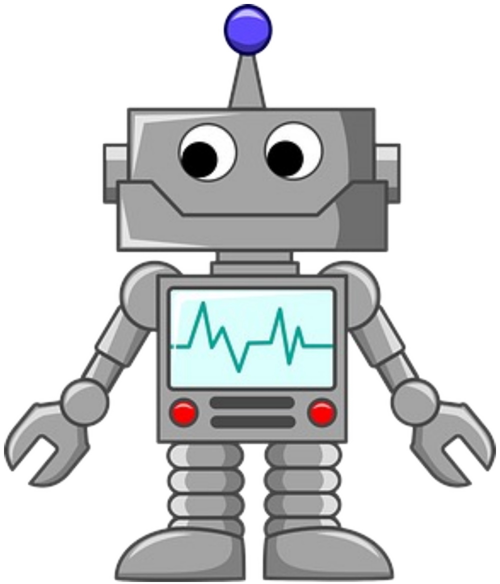
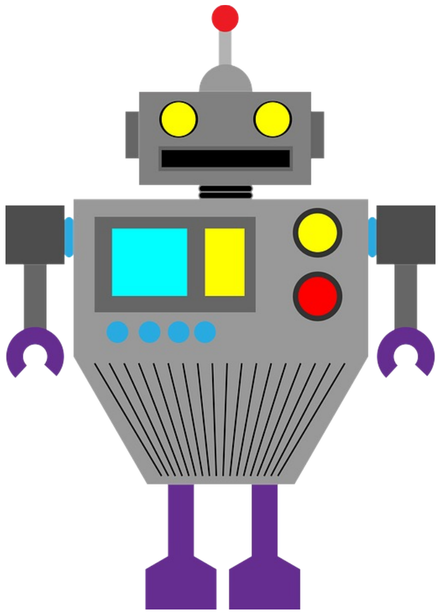
Demystifying Artificial Intelligence

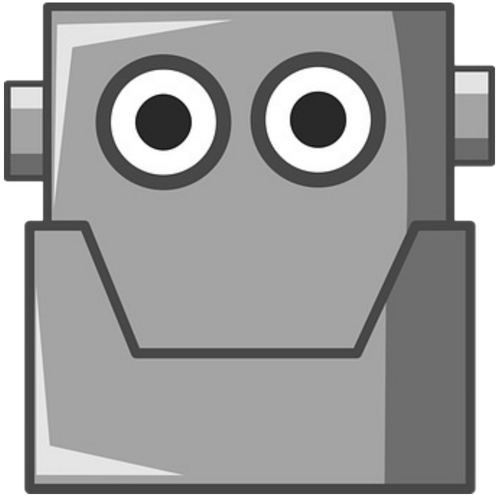


Instructor: Devin Bristow

What is Artificial Intelligence?



Definition



Artificial – “lacking in natural or spontaneous quality.” (Merriam-Webster)

Intelligence – “the ability to learn or understand or to deal with new or trying situations.” (Merriam-Webster)

Definition

Artificial Intelligence – "A branch of computer science dealing with the simulation of intelligent behavior in computers." (Merriam-Webster)

Definition

Artificial Intelligence –
Intelligence used by machinery.
(Devin Bristow)

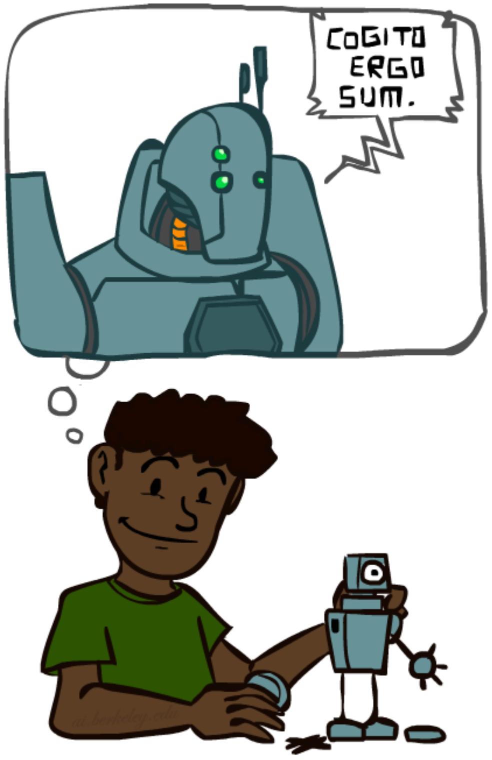
Video

Link:

https://www.youtube.com/watch?v=cIFqS2rMG3g&ab_channel=Freethink

A (Short) History of AI

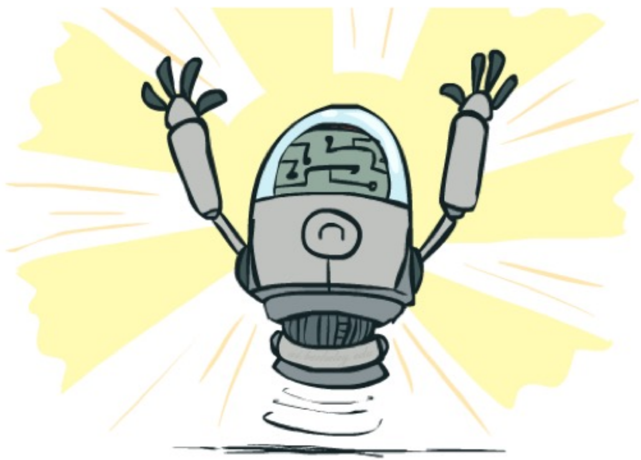
- 1940-1950: Early days
  - 1943: McCulloch & Pitts: Boolean circuit model of brain
  - 1950: Turing's “Computing Machinery and Intelligence”
- 1950—70: Excitement: Look, Ma, no hands!
  - 1950s: Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
  - 1956: Dartmouth meeting: “Artificial Intelligence” adopted
  - 1965: Robinson's complete algorithm for logical reasoning
- 1970—90: Knowledge-based approaches
  - 1969—79: Early development of knowledge-based systems
  - 1980—88: Expert systems industry booms
  - 1988—93: Expert systems industry busts: “AI Winter”
- 1990—: Statistical approaches
  - Resurgence of probability, focus on uncertainty
  - General increase in technical depth
  - Agents and learning systems… “AI Spring”?
- 2000—: Where are we now? Where are we headed?



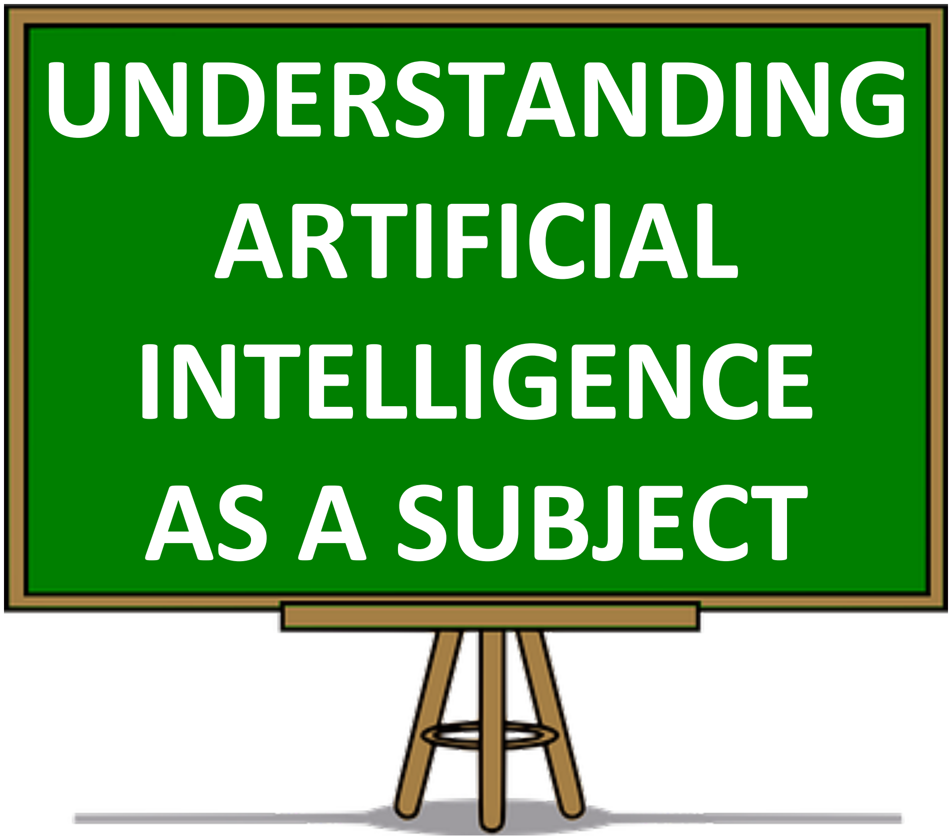
What Can AI Do?

Quiz: Which of the following can be done at present?

- ✓ Play a decent game of table tennis?
- ✓ Play a decent game of Jeopardy?
- ✓ Drive safely along a curving mountain road?
- ? Drive safely along Telegraph Avenue?
- ✓ Buy a week's worth of groceries on the web?
- ✗ Buy a week's worth of groceries at Berkeley Bowl?
- ? Discover and prove a new mathematical theorem?
- ✗ Converse successfully with another person for an hour?
- ? Perform a surgical operation?
- ✓ Put away the dishes and fold the laundry?
- ✓ Translate spoken Chinese into spoken English in real time?
- ✗ Write an intentionally funny story?



Subject

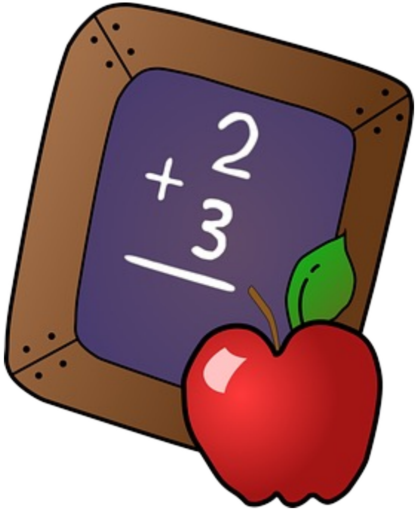


Course Overview

- Mathematics
  - Complexity Theory
  - Linear Algebra
  - Probability
  - Graph Theory
- Computer Science
  - Algorithms
  - Data Structures
  - Alan Turing
  - Logic
  - Prolog
  - Python
- Psychology
  - Behavior
  - Consciousness
  - Understanding
- Neuroscience
  - Parts of the Brain
- Artificial Intelligence
  - Artificial Neural Networks
  - Machine Learning
  - Deep Learning
  - Reinforcement Learning
  - Bayesian Networks
- Applications
  - Virtual Reality
  - Augmented Reality
  - Robotics
  - Gaming
  - Voice Recognition
  - Natural Language Processing

Course Overview

Mathematics



Complexity Theory

Big O, Big Ω, Big θ; Efficiency of algorithms

Linear Algebra

Vectors, Matrices

Probability

Probabilistic models, Distributions, Error functions, Expectations

Graph Theory

Nodes, Edges, Directed, Weighted, Cyclic

Course Overview

Computer Science

Algorithms

Greedy, Heuristics, Search

Data Structures

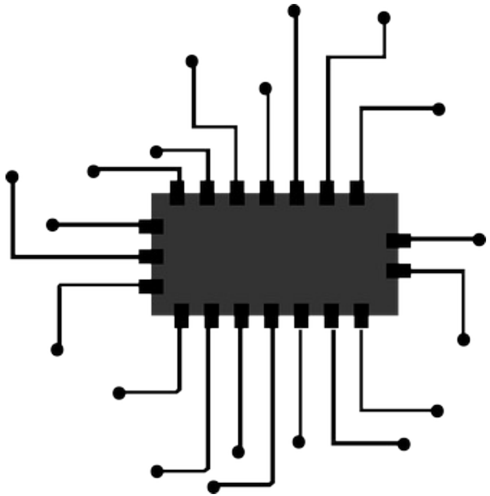
Arrays, Trees, Graphs

Alan Turing

Turing Machines, Turing Tests

Logic

Propositional Logic, Predicate Logic, Truth Tables



Course Overview

Computer Science

Prolog

Facts, Rules, Queries, Syntax, Structs, Arithmetic Operations, etc.

Python

Strings, Variables, Data Types, if-statements, for-loops, etc.

Course Overview

Psychology

Behaviors

Neural Communication

Consciousness

Dual-Track Mind

Understanding

Thinking, Learning, Intelligence, Language and Memory

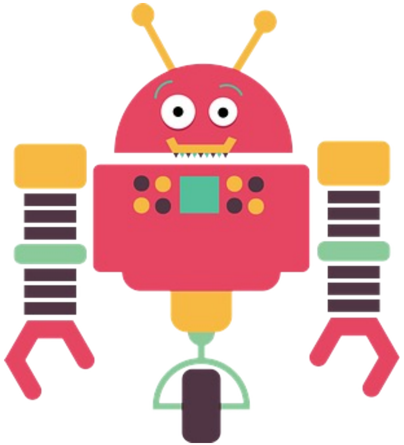
Course Overview

Neuroscience

<u>Parts of the Brain</u>

- Cerebral cortex
- Frontal lobe
- Occipital lobe
- Parietal lobe
- Temporal lobe

Course Overview



Artificial Intelligence

Artificial Neural Networks

Perceptrons

Machine Learning

Supervised Learning, Unsupervised Learning; Evaluating Regression

Deep Learning

Feedforward and Backward Propagations, Gradient Descent

Reinforcement Learning

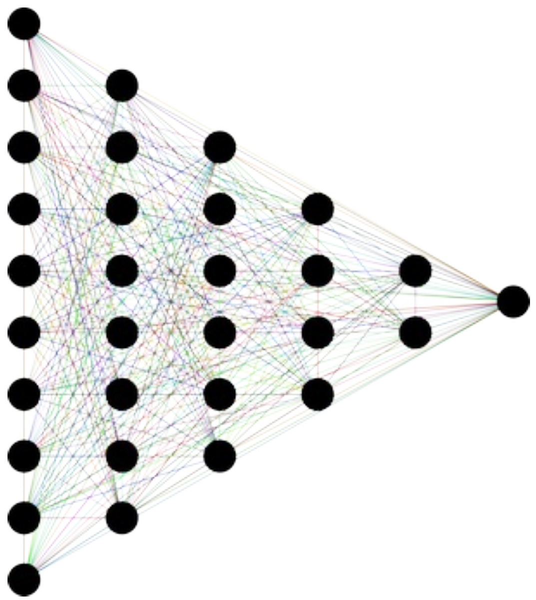
Agents, Environment, Action, Rewards (“A EAR”)

Course Overview

Artificial Intelligence

Bayesian Networks

Decisions; Accuracy



Applications

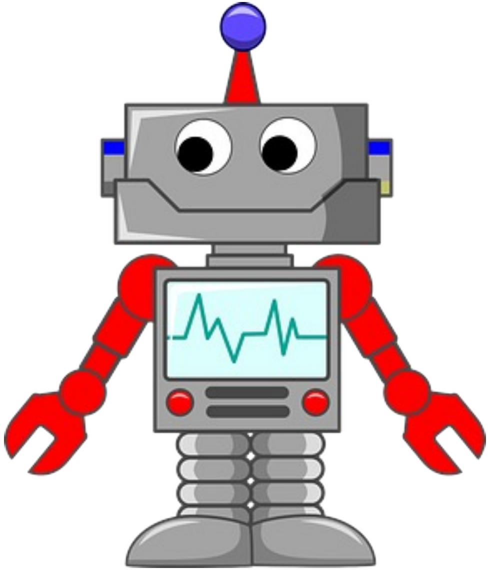
Virtual Reality

Interaction, Immersion, Imagination; Simulation

General Comments

- Augmented Reality
- Robotics
- Gaming
- Voice Recognition
- Natural Language Processing

Let’s dive right in!



What is Complexity Theory?

Complexity Theory

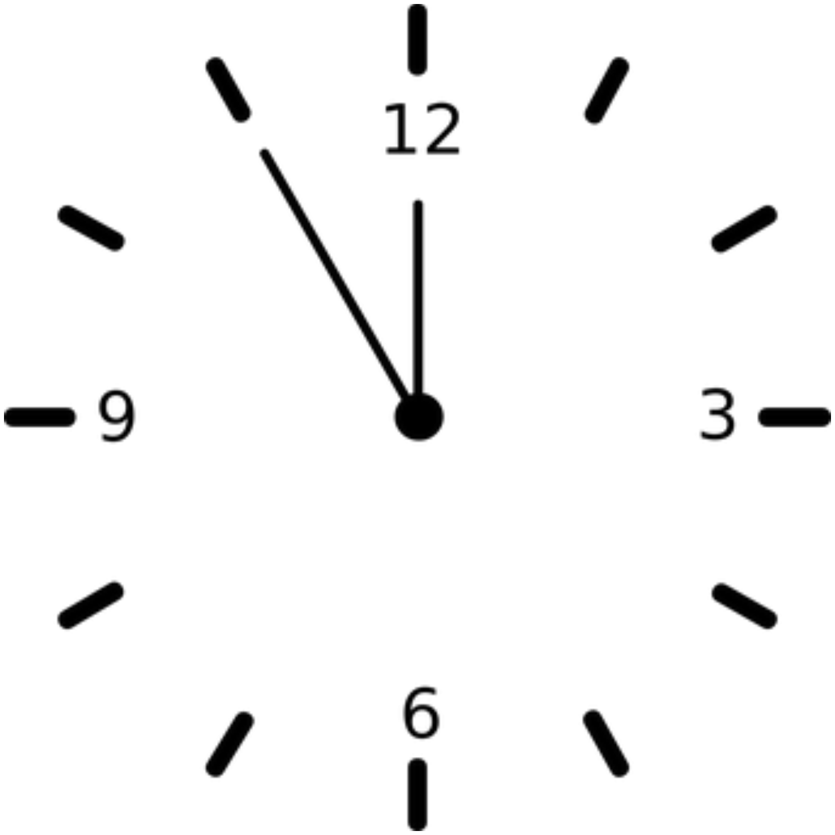
For computers:

- Some problems are easy.
- Some problems are hard.

Complexity Theory

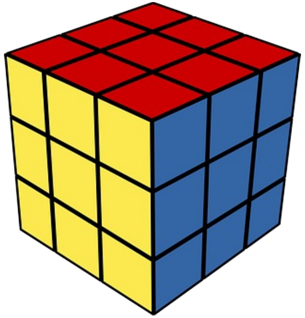
Time decides difficulty

Complexity Analysis



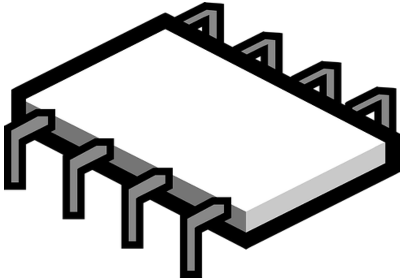
Polynomial-Time (P)

Those problems that can be solved in time O(n^k) for some constant integer k.



Non-deterministic Polynomial-Time (NP)

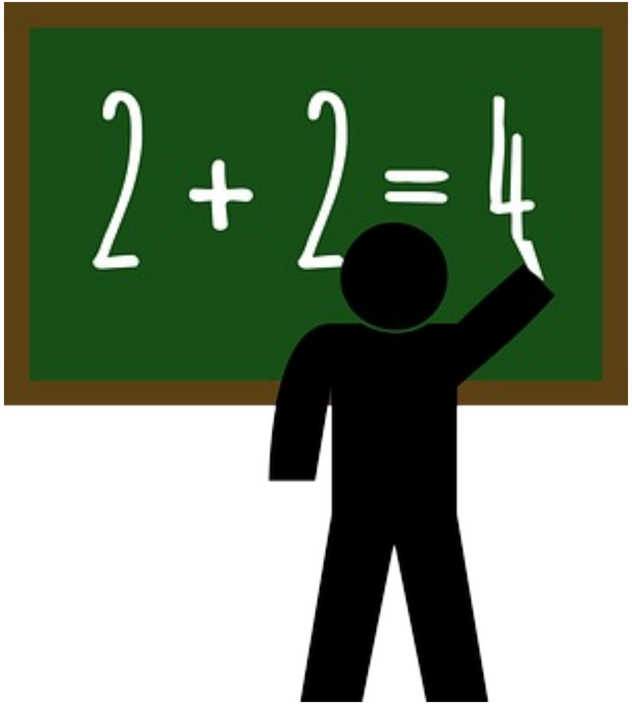
These problems are ones in which an algorithm can guess a solution and end up verifying whether the guess was correct in polynomial time.



Open-Question

Does P = NP?

Complexity Theory

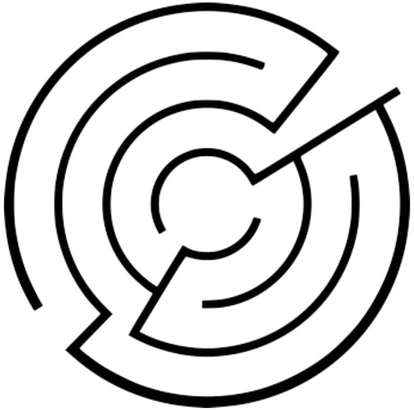


Problems are one thing;

Solving them is another.

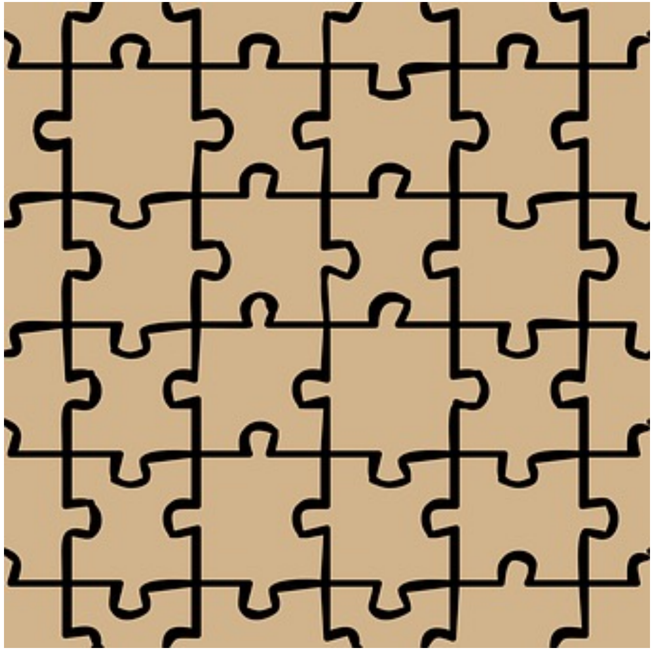
Algorithms

To solve problems, we'll need a sequence of steps to follow.



This is known as an algorithm.

Algorithms



There are many ways to solve the same problem.

Algorithms

The best way to solve a problem is said to be the most efficient way.

Time Complexity

We use a particular metric to determine an algorithm's efficiency.

Time Complexity

O

Time Complexity

Pronounced ‘Big Oh.’

Big O is the metric we use to determine an algorithm’s efficiency.

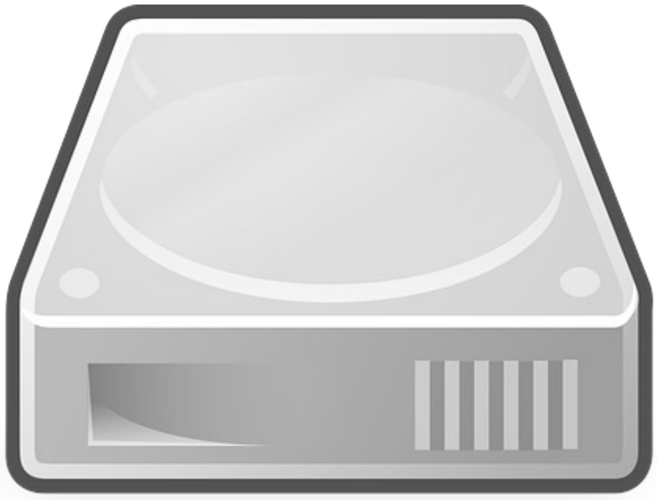
Time Complexity

Time Complexity is about the amount of time it takes for an algorithm to be completed.

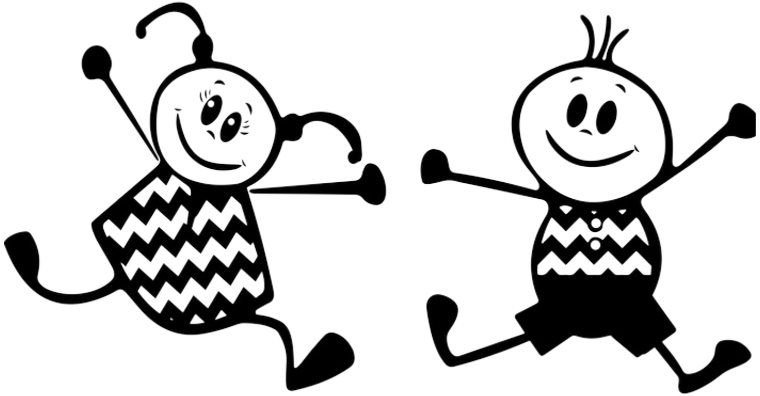
Imagine this scenario

Time Complexity

You have a file on a hard drive.



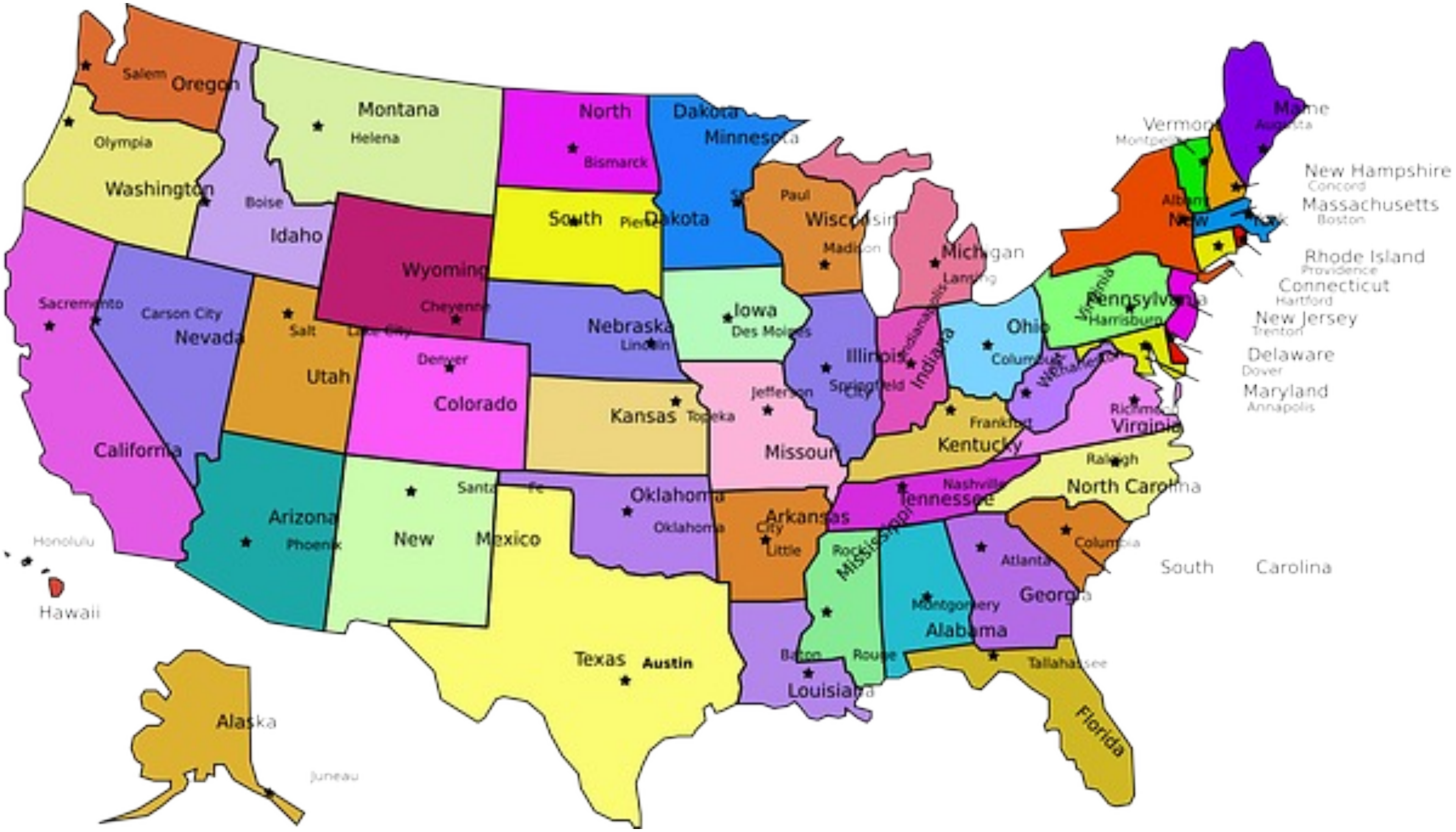
Time Complexity



You need to send your file to a friend ASAP.

Time Complexity

Your friend lives across the country.



Time Complexity

Here’s the question:

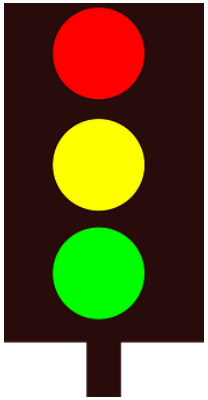
How should you send your friend the file?

Time Complexity

Popular Answers:

1. Email
2. File Transfer Protocol (FTP)
3. Other electronic means

Time Complexity



Are those answers right or wrong?

Time Complexity

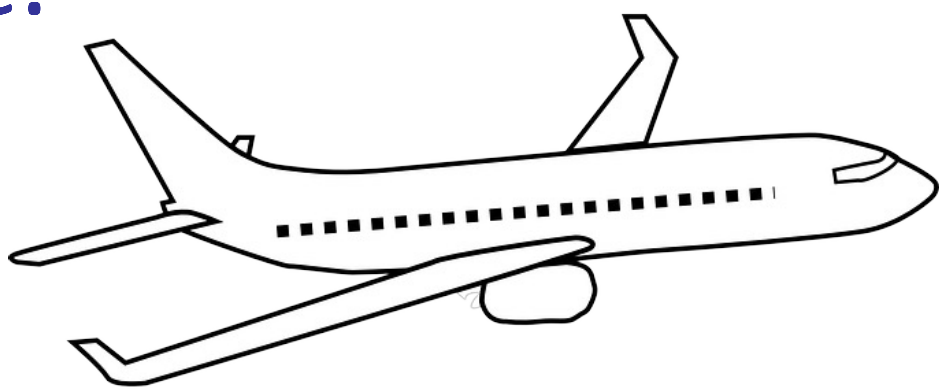
They are half right.

Time Complexity

If the file is small, then those answers are correct.

Time Complexity

It would take 5 hours to hop on a flight and send the file.



Time Complexity

BUT…

What if the file was really large?

Time Complexity

If the file was large, then it would be faster to send it by airplane!!!

Time Complexity

This is what the concept of Big O, or 'asymptotic runtime' is about.

Time Complexity

Data transfer "algorithm" runtime can be described as follows:

- O(s)
- O(1)

Time Complexity

O(s) is Electronic transfer.

The letter ‘s’ stands for the size of the file.

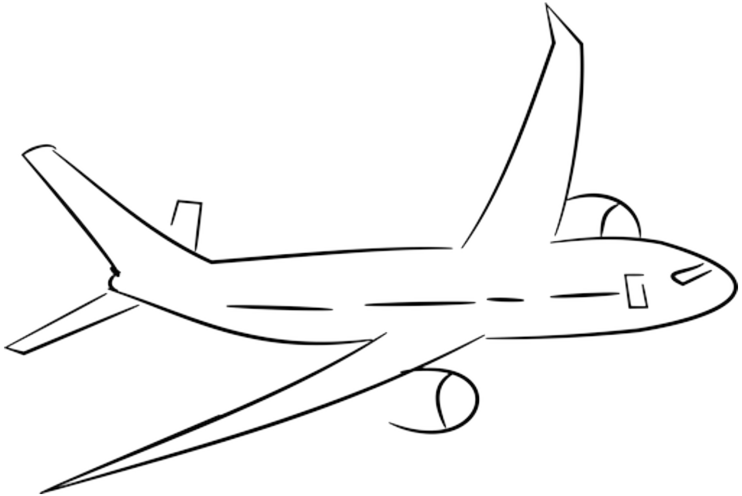
Time Complexity

O(s) means the time to transfer the file increases linearly as the size of the file increases.

Time Complexity

O(1) is Airplane transfer.

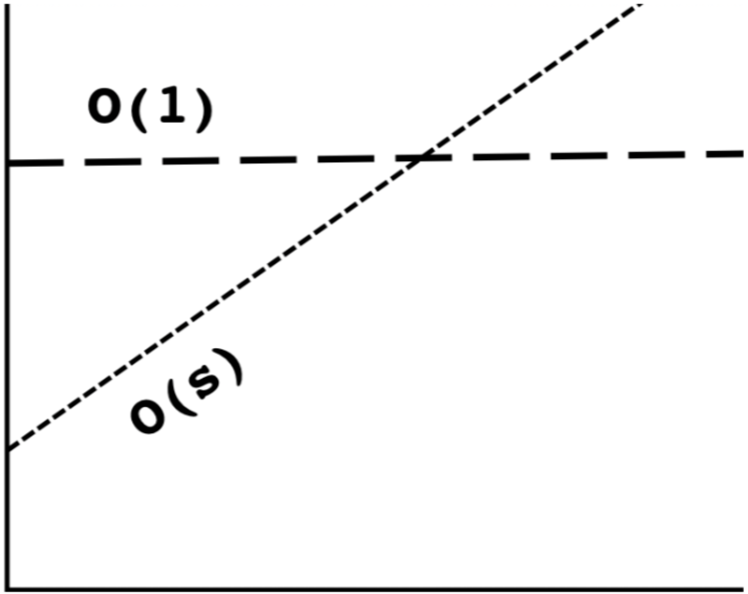
The number 1 is a fixed number.



Time Complexity

O(1) means as the size of the file increases, it won't take you any longer to get the file to your friend.

Time Complexity



[Figure 1: $O(s)$ vs. $O(1)$]

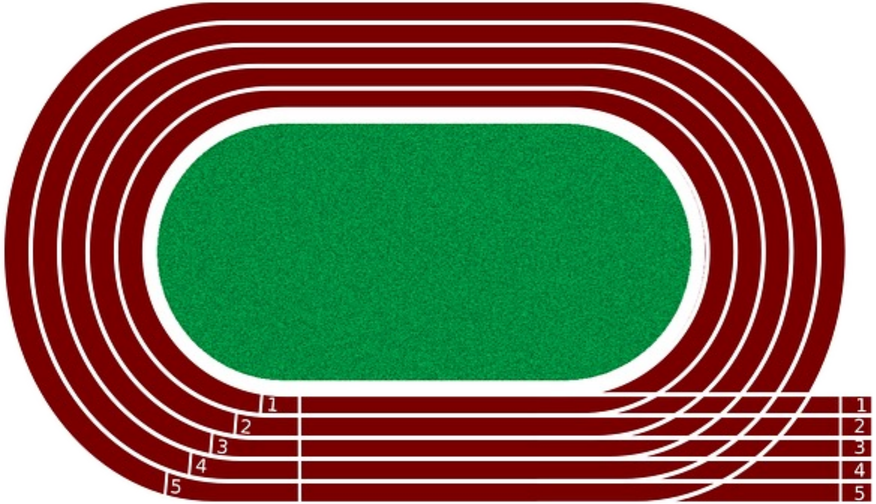
Time Complexity

There are plenty more runtimes than O(s) and O(1).

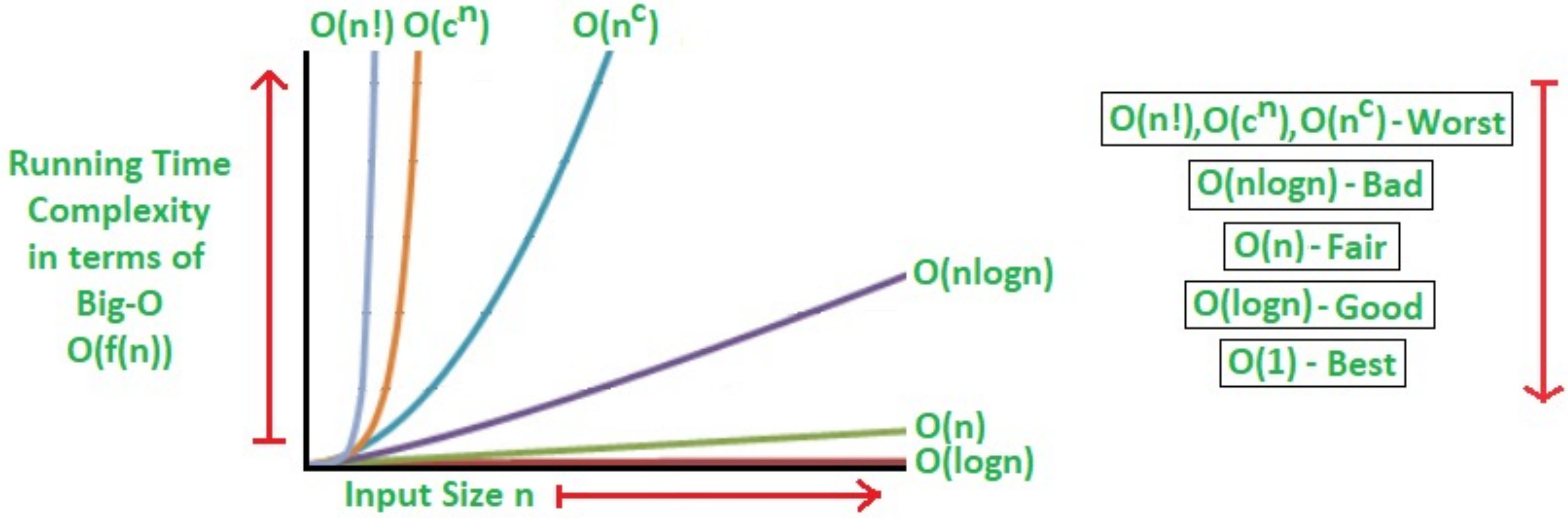
Time Complexity

Other examples of runtimes:

- $O(n)$
- $O(n^2)$
- $O(2^n)$
- $O(\log(n))$

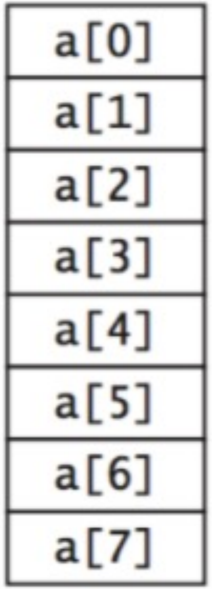


Runtime graphs



Big O example:

Array:



Time Complexity

Array – "a number of things considered as a unit." (Merriam-Webster)

Time Complexity

Array – “A collection of values.” (Devin Bristow)

Time Complexity

Big O example:

An algorithm that prints all the values in an array is done in O(n).

Time Complexity

We can think of Big O as an upper bound.

Time Complexity

You cannot walk higher than the tip of this mountain.

Time Complexity

There’s another metric.

Time Complexity

$$\Omega$$

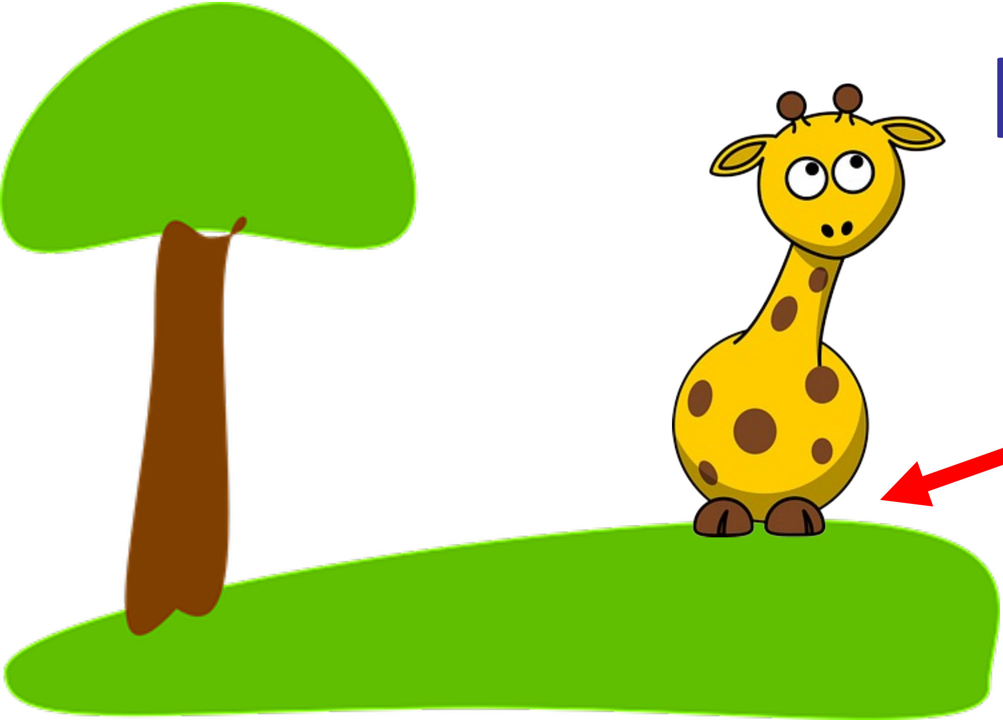
Time Complexity

This metric is called Big Ω (pronounced 'Big Omega').

We can think of Big Ω as a lower bound.

Time Complexity

You cannot go below the ground.



Time Complexity

One last metric.

Time Complexity

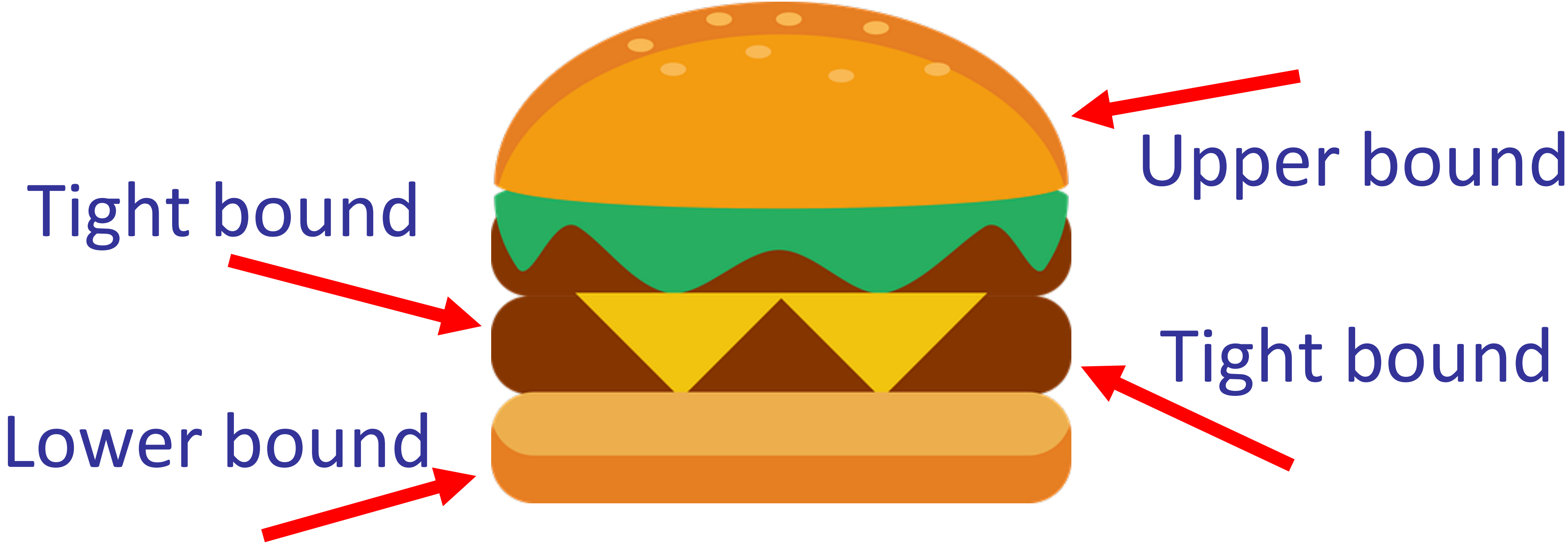
$$\Theta$$

Time Complexity

This metric is called Big θ (Pronounced 'Big Theta').

We can think of Big θ as a tight bound.

Time Complexity



The hamburger sits in between two buns.

Space Complexity

Some algorithms require machines with lots of space.

Space Complexity

The amount of memory or space needed to complete an algorithm is called Space Complexity.

Space Complexity

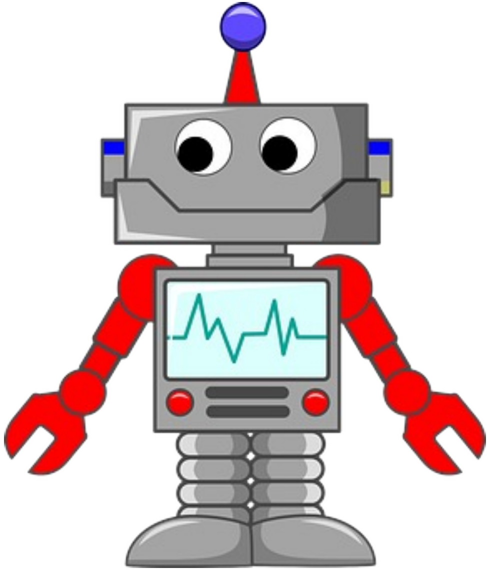
Example:

Suppose we created an array of size *n*.

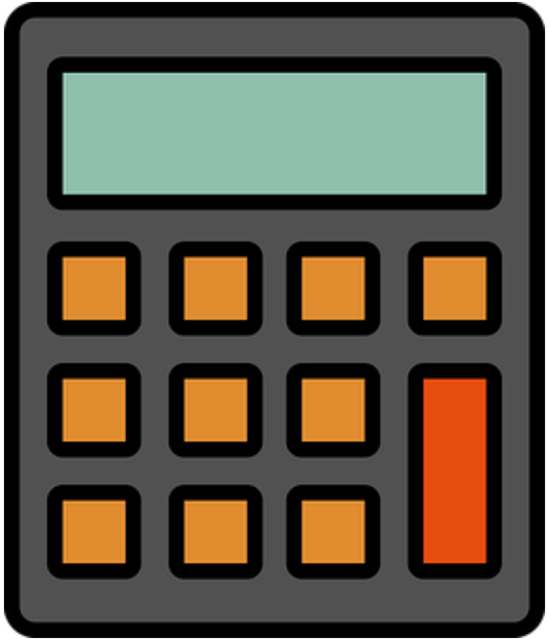
Space Complexity

Then that array would require $O(n)$ of space to run.

Linear Algebra



What is
Linear Algebra?



Definition

Linear – "… having a graph that is a line and especially a straight line : STRAIGHT." (Merriam-Webster)

Algebra – "a generalization of arithmetic in which letters representing numbers are combined…" (Merriam-Webster)

Linear Algebra

Linear Algebra is about arithmetic that produces straight lines.

Linear Algebra

When we say arithmetic, we mean:

- Adding
- Subtracting
- Multiplying
- Dividing

Linear Algebra

Linear Algebra is the study of linear equations.

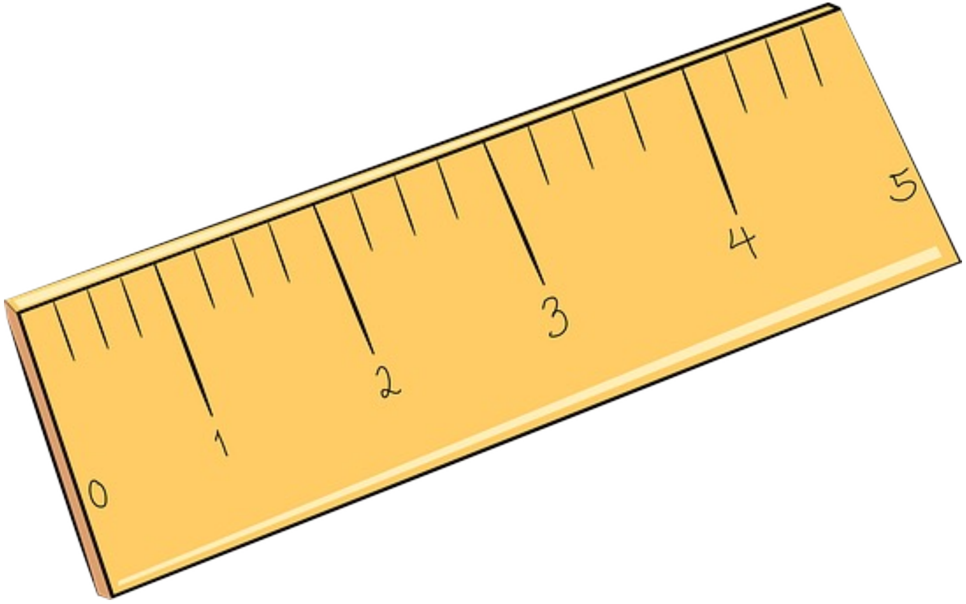
Linear Algebra

An equation is any statement with an equals (=) sign.

Linear equations produce straight lines.

Examples of Linear Equations:

- y=x
- y=2x
- y=x+3
- y=1



Let’s see how to produce straight lines from linear equations.

To do so, we’ll use desmos.com

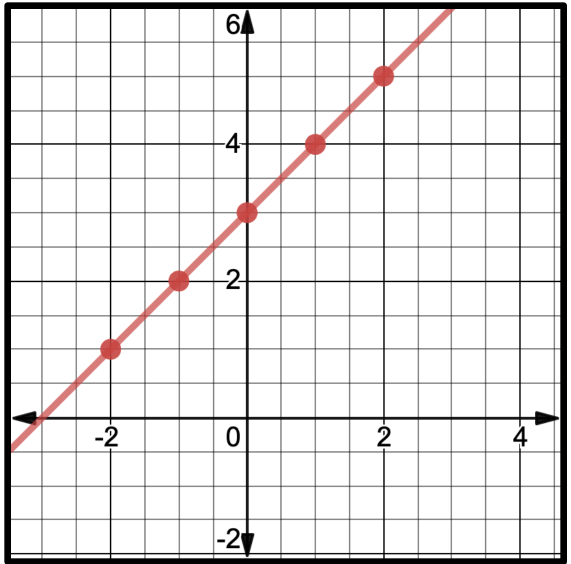
Exercise

Graph the equation $y = x+3$

Make a table:

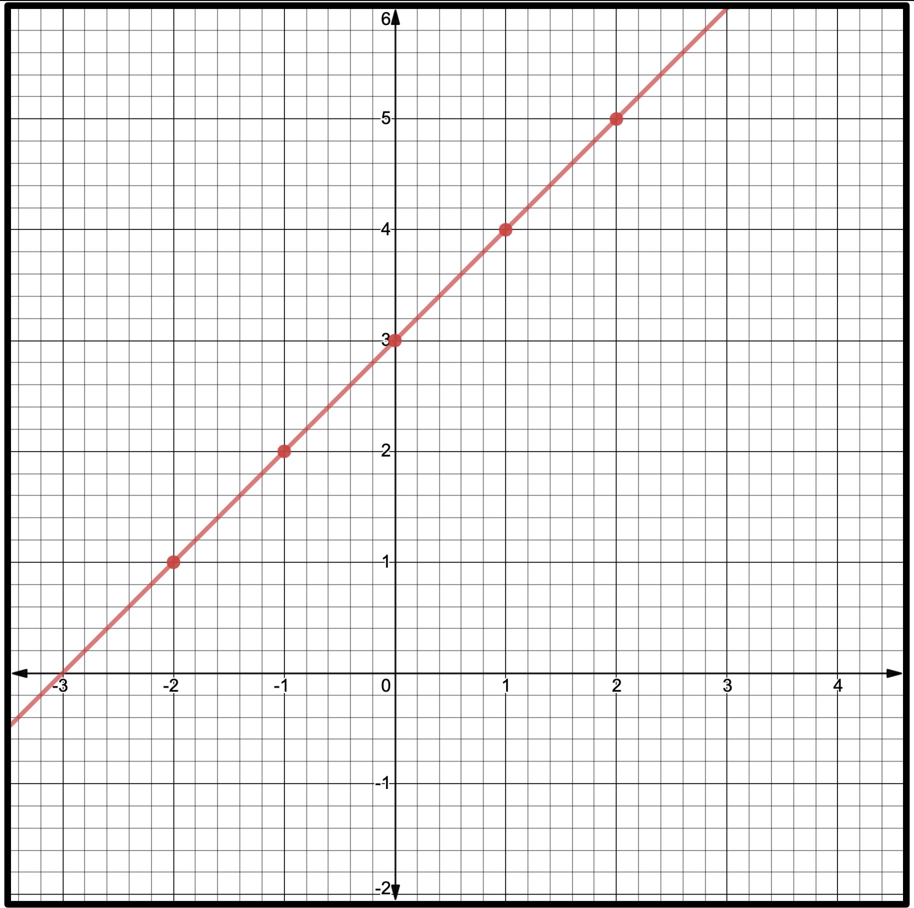
Input (x)	**Output (y = x + 3)**
x = -2	y = -2 + 3 = 1
x = -1	y = -1 + 3 = 2
x = 0	y = 0 + 3 = 3
x = 1	y = 1 + 3 = 4
x = 2	y = 2 + 3 = 5

Use values on table to plot the graph of $y=x+3$.



Linear Algebra

Same graph, just zoomed out more



General form of a Linear Equation:

$$y = ax + b$$

Linear Algebra

‘a’ and ‘b’ are numbers.

‘x’ and ‘y’ are variables.

Linear Algebra

In the equation $y = x + 3$,

$a = 1$

$b = 3$

Linear Algebra

In the equation $y = 2x + 5$,

$a = 2$

$b = 5$

Another general form of a linear equation:

$$ax + by = c$$

Linear Algebra

‘a’, ‘b’, and ‘c’ are numbers.

‘x’ and ‘y’ are variables.

Linear Algebra

In the equation $-4x + 5y = 10$,

$a = -4$

$b = 5$

$c = 10$

$-4x + 5y = 10$ is another equation that produces a straight line.

To see this, we must 'solve for y.'

$$-4x + 5y = 10$$

(Write down original equation)

$$5y = 4x + 10$$

(Add '4x' to both sides of of the equation).

$$y = \frac{4x + 10}{5}$$

(Divide both sides by 5)

Linear Algebra

When we say, "solve for y,”

This means get ‘y’ to be alone.

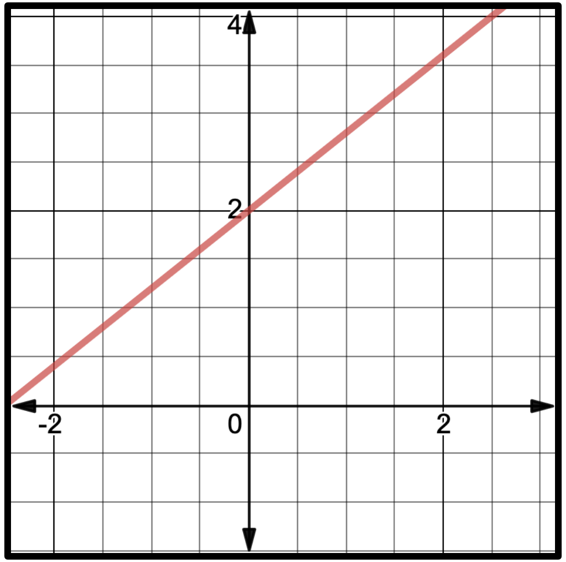
Exercise

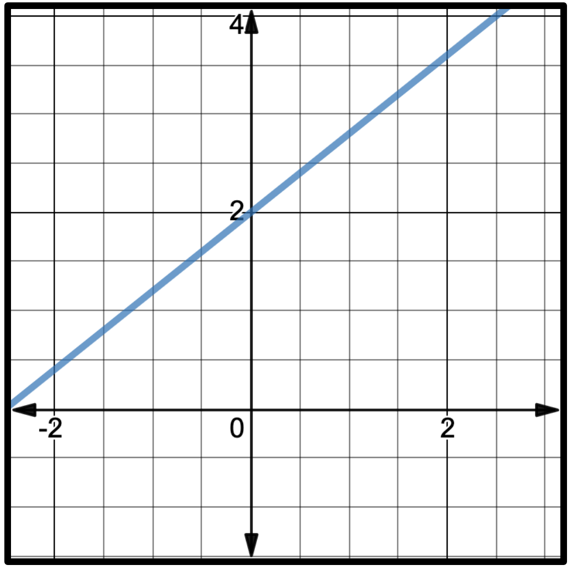
Graph the equations:

- $-4x + 5y = 10$
- $y = \frac{4x+10}{5}$

Linear Algebra

Use values on table to plot the graphs.





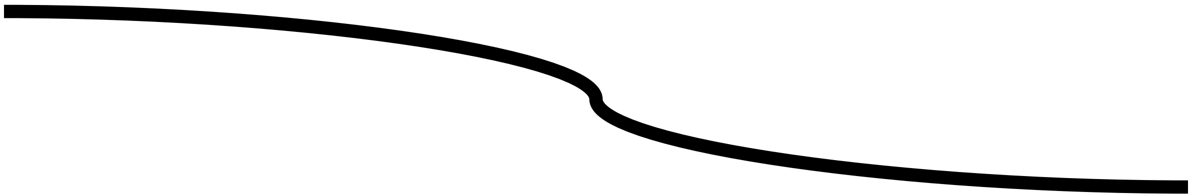
- $-4x + 5y = 10$
- $y = \frac{4x+10}{5}$

Linear Algebra

The moral of the story is:

We want straight lines.

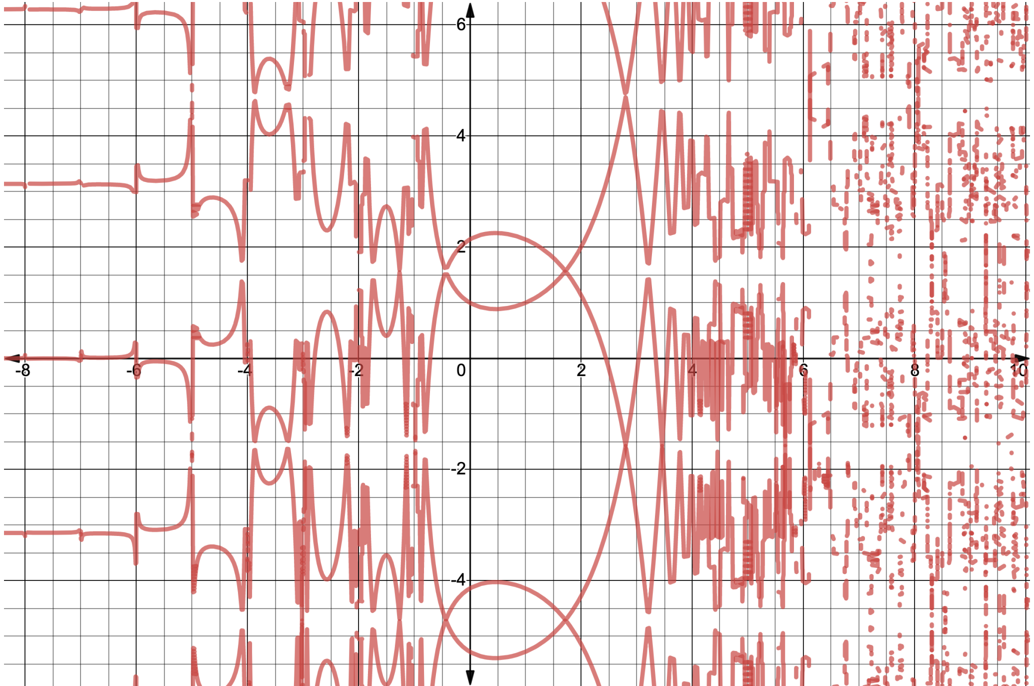
If a graph is not straight, we say it's non-linear.



Linear Algebra

Equations with straight lines are nicer to play with.

Exercise: try solving for y



LOL!

Matrices

Linear equations can be represented using a matrix.

Matrices

A matrix is a rectangular array of numbers.

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}$$

Matrices

A matrix consists of rows and columns.

Matrices

Matrix **A** has 3 rows and 3 columns. It's a 3×3 matrix.

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}$$

Matrices

Rows are horizontal —

Columns are vertical |

Matrices

Like numbers, we can add matrices:

$$A + B = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} + \begin{pmatrix} 3 & 4 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 4 & 7 \\ 9 & 12 \end{pmatrix}$$

Matrices

Like numbers, we can multiply matrices by whole numbers:

$$2A = 2 \cdot \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 6 \\ 4 & 8 \end{pmatrix}$$

Matrices

We can multiply matrices together:

$$AB = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} 3 & 4 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} (1)(3) + (3)(7) & (1)(4) + (3)(8) \\ (2)(3) + (4)(7) & (2)(4) + (4)(8) \end{pmatrix}$$

$$= \begin{pmatrix} 24 & 28 \\ 34 & 40 \end{pmatrix}$$

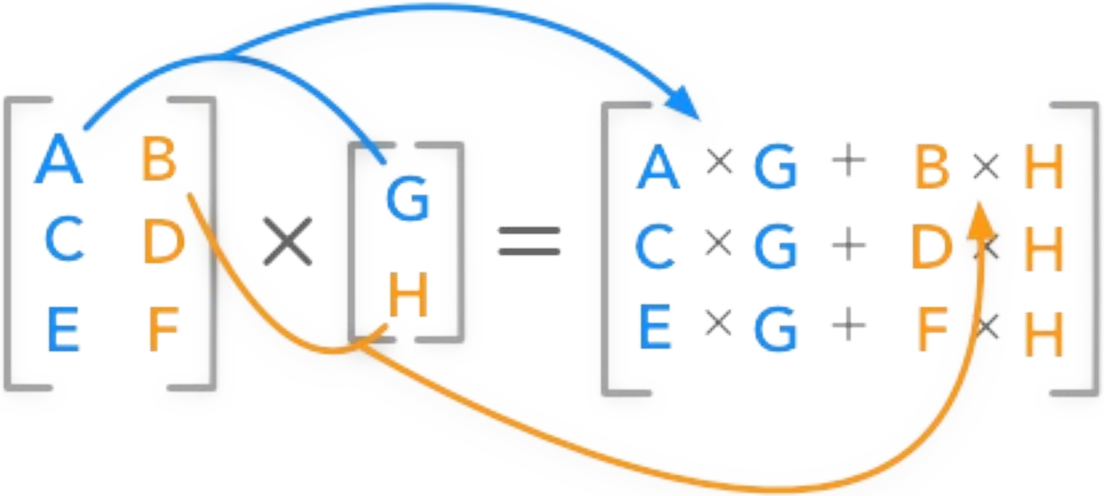
Matrices

General example:

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}$$

Matrices

Another example:



Matrices

There's something called the determinant of a matrix.

$$det\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} = (1)(4) - (2)(3)$$

$$= 4 - 6$$

$$= -2$$

Matrices

Finding the determinant of a matrix is similar to finding the volume of an object.

We can take the inverse of a matrix.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^{-1} = \frac{1}{(1)(4)-(2)(3)} \cdot \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix}$$

$$= \frac{1}{4-6} \cdot \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix}$$

$$= \frac{1}{-2} \cdot \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix} = \begin{pmatrix} -\frac{4}{2} & -\frac{2}{-2} \\ -\frac{3}{-2} & -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

Matrices

As you can see, matrices can achieve a lot!

They can help us achieve another thing: organization.

Matrices

Matrices help us organize linear (system of) equations!

$$\begin{aligned} 2x + 3y &= 1 \\ 8x - 2y &= 5 \\ 0x + 7y &= 6 \end{aligned} \quad \Longleftrightarrow \quad \begin{pmatrix} 2 & 3 & 1 \\ 8 & -2 & 5 \\ 0 & 7 & 6 \end{pmatrix}$$

Vectors

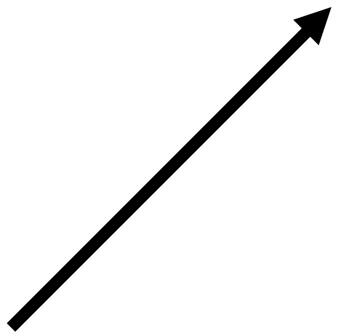
Let's switch gears.

Vectors

In linear algebra, there are objects called vectors.

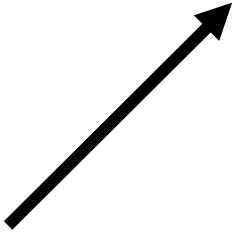
Vectors

What is a vector?



Vectors

There's three ways to think of a vector.



Physics

$$\vec{\mathbf{v}}$$

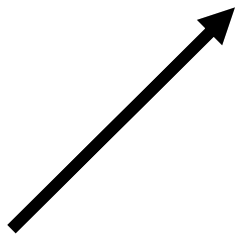
Mathematics

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Computer Science

Vectors

Physicists think of vectors as an arrow with a tip at the end.



This arrow denotes direction.

Vectors

Computer scientists think of vectors as a matrix.

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

This matrix denotes position.

Vectors

By position, we mean the point. For example:

$\binom{1}{2}$ equals the point (1, 2) on the xy-plane.

Vectors

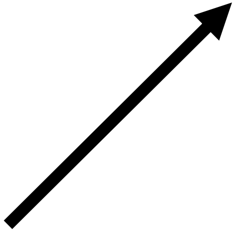
Mathematicians think of vectors as elements.

$$\vec{\mathbf{v}}$$

In mathematics, vectors are a generalized idea.

Vectors

In mathematics, we do things like:



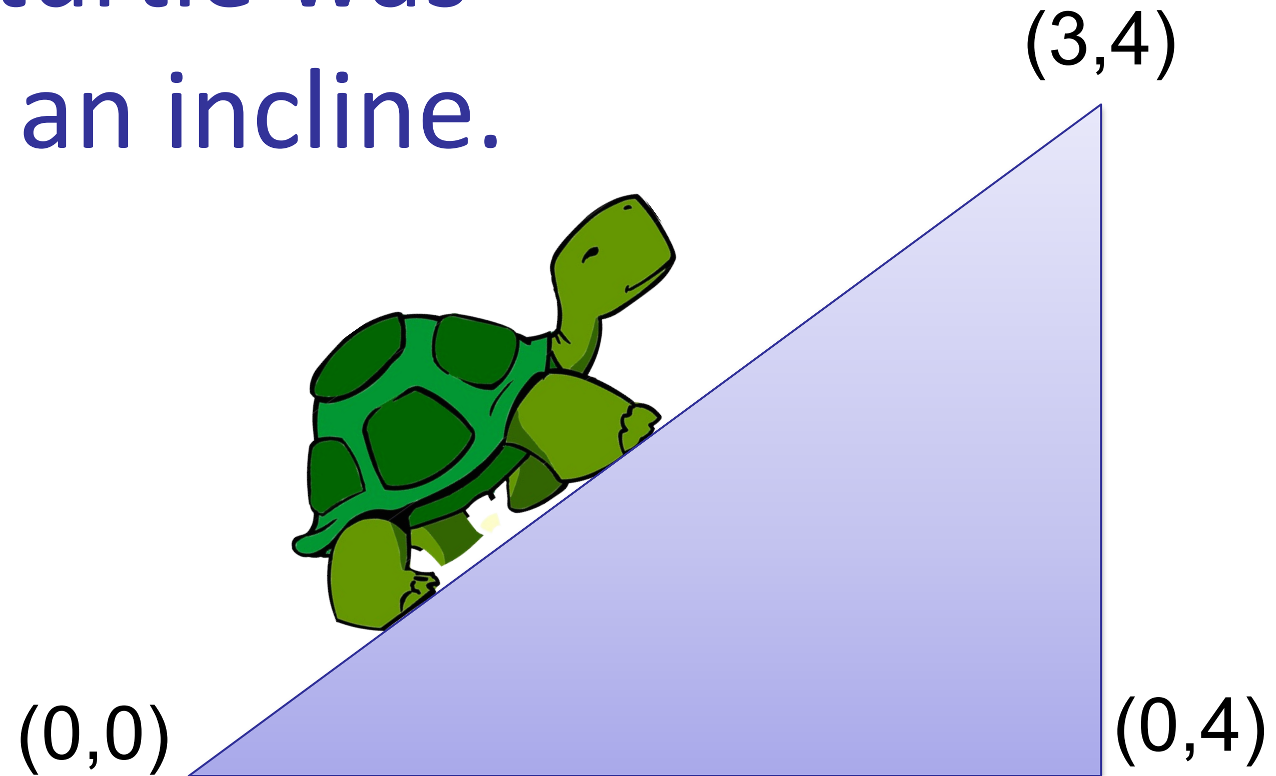
$$= \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Vectors

Vectors give us an intuitive notion of distance and direction.

Vectors

Suppose a turtle was crawling up an incline.

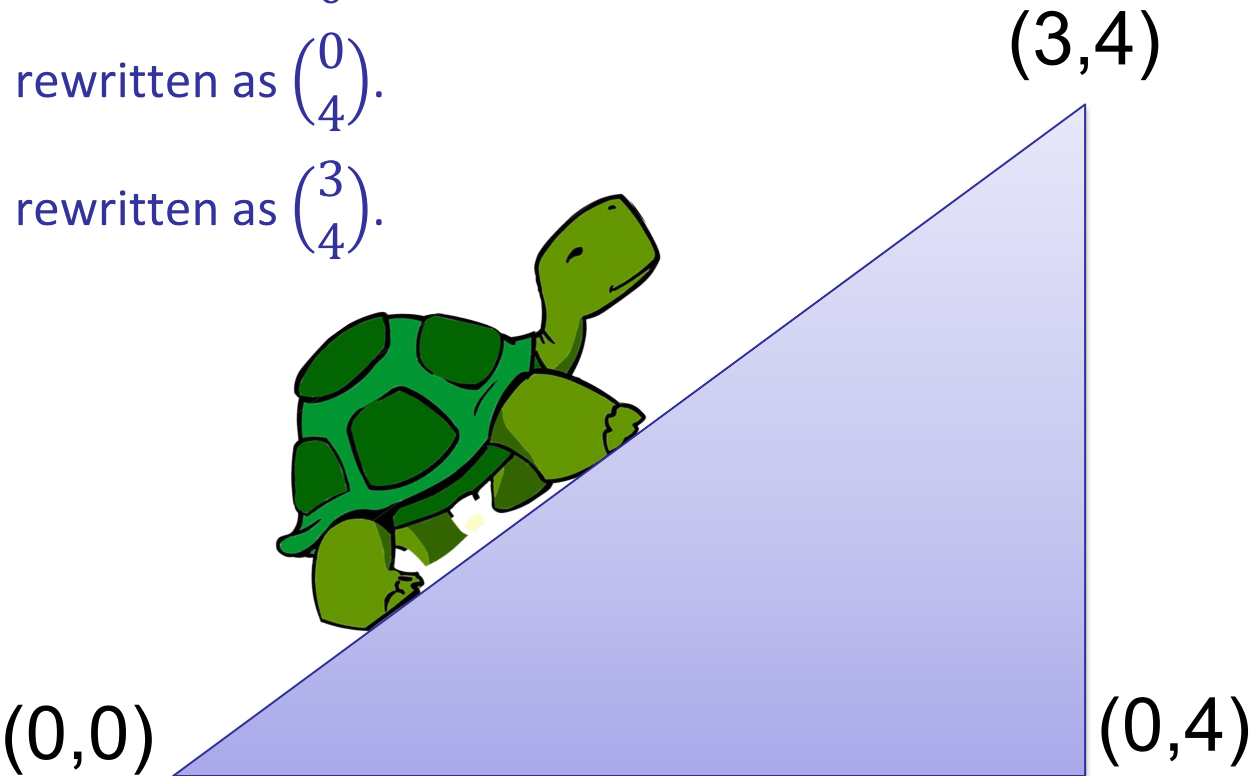


Vectors

The point (0,0) can be rewritten as $\binom{0}{0}$.

The point (0,4) can be rewritten as $\binom{0}{4}$.

The point (3,4) can be rewritten as $\binom{3}{4}$.



Vectors

We can use vectors to solve several kinds of problems.

Vectors

Suppose $\vec{\boldsymbol{x}}$ and $\vec{\boldsymbol{y}}$ is a vector.

Then we can add them together.

So like $\vec{\boldsymbol{x}} + \vec{\boldsymbol{y}}$.

Vectors

Suppose $\vec{\boldsymbol{x}}$ is a vector, and $\boldsymbol{a}$ is a scalar (or a number).

Then we can multiply them together.

Vectors

Example:

$$a \cdot \vec{x}$$

Vectors

Vectors are not numbers. But they can include numbers, e.g.

$$\vec{\boldsymbol{x}} = \langle 3,4 \rangle$$

$$\vec{\boldsymbol{y}} = \langle 0,2 \rangle$$

Vectors

Let's add these two vectors together:

$$\vec{\boldsymbol{x}} = <3,4>$$

$$\vec{\boldsymbol{y}} = <0,2>$$

Vectors

$$\vec{x}+\vec{y} = <3,4> + <0,2>$$

$$= <3+0,\ 4+2>$$

$$= <3,6>$$

Vectors

Let's multiply them: $\boldsymbol{a} \cdot \vec{\boldsymbol{x}}$

$\vec{\boldsymbol{x}}$ = <3,4>

$\boldsymbol{a} = 5$

Vectors

Let's multiply them: $\boldsymbol{a} \cdot \vec{\boldsymbol{x}}$

$$\boldsymbol{a} \cdot \vec{\boldsymbol{x}} = 5 \cdot <3,4>$$

$$= <5 \cdot 3,\ 5 \cdot 4>$$

$$= <15,\ 20>$$

Vectors

In general, we can write vectors like this:

$$\vec{x} = < x_1, x_2, \ldots, x_n >$$

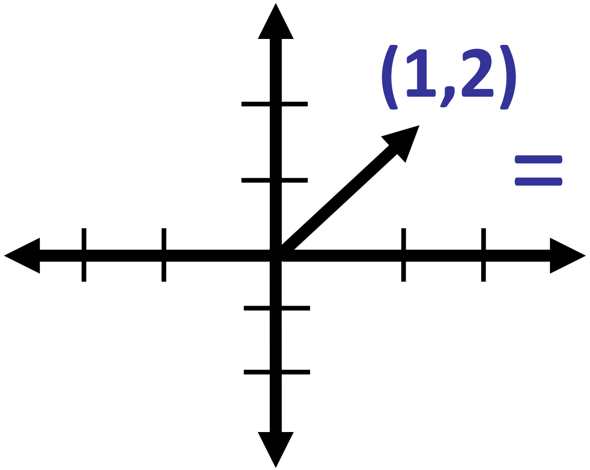
where $x_1, x_2, \ldots, x_n$ are numbers.

Vectors

We can also take the length of vectors.

Vectors

We can also take the length of vectors.



$$= \begin{pmatrix} 1 \\ 2 \end{pmatrix} = < 1,2 > = \vec{\boldsymbol{x}}$$

Vectors

The length of a vector is written in this form:

$$|\vec{\boldsymbol{x}}|$$

Vectors

If $\vec{\boldsymbol{x}} = <1,2>$

Then:

$$|\vec{\boldsymbol{x}}| = \sqrt{1^2 + 2^2}$$

$$= \sqrt{5}$$

Vectors

Note: You cannot multiply two vectors!

$$\cancel{\vec{\boldsymbol{x}} \times \vec{\boldsymbol{y}}}$$

Vectors

But you can do something else.
You can take their dot product.

$$\vec{\boldsymbol{x}} \cdot \vec{\boldsymbol{y}}$$

Vectors

The dot product is not multiplication in the sense of numbers.

$$\vec{\boldsymbol{x}} \cdot \vec{\boldsymbol{y}}$$

Vectors

Example:

$\vec{x} = \langle 3,4 \rangle$

$\vec{y} = \langle 0,2 \rangle$

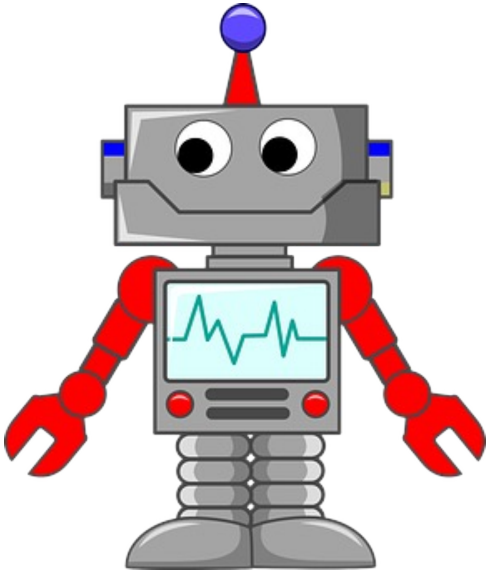
$$\vec{x} \cdot \vec{y} = (3 \times 0) + (4 \times 2)$$
$$= 8$$

Vectors

In general, we can write the dot product of two vectors like this:

$$\vec{x} \cdot \vec{y} = \sum_i x_i y_i$$

Probability



Probability

Imagine this scenario

Probability

Suppose you had a coin.

Probability

The coin has two sides:

- Heads
- Tails

Probability

If you flip the coin 10 times, how many times would it land on Heads?

Probability

How many times would it land on Tails?

Probability

What if you flipped it 100 times?

How many times would it land on Heads? What about Tails?

Probability

Now…

What if you flipped the coin 10,000 times? How many times would it land on Heads or Tails?

Probability

The answer to these questions is this:

We do not know with 100% certainty which side a coin will land on in 10,000, 100, 10, or even 1 try.

Probability

But…

We can take a really good guess as to how many times the coin will land on Heads or Tails.

Probability

Because certainty is out of the question, it makes sense to ask another question.

Probability

Here's a better question: What's the likelihood that the coin will land on Heads or Tails?

Probability

If the coin is flipped one time, the likelihood that the coin will land on Heads is 50%.

Probability

Likewise, if the coin is flipped one time, the likelihood that the coin will land on Tails is 50%.

Probability

Why do we know this?

Answer: Probability

Probability

If we flipped a coin once, what is the likelihood it will land on Heads?

$$\mathbf{Pr\{\mathit{Heads}\}} = \frac{\boldsymbol{Number\ of\ outcomes\ to\ get\ Heads}}{\boldsymbol{Number\ of\ possible\ outcomes}}$$

$$= \frac{\mathbf{1}}{\mathbf{2}}$$

$$= \mathbf{.50 = 50\%}$$

Probability

If we flipped a coin twice, what is the likelihood it will land on Heads?

$$\Pr\{Heads\} = \frac{Number\ of\ outcomes\ to\ get\ Heads}{Number\ of\ possible\ outcomes}$$

$$= \frac{2}{4}$$

$$= .50 = 50\%$$

Probability

Sometimes, events happen at the same time.

Probability

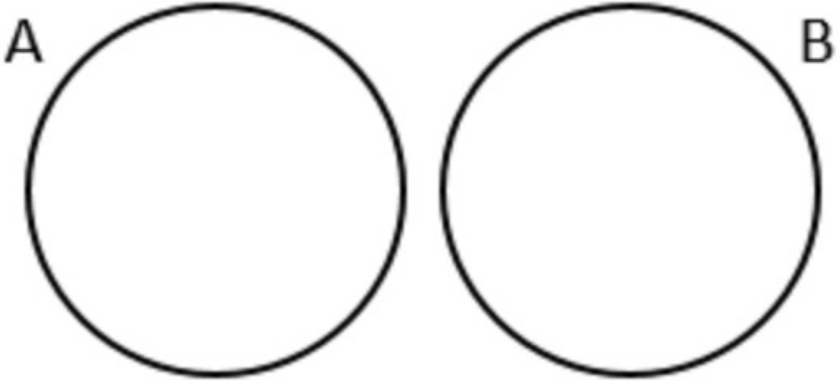
Other times, they do not happen at the same time.

Probability

If two events don't happen at the same time, they are said to be mutually exclusive.

Probability

Mutually Exclusive Events



$$P(A \text{ or } B) = P(A) + P(B)$$

Probability

If *A* and *B* are mutually exclusive events, then the probability of *A* happening **OR** the probability of *B* happening is P(*A*) + P(*B*).

Example:

If we toss a coin, either Heads or Tails might turn up, but not Heads and Tails at the same time.

Probability

Another way to write

$P(A \text{ or } B) = P(A) + P(B)$

is

$P(A \cup B) = P(A) + P(B)$

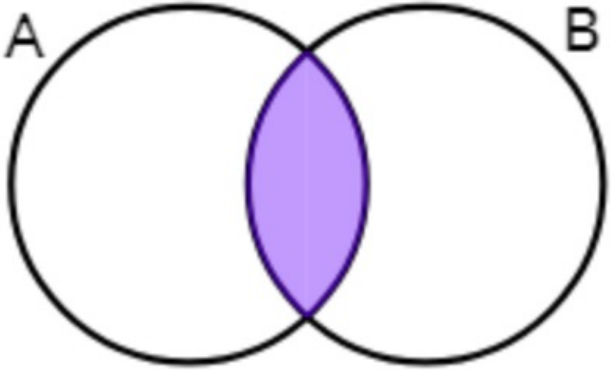
Probability

If *A* and *B* are non-mutually exclusive events, then the probability of *A* happening **OR** the probability of *B* happening is:

$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B).$$

Probability

Non-Mutually Exclusive Events



P(A or B) = P(A) + P(B) – P(A and B)

Probability

Another way to write

P(A or B) = P(*A*) + P(*B*) – P(A and B)

is

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Probability

Probabilistic models consist of a sample space of mutually exclusive possible outcomes, together with a probability for each outcome.

Probability

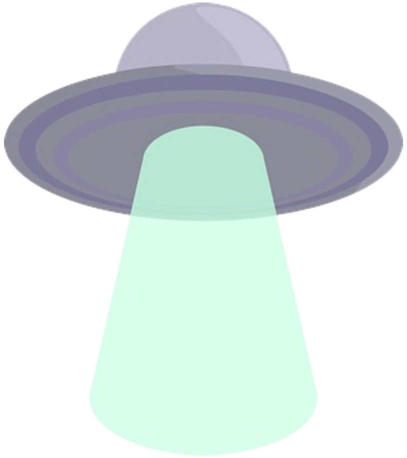
Example:

In a model of the weather tomorrow,

The outcomes might be **sunny**, **cloudy**, **rainy**, and **snowy**.

Probability

A subset of these outcomes is called an event.



Probability

For example:

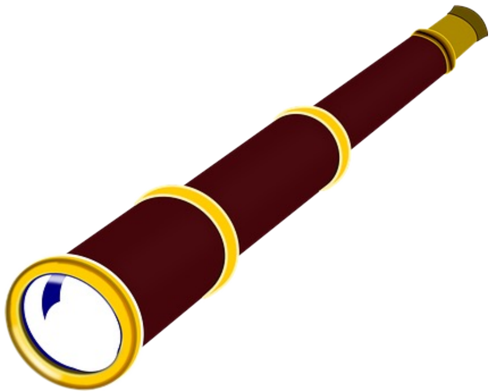
The event of precipitation is the subset consisting for {**rainy**, **snowy**}.

Probability

Now onto Distributions

Probability

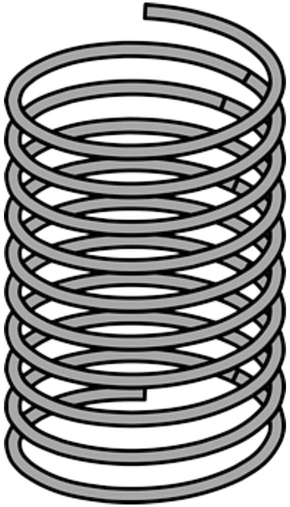
Distributions describe the relationship of observations in a sample space.



There are several kinds of distributions.

- Frequency Distributions
- Probability Distributions
- Normal Distributions
- Binomial Distributions
- Etc.

Distributions are described by density functions.



Probability

Density functions describe the likelihood of the proportion of observational changes in a distribution.

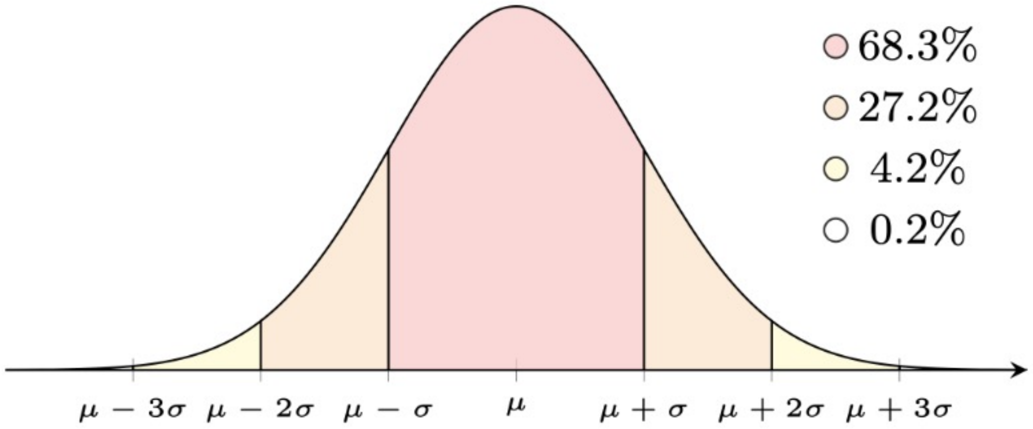
We will take a look at the most popular distribution:

Normal distributions.

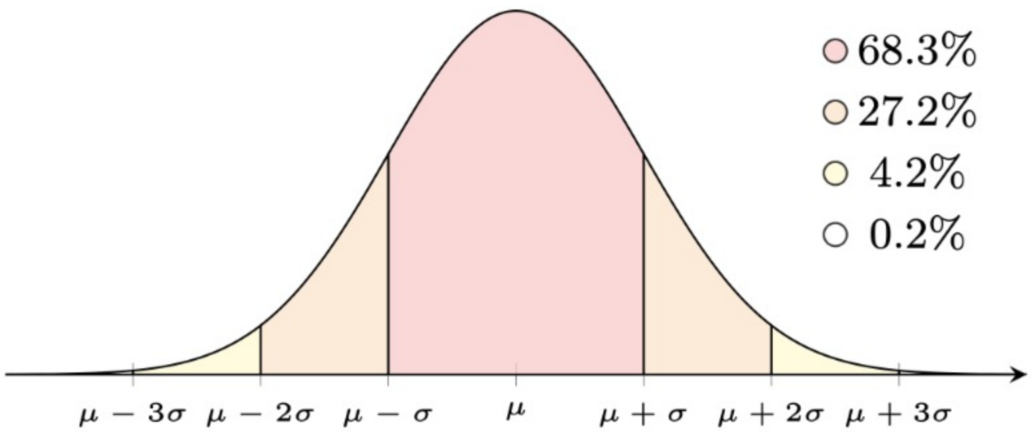
Normal Distributions have two parameters:

- Mean ($\boldsymbol{\mu}$)
- Standard Deviation ($\boldsymbol{\sigma}$)

Normal Distribution:



Normal Distribution:

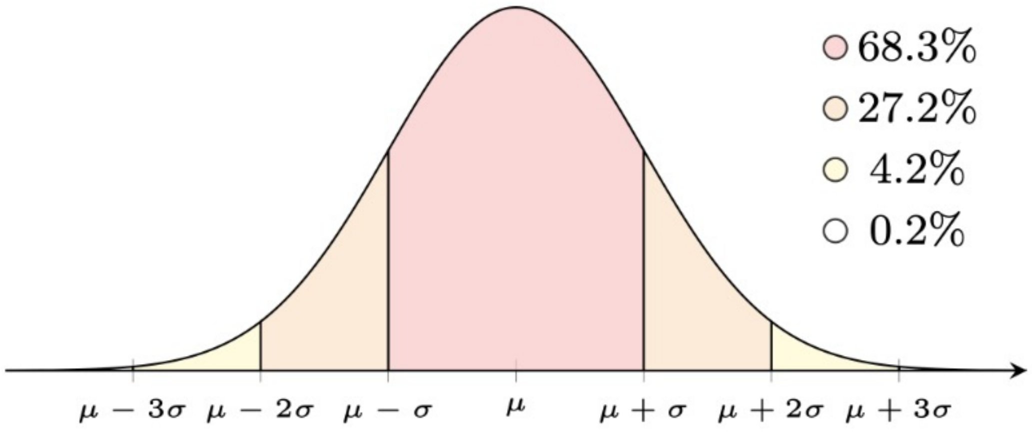


68% of data is within 1 standard deviation of the mean $\boldsymbol{\mu}$.

95% of data is within 2 standard deviations of the mean $\boldsymbol{\mu}$.

99.7% of data is within 3 standard deviations of the mean $\boldsymbol{\mu}$.

Normal Distribution:



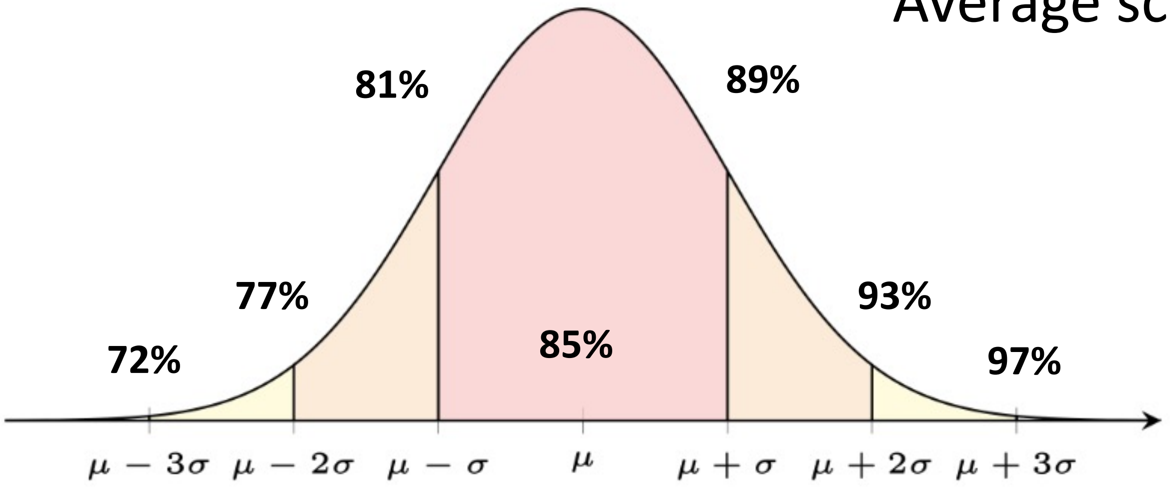
Being 1 standard deviation of the mean is $\boldsymbol{\mu \pm \sigma}$.

Being 2 standard deviations of the mean is $\boldsymbol{\mu \pm 2\sigma}$.

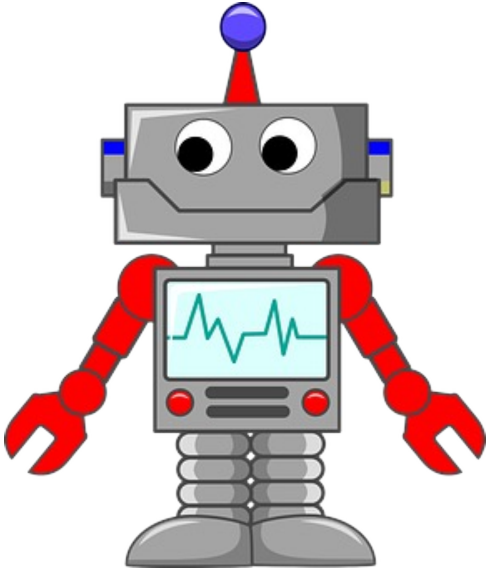
Being 3 standard deviations of the mean is $\boldsymbol{\mu \pm 3\sigma}$.

Example: Test Scores

Average score: 85%

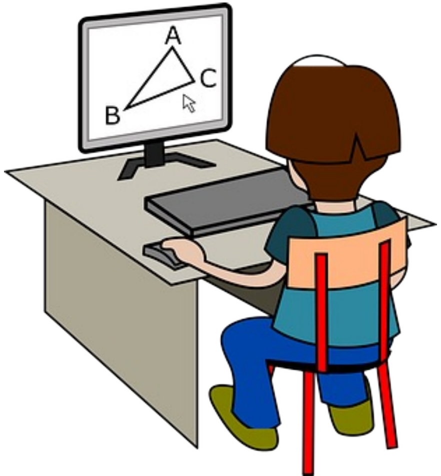


Graph Theory

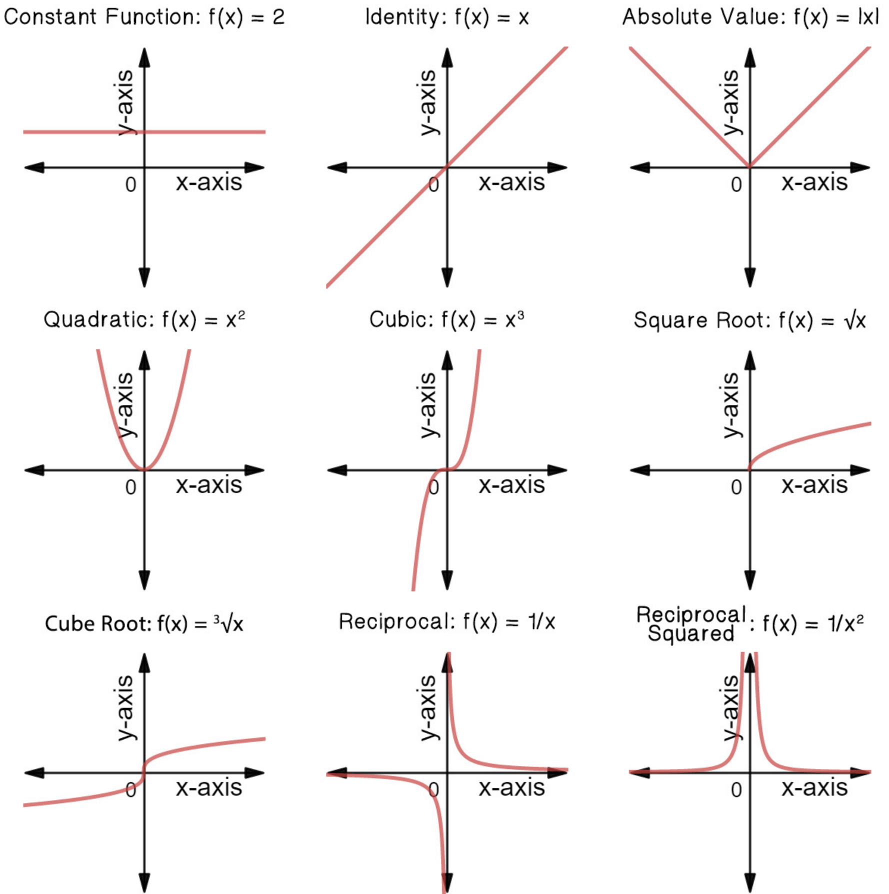


Graph Theory

Graphs in graph theory are different from the graphs you've been previously exposed to.

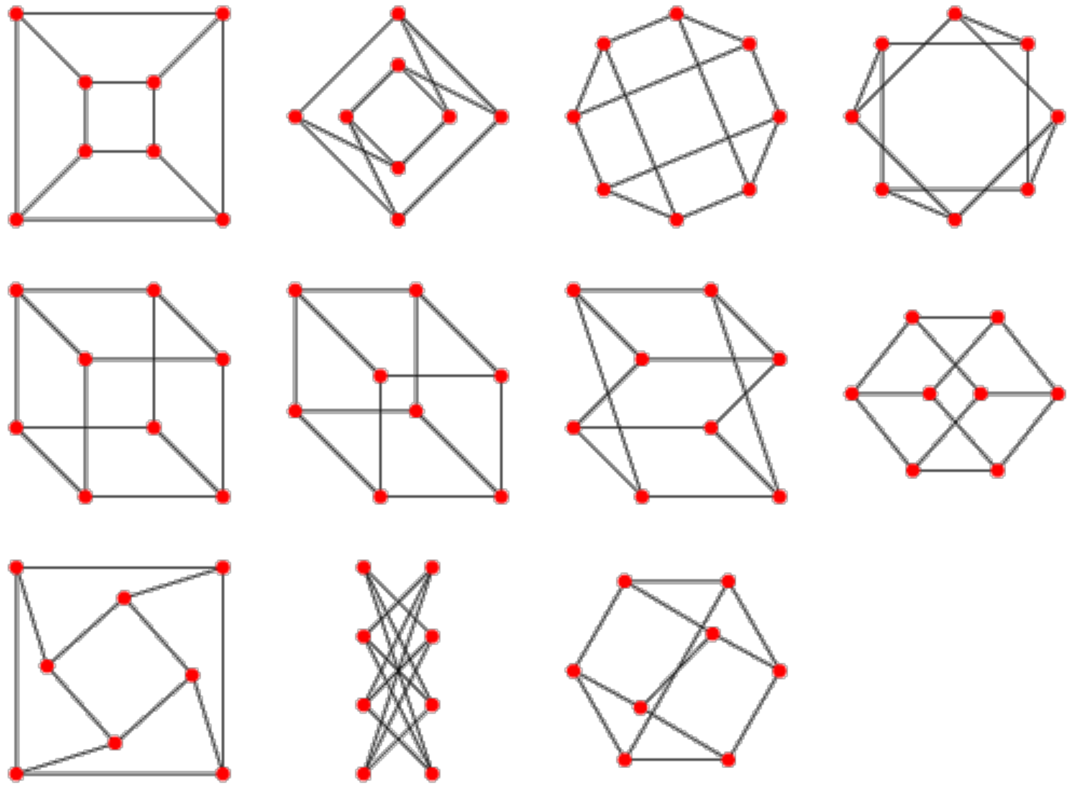


Graph Theory



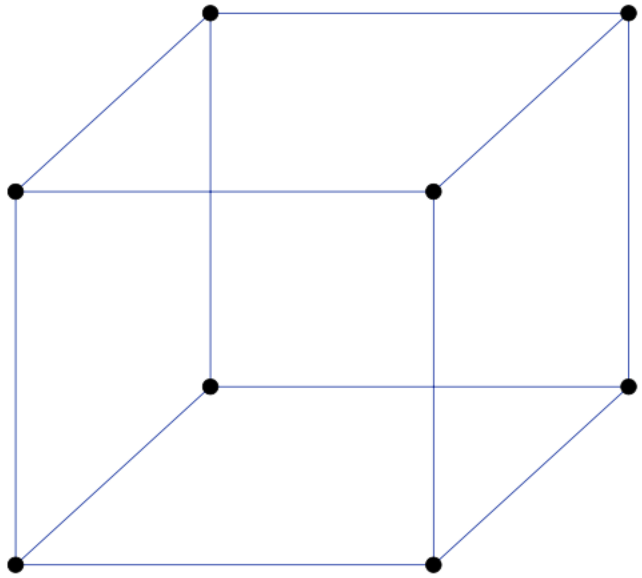
Graphs in Algebra and Calculus:

Graph Theory



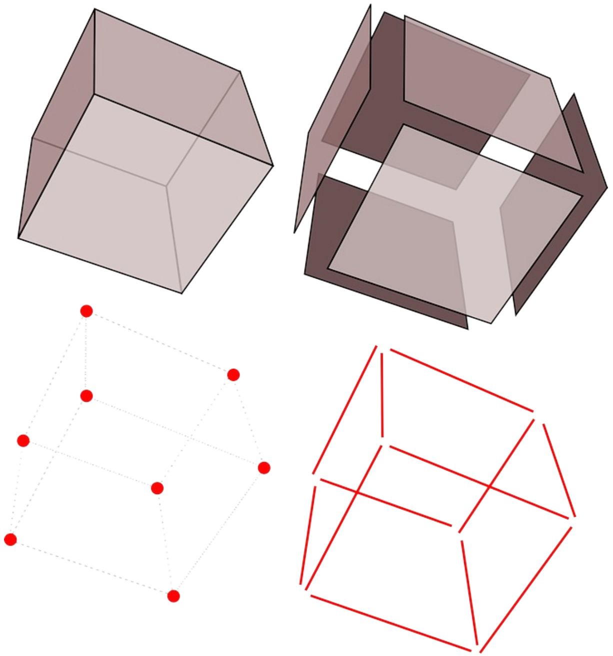
Graphs in Graph Theory:

A closer example of a graph in Graph Theory



Graph Theory

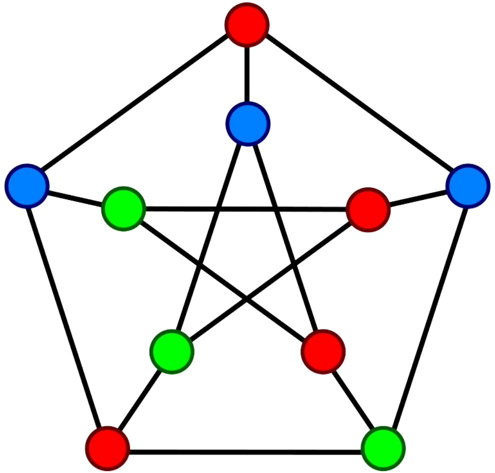
Geometric intuition behind graphs in graph theory.



Graph Theory

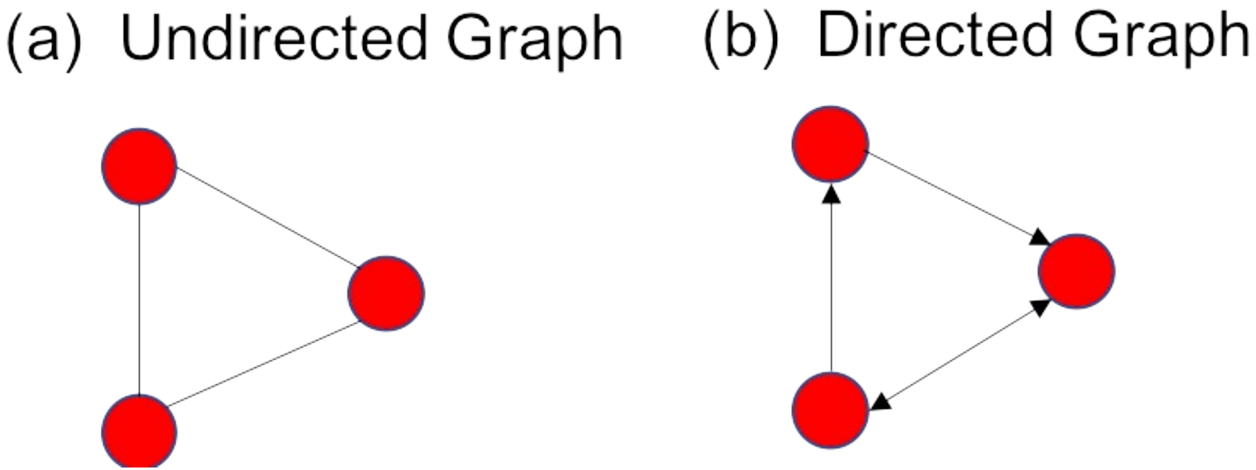
A graph consists of:

- vertices (dots)
- edges (lines)



Graph Theory

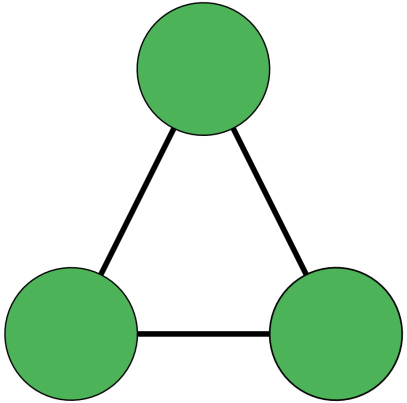
Graphs can be undirected or directed.



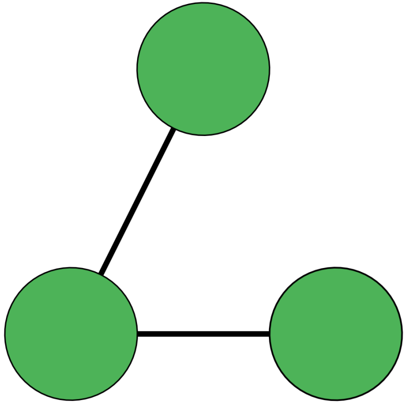
Graph Theory

Graphs can be cyclic or acyclic.

Cyclic

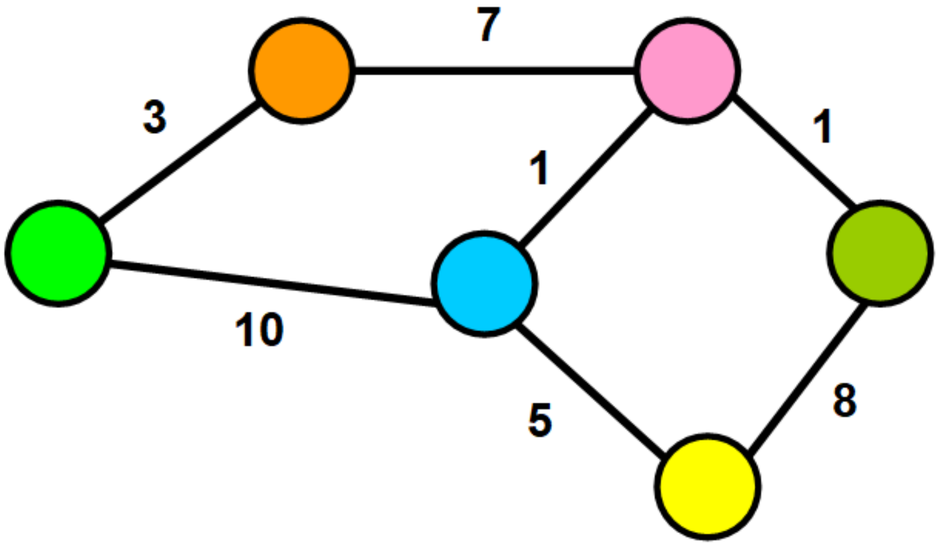


Acyclic

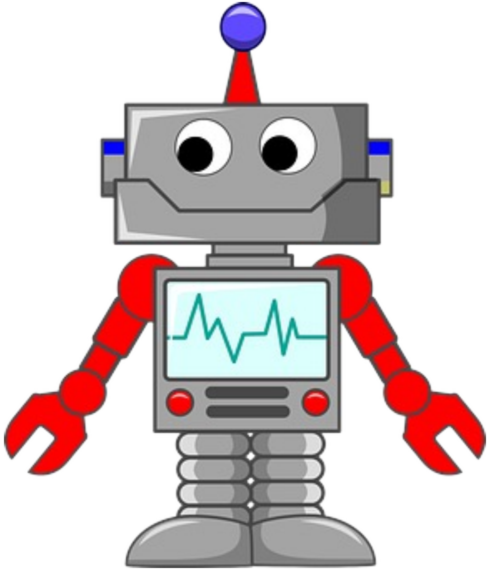


Graph Theory

Graphs can be weighted (or unweighted).



Algorithms



Heuristic Algorithms

A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot.

Heuristic Algorithms

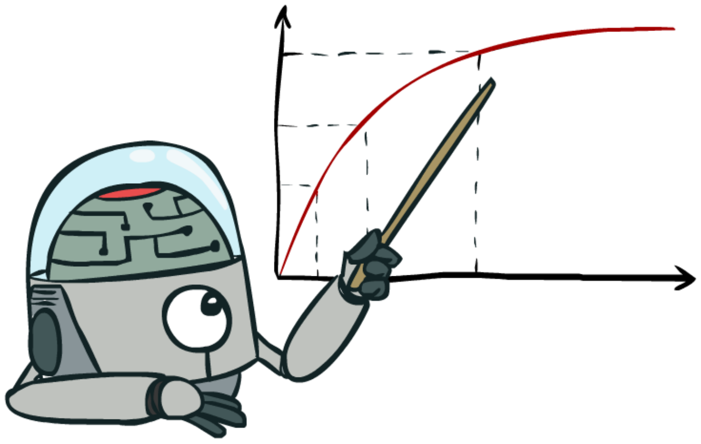
This is a kind of shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed.

Heuristic Algorithms

So *why do we need heuristics?*

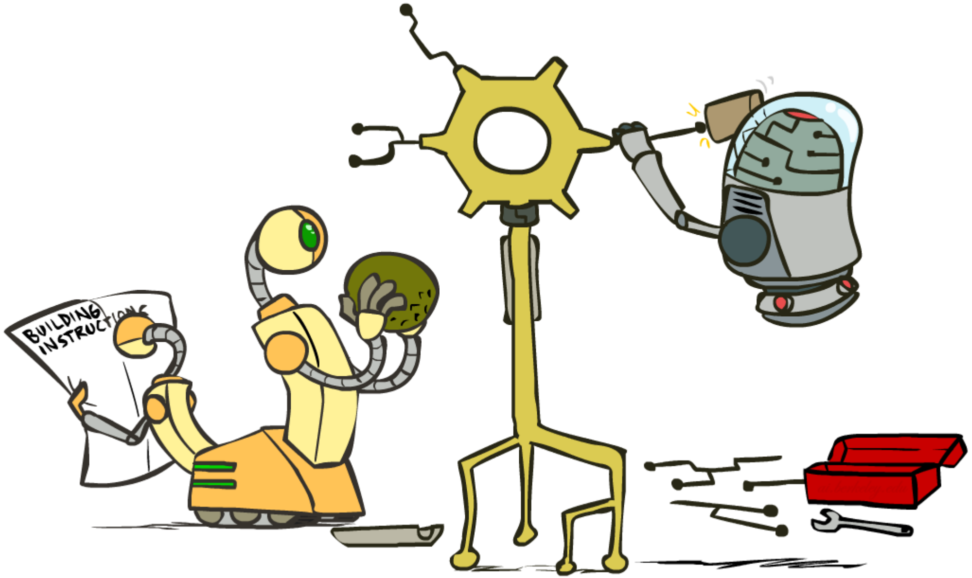
Heuristic Algorithms

One reason is to produce, in a reasonable amount of time, a solution that is good enough for the problem in question.



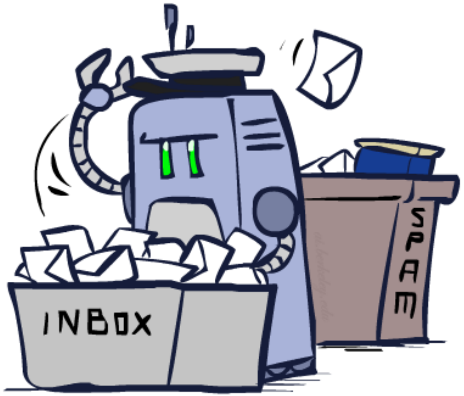
Heuristic Algorithms

It doesn't have to be the best—an approximate solution will do since this is fast enough



Greedy Algorithms

Greedy algorithms are an approach to solving certain kinds of optimization problems.

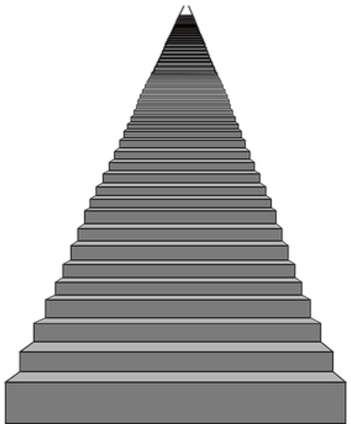


Greedy Algorithms

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment, without worrying about the future result it would bring. In other words, the locally best choices aim at producing globally best results.

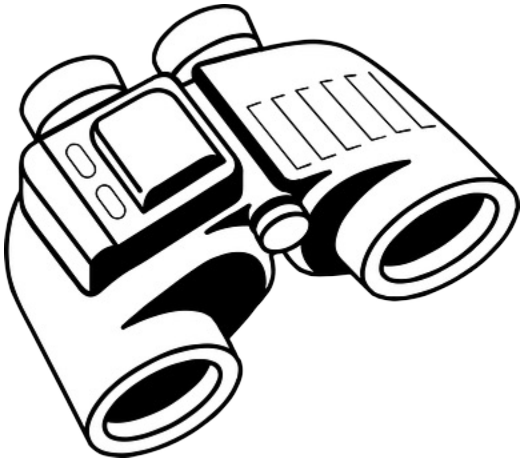
Greedy Algorithms

A greedy algorithm builds a solution by going one step at a time through the feasible solutions, applying a heuristic to determine the best choice.



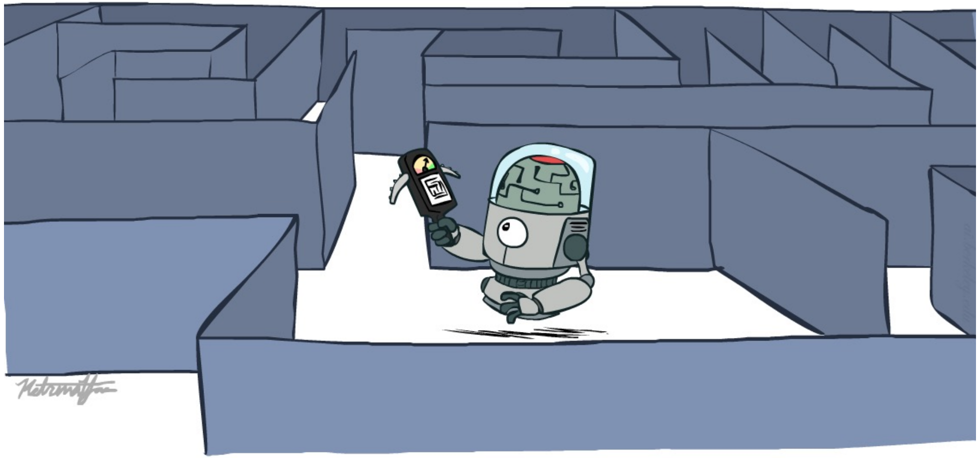
Greedy Algorithms

A heuristic applies an insight to solving the problem, such as always choose the largest, smallest, etc.



Search Algorithms

The main purpose of searching algorithms is to check an element or retrieve it from any data structure.



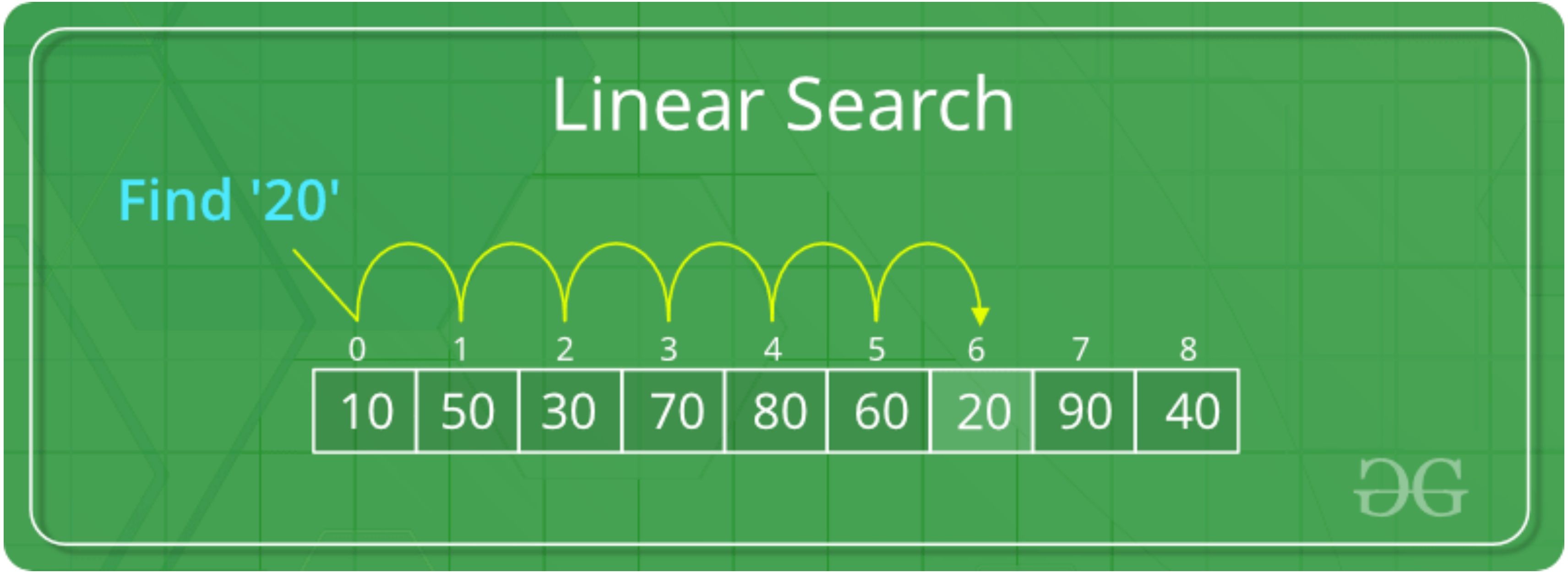
Search Algorithms

These searching algorithms are classified into two different parts generally based on the type of searching:

Sequential search: List or array is traversed sequentially and every element is checked. (e.g.: **Linear Search**)

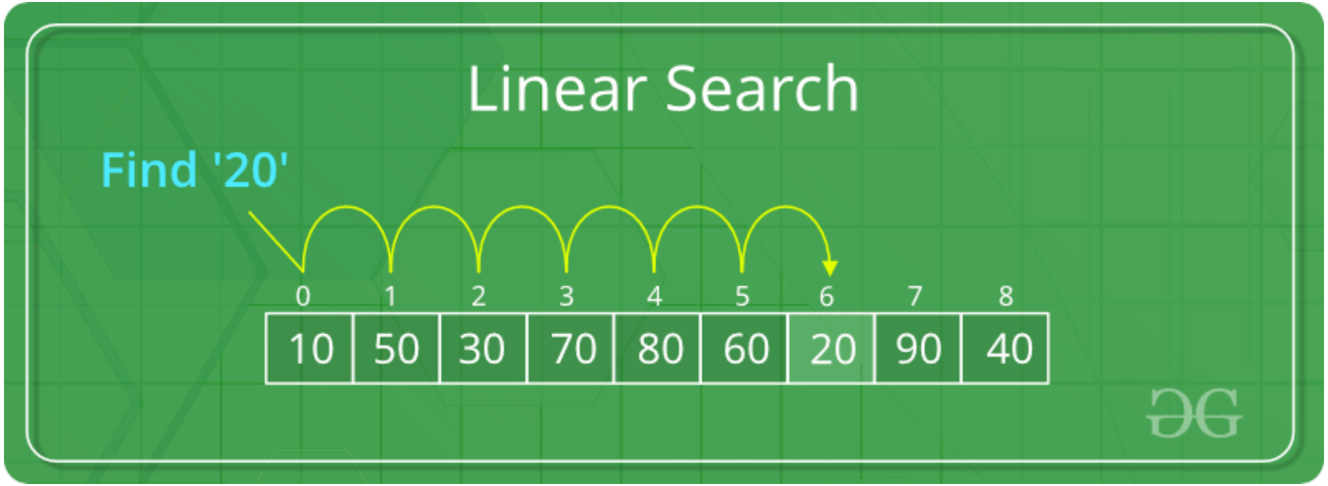
Interval search: Designed for sorted data structures and more efficient than sequential search algorithms as these are repeatedly target the center of the data structure and divide the search space in half. (e.g.: **Binary Search**)

Linear Search



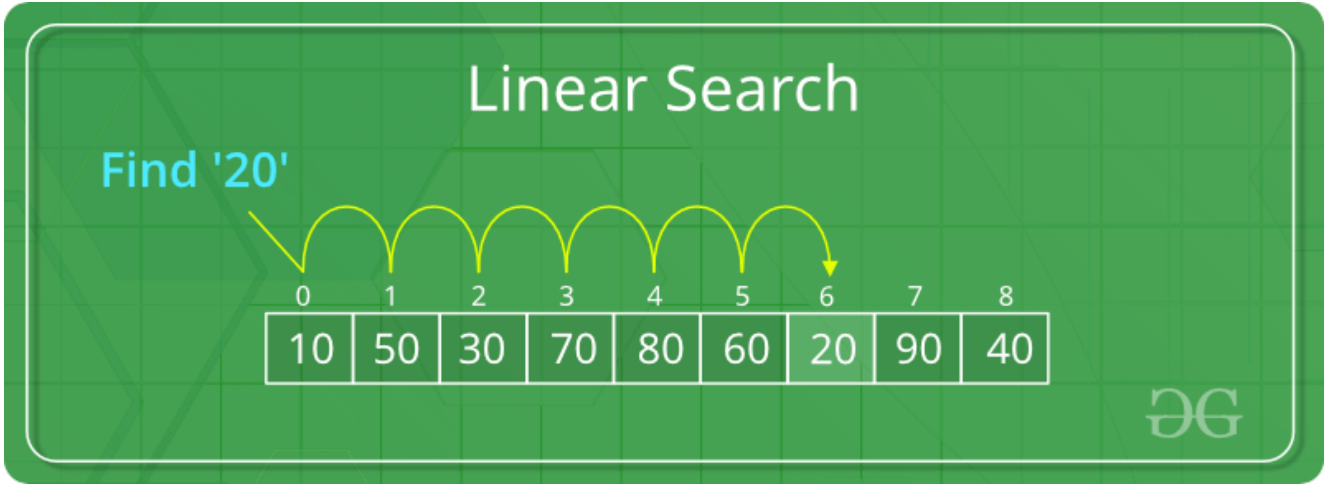
Linear Search

This algorithm is a very simple algorithm. Here, a sequential search is made throughout every element in the data structure one by one.

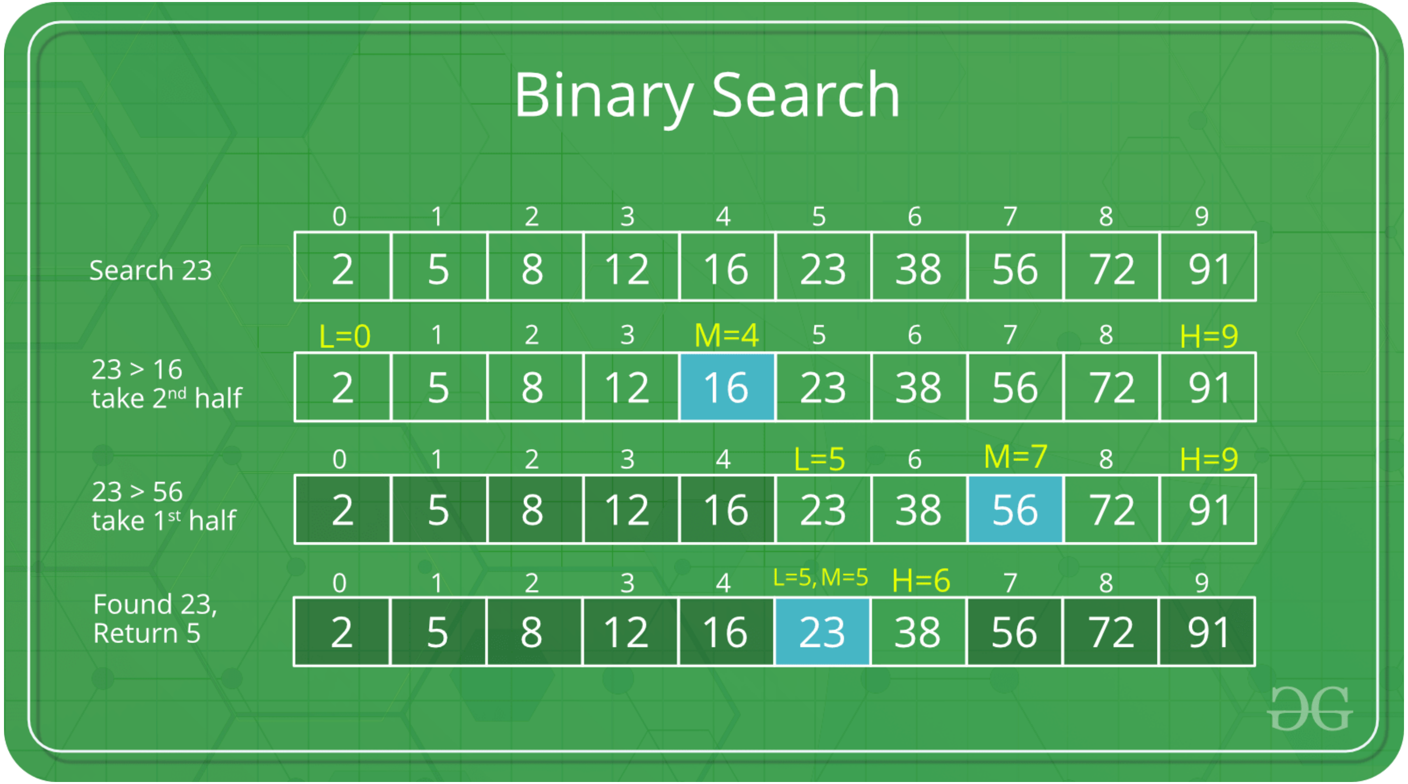


Linear Search

If the match is found, it is returned otherwise searching process continues until the end of the data structure.

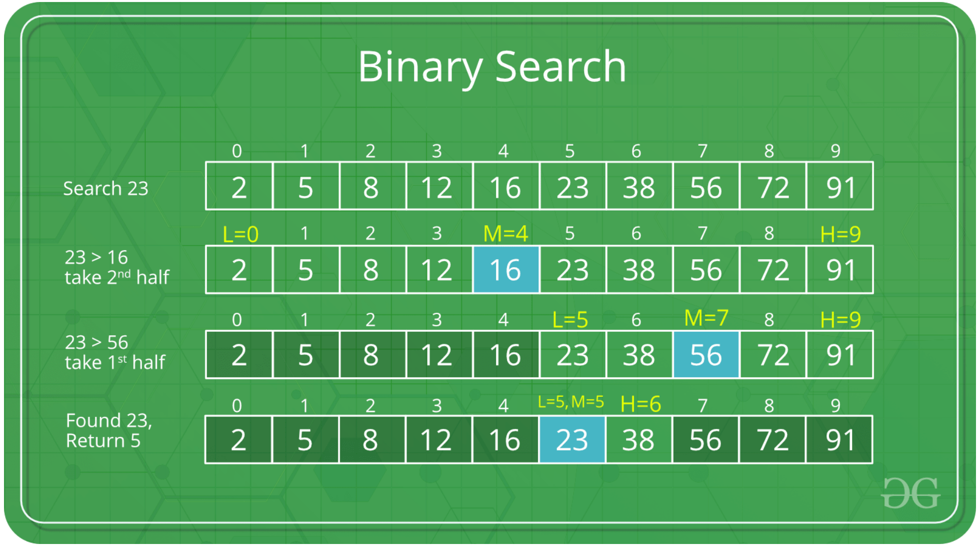


Binary Search



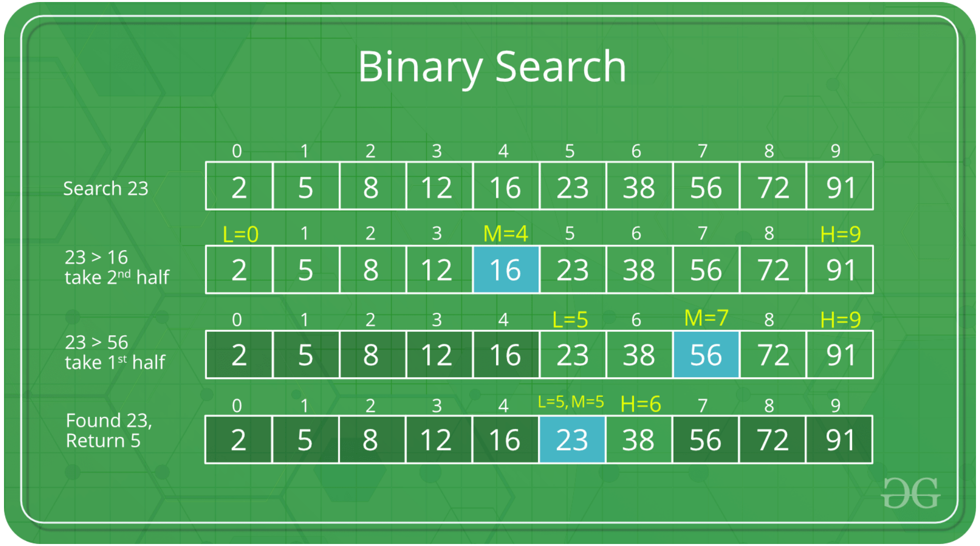
Binary Search

This is a fast searching algorithm of the runtime complexity of O(log N).



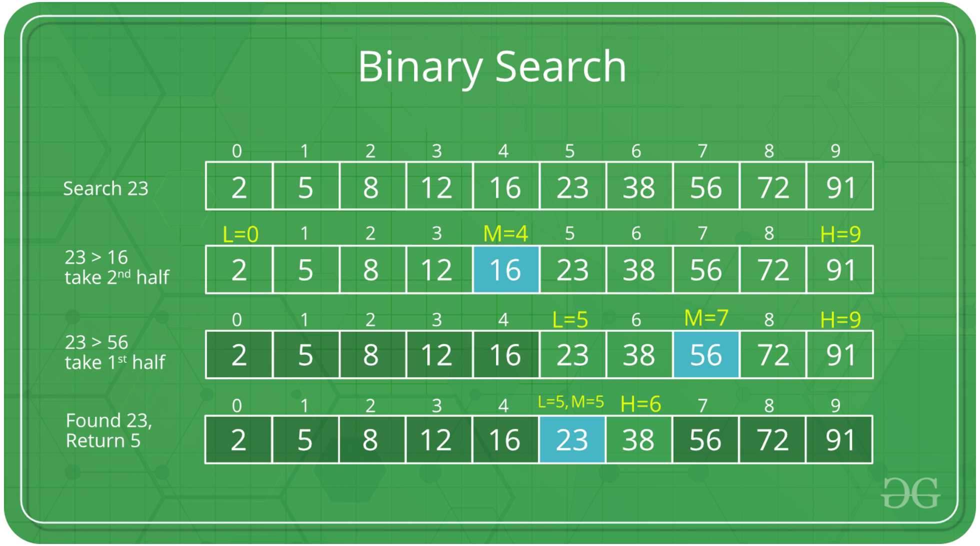
Binary Search

The data collection should be in the sorted form in order to work this algorithm correctly.



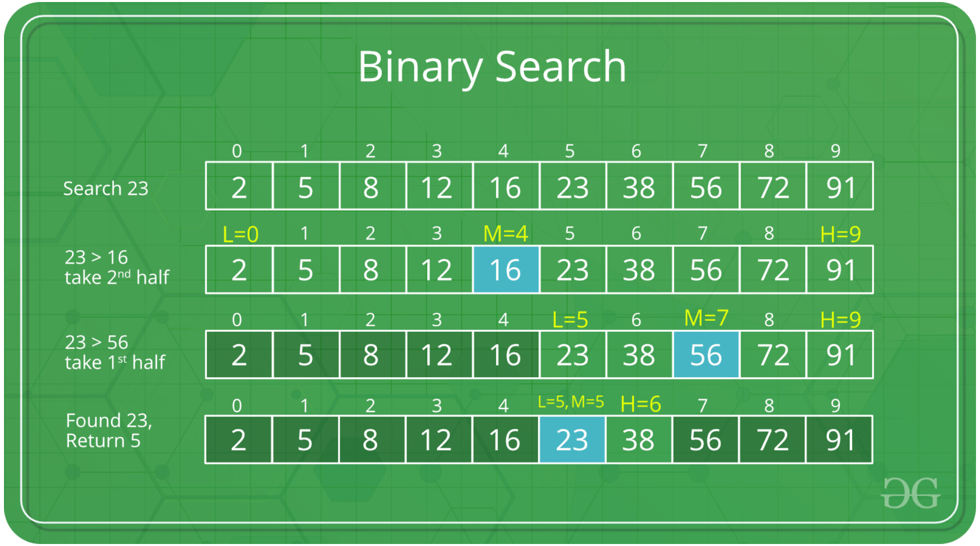
Binary Search

How does Binary Search work?



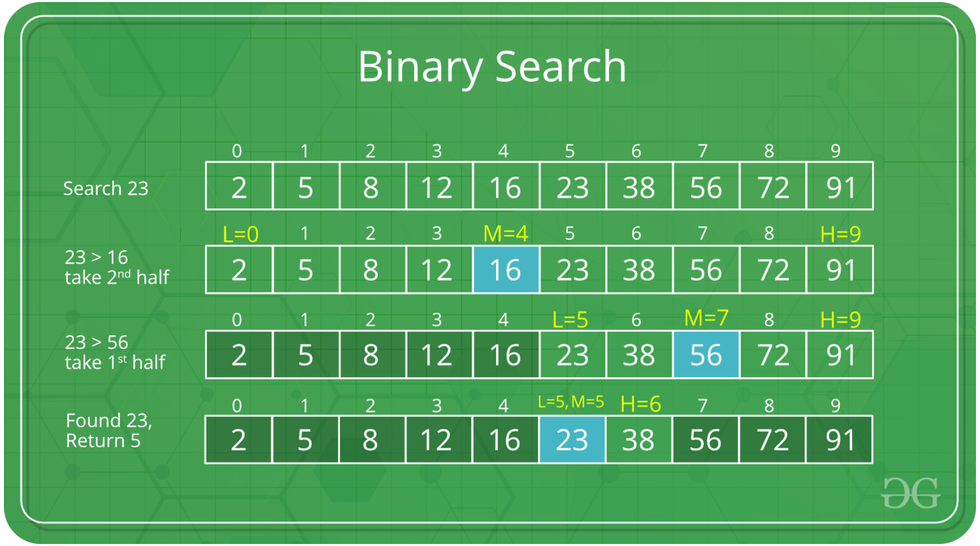
Binary Search

Binary search looks for a particular item by comparing the middlemost item of the collection.



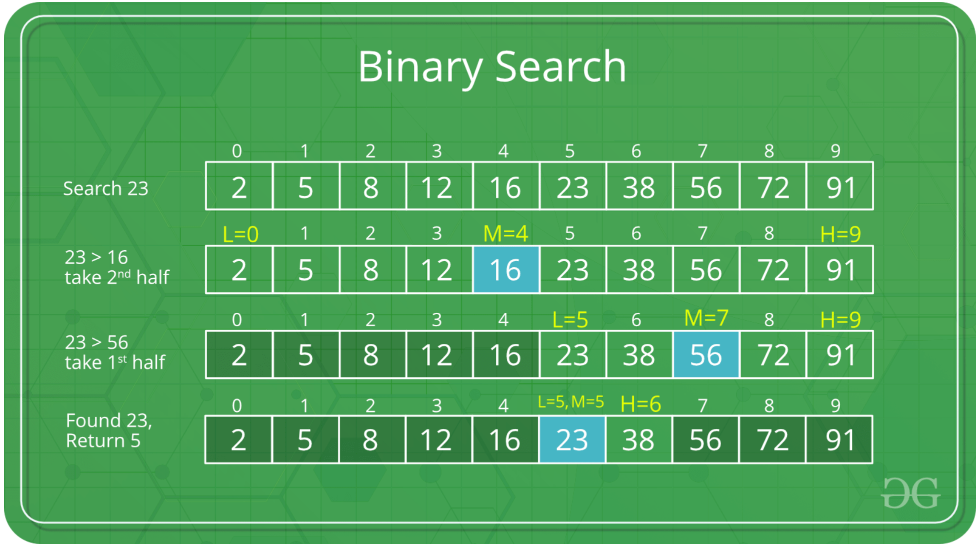
Binary Search

If a match occurs, then the index of the item is returned.



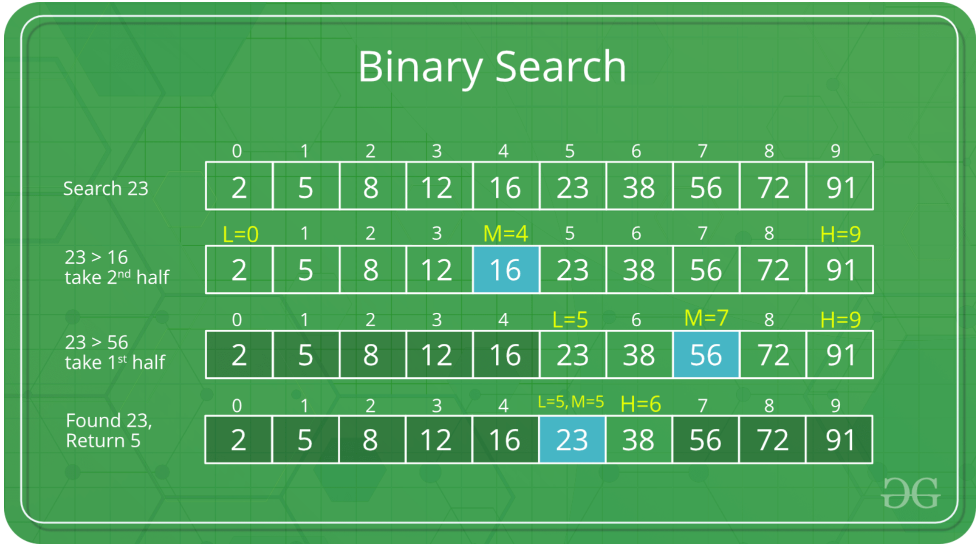
Binary Search

If the match does not occur, it checks whether the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item.



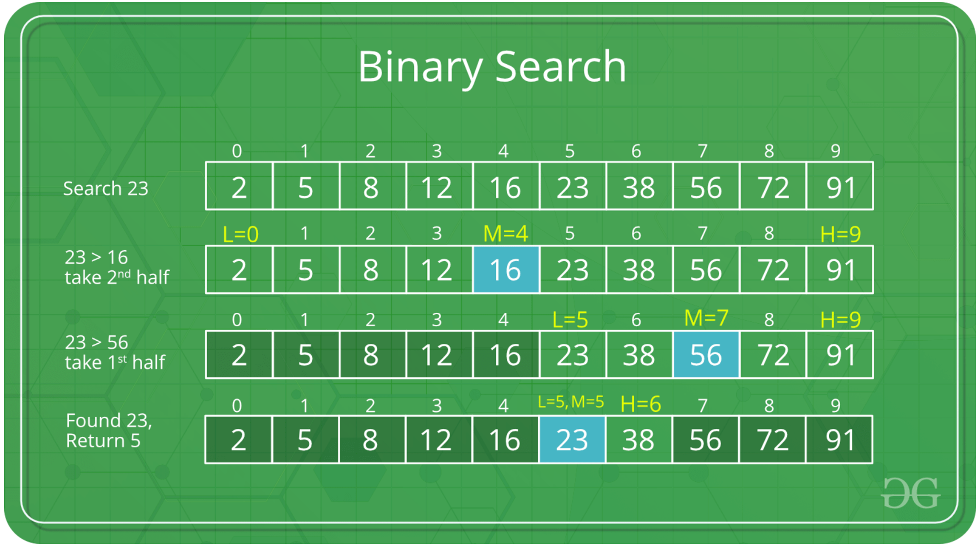
Binary Search

Otherwise, the item is searched for in the sub-array to the right of the middle item.



Binary Search

Until the subarray size reduces to zero this process continues on the sub-array as well.



Alan Turing

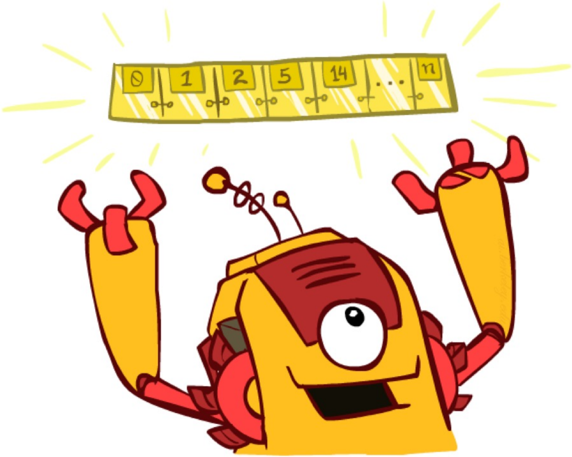
Known as The Father of Modern Computer Science

Alan Turing

Alan Turing was an amazing mathematician of the 20th century. He solved all sorts of problems!

Alan Turing

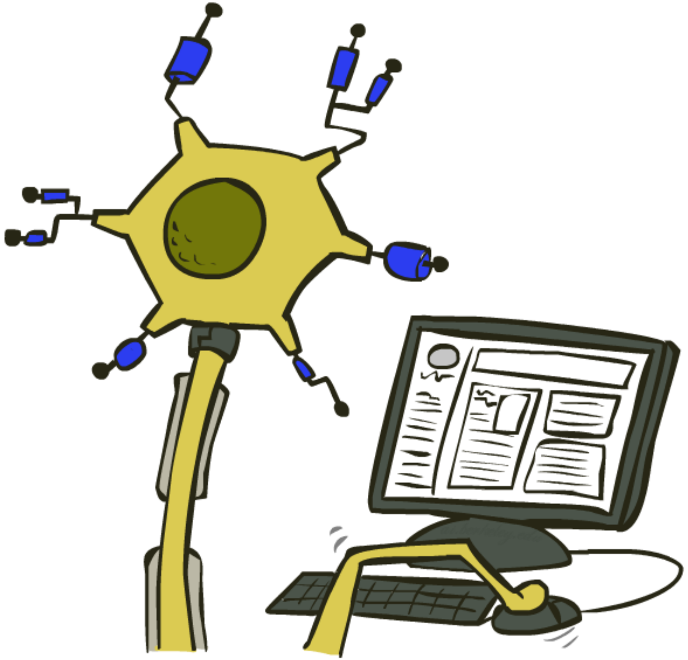
He’s very well known for his development of a Turing Machine.



Alan Turing

A Turing Machine is an abstract machine which manipulates symbols on a strip of tape according to a table of rules.

Alan Turing

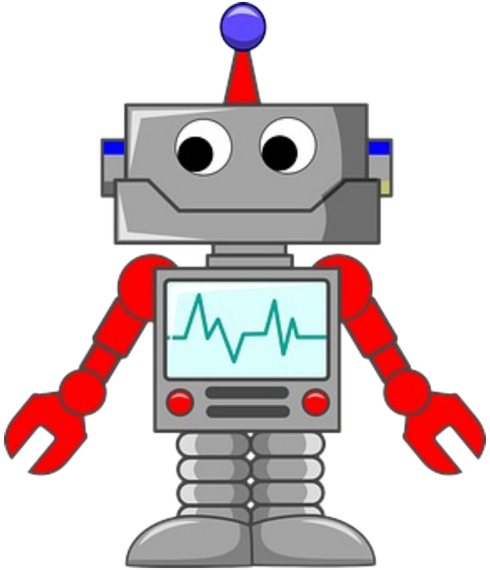


He's also known for what is called the Turing Test.

Alan Turing

The Turing Test is a test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human.

Python



Python

Python Programming Language

Python

What is Python?

Python is a high-level and general-purpose programming language.

Python

Programiz

Programiz is an online compiler

Python

We will use Programiz to compile code in Python.

Python

We're going to look at a few examples of how programming is done.

Programs we wrote:

- Print 'Hello World!'
- Added numbers
- Concatenated symbols
- Initialized variables
- Imported Packages

Programs we wrote:

- Allow the user to enter descriptions of four different types of shapes and ultimately output the perimeter/circumference and area of the figures.

Elements of Programming

In Python

Programming in Python

To program in Python, you need to:

- Compose a program by typing it into a file named, say, `myprogram.py`.
- Run (or execute) it by typing "`python myprogram.py`" in the terminal window.

Sample program in Python

```
Import stdio

# Write ‘Hello, World’ to standard output.
stdio.writeln(‘Hello, World’)
```

Analysis of the program

Line 1 contains an import statement.

That statement tells Python that you intend to use the features defined in the stdio module (also known as: stdio.py)

Analysis of the program

Line 2 is a blank line. Python ignores blank lines!

Analysis of the program

Line 3 contains a comment which serves to document the program.

In Python, a comment begins with the ‘#’ character and extends to the entire end of the line.

Python ignores comments.

Analysis of the program

Line 4 is the heart of the program. It is the statement that calls the `stdio.writeln()` function to write one line with the given text on it.

Built-In Types of Data

A data type is a set of values and a set of operations defined on those values.

Examples of built-in data types are `int` (for integers), `float` (for floating-point numbers), `str` (for sequences of characters) and `bool` (for true-false values).

Built-In Types of Data

Type	set of values	common operators	sample literals
int	integers	+ - * // % **	99 12 2147483647
float	float-point numbers	+ - * / **	3.14 2.5 6.022e23
bool	true-false values	and or not	True False
str	sequences of characters	+	'AB' 'Hello' '2.5'

More terminology

Sample Code

```
a = 1234
b = 99
c = a + b
```

Terms

The code creates:

Three objects (each of type `int`)

Literals – `1234 and 99`

Expression – `a + b`

Terms

Literal: A Python-code representation of a data-type value.

Operators: A Python-code representation of a data-type operation. For example, uses + and * to represent addition and multiplication.

Terms

The code binds variables a, b, and c to those objects using assignment statements.

The end result is that variable c is bound to an object of type `int` whose value is 1333.

Objects

All data values in a Python program are represented by objects and relationships among objects.

An object is an in-computer-memory representation of a value from a particular data type.

Objects

Objects are characterized by three qualities:

- Identity – Uniquely identifies the object.
- Type – Completely specifies its behavior.
- Value – Is the data-type value that it represents.

Objects

Each object stores one value. For example:

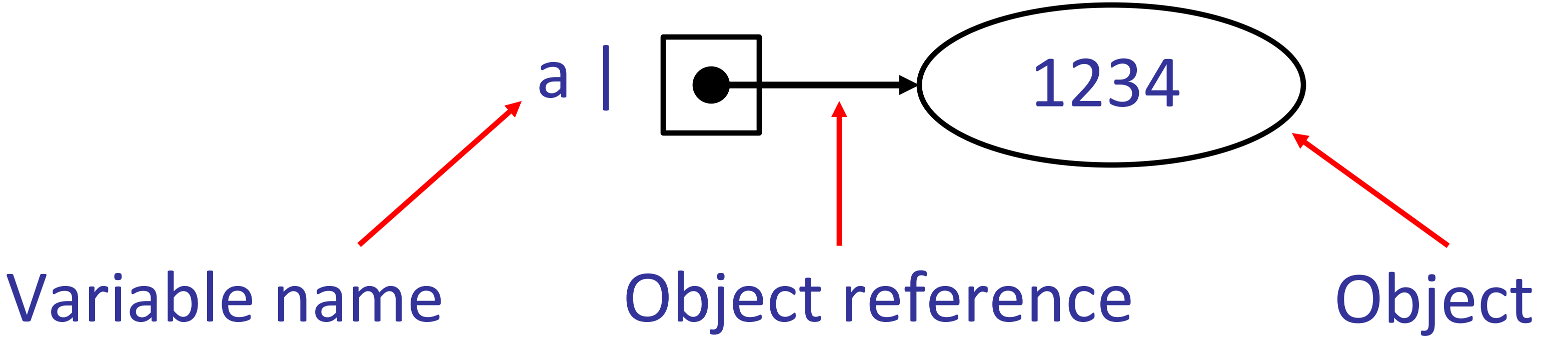
An object of type `int` can store the value 1234 or the value 99 or the value 1333.

OR

An object of type `str` can store the value ‘hello.’

Variables

A variable is a name for an object reference.



Expressions

An expression is a combination of literals, variables, and operators.

Operands (and expressions)

4 * (x - 3)

Operator

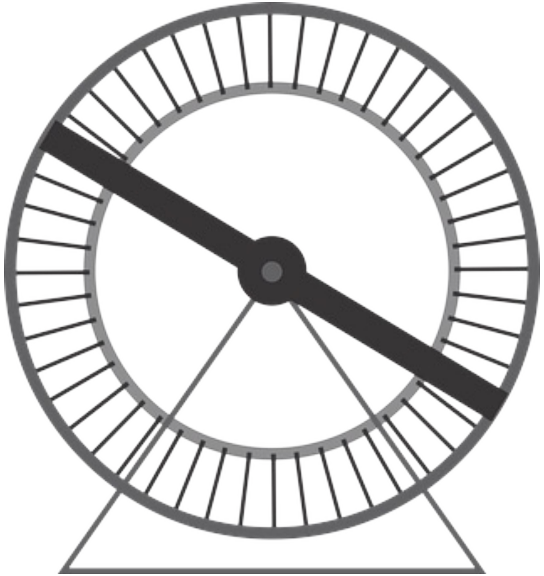
Assignment Statements

An assignment statement is a directive to Python to bind the variable on the left side of the = operator to the object produced by evaluating the expression on the right side.

Example: `c = a + b` means "associate the variable c with the sum of the values associated with the variables a and b"

Conditionals and Loops

- if-statements
- While-loops
- For-loops
- Nesting



If statements

Structure:

```
if (expression):
    statement
else:
    statement
```

If statements

Quick example

Maximum of x and y:

```
if x > y: maximum = x
else:     maximum = y
```

While-loops

Structure:

```
while (expression):
    statement
```

While-loops

Quick example

```
Import stdio
stdio.writeln('1st Hello')
stdio.writeln('2nd Hello')
stdio.writeln('3rd Hello')

i = 4
while i <= 10:
  stdio.writeln(str(i) + 'th Hello')
  i = i + 1
```

While-loops

Output:

```
1st Hello
2nd Hello
3rd Hello
4th Hello
5th Hello
6th Hello
7th Hello
8th Hello
9th Hello
10th Hello
```

For-loop

Structure:

```
for (variable) in range(expression):
    statement
```

For-loop

Quick example (**For-Loop version!**)

```
Import stdio
stdio.writeln('1st Hello')
stdio.writeln('2nd Hello')
stdio.writeln('3rd Hello')

for i in range(4, 11):
 stdio.writeln(str(i) + 'th Hello')
```

For-loop

Output:

```
1st Hello
2nd Hello
3rd Hello
4th Hello
5th Hello
6th Hello
7th Hello
8th Hello
9th Hello
10th Hello
```

For-loop

Quick example

total += i <=> total = total + i

Write a sum: 1 + 2 + … + n

```
total = 0
for i in range(1, n+1):
    total += i
stdio.writeln(total)
```

0 = 0 + 1 == 1
1 = 1 + 2 == 3
3 = 3 + 3 == 6
6 = 6 + 4 == 10
10 = 10 + 5 == 15
15 = 15 + 6 == 21
.
.
.
total = total + n

Nesting

Structure:

```
if (expression):
    statement
else:
    if (expression):
        statement
    else:
        if (expression):
            statement
        else:
            statement
```

Nesting

Example:

```
if income < 0.0:
    rate = 0.00
else:
    if income < 8925:
        rate = 0.10
    else:
        if income < 36250:
            rate = 0.15
```

Arrays

Arrays

A data structure is a way to organize data that we wish to process with a computer program.

A 1-dimensional array is a data structure that stores a sequence of (references to) objects.

We refer to the objects in an array as elements.

Arrays

The method that we use to refer to elements in an array is *numbering* and then *indexing* them.

If we have n elements in the sequence, we think of them as being numbered from 0 to n - 1.

Then, we can unambiguously specify one of them by referring to the ith element for any integer i in this range.

Arrays

A *two-dimensional array* is an array of (references to) one-dimensional arrays.

Whereas the elements of a one-dimensional array are indexed by a single integer, the elements of a two-dimensional array are indexed by a pair of integers:

- the first specifying a row, and the second specifying a column.

Arrays

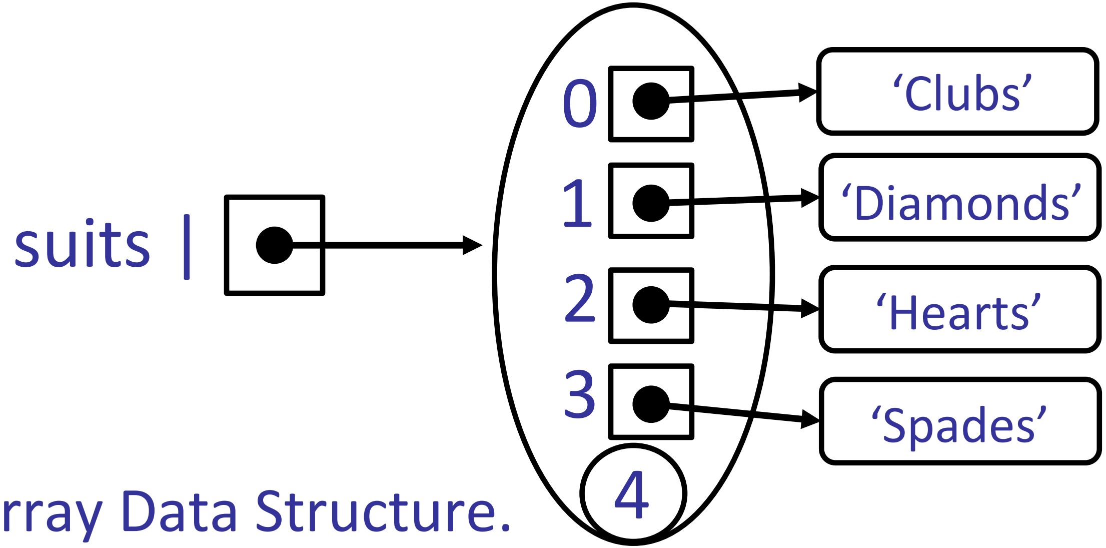
Example:

SUITS = [‘Clubs’, ‘Diamonds’, ‘Hearts’, ‘Spades’]
x = [0.30, 0.60, 0.10]
y = [0.50, 0.10, 0.40]

This creates an array SUITS[ ] with four strings, and creates arrays x[ ] and y[ ], each with three floats.

Arrays

A model for what arrays look like in the computer's memory.



This is the Array Data Structure.

Arrays

Mutability

An object is *mutable* if its value can change. Arrays are mutable objects because we can change their elements.

For example, if we create an array with the code x = [.30, .60, .10], then the assignment statement x[1] = .99 changes it to the array [.30, .99, .10].

Arrays

Mutability

The following code reverses the order of the elements in an array a[ ]:

```
n = len(a)
for i in range(n // 2):
    temp = a[ i ]
    a[ i ] = a[n-1-i]
    a[n-1-i] = temp
```

Arrays

Iteration

The following code iterates over all elements of an array to compute the average of the floats that it contains:

```
total = 0.0
for i in range(len(a)):
    total += a[ i ]
average = total / len(a)
```

Arrays

Iteration

Python also supports iterating over the elements in an array without referring to the indices explicitly. To do so, put the array name after the in keyword in a for statement, as follows:

```
total = 0.0
for v in a:
    total += v
average = total / len(a)
```

Arrays

Now is worthwhile to examine two fundamental array-processing operations in more detail.

- Array Aliases
- Array Copies

Arrays

Aliasing

If x[ ] and y[ ] are arrays, the statement x = y causes x and y to reference the same array.

It is natural to think of x and y as references to two independent arrays.

Arrays

Example:

```
x = [.30, .60, .10]
y = x
x[1] = .99
```

Therefore, y[1] is also .99, even though the code does not directly refer to y[1].

Arrays

Aliasing:

This situation — whenever two variables refer to the same object — is known as *aliasing.*

Arrays

A fundamental operation that is found in nearly every array-processing program is to create an array of *n* elements, each initialized to a given value:

```
a = [ ]
for i in range(n):
    a += [0.0]
```

Arrays

Example: **Representing playing cards.**

Suppose that we want to compose programs that process playing cards. We might start with the following code:

```
SUITS = ['Clubs', 'Diamonds', 'Hearts', 'Spades’]
RANKS = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']
```

Arrays

Example: **Representing playing cards.**

A more typical situation is when we compute the values to be stored in an array. For example, we might use the following code to initialize an array of length 52 that represents a deck of playing cards, using the two arrays just defined:

```
deck = [ ]
    for rank in RANKS:
        for suit in SUITS:
            card = rank + ' of ' + suit
            deck += [card]
```

Arrays

Example: **Exchange**

Frequently, we wish to exchange two elements in an array. Continuing our example with playing cards, the following code exchanges the cards at indices i and j:

```
temp = deck[ i ]
deck[ i ] = deck[ j ]
deck[ j ] = temp
```

Here, the variable temp means temporary.

Arrays

Example: **Shuffle**

The following code shuffles our deck of cards:

```
n = len(deck)
for i in range(n):
    r = random.randrange(i, n)
    temp = deck[ r ]
    deck[ r ] = deck[ i ]
    deck[ i ] = temp
```

Proceeding from left to right, we pick a random card from deck[i] through deck[n-1] (each card equally likely) and exchange it with deck[i].

Arrays

Two-Dimensional Arrays

In many applications, a convenient way to store information is to use a table of numbers organized in a rectangular table and refer to rows and columns in the table.

The mathematical abstraction corresponding to such tables is a *matrix*; the corresponding data structure is a *two-dimensional array*.

Arrays

Two-Dimensional Arrays

The simplest way to create a two-dimensional array is to place comma-separated one-dimensional arrays between matching square brackets.

Arrays

Two-Dimensional Arrays

For example, this matrix of integers having two rows and three columns:

$$\begin{pmatrix} 18 & 19 & 20 \\ 21 & 22 & 23 \end{pmatrix}$$

We call such an array a *2-by-3* array.

could be represented in Python using this array of arrays:

```
a = [[18, 19, 20], [21, 22, 23]]
```

Arrays

Two-Dimensional Arrays

More generally, Python represents an *m*-by-*n* array as an array that contains *m* objects, each of which is an array that contains *n* objects.

For example, this Python code creates an *m*-by-*n* array a[ ][ ] of floats, with all elements initialized to 0.0:

```
a = [ ]
for i in range(m):
    row = [0.0] * n
    a += [row]
```

Arrays

Two-Dimensional Arrays

Indexing:

When a[ ][ ] is a two-dimensional array, the syntax a[i] denotes a reference to its ith row.

The syntax a[i][j] refers to the object at row i and column j.

To access each of the elements in a two-dimensional array, we use two nested for loops.

Arrays

Two-Dimensional Arrays

For example, this code writes each object of the *m*-by-*n* array a[ ][ ], one row per line.

```
for i in range(m):
    for j in range(n):
        stdio.write(a[ i ][ j ])
        stdio.write(' ‘)
    stdio.writeln()
```

Arrays

Two-Dimensional Arrays

This code achieves the same effect without using indices:

```
for row in a:
    for v in row:
        stdio.write(v)
        stdio.write(' ‘)
    stdio.writeln()
```

Functions

Functions support a key concept that will pervade your approach to programming from this point forward: *Whenever you can clearly separate tasks within a computation, you should do so*.

You can define a function using a def statement that specifies the function signature, followed by a sequence of statements that constitute the function.

Functions

Structure:

```
def FunctionName (argument):
        statement
        statement
        .
        .
        statement
```

Functions

<u>Example:</u>

```
def     harmonic     ( n ) :
        total = 0.0
        for i in range (1, n+1):
                total += 1.0 / i
        return total
```

Python

parameter variable

signature

function name

function body

local variable

```
def harmonic (n) :
    total = 0.0
    for i in range (1, n+1):
        total += 1.0 / i
    return total
```

return statement

return value

Functions

The first line of a function definition, known as its *signature*, gives a name to the function and to each parameter variable.

Functions

The signature consists of the keyword def; the function name; a sequence of zero or more parameter variable names separated by commas and enclosed in parentheses; and a colon.

Functions

The indented statements following the signature define the function body. The function body can consist of the kinds of statements that we discussed in Chapter 1.

Functions

It also can contain a return statement, which transfers control back to the point where the function was called and returns the result of the computation or return value.

Functions

The body may also define *local variables*, which are variables that are available only inside the function in which they are defined.

Functions

Scope

The *scope* of a variable is the set of statements that can refer to that variable directly. The scope of a function's local and parameter variables is limited to that function; the scope of a variable defined in global code — known as a global variable — is limited to the .py file containing that variable.

Functions

Scope

Therefore, global code cannot refer to either a function's local or parameter variables. Nor can one function refer to either the local or parameter variables that are defined in another function.

Python

Functions

Program outputs the harmonic numbers

This code should not refer to arg or value

```
import stdio
import sys
```

Scope of n and total

```
# Return the nth harmonic number.
def harmonic(n):
    total = 0.0
    for i in range(1, n+1):
        total += 1.0 / float(i)
    return total
```

scope of i

Two different variables

```
# Write to standard output the harmonic numbers specified as command-
line arguments.
```

```
for j in range(1, len(sys.argv)):
    arg = int(sys.argv[j])
    value = harmonic(arg)
    stdio.writeln(value)
```

scope of i

scope of arg and value

This code cannot refer to n or total

Scope of local and parameter variables.

Functions

Example: Primality Test

```
def    isPrime(n) :
       if n < 2: return False
       i = 2
       while i * i <= n:
             if n % i == 0: return False
             i += 1
       return True
```

Modules and Clients

Each program that we've composed so far consists of Python code that can reside in a single .py file.

For large programs, keeping all the code in a single file is restrictive and unnecessary.

Fortunately, it is easy in Python to call a function that is defined in another file.

Modules and Clients

We distinguish two types of Python programs:

- A *module* contains functions that are available for use by other programs.

- A *client* is a program that makes use of a function in a module.

Modules and Clients

Let’s consider a program that computes the Gaussian distribution functions.

This program will be named gaussian.py

We will deem this file the module.

Modules and Clients

Now let's consider another program that will use the functions in the gaussian.py file.

This program will be named gaussiantable.py

We will deem this file the client.

Modules and Clients

To summarize, the client named gaussiantable.py will use the the functions in module gaussian.py.

Modules and Clients

But how is this process done?

Python

gaussian.py | Module

```
import sys
import stdio
import math
#-----------------------------------------------------------------------
# Return the value of the Gaussian probability function with mean mu
# and standard deviation sigma at the given x value.
def pdf(x, mu=0.0, sigma=1.0):
    x = float(x - mu) / sigma
    return math.exp(-x*x/2.0) / math.sqrt(2.0*math.pi) / sigma
#-----------------------------------------------------------------------
# Return the value of the cumulative Gaussian distribution function
# with mean mu and standard deviation sigma at the given z value.
def cdf(z, mu=0.0, sigma=1.0):
    z = float(z - mu) / sigma
    if z < -8.0: return 0.0
    if z > +8.0: return 1.0
    total = 0.0
    term = z
    i = 3
    while total != total + term:
        total += term term *= z * z / i
        i += 2
    return 0.5 + total * pdf(z)
#-----------------------------------------------------------------------
# Accept floats z, mu, and sigma as command-line arguments. Use them
# to test the phi() and Phi() functions. Write the
# results to standard output.
def main():
    z = float(sys.argv[1])
    mu = float(sys.argv[2])
    sigma = float(sys.argv[3])
    stdio.writeln(cdf(z, mu, sigma))
if __name__ == '__main__’:
    main()
```

gaussiantable.py | Client

Imported the gaussian.py file.

```
import sys
import stdio
import gaussian
#-----------------------------------------------------------------------
# Accept a mean and standard deviation as command-line arguments.
# Write to standard output a table of the percentage of students
# scoring below certain scores on the SAT, assuming the test scores
# obey a Gaussian distribution with the given mean and standard
# deviation.
mu = float(sys.argv[1])
sigma = float(sys.argv[2])

for score in range(400, 1600+1, 100):
    percent = gaussian.cdf(score, mu, sigma)
    stdio.writef('%4d %.4f\n', score, percent)
```

Modules and Clients

Private Functions

Sometimes we wish to define in a module a helper function that is not intended to be called directly by clients.

We refer to such a function as a *private function*.

Modules and Clients

Private Functions

By convention, Python programmers use an underscore as the first character in the name of a private function.

Modules and Clients

Private Functions

For example:

```
def _private():
    print(“This function is private.”)

def public():
    print(“This function is public.”)
```

Recursion

The idea of calling one function from another immediately suggests the possibility of a function calling *itself*.

The function-call mechanism in Python supports this possibility, which is known as *recursion*.

Recursion

Recursion is a powerful general-purpose programming technique, and is the key to numerous critically important computational applications, ranging from combinatorial search and sorting methods methods that provide basic support for information processing.

Recursion

Your First Recursive Program

The "HelloWorld" program for recursion is to implement the *factorial* function, which is defined for positive integers `n` by the equation:

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$$

Recursion

Your First Recursive Program

n! is easy to compute with a for loop, but an even easier method is to use the following recursive function:

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)
```

Recursion

Our `factorial()` implementation exhibits the two main components that are required for every recursive function.

- The *base case* returns a value without making any subsequent recursive calls. It does this for one or more special input values for which the function can be evaluated without recursion. For `factorial()`, the base case is `n = 1`.

- The *reduction step* is the central part of a recursive function. It relates the function at one (or more) inputs to the function evaluated at one (or more) other inputs. Furthermore, the sequence of parameter values must *converge* to the base case. For `factorial()`, the reduction step is `n * factorial(n-1)` and `n` decreases by one for each call, so the sequence of parameter values converges to the base case of `n = 1`.

Mathematical Induction

Recursive programming is directly related to *mathematical induction*, a technique for proving facts about discrete functions.

Proving that a statement involving an integer n is true for infinitely many values of n by mathematical induction involves two steps.

Mathematical Induction

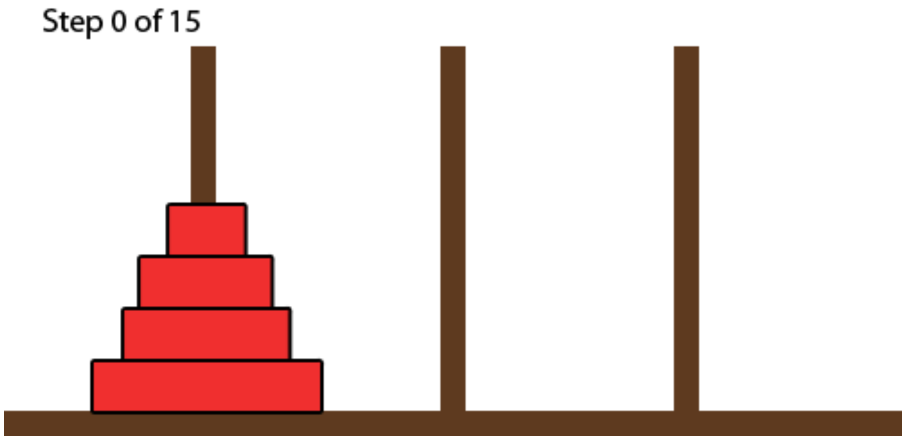
Those two steps are the following:

- The *base case* is to prove the statement true for some specific value or values of *n* (usually 0 or 1).

- The *induction step* is the central part of the proof. For example, we typically assume that a statement is true for all positive integers less than *n*, then use that fact to prove it true for *n*.

Towers of Hanoi

No discussion of recursion would be complete without the ancient *Towers of Hanoi* problem.



Towers of Hanoi

We have three poles and *n* discs that fit onto the poles.

The discs differ in size and are initially arranged on one of the poles, in order from largest (disc *n*) at the bottom to smallest (disc 1) at the top.

Towers of Hanoi

The task is to move the stack of discs to another pole, while obeying the following rules:

- Move only one disc at a time.
- Never place a disc on a smaller one.

Towers of Hanoi

Recursion provides the plan that we need, based on the following idea:

- First, we move the top *n*-1 discs to an empty pole.
- Then we move the largest disc to the other empty pole (where it does not interfere with the smaller ones).
- Lastly, we compete the job by moving the *n*-1 discs onto the largest disc

Python

Towers of Hanoi

```
import sys
import stdio
#---------------------------------------------------------------------
# Write to standard output instructions to move n Towers of Hanoi
# disks to the left (if parameter left is True) or to the right (if
# parameter left is False).
def moves(n, left):
    if n == 0:
        return
    moves(n-1, not left)
    if left:
        stdio.writeln(str(n) + ' left')
    else:
        stdio.writeln(str(n) + ' right')
    moves(n-1, not left)
```

More on Data Types

At first, our programs were confined to operations on numbers, booleans, and strings.

The reason is that the Python data types that we've encountered so far — int, float, bool, and str — manipulate numbers, booleans, and strings, using familiar operations.

Now, we begin to consider other data types.

Data Types

Methods

A *method* is a function associated with a specified object (and, by extension, with the type of that object).

That is, a method corresponds to a data-type operation.

Data Types

Methods

We can *call* (or *invoke*) a method by using a variable name, followed by the *dot operator* (.), followed by the method name, followed by a list of arguments delimited by commas and enclosed in parentheses.

Python

Data Types

Methods

As a simple example, Python's built-in int type has a method named bit_length(), so you can determine the number of bits in the binary representation of an int value as follows:

```
x = 3 ** 100
bits = x.bit_length()
stdio.writeln(bits)
```

Create an `int` object

Call on an `int` method

Module name

Call on an `stdio` function

Data Types

Methods

This code writes 159 to standard output, telling you that 3^{100} (a huge integer) has 159 bits when expressed in binary.

```
x = 3 ** 100
bits = x.bit_length()
stdio.writeln(bits)
```

Create an `int` object

Call on an `int` method

Module name

Call on an `stdio` function

Data Types

Difference between method and function

The key difference between a function and a method is that a method is associated with a specified object.

Data Types

Difference between method and function

	method	function
Sample call	x.bit_length()	stdio.writeln(bits)
Typically invoked with	variable name	module name
Parameters	Object reference and argument(s)	argument(s)
Primary Purpose	Manipulate object value	compute return value

User-Defined Data Types

As a running example of a user-defined data type, we will consider a data type `Charge` for charged particles.

User-Defined Data Types

In particular, we are interested in a two-dimensional model that uses *Coulomb's law*, which tells us that the electric potential at a point due to a given charged particle is represented by $V = kq/r$, where q is the charge value, r is the distance from the point to the charge, and $k = 8.99 \times 10^9$ N m^2/C^2 is a constant known as the electrostatic constant, or *Coulomb's constant.*

User-Defined Data Types

For consistency, we use SI (Systeme International d'Unites): in this formula, N designates newtons (force), m designates meters (distance), and C represent coulombs (electric charge).

User-Defined Data Types

When there are multiple charged particles, the electric potential at any point is the sum of the potentials due to each charge.

User-Defined Data Types

Application Programming Interface

We specify the behavior of the `Charge` data type by listing its operations in an API.

User-Defined Data Types

API for our user-defined Charge data type

operation	description
Charge(x0, y0, q0)	A new charge centered at (x0, y0) with charge value q0
c.potentialAt(x, y)	Electric potential of charge c at point (x, y)
str(c)	‘q0 at (x0, y0)’ (string representation of charge c)

User-Defined Data Types

Application Programming Interface

The first entry in the API, which has the same name as the data type, is known as a constructor.

Each call to the `Charge` constructor creates exactly one new `Charge` object.

User-Defined Data Types

Application Programming Interface

The other two entries define the data-type operations:

- The first is a method `potentialAt()`, which computes and returns the potential due to the charge at the given point (x, y).
- The second is the built-in function `str()`, which returns a string representation of the charged particle.

User-Defined Data Types

Creating Objects

To create an object from a user-defined data type, you call its constructor, which directs Python to create a new individual object.

User-Defined Data Types

Creating Objects

You call a constructor just as if it were a function, using the name of the data type, followed by the constructor's arguments enclosed in parentheses and separated by commas.

User-Defined Data Types

Creating Objects

Example:

Charge(x0, y0, q0) creates a new `Charge` object with position (x0, y0) and `charge` value q0 and returns a reference to the new object.

User-Defined Data Types

Calling a method

As was mentioned before, we typically use a variable name to identify the object to be associated with the method we intend to call.

For our example, the method call c1.potentialAt(.20, .50) returns a float that represents the potential at the query point (0.20, 0.50) due to the `Charge` object referenced by c1.

User-Defined Data Types

String Representation

In any data-type implementation, it is typically worthwhile to include an operation that converts an object's value to a string.

Python has a built-in function `str()` for this purpose, which you been using from the beginning to convert integers and floats to strings for output.

User-Defined Data Types

String Representation

Since our Charge API has a `str()` implementation, any client can call str() to get a string representation of a `Charge` object.

Example: the call `str(c1)` returns the string '21.3 at (0.51, 0.63)'.

Classes

In Python, we implement a data type as a *class*.

To define a class, we use the keyword class, followed by the class name, followed by a colon, and then a list of method definitions.

Classes

Our class defines a constructor, instance variables, and methods, which we will address in detail next.

Python

Charge program

class

```
import sys
import math
import stdio
#-----------------------------------------------------------------------
class Charge:
    # Construct self centered at (x, y) with charge q.
    def __init__(self, x0, y0, q0):
        self._rx = x0 # x value of the query point
        self._ry = y0 # y value of the query point
        self._q = q0 # Charge

    # Return the potential of self at (x, y).
    def potentialAt(self, x, y):
        COULOMB = 8.99e09
        dx = x - self._rx
        dy = y - self._ry
        r = math.sqrt(dx*dx + dy*dy)
        if r == 0.0: # Avoid division by 0
            if self._q >= 0.0:
                return float('inf')
            else:
                return float('-inf')
        return COULOMB * self._q / r

    # Return a string representation of self.
    def __str__(self):
        result = str(self._q) + ' at ('
        result += str(self._rx) + ', ' + str(self._ry) + ')'
        return result
```

Python

Charge program

```python
import sys
import math
import stdio
#-----------------------------------------------------------------------
class Charge:
    # Construct self centered at (x, y) with charge q.
    def __init__(self, x0, y0, q0):
        self._rx = x0 # x value of the query point
        self._ry = y0 # y value of the query point
        self._q = q0 # Charge

    # Return the potential of self at (x, y).
    def potentialAt(self, x, y):
        COULOMB = 8.99e09
        dx = x - self._rx
        dy = y - self._ry
        r = math.sqrt(dx*dx + dy*dy)
        if r == 0.0: # Avoid division by 0
            if self._q >= 0.0:
                return float('inf')
            else:
                return float('-inf')
        return COULOMB * self._q / r

    # Return a string representation of self.
    def __str__(self):
        result = str(self._q) + ' at ('
        result += str(self._rx) + ', ' + str(self._ry) + ')'
        return result
```

Class definition

Class name

Parameter variable

Define and initialize instance variables

constructor

Reference to invoking Object, or special parameter Variable.

methods

Ordinary method

Instance variables

Special method to support Str().

Constructor

A constructor creates an object of the specified type and returns a reference to that object.

Example: the code `c = Charge(x0, y0, q0)` returns a new Charge object, suitably initialized.

Designing Data Types

Developing APIs is a critical step in the development of any program.

Designing Data Types

Encapsulation

The process of separating clients from implementations by hiding information is known as *encapsulation*.

Details of the implementation are kept hidden from clients, and implementations have no way of knowing details of client code, which may even be created in the future.

Designing Data Types

Immutability

An object from a data type is *immutable* if its data-type value cannot change once created.

An immutable data type, such as a Python string, is one in which all objects of that type are immutable.

Designing Data Types

Tuples

Python's built-in tuple data type represents an *immutable* sequence of objects.

It is similar to the built-in list data type (which we use for arrays), except that once you create a tuple, you cannot change its items.

Designing Data Types

Polymorphism

Often, when we compose methods (or functions), we intend for them to work only with objects of specific types.

Sometimes, we want them to work with objects of different types.

A method (or function) that can take arguments with different types is said to be *polymorphic*.

Designing Data Types

Overloading

The ability to define a data type that provides its own definitions of operators is a form of polymorphism known as *operator overloading*.

In Python, you can overload almost every operator, including operators for arithmetic, comparisons, indexing, and slicing.

Designing Data Types

Hashing

We now consider a fundamental operation related to equality testing, known as *hashing*, that maps an object to an integer, known as a *hash code*.

Designing Data Types

Inheritance

Python provides language support for defining relationships among classes, known as *inheritance*.

Software developers use inheritance widely, so you will study it in detail if you take a course in software engineering.

When used properly, inheritance enables a form of code reuse known as *subclassing*.

Designing Data Types

Inheritance

The idea is to define a new class (*subclass*, or *derived class*) that inherits instance variables and methods from another class (*superclass*, or *base class*).

The subclass contains more methods than the superclass.

Prolog

Prologic!

Logic

What is Logic?

Logic is the science of reasoning

Logic

It’s about determining the validity of an argument.

Reasoning is a special mental activity called inferring, or making inferences.

To infer is to draw conclusions from premises (premises are data, information, or facts).

Logic

What is an argument?

An argument is a collection of statements, one of which is designated as the conclusion and the remainder of which are designated as the premises.

Logic

The premises of an argument are intended to support (or justify) the conclusion of the argument.

Logic

Many times, we make all kinds of statements.

A statement is a declarative sentence, which is to say a sentence that is capable of being true or false. This is also known as a propositional statement.

Logic

Examples of propositional statements:

- It is raining
- I am hungry
- 2 + 2 = 4
- God exists

Logic

We can make conclusions based on what premises tell us.

There were 20 persons originally	(premise)
There are 19 persons currently	(premise)
Therefore, someone is missing	(conclusion)

Logic

In an argument, the premises are intended to support (or justify) the conclusion.

Deductive Logic Versus Inductive Logic

Two examples:

a) There is smoke;
Therefore, there is fire.

b) There were 20 people originally;
There are 19 persons currently;
Therefore, someone is missing.

Deductive Logic Versus Inductive Logic

The example of a) is what we call inductive logic. Why?

We know that the existence of smoke does not guarantee (ensure) the existence of fire; it only makes the existence of fire likely or probable. Yet, we may be wrong in asserting that there is fire due solely to smoke, and so this argument is fallible.

Deductive Logic Versus Inductive Logic

Inductive logic investigates the process of drawing probable (likely, plausible) though fallible conclusions from premises.

Deductive Logic Versus Inductive Logic

The example of b) is what we call deductive logic. Why?

If the premises are in fact true, then the conclusion is certainly also true.

Said differently, the truth of the premises necessitates the truth of the conclusion.

Deductive Logic Versus Inductive Logic

Typically, a course in logic is about deductive logic.

Fundamental Facts

An argument is factually correct if and only if all of its premises are true.

An argument is valid if and only if its conclusion follows from its premises.

An argument is sound if and only if it is both factually correct and valid.

General Forms

All X are Y

Some X are Y

No X are Y

Some X are not Y

P’s and Q’s

We can represent sentences using letters, like p and q.

P = “I went to the store.”

Q = “I bought candy.”

P’s and Q’s

We can negate the sentences (or discuss their opposites) like so:

P = “I went to the store.”

¬P = “I did not go to the store.”

Q = “I bought candy.”

¬Q = “I did not buy candy.”

P's and Q's

We can also combine them using what's called conjunction.

P ^ Q means "P and Q."

P = "I went to the store."

Q = "I bought candy."

P ^ Q = "I went to the store AND I bought candy."

P’s and Q’s

We may use another connective called disjunctions.

P v Q means “P or Q.”

P = “I went to the store.”

Q = “I bought candy.”

P v Q = “I went to the store OR I bought candy.”

P’s and Q’s

Conditionals use a right arrow, like this $\rightarrow$.

$P \rightarrow Q$ means “If P, then Q.”

P = “I went to the store.”

Q = “I bought candy.”

$P \rightarrow Q$ = “If I went to the store, then I bought candy.”

P's and Q's

Biconditionals use a line with both arrows at the end, like this ↔.

P ↔ Q means "If P, then Q" **and** "If Q, then P." Therefore, P = Q.

P = "I went to the store."

Q = "I bought candy."

P ↔ Q = "If I went to the store, then I bought candy," and "If I bought candy, then I went to the store."

P’s and Q’s

P and Q can be true. BUT, they can also be false.

Saying P = “I went to the store” could be a true or false statement.

If P is true, then we give it a truth value denoted by **T**.

If P is false, then we give it a truth value denoted by **F**.

Truth Tables

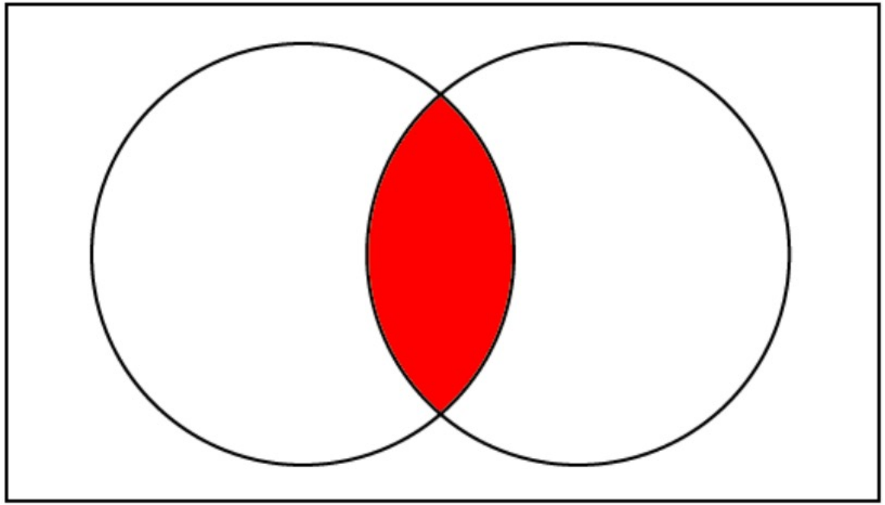
A truth table helps us check for the validity of an argument.

p	q
T	T
T	F
F	T
F	F

Truth Tables

What are the truth values of ‘P AND Q’ or ‘P ^ Q’

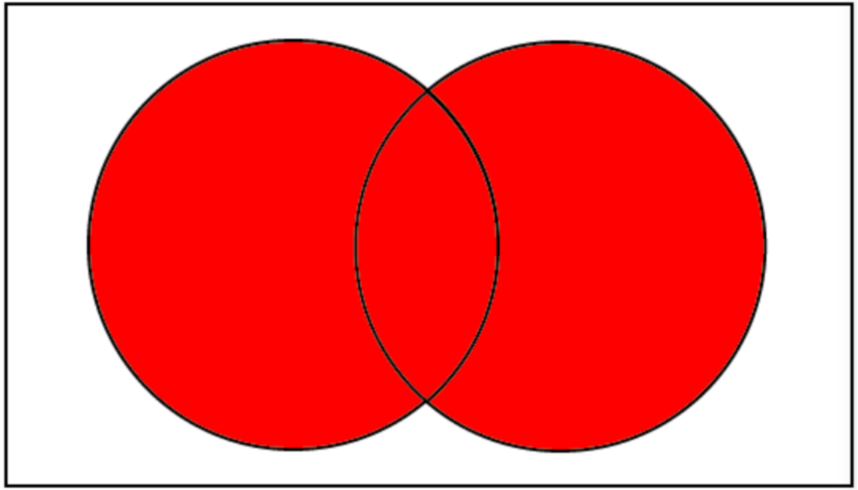
p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F



Truth Tables

What are the truth values of ‘P OR Q’ or ‘P v Q’

p	q	p∨q
T	T	T
T	F	T
F	T	T
F	F	F



Truth Tables

What are the truth values of ‘P → Q’ or ‘If P, then Q’ or ‘P⊂Q’

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Prolog

P = I went to the store
Q = I bought candy

Truth Tables

What are the truth values of ‘P ↔ Q’ or ‘P = Q’

If I went to the store,
Then I bought candy.

AND

If I bought candy,
Then I went to the store.

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Propositional Logic

The simplest, and most abstract logic we can study is called propositional logic.

Propositional Logic

Definition: A proposition is a statement that can be either true or false; it must be one or the other, and it cannot be both.

The following are propositions:

– The reactor is on.

–The wing-flaps are up.

– John Major is prime minister.

Propositional Logic

The following are not propositions:

– Are you going out somewhere?

– 2+3

Propositional Logic

We now define atomic propositions. Intuitively, these are the set of smallest propositions.

Propositional Logic

Definition: An atomic proposition is one whose truth or falsity does not depend on the truth or falsity of any other proposition.

Propositional Logic

So all the propositions we went over are atomic.

Propositional Logic

Now, rather than write out propositions in full, we will abbreviate them by using propositional variables.

Propositional Logic

It is standard practice to use the lower-case roman letters p, q, r, . . . to stand for propositions.

First-Order Logic

Why not propositional logic?

Consider the following statements:

– All monitors are ready.

– X12 is a monitor.

First-Order Logic

We saw in an earlier slide, these statements are propositions: their meaning is either true or false.

First-Order Logic

Propositional logic is the most abstract level at which we can study logic.

First-Order Logic

As we shall say, it is too coarse grained to allow us to represent and reason about the kind of statement we need to write in formal specification.

First-Order Logic

We shall now introduce a generalization of propositional logic called first-order logic (FOL). This new logic affords us much greater expressive power.

First-Order Logic

First, we shall look at how the language of first-order logic is put together.

The basic components of FOL are called terms.

First-Order Logic

Essentially, a term is an object that denotes some object other than true or false.

First-Order Logic

The simplest kind of term is a constant.

Example: Any number, like 8, can be considered a constant.

The second simplest kind of term is a variable.

Example: The letter x can be considered a variable.

A more complex class of terms — functions.

Example: plus(2,3) can be used to represent '2 + 3.'

First-Order Logic

In addition to having terms, FOL has relational operators, which capture relationships between objects.

First-Order Logic

The language of FOL contains a stock of predicate symbols.

These symbols stand for relationships between objects.

First-Order Logic

Again, each predicate symbol has an associated arity. . .

. . . and each argument has a type.

First-Order Logic

Definition: Let *P* be a predicate symbol of arity n ∈ $\mathbb{N}$, which takes arguments of types $T_1, \ldots, T_n$. Then if $\tau_1, \ldots, \tau_n$ are terms of type $T_1, \ldots, T_n$, respectively, then $P(\tau_1, \ldots, \tau_n)$ is a predicate, which will either be true or false under some interpretation.

First-Order Logic

EXAMPLE. Let gt be a predicate symbol with the intended interpretation ‘greater than’. It takes two arguments, each of which is a natural number.

Then:

– gt(4, 3) is a predicate, which evaluates to true;

– gt(3, 4) is a predicate, which evaluates to false.

but gt(−1, 2) isn’t a predicate.

First-Order Logic

So a predicate just expresses a relationship between some values.

First-Order Logic

What happens if a predicate contains variables: can we tell if it is true or false?

First-Order Logic

Not usually; we need to know an interpretation for the variables.

First-Order Logic

A predicate that contains no variables is a proposition.

SWI-Prolog

SWI-Prolog is a free Prolog environment. We will use it write code in Prolog.

Prolog

Syntax

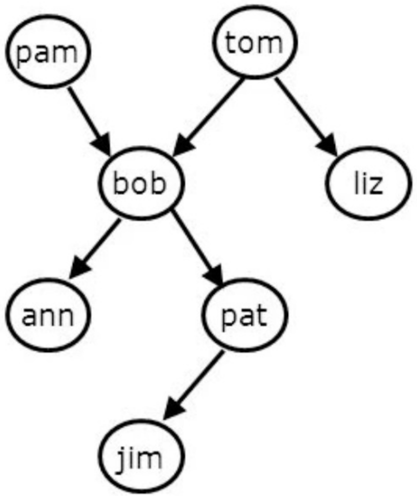
Facts:	Prolog:
Johnny is nice	nice(Johnny).
The dog is brown	brown(dog).
Suzie likes Bobby	likes(Suzie, Bobby).

Overview of prolog

AND	,
IF	:-
OR	;
NOT	not

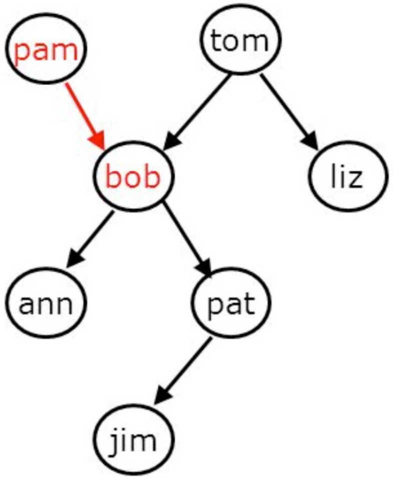
Defining Relations by facts

Here's a family tree:



Defining Relations by facts

Here's a family tree:

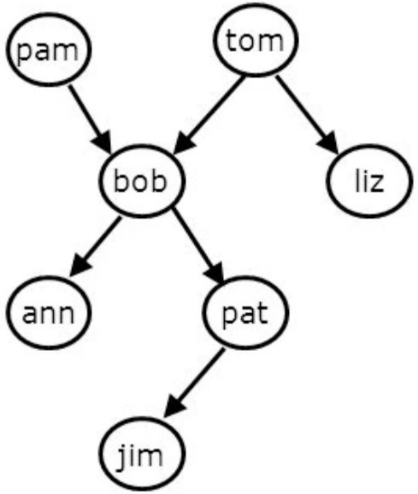


The tree defined by a program in prolog.

```
parent( pam, bob).
  % Pam is a parent of Bob
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

Defining Relations by facts

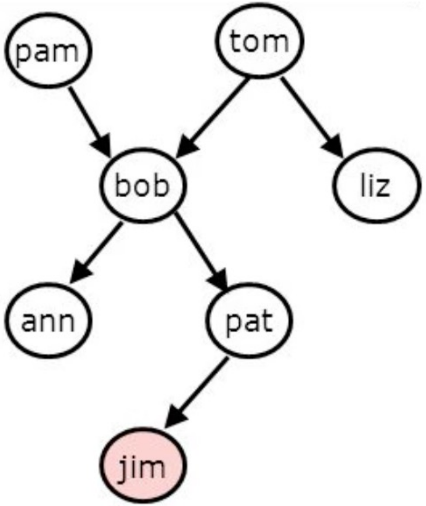
Family tree



- Is Bob a parent of Pat?
  - ?- parent(bob, pat).
  - ?- parent(liz, pat).
  - ?- parent(tom, ben).
- Who is Liz's parent?
  - ?- parent(X, liz).
- Who are Bob's children?
  - ?- parent(bob, X).

Defining Relations by facts

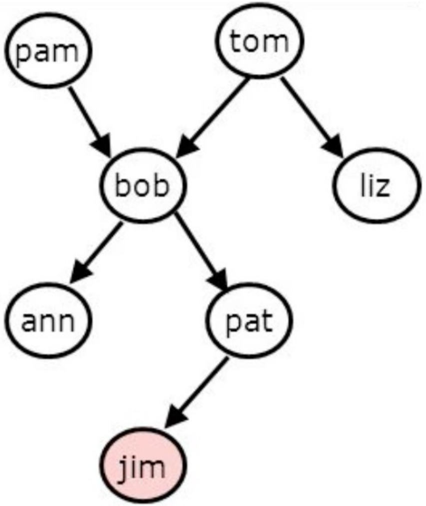
Family tree



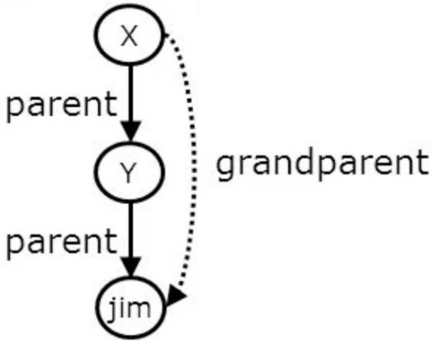
- Who is a parent of whom?
  - Find X and Y such that X is a parent of Y.
  - ?- parent(X, Y).
- Who is a grandparent of Jim?
  - ?- parent(Y, jim),
    parent(X, Y).

Defining Relations by facts

Family tree

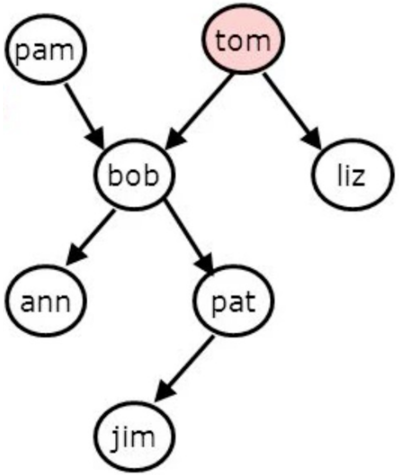


- Who is a grandparent of Jim?
  - ?- parent(Y, jim),
    parent(X, Y).



Defining Relations by facts

Family tree

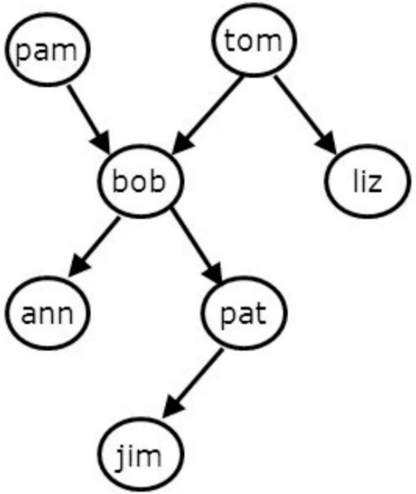


- Who are Tom's grandchildren?
  - ?- parent(tom, X),
    parent(X, Y).

- Do Ann and Pat have a common parent?
  - ?- parent(X, ann),
    parent(X, pat).

Defining Relations by facts

Family tree

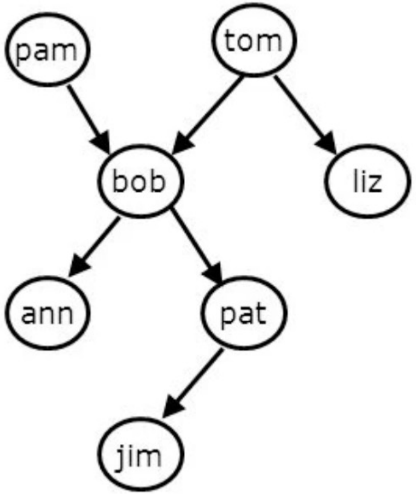


Facts:

- female(pam). % Pam is female
- female(liz).
- female(ann).
- female(pat).
- male(tom). % Tom is male
- male(bob).
- male(jim).

Defining Relations by rules

Family tree



Rules have:

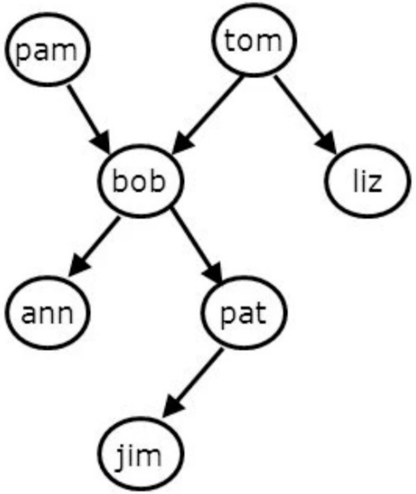
- A condition part (body)
  - the right-hand side of the rule
- A conclusion part (head)
  - the left-hand side of the rule

Rule: **offspring(Y, X) :- parent(X, Y).**

- For all X and Y,
  Y is an offspring of X if
  X is a parent of Y.

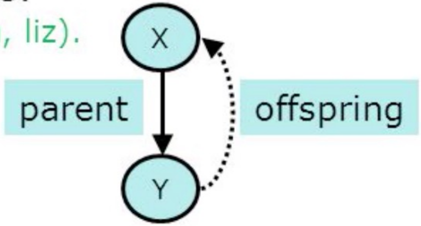
Defining Relations by rules

Family tree



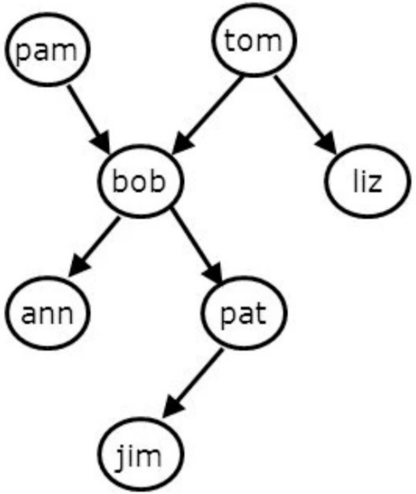
Example:

- **offspring(Y, X) :- parent(X, Y).**
- The rule is general in the sense that it is applicable to any objects X and Y.
- A special case of the general rule:
  - offspring(liz, tom) :- parent(tom, liz).
- ?- offspring(liz, tom).
- ?- offspring(X, Y).



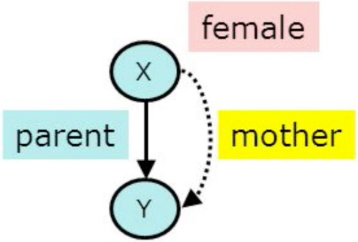
Defining Relations by rules

Family tree



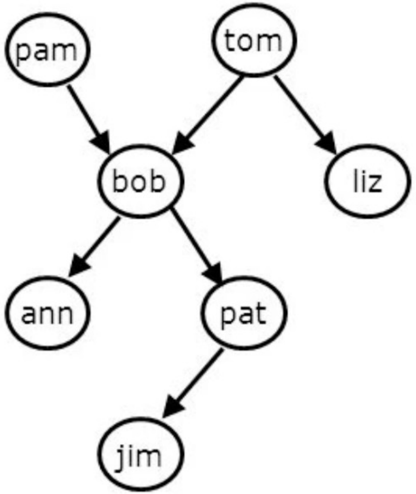
Define the “mother” relation:

- **mother(X, Y) :- parent(X, Y), female(X).**
- For all X and Y,
  X is the mother of Y if
  X is a parent of Y and
  X is a female.



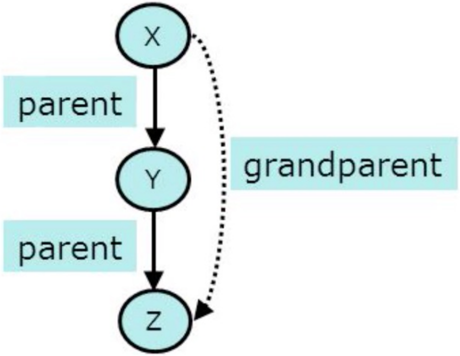
Defining Relations by rules

Family tree



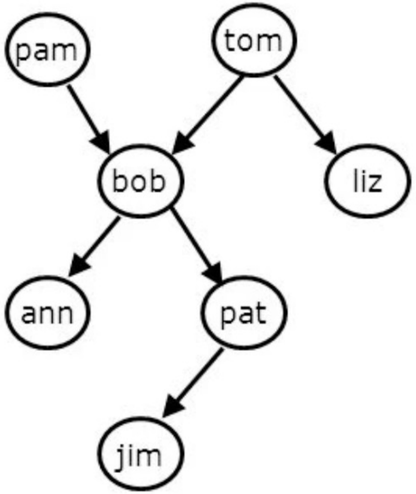
Define the "grandparent" relation:

- **grandparent(X, Z) :-**
  parent(X, Y), parent(Y, Z).



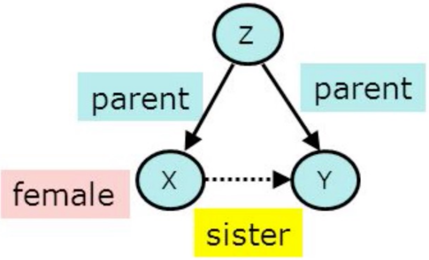
Defining Relations by rules

Family tree



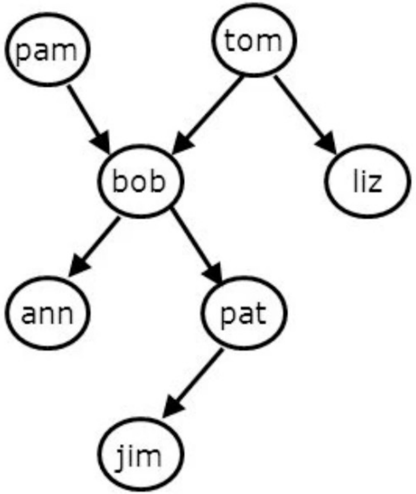
Define the “sister” relation:

- **sister(X, Y) :-**
  parent(Z, X), parent(Z, Y), female(X).
- For any X and Y,
  X is a sister of Y if
  (1) both X and Y have the same parent, and
  (2) X is female.
- ?- sister(ann, pat).
- ?- sister(X, pat).
- ?- sister(pat, pat).
  - Pat is a sister to herself?!



Defining Relations by rules

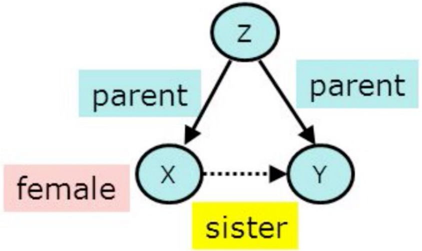
Family tree



To correct the "sister" relation:

- **sister(X, Y) :-**
  parent(Z, X), parent(Z, Y), female(X),
  different(X, Y).
- different (X, Y) is satisfied if and only if X and Y are not equal. (Please try to define this function)

different(x,y) :- not(x,y), not(y,x).



Defining Relations by rules

Example:

- Axioms: All men are fallible
  Socrates is a man.
- Theorem: Socrates is fallible.
- For all X, if X is a man then X is fallible.

```
fallible( X) :- man( X).
man( socrates).
```

  - ?- fallible(socrates).

Prolog Examples

Facts:

```
likes(ryan, brittney).
likes(brittney, ryan).
likes(dan, mary).
```

Rule:

```
dating(X, Y) :-
likes(X, Y),
likes(Y, X).
```

Prolog Examples

```
likes(ryan, brittney).
likes(brittney, ryan).
likes(dan, mary).

dating(X, Y) :-
likes(X, Y),
likes(Y, X).
```

?- likes(dan, mary).
 output: true

?- likes(dan, brittney).
 output: false

?- dating(ryan, brittney).
 output: true

Prolog Examples

```
/*weather(City, Season, Temp).*/

weather(phoenix, summer, hot).
weather(la, summer, warm).
weather(phoenix, winter, warm).
```

Prolog Examples

Now input the following into the terminal:

```
?- Weather(City, summer, hot), weather(City, winter, warm).
```

Output: City = phoenix.

Prolog Examples

Now input the following into the terminal:

```
?- warmer_than(phoenix, la)
Phoenix is warmer_than la
```

Output: true.

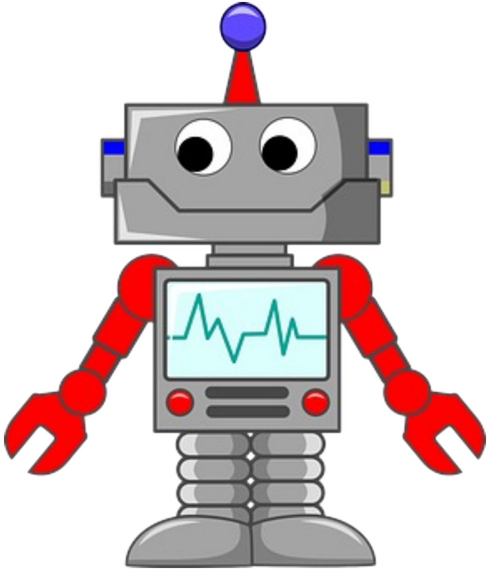
Prolog Examples

Now input the following into the terminal:

```
?- warmer_than(la, phoenix)
```

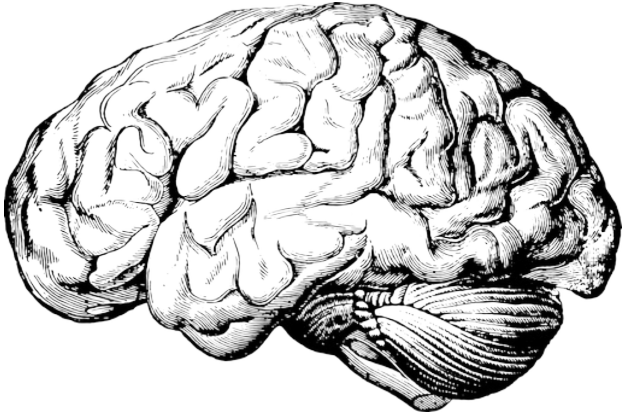
Output: false.

Psychology



Psychology

Psychology is the science of behavior and mental processes.



Psychology

Behavior is anything an organism does—any action we can observe and record.

Yelling, smiling, blinking, sweating, talking, and questionnaire marking are all observable behaviors.

Psychology

Mental processes are the internal, subjective experiences we infer from behavior –sensations, perceptions, dreams, thoughts, beliefs, and feelings.

Consciousness

Consciousness is our awareness of ourselves and our environment.

Selective Attention

The focusing of conscious awareness on a particular stimulus.

Selective Attention

We may think we can fully attend to a conversation or a class lecture while checking and returning text messages.

Selective Attention

Actually, our consciousness focuses on but one thing at a time.

Selective Attention

By one estimate, our five senses take in 11,000,000 bits of information per second, of which we consciously process about 40.

Selective Attention

Yet our mind's unconscious track intuitively makes great use of the other 10,999,960 bits.

Dual Processing: The Two-Track Mind

This is the principle that information is often simultaneously processed on separate conscious and unconscious tracks.

Perception

The process of organizing and interpreting sensory information, enabling us to recognize meaningful objects and events.

Learning

The process of acquiring through experiencing new and relatively enduring information or behaviors.

Associative Learning

Learning that certain events occur together. The events may be two stimuli (as in classical conditioning) or a response and its consequences (as an operant conditioning).

Stimulus

Any event or situation that evokes a response.

Cognitive Learning

The acquisition of mental information, whether by observing events, by watching others, or through language.

Memory

Memory is the learning that persists over time; it is the information that has been acquired and stored and can be retrieved.

Memory

Evidence that learning persists includes these three measures of retention:

- Recall
- Recognition
- Relearning

Memory

Recall is a measure of memory in which the person must retrieve information learned earlier, as on a fill-in-the-blank test.

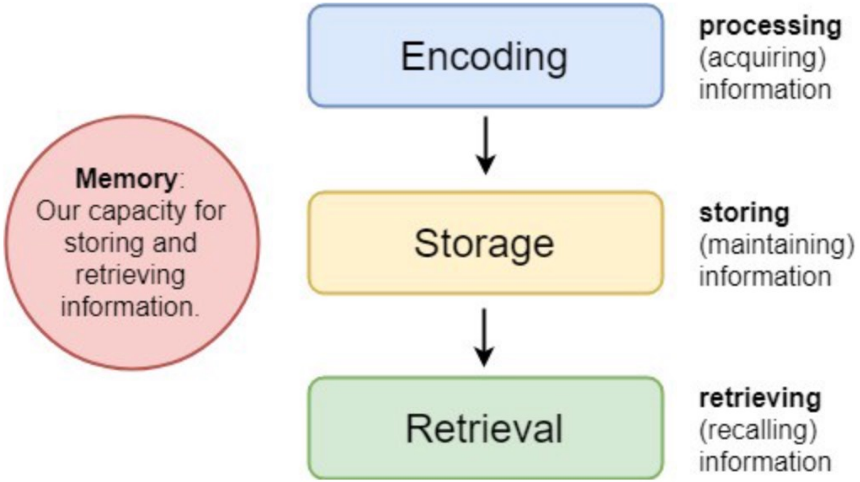
Memory

Recognition is a measure of memory in which the person identifies items previously learned, as on a multiple-choice test.

Memory

Relearning is a measure of memory that assess the amount of time saved when learning material again.

Memory


Memory:
Our capacity for storing and retrieving information.
Encoding
processing (acquiring) information
Storage
storing (maintaining) information
Retrieval
retrieving (recalling) information

Thinking

Cognition is all the mental activities associated with thinking, knowing, remembering and communicating.

Thinking

Heuristics are simpler thinking strategies. A simple thinking strategy that often allows us to make judgements and solve problems efficiently; usually speedier but also more error-prone than algorithms.

Thinking

Insight is a sudden realization of a problem's solution; contrasts with strategy-based solutions.

Thinking

Intuition is an effortless, immediate, automatic feeling or thought, as contrasted with explicit, conscious reasoning.

Language

Language is our spoken, written, or signed words and the ways we combine them to communicate meaning.

Intelligence

The mental potential to learn from experience, solve problems, and use knowledge to adapt to new situations.

Sternberg’s Three Intelligences

Analytical intelligence is school smarts; traditional academic problem solving.

Creative intelligence is the ability to react adaptively to new situations and generate novel ideas.

Practical intelligence is street smarts; skill at handling everyday tasks, which may be ill defined, with multiple solutions.

IQ

IQ stands for Intelligence Quotient, and was defined originally as the ratio of mental age (ma) to chronological age (ca) multiplied by 100

(thus, IQ = ma/ca *100).

IQ

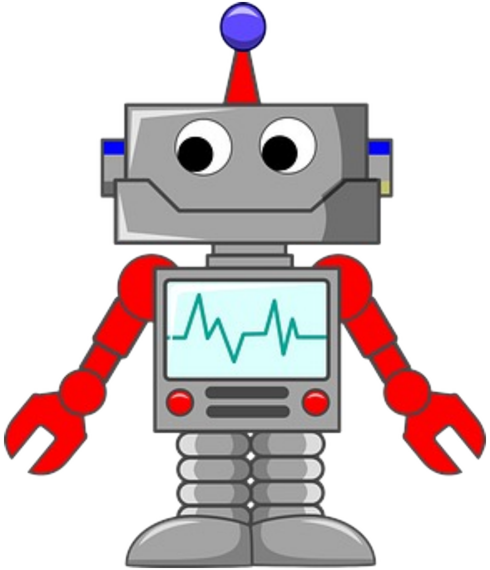
On the contemporary intelligent tests, the average performance for a given age is assigned a score of 100.

IQ

Example:

$$\text{IQ} = \frac{mental\ age\ of\ 13}{chronological\ age\ of\ 10} \times 100 = 130$$

Neuroscience



What is Neuroscience

Neuroscience is the scientific study of the nervous system.

Neural System

A **neuron** is a nerve cell; the basic building block of the nervous system.

Dendrites are a neuron's often bushy, branching extensions that receive messages and conduct impulses toward the cell body.

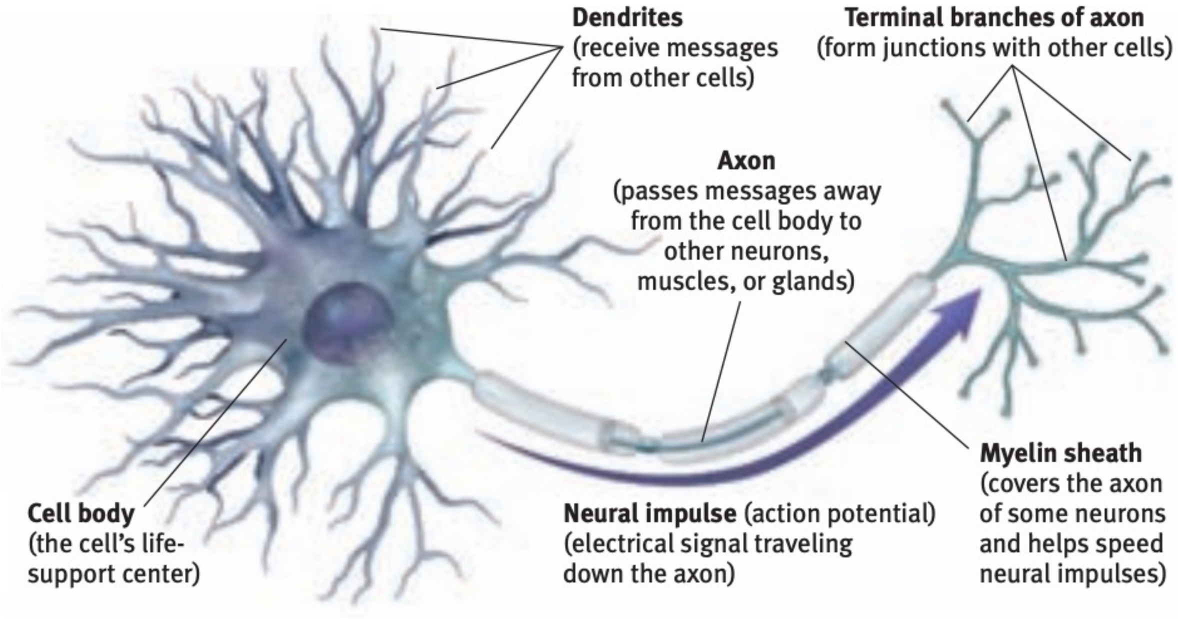
Axons are the neuron extension that passes messages through its branches to other neurons or to muscles or glands.

Neural System

The **Myelin Sheath** is a fatty tissue layer segmentally encasing the axons of some neurons; enables vastly greater transmission speed as neural impulses hop from one node to the next.

Synapse is the junction between the axon tip of the sending neuron and the dendrite or cell node of the receiving neuron. The tiny gap at this junction is called the synaptic gap or synaptic cleft.

Neural System



Neural System

Neurotransmitters are the chemical messengers that cross the synaptic gaps between neurons. When released by the sending neuron, neurotransmitters travel across the synapse and bind to receptor sites on the receiving neuron, thereby influencing whether that neuron will generate neural impulse.

Neural System

Examples of neurotransmitters are:

- Acetylcholine (ACh) – enables muscle action, learning, and memory.
- Dopamine – influences movement, learning, attention, and emotion.
- Serotonin – affects mood, hunger, sleep, and arousal.
- Norepinephrine – helps control alertness and arousal.
- GABA (Gamma-aminobutyric acid) – a major inhibitory neurotransmitter.
- Glutamate – a major excitatory neurotransmitter; involved in memory.
- Endorphins – neurotransmitters that influence the perception of pain or pleasure.

The Brain

The limbic system is the neural system (including the amygdala, hypothalamus, and hippocampus) located below the cerebral hemispheres; associated with emotions and drives.

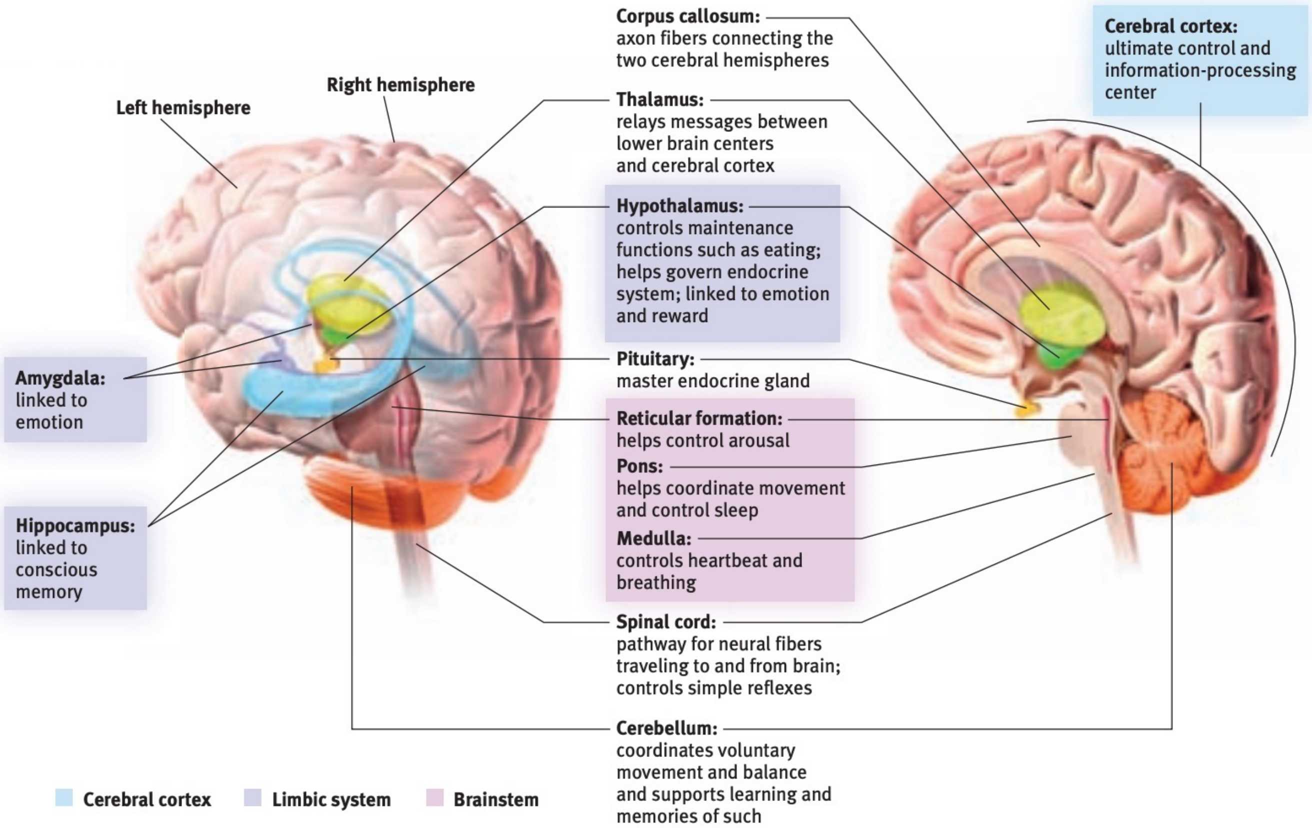
The Brain

The amygdala is the two lima-bean sized neural clusters in the limbic system; linked to emotion.

The Brain

The Hypothalamus a neural structure lying below the thalamus; it directs several maintenance activities (eating, drinking, body temperature), helps govern the endocrine system via the pituitary gland, and is linked to emotion and reward.

Neuroscience



The Brain

The **Cerebral Cortex** is the intricate fabric of interconnected neural cells covering the cerebral hemispheres; the body's ultimate control and information-processing center.

The **frontal lobes** are the portion of the cerebral cortex lying just behind the forehead; involved in speaking and muscle movements and in making plans and judgments.

The Brain

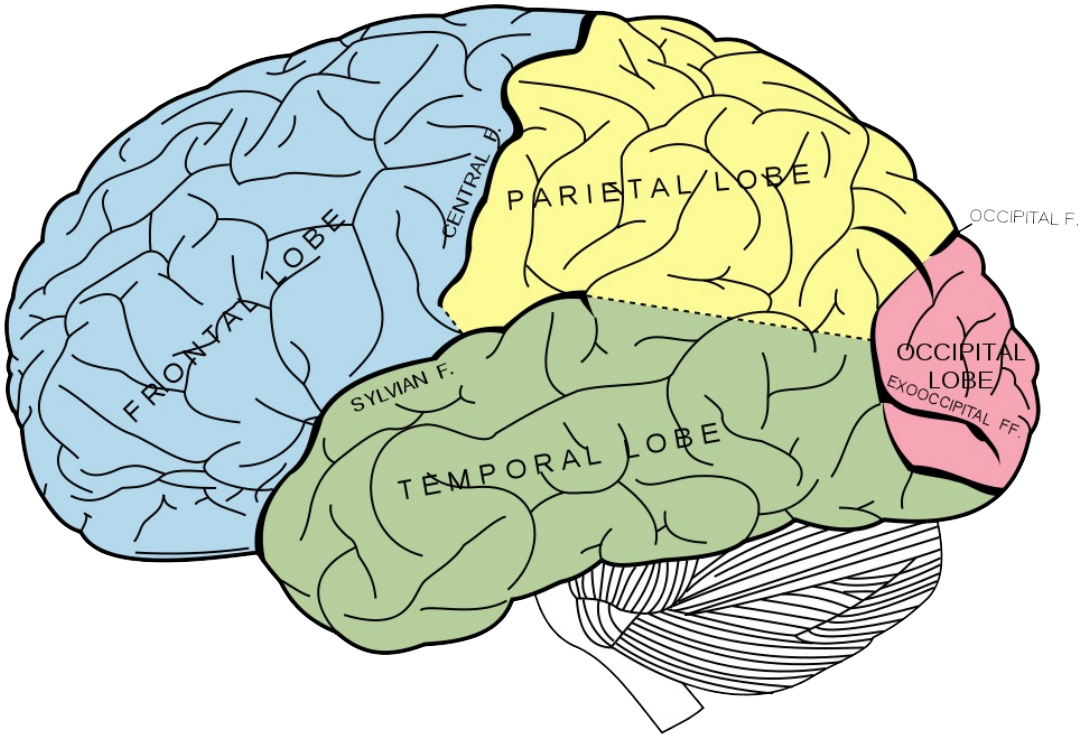
The **parietal lobes** are the portion of the cerebral cortex lying at the top of the head and toward the rear; receives sensory input for touch and body position.

The **occipital lobes** are the portion of the cerebral cortex lying at the back of the head; includes areas that receive information from the visual fields.

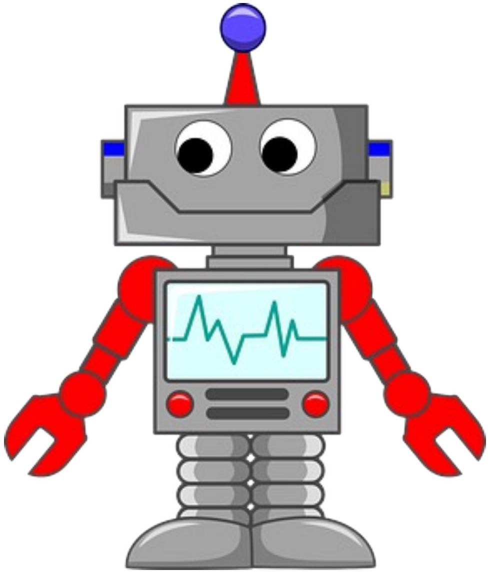
The Brain

The **temporal lobes** are the portion of the cerebral cortex lying roughly above the ears; includes the auditory areas, each receiving information primarily from the opposite ear.

Neuroscience



Artificial Intelligence



Artificial Neural Networks

Artificial Neural Networks are modeled after biological neural networks and attempt to allow computers to learn in a similar manner to humans - reinforcement learning.

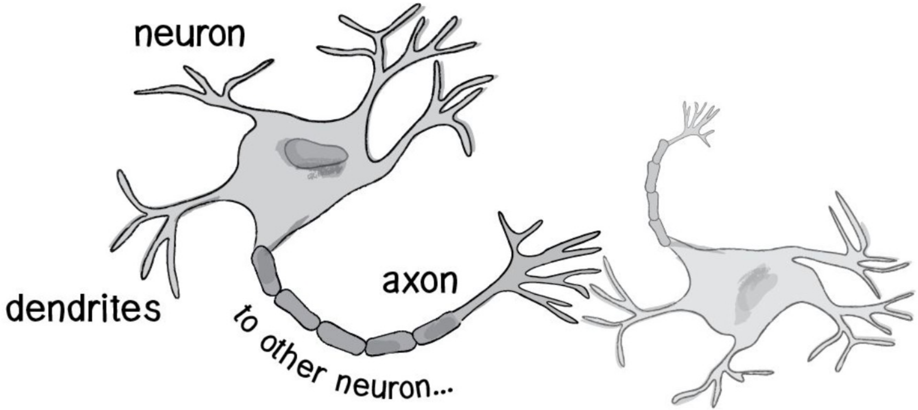
Artificial Neural Networks

ANNs are used for the following:

- Pattern Recognition
- Time Series Predictions
- Signal Processing
- Anomaly Detection
- Control

Artificial Neural Networks

The human brain has interconnected neurons with dendrites that receive inputs, and then based on those inputs, produce an electrical signal output through the axon.



Artificial Neural Networks

There are problems that are difficult for humans but easy for computers (e.g. calculating large arithmetic problems).

Then there are problems easy for humans, but difficult for computers (e.g. recognizing a picture of a person from the side).

Artificial Neural Networks

Neural Networks attempt to solve problems that would normally be easy for humans but hard for computers!

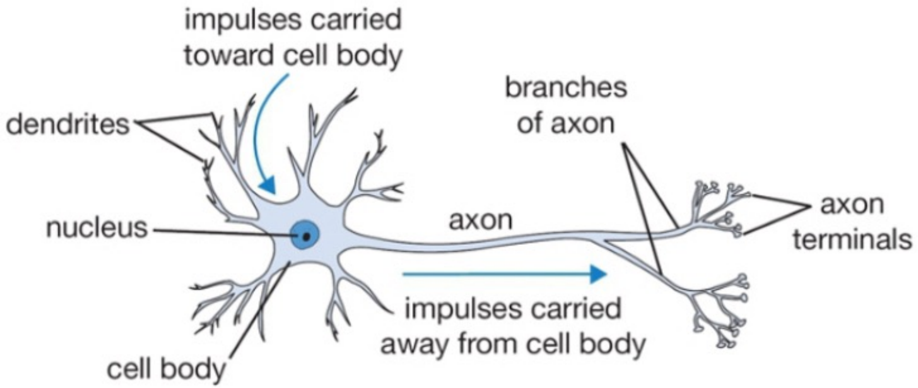
Let’s start by looking at the simplest Neural network possible - the perceptron.

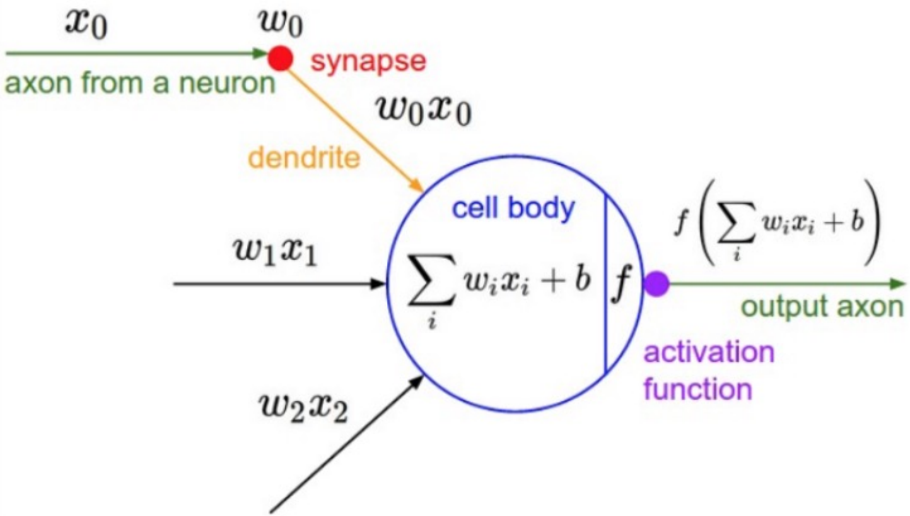
Artificial Neural Networks

A perceptron consists of one or more inputs, a processor, and a single output.

A perceptron follows the “feed-forward” model, meaning inputs are sent into the neuron, are processed, and result in an output.

Artificial Neural Networks

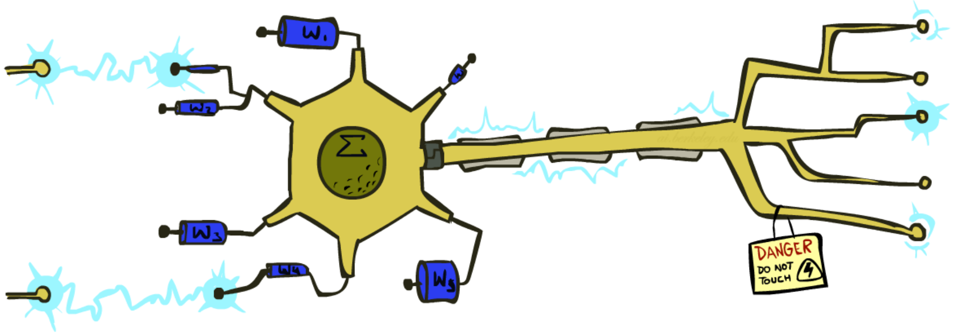




Artificial Neural Networks

Perceptrons consists of:

- input values
- synaptic weights
- a bias
- activation function



Artificial Neural Networks

The way it works is as follows: suppose we have inputs $x_0, x_1, x_2, \dots, x_n$, where each are assigned a given weight, call them $w_0, w_1, w_2, \dots, w_n$.

$$x_0 w_0 + x_1 w_1 + x_2 w_2 + \cdots + x_n w_n = \sum x_i w_i$$

Here, the weights represent how strong a given node is.

Artificial Neural Networks

Given the result of our partial sum, we may now construct the **activation function**, which is $f(\sum x_i w_i + b)$.

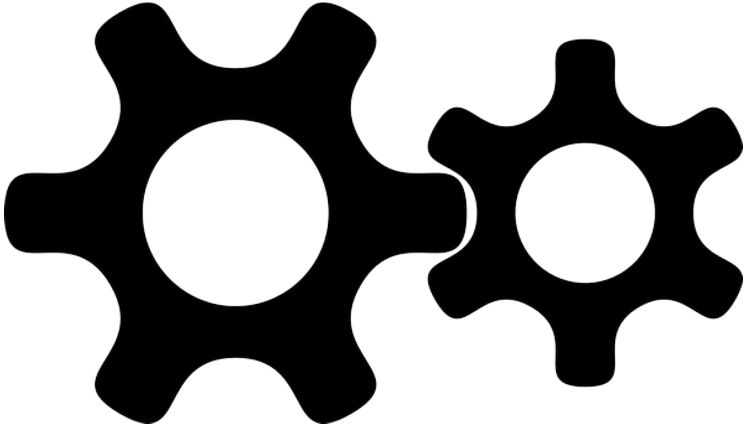
The bias *b* value brings the activation function up or down.

An activation function helps transforms an input signal to an output signal.

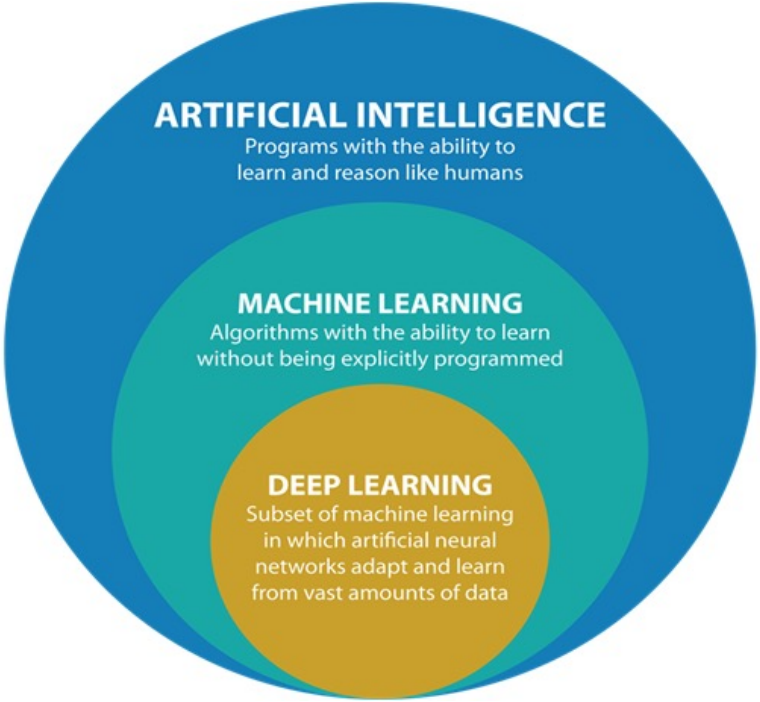
Artificial Neural Networks

There are several kinds of activation functions:

- Threshold Activation Function
- Sigmoid Activation Function
- Hyperbolic Tangent Function
- Rectified Linear Units



AI vs. Machine Learning vs. Deep Learning



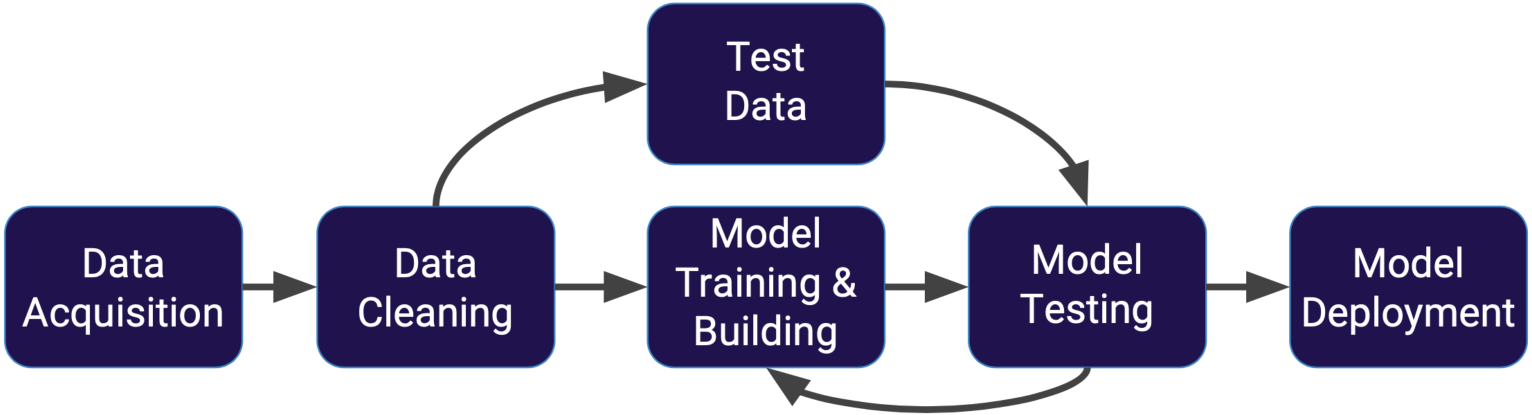
Machine Learning

Machine learning is a method of data analysis that automates analytical model building.

Machine Learning

Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look.

Machine Learning



Supervised Learning

Supervised learning algorithms are trained using **labeled** examples, such as an input where the desired output is known.

Supervised Learning

The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors.

It then modifies the model accordingly.

Supervised Learning

Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabeled data.

Supervised learning is commonly used in applications where historical data predicts likely future events.

Supervised Learning

For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim.

Or it can attempt to predict the price of a house based on different features for houses for which we have historical price data.

Unsupervised Learning

Unsupervised learning is used against data that has no historical labels.

Unsupervised Learning

The system is not told the "right answer." The algorithm must figure out what is being shown.

The goal is to explore the data and find some structure within.

Unsupervised Learning

Or it can find the main attributes that separate customer segments from each other.

Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decom

Reinforcement Learning

Reinforcement learning is often used for robotics, gaming and navigation.

Reinforcement Learning

With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards.

Reinforcement Learning

This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do.)

Reinforcement Learning

The objective is for the agent to choose actions that maximize the expected reward over a given amount of time.

Reinforcement Learning

The agent will reach the goal much faster by following a good policy.

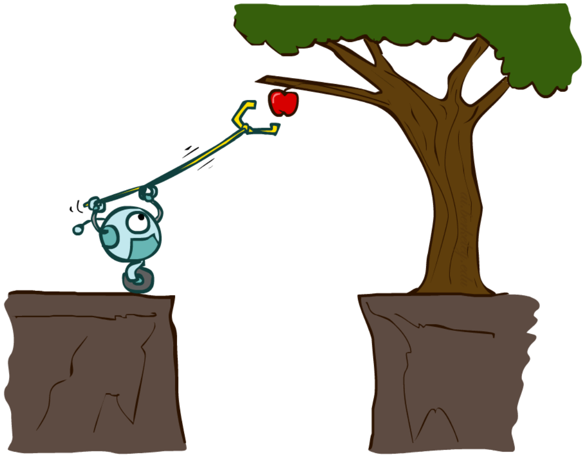
So the goal in reinforcement learning is to learn the best policy.

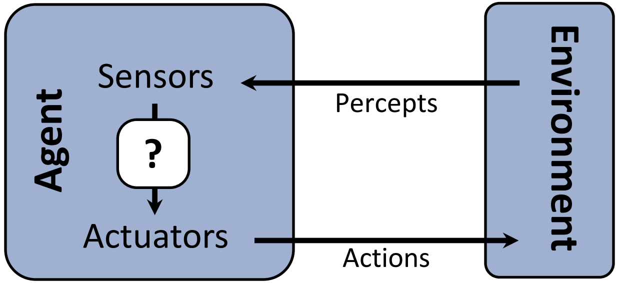
Designing Rational Agents

An **agent** is an entity that *perceives* and *acts*.

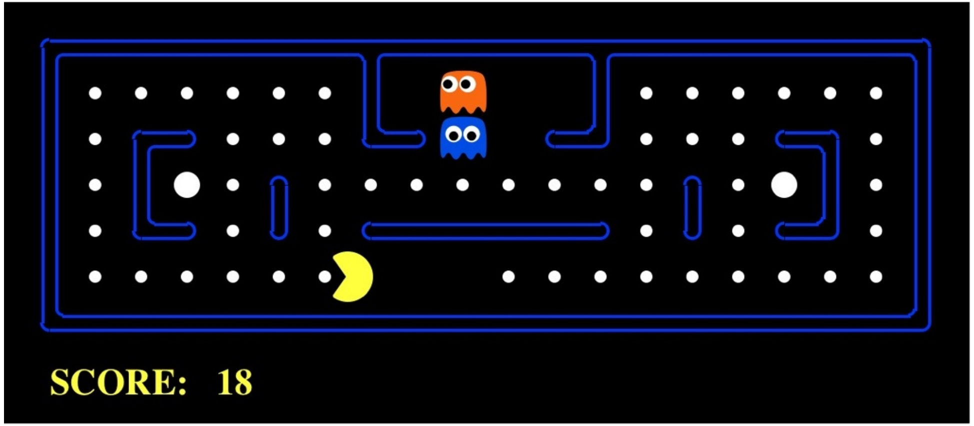
A **rational agent** selects actions that maximize its (expected) **utility**.

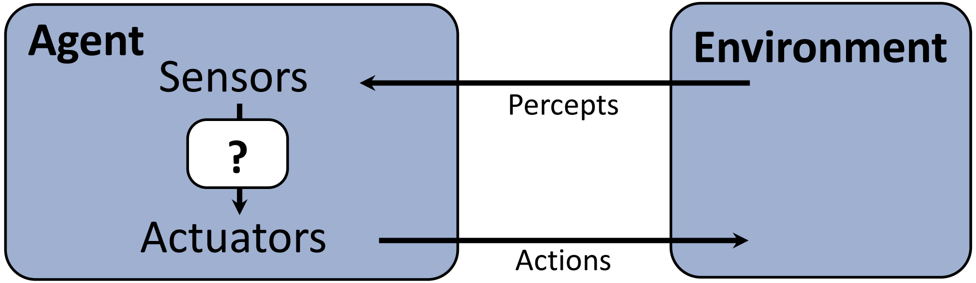
Characteristics of the **percepts, environment,** and **action space** dictate techniques for selecting rational actions.





Pac-Man as an Agent





Pac-Man is a registered trademark of Namco-Bandai Games, used here for educational purposes

Bayesian Networks

There are instances which call for the need to calculate the probability of an uncertain event or cause based on an observation.

Bayesian Networks

Really useful way of doing so is through graphical models known as Bayesian Networks.

Bayesian Networks

A Bayesian network is a probabilistic graphical model that represent information about an uncertain event, cause, or domain.

Bayesian Networks

This gives us the chance to classify entities, quantities, or other pieces of information through observation.

Bayesian Networks

In order to understand what Bayesian networks are, we must understand what they consist of, which is a set of conditional probability distributions and a directed acyclic graph.

Bayesian Networks

Recall, a directed acyclic graph is a directed graph in which has no directed cycles.

A conditional probability distribution (CPD) is a data set where the probability of the events being true is based on another event being true.

Bayesian Networks

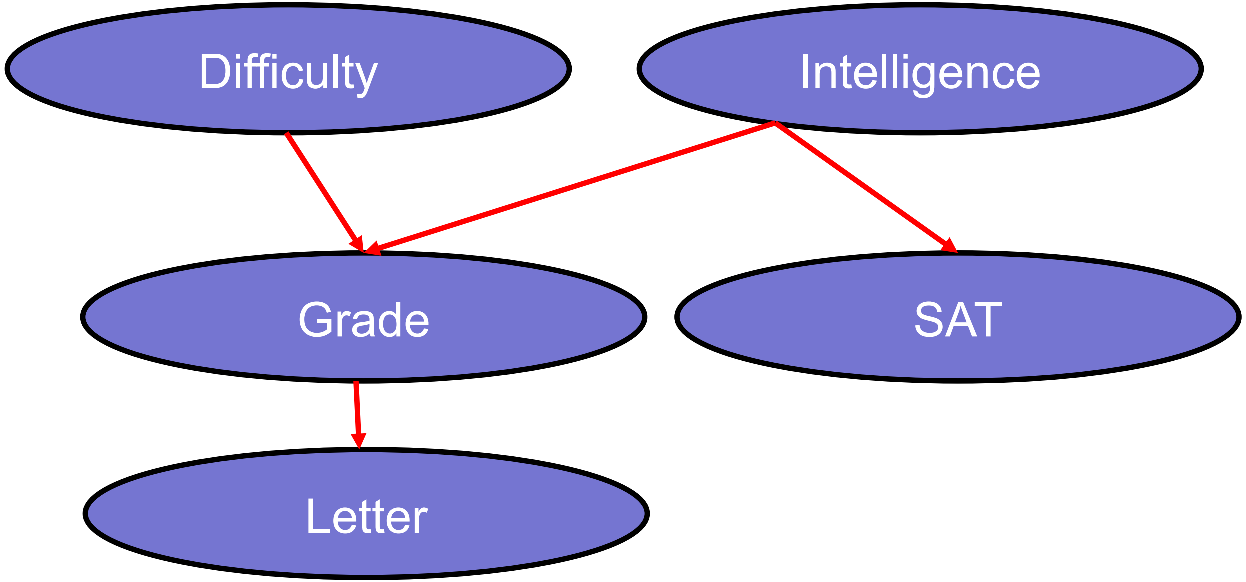
The goal of Bayesian Network is to compute the CPD of each of the unobserved caused, supposing there are observed evidences.

Bayesian Networks

Determine the side effects a drug will cause with a given patient:

Side effects are negative reactions to medical treatment. Here, we can represent the probabilistic relationship on the side effects to predict what drug caused it. Bayesian Networks are useful in this case because they will observe the symptoms of a patient, and then based on those observations, predict which drug resulted in the negative reactions.

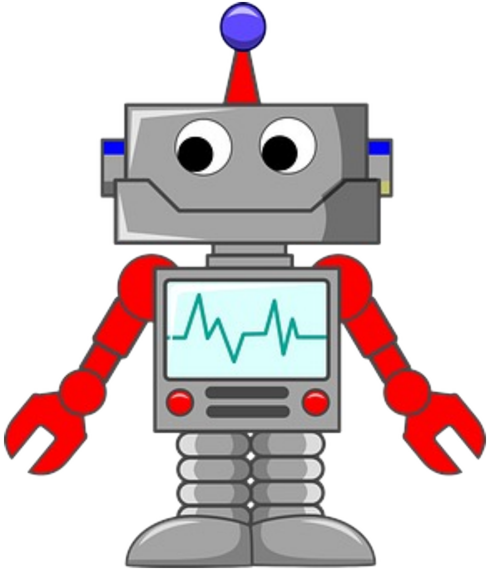
Bayesian Model



This model says there are independent states difficulty (based on difficulty of exam) and other is intelligence (based on intelligence).

Based on difficulty and intelligence, the student gets a grade and according to that grade, they get a letter of recommendation for college.

Applications



Virtual Reality

Definition of VR: Inducing targeted behavior in an organism by using artificial sensory stimulation, while the organism has little or no awareness of the interference.

Artificial Intelligence

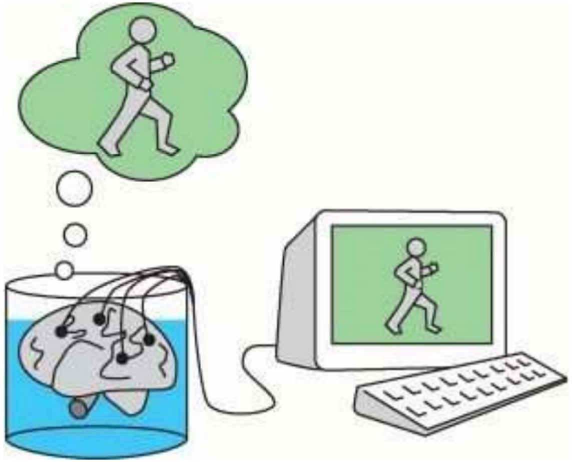
Virtual Reality

Four key components appear in the definition:

- **Targeted behavior**: The organism is having an "experience" that was designed by the creator. Examples include flying, walking, exploring, watching a movie, and socializing with other organisms.
- **Organism**: This could be you, someone else, or even another life form such as a fruit fly, cockroach, fish, rodent, or monkey (scientists have used VR technology on all of these!).
- **Artificial sensory stimulation**: Through the power of engineering, one or more senses of the organism become co-opted, at least partly, and their ordinary inputs are replaced or enhanced by artificial stimulation.
- **Awareness**: While having the experience, the organism seems unaware of the interference, thereby being "fooled" into feeling present in a virtual world. This unawareness leads to a sense of presence in an altered or alternative world. It is accepted as being natural.

Virtual Reality

A VR system causes a perceptual illusion to be maintained for the organism.



Virtual Reality

We also cannot help wondering whether we are always being fooled, and some greater reality has yet to reveal itself to us. This problem has intrigued the greatest philosophers over many centuries. One of the oldest instances is the Allegory of the Cave, presented by Plato in Republic. In this, Socrates describes the perspective of people who have spent their whole lives chained to a cave wall. They face a blank wall and only see shadows projected onto the walls as people pass by. He explains that the philosopher is like one of the cave people being finally freed from the cave to see the true nature of reality, rather than being only observed through projections.

Virtual Reality

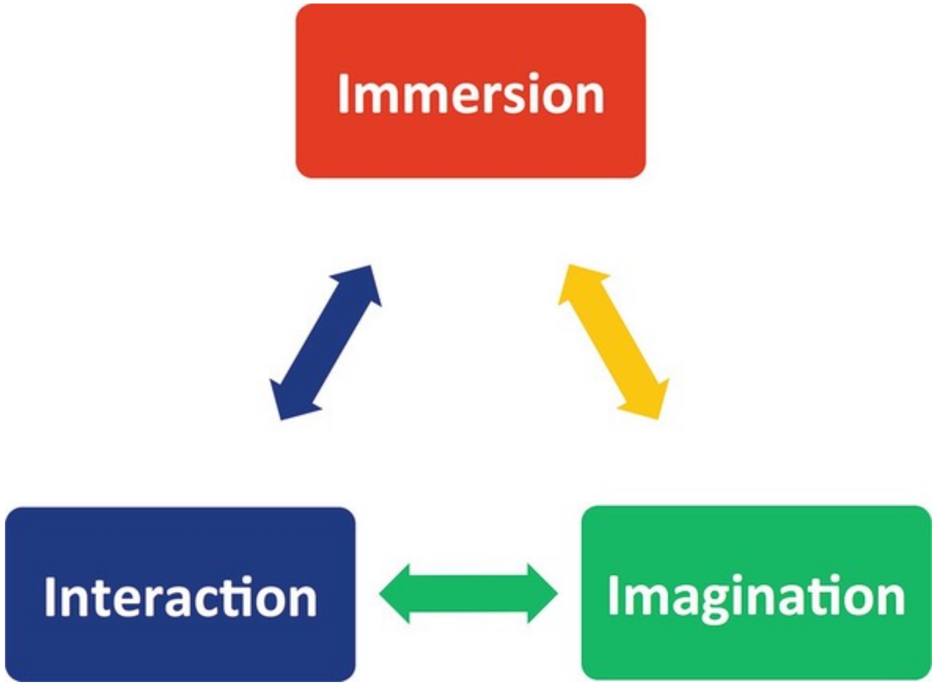
This idea has been repeated and popularized throughout history, and also connects deeply with spirituality and religion.

Virtual Reality

The basis of the 1999 movie The Matrix. In that story, machines have fooled the entire human race by connecting to their brains to a convincing simulated world, while harvesting their real bodies. The lead character Neo must decide whether to face the new reality or take a memory-erasing pill that will allow him to comfortably live in the simulation without awareness of the ruse.

Artificial Intelligence

Virtual Reality



Virtual Reality

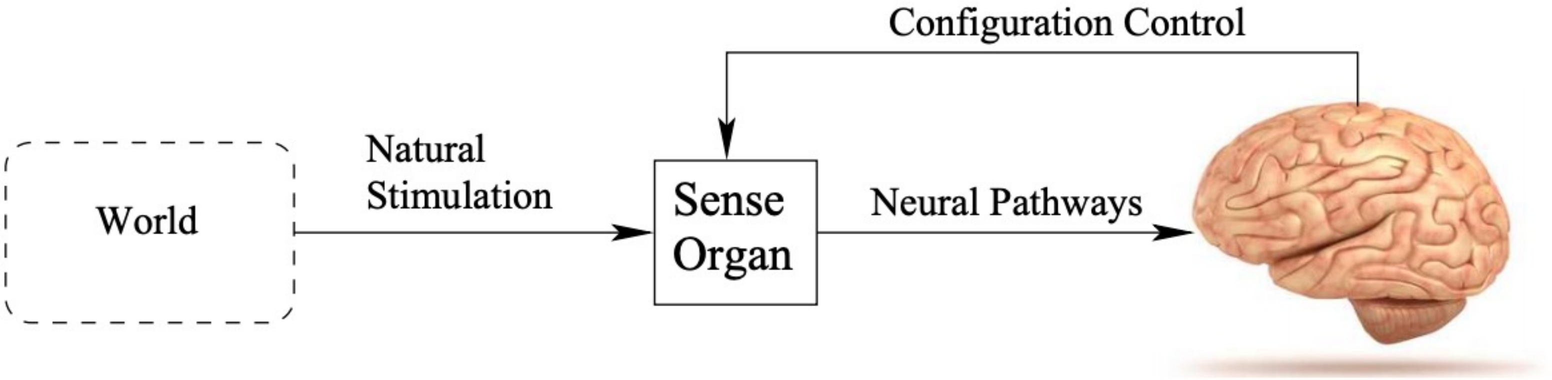
Immersion means that the users feel that they are part of the virtual world in the VR scene, as if they are immersed.

Interaction refers to the natural interaction between the user and the virtual scene. It provides the users with the same feeling as the real world through feedback.

Imagination refers to the use of multi-dimensional perception information provided by VR scenes to acquire the same feelings as the real world while acquiring the feelings that are not available in the real world.

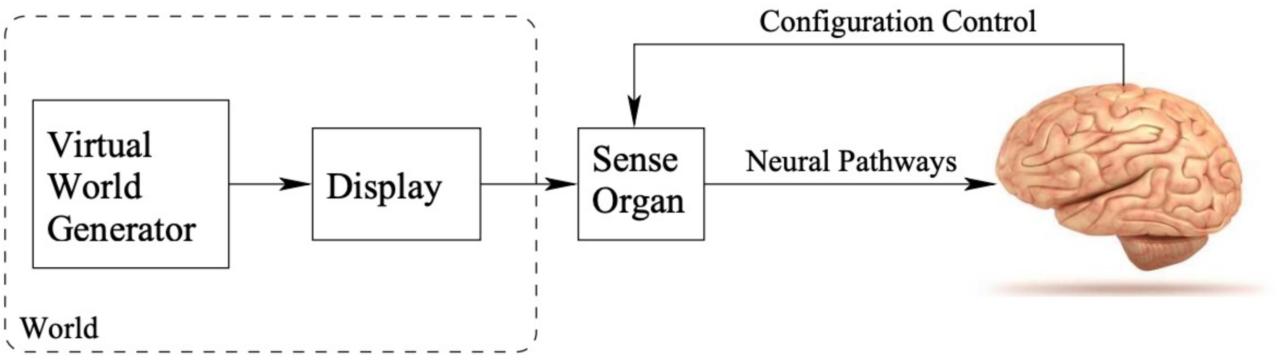
Virtual Reality

Under normal conditions, the brain (and body parts) control the configuration of sense organs (eyes, ears, fingertips) as they receive natural stimulation from the surrounding, physical world.



Virtual Reality

In comparison to the figure before, a VR system “hijacks” each sense by replacing the natural stimulation with artificial stimulation that is provided by hardware called a display. Using a computer, a virtual world generator maintains a coherent, virtual world. Appropriate “views” of this virtual world are rendered to the display.



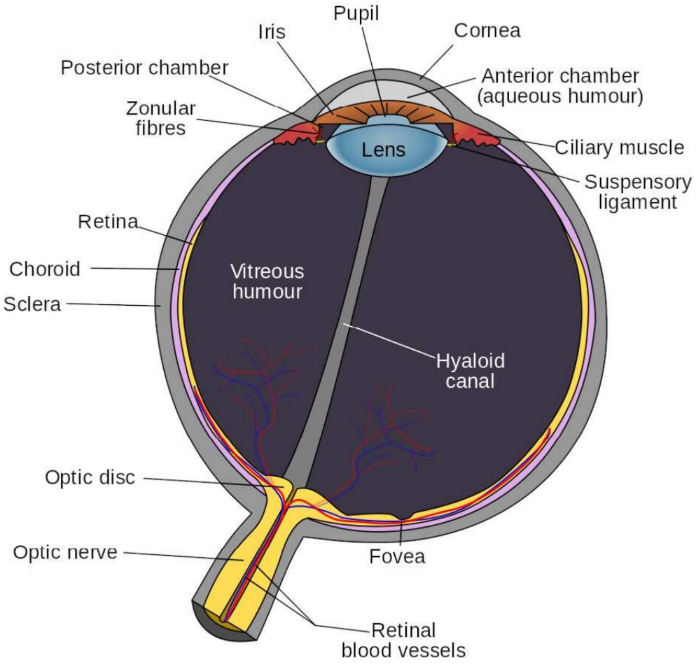
Virtual Reality

Augmented Reality

Augmented reality (AR) refers to systems in which most of the visual stimuli are propagated directly through glass or cameras to the eyes, and some additional structures, such as text and graphics, appear to be superimposed onto the user's world.

Augmented Reality

Physiology of Human Vision



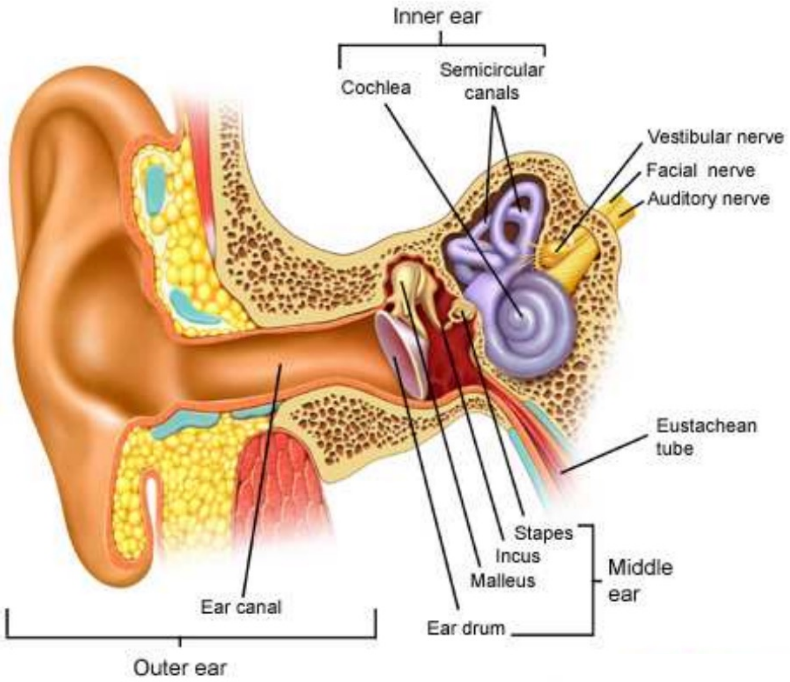
Physiology of Human Vision

Photoreceptors

The retina contains two kinds of photoreceptors for vision:

1) Rods, which are triggered by very low levels of light
2) Cones, which require more light and are designed to distinguish between colors

Physiology of Human Auditory System



Artificial Intelligence

Robotics

Robots are typically defined as physical agents that perform a variety of tasks by manipulating the physical world.

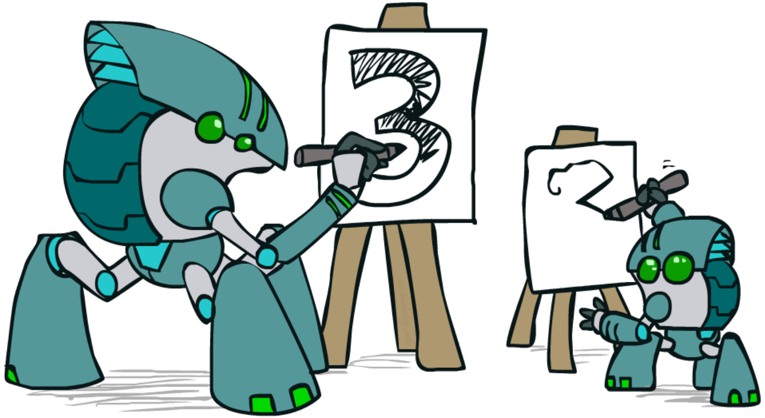
Robotics

Robots are equipped with effectors like legs, wheels, arms & grippers. The single purpose of effectors is to exert physical force on the environment.

Robotics

Mostly, today's robots fall into one of three primary categories:

- *Manipulators*
- *Mobile Robots*
- *Mobile Manipulators*



Robotics

Manipulators: Often referred to as robot arms, manipulators are physically anchored to their workplace.

Robotics

Mobile robots: Mobile robots have the ability to move about using wheels or robotic legs.

- Unmanned Ground Vehicles (**UGVs**) can drive autonomously on streets, highways and also off-roads.
- Unmanned Air Vehicles (**UAVs** aka drones) are commonly used for surveillance, military operations, crop spraying and even deliveries.
- Autonomous Underwater Vehicles (**AUVs**) are used in deep-sea explorations and underwater searches.

Robotics

Mobile Manipulators: Sometimes referred to as Humanoid Robots, these typically mimic the human torso. Mobile manipulators can apply their effectors over a much larger area than typical manipulators, which are anchored. However their task is made more difficult since they lack the rigidity that anchors provide.

Robotics

Real robots must cope with environments that are partially observable, stochastic, dynamic and continuous.

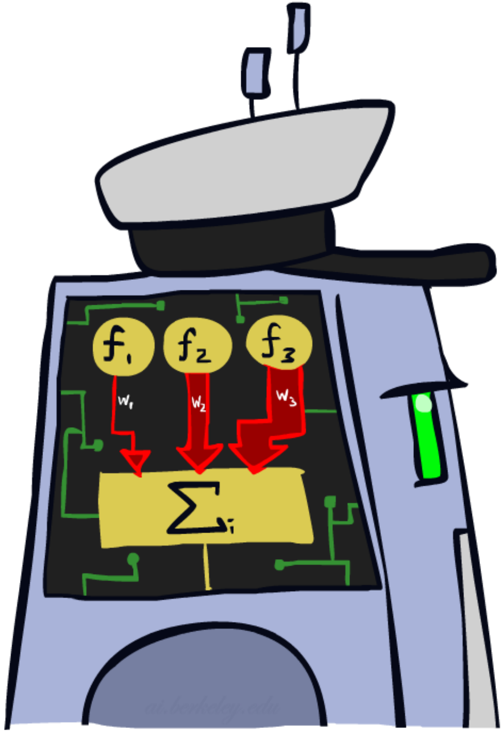
Robotics

Robotics today brings together several concepts from AI and Machine Learning like probabilistic state estimation, perception, unsupervised learning and reinforcement learning, among others.

Artificial Intelligence

Gaming

AI has come to video games.



Gaming

AI needs to understand what a player does and how a player feels during the play.

Gaming

To gauge a player’s in-game experience, developers use machine learning methods, such as supervised learning like support vector machines or neural networks to build the models of player experience.

Artificial Intelligence

Gaming

Zombies in Call of Duty are a prime example of how AI is used in video games.

Voice Recognition

In order for voice recognition to occur, we need to convert soundwaves into bits.

The computer then interprets those bits.

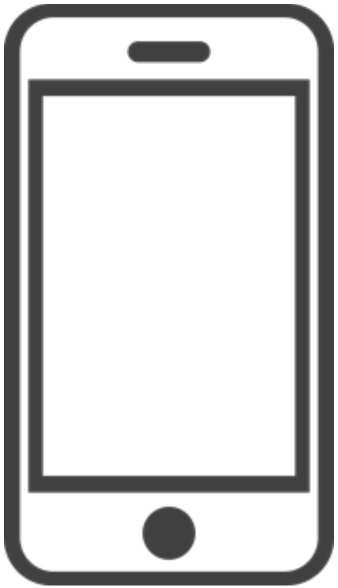
Voice Recognition

Voice recognition is everywhere!

Voice Recognition

It's in our:

- phones
- game consoles
- watches
- TV's
- etc.

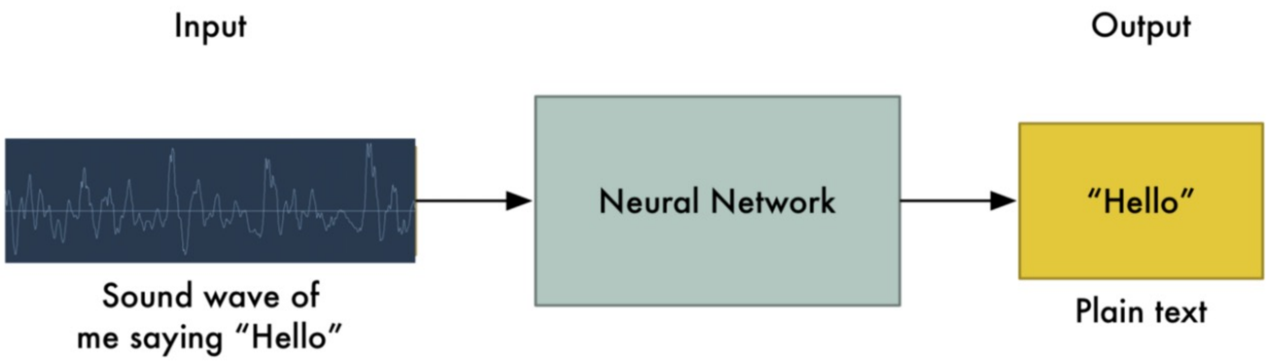


Voice Recognition

Ever heard of Amazon's Echo Dot?

Voice Recognition

Here's a visual of how voice recognition occurs.



Voice Recognition

Sometimes, we either say letters or words that sound like another.

In that case, our system will need to determine what we meant to say based on what it has heard us say before.

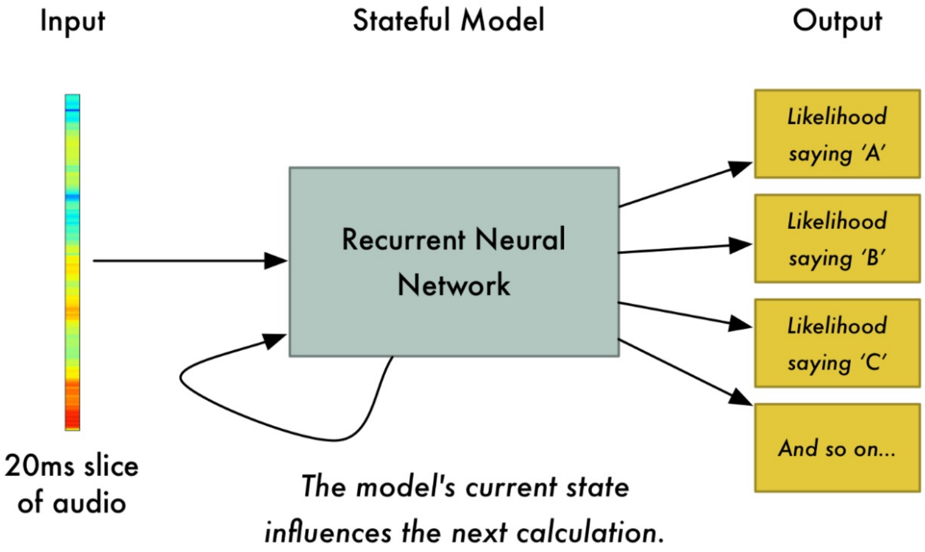
Voice Recognition

To treat this phenomenon, we can make use of a recurrent neural network.

A recurrent neural network has a memory that influences future predictions.

Voice Recognition

An example of a recurrent neural network



Natural Language Processing

Natural Language Processing, or *NLP*, is the sub-field of AI that is focused on enabling computers to understand and process human languages.

Natural Language Processing

Can Computers Understand Language?

Natural Language Processing

As long as computers have been around, programmers have been trying to write programs that understand languages like English.

Natural Language Processing

The reason is pretty obvious — humans have been writing things down for thousands of years and it would be really helpful if a computer could read and understand all that data.

Natural Language Processing

Computers can’t yet truly understand English in the way that humans do — but they can already do a lot!

Natural Language Processing

In certain limited areas, what you can do with NLP already seems like magic.

Natural Language Processing

Extracting Meaning from Text is Hard

The process of reading and understanding English is very complex — and that's not even considering that English doesn't follow logical and consistent rules. For example, what does this news headline mean?

"Environmental regulators grill business owner over illegal coal fires."

Are the regulators questioning a business owner about burning coal illegally? Or are the regulators literally cooking the business owner? As you can see, parsing English with a computer is going to be complicated.

Natural Language Processing

Doing anything complicated in machine learning usually means *building a pipeline*.

Natural Language Processing

The idea is to break up your problem into very small pieces and then use machine learning to solve each smaller piece separately.

Natural Language Processing

Then by chaining together several machine learning models that feed into each other, you can do very complicated things.

Natural Language Processing

And that's exactly the strategy we are going to use for NLP.

Natural Language Processing

London is the capital and most populous city of England and the United Kingdom.

It would be great if a computer could read this text and understand that London is a city and London is located in England.

Natural Language Processing

Word Tokenization

In our pipeline, we will break this sentence into separate words or *tokens*. This is called *tokenization*. This is the result:

“London”, “is”, “ the”, “capital”, “and”, “most”, “populous”, “city”, “of”, “England”, “and”, “the”, “United”, “Kingdom”, “.”

Tokenization is easy to do in English. We’ll just split apart words whenever there’s a space between them. And we’ll also treat punctuation marks as separate tokens since punctuation also has meaning.

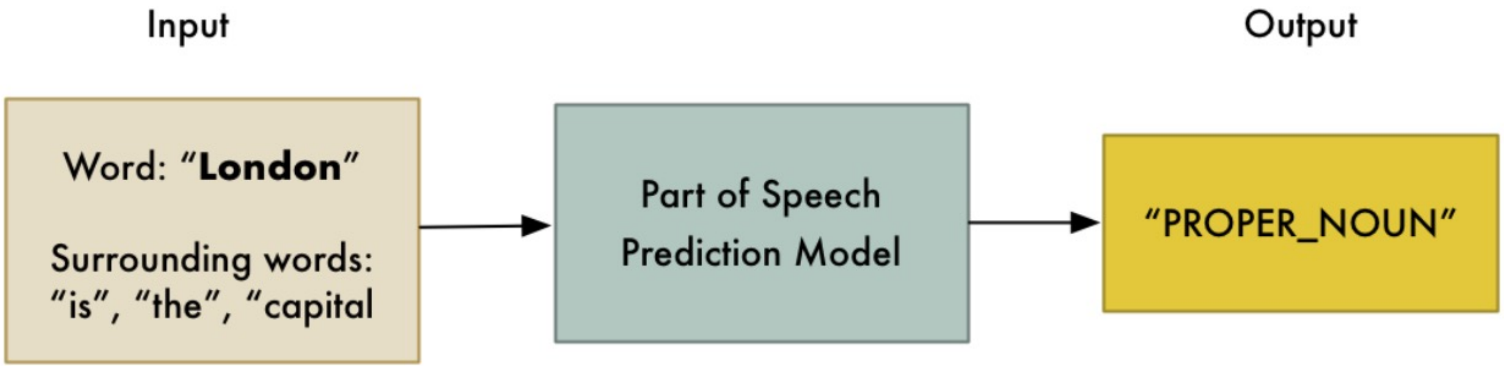
Natural Language Processing

Predicting Parts of Speech for Each Token

Next, we'll look at each token and try to guess its part of speech — whether it is a noun, a verb, an adjective and so on. Knowing the role of each word in the sentence will help us start to figure out what the sentence is talking about.

Natural Language Processing

We can do this by feeding each word (and some extra words around it for context) into a pretrained part-of-speech classification model:



Natural Language Processing

The part-of-speech model was originally trained by feeding it millions of English sentences with each word’s part of speech already tagged and having it learn to replicate that behavior.

Natural Language Processing

Keep in mind that the model is completely based on statistics — it doesn’t actually understand what the words mean in the same way that humans do. It just knows how to guess a part of speech based on similar sentences and words it has seen before.

Natural Language Processing

After processing the whole sentence, we'll have a result like this:

Natural Language Processing

Text Lemmatization

In English (and most languages), words appear in different forms. Look at these two sentences:

- I had a **pony**.
- I had two **ponies**.

Natural Language Processing

Both sentences talk about the noun **pony,** but they are using different inflections.

Natural Language Processing

When working with text in a computer, it is helpful to know the base form of each word so that you know that both sentences are talking about the same concept.

Natural Language Processing

Otherwise the strings “pony” and “ponies” look like two totally different words to a computer.

Natural Language Processing

In NLP, we call finding this process *lemmatization* — figuring out the most basic form or *lemma* of each word in the sentence.

Natural Language Processing

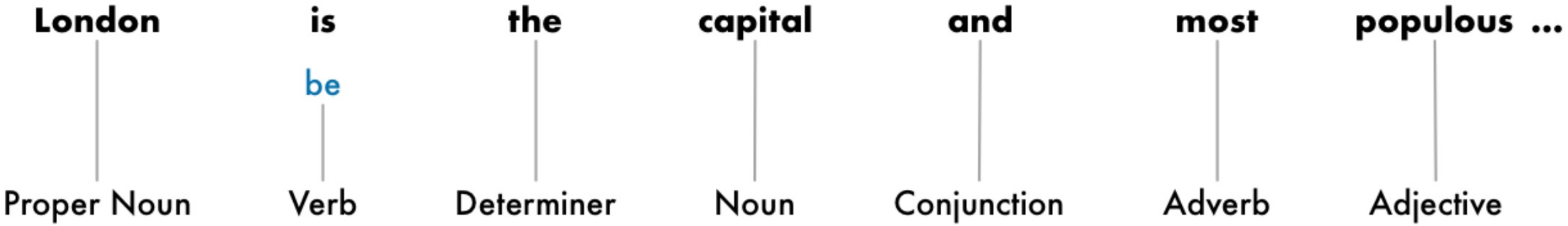
The same thing applies to verbs. We can also lemmatize verbs by finding their root, unconjugated form. So “**I had two ponies**” becomes “**I [have] two [pony].**”

Natural Language Processing

Lemmatization is typically done by having a look-up table of the lemma forms of words based on their part of speech and possibly having some custom rules to handle words that you’ve never seen before.

Natural Language Processing

Here's what our sentence looks like after lemmatization adds in the root form of our verb:



The only change we made was turning "is" into "be".

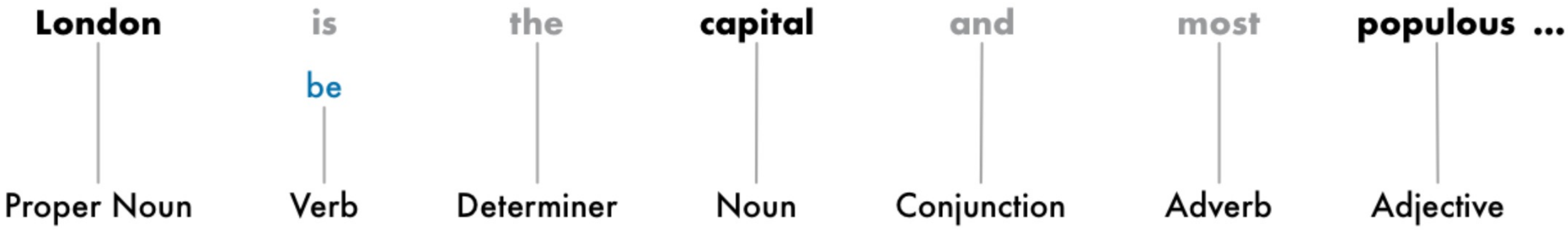
Natural Language Processing

Identifying Stop Words

Next, we want to consider the importance of each word in the sentence. English has a lot of filler words that appear very frequently like “and”, “the”, and “a”. When doing statistics on text, these words introduce a lot of noise since they appear way more frequently than other words. Some NLP pipelines will flag them as **stop words** —that is, words that you might want to filter out before doing any statistical analysis.

Natural Language Processing

Here's how our sentence looks with the stop words grayed out:



Natural Language Processing

Stop words are usually identified by just by checking a hardcoded list of known stop words. But there's no standard list of stop words that is appropriate for all applications. The list of words to ignore can vary depending on your application.

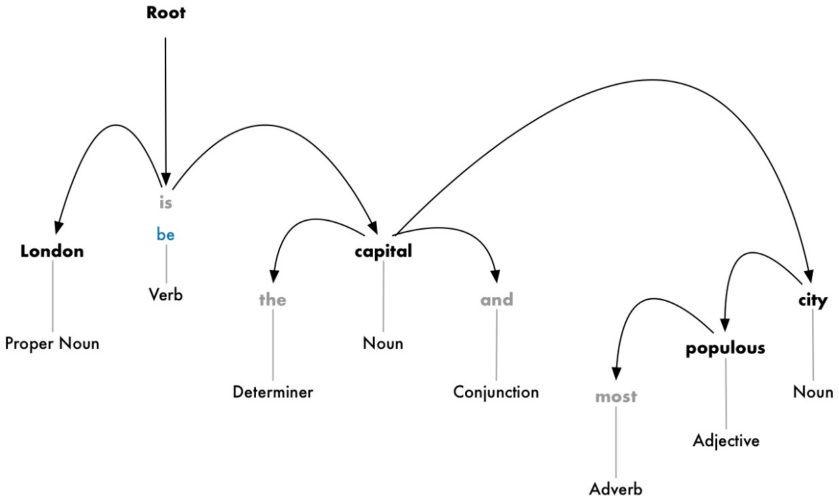
Natural Language Processing

Dependency Parsing

The next step is to figure out how all the words in our sentence relate to each other. This is called *dependency parsing*.

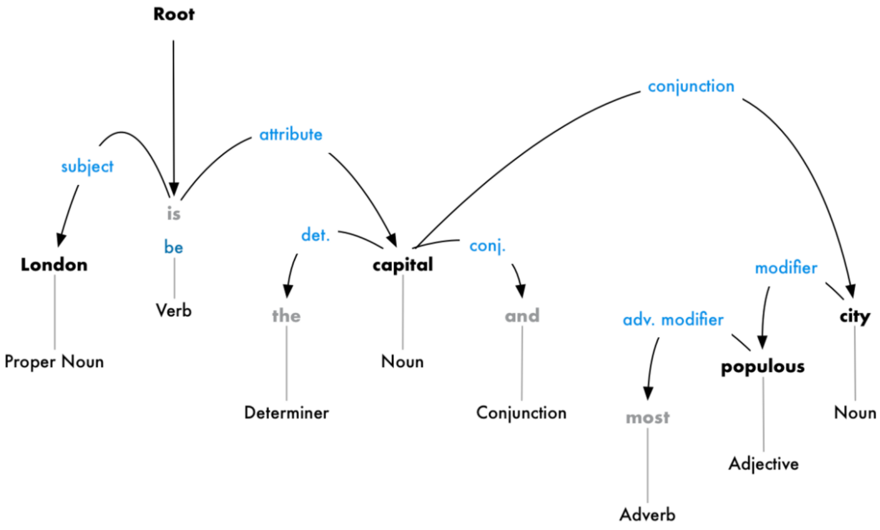
Natural Language Processing

The goal is to build a tree that assigns a single **parent** word to each word in the sentence. The root of the tree will be the main verb in the sentence. Here's what the beginning of the parse tree will look like for our sentence:



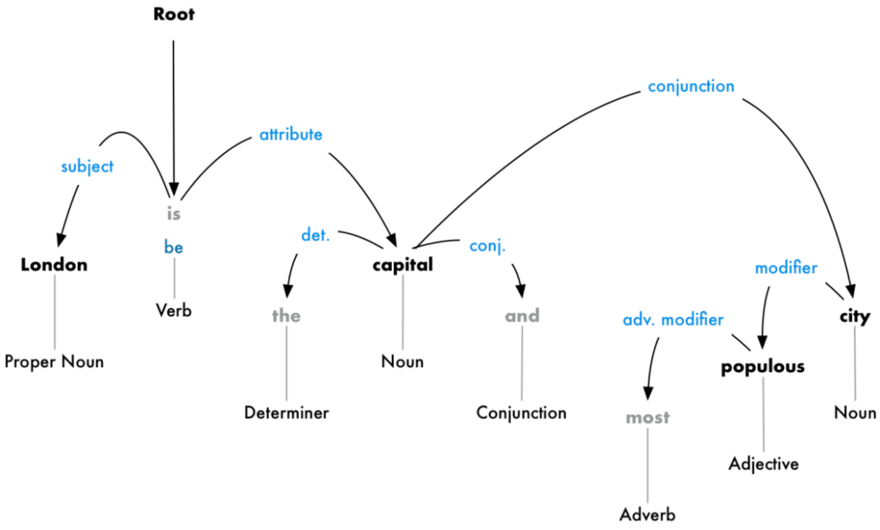
Natural Language Processing

But we can go one step further. In addition to identifying the parent word of each word, we can also predict the type of relationship that exists between those two words:



Natural Language Processing

This parse tree shows us that the subject of the sentence is the noun "*London*" and it has a "*be*" relationship with "*capital*". We finally know something useful — *London* is a *capital*! And if we followed the complete parse tree for the sentence (beyond what is shown), we would even found out that London is the capital of the *United Kingdom*.



Natural Language Processing

It’s also important to remember that many English sentences are ambiguous and just really hard to parse.

Natural Language Processing

In those cases, the model will make a guess based on what parsed version of the sentence seems most likely but it’s not perfect and sometimes the model will be embarrassingly wrong.

Natural Language Processing

But over time our NLP models will continue to get better at parsing text in a sensible way.

Natural Language Processing

Finding Noun Phrases

So far, we've treated every word in our sentence as a separate entity. But sometimes it makes more sense to group together the words that represent a single idea or thing. We can use the information from the dependency parse tree to automatically group together words that are all talking about the same thing.

Natural Language Processing

For example, instead of this:

London	is	the	capital	and	most	populous ...
	be					
Proper Noun	Verb	Determiner	Noun	Conjunction	Adverb	Adjective

We can group the noun phrases to generate this:

London	is	the capital	and	most populous city ...
	be			
Proper Noun	Verb	Noun	Conjunction	Noun

Natural Language Processing

Whether or not we do this step depends on our end goal. But it’s often a quick and easy way to simplify the sentence if we don’t need extra detail about which words are adjectives and instead care more about extracting complete ideas.

Natural Language Processing

Named Entity Recognition (NER)

Now that we've done all that hard work, we can finally move beyond grade-school grammar and start actually extracting ideas.

Natural Language Processing

In our sentence, we have the following nouns:

London is the capital and most populous city of England and the United Kingdom.

Natural Language Processing

Some of these nouns present real things in the world.

For example, *“London”*, *“England”* and *“United Kingdom”* represent physical places on a map.

Natural Language Processing

It would be nice to be able to detect that!
With that information, we could automatically extract a list of real-world places mentioned in a document using NLP.

Natural Language Processing

The goal of *Named Entity Recognition*, or *NER*, is to detect and label these nouns with the real-world concepts that they represent. Here's what our sentence looks like after running each token through our NER tagging model:

Natural Language Processing

But NER systems aren’t just doing a simple dictionary lookup.

Natural Language Processing

Instead, they are using the context of how a word appears in the sentence and a statistical model to guess which type of noun a word represents.

Natural Language Processing

A good NER system can tell the difference between "*Brooklyn Decker*" the person and the place "*Brooklyn*" using context clues.

Natural Language Processing

Here are just some of the kinds of objects that a typical NER system can tag:

- People’s names
- Company names
- Geographic locations (Both physical and political)
- Product names
- Dates and times
- Amounts of money
- Names of events

Natural Language Processing

If we had more than one sentence, we would have to consider the idea that English is full of pronouns — words like *he*, *she*, and *it*. These are shortcuts that we use instead of writing out names over and over in each sentence.

Natural Language Processing

Humans can keep track of what these words represent based on context. But our NLP model doesn’t know what pronouns mean because it only examines one sentence at a time.

Natural Language Processing

One way to dodge this is to use goal of coreference resolution, which figures out a mapping by tracking pronouns across sentences. We want to figure out all the words that are referring to the same entity.

Natural Language Processing

With coreference information combined with the parse tree and named entity information, we should be able to extract a lot of information out of this document!

Natural Language Processing

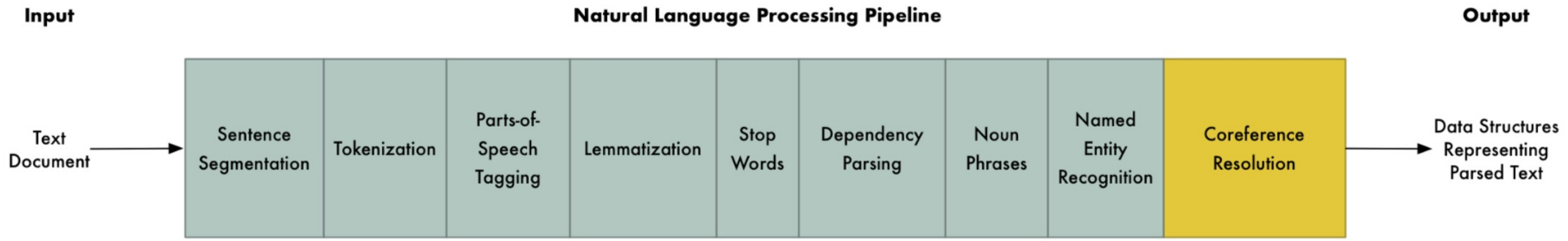
Coreference resolution is one of the most difficult steps in our pipeline to implement.

Natural Language Processing

It’s even more difficult than sentence parsing. Recent advances in deep learning have resulted in new approaches that are more accurate, but it isn’t perfect yet.

Natural Language Processing

Model of NLP Pipeline



Note, we did not have to use sentence segmentation and coreference resolution because we did not have multiple sentences.

Sentence segmentation breaks a paragraph of sentences into separate, single sentences.

CONGRATULATIONS

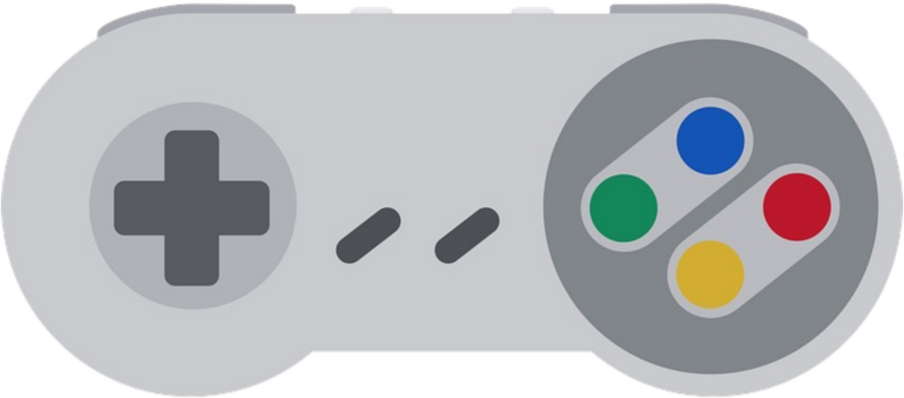
Course completed

YOU'VE COMPLETED THE ENTIRE COURSE!

CONGRATULATIONS

Course completed

Now we will create a game. This game will be a first-person shooter, and we will use artificial intelligence and unreal gaming engine to build it!



Thank You ☺