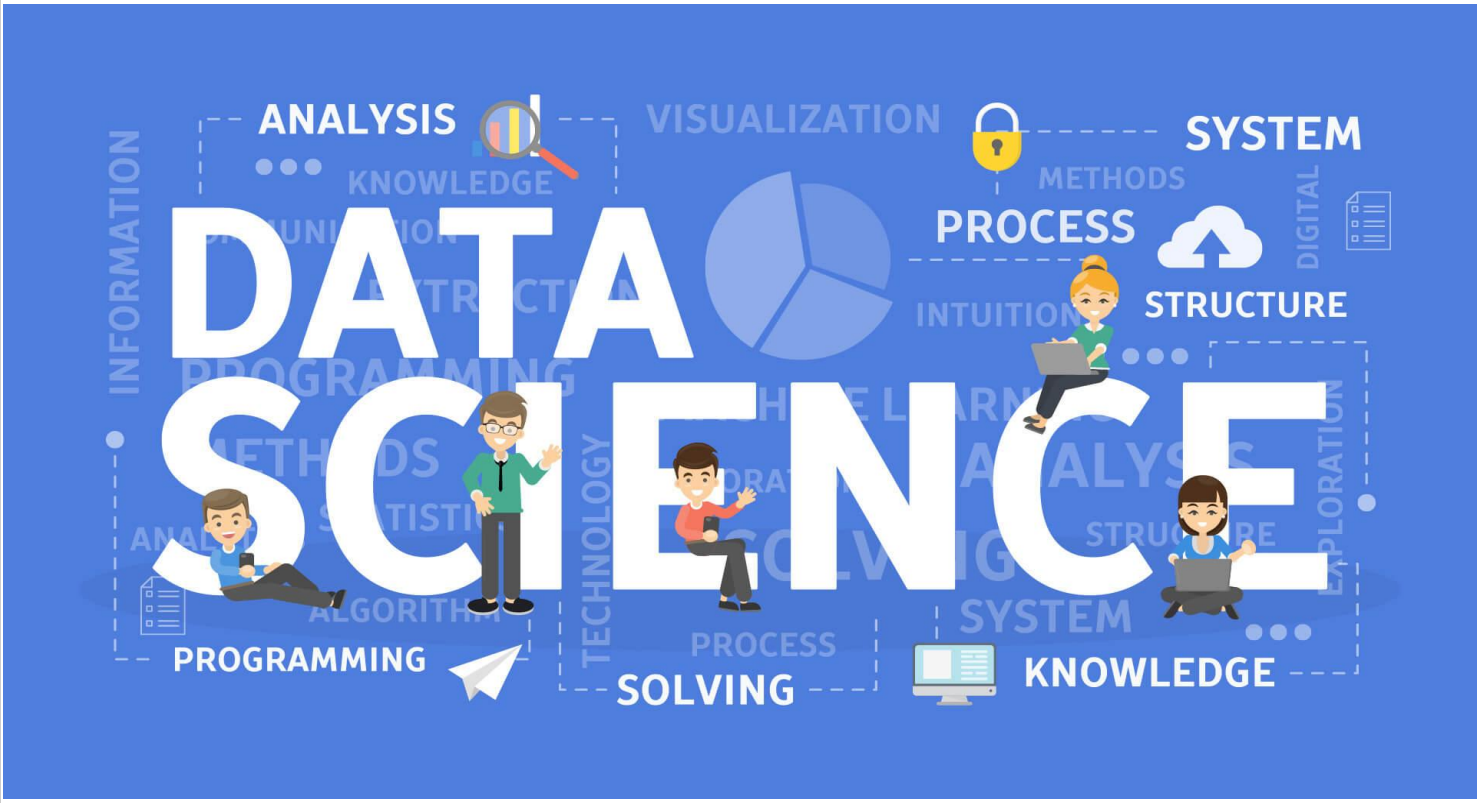
ADITI DIGITAL SOLUTIONS

CORPORATE TRAINING | SOFTWARE DEVELOPMENT | CONSULTING SERVICES



www.aditidigitalsolutions.com

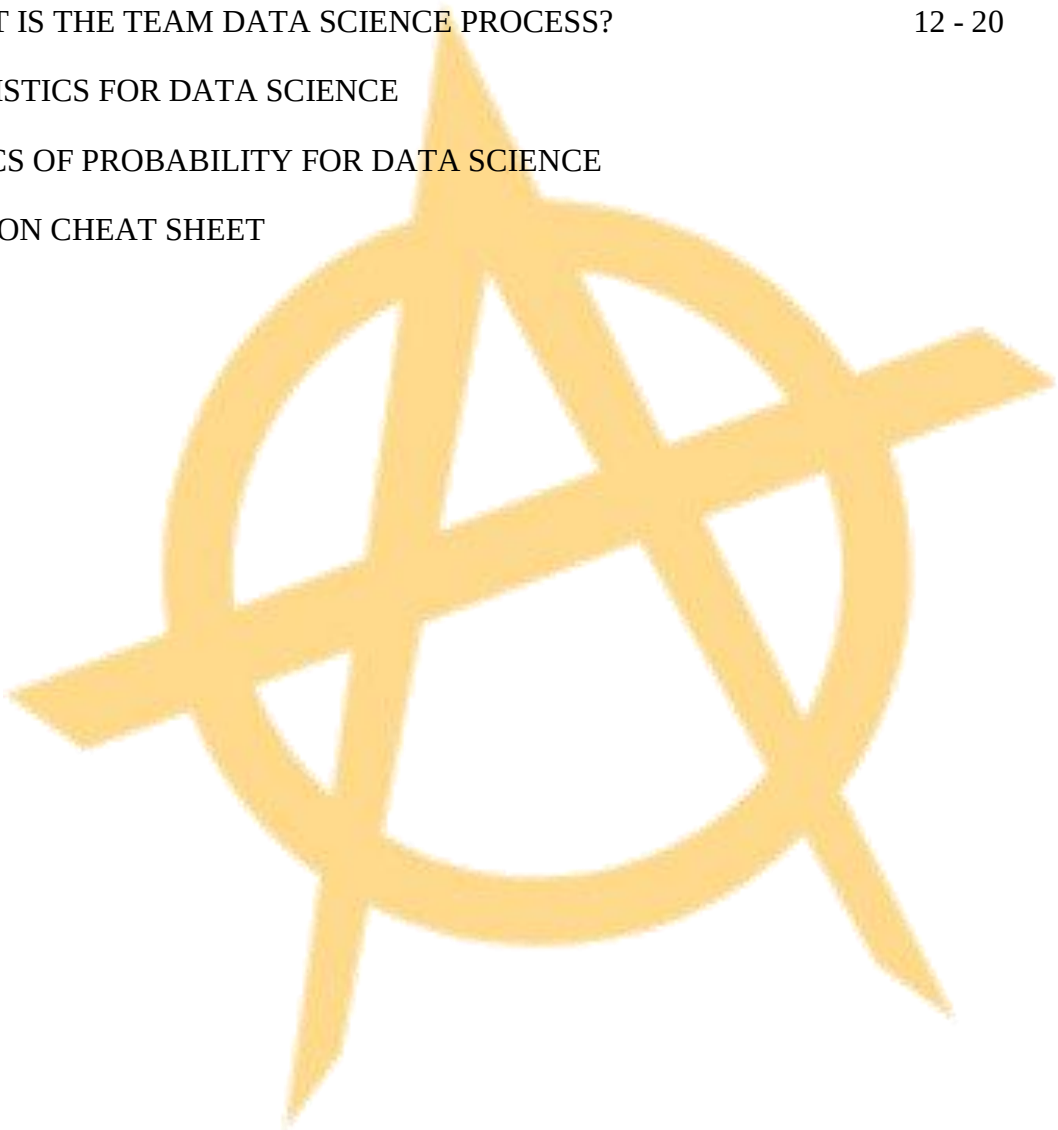
VOLUME -1

040 66 844 999 | +91 836 748 7105 | +91 965 264 8372

admin@aditidigitalsolutions.com

THIRD FLOOR, VVR COMPLEX, KPHB ROAD NO 2, PHASE 1, HYD – 72, TELANGANA, INDIA

CONTENTS



BRIEF INTRODUCTION TO BIG DATA AND DATA ANALYTICS

WHAT IS BIG DATA ANALYTICS?

Big data analytics examines large amounts of data to uncover hidden patterns, correlations and other insights. With today‟s technology, it‟s possible to analyze your data and get answers from it immediately. Big Data Analytics helps you to understand your organization better. With the use of Big data analytics, one can make the informed decisions without blindly relying on guesses.

And it can help answer the following types of questions:

- What actually happened?
- How or why did it happen?
- What‟s happening now?
- What is likely to happen next?

VALUES OF BIG DATA ANALYTICS

Big data analytics helps organizations harness their data and use it to identify new opportunities. That, in turn, leads to smarter business moves, more efficient operations, higher profits and happier customers. Here are the most important values of Big Data,

1. Cost reduction: Big data technologies such as Hadoop and cloud-based analytics bring significant cost advantages when it comes to storing large amounts of data – plus they can identify more efficient ways of doing business.
2. Faster, better decision making: With the speed of Hadoop and in-memory analytics, combined with the ability to analyze new sources of data, businesses are able to analyze information immediately – and make decisions based on what they‟ve learned.
3. New products and services: With the ability to gauge customer needs and satisfaction through analytics comes the power to give customers what they want. Davenport points out that with big data analytics, more companies are creating new products to meet customers‟ needs.

The term is often used synonymously with related concept such as *Business Intelligence (BI) and data mining.* It is true that all these terms is about analyzing data and in many cases advanced analytics . But big data concept is different from the two others when data volumes, number of transactions and the number of data sources are so big and complex that they require special methods and technologies in order to draw insight out of data (for instance, traditional data warehouse solutions may fall short when dealing with big data). This also forms the basis for the most used definition of big data, the three V: Volume, Velocity and Variety.

USES OF BIG DATA ANALYTICS ACROSS DIFFERENT INDUSTRIES

Banking : Large amounts of information will be streaming in into banks, managing all this data and getting proper insights would be possible only with big data analytics. This is important to understand customers and boost their satisfaction, and also to minimize risk and fraud.

Government: When government agencies are able to harness and apply analytics to their big data, they gain significant ground when it comes to managing utilities, running agencies, dealing with traffic congestion or preventing crime.

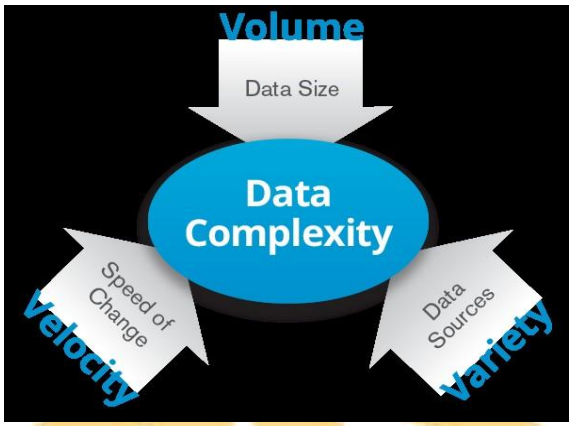
Health Care: Patient records, Treatment plans, Prescription information. When it comes to health care, everything needs to be done quickly, accurately. And, in some cases, with enough transparency to satisfy stringent industry regulations. When big data is managed effectively, health care providers can uncover hidden insights that improve patient care.

Education: Educators armed with data-driven insight can make a significant impact on school systems, students, and curriculums. By analyzing big data, they can identify at-risk students, make sure students are making adequate progress, and can implement a better system for evaluation and support of teachers and principals.

Manufacturing: Armed with insight that big data can provide, manufacturers can boost quality and output while minimizing waste – processes that are key in today"s highly competitive market. More and more manufacturers are working in an analytics-based culture, which means they can solve problems faster and make more agile business decisions.

Retail: Customer relationship building is critical to the retail industry. And the best way to manage that is to manage big data. Retailers need to know the best way to market to customers. The most effective way to handle transactions, and the most strategic way to bring back lapsed business. Big data remains at the heart of all those things.

THE THREE V OF BIG DATA



1. **Volume:** Large amounts of data , from datasets with sizes of terabytes to zettabyte.

    - Aggregation that used to be measured in petabytes (PB) is now referenced by a term: zettabytes (ZB)
    - A zettabyte is a trillion gigabytes (GB)
    - or a billion terabytes

2. **Variety:** Data come from different data sources. For the first, data can come from both internal and external data source. More importantly, data can come in various formats such as transaction and log data from various applications, structured data as database table, semi-structured data such as XML data, unstructured data such as text, images, video streams, audio statement, and more. There is a shift from sole structured data to increasingly more unstructured data or the combination of the two.

    - Relational Data (Tables/Transaction/Legacy Data)
    - Text Data (Web)
    - Semi-structured Data (XML)
    - Graph Data – Social Network, Semantic Web (RDF), …
    - Streaming Data – You can only scan the data once

The type of data generated and stored varies by sector[1]

Penetration: High, Medium, Low

	Video	Image	Audio	Text/ numbers
Banking	Medium	Medium	Medium	High
Insurance	Low	Low	Low	High
Securities and investment services	Low	Low	Low	High
Discrete manufacturing	Medium	Medium	Low	High
Process manufacturing	Medium	Medium	Low	High
Retail	Medium	Low	Low	High
Wholesale	Low	Low	Low	High
Professional services	Medium	Medium	Medium	High
Consumer and recreational services	Medium	Low	Medium	Medium
Health care	Low	High	Low	High
Transportation	Low	Low	Low	High
Communications and media[2]	High	Medium	High	High
Utilities	Low	Low	Low	High
Construction	Low	Medium	Low	Medium
Resource industries	Medium	Medium	Low	High
Government	High	Medium	High	High
Education	High	Medium	High	Medium

1 We compiled this heat map using units of data (in files or minutes of video) rather than bytes.
2 Video and audio are high in some subsectors.
SOURCE: McKinsey Global Institute analysis

3. **Velocity :** Velocity-velocity is the rate at which data arrives at the enterprise and is processed or well understood. In other terms "How long does it take you to do something about it or know it has even arrived?"

Today, it is possible using real time analytics to optimize buttons across website and on Facebook

- Facebook use anonymized data to show the number of times people
- Saw the like buttons
- Clicked like buttons
- Saw like stories on face book
- And clicked stories to visit a given website

This leads us to the most widely used definition in the industry. Gartner (2012) defines Big Data in the following. Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation. It should by now be clear that the "big" in big data is not just about volume. While big data certainly involves having a lot of data, big data does not refer to data volume alone. What it means is that you are not only getting a lot of data. It is also coming at you fast, it is coming at you in complex format, and it is coming at you from a variety of sources.

SIX KEY ESSENTIALS OF BIG DATA PLATFORM

	Big Data Platform Imperatives		Technology Capability
1	Discover, explore, and navigate Big Data sources		Federated Discovery, Search, and Navigation
2	Extreme performance–run analytics closer to data		Massively Parallel Processing Analytic appliances
3	Manage and analyze unstructured data		Hadoop File System/MapReduce Text Analytics
4	Analyze data in motion		Stream Computing
5	Rich library of analytical functions and tools		In-Database Analytics Libraries Big Data Visualization
6	Integrate and govern all data sources		Integration, Data Quality, Security, Lifecycle Management, MDM, etc

DATA ANALYTICS

Analytics Characteristics are not new.

- **Value** : produced when the analytics output is put into action.
- **Veracity** : measure of accuracy and timeliness.
- **Quality** :
  - – well-formed data
  - – Missing values
  - – cleanliness
- **Latency** : time between measurement and availability.
- Data types have differing pre-analytics needs.

THE REAL TIME ROLL

EXAMPLE OF ANALYTICS

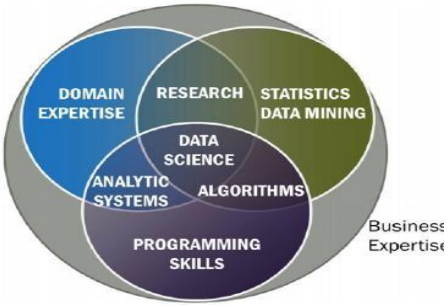
- **Counting** – How many request/day?
  – What‟s the average latency?
  – How many signups, sms, tweets?

- **Correlating** – Desktop vs Mobile user ?
  – What devices fail at the same time?
  – What features get user hooked?

- **Researching** – What features get re-tweeted
  – Duplicate detection
  – Sentiment analysis

SKILLS REQUIRED FOR BIG DATA ANALYTICS

- **Store and Process**
  - – Large scale databases
  - – Software Engineering
  - – System/network Engineering



- **Analyze and Model**
  - – Reasoning
  - – Knowledge Representation
  - – Multimedia Retrieval
  - – Modelling and Simulation
  - – Machine Learning
  - – Information Retrieval

- **Understand and Design**
  - – Decision theory
  - – Visual analytics
  - – Perception Cognition

DIFFERENT LEVELS OF ANALYTICS

1. **DESCRIPTIVE ANALYTICS :** *Descriptive analytics* answers the questions about "what happened in the past?" This involves typically reporting. We can look at some example questions that are typically addressed here.
   - What was the sales revenue in the first quarter of the year? Is additional sales effort needed to meet our target?

- Which is our most profitable product/region/customer?
- How many customers did we win/loose in the first half-year? How many did we win/loose in Oslo area, How many in Mid Norway?
- How many of the won customers can be attributed to the promotional campaign (e.g. via a recorded promotional code) that was launched in Hyderabad last month? Was the campaign successful?

2. **PREDECTIVE ANALYTICS :** *Predictive analytics* aim to something about "what might happen next?" This is harder and it involves extrapolating trends and patterns to the future. Some example questions look like this.
   - What will the number of complaints to our call center next quarter?
   - Which customer are most likely to quit (e.g. cancel her subscription)?
   - What is the next best offer for this customer?

3. **PRESCRIPTIVE ANALYTICS :** *Prescriptive analytics* tries to answer, "how do I deal with this". This is where analytics gets operational. It is totally business and use case dependent. Some examples to illustrate the point.
   - We know that this person has a high chance to quit, we can offer her a value package.
   - We know the viewing history of this customer on our news site, we can recommend articles that we think she would like to read next.
   - From analyzing various sensor data we know that part A of windmill 101 is about to break, replacement part is automatic ordered through supply chain.

THE PERSON THAT DOES THE BIG DATA ANALYTIC JOB IS CALLED DATA SCIENTIST NOW A DAYS.

The main technological components in a Big Data ecosystem

1. **Technologies for capturing, storing and accessing big data**

Traditionally, data are stored in relational database (for example a CRM system for customer data, a supply chain management software for vendor related information) and some of these data are extracted periodically from the operational database, transformed and loaded into data warehouse for reporting and further analysis. This is typically in the realm of Business Intelligence. Such process and tool set fall short when dealing with big data. For instance, one of the largest publicly discussed Hadoop cluster (Yahoo‟s) was at 455 petabytes in 2014 and it's grown since then. There simply is no parallel relational databases or data warehouse that have come even close to those kinds of numbers. Another sweet spot for Hadoop (over relational technology) is when data comes in unstructured format, such as audio, video, text.

2. **Analytical techniques**

Most of the widely used analytical techniques falls into one of the following categories.

- ✓ Statistical methods, forecasting, regression analysis
- ✓ Database querying
- ✓ Data warehouse
- ✓ Machine learning and data mining

3. **Visualization**

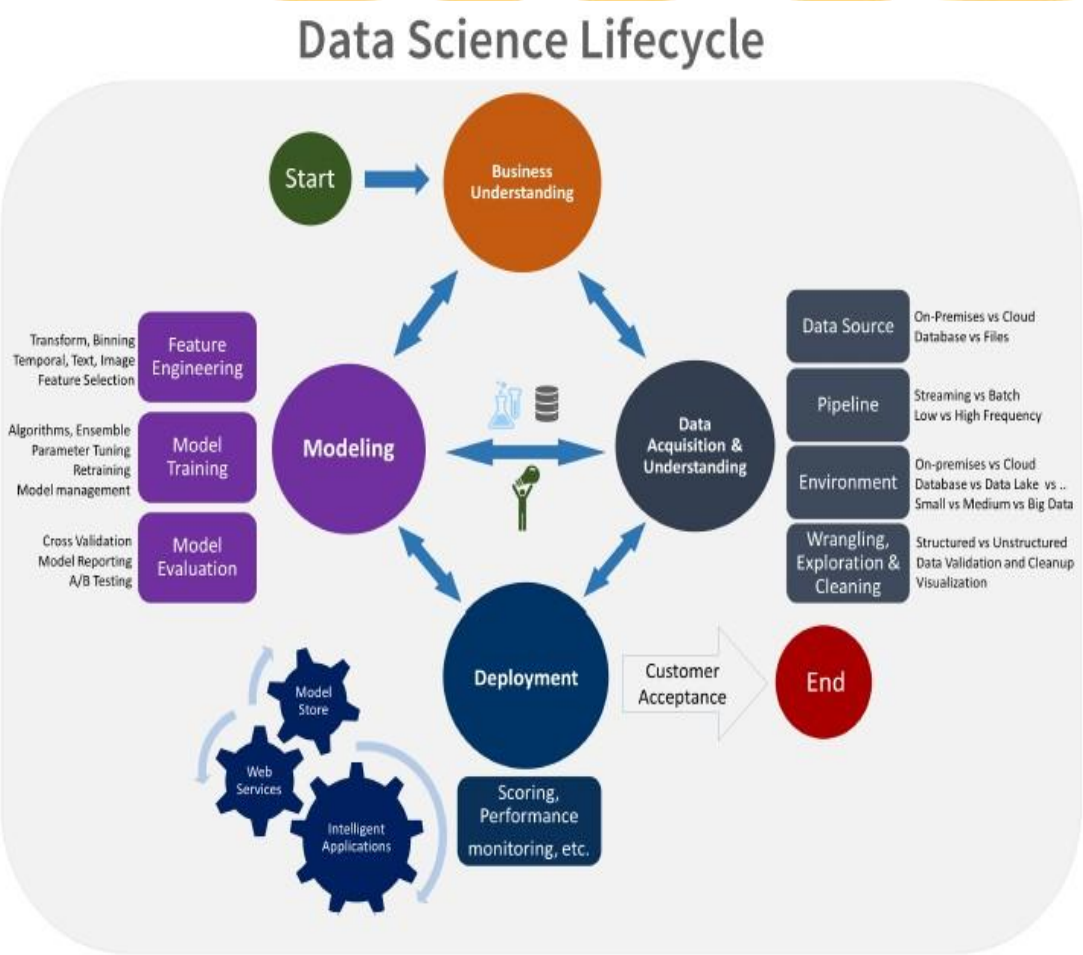
When analysis is done, the results need to be communicated to various stakeholders. One of the hardest parts of an analysis is producing quality supporting graphics. Conversely, a good graph is one of the best ways to present findings. Graphics are used primarily for two reasons: exploratory data analysis and presenting results

WHAT IS THE TEAM DATA SCIENCE PROCESS?

The Team Data Science Process (TDSP) is an agile, iterative data science methodology to deliver predictive analytics solutions and intelligent applications efficiently. TDSP helps improve team collaboration and learning. It contains a distillation of the best practices and structures from Microsoft and others in the industry that facilitate the successful implementation of data science initiatives. The goal is to help companies fully realize the benefits of their analytics program.

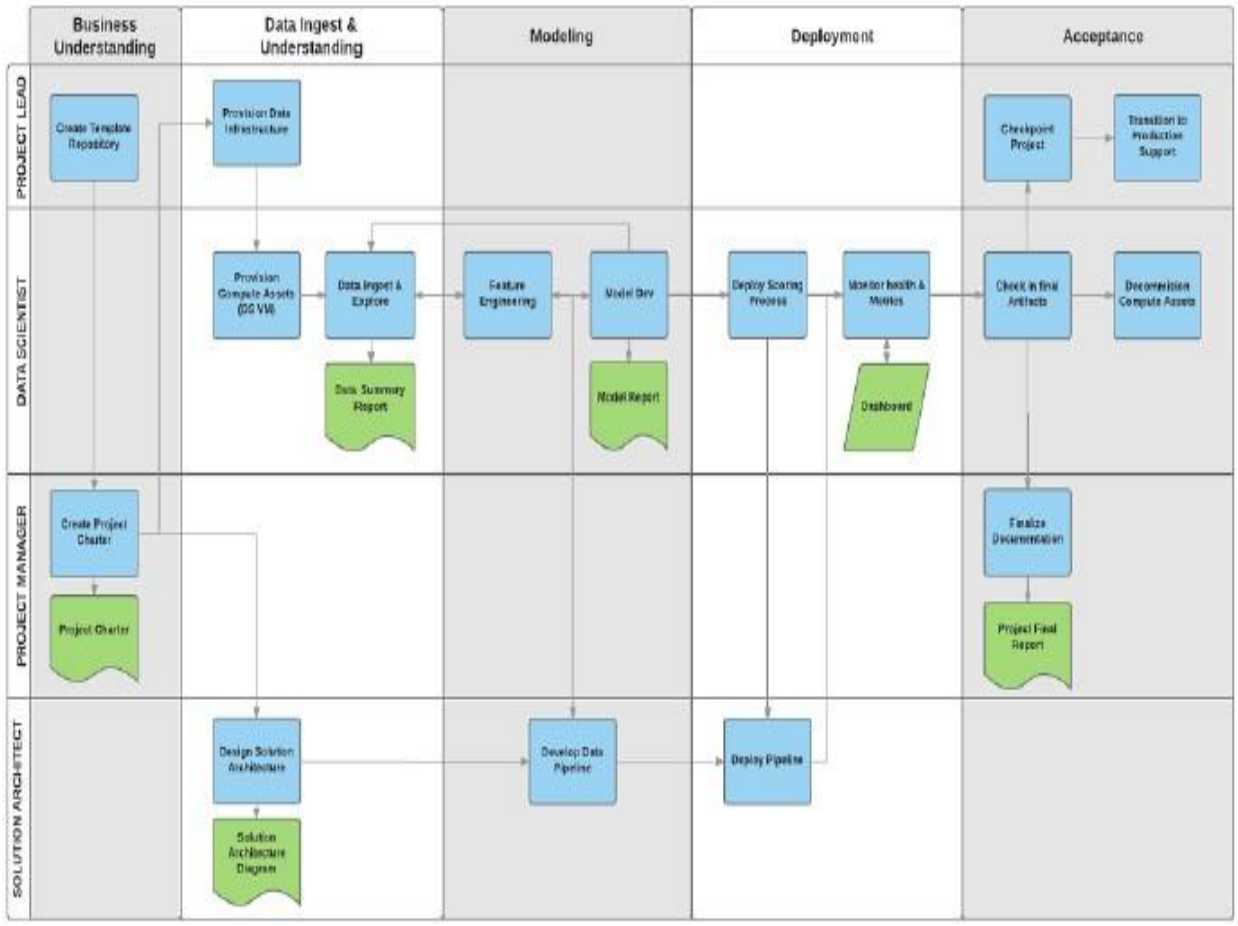
TDSP comprises of the following key components:

- A **data science lifecycle** definition
- A **standardized project structure**
- **Infrastructure and resources** for data science projects
- **Tools and utilities** for project execution



The goals, tasks, and documentation artifacts for each stage of the lifecycle in TDSP are described in the Team Data Science Process lifecycle topic. These tasks and artifacts are associated with project roles:

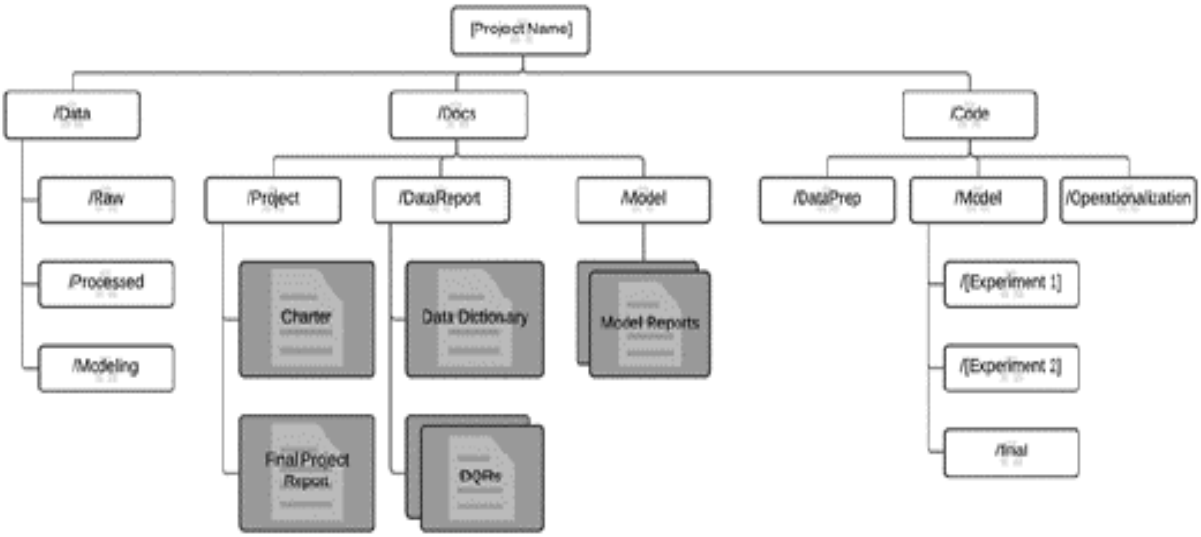
1. Solution architect
2. Project manager
3. Data scientist
4. Project lead



STANDARDIZED PROJECT STRUCTURE

Having all projects share a directory structure and use templates for project documents makes it easy for the team members to find information about their projects. All code and documents are stored in a version control system (VCS) like Git, TFS, or Subversion to enable team collaboration. Tracking tasks and features in an agile project tracking system like Jira, Rally, and Azure DevOps allows closer tracking of the code for individual features. Such tracking also enables teams to obtain better cost estimates. TDSP recommends creating a separate repository for each project on the VCS for versioning, information security, and collaboration. The standardized structure for all projects helps build institutional knowledge across the organization.

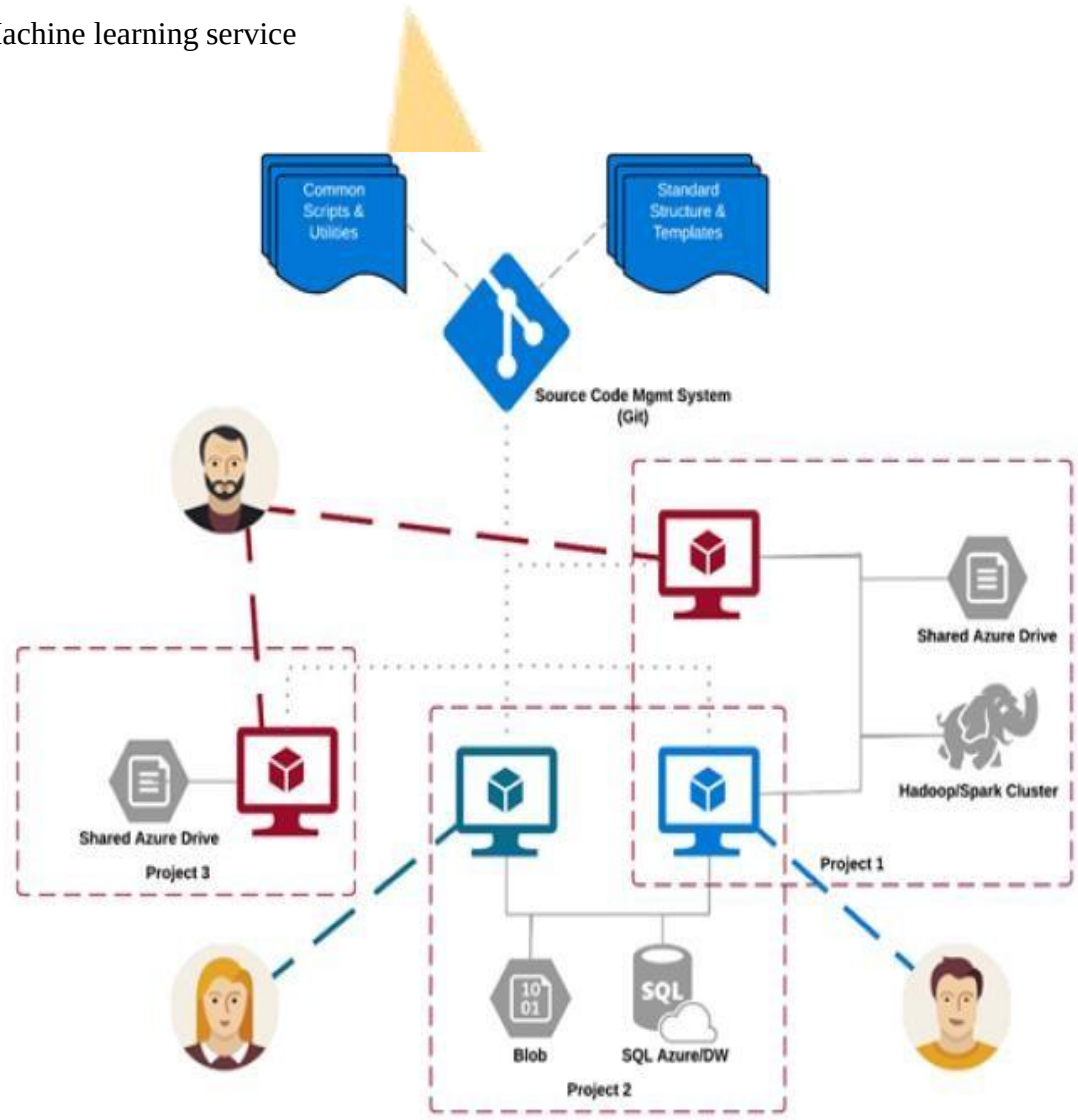
We provide templates for the folder structure and required documents in standard locations. This folder structure organizes the files that contain code for data exploration and feature extraction, and that record model iterations. These templates make it easier for team members to understand work done by others and to add new members to teams. It is easy to view and update document templates in markdown format. Use templates to provide checklists with key questions for each project to ensure that the problem is well-defined and that deliverables meet the quality expected. Examples include:



INFRASTRUCTURE AND RESOURCES FOR DATA SCIENCE PROJECTS

TDSP provides recommendations for managing shared analytics and storage infrastructure such as:

1. Cloud file systems for storing datasets
2. Databases
3. Big data (Hadoop or Spark) clusters
4. Machine learning service



1. Business understanding
2. Data acquisition and understanding
3. Modeling
4. Deployment
5. Customer acceptance

1. BUSINESS UNDERSTANDING:

- Specify the key variables that are to serve as the model targets and whose related metrics are used determine the success of the project.
- Identify the relevant data sources that the business has access to or needs to obtain.

HOW TO DO IT :

- There are two main tasks addressed in this stage:
- **Define objectives**: Work with your customer and other stakeholders to understand and identify the business problems. Formulate questions that define the business goals that the data science techniques can target.
- **Identify data sources** : Find the relevant data that helps you answer the questions that define the objectives of the project.

DEFINE OBJECTIVES

- A central objective of this step is to identify the key business variables that the analysis needs to predict. We refer to these variables as the *model targets*, and we use the metrics associated with them to determine the success of the project. Two examples of such targets are sales forecasts or the probability of an order being fraudulent.
- Define the project goals by asking and refining "sharp" questions that are relevant, specific, and unambiguous. Data science is a process that uses names and numbers to answer such questions. You typically use data science or machine learning to answer five types of questions:

- How much or how many? (regression)
- Which category? (classification)
- Which group? (clustering)
- Is this weird? (anomaly detection)
- Which option should be taken? (recommendation)

- Determine which of these questions you're asking and how answering it achieves your business goals.
- Define the project team by specifying the roles and responsibilities of its members. Develop a high-level milestone plan that you iterate on as you discover more information.
- Define the success metrics. For example, you might want to achieve a customer churn prediction. You need an accuracy rate of "x" percent by the end of this three-month project. With this data, you can offer customer promotions to reduce churn. The metrics must be

SMART :

- **S**pecific
- **M**easurable
- **A**chievable
- **R**elevant
- **T**ime-bound

IDENTIFY DATA SOURCES

Identify data sources that contain known examples of answers to your sharp questions. Look for the following data:

- Data that's relevant to the question. Do you have measures of the target and features that are related to the target?
- Data that's an accurate measure of your model target and the features of interest.

1. **DATA ACQUISITION AND UNDERSTANDING :**

- Produce a clean, high-quality data set whose relationship to the target variables is understood. Locate the data set in the appropriate analytics environment so you are ready to model.
- Develop solution architecture of the data pipeline that refreshes and scores the data regularly.

HOW TO DO IT :

There are three main tasks addressed in this stage:

- **Ingest the data** into the target analytic environment.
- **Explore the data** to determine if the data quality is adequate to answer the question.
- **Set up a data pipeline** to score new or regularly refreshed data.

INGEST THE DATA

Set up the process to move the data from the source locations to the target locations where you run analytics operations, like training and predictions. For technical details and options on how to move the data with various Azure data services, see Load data into storage environments for analytics.

EXPLORE THE DATA

Before you train your models, you need to develop a sound understanding of the data. Real-world data sets are often noisy, are missing values, or have a host of other discrepancies. You can use data summarization and visualization to audit the quality of your data and provide the information you need to process the data before it's ready for modeling. This process is often iterative.

After you're satisfied with the quality of the cleansed data, the next step is to better understand the patterns that are inherent in the data. This helps you choose and develop an appropriate predictive model for your target. Look for evidence for how well connected the data is to the target. Then determine whether there is sufficient data to move forward with the next modeling steps. Again, this process is often iterative. You might need to find new data sources with more accurate or more relevant data to augment the data set initially identified in the previous stage.

SET UP A DATA PIPELINE

In addition to the initial ingestion and cleaning of the data, you typically need to set up a process to score new data or refresh the data regularly as part of an ongoing learning process. You do this by setting up a data pipeline or workflow. The Move data from an on-premises SQL Server instance to Azure SQL Database with Azure Data Factory article gives an example of how to set up a pipeline with Azure Data Factory.

In this stage, you develop solution architecture of the data pipeline. You develop the pipeline in parallel with the next stage of the data science project. Depending on your business needs and the constraints of your existing systems into which this solution is being integrated, the pipeline can be one of the following:

- Batch-based
- Streaming or real time
- A hybrid

2. MODELING STAGE :

- Determine the optimal data features for the machine-learning model.
- Create an informative machine-learning model that predicts the target most accurately.
- Create a machine-learning model that's suitable for production.

HOW TO DO IT :

There are three main tasks addressed in this stage:

- **Feature engineering:** Create data features from the raw data to facilitate model training.
- **Model training**: Find the model that answers the question most accurately by comparing their success metrics.
- **Determine if your model is suitable for production**

3. DEPLOYMENT STAGE :

Deploy models with a data pipeline to a production or production-like environment for final user acceptance.

HOW TO DO IT :

The main task addressed in this stage:

Operationalize the model: Deploy the model and pipeline to a production or production-like environment for application consumption. After you have a set of models that perform well, you can operationalize them for other applications to consume. Depending on the business requirements, predictions are made either in real time or on a batch basis. To deploy models, you expose them with an open API interface. The interface enables the model to be easily consumed from various applications, such as:

- Online websites
- Spreadsheets
- Dashboards
- Line-of-business applications
- Back-end applications

4. CUSTOMER ACCEPTANCE STAGE :

Finalize the project deliverables: Confirm that the pipeline, the model, and their deployment in a production environment satisfy the customer's objectives
There are two main tasks addressed in this stage:

- **System validation**: Confirm that the deployed model and pipeline meet the customer's needs.
- **Project hand-off**: Hand the project off to the entity that's going to run the system in production.

The customer should validate that the system meets their business needs and that it answers the questions with acceptable accuracy to deploy the system to production for use by their client's application. All the documentation is finalized and reviewed. The project is handed-off to the entity responsible for operations. This entity might be, for example, an IT or customer data-science team or an agent of the customer that's responsible for running the system in production.

STATISTICS

Statistics consists of a body of methods for collecting and analyzing data.

From above, it should be clear that statistics is much more than just the tabulation of numbers and the graphical presentation of these tabulated numbers. Statistics is the science of gaining information from numerical and categorical data. Statistical methods can be used to find answers to the questions like:

- What kind and how much data need to be collected?
- How should we organize and summarize the data?
- How can we analyze the data and draw conclusions from it?
- How can we assess the strength of the conclusions and evaluate their uncertainty?

That is, statistics provides methods for

1. Design: Planning and carrying out research studies.
2. Description: Summarizing and exploring data.
3. Inference: Making predictions and generalizing about phenomena represented by the data.

Descriptive and Inferential Statistics

- (Descriptive Statistics). Descriptive statistics consist of methods for organizing and summarizing information.
- (Inferential Statistics). Inferential statistics consist of methods for drawing and measuring the reliability of conclusions about population based on information obtained from a sample of the population.

Descriptive Statistics includes the construction of graphs, charts, and tables, and the calculation of various descriptive measures such as averages, measures of variation, and percentiles. In fact, the most part of this course deals with descriptive statistics.

Inferential Statistics includes methods like point estimation, interval estimation and hypothesis testing which are all based on probability theory.

Example (Descriptive and Inferential Statistics) Consider event of tossing dice. The dice is rolled 100 times and the results are forming the sample data. Descriptive statistics is used to grouping the sample data to the following table.

Outcome of the roll	Frequencies in the sample data
1	10
2	20
3	18
4	16
5	11
6	25

Inferential statistics can now be used to verify whether the dice is a fair or not. Descriptive and inferential statistics are interrelated. It is almost always necessary to use methods of descriptive statistics to organize and summarize the information obtained from a sample before methods of inferential statistics can be used to make more thorough analysis of the subject under investigation.

THE MEAN

The most basic estimate of location is the mean, or *average* value. The mean is the sum of all the values divided by the number of values. Consider the following set of numbers: {3 5 1 2}. The mean is (3 + 5 + 1 + 2) / 4 = 11 / 4 = 2.75. You will encounter the symbol $\bar{x}$ (pronounced "x-bar") to represent the mean of a sample from a population. The formula to compute the mean for a set of *n* values $x_1, x_2, x_3, \ldots x_n$

$$\text{Mean} = \bar{x} = \frac{\sum_i^n x_i}{n}$$

NOTE

N (or *n*) refers to the total number of records or observations. In statistics it is capitalized if it is referring to a population, and lowercase if it refers to a sample from a population. In data science, that distinction is not vital so you may see it both ways.

The accompanying summary data on Na particle sizes (nm) under certain experimental conditions was read from a graph in the article as below

3.0-<3.5	3.5-<4.0	4.0-<4.5	4.5-<5.0	5.0-<5.5	5.5-<6.0	6.0-<6.5	6.5-<7.0	7.0-<7.5	7.5-<8.0
5	15	27	34	22	14	7	2	4	1

Q1.What proportion of the observations are less than 5?

Q2.What proportion of the observations are at least 6?

TRIMMED MEAN

A variation of the mean is a *trimmed mean*, which you calculate by dropping a fixed number of sorted values at each end and then taking an average of the remaining values.

$$\text{Trimmed mean} = \bar{x} = \frac{\sum_{i=p+1}^{n-p} x_{(i)}}{n-2p}$$

The 10% trimmed mean is the mean computed by excluding the 10% largest and 10% smallest values from the sample and taking the arithmetic mean of the remaining 80% of the sample (other trimmed means are possible: 5%, 20%,, etc.)

Example Consider the data (sample)
5, 4, 7, 6, 8, 10, 11, 0, 7 , 18
whose order statistics (rearranged from smallest to largest) are
0, 4, 5, 6, 7, 7, 8, 10, 11, 18
The 10% trimmed mean omits 0 and 18 and yields

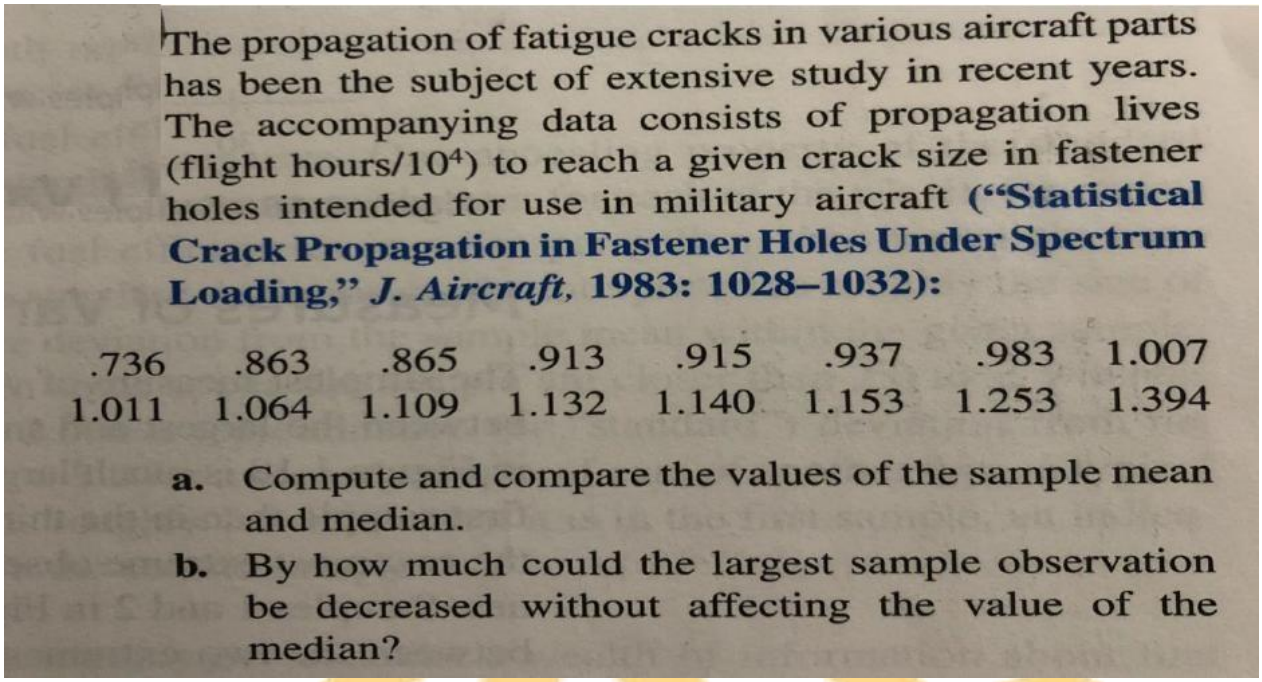
$$\bar{x}_{.10} = \frac{4+5+6+7+7+8+10+11}{8} = 7.25.$$

Note that $\bar{x} = 7.6$ and $\tilde{x} = 7$

Trimmed means are examples of robust statistics (resistant to gross error).

The 20% trimmed mean excludes the 2 smallest and 2 largest values in the sample above, and

$$\bar{x}_{.20} = \frac{5+6+7+7+8+10}{6} = 7.1667.$$



The propagation of fatigue cracks in various aircraft parts has been the subject of extensive study in recent years. The accompanying data consists of propagation lives (flight hours/10^4) to reach a given crack size in fastener holes intended for use in military aircraft (**"Statistical Crack Propagation in Fastener Holes Under Spectrum Loading," *J. Aircraft*, 1983: 1028–1032):**

.736	.863	.865	.913	.915	.937	.983	1.007
1.011	1.064	1.109	1.132	1.140	1.153	1.253	1.394

a. Compute and compare the values of the sample mean and median.

b. By how much could the largest sample observation be decreased without affecting the value of the median?

WEIGHTED MEAN

Sample problem: You take three 100-point exams in your statistics class and score 80, 80 and 95. The last exam is *much* easier than the first two, so your professor has given it less weight. The weights for the three exams are:

- Exam 1: 40 % of your grade. (Note: 40% as a decimal is .4)
- Exam 2: 40 % of your grade.
- Exam 3: 20 % of your grade.

What is your final weighted average for the class?

1. Multiply the numbers in your data set by the weights:
   .4(80) = 32
   .4(80) = 32
   .2(95) = 19
2. Add the numbers up. 32 + 32 + 19 = 83.

WEIGHTED MEAN FORMULA

$$\bar{x} = \frac{\sum_{i=1}^{n} (x_i * w_i)}{\sum_{i=1}^{n} w_i}$$

The above image is the technical formula for the weighted mean. In simple terms, the formula can be written as:
Weighted mean = Σwx/Σw
Σ = the sum of (in other words…add them up!).
w = the weights.
x = the value.

In the sample grades problem above, all of the weights add up to 1 (.4 + .4 + .2) so you would divide your answer (83) by 1:
83 / 1 = 83.

However, let‟s say your weighted means added up to 1.2 instead of 1. You‟d divide 83 by 1.2 to get:
83 / 1.2 = 69.17.

MEASURES OF CENTER

1. **The Mode :** The sample mode of a qualitative or a discrete quantitative variable is that value of the variable which occurs with the greatest frequency in a data set.

   Obtain the frequency of each observed value of the variable in a data and note the greatest frequency.

   - If the greatest frequency is 1 (i.e. no value occurs more than once), then the variable has no mode.
   - If the greatest frequency is 2 or greater, then any value that occurs with that greatest frequency is called a sample mode of the variable.

2. **The Median :** Arrange the observed values of variable in a data in increasing order.

   - If the number of observation is odd, then the sample median is the observed value exactly in the middle of the ordered list.
   - If the number of observation is even, then the sample median is the number halfway between the two middle observed values in the ordered list.

   Example: **13, 18, 13, 14, 13, 16, 14, 21, 13**

   The median is the middle value, so first I'll have to rewrite the list in numerical order13, 13, 13, 13, 14, 14, 16, 18, 21. **So the median is 14.**

3. **The Mean :** The sample mean of the variable is the sum of observed values in a data divided by the number of observations.

   Example:**13, 18, 13, 14, 13, 16, 14, 21, 13**

   The mean is the usual average, so I'll add and then divide:

   (13 18 + 13 + 14 + 13 + 16 + 14 + 21 + 13) ÷ 9 = 15

4. **Range :** The sample range of the variable is the difference between its maximum and minimum values in a data set

   Range = Max −Min.

   0.11,0.17,0.11,0.15,0.10,0.11,0.21,0.20,0.14,0.14,0.23,0.25,0.07,0.09,0.10,0.10,0.19,0.11,0.19,0.17,0.12,0.12,0.12,0.10,0.11,0.13,0.10,0.09,0.11,0.15,0.13,0.10,0.18,0.09,0.07,0.08,0.06,0.08,0.05,0.07,0.08,0.08,0.07,0.09,0.06,0.07,0.08,0.07,0.07,0.07,0.08,0.06, 0.07,0.06

   Minimum = 0.05 , Maximum = 0.25

   Range is = 0.20

5. **Standard Deviation :** For a variable x, the sample standard deviation, denoted by sx (or when no confusion arise, simply by s), is

$$s_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}.$$

Since the standard deviation is defined using the sample mean $\bar{x}$ of the variable x, it is preferred measure of variation when the mean is used as the measure of center (i.e. in case of symmetric distribution). Note that the stardard deviation is always positive number, i.e., $S_x >= 0$.

6. **LINEAR ALGEBRA :** Linear algebra provides the first steps into *vectorization*, presenting a deeper way of thinking about parallelization of certain operations. Algorithms written in standard 'for-loop' notation can be reformulated as matrix equations providing significant gains in computational efficiency.

Such methods are used in the major Python libraries such as NumPy, SciPy, Scikit-Learn, Pandas and Tensorflow. GPUs have been designed to carry out optimised linear algebra operations. The explosive growth in deep learning can partially be attributed to the highly parallelised nature of the underlying algorithms on commodity GPU hardware.

Applications of Linear Algebra

As linear algebra is the mathematics of data, the tools of linear algebra are used in many domains.

- Matrices in Engineering, such as a line of springs.
- Graphs and Networks, such as analyzing networks.
- Markov Matrices, Population, and Economics, such as population growth.
- Linear Programming, the simplex optimization method.
- Fourier Series: Linear Algebra for functions, used widely in signal processing.
- Linear Algebra for statistics and probability, such as least squares for regression.
- Computer Graphics, such as the various translation, rescaling and rotation of images.

7. **VECTORS AND MATRICES :** The two primary mathematical entities that are of interest in linear algebra are the vector and the matrix. They are examples of a more general entity known as a tensor. Tensors possess an order (or rank), which determines the number of dimensions in an array required to represent it.

SCALARS are single numbers and are an example of a 0th-order tensor. In mathematics it is necessary to describe the set of values to which a scalar belongs. The notation
X € R states that the (lowercase) scalar value X is an element of (or member of) the set of real-valued numbers R.

There are various sets of numbers of interest within machine learning. N represents the set of positive integers (1,2,3,…). Z represents the integers, which include positive, negative and zero values. Q represents the set of rational numbers that may be expressed as a fraction of two integers.

VECTORS are ordered arrays of single numbers and are an example of 1st-order tensor. Vectors are members of objects known as vector spaces. A vector space can be thought of as the entire collection of all possible vectors of a particular length (or dimension). The three-dimensional real-valued vector space, denoted by R3 is often used to represent our real-world notion of three-dimensional space mathematically.

More formally a vector space is an n dimensional Cartesian product of a set with itself, along with proper definitions on how to add vectors and multiply them with scalar values. If all of the scalars in a vector are real-valued then the notation x∈Rn states that the (boldface lowercase) vector value x is a member of the n -dimensional vector space of real numbers, Rn.

A vector is an ordered finite list of numbers. Vectors are usually written as vertical arrays, surrounded by square or curved brackets, as in

$$\begin{bmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{bmatrix} \quad \text{or} \quad \begin{pmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{pmatrix}$$

They can also be written as numbers separated by commas and surrounded by parentheses. In this notation style, the vector above is written as $(1.1, 0.0, 3.6, 7.2)$

The elements (or entries, coefficients, components) of a vector are the values in the array. The size (also called dimension or length) of the vector is the number of elements it contains. The vector above, for example, has size four; its third entry is 3:6. A vector of size n is called an n-vector. A 1-vector is considered to be the same as a number, i.e., we do not distinguish between the 1-vector [1:3] and the number 1:3.

Two vectors a and b are equal, which we denote a = b, if they have the same size, and each of the corresponding entries is the same. If a and b are n-vectors, then a = b means a1 = b1, . . . , an = bn.

Basics of Statistics for Data Science
By Dr.Mohammed Habeeb Vulla

Statistics Cheat Sheet

Population

The entire group one desires information about

Sample

A subset of the population taken because the entire population is usually too large to analyze
Its characteristics are taken to be representative of the population

Mean

Also called the arithmetic mean or average
The sum of all the values in the sample divided by the number of values in the sample/population
μ is the mean of the population; $\bar{x}$ is the mean of the sample

Median

The value separating the higher half of a sample/population from the lower half
Found by arranging all the values from lowest to highest and taking the middle one (or the mean of the middle two if there are an even number of values)

Variance

Measures dispersion around the mean
Determined by averaging the squared differences of all the values from the mean
Variance of a population is σ^2

$$\sigma^2 = \frac{\sum (x-\mu)^2}{n}$$

Can be calculated by subtracting the square of the mean from the average of the squared scores:

$$\sigma^2 = \frac{\sum x^2}{n} - \mu^2$$

Variance of a sample is s^2; note the *n-1*

$$s^2 = \frac{\sum (x-\bar{x})^2}{n-1}$$

Can be calculated by:

$$s^2 = \frac{\sum x^2 - \frac{(\sum x)^2}{n}}{n-1}$$

Standard Deviation

Square root of the variance
Also measures dispersion around the mean but in the same units as the values (instead of square units with variance)
σ is the standard deviation of the population and *s* is the standard deviation of the sample

Standard Error

An estimate of the standard deviation of the sampling distribution—the set of all samples of size *n* that can be taken from a population
Reflects the extent to which a statistic changes from sample to sample

For a mean, $\frac{s}{\sqrt{n}}$

For the difference between two means,

Assuming equal variances $\sqrt{s^2\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}$; unequal variances $\sqrt{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}$

T-test

One-Sample

Tests whether the mean of a normally distributed population is different from a specified value

Null Hypothesis (H_0): states that the population mean is equal to some value (μ_0)
Alternative Hypothesis (H_a): states that the mean does not equal/is greater than/is less than μ_0
t-statistic: standardizes the difference between $\bar{x}$ and μ_0

$$t = \frac{\bar{x} - \mu_0}{\frac{s}{\sqrt{n}}}$$ Degrees of freedom (df) = *n-1*

Read the table of t-distribution critical values for the p-value (probability that the sample mean was obtained by chance given μ_0 is the population mean) using the calculated t-statistic and degrees of freedom.

- H_a: $\mu > \mu_0$ → the t-statistic is likely positive; read table as given
- H_a: $\mu < \mu_0$ → the t-statistic is likely negative; the t-distribution is symmetrical so read the probability as if the t-statistic were positive
- Note: if the t-statistic is of the 'wrong' sign, the p-value is 1 minus the *p* given in the chart
- H_a: $\mu \neq \mu_0$ → read the p-value as if the t-statistic were positive and double it (to consider both less than and greater than)

If the p-value is less than the predetermined value for significance (called α and is usually 0.05), reject the null hypothesis and accept the alternative hypothesis.

Example:
You are experiencing hair loss and skin discoloration and think it might be because of selenium toxicity. You decide to measure the selenium levels in your tap water once a day for one week. Your results are given below. The EPA maximum contaminant level for safe drinking water is 0.05 mg/L. Does the selenium level in your tap water exceed the legal limit (assume α=0.05)?

Day	Selenium mg/L
1	0.051
2	0.0505
3	0.049
4	0.0516
5	0.052
6	0.0508
7	0.0506

H_0: $\mu = 0.05$; H_a: $\mu > 0.05$
Calculate the mean and standard deviation of your sample:

$$\bar{x} = 0.0508$$

$$s^2 = \frac{\sum (x - \bar{x})^2}{n-1} = \frac{(0.051 - 0.0508)^2 + (0.0505 - 0.0508)^2 + etc\ldots}{6} = 9.15 \times 10^{-7}$$

$$s = \sqrt{s^2} = 9.56 \times 10^{-4}$$

The t-statistic is: $t = \frac{\bar{x} - \mu_0}{\frac{s}{\sqrt{n}}} = \frac{0.0508 - 0.05}{\frac{9.56 \times 10^{-4}}{\sqrt{7}}} = 2.17$ and the degrees of freedom are *n-1* = 7-1 = 6

Looking at the t-distribution of critical values table, 2.17 with 6 degrees of freedom is between *p*=0.05 and *p*=0.025. This means that the p-value is less than 0.05, so you can reject H_0 and conclude that the selenium level in your tap water exceeds the legal limit.

T-test

Two-Sample

Tests whether the means of two populations are significantly different from one another

Paired

Each value of one group corresponds directly to a value in the other group; ie: before and after values after drug treatment for each individual patient

Subtract the two values for each individual to get one set of values (the differences) and use $\mu_0 = 0$ to perform a one-sample t-test

Unpaired

The two populations are independent

H_0: states that the means of the two populations are equal ($\mu_1 = \mu_2$)

H_a: states that the means of the two populations are unequal or one is greater than the other ($\mu_1 \neq \mu_2$, $\mu_1 > \mu_2$, $\mu_1 < \mu_2$)

t-statistic:

assuming equal variances: $t = \dfrac{\bar{x}_1 - \bar{x}_2}{\sqrt{s^2\left(\dfrac{1}{n_1} + \dfrac{1}{n_2}\right)}}$ assuming unequal variances: $t = \dfrac{\bar{x}_1 - \bar{x}_2}{\sqrt{\left(\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}\right)}}$

degrees of freedom = $(n_1\text{-}1)+(n_2\text{-}1)$

Read the table of t-distribution critical values for the p-value using the calculated t-statistic and degrees of freedom. Remember to keep the sign of the t-statistic clear (order of subtracting the sample means) and to double the p-value for an H_a of $\mu_1 \neq \mu_2$.

Example:

Consider the lifespan of 18 rats. 12 were fed a restricted calorie diet and lived an average of 700 days (standard deviation=21 days). The other 6 had unrestricted access to food and lived an average of 668 days (standard deviation=30 days). Does a restricted calorie diet increase the lifespan of rats (assume α=0.05)?

μ_1=700, s_1=21, n_1=12; μ_2=668, s_2=30, n_2=6

H_0: $\mu_1=\mu_2$

H_a: $\mu_1>\mu_2$ (because we are only asking if a restricted calorie diet increases lifespan)

We cannot assume that the variances of the two populations are equal because the different diets could also affect the variability in lifespan.

The t-statistic is: $t = \dfrac{\bar{x}_1 - \bar{x}_2}{\sqrt{\left(\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}\right)}} = \dfrac{700 - 668}{\sqrt{\dfrac{21^2}{12} + \dfrac{30^2}{6}}} = 2.342$

Degrees of freedom = $(n_1\text{-}1)+(n_2\text{-}1)$ = (12-1)+(6-1)=16

From the t-distribution table, the p-value falls between 0.01 and 0.02, so we do reject H_0. The restricted calorie diet does increase the lifespan of rats.

Chi-Square Test

For Goodness of Fit

Checks whether or not an observed pattern of data fits some given distribution

H_0: the observed pattern fits the given distribution

H_a: the observed pattern does not fit the given distribution

The chi-square statistic is: $\chi^2 = \sum \dfrac{(O-E)^2}{E}$ (O is the observed value and E is the expected value)

Degrees of freedom = number of categories in the distribution – 1

Get the p-value from the table of χ^2 critical values using the calculated χ^2 and df values. If the p-value is less than α, the observed data does not fit the expected distribution. If $p>\alpha$, the data likely fits the expected distribution

***Example 1*:**

You breed puffskeins and would like to determine the pattern of inheritance for coat color and purring ability. Puffskeins come in either pink or purple and can either purr or hiss. You breed a purebred, pink purring male with a purebred, purple hissing female. All individuals of the F_1 generation are pink and purring. The F_2 offspring are shown below. Do the alleles for coat color and purring ability assort independently (assume α=0.05)?

Pink and Purring	Pink and Hissing	Purple and Purring	Purple and Hissing
143	60	55	18

Independent assortment means a phenotypic ratio of 9:3:3:1, so:

H_0: the observed distribution of F_2 offspring fits a 9:3:3:1 distribution

H_a: the observed distribution of F_2 offspring does not fit a 9:3:3:1 distribution

The expected values are:

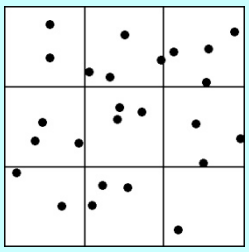
Pink and Purring	Pink and Hissing	Purple and Purring	Purple and Hissing
155.25	51.75	51.75	17.25

$$\chi^2 = \sum \frac{(O-E)^2}{E} = \frac{(143-155.25)^2}{155.25} + \frac{(60-51.75)^2}{51.75} + \frac{(55-51.75)^2}{51.75} + \frac{(18-17.25)^2}{17.25} = 2.519$$

df=4-1=3

From the table of χ^2 critical values, the p-value is greater than 0.25, so the alleles for coat color and purring ability do assort independently in puffskeins.

Example 2:



You are studying the pattern of dispersion of king penguins and the diagram on the right represents an area you sampled. Each dot is a penguin. Do the penguins display a uniform distribution (assume α=0.05)?

H_0: there is a uniform distribution of penguins

H_a: there is not a uniform distribution of penguins

There are a total of 25 penguins, so if there is a uniform distribution, there should be 2.778 penguins per square. There actual observed values are 2, 4, 4, 3, 3, 3, 2, 3, 1, so the χ^2 statistic is:

$$\chi^2 = \sum \frac{(O-E)^2}{E} = \frac{(1-2.778)^2}{2.778} + 2\left(\frac{(2-2.778)^2}{2.778}\right) + 4\left(\frac{(3-2.778)^2}{2.778}\right) + 2\left(\frac{(4-2.778)^2}{2.778}\right) = 2.72$$

df=9-1=8

From the table of χ^2 critical values, the p-value is greater than 0.25, so we do not reject H_0. The penguins do display a uniform distribution.

Chi-Square Test

For Independence

Checks whether two categorical variables are related or not (independence)

H_0: the two variables are independent

H_a: the two variables are not independent

Does not make any assumptions about an expected distribution

The observed values ($\#_1$, $\#_2$, $\#_3$, and $\#_4$) are usually presented as a table. Each row is a category of variable 1 and each column is a category of variable 2.

		Variable 1		Totals
		Category *x*	Category *y*	
Variable 2	Category *a*	$\#_1$	$\#_2$	$\#_1+\#_2$
	Category *b*	$\#_3$	$\#_4$	$\#_3+\#_4$
Totals		$\#_1+\#_3$	$\#_2+\#_4$	$\#_1+\#_2+\#_3+\#_4$

The proportion of category *x* of variable 1 is the number of individuals in category *x* divided by the total number of individuals $\left(\frac{\#_1+\#_3}{\#_1+\#_2+\#_3+\#_4}\right)$. Assuming independence, the expected number of individuals that fall within category *a* of variable 2 is the proportion of category *x* multiplied by the number of individuals in category *a* $\left(\frac{\#_1+\#_3}{\#_1+\#_2+\#_3+\#_4}\right)(\#_1+\#_2)$. Thus, the expected value is:

$$E = \frac{(\#_1+\#_3)(\#_1+\#_2)}{\#_1+\#_2+\#_3+\#_4} = \frac{(row\ total)(column\ total)}{grand\ total}$$

Degrees of freedom = $(r-1)(c-1)$ where *r* is the number of rows and *c* is the number of columns

The chi-square statistic is still $\chi^2 = \sum \frac{(O-E)^2}{E}$

Read the p-values from the table of χ^2 critical values.

Example:

Given the data below, is there a relationship between fitness level and smoking habits (assume α=0.05)?

	Fitness Level				
	Low	Medium-Low	Medium-High	High	
Never smoked	113	113	110	159	495
Former smokers	119	135	172	190	616
1 to 9 cigarettes daily	77	91	86	65	319
≥ 10 cigarettes daily	181	152	124	73	530
	490	491	492	487	1960

H_0: fitness level and smoking habits are independent
H_a: fitness level and smoking habits are not independent
First, we calculate the expected counts. For the first cell, the expected count is:

$$E = \frac{(row\ total)(column\ total)}{grand\ total} = \frac{(495)(490)}{1960} = 123.75$$

	Fitness Level			
	Low	Medium-Low	Medium-High	High
Never smoked	123.75	124	124.26	122.99
Former smokers	154	154.31	154.63	153.06
1 to 9 cigarettes daily	79.75	79.91	80.08	79.26
≥ 10 cigarettes daily	132.5	132.77	133.04	131.69

$$\chi^2 = \sum \frac{(O-E)^2}{E} = \frac{(113-123.75)^2}{123.75} + \frac{(113-124)^2}{124} + \frac{(110-124.26)^2}{124.26} + etc... = 91.73$$

df=(*r*-1)(*c*-1)=(4-1)(4-1)=9

From the table of χ^2 critical values, the p-value is less than 0.001, so we reject H_0 and conclude that there is a relationship between fitness level and smoking habits.

Type I error

The probability of rejecting a true null hypothesis
Equals α

Type II error

The probability of failing to reject a false null hypothesis

Probability

Joint Probability

The probability of events A and B occurring
$P(A \text{ and } B) = P(A) \times P(B)$ when events A and B are independent

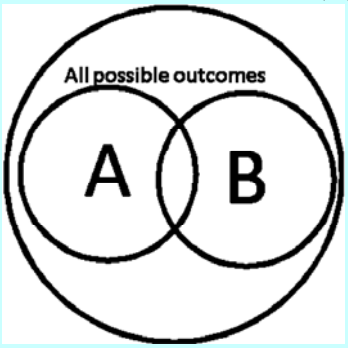
Union of Events

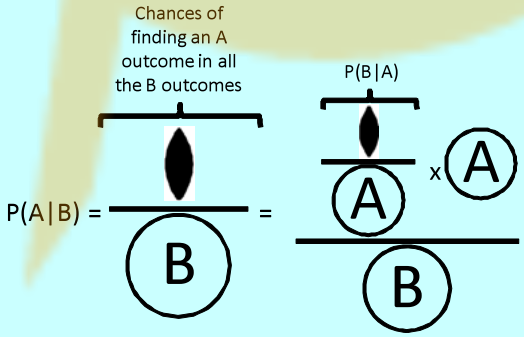
The probability of either event A or event B occurring
$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$

Conditional Probability

The probability of event A occurring given that event B has occurred

$$P(A \mid B) = \frac{P(A \text{ and } B)}{P(B)} \quad \text{or} \quad P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$$





Example 1:
Assume that eye color is an autosomally inherited trait controlled by one gene with two alleles. Brown is dominant to blue. A brown-eyed man with genotype Bb and a blue-eyed woman have three children. The first has blue eyes. What is the probability that all three children have blue eyes?

Without considering the first child, the probability that the couple has three children with blue eyes is
$0.5 \times 0.5 \times 0.5 = 0.125 = P(A \text{ and } B) = P(2 \text{ children} = \text{bb and 1st child bb})$
With his parents, the probability that the 1st child is bb is: $P(B) = P(1\text{st child} = \text{bb}) = 0.5$
Therefore, $P(2 \text{ children} = \text{bb} \mid \text{1st child bb}) = P(A \mid B) = \dfrac{P(A \text{ and } B)}{P(B)} = \dfrac{0.125}{0.5} 0.25$

Example 2:
Based on an analysis of her pedigree, it is determined that a woman has a 70% chance of being Zz and a 30% chance of being ZZ for a sex-linked trait, where Z is dominant to z. If she now has a son with the Z phenotype, what is the probability of her being Zz?

We're looking for: P(W=Zz|S=Z)
But it's hard to find P(W=Zz and S=Z) because the two events are not independent. Instead, let us use:
$P(A \mid B) = \dfrac{P(B \mid A) \times P(A)}{P(B)}$
$P(S = Z \mid W = Zz) = 0.5$ (50% chance of passing on the Z allele)
$P(W = Zz) = 0.7$ (given)
$P(S = Z) = (0.7 \times 0.5) + (0.3 \times 1) = 0.65$ (son can be Z from the woman being either Zz or ZZ)
$P(W = Zz \mid S = Z) = \dfrac{0.5 \times 0.7}{0.65} = 0.538$

Multiple Experiments

Binomial distribution

For when you are not concerned about the order of the events, only that they occur

$$P(X = m) = \frac{n! \times p^m \times (1-p)^{(n-m)}}{m! \times (n-m)!}$$

for *m* outcomes of event X in *n* total trials with *p*=probability of X occurring once

Example:
What is the probability that a couple has one boy out of five children?

$$P(1 \text{ boy of } 5 \text{ children}) = \frac{5! \times 0.5^1 \times 0.5^4}{1! \times (4)!} = 0.15625$$

Poisson distribution

The binomial distribution works for a small number of trials but as *n* gets too large, the factorials become unwieldy.
The Poisson distribution is an estimate of the binomial distribution for large *n*.

$$P(X = m) = \frac{e^{-np} \times (n \times p)^m}{m!}$$

Note: *np* is also known as the number of expected outcomes for event X

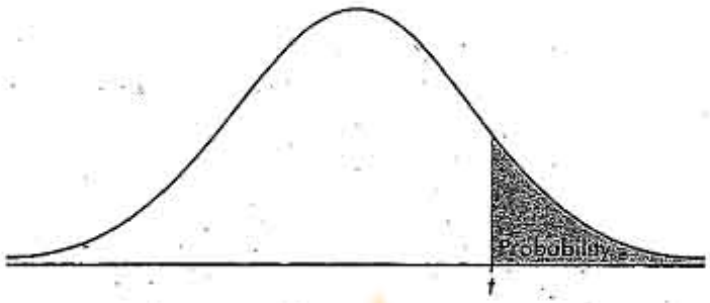


TABLE B: *t*-DISTRIBUTION CRITICAL VALUES

	Tail probability p											
df	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001	.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	15.89	31.82	63.66	127.3	318.3	636.6
2	.816	1.061	1.386	1.886	2.920	4.303	4.849	6.965	9.925	14.09	22.33	31.60
3	.765	.978	1.250	1.638	2.353	3.182	3.482	4.541	5.841	7.453	10.21	12.92
4	.741	.941	1.190	1.533	2.132	2.776	2.999	3.747	4.604	5.598	7.173	8.610
5	.727	.920	1.156	1.476	2.015	2.571	2.757	3.365	4.032	4.773	5.893	6.869
6	.718	.906	1.134	1.440	1.943	2.447	2.612	3.143	3.707	4.317	5.208	5.959
7	.711	.896	1.119	1.415	1.895	2.365	2.517	2.998	3.499	4.029	4.785	5.408
8	.706	.889	1.108	1.397	1.860	2.306	2.449	2.896	3.355	3.833	4.501	5.041
9	.703	.883	1.100	1.383	1.833	2.262	2.398	2.821	3.250	3.690	4.297	4.781
10	.700	.879	1.093	1.372	1.812	2.228	2.359	2.764	3.169	3.581	4.144	4.587
11	.697	.876	1.088	1.363	1.796	2.201	2.328	2.718	3.106	3.497	4.025	4.437
12	.695	.873	1.083	1.356	1.782	2.179	2.303	2.681	3.055	3.428	3.930	4.318
13	.694	.870	1.079	1.350	1.771	2.160	2.282	2.650	3.012	3.372	3.852	4.221
14	.692	.868	1.076	1.345	1.761	2.145	2.264	2.624	2.977	3.326	3.787	4.140
15	.691	.866	1.074	1.341	1.753	2.131	2.249	2.602	2.947	3.286	3.733	4.073
16	.690	.865	1.071	1.337	1.746	2.120	2.235	2.583	2.921	3.252	3.686	4.015
17	.689	.863	1.069	1.333	1.740	2.110	2.224	2.567	2.898	3.222	3.646	3.965
18	.688	.862	1.067	1.330	1.734	2.101	2.214	2.552	2.878	3.197	3.611	3.922
19	.688	.861	1.066	1.328	1.729	2.093	2.205	2.539	2.861	3.174	3.579	3.883
20	.687	.860	1.064	1.325	1.725	2.086	2.197	2.528	2.845	3.153	3.552	3.850
21	.686	.859	1.063	1.323	1.721	2.080	2.189	2.518	2.831	3.135	3.527	3.819
22	.686	.858	1.061	1.321	1.717	2.074	2.183	2.508	2.819	3.119	3.505	3.792
23	.685	.858	1.060	1.319	1.714	2.069	2.177	2.500	2.807	3.104	3.485	3.768
24	.685	.857	1.059	1.318	1.711	2.064	2.172	2.492	2.797	3.091	3.467	3.745
25	.684	.856	1.058	1.316	1.708	2.060	2.167	2.485	2.787	3.078	3.450	3.725
26	.684	.856	1.058	1.315	1.706	2.056	2.162	2.479	2.779	3.067	3.435	3.707
27	.684	.855	1.057	1.314	1.703	2.052	2.158	2.473	2.771	3.057	3.421	3.690
28	.683	.855	1.056	1.313	1.701	2.048	2.154	2.467	2.763	3.047	3.408	3.674
29	.683	.854	1.055	1.311	1.699	2.045	2.150	2.462	2.756	3.038	3.396	3.659
30	.683	.854	1.055	1.310	1.697	2.042	2.147	2.457	2.750	3.030	3.385	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.123	2.423	2.704	2.971	3.307	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.109	2.403	2.678	2.937	3.261	3.496
60	.679	.848	1.045	1.296	1.671	2.000	2.099	2.390	2.660	2.915	3.232	3.460
80	.678	.846	1.043	1.292	1.664	1.990	2.088	2.374	2.639	2.887	3.195	3.416
100	.677	.845	1.042	1.290	1.660	1.984	2.081	2.364	2.626	2.871	3.174	3.390
1000	.675	.842	1.037	1.282	1.646	1.962	2.056	2.330	2.581	2.813	3.098	3.300
∞	.674	.841	1.036	1.282	1.645	1.960	2.054	2.326	2.576	2.807	3.091	3.291
	50%	60%	70%	80%	90%	95%	96%	98%	99%	99.5%	99.8%	99.9%
	Confidence level C											

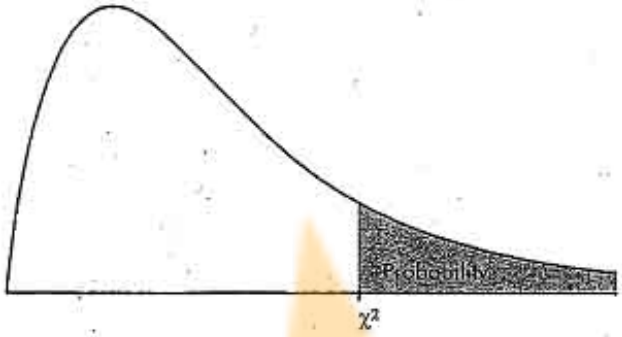


TABLE C: χ^2 CRITICAL VALUES

	Tail probability p										
df	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001
1	1.32	1.64	2.07	2.71	3.84	5.02	5.41	6.63	7.88	9.14	10.83
2	2.77	3.22	3.79	4.61	5.99	7.38	7.82	9.21	10.60	11.98	13.82
3	4.11	4.64	5.32	6.25	7.81	9.35	9.84	11.34	12.84	14.32	16.27
4	5.39	5.99	6.74	7.78	9.49	11.14	11.67	13.28	14.86	16.42	18.47
5	6.63	7.29	8.12	9.24	11.07	12.83	13.39	15.09	16.75	18.39	20.51
6	7.84	8.56	9.45	10.64	12.59	14.45	15.03	16.81	18.55	20.25	22.46
7	9.04	9.80	10.75	12.02	14.07	16.01	16.62	18.48	20.28	22.04	24.32
8	10.22	11.03	12.03	13.36	15.51	17.53	18.17	20.09	21.95	23.77	26.12
9	11.39	12.24	13.29	14.68	16.92	19.02	19.68	21.67	23.59	25.46	27.88
10	12.55	13.44	14.53	15.99	18.31	20.48	21.16	23.21	25.19	27.11	29.59
11	13.70	14.63	15.77	17.28	19.68	21.92	22.62	24.72	26.76	28.73	31.26
12	14.85	15.81	16.99	18.55	21.03	23.34	24.05	26.22	28.30	30.32	32.91
13	15.98	16.98	18.20	19.81	22.36	24.74	25.47	27.69	29.82	31.88	34.53
14	17.12	18.15	19.41	21.06	23.68	26.12	26.87	29.14	31.32	33.43	36.12
15	18.25	19.31	20.60	22.31	25.00	27.49	28.26	30.58	32.80	34.95	37.70
16	19.37	20.47	21.79	23.54	26.30	28.85	29.63	32.00	34.27	36.46	39.25
17	20.49	21.61	22.98	24.77	27.59	30.19	31.00	33.41	35.72	37.95	40.79
18	21.60	22.76	24.16	25.99	28.87	31.53	32.35	34.81	37.16	39.42	42.31
19	22.72	23.90	25.33	27.20	30.14	32.85	33.69	36.19	38.58	40.88	43.82
20	23.83	25.04	26.50	28.41	31.41	34.17	35.02	37.57	40.00	42.34	45.31
21	24.93	26.17	27.66	29.62	32.67	35.48	36.34	38.93	41.40	43.78	46.80
22	26.04	27.30	28.82	30.81	33.92	36.78	37.66	40.29	42.80	45.20	48.27
23	27.14	28.43	29.98	32.01	35.17	38.08	38.97	41.64	44.18	46.62	49.73
24	28.24	29.55	31.13	33.20	36.42	39.36	40.27	42.98	45.56	48.03	51.18
25	29.34	30.68	32.28	34.38	37.65	40.65	41.57	44.31	46.93	49.44	52.62
26	30.43	31.79	33.43	35.56	38.89	41.92	42.86	45.64	48.29	50.83	54.05
27	31.53	32.91	34.57	36.74	40.11	43.19	44.14	46.96	49.64	52.22	55.48
28	32.62	34.03	35.71	37.92	41.34	44.46	45.42	48.28	50.99	53.59	56.89
29	33.71	35.14	36.85	39.09	42.56	45.72	46.69	49.59	52.34	54.97	58.30
30	34.80	36.25	37.99	40.26	43.77	46.98	47.96	50.89	53.67	56.33	59.70
40	45.62	47.27	49.24	51.81	55.76	59.34	60.44	63.69	66.77	69.70	73.40
50	56.33	58.16	60.35	63.17	67.50	71.42	72.61	76.15	79.49	82.66	86.66
60	66.98	68.97	71.34	74.40	79.08	83.30	84.58	88.38	91.95	95.34	99.61
80	88.13	90.41	93.11	96.58	101.9	106.6	108.1	112.3	116.3	120.1	124.8
100	109.1	111.7	114.7	118.5	124.3	129.6	131.1	135.8	140.2	144.3	149.4

Linear algebra for Data Science

By Dr Mohammed Habeeb Vulla

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \in \begin{bmatrix} \mathbb{R} & \mathbb{R} \\ \mathbb{R} & \mathbb{R} \\ \mathbb{R} & \mathbb{R} \end{bmatrix} \equiv \mathbb{R}^{3\times 2}.$$

The purpose of this document is to introduce you to the mathematical operations that we can perform on vectors and matrices and to give you a feel of the power of linear algebra. Many problems in science, business, and technology can be described in terms of vectors and matrices so it is important that you understand how to work with these.

Prerequisites

The only prerequisite for this tutorial is a basic understanding of high school math concepts[1] like numbers, variables, equations, and the fundamental arithmetic operations on real numbers: addition (denoted $+$), subtraction (denoted $-$), multiplication (denoted implicitly), and division (fractions).

You should also be familiar with *functions* that take real numbers as inputs and give real numbers as outputs, $f : \mathbb{R} \to \mathbb{R}$. Recall that, by definition, the *inverse function* f^{-1} *undoes* the effect of f. If you are given $f(x)$ and you want to find x, you can use the inverse function as follows: $f^{-1}(f(x)) = x$. For example, the function $f(x) = \ln(x)$ has the inverse $f^{-1}(x) = e^x$, and the inverse of $g(x) = \sqrt{x}$ is $g^{-1}(x) = x^2$.

II. Definitions

A. Vector operations

We now define the math operations for vectors. The operations we can perform on vectors $\vec{u} = (u_1, u_2, u_3)$ and $\vec{v} = (v_1, v_2, v_3)$ are: addition, subtraction, scaling, norm (length), dot product, and cross product:

$$\begin{aligned} \vec{u} + \vec{v} &= (u_1 + v_1, u_2 + v_2, u_3 + v_3) \\ \vec{u} - \vec{v} &= (u_1 - v_1, u_2 - v_2, u_3 - v_3) \\ \alpha\vec{u} &= (\alpha u_1, \alpha u_2, \alpha u_3) \\ ||\vec{u}|| &= \sqrt{u_1^2 + u_2^2 + u_3^2} \\ \vec{u} \cdot \vec{v} &= u_1 v_1 + u_2 v_2 + u_3 v_3 \\ \vec{u} \times \vec{v} &= (u_2 v_3 - u_3 v_2,\ u_3 v_1 - u_1 v_3,\ u_1 v_2 - u_2 v_1) \end{aligned}$$

The dot product and the cross product of two vectors can also be described in terms of the angle θ between the two vectors. The formula for the dot product of the vectors is $\vec{u} \cdot \vec{v} = \|\vec{u}\|\|\vec{v}\| \cos\theta$. We say two vectors $\vec{u}$ and $\vec{v}$ are *orthogonal* if the angle between them is $90°$. The dot product of orthogonal vectors is zero: $\vec{u} \cdot \vec{v} = \|\vec{u}\|\|\vec{v}\| \cos(90°) = 0$.

The *norm* of the cross product is given by $\|\vec{u} \times \vec{v}\| = \|\vec{u}\|\|\vec{v}\| \sin\theta$. The cross product is not commutative: $\vec{u} \times \vec{v} \neq \vec{v} \times \vec{u}$, in fact $\vec{u} \times \vec{v} = -\vec{v} \times \vec{u}$.

B. Matrix operations

We denote by A the matrix as a whole and refer to its entries as a_{ij}. The mathematical operations defined for matrices are the following:

- addition (denoted $+$)

$$C = A + B \qquad \Leftrightarrow \qquad c_{ij} = a_{ij} + b_{ij}.$$

- subtraction (the inverse of addition)
- matrix product. The product of matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times \ell}$ is another matrix $C \in \mathbb{R}^{m\times \ell}$ given by the formula

$$C = AB \qquad \Leftrightarrow \qquad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$

- matrix inverse (denoted A^{-1})
- matrix transpose (denoted $^{\mathsf{T}}$):

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \end{bmatrix}^{\mathsf{T}} = \begin{bmatrix} \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \end{bmatrix}.$$

- matrix trace: $\mathrm{Tr}[A] \equiv \sum_{i=1}^{n} a_{ii}$
- determinant (denoted $\det(A)$ or $|A|$)

Note that the matrix product is not a commutative operation: $AB \neq BA$.

C. Matrix-vector product

The matrix-vector product is an important special case of the matrix-matrix product. The product of a 3×2 matrix A and the 2×1 column vector $\vec{x}$ results in a 3×1 vector $\vec{y}$ given by:

$$\vec{y} = A\vec{x} \qquad \Leftrightarrow \qquad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \\ a_{31}x_1 + a_{32}x_2 \end{bmatrix}$$

$$= x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix} \tag{C}$$

$$= \begin{bmatrix} (a_{11}, a_{12}) \cdot \vec{x} \\ (a_{21}, a_{22}) \cdot \vec{x} \\ (a_{31}, a_{32}) \cdot \vec{x} \end{bmatrix}. \tag{R}$$

There are two[2] fundamentally different yet equivalent ways to interpret the matrix-vector product. In the column picture, **(C)**, the multiplication of the matrix A by the vector $\vec{x}$ produces a **linear combination of the columns of the matrix**: $\vec{y} = A\vec{x} = x_1 A_{[:,1]} + x_2 A_{[:,2]}$, where $A_{[:,1]}$ and $A_{[:,2]}$ are the first and second columns of the matrix A.

In the row picture, **(R)**, multiplication of the matrix A by the vector $\vec{x}$ produces a column vector with coefficients equal to the **dot products of rows of the matrix** with the vector $\vec{x}$.

D. Linear transformations

The matrix-vector product is used to define the notion of a *linear transformation*, which is one of the key notions in the study of linear algebra. Multiplication by a matrix $A \in \mathbb{R}^{m\times n}$ can be thought of as computing a *linear transformation* T_A that takes n-vectors as inputs and produces m-vectors as outputs:

$$T_A : \mathbb{R}^n \to \mathbb{R}^m.$$

[1] Dr.Mohammed Habeeb Vulla

Instead of writing $\vec{y} = T_A(\vec{x})$ for the linear transformation T_A applied to the vector $\vec{x}$, we simply write $\vec{y} = A\vec{x}$. Applying the linear transformation T_A to the vector $\vec{x}$ corresponds to the product of the matrix A and the column vector $\vec{x}$. We say T_A is *represented by* the matrix A.

You can think of linear transformations as "vector functions" and describe their properties in analogy with the regular functions you are familiar with:

function $f : \mathbb{R} \to \mathbb{R}$	$\Leftrightarrow$	linear transformation $T_A : \mathbb{R}^n \to \mathbb{R}^m$
input $x \in \mathbb{R}$	$\Leftrightarrow$	input $\vec{x} \in \mathbb{R}^n$
output $f(x)$	$\Leftrightarrow$	output $T_A(\vec{x}) = A\vec{x} \in \mathbb{R}^m$
$g \circ f = g(f(x))$	$\Leftrightarrow$	$T_B(T_A(\vec{x})) = BA\vec{x}$
function inverse f^{-1}	$\Leftrightarrow$	matrix inverse A^{-1}
zeros of f	$\Leftrightarrow$	$\mathcal{N}(A) \equiv$ null space of A
range of f	$\Leftrightarrow$	$\mathcal{C}(A) \equiv$ column space of A = range of T_A

Note that the combined effect of applying the transformation T_A followed by T_B on the input vector $\vec{x}$ is equivalent to the matrix product $BA\vec{x}$.

E. Fundamental vector spaces

A *vector space* consists of a set of vectors and all linear combinations of these vectors. For example the vector space $\mathcal{S} = \text{span}\{\vec{v}_1, \vec{v}_2\}$ consists of all vectors of the form $\vec{v} = \alpha\vec{v}_1 + \beta\vec{v}_2$, where α and β are real numbers. We now define three fundamental vector spaces associated with a matrix A.

The *column space* of a matrix A is the set of vectors that can be produced as linear combinations of the columns of the matrix A:

$$\mathcal{C}(A) \equiv \{\vec{y} \in \mathbb{R}^m \mid \vec{y} = A\vec{x} \text{ for some } \vec{x} \in \mathbb{R}^n\}.$$

The column space is the *range* of the linear transformation T_A (the set of possible outputs). You can convince yourself of this fact by reviewing the definition of the matrix-vector product in the column picture **(C)**. The vector $A\vec{x}$ contains x_1 times the 1st column of A, x_2 times the 2nd column of A, etc. Varying over all possible inputs $\vec{x}$, we obtain all possible linear combinations of the columns of A, hence the name "column space."

The *null space* $\mathcal{N}(A)$ of a matrix $A \in \mathbb{R}^{m \times n}$ consists of all the vectors that the matrix A sends to the zero vector:

$$\mathcal{N}(A) \equiv \{\vec{x} \in \mathbb{R}^n \mid A\vec{x} = \vec{0}\}.$$

The vectors in the null space are *orthogonal* to all the rows of the matrix. We can see this from the row picture **(R)**: the output vectors is $\vec{0}$ if and only if the input vector $\vec{x}$ is orthogonal to all the rows of A.

The *row space* of a matrix A, denoted $\mathcal{R}(A)$, is the set of linear combinations of the rows of A. The row space $\mathcal{R}(A)$ is the orthogonal complement of the null space $\mathcal{N}(A)$. This means that for all vectors $\vec{v} \in \mathcal{R}(A)$ and all vectors $\vec{w} \in \mathcal{N}(A)$, we have $\vec{v} \cdot \vec{w} = 0$. Together, the null space and the row space form the domain of the transformation T_A, $\mathbb{R}^n = \mathcal{N}(A) \oplus \mathcal{R}(A)$, where $\oplus$ stands for *orthogonal direct sum*.

F. Matrix inverse

By definition, the inverse matrix A^{-1} *undoes* the effects of the matrix A. The cumulative effect of applying A^{-1} after A is the identity matrix $\mathbb{1}$:

$$A^{-1}A = \mathbb{1} \equiv \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}.$$

The identity matrix (ones on the diagonal and zeros everywhere else) corresponds to the identity transformation: $T_{\mathbb{1}}(\vec{x}) = \mathbb{1}\vec{x} = \vec{x}$, for all $\vec{x}$.

The matrix inverse is useful for solving matrix equations. Whenever we want to get rid of the matrix A in some matrix equation, we can "hit" A with its inverse A^{-1} to make it disappear. For example, to solve for the matrix X in the equation $XA = B$, multiply both sides of the equation by A^{-1} from the right: $X = BA^{-1}$. To solve for X in $ABCXD = E$, multiply both sides of the equation by D^{-1} on the right and by A^{-1}, B^{-1} and C^{-1} (in that order) from the left: $X = C^{-1}B^{-1}A^{-1}ED^{-1}$.

III. Computational linear algebra

Okay, I hear what you are saying "Dude, enough with the theory talk, let's see some calculations." In this section we'll look at one of the fundamental algorithms of linear algebra called Gauss–Jordan elimination.

A. Solving systems of equations

Suppose we're asked to solve the following system of equations:

$$\begin{aligned} 1x_1 + 2x_2 &= 5, \\ 3x_1 + 9x_2 &= 21. \end{aligned} \tag{1}$$

Without a knowledge of linear algebra, we could use substitution, elimination, or subtraction to find the values of the two unknowns x_1 and x_2.

Gauss–Jordan elimination is a systematic procedure for solving systems of equations based the following *row operations*:

α) Adding a multiple of one row to another row
β) Swapping two rows
γ) Multiplying a row by a constant

These row operations allow us to simplify the system of equations without changing their solution.

To illustrate the Gauss–Jordan elimination procedure, we'll now show the sequence of row operations required to solve the system of linear equations described above. We start by constructing an *augmented matrix* as follows:

$$\left[\begin{array}{cc|c} \mathbf{1} & 2 & 5 \\ 3 & 9 & 21 \end{array}\right].$$

The first column in the augmented matrix corresponds to the coefficients of the variable x_1, the second column corresponds to the coefficients of x_2, and the third column contains the constants from the right-hand side.

The Gauss-Jordan elimination procedure consists of two phases. During the first phase, we proceed left-to-right by choosing a row with a leading one in the leftmost column (called a *pivot*) and systematically subtracting that row from all rows below it to get zeros below in the entire column. In the second phase, we start with the rightmost pivot and use it to eliminate all the numbers above it in the same column. Let's see this in action.

1) The first step is to use the pivot in the first column to eliminate the variable x_1 in the second row. We do this by subtracting three times the first row from the second row, denoted $R_2 \leftarrow R_2 - 3R_1$,
$$\left[\begin{array}{cc|c} \mathbf{1} & 2 & 5 \\ 0 & 3 & 6 \end{array}\right].$$
2) Next, we create a pivot in the second row using $R_2 \leftarrow \frac{1}{3}R_2$:
$$\left[\begin{array}{cc|c} 1 & 2 & 5 \\ 0 & \mathbf{1} & 2 \end{array}\right].$$
3) We now start the backward phase and eliminate the second variable from the first row. We do this by subtracting two times the second row from the first row $R_1 \leftarrow R_1 - 2R_2$:
$$\left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 2 \end{array}\right].$$
The matrix is now in *reduced row echelon form* (RREF), which is its "simplest" form it could be in. The solutions are: $x_1 = 1$, $x_2 = 2$.

B. Systems of equations as matrix equations

We will now discuss another approach for solving the system of equations. Using the definition of the matrix-vector product, we can express this system of equations (1) as a matrix equation:

$$\begin{bmatrix} 1 & 2 \\ 3 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 21 \end{bmatrix}.$$

This matrix equation had the form $A\vec{x} = \vec{b}$, where A is a 2×2 matrix, $\vec{x}$ is the vector of unknowns, and $\vec{b}$ is a vector of constants. We can solve for $\vec{x}$ by multiplying both sides of the equation by the matrix inverse A^{-1}:

$$A^{-1}A\vec{x} = \mathbb{1}\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = A^{-1}\vec{b} = \begin{bmatrix} 3 & -\frac{2}{3} \\ -1 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 5 \\ 21 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

But how did we know what the inverse matrix A^{-1} is?

IV. Computing the inverse of a matrix

In this section we'll look at several different approaches for computing the inverse of a matrix. The matrix inverse is *unique* so no matter which method we use to find the inverse, we'll always obtain the same answer.

A. Using row operations

One approach for computing the inverse is to use the Gauss–Jordan elimination procedure. Start by creating an array containing the entries of the matrix A on the left side and the identity matrix on the right side:

$$\left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 3 & 9 & 0 & 1 \end{array}\right].$$

Now we perform the Gauss-Jordan elimination procedure on this array.

1) The first row operation is to subtract three times the first row from the second row: $R_2 \leftarrow R_2 - 3R_1$. We obtain:
$$\left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 0 & 3 & -3 & 1 \end{array}\right].$$
2) The second row operation is divide the second row by 3: $R_2 \leftarrow \frac{1}{3}R_2$
$$\left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 0 & 1 & -1 & \frac{1}{3} \end{array}\right].$$
3) The third row operation is $R_1 \leftarrow R_1 - 2R_2$
$$\left[\begin{array}{cc|cc} 1 & 0 & 3 & -\frac{2}{3} \\ 0 & 1 & -1 & \frac{1}{3} \end{array}\right].$$

The array is now in reduced row echelon form (RREF). The inverse matrix appears on the right side of the array.

Observe that the sequence of row operations we used to solve the specific system of equations in $A\vec{x} = \vec{b}$ in the previous section are the same as the row operations we used in this section to find the inverse matrix. Indeed, in both cases the combined effect of the three row operations is to "undo" the effects of A. The right side of the 2×4 array is simply a convenient way to record this sequence of operations and thus obtain A^{-1}.

B. Using elementary matrices

Every row operation we perform on a matrix is equivalent to a left-multiplication by an *elementary matrix*. There are three types of elementary matrices in correspondence with the three types of row operations:

$$\mathcal{R}_\alpha : R_1 \leftarrow R_1 + mR_2 \quad \Leftrightarrow \quad E_\alpha = \begin{bmatrix} 1 & m \\ 0 & 1 \end{bmatrix}$$

$$\mathcal{R}_\beta : R_1 \leftrightarrow R_2 \quad \Leftrightarrow \quad E_\beta = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\mathcal{R}_\gamma : R_1 \leftarrow mR_1 \quad \Leftrightarrow \quad E_\gamma = \begin{bmatrix} m & 0 \\ 0 & 1 \end{bmatrix}$$

Let's revisit the row operations we used to find A^{-1} in the above section representing each row operation as an elementary matrix multiplication.

1) The first row operation $R_2 \leftarrow R_2 - 3R_1$ corresponds to a multiplication by the elementary matrix E_1:
$$E_1 A = \begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}.$$
2) The second row operation $R_2 \leftarrow \frac{1}{3}R_2$ corresponds to a matrix E_2:
$$E_2(E_1 A) = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}.$$
3) The final step, $R_1 \leftarrow R_1 - 2R_2$, corresponds to the matrix E_3:
$$E_3(E_2 E_1 A) = \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Note that $E_3 E_2 E_1 A = \mathbb{1}$, so the product $E_3 E_2 E_1$ must be equal to A^{-1}:

$$A^{-1} = E_3 E_2 E_1 = \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -\frac{2}{3} \\ -1 & \frac{1}{3} \end{bmatrix}.$$

The elementary matrix approach teaches us that every invertible matrix can be decomposed as the product of elementary matrices. Since we know $A^{-1} = E_3 E_2 E_1$ then $A = (A^{-1})^{-1} = (E_3 E_2 E_1)^{-1} = E_1^{-1} E_2^{-1} E_3^{-1}$.

C. Using a computer

The last (and most practical) approach for finding the inverse of a matrix is to use a computer algebra system like the one at `live.sympy.org`.

```
>>> A = Matrix( [[1,2],[3,9]] )            # define A
    [1, 2]
    [3, 9]

>>> A.inv()                 # calls the inv method on A
    [ 3, -2/3]
    [-1,  1/3]
```

You can use `sympy` to "check" your answers on homework problems.

V. Other topics

We'll now discuss a number of other important topics of linear algebra.

A. Basis

Intuitively, a basis is any set of vectors that can be used as a coordinate system for a vector space. You are certainly familiar with the standard basis for the xy-plane that is made up of two orthogonal axes: the x-axis and the y-axis. A vector $\vec{v}$ can be described as a coordinate pair (v_x, v_y) with respect to these axes, or equivalently as $\vec{v} = v_x\hat{\imath} + v_y\hat{\jmath}$, where $\hat{\imath} \equiv (1, 0)$ and $\hat{\jmath} \equiv (0, 1)$ are unit vectors that point along the x-axis and y-axis respectively. However, other coordinate systems are also possible.

Definition (Basis). *A basis for a n-dimensional vector space $\mathcal{S}$ is any set of n linearly independent vectors that are part of $\mathcal{S}$.*

Any set of two linearly independent vectors $\{\hat{e}_1, \hat{e}_2\}$ can serve as a basis for $\mathbb{R}^2$. We can write any vector $\vec{v} \in \mathbb{R}^2$ as a linear combination of these basis vectors $\vec{v} = v_1\hat{e}_1 + v_2\hat{e}_2$.

Note the *same* vector $\vec{v}$ corresponds to different coordinate pairs depending on the basis used: $\vec{v} = (v_x, v_y)$ in the standard basis $B_s \equiv \{\hat{\imath}, \hat{\jmath}\}$, and $\vec{v} = (v_1, v_2)$ in the basis $B_e \equiv \{\hat{e}_1, \hat{e}_2\}$. Therefore, it is important to keep in mind the basis with respect to which the coefficients are taken, and if necessary specify the basis as a subscript, e.g., $(v_x, v_y)_{B_s}$ or $(v_1, v_2)_{B_e}$.

Converting a coordinate vector from the basis B_e to the basis B_s is performed as a multiplication by a *change of basis* matrix:

$$\left[\vec{v}\right]_{B_s} = {}_{B_s}\left[\ \mathbb{1}\ \right]_{B_e} \left[\vec{v}\right]_{B_e} \quad \Leftrightarrow \quad \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \hat{\imath}\cdot\hat{e}_1 & \hat{\imath}\cdot\hat{e}_2 \\ \hat{\jmath}\cdot\hat{e}_1 & \hat{\jmath}\cdot\hat{e}_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}.$$

Note the change of basis matrix is actually an identity transformation. The vector $\vec{v}$ remains unchanged—it is simply expressed with respect to a new coordinate system. The change of basis from the B_s-basis to the B_e-basis is accomplished using the inverse matrix: ${}_{B_e}[\mathbb{1}]_{B_s} = ({}_{B_s}[\mathbb{1}]_{B_e})^{-1}$.

B. Matrix representations of linear transformations

Bases play an important role in the representation of linear transformations $T\colon \mathbb{R}^n \to \mathbb{R}^m$. To fully describe the matrix that corresponds to some linear transformation T, it is sufficient to know the effects of T to the n vectors of the standard basis for the input space. For a linear transformation $T\colon \mathbb{R}^2 \to \mathbb{R}^2$, the matrix representation corresponds to

$$M_T = \begin{bmatrix} | & | \\ T(\hat{\imath}) & T(\hat{\jmath}) \\ | & | \end{bmatrix} \in \mathbb{R}^{2\times 2}.$$

As a first example, consider the transformation Π_x which projects vectors onto the x-axis. For any vector $\vec{v} = (v_x, v_y)$, we have $\Pi_x(\vec{v}) = (v_x, 0)$. The matrix representation of Π_x is

$$M_{\Pi_x} = \begin{bmatrix} \Pi_x\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) & \Pi_x\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.$$

As a second example, let's find the matrix representation of R_θ, the counterclockwise rotation by the angle θ:

$$M_{R_\theta} = \begin{bmatrix} R_\theta\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) & R_\theta\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

The first column of M_{R_θ} shows that R_θ maps the vector $\hat{\imath} \equiv 1\angle 0$ to the vector $1\angle\theta = (\cos\theta, \sin\theta)^\mathsf{T}$. The second column shows that R_θ maps the vector $\hat{\jmath} = 1\angle\frac{\pi}{2}$ to the vector $1\angle(\frac{\pi}{2} + \theta) = (-\sin\theta, \cos\theta)^\mathsf{T}$.

C. Dimension and bases for vector spaces

The *dimension* of a vector space is defined as the number of vectors in a basis for that vector space. Consider the following vector space $\mathcal{S} = \text{span}\{(1,0,0),(0,1,0),(1,1,0)\}$. Seeing that the space is described by three vectors, we might think that $\mathcal{S}$ is 3-dimensional. This is not the case, however, since the three vectors are not linearly independent so they don't form a basis for $\mathcal{S}$. Two vectors are sufficient to describe any vector in $\mathcal{S}$; we can write $\mathcal{S} = \text{span}\{(1,0,0),(0,1,0)\}$, and we see these two vectors are linearly independent so they form a basis and $\dim(\mathcal{S}) = 2$.

There is a general procedure for finding a basis for a vector space. Suppose you are given a description of a vector space in terms of m vectors $\mathcal{V} = \text{span}\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_m\}$ and you are asked to find a basis for $\mathcal{V}$ and the dimension of $\mathcal{V}$. To find a basis for $\mathcal{V}$, you must find a set of linearly independent vectors that span $\mathcal{V}$. We can use the Gauss–Jordan elimination procedure to accomplish this task. Write the vectors $\vec{v}_i$ as the rows of a matrix M. The vector space $\mathcal{V}$ corresponds to the row space of the matrix M. Next, use row operations to find the reduced row echelon form (RREF) of the matrix M. Since row operations do not change the row space of the matrix, the row space of reduced row echelon form of the matrix M is the same as the row space of the original set of vectors. The nonzero rows in the RREF of the matrix form a basis for vector space $\mathcal{V}$ and the numbers of nonzero rows is the dimension of $\mathcal{V}$.

D. Row space, columns space, and rank of a matrix

Recall the fundamental vector spaces for matrices that we defined in Section II-E: the column space $\mathcal{C}(A)$, the null space $\mathcal{N}(A)$, and the row space $\mathcal{R}(A)$. A standard linear algebra exam question is to give you a certain matrix A and ask you to find the dimension and a basis for each of its fundamental spaces.

In the previous section we described a procedure based on Gauss–Jordan elimination which can be used "distill" a set of linearly independent vectors which form a basis for the row space $\mathcal{R}(A)$. We will now illustrate this procedure with an example, and also show how to use the RREF of the matrix A to find bases for $\mathcal{C}(A)$ and $\mathcal{N}(A)$.

Consider the following matrix and its reduced row echelon form:

$$A = \begin{bmatrix} 1 & 3 & 3 & 3 \\ 2 & 6 & 7 & 6 \\ 3 & 9 & 9 & 10 \end{bmatrix} \qquad \text{rref}(A) = \begin{bmatrix} \mathbf{1} & 3 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix}.$$

The reduced row echelon form of the matrix A contains three pivots. The locations of the pivots will play an important role in the following steps.

The vectors $\{(1,3,0,0),(0,0,1,0),(0,0,0,1)\}$ form a basis for $\mathcal{R}(A)$.

To find a basis for the column space $\mathcal{C}(A)$ of the matrix A we need to find which of the columns of A are linearly independent. We can do this by identifying the columns which contain the leading ones in $\text{rref}(A)$. The corresponding columns in the original matrix form a basis for the column space of A. Looking at $\text{rref}(A)$ we see the first, third, and fourth columns of the matrix are linearly independent so the vectors $\{(1,2,3)^\mathsf{T}, (3,7,9)^\mathsf{T}, (3,6,10)^\mathsf{T}\}$ form a basis for $\mathcal{C}(A)$.

Now let's find a basis for the null space, $\mathcal{N}(A) \equiv \{\vec{x} \in \mathbb{R}^4 \mid A\vec{x} = \vec{0}\}$. The second column does not contain a pivot, therefore it corresponds to a *free variable*, which we will denote s. We are looking for a vector with three unknowns and one free variable $(x_1, s, x_3, x_4)^\mathsf{T}$ that obeys the conditions:

$$\begin{bmatrix} \mathbf{1} & 3 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix} \begin{bmatrix} x_1 \\ s \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \Rightarrow \quad \begin{aligned} 1x_1 + 3s &= 0 \\ 1x_3 &= 0 \\ 1x_4 &= 0 \end{aligned}$$

Let's express the unknowns x_1, x_3, and x_4 in terms of the free variable s. We immediately see that $x_3 = 0$ and $x_4 = 0$, and we can write $x_1 = -3s$. Therefore, any vector of the form $(-3s, s, 0, 0)$, for any $s \in \mathbb{R}$, is in the null space of A. We write $\mathcal{N}(A) = \text{span}\{(-3,1,0,0)^\mathsf{T}\}$.

Observe that the $\dim(\mathcal{C}(A)) = \dim(\mathcal{R}(A)) = 3$, this is known as the *rank* of the matrix A. Also, $\dim(\mathcal{R}(A)) + \dim(\mathcal{N}(A)) = 3 + 1 = 4$, which is the dimension of the input space of the linear transformation T_A.

E. Invertible matrix theorem

There is an important distinction between matrices that are invertible and those that are not as formalized by the following theorem.

Theorem. *For an $n \times n$ matrix A, the following statements are equivalent:*

1) *A is invertible*
2) *The RREF of A is the $n \times n$ identity matrix*
3) *The rank of the matrix is n*
4) *The row space of A is $\mathbb{R}^n$*
5) *The column space of A is $\mathbb{R}^n$*
6) *A doesn't have a null space (only the zero vector $\mathcal{N}(A) = \{\vec{0}\}$)*
7) *The determinant of A is nonzero $\det(A) \neq 0$*

For a given matrix A, the above statements are either all true or all false.

An invertible matrix A corresponds to a linear transformation T_A which maps the n-dimensional input vector space $\mathbb{R}^n$ to the n-dimensional output vector space $\mathbb{R}^n$ such that there exists an inverse transformation T_A^{-1} that can faithfully undo the effects of T_A.

On the other hand, an $n \times n$ matrix B that is not invertible maps the input vector space $\mathbb{R}^n$ to a subspace $\mathcal{C}(B) \subsetneq \mathbb{R}^n$ and has a nonempty null space. Once T_B sends a vector $\vec{w} \in \mathcal{N}(B)$ to the zero vector, there is no T_B^{-1} that can undo this operation.

F. Determinants

The determinant of a matrix, denoted $\det(A)$ or $|A|$, is a special way to combine the entries of a matrix that serves to check if a matrix is invertible or not. The determinant formulas for 2×2 and 3×3 matrices are

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}, \qquad \text{and}$$

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}.$$

If the $|A| = 0$ then A is not invertible. If $|A| \neq 0$ then A is invertible.

G. Eigenvalues and eigenvectors

The set of eigenvectors of a matrix is a special set of input vectors for which the action of the matrix is described as a simple *scaling*. When a matrix is multiplied by one of its eigenvectors the output is the same eigenvector multiplied by a constant $A\vec{e}_\lambda = \lambda\vec{e}_\lambda$. The constant λ is called an *eigenvalue* of A.

To find the eigenvalues of a matrix we start from the eigenvalue equation $A\vec{e}_\lambda = \lambda\vec{e}_\lambda$, insert the identity $\mathbb{1}$, and rewrite it as a null-space problem:

$$A\vec{e}_\lambda = \lambda\mathbb{1}\vec{e}_\lambda \qquad \Rightarrow \qquad (A - \lambda\mathbb{1})\,\vec{e}_\lambda = \vec{0}.$$

This equation will have a solution whenever $|A - \lambda\mathbb{1}| = 0$. The eigenvalues of $A \in \mathbb{R}^{n\times n}$, denoted $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, are the roots of the *characteristic polynomial* $p(\lambda) = |A - \lambda\mathbb{1}|$. The *eigenvectors* associated with the eigenvalue λ_i are the vectors in the null space of the matrix $(A - \lambda_i\mathbb{1})$.

Certain matrices can be written entirely in terms of their eigenvectors and their eigenvalues. Consider the matrix Λ that has the eigenvalues of the matrix A on the diagonal, and the matrix Q constructed from the eigenvectors of A as columns:

$$\Lambda = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix}, \quad Q = \begin{bmatrix} | & & | \\ \vec{e}_{\lambda_1} & \cdots & \vec{e}_{\lambda_n} \\ | & & | \end{bmatrix}, \quad \text{then} \quad A = Q\Lambda Q^{-1}.$$

Matrices that can be written this way are called diagonalizable.

The decomposition of a matrix into its eigenvalues and eigenvectors gives valuable insights into the properties of the matrix. Google's original `PageRank` algorithm for ranking webpages by "importance" can be formalized as an eigenvector calculation on the matrix of web hyperlinks.

Basics Of Probability for Data Science
By Dr.Mohammed Habeeb Vulla

Basic Probability

Introduction

The world is an uncertain place. Making predictions about something as seemingly mundane as tomorrow's weather, for example, is actually quite a difficult task. Even with the most advanced computers and models of the modern era, weather forecasters still cannot say with absolute certainty whether it will rain tomorrow. The best they can do is to report their best estimate of the *chance* that it will rain tomorrow. For example, if the forecasters are fairly confident that it will rain tomorrow, they might say that there is a 90% chance of rain. You have probably heard statements like this your entire life, but have you ever asked yourself what exactly it means to say that there is a 90% chance of rain?

Let us consider an even more basic example: tossing a coin. If the coin is fair, then it is just as likely to come up heads as it is to come up tails. In other words, if we were to repeatedly toss the coin many times, we would expect about about half of the tosses to be heads and and half to be tails. In this case, we say that the **probability** of getting a head is 1/2 or 0.5.

Note that when we say the probability of a head is 1/2, we are *not* claiming that any sequence of coin tosses will consist of exactly 50% heads. If we toss a fair coin ten times, it would not be surprising to observe 6 heads and 4 tails, or even 3 heads and 7 tails. But as we continue to toss the coin over and over again, we expect the long-run frequency of heads to get ever closer to 50%. In general, it is important in statistics to understand the distinction between *theoretical* and *empirical* quantities. Here, the true (theoretical) probability of a head was 1/2, but any realized (empirical) sequence of coin tosses may have more or less than exactly 50% heads. (See Figures 1 – 3.)

Now suppose instead that we were to toss an unusual coin with heads on both of its faces. Then every time we flip this coin we will observe a head — we say that the probability of a head is 1. The probability of a tail, on the other hand, is 0. Note that there is no way

Figure 1: The true probability of a head is 1/2 for a fair coin.

Figure 2: A sequence of 10 flips happened to contain 3 head. The empirical frequency of heads is thus 3/10, which is quite different from 1/2.

Figure 3: A sequence of 100 flips happened to contain 45 heads. The empirical frequency of heads is 45/100, which is much closer to 1/2.

we can further modify the coin to make flipping a head even more likely. Thus, *a probability is always a number between 0 and 1 inclusive.*

First Concepts

Terminology

When we later discuss examples that are more complicated than flipping a coin, it will be useful to have an established vocabulary for working with probabilities. A probabilistic **experiment** (such as tossing a coin or rolling a die) has several components. The **sample space** is the set of all possible **outcomes** in the experiment. We usually denote the sample space by Ω, the Greek capital letter "Omega." So in a coin toss experiment, the sample space is

$$\Omega = \{\mathsf{H}, \mathsf{T}\},$$

since there are only two possible outcomes: heads (H) or tails (T). Different experiments have different sample spaces. So if we instead consider an experiment in which we roll a standard six-sided die, the sample space is

$$\Omega = \{1, 2, 3, 4, 5, 6\}.$$

Collections of outcomes in the sample space Ω are called **events**, and we often use capital Roman letters to denote these collections. We might be interested in the event that we roll an even number, for example. If we call this event E, then

$$E = \{2, 4, 6\}.$$

Any subset of Ω is a valid event. In particular, one-element subsets are allowed, so we can speak of the event F of rolling a 4, $F = \{4\}$.

Assigning probabilities to dice rolls and coin flips

In a random experiment, every event gets assigned a probability. Notationally, if A is some event of interest, then $P(A)$ is the probability that A occurs. The probabilities in an experiment are not arbitrary; they must satisfy a set of rules or **axioms**. We first require that *all probabilities be nonnegative.* In other words, in an experiment with sample space Ω, it must be the case that

$$P(A) \geq 0 \tag{1}$$

for any event $A \subseteq \Omega$. This should make sense given that we've already said that a probability of 0 is assigned to an impossible event,

and there is no way for something to be less likely than something that is impossible!

The next axiom is that *the sum of the probabilities of all the outcomes in Ω must be 1.* We can restate this requirement by the equation

$$\sum_{\omega \in \Omega} P(\omega) = 1. \tag{2}$$

This rule can sometimes be used to deduce the probability of an outcome in certain experiments. Consider an experiment in which we roll a fair die, for example. Then each outcome (i.e. each face of the die) is equally likely. That is,

$$P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = a,$$

for some number a. Equation (2) now allows us to conclude

$$1 = \sum_{k=1}^{6} P(k) = \sum_{k=1}^{6} a = 6a,$$

so $a = 1/6$. In this example, we were able to use the symmetry of the experiment along with one of the probability axioms to determine the probability of rolling any number.

Once we know the probabilties of the outcomes in an experiment, we can compute the probability of any event. This is because *the probability of an event is the sum of the probabilities of the outcomes it comprises.* In other words, for an event $A \subseteq \Omega$, the probability of A is

$$P(A) = \sum_{\omega \in A} P(\omega). \tag{3}$$

To illustrate this equation, let us find the probability of rolling an even number, an event which we will denote by E. Since $E = \{2, 4, 6\}$, we simply add the probabilities of these three outcomes to obtain

$$\begin{aligned} P(E) &= \sum_{\omega \in E} P(\omega) \\ &= P(2) + P(4) + P(6) \\ &= \frac{1}{6} + \frac{1}{6} + \frac{1}{6} \\ &= \frac{1}{2}. \end{aligned}$$

What is the probability that we get at least one H?

Solution. One way to solve this problem is to add up the probabilities

of all outcomes that have at least one H. We would get

$$\begin{aligned} P(\text{flip at least one H}) &= P(\text{HH}) + P(\text{HT}) + P(\text{TH}) \\ &= p^2 + p \cdot (1-p) + (1-p) \cdot p \\ &= p^2 + 2 \cdot (p - p^2) \\ &= 2p - p^2 \\ &= p \cdot (2-p). \end{aligned}$$

Another way to do this is to find the probability that we **don't** flip at least one H, and subtract that probability from 1. This would give us the probability that we **do** flip at least one H.

The only outcome in which we don't flip at least one H is if we flip T both times. We would then compute

$$P(\text{don't flip at least one H}) = P(\text{TT}) = (1-p)^2$$

Then to get the **complement** of this event, i.e. the event where we **do** flip at least one H, we subtract the above probability from 1. This gives us

$$\begin{aligned} P(\text{flip at least one H}) &= 1 - P(\text{don't flip at least one H}) \\ &= 1 - (1-p)^2 \\ &= 1 - (1 - 2p + p^2) \\ &= 2p - p^2 \\ &= p \cdot (2-p). \end{aligned}$$

Wowee! Both methods for solving this problem gave the same answer. Notice that in the second calculation, we had to sum up fewer probabilities to get the answer. It can often be the case that computing the probability of the complement of an event and subtracting that from 1 to find the probability of the original event requires less work. □

Independence

If two events A and B don't influence or give any information about the other, we say A and B are independent. Remember that this is not the same as saying A and B are disjoint. If A and B were disjoint, then given information that A happened, we would know with certainty that B did *not* happen. Hence if A and B are disjoint they could never be independent. The mathematical statement of independent events is given below.

Definition 0.0.1. *Let A and B both be subsets of our sample space Ω. Then we say A and B are independent if*

$$P(A \cap B) = P(A)P(B)$$

In other words, if the probability of the intersection factors into the product of the probabilities of the individual events, they are independent.

We haven't defined set intersection in this section, but it is defined in the set theory chapter. The $\cap$ symbol represents *A and B* happening, i.e. the intersection of the events.

Example 0.0.1. *Returning to our double coin flip example, our sample space was*

$$\Omega = \{HH, HT, TH, TT\}$$

Define the events

$$A \doteq \{\textit{first flip heads}\} = \{HH, HT\}$$
$$B \doteq \{\textit{second flip heads}\} = \{HT, TT\}$$

Notation: *We write the sign $\doteq$ to represent that we are defining something. In the above expression, we are defining the arbitrary symbols A and B to represent events.*

Intuitively, we suspect that A and B are independent events, since the first flip has no effect on the outcome of the second flip. This intuition aligns with the definition given above, as

$$P(A \cap B) = P(\{HT\}) = \frac{1}{4}$$

and

$$P(A) = P(B) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}.$$

We can verify that

$$P(A \cap B) = \frac{1}{4} = \frac{1}{2} \cdot \frac{1}{2} = P(A)P(B)$$

Hence A and B are independent. This may have seemed like a silly exercise, but in later chapters, we will encounter pairs of sets where it is not intuitively clear whether or not they are independent. In these cases, we can simply verify this mathematical definition to conclude independence.

Expectation

Consider the outcome of a single die roll, and call it X. A reasonable question one might ask is "What is the average value of X?". We define this notion of "average" as a weighted sum of outcomes.

Since X can take on 6 values, each with probability $\frac{1}{6}$, the weighted average of these outcomes should be

$$\begin{aligned}\text{Weighted Average} &= \frac{1}{6}\cdot 1 + \frac{1}{6}\cdot 2 + \frac{1}{6}\cdot 3 + \frac{1}{6}\cdot 4 + \frac{1}{6}\cdot 5 + \frac{1}{6}\cdot 6 \\ &= \frac{1}{6}\cdot(1+2+3+4+5+6) \\ &= \frac{21}{6} \\ &= 3.5\end{aligned}$$

This may seem dubious to some. How can the average roll be a non-integer value? The confusion lies in the interpretation of the phrase *average roll*. A more correct interpretation would be the long term average of the die rolls. Suppose we rolled the die many times, and recorded each roll. Then we took the average of all those rolls. This average would be the fraction of 1's, times 1, plus the fraction of 2's, times 2, plus the fraction of 3's, times 3, and so on. But this is exactly the computation we have done above! In the long run, the fraction of each of these outcomes is nothing but their probability, in this case, $\frac{1}{6}$ for each of the 6 outcomes.

From this very specific die rolling example, we can abstract the notion of the *average value* of a random quantity. The concept of average value is an important one in statistics, so much so that it even gets a special bold faced name. Below is the mathematical definition for the **expectation**, or average value, of a random quantity X.

Definition 0.0.2. *The **expected value**, or **expectation** of X, denoted by $E(X)$, is defined to be*

$$E(X) = \sum_{x \in X(\Omega)} xP(X = x)$$

This expression may look intimidating, but it is actually conveying a very simple set of instructions, the same ones we followed to compute the average value of X.

The $\sum$ sign means to sum over, and the indices of the items we are summing are denoted below the $\sum$ sign. The $\in$ symbol is shorthand for "contained in", so the expression below the $\sum$ is telling us to sum over all items *contained in* our sample space Ω. We can think of the expression to the right of the $\sum$ sign as the actual items we are summing, in this case, the weighted contribution of each item in our sample space.

The notation $X(\Omega)$ is used to deal with the fact that Ω may not be a set of numbers, so a weighted sum of elements in Ω isn't even well defined. For instance, in the case of a coin flip, how can we compute $\mathtt{H} \cdot \frac{1}{2} + \mathtt{T} \cdot \frac{1}{2}$? We would first need to assign *numerical values* to $\mathtt{H}$ and $\mathtt{T}$ in order to compute a meaningful expected value. For a coin flip we typically make the following assignments,

$$\mathtt{T} \mapsto 0$$
$$\mathtt{H} \mapsto 1$$

So when computing an expectation, the indices that we would sum over are contained in the set

$$X(\Omega) = \{0, 1\}$$

Let's use this set of instructions to compute the expected value for a coin flip.

Expectation of a Coin Flip

Now let X denote the value of a coin flip with bias p. That is, with probability p we flip $\mathtt{H}$, and in this case we say $X = 1$. Similarly, with probability $1 - p$ we flip $\mathtt{T}$, and in this case we say $X = 0$. The expected value of the random quantity X is then

$$\begin{aligned} E(X) &= \sum_{x \in X(\Omega)} xP(X = x) \\ &= \sum_{x \in \{0,1\}} xP(X = x) \\ &= 0 \cdot P(X = 0) + 1 \cdot P(X = 1) \\ &= 0 \cdot P(\mathtt{T}) + 1 \cdot P(\mathtt{H}) \\ &= 0 \cdot (1 - p) + 1 \cdot p \\ &= p \end{aligned}$$

So the expected value of this experiment is p. If we were flipping a fair coin, then $p = \frac{1}{2}$, so the average value of X would be $\frac{1}{2}$.

Again, we can never get an outcome that would yield $X = \frac{1}{2}$, but this is not the interpretation of the expectation of X. Remember, the correct interpretation is to consider what would happen if we flipped the coin many times, obtained a sequence of 0's and 1's, and took the average of those values. We would expect around half of the flips to give 0 and the other half to give 1, giving an average value of $\frac{1}{2}$.

Exercise 0.0.1. *Show the following properties of expectation.*

(a) If X and Y are two random variables, then

$$E(X + Y) = E(X) + E(Y)$$

(b) If X is a random variable and c is a constant, then

$$E(cX) = cE(X)$$

(c) If X and Y are independent random variables, then

$$E[XY] = E[X]E[Y]$$

Proof. For now, we will take (a) and (c) as a fact, since we don't know enough to prove them yet (and we haven't even defined independence of random variables!). (b) follows directly from the definition of expectation given above. □

Variance

The variance of a random variable X is a nonnegative number that summarizes on average how much X differs from its mean, or expectation. The first expression that comes to mind is

$$X - E(X)$$

i.e. the difference between X and its mean. This itself is a random variable, since even though EX is just a number, X is still random. Hence we would need to take an expectation to turn this expression into the average amount by which X differs from its expected value. This leads us to

$$E(X - EX)$$

This is almost the definition for variance. We require that the variance always be nonnegative, so the expression inside the expectation should always be ≥ 0. Instead of taking the expectation of the difference, we take the expectation of the squared difference.

Definition 0.0.3. *The **variance** of X, denoted by $Var(X)$ is defined*

$$Var(X) = E[(X - EX)^2]$$

Below we give and prove some useful properties of the variance.

Proposition 0.0.1. *If X is a random variable with mean EX and $c \in$ is a real number,*

(a) $Var(X) \geq 0$.

(b) $Var(cX) = c^2 Var(X)$.

(c) $Var(X) = E(X^2) - E(X)$.

(d) If X and Y are independent random variables, then

$$Var(X+Y) = Var(X) + Var(Y)$$

Proof.

(a) Since $(X - EX)^2 \geq 0$, its average is also ≥ 0. Hence $E[(X - EX)^2] \geq 0$.

(b) Going by the definition, we have

$$\begin{aligned}
\text{Var}(cX) &= E[(cX - E[cX])^2] \\
&= E[(cX - cEX)^2] \\
&= E[c^2(X - EX)^2] \\
&= c^2 E[(X - EX)^2] \\
&= c^2 \text{Var}(X)
\end{aligned}$$

(c) Expanding out the square in the definition of variance gives

$$\begin{aligned}
\text{Var}(X) &= E[(X - EX)^2] \\
&= E[X^2 - 2XEX + (EX)^2] \\
&= E[X^2] - E(2XEX) + E((EX)^2) \\
&= E[X^2] - 2EXEX + (EX)^2 \\
&= E[X^2] - (EX)^2
\end{aligned}$$

where the third equality comes from linearity of E (Exercise 2.3 (a)) and the fourth equality comes from Exercise 2.3 (b) and the fact that since EX and $(EX)^2$ are constants, their expectations are just EX and $(EX)^2$ respectively.

(d) By the definition of variance,

$$\begin{aligned}
\text{Var}(X+Y) &= E[(X+Y)^2] - (E[X+Y])^2 \\
&= E[X^2 + 2XY + Y^2] - \left((E[X])^2 + 2E[X]E[Y] + (E[Y])^2\right) \\
&= E[X^2] - (E[X])^2 + E[Y^2] - (E[Y]^2) + 2E[XY] - 2E[X]E[Y] \\
&= E[X^2] - (E[X])^2 + E[Y^2] - (E[Y]^2) \\
&= \text{Var}(X) + \text{Var}(Y)
\end{aligned}$$

where the fourth equality comes from the fact that if X and Y are independent, then $E[XY] = E[X]E[Y]$. Independence of random variables will be discussed in the "Random Variables" section, so don't worry if this proof doesn't make any sense to you yet.

□

Exercise 0.0.2. *Compute the variance of a die roll, i.e. a uniform random variable over the sample space* $\Omega = \{1, 2, 3, 4, 5, 6\}$.

Solution. Let X denote the outcome of the die roll. By definition, the variance is

$$\begin{aligned}
\text{Var}(X) &= E[(X - EX)]^2 \\
&= E(X^2) - (EX)^2 && \text{(Proposition 2.11 (c))} \\
&= \Big(\sum_{k=1}^{6} k^2 \cdot \frac{1}{6} \Big) - (3.5)^2 && \text{(Definition of Expectation)} \\
&= \frac{1}{6} \cdot (1 + 4 + 9 + 16 + 25 + 36) - 3.5^2 \\
&= \frac{1}{6} \cdot 91 - 3.5^2 \\
&\approx 2.92
\end{aligned}$$

□

Remark 0.0.1. *The square root of the variance is called the* ***standard deviation****.*

Markov's Inequality

Here we introduce an inequality that will be useful to us in the next section. Feel free to skip this section and return to it when you read "Chebyschev's inequality" and don't know what's going on.

Markov's inequality is a bound on the probability that a nonnegative random variable X exceeds some number a.

Theorem 0.0.1 (Markov's inequality)**.** *Suppose X is a nonnegative random variable and $a \in$ is a positive constant. Then*

$$P(X \geq a) \leq \frac{EX}{a}$$

Proof. By definition of expectation, we have

$$\begin{aligned}
EX &= \sum_{k \in X(\Omega)} kP(X = k) \\
&= \sum_{k \in X(\Omega) \text{ s.t. } k \geq a} kP(X = k) + \sum_{k \in X(\Omega) \text{ s.t. } k < a} kP(X = k) \\
&\geq \sum_{k \in X(\Omega) \text{ s.t. } k \geq a} kP(X = k) \\
&\geq \sum_{k \in X(\Omega) \text{ s.t. } k \geq a} aP(X = k) \\
&= a \sum_{k \in X(\Omega) \text{ s.t. } k \geq a} P(X = k) \\
&= aP(X \geq a)
\end{aligned}$$

where the first inequality follows from the fact that X is nonnegative and probabilities are nonnegative, and the second inequality follows from the fact that $k \geq a$ over the set $\{k \in X(\Omega) \text{ s.t. } k \geq a\}$.

Notation: "s.t." stands for "such that".

Dividing both sides by a, we recover

$$P(X \geq a) \leq \frac{EX}{a}$$

□

Corollary 0.0.1 (Chebyschev's inequality). *Let X be a random variable. Then*

$$P(|X - EX| > \epsilon) \leq \frac{Var(X)}{\epsilon^2}$$

Proof. This is marked as a corollary because we simply apply Markov's inequality to the nonnegative random variable $(X - EX)^2$. We then have

$$\begin{aligned} P(|X - EX| > \epsilon) &= P((X - EX)^2 > \epsilon^2) && \text{(statements are equivalent)} \\ &\leq \frac{E[(X - EX)^2]}{\epsilon^2} && \text{(Markov's inequality)} \\ &= \frac{\text{Var}(X)}{\epsilon^2} && \text{(definition of variance)} \end{aligned}$$

□

Estimation

One of the main reasons we do statistics is to make inferences about a population given data from a subset of that population. For example, suppose there are two candidates running for office. We could be interested in finding out the true proportion of the population that supports a particular political candidate. Instead of asking every single person in the country their preferred candidate, we could randomly select a couple thousand people from across the country and record their preference. We could then estimate the true proportion of the population that supports the candidate using this sample proportion. Since each person can only prefer one of two candidates, we can model this person's preference as a coin flip with bias $p =$ the true proportion that favors candidate 1.

Estimating the Bias of a Coin

Suppose now that we are again flipping a coin, this time with bias p. In other words, our coin can be thought of as a random quantity X defined

$$X = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1-p \end{cases}$$

where 1 represents `H` and 0 represents `T`. If we were just handed this coin, and told that it has some bias $0 \leq p \leq 1$, how would we estimate p? One way would be to flip the coin n times, count the number of heads we flipped, and divide that number by n. Letting X_i be the outcome of the i^{th} flip, our estimate, denoted $\hat{p}$, would be

$$\hat{p} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

As the number of samples n gets bigger, we would expect $\hat{p}$ to get closer and closer to the true value of p.

Estimating π

In the website's visualization, we are throwing darts uniformly at a square, and inside that square is a circle. If the side length of the square that inscribes the circle is L, then the radius of the circle is $R = \frac{L}{2}$, and its area is $A = \pi(\frac{L}{2})^2$. At the i^{th} dart throw, we can define

$$X_i = \begin{cases} 1 & \text{if the dart lands in the circle} \\ 0 & \text{otherwise} \end{cases}$$

The event “dart lands in the circle" has probability

$$p = \frac{\text{Area of Circle}}{\text{Area of Square}} = \frac{\pi\left(\frac{L}{2}\right)^2}{L^2} = \frac{\pi}{4}$$

So with probability $p = \frac{\pi}{4}$, a dart lands in the circle, and with probability $1 - \frac{\pi}{4}$, it doesn't.

By the previous section, we can estimate p using $\hat{p} = \frac{1}{n}\sum_{i=1}^{n} X_i$ so that for large enough n, we have

$$\hat{p} \approx p = \frac{\pi}{4}$$

so that rearranging for π yields

$$\pi \approx 4\hat{p}$$

Hence our approximation gets closer and closer to π as the number of sample $n \to \infty$ causing $\hat{p} \to p$.

Consistency of Estimators

What exactly do we mean by “closer and closer"? In this section, we describe the concept of **consistency** in order to make precise this notion of convergence. Our estimator in the last section, $4\hat{p}$ is itself random, since it depends on the n sample points we used to compute it. If we were to take a different set of n sample points, we would likely get a different estimate. Despite this randomness, intuitively we believe that as the number of samples n tends to infinity, the estimator $4\hat{p}$ will converge in some probabilistic sense, to π.

Another way to formulate this is to say, no matter how small a number we pick, say 0.001, we should always be able to conclude that the probability that our estimate differs from π by more than 0.001, goes to 0 as the number of samples goes to infinity. We chose 0.001 in this example, but this notion of probabilistic convergence should hold for any positive number, no matter how small. This leads us to the following definition.

Definition 0.0.4. *We say an estimator $\hat{p}$ is a **consistent** estimator of p if for any $\epsilon > 0$,*

$$\lim_{n\to\infty} P(|\hat{p} - p| > \epsilon) = 0.$$

Let's show that $4\hat{p}$ is a *consistent* estimator of π.

Proof. Choose any $\epsilon > 0$. By Chebyshev's inequality (Corollary 2.13),

$$
\begin{aligned}
P(|4\hat{p} - \pi| > \epsilon) &\leq \frac{\text{Var}(4\hat{p})}{\epsilon^2} \\
&= \frac{\text{Var}\left(4 \cdot \frac{1}{n}\sum_{i=1}^{n} X_i\right)}{\epsilon^2} && \text{(Definition of } \hat{p}) \\
&= \frac{\frac{16}{n^2}\text{Var}\left(\sum_{i=1}^{n} X_i\right)}{\epsilon^2} && (\text{Var}(cY) = c^2\text{Var}(Y)) \\
&= \frac{\frac{16}{n^2}\sum_{i=1}^{n}\text{Var}(X_i)}{\epsilon^2} && (X_i\text{'s are independent}) \\
&= \frac{\frac{16}{n^2} \cdot n \cdot \text{Var}(X_1)}{\epsilon^2} && (X_i\text{'s are identically distributed}) \\
&= \frac{\frac{16}{n} \cdot p(1-p)}{\epsilon^2} && (\text{Var}(X_i) = p(1-p)) \\
&= \frac{16 \cdot \frac{\pi}{4}\left(1 - \frac{\pi}{4}\right)}{n\epsilon^2} && \left(p = \frac{\pi}{4}\right) \\
&\to 0
\end{aligned}
$$

as $n \to \infty$. Hence we have shown that $4\hat{p}$ is a consistent estimator of π. □

Python 3 Cheat Sheet

Base Types

integer, float, boolean, string, bytes

```
int 783  0  -192  0b010  0o642  0xF3
                  zero  binary  octal  hexa
float 9.23  0.0  -1.7e-6
                      x10^-6
bool True  False
str "One\nTwo"        Multiline string:
    escaped new line  """X\tY\tZ
    'I\'m'            1\t2\t3"""
    escaped '         escaped tab
bytes b"toto\xfe\775"
                  hexadecimal  octal
```

☝ immutables

Container Types

- **ordered sequences**, fast index access, repeatable values

```
list [1,5,9]   ["x",11,8.9]   ["mot"]   []
tuple (1,5,9)  11,"y",7.4     ("mot",)  ()
```

Non modifiable values (immutables) — ☝ *expression with only comas →* `tuple`

```
str bytes   (ordered sequences of chars / bytes)   ""  b""
```

- **key containers**, no *a priori* order, fast key access, each key is unique

```
dictionary  dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
(key/value associations)  {1:"one",3:"three",2:"two",3.14:"n"}
collection  set {"key1","key2"}   {1,9,3,0}   set()
```

☝ *keys=hashable values (base types, immutables…)* `frozenset` *immutable set* *empty*

Identifiers

for variables, functions, modules, classes… names

`a…zA…Z_` followed by `a…zA…Z_0…9`

- diacritics allowed but should be avoided
- language keywords forbidden
- lower/UPPER case discrimination

☺ `a toto x7 y_max BigOne`
☹ ~~`3y and for`~~

Variables assignment

`=`

☝ assignment ⇔ **binding** of a *name* with a *value*

1) *evaluation of right side expression value*
2) *assignment in order with left side names*

```
x=1.2+8+sin(y)
a=b=c=0        assignment to same value
y,z,r=9.2,-7.6,0   multiple assignments
a,b=b,a        values swap
a,*b=seq       unpacking of sequence in
*a,b=seq       item and list
x+=3           increment ⇔ x=x+3      and *= /= %= …
x-=2           decrement ⇔ x=x-2
x=None         « undefined » constant value
del x          remove name x
```

Conversions

`type(`*expression*`)`

```
int("15") → 15
int("3f",16) → 63        can specify integer number base in 2nd parameter
int(15.56) → 15          truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6      rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x , None or False x ; True for other x
str(x)→ "…"              representation string of x for display (cf. formatting on the back)
chr(64)→'@'  ord('@')→64     code ↔ char
repr(x)→ "…"             literal representation string of x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str → assembled str
   ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str splitted on whitespaces → list of str
   "words with  spaces".split() → ['words','with','spaces']
str splitted on separator str → list of str
   "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via list comprehension)
   [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Sequence Containers Indexing

for lists, tuples, strings, bytes…

```
negative index   -5  -4  -3  -2  -1
positive index    0   1   2   3   4
lst=[10, 20, 30, 40, 50]
positive slice    0   1   2   3   4   5
negative slice   -5  -4  -3  -2  -1
```

Items count `len(lst)→5`

☝ **index from 0** (here from 0 to 4)

Individual access to **items** via `lst[`*index*`]`

```
lst[0]→10   ⇒ first one      lst[1]→20
lst[-1]→50  ⇒ last one       lst[-2]→40
```

On mutable sequences (`list`), remove with `del lst[3]` *and modify with assignment* `lst[4]=25`

Access to **sub-sequences** via `lst[`*start slice*`:`*end slice*`:`*step*`]`

```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]     lst[::-2]→[50,30,10]         lst[-3:-1]→[30,40]  lst[3:]→[40,50]
lst[::2]→[10,30,50]      lst[:]→[10,20,30,40,50]  shallow copy of sequence
```

Missing slice indication → from start / up to end.

On mutable sequences (`list`), remove with `del lst[3:5]` *and modify with assignment* `lst[1:4]=[15,25]`

Boolean Logic

Comparisons : `<` `>` `<=` `>=` `==` `!=` (*boolean results*) ≤ ≥ = ≠

`a and b` logical and — *both simultaneously*

`a or b` logical or — *one or other or both*

☝ pitfall : `and` *and* `or` *return value of* `a` *or of* `b` *(under shortcut evaluation).*
⇒ *ensure that* `a` *and* `b` *are booleans.*

`not a` logical not

`True` `False` — True and False constants

Statements Blocks

parent statement :
    statement block 1…
    ⋮
parent statement :
    statement block2…
    ⋮
next statement after block 1

indentation !

☝ *configure editor to insert 4 spaces in place of an indentation tab.*

Modules/Names Imports

module `truc` ⇔ *file* `truc.py`

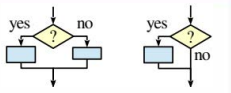
```
from monmod import nom1,nom2 as fct
                  →direct access to names, renaming with as
import monmod  →access via monmod.nom1 …
```

☝ modules and packages searched in *python path* (cf `sys.path`)

Conditional Statement

statement block executed only if a condition is true

```
if logical condition :
    statements block
```



Can go with several *elif*, *elif*… and only one final *else*. Only the block of first true condition is executed.

```
if age<=18:
    state="Kid"
elif age>65:
    state="Retired"
else:
    state="Active"
```

☝ *with a var* `x`:
`if bool(x)==True:` ⇔ `if x:`
`if bool(x)==False:` ⇔ `if not x:`

Maths

☝ *floating numbers… approximated values* *angles in radians*

Operators: `+` `-` `*` `/` `//` `%` `**`
Priority `(…)` × ÷ ↑ ↑ a^b — integer ÷ — ÷ remainder

`@` → matrix × *python3.5+numpy*

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

☝ *usual order of operations*

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

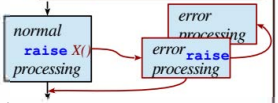
modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

Exceptions on Errors

Signaling an error:
`raise` *ExcClass(…)*

Errors processing:

```
try:
    normal procesising block
except Exception as e:
    error processing block
```



☝ `finally` *block for final processing in all cases.*

Conditional Loop Statement

*statements block executed **as long as** condition is true*

```
while logical condition :
    statements block
```


```
s = 0          } initializations before the loop
i = 1
while i <= 100:    } condition with a least one variable value (here i)
    s = s + i**2
    i = i + 1      # make condition variable change !
print("sum:",s)
```

Algo: $s = \sum_{i=1}^{i=100} i^2$

beware of infinite loops!

Loop Control

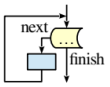
```
break      immediate exit
continue   next iteration
```

`else` *block for **normal** loop exit.*

Iterative Loop Statement

*statements block executed **for each** item of a container or iterator*

```
for var in sequence :
    statements block
```



Go over sequence's **values**

```
s = "Some text"   } initializations before the loop
cnt = 0
for c in s:       # loop variable, assignment managed by for statement
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

Algo: count number of `e` *in the string.*

loop on dict/set ⇔ loop on keys sequences
use *slices* to loop on a subset of a sequence

Go over sequence's **index**

- modify item at index
- access items around index (before / after)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

Algo: limit values greater than 15, memorizing of lost values.

Go simultaneously over sequence's **index** and **values**:

```
for idx,val in enumerate(lst):
```

good habit : don't modify loop variable

Display

```
print("v=",3,"cm :",x,",",y+4)
```

items to display : literal values, variables, expressions

`print` options:

- `sep=" "` items separator, default space
- `end="\n"` end of print, default new line
- `file=sys.stdout` print to file, default standard output

Input

```
s = input("Instructions:")
```

`input` always returns a **string**, convert it to required type (cf. boxed *Conversions* on the other side).

Generic Operations on Containers

`len(c)` → items count
`min(c)` `max(c)` `sum(c)`
`sorted(c)` → `list` sorted *copy*
`val in c` → boolean, membership operator `in` (absence `not in`)
`enumerate(c)` → *iterator* on (index, value)
`zip(c1,c2…)` → *iterator* on tuples containing c_i items at same index
`all(c)` → `True` if **all** `c` items evaluated to true, else `False`
`any(c)` → `True` if **at least one** item of `c` evaluated true, else `False`

*Note: For dictionaries and sets, these operations use **keys**.*

*Specific to **ordered sequences containers** (lists, tuples, strings, bytes…)*

`reversed(c)` → *inversed iterator* `c*5` → duplicate `c+c2` → concatenate
`c.index(val)` → *position* `c.count(val)` → *events count*

```
import copy
copy.copy(c) → shallow copy of container
copy.deepcopy(c) → deep copy of container
```

Operations on Lists

modify original list

`lst.append(val)`	add item at end
`lst.extend(seq)`	add sequence of items at end
`lst.insert(idx,val)`	insert item at index
`lst.remove(val)`	remove first item with value *val*
`lst.pop([idx])` →*value*	remove & return item at index *idx* (default last)
`lst.sort()` `lst.reverse()`	sort / reverse liste *in place*

Operations on Dictionaries

`d[key]=value` `d.clear()`
`d[key]` → *value* `del d[key]`
`d.update(d2)` update/add associations
`d.keys()` `d.values()` `d.items()` →*iterable views on keys/values/associations*
`d.pop(key[,default])` → *value*
`d.popitem()` → *(key,value)*
`d.get(key[,default])` → *value*
`d.setdefault(key[,default])` →*value*

Operations on Sets

Operators:
`|` → union (vertical bar char)
`&` → intersection
`-` `^` → difference/symmetric diff.
`<` `<=` `>` `>=` → inclusion relations
Operators also exist as methods.

`s.update(s2)` `s.copy()`
`s.add(key)` `s.remove(key)`
`s.discard(key)` `s.clear()`
`s.pop()`

Files

storing data on disk, and reading it back

```
f = open("file.txt","w",encoding="utf8")
```

- file **variable** for operations
- **name** of file on disk (+path…)
- opening **mode**
  - `'r'` read
  - `'w'` write
  - `'a'` append
  - …`'+'` `'x'` `'b'` `'t'`
- **encoding** of chars for *text files*: `utf8` `ascii` `latin1` …

cf. modules `os`, `os.path` and `pathlib`

writing

```
f.write("coucou")
f.writelines(list of lines)
```

reading

read empty string if end of file

```
f.read([n])      → next chars
                   if n not specified, read up to end !
f.readlines([n]) → list of next lines
f.readline()     → next line
```

*text mode `t` by default (read/write `str`), possible binary mode `b` (read/write `bytes`). **Convert from/to required type** !*

`f.close()` dont forget to **close the file** after use !

`f.flush()` write cache
`f.truncate([size])` resize

reading/writing progress sequentially in the file, modifiable with:
`f.tell()` →*position*
`f.seek(position[,origin])`

Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:

```
with open(…) as f:
    for line in f :
        # processing of line
```

Integer Sequences

```
range([start,] end [,step])
```

start default 0, *end* not included in sequence, *step* signed, default 1

`range(5)` → 0 1 2 3 4
`range(3,8)` → 3 4 5 6 7
`range(2,12,3)` → 2 5 8 11
`range(20,5,-5)` → 20 15 10
`range(len(seq))` → *sequence of index of values in seq*

range provides an immutable sequence of int constructed as needed

Function Definition

function name (identifier) — named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

result value of the call, if no computed result to return: `return None`

parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`

**args variable positional arguments (→`tuple`), default values,*
***kwargs variable named arguments (→`dict`)*

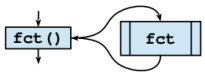
Function Call

```
r = fct(3,i+2,2*i)
```

storage/use of returned value — *one argument per parameter*

this is the use of function name *with parentheses* which does the call

*Advanced: *sequence **dict*



Operations on Strings

`s.startswith(prefix[,start[,end]])` `s.endswith(suffix[,start[,end]])` `s.strip([chars])`
`s.count(sub[,start[,end]])` `s.partition(sep)` → *(before,sep,after)*
`s.index(sub[,start[,end]])` `s.find(sub[,start[,end]])`
`s.is…()` *tests on chars categories (ex. `s.isalpha()`)*
`s.upper()` `s.lower()` `s.title()` `s.swapcase()`
`s.casefold()` `s.capitalize()` `s.center([width,fill])`
`s.ljust([width,fill])` `s.rjust([width,fill])` `s.zfill([width])`
`s.encode(encoding)` `s.split([sep])` `s.join(seq)`

Formatting

formating directives — values to format

```
"modele{} {} {}".format(x,y,r) → str
```

`"{`*selection* `:` *formatting* `!` *conversion*`}"`

- **Selection** :
  `2`
  `nom`
  `0.nom`
  `4[key]`
  `0[2]`
- **Formatting** :
  fill char *alignment* *sign* *mini width* `.` *precision~maxwidth* *type*
  `< > ^ =` `+ - space` `0` at start for filling with 0
  integer: `b` binary, `c` char, `d` decimal (default), `o` octal, `x` or `X` hexa…
  float: `e` or `E` exponential, `f` or `F` fixed point, `g` or `G` appropriate (default),
  string: `s` … `%` percent
- **Conversion** : `s` (readable text) or `r` (literal representation)

Examples:

```
"{:+2.3f}".format(45.72793)
→'+45.728'
"{1:>10s}".format(8,"toto")
→'      toto'
"{x!r}".format(x="I'm")
→'"I\'m"'
```

While loops

A while loop repeats a block of code as long as a certain condition is true.

A simple while loop

```
current_value = 1
while current_value <= 5:
    print(current_value)
    current_value += 1
```

Letting the user choose when to quit

```
msg = ''
while msg != 'quit':
    msg = input("What's your message? ")
    print(msg)
```

Functions

Functions are named blocks of code, designed to do one specific job. Information passed to a function is called an argument, and information received by a function is called a parameter.

A simple function

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
```

Passing an argument

```
def greet_user(username):
    """Display a personalized greeting."""
    print("Hello, " + username + "!")

greet_user('jesse')
```

Default values for parameters

```
def make_pizza(topping='bacon'):
    """Make a single-topping pizza."""
    print("Have a " + topping + " pizza!")

make_pizza()
make_pizza('pepperoni')
```

Returning a value

```
def add_numbers(x, y):
    """Add two numbers and return the sum."""
    return x + y

sum = add_numbers(3, 5)
print(sum)
```

Classes

A class defines the behavior of an object and the kind of information an object can store. The information in a class is stored in attributes, and functions that belong to a class are called methods. A child class inherits the attributes and methods from its parent class.

Creating a dog class

```
class Dog():
    """Represent a dog."""

    def __init__(self, name):
        """Initialize dog object."""
        self.name = name

    def sit(self):
        """Simulate sitting."""
        print(self.name + " is sitting.")

my_dog = Dog('Peso')

print(my_dog.name + " is a great dog!")
my_dog.sit()
```

Inheritance

```
class SARDog(Dog):
    """Represent a search dog."""

    def __init__(self, name):
        """Initialize the sardog."""
        super().__init__(name)

    def search(self):
        """Simulate searching."""
        print(self.name + " is searching.")

my_dog = SARDog('Willie')

print(my_dog.name + " is a search dog.")
my_dog.sit()
my_dog.search()
```

Infinite Skills

If you had infinite programming skills, what would you build?

As you're learning to program, it's helpful to think about the real-world projects you'd like to create. It's a good habit to keep an "ideas" notebook that you can refer to whenever you want to start a new project. If you haven't done so already, take a few minutes and describe three projects you'd like to create.

Working with files

Your programs can read from files and write to files. Files are opened in read mode ('r') by default, but can also be opened in write mode ('w') and append mode ('a').

Reading a file and storing its lines

```
filename = 'siddhartha.txt'
with open(filename) as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line)
```

Writing to a file

```
filename = 'journal.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
```

Appending to a file

```
filename = 'journal.txt'
with open(filename, 'a') as file_object:
    file_object.write("\nI love making games.")
```

Exceptions

Exceptions help you respond appropriately to errors that are likely to occur. You place code that might cause an error in the try block. Code that should run in response to an error goes in the except block. Code that should run only if the try block was successful goes in the else block.

Catching an exception

```
prompt = "How many tickets do you need? "
num_tickets = input(prompt)

try:
    num_tickets = int(num_tickets)
except ValueError:
    print("Please try again.")
else:
    print("Your tickets are printing.")
```

Zen of Python

Simple is better than complex

If you have a choice between a simple and a complex solution, and both work, use the simple solution. Your code will be easier to maintain, and it will be easier for you and others to build on that code later on.

Beginner's Python Cheat Sheet - Lists

What are lists?

A list stores a series of items in a particular order. Lists allow you to store sets of information in one place, whether you have just a few items or millions of items. Lists are one of Python's most powerful features readily accessible to new programmers, and they tie together many important concepts in programming.

Defining a list

Use square brackets to define a list, and use commas to separate individual items in the list. Use plural names for lists, to make your code easier to read.

Making a list

```
users = ['val', 'bob', 'mia', 'ron', 'ned']
```

Accessing elements

Individual elements in a list are accessed according to their position, called the index. The index of the first element is 0, the index of the second element is 1, and so forth. Negative indices refer to items at the end of the list. To get a particular element, write the name of the list and then the index of the element in square brackets.

Getting the first element

```
first_user = users[0]
```

Getting the second element

```
second_user = users[1]
```

Getting the last element

```
newest_user = users[-1]
```

Modifying individual items

Once you've defined a list, you can change individual elements in the list. You do this by referring to the index of the item you want to modify.

Changing an element

```
users[0] = 'valerie'
users[-2] = 'ronald'
```

Adding elements

You can add elements to the end of a list, or you can insert them wherever you like in a list.

Adding an element to the end of the list

```
users.append('amy')
```

Starting with an empty list

```
users = []
users.append('val')
users.append('bob')
users.append('mia')
```

Inserting elements at a particular position

```
users.insert(0, 'joe')
users.insert(3, 'bea')
```

Removing elements

You can remove elements by their position in a list, or by the value of the item. If you remove an item by its value, Python removes only the first item that has that value.

Deleting an element by its position

```
del users[-1]
```

Removing an item by its value

```
users.remove('mia')
```

Popping elements

If you want to work with an element that you're removing from the list, you can "pop" the element. If you think of the list as a stack of items, pop() takes an item off the top of the stack. By default pop() returns the last element in the list, but you can also pop elements from any position in the list.

Pop the last item from a list

```
most_recent_user = users.pop()
print(most_recent_user)
```

Pop the first item in a list

```
first_user = users.pop(0)
print(first_user)
```

List length

The len() function returns the number of items in a list.

Find the length of a list

```
num_users = len(users)
print("We have " + str(num_users) + " users.")
```

Sorting a list

The sort() method changes the order of a list permanently. The sorted() function returns a copy of the list, leaving the original list unchanged. You can sort the items in a list in alphabetical order, or reverse alphabetical order. You can also reverse the original order of the list. Keep in mind that lowercase and uppercase letters may affect the sort order.

Sorting a list permanently

```
users.sort()
```

Sorting a list permanently in reverse alphabetical order

```
users.sort(reverse=True)
```

Sorting a list temporarily

```
print(sorted(users))
print(sorted(users, reverse=True))
```

Reversing the order of a list

```
users.reverse()
```

Looping through a list

Lists can contain millions of items, so Python provides an efficient way to loop through all the items in a list. When you set up a loop, Python pulls each item from the list one at a time and stores it in a temporary variable, which you provide a name for. This name should be the singular version of the list name.

The indented block of code makes up the body of the loop, where you can work with each individual item. Any lines that are not indented run after the loop is completed.

Printing all items in a list

```
for user in users:
    print(user)
```

Printing a message for each item, and a separate message afterwards

```
for user in users:
    print("Welcome, " + user + "!")

print("Welcome, we're glad to see you all!")
```

The range() function

You can use the range() function to work with a set of numbers efficiently. The range() function starts at 0 by default, and stops one number below the number passed to it. You can use the list() function to efficiently generate a large list of numbers.

Printing the numbers 0 to 1000

```
for number in range(1001):
    print(number)
```

Printing the numbers 1 to 1000

```
for number in range(1, 1001):
    print(number)
```

Making a list of numbers from 1 to a million

```
numbers = list(range(1, 1000001))
```

Simple statistics

There are a number of simple statistics you can run on a list containing numerical data.

Finding the minimum value in a list

```
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
youngest = min(ages)
```

Finding the maximum value

```
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
oldest = max(ages)
```

Finding the sum of all values

```
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
total_years = sum(ages)
```

Slicing a list

You can work with any set of elements from a list. A portion of a list is called a slice. To slice a list start with the index of the first item you want, then add a colon and the index after the last item you want. Leave off the first index to start at the beginning of the list, and leave off the last index to slice through the end of the list.

Getting the first three items

```
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
first_three = finishers[:3]
```

Getting the middle three items

```
middle_three = finishers[1:4]
```

Getting the last three items

```
last_three = finishers[-3:]
```

Copying a list

To copy a list make a slice that starts at the first item and ends at the last item. If you try to copy a list without using this approach, whatever you do to the copied list will affect the original list as well.

Making a copy of a list

```
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
copy_of_finishers = finishers[:]
```

List comprehensions

You can use a loop to generate a list based on a range of numbers or on another list. This is a common operation, so Python offers a more efficient way to do it. List comprehensions may look complicated at first; if so, use the for loop approach until you're ready to start using comprehensions.

To write a comprehension, define an expression for the values you want to store in the list. Then write a for loop to generate input values needed to make the list.

Using a loop to generate a list of square numbers

```
squares = []
for x in range(1, 11):
    square = x**2
    squares.append(square)
```

Using a comprehension to generate a list of square numbers

```
squares = [x**2 for x in range(1, 11)]
```

Using a loop to convert a list of names to upper case

```
names = ['kai', 'abe', 'ada', 'gus', 'zoe']

upper_names = []
for name in names:
    upper_names.append(name.upper())
```

Using a comprehension to convert a list of names to upper case

```
names = ['kai', 'abe', 'ada', 'gus', 'zoe']

upper_names = [name.upper() for name in names]
```

Styling your code

Readability counts

- Use four spaces per indentation level.
- Keep your lines to 79 characters or fewer.
- Use single blank lines to group parts of your program visually.

Tuples

A tuple is like a list, except you can't change the values in a tuple once it's defined. Tuples are good for storing information that shouldn't be changed throughout the life of a program. Tuples are designated by parentheses instead of square brackets. (You can overwrite an entire tuple, but you can't change the individual elements in a tuple.)

Defining a tuple

```
dimensions = (800, 600)
```

Looping through a tuple

```
for dimension in dimensions:
    print(dimension)
```

Overwriting a tuple

```
dimensions = (800, 600)
print(dimensions)

dimensions = (1200, 900)
```

Visualizing your code

When you're first learning about data structures such as lists, it helps to visualize how Python is working with the information in your program. pythontutor.com is a great tool for seeing how Python keeps track of the information in a list. Try running the following code on pythontutor.com, and then run your own code.

Build a list and print the items in the list

```
dogs = []
dogs.append('willie')
dogs.append('hootz')
dogs.append('peso')
dogs.append('goblin')

for dog in dogs:
    print("Hello " + dog + "!")
print("I love these dogs!")

print("\nThese were my first two dogs:")
old_dogs = dogs[:2]
for old_dog in old_dogs:
    print(old_dog)

del dogs[0]
dogs.remove('peso')
print(dogs)
```

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet — Dictionaries

What are dictionaries?

Python's dictionaries allow you to connect pieces of related information. Each piece of information in a dictionary is stored as a key-value pair. When you provide a key, Python returns the value associated with that key. You can loop through all the key-value pairs, all the keys, or all the values.

Defining a dictionary

Use curly braces to define a dictionary. Use colons to connect keys and values, and use commas to separate individual key-value pairs.

Making a dictionary

```
alien_0 = {'color': 'green', 'points': 5}
```

Accessing values

To access the value associated with an individual key give the name of the dictionary and then place the key in a set of square brackets. If the key you're asking for is not in the dictionary, an error will occur.

You can also use the get() method, which returns None instead of an error if the key doesn't exist. You can also specify a default value to use if the key is not in the dictionary.

Getting the value associated with a key

```
alien_0 = {'color': 'green', 'points': 5}

print(alien_0['color'])
print(alien_0['points'])
```

Getting the value with get()

```
alien_0 = {'color': 'green'}

alien_color = alien_0.get('color')
alien_points = alien_0.get('points', 0)

print(alien_color)
print(alien_points)
```

Adding new key-value pairs

You can store as many key-value pairs as you want in a dictionary, until your computer runs out of memory. To add a new key-value pair to an existing dictionary give the name of the dictionary and the new key in square brackets, and set it equal to the new value.

This also allows you to start with an empty dictionary and add key-value pairs as they become relevant.

Adding a key-value pair

```
alien_0 = {'color': 'green', 'points': 5}

alien_0['x'] = 0
alien_0['y'] = 25
alien_0['speed'] = 1.5
```

Adding to an empty dictionary

```
alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
```

Modifying values

You can modify the value associated with any key in a dictionary. To do so give the name of the dictionary and enclose the key in square brackets, then provide the new value for that key.

Modifying values in a dictionary

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

# Change the alien's color and point value.
alien_0['color'] = 'yellow'
alien_0['points'] = 10
print(alien_0)
```

Removing key-value pairs

You can remove any key-value pair you want from a dictionary. To do so use the del keyword and the dictionary name, followed by the key in square brackets. This will delete the key and its associated value.

Deleting a key-value pair

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

del alien_0['points']
print(alien_0)
```

Visualizing dictionaries

Try running some of these examples on pythontutor.com.

Looping through a dictionary

You can loop through a dictionary in three ways: you can loop through all the key-value pairs, all the keys, or all the values.

A dictionary only tracks the connections between keys and values; it doesn't track the order of items in the dictionary. If you want to process the information in order, you can sort the keys in your loop.

Looping through all key-value pairs

```
# Store people's favorite languages.
fav_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
    }

# Show each person's favorite language.
for name, language in fav_languages.items():
    print(name + ": " + language)
```

Looping through all the keys

```
# Show everyone who's taken the survey.
for name in fav_languages.keys():
    print(name)
```

Looping through all the values

```
# Show all the languages that have been chosen.
for language in fav_languages.values():
    print(language)
```

Looping through all the keys in order

```
# Show each person's favorite language,
#   in order by the person's name.
for name in sorted(fav_languages.keys()):
    print(name + ": " + language)
```

Dictionary length

You can find the number of key-value pairs in a dictionary.

Finding a dictionary's length

```
num_responses = len(fav_languages)
```

Nesting — A list of dictionaries

It's sometimes useful to store a set of dictionaries in a list; this is called nesting.

Storing dictionaries in a list

```
# Start with an empty list.
users = []

# Make a new user, and add them to the list.
new_user = {
    'last': 'fermi',
    'first': 'enrico',
    'username': 'efermi',
    }
users.append(new_user)

# Make another new user, and add them as well.
new_user = {
    'last': 'curie',
    'first': 'marie',
    'username': 'mcurie',
    }
users.append(new_user)

# Show all information about each user.
for user_dict in users:
    for k, v in user_dict.items():
        print(k + ": " + v)
    print("\n")
```

You can also define a list of dictionaries directly, without using append():

```
# Define a list of users, where each user
#   is represented by a dictionary.
users = [
    {
        'last': 'fermi',
        'first': 'enrico',
        'username': 'efermi',
    },
    {
        'last': 'curie',
        'first': 'marie',
        'username': 'mcurie',
    },
]

# Show all information about each user.
for user_dict in users:
    for k, v in user_dict.items():
        print(k + ": " + v)
    print("\n")
```

Nesting — Lists in a dictionary

Storing a list inside a dictionary alows you to associate more than one value with each key.

Storing lists in a dictionary

```
# Store multiple languages for each person.
fav_languages = {
    'jen': ['python', 'ruby'],
    'sarah': ['c'],
    'edward': ['ruby', 'go'],
    'phil': ['python', 'haskell'],
}

# Show all responses for each person.
for name, langs in fav_languages.items():
    print(name + ": ")
    for lang in langs:
        print("- " + lang)
```

Nesting — A dictionary of dictionaries

You can store a dictionary inside another dictionary. In this case each value associated with a key is itself a dictionary.

Storing dictionaries in a dictionary

```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
        },
    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
        },
    }

for username, user_dict in users.items():
    print("\nUsername: " + username)
    full_name = user_dict['first'] + " "
    full_name += user_dict['last']
    location = user_dict['location']

    print("\tFull name: " + full_name.title())
    print("\tLocation: " + location.title())
```

Levels of nesting

Nesting is extremely useful in certain situations. However, be aware of making your code overly complex. If you're nesting items much deeper than what you see here there are probably simpler ways of managing your data, such as using classes.

Using an OrderedDict

Standard Python dictionaries don't keep track of the order in which keys and values are added; they only preserve the association between each key and its value. If you want to preserve the order in which keys and values are added, use an OrderedDict.

Preserving the order of keys and values

```
from collections import OrderedDict

# Store each person's languages, keeping
#  track of who respoded first.
fav_languages = OrderedDict()

fav_languages['jen'] = ['python', 'ruby']
fav_languages['sarah'] = ['c']
fav_languages['edward'] = ['ruby', 'go']
fav_languages['phil'] = ['python', 'haskell']

# Display the results, in the same order they
#  were entered.
for name, langs in fav_languages.items():
    print(name + ":")
    for lang in langs:
        print("- " + lang)
```

Generating a million dictionaries

You can use a loop to generate a large number of dictionaries efficiently, if all the dictionaries start out with similar data.

A million aliens

```
aliens = []

# Make a million green aliens, worth 5 points
#  each. Have them all start in one row.
for alien_num in range(1000000):
    new_alien = {}
    new_alien['color'] = 'green'
    new_alien['points'] = 5
    new_alien['x'] = 20 * alien_num
    new_alien['y'] = 0
    aliens.append(new_alien)

# Prove the list contains a million aliens.
num_aliens = len(aliens)

print("Number of aliens created:")
print(num_aliens)
```

Beginner's Python Cheat Sheet — If Statements and While Loops

What are if statements? What are while loops?

If statements allow you to examine the current state of a program and respond appropriately to that state. You can write a simple if statement that checks one condition, or you can create a complex series of if statements that idenitfy the exact conditions you're looking for.

While loops run as long as certain conditions remain true. You can use while loops to let your programs run as long as your users want them to.

Conditional Tests

A conditional test is an expression that can be evaluated as True or False. Python uses the values True and False to decide whether the code in an if statement should be executed.

Checking for equality

A single equal sign assigns a value to a variable. A double equal sign (==) checks whether two values are equal.

```
>>> car = 'bmw'
>>> car == 'bmw'
True
>>> car = 'audi'
>>> car == 'bmw'
False
```

Ignoring case when making a comparison

```
>>> car = 'Audi'
>>> car.lower() == 'audi'
True
```

Checking for inequality

```
>>> topping = 'mushrooms'
>>> topping != 'anchovies'
True
```

Numerical comparisons

Testing numerical values is similar to testing string values.

Testing equality and inequality

```
>>> age = 18
>>> age == 18
True
>>> age != 18
False
```

Comparison operators

```
>>> age = 19
>>> age < 21
True
>>> age <= 21
True
>>> age > 21
False
>>> age >= 21
False
```

Checking multiple conditions

You can check multiple conditions at the same time. The and operator returns True if all the conditions listed are True. The or operator returns True if any condition is True.

Using and to check multiple conditions

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 and age_1 >= 21
False
>>> age_1 = 23
>>> age_0 >= 21 and age_1 >= 21
True
```

Using or to check multiple conditions

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 or age_1 >= 21
True
>>> age_0 = 18
>>> age_0 >= 21 or age_1 >= 21
False
```

Boolean values

A boolean value is either True or False. Variables with boolean values are often used to keep track of certain conditions within a program.

Simple boolean values

```
game_active = True
can_edit = False
```

If statements

Several kinds of if statements exist. Your choice of which to use depends on the number of conditions you need to test. You can have as many elif blocks as you need, and the else block is always optional.

Simple if statement

```
age = 19

if age >= 18:
    print("You're old enough to vote!")
```

If-else statements

```
age = 17

if age >= 18:
    print("You're old enough to vote!")
else:
    print("You can't vote yet.")
```

The if-elif-else chain

```
age = 12

if age < 4:
   price = 0
elif age < 18:
    price = 5
else:
    price = 10

print("Your cost is $" + str(price) + ".")
```

Conditional tests with lists

You can easily test whether a certain value is in a list. You can also test whether a list is empty before trying to loop through the list.

Testing if a value is in a list

```
>>> players = ['al', 'bea', 'cyn', 'dale']
>>> 'al' in players
True
>>> 'eric' in players
False
```

Conditional tests with lists (cont.)

Testing if a value is not in a list

```
banned_users = ['ann', 'chad', 'dee']
user = 'erin'

if user not in banned_users:
    print("You can play!")
```

Checking if a list is empty

```
players = []

if players:
    for player in players:
        print("Player: " + player.title())
else:
    print("We have no players yet!")
```

Accepting input

You can allow your users to enter input using the input() statement. In Python 3, all input is stored as a string.

Simple input

```
name = input("What's your name? ")
print("Hello, " + name + ".")
```

Accepting numerical input

```
age = input("How old are you? ")
age = int(age)

if age >= 18:
    print("\nYou can vote!")
else:
    print("\nYou can't vote yet.")
```

Accepting input in Python 2.7

Use raw_input() in Python 2.7. This function interprets all input as a string, just as input() does in Python 3.

```
name = raw_input("What's your name? ")
print("Hello, " + name + ".")
```

While loops

A while loop repeats a block of code as long as a condition is True.

Counting to 5

```
current_number = 1

while current_number <= 5:
    print(current_number)
    current_number += 1
```

While loops (cont.)

Letting the user choose when to quit

```
prompt = "\nTell me something, and I'll "
prompt += "repeat it back to you."
prompt += "\nEnter 'quit' to end the program. "

message = ""
while message != 'quit':
    message = input(prompt)

    if message != 'quit':
        print(message)
```

Using a flag

```
prompt = "\nTell me something, and I'll "
prompt += "repeat it back to you."
prompt += "\nEnter 'quit' to end the program. "

active = True
while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

Using break to exit a loop

```
prompt = "\nWhat cities have you visited?"
prompt += "\nEnter 'quit' when you're done. "

while True:
    city = input(prompt)

    if city == 'quit':
        break
    else:
        print("I've been to " + city + "!")
```

Accepting input with Sublime Text

Sublime Text doesn't run programs that prompt the user for input. You can use Sublime Text to write programs that prompt for input, but you'll need to run these programs from a terminal.

Breaking out of loops

You can use the break statement and the continue statement with any of Python's loops. For example you can use break to quit a for loop that's working through a list or a dictionary. You can use continue to skip over certain items when looping through a list or dictionary as well.

While loops (cont.)

Using continue in a loop

```
banned_users = ['eve', 'fred', 'gary', 'helen']

prompt = "\nAdd a player to your team."
prompt += "\nEnter 'quit' when you're done. "

players = []
while True:
    player = input(prompt)
    if player == 'quit':
        break
    elif player in banned_users:
        print(player + " is banned!")
        continue
    else:
        players.append(player)

print("\nYour team:")
for player in players:
    print(player)
```

Avoiding infinite loops

Every while loop needs a way to stop running so it won't continue to run forever. If there's no way for the condition to become False, the loop will never stop running.

An infinite loop

```
while True:
    name = input("\nWho are you? ")
    print("Nice to meet you, " + name + "!")
```

Removing all instances of a value from a list

The remove() method removes a specific value from a list, but it only removes the first instance of the value you provide. You can use a while loop to remove all instances of a particular value.

Removing all cats from a list of pets

```
pets = ['dog', 'cat', 'dog', 'fish', 'cat',
        'rabbit', 'cat']
print(pets)

while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

Beginner's Python Cheat Sheet — Functions

What are functions?

Functions are named blocks of code designed to do one specific job. Functions allow you to write code once that can then be run whenever you need to accomplish the same task. Functions can take in the information they need, and return the information they generate. Using functions effectively makes your programs easier to write, read, test, and fix.

Defining a function

The first line of a function is its definition, marked by the keyword def. The name of the function is followed by a set of parentheses and a colon. A docstring, in triple quotes, describes what the function does. The body of a function is indented one level.

To call a function, give the name of the function followed by a set of parentheses.

Making a function

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
```

Passing information to a function

Information that's passed to a function is called an argument; information that's received by a function is called a parameter. Arguments are included in parentheses after the function's name, and parameters are listed in parentheses in the function's definition.

Passing a single argument

```
def greet_user(username):
    """Display a simple greeting."""
    print("Hello, " + username + "!")

greet_user('jesse')
greet_user('diana')
greet_user('brandon')
```

Positional and keyword arguments

The two main kinds of arguments are positional and keyword arguments. When you use positional arguments Python matches the first argument in the function call with the first parameter in the function definition, and so forth.

With keyword arguments, you specify which parameter each argument should be assigned to in the function call. When you use keyword arguments, the order of the arguments doesn't matter.

Using positional arguments

```
def describe_pet(animal, name):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")

describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
```

Using keyword arguments

```
def describe_pet(animal, name):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")

describe_pet(animal='hamster', name='harry')
describe_pet(name='willie', animal='dog')
```

Default values

You can provide a default value for a parameter. When function calls omit this argument the default value will be used. Parameters with default values must be listed after parameters without default values in the function's definition so positional arguments can still work correctly.

Using a default value

```
def describe_pet(name, animal='dog'):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")

describe_pet('harry', 'hamster')
describe_pet('willie')
```

Using None to make an argument optional

```
def describe_pet(animal, name=None):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    if name:
        print("Its name is " + name + ".")

describe_pet('hamster', 'harry')
describe_pet('snake')
```

Return values

A function can return a value or a set of values. When a function returns a value, the calling line must provide a variable in which to store the return value. A function stops running when it reaches a return statement.

Returning a single value

```
def get_full_name(first, last):
    """Return a neatly formatted full name."""
    full_name = first + ' ' + last
    return full_name.title()

musician = get_full_name('jimi', 'hendrix')
print(musician)
```

Returning a dictionary

```
def build_person(first, last):
    """Return a dictionary of information
    about a person.
    """
    person = {'first': first, 'last': last}
    return person

musician = build_person('jimi', 'hendrix')
print(musician)
```

Returning a dictionary with optional values

```
def build_person(first, last, age=None):
    """Return a dictionary of information
    about a person.
    """
    person = {'first': first, 'last': last}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', 27)
print(musician)

musician = build_person('janis', 'joplin')
print(musician)
```

Visualizing functions

Try running some of these examples on pythontutor.com.

Passing a list to a function

You can pass a list as an argument to a function, and the function can work with the values in the list. Any changes the function makes to the list will affect the original list. You can prevent a function from modifying a list by passing a copy of the list as an argument.

Passing a list as an argument

```
def greet_users(names):
    """Print a simple greeting to everyone."""
    for name in names:
        msg = "Hello, " + name + "!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

Allowing a function to modify a list

The following example sends a list of models to a function for printing. The original list is emptied, and the second list is filled.

```
def print_models(unprinted, printed):
    """3d print a set of models."""
    while unprinted:
        current_model = unprinted.pop()
        print("Printing " + current_model)
        printed.append(current_model)

# Store some unprinted designs,
#  and print each of them.
unprinted = ['phone case', 'pendant', 'ring']
printed = []
print_models(unprinted, printed)

print("\nUnprinted:", unprinted)
print("Printed:", printed)
```

Preventing a function from modifying a list

The following example is the same as the previous one, except the original list is unchanged after calling print_models().

```
def print_models(unprinted, printed):
    """3d print a set of models."""
    while unprinted:
        current_model = unprinted.pop()
        print("Printing " + current_model)
        printed.append(current_model)

# Store some unprinted designs,
#  and print each of them.
original = ['phone case', 'pendant', 'ring']
printed = []

print_models(original[:], printed)
print("\nOriginal:", original)
print("Printed:", printed)
```

Passing an arbitrary number of arguments

*Sometimes you won't know how many arguments a function will need to accept. Python allows you to collect an arbitrary number of arguments into one parameter using the * operator. A parameter that accepts an arbitrary number of arguments must come last in the function definition.*

*The ** operator allows a parameter to collect an arbitrary number of keyword arguments.*

Collecting an arbitrary number of arguments

```
def make_pizza(size, *toppings):
    """Make a pizza."""
    print("\nMaking a " + size + " pizza.")
    print("Toppings:")
    for topping in toppings:
        print("- " + topping)

# Make three pizzas with different toppings.
make_pizza('small', 'pepperoni')
make_pizza('large', 'bacon bits', 'pineapple')
make_pizza('medium', 'mushrooms', 'peppers',
        'onions', 'extra cheese')
```

Collecting an arbitrary number of keyword arguments

```
def build_profile(first, last, **user_info):
    """Build a user's profile dictionary."""
    # Build a dict with the required keys.
    profile = {'first': first, 'last': last}

    # Add any other keys and values.
    for key, value in user_info.items():
        profile[key] = value

    return profile

# Create two users with different kinds
#   of information.
user_0 = build_profile('albert', 'einstein',
        location='princeton')
user_1 = build_profile('marie', 'curie',
        location='paris', field='chemistry')

print(user_0)
print(user_1)
```

What's the best way to structure a function?

As you can see there are many ways to write and call a function. When you're starting out, aim for something that simply works. As you gain experience you'll develop an understanding of the more subtle advantages of different structures such as positional and keyword arguments, and the various approaches to importing functions. For now if your functions do what you need them to, you're doing well.

Modules

You can store your functions in a separate file called a module, and then import the functions you need into the file containing your main program. This allows for cleaner program files. (Make sure your module is stored in the same directory as your main program.)

Storing a function in a module

File: pizza.py

```
def make_pizza(size, *toppings):
    """Make a pizza."""
    print("\nMaking a " + size + " pizza.")
    print("Toppings:")
    for topping in toppings:
        print("- " + topping)
```

Importing an entire module

File: making_pizzas.py
Every function in the module is available in the program file.

```
import pizza

pizza.make_pizza('medium', 'pepperoni')
pizza.make_pizza('small', 'bacon', 'pineapple')
```

Importing a specific function

Only the imported functions are available in the program file.

```
from pizza import make_pizza

make_pizza('medium', 'pepperoni')
make_pizza('small', 'bacon', 'pineapple')
```

Giving a module an alias

```
import pizza as p

p.make_pizza('medium', 'pepperoni')
p.make_pizza('small', 'bacon', 'pineapple')
```

Giving a function an alias

```
from pizza import make_pizza as mp

mp('medium', 'pepperoni')
mp('small', 'bacon', 'pineapple')
```

Importing all functions from a module

Don't do this, but recognize it when you see it in others' code. It can result in naming conflicts, which can cause errors.

```
from pizza import *

make_pizza('medium', 'pepperoni')
make_pizza('small', 'bacon', 'pineapple')
```

Beginner's Python Cheat Sheet - Classes

What are classes?

Classes are the foundation of object-oriented programming. Classes represent real-world things you want to model in your programs: for example dogs, cars, and robots. You use a class to make objects, which are specific instances of dogs, cars, and robots. A class defines the general behavior that a whole category of objects can have, and the information that can be associated with those objects.

Classes can inherit from each other – you can write a class that extends the functionality of an existing class. This allows you to code efficiently for a wide variety of situations.

Creating and using a class

Consider how we might model a car. What information would we associate with a car, and what behavior would it have? The information is stored in variables called attributes, and the behavior is represented by functions. Functions that are part of a class are called methods.

The Car class

```
class Car():
    """A simple attempt to model a car."""

    def __init__(self, make, model, year):
        """Initialize car attributes."""
        self.make = make
        self.model = model
        self.year = year

        # Fuel capacity and level in gallons.
        self.fuel_capacity = 15
        self.fuel_level = 0

    def fill_tank(self):
        """Fill gas tank to capacity."""
        self.fuel_level = self.fuel_capacity
        print("Fuel tank is full.")

    def drive(self):
        """Simulate driving."""
        print("The car is moving.")
```

Creating and using a class (cont.)

Creating an object from a class

```
my_car = Car('audi', 'a4', 2016)
```

Accessing attribute values

```
print(my_car.make)
print(my_car.model)
print(my_car.year)
```

Calling methods

```
my_car.fill_tank()
my_car.drive()
```

Creating multiple objects

```
my_car = Car('audi', 'a4', 2016)
my_old_car = Car('subaru', 'outback', 2013)
my_truck = Car('toyota', 'tacoma', 2010)
```

Modifying attributes

You can modify an attribute's value directly, or you can write methods that manage updating values more carefully.

Modifying an attribute directly

```
my_new_car = Car('audi', 'a4', 2016)
my_new_car.fuel_level = 5
```

Writing a method to update an attribute's value

```
def update_fuel_level(self, new_level):
    """Update the fuel level."""
    if new_level <= self.fuel_capacity:
        self.fuel_level = new_level
    else:
        print("The tank can't hold that much!")
```

Writing a method to increment an attribute's value

```
def add_fuel(self, amount):
    """Add fuel to the tank."""
    if (self.fuel_level + amount
            <= self.fuel_capacity):
        self.fuel_level += amount
        print("Added fuel.")
    else:
        print("The tank won't hold that much.")
```

Naming conventions

In Python class names are written in CamelCase and object names are written in lowercase with underscores. Modules that contain classes should still be named in lowercase with underscores.

Class inheritance

If the class you're writing is a specialized version of another class, you can use inheritance. When one class inherits from another, it automatically takes on all the attributes and methods of the parent class. The child class is free to introduce new attributes and methods, and override attributes and methods of the parent class.

To inherit from another class include the name of the parent class in parentheses when defining the new class.

The __init__() method for a child class

```
class ElectricCar(Car):
    """A simple model of an electric car."""

    def __init__(self, make, model, year):
        """Initialize an electric car."""
        super().__init__(make, model, year)

        # Attributes specific to electric cars.
        # Battery capacity in kWh.
        self.battery_size = 70
        # Charge level in %.
        self.charge_level = 0
```

Adding new methods to the child class

```
class ElectricCar(Car):
    --snip--
    def charge(self):
        """Fully charge the vehicle."""
        self.charge_level = 100
        print("The vehicle is fully charged.")
```

Using child methods and parent methods

```
my_ecar = ElectricCar('tesla', 'model s', 2016)

my_ecar.charge()
my_ecar.drive()
```

Finding your workflow

There are many ways to model real world objects and situations in code, and sometimes that variety can feel overwhelming. Pick an approach and try it – if your first attempt doesn't work, try a different approach.

Class inheritance (cont.)

Overriding parent methods

```
class ElectricCar(Car):
    --snip--
    def fill_tank(self):
        """Display an error message."""
        print("This car has no fuel tank!")
```

Instances as attributes

A class can have objects as attributes. This allows classes to work together to model complex situations.

A Battery class

```
class Battery():
    """A battery for an electric car."""

    def __init__(self, size=70):
        """Initialize battery attributes."""
        # Capacity in kWh, charge level in %.
        self.size = size
        self.charge_level = 0

    def get_range(self):
        """Return the battery's range."""
        if self.size == 70:
            return 240
        elif self.size == 85:
            return 270
```

Using an instance as an attribute

```
class ElectricCar(Car):
    --snip--

    def __init__(self, make, model, year):
        """Initialize an electric car."""
        super().__init__(make, model, year)

        # Attribute specific to electric cars.
        self.battery = Battery()

    def charge(self):
        """Fully charge the vehicle."""
        self.battery.charge_level = 100
        print("The vehicle is fully charged.")
```

Using the instance

```
my_ecar = ElectricCar('tesla', 'model x', 2016)

my_ecar.charge()
print(my_ecar.battery.get_range())
my_ecar.drive()
```

Importing classes

Class files can get long as you add detailed information and functionality. To help keep your program files uncluttered, you can store your classes in modules and import the classes you need into your main program.

Storing classes in a file

car.py

```
"""Represent gas and electric cars."""

class Car():
    """A simple attempt to model a car."""
    --snip—

class Battery():
    """A battery for an electric car."""
    --snip--

class ElectricCar(Car):
    """A simple model of an electric car."""
    --snip--
```

Importing individual classes from a module

my_cars.py

```
from car import Car, ElectricCar

my_beetle = Car('volkswagen', 'beetle', 2016)
my_beetle.fill_tank()
my_beetle.drive()

my_tesla = ElectricCar('tesla', 'model s',
2016)
my_tesla.charge()
my_tesla.drive()
```

Importing an entire module

```
import car

my_beetle = car.Car(
        'volkswagen', 'beetle', 2016)
my_beetle.fill_tank()
my_beetle.drive()

my_tesla = car.ElectricCar(
        'tesla', 'model s', 2016)
my_tesla.charge()
my_tesla.drive()
```

Importing all classes from a module

(Don't do this, but recognize it when you see it.)

```
from car import *

my_beetle = Car('volkswagen', 'beetle', 2016)
```

Classes in Python 2.7

Classes should inherit from object

```
class ClassName(object):
```

The Car class in Python 2.7

```
class Car(object):
```

Child class __init__() method is different

```
class ChildClassName(ParentClass):
    def __init__(self):
        super(ClassName, self).__init__()
```

The ElectricCar class in Python 2.7

```
class ElectricCar(Car):
    def __init__(self, make, model, year):
        super(ElectricCar, self).__init__(
                make, model, year)
```

Storing objects in a list

A list can hold as many items as you want, so you can make a large number of objects from a class and store them in a list.

Here's an example showing how to make a fleet of rental cars, and make sure all the cars are ready to drive.

A fleet of rental cars

```
from car import Car, ElectricCar

# Make lists to hold a fleet of cars.
gas_fleet = []
electric_fleet = []

# Make 500 gas cars and 250 electric cars.
for _ in range(500):
    car = Car('ford', 'focus', 2016)
    gas_fleet.append(car)
for _ in range(250):
    ecar = ElectricCar('nissan', 'leaf', 2016)
    electric_fleet.append(ecar)

# Fill the gas cars, and charge electric cars.
for car in gas_fleet:
    car.fill_tank()
for ecar in electric_fleet:
    ecar.charge()

print("Gas cars:", len(gas_fleet))
print("Electric cars:", len(electric_fleet))
```

Beginner's Python Cheat Sheet — Files and Exceptions

What are files? What are exceptions?

Your programs can read information in from files, and they can write data to files. Reading from files allows you to work with a wide variety of information; writing to files allows users to pick up where they left off the next time they run your program. You can write text to files, and you can store Python structures such as lists in data files.

Exceptions are special objects that help your programs respond to errors in appropriate ways. For example if your program tries to open a file that doesn't exist, you can use exceptions to display an informative error message instead of having the program crash.

Reading from a file

To read from a file your program needs to open the file and then read the contents of the file. You can read the entire contents of the file at once, or read the file line by line. The with statement makes sure the file is closed properly when the program has finished accessing the file.

Reading an entire file at once

```
filename = 'siddhartha.txt'

with open(filename) as f_obj:
    contents = f_obj.read()

print(contents)
```

Reading line by line

Each line that's read from the file has a newline character at the end of the line, and the print function adds its own newline character. The rstrip() method gets rid of the the extra blank lines this would result in when printing to the terminal.

```
filename = 'siddhartha.txt'

with open(filename) as f_obj:
    for line in f_obj:
        print(line.rstrip())
```

Reading from a file (cont.)

Storing the lines in a list

```
filename = 'siddhartha.txt'

with open(filename) as f_obj:
    lines = f_obj.readlines()

for line in lines:
    print(line.rstrip())
```

Writing to a file

Passing the 'w' argument to open() tells Python you want to write to the file. Be careful; this will erase the contents of the file if it already exists. Passing the 'a' argument tells Python you want to append to the end of an existing file.

Writing to an empty file

```
filename = 'programming.txt'

with open(filename, 'w') as f:
    f.write("I love programming!")
```

Writing multiple lines to an empty file

```
filename = 'programming.txt'

with open(filename, 'w') as f:
    f.write("I love programming!\n")
    f.write("I love creating new games.\n")
```

Appending to a file

```
filename = 'programming.txt'

with open(filename, 'a') as f:
    f.write("I also love working with data.\n")
    f.write("I love making apps as well.\n")
```

File paths

When Python runs the open() function, it looks for the file in the same directory where the program that's being excuted is stored. You can open a file from a subfolder using a relative path. You can also use an absolute path to open any file on your system.

Opening a file from a subfolder

```
f_path = "text_files/alice.txt"

with open(f_path) as f_obj:
    lines = f_obj.readlines()

for line in lines:
    print(line.rstrip())
```

File paths (cont.)

Opening a file using an absolute path

```
f_path = "/home/ehmatthes/books/alice.txt"

with open(f_path) as f_obj:
    lines = f_obj.readlines()
```

Opening a file on Windows

Windows will sometimes interpret forward slashes incorrectly. If you run into this, use backslashes in your file paths.

```
f_path = "C:\Users\ehmatthes\books\alice.txt"

with open(f_path) as f_obj:
    lines = f_obj.readlines()
```

The try-except block

When you think an error may occur, you can write a try-except block to handle the exception that might be raised. The try block tells Python to try running some code, and the except block tells Python what to do if the code results in a particular kind of error.

Handling the ZeroDivisionError exception

```
try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Handling the FileNotFoundError exception

```
f_name = 'siddhartha.txt'

try:
    with open(f_name) as f_obj:
        lines = f_obj.readlines()
except FileNotFoundError:
    msg = "Can't find file {0}.".format(f_name)
    print(msg)
```

Knowing which exception to handle

It can be hard to know what kind of exception to handle when writing code. Try writing your code without a try block, and make it generate an error. The traceback will tell you what kind of exception your program needs to handle.

The else block

The try block should only contain code that may cause an error. Any code that depends on the try block running successfully should be placed in the else block.

Using an else block

```
print("Enter two numbers. I'll divide them.")

x = input("First number: ")
y = input("Second number: ")

try:
    result = int(x) / int(y)
except ZeroDivisionError:
    print("You can't divide by zero!")
else:
    print(result)
```

Preventing crashes from user input

Without the except block in the following example, the program would crash if the user tries to divide by zero. As written, it will handle the error gracefully and keep running.

```
"""A simple calculator for division only."""

print("Enter two numbers. I'll divide them.")
print("Enter 'q' to quit.")

while True:
    x = input("\nFirst number: ")
    if x == 'q':
        break
    y = input("Second number: ")
    if y == 'q':
        break

    try:
        result = int(x) / int(y)
    except ZeroDivisionError:
        print("You can't divide by zero!")
    else:
        print(result)
```

Deciding which errors to report

Well-written, properly tested code is not very prone to internal errors such as syntax or logical errors. But every time your program depends on something external such as user input or the existence of a file, there's a possibility of an exception being raised.

It's up to you how to communicate errors to your users. Sometimes users need to know if a file is missing; sometimes it's better to handle the error silently. A little experience will help you know how much to report.

Failing silently

Sometimes you want your program to just continue running when it encounters an error, without reporting the error to the user. Using the pass statement in an else block allows you to do this.

Using the pass statement in an else block

```
f_names = ['alice.txt', 'siddhartha.txt',
        'moby_dick.txt', 'little_women.txt']

for f_name in f_names:
    # Report the length of each file found.
    try:
        with open(f_name) as f_obj:
            lines = f_obj.readlines()
    except FileNotFoundError:
        # Just move on to the next file.
        pass
    else:
        num_lines = len(lines)
        msg = "{0} has {1} lines.".format(
            f_name, num_lines)
        print(msg)
```

Avoid bare except blocks

Exception-handling code should catch specific exceptions that you expect to happen during your program's execution. A bare except block will catch all exceptions, including keyboard interrupts and system exits you might need when forcing a program to close.

If you want to use a try block and you're not sure which exception to catch, use Exception. It will catch most exceptions, but still allow you to interrupt programs intentionally.

Don't use bare except blocks

```
try:
    # Do something
except:
    pass
```

Use Exception instead

```
try:
    # Do something
except Exception:
    pass
```

Printing the exception

```
try:
    # Do something
except Exception as e:
    print(e, type(e))
```

Storing data with json

The json module allows you to dump simple Python data structures into a file, and load the data from that file the next time the program runs. The JSON data format is not specific to Python, so you can share this kind of data with people who work in other languages as well.

Knowing how to manage exceptions is important when working with stored data. You'll usually want to make sure the data you're trying to load exists before working with it.

Using json.dump() to store data

```
"""Store some numbers."""

import json

numbers = [2, 3, 5, 7, 11, 13]

filename = 'numbers.json'
with open(filename, 'w') as f_obj:
    json.dump(numbers, f_obj)
```

Using json.load() to read data

```
"""Load some previously stored numbers."""

import json

filename = 'numbers.json'
with open(filename) as f_obj:
    numbers = json.load(f_obj)

print(numbers)
```

Making sure the stored data exists

```
import json

f_name = 'numbers.json'

try:
    with open(f_name) as f_obj:
        numbers = json.load(f_obj)
except FileNotFoundError:
    msg = "Can't find {0}.".format(f_name)
    print(msg)
else:
    print(numbers)
```

Practice with exceptions

Take a program you've already written that prompts for user input, and add some error-handling code to the program.

Beginner's Python Cheat Sheet — Testing Your Code

Why test your code?

When you write a function or a class, you can also write tests for that code. Testing proves that your code works as it's supposed to in the situations it's designed to handle, and also when people use your programs in unexpected ways. Writing tests gives you confidence that your code will work correctly as more people begin to use your programs. You can also add new features to your programs and know that you haven't broken existing behavior.

A unit test verifies that one specific aspect of your code works as it's supposed to. A test case is a collection of unit tests which verify your code's behavior in a wide variety of situations.

Testing a function: A passing test

Python's unittest module provides tools for testing your code. To try it out, we'll create a function that returns a full name. We'll use the function in a regular program, and then build a test case for the function.

A function to test

Save this as full_names.py

```
def get_full_name(first, last):
    """Return a full name."""
    full_name = "{0} {1}".format(first, last)
    return full_name.title()
```

Using the function

Save this as names.py

```
from full_names import get_full_name

janis = get_full_name('janis', 'joplin')
print(janis)

bob = get_full_name('bob', 'dylan')
print(bob)
```

Testing a function (cont.)

Building a testcase with one unit test

To build a test case, make a class that inherits from `unittest.TestCase` *and write methods that begin with* `test_`*. Save this as test_full_names.py*

```
import unittest
from full_names import get_full_name

class NamesTestCase(unittest.TestCase):
    """Tests for names.py."""

    def test_first_last(self):
        """Test names like Janis Joplin."""
        full_name = get_full_name('janis',
                'joplin')
        self.assertEqual(full_name,
                'Janis Joplin')

unittest.main()
```

Running the test

Python reports on each unit test in the test case. The dot reports a single passing test. Python informs us that it ran 1 test in less than 0.001 seconds, and the OK lets us know that all unit tests in the test case passed.

```
.
--------------------------------------
Ran 1 test in 0.000s

OK
```

Testing a function: A failing test

Failing tests are important; they tell you that a change in the code has affected existing behavior. When a test fails, you need to modify the code so the existing behavior still works.

Modifying the function

We'll modify `get_full_name()` *so it handles middle names, but we'll do it in a way that breaks existing behavior.*

```
def get_full_name(first, middle, last):
    """Return a full name."""
    full_name = "{0} {1} {2}".format(first,
                                     middle, last)
    return full_name.title()
```

Using the function

```
from full_names import get_full_name

john = get_full_name('john', 'lee', 'hooker')
print(john)

david = get_full_name('david', 'lee', 'roth')
print(david)
```

A failing test (cont.)

Running the test

When you change your code, it's important to run your existing tests. This will tell you whether the changes you made affected existing behavior.

```
E
======================================================
ERROR: test_first_last (__main__.NamesTestCase)
Test names like Janis Joplin.
------------------------------------------------------
Traceback (most recent call last):
  File "test_full_names.py", line 10,
    in test_first_last
    'joplin')
TypeError: get_full_name() missing 1 required
  positional argument: 'last'

------------------------------------------------------
Ran 1 test in 0.001s

FAILED (errors=1)
```

Fixing the code

When a test fails, the code needs to be modified until the test passes again. (Don't make the mistake of rewriting your tests to fit your new code.) Here we can make the middle name optional.

```
def get_full_name(first, last, middle=''):
    """Return a full name."""
    if middle:
        full_name = "{0} {1} {2}".format(first,
                                         middle, last)
    else:
        full_name = "{0} {1}".format(first,
                                     last)
    return full_name.title()
```

Running the test

Now the test should pass again, which means our original functionality is still intact.

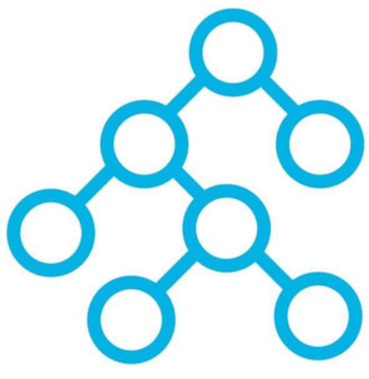
```
.
--------------------------------------
Ran 1 test in 0.000s

OK
```

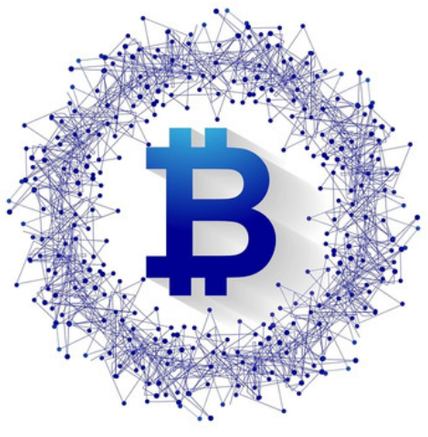
ONLY 10 STUDENTS PER BATCH ARE ALLOWED

WE OFFER EXCLUSIVE TRAINING IN

DIGITAL MARKETING



DATA SCIENCE



BLOCKCHAIN

PYTHON

MACHINE LEARNING

ADITI DIGITAL SOLUTIONS

+91 836 748 7105

THIRD FLOOR - VVR COMPLEX
BESIDE KPHB METRO STATION
KPHB ROAD NO 2
HYDERABAD - 500072
TELANGANA - INDIA