**INDUSTRY INTERNSHIP REPORT**

**ON**

**“Web Scraping on Covid 19 Cases (GUI Based)”**

**AT**

**Remark skill**

**Ramnagar, Delhi**

**AN INDUSTRY INTERNSHIP REPORT SUBMITTED**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS**

**FOR THE AWARD OF DEGREE OF**

**BACHELOR OF ENGINEERING**

**In**

**Computer Science Engineering**

**SUBMITTED BY**

Shafin Showkat

Roll Number:2020A1R084

**A picture containing text, clipart

Description automatically generated**

**SUBMITTED TO**

**Computer Science Department**

**Model Institute of Engineering and Technology (Autonomous)**

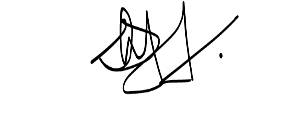
**Jammu, India**

**2022**

**CANDIDATES’ DECLARATION**

I, **Shafin Showkat, roll number 2020A1R084,** hereby declare that the work which is being presented in the Industry Internship Report entitled, “**Web Scraping of Covid19 Cases**” in partial fulfillment of requirement for the award of degree of B.E. (Computer Science Engineering ) and submitted in the Computer Science Department, Model Institute of Engineering and Technology (Autonomous), Jammu is an authentic record of my own work carried by me at “Remark skill, Ramnagar Delhi” under the supervision and mentorship of Mr. Saksham Madan Robotic Software Developer, Remark Skill, Ramnagar Delhi and MS Vishalika (Assistant Professor, Computer Science Department, Model Institute of Engineering and Technology) respectively. The matter presented in this report has not been submitted in this or any other University / Institute for the award of B.E. Degree.

*Signature of the Student*  *Dated*: 04-11-22



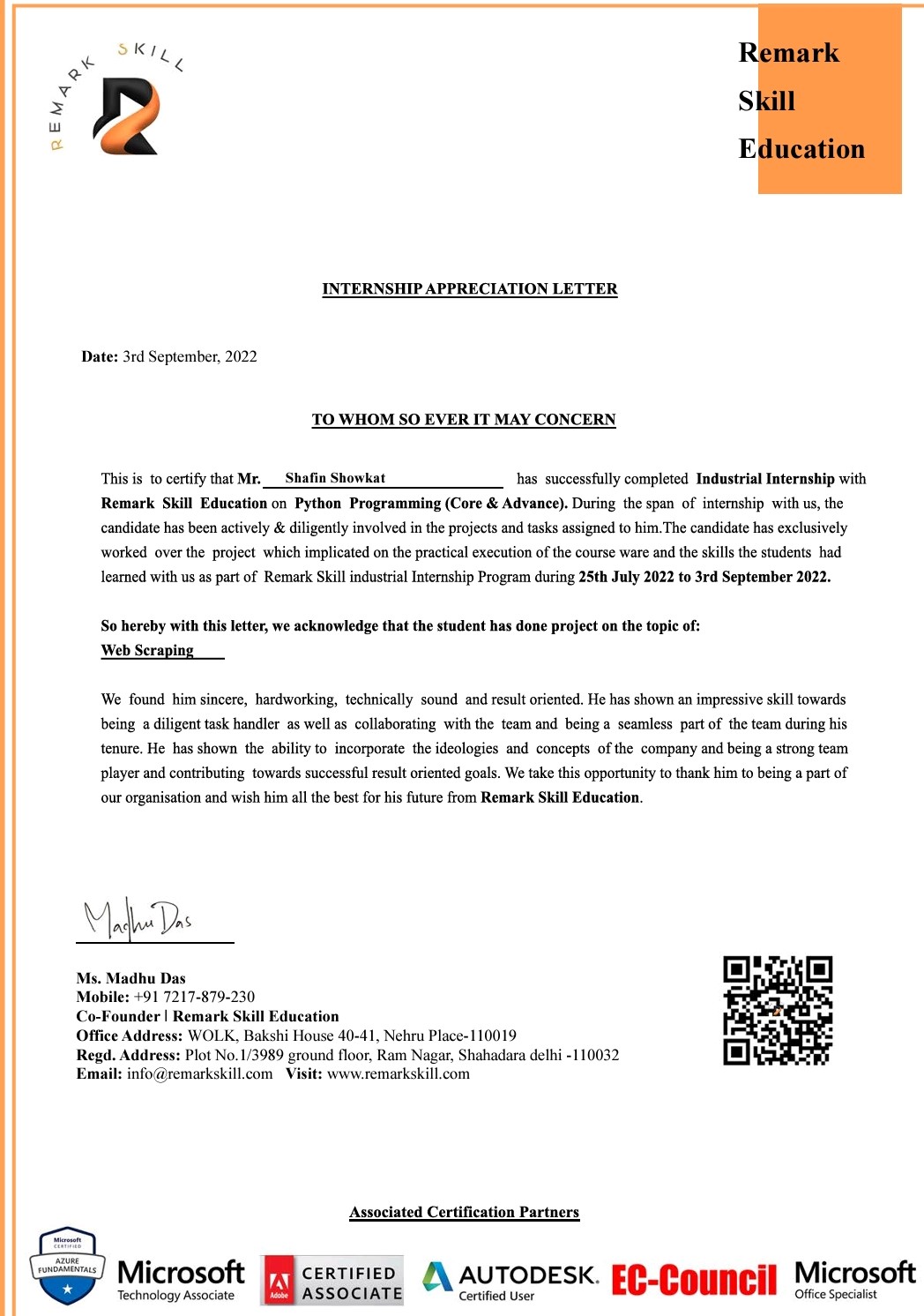
**Shafin Showkat**

**Roll Number 2020A1R084**

**INTERNSHIP CERTIFICATE**



**EXPERIENCE LETTER**

****

**COMPUTER SCIENCE ENGINEERING**

**Model Institute of Engineering and Technology (Autonomous) Kot Bhalwal, Jammu, India**

***(NAAC “A” Grade Accredited)***

**Ref. No.: Date:**

**CERTIFICATE**

Certified that this Industry Internship Report entitled **“Web Scraping of Covid19 Cases (GUI Based)”** is the bona fide work of “**Shafin Showkat, Roll No. 2020A1R084, of 5th Semester, Computer Science Engineering, Model Institute of Engineering and Technology (Autonomous), Jammu”,** who carried out the Industry Internship at “Remark skill, Ramnagar Delhi” work under my mentorship during July, 2022-August, 2022.

**MS Vishalika**

**Mentor-Internal Supervisor**

**Assistant Professor**

**Computer Science Engineering, MIET**

*This is to certify that the above statement is correct to the best of my knowledge.*

**Dr (Professor) Ashok Kumar**

**Dean Academic Affairs MIET**

**Head of the Department**

**Computer Science Engineering, MIET**

**ACKNOWLEDGEMENTS**

This summer internship opportunity was a great chance for learning and professional development. I am grateful for having a chance to meet so many wonderful people and professionals who led me through this internship period.

It is my pleasant duty to pay my heartfelt gratitude to Mr. Saksham Madan Robotic Software Developer, Remark skill, Ramnagar Delhi who has guided me through the course of this Internship.

I must record my deep sense of gratitude to Dr. (Prof.) Ankur Gupta (Director, MIET) and Dr. (Prof.) Ashok Kumar (Dean Academic Affairs; HOD CSE, MIET) for their guidance, constant inspiration and encouragement, and their keen involvement throughout the course of the present work.

Gratitude and appreciation although mean a very small thing to convey my thanks to my parents who have always given me a parental source of love, motivation and strength right from the journey of my life.

Bearing in mind the previous I am using this opportunity to express my deepest gratitude and special thanks to the teachers who in spite of being extraordinarily busy with their duties, took time out to hear, guide and keep me on the correct path and allowing me to carry out my project at their esteemed organization and extending during the training.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives.

Hope to continue cooperation with all of you in the future.

I express my sincere gratitude to Remark Skill and Model Institute of Engineering and Technology (Autonomous), Jammu for giving me the opportunity.

**Shafin Showkat**

**2020A1R084**

**ABSTRACT**

In 2019, COVID-19 quickly spread across the world, infecting billions of people and disrupting the normal lives of citizens in every country. Governments, organizations, and research institutions all over the world are dedicating vast resources to research effective strategies to fight this rapidly propagating virus. With virus testing, most countries publish the number of confirmed cases, dead cases, recovered cases, and locations routinely through various channels and forms. This important data source has enabled researchers worldwide to perform different COVID-19 scientific studies, such as modeling this virus’s spreading patterns, developing prevention strategies, and studying the impact of COVID-19 on other aspects of society. However, one major challenge is that there is no standardized, updated, and high-quality data product that covers COVID-19 cases data.

Covid data for different areas/states is not available collectively for research and visualization so different organizations/individuals have to go across sites to collect this data. With scrapper the different data can be collected and a single dataset can be created.

The aim was to develop an interface which will show the active cases, total cases, discharged and deaths in India. The data has been taken from the website ([www.mygov.in](http://www.mygov.in)). This interface allows real time monitoring which means the information is updated time to time. It has a GUI based desktop which shows the active cases, total cases, deaths, and discharged patients. The data is automatically changed if there is any change in the Covid cases in India. The graphical user interface is made using Tkinter module because of its classes and methods present in it. The requests and Beautifulsoup is used for web scraping. Request is an HTTP library that provides easy functionality to deal with http request/response in web applications. The beautiful soup is a python library to pulling out data from HTML and Xml files.

**Contents**

Candidates’ Declaration i

Internship Certificate ii

Experience Letter iii

Certificate iv

Acknowledgement v

Abstract vi

Contents vii List of Tables viii

List of Figures ix Abbreviations Used x

**Chapter 1 Basics of Programming**

1.1What is programming**?**

1.2 Algorithm and Flowcharts

1.3 Use of Python Programming

1.4 Programming and Interpretation

**Chapter 2 Program structure in Python**

2.1 What are packages in python?

**2.2 Application of API’s**

**2.3 Back-end of python**

**Chapter 3 Data types and variable scope**

4.1Data types examples

4.2 Understanding variable scope

**Chapter 4 Flow Control Statements**

6.1 If-else statements

6.2 Loops control Statements in python

**Chapter 5 Exception Handling**

7.1 What are exceptions?

7.2 Types of exceptions

**Chapter 6 File handling using python**

8.1 Opening and Reading a file

8.2 Writing Bytes to a text file

**Chapter 7 Modules in Python**

9.1 Importing statements

**Chapter 8 Concept of OOPS and their real-life implementattiom**

10.1 History of OOPS

Chapter 9 Introduction to Data Analystics and Visualization in Python

9.1 Introduction to Numpy, Pandas, and Matplotlib

9.2 Data Wrangling

9.3 Data Loading

**REFERENCES**

**Chapter 1**

**Basics OF Programming**

* 1. **What is programming?**

Programming is writing computer code to create a program, to solve a problem. Programs are created to implement algorithms. Algorithms can be represented as pseudocode or a flowchart, and programming is the translation of these into a computer program.

To tell a computer to do something, a program must be written to tell it exactly what to do and how to do it. If an algorithm has been designed, the computer program will follow this algorithm, step-by-step, which will tell the computer exactly what it should do.

* 1. **Algorithm and Flowcharts**

**Algorithms:-**

 A set of instructions or rules expressed in a step-by-step fashion that represents the procedure to solve the problem to reach the required output is called an Algorithm.

* Irrespective of the programming language, the algorithm is the first step to solve any problem. Hence, an algorithm is independent of the language in which we want to write the program.
* We can develop a flowchart from the algorithm to analyze the problem.

### Characteristics of algorithms:

The difference between just a random explanation of how to solve a problem and representing an algorithm lies in the characteristics of an algorithm.

An algorithm must be:

1. **Clear and unambiguous:** Writing an algorithm aims to get a clarified view of the problem. Hence, the algorithms must be an explanation and should not rise to more confusion. The steps must be clear and should have a straightforward meaning.
2. **Well-defined I/p and O/p:** The inputs and the outputs must be mentioned, and every algorithm must have 0 or more inputs and should generate at least 1 output.
3. **Finite:** The algorithm can be lengthy depending on the need of the problem, but it should have a finite number of steps.
4. **Feasible:** Writing a program based on the algorithm must be feasible for the programmer.
5. **Independent of the platform:** The algorithm for a particular problem must work the same in all programming languages.

### Developing an algorithm:

There are no standards on how an algorithm must be. Any algorithm must have all the characteristics mentioned above and be understandable to a new reader. We can use common concepts of programming languages like conditional statements and loops while developing an algorithm.

## Aim: To add two integer numbers.

**Algorithm:**

**Step 1**: START

**Step 2**: Declare three integers: a, b and sum.

**Step 3**: Take the values of the two numbers a and b from the user.

**Step 4**: Add a and b and store the value in the variable sum.

**Step 5**: Print the value of the sum

**Step 6**: STOP

Adding two numbers is easy. Hence, this algorithm does not have much to gain to write the program. But algorithms play a major role in solving bigger and more complex problems.

Rather than writing definitions, we can even simplify an algorithm as below:

**Step 1**: START

**Step 2**: Take values of a and b

**Step 3**: sum <- a + b

**Step 4**: Display the sum

**Step 5**: STOP

### Advantages of writing algorithms:

1. Gives a clear view to the programmer as it makes the program easy to understand.

### Disadvantages:

1. Creating an algorithm for big problems can be time-taking.
2. As the problem becomes complex, the algorithm becomes bulky, and the level of simplicity decreases, making it hard to grasp.
3. Loops and branches are hard to show in the algorithms.

**Flowcharts**

A flowchart is a graphical or symbolic representation of a process. Each step in the process is represented by a different symbol and contains a short description of the process step. The flowchart symbols are linked together with arrows showing the flow of the process.

* A flowchart is basically a plan to be followed when the program is being written. It acts as a road map for a programmer and guides him on how to go about it from the starting point to the final point while writing the computer program.

For a beginner, it is strongly recommended that the flowchart must be drawn first before writing the program to reduce errors and omissions. Moreover, the flowchart is helpful in testing and modifying the program.

**Guidelines for Drawing a Flowchart**

(a) There should be only one entry/starting point and one exit point of the flowchart.  
(b) Use the correct symbol at each stage in the flowchart.

(c) The logical flow should clearly be shown using arrows.

(d) Use connectors to reduce the number of flow lines.  
(e) Lines should ideally be vertical or horizontal.

**Flowcharting Rules**

1) First, think of the mainline of logic, and then incorporate details.

2) Do not give every detail on the flowchart. A reader who is interested in greater details can refer to the program itself.

3) Statements in the flowchart symbols should be precise and easy to understand.

4) Be consistent in using names and variables in the flowchart. Use descriptive variable names.

5)

Go from left to right and top-to-bottom while constructing flowcharts.

6) Keep the flowchart simple. The crossing of flow lines should be avoided.

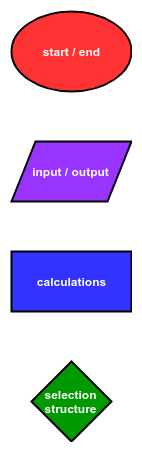
7) If a new flowcharting page is needed, it is recommended that the flowchart is broken at an input or output point. Moreover, properly labeled connectors should be used to link the portions of the flowchart on different pages.

**Advantages of using Flowcharts Convey Better Meaning**

Since a flowchart is a pictorial representation of the solution of a problem, it is easier for a programmer to understand the logic of the program.

There are four basic shapes used in a flow chart. Each shape has a specific use:

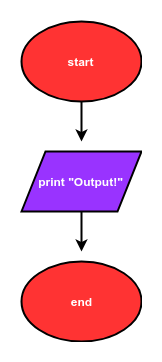
* oval: start / end
* parallelogram: input / output
* rectangle: calculations
* diamond: selection structures



Arrows connect the basic shapes in a flowchart. The shapes and arrows of a flowchart describe the flow of a program from start to end. Flowcharts typically flow from the top to the bottom or flow from the left to the right. Below is the description of a simple program:

The program starts. Then the program prints out "Output!". Finally, the program ends.

A flowchart that describes this simple program is shown.



The Python code that corresponds to this flowchart is:

# Start

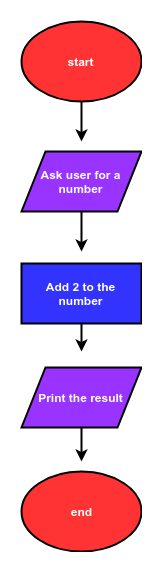
Print("Output!")

# End

A description of a program that includes a calculation is below:

The program starts. Next, the program asks a user for a number. Two is added to the number. Next, the resulting sum is printed. Finally, the program ends.

A flowchart that describes this program is is shown.



The Python code that corresponds to this flow chart is:

# start

num = input("Enter a number: ")

num = float(num)

num\_plus\_2 = num + 2

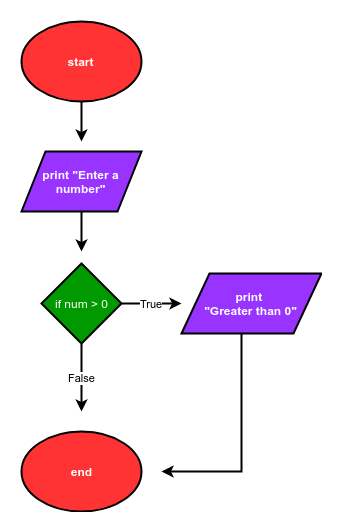
print(num\_plus\_2)

# end

The description of another program is below:

The program starts. Next the program asks a user for a number. If the number is greater than zero, the program prints "Greater than 0", then the program ends.

A flow chart that describes this program is shown.



The Python code that corresponds to this flow chart is:

# start

num = input("Enter a number: ")

num = float(num)

if num>0:

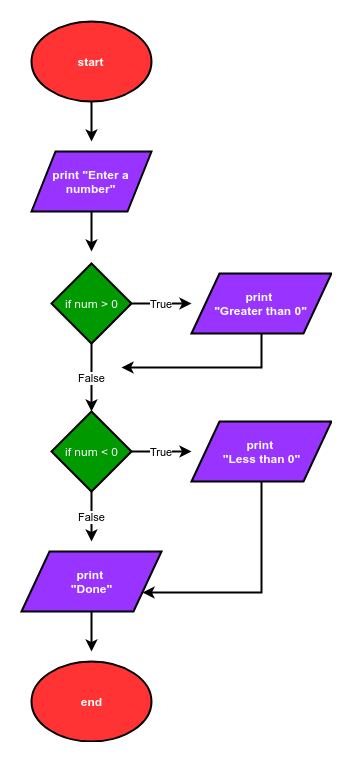
print("Greater than 0")

# end

The description of a more complex program is below:

The program starts. Next, the program asks a user for a number. If the number is greater than zero, the program prints "Greater than 0". If the number is less than zero, the program prints "Less than 0". Then the program prints "Done" and the program ends.

A flowchart that describes this program is below:



The Python code that corresponds to this flow chart is:

# start

num = input('Enter a number: ')

num = float(num)

if num>0:

print('num greater than zero')

if num<0:

print('num less than zero')

print('Done')

# end

**1.3 Use of Python Programming**

Python is a open-source, object-oriented and straightforward programming language. It is a dynamic type, high-level, and interpreted coding language, making it one of the easiest programming languages to learn. This is also used for error debugging and to encourage the rapid development of application prototypes and the use of it as a programming language.

Moreover, the Python programming language has been growing in popularity over the past few years. Indeed, it is now one of the most popular programming languages out there. It is used in a wide variety of areas from web development to artificial intelligence and machine learning. This blog will explore some of the top uses of Python programming.

## ****Top Uses of Python Programming****

### ****1. Web Development****

Python, well-known as the go-to programming language for web development, plays an important role in it.

Python includes several web development frameworks, including Pyramid, Django, and Flask. These frameworks include standard libraries that facilitate protocol integration and reduce development time.

### ****2. Desktop GUI****

Desktop GUI is another uses of of Python programming. It allows game developers to create interactive games.

Additionally, it makes use of various libraries, such as PySoy, a 3D game engine that supports Python 3. Various game developers use another library PyGame, which provides functionality for games such as Disney’s Toon town Online, Civilization-IV, Vega Strike, and many more, all of which are written in Python.

### ****3. Data Science****

We are all aware that data science is one of the most sought-after skills in the market. Data science knowledge is in high demand in IT, manufacturing, and ecommerce. Python comes into play here.

However, it’s numerous libraries, such as Pandas, Tensor Flow, NumPy, and others, aid in the extraction of valuable information from data. Libraries such as Matplotlib and Sea born allow a data scientist to focus on data visualization through graphs and charts. It is not an exaggeration to say that Python is the first language that any data science professional should learn.

### ****4. Development of games****

It is another uses of Python programming language, as it allows game developers to create interactive games. It makes use of various libraries, such as PySoy, a 3D game engine that supports Python 3.

However, various game developers use another library PyGame, which provides functionality for games such as Disney’s Toon town Online, Civilization-IV, Vega Strike, and many more, all of which are written in Python.

### ****5. Artificial Intelligence and Machine Learning****

AI is one of Python’s most major uses. The reason for this is that Python is a stable language capable of handling the computations needed to build machine learning models.

Furthermore, its libraries, such as Keras, Pandas, and NumPy, are suitable for -learning applications. It is also used in a variety of AI solutions such as advanced computing, image recognition, data processing, and others.

### ****6. Business Applications****

This is another uses of Python programming because business applications differ from standard applications that cover domains such as ERP, E-Commerce, and many more.

However, they require extensible, scalable, and readable applications. Python provides all of these features to its users. Other platforms, such as Tryton, are used to develop these business applications.

### ****7. Web Scraping Application****

This programming language is used to collect a large amount of information from other websites. Which can be useful in a variety of processes such as job listings, price comparison, development and research, and so on.

Additionally, it enables the Beautifulsoup library, which is used to collect information that can then be processed.

1.4 Python Interpreter

An Interpreter is a program that converts the code a developer writes into an intermediate language, called the byte code. It converts the code line by line, one at a time. It translates till the end and stops at that line where an error occurs, if any, making the debugging process easy.

Python and Java are two examples of the interpreted programming languages.

### What happens when you run a Python Code?

On running a Python code, say a simple program that prints “Hello, World!”, then the following steps happen:

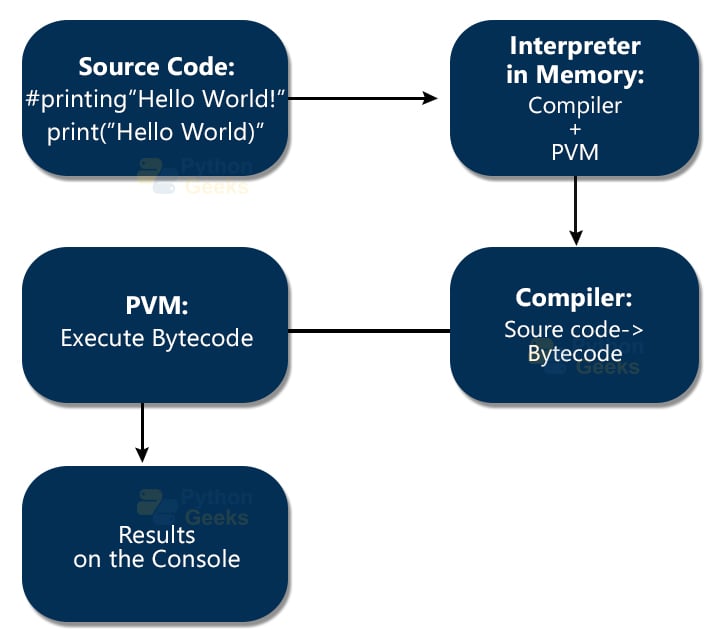
1. The program gets converted into an intermediate code, called bytecode. This is also binary but is not understood by the processor.

2. Python Interpreter, stored in the memory as a collection of instructions in binary form, does the above conversion. These instructions can be thought of as a combination of a compiler and a Python Virtual Machine (PVM).

3. The compiler converts the source code, the “Hello, World!” program, into the byte code, which is platform-independent and understood by the PVM.

4. PVM reads the byte code and executes it on the hardware, the job of a processor.

This is how a Python interpreter works. The following chart summarizes the above process.

[](https://pythongeeks.org/wp-content/uploads/2021/06/Internal-process-happening-on-running-the-code.jpg)

### Features of Python Interpreter

The Python Interpreter is user friendly and its features include:

* Interactive editing
* Use of variables initialized in the previous prompts
* Writing the complete code in it with a redline facility.
* To get command-line editing, one can press Ctrl+P, which gives a beep indicating the mode is activated.

### Getting started with the Python Interpreter

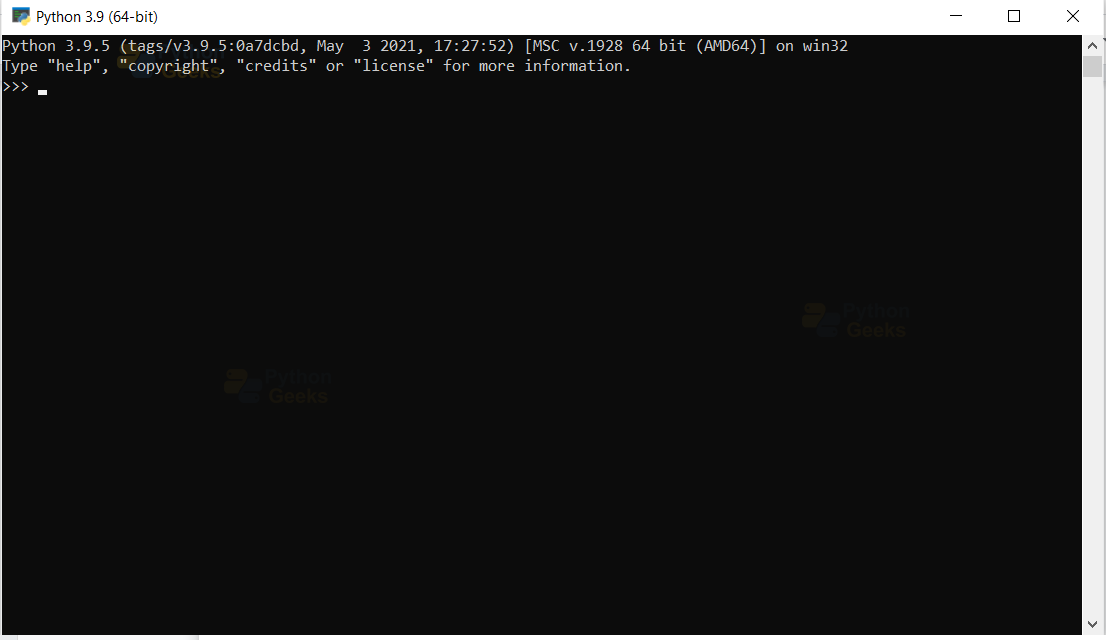
To open the Python interpreter, installed in the system, search in the Start menu. Then click on Python 3.9 or other, depending on the installed version.

* In Windows, it looks like Command Prompt.
* And on Mac, it looks like a terminal.

The interpreter environment works using REPL:

* Read the lines of the code
* Evaluate and execute the code
* Print the output of the code (if any) to the console
* Looping to repeat the above process

The picture below shows the Python interpreter once opened. Small description of the Python version, storage, etc. Along with this, you can see three arrows (>>>), called prompt. This is the place where you have to write the code.

[](https://pythongeeks.org/wp-content/uploads/2021/06/1.png)

### Coding in the Python Interpreter Environment

The interpreter allows interactive editing. Once you write the code, press the “Enter” button to see the results. The below example shows a simple code of printing “Hello, World!”

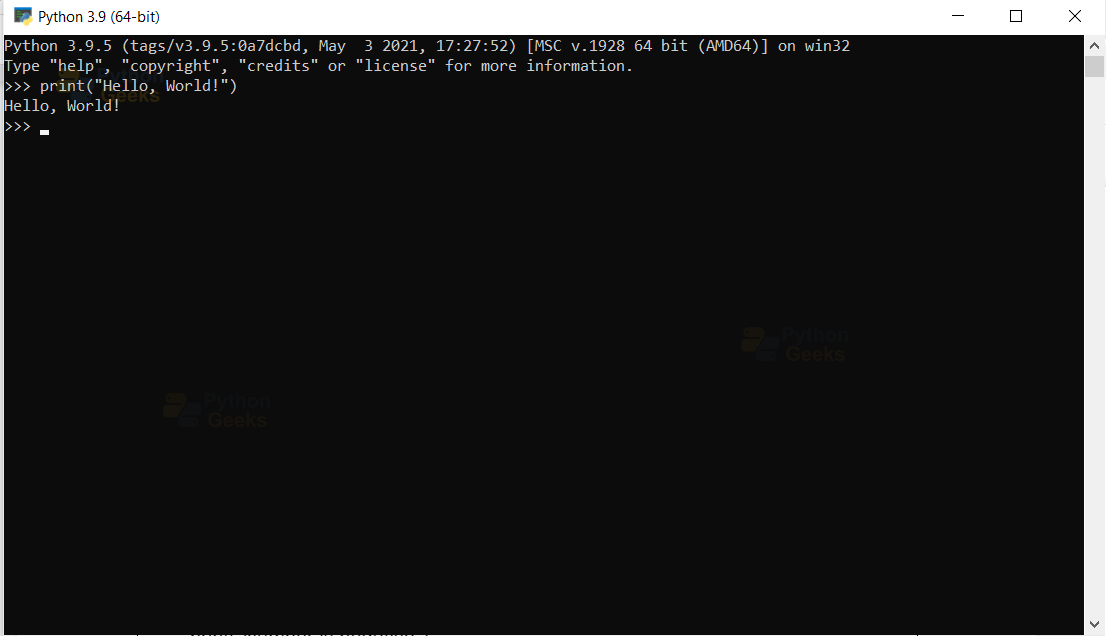
**“Hello, World!” program in Interpreter:**

print("Hello, World!")

**Output**

Hello, World!

After the result, the interpreter gives another prompt for the next program/code. You can see this in the below picture.

[](https://pythongeeks.org/wp-content/uploads/2021/06/2.png)

To write multiple line codes, the interpreter allows for the continuation.

**Chapter2-Program Structure in Python:**

**2.1 packages in python**

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-subpackages, and so on.

Consider a file Pots.py available in Phone directory. This file has following line of source code −

#!/usr/bin/python

def Pots():

print "I'm Pots Phone"

Similar way, we have another two files having different functions with the same name as above −

* Phone/Isdn.py file having function Isdn()
* Phone/G3.py file having function G3()

Now, create one more file \_\_init\_\_.py in Phone directory −

* Phone/\_\_init\_\_.py

To make all of your functions available when you've imported Phone, you need to put explicit import statements in \_\_init\_\_.py as follows −

from Pots import Pots

from Isdn import Isdn

from G3 import G3

After you add these lines to \_\_init\_\_.py, you have all of these classes available when you import the Phone package.

#!/usr/bin/python

# now import your Phone Package.

Import Phone

Phone.Pots ()

Phone.Isdn ()

Phone.G3 ()

When the above code is executed, it produces the following result −

I'm Pots Phone

I'm 3G Phone

I'm ISDN Phone

In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

2.2 Application of APIs

An Application Programming interface is a software interface that helps in connecting between the computers or between computer programs. It is an interface that provides the accessibility of information such that weather forecasting. In simple words, you can say it is a software interface that offers service to the other pieces of software.

**Example –**  
Best examples of web services APIs are- SOAP REST

**Features:**

* An application programming interface is a software that allows two applications to talk to each other.
* Application programming interface helps in enabling applications to exchange data and functionality easily.
* The application programming interface is also called a middle man between two systems.
* Application programming interface helps in data monetization.
* Application programming interface helps in improving collaboration.

**Different types of APIs :**  
These are common types of APIs as follows.

1. **Open APIs –**  
   It is also called public APIs which are available to any other users. Open APIs help external users to access the data and services. It is an open-source application programming interface in which we access with HTTP protocols.
2. **Internal APIs –**   
   It is also known as private APIs, only an internal system exposes this type of APIs. These are designed for the internal use of the company rather than the external users.
3. **Composite APIs –**   
   It is a type of APIs that combines different data and services. The main reason to use Composites APIs is to improve the performance and to speed the execution process and improve the performance of the listeners in the web interfaces.
4. **Partner APIs –**    
   It is a type of APIs in which a developer needs specific rights or licenses in order to access. Partner APIs are not available to the public.

**Applications of API**  
Here, are some real-world applications are as follows.

* **Weather snippets –**  
  In weather snippets, APIs are generally used to access a large set of datasets to access the information of weather forecast which is very helpful information in day-to-day life.
* **Login –**  
  In this functionality, APIs are widely used to log in via Google, Linked In, Git Hub, and Twitter and allow users to access the log-in portal by using the API interface.
* **Entertainment –**  
  In this field, APIs are used to access and provide a huge set of databases to access movies, web series, comedy, etc.
* **E-commerce website –**   
  In this, APIs provide the functionality like if you have purchase something, and now you want to pay so, API provides interface like you can pay using different bank debit cards, UPI(Unified Payments Interface), credit card, wallet, etc.
* **Gaming –**   
  In gaming, it provides an interface like you can access the information of the game, and you can connect to different users and play with different-different users at the same time.

**2.3 Back-end of Python**

The backend is the server side of the website. It stores and arranges data, and also makes sure everything on the client side of the website works fine. It is the part of the website that you cannot see and interact with. It is the portion of software that does not come in direct contact with the users. The parts and characteristics developed by backend designers are indirectly accessed by users through a front-end application. Activities, like writing APIs, creating libraries, and working with system components without user interfaces or even systems of scientific programming, are also included in the backend.

### ****Back End Languages****

The back-end portion is built by using some languages which are discussed below:

* **PHP:** PHP is a server-side scripting language designed specifically for web development. Since PHP code is executed on the server side, so it is called a server-side scripting language.
* [**C++**](https://www.geeksforgeeks.org/cpp-tutorial/)**:** It is a general-purpose programming language and is widely used nowadays for competitive programming. It is also used as a backend language.
* [**Java**](https://www.geeksforgeeks.org/java/)**:** Java is one of the most popular and widely used programming languages and platforms. It is highly scalable. Java components are easily available and for learning this one of the most popular languages. It will help us to understand the proper framework, concepts, functions, and more.
* [**Python**](https://www.geeksforgeeks.org/python-programming-language/)**:** Python is a programming language that lets you work quickly and integrate systems more efficiently. It is also a very important language for the back end..
* [**Node.js**](https://www.geeksforgeeks.org/nodejs/)**:** Node.js is an open-source and cross-platform runtime environment for executing JavaScript code outside a browser. You need to remember that NodeJS is not a framework, and it’s not a programming language. Most people are confused and understand it’s a framework or a programming language. We often use Node.js for building back-end services like APIs like Web App or Mobile App. It’s used in production by large companies such as Paypal, Uber, Netflix, Walmart, and so on.

### ****Back-End Frameworks:****

* **Express –**Express is a Nodejs framework used for backend/server-side development. It is used to build single-page, multi-page, and hybrid web applications. With its help, you can handle multiple different HTTP requests.
* [**Django**](https://www.geeksforgeeks.org/django-tutorial/)– Django is a Python web-based framework, following the model-template-views pattern. It is used to build large and complex web applications. Its features include being fast, secure, and scalable.
* Ruby on rails– Ruby on Rails is a server-side framework following the model-view-controller architecture pattern. It provides default structures such as web services, web pages, and databases.
* Laravel– Laravel is a web application framework for PHP and is robust. The feature which makes it perfect is reusing the components of different frameworks for creating a web application.
* – This server-side framework provides infrastructure support for Java applications. It acts as a support to various frameworks like Hibernate, Struts, EJB, etc. It also has extensions that help in developing Java applications quickly and easily.
* Some more back-end programming/scripting languages are C#, Ruby, GO, etc.

**Chapter3- Data,Object type and Variable type**

**3.1 Data Type Examples**

In Python programming language, a data type defines the type of a variable. Python programming does not require defining the data type of a variable. It is a dynamic type of language. The interpreter implicitly binds the value with its type.

There are various data types in Python. Some of the important types are listed below:

* Numbers Data Type
* List Data Type
* Tuple Data Type
* Strings Data Type
* Set Data Type
* Dictionary Data Type

**Note:-**If you want to check the data type of a variable, what is the data type of the variable, then type () function is used to check the data type of a variable in Python.

In Python, below is the program for data type checking:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | a = 'string'  b = 50  c = 10.5    print(type(a));  print(type(b));  print(type(c)); |

The output of the above program is following:

|  |  |
| --- | --- |
| 1  2  3 | <type 'str'>  <type 'int'>  <type 'float'> |

### Numbers Data Type

In python programming, Integer, floating-point numbers, and complex numbers under the category of Python numbers. Those you define as int, float and complex classes.

**For example:**

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8 | a = 10  print(a, "is of type", type(a))    a = 50.0  print(a, "is of type", type(a))    a = 11+2j  print(a, "is complex number?", isinstance(11+2j,complex))  **Output of the above program is following:**   |  |  | | --- | --- | | 1  2  3 | 10 is of type <class 'int'>  50.0 is of type <class 'float'>  (11+2j) is complex number? True | |

### List Data Type

In python programming, list data types hold different types of data. it does not need to be of the same type. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

**Note:-** List data types are alike to arrays in PHP, C

You can use a slicing operator [] to extract items from a list.

**For example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | a = [10, 5.5, 'python']    print(a)    a = [5,10,15,20,25,30,35,40]    # a[2] = 15  print("a[2] = ", a[2])    # a[0:3] = [5, 10, 15]  print("a[0:3] = ", a[0:3])    # a[5:] = [30, 35, 40]  print("a[5:] = ", a[5:]) |

The output of the above program is following:

|  |  |
| --- | --- |
| 1  2  3  4 | [10, 5.5, 'python']  a[2] =  15  a[0:3] =  [5, 10, 15]  a[5:] =  [30, 35, 40] |

### Tuple Data Type

In python programming, a tuples data type is alike to list data type. But it is only one difference, once tuples created can not be changed/modify.

Tuples contain the collection of the items of different data types alike list data type. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

**For example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | a = (8,'python', 1+2j)    # a[1] = 'program'  print("a[1] = ", a[1])    # a[0:3] = (5, 'program', (1+3j))  print("a[0:3] = ", a[0:3]) |

Output of the above code is following:

|  |  |
| --- | --- |
| 1  2 | a[1] =  python  a[0:3] =  (8, 'python', (1+2j)) |

### Strings Data Type

In python programming, A string is a sequence of characters in Python. In python, Strings are either enclosed with single quotes, double, and triple quotes.

Note:- In python, Multi-line strings can be denoted using triple quotes, ”’ or “””.

**For example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | string = 'Hello world!'    # string[1] = 'e'  print("string[1] = ", string[1])    mString = """Hello world!                this is multi                line string example"""    print(mString) |

Output of the above program is the following:

|  |  |
| --- | --- |
| 1  2  3  4 | string[1] =  e  Hello world!                this is multi                line string example |

### Set Data Type

In python programming, set data types hold unordered collection of unique items. The items stored in the set data types are separated with a comma (,) and enclosed within square brackets { }.

**For example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | abc = {5,2,3,1,4}    # printing set variable  print("abc = ", abc)    # data type of variable a  print(type(abc)) |

Output of the above program is following:

|  |  |
| --- | --- |
| 1  2 | abc =  {1, 2, 3, 4, 5}  <class 'set'> |

### Dictionary Data Type

In python programming, Dictionary data types is held data in key and value pairs.

**Note:-** A dictionary data type does not allow duplicate keys in collection. But the values ​​can be duplicate.

**For example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | dict = {1:"john","lastname":"smith", "age":25}    # prints the value where key value is 1  print(dict[1])    # prints the value where key value is "lastname"  print(dict["lastname"])    # prints the value where key value is "age"  print(dict["age"]) |

Output of the above program is following:

|  |  |
| --- | --- |
| 1  2  3 | john  smith  25 |

**3.2 Understanding Variable Scope**

a variable is a name that stores the values/data. For example, to store the text “PythonGeeks” we can use the variable ‘text’ to store as shown below:

**Example of variable in Python:**

text="PythonGeeks" #variable 'text' stores "PythonGeeks"

print(text)

**Output:**

PythonGeeks

In Python, the scope of a variable refers to the part of the code where we can access the variable. We cannot access all the variables in the code from any part. For instance, look at the below example. Here we use ‘def func()’ to create a function with some statements to perform operations.

**Example of scope of Variable in python:**

name="PythonGeeks"

**def** func():

name2="Python"

print(name2)

print(name)

func()

**Output:**

Python  
PythonGeeks

We could access both the variables ‘name’ and ‘name2’ inside the function ‘func()’. What happens if we try to do the same outside the ‘func()’?

**Example to show that scope of a variable in a function ends after the function:**  
print(name)  
print(name2)

**Outputs:**

PythonGeeks  
Traceback (most recent call last):  
File “<pyshell#8>”, line 1, in <module>  
print(name2)  
NameError: name ‘name2’ is not defined

We could access ‘name’ but we got an error ‘name2’. This is because the scope of ‘name2’ is only within the ‘func()’.

### Different types of variable scope in Python

Learning about what variable scope is, we will now dive into different types of scope. Based on the part of the code where we do the initialization of the variable, their scopes are of four types:

#### a. Local Scope in Python

When a variable is inside a function, then it is accessible only inside that function. This variable exists only while the function is executing and is called a local variable.

**Example of local variable and showing that we cannot access it outside a function:**

**def** func():

a="PythonGeeks"

print("Inside the function: value of a is ",a)

func()

print(a)

**Output:**

Inside the function: value of a is PythonGeeks

Traceback (most recent call last):  
File “<pyshell#5>”, line 1, in <module>  
print(a)  
NameError: name ‘a’ is not defined

We see that we can print the variable ‘a’ only inside the ‘func(). But we get an error accessing it outside the function. This variable has local scope, that is the function ‘func()’.

#### b. Global Scope of variable in Python

This is the scope of the variable that is available from any part of the code.The variable is initialized outside the functions and is called global variable.

**Example of creating a global variable:**

text="PythonGeeks"

**def** func():

print("Inside the function, the value of text is ", text)

func()

print(text)

**Output:**

Inside the function, the value of text is PythonGeeks  
PythonGeeks

In the above example, the variable ‘text’ has global scope. This is because we could access it outside and inside the function ‘func().

#### c. Enclosed Scope in Python

This is the scope of the variable inside a function with a nested function (i.e., function inside another function). This variable is neither global nor local, so it is also called nonlocal scope.

To understand it clearly, look at the following example:

**Example of enclosed scope:**

**def** func():

s="PythonGeeks"

**def** nested\_func(): #nested function

p="Python"

print("s=",s)

print("p=",p)

nested\_func() #calling nested function

print("s=",s)

func()

**Output:**

s= PythonGeeks  
p= Python  
s= PythonGeeks

Here, ‘p’ has the local scope of the ‘nested\_func()’, and ‘s’ has a nonlocal scope with ‘nested\_func().

#### d. Built-in Scope in Python

All the keywords come under this scope. We can call them from any part of the program and have specific predefined purposes. These get loaded when the interpreter starts and there is no need for importing them separately.

This is the largest scope. Some examples of keywords are: print, def, True, if, else, type, and so on.

**Example of built-in variable:**

s="Python"

print(s)

**def** func():

**if**(**True**):

print("Inside the function")

func()

**Output:**

Python

Inside the function

In the above example we could use the keywords at any part of the program.

**Chapter4 Flow Control Statements**

**4.1 If Else Statements**

An**if else statement** in Python means: "When the if expression evaluates to True, then execute the code that follows it. But if it evalates to False, then run the code that follows the else statement" The else statement is written on a new line after the last line of indented code and it can't be written by itself.

## How do we use the if-else statement?

If-else statements are a form of conditional logic. Essentially, what that means is

1. We test a condition. For example, whether a given variable equals another given variable.
2. If the condition is true, we execute the following block of code.
3. And if the condition is false, we execute a different block of code.

This is absolutely critical to any sort of programming. You cannot have turing-complete programming languages without some sort of conditional logic. In Python, that means lots of if-else statements.

4.2 Loop control statements in Python

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements. Click the following links to check their detail.

Let us go through the loop control statements briefly

| **Sr.No** | **Operator & Description** |
| --- | --- |
| 1 | **Break statement**  terminates the loop statement and transfers execution to the statement immediately following the loop. |
| 2 | [continue statement](https://www.tutorialspoint.com/python/python_continue_statement.htm) Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| 3 | [pass statement](https://www.tutorialspoint.com/python/python_pass_statement.htm) The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. |

**Chapter 5 Exception Handling**

**5.1 What are Exceptions?**

n exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

## Handling an exception

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a **try**: block. After the try: block, include an **except**: statement, followed by a block of code which handles the problem as elegantly as possible.

## Syntax

Here is simple syntaxof try....except...else blocks −

try:

   You do your operations here;

   ......................

except ExceptionI:

   If there is ExceptionI, then execute this block.

except ExceptionII:

   If there is ExceptionII, then execute this block.

   ......................

else:

   If there is no exception then execute this block.

Here are few important points about the above-mentioned syntax −

* A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
* You can also provide a generic except clause, which handles any exception.
* After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
* The else-block is a good place for code that does not need the try: block's protection.

## Example

This example opens a file, writes content in the, file and comes out gracefully because there is no problem at all

#!/usr/bin/python

try:

   fh = open("testfile", "w")

   fh.write("This is my test file for exception handling!!")

except IOError:

   print "Error: can\'t find file or read data"

else:

   print "Written content in the file successfully"

   fh.close()

## Output

This produces the following result −

Written content in the file successfully

5.2 types of exceptions

Exceptions can either belong to the in-built errors/exceptions or have custom exceptions. Some of the **common in-built exceptions** are as follows:

1. Zero Division Error
2. Name Error
3. Indentation Error
4. IO Error
5. EOF Error

**Chapter 6-File handling using Python**

**6.1 Opening and Reading a File**

Python has a well-defined methodology for opening, reading, and writing files. Some applications for file manipulation in Python include: reading data for algorithm training and testing, reading files to create generative art, reporting, and reading configuration files.

Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).

Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.

When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order:

1. Open a file
2. Read or write (perform operation)
3. Close the file

## Opening Files in Python

Python has a built-in open() function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

>>> f = open("test.txt") # open file in current directory

>>> f = open("C:/Python38/README.txt") # specifying full path

We can specify the mode while opening a file. In mode, we specify whether we want to read r, write w or append a to the file. We can also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file.

On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like images or executable files.

|  |  |
| --- | --- |
| Mode | Description |
| r | Opens a file for reading. (default) |
| w | Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists. |
| x | Opens a file for exclusive creation. If the file already exists, the operation fails. |
| a | Opens a file for appending at the end of the file without truncating it. Creates a new file if it does not exist. |
| t | Opens in text mode. (default) |
| b | Opens in binary mode. |
| + | Opens a file for updating (reading and writing) |

f = open("test.txt") # equivalent to 'r' or 'rt'

f = open("test.txt",'w') # write in text mode

f = open("img.bmp",'r+b') # read and write in binary mode

Unlike other languages, the character a does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).

Moreover, the default encoding is platform dependent. In windows, it is cp1252 but utf-8 in Linux.

So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

f = open("test.txt", mode='r', encoding='utf-8')

## Closing Files in Python

When we are done with performing operations on the file, we need to properly close the file.

Closing a file will free up the resources that were tied with the file. It is done using the close() method available in Python.

Python has a garbage collector to clean up unreferenced objects but we must not rely on it to close the file.

f = open("test.txt", encoding = 'utf-8')

# perform file operations

f.close()

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a [try...finally](https://www.programiz.com/python-programming/exception-handling) block.

try:

f = open("test.txt", encoding = 'utf-8')

# perform file operations

finally:

f.close()

This way, we are guaranteeing that the file is properly closed even if an exception is raised that causes program flow to stop.

The best way to close a file is by using the with statement. This ensures that the file is closed when the block inside the with statement is exited.

We don't need to explicitly call the close() method. It is done internally.

with open("test.txt", encoding = 'utf-8') as f:

# perform file operations

## Writing to Files in Python

In order to write into a file in Python, we need to open it in write w, append a or exclusive creation x mode.

We need to be careful with the w mode, as it will overwrite into the file if it already exists. Due to this, all the previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using the write() method. This method returns the number of characters written to the file.

with open("test.txt",'w',encoding = 'utf-8') as f:

f.write("my first file\n")

f.write("This file\n\n")

f.write("contains three lines\n")

This program will create a new file named test.txt in the current directory if it does not exist. If it does exist, it is overwritten.

We must include the newline characters ourselves to distinguish the different lines.

## Reading Files in Python

To read a file in Python, we must open the file in reading r mode.

There are various methods available for this purpose. We can use the read(size) method to read in the size number of data. If the size parameter is not specified, it reads and returns up to the end of the file.

We can read the text.txt file we wrote in the above section in the following way:

>>> f = open("test.txt",'r',encoding = 'utf-8')

>>> f.read(4) # read the first 4 data

'This'

>>> f.read(4) # read the next 4 data

' is '

>>> f.read() # read in the rest till end of file

'my first file\nThis file\ncontains three lines\n'

>>> f.read() # further reading returns empty sting

''

We can see that the read() method returns a newline as '\n'. Once the end of the file is reached, we get an empty string on further reading.

We can change our current file cursor (position) using the seek() method. Similarly, the tell() method returns our current position (in number of bytes).

>>> f.tell() # get the current file position

56

>>> f.seek(0) # bring file cursor to initial position

0

>>> print(f.read()) # read the entire file

This is my first file

This file

contains three lines

We can read a file line-by-line using a [for loop](https://www.programiz.com/python-programming/for-loop). This is both efficient and fast.

>>> for line in f:

... print(line, end = '')

...

This is my first file

This file

contains three lines

In this program, the lines in the file itself include a newline character \n. So, we use the end parameter of the print() function to avoid two newlines when printing.

Alternatively, we can use the readline() method to read individual lines of a file. This method reads a file till the newline, including the newline character.

>>> f.readline()

'This is my first file\n'

>>> f.readline()

'This file\n'

>>> f.readline()

'contains three lines\n'

>>> f.readline()

''

Lastly, the readlines() method returns a list of remaining lines of the entire file. All these reading methods return empty values when the end of file (EOF) is reached.

>>> f.readlines()

['This is my first file\n', 'This file\n', 'contains three lines\n']

**6.2 Writing Bytes to a text file**

Files are used in order to store data permanently. File handling is performing various operations (read, write, delete, update, etc.) on these files. In Python, [file handling](https://www.geeksforgeeks.org/file-handling-python/) process takes place in the following steps:

1. Open file
2. Perform operation
3. Close file

There are four basic modes in which a file can be opened― read, write, append, and exclusive creations. In addition, Python allows you to specify two modes in which a file can be handled― binary and text. Binary mode is used for handling all kinds of non-text data like image files and executable files.

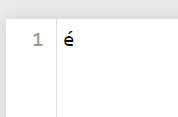
## Write Bytes to File in Python

**Example 1:**Open a file in binary write mode and then specify the contents to write in the form of bytes. Next, use the write function to write the byte contents to a binary file.

## Python3

|  |
| --- |
| some\_bytes = b'\xC3\xA9'    # Open in "wb" mode to  # write a new file, or  # "ab" mode to append  with open("my\_file.txt", "wb") as binary\_file:        # Write bytes to file      binary\_file.write(some\_bytes) |

**Output:**



*my\_file.txt*

**Chapter 7 Modules in Python**

**7.1 python import statement**

Python **Import** statement enables the user to import particular modules in the corresponding program.

It resembles the #include header\_file in C/C++.

As soon as the interpreter encounters the import statement in a particular code, it searches for the same in the local scope and imports the module, if present in the search path. It searches for a particular module in its built-in modules section at first. If it’s not found, it searches those modules in its current directory.

**A module is loaded only once in a particular program, without being affected by the number of times the module is imported.**

**Syntax:**

import module\_name

**Example:**

|  |
| --- |
| import collections |

### 1. Importing class/functions from a module

We can import classes/functions from a module using the syntax:

from {module} import {class/function}

**Example:**

|  |
| --- |
| from collections import OrderedDict  from os import path  from math import pi  print(pi) |

**Output:**

3.141592653589793

### 2. The import \* Statement

All the methods and constants of a particular module can be imported using import \* operator.

|  |
| --- |
| from math import \*  print(pi)  print(floor(3.15)) |

**Output:**

3.141592653589793

3

### 3. Python’s import as Statement

The import as statement helps the user provide an alias name to the original module name.

|  |
| --- |
| # python import as  import math as M    print(M.pi)  print(M.floor(3.18)) |

**Output:**

Report this ad

3.141592653589793

3

### 4. Importing user-defined modules

We can import the functions of one program into another using its name.

Initially, we need to create a python code

|  |
| --- |
| def sub(a, b):      return int(a) - int(b)    def lower\_case(str1):      return str(str1).lower() |

Then create another python script, wherein we need to import the above create test.py script.

|  |
| --- |
| import test    print(test.sub(5,4))  print(test.lower\_case('SafA')) |

**Output:**

1

safa

### 5. Importing from another directory

The **importlib** library is used to import a script from another directory.

Initially, we need to create a python script and define functions in it.

|  |
| --- |
| def sub(a, b):      return int(a) - int(b)    def lower\_case(str1):      return str(str1).lower() |

Then, we will create another python script and save it into another directory and then import the functionalities from test1.py (which resides into another directory).

|  |
| --- |
| import importlib, importlib.util    def module\_directory(name\_module, path):      P = importlib.util.spec\_from\_file\_location(name\_module, path)      import\_module = importlib.util.module\_from\_spec(P)      P.loader.exec\_module(import\_module)      return import\_module    result = module\_directory("result", "../inspect\_module/test1.py")    print(result.sub(3,2))  print(result.lower\_case('SaFa')) |

**Output:**

1

safa

Another alternative way is to add the module directory to the **sys.path** list.

### 6. Importing class from another file

**tests.py**

|  |
| --- |
| class Employee:      designation = ""        def \_\_init\_\_(self, result):          self.designation = result        def show\_designation(self):          print(self.designation)      class Details(Employee):      id = 0        def \_\_init\_\_(self, ID, name):          Employee.\_\_init\_\_(self, name)          self.id = name        def get\_Id(self):          return self.id |

**design.py**

|  |
| --- |
| import importlib, importlib.util    def module\_directory(name\_module, path):      P = importlib.util.spec\_from\_file\_location(name\_module, path)      import\_module = importlib.util.module\_from\_spec(P)      P.loader.exec\_module(import\_module)      return import\_module    result = module\_directory("result", "../Hello/tests.py")    a = result.Employee('Project Manager')  a.show\_designation()    x = result.Details(4001,'Safa')  x.show\_designation()  print(x.get\_Id()) |

**Output:**

Project Manager

Safa

Safa

## Conclusion

Thus, in this article, we have understood the functionalities offered by the import statement.]

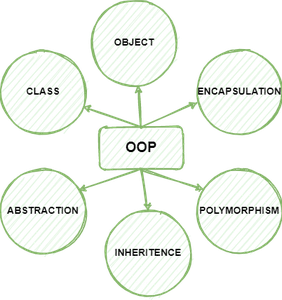
**Chapter 8: Concept of OOPs and their real life implementation**

**8.1History of OOPs**

In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

## Main Concepts of Object-Oriented Programming (OOPs)

* Class
* Objects
* Polymorphism
* Encapsulation
* Inheritance
* Data Abstraction



***OOPs***

## Class

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

To understand the need for creating a class let’s consider an example, let’s say you wanted to track the number of dogs that may have different attributes like breed, age. If a list is used, the first element could be the dog’s breed while the second element could represent its age. Let’s suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it’s the exact need for classes.

**Some points on Python class:**

* Classes are created by keyword class.
* Attributes are the variables that belong to a class.
* Attributes are always public and can be accessed using the dot (.) operator. E.g.: Myclass.Myattribute

**Class Definition Syntax:**

class ClassName:

# Statement-1

.

.

.

# Statement-N

### Example: Creating an empty Class in Python

## Python

|  |
| --- |
| # Python3 program to  # demonstrate defining  # a class    class Dog:      pass |

In the above example, we have created a class named dog using the class keyword.

## Objects

The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 12 is an object, the string “Hello, world” is an object, a list is an object that can hold other objects, and so on. You’ve been using objects all along and may not even realize it.

**An object consists of :**

* **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
* **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
* **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

To understand the state, behavior, and identity let us take the example of the class dog (explained above).

* The identity can be considered as the name of the dog.
* State or Attributes can be considered as the breed, age, or color of the dog.
* The behavior can be considered as to whether the dog is eating or sleeping.

### Example: Creating an object

## Python3

|  |
| --- |
| obj = Dog() |

This will create an object named obj of the class Dog defined above. Before diving deep into objects and class let us understand some basic keywords that will we used while working with objects and classes.

### ****The self****

1. Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it
2. If we have a method that takes no arguments, then we still have to have one argument.
3. This is similar to this pointer in C++ and this reference in Java.

When we call a method of this object as myobject. Method (arg1, arg2), this is automatically converted by Python into MyClass.method (myobject, arg1, arg2) – this is all the special self is about.

**Note:** For more information, refer to [self in Python class](https://www.geeksforgeeks.org/self-in-python-class/)

### ****The \_\_init\_\_ method****

The [\_\_init\_\_ method](https://www.geeksforgeeks.org/__init__-in-python/) is similar to constructors in C++ and Java. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

Now let us define a class and create some objects using the self and \_\_init\_\_ method.

### Example 1: Creating a class and object with class and instance attributes

## Python3

|  |
| --- |
| class Dog:        # class attribute      attr1 = "mammal"        # Instance attribute      def \_\_init\_\_(self, name):          self.name = name    # Driver code  # Object instantiation  Rodger = Dog("Rodger")  Tommy = Dog("Tommy")    # Accessing class attributes  print("Rodger is a {}".format(Rodger.\_\_class\_\_.attr1))  print("Tommy is also a {}".format(Tommy.\_\_class\_\_.attr1))    # Accessing instance attributes  print("My name is {}".format(Rodger.name))  print("My name is {}".format(Tommy.name)) |

**Output**

Rodger is a mammal

Tommy is also a mammal

My name is Rodger

My name is Tommy

### Example 2: Creating Class and objects with methods

## Python3

|  |
| --- |
| class Dog:        # class attribute      attr1 = "mammal"        # Instance attribute      def \_\_init\_\_(self, name):          self.name = name        def speak(self):          print("My name is {}".format(self.name))    # Driver code  # Object instantiation  Rodger = Dog("Rodger")  Tommy = Dog("Tommy")    # Accessing class methods  Rodger.speak()  Tommy.speak() |

**Output**

My name is Rodger

My name is Tommy

## Inheritance

Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class. The benefits of inheritance are:

* It represents real-world relationships well.
* It provides the reusability of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
* It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

#### ****Types of Inheritance –****

**Single Inheritance**:  
Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.

**Multilevel Inheritance:**  
Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.

**Hierarchical Inheritance:**  
Hierarchical level inheritance enables more than one derived class to inherit properties from a parent class.

**Multiple Inheritance:**  
Multiple level inheritance enables one derived class to inherit properties from more than one base class.

### Example: Inheritance in Python

|  |
| --- |
| # Python code to demonstrate how parent constructors  # are called.    # parent class  class Person(object):        # \_\_init\_\_ is known as the constructor      def \_\_init\_\_(self, name, idnumber):          self.name = name          self.idnumber = idnumber        def display(self):          print(self.name)          print(self.idnumber)        def details(self):          print("My name is {}".format(self.name))          print("IdNumber: {}".format(self.idnumber))    # child class  class Employee(Person):      def \_\_init\_\_(self, name, idnumber, salary, post):          self.salary = salary          self.post = post            # invoking the \_\_init\_\_ of the parent class          Person.\_\_init\_\_(self, name, idnumber)        def details(self):          print("My name is {}".format(self.name))          print("IdNumber: {}".format(self.idnumber))          print("Post: {}".format(self.post))      # creation of an object variable or an instance  a = Employee('Rahul', 886012, 200000, "Intern")    # calling a function of the class Person using  # its instance  a.display()  a.details() |

**Output**

Rahul

886012

My name is Rahul

IdNumber: 886012

Post: Intern

In the above article, we have created two classes i.e. Person (parent class) and Employee (Child Class). The Employee class inherits from the Person class. We can use the methods of the person class through employee class as seen in the display function in the above code. A child class can also modify the behavior of the parent class as seen through the details() method.

## Polymorphism

Polymorphism simply means having many forms. For example, we need to determine if the given species of birds fly or not, using polymorphism we can do this using a single function.

### Example: Polymorphism in Python

|  |
| --- |
| class Bird:        def intro(self):          print("There are many types of birds.")        def flight(self):          print("Most of the birds can fly but some cannot.")    class sparrow(Bird):        def flight(self):          print("Sparrows can fly.")    class ostrich(Bird):        def flight(self):          print("Ostriches cannot fly.")    obj\_bird = Bird()  obj\_spr = sparrow()  obj\_ost = ostrich()    obj\_bird.intro()  obj\_bird.flight()    obj\_spr.intro()  obj\_spr.flight()    obj\_ost.intro()  obj\_ost.flight() |

**Output**

There are many types of birds.

Most of the birds can fly but some cannot.

There are many types of birds.

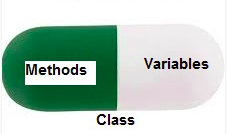
Sparrows can fly.

There are many types of birds.

## Encapsulation

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object’s variable can only be changed by an object’s method. Those types of variables are known as private variables.

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.



### Example: Encapsulation in Python

|  |
| --- |
| # Python program to  # demonstrate private members    # Creating a Base class  class Base:      def \_\_init\_\_(self):          self.a = "GeeksforGeeks"          self.\_\_c = "GeeksforGeeks"    # Creating a derived class  class Derived(Base):      def \_\_init\_\_(self):            # Calling constructor of          # Base class          Base.\_\_init\_\_(self)          print("Calling private member of base class: ")          print(self.\_\_c)      # Driver code  obj1 = Base()  print(obj1.a)    # Uncommenting print(obj1.c) will  # raise an AttributeError    # Uncommenting obj2 = Derived() will  # also raise an AtrributeError as  # private member of base class  # is called inside derived class |

**Output**

GeeksforGeeks

In the above example, we have created the c variable as the private attribute. We cannot even access this attribute directly and can’t even change its value.

**Note:**for more information, refer to our [Encapsulation in Python](https://www.geeksforgeeks.org/encapsulation-in-python/) Tutorial.

## ****Data Abstraction****

It hides the unnecessary code details from the user. Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction came.

Data Abstraction in Python can be achieved through creating abstract classes.

**History: -** The basis for OOP started in the early 1960s. A breakthrough involving instances and objects was achieved at MIT with the PDP-1, and the first programming language to use objects was Simula 67. It was designed for the purpose of creating simulations, and was developed by Kristen Nygaard and Ole-Johan Dahl in Norway.

They were working on simulations that deal with exploding ships, and realized they could group the ships into different categories. Each ship type would have its own class, and the class would generate its unique behavior and data. Simula was not only responsible for introducing the concept of a class, but it also introduced the instance of a class.  
  
The term "object oriented programming " was first used by Xerox PARC in their Smalltalk programming language. The term was used to refer to the process of using objects as the foundation for computation. The Smalltalk team was inspired by the Simula 67 project, but they designed Smalltalk so that it would be dynamic. The objects could be changed, created, or deleted, and this was different from the static systems that were commonly used. Smalltalk was also the first programming language to introduce the inheritance concept. It is this feature that allowed Smalltalk to surpass both Simula 67 and the analog programming systems. While these systems were advanced for their time, they did not use the inheritance concept.  
  
Simula 67 was a groundbreaking system that has inspired a large number of other programming languages, and some of these include Pascal and Lisp. By the 1980s, object oriented programming had become prominent, and the primary factor in this is C++. Object oriented programming was also important for the development of Graphical user interfaces. The Cocoa structure that exists within Mac OS X is a good example of a dynamic GUI that works with an object oriented programming language. This paradigm of programming has also played an important role in the development of event-driven programming.  
  
Niklaus Wirth and his associates were looking at areas such as modular programming and data abstraction, and they developed two systems which incorporated these elements. These two systems are Oberon and Modula-2. Oberon used a unique approach to classes and object orientation that is much different than C++ or Smalltalk. Since the introduction of OOP, a large number of modern programming languages are now using the concept. Some of these are Fortran, BASIC, and Pascal. There have been some compatibility issues, because many programs were not designed with a OOPs approach in mind. Object oriented programming languages that were "pure" did not have many of the functions that programmers needed.  
  
To solve these problems, a number of researchers have been attempting to design new programming languages that used object oriented concepts but still retained many of the functions that programmers needed. One example of a programming language that has achieved this to some degree is Eiffel. Another programming language that has attempted to solve this problem is Java. Java has become popular because it uses a virtual machine, and it is very similar to C++ and C. The virtual machine is important because it allows code to be run on multiple platforms without having to be changed. Another system that is similar is Microsoft’s .NET. Many developers now understand the importance of OOP, and are actively using it within their own programs. Many researchers have continued to make advancements by using the object oriented approach.  
  
There are a number of other languages that have successfully combined the object oriented approach with procedures that are useful to programmers. Python is one example, and Ruby uses a similar approach as well.

The use of an object oriented approach has led to advancements in modeling languages, design patterns, and a number of other areas. It is likely that OOP is a programming paradigm that will continue to evolve as we move forward into the future. It is a powerful language which has continued to improve over the years.

**Chapter 9 Introduction to Data Analytics and Visualization in python**

**9.1 introduction to Numpy,Pandas and Matplotlib**

## Data Analysis

Data Analysis is a process of inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making.

Stpes for Data Analysis, Data Manipulation and Data Visualization:

1. Tranform Raw Data in a Desired Format
2. Clean the Transformed Data (Step 1 and 2 also called as a Pre-processing of Data)
3. Prepare a Model
4. Analyse Trends and Make Decisions

## NumPy

NumPy is a package for scientific computing.

1. Multi dimensional array
2. Methods for processing arrays
3. Element by element operations
4. Mathematical operations like logical, Fourier transform, shape manipulation, linear algebra and random number generation

In [1]:

import numpy as np

### Ndarray - NumPy Array

The ndarray is a multi-dimensional array object consisting of two parts -- the actual data, some metadata which describes the stored data. They are indexed just like sequence are in Python, starting from 0

1. Each element in ndarray is an object of data-type object called dtype
2. An item extracted from ndarray, is represented by a Python object of an array scalar type

### Single Dimensional Array

### Creating a Numpy Array

In [2]:

*# Creating a single-dimensional array*

a = np.array([1,2,3]) *# Calling the array function*

print(a)

[1 2 3]

In [3]:

*# Creating a multi-dimensional array*

*# Each set of elements within a square bracket indicates a row*

*# Array of two rows and two columns*

b = np.array([[1,2], [3,4]])

print(b)

[[1 2]

[3 4]]

In [4]:

*# Creating an ndarray by wrapping a list*

list1 = [1,2,3,4,5] *# Creating a list*

arr = np.array(list1) *# Wrapping the list*

print(arr)

[1 2 3 4 5]

In [5]:

*# Creating an array of numbers of a specified range*

arr1 = np.arange(10, 100) *# Array of numbers from 10 up to and excluding 100*

print(arr1)

[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33

34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57

58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81

82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99]

In [6]:

linkcode

*# Creating a 5x5 array of zeroes*

arr2 = np.zeros((5,5))

print(arr2)

[[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]]

In [7]:

*# Creating a linearly spaced vector, with spacing*

vector = np.linspace(0, 20, 5) *# Start, stop, step*

print(vector)

[ 0. 5. 10. 15. 20.]

In [8]:

*# Creating Arrays from Existing Data*

x = [1,2,3]

*# Used for converting Python sequences into ndarrays*

c = np.asarray(x) *#np.asarray(a, dtype = None, order = None)*

print(c)

[1 2 3]

In [9]:

*# Converting a linear array of 8 elements into a 2x2x2 3D array*

arr3 = np.zeros(8) *# Flat array of eight zeroes*

arr3d = arr3.reshape((2,2,2)) *# Restructured array*

print(arr3d)

[[[0. 0.]

[0. 0.]]

[[0. 0.]

[0. 0.]]]

In [10]:

*# Flatten rgw 3d array to get back the linear array*

arr4 = arr3d.ravel()

print(arr4)

[0. 0. 0. 0. 0. 0. 0. 0.]

### Indexing of NumPy Arrays

In [11]:

*# NumPy array indexing is identical to Python's indexing scheme*

arr5 = np.arange(2, 20)

element = arr5[6]

print(element)

8

In [12]:

*# Python's concept of lists slicing is extended to NumPy.*

*# The slice object is constructed by providing start, stop, and step parameters to slice()*

arr6 = np.arange(20)

arr\_slice = slice(1, 10, 2) *# Start, stop & step*

element2 = arr6[6]

print(arr6[arr\_slice])

[1 3 5 7 9]

In [13]:

*# Slicing items beginning with a specified index*

arr7 = np.arange(20)

print(arr7[2:])

[ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

In [14]:

*# Slicing items until a specified index*

print(arr7[:15])

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]

In [15]:

*# Extracting specific rows and columns using Slicing*

d = np.array([[1,2,3], [3,4,5], [4,5,6]])

print(d[0:2, 0:2]) *# Slice the first two rows and the first two columns*

[[1 2]

[3 4]]

### NumPy Array Attributes

In [16]:

print(d.shape) *# Returns a tuple consisting of array dimensions*

print(d.ndim) *# Attribute returns the number of array dimensions*

print(a.itemsize) *# Returns the length of each element of array in bytes*

(3, 3)

2

8

In [17]:

y = np.empty([3,2], dtype = int) *# Creates an uninitialized array of specified shape and dtype*

print(y)

[[140468392404648 140468392404648]

[ 0 0]

[ 0 0]]

In [18]:

*# Returns a new array of specified size, filled with zeros*

z = np.zeros(5) *# np.zeros(shape, dtype = float)*

print(z)

[0. 0. 0. 0. 0.]

### Reading & Writing from Files

In [19]:

*# NumPy provides the option of importing data from files directly into ndarray using the loadtxt function*

*# The savetxt function can be used to write data from an array into a text file*

*#import os*

*#print(os.listdir('../input'))*

arr\_txt = np.loadtxt('../input/data\_file.txt')

np.savetxt('newfilex.txt', arr\_txt)

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-19-a97aa75f0572> in <module>()

**3** #import os

**4** #print(os.listdir('../input'))

----> 5 arr\_txt = np.loadtxt('../input/data\_file.txt')

**6** np.savetxt('newfilex.txt', arr\_txt)

/opt/conda/lib/python3.6/site-packages/numpy/lib/npyio.py in loadtxt(fname, dtype, comments, delimiter, converters, skiprows, usecols, unpack, ndmin, encoding)

**1099** # converting the data

**1100** X = None

-> 1101 for x in read\_data(\_loadtxt\_chunksize):

**1102** if X is None:

**1103** X = np.array(x, dtype)

/opt/conda/lib/python3.6/site-packages/numpy/lib/npyio.py in read\_data(chunk\_size)

**1026**

**1027** # Convert each value according to its column and store

-> 1028 items = [conv(val) for (conv, val) in zip(converters, vals)]

**1029**

**1030** # Then pack it according to the dtype's nesting

/opt/conda/lib/python3.6/site-packages/numpy/lib/npyio.py in <listcomp>(.0)

**1026**

**1027** # Convert each value according to its column and store

-> 1028 items = [conv(val) for (conv, val) in zip(converters, vals)]

**1029**

**1030** # Then pack it according to the dtype's nesting

/opt/conda/lib/python3.6/site-packages/numpy/lib/npyio.py in floatconv(x)

**744** if '0x' in x:

**745** return float.fromhex(x)

--> 746 return float(x)

**747**

**748** typ = dtype.type

ValueError: could not convert string to float: 'account\_number,name,item\_code,category,quantity,unit'

In [20]:

*# NumPy arrays can be dumped into CSV files using the savetxt function and the comma delimiter*

*# The genfromtxt function can be used to read data from a CSV file into a NumPy array*

arr\_csv = np.genfromtxt('../input/Hurricanes.csv', delimiter = ',')

np.savetxt('newfilex.csv', arr\_csv, delimiter = ',')

## Pandas

Pandas is an open-source Python library providing efficient, easy-to-use data structure and data analysis tools. The name Pandas is derived from "Panel Data" - an Econometrics from Multidimensional Data. Pandas is well suited for many different kinds of data:

1. Tabular data with heterogeneously-type columns.
2. Ordered and unordered time series data.
3. Arbitary matrix data with row and column labels.
4. Any other form observational/statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure.

Pandas provides three data structure - all of which are build on top of the NumPy array - all the data structures are value-mutable

1. Series (1D) - labeled, homogenous array of immutable size
2. DataFrames (2D) - labeled, heterogeneously typed, size-mutable tabular data structures
3. Panels (3D) - Labeled, size-mutable array

In [21]:

import pandas as pd

### Series

1. A Series is a single-dimensional array structures that stores homogenous data i.e., data of a single type.
2. All the elements of a Series are value-mutable and size-immutable
3. Data can be of multiple data types such as ndarray, lists, constants, series, dict etc.
4. Indexes must be unique, hashable and have the same length as data. Defaults to np.arrange(n) if no index is passed.
5. Data type of each column; if none is mentioned, it will be inferred; automatically
6. Deep copies data, set to false as default

### Creating a Series

In [22]:

*# Creating an empty Series*

series = pd.Series() *# The Series() function creates a new Series*

print(series)

Series([], dtype: float64)

In [23]:

*# Creating a series from an ndarray*

*# Note that indexes are a assigned automatically if not specifies*

arr = np.array([10,20,30,40,50])

series1 = pd.Series(arr)

print(series1)

0 10

1 20

2 30

3 40

4 50

dtype: int64

In [24]:

*# Creating a series from a Python dict*

*# Note that the keys of the dictionary are used to assign indexes during conversion*

data = {'a':10, 'b':20, 'c':30}

series2 = pd.Series(data)

print(series2)

a 10

b 20

c 30

dtype: int64

In [25]:

*# Retrieving a part of the series using slicing*

print(series1[1:4])

1 20

2 30

3 40

dtype: int64

### DataFrames

1. A DataFrame is a 2D data structure in which data is aligned in a tabular fashion consisting of rows & columns
2. A DataFrame can be created using the following constructor - pandas.DataFrame(data, index, dtype, copy)
3. Data can be of multiple data types such as ndarray, list, constants, series, dict etc.
4. Index Row and column labels of the dataframe; defaults to np.arrange(n) if no index is passed
5. Data type of each column
6. Creates a deep copy of the data, set to false as default

### Creating a DataFrame

In [26]:

*# Converting a list into a DataFrame*

list1 = [10, 20, 30, 40]

table = pd.DataFrame(list1)

print(table)

0

0 10

1 20

2 30

3 40

In [27]:

*# Creating a DataFrame from a list of dictionaries*

data = [{'a':1, 'b':2}, {'a':2, 'b':4, 'c':8}]

table1 = pd.DataFrame(data)

print(table1)

*# NaN (not a number) is stored in areas where no data is provided*

a b c

0 1 2 NaN

1 2 4 8.0

In [28]:

*# Creating a DataFrame from a list of dictionaries and accompaying row indices*

table2 = pd.DataFrame(data, index = ['first', 'second'])

*# Dict keys become column lables*

print(table2)

a b c

first 1 2 NaN

second 2 4 8.0

In [29]:

*# Converting a dictionary of series into a DataFrame*

data1 = {'one':pd.Series([1,2,3], index = ['a', 'b', 'c']),

'two':pd.Series([1,2,3,4], index = ['a', 'b', 'c', 'd'])}

table3 = pd.DataFrame(data1)

print(table3)

*# the resultant index is the union of all the series indexes passed*

one two

a 1.0 1

b 2.0 2

c 3.0 3

d NaN 4

### DataFrame - Addition & Deletion of Columns

In [30]:

*# A new column can be added to a DataFrame when the data is passed as a Series*

table3['three'] = pd.Series([10,20,30], index = ['a', 'b', 'c'])

print(table3)

one two three

a 1.0 1 10.0

b 2.0 2 20.0

c 3.0 3 30.0

d NaN 4 NaN

In [31]:

*# DataFrame columns can be deleted using the del() function*

del table3['one']

print(table3)

two three

a 1 10.0

b 2 20.0

c 3 30.0

d 4 NaN

In [32]:

*# DataFrame columns can be deleted using the pop() function*

table3.pop('two')

print(table3)

three

a 10.0

b 20.0

c 30.0

d NaN

### DataFrame - Addition & Deletion of Rows

In [33]:

*# DataFrame rows can be selected by passing the row lable to the loc() function*

print(table3.loc['c'])

three 30.0

Name: c, dtype: float64

In [34]:

*# Row selection can also be done using the row index*

print(table3.iloc[2])

three 30.0

Name: c, dtype: float64

In [35]:

*# The append() function can be used to add more rows to the DataFrame*

data2 = {'one':pd.Series([1,2,3], index = ['a', 'b', 'c']),

'two':pd.Series([1,2,3,4], index = ['a', 'b', 'c', 'd'])}

table5 = pd.DataFrame(data2)

table5['three'] = pd.Series([10,20,30], index = ['a', 'b', 'c'])

row = pd.DataFrame([[11,13],[17,19]], columns = ['two', 'three'])

table6 = table5.append(row)

print(table6)

one three two

a 1.0 10.0 1

b 2.0 20.0 2

c 3.0 30.0 3

d NaN NaN 4

0 NaN 13.0 11

1 NaN 19.0 17

/opt/conda/lib/python3.6/site-packages/pandas/core/frame.py:6211: FutureWarning: Sorting because non-concatenation axis is not aligned. A future version

of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

sort=sort)

In [36]:

*# The drop() function can be used to drop rows whose labels are provided*

table7 = table6.drop('a')

print(table7)

one three two

b 2.0 20.0 2

c 3.0 30.0 3

d NaN NaN 4

0 NaN 13.0 11

1 NaN 19.0 17

### Importing & Exporting Data

In [37]:

*# Data can be loaded into DataFrames from input data stored in the CSV format using the read\_csv() function*

table\_csv = pd.read\_csv('../input/Cars2015.csv')

In [38]:

*# Data present in DataFrames can be written to a CSV file using the to\_csv() function*

*# If the specified path doesn't exist, a file of the same name is automatically created*

table\_csv.to\_csv('newcars2015.csv')

In [39]:

*# Data can be loaded into DataFrames from input data stored in the Excelsheet format using read\_excel()*

sheet = pd.read\_excel('cars2015.xlsx')

---------------------------------------------------------------------------

FileNotFoundError Traceback (most recent call last)

<ipython-input-39-a9a8c848b22a> in <module>()

**1** # Data can be loaded into DataFrames from input data stored in the Excelsheet format using read\_excel()

----> 2 sheet = pd.read\_excel('cars2015.xlsx')

/opt/conda/lib/python3.6/site-packages/pandas/util/\_decorators.py in wrapper(\*args, \*\*kwargs)

**176** else:

**177** kwargs[new\_arg\_name] = new\_arg\_value

--> 178 return func(\*args, \*\*kwargs)

**179** return wrapper

**180** return \_deprecate\_kwarg

/opt/conda/lib/python3.6/site-packages/pandas/util/\_decorators.py in wrapper(\*args, \*\*kwargs)

**176** else:

**177** kwargs[new\_arg\_name] = new\_arg\_value

--> 178 return func(\*args, \*\*kwargs)

**179** return wrapper

**180** return \_deprecate\_kwarg

/opt/conda/lib/python3.6/site-packages/pandas/io/excel.py in read\_excel(io, sheet\_name, header, names, index\_col, usecols, squeeze, dtype, engine, converters, true\_values, false\_values, skiprows, nrows, na\_values, parse\_dates, date\_parser, thousands, comment, skipfooter, convert\_float, \*\*kwds)

**305**

**306** if not isinstance(io, ExcelFile):

--> 307 io = ExcelFile(io, engine=engine)

**308**

**309** return io.parse(

/opt/conda/lib/python3.6/site-packages/pandas/io/excel.py in \_\_init\_\_(self, io, \*\*kwds)

**392** self.book = xlrd.open\_workbook(file\_contents=data)

**393** elif isinstance(self.\_io, compat.string\_types):

--> 394 self.book = xlrd.open\_workbook(self.\_io)

**395** else:

**396** raise ValueError('Must explicitly set engine if not passing in'

/opt/conda/lib/python3.6/site-packages/xlrd/\_\_init\_\_.py in open\_workbook(filename, logfile, verbosity, use\_mmap, file\_contents, encoding\_override, formatting\_info, on\_demand, ragged\_rows)

**114** peek = file\_contents[:peeksz]

**115** else:

--> 116 with open(filename, "rb") as f:

**117** peek = f.read(peeksz)

**118** if peek == b"PK\x03\x04": # a ZIP file

FileNotFoundError: [Errno 2] No such file or directory: 'cars2015.xlsx'

In [40]:

*# Data present in DataFrames can be written to a spreadsheet file using to\_excel()*

*#If the specified path doesn't exist, a file of the same name is automatically created*

sheet.to\_excel('newcars2015.xlsx')

---------------------------------------------------------------------------

NameError Traceback (most recent call last)

<ipython-input-40-f689e9aa0038> in <module>()

**1** # Data present in DataFrames can be written to a spreadsheet file using to\_excel()

**2** #If the specified path doesn't exist, a file of the same name is automatically created

----> 3 sheet.to\_excel('newcars2015.xlsx')

NameError: name 'sheet' is not defined

## Matplotlib

1. Matplotlib is a Python library that is specially designed for the development of graphs, charts etc., in order to provide interactive data visualisation
2. Matplotlib is inspired from the MATLAB software and reproduces many of it's features

In [41]:

*# Import Matplotlib submodule for plotting*

import matplotlib.pyplot as plt

### Plotting in Matplotlib

In [42]:

plt.plot([1,2,3,4]) *# List of vertical co-ordinates of the points plotted*

plt.show() *# Displays plot*

*# Implicit X-axis values from 0 to (N-1) where N is the length of the list*

In [43]:

*# We can specify the values for both axes*

x = range(5) *# Sequence of values for the x-axis*

*# X-axis values specified - [0,1,2,3,4]*

plt.plot(x, [x1\*\*2 for x1 **in** x]) *# vertical co-ordinates of the points plotted: y = x^2*

plt.show()

In [44]:

*# We can use NumPy to specify the values for both axes with greater precision*

x = np.arange(0, 5, 0.01)

plt.plot(x, [x1\*\*2 for x1 **in** x]) *# vertical co-ordinates of the points plotted: y = x^2*

plt.show()

### Multiline Plots

In [45]:

*# Multiple functions can be drawn on the same plot*

x = range(5)

plt.plot(x, [x1 for x1 **in** x])

plt.plot(x, [x1\*x1 for x1 **in** x])

plt.plot(x, [x1\*x1\*x1 for x1 **in** x])

plt.show()

In [46]:

*# Different colours are used for different lines*

x = range(5)

plt.plot(x, [x1 for x1 **in** x],

x, [x1\*x1 for x1 **in** x],

x, [x1\*x1\*x1 for x1 **in** x])

plt.show()

### Grids

In [47]:

*# The grid() function adds a grid to the plot*

*# grid() takes a single Boolean parameter*

*# grid appears in the background of the plot*

x = range(5)

plt.plot(x, [x1 for x1 **in** x],

x, [x1\*2 for x1 **in** x],

x, [x1\*4 for x1 **in** x])

plt.grid(True)

plt.show()

### Limiting the Axes

In [48]:

*# The scale of the plot can be set using axis()*

x = range(5)

plt.plot(x, [x1 for x1 **in** x],

x, [x1\*2 for x1 **in** x],

x, [x1\*4 for x1 **in** x])

plt.grid(True)

plt.axis([-1, 5, -1, 10]) *# Sets new axes limits*

plt.show()

In [49]:

*# The scale of the plot can also be set using xlim() and ylim()*

x = range(5)

plt.plot(x, [x1 for x1 **in** x],

x, [x1\*2 for x1 **in** x],

x, [x1\*4 for x1 **in** x])

plt.grid(True)

plt.xlim(-1, 5)

plt.ylim(-1, 10)

plt.show()

### Adding Labels

In [50]:

*# Labels can be added to the axes of the plot*

x = range(5)

plt.plot(x, [x1 for x1 **in** x],

x, [x1\*2 for x1 **in** x],

x, [x1\*4 for x1 **in** x])

plt.grid(True)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.show()

### Adding the Title

In [51]:

*# The title defines the data plotted on the graph*

x = range(5)

plt.plot(x, [x1 for x1 **in** x],

x, [x1\*2 for x1 **in** x],

x, [x1\*4 for x1 **in** x])

plt.grid(True)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title("Polynomial Graph") *# Pass the title as a parameter to title()*

plt.show()

### Adding a Legend

In [52]:

*# Legends explain the meaning of each line in the graph*

x = np.arange(5)

plt.plot(x, x, label = 'linear')

plt.plot(x, x\*x, label = 'square')

plt.plot(x, x\*x\*x, label = 'cube')

plt.grid(True)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title("Polynomial Graph")

plt.legend()

plt.show()

### Adding a Markers

In [53]:

x = [1, 2, 3, 4, 5, 6]

y = [11, 22, 33, 44, 55, 66]

plt.plot(x, y, 'bo')

for i **in** range(len(x)):

x\_cord = x[i]

y\_cord = y[i]

plt.text(x\_cord, y\_cord, (x\_cord, y\_cord), fontsize = 10)

plt.show()

### Saving Plots

In [54]:

*# Plots can be saved using savefig()*

x = np.arange(5)

plt.plot(x, x, label = 'linear')

plt.plot(x, x\*x, label = 'square')

plt.plot(x, x\*x\*x, label = 'cube')

plt.grid(True)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title("Polynomial Graph")

plt.legend()

plt.savefig('plot.png') *# Saves an image names 'plot.png' in the current directory*

plt.show()

### Plot Types

Matplotlib provides many types of plot formats for visualising information

1. Scatter Plot
2. Histogram
3. Bar Graph
4. Pie Chart

### Histogram

In [55]:

*# Histograms display the distribution of a variable over a range of frequencies or values*

y = np.random.randn(100, 100) *# 100x100 array of a Gaussian distribution*

plt.hist(y) *# Function to plot the histogram takes the dataset as the parameter*

plt.show()

In [56]:

*# Histogram groups values into non-overlapping categories called bins*

*# Default bin value of the histogram plot is 10*

y = np.random.randn(1000)

plt.hist(y, 100)

plt.show()

### Bar Chart

In [57]:

*# Bar charts are used to visually compare two or more values using rectangular bars*

*# Default width of each bar is 0.8 units*

*# [1,2,3] Mid-point of the lower face of every bar*

*# [1,4,9] Heights of the successive bars in the plot*

plt.bar([1,2,3], [1,4,9])

plt.show()

In [58]:

dictionary = {'A':25, 'B':70, 'C':55, 'D':90}

for i, key **in** enumerate(dictionary):

plt.bar(i, dictionary[key]) *# Each key-value pair is plotted individually as dictionaries are not iterable*

plt.show()

In [59]:

dictionary = {'A':25, 'B':70, 'C':55, 'D':90}

for i, key **in** enumerate(dictionary):

plt.bar(i, dictionary[key])

plt.xticks(np.arange(len(dictionary)), dictionary.keys()) *# Adds the keys as labels on the x-axis*

plt.show()

### Pie Chart

In [60]:

plt.figure(figsize = (3,3)) *# Size of the plot in inches*

x = [40, 20, 5] *# Proportions of the sectors*

labels = ['Bikes', 'Cars', 'Buses']

plt.pie(x, labels = labels)

plt.show()

### Scatter Plot

In [61]:

*# Scatter plots display values for two sets of data, visualised as a collection of points*

*# Two Gaussion distribution plotted*

x = np.random.rand(1000)

y = np.random.rand(1000)

plt.scatter(x, y)

plt.show()

### Styling

In [62]:

*# Matplotlib allows to choose custom colours for plots*

y = np.arange(1, 3)

plt.plot(y, 'y') *# Specifying line colours*

plt.plot(y+5, 'm')

plt.plot(y+10, 'c')

plt.show()

Color code:

1. b = Blue
2. c = Cyan
3. g = Green
4. k = Black
5. m = Magenta
6. r = Red
7. w = White
8. y = Yellow

In [63]:

*# Matplotlib allows different line styles for plots*

y = np.arange(1, 100)

plt.plot(y, '--', y\*5, '-.', y\*10, ':')

plt.show()

*# - Solid line*

*# -- Dashed line*

*# -. Dash-Dot line*

*# : Dotted Line*

In [64]:

*# Matplotlib provides customization options for markers*

y = np.arange(1, 3, 0.2)

plt.plot(y, '\*',

y+0.5, 'o',

y+1, 'D',

y+2, '^',

y+3, 's') *# Specifying line styling*

plt.show()

**9.2 Data Wrangling**

## Data Wrangling is the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging. **Importance of Data Wrangling**

Data wrangling is a very important step. The below example will explain its importance as:

Books selling Website want to show top-selling books of different domains, according to user preference. For example, a new user search for motivational books, then they want to show those motivational books which sell the most or having a high rating, etc.

But on their website, there are plenty of raw data from different users. Here the concept of Data Munging or Data Wrangling is used. As we know Data is not Wrangled by System. This process is done by Data Scientists. So, the data Scientist will wrangle data in such a way that they will sort that motivational books that are sold more or have high ratings or user buy this book with these package of Books, etc. On the basis of that, the new user will make choice. This will explain the importance of Data wrangling.

Data Wrangling is the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging.

## ****Importance Of Data Wrangling****

Data Wrangling is a very important step. The below example will explain its importance as :

Books selling Website want to show top-selling books of different domains, according to user preference. For example, a new user search for motivational books, then they want to show those motivational books which sell the most or having a high rating, etc.

But on their website, there are plenty of raw data from different users. Here the concept of Data Munging or Data Wrangling is used. As we know Data is not Wrangled by System. This process is done by Data Scientists. So, the data Scientist will wrangle data in such a way that they will sort that motivational books that are sold more or have high ratings or user buy this book with these package of Books, etc. On the basis of that, the new user will make choice. This will explain the importance of Data wrangling.

Data Wrangling is a crucial topic for Data Science and Data Analysis. Pandas Framework of Python is used for Data Wrangling. Pandas is an open-source library specifically developed for Data Analysis and Data Science. The process like data sorting or filtration, Data grouping, etc.

Data wrangling in python deals with the below functionalities:

1. **Data exploration:**In this process, the data is studied, analyzed and understood by visualizing representations of data.
2. **Dealing with missing values:**Most of the datasets having a vast amount of data contain missing values of *NaN, they are needed to be taken*careof by replacing them with mean, mode, the most frequent value of the column or simply by dropping the row having a *NaN*value.
3. **Reshaping data:**In this process, data is manipulated according to the requirements, where new data can be added or pre-existing data can be modified.
4. **Filtering data:**Some times datasets are comprised of unwanted rows or columns which are required to be removed or filtered
5. **Other:** After dealing with the raw dataset with the above functionalities we get an efficient dataset as per our requirements and then it can be used for a required purpose like data analyzing, machine learning, data visualization, model training etc.

#### Below is an example which implements the above functionalities on a raw dataset:

* **Data exploration**, here we assign the data, and then we visualize the data in a tabular format

|  |
| --- |
| # Import pandas package  import pandas as pd    # Assign data  data = {'Name': ['Jai', 'Princi', 'Gaurav',                   'Anuj', 'Ravi', 'Natasha', 'Riya'],          'Age': [17, 17, 18, 17, 18, 17, 17],          'Gender': ['M', 'F', 'M', 'M', 'M', 'F', 'F'],          'Marks': [90, 76, 'NaN', 74, 65, 'NaN', 71]}    # Convert into DataFrame  df = pd.DataFrame(data)    # Display data  df |

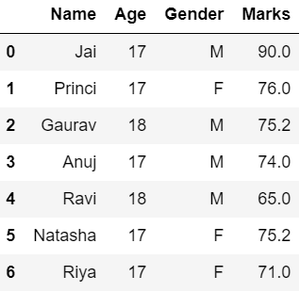
**Output:**



* **Dealing with missing values**, as we can see from the previous output, there are *NaN*values present in the *MARKS*column which are going to be taken care of by replacing them with the column mean.

|  |
| --- |
| #Compute average  c = avg = 0  for ele in df['Marks']:      if str(ele).isnumeric():          c += 1          avg += ele  avg /= c    # Replace missing values  df = df.replace(to\_replace="NaN",                  value=avg)    # Display data  df |

**Output:**



* **Reshaping data**, in the *GENDER*column, we can reshape the data by categorizing them into different numbers.

|  |
| --- |
| # Categorize gender  df['Gender'] = df['Gender'].map({'M': 0,                                   'F': 1, }).astype(float)    # Display data  df |

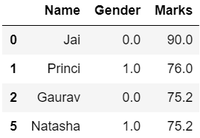
**Output:**



* **Filtering data**, suppose there is a requirement for the details regarding name, gender, marks of the top-scoring students. Here we need to remove some unwanted data.

|  |
| --- |
| # Filter top scoring students  df = df[df['Marks'] >= 75]    # Remove age row  df = df.drop(['Age'], axis=1)    # Display data  df |

**Output:**



Hence, we have finally obtained an efficient dataset which can be further used for various purposes.

Now that we know the basics of data wrangling. Below we will discuss various operations using which we can perform data wrangling:

## ****Wrangling Data Using Merge Operation****

Merge operation is used to merge raw data and into the desired format.

**Syntax:**

pd.merge( data\_frame1,data\_frame2, on="field ")

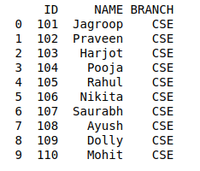
Here the field is the name of the column which is similar on both data-frame.

For example: Suppose that a Teacher has two types of Data, first type of Data consist of Details of Students and Second type of Data Consist of Pending Fees Status which is taken from Account Office. So The Teacher will use merge operation here in order to merge the data and provide it meaning. So that teacher will analyze it easily and it also reduces time and effort of Teacher from Manual Merging.

**FIRST TYPE OF DATA:**

|  |
| --- |
| # import module  import pandas as pd    # creating DataFrame for Student Details  details = pd.DataFrame({      'ID': [101, 102, 103, 104, 105, 106,             107, 108, 109, 110],      'NAME': ['Jagroop', 'Praveen', 'Harjot',               'Pooja', 'Rahul', 'Nikita',               'Saurabh', 'Ayush', 'Dolly', "Mohit"],      'BRANCH': ['CSE', 'CSE', 'CSE', 'CSE', 'CSE',                 'CSE', 'CSE', 'CSE', 'CSE', 'CSE']})    # printing details  print(details) |

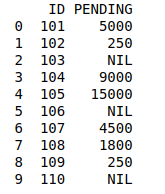
**Output:**



**SECOND TYPE OF DATA**

|  |
| --- |
| # Import module  import pandas as pd    # Creating Dataframe for Fees\_Status  fees\_status = pd.DataFrame(      {'ID': [101, 102, 103, 104, 105,              106, 107, 108, 109, 110],       'PENDING': ['5000', '250', 'NIL',                   '9000', '15000', 'NIL',                   '4500', '1800', '250', 'NIL']})    # Printing fees\_status  print(fees\_status) |

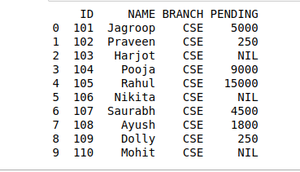
**Output:**



**WRANGLING DATA USING MERGE OPERATION:**

|  |
| --- |
| # Import module  import pandas as pd    # Creating Dataframe  details = pd.DataFrame({      'ID': [101, 102, 103, 104, 105,             106, 107, 108, 109, 110],      'NAME': ['Jagroop', 'Praveen', 'Harjot',               'Pooja', 'Rahul', 'Nikita',               'Saurabh', 'Ayush', 'Dolly', "Mohit"],      'BRANCH': ['CSE', 'CSE', 'CSE', 'CSE', 'CSE',                 'CSE', 'CSE', 'CSE', 'CSE', 'CSE']})    # Creating Dataframe  fees\_status = pd.DataFrame(      {'ID': [101, 102, 103, 104, 105,              106, 107, 108, 109, 110],       'PENDING': ['5000', '250', 'NIL',                   '9000', '15000', 'NIL',                   '4500', '1800', '250', 'NIL']})    # Merging Dataframe  print(pd.merge(details, fees\_status, on='ID')) |

**Output:**



## ****Wrangling Data using Grouping Method****

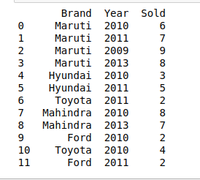
The grouping method in Data analysis is used to provide results in terms of various groups taken out from Large Data. This method of pandas is used to group the outset of data from the large data set.

Example: There is a Car Selling company and this company have different Brands of various Car Manufacturing Company like Maruti, Toyota, Mahindra, Ford, etc. and have data where different cars are sold in different years. So the Company wants to wrangle only that data where cars are sold during the year 2010. For this problem, we use another Wrangling technique that is*groupby()*method.

**CARS SELLING DATA:**

|  |
| --- |
| # Import module  import pandas as pd    # Creating Data  car\_selling\_data = {'Brand': ['Maruti', 'Maruti', 'Maruti',                                'Maruti', 'Hyundai', 'Hyundai',                                'Toyota', 'Mahindra', 'Mahindra',                                'Ford', 'Toyota', 'Ford'],                      'Year':  [2010, 2011, 2009, 2013,                                2010, 2011, 2011, 2010,                                2013, 2010, 2010, 2011],                      'Sold': [6, 7, 9, 8, 3, 5,                               2, 8, 7, 2, 4, 2]}    # Creating Dataframe of car\_selling\_data  df = pd.DataFrame(car\_selling\_data)    # printing Dataframe  print(df) |

**Output:**



**DATA OF THE** **YEAR 2010:**

|  |
| --- |
| # Import module  import pandas as pd    # Creating Data  car\_selling\_data = {'Brand': ['Maruti', 'Maruti', 'Maruti',                                'Maruti', 'Hyundai', 'Hyundai',                                'Toyota', 'Mahindra', 'Mahindra',                                'Ford', 'Toyota', 'Ford'],                      'Year':  [2010, 2011, 2009, 2013,                                2010, 2011, 2011, 2010,                                2013, 2010, 2010, 2011],                      'Sold': [6, 7, 9, 8, 3, 5,                               2, 8, 7, 2, 4, 2]}    # Creating Dataframe for Provided Data  df = pd.DataFrame(car\_selling\_data)    # Group the data when year = 2010  grouped = df.groupby('Year')  print(grouped.get\_group(2010)) |

**Output:**



## ****Wrangling data by removing Duplication****

Pandas *duplicates()* method helps us to remove duplicate values from Large Data. An important part of Data Wrangling is removing Duplicate values from the large data set.

**Syntax:**

DataFrame.duplicated(subset=None, keep='first')

Here subset is the column value where we want to remove Duplicate value.

In*keep*, we have 3 options :

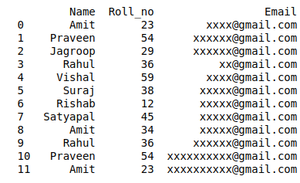
* if  *keep =’first’* then the first value is marked as original rest all values if occur will be removed as it is considered as duplicate.
* if *keep=’last’* then the last value is marked as original rest all above same values will be removed as it is considered as duplicate values.
* if *keep =’false’* the all the values which occur more than once will be removed as all considered as a duplicate value.

For example, A University will organize the event. In order to participate Students have to fill their details in the online form so that they will contact them. It may be possible that a student will fill the form multiple time. It may cause difficulty for the event organizer if a single student will fill multiple entries. The Data that the organizers will get can be Easily Wrangles by removing duplicate values.

**DETAILS STUDENTS DATA WHO WANT TO PARTICIPATE IN THE EVENT:**

|  |
| --- |
| # Import module  import pandas as pd    # Initializing Data  student\_data = {'Name': ['Amit', 'Praveen', 'Jagroop',                           'Rahul', 'Vishal', 'Suraj',                           'Rishab', 'Satyapal', 'Amit',                           'Rahul', 'Praveen', 'Amit'],                    'Roll\_no': [23, 54, 29, 36, 59, 38,                              12, 45, 34, 36, 54, 23],                    'Email': ['xxxx@gmail.com', 'xxxxxx@gmail.com',                            'xxxxxx@gmail.com', 'xx@gmail.com',                            'xxxx@gmail.com', 'xxxxx@gmail.com',                            'xxxxx@gmail.com', 'xxxxx@gmail.com',                            'xxxxx@gmail.com', 'xxxxxx@gmail.com',                            'xxxxxxxxxx@gmail.com', 'xxxxxxxxxx@gmail.com']}    # Creating Dataframe of Data  df = pd.DataFrame(student\_data)    # Printing Dataframe  print(df) |

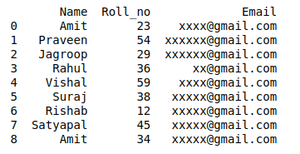
**Output:**



**DATA WRANGLED BY REMOVING DUPLICATE ENTRIES:**

|  |
| --- |
| # import module  import pandas as pd    # initializing Data  student\_data = {'Name': ['Amit', 'Praveen', 'Jagroop',                           'Rahul', 'Vishal', 'Suraj',                           'Rishab', 'Satyapal', 'Amit',                           'Rahul', 'Praveen', 'Amit'],                    'Roll\_no': [23, 54, 29, 36, 59, 38,                              12, 45, 34, 36, 54, 23],                  'Email': ['xxxx@gmail.com', 'xxxxxx@gmail.com',                            'xxxxxx@gmail.com', 'xx@gmail.com',                            'xxxx@gmail.com', 'xxxxx@gmail.com',                            'xxxxx@gmail.com', 'xxxxx@gmail.com',                            'xxxxx@gmail.com', 'xxxxxx@gmail.com',                            'xxxxxxxxxx@gmail.com', 'xxxxxxxxxx@gmail.com']}    # creating dataframe  df = pd.DataFrame(student\_data)    # Here df.duplicated() list duplicate  Entries in ROllno.  # So that ~(NOT) is placed in order to get non duplicate values.  non\_duplicate = df[~df.duplicated('Roll\_no')]    # printing non-duplicate values  print(non\_duplicate) |

**Output:**



9.3 Data Loading

Data loading is the process of copying and loading data or data sets from a source file, folder or application to a database or similar application. It is usually implemented by copying digital data from a source and pasting or loading the data to a data storage or processing utility.

Data loading is used in database-based extraction and loading techniques. Typically, such data is loaded into the destination application as a different format than the original source location.

For example, when data is copied from a word processing file to a database application, the data format is changed from .doc or .txt to a .CSV or DAT format. Usually, this process is performed through or the last phase of the Extract, Transform and Load (ETL) process. The data is extracted from an external source and transformed into the destination application's supported format, where the data is further loaded.

# Data Loading:

Data Loading is defined as copying data from one electronic file or database into another. Data loading implies converting from one format into another; for example, from one type

## Introduction to Data Loading

Data loading defines the LOAD component of the ETL process. ETL stands for Extraction, Transformation, and Load. Extraction deals with the retrieval and combining of data from multiple sources. Transformation deals with cleaning and formatting of the Extracted Data. Data Loading deals with data getting loaded into a storage system, such as a cloud data warehouse.

ETL aids in the data integration process that standardizes diverse data types to make them available for querying, manipulation, or reporting for many different individuals and teams. Because today’s organizations are increasingly dependent upon their own data to make smarter, faster business decisions, ETL needs to be scalable and streamlined to provide the most benefit.

Data loading is quite simply the process of packing up your data and moving it to a designated data warehouse. It is at the beginning of this transitory phase where you can begin planning a roadmap, outlining where you would like to move forward with your data and how you would like to use it.

Data Loading is the ultimate step in the ETL process. In this step, the extracted data and the transformed data are loaded into the target database. To make the data loading efficient, it is necessary to index the database and disable the constraints before loading the data. All three steps in the ETL process can be run parallel. Data extraction takes time and therefore the second phase of the transformation process is executed simultaneously. This prepares the data for the third stage that is data loading. As soon as some data is ready, data loading is done without waiting for the previous steps to be completed.

## ****Simplify Data Analysis with Hevo’s No-code Data Pipeline****

[Hevo Data,](https://hevodata.com/) a No-code Data Pipeline helps to load data from any data source such as Google Search Console, Databases, SaaS applications, Cloud Storage, SDKs, and Streaming Services and simplifies the ETL process. It supports [100+ data sources](https://hevodata.com/integrations/) (**including 30+ free data sources**) and is a 3-step process by just selecting the data source, providing valid credentials, and choosing the destination. Hevo not only loads the data onto the desired Data Warehouse/destination but also enriches the data and transforms it into an analysis-ready form without having to write a single line of code.

Its completely automated pipeline offers data to be delivered in real-time without any loss from source to destination. Its fault-tolerant and scalable architecture ensure that the data is handled in a secure, consistent manner with zero data loss and supports different forms of data. The solutions provided are consistent and work with different BI tools as well.

**Check out why Hevo is the Best:**

* **Secure**: Hevo has a fault-tolerant architecture that ensures that the data is handled in a secure, consistent manner with zero data loss.
* **Schema Management**: Hevo takes away the tedious task of schema management & automatically detects the schema of incoming data and maps it to the destination schema.
* **Minimal Learning**: Hevo, with its simple and interactive UI, is extremely simple for new customers to work on and perform operations.
* **Hevo Is Built To Scale**: As the number of sources and the volume of your data grows, Hevo scales horizontally, handling millions of records per minute with very little latency.
* **Incremental Data Load**: Hevo allows the transfer of data that has been modified in real-time. This ensures efficient utilization of bandwidth on both ends.
* **Live Support**: The Hevo team is available round the clock to extend exceptional support to its customers through chat, email, and support calls.
* **Live Monitoring**: Hevo allows you to monitor the data flow and check where your data is at a particular point in time.

## Challenges with Data Loading

Many ETL solutions are cloud-based, which accounts for their speed and scalability. But large enterprises with traditional, on-premise infrastructure and data management processes often use custom-built scripts to collect and perform data loading on their own data into storage systems through customized configurations. This can:

* **Slow down analysis.** Each time a data source is added or changed, the system has to be reconfigured, which takes time and hampers the ability to make quick decisions.
* **Increase the likelihood of errors.** Changes and reconfigurations open up the door for human error, duplicate or missing data, and other problems.
* **Require specialized knowledge.** In-house IT teams often lack the skill (and bandwidth) needed to code and monitor ETL functions themselves.
* **Require costly equipment.** In addition to investment in the right human resources, organizations have to purchase, house, and maintain hardware and other equipment to run the process on-site.
* **Unorganized Data:** Loading your data can become unorganized very fast. For ETL voyagers, common roadblocks that many encounters early on can be resolved with proper planning and delivery.
* **Universal formatting:** Before you begin loading your data, make sure that you identify where it is coming from and where you want to go.
* **Loss of data:** Tracking the status of all data is critical for a smooth loading process.
* **Speed:** Although it’s exciting to be closer to your final destination, do not rush through this phase. Errors are most likely to occur during this time.

## Methods for Data Loading

Since data loading is part of the larger ETL process, organizations need a proper understanding of the types of ETL tools and methods available, and which one(s) work best for their needs, budget, and structure.

In the process of Data Loading the data is physically moved to the data warehouse. The Data Loading takes place within a “load window. The tendency is close to real-time updates for data warehouses as warehouses are growing used for operational applications.

**Cloud-based.** ETL tools in the cloud are built for speed and scalability, and often enable real-time data processing. They also include the ready-made infrastructure and expertise of the vendor, who can advise on best practices for each organization’s unique setup and needs.

**Batch processing.** ETL tools that work off batch processing move data at the same scheduled time every day or week. It works best for large volumes of data and for organizations that don’t necessarily need real-time access to their data.

**Open-source.** Many open-source ETL tools are quite cost-effective as their codebase is publicly accessible, modifiable, and shareable. While a good alternative to commercial solutions, these tools can still require some customization or hand-coding.

## Types of Data Loading

Soon after your departure from the extraction phase, you will be faced with the decision of which loading process that you would like to deploy. The **data** **loading** process is the physical movement of the data from the computer systems storing the source database(s) to that which will store the data warehouse database. The entire process of transferring data to a data warehouse repository is referred to in the following ways:

* **Full Load:**This is where all of your data is selected, moved in bulk, and then replaced by new data. Although it is not as complex to navigate through, loading time is much slower. With the overwhelming amount of data being moved at once, it is much easier for data to get lost within the big move.
* **Incremental Load:** This is where you are moving new data in intervals. Due to its intricate nature, delivery time is much faster than its counterpart. However, this speed comes at a cost. Incremental loads are more likely to encounter problems due to the nature of having to manage them as individual batches rather than one big group. **Incremental Load** Periodically applies ongoing changes as per the requirement. After the data is loaded into the data warehouse database, verify the referential integrity between the dimensions and the fact tables to ensure that all records belong to the appropriate records in the other tables. The DBA must verify that each record in the fact table is related to one record in each dimension table that will be used in combination with that fact table.
* **Initial Load**: For the very first time loading all the data warehouse tables.
* **Full Refresh**: Deleting the contents of a table and reloading it with fresh data.

## Data Loading: Refresh versus Update

After the initial load, the data warehouse needs to be maintained and updated and this can be done by the following two methods:

* **Update**-application of incremental changes in the data sources.
* **Refresh**-complete reloads at specified intervals.

### Cloud-Based ETL Tools

In the present-day market, ETL equipment is of great value, and it is very important to recognize the classified method of extraction, transformation, and data loading method.

### 1) Hevo Data

[Hevo Data](https://hevodata.com/), a No-code Data Pipeline, helps to transfer data from [100+ sources](https://hevodata.com/integrations/) to your desired data warehouse/ destination and visualize it in a BI tool. Hevo is fully managed and completely automates the process of not only loading data from your desired source but also enriching the data and transforming it into an analysis-ready form without having to write a single line of code. Its fault-tolerant architecture ensures that the data is handled in a secure, consistent manner with zero data loss.

#### ****Hevo Data Use Case****

Hevo provides a seamless data pipeline experience to companies. Hevo supports pre-built integration with [100+ data sources](https://hevodata.com/integrations/) and allows data migration in real-time. With its ETL, ELT and data transformation capabilities, you will always have analysis-ready data.

### 2) Skyvia

Skyvia is one of the most popular Cloud ETL Tools that provide users with robust data integration, migration and backup support. Being a SaaS application, it only requires users to have smooth internet connectivity and a web browser to be able to access it.

Skyvia’s impeccable no-code data integration wizard allows users to bring in data from a variety of sources such as databases, cloud applications, CSV files, etc. to data warehouses of their choice such as [Google BigQuery](https://cloud.google.com/bigquery), [Amazon Redshift](https://aws.amazon.com/redshift/), etc.

Some of the common issues that you might encounter while using Skyvia is that it doesn’t have fast customer support response times. Similarly, another problem with Skyvia is that it provides less integration support and transformation functionalities.

#### Skyvia Use Case

Skyvia can be a suitable choice for you if you’re looking for a tool that provides a no-code solution to help you automate your ETL pipelines, and you’re okay with minimal data transformation functionalities.

### 3) Xplenty

Xplenty is a robust Cloud ETL Tool that provides an easy-to-use data integration platform and helps you integrate data from a diverse set of sources. Its intuitive user interface lets users set up data pipelines with ease.

It houses powerful data transformation functionalities that allow users to clean, transform and normalise their data into an analysis-ready form. It provides integration support with a diverse set of sources such as on-premise databases, cloud applications, SaaS offerings, etc. such as MongoDB, MySQL, PostgreSQL, etc.

#### Xplenty Use Case

Xplenty can prove to be the right choice for companies that want an easy-to-use no-code data integration platform to manage their ELT and ETL workloads. It can be a good choice for businesses that don’t want to invest much in their engineering bandwidth and prefer leveraging pre-built integrations and functionalities such as drag and drop features.

### 4) Talend

Talend is an open-source Cloud ETL Tool that provides more than 100 pre-built integrations and helps users bring in data from both on-premise and cloud-based applications and store it in the destination of their choice.

With Talend, you can seamlessly work with complex process workflows by making use of the large suite of apps provided by Talend. You can manage the design, testing and deployment of your integrations. It also provides a smooth drag and drops functionality along with an [open studio feature](https://www.talend.com/products/talend-open-studio/) for beginners.

#### ****Talend Use Case****

Talend is a suitable choice for companies that require the flexibility of a diverse set of pre-built integrations and are looking for an open-source ETL solution.

### 5) Informatica PowerCenter

Informatica PowerCenter is an enterprise-grade data integration platform. It is one of the most robust and well-reputed Cloud ETL Tools in the market and is available as one of the tools in the Informatica cloud data management suite.

It performs exceptionally well and helps integrate data from numerous data sources, including various SQL and NoSQL databases. PowerCenter’s data integration platform is highly scalable, and scales as your business grows to manage your business and data needs and helps transform fragmented data into an analysis-ready form.

Some of the common issues you might face using Informatica is that it has a steep learning curve and requires users some time to learn and understand the platform. Similarly, it can turn out to be an expensive solution for various small businesses.

#### ****Informatica PowerCenter Use Case****

If your company is a large enterprise that can support expensive ETL solutions and has a challenging workload that requires high-end performance, then Informatica can be the right choice. You must also be ready to invest a large amount of time in learning the platform as it has a steep learning curve.

### ****6) Fivetran****

Fivetran is a cloud-based ETL tool that delivers high-end performance and provides one of the most versatile integration support, supporting over 90+ SaaS sources apart from various databases and other custom integrations.

It is fully managed and helps deploy automated ETL pipelines in a matter of minutes. It has an easy-to-use platform with a minimal learning curve that allows you to integrate and load data to various data-warehouses such as Google BigQuery, Amazon Redshift, etc. It further adapts to changes in the API and schema easily.

One of the common issues that you might face while using Fivetran is that if there’s an error or technical issue, it becomes challenging to figure out the cause of it. Further, Fivetran customer support tends to be slow in responding to your queries.

#### Fivetran Use Case

Fivetran is a suitable choice for companies that require the flexibility of a diverse set of pre-built integrations.

### 7) Stitch Data

Stitch Data is an open-source cloud-based ETL tool that is suitable for businesses of all kinds, even large enterprises. It provides users with intuitive self-service ELT pipelines that are fully-automated, allowing users to integrate data from various data sources such as SaaS applications, databases and store it in data warehouses, data lakes, etc.

Stitch doesn’t support much transformation functionalities and requires users to load the data and then transform it. It provides more advanced features to users as they go higher in the pricing tiers.

One common issue that most Stitch users face is the lack of support for some data sources and minor technical errors that occur frequently. Although Stitch has an easy-to-use UI, it can take some time to adjust to the UI.

#### ****Stitch Use Case****

Stitch is suitable for companies that are looking for an open-source tool that provides a no-code solution to help them automate their ETL pipelines, and are okay with having minimal data transformation functionalities.

## Conclusion

This gives a comprehensive overview of the Data Loading component of the ETL process. It also gave loads of tools that are cloud-based and can ease the process of ETL.

To make things easier, Hevo comes into the picture.[Hevo Data](https://hevodata.com/) is a No-code Data Pipeline and has awesome 100+ pre-built Integrations that you can choose from.