WM-ASDS04: Introduction to Data Science with Python[1]

Dr. Md. Rezaul Karim
PhD(KU Leuven & UHasselt), MS(Biostatistics, UHasselt), MS(Statistics, JU)
Professor, Department of Statistics and Data Science
Jahangirnagar University (JU), Savar, Dhaka - 1342, Bangladesh
Mobile: 01912605556, Email: rezaul@juniv.edu

Summer - 2024

[1] These course slides should not be reproduced nor used by others (without permission).

Lecture Outline I

Lecture Outline II

Lecture Outline III

Lecture Outline IV

Lecture Outline V

Lecture Outline VI

Chapter 1: Introduction to Data Science

1 Chapter 1: Introduction to Data Science

Text and Reference Book List

Text Book

1. Laura Lgual, L. and Seguí, S. (2017): *Introduction to Data Science A Python Approach to Concepts Techniques and Applications*, Springer, Switzerland.

Reference list

1. Lutz, M. (2013). *Learning Python: Powerful Object-Oriented Programming*. 5th Edition, O'Reilly Media, Inc.
2. Lecture Slides

Course Assessments

Our department suggested implementing the following strategies in course assessments:

- Two (2) Quizzes and two (2) assignments, one midterm, and a final exam will be the components of Assessments.
- Attendance 5%, Scientific Report (Assignment) and homework 10%, Quiz (and Class Performance) 15%, Midterm 30%, Final 40% - Total 100%
- Midterm will be taken after the 6th lecture (in cases it can be extended upto 8th lecture) course contents are covered in classes.
- An average of two quizzes and an average of two assignments will be considered.

Problem & Motivation

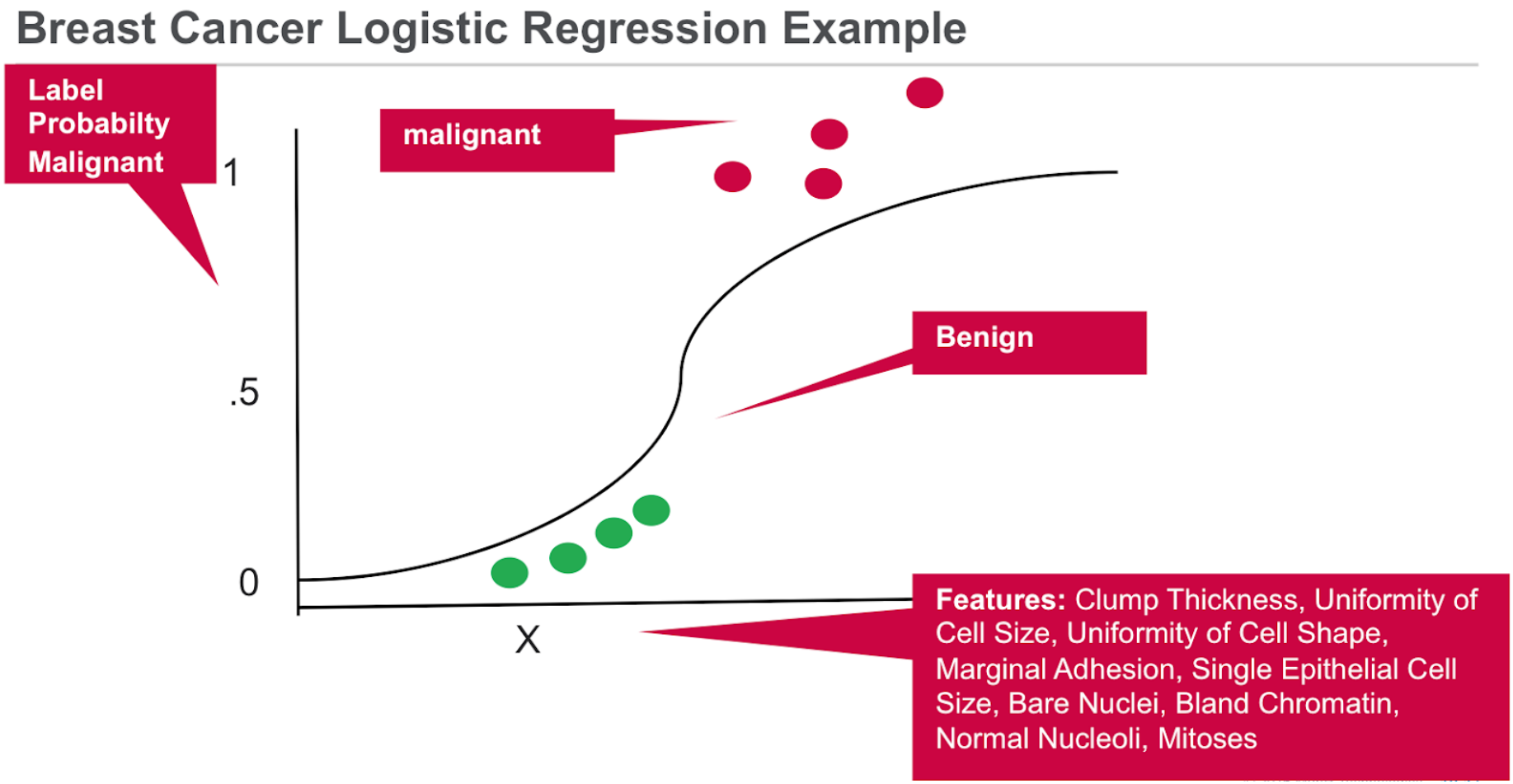
Can you understand this dataset? What insights can you obtain?

id	diagnosis	radius_me	texture_n	perimeter	area_mea	smoothne	compactn	concavity_	concave p	symmetry	fractal_dir	radius_se	texture_s	perimeter
842302	M	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871	1.095	0.9053	8.589
842517	M	20.57	17.77	132.9	1326	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398
8.4E+07	M	19.69	21.25	130	1203	0.1096	0.1599	0.1974	0.1279	0.2069	0.05999	0.7456	0.7869	4.585
8.4E+07	M	11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.2597	0.09744	0.4956	1.156	3.445
8.4E+07	M	20.29	14.34	135.1	1297	0.1003	0.1328	0.198	0.1043	0.1809	0.05883	0.7572	0.7813	5.438
843786	M	12.45	15.7	82.57	477.1	0.1278	0.17	0.1578	0.08089	0.2087	0.07613	0.3345	0.8902	2.217
844359	M	18.25	19.98	119.6	1040	0.09463	0.109	0.1127	0.074	0.1794	0.05742	0.4467	0.7732	3.18
8.4E+07	M	13.71	20.83	90.2	577.9	0.1189	0.1645	0.09366	0.05985	0.2196	0.07451	0.5835	1.377	3.856
844981	M	13	21.82	87.5	519.8	0.1273	0.1932	0.1859	0.09353	0.235	0.07389	0.3063	1.002	2.406
8.5E+07	M	12.46	24.04	83.97	475.9	0.1186	0.2396	0.2273	0.08543	0.203	0.08243	0.2976	1.599	2.039
845636	M	16.02	23.24	102.7	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697	0.3795	1.187	2.466
8.5E+07	M	15.78	17.89	103.6	781	0.0971	0.1292	0.09954	0.06606	0.1842	0.06082	0.5058	0.9849	3.564
846226	M	19.17	24.8	132.4	1123	0.0974	0.2458	0.2065	0.1118	0.2397	0.078	0.9555	3.568	11.07
846381	M	15.85	23.95	103.7	782.7	0.08401	0.1002	0.09938	0.05364	0.1847	0.05338	0.4033	1.078	2.903
8.5E+07	M	13.73	22.61	93.6	578.3	0.1131	0.2293	0.2128	0.08025	0.2069	0.07682	0.2121	1.169	2.061
8.5E+07	M	14.54	27.54	96.73	658.8	0.1139	0.1595	0.1639	0.07364	0.2303	0.07077	0.37	1.033	2.879
848406	M	14.68	20.13	94.74	684.5	0.09867	0.072	0.07395	0.05259	0.1586	0.05922	0.4727	1.24	3.195
8.5E+07	M	16.13	20.68	108.1	798.8	0.117	0.2022	0.1722	0.1028	0.2164	0.07356	0.5692	1.073	3.854
849014	M	19.81	22.15	130	1260	0.09831	0.1027	0.1479	0.09498	0.1582	0.05395	0.7582	1.017	5.865
8510426	B	13.54	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.04781	0.1885	0.05766	0.2699	0.7886	2.058
8510653	B	13.08	15.71	85.63	520	0.1075	0.127	0.04568	0.0311	0.1967	0.06811	0.1852	0.7477	1.383
8510824	B	9.504	12.44	60.34	273.9	0.1024	0.06492	0.02956	0.02076	0.1815	0.06905	0.2773	0.9768	1.909
8511133	M	15.34	14.26	102.5	704.4	0.1073	0.2135	0.2077	0.09756	0.2521	0.07032	0.4388	0.7096	3.384
851509	M	21.16	23.04	137.2	1404	0.09428	0.1022	0.1097	0.08632	0.1769	0.05278	0.6917	1.127	303
852552	M	16.65	21.38	110	904.6	0.1121	0.1457	0.1525	0.0917	0.1995	0.0633	0.8068	0.9017	5.455
852631	M	17.14	16.4	116	912.7	0.1186	0.2276	0.2229	0.1401	0.304	0.07413	1.046	0.976	7.276

Breast Cancer Wisconsin Dataset - Variable Descriptions

- **Radius**: Mean of distances from the center to points on the perimeter.
- **Texture**: Standard deviation of gray-scale values.
- **Perimeter**: Perimeter of the tumor.
- **Area**: Area of the tumor.
- **Smoothness**: Local variation in radius lengths.
- **Compactness**: $(\frac{\text{Perimeter}^2}{\text{Area}}) - 1$
- **Concavity**: Severity of concave portions of the contour.
- **Concave Points**: Number of concave portions of the contour.
- **Symmetry**: Symmetry of the tumor.
- **Fractal Dimension**: Coastline approximation – fractal dimension of the tumor.
- **Class**: 1 = Malignant, 0 = Benign (target variable).

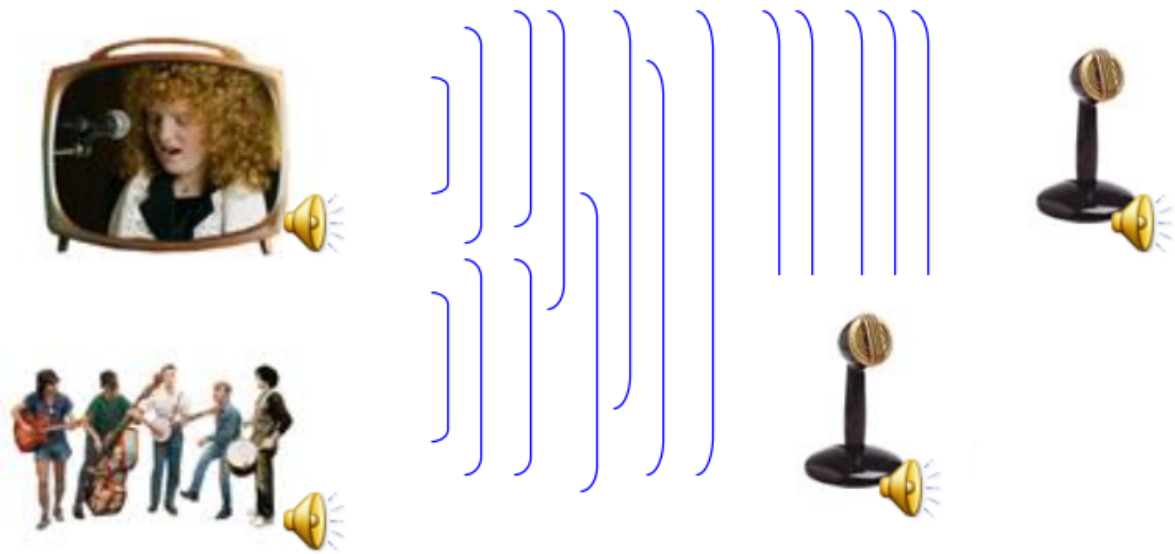
Problem & Motivation




?
?
Is this Koala?
Yes
No

Motivation - Cocktail-Party Problem

- Simple scenario:
  - Two people speaking simultaneously in a room.
  - Speeches are recorded by two microphones in separate locations.



Motivation - Cocktail-Party Problem

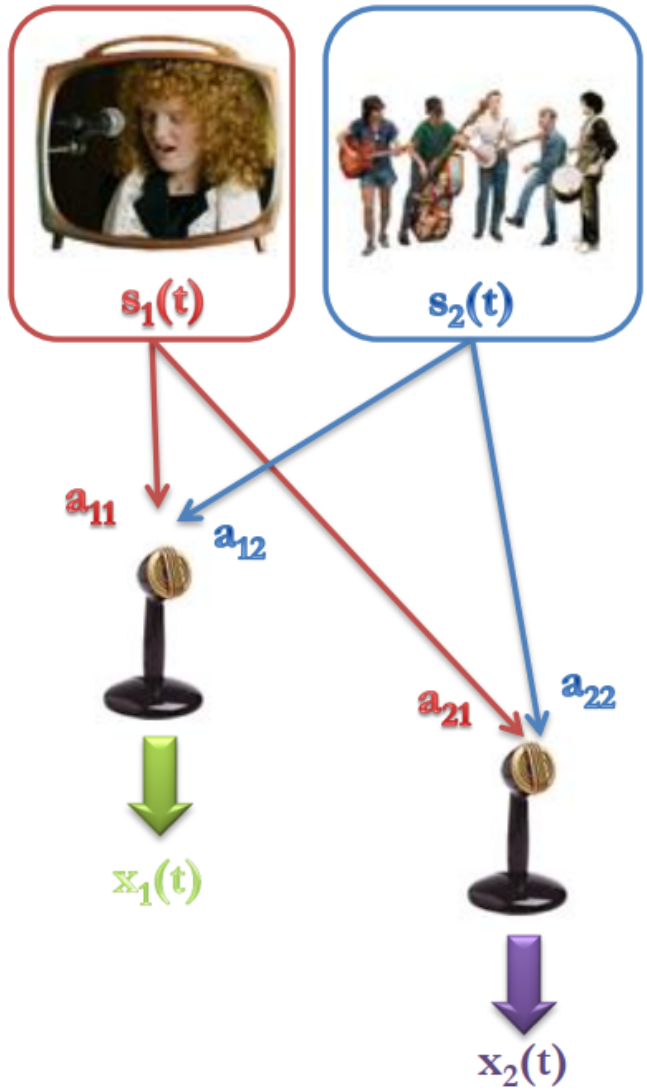
- Let $\mathbf{s_1(t)}$, $\mathbf{s_2(t)}$ be the speech signals emitted by the two speakers.
- Recorded time signals, by the two microphones, are denoted by $\mathbf{x_1(t)}$, $\mathbf{x_2(t)}$.
- The recorded time signals can be expressed as a linear equation:

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t)$$

$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t)$$

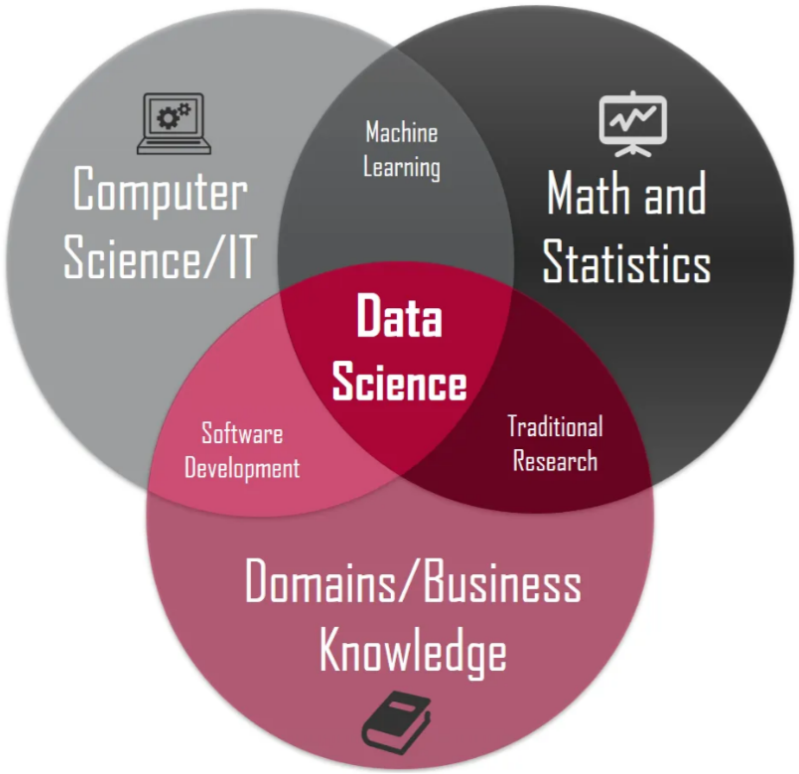
where parameters in matrix **A** depend on distances of the microphones to the speaker, along with other microphone properties

- Assume $\mathbf{s_1(t)}$ and $\mathbf{s_2(t)}$ are *statistically independent*.



What is Data Science?

- Data science is the art of deriving meaningful insights and solutions from vast amounts of data through the application of statistical analysis, machine learning, and computational methods.
- It involves the application of scientific methods, algorithms, and systems to extract insights and knowledge from large volumes of data.



- The term "data science" was first used by the Danish computer scientist Peter Naur in 1960. He used it as a substitute for "computer science" in his work, particularly in his influential book "Concise Survey of Computer Methods," published in 1974.

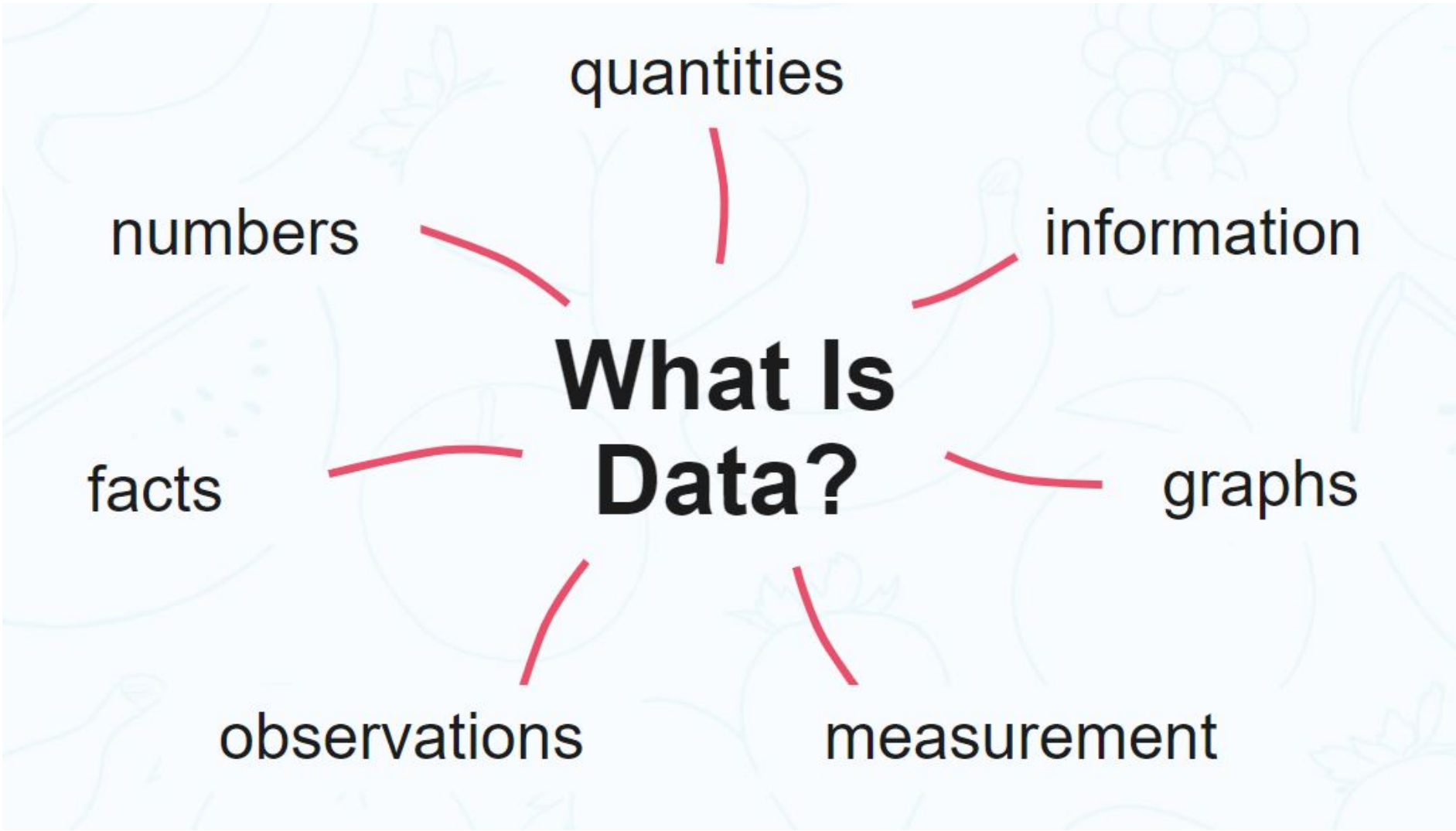
What is Data?

Data

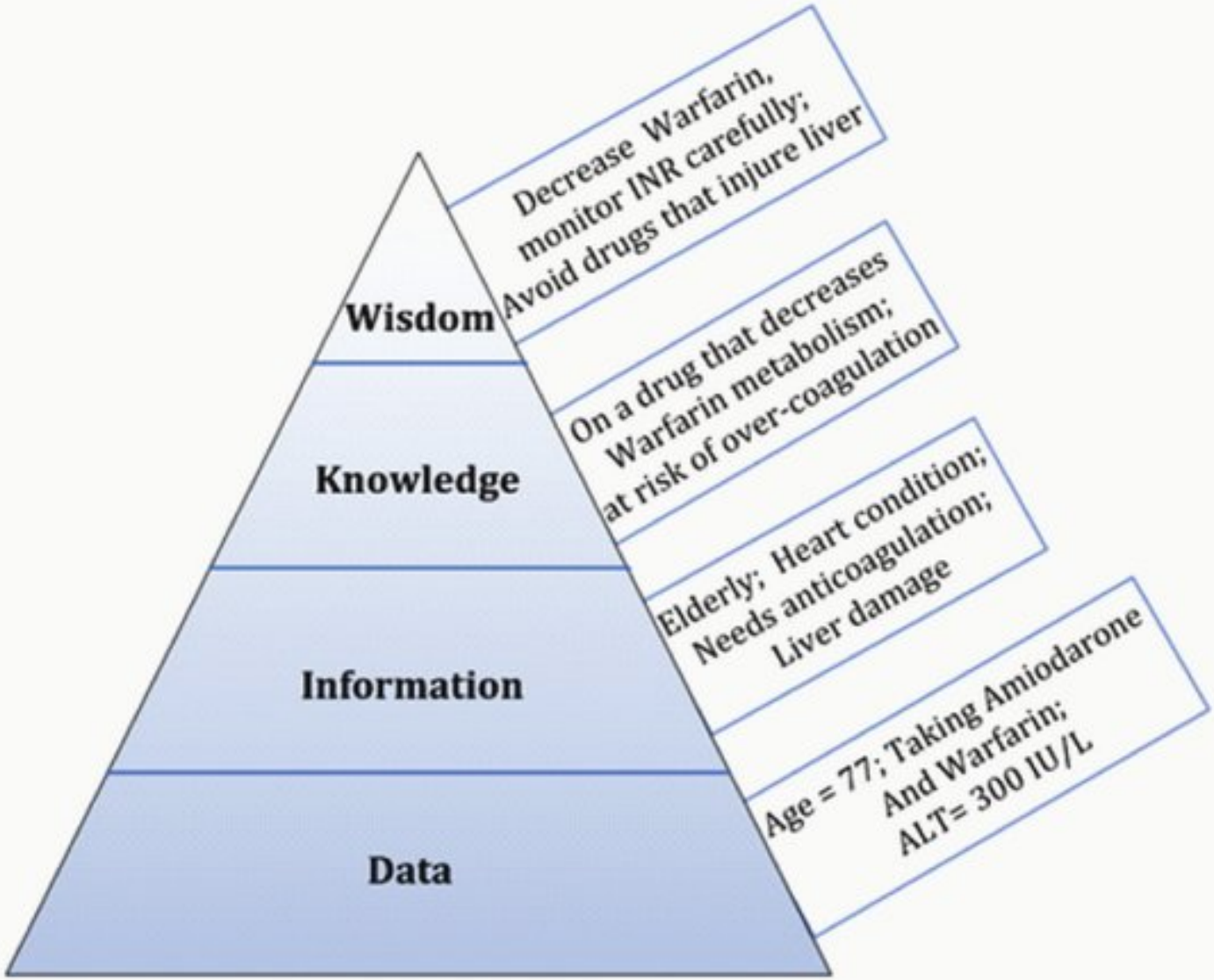
- data are individual units of information
- these facts and figures are collected, analyzed, and summarized for presentation and interpretation
- all the data collected in a particular study are referred to as the data set for the study

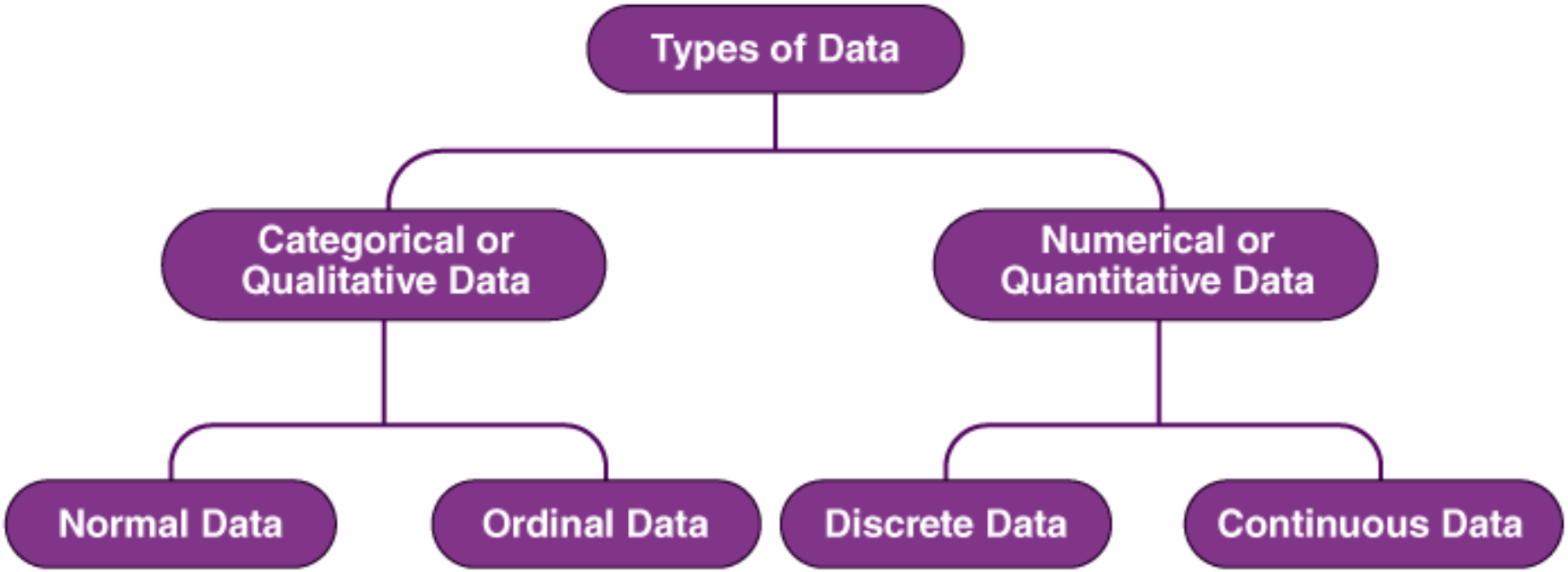
Datum

- datum (a singular form of "data") is a single piece of information


quantities
numbers
information
What Is Data?
facts
graphs
observations
measurement

DIKW (Data-Information-Knowledge-Wisdom) Pyramid




Types of Data
Categorical or Qualitative Data
Numerical or Quantitative Data
Normal Data
Ordinal Data
Discrete Data
Continuous Data

Types of data

1. quantitative data —ex: Family size, Score, Age
   - continuous data
     - It can take any value within a range and can be measured with precision, like height, weight, or temperature.
   - discrete data
     - Discrete data occur when the data can only take certain values. The possible values of discrete data are generally integers. For instance, if we also collect data on the number of cases of cold (0, 1, 2, ...) in 2020 for the 7-year-old boys, then they are discrete data.
2. qualitative data (also called categorical data)
   - nominal-level data (ex: Gender, ID number, cell number, blood type)
   - ordinal-level data (ex: Cancer stage, Grade, the treatment effect of a disease — cured, effective, improved, ineffective, and deteriorated.)

Types of data based on sources

1. primary data
   (See details in the course WM-ASDS02: Sampling Methodology)
2. secondary data

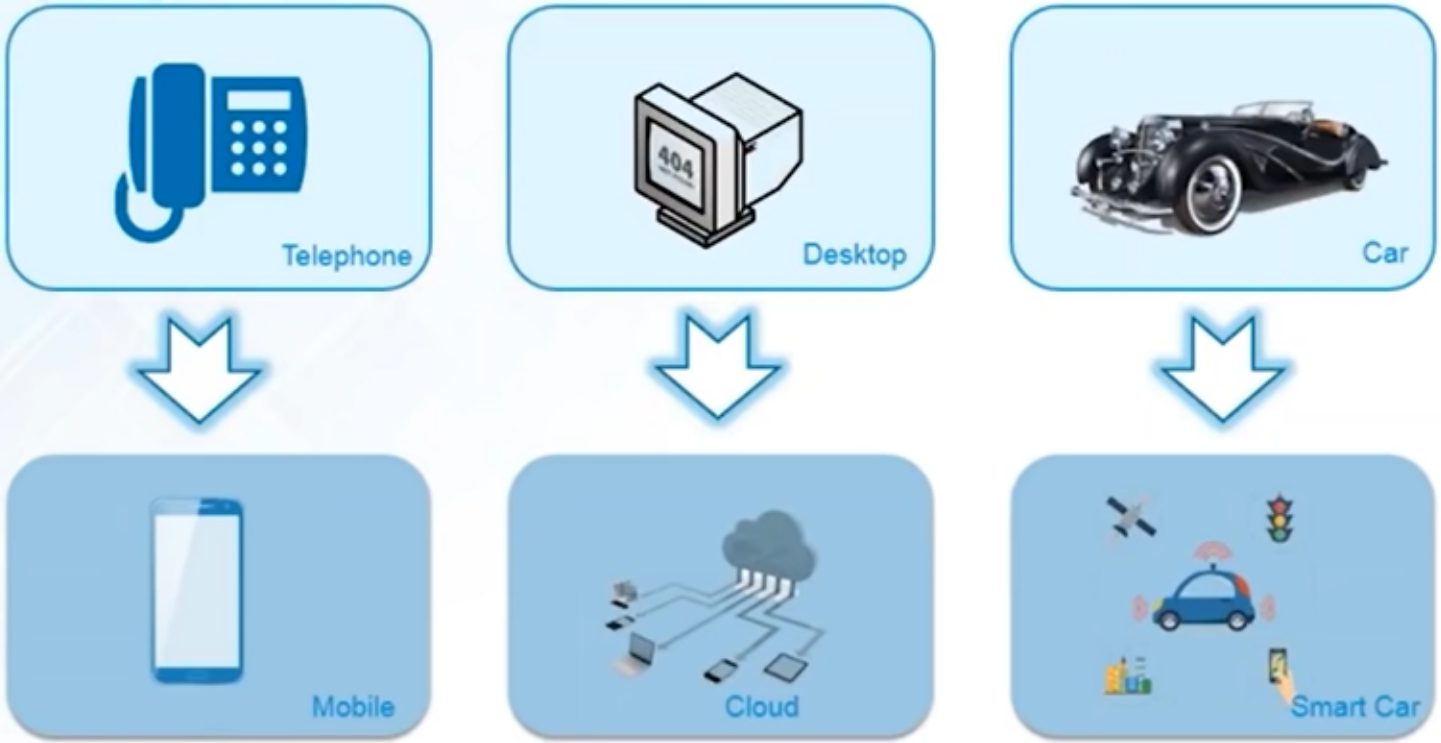
Classroom Exercise

What kind of data is the following?

1. hypertensive (1 for hypertensive and 0 for not hypertensive)
2. degrees of hypertension
   - Normal: Below SBP 120/DBP 80.
   - Elevated: SBP 120 to 129/DBP less than 80.
   - Stage 1 high blood pressure: SBP 130 to 139/DBP 80 to 89.
   - Stage 2 high blood pressure: SBP 140 and above/DBP 90 and above.
   - Hypertension crisis: SBP above 180/DBP above 120.
3. degree of the blood pressure (low, normal, high)
4. number of white blood cells (WBC) in 1 mm^3 of blood
5. hemoglobin level
6. number of leaves on a plant
7. number of giraffes visiting a water hole
8. number of WBC in CBC test

Evolution of Technology

1. Evolution of Technology
2. IOT
3. Social Media
4. Data evolved to Big Data

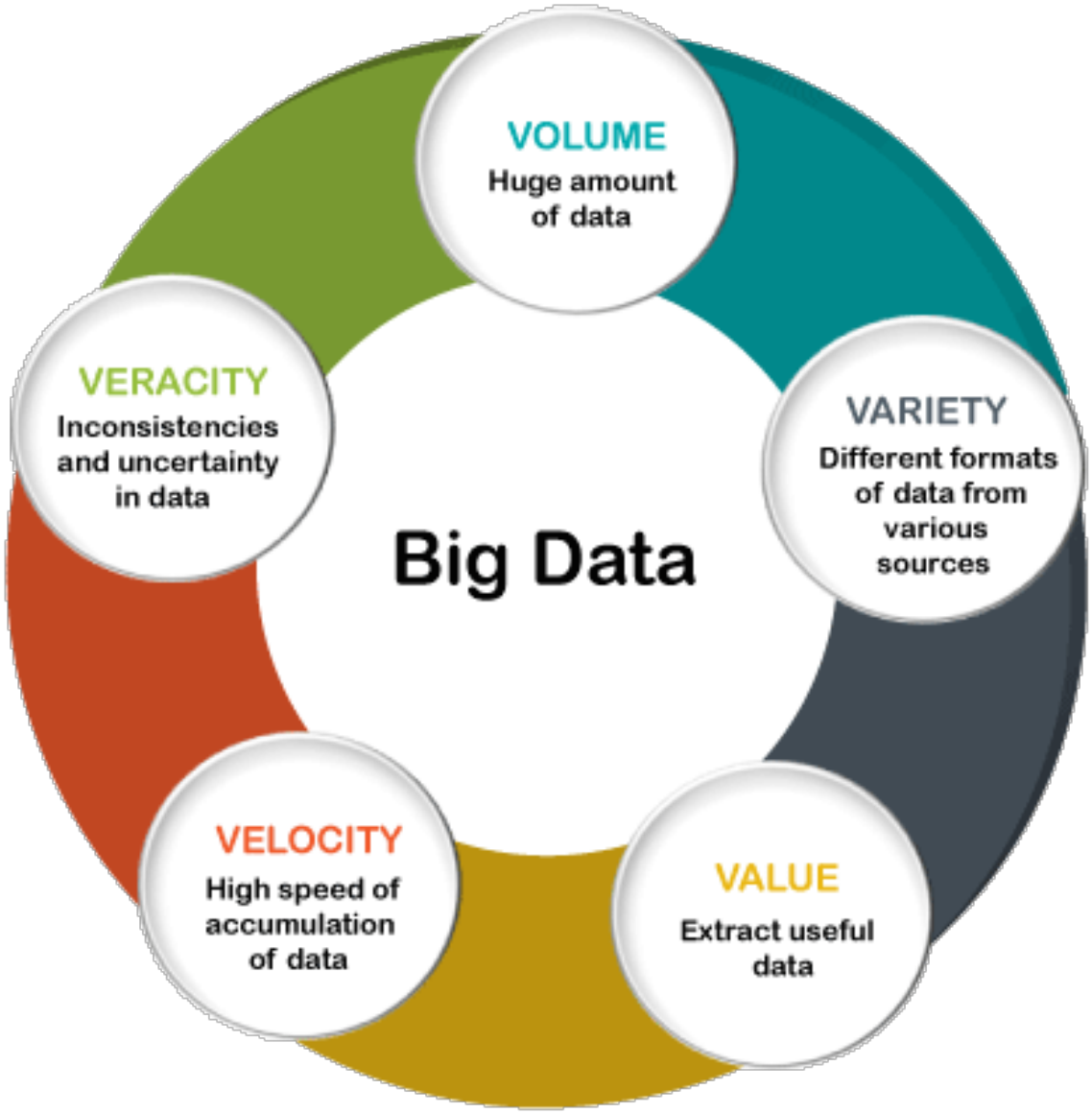


Remarks:

- 1 zettabyte is equal to 1,024 exabytes (EB).
- 1 exabyte is equal to 1,024 petabytes (PB).
- 1 petabyte is equal to 1,024 terabytes (TB).

Therefore, 1 zettabyte is equal to approximately 1.07 trillion terabytes exactly $1{,}024^3$, which is approximately 1073741824 terabytes).

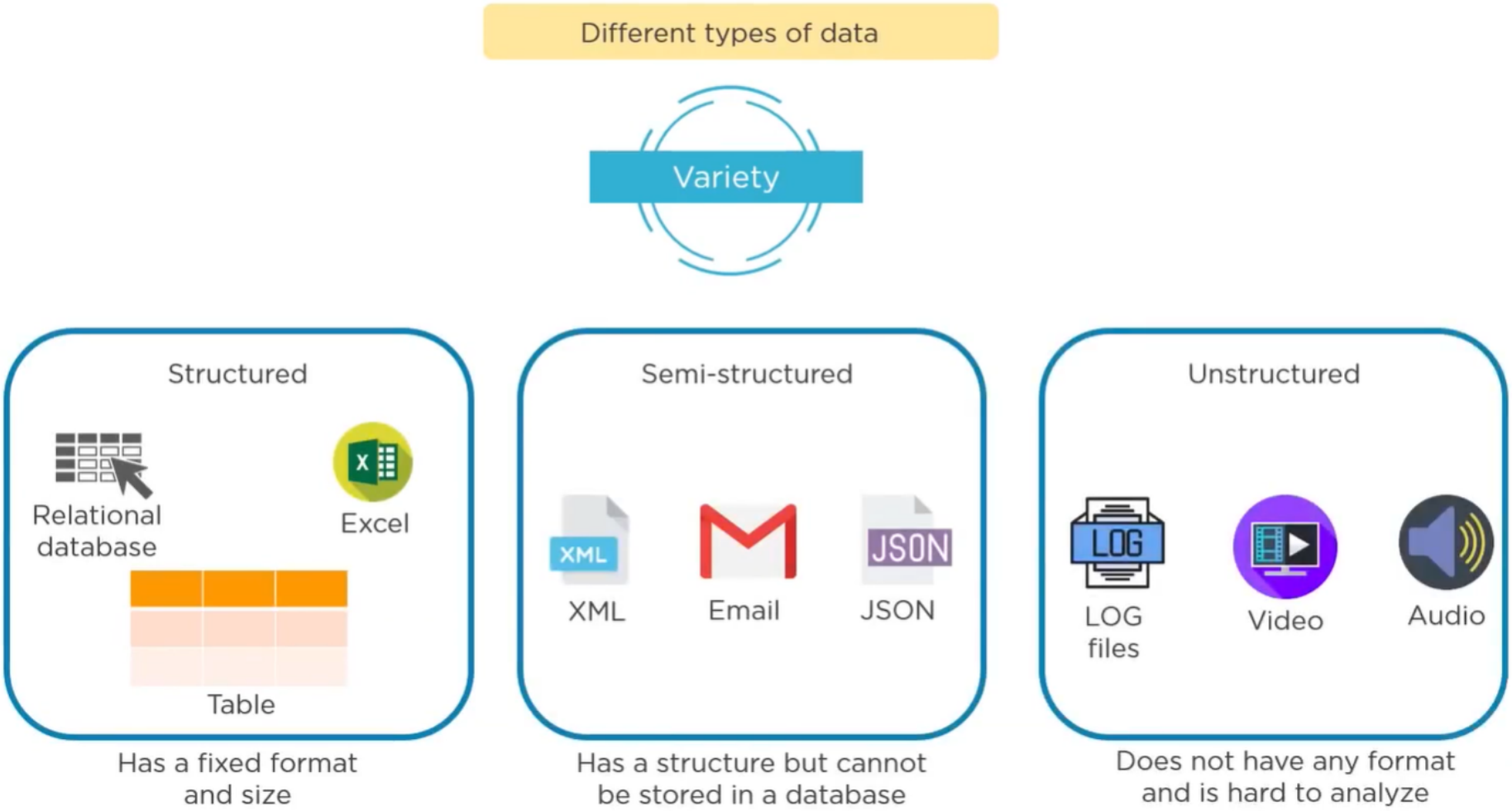
What is Big Data?



Big Data can be

- structured
  - In Structured schema, along with all the required columns. It is in a tabular form. Structured Data is stored in the relational database management system.
- unstructured
  - All the unstructured files, log files, audio files, and image files are included in the unstructured data. Some organizations have much data available, but they did not know how to derive the value of data since the data is raw.
- semi-structured
  - In Semi-structured, the schema is not appropriately defined, e.g., JSON, XML, CSV, TSV, and email. OLTP (Online Transaction Processing) systems are built to work with semi-structured data. It is stored in relations, i.e., tables.

that are being collected from different sources.


Different types of data
Variety
Structured
Relational database
Excel
Table
Has a fixed format and size
Semi-structured
XML
Email
JSON
Has a structure but cannot be stored in a database
Unstructured
LOG files
Video
Audio
Does not have any format and is hard to analyze

Data File

application_log - Notepad

File Edit Format View Help

```
2015-12-31 02:16:59,929 WARN [main] org.apache.hadoop.metrics2.impl.MetricsConfig: Cannot locate configuration:
-maptask.properties,hadoop-metrics2.properties
2015-12-31 02:17:00,004 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: Scheduled snapshot perio
2015-12-31 02:17:00,004 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: MapTask metrics system s
2015-12-31 02:17:00,016 INFO [main] org.apache.hadoop.mapred.YarnChild: Executing with tokens:
2015-12-31 02:17:00,016 INFO [main] org.apache.hadoop.mapred.YarnChild: Kind: mapreduce.job, Service: job_14505
(org.apache.hadoop.mapreduce.security.token.JobTokenIdentifier@14e50583)
2015-12-31 02:17:00,126 INFO [main] org.apache.hadoop.mapred.YarnChild: Sleeping for 0ms before retrying again.
2015-12-31 02:17:00,562 INFO [main] org.apache.hadoop.mapred.YarnChild: mapreduce.cluster.local.dir for child:
/hd_data/disk1/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk2/had
e/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk3/hadoop/yarn/local/usercache/hsgctcrp/appcach
8170_12821,/hd_data/disk4/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_dat
al/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk6/hadoop/yarn/local/usercache/hsgct
n_1450565638170_12821,/hd_data/disk7/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12
op/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk9/hadoop/yarn/local/user
/application_1450565638170_12821,/hd_data/disk10/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450
a/disk11/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk12/hadoop/y
ctcrp/appcache/application_1450565638170_12821,/hd_data/disk13/hadoop/yarn/local/usercache/hsgctcrp/appcache/ap
_12821,/hd_data/disk14/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/d
/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk16/hadoop/yarn/local/usercache/hsgctc
_1450565638170_12821,/hd_data/disk17/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12
oop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk19/hadoop/yarn/local/us
he/application_1450565638170_12821,/hd_data/disk20/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_14
ata/disk21/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk22/hadoop
sgctcrp/appcache/application_1450565638170_12821,/hd_data/disk23/hadoop/yarn/local/usercache/hsgctcrp/appcache/
70_12821,/hd_data/disk24/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821
2015-12-31 02:17:01,275 INFO [main] org.apache.hadoop.conf.Configuration.deprecation: session.id is deprecated.
dfs.metrics.session-id
```

Structured

application_log - Notepad

File Edit Format View Help

```
2015-12-31 02:16:59,929 WARN [main] org.apache.hadoop.metrics2.impl.MetricsConfig: Cannot locate configuration: tried hadoop-metrics2
-maptask.properties,hadoop-metrics2.properties
2015-12-31 02:17:00,004 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: Scheduled snapshot period at 10 second(s).
2015-12-31 02:17:00,004 INFO [main] org.apache.hadoop.metrics2.impl.MetricsSystemImpl: MapTask metrics system started
2015-12-31 02:17:00,016 INFO [main] org.apache.hadoop.mapred.YarnChild: Executing with tokens:
2015-12-31 02:17:00,016 INFO [main] org.apache.hadoop.mapred.YarnChild: Kind: mapreduce.job, Service: job_1450565638170_12821, Ident:
(org.apache.hadoop.mapreduce.security.token.JobTokenIdentifier@14e50583)
2015-12-31 02:17:00,126 INFO [main] org.apache.hadoop.mapred.YarnChild: Sleeping for 0ms before retrying again. Got null now.
2015-12-31 02:17:00,562 INFO [main] org.apache.hadoop.mapred.YarnChild: mapreduce.cluster.local.dir for child:
/hd_data/disk1/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk2/hadoop/yarn/local/usercach
e/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk3/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_145056563
8170_12821,/hd_data/disk4/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk5/hadoop/yarn/loc
al/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk6/hadoop/yarn/local/usercache/hsgctcrp/appcache/applicatio
n_1450565638170_12821,/hd_data/disk7/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk8/hado
op/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk9/hadoop/yarn/local/usercache/hsgctcrp/appcache
/application_1450565638170_12821,/hd_data/disk10/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_dat
a/disk11/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk12/hadoop/yarn/local/usercache/hsg
ctcrp/appcache/application_1450565638170_12821,/hd_data/disk13/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170
_12821,/hd_data/disk14/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk15/hadoop/yarn/local
/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk16/hadoop/yarn/local/usercache/hsgctcrp/appcache/application
_1450565638170_12821,/hd_data/disk17/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk18/had
oop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk19/hadoop/yarn/local/usercache/hsgctcrp/appcac
he/application_1450565638170_12821,/hd_data/disk20/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_d
ata/disk21/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821,/hd_data/disk22/hadoop/yarn/local/usercache/h
sgctcrp/appcache/application_1450565638170_12821,/hd_data/disk23/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_14505656381
70_12821,/hd_data/disk24/hadoop/yarn/local/usercache/hsgctcrp/appcache/application_1450565638170_12821
2015-12-31 02:17:01,275 INFO [main] org.apache.hadoop.conf.Configuration.deprecation: session.id is deprecated. Instead, use
dfs.metrics.session-id
2015-12-31 02:17:01,787 INFO [main] org.apache.hadoop.mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
2015-12-31 02:17:02,122 INFO [main] org.apache.hadoop.mapred.MapTask: Processing split:
Paths:/EDMBD/PROJECTS/GCT/OCODS/CLAIMS_XML/BIX_ODS/XML_DELTA_FEED/UELEMENTARY_CLAIM/part-m-
```

Semi-Structured

Sources of Data

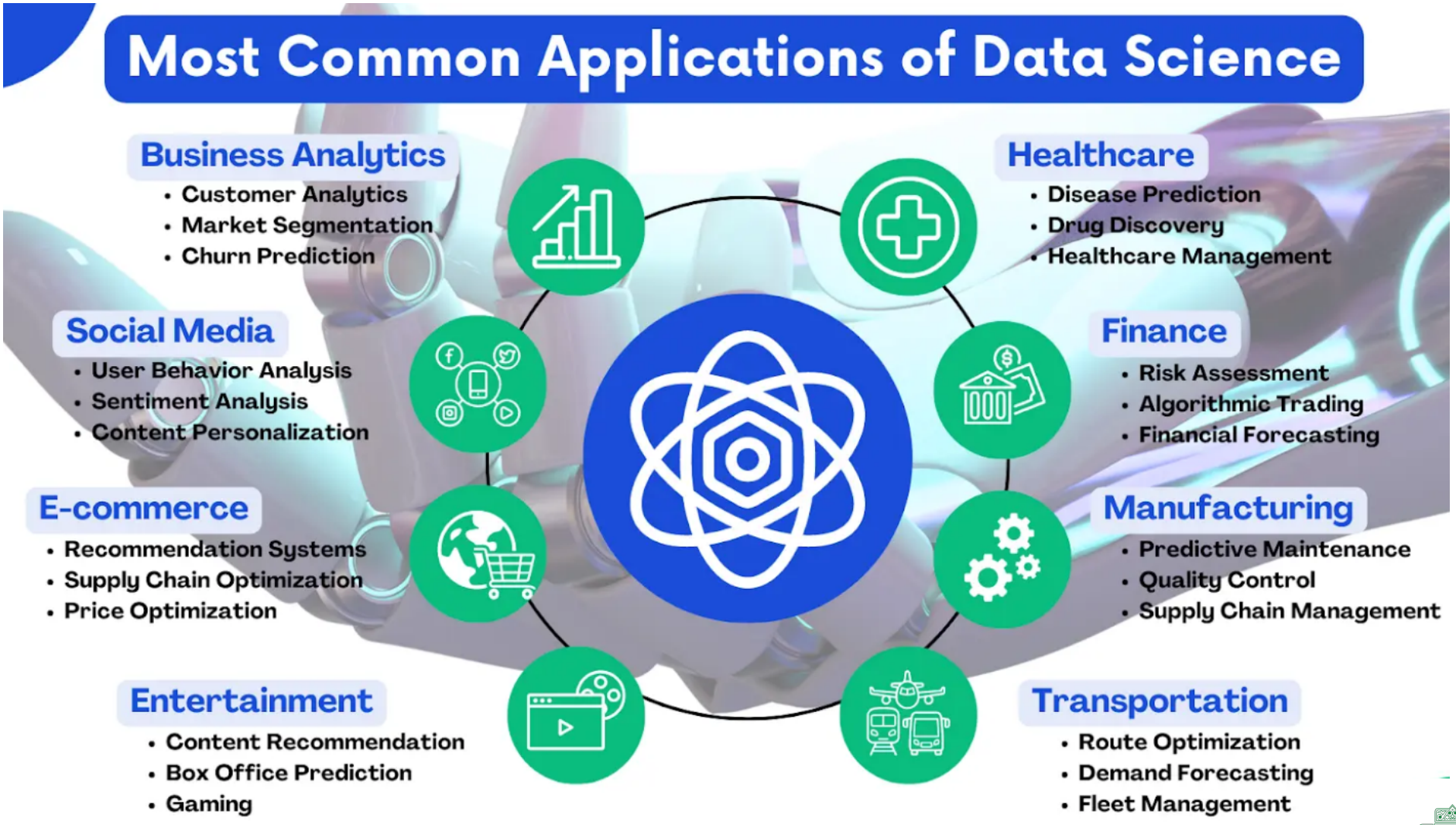
- the main sources of big data are the operation and trading information in enterprises, logistic and sensing information in the IoT, human interaction information and position information in the Internet world, and data generated in scientific research, etc.

1. Data generation
   - enterprise data (mainly consists of online trading data)
   - IoT data (data may come from industry, agriculture, traffic and transportation, medical care, public departments, and households)
   - internet data (searching entries, internet forum posts, chatting records, and microblog messages, etc.)
   - bio-medical data (inventing new drug project data, Genome data, etc.)
   - data generation from other fields

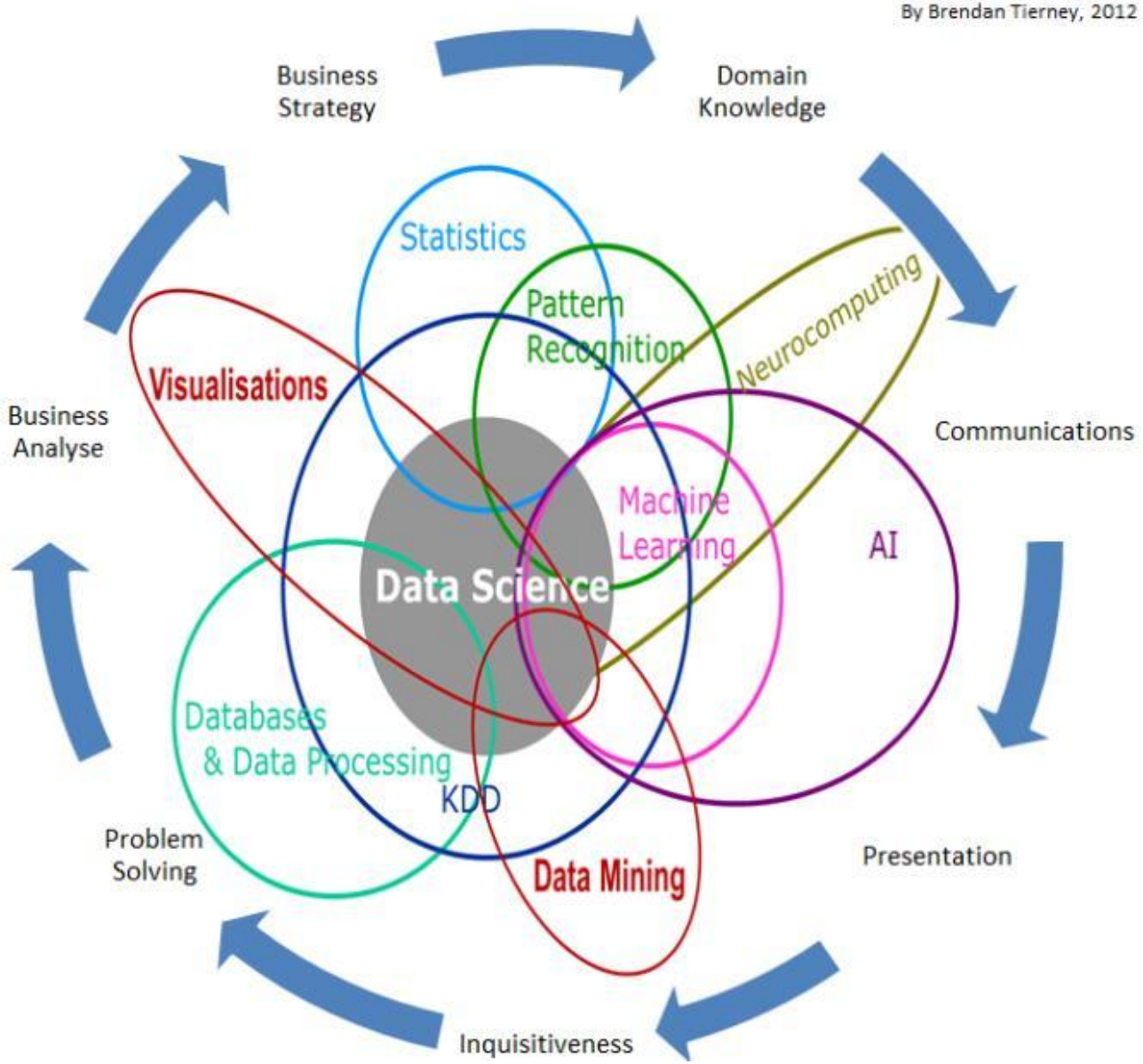
Steps of Data Science Process

1. Problem Definition: Clearly defining the problem and identifying the goal of the analysis.
2. Data Collection: Gathering and acquiring data from various sources, including data cleaning and preparation.
3. Data Exploration: Exploring the data to gain insights and identify trends, patterns, and relationships.
4. Data Modeling: Building mathematical models and algorithms to solve problems and make predictions.
5. Evaluation: Evaluating the model's performance and accuracy using appropriate metrics.
6. Deployment: Deploying the model in a production environment to make predictions or automate decision-making processes.
7. Monitoring and Maintenance: Monitoring the model's performance over time and making updates as needed to improve accuracy.

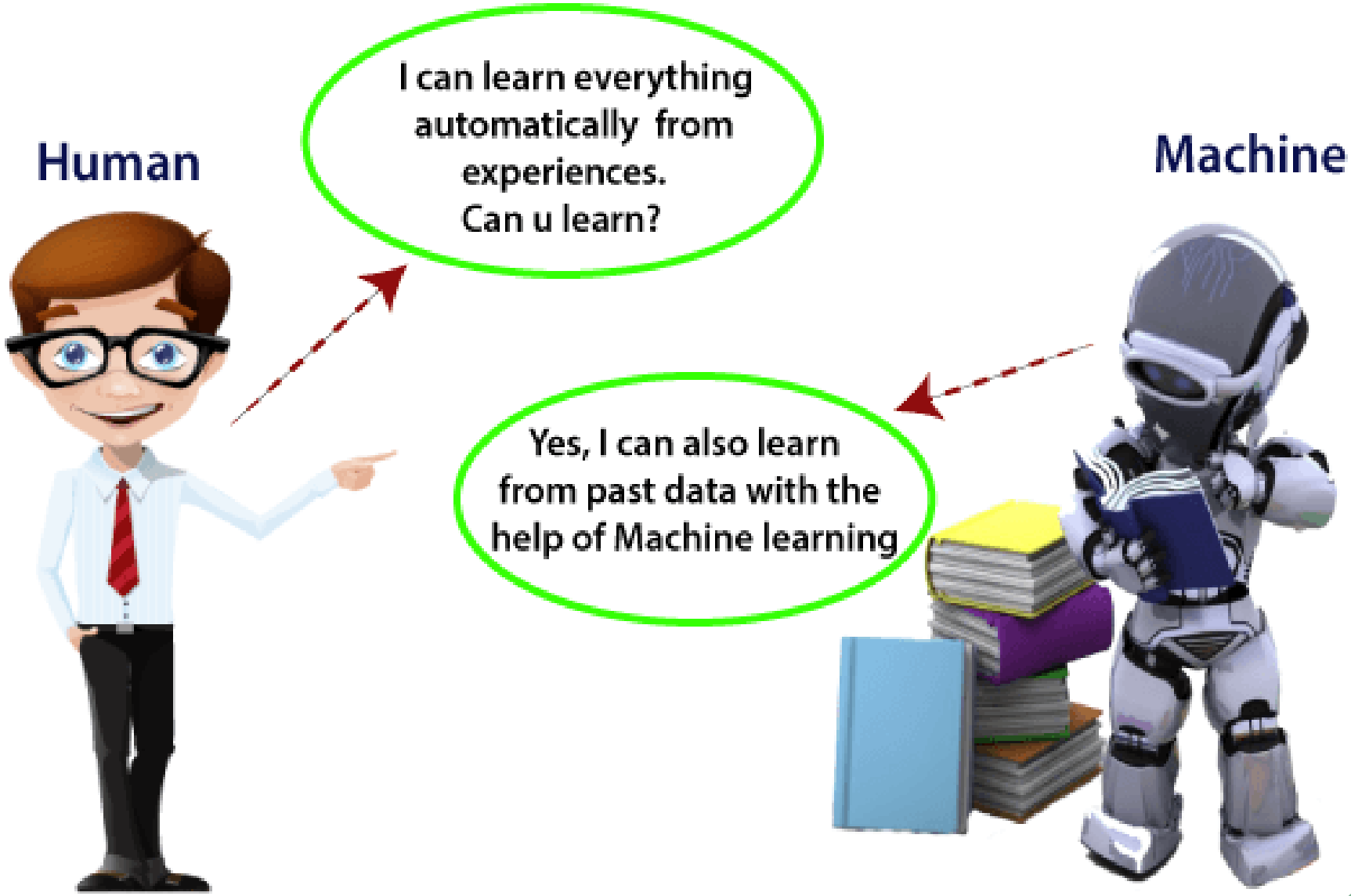
Applications of Data Science



Data Science Is Multidisciplinary



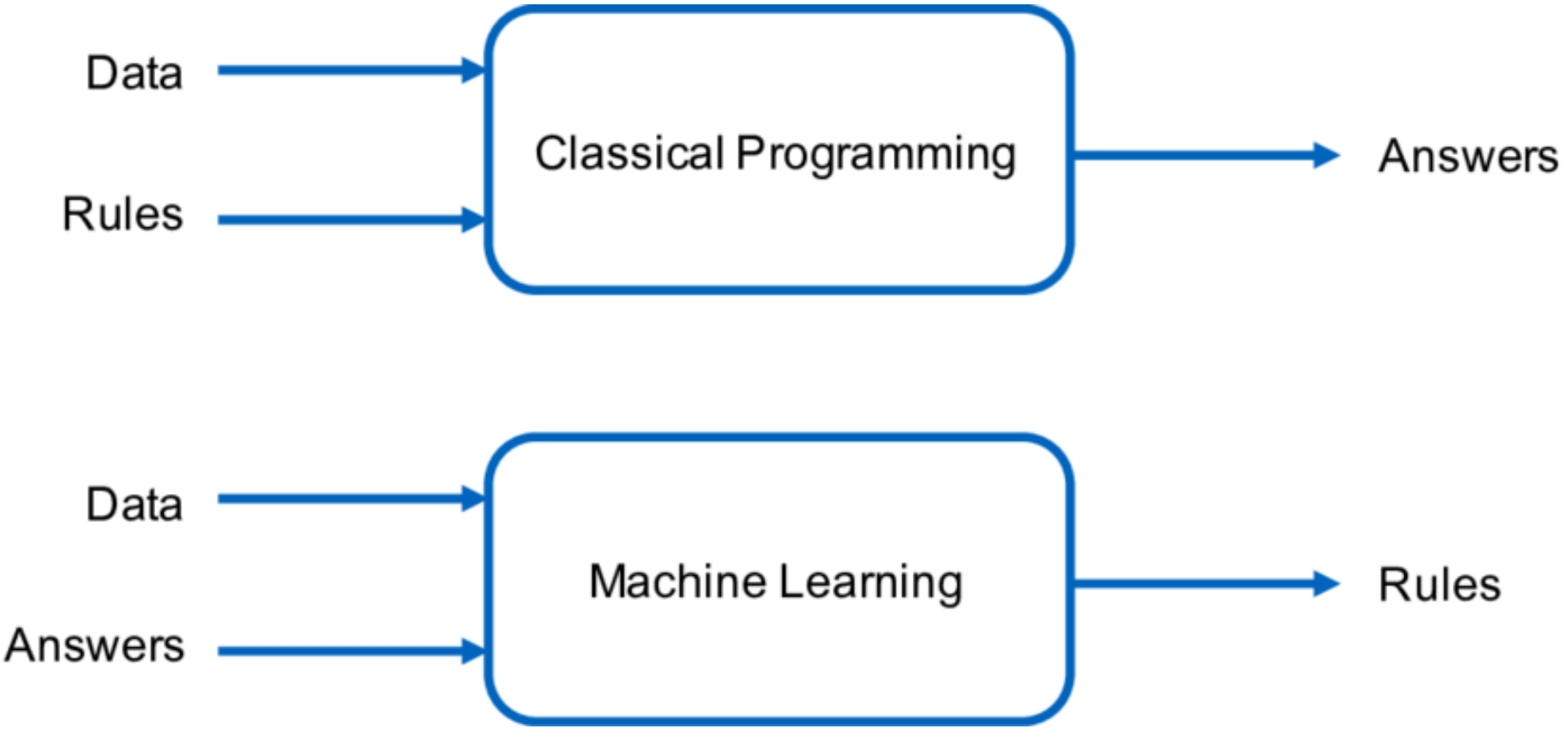
Machine Learning



What is Machine Learning?

- Machine Learning (ML) is the kind of programming technology which gives computers the capability to automatically learn from past data without being explicitly programmed. This means in other words that these programs change their behaviour by learning from data.
- ML algorithms use historical data as input to predict new output values
- Currently, it is being used for various tasks such as
  - image recognition
  - speech recognition
  - email filtering
  - Facebook auto-tagging
  - recommender system
  - ...

Tradition Programming vs Machine Learning



What are the different types of machine learning?

There are four basic approaches:

1. supervised learning
2. unsupervised learning
3. semi-supervised learning
4. reinforcement learning

The type of algorithm data scientists choose to use depends on what type of data they want to predict.

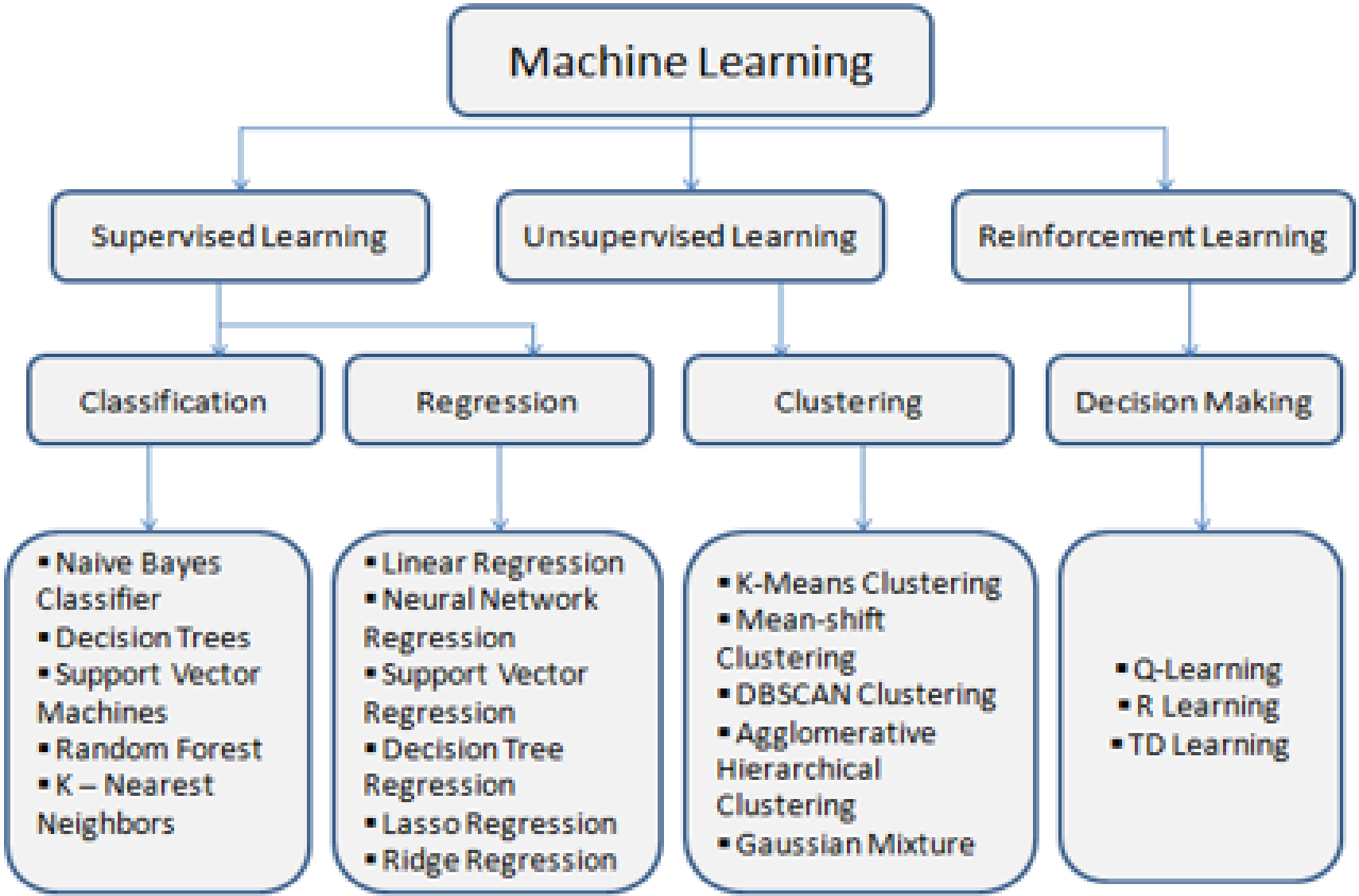
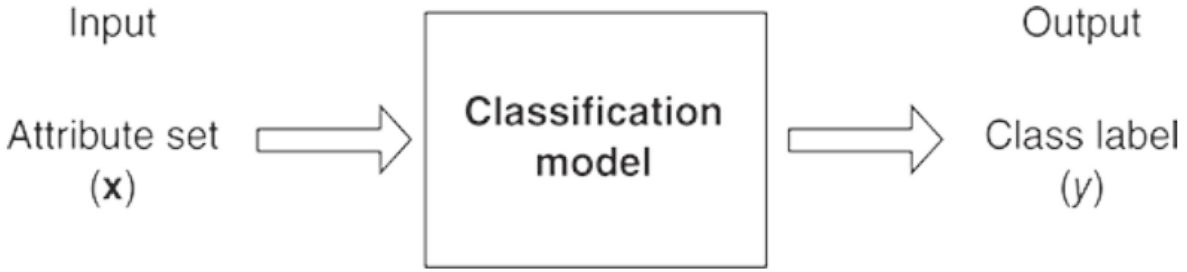


Fig: 2 Classification of Machine Learning algorithms

Supervised Machine Learning

supervised learning

- is the learning of the model where with input variable (say, x) and an output variable (say, Y) and an algorithm to map the input to the output, i. e., $Y = f(x)$
- the basic aim is to approximate the mapping function so well that when there is a new input data (x) then the corresponding output variable can be predicted
- why supervised learning?
  — it is called supervised learning because the process of an learning (from the training dataset) can be thought of as a teacher who is supervising the entire learning process



Examples of supervised learning

Task	Attribute set	Class label
Spam filtering	Features extracted from email message header and content	spam or non-spam
Tumor identification	Features extracted from magnetic resonance imaging (MRI) scans	malignant or benign
Galaxy classification	Features extracted from telescope images	elliptical, spiral, or irregular-shaped

Supervised Machine Learning

- Supervised learning, also known as supervised machine learning, is defined by its use of labeled datasets to train algorithms to classify data or predict outcomes accurately.
- The machine learning program is both given the input data and the corresponding labelling. This means that the learn data has to be labelled by a human being beforehand.
- Supervised learning helps organizations solve a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.
- Some methods used in supervised learning include
  - naïve bayes
  - neural networks
  - decision trees
  - random forest
  - support vector machine (SVM)
  - boosting and bagging
  - ...

Unsupervised Machine Learning

unsupervised learning

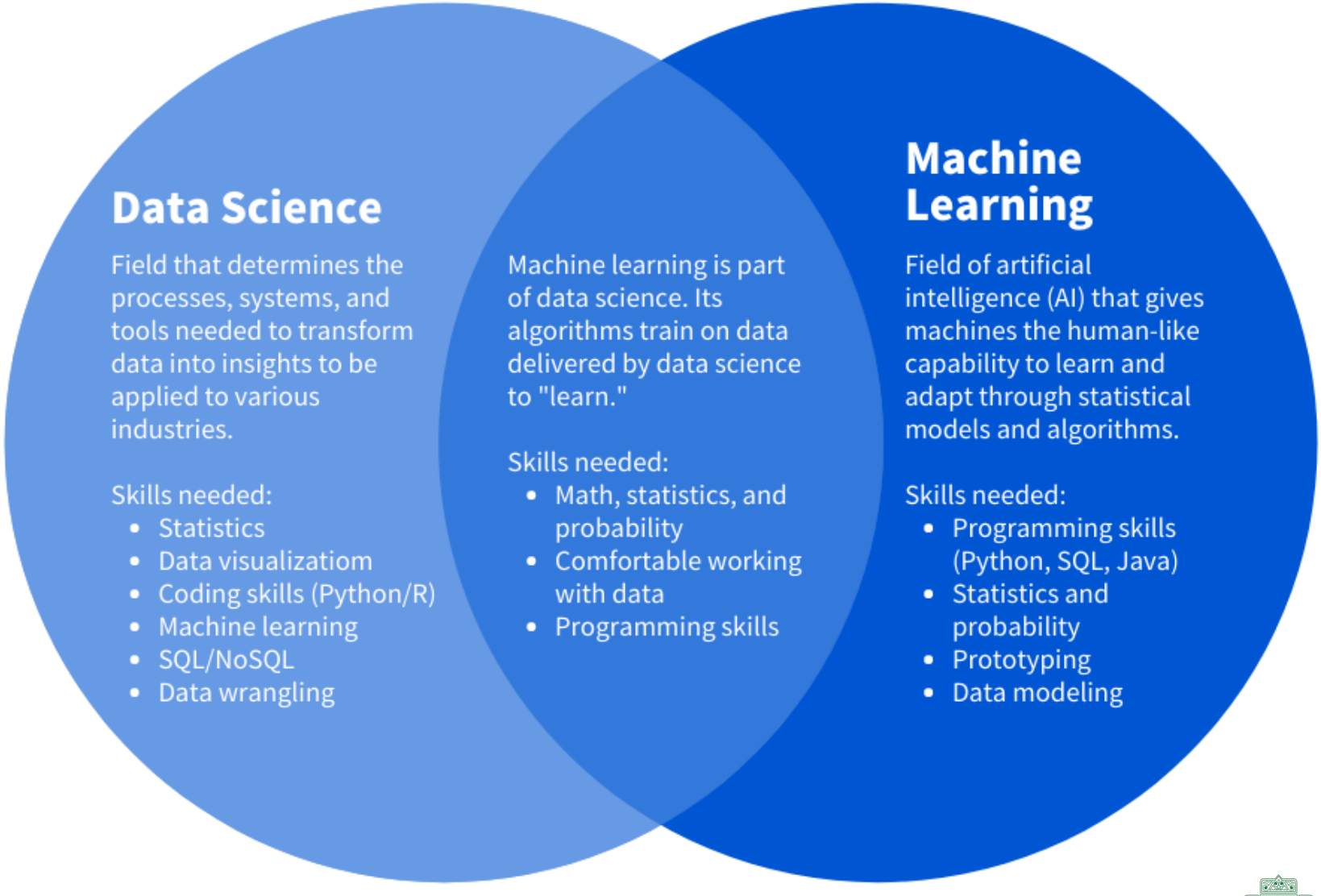
- no labels are provided to the learning algorithm. The algorithm has to figure out the a clustering of the input data.
- the main aim of unsupervised learning is to model the distribution in the data in order to learn more about the data
- it is called so, because there is no correct answer and there is no such teacher (unlike supervised learning)
- algorithms are left to their own devises to discover and present the interesting structure in the data
- examples of unsupervised learning
  - cluster analysis
  - association mining
  - ...

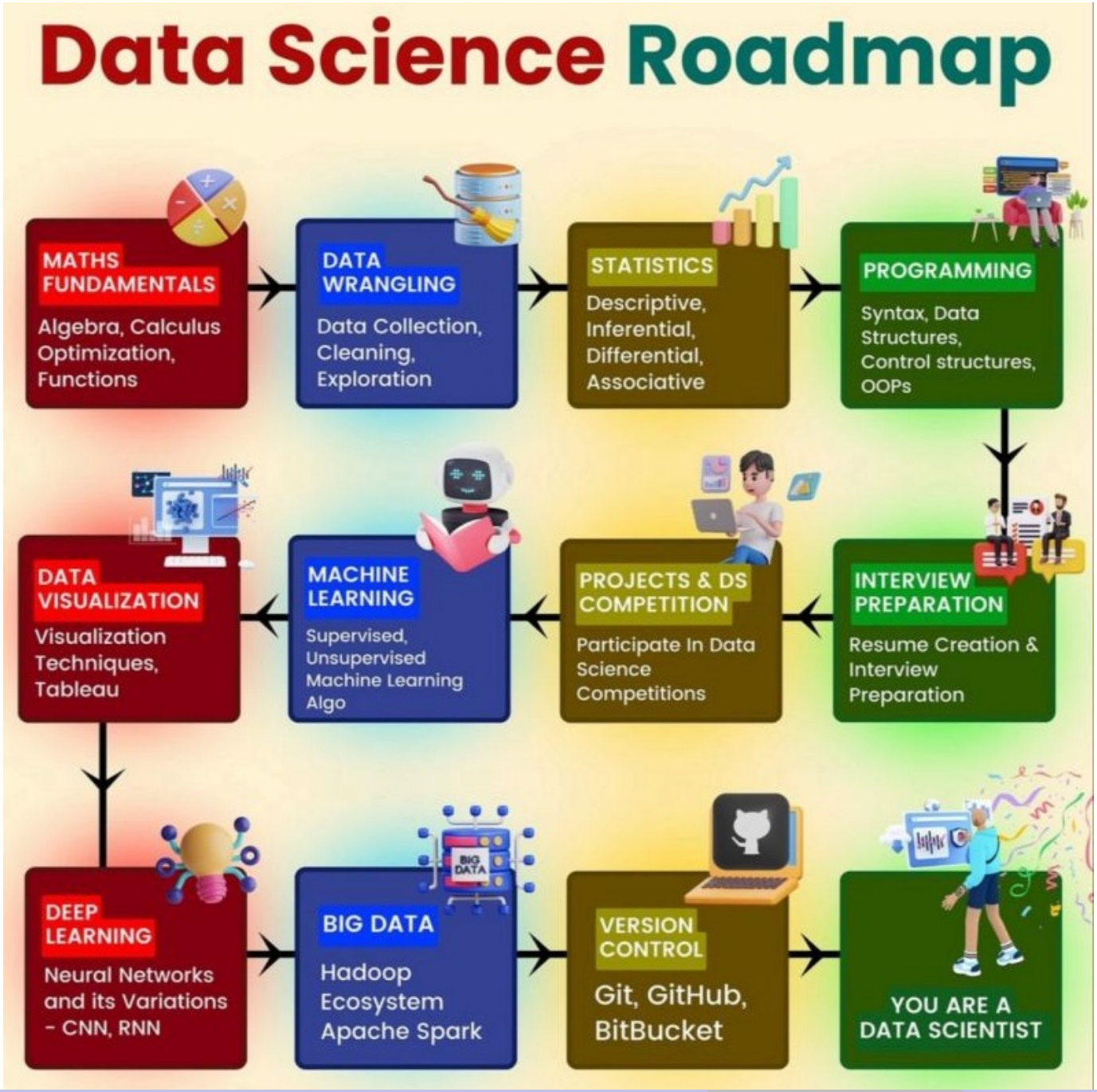
Semi-supervised learning

- is a machine learning technique that uses a small portion of labeled data and lots of unlabeled data to train a predictive model
- during training, it uses a smaller labeled data set to guide classification and feature extraction from a larger, unlabeled data set

Reinforcement Machine Learning

- Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action.
- The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it.
- The goal of an agent is to get the most reward points, and hence, it improves its performance.
- Reinforcement learning solves a specific type of problem where decision-making is sequential and the goal is long-term, such as game-playing, robotics, etc.


Data Science
Field that determines the processes, systems, and tools needed to transform data into insights to be applied to various industries.
Skills needed:
• Statistics
• Data visualizatiom
• Coding skills (Python/R)
• Machine learning
• SQL/NoSQL
• Data wrangling
Machine learning is part of data science. Its algorithms train on data delivered by data science to "learn."
Skills needed:
• Math, statistics, and probability
• Comfortable working with data
• Programming skills
Machine Learning
Field of artificial intelligence (AI) that gives machines the human-like capability to learn and adapt through statistical models and algorithms.
Skills needed:
• Programming skills (Python, SQL, Java)
• Statistics and probability
• Prototyping
• Data modeling


Data Science Roadmap
MATHS FUNDAMENTALS
Algebra, Calculus Optimization, Functions
DATA WRANGLING
Data Collection, Cleaning, Exploration
STATISTICS
Descriptive, Inferential, Differential, Associative
PROGRAMMING
Syntax, Data Structures, Control structures, OOPs
INTERVIEW PREPARATION
Resume Creation & Interview Preparation
PROJECTS & DS COMPETITION
Participate In Data Science Competitions
MACHINE LEARNING
Supervised, Unsupervised Machine Learning Algo
DATA VISUALIZATION
Visualization Techniques, Tableau
DEEP LEARNING
Neural Networks and its Variations - CNN, RNN
BIG DATA
Hadoop Ecosystem Apache Spark
VERSION CONTROL
Git, GitHub, BitBucket
YOU ARE A DATA SCIENTIST

Software for Data Science

- Python
- R (or RStudio)
- MATLAB
- SAS (Statistical Analysis System)
- SPSS (Statistical Package for the Social Sciences)
- Tableau
- Apache Spark

Python is a versatile language widely used in data science due to its rich ecosystem of libraries such as `NumPy`, `pandas`, `scikit-learn`, `TensorFlow`, and `PyTorch`.

Chapter 2: Getting Started with Scientific Python

2 Chapter 2: Getting Started with Scientific Python

Python!

- is
  - a high-level programming language
  - an interpreted
  - an object-oriented
- is *fundamental* to data science and machine learning, as well as an everexpanding list of areas including cyber-security, and web programming
- is a language designed for scientists and engineers who might not have formal training in software development.
- created in 1991 by Guido van Rossum
  — Named for Monty Python
- useful as a scripting language or glue language
- Filename extensions: .py,.pyi,.pyc,.pyd,.pyo (prior to 3.5),.pyw,.pyz (since 3.5)

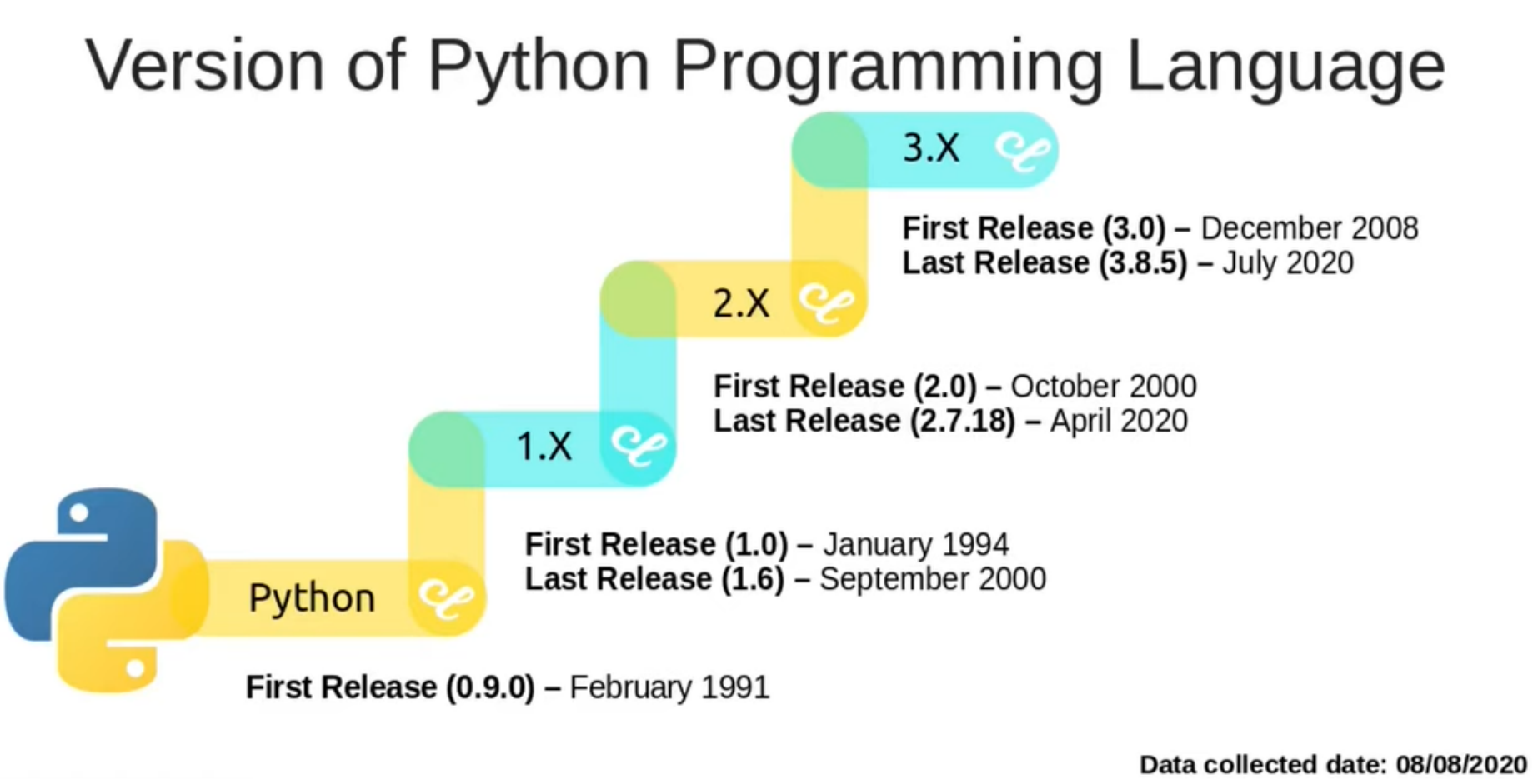
Guido van Rossum is a Dutch programmer best known as the creator of the Python programming language.

Why should you learn Python?

- beginner friendly, it has simple syntax similar to the English language
- write fewer lines than some other programming languages
- Python works on different platforms (Windows, Mac, Linus, etc.)
- used by many popular organization such as
  - google, yahoo!, youtube, NASA, etc.
  - many Linux distributions
  - games and apps (e.g. Eve Online)
- Large community support
- Versatile (used in AI, web, automation, and more)

What is Python used for?

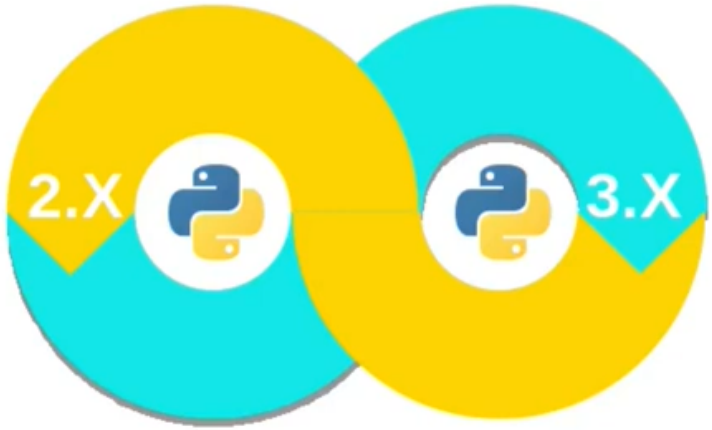
- web development
- desktop applications with PyQt, Gtk, wxWidgets and many more
- mobile applications using kivy or BeeWare
- Data Science and Visualization using *NumPy*, *Pandas* and *Matplotlib*
- Machine learning with `Tensorflow` and `Scikit-learn`
- artificial intelligence



On 7 September 2022, four new releases were made due to a potential denial-of-service attack: 3.10.7, 3.9.14, 3.8.14, and 3.7.14. As of October 2023, Python 3.12 is the stable release, and 3.12 and 3.11 are the only versions with active (as opposed to just security) support.

Python 2.X vs 3.X

- Not compatible - libraries
- print(" ") or print " "
- Division – integer or decimal no
- Comprehension – change value
- ASCII Code



- Compatible - libraries
- print(" ")
- Division – decimal no
- Comprehension – not change value
- Unicode

Getting Python!

1. Basic Python 3.12.3 download (i.e., IDE then e.g. PyChanrm)
   `https://www.python.org/downloads/`
2. Python 3.7 from the "Anaconda Distribution"–Spyder, jupyter notebook
   `https://www.anaconda.com/download/`
3. PyCharm (Best for large projects)
4. Jupyter Notebook (Good for data science)
5. IDLE (Built-in Python IDE)
6. Online Python (the best one!)
   `https://colab.research.google.com/notebooks/intro.ipynb`

CAUTION: the "basic" Python installation from https://www.python.org will NOT be sufficient for all the data-analysis!

How to Run Python Scripts From an IDE or a Text Editor?

- when developing larger and more complex applications, it is recommended that you use an integrated development environment (IDE) or an advanced text editor
- to execute your script, choose the Run or Build command, which is usually available from the tool bar or from the main menu
- Python's standard distribution includes IDLE as the default IDE, and you can use it to write, debug, modify, and run your modules and scripts
- other IDEs such as jupyter notebook, Spyder, Eclipse-PyDev, PyCharm, Eric, and NetBeans also allow you to run Python scripts from inside the environment
- advanced text editors like Sublime Text and Visual Studio Code also allow you to run your scripts

Overview of Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

- Supports many programming languages (Python, R, Julia, etc.)
- Interactive environment for data analysis, visualization, and learning.
- Combines code execution with text and multimedia.
- Widely used in data science, machine learning, and academic research.

The notebook is composed of **cells**:

- **Code Cells:** Execute Python code (or other languages).
- **Markdown Cells:** Write explanations, formulas, and more using Markdown.

Jupyter Notebook

two modes in Jupyter Notebook

1. command mode
2. edit mode

```
# See installed package
!pip list

# See the directiory
%pwd

# list of the folder in the directory
%ls

# See availabe line magics command
%lsmagic
```

The symbol is used to run system commands from within a Jupyter notebook or IPython environment (like in Google Colab or JupyterLab).

Jupyter Notebook Keyboard Shortcuts

1. **Command Mode Shortcuts (Esc to enter):**
   - `A`: Insert cell above
   - `B`: Insert cell below
   - `M`: Change cell to Markdown
   - `Y`: Change cell to Code
   - `D, D`: Delete selected cell
   - `Z`: Undo last cell deletion
   - `Shift + Up/Down`: Select multiple cells
   - `C`: Copy selected cell
   - `X`: Cut selected cell
   - `V`: Paste cell below
   - `Shift + V`: Paste cell above
   - `Ctrl + S`: Save notebook
2. **Edit Mode Shortcuts (Enter to enter):**
   - `Ctrl + Enter`: Run the current cell
   - `Shift + Enter`: Run the current cell and move to the next one
   - `Alt + Enter`: Run the current cell and insert a new one below
   - `Ctrl + /`: Toggle comment on selected line(s)

Using print(...) command, # and \ characters

- use command print(...) to print your output or comment(s)

```
print("Hello Dr. M R Karim")
print("Hello \n Dr. M R Karim")
print("Hello \n Dr. M R Karim! \n Is your mobile
    number 01912605556?")
print("Hello \n Dr. M R Karim! \t Is your mobile
    number 01912605556?")
print("Hello \n \" Dr. M R Karim!\" \t Is your mobile
    number 01912605556?")
```

- use of hash (#) symbol before single comment (or "'....'" for multiple comments to stop printing your comment(s)
- backslash (\) is used for making nice output (e.g. \n used for a new line)
- backslash character is also known as escape sequence

Using Python as a Calculator

- input and output are distinguished by the presence or absence of prompts (>>> and ···)
- expression syntax is straightforward: the operators $+, -, *, ()$ and / work just like in most other languages

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5 # division always returns a floating point number
1.6
>>> 17 / 3 # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3 # floor division discards the fractional part
5
>>> 17 % 3 # the % operator returns the remainder of the
    division
2
```


Using Python as a Calculator

1. use the ** operator to calculate powers

```
>>> 5 ** 2 # 5 squared
25
>>> 2 ** 7 # 2 to the power of 7
128
```

2. equal sign (=) is used to assign a value to a variable

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```


3. the last printed expression is assigned to the variable _

```
>>> price=110.0625
>>> a=2
>>> b=3
>>>a*b
>>> price + _ # Add the value of price to the result
    of the previous expression
116.0625
>>> round(_, 2)
116.06
```

Mathematical & Logical Operator

- < (less than)
- > (greater than)
- == (equal to)
- <= (less than or equal to)
- >= (greater than or equal to) and
- ! = (not equal to)
- logical operator: | (OR) and & (AND)

N. B.:

1. you have to type a tab or space(s) for each indented line in the body part of the program
2. when a compound statement is entered interactively, it must be followed by a blank line to indicate completion (since the parser cannot guess when you have typed the last line)

Symbol	Explanation/Note
+	Addition: Adds two numbers.
−	Subtraction: Subtracts the second number from the first.
*	Multiplication: Multiplies two numbers.
/	Division: Divides the first number by the second (returns a flo
//	Floor Division: Divides and returns the integer part (rounds do
%	Modulus: Returns the remainder of the division.
**	Exponentiation: Raises the first number to the power of the se
math.sqrt()	Square Root: Returns the square root of a number.
abs()	Absolute Value: Returns the absolute (non-negative) value of
round()	Round: Rounds a number to a specified number of decimal pl
math.log()	Logarithm: Returns the logarithm of the number with a specif

Operation	Result	Notes	Full documentation
`x + y`	sum of *x* and *y*		
`x - y`	difference of *x* and *y*		
`x * y`	product of *x* and *y*		
`x / y`	quotient of *x* and *y*		
`x // y`	floored quotient of *x* and *y*	(1)	
`x % y`	remainder of `x / y`	(2)	
`-x`	*x* negated		
`+x`	*x* unchanged		
`abs(x)`	absolute value or magnitude of *x*		`abs()`
`int(x)`	*x* converted to integer	(3)(6)	`int()`
`float(x)`	*x* converted to floating point	(4)(6)	`float()`
`complex(re, im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.	(6)	`complex()`
`c.conjugate()`	conjugate of the complex number *c*		
`divmod(x, y)`	the pair `(x // y, x % y)`	(2)	`divmod()`
`pow(x, y)`	*x* to the power *y*	(5)	`pow()`
`x ** y`	*x* to the power *y*	(5)	

Logical Operator

1. you can also make use of the logical operators | (OR) and & (AND)

```
import numpy as np
# Logical operator
x = np.array([1,3,4,5,6,7,8,2,3])
bigger_than_3 = (x > 3) | (x == 3)
print(bigger_than_3)
```

2. details can be found in
https://docs.python.org/3/library/stdtypes.html#dict

Strings in Python

1. strings can be enclosed in single quotes ('...') or double quotes ("...") with the same result
2. \ can be used to escape quotes

```
>>> 'spam eggs' # single quotes
'spam eggs'
>>> 'doesn\'t' # use \' to escape the single quote...
"doesn't"
>>> "doesn't" # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```


Strings in Python

1. the print() function produces a more readable output

```
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
>>> print('"Isn\'t," they said.')
"Isn't," they said.
>>> s = 'First line.\nSecond line.' # \n means newline
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s) # with print(), \n produces a new line
First line.
Second line.
```

2. if you don't want characters prefaced by backslash (\) to be interpreted as special characters, you can use raw strings by adding an *r* before the first quote:

```
>>> print('C:\some\name') # here \n means newline!
C:\some
ame
>>> print(r'C:\some\name') # note the r before the quote
C:\some\name
```

Strings in Python

1. strings can be concatenated (glued together) with the + operator, and repeated with ∗:

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

2. two or more string literals (i.e. the ones enclosed between quotes) next to each other are automatically concatenated

```
>>> 'Py' 'thon'
'Python'
>>> text = ('Put several strings within parentheses '
'to have them joined together.')
>>> text
'Put several strings within parentheses to have them
    joined together.'
```

Strings can be indexed!

- strings can be indexed (subscripted), with the first character having index 0

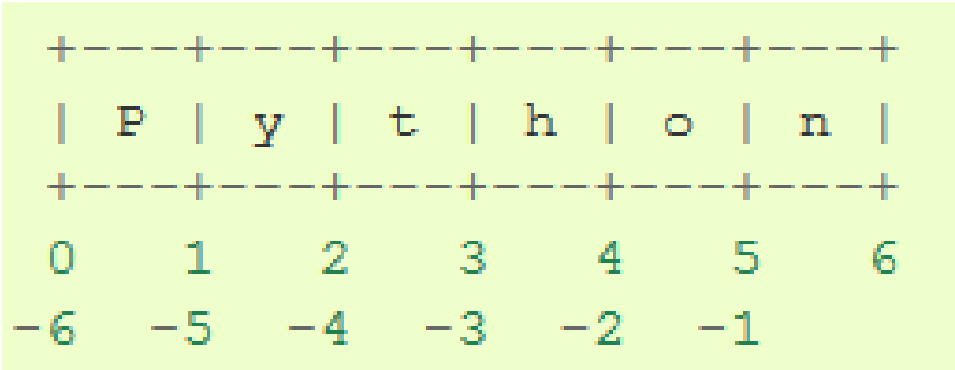
```
>>> word = 'Python'
>>> word[0] # character in position 0
'P'
>>> word[5] # character in position 5
'n'
```

- indices may also be negative numbers, to start counting from the right:

```
>>> word[-1] # last character
'n'
>>> word[-2] # second-last character
'o'
>>> word[-6]
'P'
>>> word[0:2] # characters from position 0 (included)
    to 2 (excluded)
'Py'
>>> word[2:5] # characters from position 2 (included)
    to 5 (excluded)
'tho'
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

- note that since -0 is the same as 0, negative indices start from -1

```
>>> word[:2]  # character from the beginning to position 2
    (excluded)
'Py'
>>> word[4:]  # characters from position 4 (included) to the
    end
'on'
>>> word[-2:] # characters from the second-last (included)
    to the end
'on'
```

1. the most versatile is the list, which can be written as a list of comma-separated values (items) between square brackets

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
>>> squares[0] # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:] # slicing returns a new list
[9, 16, 25]
>>> squares[:]
[1, 4, 9, 16, 25
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```


1. you can also add new items at the end of the list, by using the append() method

```
>>> cubes=[1, 8, 27, 64, 125]
>>> cubes.append(216) # add the cube of 6
>>> cubes.append(7 ** 3) # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

2. the built-in function len() also applies to lists:

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
```

True/False Logic

```
>>> saarc = ["Bangladesh", "Afghanistan", "Bhutan", "Nepal",
    "India", "Pakistan", "Sri Lanka"]
>>> print(saarc)
['Bangladesh', 'Afghanistan', 'Bhutan', 'Nepal', 'India',
    'Pakistan', 'Sri Lanka']
>>> print(saarc[0])
Bangladesh
>>> print("Number of countries in SAARC:", len(saarc))
Number of countries in SAARC: 7
>>> "Bangladesh" in saarc
True
>>> "China" in saarc
False
>>> "China" not in saarc
True
>>> "India" not in saarc
False
```

Built-in Data Types

Python has the following data types built-in by default, in these categories:

1. Text type: `str`
2. Numeric Types: `int`, `float`, `complex`
3. Sequence Types: `list`, `tuple`, `range`
4. Mapping Type: `dict`
5. Boolean Type: `bool`
6. Binary Types: `bytes`, `bytearray`, `memoryview`

Example	Data Type	Example	Data Type
x = "Hello World"	str	x = {"name" : "John", "age" : 36}	dict
x = 20	int	x = {"apple", "banana", "cherry"}	set
x = 20.5	float	x = frozenset({"apple", "banana", "cherry"})	frozenset
x = 1j	complex	x = True	bool
x = ["apple", "banana", "cherry"]	list	x = b"Hello"	bytes
x = ("apple", "banana", "cherry")	tuple	x = bytearray(5)	bytearray
x = range(6)	range	x = memoryview(bytes(5))	memoryview

```
print(type(x))
```

Example: Variable Type

```
>>> a = 1
>>> type(a)
<class 'int'>
>>> b = 3.1416
>>> type(b)
<class 'float'>
>>> c = "Hello World!"
>>> type(c)
<class 'str'>
>>> d = True
>>> type(d)
<class 'bool'>
```


Differences between Lists, Tuples, Dictionaries, and Sets in Python

List:

- Ordered collection of elements
- Mutable
- Elements accessed by index
- Defined with square brackets []
- Can contain duplicates

Tuple:

- Ordered collection of elements
- Immutable
- Defined with parentheses ()
- Can contain duplicates

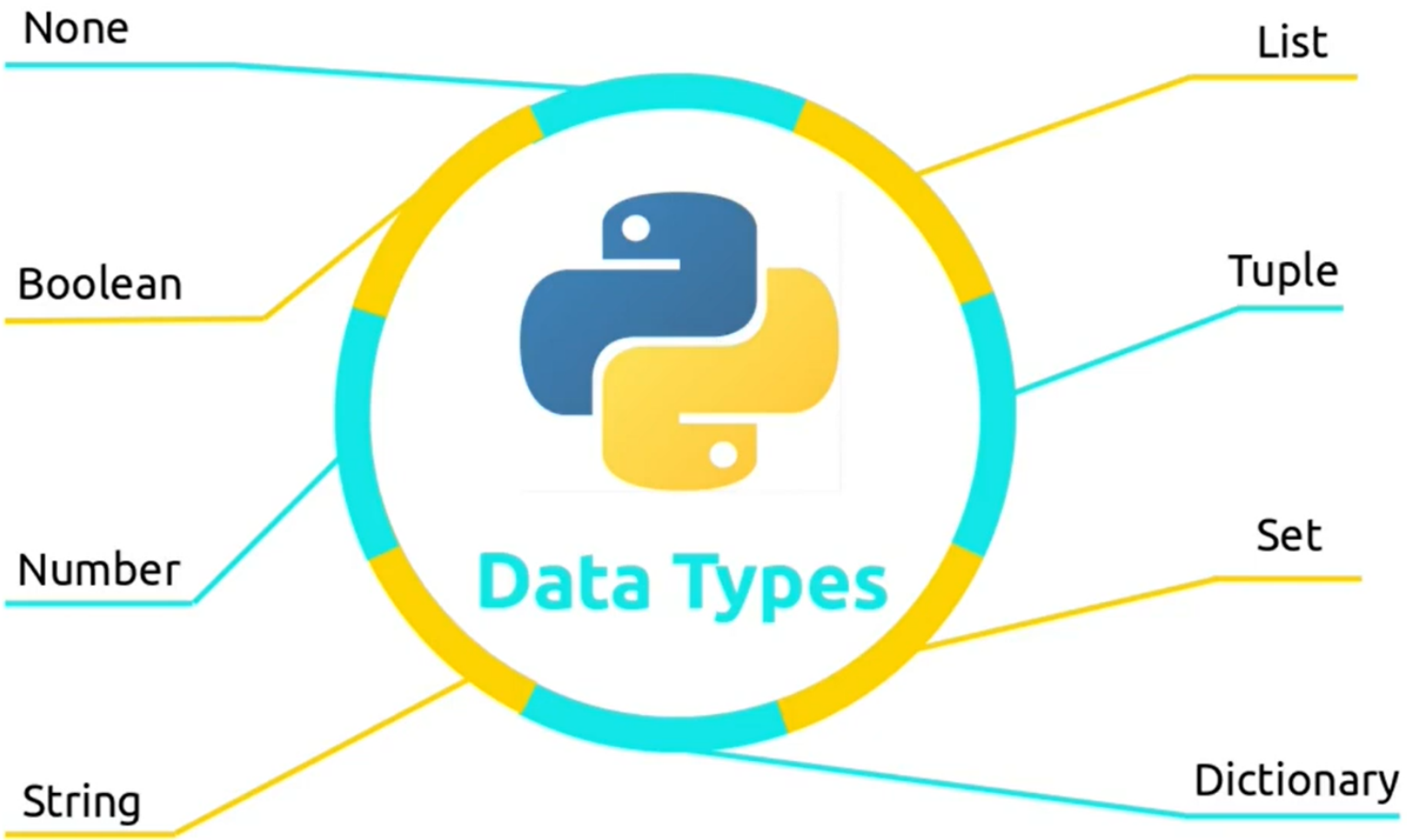
Dictionary:

- Unordered collection of key-value pairs
- Mutable
- Keys are unique and immutable
- Defined with curly braces {}

Set:

- Unordered collection of unique elements
- Mutable
- Defined with curly braces {} or set()

Data Type and String Method



Understanding List Assignment in Python

Code:

```
# Define Position
x = [1, 2, 3] # Step 1
y = x         # Step 2
y[0] = 10     # Step 3
print(x)      # Step 4
```

Explanation:

- **Step 1:** `x = [1, 2, 3]`
  Creates a list `x` with elements {1, 2, 3}.
- **Step 2:** `y = x`
  Assigns `y` to reference the same list object as `x`. `y` is not a copy of `x`, but a reference to the same list.

- **Step 3:** `y[0] = 10`
  Modifies the first element of the list via `y`. Since `y` and `x` refer to the same list, the modification affects both.
- **Step 4:** `print(x)`
  Prints the modified list, which is now {10, 2, 3}.
  Output:

```
[10, 2, 3]
```

String - method

1	capitalize()	9	center()	17	splitlines()	25	istitle()
2	title()	10	ljust()	18	partition()	26	isupper()
3	upper()	11	rjust()	19	swapcase()	27	isnumeric()
4	casefold()	12	count()	20	endswith()	28	isdigit()
5	lower()	13	strip()	21	startswith()	29	isdecimal()
6	join()	14	lstrip()	22	zfill()	30	isalpha()
7	find()	15	rstrip()	23	replace()	31	isalnum()
8	rfind()	16	split()	24	islower()	32	isspace()

String - method

1. maketrans()
2. translate()

Chapter 3: First Steps Towards Programming

3 Chapter 3: First Steps Towards Programming

Rules for Assigning Variables in Python

1. **Basic Assignment**: Use the = operator to assign a value to a variable.

```
x = 10
name = "Alice"
is_active = True
```

2. **Variable Naming Rules**
   - Variable names are case-sensitive (myVar and myvar are different).
   - The rest can contain letters, digits, or underscores.
3. Variable names must start with a letter (a-z, A-Z) or an underscore (_)

```
valid_name = 100
_hidden_variable = "secret"
var_1 = 5
user2 = "Bob"
```

4. **Never use "Keyword" as a variable name**

```
# Invalid
class = "biology"
return = 50

# Valid alternative
class_name = "biology"
return_value = 50
```

5. **Descriptive Names**: Use descriptive names that make the purpose of the variable clear.

```
# Bad
a = "John"
b = 30

# Good
first_name = "John"
user_age = 30
```

Keywords in Python programming language				
False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

6. **Multiple Assignments**: You can assign values to multiple variables in one line.

```
a, b, c = 1, 2, 3
name, age, is_active = "Alice", 30, True
```

7. **Swapping Values**: You can swap values of two variables without a temporary variable.

```
x, y = y, x
```

8. **Type Consistency**: While Python is dynamically typed (you don't declare variable types), it's good practice to use variables consistently.

```
count = 10  # integer
count = "ten" # avoid changing type, though it's allowed
```

9. **Constant Naming**: Use all uppercase letters for variable names that are meant to be constants.

```
MAX_USERS = 100
PI = 3.14159
```

10. **Underscore for Ignoring Values**: Use an underscore _ to ignore specific values during assignment.

```
_, b, _ = 1, 2, 3 # only 'b' is assigned
```

11. **Type Hints (Optional)**: You can use type hints to indicate the expected data type for a variable (optional).

```
age: int = 25
name: str = "Alice"
active: bool = True
```

How to define Variable(s) with print command in Python?

```
print("Our new student name is Mr. Hassan")
print("Mr. Hassan lives in Dhaka")
print("He is currently 24 years old")
print("At the age of 24, he has started to learn Python")
print("Mr. Hassan has scored 3.95 in his Secondary exam")

## Defining Variable
name="Mr. Hassan"
age=24
score=3.95
print("Our new student name is " + name)
print(name + " lives in Dhaka")
print("He is currently", age,"years old")
print("At the age of", age,", he has started to learn
    Python")
print(name+"has scored", score,"in his Secondary exam")
```

Multiple Variables with print command

- Assign value to multiple variables

```
# Example: 1
x,y,z="Md.","Rezaul","Karim"
print(x)
print(y)
print(z)
# Exaple: 2
x=y=z="Md. Rezaul Karim"
print(x)
print(y)
print(z)
```


input function

- Python has an input function which lets you ask a user for some text input
- you call this function to tell the program to stop and wait for the user to key in the data

```
beautiful_number = input("tell me a beautiful number ")
print("I am very happy to see your beautiful
    number",beautiful_number,".")
## Defining Variable and Input
name=input("Enter your name: ")
age=input("Enter your age: ")
score=input("Enter your GPA score: ")
print("Student Information")
print("--------------------------")
print("Our new student name is " + name)
print(name + " lives in Dhaka")
print("He is currently", age,"years old")
print("At the age of", age,", he has started to learn
    Python")
print(name+"has scored", score,"in his Secondary exam")
```

Type Casting or Type Conversion

What is wrong with the following code?

```
# Type Casting
num1=input("Enter first number: ")
num2=input("Enter second number: ")
sum=num1+num2
print(sum) # wrong result
print("the sum is"+ sum) # no error!!!
```



```
[42] # Type Casting
     num1=input("Enter first number: ")
     num2=input("Enter second number: ")
     sum=num1+num2
     print(sum) # wrong result
```

```
Enter first number: 2
Enter second number: 3
23
```

- The code is providing incorrect results because the variables num1 and num2 are being treated as strings, not as numerical values. When you use the `input()` function in Python, it always returns a string, even if the user inputs a number.
- In the line `sum=num1+num2`, Python concatenates the strings stored in num1 and num2, rather than performing addition. This behavior is known as string concatenation.
- To fix this issue, you need to convert the input strings to numerical values before performing arithmetic operations. This process is known as type casting.

How to correct this code?

```
sum=int(num1)+int(num2)
print(sum) # correct result
print("the sum is"+sum) # error result. why?
print("the sum is",sum) # modified correct result

# ignore writting int in each time
num1=int(input("Enter first number: "))
num2=int(input("Enter second number: "))
sum=num1+num2
print(sum) # correct result
```

Format

To format the output in Python, you can use various methods such as string formatting, f-strings, or the `format()` method.

```
## Output Format
num1=20
num2=30
# f use for string in formating
# {} use for number
print(f"{num1}+{num2}={num1+num2}")
```

- In Python, the `print()` function by default ends with a newline character

  `(\n)`

  which means each subsequent call to `print()` will start printing on a new line.

```
## output in two lines
print("Dr. Md. Rezaul Karim")
print("01912605556")

# output in one line
print("Dr. Md. Rezaul Karim.", end=" ")
print("Mobile Number: 01912605556")
```

- However, `end=" "` tells Python to use a space character instead of a newline character as the ending character for the output of the `print()` function.

```
#String Formatting:
print("Name: {}. Mobile Number: {}".format("Dr. Md. Rezaul
    Karim", "01912605556"))
name = "Dr. Md. Rezaul Karim"
mobile_number = "01912605556"
print(f"Name: {name}. Mobile Number: {mobile_number}")
```


- "Name: {}. Mobile Number: {}": This is a format string that serves as a template for the output. It contains placeholders {} where values will be inserted.
- Inside the format string, there are two placeholders {}. These will be replaced by the values provided in the .format() method.

Homework-1

1. Write the Python for calculating the area of triangle.
2. Write the Python for calculating the area square.
3. Write the Python for calculating the area of circle.

```
## Area calcuation for circle
radious=float(input("Inter Radious = "))
area=3.14*radious*radious
print(area)
# or
import math
radious=float(input("Inter Radious = "))
area=math.pi*radious*radious
print(area)
```

Rules for data reading in Python

1. Provide the correct file path.
2. Specify the file format (e.g., CSV, Excel).
3. Check for headers (column names) and handle them appropriately.
4. Specify the delimiter for delimited files (e.g., CSV).

```
import pandas as pd
# Define the file path
file_path = r'C:\Users\Admin\OneDrive\Lecture Slides\Machine
    Learning\data\Breast Cancer Wisconsin.csv'
# Read the CSV file into a DataFrame
data = pd.read_csv(file_path)
# Print the entire DataFrame
print(data)
# Display the first few rows of the DataFrame
print(data.head())
# Display Descriptive statistics
data.describe()
```

```
In [88]: data.describe()
```

Out[88]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	s
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	

8 rows × 32 columns

Central Tendency in Python

```
mean_radius_mean = data['radius_mean'].mean()

# Print the mean value for the "radius_mean" variable
print("Mean of radius_mean:", mean_radius_mean)

# Calculate the median for the "radius_mean" variable
median_radius_mean = data['radius_mean'].median()

# Calculate the mode for the "radius_mean" variable
mode_radius_mean = data['radius_mean'].mode()
# Print the median and mode for the "radius_mean" variable
print("Median of radius_mean:", median_radius_mean)
print("Mode of radius_mean:", mode_radius_mean)
```

Dispersion in Python

```
# Calculate the variance for the "radius_mean" variable
variance_radius_mean = data['radius_mean'].var()

# Calculate the standard deviation for the "radius_mean" variable
std_deviation_radius_mean = data['radius_mean'].std()

# Calculate the coefficient of variation (CV) for the
    "radius_mean" variable
cv_radius_mean = (std_deviation_radius_mean /
    data['radius_mean'].mean()) * 100

print("Variance of radius_mean:", variance_radius_mean)
print("Standard deviation of radius_mean:",
    std_deviation_radius_mean)
print("Coefficient of variation (CV) of radius_mean:",
    cv_radius_mean)
```

Chapter 4: Python Conditions and If statements

4 Chapter 4: Python Conditions and If statements

Let's discuss the different types of `if` statements in Python, which are used for decision-making to execute code conditionally.

Indentation in Python

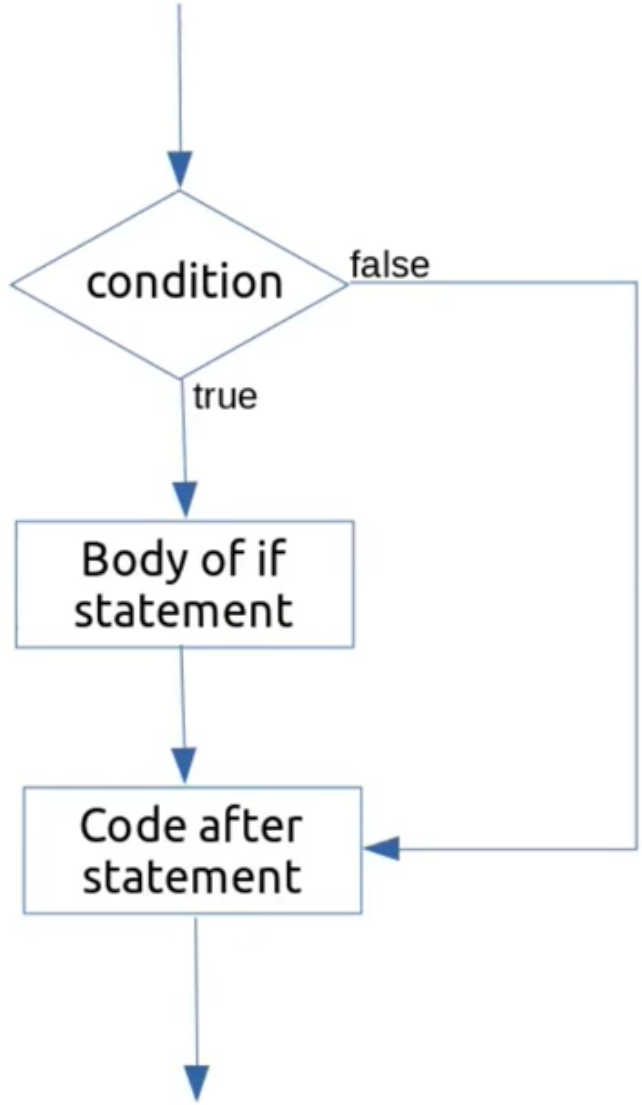
Indentation in Python is used to define the scope and structure of code blocks. Unlike many other programming languages that use braces to delimit code blocks, Python relies on indentation levels to group statements. Proper indentation is crucial for defining the body of loops, conditionals, functions, and other structures.

Why indentation is important in Python?

- **Defining Code Blocks:** Indentation is used to define the body of loops, functions, classes, and conditional statements.
- **Readability:** Proper indentation improves the readability of the code, making it easier to understand the structure and flow of the program.
- **Syntax Requirement:** Incorrect indentation leads to syntax errors. Python will throw an `IndentationError` if the indentation is not consistent.

If - statement

```
#------code------
if condition:
    #condition is true
    #write your code
#------code------
```



Symbols Used In Flowchart

start/stop

processing

input/output

decision

flow line

if statement

- if statement: The basic if statement executes a block of code if a specified condition is True.
- Example-1:

```
x = 10
if x > 5:
  print("x is greater than 5")
```


- Example-2:

```
### if statement
saarc = ["Bangladesh", "Afghanistan", "Bhutan",
    "Nepal", "India", "Pakistan", "Sri Lanka"]

country = input("Enter the name of the country: ")
if country in saarc:
  print(country, "is a member of SAARC")
print("Program terminated")
```

if-else statement

- if-else statement: The if-else statement executes one block of code if the condition is True, and another block of code if the condition is False.
- Example-1:

```
x = 3
if x > 5:
  print("x is greater than 5")
else:
  print("x is not greater than 5")
```

- Example-2:

```
# if and else statement
saarc = ["Bangladesh", "Afghanistan", "Bhutan",
    "Nepal", "India", "Pakistan", "Sri Lanka"]
country = input("Enter the name of the country: ")
if country in saarc:
  print(country, "is a member of SAARC")

else:
  print(country, "is not a member of SAARC")

print("Program terminated")
```

What is Indentation?

- Indentation refers to the spaces at the beginning of a line of code.
- It is used to define blocks of code in Python.
- Unlike other languages that use '{}' brackets, Python enforces indentation.
- Improper indentation will result in an IndentationError.

Why is Indentation Important?

- It improves code readability.
- It defines logical structure in control flow (if, loops, functions, classes).
- It is a strict requirement in Python (no indentation = error!).

Rules of Indentation in Python

1. Use 4 spaces per indentation level (PEP 8 standard).
2. Do not mix tabs and spaces.
3. All statements inside control structures (if, loops, functions) must be indented.
4. Nested blocks must be indented further than the parent block.
5. Maintain consistent indentation throughout the code.

Correct Indentation Example

```
if True:
  print("Hello, Python!") # Indented properly

def greet():
  print("Welcome!")

greet()
```


Incorrect Indentation Example

```
if True:
print("Hello, Python!") # IndentationError
```

Error: IndentationError: expected an indented block

Nested Indentation Example

```
def check_age(age):
  if age >= 18:
    print("Adult")
    if age >= 65:
      print("Senior Citizen")
  else:
    print("Minor")

check_age(70)
```


Best Practices for Indentation

- Always use 4 spaces per indentation level.
- Configure your editor to insert spaces instead of tabs.
- Follow PEP 8 guidelines for clean and readable code.
- Use a linter to detect indentation errors.

PEP 8 Guidelines for Python Code

PEP 8 (Python Enhancement Proposal 8) is the official style guide for Python code. It provides guidelines for writing readable, consistent, and maintainable Python programs.

- PEP 8 is the Python Enhancement Proposal that defines the coding style for Python code.
- It aims to improve the readability and consistency of Python code across projects.
- It covers indentation, line length, naming conventions, and other style recommendations.

Key Recommendations of PEP 8

1. **Indentation:** Use 4 spaces per indentation level. Avoid using tabs.
2. **Line Length:** Limit all lines to a maximum of 79 characters.
3. **Blank Lines:** Separate functions, classes, and code blocks with blank lines. Use 2 blank lines.
4. **Imports:**
   - place imports at the top of the file.
   - Imports one module per line.
   - Standard library imports first, followed by third-party imports, then local imports.
5. **Whitespace:** Avoid unnecessary whitespace in expressions and statements. Avoid extra space inside parentheses, brackets, or braces.

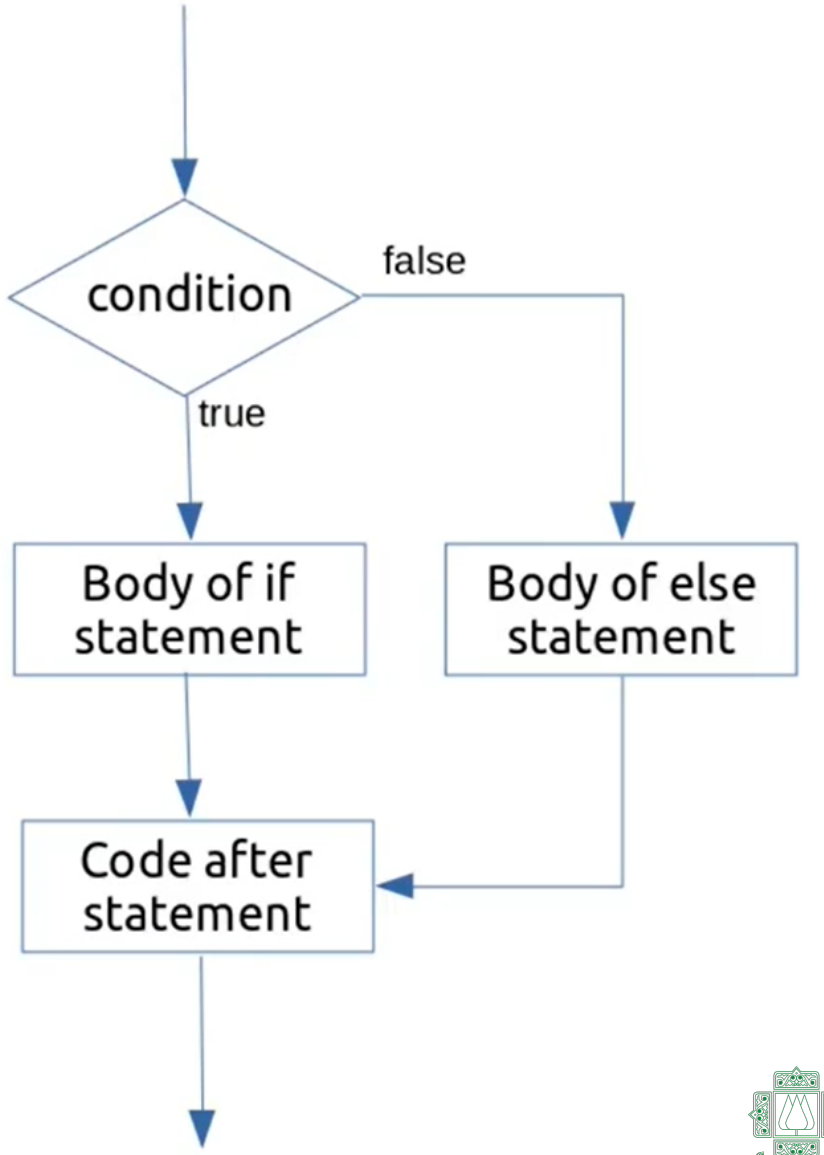
6. **Naming Conventions**
   - **Variable names:** Use lowercase with words separated by underscores (`snake_case`).
   - **Function names:** Use lowercase with words separated by underscores (`snake_case`).
   - **Class names:** Use capitalized words with no underscores (`CamelCase`).
   - **Constants:** Use all uppercase with words separated by underscores (`UPPERCASE`).
7. **Docstrings:**
   - Use triple quotes (`"""docstring"""`) for all public modules, functions, and classes.
   - Describe the purpose and behavior of the function or class.
8. **Comments:**
   - Comments should be complete sentences.
   - Use block comments and use # for inline comments and triple quotes for docstrings.
   - Leave at least two spaces between code and inline comments.

Why PEP 8 Matters?

- Improves the readability of the code, making it easier to understand and maintain.
- Helps standardize code across different projects and teams.
- Facilitates better collaboration and code sharing.

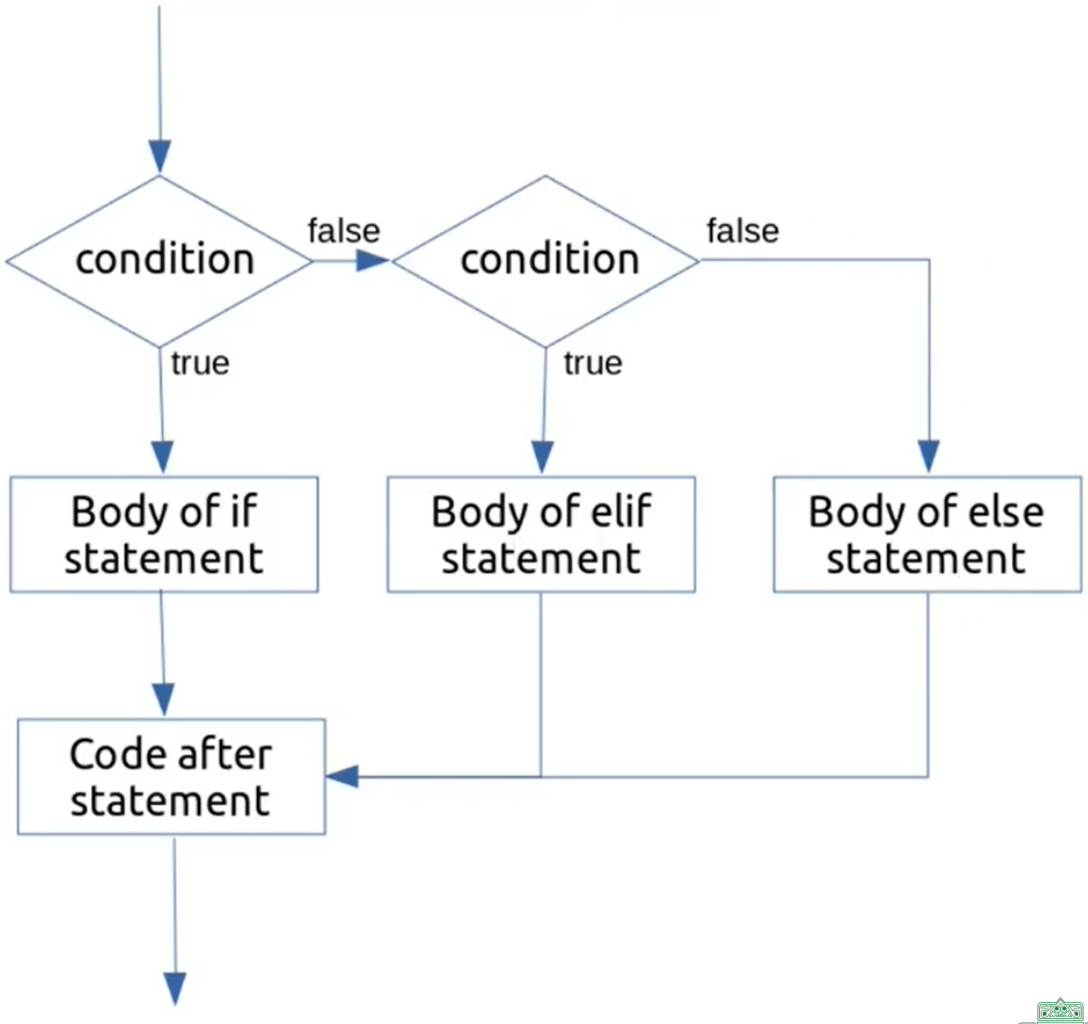
else - statement

```
#------code------
if condition:
        #condition is true
        #write your code
else:
        #condition is false
        #write your code
#------code------
```



elif / multiple - statement

```
#------code------
if condition:
        #condition is true
        #write your code
elif condition:
        #condition is true
        #write your code
else:
        #condition is false
        #write your code
#------code------
```



if-elif-else statement

- if-else statement

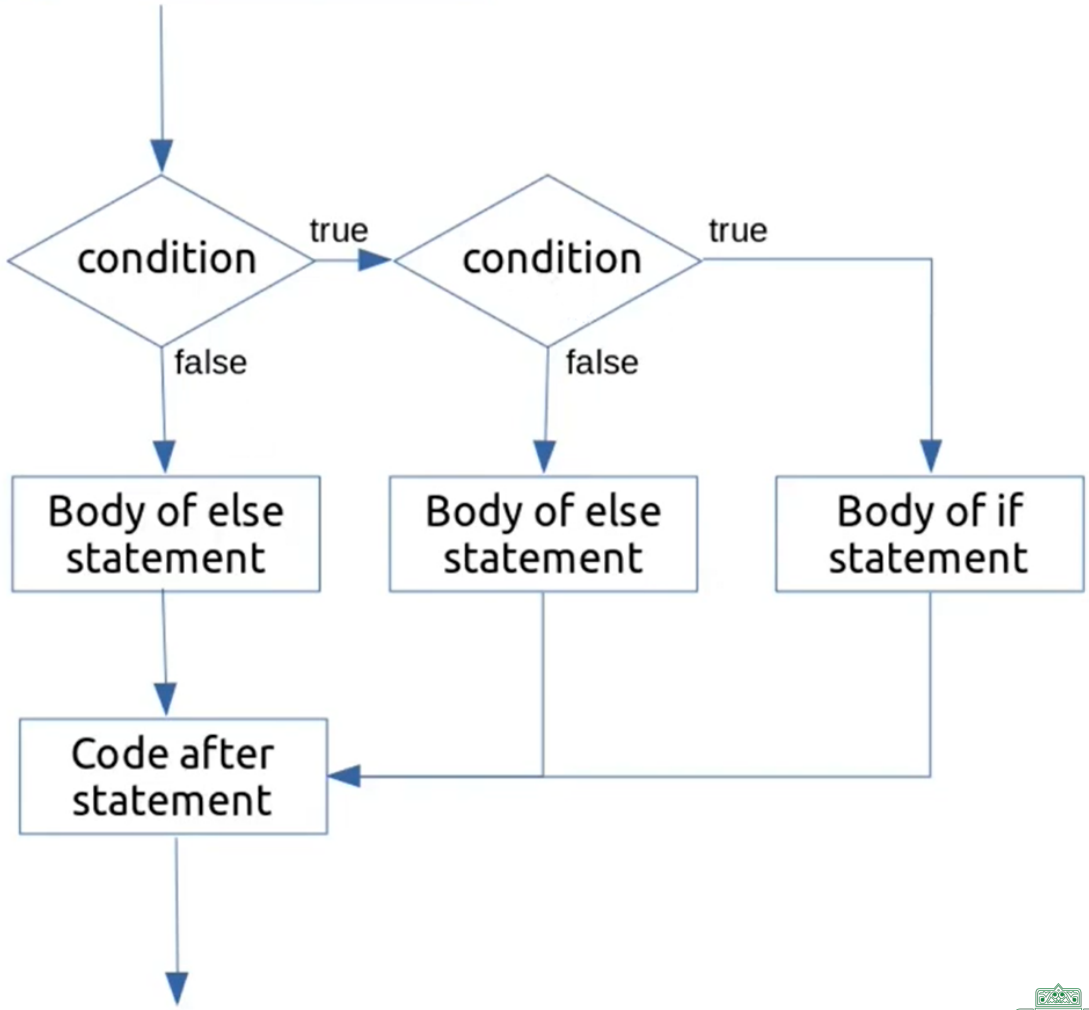
```
## if-elif-else statement
marks = input("Please enter your marks: ")
marks = int(marks)
if marks >= 80:
  grade = "A+"
elif marks >= 70:
  grade = "A"
elif marks >= 60:
  grade = "A-"
elif marks >= 50:
  grade = "B"
else:
  grade = "F"
print("Your grade is", grade)
```


Classroom Exercise: Grade Calculator

- Write a Python program that takes input from the user for a test score and outputs the corresponding grade.
- Grading scale:
  - A: 90-100
  - B: 80-89
  - C: 70-79
  - D: 60-69
  - F: Below 60

Nested if - statement

```
#-----code------
if condition:
        #condition is true
        #write your code
        if condition:
                #condition is true
                #write your code
        else:
        #condition is false
        #write your code
else:
        #condition is false
        #write your code
#-----code------
```



Nested if-else statement (Program to Check Leap Year)

- A leap year is exactly divisible by 4 except for century years (years ending with 00). The century year is a leap year only if it is perfectly divisible by 400. For example, 2017 is not a leap year, 1900 is a not leap year, 2012 is a leap year, 2000 is a leap year

```
## Example: Python Program to Check Leap Year
year = int(input("Please enter year:"))
if year % 4 != 0:
  print("No,", year, "is not a leap year")
else:
  if year % 100 == 0:
    if year % 400 == 0:
      print("Yes,", year, "is a leap year")
    else:
      print("No,", year, "is not a leap year")
    else:
      print("Yes,", year, "is a leap year")
```

if-elif-else statement

- alternatively: Python Program to Check Leap Year

```
## if-elif-else statement
# Alternativee code for leaf year
year = int(input("Please enter year:"))
if year % 100 != 0 and year % 4 == 0:
  print("Yes,", year, "is a leap year")
elif year % 100 == 0 and year % 400 == 0:
  print("Yes")
else:
  print("No,", year, "is not a leap year")
```

Classroom Exercise: Leap Year Checker

- Write a Python program that takes a year as input and determines whether it is a leap year or not.
- If it's a leap year, print "Leap year".
- Otherwise, print "Not a leap year".
- Hint: A leap year is divisible by 4, but not by 100 unless it is also divisible by 400.

Homework-2

1. **Exercise 1: Temperature Converter**
   - Write a Python program that takes input from the user in Celsius and converts it to Fahrenheit.
   - If the converted temperature is above 100°F, print "Hot day".
   - If it's between 50°F and 100°F, print "Moderate day".
   - If it's below 50°F, print "Cold day".
2. **Exercise 2: Guessing Game**
   - Write a Python program that generates a random number between 1 and 100 and asks the user to guess it.
   - If the user's guess is higher than the actual number, print "Too high".
   - If it's lower, print "Too low".
   - If it's correct, print "Congratulations!".

3. Exercise 3: BMI Calculator
   - Write a Python program that calculates the Body Mass Index (BMI) of a person.
   - Take input for height (in meters) and weight (in kilograms) from the user.
   - Classify the BMI according to the following criteria:
     - Underweight: BMI < 18.5
     - Normal weight: $18.5 \leq$ BMI < 25
     - Overweight: $25 \leq$ BMI < 30
     - Obese: BMI ≥ 30
4. Write a Python program that prompts the user to choose between calculating the area of a triangle or a circle. For a triangle, ask for the base and height. For a circle, ask for the radius. Compute and display the chosen area.

Chapter 5: Loops in Python – For, While and Nested Loops

5 Chapter 5: Loops in Python – For, While and Nested Loops

Let's discuss loops in Python in an easy way, focusing on the two main types: `for` loops and `while` loops. Then we will focus on nested loops.

Loop in Python

In Python, loops are essential for iterating over sequences, performing repetitive tasks, and controlling program flow. Some of the important loops used in Python code include:

- "for" loop
- "while" loops
- Nested Loops:
  - "break" statement
  - "continue" statement
  - "pass" statement

range function

- The range() function generates a sequence of numbers.
- Commonly used in for loops for iteration.
- Memory efficient and immutable.

Syntax of range()

- range(stop)
- range(start, stop)
- range(start, stop, step)

Example

```
range(5)        # Generates 0, 1, 2, 3, 4
range(2, 7)     # Generates 2, 3, 4, 5, 6
range(1, 10, 2) # Generates 1, 3, 5, 7, 9
```

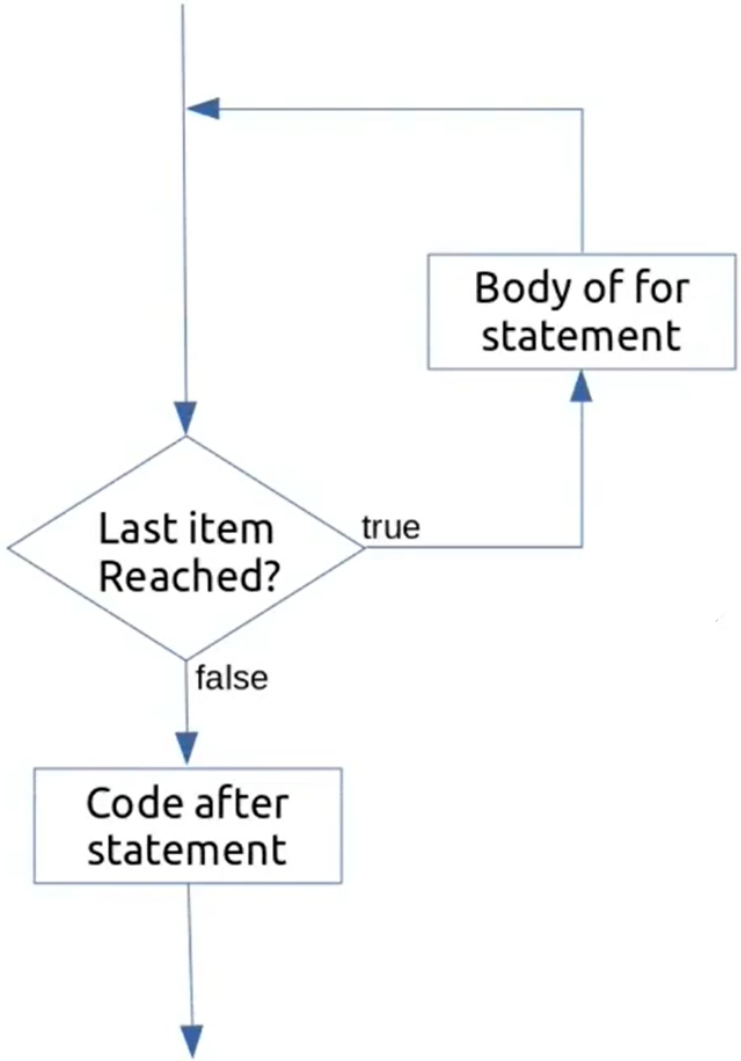
range Function

- the given end point is never part of the generated sequence; range(10) generates 10 values, the legal indices for items of a sequence of length 10.

```
range(5)
print(range(5))
# output: range(0, 5)
# that is  0, 1, 2, 3, 4
range(5, 10)
# that is 5, 6, 7, 8, 9

range(0, 10, 3)
# that is 0, 3, 6, 9

range(-10, -100, -30)
# that is -10, -40, -70
```

for - loop

```
#------code------
for variable in sequence:
      #item reached
      #write your code
#------code------
```



for statement and range() Function

- if you do need to iterate over a sequence of numbers, the built-in function range() comes in handy

```
for i in range(5):
  print(i)
# output
0
1
2
3
4

## Example
for i in range(10):
  print("I want to be a great programmer.")
```


Loop by using for statement

- to iterate over the indices of a sequence, you can combine range() and len() as follows:

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
  print(i, a[i])
```

```
# Output
0 Mary
1 had
2 a
3 little
4 lamb
```

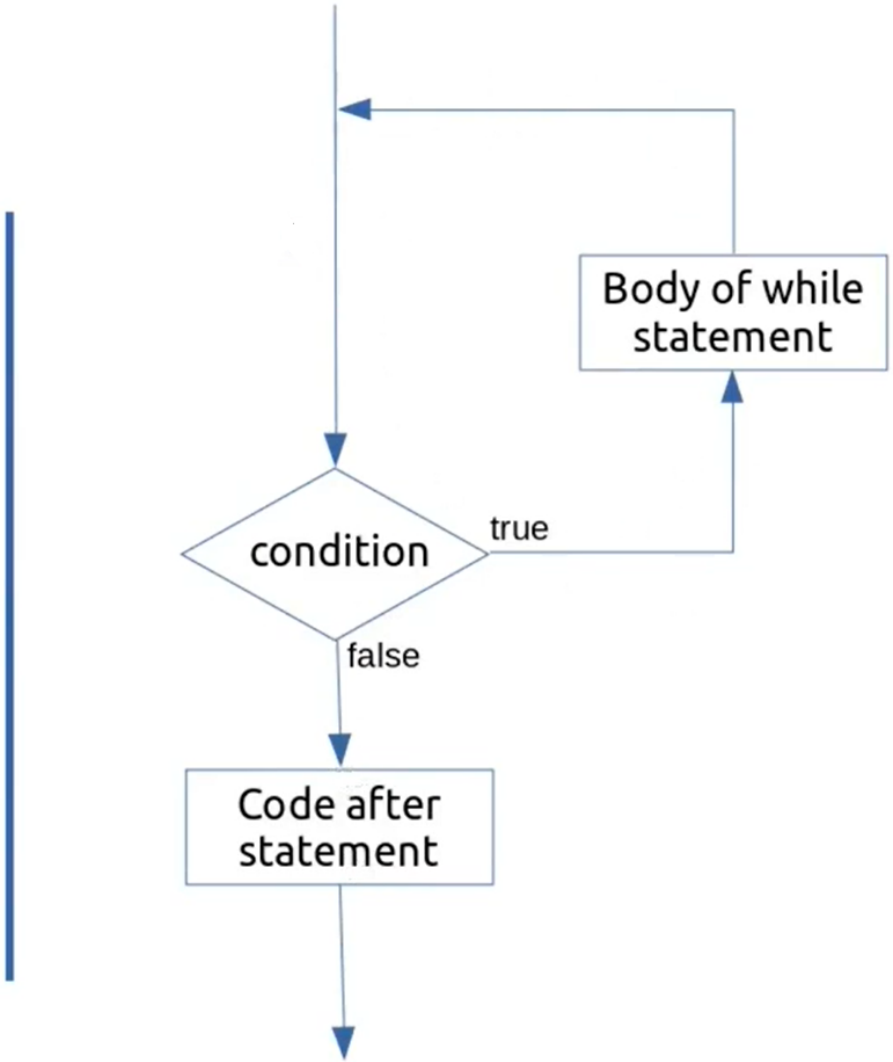
```
## Example
saarc = ["Bangladesh", "Afghanistan", "Bhutan", "Nepal",
    "India", "Pakistan", "Sri Lanka"]
```


while Loop

- The while loop is a fundamental control structure for repeated execution.
- Ensure loop conditions and increments are correctly implemented to avoid infinite loops.
- Practice with various examples to solidify understanding.
- while loop is useful for situations where the number of iterations is not known beforehand.

While - loop

```
#------code------
while condition:
      #condition is true
      #write your code
#------code------
```



while statement

- you can use Python for more complicated tasks than adding two and two together
- For instance, we can write an initial sub-sequence of the Fibonacci series as follows:

```
# Fibonacci series:
# the sum of two elements defines the next
a, b =0,1
while a<10:
  print(a)
  a,b=b,a+b

# output
0
1
1
2
3
5
8
```


while statement

- Make a Multiplication table

```
# Printing the Multiplication Table
n = input("Please enter a positive integer: ")
n = int(n)
m = 1
while m <= 10:
  print(n, "x", m, "=", n*m)
  m += 1 #This line increases the value of m by 1.
```

```
# Search for a number in a list and stop when found

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
search_for = 5

for number in numbers:
 if number == search_for:
   print("Number found!")
   break # Exit the loop immediately when the number is found
print("Loop finished.")
```


```
# Search for a number in a list and stop when found

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
search_for = 5
position = None

for index, number in enumerate(numbers):
 if number == search_for:
   position = index # Store the position of the number
   print("Number found at position:", position)
   break  # Exit the loop immediately when the number is found

print("Loop finished.")
```

continue statement

- Find the even number

```
# Example using continue statement
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Print only odd numbers from the list
for num in numbers:
  if num % 2 == 0:
    continue # Skip even numbers
  print(num)
```


- Find the numbers from 1 to 10, skipping multiples of 3

```
# Print numbers from 1 to 10, skipping multiples of 3

for i in range(1, 11):
if i % 3 == 0:
  continue # Skip the current iteration if i is a
      multiple of 3
print(i)
```

pass statement

The pass statement in Python is used as a placeholder for future code. It allows you to create an empty code block that does nothing when executed, which can be useful in various situations. Here are some common scenarios where you might use the pass statement:

```
# Check if numbers in a list are positive, negative, or zero
numbers = [10, -3, 0, 7, -1, 5]

for number in numbers:
  if number > 0:
    print(number, "is positive")
  elif number < 0:
    print(number, "is negative")
  else:
    pass  # Placeholder for future code

print("Finished checking numbers.")
```

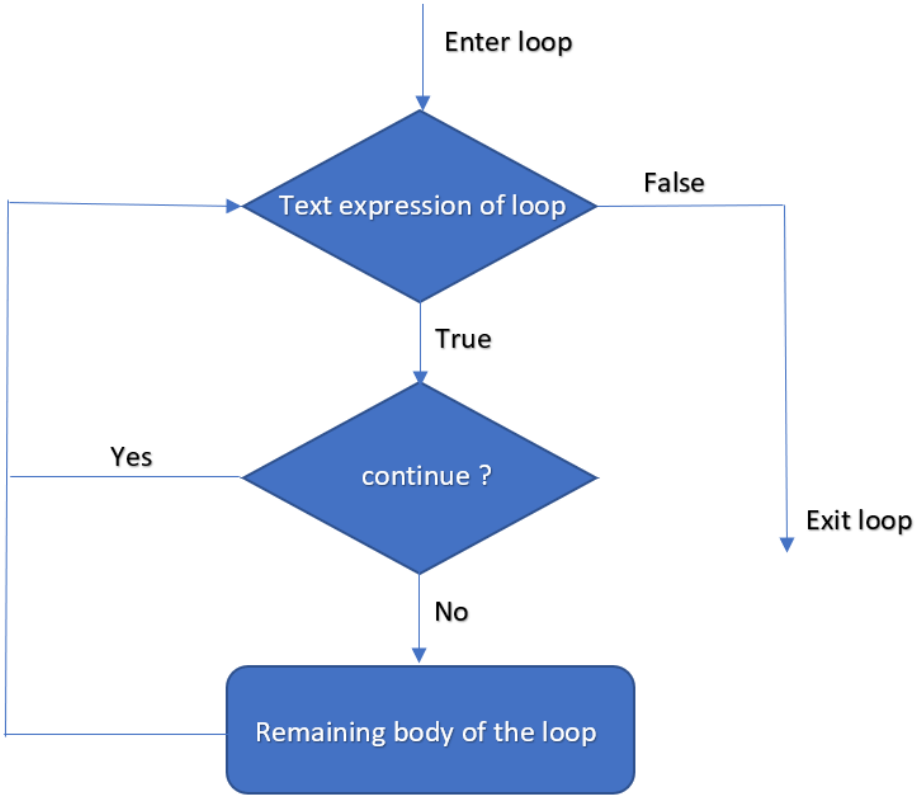

Differences Between Pass and Continue in Python

```
# Python program to demonstrate
# difference between pass and
# continue statements

s = "geeks"

# Pass statement
for i in s:
if i == 'k':
print('Pass executed')
pass
print(i)

print()

# Continue statement
for i in s:
if i == 'k':
```

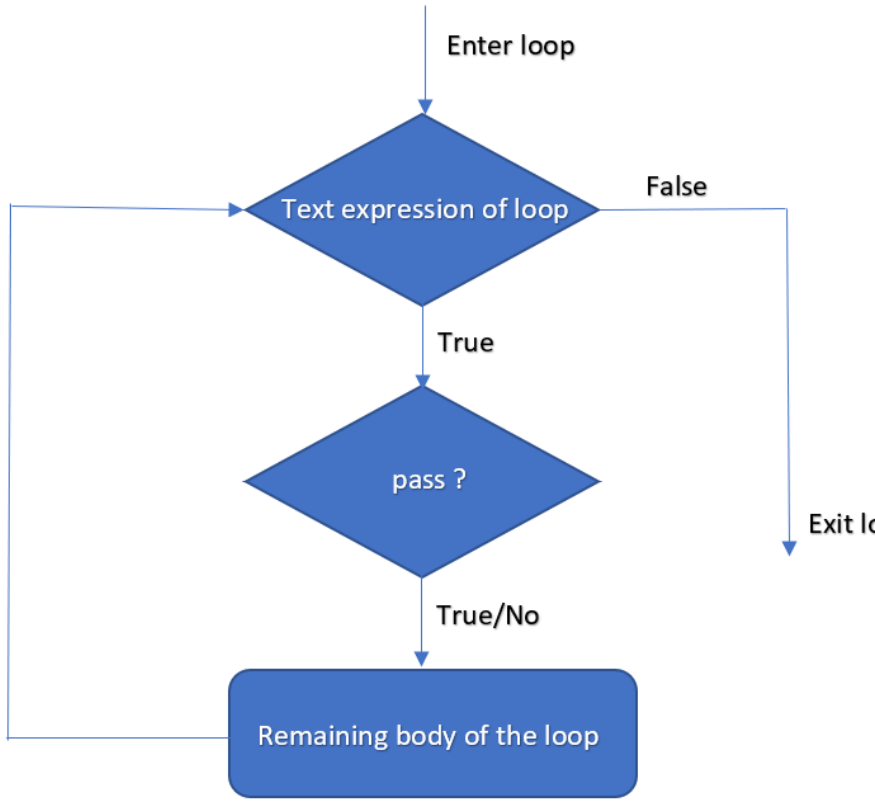

Output:

```
g
e
e
Pass executed
k
s

g
e
e
Continue executed
s
```

Continue



Pass



- pass and continue statements are not interchangeable in Python. A pass statement signals to a loop that there is "no code to execute here." It's a placeholder for future code. A continue statement is used to force the loop to skip the remaining code and start the next iteration.

```
for num in range(0,5):
pass
print(f'Iteration: {num}')
print("This statement is after 'pass'")
continue
print("This statement is after 'continue'")
```

Double loop and break statement

Code for the finding prime numbers

- the break statement, like in C, breaks out of the innermost enclosing for or while loop.

```
for n in range(2, 10):
  for x in range(2, n):
    if n % x == 0:
      print(n, 'equals', x, '*', n // x)
      break
  else:
  # Loop fell through without finding a factor
    print(n, 'is a prime number')

    # output
    2 is a prime number
    3 is a prime number
    4 equals 2 * 2
    5 is a prime number
    6 equals 2 * 3
    7 is a prime number
    8 equals 2 * 4
    9 equals 3 * 3
```

Expiation of this Code

- **Outer Loop:** `for n in range(2, 10):` iterates over numbers from 2 to 9 (inclusive). These numbers will be checked for primality.
- **Inner Loop:** `for x in range(2, n):` iterates over numbers from 2 to n-1. Each x is a potential divisor of n.
- **Prime Check:** if `n % x == 0:` checks if n is divisible by x. If it is, n is not a prime number, and the loop breaks.
- **Else Block:** The `else` block after the inner `for` loop executes only if the loop completes without encountering a `break` statement, indicating that n is a prime number.
- **Output:** If n is found to be divisible by any number x in the range of 2 to n-1, it prints the factorization of n. Otherwise, if no divisor is found, it prints that n is a prime number.

list()

The list() function creates a list object. A list object is a collection which is ordered and changeable

```
>>> li = list(range(11))
>>> print(li)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> li = list(range(2, 21, 2))
>>> print(li)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```


Homework-3

1. Sum of First N Natural Numbers: Write a Python program that asks the user to enter a positive integer N and then calculates the sum of the first N natural numbers using a `while` loop.

```
# Example Output:
# Enter a positive integer: 5
# The sum of the first 5 natural numbers is 15
```

2. Fibonacci Sequence: Write a Python program that prints the Fibonacci sequence up to a certain number of terms. The user should specify how many terms of the sequence they want to see.

```
# Example Output:
# Enter the number of terms: 7
# Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8
```

3. Sum of Elements in a List: Write a Python program that calculates and prints the sum of all elements in a list using a `for` loop.

```
# Example Output:
# List of numbers: [1, 2, 3, 4, 5]
# The sum of the elements is: 15
```

4. Find the Maximum Number: Write a Python program that finds and prints the maximum number in a list using a `for` loop.

```
# Example Output:
# List of numbers: [3, 5, 7, 2, 8]
# The maximum number is: 8
```

5. Reverse a List: Write a Python program that reverses a list using a `for` loop and prints the reversed list.

```
# Example Output:
# Original list: [1, 2, 3, 4, 5]
# Reversed list: [5, 4, 3, 2, 1]
```

6. Calculate Factorial: Write a Python program that calculates the factorial of a given positive integer using a `for` loop.

```
# Example Output:
# Enter a positive integer: 5
# The factorial of 5 is: 120
```

7. Filter Even Numbers: Write a Python program that prints only the even numbers from a given list using a `for` loop.

```
# Example Output:
# List of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Even numbers: [2, 4, 6, 8, 10]
```