



# *****Road Map*****

1. Programming Language
2. Data Structures and Algorithms 🡪 Problem solving skills.
   1. Array 🡪 Easy to Moderate level 🡪 LeetCode
   2. Strings 🡪 Easy to Moderate level 🡪 LeetCode
3. Mathematics 🡪 Linear Algebra, Calculus, Probability and Random Variables, Matrix Theory, Statistics
4. Database 🡪 SQL
5. Data Science Life cycle
   1. Data Acquisition
   2. Data Pipeline
   3. EDA (Exploratory Data Analysis)
      1. Null values
      2. Outliers
      3. Mean, median, mode.
      4. Categorical handling etc.…...
   4. Data Modeling 🡪 Applying..
      1. ML algorithms depending upon scenario.
         1. Supervised
         2. Unsupervised
      2. Deep Learning Algorithms (Tensor flow, Pytorch, etc.)
      3. NLP
   5. Model Evaluation 🡪 Confusion Matrix, Accuracy etc.
   6. Model Deployment
6. Data Science Projects
7. Prepare for Interview

Table Of Contents

[Road Map 1-2](#_Toc145268696)

[Chapter 1 Data 1-11](#_Toc145268697)

[A diagram of information and data

Description automatically generated 1-11](#_Toc145268698)

[Data? 1-12](#_Toc145268699)

[Categories of Data 1-12](#_Toc145268700)

[Quantitative data: 1-13](#_Toc145268701)

[Qualitative data: 1-13](#_Toc145268702)

[Data Examples 1-13](#_Toc145268703)

[Types of Data 1-14](#_Toc145268704)

[DBMS 1-17](#_Toc145268705)

[Database 1-17](#_Toc145268706)

[Types of Databases 1-17](#_Toc145268707)

[Components of Database 1-18](#_Toc145268708)

[Operations on a Database 1-18](#_Toc145268709)

[Database Concepts 1-19](#_Toc145268710)

[Database Administration and Security 1-19](#_Toc145268711)

[Relational Database: 1-19](#_Toc145268712)

[DBMS Architecture 1-level, 2-Level, 3-Level 1-19](#_Toc145268713)

[Types of DBMS Architecture 1-19](#_Toc145268714)

[1-Tier Architecture 1-20](#_Toc145268715)

[2-Tier Architecture 1-21](#_Toc145268716)

[3-Tier Architecture 1-22](#_Toc145268717)

[Difference Between Two-Tier And Three-Tier database architecture 1-29](#_Toc145268718)

[ER-Model 1-32](#_Toc145268719)

[Some Important Terms 1-33](#_Toc145268720)

[Database Design 1-35](#_Toc145268721)

[Data Retrieval (SQL, RA) 1-39](#_Toc145268722)

[File Structure 1-44](#_Toc145268723)

[Indexing Type: 1-46](#_Toc145268724)

[1. Single level Index 1-46](#_Toc145268725)

[2. Multilevel Index 1-47](#_Toc145268726)

[Transaction and Concurrency Control 1-48](#_Toc145268727)

[Database Languages in DBMS 1-53](#_Toc145268728)

[Types of Database Languages 1-54](#_Toc145268729)

[1. Data Definition Language (DDL) 1-54](#_Toc145268730)

[2. Data Manipulation Language (DML) 1-55](#_Toc145268731)

[3. Data Control Language (DCL) 1-55](#_Toc145268732)

[4. Transaction Control Language (TCL) 1-56](#_Toc145268733)

[Chapter 2 Python 2-57](#_Toc145268734)

[OOPS 2-57](#_Toc145268735)

[What is OOP? 2-57](#_Toc145268736)

[Why Python is not a pure OOP language? 2-57](#_Toc145268737)

[Why OOP? 2-58](#_Toc145268738)

[Principles 2-59](#_Toc145268739)

[Basic Concepts of OOP 2-59](#_Toc145268740)

[Object 2-59](#_Toc145268741)

[Class 2-60](#_Toc145268742)

[Chapter 3 DSA 3-62](#_Toc145268743)

[ 3-62](#_Toc145268744)

[Algorithm Analysis 3-62](#_Toc145268745)

[Recursion 3-62](#_Toc145268746)

[Array Based Sequences 3-62](#_Toc145268747)

[Stacks, Queues and Deques 3-62](#_Toc145268748)

[Linked Lists 3-62](#_Toc145268749)

[Trees 3-62](#_Toc145268750)

[Priority Queues 3-62](#_Toc145268751)

[Maps, Hash Tables and Skip Lists 3-62](#_Toc145268752)

[Search 3-62](#_Toc145268753)

[Sorting and Selection 3-62](#_Toc145268754)

[Text Processing 3-62](#_Toc145268755)

[Graph Algorithms 3-62](#_Toc145268756)

[Memory Management and Binary Trees 3-62](#_Toc145268757)

[Chapter 4 Linear Algebra 4-65](#_Toc145268758)

[Chapter 5 Calculus 5-67](#_Toc145268759)

[Chapter 6 Statistics 6-69](#_Toc145268760)

[Chapter 7 Probability 7-72](#_Toc145268761)

[Chapter 8 Data Manipulation and Analysis 8-74](#_Toc145268762)

[Numpy 8-74](#_Toc145268763)

[Pandas 8-74](#_Toc145268764)

[Matplotlib 8-74](#_Toc145268765)

[Seaborn 8-74](#_Toc145268766)

[Scrapy or Beautiful Soup 8-74](#_Toc145268767)

[Scikit-learn 8-74](#_Toc145268768)

[TensorFlow (from Google) 8-74](#_Toc145268769)

[Pytorch (from Facebook) 8-74](#_Toc145268770)

[Spacy (NLP)/NLTK 8-74](#_Toc145268771)

[OpenCV (Image Processing) 8-74](#_Toc145268772)

[Chapter 9 Machine Learning 9-76](#_Toc145268773)

[Chapter 10 Deep Learning 10-78](#_Toc145268774)

[Chapter 11 Natural Language Processing (NLP) 11-80](#_Toc145268775)

[Chapter 12 AI 12-82](#_Toc145268776)

[Chapter 13 Data 13-84](#_Toc145268777)

[Chapter 14 Machine Learning 14-85](#_Toc145268778)

[OOPS 14-85](#_Toc145268779)

[What is OOP? 14-85](#_Toc145268780)

[Why OOP? 14-85](#_Toc145268781)

[When OOP? 14-85](#_Toc145268782)

[How OOP? 14-85](#_Toc145268783)

[Where OOP? 14-85](#_Toc145268784)

[Goals, Principles, Design Pattern 14-85](#_Toc145268785)

[Chapter 15 Study Plan 15-87](#_Toc145268786)

[Week 1: Python Basics and Advanced Features 15-87](#_Toc145268787)

[Week 2: Python Libraries for Data Science and Intro to Data Analysis 15-87](#_Toc145268788)

[Week 3: Data Visualization 15-87](#_Toc145268789)

[Week 4: Exploratory Data Analysis (EDA) 15-87](#_Toc145268790)

[Week 5: Machine Learning with Scikit-learn 15-88](#_Toc145268791)

[Week 6: Advanced Machine Learning Concepts 15-88](#_Toc145268792)

[Week 7: Intro to Deep Learning and Neural Networks with TensorFlow/Keras 15-88](#_Toc145268793)

[Week 8: Capstone Projects and Further Resources 15-88](#_Toc145268794)

Learning data science and AI is an exciting journey that requires a combination of theoretical understanding and hands-on practice. Here's a roadmap to help you get started:

\*\*1. **Foundations:**

* **Mathematics:** Brush up on linear algebra, calculus, and probability/statistics. These are essential for understanding the algorithms and concepts in data science and AI.

**2. Programming:**

* **Python:** Learn Python, a widely-used programming language in data science and AI. Familiarize yourself with libraries like NumPy, Pandas, and Matplotlib for data manipulation, analysis, and visualization.

**3. Data Manipulation and Analysis:**

* **Pandas:** Learn how to clean, manipulate, and analyze data using the Pandas library.
* **Data Visualization:** Learn Matplotlib and Seaborn for creating meaningful visualizations to understand your data.

**4. Machine Learning:**

* **Introduction to Machine Learning:** Understand the basics of supervised, unsupervised, and reinforcement learning.
* **Scikit-Learn:** Explore the Scikit-Learn library for implementing various machine learning algorithms.
* **Model Evaluation:** Learn about metrics, cross-validation, and hyperparameter tuning.
* **Feature Engineering:** Understand how to preprocess and engineer features for better model performance.

**5. Deep Learning:**

* **Neural Networks:** Learn the basics of artificial neural networks and their components.
* **Deep Learning Frameworks:** Choose a deep learning framework like TensorFlow or PyTorch and learn to build and train neural networks.
* **Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs):** Dive into advanced neural network architectures for image and sequence data.

**6. Data Preprocessing and Feature Engineering:**

* **Feature Scaling, Normalization:** Understand techniques to scale and normalize data for better model performance.
* **Handling Missing Data:** Learn strategies to handle missing values in datasets.
* **One-Hot Encoding, Label Encoding:** Explore methods to transform categorical data into numerical formats.

**7. Advanced Topics:**

* **Natural Language Processing (NLP):** Learn how to process and analyze text data using techniques like tokenization, sentiment analysis, and text generation.
* **Reinforcement Learning:** Dive into the field of reinforcement learning, which deals with training agents to make sequential decisions.
* **Time Series Analysis:** Understand techniques for analyzing and predicting time-dependent data.

**8. Projects:**

* Work on real-world projects to apply your knowledge and gain practical experience.
* Start with small projects and gradually work your way up to more complex ones.

**9. Online Resources:**

* Utilize online platforms like Coursera, edX, Udacity, and Khan Academy for structured courses on data science and AI.
* Follow blogs, YouTube channels, and forums to stay updated on the latest trends and techniques.

**10. Kaggle:**

* Join Kaggle, a platform for data science competitions. Participate in competitions to solve real-world problems and learn from others.

**11. Books:**

* Consider reading foundational books like "Python for Data Analysis" by Wes McKinney, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron, and "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

Remember that learning data science and AI is an ongoing process. The field is constantly evolving, so staying curious and open to new ideas is crucial. Practical experience and building a strong portfolio of projects will be invaluable as you progress in your journey.

youtube channels to learn datascience and ai

ChatGPT

There are numerous YouTube channels that offer high-quality tutorials and educational content on data science and AI. Here are some popular YouTube channels that you might find valuable for learning:

1. **Sentdex** - Provides tutorials on various topics including Python programming, machine learning, deep learning, and data analysis. Link: [Sentdex YouTube Channel](https://www.youtube.com/user/sentdex)
2. **3Blue1Brown** - Offers visually engaging videos that explain complex mathematical and AI concepts, making them easier to understand. Link: [3Blue1Brown YouTube Channel](https://www.youtube.com/c/3blue1brown)
3. **StatQuest with Josh Starmer** - Breaks down statistical concepts and machine learning algorithms using easy-to-understand animations. Link: [StatQuest YouTube Channel](https://www.youtube.com/user/joshstarmer)
4. **Krish Naik** - Covers a wide range of topics in data science, machine learning, and AI through tutorials and practical demonstrations. Link: [Krish Naik YouTube Channel](https://www.youtube.com/user/krishnaik06)
5. **Tech With Tim** - Offers tutorials on Python programming, data science, machine learning, and game development. Link: [Tech With Tim YouTube Channel](https://www.youtube.com/c/TechWithTim)
6. **Codebasics** - Provides tutorials on Python, machine learning, and data science, with a focus on practical examples. Link: [Codebasics YouTube Channel](https://www.youtube.com/c/codebasics)
7. **Data School** - Offers tutorials on Python, Pandas, machine learning, and data visualization. Link: [Data School YouTube Channel](https://www.youtube.com/user/dataschool)
8. **Siraj Raval** - Covers a variety of AI and machine learning topics, often discussing cutting-edge developments and concepts. Link: [Siraj Raval YouTube Channel](https://www.youtube.com/c/SirajRaval)
9. **AI Engineering** - Focuses on topics related to artificial intelligence, machine learning, and data science, with in-depth tutorials and explanations. Link: [AI Engineering YouTube Channel](https://www.youtube.com/c/AIEngineering)
10. **DataCamp** - Provides tutorials, courses, and webinars on data science and machine learning topics. Link: [DataCamp YouTube Channel](https://www.youtube.com/c/datacamp)

Top of Form

# *****Data*****

# A diagram of information and data Description automatically generated

***Data***

https://www.mathsisfun.com/data/index.html

Data comes in various forms such as numbers, words, etc. that describes discrete facts about an objective reality.

Example: BuyMore, consumer electronic goods, 12%, South India, 80%, North India

***Information:***

It comes from providing context to data. It is also the description of data. Information helps to understand patterns between factual data and give it a meaning. Information helps us to answers questions like who, when, what, where etc.

**Example:** BuyMore is a retail chain that has had done  12%  sales of consumer electronic goods in south India 80% sales in north India.

***Knowledge:***

It is synthesis of data and information leads us to answer the how question and take business decisions.

**Example:** BuyMore management got a knowledge that sales of consumer electronic goods is poor in south India compared to north India  and hence management of BuyMore retail store initiates some strategy to promote the products.

## ***Data?***

Data is defined as a collection of individual facts or statistics.

(While “datum” is technically the singular form of “data,” it’s not commonly used in everyday language.)

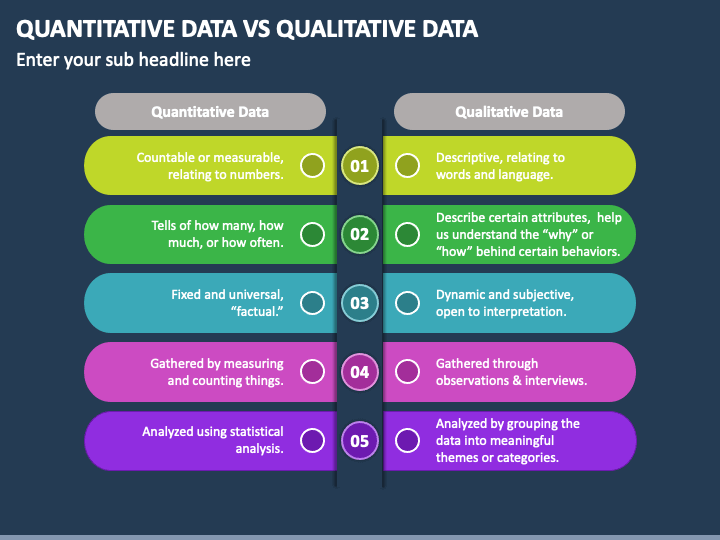
Data can be in the form of text, figures, images, numbers, graphs, or symbols. Data gathered by observations, measurements, research, or analysis.

Data is a raw form of knowledge and simple. It may even seem useless until it is analyzed, organized, and interpreted.

### ***Categories of Data***

Data can be broadly categorized as

* **Quantitative data** is provided in numerical form that can answer questions like “How many”, “How much”, “How often”. For example, the weight, volume, or cost of an item.
* **Qualitative data** is descriptive, but non-numerical, like the name, sex, or eye color of a person.



### *Quantitative data:*

**Quantitative data** can be Discrete or Continuous:

* **Discrete data** can only take certain values (like whole numbers)

Example: the number of students in a class

* **Continuous data** can take any value (within a range)

Example: A person's height: could be any value (within the range of human heights), not just certain fixed heights.

Time in a race: you could even measure it to fractions of a second,

simply: **Discrete data** is counted; **Continuous data** is measured.

Data are easily amenable (responsive)

### *Qualitative data:*

Qualitative data, also known as the categorical data, describes the data that fits into the categories. Qualitative data are not numerical. Categorical measures are defined in terms of natural language specifications, but not in terms of numbers. One of the examples is a grouped data. More precisely, categorical data could be derived from qualitative data analysis that are countable, or from quantitative data analysis grouped within given intervals.

### ***Data Examples***

* The number of visitors to a website in one month
* Inventory levels in a warehouse on a specific date
* Individual satisfaction scores on a customer service survey
* The price of a competitors’ product.Top of Form

### ***Types of Data***

* **Structured Data:**

Data that is organized in a tabular format, often found in databases. Each row corresponds to a record, and each column corresponds to a field.

* **Unstructured Data:**

Data that does not have a predefined structure, such as text documents, images, and videos.

* **Semi-Structured Data:**

Data that is not strictly organized like structured data but still has some level of structure, such as JSON or XML files.

Data can be categorized into various types based on its nature, structure, and characteristics. The main types of data include:

**\*\*1. Numerical Data:**

* Numerical data consists of numbers and is used for quantitative measurements.
* Two subtypes:
  + **Continuous Data**: Data that can take any value within a specific range. For example, height, weight, temperature.
  + **Discrete Data**: Data that can only take specific, distinct values. For example, the number of cars in a parking lot, the count of items sold.

**2. Categorical Data:**

* Categorical data represents categories or labels and is used for qualitative analysis.
* Two subtypes:
  + Nominal Data: Categories without any inherent order or ranking. For example, colors, types of animals.
  + Ordinal Data: Categories with a meaningful order or ranking. For example, educational levels (high school, bachelor's, master's).

**3. Text Data:**

* Text data consists of unstructured textual content, such as documents, articles, emails, and social media posts.
* Often used in natural language processing (NLP) tasks like sentiment analysis and text classification.

**4. Time Series Data:**

* Time series data is collected over successive time intervals. It's used for analyzing trends, patterns, and seasonality.
* Common in fields like finance, economics, and environmental monitoring.

**5. Image Data:**

* Image data represents visual information in the form of pixels.
* Used in computer vision tasks like image recognition, object detection, and image generation.

**6. Audio Data:**

* Audio data represents sound waves and is used in applications like speech recognition and music analysis.

**7. Geospatial Data:**

* Geospatial data includes information related to geographical locations, coordinates, and maps.
* Used in geographic information systems (GIS) and location-based services.

**8. Binary Data:**

* Binary data consists of only two possible values, often represented as 0s and 1s.
* Used in computer systems, cryptography, and digital communication.

**9. Mixed Data:**

* Mixed data involves a combination of different data types within a dataset. For instance, a dataset about customers might include numerical attributes (age), categorical attributes (gender), and text attributes (address).

Understanding the types of data is crucial for determining appropriate analysis methods, data preprocessing techniques, and modeling approaches. Different types of data require different methods for handling, visualizing, and extracting insights.

**Importance of Data**

Data serves as the foundation for making informed decisions, deriving insights, identifying trends, and developing models in various fields such as business, science, healthcare, and more.

**Data Collection**

The process of gathering data from various sources, which can include surveys, sensors, websites, databases, and more.

**Attributes or Features**

These are the individual characteristics or variables that make up a dataset. For example, in a dataset about customers, attributes could include name, age, and location.

SQL is a powerful language used for managing and manipulating relational databases. Below is a basic SQL course outline to get you started:

**1. Introduction to Databases:**

* Understand what databases are and why they are used.
* Learn about the differences between relational and non-relational databases.

**2. Basics of SQL:**

* Learn about SQL syntax, statements, and clauses.
* Explore SELECT statements to retrieve data from a database.

**3. Filtering and Sorting Data:**

* Use WHERE clause to filter data based on conditions.
* Learn to use ORDER BY clause to sort query results.

**4. Working with Functions:**

* Understand various SQL functions like COUNT, SUM, AVG, MAX, MIN, etc.
* Explore how to perform calculations on data.

**5. Joins and Relationships:**

* Learn about different types of joins (INNER, LEFT, RIGHT, FULL) to combine data from multiple tables.
* Understand primary keys and foreign keys for establishing relationships between tables.

**6. Grouping and Aggregating Data:**

* Use GROUP BY clause to group data based on specific columns.
* Learn about HAVING clause to filter grouped results.

**7. Subqueries and Derived Tables:**

* Explore subqueries to perform operations within queries.
* Understand derived tables (also known as subquery factoring or Common Table Expressions).

**8. Data Modification:**

* Learn how to INSERT, UPDATE, and DELETE data from tables.
* Understand the importance of data integrity and transaction management.

**9. Advanced SQL Topics:**

* Study concepts like indexes, views, stored procedures, and triggers.
* Explore techniques for optimizing SQL queries.

1. **Practical Projects:** - Apply SQL concepts to real-world scenarios by working on projects that involve data manipulation, analysis, and reporting.

## ***DBMS***

### ***Database***

A database is an organized collection of structured information of data that can be easily accessed, managed, and updated typically stored electronically in a computer system.

Data can’t be accessed directly so user have to use some application or software called as “Database Software” . Database software is used to create, edit, and maintain database files and records and it enables easier file control, handles data storage, backup and reporting, multi-access control, and security. Database software is sometimes also referred to as a “database management system” (DBMS).

Strong database security is especially important today, as data theft becomes more frequent.

Database software makes data management simpler by enabling users to store data in a structured form and then access it. It typically has a graphical interface to help create and manage the data and, in some cases, users can construct their own databases by using database software. A DBMS serves as an interface between the database and its end users or programs, allowing users to retrieve, update, and manage how the information is organized and optimized.

Databases can be classified based on types of content: bibliographic, full-text, numeric, and images.

### ***Types of Databases***

1. **Relational Database (RDBMS)**: Organized into tables with rows and columns. Examples include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.
2. **NoSQL Database**: Non-relational databases that can store and retrieve data in ways other than tabular relations. Subtypes include:
   * **Document-based databases** (e.g., MongoDB): Data is stored in document-like structures.
   * **Key-Value stores** (e.g., Redis, Riak): Use keys and values for data storage and retrieval.
   * **Column-based stores** (e.g., Cassandra, HBase): Data is stored in cells grouped in columns.
   * **Graph-based databases** (e.g., Neo4j): Used for data whose relations are best represented as a graph.
3. **Object-Oriented Databases**: Data is stored as objects, which can be manipulated using a programming language.
4. **Distributed Databases**: A single logical database that is spread across more than one physical computer or platform. Useful for cloud-based systems.
5. **In-memory Databases** (e.g., Redis, SAP HANA): Data is stored in main memory (rather than on disk) to produce near-instantaneous response times.
6. **Time-series Databases** (e.g., Influx DB, Timescale DB): Optimized for time-stamped or time-series data.
7. **Hierarchical Databases**: Data is stored hierarchically, in parent-child relationships.



1. **Network Databases**: More flexible than hierarchical databases, allowing for many-to-many relationships.

### ***Components of Database***

* **Tables or Collections**: Where the data is stored. In relational databases, these are tables. In document-based NoSQL databases like MongoDB, these are called collections.
* **Schema**: Defines the structure of the database. It describes the shape of the data, the types, constraints, relationships, and other aspects.
* **Database Engine**: Software component that helps to store, retrieve, and manage data in the database.
* **Query Processor**: Processes the database queries.
* **Index**: Enhances the speed of data retrieval operations on a database.

### ***Operations on a Database***

* **CRUD Operations**: Basic operations one can perform on database data. Stands for Create, Read, Update, and Delete.
* **Querying**: The process of requesting specific data from a database. SQL (Structured Query Language) is the standard language for querying relational databases.
* **Transactions**: Represents any change in the database. Databases that support transactions (like most RDBMS systems) are called transactional databases.

### ***Database Concepts***

* **ACID Properties**: Properties that ensure reliable processing in a database. Stands for Atomicity, Consistency, Isolation, and Durability.
* **Normalization**: The process of efficiently organizing data in a database.
* **Join**: Combines rows from two or more tables based on a related column between them.

### ***Database Administration and Security***

* **Backup and Recovery**: Protecting the database against data loss.
* **Concurrency Control**: Ensuring that multiple transactions can occur concurrently without leading to inconsistencies.
* **Security**: Protecting the database from unauthorized access.

Understanding databases is fundamental in the fields of computer science and information technology. Whether you're a developer, a data analyst, or in another role, databases often play a crucial role in software and web development, analytics, and more.

### ***Relational Database***:Top of Form

# DBMS Architecture 1-level, 2-Level, 3-Level

A Database store a lot of critical information to access data quickly and securely. Hence it is important to select the correct architecture for efficient data management. DBMS Architecture helps users to get their requests done while connecting to the database. We choose database architecture depending on several factors like the size of the database, number of users, and relationships between the users. There are two types of database models that we generally use, are logical model and physical model. Several types of architecture are there in the database which we will deal with in the next section.

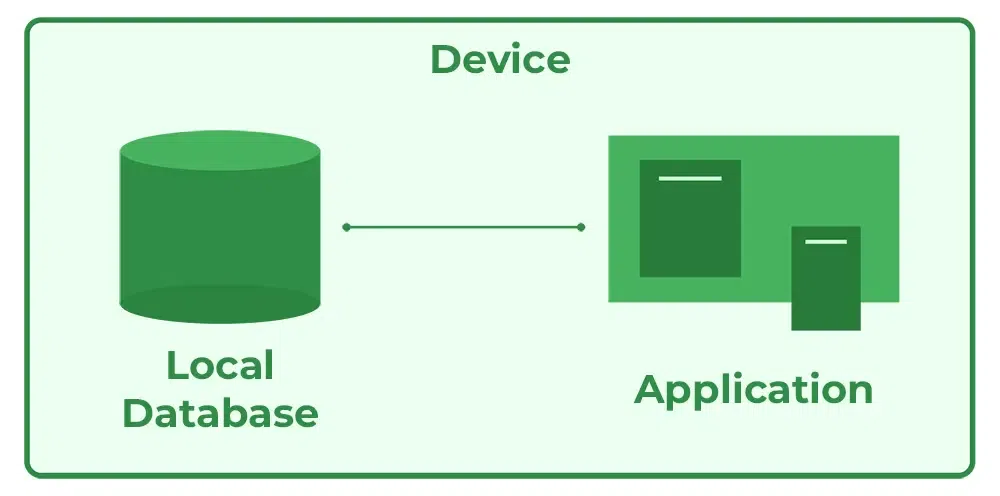
## Types of DBMS Architecture

There are several types of DBMS Architecture that we use according to the usage requirements. Types of DBMS Architecture are discussed here.

* 1-Tier Architecture
* 2-Tier Architecture
* 3-Tier Architecture

### **1-Tier Architecture**

In 1-Tier Architecture the database is directly available to the user, the user can directly sit on the DBMS and use it that is, the client, server, and Database are all present on the same machine. For Example: to learn SQL we set up an SQL server and the database on the local system. This enables us to directly interact with the relational database and execute operations. The industry won’t use this architecture they logically go for 2-Tier and 3-Tier Architecture.



*DBMS 1-Tier Architecture*

**Advantages of 1-Tier Architecture**

Below mentioned are the advantages of 1-Tier Architecture.

* **Simple Architecture:** 1-Tier Architecture is the most simple architecture to set up, as only a single machine is required to maintain it.
* **Cost-Effective:** No additional hardware is required for implementing 1-Tier Architecture, which makes it cost-effective.
* **Easy to Implement:**1-Tier Architecture can be easily deployed, and hence it is mostly used in small projects.

### **2-Tier Architecture**

The 2-tier architecture is similar to a basic[client-server model](https://www.geeksforgeeks.org/client-server-model/). The application at the client end directly communicates with the database on the server side. APIs like ODBC and JDBC are used for this interaction. The server side is responsible for providing query processing and transaction management functionalities. On the client side, the user interfaces and application programs are run. The application on the client side establishes a connection with the server side in order to communicate with the DBMS.   
An advantage of this type is that maintenance and understanding are easier, and compatible with existing systems. However, this model gives poor performance when there are a large number of users.



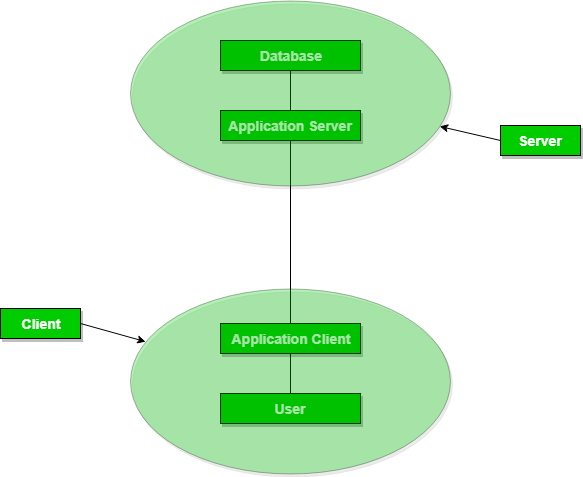
*DBMS 2-Tier Architecture*

**Advantages of 2-Tier Architecture**

* **Easy to Access:** 2-Tier Architecture makes easy access to the database, which makes fast retrieval.
* **Scalable:** We can scale the database easily, by adding clients or by upgrading hardware.
* **Low Cost:** 2-Tier Architecture is cheaper than 3-Tier Architecture and [Multi-Tier Architecture](https://www.geeksforgeeks.org/multi-tier-architecture-of-data-warehouse/).
* **Easy Deployment:** 2-Tier Architecture is easy to deploy than 3-Tier Architecture.
* **Simple:** 2-Tier Architecture is easily understandable as well as simple because of only two components.

### **3-Tier Architecture**

In [3-Tier Architecture](https://www.geeksforgeeks.org/introduction-of-3-tier-architecture-in-dbms-set-2/), there is another layer between the client and the server. The client does not directly communicate with the server. Instead, it interacts with an application server which further communicates with the database system and then the query processing and transaction management takes place. This intermediate layer acts as a medium for the exchange of partially processed data between the server and the client. This type of architecture is used in the case of large web applications.



*DBMS 3-Tier Architecture*

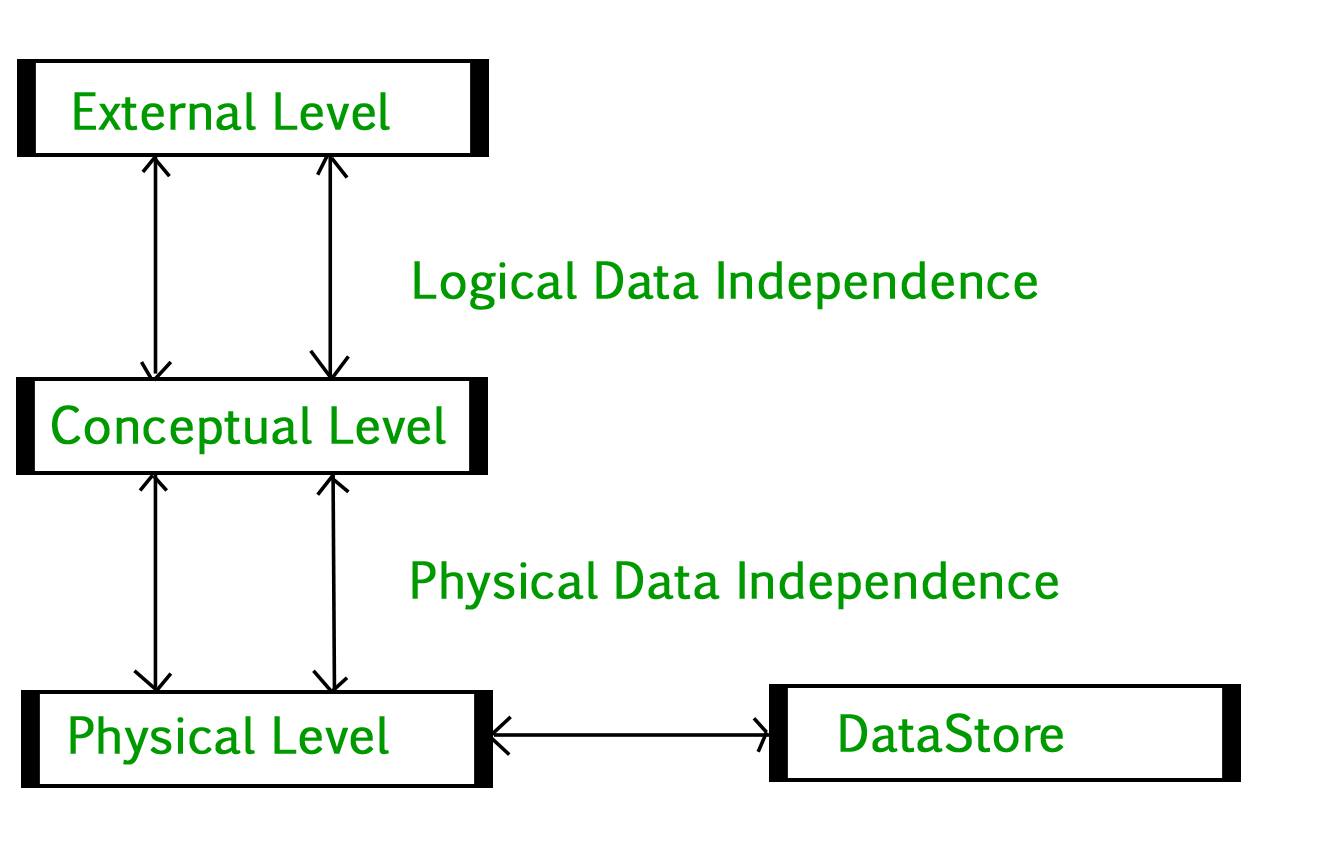
**Advantages of 3-Tier Architecture**

* **Enhanced scalability:** Scalability is enhanced due to distributed deployment of application servers. Now, individual connections need not be made between the client and server.
* **Data Integrity:** 3-Tier Architecture maintains Data Integrity. Since there is a middle layer between the client and the server, data corruption can be avoided/removed.
* **Security:**3-Tier Architecture Improves Security. This type of model prevents direct interaction of the client with the server thereby reducing access to unauthorized data.

**Disadvantages of 3-Tier Architecture**

* **More Complex:**3-Tier Architecture is more complex in comparison to 2-Tier Architecture. Communication Points are also doubled in 3-Tier Architecture.
* **Difficult to Interact:** It becomes difficult for this sort of interaction to take place due to the presence of middle layers.

**DBMS 3-tier Architecture**

DBMS 3-tier architecture divides the complete system into three inter-related but independent modules as shown below: [](https://media.geeksforgeeks.org/wp-content/uploads/dbms-3tier.jpg)

1. **Physical Level:** At the physical level, the information about the location of database objects in the data store is kept. Various users of DBMS are unaware of the locations of these objects.In simple terms,physical level of a database describes how the data is being stored in secondary storage devices like disks and tapes and also gives insights on additional storage details.
2. **Conceptual Level:**At conceptual level, data is represented in the form of various database tables. For Example, STUDENT database may contain STUDENT and COURSE tables which will be visible to users but users are unaware of their storage.Also referred as logical schema,it describes what kind of data is to be stored in the database.
3. **External Level:** An external level specifies a view of the data in terms of conceptual level tables.  Each external level view is used to cater to the needs of a particular category of users. For Example, FACULTY of a university is interested in looking course details of students, STUDENTS are interested in looking at all details related to academics, accounts, courses and hostel details as well. So, different views can be generated for different users. The main focus of external level is data abstraction.

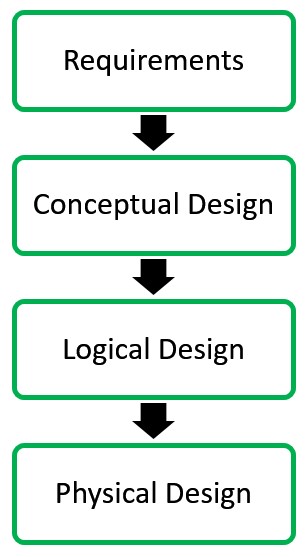
**Data Independence**

Data independence means a change of data at one level should not affect another level. Two types of data independence are present in this architecture:

1. **Physical Data Independence:** Any change in the physical location of tables and indexes should not affect the conceptual level or external view of data. This data independence is easy to achieve and implemented by most of the DBMS.
2. **Conceptual Data Independence:** The data at conceptual level schema and external level schema must be independent. This means a change in conceptual schema should not affect external schema. e.g.; Adding or deleting attributes of a table should not affect the user’s view of the table. But this type of independence is difficult to achieve as compared to physical data independence because the changes in conceptual schema are reflected in the user’s view.

**Phases of database design**

Database designing for a real-world application starts from capturing the requirements to physical implementation using DBMS software which consists of following steps shown below:

[](https://media.geeksforgeeks.org/wp-content/uploads/20190506212511/dbms-phases.jpg)

**Conceptual Design:**The requirements of database are captured using high level conceptual data model. For Example, the ER model is used for the conceptual design of the database. **Logical Design:**Logical Design represents data in the form of relational model. ER diagram produced in the conceptual design phase is used to convert the data into the Relational Model. **Physical Design:** In physical design, data in relational model is implemented using commercial DBMS like Oracle, DB2.

**Advantages of DBMS**

DBMS helps in efficient organization of data in database which has following advantages over typical file system:

* **Minimized redundancy and data inconsistency:** Data is normalized in DBMS to minimize the redundancy which helps in keeping data consistent. For Example, student information can be kept at one place in DBMS and accessed by different users.This minimized redundancy is due to primary key and foreign keys
* **Simplified Data Access:** A user need only name of the relation not exact location to access data, so the process is very simple.
* **Multiple data views:** Different views of same data can be created to cater the needs of different users. For Example, faculty salary information can be hidden from student view of data but shown in admin view.
* **Data Security:** Only authorized users are allowed to access the data in DBMS. Also, data can be encrypted by DBMS which makes it secure.
* **Concurrent access to data:**Data can be accessed concurrently by different users at same time in DBMS.
* **Backup and Recovery mechanism:** DBMS backup and recovery mechanism helps to avoid data loss and data inconsistency in case of catastrophic failures.

Also see

* [All DBMS Articles](https://www.geeksforgeeks.org/category/dbms/)
* [DBMS Quizzes](https://www.geeksforgeeks.org/quiz-corner-gq/)

In DBMS, the 3-tier architecture is a client-server architecture that separates the user interface, application processing, and data management into three distinct tiers or layers. The 3-tier architecture is widely used in modern web applications and enterprise systems because it offers scalability, flexibility, and security. Here is a brief description of each tier in the 3-tier architecture:

1. Presentation Tier: The presentation tier is the user interface or client layer of the application. It is responsible for presenting data to the user and receiving input from the user. This tier can be a web browser, mobile app, or desktop application.
2. Application Tier: The application tier is the middle layer of the 3-tier architecture. It is responsible for processing and managing the business logic of the application. This tier communicates with the presentation tier to receive user input and communicates with the data management tier to retrieve or store data. This tier may include application servers, web servers, or APIs.
3. Data Management Tier: The data management tier is the bottom layer of the 3-tier architecture. It is responsible for managing and storing data. This tier can include databases, data warehouses, or data lakes. The data management tier communicates with the application tier to receive or store data.
4. The 3-tier architecture in DBMS provides several benefits, including:
5. Scalability: The architecture separates the application processing and data management layers, which allows for easy scalability of each layer independently.
6. Flexibility: The architecture allows for the replacement or upgrade of one layer without affecting the other layers.
7. Security: The architecture provides an additional layer of security, as the data management tier can be isolated from the application and presentation tiers, reducing the risk of unauthorized access.

Overall, the 3-tier architecture in DBMS is a flexible, scalable, and secure approach to building modern web applications and enterprise systems. It separates the user interface, application processing, and data management into distinct layers, providing clear boundaries between each layer and improving system performance, reliability, and maintainability.

# Difference Between Two-Tier And Three-Tier database architecture

**1. Two-Tier Database Architecture –**  
In two-tier, the application logic is either buried inside the User Interface on the client or within the database on the server (or both). With two-tier client/server architectures, the user system interface is usually located in the user’s desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients.

**2. Three-Tier Database Architecture –**  
In three-tier, the application logic or process lives in the middle-tier, it is separated from the data and the user interface. Three-tier systems are more scalable, robust and flexible. In addition, they can integrate data from multiple sources. In the three-tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers.  
**Difference Between Two-Tier And Three-Tier Database Architecture**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Two-Tier Database Architecture** | **Three-Tier Database Architecture** |
| 1 | It is a Client-Server Architecture. | It is a Web-based application. |
| 2 | In two-tier, the application logic is either buried inside the user interface on the client or within the database on the server (or both). | In three-tier, the application logic or process resides in the middle-tier, it is separated from the data and the user interface. |
| 3 | Two-tier architecture consists of two layers : Client Tier and Database (Data Tier). | Three-tier architecture consists of three layers : Client Layer, Business Layer and Data Layer. |
| 4 | It is easy to build and maintain. | It is complex to build and maintain. |
| 5 | Two-tier architecture runs slower. | Three-tier architecture runs faster. |
| 6 | It is less secured as client can communicate with database directly. | It is secured as client is not allowed to communicate with database directly. |
| 7 | It results in performance loss whenever the users increase rapidly. | It results in performance loss whenever the system is run on Internet but gives more performance than two-tier architecture. |
| 8 | Example – Contact Management System created using MS-Access or Railway Reservation System, etc. | Example – Designing registration form which contains text box, label, button or a large website on the Internet, etc. |

**DBMS over File System**

The file system has numerous issues, which were resolved with the help of DBMS, the issues with the file system are:

* **Data Redundancy:**Same data can be stored at multiple places.
* **Data Inconsistency:** If multiple copies of the same data have different content in each copy. Like, the phone number of students is different in academic and accounts files.
* **Data access:** In a file system, accessing data was difficult and insecure as well. Accessing data concurrently was not possible.
* **No Backup and Recovery:** There is no backup and recovery in the file system that can lead to data loss.

## ER-Model

**ER Diagram:**An ER diagram is a model of a logical view of the database which is represented using the following components:

* **Entity:**The entity is a real-world object, represented using a rectangular box.
  + **Strong Entity:**A strong entity set has a primary key and all the tuples of the set can be identified using that primary key
  + **Weak entity:**When an entity does not have sufficient attributes to form a primary key. Weak entities are associated with another strong entity set also known as identifying an entity. A weak entity’s existence depends upon the existence of its identifying entity. The weak entity is represented using a double-lined or bold-lined rectangle.
* **Attribute:** Attribute is the properties or characteristics of the real-world object. It is represented using an oval.
  + **Key attribute:** The attribute which determines each entity uniquely is known as the Key attribute. It is represented by an oval with an underlying line.
  + **Composite Attribute:**An attribute that is composed of many other attributes. E.g. address is an attribute it is formed of other attributes like state, district, city, street, etc. It is represented using an oval comprises of many other ovals.
  + **Multivalued Attribute:** An attribute that can have multiple values, like a mobile number. It is represented using a double-lined oval.
  + **Derived attribute:**An attribute that can be derived from other attributes. E.g. Age is an attribute that can be derived from another attribute Data of Birth. It is represented using a dashed oval.
* **Relationship:**A relationship is an association between two or more entities. Entities are connected or related to each other and this relationship is represented using a diamond.

### **Some Important Terms**

**Cardinality of DBMS:**Cardinality of relation expresses the maximum number of possible relationship occurrences for an entity participating in a relationship. Cardinality of a relationship can be defined as the number of times an entity of an entity set participates in a relationship set. Let’s suppose a binary relationship R between two entity sets A and B.  The relationship must have one of the following mapping cardinalities:

* **One-to-One:**When one entity of A is related to at most one entity of B, and vice-versa.
* **One-to-Many:** When one entity of A is related to one or more than one entity of B. Whereas B is associated with at most one entity in A.
* **Many-to-One:**When one entity of B is related to one or more than one entity of A. Whereas A is associated with at most one entity in B.
* **Many-to-Many:**Any number of entities of A is related to any number of entities of B, and vice-versa.

The most commonly asked question in ER diagram is the minimum number of tables required for a given ER diagram. Generally, the following criteria are used:

| **Cardinality** | **Minimum No. of tables** |
| --- | --- |
| 1:1 cardinality with partial participation of both entities | 2 |
| 1:1 cardinality with a total participation of at least 1 entity | 1 |
| 1:n cardinality | 2 |
| m:n cardinality | 3 |

* If the relation is one-to-many or many-to-one then two or more relational tables can be combined.
* If the relation is many-to-many two tables cannot be combined.
* If the relation is one-to-one and there is total participation of one entity then that entity can be combined with a relational table.
* If there is total participation of both entities then one table can be obtained by combining one table and both entities of the relation.

**Note:** This is a general observation. Special cases need to be taken care of. We may need an extra table if the attribute of a relationship can’t be moved to any entity side.

**Specialization:**It is a  top-down approach in which one entity is divided/specialized into two or more sub-entities based on its characteristics.

**Generalization:**It is a bottom-up approach in which common properties of two or more subentities are combined/generalized to form one entity. It is exactly the reverse of Specialization. In this, two or lower level entities are generalized to one higher level entity.

**Aggregation:**Aggregation is an abstraction process through which relationships are represented as higher-level entity sets.

**Participation Constraint:**It specifies the maximum or a minimum number of relationship instances in which any entity can participate. In simple words, participation means how an entity is linked to a relationship.

* **Total Participation:**Each entity of an entity set participates in at least one relationship.
* **Partial Participation:** Some entities of the entity set may not participate in any relationship.

## Database Design

Database design Goals: The prime goal of designing a database is:

* To have zero redundancy in the system
* Loss-less join
* Dependency preservation
* Overcome all the shortcomings of conventional file system

According to [E.F. Codd](https://www.geeksforgeeks.org/introduction-of-relational-model-and-codd-rules-in-dbms/), “All the records of the table must be unique”.

[**Keys of a relation**](https://www.geeksforgeeks.org/dbms-keys-candidate-super-primary-alternate-and-foreign/): There are various types of keys in a relation which are:

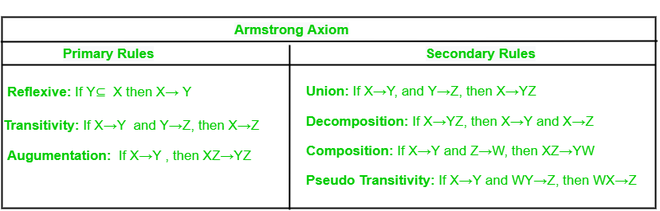
* **Candidate Key:**The minimal set of attributes that can determine a tuple uniquely. There can be more than 1 candidate key of a relation and its proper subset can’t determine tuple uniquely and it can’t be NULL.
* **Super Key:**The set of attributes that can determine a tuple uniquely. A candidate key is always a super key but vice versa is not true.
* **Primary Key and Alternate Key:** Among various candidate keys, one key is taken as the primary key and others are alternate keys.
* **Foreign Key:** Foreign Key is a set of attributes in a table that is used to refer to the primary key or alternative key of the same or another table.

[**Functional Dependency**](https://www.geeksforgeeks.org/types-of-functional-dependencies-in-dbms/)**:**It is a constraint that specifies the association/ relationship between a set of attributes. In functional dependency, one set can accurately determine the value of another set.  It is represented as A𡤫, where set A can determine the values of set B correctly. The A is known as the **Determinant**, and B is known as the **Dependent.**

Functional dependencies are further categorized into two types:

* **Trival Functional Dependency:**In functional dependency, if B is a subset of A, then such dependency is known as trivial functional dependency.
* **Non-Trivial Functional Dependency:**In functional dependency, if B is not a subset of A, then such dependency is known as non-trivial functional dependency.

**Armstrong’s Axioms:**It is a statement that is always considered true and used as a starting point for further arguments. Armstrong axiom is used to generate a closure set in a relational database.



*Armstrong Axiom*

[**Attribute Closure(X+):**](https://www.geeksforgeeks.org/functional-dependency-and-attribute-closure/)All attributes of the set are functionally determined by X.

* **Prime Attribute:** An attribute that is part of one candidate key.
* **Non-prime Attribute:**An attribute that is not a part of any candidate key.

**Example:** If the relation R(ABCD) {A𡤫, B𡤬, C𡤭}, then the attribute closure of

A will be (A+)={ABCD} [A can determine B, B can determine C, C can determine D]  
B will be (B+)={BCD}    [B can determine C, C can determine D]  
C will be (C+)={CD}      [C can determine D]        
D will be (D+)={D}        [D can determine itself]

Note: With the help of Attribute closure, we can easily determine the Superkey [The set of attributes whose closure contains all attributes of a relation] of a relation, So in the above example A is the superkey of the given relation.  There can be more than one superkey in a relationship.

**Example:**If the relation R(ABCDE) {A�, CD𡤮, B𡤭, E𡤪}, then the attribute closure will be

A+= {ABCDE}  
B+= {BD}  
C+= {C}  
D+= {D}  
E+= {ABCDE}

[**Equivalence sets of Functional Dependency**](https://www.geeksforgeeks.org/equivalence-of-functional-dependencies-sets/)**:**If two sets of a functional dependency are equivalent, i.e. if A+= B+. Every FD in A can be inferred from B, and every FD in B can be inferred from A, then A and B are functionally equivalent.

[**Minimal set of Functional Dependency**](https://www.geeksforgeeks.org/canonical-cover-of-functional-dependencies-in-dbms/)**:**A set of FD  will be minimal if it satisfies the following conditions:

* F logically implies all dependencies in F+
* F+ logically implies all dependencies in F.
* No functional dependency in F+ contains an extraneous attribute.
* Each left side of a functional dependency in F+ is unique. That is, there are no two dependencies A𡤫 and C𡤭 in such that A𡤬.

[**Normalization:**](https://www.geeksforgeeks.org/introduction-of-database-normalization/)Normalization is used to eliminate the following [anomalies](https://www.geeksforgeeks.org/anomalies-in-relational-model/):

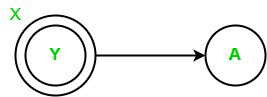
* **Insertion anomaly**
* **Deletion anomaly**
* **Updation Anomaly**
* **Join anomaly**

Normalization was introduced to achieve integrity in the database and make the database more maintainable.

**1. First Normal Form:**A relation is in first normal form if it does not contain any multi-valued or composite attribute.  If the data is in 1NF then it will have high redundancy.

**2. Second Normal Form:**A relation is in the second normal form if it is in the first normal form and if it does not contain any partial dependency.

* **Partial Dependency:**A dependency is called partial dependency if any proper subset of candidate key determines non-prime (which are not part of candidate key) attribute.  
  Let R be the relational schema and X, Y, A is the set of attributes. Suppose X is any candidate key, Y is a proper subset of candidate key, and A is a Non-prime attribute.



*Partial Dependency*

           Y𡤪 will be partial dependency iff, Y is a proper subset of candidate key, and A is a non-prime attribute.

* **Full Functional Dependency:** If A and B are an attribute set of a relation, B is fully functional dependent on A, if B is functionally dependent on A but not on any proper subset of A.

**3. Third Normal Form:**A relation is in the third normal form if it is in the second normal form and it does not contain any transitive dependency. For a relation to be in Third Normal Form, either LHS of FD should be super key or RHS should be the prime attribute.

**4. Boyce-Codd Normal Form:**A relation is inBoyce-CoddNormal Form if the LHS of every FD is super key. The relationship between Normal Forms can be represented as **1NF𢠲NF 𢠳NF �NF**

| **Design Goal** | **1NF** | **2NF** | **3NF** | **BCNF** |
| --- | --- | --- | --- | --- |
| Zero Redundancy | High redundancy | Less than 1NF | Less than 2NF | No redundancy |
| Loss-less decomposition | Always | Always | Always | Always |
| Dependency preservation | Always | Always | Always | Sometimes Not possible |

**Properties of Decomposition:**

* **Loss-less Join Decomposition:**There should not be the generation of any new tuple because of the decomposition.

**i.e. R≡R1 ⨝R2**

* **Dependency Preserving Decomposition:**There should not be the loss of any tuple because of the decomposition. Let R be a relation with Functional dependency F. After decomposition R is decomposed into R1, R2, R3……Rn with FD set F1, F2, F3……Fn respectively.

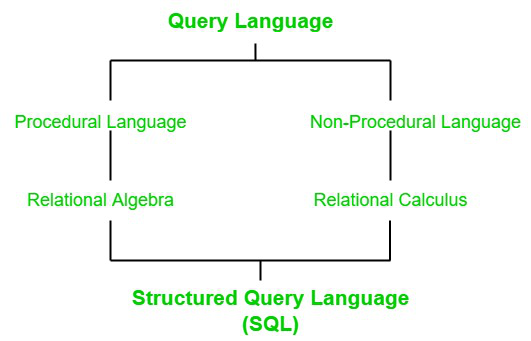
If F1𢊯2𢊯3…..Fn **≡** F, then the decomposition is dependency preserving otherwise not.

## Data Retrieval (SQL, RA)

**Commands to Access Database:**For efficient data retrieval, insertion, deletion, updation, etc. The commands in the Database are categorized into three categories, which are as follows:

* **DDL [Data Definition language]:**It deals with how data should store in the database. DDL commands include CREATE, ALTER, DROP, COMMENT, and TRUNCATE
* **DML [Data Manipulation language]:**It deals with data manipulation like modifying, updating, deleting, etc. DML commands include SELECT, INSERT, DELETE, UPDATE, LOCK TABLE, MERGE, CALL, AND EXPLAIN PLAN.
* **DCL [Data Control Language]:**It acts as an access specifier, and includes GRANT, AND REVOKE.

**Query Language:** Language using which any user can retrieve some data from the database.



**Note:**Relational model is a theoretical framework RDBMS is its implementation.

[**Relational Algebra**](https://www.geeksforgeeks.org/database-management-system-relational-algebra/): Procedural language with basic and extended operators.

| **Basic Operator** | **Semantic** |
| --- | --- |
| σ(Selection) | Select rows based on a given condition |
| ∏(Projection) | Project some columns |
| X (Cross Product/ Cartesian Product) | Cross product of relations, returns **m\*n** rows where m and n are numbers of rows in R1 and R2 respectively. |
| U (Union) | Return those tuples which are either in R1 or R2.  Maximum number of rows returned **= m+n** Minimum number of rows returned = **max(m,n)** |
| −(Minus) | R1-R2 returns those tuples which are in R1 but not in R2.  Maximum number of rows returned = **m** Minimum number of rows returned = **m-n** |
| ρ(Rename) | Renaming a relation to another relation. |

| **Extended Operator** | **Semantic** |
| --- | --- |
| ∩ (Intersection) | Returns those tuples which are in both relation R1 and R2.  Maximum number of rows returned = min(m,n)  Minimum number of rows returned = 0 |
| ⋈c (Conditional Join) | Selection from two or more tables based on some condition (Cross product followed by selection) |
| ⋈(Equi Join) | It is a special case of conditional join when only an equality condition is applied between attributes. |
| ⋈ (Natural Join) | In natural join, equality condition on common attributes holds, and duplicate attributes are removed by default. **Note:** Natural Join is equivalent to the cross product of two relations have no attribute in common and the natural join of a relation R with itself will return R only. |
| ⟕(Left Outer Join) | When applying join on two relations R and S, Left Outer Joins gives all tuples of R in the result set. The tuples of R which do not satisfy the join condition will have values as NULL for attributes of S. |
| ⟖(Right Outer Join) | When applying join on two relations R and S, Right Outer Joins gives all tuples of S in the result set. The tuples of S which do not satisfy the join condition will have values as NULL for attributes of R. |
| ⟗(Full Outer Join) | When applying join on two relations R and S, Full Outer Joins gives all tuples of S and all tuples of R in the result set. The tuples of S which do not satisfy the join condition will have values as NULL for attributes of R and vice versa. |
| /     (Division Operator) | Division operator A/B will return those tuples in A which is associated with every tuple of B. **Note:**Attributes of B should be a proper subset of attributes of A.  The attributes in A/B will be Attributes of A- Attribute of B. |

**Relational Calculus:**Relational calculus is a non-procedural query language. It explains what to do but not how to do it. It is of two types:

* [**Tuple Relational Calculus:**](https://www.geeksforgeeks.org/tuple-relational-calculus-trc-in-dbms/)The tuple relational calculus is based on specifying the number of tuple variables. Each variable usually ranges over a particular database relation. It is of the form   
  **{t| cond(t)}**  
  where**t** is the tuple variable and **cond(t)** is a conditional expression involving t. The result of such query is the set of tuples of t that satisfy cond(t).
* [**Domain Relational Calculus:**](https://www.geeksforgeeks.org/domain-relational-calculus-in-dbms/)Unlike tuple calculus in Domain calculus variables range oer single values from domains of attributes. It is of the form  
  **{x1, x2, ……, xn | cond (x1, x2, ……., xn, xn+1, xn+2, …….., xn+m)}**  
  where**, x1, x2, ……., xn, xn+1, xn+2, …….., xn+m**are domain variables ranging over domains, and**cond**is a condition.

[**SQL**](https://www.geeksforgeeks.org/sql-tutorial/): Structured Query Language, lets you access or modify databases. SQL can execute queries, retrieve data, insert records, update records, delete records, create a new database, create new tables, create views, and set permissions on tables, procedures, or views.

[**SQL Commands:**](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/)

| **Operator** | **Meaning** |
| --- | --- |
| [**SELECT**](https://www.geeksforgeeks.org/sql-select-query/) | Selects columns from a relation or set of relations. It defines WHAT is to be returned.  **Note:**As opposed to Relational Algebra, it may give duplicate tuples for the repeated values of an attribute. |
| [**FROM**](https://www.geeksforgeeks.org/sql-sub-queries-clause/) | **FROM** is used to define the Table(s) or View(s) used by the SELECT or WHERE statements |
| [**WHERE**](https://www.geeksforgeeks.org/sql-where-clause/) | **WHERE**is used to define what records are to be included in the query. It uses conditional operators. |
| [**EXISTS**](https://www.geeksforgeeks.org/sql-exists/) | **EXISTS**is used to check whether the result of a correlated nested query is empty (contains no tuples) or not. |
| [**GROUP BY**](https://www.geeksforgeeks.org/sql-group-by/) | **GROUP BY**is used to group the tuples based on some attribute or set of attributes like counting the number of students GROUP BY the department. |
| [**ORDER BY**](https://www.geeksforgeeks.org/sql-order-by/) | **ORDER BY**is used to sort the fetched data in either ascending or descending according to one or more columns. |
| [**Aggregate functions**](https://www.geeksforgeeks.org/database-management-system-aggregate-functions/) | Find the aggregated value of an attribute. Used mostly with GROUP BY. e.g.; count, sum, min max.  **select count(\*) from the student group by dept\_id** **Note:** we can select only those columns which are part of GROUP BY. |
| [**Nested Queries**](https://www.geeksforgeeks.org/nested-queries-sql/) | When one query is a part of another query. |
| [**UPDATE**](https://www.geeksforgeeks.org/sql-update-statement/) | It is used to update records in a table. |
| [**DELETE**](https://www.geeksforgeeks.org/sql-delete-statement/) | It is used to delete rows in a table. |
| [**LIKE**](https://www.geeksforgeeks.org/sql-like/) | LIKE operator is used with the WHERE clause to search a specified pattern in a column. |
| [**IN**](https://www.geeksforgeeks.org/sql-between-in-operator/) | IN operator is used to specify multiple values in the WHERE clause. |
| [**BETWEEN**](https://www.geeksforgeeks.org/sql-between-in-operator/) | It selects values within a range. |
| [**Aliases**](https://www.geeksforgeeks.org/sql-aliases/) | It is used to temporarily rename a table or a column heading. |
| [**HAVING**](https://www.geeksforgeeks.org/having-clause-in-ms-sql-server/) | The HAVING clause was added because the WHERE keyword could be used with aggregate functions. |

## File Structure

[**File organization:**](https://www.geeksforgeeks.org/file-organization-in-dbms-set-1/)It isthe logical relation between records and it defines how file records are mapped into disk blocks(memory). A database is a collection of files, each file is a collection of records, and each record contains a sequence of fields. **The blocking Factor** is the average number of records per block.

Strategies for storing files of records in block:

* **Spanned Strategy:**It allows a partial part of the record to be stored in the block. It is suitable for variable-length records. No wastage of memory in spanned strategy but block access time gets increases.
* **Unspanned Strategy:** Data cannot be stored partially, the whole block will be occupied, this can lead to internal fragmentation and wastage of memory but block access time is reduced. This is suitable for fixed-length records.

File organizations is of following types:

* [Sequential File Organization](https://www.geeksforgeeks.org/file-organization-in-dbms-set-1/)
* [Heap File Organization](https://www.geeksforgeeks.org/file-organization-in-dbms-set-1/)
* [Hash File Organization](https://www.geeksforgeeks.org/file-organization-in-dbms-set-4/)
* [B+ Tree File Organization](https://www.geeksforgeeks.org/file-organization-in-dbms-set-3/)
* [Clustered File Organization](https://www.geeksforgeeks.org/file-organization-in-dbms-set-3/)

**Sequential File:**In this method, files are stored in sequential order one after another.

* **Blocking factor:**
* **Number of record blocks:**
* **Average number of blocks accessed by linear search:**
* **Average number of blocks accessed by binary search:**

**Index File:**

* **Index blocking factor:**
* **First level index block:**
* **Number of block accesses:**

## [**Indexing Type**](https://www.geeksforgeeks.org/indexing-in-databases-set-1/)**:**

### **1. Single level Index**

* **Primary Index(Sparse):**A primary index is an ordered file(ordered with key field), records of fixed length with two fields. The first field is the same as the primary key of the data file and the second field is a pointer to a data block, where the key is available. In Sparse indexing,  for a set of database records there exists a single entry in the index file.
  + **Number of index file entries≤ Number of database records.**
* **Secondary Index (Dense):** Secondary index provides secondary means of accessing a file for which primary access already exists. In Dense indexing, for every database record, there exists an entry in the index file. The index blocking factor is the same for all indexes.
  + **Number of database records = Number of entries in the index file**
  + **Number of block accesses=**
* **Clustered Index(Sparse):** A clustering index is created on a data file whose records are physically ordered on a non-key field (called a Clustering field).Almost one clustering index is possible.
  + **Single-level index blocks=**
  + **Number of block accesses=**

### **2. Multilevel Index**

* **Indexed sequential access method:**Second level index is always sparse.
  + **Level 1**= “first-level index blocks” computed by index
  + **Level 2** =
  + **Level n**=
  + **Number of blocks** =
  + **Number of block access**= n+1
* **B-Tree:**Also known as Baye’s or balanced Search Tree. At every level, we have Key and Data pointers, and data pointer points either block or record.
  + **Root node:**B-tree can have children between **2** and **p**, where p is the Order of the tree.
  + **Internal Node:** to n children.
  + **Leaf nodes** all are at the same level.
  + **Block size**= p × (size of block pointer) + (p-1)× (Size of key field + size of record pointer)
  + **Minimum number of nodes** =
  + **Maximum number of nodes** = \frac{p^{h+1}-1}{p-1}
  + **Minimum height**= \left ( \left \lceil log\_p \ l \right \rceil \right ) l is the number of leaves
  + **Maximum height** = \left \lfloor 1 +log\_{\frac {p}{2}} \frac{l}{2} \right \rfloor
* **B+ Tree:**It is the same as B-tree. All the records are available at the leaf (last) level. B+ tree allows both sequential and random access whereas in B-tree only random access was allowed. Each leaf node has one block pointer and all the leaf nodes are connected to the next leaf node using a block pointer.
  + **Order of non-leaf node**= [p × size of block pointer] + [(p-1) × size of key field]≤ Block size.
  + **Order of Leaf node**= [(pleaf -1) × (size of key field + size of record pointer) + p × (size of block pointer)𢙋lock size]

## Transaction and Concurrency Control

A transaction is a unit of instruction or set of instructions that performs a logical unit of work. Transaction processes are always atomic in nature either they will execute completely or do not execute.

[**Transaction Properties:**](https://www.geeksforgeeks.org/acid-properties-in-dbms/)

* **Atomicity:** Either execute all operations or none of them. It is managed by the transaction Management Component.
* **Consistency:** Database must be consistent before and after the execution of the transaction. If atomicity, isolation, and durability are implemented accurately, consistency will be achieved automatically.
* **Isolation:** In concurrent transactions, the execution of one transaction must not affect the execution of another transaction. It is managed by the Concurrency Control component.
* **Durability:** After the commit operation, the changes should be durable and persist always in the database. It is managed by the Recovery Management component.

**Schedule:**Sequences in which instructions of the concurrent transactions get executed. Schedules are of two types:

* **Serial Schedule:**Transactions execute one by one, another transaction will begin after the commit of the first transaction. It is inconsistent and the system’s efficiency is so poor due to no concurrency.
  + The number of possible serial schedules with n transactions = **n!**
* **Non-Serial Schedule:**When two or more transactions can execute simultaneously. This may lead to inconsistency, but have better throughput and less response time.
  + The number of possible non-serial schedules with n transactions = Total Schedule – Serial Schedule

**∩ (Intersection)Serializability:** A schedule is said to be serializable if it is equivalent to a serial schedule. It is categorized into two categories: Conflict Seriablizability, and View Serializability.

[**Conflict Serializability:**](https://www.geeksforgeeks.org/conflict-serializability-in-dbms/)A schedule will be conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. It is a polynomial-time problem.

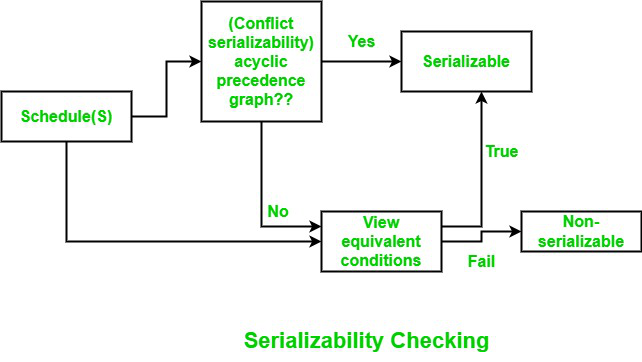
* **Conflicting operations:**Two operations will be conflicting if
  + They belong to different transactions.
  + They are working on the same data item.
  + At least one of them is the Write operation.

[**View Serializability:**](https://www.geeksforgeeks.org/view-serializability-in-dbms/)A schedule will be view serializable if it is view equivalent to a serial schedule. It is an NP-Complete Problem.

* Check whether it is conflict serializable or not, if Yes then it is view serializable.
* If the schedule does not conflict with serializable then check whether it has blind write or not. If it does not have blind write then it is not view serializable. [To be view serializable a schedule must have a blind write]
* If the schedule has blind write, Now check whether the schedule is view-equivalent to any other serial schedule.
* Now, draw a precedence graph using given dependencies. If no cycle/loop exists in the graph, then the schedule would be a View-Serializable otherwise not.

[**Types of Schedule based recoverability:**](https://www.geeksforgeeks.org/types-of-schedules-based-recoverability-in-dbms/)

* **Irrecoverable Schedule:** A transaction is impossible to roll back once the commit operation is done.
* [**Recoverable Schedule:**](https://www.geeksforgeeks.org/recoverability-in-dbms/)A schedule is recoverable if a transaction Ti reads a data item previously written by Transaction Tj, the commit operation  Tj appears before the commit operation of Ti.
* [**Cascadeless Recoverable Schedule**](https://www.geeksforgeeks.org/cascadeless-in-dbms/)**:**Cascadeless Schedule avoids cascading aborts/rollbacks (ACA). Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Avoids that a single transaction abort leads to a series of transaction rollbacks.
  + **Cascading rollback:** When failure of a single transaction leads to a series of transaction rollbacks.
* **Strict Recoverable Schedule:** If there is no read or write in the schedule before the commit, then such schedule are known as a Strict recoverable schedule.



**Concurrency Control with Locks:**

To achieve consistency,  isolation is the most important concept. Isolation can be achieved using locking very easily. A schedule acquires a lock prior to accessing the transaction and the lock is released when the transaction is completed. A locking protocol is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

**Lock Types:**

**Binary Locks**

* It is in two states: Locked(1) or Unlocked(0)
* When an object is locked it is unavailable to other objects.
* When an object is unlocked then it is open to transactions.
* An object is unlocked when the transaction is unlocked.
* Every transaction locks a data item before use and unlocks/releases it after use.
* Issues with binary locks: Irrecoverability, Deadlock, and Low concurrency.

**Shared/Exclusive Locks:**

* **Shared (S Mode):**It is denoted by lock-S(Q), the transaction can perform a read operation, and any other transaction can also obtain the same lock on same data item at the same time and can also perform a read operation only.
* **Exclusive (X Mode):**It is denoted by lock- X(Q), the transaction can perform both read and write operations, any other transaction can not obtain either shared/exclusive lock.

**Two-Phase Locking:**This protocol requires that each transaction in a schedule will be two phases: i.e. Growing phase and the shrinking phase.

* In the growing phase, transactions can only obtain locks but cannot release any lock.
* In the shrinking phase, transactions can only release locks but can not obtain any lock.
* The transaction can perform read/write operations in both the growing as well as in shrinking phase.
* **Rules for 2-PL:**
  + Two transactions cannot have conflicting locks.
  + No unlock operation can precede a lock operation in the same transaction.
  + No data are affected until all locks are obtained.
* **Basic 2-PL:**
  + Equal serial schedule based on lock point.
  + It has recoverability, deadlock, and starvation.
* **Strict 2-PL:**
  + Basic 2-PL with all exclusive locks should be hold until commit/roll back.
  + It ensures serializability strict recoverable, but it has starvation and irrecoverability.
* **Rigorous 2-PL:**
  + Basic 2PL with all locks (S/X) should be hold until commit.
  + Equivalent serial schedule based on the order of commit.

[**Time Stamp Ordering Protocols:**](https://www.geeksforgeeks.org/timestamp-based-concurrency-control/)Time stamp protocols assign a globally unique timestamp value to each transaction and produce the order for transaction submission. The timestamp protocol is free from deadlock, it may suffer from starvation but it ensures serializability.

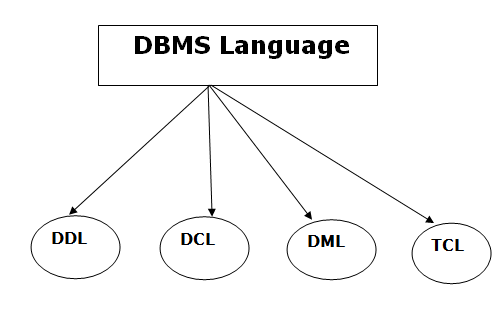
**Thomas Write Rule:** It modify time-stamping protocol in obsolete write case, when Ti requests write (Q) if TS (Ti) < WTS(Q) then roll back T. Otherwise execute successfully. Set RTS (A)= max{TS(T), RTS (A)}.

Thomas write rule is View serializable, hence it guarantees consistency, but there is no guarantee of cascadeness and recoverable system because there is no restriction on write.

# Database Languages in DBMS

* A DBMS has appropriate languages and interfaces to express database queries and updates.
* Database languages can be used to read, store and update the data in the database.

## **Types of Database Languages**



## **1. Data Definition Language (DDL)**

* **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
* It is used to create schema, tables, indexes, constraints, etc. in the database.
* Using the DDL statements, you can create the skeleton of the database.
* Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

* **Create:** It is used to create objects in the database.
* **Alter:** It is used to alter the structure of the database.
* **Drop:** It is used to delete objects from the database.
* **Truncate:** It is used to remove all records from a table.
* **Rename:** It is used to rename an object.
* **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

## **2. Data Manipulation Language (DML)**

**DML** stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

Backward Skip 10sPlay VideoForward Skip 10s

* **Select:** It is used to retrieve data from a database.
* **Insert:** It is used to insert data into a table.
* **Update:** It is used to update existing data within a table.
* **Delete:** It is used to delete all records from a table.
* **Merge:** It performs UPSERT operation, i.e., insert or update operations.
* **Call:** It is used to call a structured query language or a Java subprogram.
* **Explain Plan:** It has the parameter of explaining data.
* **Lock Table:** It controls concurrency.

## **3. Data Control Language (DCL)**

* **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
* The DCL execution is transactional. It also has rollback parameters.

(But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

* **Grant:** It is used to give user access privileges to a database.
* **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

## **4. Transaction Control Language (TCL)**

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

* **Commit:** It is used to save the transaction on the database.
* **Rollback:** It is used to restore the database to original since the last Commit.

**See Last Minute Notes on all subjects**[**here**](https://www.geeksforgeeks.org/lmns-gq/)**.**

# *****Python*****

***Road Map***

## ***Print Output***

In Python, there are several ways to produce output. Below are some common methods to display output to the console or other output streams:

1. \*\*print() Function\*\*: This is the most common way to produce output.

print("Hello, World!")

You can format output using string concatenation or string formatting:

name = "John"

print("Hello, " + name + "!") # Concatenation

2. \*\*Formatted String Literals (f-strings)\*\*: Available in Python 3.6 and newer, you can embed expressions inside string literals using `{}`.

name = "John"

print(f"Hello, {name}!")

3. \*\*str.format()\*\*: Allows you to format strings by placing `{}` where you want to insert values.

name = "John"

print("Hello, {}!".format(name))

For multiple variables:

first\_name = "John"

last\_name = "Doe"

print("Hello, {} {}!".format(first\_name, last\_name))

4. \*\*Using the `%` Operator\*\*: This is an older method of formatting strings in Python.

name = "John"

print("Hello, %s!" % name)

For multiple variables:

first\_name = "John"

last\_name = "Doe"

print("Hello, %s %s!" % (first\_name, last\_name))

5. \*\*Write to `sys.stdout`\*\*: The `print()` function writes to the standard output stream (`sys.stdout` by default). You can directly write to this stream:

import sys

sys.stdout.write("Hello, World!\n")

6. \*\*Using a Third-party Library\*\*: Libraries like `logging` can also produce output, though they are more commonly used for logging purposes:

import logging

logging.basicConfig(level=logging.INFO)

logging.info("Hello, World!")

7. \*\*Display in Interactive Environments\*\*: In interactive environments like Jupyter notebooks or IPython, just typing the variable or expression at the end of a cell will display its value.

message = "Hello, World!"

message

These are the most common ways to print output in Python. Depending on the context (e.g., writing to files, web development, GUI applications), there may be other ways to produce or display output. The `print()` function and formatted string methods are generally the most commonly used for console output.

## ***OOPS***

<https://python-programming.quantecon.org/intro.html>

### ***What is OOP?***

Object-oriented Programming (OOPs) is a programming pattern that based on objects, which contain data in the form of fields and classes in programming that focus on objects and data rather than actions and logic.

### ***Why Python is not a pure OOP language?***

Python is often referred to as an "object-oriented programming language." However, it's not considered a "pure" object-oriented language for several reasons:

1. **Primitives Are Not Objects:** Almost everything in programming languages like Java is an object (with a few exceptions like primitives, however wrapper classes exist). While many of the entities in Python are objects, some things like integers and floats are not necessarily objects. However, one can argue against this point because in Python, functions like 'id()' and 'type()' can be used with primitives, referring to that they are really objects.

2. **Multiple Paradigms**: Python is adaptable and supports a number of different programming approaches. In addition to object-oriented programming, Python supports imperative, functional, and procedural programming. A purely object-oriented language would not provide such flexibility.

3. **Lack of Access Specifiers:** Access modifiers like "private," "protected," and "public" are explicitly available in object-oriented languages. These keywords limit the visibility of attributes and methods outside of the class or package. Python does not have such strict encapsulation. Instead, it relies on conventions, such as prefixing an attribute with ‘**\_’** for "protected" or ‘**\_ \_’** for "private".

4. **Static Methods & Class Methods**: In Python, you can define static methods using the **@staticmethod** decorator and class methods using the **@classmethod** decorator. Static methods, in particular, don't need a class instance, and they don't change the class's state, which deviates from the pure OOP approach.

5. **Global Functions and Variables**: Python allows you to define functions and variables outside classes. In a pure object-oriented language, functions (methods) and attributes would always be encapsulated within classes.

6. **Functional Programming Features**: Python includes features that are borrowed from functional programming languages, such as **lambda**, **map()**, **filter()**, and **reduce()**. While these tools are convenient, they don't align with the strict OOP paradigm.

### ***Why OOP?***

1. **Modeling the Real World**: OOP is inherently designed to mirror how entities interact in the real world. Objects in OOP can represent tangible real-world entities (like cars, books, or users) or more abstract concepts (like transactions, processes, or policies). This makes the design more intuitive and easier to understand.

2. **Encapsulation**: OOP allows data and the methods that operate on it to be bundled together as a single unit, an object. This encapsulation ensures that an object's internal state is shielded from outside interference and misuse. Encapsulation simplifies interface design and protects the integrity of the data.

3. **Modularity and Structure**: OOP provides a clear modular structure for programs. Each object is a self-contained module with its attributes and operations. Such modularity facilitates more organized, manageable, and modular system design.

4.**Code Reusability**: Through inheritance, a "child" class can inherit attributes and methods from a "parent" class. This means common functionality doesn't need to be repeated across multiple classes, leading to less redundant code and easier maintenance.

5.**Flexibility through Polymorphism**: Polymorphism allows objects of different classes to be treated as instances of the same class. This makes systems more flexible and scalable, as new objects can be introduced without disrupting existing interfaces or code.

6. **Maintainability**: The modular nature of OOP means that changes in one part of the application are less likely to impact other parts. This makes maintenance and adding enhancements easier and less error prone.

7. **Clearer and More Flexible Design Patterns**: Many modern design patterns, like MVC (Model-View-Controller), are inherently object-oriented. OOP provides the means to implement these patterns in a way that's consistent with their design principles, leading to clearer and more maintainable system architectures.

8. **Problem Decomposition**: OOP provides a framework to break down complex problems into more manageable pieces. By treating entities as objects, developers can focus on individual aspects of a problem one at a time, making complex system development more tractable.

9. **Enhanced Productivity**: OOP's emphasis on modularity, reusability, and maintainability can lead to increased productivity over the software's lifecycle, as code can be reused, extended, and maintained more easily.

10. **Collaboration**: OOP makes it easier for teams to work collaboratively. Different team members can work on different objects or modules without stepping on each other's toes. The modular nature ensures clear boundaries and interfaces between components.

### ***Principles***

* **DRY (Don't Repeat Yourself):** This principle is about reducing the repetition of code. Inheritance is one way OOP achieves this: instead of repeating the same code in multiple classes, common code can be placed in a superclass and inherited.
* **Modularity**: Systems are built using well-defined, simple, and interconnected modules or classes. This makes systems easier to develop, debug, and maintain.
* **Information Hiding**: Only necessary information is exposed to the user, while the inner workings are hidden for simplicity and security.

### ***Basic Concepts of OOP***

1. Class

A class is a blueprint for creating objects (specific data structures). It defines properties (attributes) that the object will have and methods that can manipulate those attributes.

1. Object

An instance of a class. If a class is like a blueprint, an object is what's built from that blueprint.

1. Method

A function defined within a class which typically performs operations on class attributes.

1. Inheritance

A way to create a new class using properties and behaviors (methods) of an existing class. This promotes code reuse.

1. Encapsulation

Restricting access to certain parts of an object, ensuring that the internal representation of the object is hidden from outside.

1. Polymorphism

Allows objects of different classes to be treated as objects of a common superclass.

### ***Object***

An *object* is a collection of data and instructions held in computer memory that consists of

1. a type
2. a unique identity
3. data (i.e., content)
4. methods
5. Type

Python provides for different types of objects, to accommodate different categories of data. For example,

s = 'This is a string'

type(s)

str

x = 42 *# Now let's create an integer*

type(x)

int

The type of an object matters for many expressions.

1. Identity

In Python, each object has a unique identifier, which helps Python (and us) keep track of the object.

The identity of an object can be obtained via the id() function

y = 2.5

z = 2.5

id(y)

140026281779056

id(z)

140026281776944

In this example, y and z happen to have the same value (i.e., 2.5), but they are not the same object.

The identity of an object is in fact just the address of the object in memory.

1. Data

If we set x = 42 then we create an object of type int that contains the data 42. In fact, it contains more, as the following example shows

x = 42

x

42

x.imag

0

x.\_\_class\_\_

int

When Python creates this integer object, it stores with it various auxiliary information, such as the imaginary part, and the type.

Any name following a dot is called an attribute of the object to the left of the dot.

* e.g.,imag and \_\_class\_\_ are attributes of x.

1. Methods

Methods are functions that are bundled with objects.

Formally, methods are attributes of objects that are callable (i.e., can be called as functions)

x = ['foo', 'bar']

callable(x.append)

True

callable(x.\_\_doc\_\_)

False

Methods typically act on the data contained in the object they belong to, or combine that data with other data

x = ['a', 'b']

x.append('c')

s = 'This is a string'

s.upper()

'THIS IS A STRING'

s.lower()

'this is a string'

s.replace('This', 'That')

'That is a string'

A great deal of Python functionality is organized around method calls.

For example, consider the following piece of code

x = ['a', 'b']

x[0] = 'aa' *# Item assignment using square bracket notation*

x

['aa', 'b']

It doesn’t look like there are any methods used here, but in fact the square bracket assignment notation is just a convenient interface to a method call.

What actually happens is that Python calls the \_\_setitem\_\_ method, as follows

x = ['a', 'b']

x.\_\_setitem\_\_(0, 'aa') *# Equivalent to x[0] = 'aa'*

x

['aa', 'b']

(If you wanted to you could modify the \_\_setitem\_\_ method, so that square bracket assignment does something totally different)

### ***Class***

In Object-Oriented Programming (OOP), classes serve as blueprints to create objects. Various types of classes can be distinguished based on their roles, functionality, and the design patterns they fit into. Here are some common types of classes encountered in OOP:

1. \*\*Concrete Class\*\*:

- A standard class that can be instantiated.

- It provides an implementation for all its methods and is not marked as abstract.

2. \*\*Base (or Parent or Super) Class\*\*:

- A class that is extended by one or more derived classes.

- The derived class inherits attributes and behaviors (methods) from the base class.

3. \*\*Derived (or Child or Sub) Class\*\*:

- A class that inherits properties and methods from a base class.

- It can also have additional properties and methods or override inherited ones.

4. \*\*Abstract Class\*\*:

- Cannot be instantiated on its own.

- It provides a base for other classes and may contain one or more abstract methods (methods without a body).

- Derived classes inheriting from an abstract class must provide an implementation for all its abstract methods.

5. \*\*Singleton Class\*\*:

- A class designed to ensure that only one instance of the class exists in the system.

- It provides a global point of access to that instance.

6. \*\*Immutable Class\*\*:

- Once an object of an immutable class is created, its state cannot be changed.

- Python's `str` and `tuple` are examples of immutable classes.

7. \*\*Mixin Class\*\*:

- A class that provides methods to other classes but isn't considered a base class of those other classes.

- Mixins are used to "mix in" additional properties or methods into a class.

8. \*\*Utility (or Helper) Class\*\*:

- A class that defines a set of methods that perform common, often re-used functions.

- Usually, these are static methods and you don't need to create an instance of the utility class.

9. \*\*Adapter Class\*\*:

- A class that bridges the gap between two interfaces that are incompatible but logically related.

- It allows classes with incompatible interfaces to work together.

10. \*\*Proxy Class\*\*:

- A class that functions as an interface to another class or resource, such as a network connection or a large object in memory.

- Useful for access control, lazy initialization, and more.

11. \*\*Factory Class\*\*:

- A class that's responsible for creating objects of other types.

- Rather than calling a class's constructor directly, a factory method is called to create the object.

12. \*\*Wrapper (or Decorator) Class\*\*:

- A class that "wraps" another, providing additional functionality or modifying existing behaviors of the wrapped class.

Different programming languages might have shades in their implementations of OOP, but these types of classes are common across many OOP languages.

In Python, the data and methods of an object are collectively referred to as attributes.

Attributes are accessed via “dotted attribute notation.”

* object\_name.data
* object\_name.method\_name()

In the example

x = [1, 5, 4]

x.sort()

x.\_\_class\_\_

list

* x is an object or instance, created from the definition for Python lists, but with its own particular data.
* x.sort() and x.\_\_class\_\_ are two attributes of x.
* dir(x) can be used to view all the attributes of x.

***Constructor***

In Python, a constructor is a special method used to initialize an object when it's created. The constructor method is named `\_\_init\_\_`.

When you create an object of a class, the constructor (`\_\_init\_\_` method) is automatically called, allowing you to set initial values for attributes or perform any other setup action.

**Basic Syntax**

*class* ClassName:

*def* \_\_init\_\_(self, [parameters]):

*# initialization code*

### \*\*Examples\*\*:

1. **A simple constructor:**

*class* Dog:

*def* \_\_init\_\_(self):

*self*.breed = "Unknown"

dog = Dog()

print(dog.breed)  *# Output: Unknown*

2. **Constructor with parameters**:

*class* Dog:

*def* \_\_init\_\_(self, breed):

*self*.breed = breed

dog = Dog("Bulldog")

print(dog.breed)  *# Output: Bulldog*

3. **Constructor with default parameters**:

*class* Dog:

*def* \_\_init\_\_(self, breed="Unknown"):

*self*.breed = breed

dog1 = Dog()

dog2 = Dog("Golden Retriever")

print(dog1.breed)  *# Output: Unknown*

print(dog2.breed)  *# Output: Golden Retriever*

**Points to Remember:**

1. **Single Constructor per Class**: Unlike some other languages, Python allows only one `\_\_init\_\_` method per class. However, you can use default arguments or `\*args` and `\*\*kwargs` to allow for a variable number of arguments.

2. **Explicitly Calling a Parent Class's Constructor**: If a class inherits from another class, it might need to call the parent class's constructor. You can use the `super()` function to do this:

*class* Animal:

*def* \_\_init\_\_(self, species):

*self*.species = species

*class* Dog(Animal):

*def* \_\_init\_\_(self, breed):

        super().\_\_init\_\_("Dog")

*self*.breed = breed

dog = Dog("Labrador")

print(dog.species)  *# Output: Dog*

print(dog.breed)    *# Output: Labrador*

3. **Constructors Don't Return Values**: The primary responsibility of a constructor is to set up the initial state of the object. While technically they can return a value, in practice, they shouldn't, and if you try to return a value other than `None`, Python will raise an error.

In essence, the constructor (`\_\_init\_\_` method) in Python is a powerful tool for setting up new objects correctly right from their creation.

***Self KeyWord:***

In Python, `self` is a convention used for the first parameter of instance methods in a class. This parameter refers to the instance of the class on which the method was called. It's similar to the `this` keyword in many other programming languages.

### 1. \*\*Reference to the Instance\*\*

When you create an instance of a class and call one of its methods, Python automatically passes the instance as the first argument to the method. The convention is to call this argument `self`.

### 2. \*\*Using `self` in a Class\*\*

When defining methods in a class, you'll use `self` to refer to instance attributes or other instance methods.

Example:

```python

class Dog:

def \_\_init\_\_(self, name):

self.name = name # 'self.name' refers to the instance attribute, 'name' is the parameter

def bark(self):

return f"{self.name} says Woof!"

```

When you create an instance of the `Dog` class and call its methods:

```python

dog = Dog("Buddy")

print(dog.bark()) # Output: Buddy says Woof!

```

In the call `dog.bark()`, the `dog` instance is automatically passed to the `bark` method as the `self` parameter.

### 3. \*\*Why Do We Need `self`?\*\*

The need for `self` arises from the way Python handles class and instance attributes. Without `self`, you wouldn't be able to differentiate between instance variables and local variables inside methods.

### 4. \*\*Is `self` a Reserved Keyword?\*\*

No, `self` is not a reserved keyword in Python. It's merely a convention. However, it's strongly recommended to use `self` because it's universally recognized by Python developers.

You could technically use another name in place of `self`, but it would be against convention and might confuse other developers:

```python

class Dog:

def \_\_init\_\_(s, name): # using 's' instead of 'self'

s.name = name

def bark(s):

return f"{s.name} says Woof!"

```

This would work, but again, using anything other than `self` is generally discouraged for the sake of clarity and consistency in the code.

### 5. \*\*How Does `self` Differ from Other OOP Languages?\*\*

In many object-oriented programming languages (like Java or C++), the reference to the instance is implicit and is usually referred to with a keyword like `this`. In Python, this reference is explicit, and by convention, it's named `self`.

# *****DSA*****

## 

<https://www.programiz.com/dsa>

https://github.com/manishbisht/Competitive-Programming/tree/master/Algorithms/Array

## ***Algorithm Analysis***

## ***Recursion***

## ***Array Based Sequences***

## ***Stacks, Queues and Deques***

## ***Linked Lists***

## ***Trees***

## ***Priority Queues***

## ***Maps, Hash Tables and Skip Lists***

## ***Search***

## ***Sorting and Selection***

## ***Text Processing***

## ***Graph Algorithms***

## ***Memory Management and Binary Trees***

**Aghsdnhg**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**Sd**

**D**

**Dg**

**G**

**G**

**G**

**G**

**G**

**Gg**

**G**

# *****Linear Algebra*****

**Linear algebra** is the branch of mathematics that deals with vector spaces.

## **Scalar**

1. Scalars are One dimensional, primitive datatype and Single Numbers.
2. They are Real or Complex
3. Scalar variables --> a, b, c....., x, y, z
4. For ML problems one has to use real values.
5. 2/7 is a scalar value (2/7 --> 0.285714).
6. 30 is a Scalar value.

**Code:**

# Scalar

marks = 99

print("The marks achieved are :",marks)

# **Output:**

The marks achieved are :99

## **Vector**

vectors are objects that can be added together to form new vectors and that can be multiplied by scalars. vectors are points in some finite-dimensional space.

* 1. **Ordered list of numbers or primitive data types.**
  2. **It is an ordered collection of data.**
  3. **One can perform Addition, Subtraction, Scalar multiplication, Transpose, Inverse and measure how two vectors align.**
  4. Vectors add component wise. Means if two vectors v and w are the same length, their sum is the vector whose first element is v[0] + w[0], whose second element is v[1] + w[1], and so on.

If they’re not have the same length, then they are not allowed to add, subtract.

**Code:**

# Vector Sum

# 1st Vector is "v1"

# 2nd Vector is "v2"

# 3rd Vector is "v3"

from numpy import vectorize

def add(v1, v2) -> vectorize:

    """Adds corresponding elements"""

    assert len(v1) == len(v2), "vectors must be the same length"

    return [v1\_i + v2\_i for v1\_i, v2\_i in zip(v1, v2)]

v3 = [9, 7, 5]

assert add([3, 2, 1], [6, 5, 4]) == v3

print(f"The Output is correct, and it is {v3}")

assert add([2, 1], [6, 5, 4]) == v3

# **Output:**

The Output is correct, and it is [9, 7, 5]

**----------------------------------------------------------------------**

**AssertionError** Traceback (most recent call last)

**LinearAlgebra.ipynb**

print(f"The Output is correct and it is {v3}")

v4 = [10, 7, 5]

**--->**  assert add([2, 1], [6, 5, 4]) == v3

**LinearAlgebra.ipynb**

def add(v1, v2) -> vectorize:

"""Adds corresponding elements"""

**---->** assert len(v1) == len(v2), "vectors must be the same length"

return [v1\_i + v2\_i for v1\_i, v2\_i in zip(v1, v2)]

**AssertionError**: vectors must be the same length

**----------------------------------------------------------------------**

**In above code vector v1 = [2,1] has m=1 and n=2 but vector v2 =[6,5,4] has m=1, n=3. Because of n value of both vectors is not same. So, the addition not happened and AssertionError occurs.**

**Code:**

# Vectors : List[vectors]

# Vector

def vector\_sum(vectors):

    """Sums all corresponding elements"""

    # Check that vectors is not empty

    assert vectors, "no vectors provided!"

    # Check the vectors are all the same size

    num\_elements = len(vectors[0])

    assert all(len(v) == num\_elements for v in vectors), "different sizes!"

    # the i-th element of the result is the sum of every vector[i]

    return [sum(vector[i] for vector in vectors)

        for i in range(num\_elements)]

vector = [15, 21]

if vector\_sum([[1, 3], [2, 4], [5, 8], [7, 6]]) == vector:

    print(f"The Vector output is {vector}")

# **Output:**

The Vector output is [15, 21]

* 1. **The result of dot product of two vectors --> Scalar.**
  2. **Vector can never have negative magnitude.**
  3. **Number of Scalars in a vector --> Dimension**
  4. **Vector contains two-dimensional array or list.**
  5. **A vector V has dimension of (m, n)**

**Where,**

**" m = number of rows ",**

**" n = number of columns ”**

* 1. **Transpose of V have (n, m) dimension or shape.**

**Code:**

# Transpose of V have (n,m) dimension or shape.

trans = np.array([[4,1,2],[-2,4,8],[6,-3,-5]])

print(" The Column Vector is \n", np.transpose(trans))

# **Output:**

The Column Vector is

[[ 4 -2 6]

[ 1 4 -3]

[ 2 8 -5]]

* 1. **Column vector --> m rows and 1 column --> shape is (m,1)**
  2. **Row vector --> 1 row and n columns --> shape is (1,n)**
  3. **Zero Vector --> A vector which contains each element as Zero --> z = [0,0,0]**
  4. **One Vector --> A vector which contains each element as One --> o = [1,1,1]**
  5. **The Magnitude of vector is |V| = $\\sqrt{(V1)^2 +(V2)^2 +(V3)^2+ ...... + (Vn)^2}$"**

## **Scalar**

**G**

# *****Calculus*****

**Aghsdnhg**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**Sd**

**D**

**Dg**

**G**

**G**

**G**

**G**

**G**

**Gg**

**Gf**

# *****Statistics*****

Road Map

1. Descriptive type (part) 🡪 Summarizing, Analyzing, Visualizing
   1. Measure of Central Tendency
      1. Mean
      2. Median
      3. Mode
   2. Measure of Dispersion
      1. Range
      2. IQR
      3. Standard Deviation
      4. Variance
   3. Outliers
      1. Box Plot
      2. Histogram, etc.
   4. Distribution
      1. Normal /Gaussian Distribution Theorem
      2. Central Limit Theorem (Real time Examples also)
      3. Skewed/ Non skewed distribution.
      4. Poisson distributions
      5. Bernoulli/Binomial distributions
      6. Other Distributions
2. Inferential type
   1. Hypothesis Testing
      1. Null
      2. Alternative
      3. Null vs Alternative
   2. Errors
      1. Type 1 (Alpha)
      2. Type 2 (Beta)
   3. P-value
   4. Confidence Interval
3. Correlation Coefficient
4. Covariation
5. Z-Score (Standard normal Distribution)
6. Z-test
7. T-test
8. Anova test
9. Log normal Distribution.
10. Univariant, bivariant , multi variant

**What is Statistics?**

**Statistics is the science of collecting, Organizing, and analyzing data. By doing this we can take *better decision making*.**

**Gg**

# *****Probability*****

**Aghsdnhg**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**Sd**

**D**

**Dg**

**G**

**G**

**G**

**G**

**G**

**Gg**

**G**

# *****Data Manipulation and Analysis*****

## ***Numpy***

## ***Pandas***

## ***Matplotlib***

## ***Seaborn***

## ***Scrapy or Beautiful Soup***

## ***Scikit-learn***

## ***TensorFlow (from Google)***

## ***Pytorch (from Facebook)***

## ***Spacy (NLP)/NLTK***

## ***OpenCV (Image Processing)***

**Aghsdnhg**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**Sd**

**D**

**Dg**

**G**

**G**

**G**

**G**

**G**

**Gg**

**G**

# *****Machine Learning*****

**Aghsdnhg**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**Sd**

**D**

**Dg**

**G**

**G**

**G**

**G**

**G**

**Gg**

**G**

# *****Deep Learning*****

**Aghsdnhg**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**Sd**

**D**

**Dg**

**G**

**G**

**G**

**G**

**G**

**Gg**

**G**

# *****Natural Language Processing (NLP)*****

**Aghsdnhg**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**A**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**S**

**Sd**

**D**

**Dg**

**G**

**G**

**G**

**G**

**G**

# *****AI*****

Road Map

1. Descriptive type (part) 🡪 Summarizing, Analyzing, Visualizing
   1. Measure of Central Tendency
      1. Mean
      2. Median
      3. Mode
   2. Measure of Dispersion
      1. Range
      2. IQR
      3. Standard Deviation
      4. Variance
   3. Outliers
      1. Box Plot
      2. Histogram, etc.
   4. Distribution
      1. Normal /Gaussian Distribution Theorem
      2. Central Limit Theorem (Real time Examples also)
      3. Skewed/ Non skewed distribution.
      4. Poisson distributions
      5. Bernoulli/Binomial distributions
      6. Other Distributions
2. Inferential type
   1. Hypothesis Testing
      1. Null
      2. Alternative
      3. Null vs Alternative
   2. Errors
      1. Type 1 (Alpha)
      2. Type 2 (Beta)
   3. P-value
   4. Confidence Interval
3. Correlation Coefficient
4. Covariation
5. Z-Score (Standard normal Distribution)
6. Z-test
7. T-test
8. Anova test
9. Log normal Distribution.
10. Univariant, bivariant , multi variant

a

a

a

a

a

a

a

# *****Data*****

# *Machine Learning*

***Road Map***

## ***OOPS***

<https://python-programming.quantecon.org/intro.html>

### ***What is OOP?***

Object-oriented Programming (OOPs) is a programming pattern that based on objects, which contain data in the form of fields and classes in programming.

### ***Why OOP?***

### ***When OOP?***

### ***How OOP?***

### ***Where OOP?***

### ***Goals, Principles, Design Pattern***

# *Study Plan*

8-week study plan for learning data science using Python. This plan assumes you have some basic knowledge of Python. If you don't, you might need extra time, especially in the first week.

### **Week 1: Python Basics and Advanced Features**

* **Day 1-3**: Review Python basics – variables, data types, loops, and functions.
* **Day 4-5**: Dive into advanced Python – list comprehensions, lambda functions, map, reduce, and filter.
* **Day 6-7**: Hands-on exercises and mini-projects using Python.

### **Week 2: Python Libraries for Data Science and Intro to Data Analysis**

* **Day 1-3**: Introduction to NumPy – arrays, array operations, and basic functions.
* **Day 4-5**: Introduction to Pandas – dataframes, series, basic operations, and data manipulation.
* **Day 6-7**: Hands-on exercises with NumPy and Pandas.

### **Week 3: Data Visualization**

* **Day 1-3**: Introduction to Matplotlib – basic plots, customizing plots, and subplots.
* **Day 4-5**: Introduction to Seaborn – data visualization with statistical plotting.
* **Day 6-7**: Visualization projects – create a variety of plots from a given dataset.

### **Week 4: Exploratory Data Analysis (EDA)**

* **Day 1-3**: Basics of EDA – understanding distributions, mean, median, mode, variance, etc.
* **Day 4-5**: Advanced EDA – correlation, regression, and hypothesis testing.
* **Day 6-7**: Hands-on EDA project using a real-world dataset.

### **Week 5: Machine Learning with Scikit-learn**

* **Day 1-2**: Introduction to machine learning and Scikit-learn.
* **Day 3-4**: Supervised learning basics – regression and classification.
* **Day 5-7**: Practical exercises on regression (like Linear Regression) and classification (like Decision Trees).

### **Week 6: Advanced Machine Learning Concepts**

* **Day 1-2**: Introduction to unsupervised learning – clustering (KMeans, Hierarchical clustering).
* **Day 3-4**: Ensemble methods – Random Forest, Gradient Boosting, etc.
* **Day 5-7**: Real-world machine learning project using supervised and/or unsupervised learning.

### **Week 7: Intro to Deep Learning and Neural Networks with TensorFlow/Keras**

* **Day 1-3**: Introduction to neural networks – perceptrons, activation functions, and backpropagation.
* **Day 4-5**: Basics of TensorFlow/Keras for building simple neural networks.
* **Day 6-7**: Hands-on exercises on creating basic neural networks.

### **Week 8: Capstone Projects and Further Resources**

* **Day 1-5**: Work on a capstone project that encapsulates everything you've learned. This could be a comprehensive data analysis, visualization, and machine learning project on a dataset of your choice.
* **Day 6**: Review what you've learned, and understand areas of strengths and weaknesses.
* **Day 7**: Explore further resources – advanced topics, other libraries/tools, MOOCs, etc.