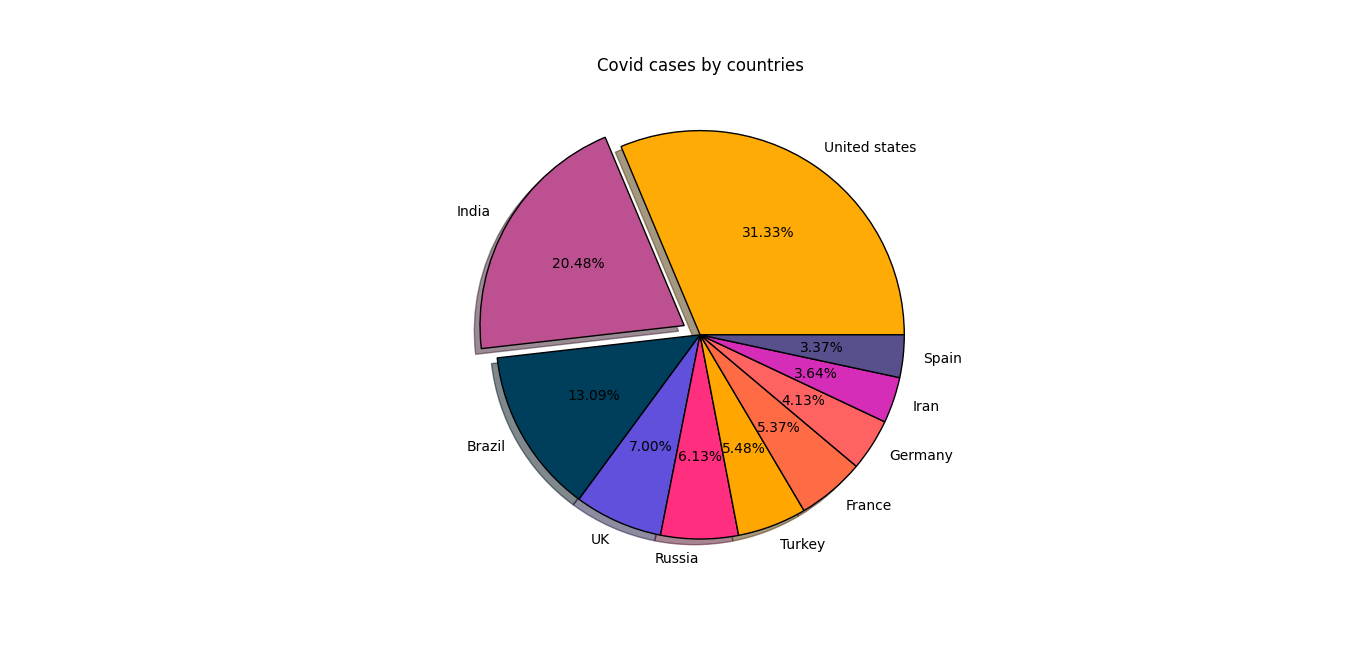
Graph 2



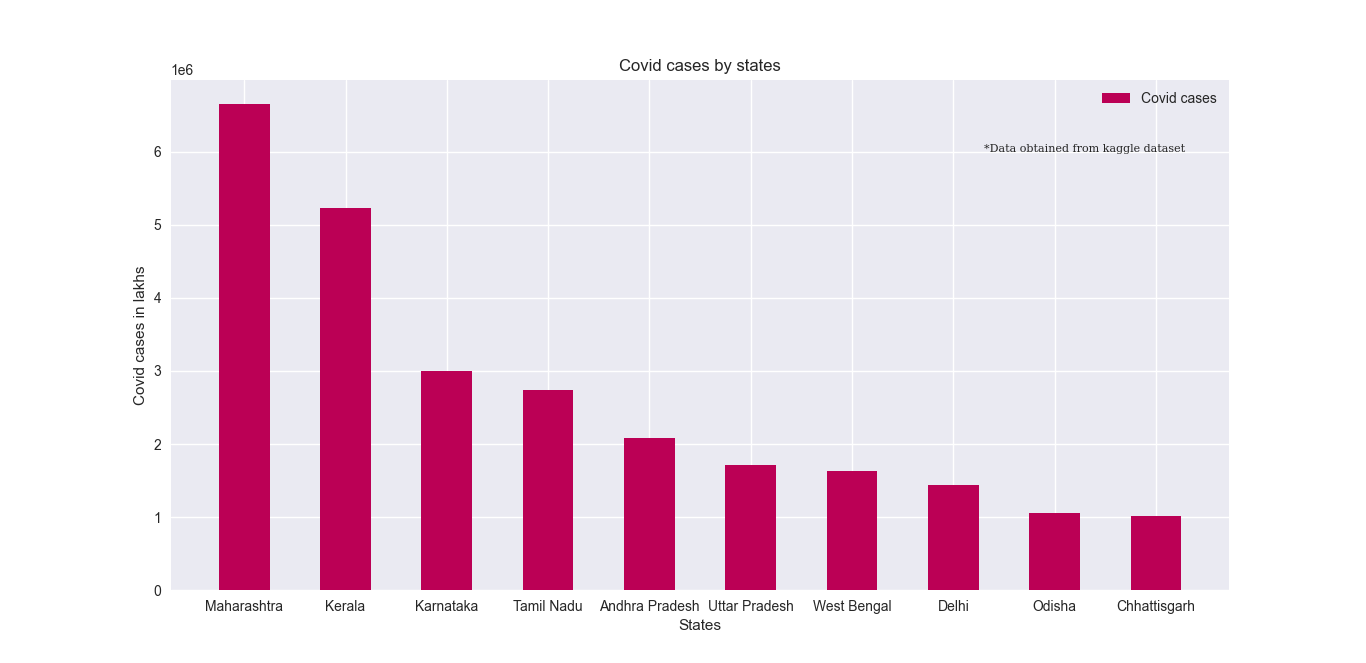
Graph 3



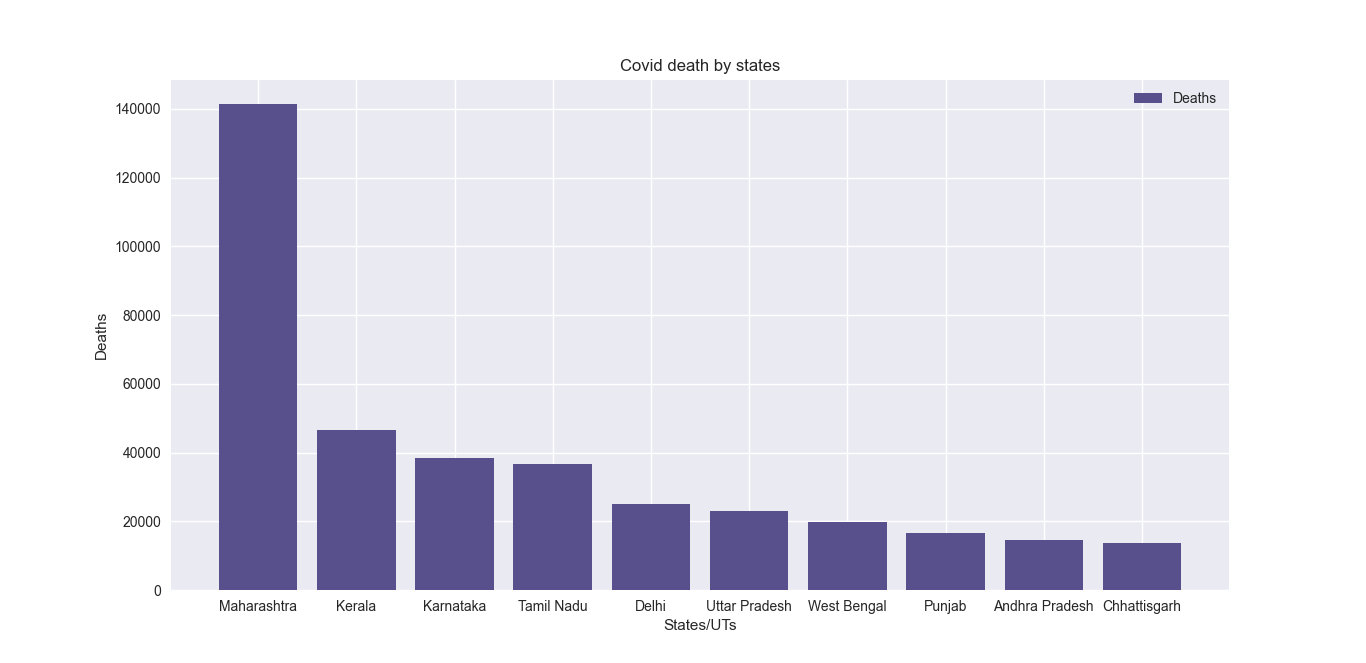
Graph 4



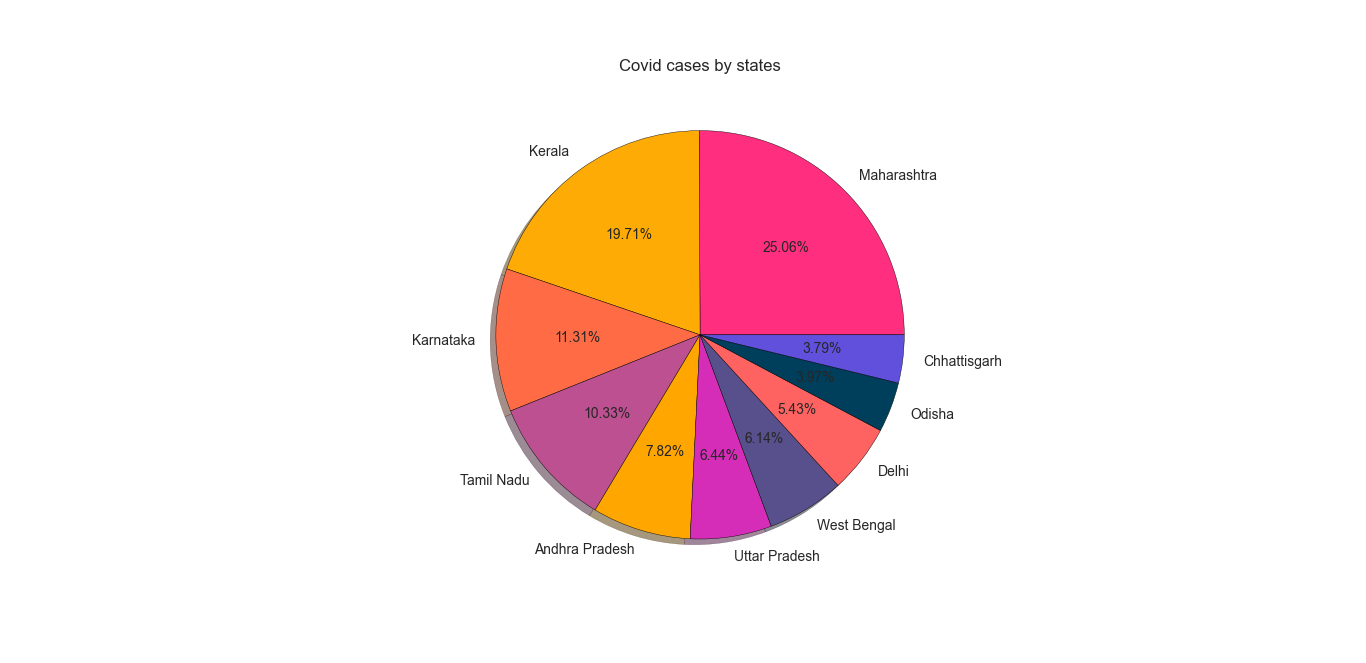
Graph 5



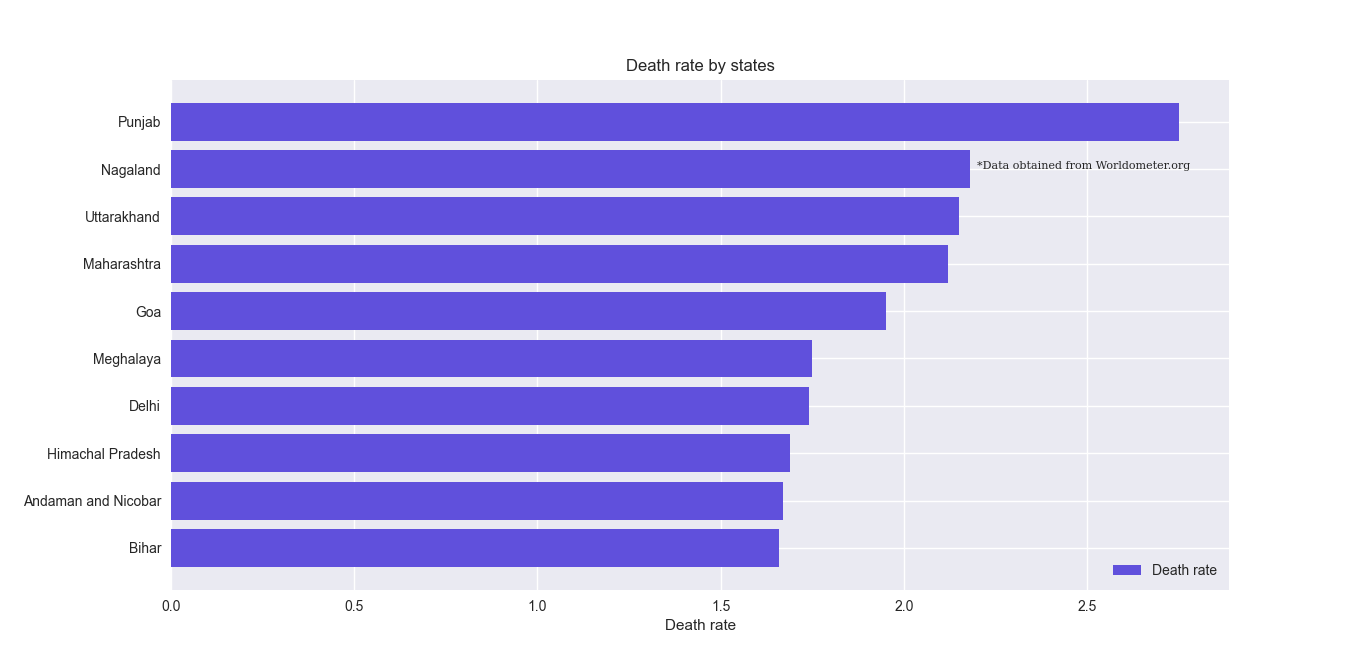
Graph 6



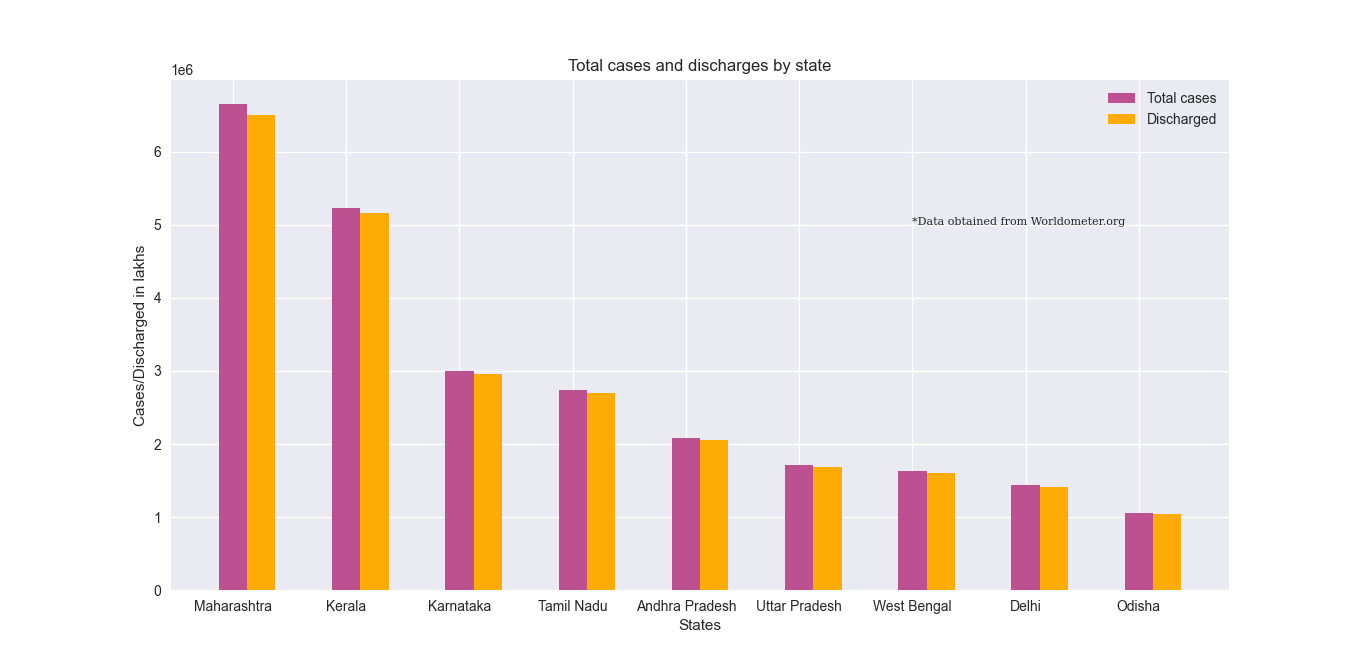
Graph 7



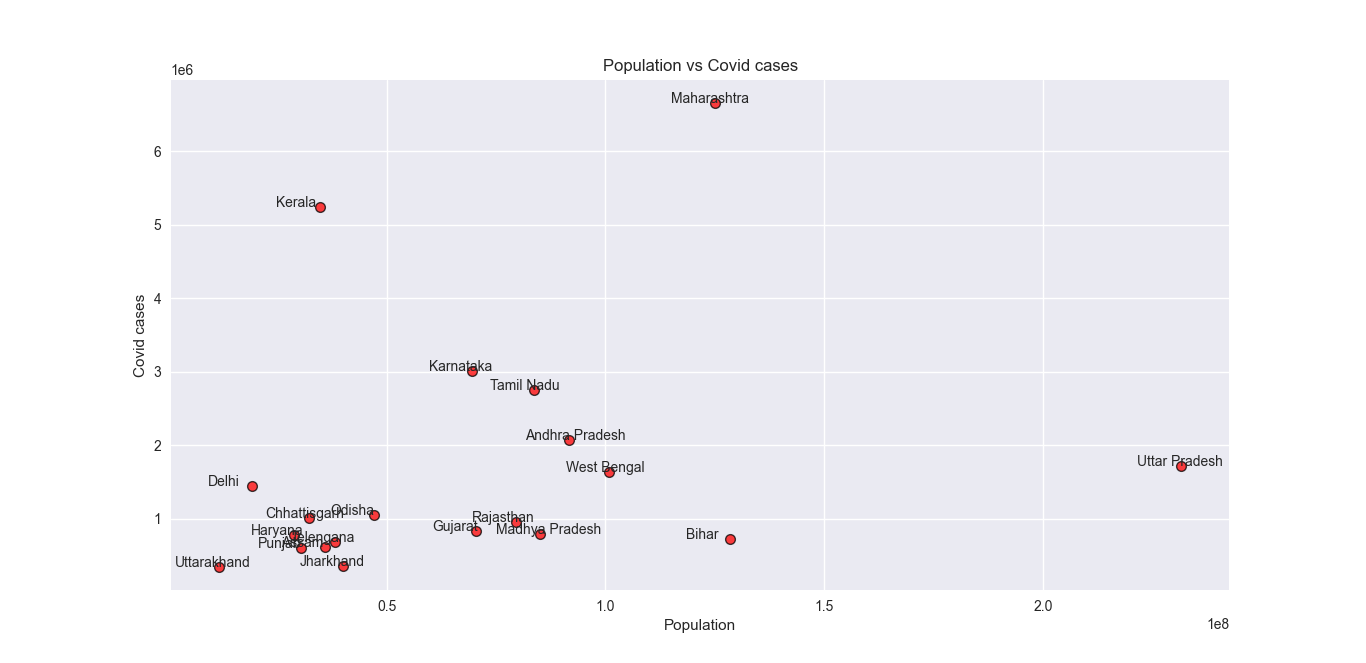
Graph 8



Graph 9

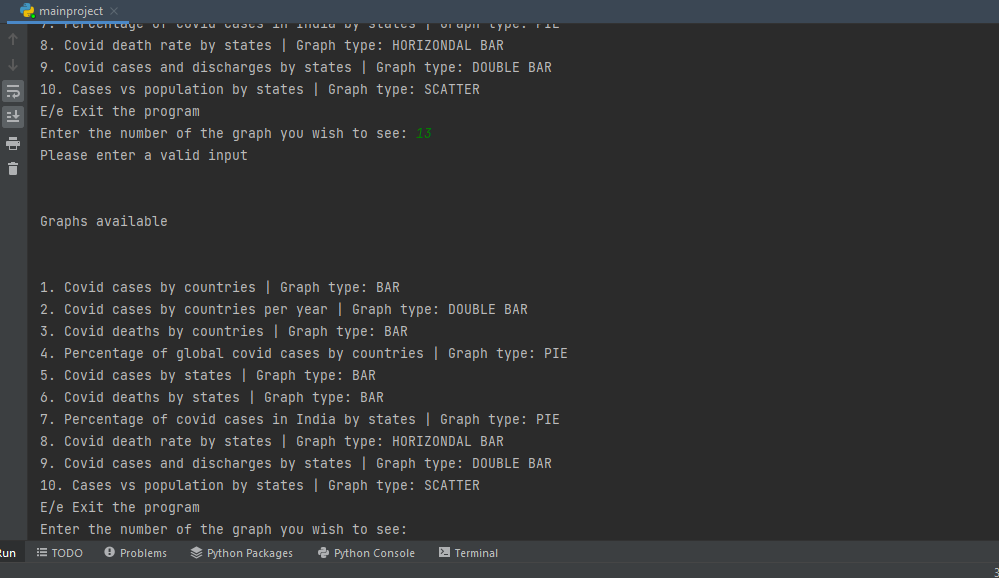


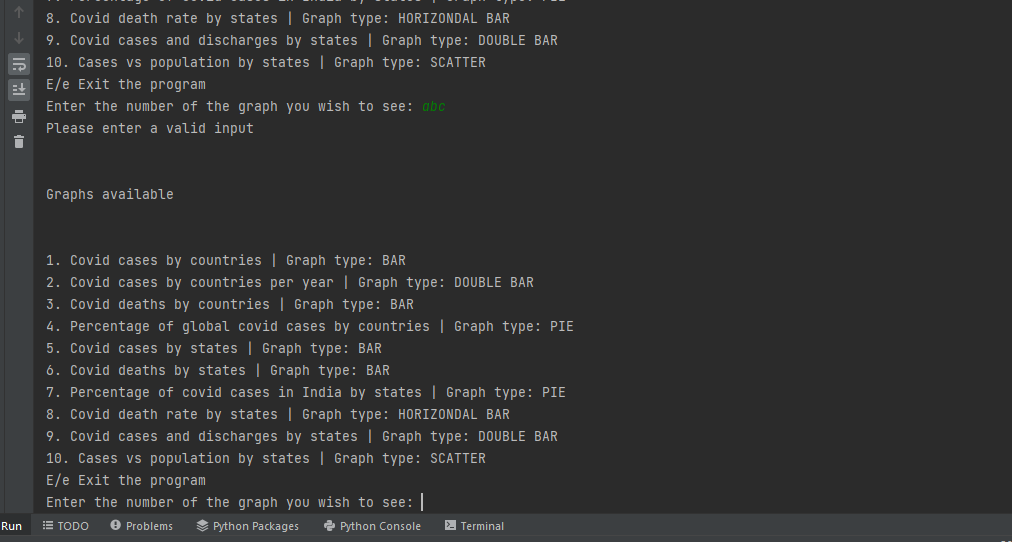
Graph 10



Invalid input cases

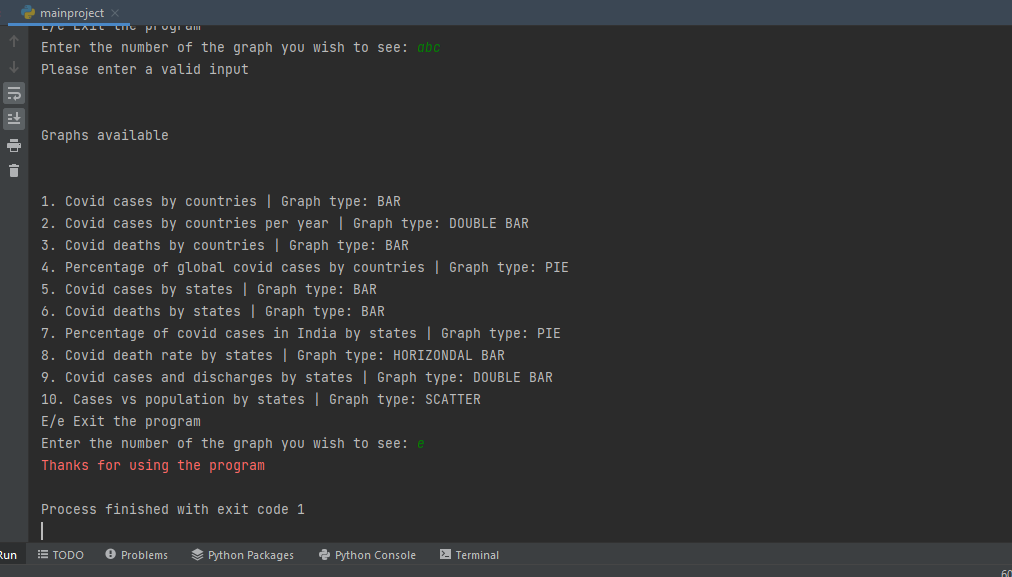
Outputs when the entered number does not belong to the displayed graph numbers or the string entered is not E/e:





Exiting the program

The user can exit the program by entering E or e. The following message is show when the program is exited



CSV files used in the program(Raw files)

FILE: Data4.csv Contains covid data of Indian states

State/UTs,Total Cases,Active,Discharged,Deaths,Active Ratio,Discharge Ratio,Death Ratio,Population  
Andaman and Nicobar,7714,9,7576,129,0.12,98.21,1.67,399001  
Andhra Pradesh,2076492,1166,2060836,14490,0.06,99.25,0.7,91702478  
Arunachal Pradesh,55332,16,55036,280,0.03,99.47,0.51,1711947  
Assam,620134,2075,611903,6156,0.33,98.67,0.99,35998752  
Bihar,726456,99,714263,12094,0.01,98.32,1.66,128500364  
Chandigarh,65774,101,64595,1078,0.15,98.21,1.64,1158040  
Chhattisgarh,1007623,330,993696,13597,0.03,98.62,1.35,32199722  
Dadra and Nagar Haveli and Daman and Diu,10691,1,10686,4,0.01,99.95,0.04,773997  
Delhi,1443352,1103,1417144,25105,0.08,98.18,1.74,19301096  
Goa,180050,449,176082,3519,0.25,97.8,1.95,1521992  
Gujarat,829359,948,818298,10113,0.11,98.67,1.22,70400153  
Haryana,772633,495,762076,10062,0.06,98.63,1.3,28900667  
Himachal Pradesh,228549,346,224330,3873,0.15,98.15,1.69,7503010  
Jammu and Kashmir,340722,1350,334849,4523,0.4,98.28,1.33,14999397  
Jharkhand,349867,361,344364,5142,0.1,98.43,1.47,40100376  
Karnataka,3004587,7447,2958828,38312,0.25,98.48,1.28,69599762  
Kerala,5235122,23372,5165164,46586,0.45,98.66,0.89,34698876  
Ladakh,22091,209,21664,218,0.95,98.07,0.99,290492  
Lakshadweep,10415,7,10357,51,0.07,99.44,0.49,66001  
Madhya Pradesh,793696,252,782912,10532,0.03,98.64,1.33,85002417  
Maharashtra,6657888,13498,6502957,141433,0.2,97.67,2.12,124904071  
Manipur,125723,182,123540,2001,0.14,98.26,1.59,3436948  
Meghalaya,84793,64,83246,1483,0.08,98.18,1.75,3772103  
Mizoram,140564,1399,138626,539,1.0,98.62,0.38,1308967  
Nagaland,32174,77,31395,702,0.24,97.58,2.18,2073074  
Odisha,1053881,1503,1043926,8452,0.14,99.06,0.8,47099270  
Puducherry,129415,127,127408,1880,0.1,98.45,1.45,1646050  
Punjab,604236,378,587220,16638,0.06,97.18,2.75,30501026  
Rajasthan,955480,318,946199,8963,0.03,99.03,0.94,79502477  
Sikkim,32480,61,32010,409,0.19,98.55,1.26,658019  
Tamil Nadu,2744037,6629,2700673,36735,0.24,98.42,1.34,83697770  
Telengana,680662,3417,673223,4022,0.5,98.91,0.59,38157311  
Tripura,85005,48,84128,829,0.06,98.97,0.98,4184959  
Uttar Pradesh,1710931,323,1687693,22915,0.02,98.64,1.34,231502578  
Uttarakhand,344779,231,337132,7416,0.07,97.78,2.15,11700099  
West Bengal,1630626,7450,1603460,19716,0.46,98.33,1.21,100896618

FILE: Data6.csv Contains covid data of countries

Country,Total Cases,Total Deaths,Total Recovered,Active Cases,Total Cases/1M population,Deaths/1M population,Total Tests,Tests/1M population,Population  
Afghanistan,158056,7356,145606.0,5094.0,3928,183,820849.0,20402.0,40234070  
Albania,208899,3212,200181.0,5506.0,72708,1118,1473154.0,512739.0,2873109  
Algeria,217647,6263,149617.0,61767.0,4833,139,230861.0,5127.0,45030899  
Andorra,22823,140,20563.0,2120.0,294684,1808,249838.0,3225839.0,77449  
Angola,78475,1760,64124.0,12591.0,2282,51,1273800.0,37042.0,34387598  
Anguilla,1674,5,1640.0,29.0,110110,329,51382.0,3379728.0,15203  
Antigua and Barbuda,4259,118,4060.0,81.0,42960,1190,18559.0,187204.0,99138  
Argentina,5556239,117111,5283910.0,155218.0,121278,2556,27790142.0,606583.0,45814218  
Armenia,344826,7968,331348.0,5510.0,116044,2681,2560870.0,861809.0,2971506  
Aruba,19021,181,16731.0,2109.0,177035,1685,177885.0,1655637.0,107442  
Australia,362691,2225,245297.0,115169.0,13981,86,54648003.0,2106627.0,25940997  
Austria,1274995,13701,1232226.0,29068.0,140371,1508,123226848.0,13566686.0,9083047  
Azerbaijan,616352,8346,598793.0,9213.0,59983,812,5853336.0,569642.0,10275459  
Bahamas,24269,716,21789.0,1764.0,60848,1795,182597.0,457812.0,398847  
Bahrain,280876,1394,277108.0,2374.0,156953,779,7983336.0,4461071.0,1789556  
Bangladesh,1585027,28070,1548811.0,8146.0,9483,168,11472815.0,68644.0,167135949  
Barbados,28063,260,26848.0,955.0,97475,903,467711.0,1624561.0,287900  
Belarus,697600,5561,690505.0,1534.0,73862,589,11022249.0,1167032.0,9444686  
Belgium,2075781,28267,1706312.0,341202.0,177952,2423,27122517.0,2325147.0,11664862  
Belize,32067,597,30545.0,925.0,78522,1462,379996.0,930489.0,408383  
Benin,24935,161,24705.0,69.0,1980,13,604310.0,47977.0,12595948  
Bermuda,6014,106,5674.0,234.0,97102,1711,685295.0,11064745.0,61935  
Bhutan,2660,3,2649.0,8.0,3392,4,1286842.0,1640802.0,784276  
Bolivia,585624,19622,523779.0,42223.0,49169,1647,2633963.0,221149.0,11910376  
Bosnia and Herzegovina,290471,13428,192218.0,84825.0,89366,4131,1479046.0,455040.0,3250365  
Botswana,212482,2439,196455.0,13588.0,87717,1007,2015878.0,832196.0,2422361  
Brazil,22263834,618870,21567845.0,77119.0,103643,2881,63776166.0,296892.0,214812911  
Brunei,15470,98,15272.0,100.0,34864,221,687114.0,1548513.0,443725  
Bulgaria,740682,30819,610724.0,99139.0,107808,4486,7386068.0,1075059.0,6870385  
Burkina Faso,17632,318,16619.0,695.0,810,15,239193.0,10994.0,21756518  
Burundi,27366,38,773.0,26555.0,2204,3,345742.0,27842.0,12417834  
Cabo Verde,39947,351,38211.0,1385.0,70704,621,389139.0,688753.0,564991  
Cambodia,120487,3012,116907.0,568.0,7061,177,2761485.0,161832.0,17063869  
Cameroon,109367,1851,106050.0,1466.0,3973,67,1751774.0,63632.0,27529766  
Canada,2102474,30248,1864799.0,207427.0,54987,791,52408105.0,1370661.0,38235634  
CAR,12163,101,6859.0,5203.0,2455,20,74519.0,15039.0,4954897  
Caribbean Netherlands,3336,23,3193.0,120.0,125480,865,30126.0,1133153.0,26586  
Cayman Islands,8386,11,4799.0,3576.0,125402,164,215963.0,3229450.0,66873  
Chad,5701,181,4874.0,646.0,333,11,180347.0,10530.0,17127409  
Channel Islands,25998,113,22189.0,3696.0,147504,641,1123134.0,6372283.0,176253  
Chile,1802891,39063,1722302.0,41526.0,93120,2018,27232477.0,1406570.0,19360907  
China,101890,4636,94691.0,2563.0,71,3,160000000.0,111163.0,1439323776  
Colombia,5138603,129866,4965217.0,43520.0,99409,2512,29479920.0,570305.0,51691542  
Comoros,6284,156,4551.0,1577.0,7004,174,,,897193  
Congo,20089,367,12421.0,7301.0,3512,64,301494.0,52706.0,5720330  
Costa Rica,570556,7353,560709.0,2494.0,110498,1424,2965435.0,574309.0,5163485  
Croatia,709678,12493,671664.0,25521.0,174476,3071,3743534.0,920355.0,4067487  
Cuba,965243,8321,956097.0,825.0,85297,735,11626920.0,1027449.0,11316301  
Curaçao,20199,187,17674.0,2338.0,122348,1133,391243.0,2369820.0,165094  
Cyprus,157928,635,124370.0,32923.0,129400,520,9477138.0,7765173.0,1220467  
Czechia,2469951,36061,2304176.0,129714.0,230006,3358,46866575.0,4364301.0,10738621  
Denmark,769766,3256,586027.0,180483.0,132202,559,106452714.0,18282583.0,5822630  
Djibouti,13619,189,13350.0,80.0,13492,187,257996.0,255590.0,1009415  
Dominica,6520,44,6173.0,303.0,90238,609,121399.0,1680193.0,72253  
Dominican Republic,416270,4246,407783.0,4241.0,37808,386,2568096.0,233249.0,11010118  
DRC,74793,1205,50930.0,22658.0,799,13,846704.0,9045.0,93613410  
Ecuador,542960,33650,443880.0,65430.0,30094,1865,2136008.0,118389.0,18042354  
Egypt,383857,21695,319225.0,42937.0,3648,206,3693367.0,35105.0,105210099  
El Salvador,121741,3823,107088.0,10830.0,18629,585,1505621.0,230393.0,6535022  
Equatorial Guinea,13637,175,13410.0,52.0,9267,119,263190.0,178851.0,1471562  
Eritrea,7948,74,7711.0,163.0,2196,20,23693.0,6546.0,3619586  
Estonia,240334,1928,223051.0,15355.0,180993,1452,2514711.0,1893799.0,1327866  
Eswatini,65473,1296,57617.0,6560.0,55576,1100,427929.0,363244.0,1178075  
Ethiopia,410445,6916,354174.0,49355.0,3443,58,4084823.0,34268.0,119203769  
Faeroe Islands,5548,13,4961.0,574.0,112902,265,638000.0,12983313.0,49140  
Fiji,53332,698,51223.0,1411.0,58857,770,471147.0,519953.0,906133  
Finland,248958,1554,46000.0,201404.0,44829,280,8621296.0,1552391.0,5553561  
France,9534357,123372,7982413.0,1428572.0,145587,1884,188795159.0,2882854.0,65488987  
French Guiana,48038,339,11254.0,36445.0,154857,1093,487548.0,1571681.0,310208  
French Polynesia,46382,636,,,163703,2245,,,283330  
Gabon,39910,288,36740.0,2882.0,17322,125,1479562.0,642187.0,2303943  
Gambia,10136,342,9688.0,106.0,4026,136,131044.0,52049.0,2517727  
Georgia,932641,13758,893064.0,25819.0,234465,3459,12872568.0,3236150.0,3977742  
Germany,7101162,112133,6332300.0,656729.0,84353,1332,89622218.0,1064604.0,84183592  
Ghana,141295,1287,130474.0,9534.0,4410,40,2096416.0,65433.0,32039115  
Greece,1134713,20636,1006345.0,107732.0,109665,1994,46279797.0,4472740.0,10347080  
Greenland,2437,1,2161.0,275.0,42817,18,135603.0,2382469.0,56917  
Grenada,6073,200,5685.0,188.0,53601,1765,93030.0,821087.0,113301  
Guadeloupe,56035,750,2250.0,53035.0,140008,1874,607573.0,1518075.0,400226  
Guatemala,626675,16102,608237.0,2336.0,34041,875,3265129.0,177363.0,18409350  
Guinea,31412,389,29815.0,1208.0,2299,28,571842.0,41853.0,13663083  
Guinea-Bissau,6476,149,6302.0,25.0,3179,73,110892.0,54432.0,2037268  
Guyana,39238,1051,37544.0,643.0,49530,1327,421110.0,531562.0,792213  
Haiti,25985,766,23289.0,1930.0,2238,66,132422.0,11406.0,11609760  
Honduras,379542,10434,124171.0,244937.0,37432,1029,1139978.0,112431.0,10139379  
Hong Kong,12631,213,12191.0,227.0,1665,28,29687335.0,3912567.0,7587688  
Hungary,1253055,39104,1106531.0,107420.0,130204,4063,9152694.0,951053.0,9623749  
Iceland,27059,37,20654.0,6368.0,78535,107,1364712.0,3960899.0,344546  
India,34822040,480860,34258778.0,82402.0,24869,343,675246143.0,482238.0,1400235552  
Indonesia,4262540,144088,4114141.0,4311.0,15342,519,62956193.0,226595.0,277836221  
Iran,6192698,131572,6034821.0,26305.0,72348,1537,41791485.0,488243.0,85595627  
Iraq,2093436,24154,2064527.0,4755.0,50363,581,16817837.0,404598.0,41566778  
Ireland,747895,5912,564203.0,177780.0,148982,1178,9975030.0,1987049.0,5020023  
Isle of Man,13487,67,13057.0,363.0,157367,782,133676.0,1559741.0,85704  
Israel,1374453,8243,1347558.0,18652.0,147379,884,37741674.0,4046931.0,9326000  
Italy,5854428,137091,5042472.0,674865.0,97041,2272,137763229.0,2283526.0,60329171  
Ivory Coast,66568,708,61424.0,4436.0,2434,26,1221161.0,44648.0,27350895  
Jamaica,93226,2469,65442.0,25315.0,31279,828,703604.0,236073.0,2980447  
Japan,1732835,18393,1711828.0,2614.0,13764,146,29312089.0,232823.0,125898724  
Jordan,1059515,12590,1011442.0,35483.0,102327,1216,13462192.0,1300171.0,10354171  
Kazakhstan,987386,13005,959635.0,14746.0,51664,680,11575012.0,605655.0,19111551  
Kenya,292237,5376,251223.0,35638.0,5262,97,3020888.0,54396.0,55535418  
Kuwait,416077,2468,411504.0,2105.0,95328,565,5921186.0,1356606.0,4364707  
Kyrgyzstan,184633,2800,179778.0,2055.0,27621,419,1907195.0,285312.0,6684604  
Laos,110054,360,7339.0,102355.0,14806,48,913451.0,122893.0,7432923  
Latvia,275517,4561,259623.0,11333.0,148521,2459,5600259.0,3018887.0,1855074  
Lebanon,719103,9087,659448.0,50568.0,106065,1340,4795578.0,707330.0,6779834  
Lesotho,28408,665,15241.0,12502.0,13106,307,322581.0,148821.0,2167575  
Liberia,6278,287,5535.0,456.0,1199,55,139824.0,26711.0,5234708  
Libya,388183,5696,375747.0,6740.0,55371,812,2025457.0,288915.0,7010575  
Liechtenstein,6066,69,5715.0,282.0,158418,1802,80413.0,2100050.0,38291  
Lithuania,517655,7373,484771.0,25511.0,194199,2766,6589925.0,2472215.0,2665595  
Luxembourg,101256,911,93608.0,6737.0,157927,1421,3809430.0,5941512.0,641155  
Madagascar,50279,1027,44708.0,4544.0,1749,36,320884.0,11161.0,28750905  
Malawi,73238,2350,59842.0,11046.0,3687,118,482503.0,24288.0,19865922  
Malaysia,2750516,31428,2677406.0,41682.0,83389,953,40895705.0,1239858.0,32984197  
Maldives,95420,262,92940.0,2218.0,172012,472,1869391.0,3369911.0,554730  
Mali,20468,658,18474.0,1336.0,969,31,532617.0,25220.0,21118989  
Malta,51070,476,40045.0,10549.0,115205,1074,1211456.0,2732831.0,443297  
Martinique,48473,779,104.0,47590.0,129320,2078,488892.0,1304300.0,374831  
Mauritania,40906,862,39112.0,932.0,8466,178,558833.0,115662.0,4831612  
Mauritius,23421,762,21881.0,778.0,18371,598,358675.0,281332.0,1274917  
Mayotte,21911,185,2964.0,18762.0,77529,655,176919.0,626003.0,282617  
Mexico,3961662,299132,3304774.0,357756.0,30253,2284,12328458.0,94144.0,130952566  
Moldova,375358,9667,361724.0,3967.0,93373,2405,2334414.0,580704.0,4019974  
Monaco,4985,38,4524.0,423.0,125703,958,54960.0,1385884.0,39657  
Mongolia,389089,2058,313256.0,73775.0,115905,613,4030048.0,1200502.0,3356968  
Montenegro,166581,2401,158516.0,5664.0,265178,3822,967203.0,1539679.0,628185  
Morocco,959098,14842,938892.0,5364.0,25533,395,10805226.0,287658.0,37562815  
Mozambique,179272,1988,153731.0,23553.0,5506,61,1076026.0,33049.0,32558881  
Myanmar,530276,19262,507806.0,3208.0,9649,351,6018475.0,109519.0,54953819  
Namibia,146720,3616,130402.0,12702.0,56226,1386,864924.0,331456.0,2609469  
Nepal,828207,11590,811746.0,4871.0,27683,387,4851470.0,162161.0,29917667  
Netherlands,3101261,20852,2600170.0,480239.0,180396,1213,21107399.0,1227786.0,17191433  
New Caledonia,12762,281,12217.0,264.0,44069,970,42756.0,147643.0,289591  
New Zealand,14057,51,12780.0,1226.0,2810,10,5561698.0,1111873.0,5002100  
Nicaragua,17487,212,4225.0,13050.0,2594,31,,,6742059  
Niger,7307,274,6900.0,133.0,286,11,203062.0,7960.0,25509885  
Nigeria,240387,3030,213491.0,23866.0,1125,14,3823309.0,17887.0,213747187  
North Macedonia,224134,7935,211228.0,4971.0,107589,3809,1586033.0,761328.0,2083246  
Norway,387196,1305,88952.0,296939.0,70599,238,9267576.0,1689788.0,5484461  
Oman,305489,4116,300381.0,992.0,57639,777,25000000.0,4716984.0,5299997  
Pakistan,1294861,28921,1255931.0,10009.0,5697,127,23324549.0,102624.0,227282382  
Palestine,439495,4662,431160.0,3673.0,83269,883,2847177.0,539440.0,5278021  
Panama,491043,7425,474829.0,8789.0,111196,1681,4462855.0,1010611.0,4415996  
Papua New Guinea,36137,590,35441.0,106.0,3928,64,249149.0,27083.0,9199532  
Paraguay,466101,16624,447870.0,1607.0,64173,2289,2027738.0,279181.0,7263155  
Peru,2287494,202584,,,67966,6019,21656119.0,643445.0,33656496  
Philippines,2841260,51373,2778115.0,11772.0,25424,460,25078068.0,224402.0,111755258  
Poland,4094608,96415,3614290.0,383903.0,108367,2552,27104556.0,717341.0,37784763  
Portugal,1330158,18921,1175217.0,136020.0,131020,1864,25760643.0,2537408.0,10152345  
Qatar,249787,617,245348.0,3822.0,88962,220,3168202.0,1128355.0,2807805  
Réunion,76602,409,69431.0,6762.0,84658,452,1347378.0,1489077.0,904841  
Romania,1807223,58714,1737543.0,10966.0,94886,3083,17051755.0,895285.0,19046177  
Russia,10479344,307948,9423227.0,748169.0,71763,2109,240000000.0,1643522.0,146027823  
Rwanda,109070,1348,45522.0,62200.0,8121,100,3965214.0,295244.0,13430291  
S. Korea,625967,5455,509957.0,110555.0,12194,106,15804065.0,307861.0,51335014  
Saint Kitts and Nevis,2918,28,2784.0,106.0,54278,521,58205.0,1082682.0,53760  
Saint Lucia,13473,295,12950.0,228.0,72875,1596,107507.0,581499.0,184879  
Saint Martin,4008,56,1399.0,2553.0,101087,1412,78366.0,1976494.0,39649  
San Marino,7655,98,6725.0,832.0,224902,2879,103193.0,3031789.0,34037  
Sao Tome and Principe,3819,57,3678.0,84.0,16957,253,29036.0,128921.0,225223  
Saudi Arabia,555417,8875,541614.0,4928.0,15592,249,33134497.0,930172.0,35621922  
Senegal,74870,1890,72482.0,498.0,4303,109,917913.0,52751.0,17400828  
Serbia,1295198,12665,1260458.0,22075.0,149133,1458,7189047.0,827768.0,8684854  
Seychelles,24546,134,23980.0,432.0,247302,1350,,,99255  
Sierra Leone,6891,123,,,838,15,259958.0,31630.0,8218637  
Singapore,278750,826,275084.0,2840.0,47094,140,21098774.0,3564608.0,5918960  
Sint Maarten,5054,75,4653.0,326.0,115912,1720,62056.0,1423237.0,43602  
Slovakia,838738,16598,777612.0,44528.0,153514,3038,5085694.0,930834.0,5463588  
Slovenia,462152,5583,439653.0,16916.0,222256,2685,2072037.0,996476.0,2079364  
Somalia,23532,1333,12948.0,9251.0,1422,81,391308.0,23639.0,16553247  
South Africa,3433554,90935,3145543.0,197076.0,56825,1505,21141715.0,349893.0,60423455  
South Sudan,15152,135,12789.0,2228.0,1330,12,294645.0,25870.0,11389386  
Spain,6133057,89331,5049396.0,994330.0,131099,1910,66213858.0,1415377.0,46781797  
Sri Lanka,586746,14962,560313.0,11471.0,27231,694,5726599.0,265770.0,21547178  
St. Vincent Grenadines,5850,81,5228.0,541.0,52482,727,93855.0,842006.0,111466  
Sudan,46518,3331,38714.0,4473.0,1025,73,562941.0,12407.0,45373938  
Suriname,52031,1189,49084.0,1758.0,87525,2000,169283.0,284763.0,594470  
Sweden,1314784,15271,1192194.0,107319.0,128986,1498,14931240.0,1464825.0,10193191  
Switzerland,1288225,12247,993334.0,282644.0,147242,1400,14141939.0,1616403.0,8749016  
Syria,50204,2889,32275.0,15040.0,2769,159,146269.0,8066.0,18133713  
Taiwan,16988,850,15871.0,267.0,711,36,9236508.0,386771.0,23881086  
Tajikistan,17095,124,16966.0,5.0,1734,13,,,9856698  
Tanzania,29306,737,,,471,12,,,62266704  
Thailand,2220324,21672,2165253.0,33399.0,31691,309,17270775.0,246510.0,70061267  
Timor-Leste,19833,122,19705.0,6.0,14628,90,226255.0,166872.0,1355858  
Togo,29416,246,26113.0,3057.0,3433,29,613086.0,71558.0,8567675  
Trinidad and Tobago,90829,2825,72093.0,15911.0,64590,2009,502726.0,357495.0,1406248  
Tunisia,724680,25556,695993.0,3131.0,60375,2129,3342848.0,278501.0,12003013  
Turkey,9402083,82059,9014490.0,305534.0,109719,958,117951480.0,1376455.0,85692215  
Turks and Caicos,3283,26,3142.0,115.0,83118,658,94789.0,2399843.0,39498  
UAE,759511,2162,744180.0,13169.0,75434,215,110915918.0,11016070.0,10068556  
Uganda,139079,3291,98379.0,37409.0,2904,69,2136079.0,44603.0,47890461  
UK,12559926,148089,10229561.0,2182276.0,183577,2164,395579689.0,5781814.0,68417926  
Ukraine,3660620,95690,3460249.0,104681.0,84460,2208,16757339.0,386633.0,43341670  
Uruguay,410311,6168,398220.0,5923.0,117511,1766,4379860.0,1254367.0,3491689  
USA,54656866,844272,41408291.0,12404303.0,163693,2529,809008481.0,2422923.0,333897730  
Uzbekistan,198769,1485,195840.0,1444.0,5813,43,1377915.0,40299.0,34192202  
Venezuela,444186,5321,433208.0,5657.0,15687,188,3359014.0,118627.0,28315717  
Vietnam,1694874,31877,1302542.0,360455.0,17182,323,74476728.0,755000.0,98644616  
Yemen,10126,1984,7020.0,1122.0,329,64,265253.0,8611.0,30804358  
Zambia,249193,3730,216666.0,28797.0,13012,195,2943337.0,153696.0,19150328  
Zimbabwe,209655,4967,173916.0,30772.0,13806,327,1829447.0,120474.0,15185418

Conclusion

Thus the program visualises covid related statistics with data imported from csv files. The visualisation makes it much easier to analyse and observe the data.

The csv files also serve as a very space efficient form of storing data and are much easier to port than other types of data files typically used.

These visualisations allows us to observe anomalies and give room to analyse our flaws in our handling of the pandemic. On the contrary, it also paints a picture on how India has handled the pandemic compared to other countries with lesser populations

Analysing this data is crucial to identify and learn from our mistakes and also to prevent pandemics like this in the future

Bibiliography

* *Computer science with python class 12 by Sumita Arora*
* *Matplotlib tutorials by Corey Schafer*
* *Stack overflow(for the dictionary reverse sorting snippet)*
* *Matplotlib documentation(specifically pyplot)*