



UTeach

Computer Science

Principles



Table of Contents

1. [Welcome to Computer Science Principles](#) 1.1
2. [UNIT 1: Computational Thinking](#) 1.2
 1. [PROJECT: Password Generator Project](#) 1.2.1
 1. [Password Generator Project](#) 1.2.1.1
 2. [TOPIC: Algorithmic Thinking](#) 1.2.2
 1. [Problem Solving](#) 1.2.2.1
 1. [Building Blocks](#) 1.2.2.1.1
 2. [Problem Solving](#) 1.2.2.1.2
 2. [Flow Patterns](#) 1.2.2.2
 1. [Flow Patterns](#) 1.2.2.2.1
 2. [Flowcharts](#) 1.2.2.2.2
 3. [Algorithms](#) 1.2.2.2.3
 3. [CODING SKILLS: Encryption](#) 1.2.3
 1. [Encryption](#) 1.2.3.1
 4. [TOPIC: Cybersecurity](#) 1.2.4
 1. [Confidentiality](#) 1.2.4.1
 2. [Integrity](#) 1.2.4.2
 3. [Availability](#) 1.2.4.3
 4. [Social Engineering](#) 1.2.4.4
 5. [BIG PICTURE: Electronic Voting](#) 1.2.5
 1. [Electronic Voting](#) 1.2.5.1
 6. [TOPIC: Programming Languages](#) 1.2.6
 1. [Grammar, Vocabulary, and Syntax](#) 1.2.6.1
 1. [Clarity and Ambiguity](#) 1.2.6.1.1
 2. [Artificial Languages](#) 1.2.6.1.2
 3. [High-level vs. Low-level Languages](#) 1.2.6.1.3
 2. [Idea to Execution](#) 1.2.6.2
 1. [Idea to Execution](#) 1.2.6.2.1
 7. [CODING SKILLS: Pseudocode](#) 1.2.7
 1. [Pseudocode](#) 1.2.7.1
 2. [Pseudocode for the AP Exam](#) 1.2.7.2
 8. [TOPIC: Solvability and Performance](#) 1.2.8
 1. [Decidability and Efficiency](#) 1.2.8.1
 2. [Moore's Law](#) 1.2.8.2
 9. [PROJECT: Password Generator Project](#) 1.2.9
 1. [Password Generator Project: Rubric Check](#) 1.2.9.1
3. [UNIT 2: Programming](#) 1.3
 1. [PROJECT: Scratch Programming Project](#) 1.3.1
 1. [Scratch Programming Project](#) 1.3.1.1

2. [**BIG PICTURE: The Who, What, and Why of Programming**](#) 1.3.2
 1. [Who Programs?](#) 1.3.2.1
 2. [Why Program?](#) 1.3.2.2
 3. [What is a Program?](#) 1.3.2.3
3. [**TOPIC: Visual Programming**](#) 1.3.3
 1. [Welcome to Scratch](#) 1.3.3.1
 1. [Personal Scratch Pages](#) 1.3.3.1.1
 2. [The Cat's Meow](#) 1.3.3.1.2
 3. [Save Early and Often](#) 1.3.3.1.3
 2. [Programming with Blocks](#) 1.3.3.2
 1. [Experimenting with "play" Blocks](#) 1.3.3.2.1
 2. [Different Ways to Broadcast](#) 1.3.3.2.2
 3. [Remixing Scratch Projects](#) 1.3.3.3
 1. [Remixing Scratch Projects](#) 1.3.3.3.1
4. [**CODING SKILLS: Choreography Notation**](#) 1.3.4
 1. [Let's Dance](#) 1.3.4.1
 2. [Choreography](#) 1.3.4.2
 3. [Animated Movie](#) 1.3.4.3
5. [**TOPIC: Program State**](#) 1.3.5
 1. [User Input](#) 1.3.5.1
 1. [Stored State](#) 1.3.5.1.1
 2. [User Input and Interaction](#) 1.3.5.1.2
 3. [Show Me Your State](#) 1.3.5.1.3
 4. [Text Input](#) 1.3.5.1.4
 2. [Variables](#) 1.3.5.2
 1. [Variables](#) 1.3.5.2.1
 2. [Names Are Important](#) 1.3.5.2.2
 3. [Game of Tag](#) 1.3.5.2.3
 4. [Randomness](#) 1.3.5.2.4
 5. [Custom Variables](#) 1.3.5.2.5
 6. [Drawing Commands](#) 1.3.5.2.6
 7. [Reviewing Variables](#) 1.3.5.2.7
6. [**TOPIC: Selection Statements**](#) 1.3.6
 1. ["if...else" Statements](#) 1.3.6.1
 1. [Decisions](#) 1.3.6.1.1
 2. [How Many Days...?](#) 1.3.6.1.2
 3. [Switching and Nesting](#) 1.3.6.1.3
 2. [Quiz Show](#) 1.3.6.2
 1. [Quiz Show](#) 1.3.6.2.1
7. [**PROJECT: Scratch Programming Project**](#) 1.3.7
 1. [Scratch Project: Documentation](#) 1.3.7.1
8. [**TOPIC: Repetition**](#) 1.3.8

1. [Repeat](#) 1.3.8.1
 1. [Repeat](#) 1.3.8.1.1
 2. [Repeat After Me](#) 1.3.8.1.2
 3. [Tempo](#) 1.3.8.1.3
 4. [Regular Polygon Generator](#) 1.3.8.1.4
 2. [Repeat Until](#) 1.3.8.2
 1. [Repeat Until](#) 1.3.8.2.1
 2. [Conditional Loops Compared](#) 1.3.8.2.2
 3. [Draw a "Spiral"](#) 1.3.8.2.3
 3. [Loops and Variables](#) 1.3.8.3
 1. [Loops and Variables Mini-Project](#) 1.3.8.3.1
 1. [Option I: Draw a Picture](#) 1.3.8.3.1.1
 2. [Option II: Electronic Keyboard](#) 1.3.8.3.1.2
 3. [Option III: Countdown](#) 1.3.8.3.1.3
 9. [CODING SKILLS: Rock, Paper, Scissors](#) 1.3.9
 1. [Rock, Paper, Scissors](#) 1.3.9.1
 10. [PROJECT: Scratch Programming Project](#) 1.3.10
 1. [Scratch Project: Rubric Check](#) 1.3.10.1
4. [UNIT 3: Data Representation](#) 1.4
 1. [PROJECT: Unintend'o Controller Project](#) 1.4.1
 1. [Unintend'o Controller Project](#) 1.4.1.1
 2. [TOPIC: Binary Encoding of Information](#) 1.4.2
 1. [Binary](#) 1.4.2.1
 1. [What Is Binary?](#) 1.4.2.1.1
 2. [Twenty Questions](#) 1.4.2.1.2
 3. [State Space](#) 1.4.2.1.3
 4. [High/Low Guessing](#) 1.4.2.1.4
 2. [Base Conversions](#) 1.4.2.2
 1. [The Amazing Binari](#) 1.4.2.2.1
 2. [Binary Finger Counting](#) 1.4.2.2.2
 3. [ASCII vs. Unicode](#) 1.4.2.3
 1. [Alphanumeric Representation](#) 1.4.2.3.1
 2. [Digital Scavenger Hunt](#) 1.4.2.3.2
 3. [Reading and Writing in ASCII](#) 1.4.2.3.3
 4. [Unicode vs. ASCII](#) 1.4.2.3.4
 3. [CODING SKILLS: Binary Birthday Cake](#) 1.4.3
 1. [Binary Birthday Cake](#) 1.4.3.1
 2. [Floating Point Numbers](#) 1.4.3.2
 4. [PROJECT: Unintend'o Controller Project](#) 1.4.4
 1. [Unintend'o Project: Binary Mapping](#) 1.4.4.1
 5. [TOPIC: Digital Approximations](#) 1.4.5
 1. [Digitization](#) 1.4.5.1

1. [Variable vs. Fixed-Width Encodings](#) 1.4.5.1.1
2. [Analog vs. Digital Data](#) 1.4.5.2
 1. [Approximating Physical Media](#) 1.4.5.2.1
 2. [Digital vs. Analog](#) 1.4.5.2.2
3. [Perfect Copies](#) 1.4.5.3
 1. [Digital Copies](#) 1.4.5.3.1
 2. [Perfect Imperfection](#) 1.4.5.3.2
6. [BIG PICTURE: Legality of Reselling Digital Music](#) 1.4.6
 1. [Reselling Digital Music](#) 1.4.6.1
7. [PROJECT: Unintend'o Controller Project](#) 1.4.7
 1. [Unintend'o Project: Programming](#) 1.4.7.1
8. [TOPIC: Lists](#) 1.4.8
 1. [Making a List](#) 1.4.8.1
 1. [Making a List](#) 1.4.8.1.1
 2. [Weird Cases in Lists](#) 1.4.8.1.2
 3. [Reading a List](#) 1.4.8.1.3
 2. [Processing a List](#) 1.4.8.2
 1. [Processing a List](#) 1.4.8.2.1
 2. [Index Variables](#) 1.4.8.2.2
 3. [Remove from a List](#) 1.4.8.2.3
 4. [Sentences as Lists](#) 1.4.8.2.4
 3. [Sorting a List](#) 1.4.8.3
 1. [Swaps](#) 1.4.8.3.1
 2. [Reorder](#) 1.4.8.3.2
 4. [Lists in Action](#) 1.4.8.4
 1. [MultiSet to Set](#) 1.4.8.4.1
9. [PROJECT: Unintend'o Controller Project](#) 1.4.9
 1. [Unintend'o Project: Rubric Check](#) 1.4.9.1
 2. [Unintend'o Project: Gallery Walk](#) 1.4.9.2
5. [UNIT 4: Digital Media Processing](#) 1.5
 1. [PROJECT: Image Filter Project](#) 1.5.1
 1. [Image Filter Project](#) 1.5.1.1
 2. [TOPIC: Procedural Programming](#) 1.5.2
 1. [Introduction to Processing](#) 1.5.2.1
 1. [Writing Code](#) 1.5.2.1.1
 2. [Scratch vs. Processing](#) 1.5.2.1.2
 3. [Scratch Constructs Revisited](#) 1.5.2.1.3
 4. [Punctuation](#) 1.5.2.1.4
 2. [Drawing](#) 1.5.2.2
 1. [Draw Shapes](#) 1.5.2.2.1
 2. [Draw a Figure](#) 1.5.2.2.2
 3. [Debugging with println\(\)](#) 1.5.2.2.3

3. [Mouse Interaction](#) 1.5.2.3
 1. [Movement](#) 1.5.2.3.1
 2. [Animate Your Figure](#) 1.5.2.3.2
4. [Keyboard Interaction](#) 1.5.2.4
 1. [Keyboard Input](#) 1.5.2.4.1
 2. [Loops](#) 1.5.2.4.2
3. [TOPIC: Image Manipulation](#) 1.5.3
 1. [RGB Color](#) 1.5.3.1
 1. [Calculating Colors](#) 1.5.3.1.1
 2. [Hexadecimal](#) 1.5.3.1.2
 3. [Color](#) 1.5.3.1.3
 2. [Raster Images](#) 1.5.3.2
 1. [Raster Images](#) 1.5.3.2.1
 2. [Eliminating Digital Noise](#) 1.5.3.2.2
 3. [Raster Image Manipulation](#) 1.5.3.3
 1. [Raster Image Manipulation](#) 1.5.3.3.1
 4. [Encoding Schemes](#) 1.5.3.4
 1. [Digital Image File Extensions](#) 1.5.3.4.1
 2. [Encoding Schemes](#) 1.5.3.4.2
 3. [Picture Logic](#) 1.5.3.4.3
 5. [Manipulating Digital Images](#) 1.5.3.5
 1. [Filters](#) 1.5.3.5.1
4. [TOPIC: Audio Manipulation](#) 1.5.4
 1. [Digital Audio](#) 1.5.4.1
 1. [Digitizing Audio](#) 1.5.4.1.1
 2. [Audio Generation](#) 1.5.4.1.2
 2. [Audio Processing](#) 1.5.4.2
 1. [Post-Processing Audio](#) 1.5.4.2.1
 2. [Audio Processing](#) 1.5.4.2.2
 3. [Audio Compression](#) 1.5.4.3
 1. [X Marks the Spot](#) 1.5.4.3.1
 2. [Compression Algorithms](#) 1.5.4.3.2
5. [BIG PICTURE: Ethics of Digital Manipulation](#) 1.5.5
 1. [Original or Manipulated](#) 1.5.5.1
 2. [Original or Manipulated \(Answers\)](#) 1.5.5.2
 3. [Ethics of Digital Manipulation](#) 1.5.5.3
 4. [Creative Commons](#) 1.5.5.4
6. [PROJECT: Image Filter Project](#) 1.5.6
 1. [Image Filter Project: Rubric Check](#) 1.5.6.1
6. [UNIT 5: Big Data](#) 1.6
 1. [PROJECT: TEDxKinda Project](#) 1.6.1
 1. [TEDxKinda Project](#) 1.6.1.1

2. [TOPIC: Data Science](#) 1.6.2
 1. [Introduction to Big Data](#) 1.6.2.1
 1. [What Is Big Data?](#) 1.6.2.1.1
 2. [Exploring US Employment Data](#) 1.6.2.1.2
 2. [Usability and Usefulness of Data](#) 1.6.2.2
 1. [Usability vs. Usefulness](#) 1.6.2.2.1
3. [PROJECT: TEDxKinda Project](#) 1.6.3
 1. [TEDxKinda Project: Topics](#) 1.6.3.1
 2. [TEDxKinda Project: Big Data Sets](#) 1.6.3.2
 3. [TEDxKinda Project: Tools](#) 1.6.3.3
4. [TOPIC: Data Aggregation](#) 1.6.4
 1. [Collection](#) 1.6.4.1
 1. [Big Data Collection](#) 1.6.4.1.1
 2. [Creating Structure from Unstructured Data Sets](#) 1.6.4.1.2
 3. [Digitizing Business Cards](#) 1.6.4.1.3
 2. [Extraction](#) 1.6.4.2
 1. [The Internet's Data Structure](#) 1.6.4.2.1
 2. [Spiderbots](#) 1.6.4.2.2
 3. [Fetching Flutter-bys](#) 1.6.4.2.3
 3. [Storage](#) 1.6.4.3
 1. [Indexing Julius Caesar](#) 1.6.4.3.1
 2. [Data Persistence](#) 1.6.4.3.2
 3. [Your Filter Bubble](#) 1.6.4.3.3
 4. [Privacy vs. Utility](#) 1.6.4.3.4
5. [BIG PICTURE: Data Breaches](#) 1.6.5
 1. [Data Breaches](#) 1.6.5.1
 2. [My Data Rules](#) 1.6.5.2
6. [TOPIC: Data Analysis](#) 1.6.6
 1. [Statistical Analysis](#) 1.6.6.1
 1. [Statistical Analysis](#) 1.6.6.1.1
 2. [Justin Who?](#) 1.6.6.1.2
 2. [Data Mining](#) 1.6.6.2
 1. [Data Mining](#) 1.6.6.2.1
 2. [Association Rule Mining](#) 1.6.6.2.2
 3. [Clustering](#) 1.6.6.3
 1. [TEDxKinda Project: Identifying Clusters](#) 1.6.6.3.1
 4. [Anomaly Detection](#) 1.6.6.4
 1. [Outliers](#) 1.6.6.4.1
 2. [TEDxKinda Project: Identifying Outliers](#) 1.6.6.4.2
 5. [Regression](#) 1.6.6.5
 1. [TEDxKinda Project: Making Predictions](#) 1.6.6.5.1
 6. [Classification and Summarization](#) 1.6.6.6

1. [Classify Me](#) 1.6.6.1
 2. [TEDxKinda Project: Automated Summarization](#) 1.6.6.2
7. [TOPIC: Data Visualization](#) 1.6.7
 1. [Interactive Infographics](#) 1.6.7.1
 8. [BIG PICTURE: Wisdom of the Crowd](#) 1.6.8
 1. [ReCAPTCHA](#) 1.6.8.1
 2. [Crowdsourcing](#) 1.6.8.2
 9. [PROJECT: TEDxKinda Project](#) 1.6.9
 1. [TEDxKinda Project: Rubric Check](#) 1.6.9.1
7. [UNIT 6: Innovative Technologies](#) 1.7
 1. [PROJECT: Future Technology Project](#) 1.7.1
 1. [Future Technology Project](#) 1.7.1.1
 2. [TOPIC: Everyday Computing](#) 1.7.2
 1. [Social Networking and Communication](#) 1.7.2.1
 1. [Social Networking](#) 1.7.2.1.1
 2. [Models of Sharing](#) 1.7.2.1.2
 2. [Search, Wikis, Commerce, and News](#) 1.7.2.2
 1. [Essential Services](#) 1.7.2.2.1
 2. [Search](#) 1.7.2.2.2
 3. [Wikis](#) 1.7.2.2.3
 4. [Commerce](#) 1.7.2.2.4
 3. [Cloud Computing](#) 1.7.2.3
 1. [Cloud Computing](#) 1.7.2.3.1
 2. [Ownership of Cloud Data](#) 1.7.2.3.2
 4. [The Digital Divide](#) 1.7.2.4
 1. [The Digital Divide](#) 1.7.2.4.1
 3. [TOPIC: The Internet](#) 1.7.3
 1. [Network Infrastructure](#) 1.7.3.1
 1. [Network Infrastructure](#) 1.7.3.1.1
 2. [Communication Protocols](#) 1.7.3.1.2
 3. [Internet Protocol](#) 1.7.3.1.3
 4. [Domain Name System](#) 1.7.3.1.4
 2. [World Wide Web](#) 1.7.3.2
 1. [World Wide Web](#) 1.7.3.2.1
 4. [BIG PICTURE: Net Neutrality](#) 1.7.4
 1. [Net Neutrality](#) 1.7.4.1
 5. [TOPIC: Innovations in Computing](#) 1.7.5
 1. [Internet of Things](#) 1.7.5.1
 1. [Internet of Things](#) 1.7.5.1.1
 2. [Distributed Computing](#) 1.7.5.2
 1. [Distributed Computing](#) 1.7.5.2.1
 3. [Ethics of Autonomous Technology](#) 1.7.5.3

1. [Ethics of Autonomous Technology](#) 1.7.5.3.1
6. [PROJECT: Future Technology Project](#) 1.7.6
 1. [Future Technology Project: Rubric Check](#) 1.7.6.1
8. [Performance Tasks](#) 1.8
 1. [AP CSP Course Assessment](#) 1.8.1
 2. [The "Explore" Performance Task](#) 1.8.2
 3. [The "Create" Performance Task](#) 1.8.3
9. [UNIT A1: Artificial Intelligence: Turing Test](#) 1.9
 1. [Artificial Intelligence: Turing Test Project](#) 1.9.1
 2. Chatterbots 1.9.2
 1. [What is a chatterbot?](#) 1.9.2.1
 2. [Black-Box Testing Chatterbots](#) 1.9.2.2
 3. The “Humanity” of AI 1.9.3
 1. [Consider L'il Johnny McPixel](#) 1.9.3.1
 2. [L'il Johnny McPixel's Humanity](#) 1.9.3.2
 4. [Artificial Intelligence: Turing Test Project Draft](#) 1.9.4
 5. Programming Intelligence 1.9.5
 1. [Creating Intelligence](#) 1.9.5.1
 2. [Artificial Intelligence: Turing Test Experiment](#) 1.9.5.2
 6. Beyond Text 1.9.6
 1. [Experimenting with Visual Identification](#) 1.9.6.1
 2. [Multi-modal Approaches](#) 1.9.6.2
 3. [Dasher](#) 1.9.6.3
 7. Difficulties with Text 1.9.7
 1. [Ambiguity](#) 1.9.7.1
 2. [Ambiguity Rocks](#) 1.9.7.2
 8. Artificial Intelligence: Project 1.9.8
 1. [Artificial Intelligence: Turing Test: Trials](#) 1.9.8.1

UTeach CS Principles

Welcome to UTeach Computer Science Principles



<https://www.youtube.com/embed/DTF3fM--XMO>

Throughout this course, you will be introduced to this amazing world and the many ways that computer science has helped to shape nearly every aspect of your life. Whether it is the cell phone in your pocket, the game console connected to your television, the self-checkout register at the store, the robot-assisted surgery that saves your life, or the self-driving car that brings you to school, we are surrounded by the products made possible by centuries worth of technological advances in math, science, logic, and design.

Ever since the dawn of mankind, humans have striven to develop the knowledge and skills needed to harness the varied resources of our world. With those resources, we have created tools and other technological artifacts that have allowed us to manage the challenges and complexities of life. Whether it was spears and stone axes to help us hunt or ink and paper to help us record our thoughts for all posterity, technological advances have shaped the ways that individuals have related to the world around them and enabled us to achieve the seemingly impossible.

By the mid-20th century, these advances began to take on a new arena and the age of digital computing was born. The advent of the modern computer and its modular design, stored-program architecture, and ability to perform complex computations with both speed and accuracy opened the doors to a wealth of incredible innovations and fundamentally changed the way that we interact with others and the world around us. It is truly a remarkable time to live in, both as a consumer of digital computing and as an innovator of the new ideas that will shape tomorrow's digital world.

innovation

The process of imagining something that does not yet exist, but that

has potential value, and making it real through the application of design, implementation, and production.

Over the next several units, you will be introduced to a number of innovations in computing and digital media that have come to form the backbone of nearly all of our online and offline interactions. As you explore each new component of this digital landscape, you should get in the habit of asking yourself questions like the following:

- *What does this innovation allow me to do that I couldn't do before?*
- *How can I use this innovation to create something that nobody has ever imagined before?*
- *How can I use this innovation to create even better innovations of my own?*

As this course guides you through the art and science of digital technology and helps you develop robust, computational thinking skills, you will become a master of innovation and be fully prepared to thrive in our digital world!

UNIT 1

Computational Thinking

In order to successfully master the art of creating computational artifacts, it is important that students develop a clear understanding of the complex processes and structures that make up an algorithmic solution to a given problem. In addition, it is critical that they be able to formally express those solutions clearly and unambiguously, such as what can be achieved through the use of pseudocode or a well-specified programming language. This unit focuses on introducing students to these concepts and helping them to develop the skills that they will rely on throughout the remainder of the course.

First, students will explore a number of techniques for analyzing common problems and visualizing their solutions. They will use these techniques to investigate a number of real-world applications, such as searching, sorting, and encryption. Next, students will examine how programmers utilize various levels of abstraction in the languages that they use to write programs and communicate their intentions in a form that can be executed by a computer. Finally, students will turn their attention to the question of whether various problems are solvable and investigate the factors that affect the efficiency of a solution to a given problem.

UNIT PROJECT:

Password Generator

Highlights

- You will collaborate in pairs to design an algorithm for generating a custom, reproducible password that is uniquely different for each website (e.g., using the domain name as a seed, etc.).
- You will write pseudocode to describe each step of the algorithm used to generate a password.
- You will exchange algorithms with your peers and share feedback with each other on the clarity of the pseudocode and the strengths and weaknesses of your algorithms.
- You will construct trace tables documenting the result of each step of the algorithm in generating passwords for different domains.
- You will write about the dangers of reusing passwords across multiple websites and online services and how such behavior may be exploited.

Password Generator Project

"Choosing a hard-to-guess, but easy-to-remember password is important!" - Kevin Mitnick



<https://www.youtube.com/embed/qpxLusH4quY>

Password Vulnerabilities

In March 2013, the popular online note-taking service, Evernote, issued a [Security Notice](#) alerting users of a "service wide password reset" that they were enfoing as a result of a "coordinated attempt to access secure areas of the Evernote Service."

Fortunately, no personal information or data was breached and that by issuing a required password reset, Evernote was merely taking proactive steps in an abundance of caution.

However, such widespread password leaks and security breaches are becoming all too common as online services play a larger and larger role in our digital lives. Several times every year, the news reports of yet another hack of a popular site or a leak of passwords and other sensitive user data.

In its announcement, Evernote offered its users advice to follow that could help to ensure their data on *any* site.

Advice from an Evernote [Security Notice](#) (March 2, 2013):

- *Avoid using simple passwords based on dictionary words*
- *Never use the same password on multiple sites or services*
- *Never click on 'reset password' requests in e-mails—instead go directly to the service*

The first two bullet points on this list relate to the two most likely mistakes that users make when choosing a password. They relate to the dilemma of trying to balance the security of complex passwords with the convenience of having a single, easy-to-remember password.

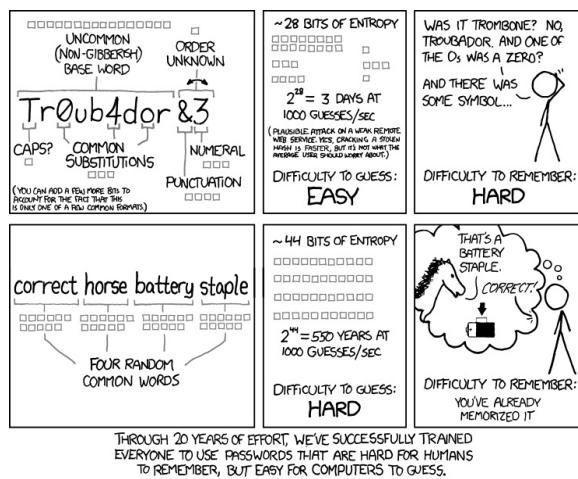
Unfortunately, despite their concerns over privacy, people often choose convenience over security. Let's take a closer look at two ways that users often leave themselves vulnerable to attack.

Easy to Remember, Hard to Guess

"Any password that can be easily remembered is vulnerable to a dictionary attack." — Bruce Schneier

The goal of any password should be to choose something that is difficult to guess, especially when some automated systems are capable of making millions of guesses per second. People use a number of techniques to increase the complexity of their passwords, such as mixing upper- and lowercase letters, substituting digits and punctuation for letters, appending extra characters or numbers, etc. Unfortunately, while these efforts might increase the effort required to guess them, it also increases the difficulty of remembering them.

A popular [xkcd](#) comic by Randall Munroe addressed the issue of how difficult it is to crack a password vs. how easy it is to remember by attempting to measure password strength in terms of "bits of entropy."



Unfortunately, as security expert Bruce Schneier notes in his article, "[Choosing Secure Passwords](#)," despite Munroe's logic, his suggested solution is actually quite vulnerable to attack due to its reliance upon common *dictionary words* that are easy to guess by brute force.

So, just how does one create a password that is strong but memorable? What steps could you take to create a secure password that is convenient for you to use? Discuss some ideas with your partner and be prepared to share your suggestions with the class.

Single Point of Failure

Another common mistake that people frequently make with their password security was also addressed by Evernote's advice to their users — "Never use the same password on multiple sites or services."

There are many reasons why users might reuse the same password for every site or service they visit, with the most obvious being that it is simply easier to remember *one* password than it is to remember *many* different passwords. It is common sense. But it is also highly insecure.

The problem is that a password is meant to be a secret credential that you use to identify yourself to someone else. This method of authentication relies on the basic assumption that there is a one-to-one relationship between knowing the password and having the right to access an account. That is, in theory, only the owner of an account can provide the secret piece of information (i.e., the password) that can confirm the individual's identity. However, if any other, third party should ever possess that secret information, then the initial assumption fails as knowing the password is no longer guaranteed to be limited to the account owner.

The real problem lies in the fact that in order for someone else (e.g., Facebook, Amazon, your bank) to be able to authenticate you, they need to know what the secret piece of information is that only you can tell them — which *itself* violates the first assumption and invalidates your use of that password with any other party.

For example, imagine you sign up with Facebook and set your password to be **fuzzybunny123**. Now you know your password. And Facebook knows your password. So far, so good. Whenever you visit Facebook, you can prove to the site that you are you by identifying yourself with the **fuzzybunny123** password that only you and Facebook know. However, you also sign up for an account with some other social sharing site, like *InstaChatOmatic*, and you use the same **fuzzybunny123** password that you use with Facebook. Now, three different parties all know the same information that, in theory, only *you* should know. A malicious employee, rogue software, or database leak at InstaChatOmatic can result in an unknown party attempting to access your Facebook account using your InstaChatOmatic password. And since the passwords are the same, Facebook will authenticate the *intruder* as if it is *you* because they will have provided that information that supposedly only *you* should have known.

In short, by reusing the same password with more than one service, you've undermined the security of your password at both sites — all in the name of making it easier for *you* to remember. This is flawed thinking. Passwords are meant for security and the strength of that security should be prioritized above anything else that undermines it.

Remember Algorithms, Not Passwords

In his article, "[Passwords Are Not Broken, but How We Choose them Sure Is.](#)" Schneier suggests what he calls the "Schneier Scheme":

"My advice is to take a sentence and turn it into a password."

Something like 'This little piggy went to market' might become t1pWENT2m. That nine-character password won't be in anyone's dictionary. Of course, don't use this one, because I've written about it. Choose your own sentence - something personal." — Bruce Schneier

The important thing to note about this approach is that rather than trying to remember an obscure collection of odd and difficult to guess letters, digits, and punctuation, one needs only to remember a personalized phrase or other mnemonic that will remind you how to easily reconstruct the password.

And if you customize the key phrase to match the site or service, a single, simple set of rules can be created that will allow you to easily reconstruct the password anytime you visit the site. For example, consider the following phrase/password scheme:

- "Facebook is where I post to my friends." ... FB8pstBFSS
- "Gmail is where I read my mail." ... GM5rdMAIL
- "Twitter is where I follow my friends." ... TW7f11wBFFS
- "YouTube is where I watch videos." ... YT7wtchVIDEOS

The single algorithm that you would need to memorize for this scheme would be something like the following:

- 1) Abbreviate the site into a 2-letter phrase.
- 2) Capitalize the site abbreviation.
- 3) Type the site abbreviation.
- 4) Type the number of letters in the site name.
- 5) Identify the verb that describes how you use the site.
- 6) Remove all vowels from the verb.
- 7) Type the vowel-less verb in lowercase letters.
- 8) Identify the subject or type of content for the site.
- 9) Capitalize the subject or type of content.
- 10) Type the capitalized subject or type of content.

What are the advantages to this particular algorithm? Are there any problems or weaknesses to this algorithm? What other algorithms can you think of that might work better? Be prepared to discuss your ideas with the class. (Of course, if you have a *really* clever idea, you might want to save that one for yourself and only discuss your rejected ideas.)

Assignment

Design an algorithm that can generate a custom, reproducible password that is uniquely different for each website.

Working in pairs, your task is to design and construct a standardized strategy for generating unique passwords for different sites that can later be regenerated by reapplying the same algorithm. Your solution should address the following concepts:

- The algorithm should generate different passwords for different sites.
- The password for any site should be reproducible simply by following the algorithm.
- The algorithm should be easy to remember and apply.
- The password should be complex and difficult to guess.
- The general algorithm should not be easily deduced from the password.

Once you've designed your solution, write out each step of your password-generating algorithm in some form of *pseudocode*. No specific format is required for your algorithm, but your pseudocode should be clear enough and detailed enough that anyone who is not familiar with how your algorithm is supposed to work can still follow along and apply its steps in generating a valid password.

Submission

Your submission will be in the form of a written algorithm (i.e., pseudocode) that explicitly states each of the discrete steps and decisions that must be made in generating a valid password. Also, you must provide at least five examples of passwords that your algorithm would generate for five different sites. One of those examples must be thoroughly annotated, showing how each step of the algorithm contributes to the final password.

Your solution and examples should demonstrate the following properties:

- Clear and readable
- Cleanly formatted
- Appropriate use of sequencing, selection, and/or iteration
- Well-documented examples

Learning Goals

Over the course of this module and this project, you will learn to:

- identify and examine a number of common features of algorithms, including sequencing, selection, and repetition
- write pseudocode to describe each step of an algorithm with clarity and precision
- construct trace tables documenting the result of each step of an algorithm
- compare the differences between different types of common algorithms
- analyze the need for artificial programming languages
- examine strategies for approaching large-scale problems
- identify factors that allow solutions to scale efficiently
- encode and decode messages using common cryptographic techniques
- examine a number of common threats to cybersecurity, including distributed denial of service attacks (DDoS), phishing, viruses, and social engineering
- examine the implications of Moore's Law on the research and development of new and

existing technologies

Rubric

Content Area	Performance Quality			
Readability	10 Algorithm is typed, organized, and nicely formatted for easy use.	7 Algorithm is organized and nicely formatted for easy use, but is not typed. —OR— Algorithm is typed, but the formatting and organization makes it somewhat difficult to use.	4 Algorithm has formatting and organization that makes it somewhat difficult to use AND is not typed. —OR— Algorithm may be typed, but the formatting and organization makes it extremely difficult to use.	0 Not enough criteria are met in order to award any credit.
Flow	5 The algorithm incorporates the appropriate use of all three types of programming structure: sequencing, selection, and iteration.	3 The algorithm incorporates the appropriate use of only two types of programming structure: sequencing, selection, and iteration.	1 The algorithm incorporates the appropriate use of only one type of programming structure: sequencing, selection, and iteration.	0 Not enough criteria are met in order to award any credit.
Correctness	10 The algorithm generates a unique and reproducible password for all sites.	7 The algorithm generates a reproducible password for all sites, however, some may not be unique. —OR— The algorithm generates a unique and reproducible password for most sites. —OR— The algorithm generates a unique password for all sites, however, it is	4 The algorithm generates a password for all sites, however, some may not be unique or reproducible. —OR— The algorithm generates a unique and reproducible password for only a few sites.	0 Not enough criteria are met in order to award any credit.

		not reproducible.		
Effectiveness	5 The algorithm cannot be easily deduced from just the password and the name of the site.	3 A few parts of the algorithm can be easily deduced from just the password and the name of the site.	1 Most parts of the algorithm can be easily deduced from just the password and the name of the site.	0 Not enough criteria are met in order to award any credit.
Examples	10 There are five sample passwords generated correctly based on the algorithm.	7 There are four sample passwords generated correctly based on the algorithm.	4 There are three or fewer sample passwords generated correctly based on the algorithm.	0 Not enough criteria are met in order to award any credit.
Documented Case	10 There is one annotated example documented at all steps of the process. —AND— It is well formatted and organized and easy to follow.	7 There is one annotated example documented at most steps of the process AND It is well formatted and organized and easy to follow. —AND— There is one annotated example documented at all steps of the process, but the organization and formatting makes it difficult to follow.	4 There is one annotated example documented at some steps of the process AND It is well formatted and organized and easy to follow. —OR— There is one annotated example documented at all steps of the process, but the organization and formatting makes it extremely difficult to follow. —OR— There is one annotated example documented at most steps of the process, but the organization and formatting make it difficult to follow.	0 Not enough criteria are met in order to award any credit.
TOTAL				50 pts

UNIT TOPIC:

Algorithmic Thinking

Problem Solving

- You will examine strategies for approaching large-scale problems.
- You will explore the benefits and applications of employing a top-down approach to problem solving.
- You will explore the benefits and applications of employing a bottom-up approach to problem solving.

Flow Patterns

- You will identify and examine a number of common features of algorithms, including sequencing, selection, and repetition.

UNIT TOPIC:

Algorithmic Thinking

Problem Solving

- You will examine strategies for approaching large-scale problems.
- You will explore the benefits and applications of employing a top-down approach to problem solving.
- You will explore the benefits and applications of employing a bottom-up approach to problem solving.

Building Blocks

People Programs

One of the major goals of this unit is to help you develop a better understanding of the methods and techniques computer scientists rely on to construct simple and elegant solutions to potentially complex problems.

Today, in order to give you a feel for what this is like, you'll be participating in a group exercise in which you'll write and execute your first program. However, most of you have not learned any programming languages yet, so for now, we'll stick with using a language that everybody in your group likely does know — English. And instead of executing your programs on an actual computer (which probably wouldn't understand English as well as people do), you and your group will role-play the parts of a simulated computer as you attempt to execute your program in much the same way a real computer would run a real program. And if programs that are run by computers are called *computer programs*, then it only seems appropriate that our programs today that are run by people should be called *people programs*.

Instructions

Working in groups of four, each student should assume one of the roles listed below. Each group will be given a construction site (grid with directions and "X," "A," and "B" spaces marked), a bag of building blocks, and a set of criteria describing what they are to build. Each group should write/type a list of detailed instructions that the Supervisor can read off to the Supplier, Worker, and Inspector so that the required set of towers is properly constructed.

Roles

Supervisor: This is the "construction foreperson" whose job it is to follow the steps in the plan exactly, ask "Yes/No" questions of the Inspector, make decisions when appropriate, and tell the Worker and Supplier what to do. The Supervisor can be thought of as being analogous to the "processor" within a computer.

- Can only read the instructions exactly as they are written.
- Cannot say anything else.
- Cannot see the blocks, site, or other crewmembers while construction is in progress.
- Can instruct him/herself to make notes, count, or remember things about the progress that has been made.

Supplier: This individual is responsible for supplying and disposing of the raw materials (i.e., blocks) for construction. Think of the Supplier as an "input/output device" like a keyboard, mouse, or computer display.

- When instructed by the Supervisor, can randomly remove a single block from the supply bag and place it on the "Loading Zone" (X).
 - When instructed by the Supervisor, can remove a single block from the "Loading Zone" (X) and randomly place it back into the supply bag.
-

Worker: This individual is solely responsible for manipulating, moving, and placing the construction materials into their designated locations. The Worker can be seen as performing the role of a "bus" that transfers data between components inside a computer.

- Can only touch one block at a time.
 - Can only manipulate blocks that are on the construction site (i.e., the grid).
 - Must follow the Supervisor's instructions exactly.
 - Knows about the grid (including direction, distance, and the three marked locations — A, B, and X).
 - May NOT use any personal judgment in selecting, manipulating, or placing blocks. The Supervisor's instructions should be clear enough.
-

Inspector: This is the "answer person" of the construction team who is responsible for giving the Supervisor essential feedback about materials and the status of constructions. The Inspector functions like a computer's "logic unit" (part of the ALU — Arithmetic Logic Unit).

- When asked a question by the Supervisor, can only answer "YES" or "NO" about the block(s) in one or both of the two "Inspection Areas" (A, B).
 - Cannot answer questions about anything not in one of the two "Inspection Areas" (A, B).
 - May not communicate anything else.
 - If at any point, a question cannot be answered strictly "YES" or "NO," then the question is considered faulty. Stop the construction process, go back to the drawing board, and rethink the set of instructions your team has written for the Supervisor.
-

Problem Solving

The Big Problem

From the time you wake up in the morning to the time you go to sleep at night, you face problems of one sort or another throughout the day. Whether it's the challenge of dragging yourself out of a cozy bed when you just want to keep sleeping, deciding what to wear for the day, making sure you arrive at each of your classes on time and with the proper materials, completing your assigned homework, or carrying on a texting conversation with your best friend, *everything* you do can be seen as a *problem* that you must *solve*.

Every task that you deal with on a day-to-day basis, no matter how large or small, requires that you perform a specific set of actions and make a particular set of decisions in very precise ways. And if you manage to perform the right sequence of steps in just the right order, you'll succeed in achieving what you set out to accomplish. That is, you'll have solved your problem.

Executing Solutions

For example, the very act of reading this paragraph right now is a problem that you are currently solving, likely without even realizing that you're solving a problem. Just a few moments ago, you had no idea what this paragraph was about, what it says, or what you're supposed to be taking away from it. But as you started reading it, your mind began subconsciously following an algorithm that you were taught many years ago for solving just this very type of problem. First, your eyes focused on the upper left-hand corner of the block of text (i.e., where it says "For example"). Then your eyes scanned left to right across the text, noting the spaces that separated the various words and then decoding the meaning of each of those words as you encountered it. When your eyes reached the right-hand margin of a line of text, they dropped down to the next row and quickly repositioned themselves back on the left-hand margin. You then repeated this left-to-right scanning/decoding process line by line until you reach the end of the paragraph. And when you reach that point, without even thinking about *how* you were doing it, you will have actually solved the "big problem" of reading and, hopefully, comprehending this paragraph. Congratulations!

Generalizing Problems

Also note that the algorithm you learned for reading the above paragraph (as well as this and every other paragraph) was a generalized algorithm. It wasn't specific to the paragraph above. Instead, that single algorithm of scanning through the text of a paragraph word-by-word and line-by-line is perfectly applicable to *any* paragraph. Imagine how difficult life would be if it only worked for a single paragraph of text and that reading every other paragraph required its own unique set of instructions.

Instead, the mark of a good problem solver is the ability to use abstraction to generalize a variety of related problems and develop a common solution that can be applied to solve any

of them. Much of computer science involves identifying these generalized solutions, codifying them in a programming language, and then letting a computer do all the work of executing your plan.

Breaking it Down

Big problems are *BIG!* They're hard. They're challenging. They can be overwhelming. If they weren't, we wouldn't call them "big problems." But, how does one solve a "big problem"?

The easy answer is that you don't. **DO NOT TRY TO SOLVE A BIG PROBLEM!** At least not *directly*.

Smaller problems are much easier to handle. They're simple. They're easy. They're trivial. Or at least they can be if you just make them small enough.

In fact, **the best way to solve a big problem** is to recognize that every *big* problem is really just a combination of many *smaller* problems. And those smaller problems, themselves, might be made up of even *smaller* problems that can be broken down even further until all you're left with are a bunch of trivially simple problems that you already know how to solve.

And here's the important thing to realize: If you can solve all of the small problems, you'll have solved the big problem!

Building Towers

The challenge is to take a good, long look at your big problem and try to reimagine it as a series of smaller problems. More than likely, you and your group members intuitively used some very clever problem-solving strategies despite whether you were aware of it.

For example, in the previous "Building Blocks" activity, you were tasked with building a series of precisely located towers, each with a specific set of properties. Building multiple towers simultaneously (big problem) is much more challenging than building a single tower (small problem). So, we can redefine the original problem (hard) as being equivalent to building a sequence of individual towers (easy).

Big Problem	Redefined Problem
Build N towers	Build 1st tower
	Build 2nd tower
	...
	Build Nth tower

But even though building a single tower is an easier problem to solve than building multiple towers, it's still a pretty big problem in and of itself. Perhaps we can break it down into even smaller problems.

Big Problem	Redefined Problem
Build a tower of M blocks	Place 1st block

	Place 2nd block
	...
	Place Mth block

For each tower we want to build, we can think of that as merely placing a series of properly oriented blocks. But how might we break down the task of placing a block?

Big Problem	Redefined Problem
Place a block	<i>Supply a block to X</i>
	Orient a block
	<i>Move a block into position</i>

For each block, we only need to supply it, orient it, and move it into its final position. But "Supply a block to X" is one of the two actions the Supplier already knows how to do! And moving a block to a desired location is something the Worker already knows how to do! Two of the three steps to "Place a block" involve built-in functionality and there's no need to redefine either of those steps any further. But, unfortunately, neither the Supplier, Worker, nor Inspector already knows how to "Orient a block," so that's still a "big problem" — break it down further!

Big Problem	Redefined Problem
Orient a block	<i>Move a block from X to A</i>
	Determine the current orientation of a block
	<i>Turn a block around a specific axis</i>

Again, the first and third steps of this operation involve manipulating the block, which is built-in functionality that the Worker can already do. But in order to "Determine the current orientation of a block," we'll need to ask a number of carefully chosen questions.

Big Problem	Redefined Problem
Determine current orientation	<i>Ask 1st Y/N question</i>
	<i>Ask 2nd Y/N question</i>
	...
	<i>Ask Nth Y/N question</i>

And since the Inspector already knows how to answer each of these Y/N questions, we can stop here. There's no need to break this big problem down any further. Instead, we can start constructing a solution.

For example, depending on the particular building criteria, the above analysis and redefinition of each problem into simpler tasks might result in a typical solution similar to the

following:

```
For each of the N towers...
  ...For each of the M blocks in a tower...
    ....SUPPLIER: Supply a block to X
    ....WORKER: Move a block from X to A
    ....INSPECTOR: Is the [TOP] face [RED]?
    ....If INSPECTOR said YES..
      ....WORKER: Turn the block on the [N/S] axis
    ....If INSPECTOR said NO..
      ....WORKER: Turn the block on the [E/W] axis
    ....WORKER: Move a block from A to [location]
```

UNIT TOPIC:

Algorithmic Thinking

Flow Patterns

- You will identify and examine a number of common features of algorithms, including sequencing, selection, and repetition.

Flow Patterns

Familiar Patterns

The algorithmic solutions developed by programmers and problem solvers can take on a virtually infinite number of diverse forms. They can be brief, requiring only a few operations to implement or they can be massive and complex, spanning millions of lines of code and a team of programmers to keep it all straight. But despite this range of diversity, all of these algorithms are characterized by the same basic elements of **sequencing**, **selection**, and **iteration**.

A good problem solver can learn to recognize these three features and easily use them to construct simple, effective, and elegant solutions to the problems they might want/need to solve.

With repeated practice throughout this course, you'll develop a trained eye that can quickly recognize each of these common patterns within your everyday actions. In fact, if you stop to take a closer look at a few of the common, everyday tasks that you perform all the time, you'll likely discover that you already make use of these patterns without even noticing.

In the next few units, we'll see that all programming languages are designed to make it easy for programmers to write code that uses these three basic patterns.

Sequencing

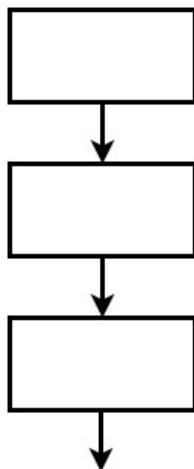
The simplest of all of the patterns consists of each step of an algorithm following the previous step. When the steps of an algorithm are sequentially arranged, each instruction is executed one at a time in the order specified by the algorithm.

- 1) put on pants
- 2) put on left shoe
- 3) put on right shoe
- 4) put on shirt

One of the most important things to recognize about the sequencing of instructions is that it enables one to explicitly state the order in which certain instructions are executed.

Depending on the situation, sometimes an instruction needs to precede one or more other instructions in the same algorithm. This is often the case if a "later" step is dependent upon the successful completion of an "earlier" step.

In the above example, `put on pants` is sequenced to come before `put on left shoe` and `put on right shoe` which makes sense, since it would be difficult to put on pants *after* putting on shoes.



However, just because sequencing imposes an order on which instructions are executed doesn't mean that order is important or even necessary.

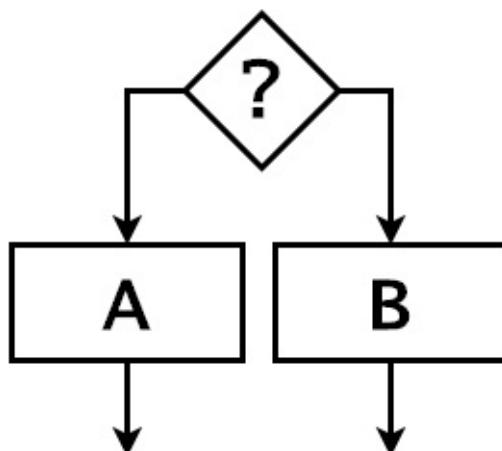
Again, in the previous example, the order of `put on left shoe` versus `put on right shoe` probably doesn't matter. That is, the algorithm could probably work just as well with the right shoe preceding the left shoe. Similarly, `put on shirt`, which is currently the last step, could probably have been placed anywhere in the algorithm, either before or after the pants and shoes, without affecting its correctness.

Selection

Algorithms that consist only of sequencing will always perform exactly the same, regardless of external conditions, which is often good for designing specific solutions to specific problems.

However, most problems would benefit from a more generalized solution that can dynamically adapt to changes in conditions. More specifically, problems that handle variable data, such as user-supplied input, oftentimes need to execute different sequences of instructions depending on the particular conditions.

In these "conditional" situations, algorithms need a way to *select* which sequence of instructions should be executed and which should not.



The classic metaphor used for this type of situation is an "if... else..." sort of structure.

Literally (since most languages actually use the keyword `if`), the algorithm asks a question (specifically a "YES/NO" question) and the answer to that question is used to determine which path of the algorithm to execute and which path of the algorithm to skip.

- 1) `if wearing pants...`
- 2) `...put on left shoe`
- 3) `...put on right shoe`
- 4) `else...`
- 5) `...put on pants`

In the above example, the `if` condition at Step 1 creates a fork in the algorithm that enables one of two possible sequences of instructions to be executed. If the condition `wearing pants` is *true* (i.e., the pants have already been put on at some earlier time in the algorithm), then it's safe to put on one's shoes. In that case, Steps 2 and 3 (`put on left shoe` and `put on right shoe`) will be executed while Step 5 (`put on pants`) will be skipped. However, if the condition was *false* (i.e., not wearing pants), then someone better put on some pants! In that case only Step 5 will be executed while Steps 2 and 3 will be skipped.

It's important to note that the nature of an `if/else` question, or *condition*, always allows for only two possible answers (e.g., yes/no, true/false, etc.) because each answer is used to select one of the two specified sequences – the first sequence/branch if, and only if, the condition is *true* or, alternatively, the second sequence/branch if, and only if, the condition is *false*.

If more than two branches are ever needed, they can be achieved by "nesting" multiple if/else clauses together. That is, one or both of the two branches can contain its own internal if/else clause. For example:

- 1) `if wearing pants...`
- 2) `...if wearing shoes`
- 3) `...put on shirt`
- 4) `...else...`
- 5) `...put on left shoe`
- 6) `...put on right shoe`
- 7) `else...`
- 8) `...put on pants`

This example tests for and handles three different situations:

1. If you're already wearing pants and shoes... put on your shirt.
2. If you're already wearing pants, but no shoes... put on your shoes.
3. If you're already not wearing pants... put some pants on ASAP!

Note that in the first and second situations, Step 8 will be ignored completely because of the pants-wearing status. Then, depending on the shoes-wearing status, either Step 3 will be executed or Steps 5 and 6 will be executed. Likewise, in the third situation (no pants!), the

condition about shoes is irrelevant as Step 2 (and its subsequent Steps 3-6) will be ignored.

Iteration

The third common pattern that you'll see in most algorithms, especially with large and complex solutions, is *repetition* – more properly referred to as *iteration*.

For example, consider the very simple act of hammering a nail into a block of wood:

- 1) grab a hammer
- 2) raise the hammer over your head
- 3) aim for the head of the nail
- 4) swing hammer down onto the head of the nail

- 5) raise the hammer over your head
- 6) aim for the head of the nail
- 7) swing hammer down onto the head of the nail

- 8) raise the hammer over your head
- 9) aim for the head of the nail
- 10) swing hammer down onto the head of the nail

- 11)
- ...
- N-1)

- N) put down the hammer

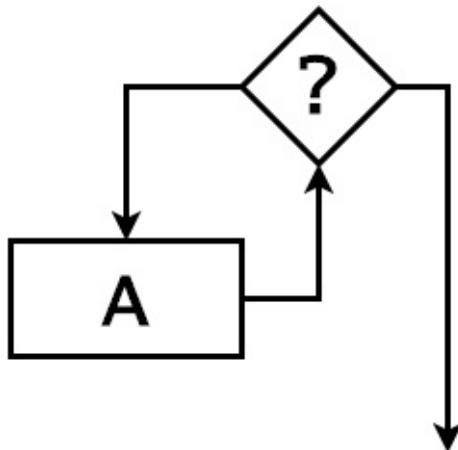
When you look at it this way, the repetition becomes immediately obvious. Steps 2–4 are repeated again as Steps 5–7 and Steps 8–10, etc. Each of these repeated sections of steps is called an *iteration*. If we were to continue this algorithm, what do you think Steps 11–13 will be? What about Steps 14–16? 17–19? How many iterations will we need? When will these instructions stop? When *should* they stop?

And more importantly, if we ever needed to actually write out these detailed instructions for someone else, like maybe a robot, to follow, even a simple task like hammering a nail might become extremely long and tedious to write. But we don't normally think of tasks like this in such overwhelming ways. Instead, we simplify the tasks in our mind. That is, we apply *abstraction* (a topic we'll cover in more detail throughout this course).

For example, in the real world, if you were actually driving a nail into a block of wood, you'd know to repeatedly swing that hammer over and over until you knew it was time to stop (i.e., when the nail head was flush with the wood surface).

Now think about that. You intuitively know to **repeat** a certain set of instructions ("Raise," "Aim," "Swing") **until** a certain **condition** is met ("Nail head is flush with the wood surface"). And that *condition* that helps you decide whether to repeat those "Raise-Aim-Swing" instructions one more time or to stop and put the hammer down is just like the *condition* we saw above in the section on *selection*.

This familiar pattern of *repeating* sections of instructions *until* a condition is met can be seen as a special case of *selection* in which one of the branches *loops* back to test the condition again.



Putting this into words, we might use a phrase like **Go back to Step #** to indicate that the flow of a program should not continue sequentially to the next step, but should instead jump back to an earlier step.

- 1) if the problem still exists...
- 2) ...take action to solve it
- 3) ...go back to Step 1
- 4) the problem no longer exists

In the above example, whatever action is being performed in Step 2 will continue to be repeated over and over again until the problem is solved. Notice how Step 3 explicitly states that the algorithm should go back to Step 1 where the condition (i.e., "Does the problem still exist?") can be tested again. The condition then determines whether another iteration is needed or if the loop can be exited and the program can continue on with whatever comes after this loop. That is, Step 4 is like an implicit "else" branch of an "if/else" clause. It will only be executed when the "if" condition in Step 1 is false.

Indefinite Loops

The previous example is an example of an "indefinite loop" — a form of iteration in which the number of times that the loop will execute is not explicitly stated in the algorithm. In these types of problems, we can use a **repeat until** notation to specify that the loop should continue indefinitely until a particular condition is reached.

- 1) repeat until...
- 2) ...

Our example of hammering a nail into a block of wood is an excellent example of an indefinite loop. The instructions are to essentially *repeat* the "Raise-Aim-Swing" action over and over *until* the nail is flush with the wood. The exact number of swings of the hammer that are needed are *indefinite*.

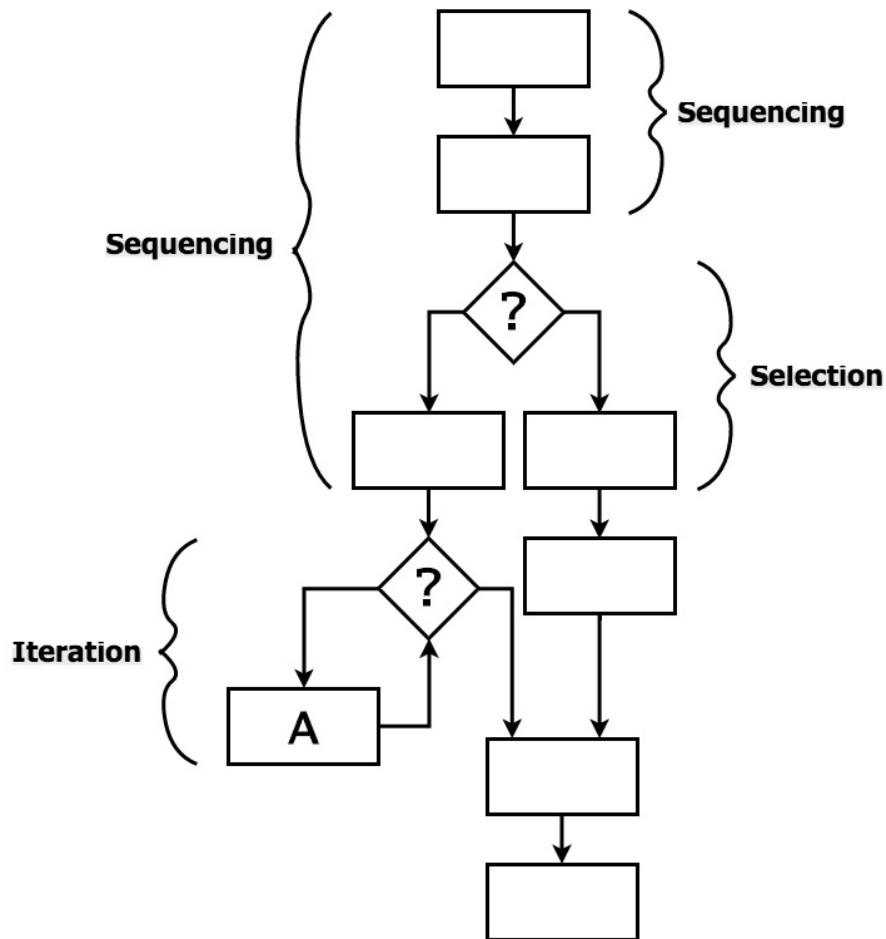
Definite Loops

However, in many other types of problems, it is often necessary to repeat an action a specific number of times. In these cases, we can use explicitly state an exact number of iterations that we need a particular section of the algorithm to be executed using the `repeat N times` notation, where `N` is the number of iterations desired.

- 1) `repeat N times...`
- 2) ...

Combinations

Finally, it's important to note that most algorithms are not limited to only one of these three common flow patterns. In fact, most programs make use of a combination of these three patterns. Oftentimes, a number of *selection* blocks are arranged together in *sequence*. Or, as we saw earlier in the hammer/nail example, a *sequential* series of steps may make up the body of the loop (i.e., the instructions that keep getting repeated).



Flowcharts

Flowcharts

When attempting to describe the structure and operation of an algorithm, it is often useful to visualize its components and their relationships to one another. Simple diagrams can be constructed that depict each step of the algorithm (usually represented by rectangles, diamonds, etc.) interconnected with arrows showing the general *flow* of the process, hence the name, *flowchart*. You've most likely seen and/or used flowcharts for years in your daily life, but perhaps you didn't consider them analog "programs" of sorts. Like programs, flowcharts serve different purposes and have different qualities.

Your task today is to create a flowchart outlining some simple instructive process (e.g., "How to brush your teeth," "How to make a sandwich," etc.). Then, you will present your flowchart to your peers by illustrating how it would work.

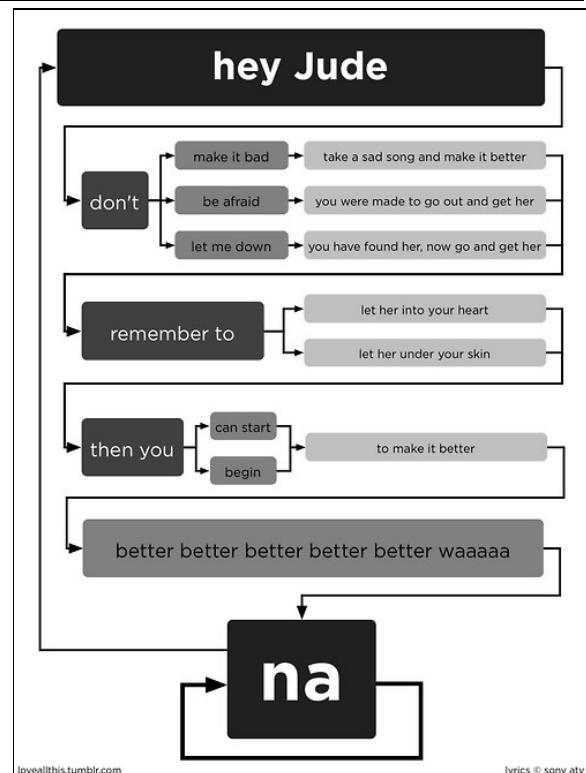
First, as a class, develop general guidelines for the flowchart. Flowcharts should contain a variety of components, such as a minimum number of steps, decisions, and inputs/outputs.

Brainstorm an idea for your flowchart with a partner quickly, then begin diagramming the flowchart. Focus on the content and function of the flowchart, but consider the usability of the flowchart, too. Be effective and efficient with your design, test it out (perhaps others will help you), and do your best.

For example, this flowchart illustrates how to sing the Beatles' song *Hey Jude* ([lyrics](#)). However, it is imperfect, like many flowcharts and programs.

- What is unclear about this flowchart?
- How could the flowchart be improved?

Consider that as you design your own flowchart, and be prepared to share, illustrate, and discuss your work afterward.



Algorithms

Recipes for Success

While computer science encompasses a lot more than just writing code, the process of constructing and expressing a detailed sequence of well-specified operations is a key component of harnessing the raw, computing power of our digital world.

As problem solvers, computer scientists and programmers must be able to develop careful and precise algorithmic solutions to whatever problem they are faced with and then communicate those solutions to other individuals and/or machines with clarity and 100% assurance that the instructions not only solve the problem, but that they do so efficiently.

Just like with a cook following a recipe in a cookbook, an algorithm is intended to spell out the step-by-step process for turning an available set of raw data (i.e., the *ingredients*) into a working solution (i.e., the *finished dish*). Each instruction represents one incremental step in that transformation and the collective total of all instructions are carefully orchestrated and arranged to perform a very specific, and oftentimes complex, task.

Why Study Algorithms?

At its core, an algorithm is really just a generalized, conceptual solution to a problem that can later be implemented in some real-world form like a computer program. Therefore, it might seem obvious that anyone wanting to be a programmer would need to be proficient at designing and working with algorithms.

But even non-programmers who might never write a program in their life can benefit from the ability to *think computationally*. Algorithms, with their logical and structured order, enable individuals to more effectively organize their thoughts and manage the complexity of modern life in this digital age.

In fact, much of this course centers on the task of helping you develop the skills to analyze a task at hand, identify what the real problem might be, recognize what you have to work with, and design an effective and efficient solution.

To help get you started, we'll examine a number of well-known solutions to common, data-oriented problems that computer scientists regularly have to deal with, such as searching and sorting algorithms. While each of the examples we'll look at are well-known solutions, they also provide an excellent look into the processes and techniques that are used in designing such solutions — techniques that you will learn to master and use in designing your own algorithms.

How Algorithms Work

As we've seen previously, most algorithms are composed of a combination of *sequencing*, *selection*, and *iteration*. Computer scientists use these familiar patterns to control the flow of execution that a computer takes through the instructions to ensure that all of the right steps

are performed in just the right way and in just the right order.

A well-written algorithm includes all of the relevant information needed to properly implement (i.e., turn ideas into code) and execute (i.e., turn code into actions) a given solution. As such, it is critical that the algorithm be communicated in some form that ensures that the individual steps of the algorithm are perfectly understood and executed exactly the way they were intended.

For example, consider the following simple algorithm:

pants, shoe, shoe, shirt

Compare that with a slightly revised version of that same algorithm:

- 1) put on pants
- 2) put on left shoe
- 3) put on right shoe
- 4) put on shirt

Both algorithms contain much the same information, but the second version is much more clearly expressed as a sequence of understandable instructions than the first version. Let's examine a few of the features that the second version has that make it more useful as an algorithm than the first version.

1) Format: The first version merely lists four words in succession, but doesn't make clear that this is an algorithm consisting of four discrete steps that must be performed. In the second version, each individual step is written on its own line, helping to make it visually clear each step is separate and unique from the other steps.

2) Imperative statements: Each instruction in the second version is written as a command with a verb phrase that clearly denotes which operation needs to be performed (in this case, "put on"). The first version lacks these verbs and leaves it unclear what is to be done with these pants, shoes, and shirt. Put them on? Take them off? Wash them? Buy them? Sell them? Who knows? The algorithm doesn't say.

3) Descriptive qualifiers: The first version only lists two shoes, but does not make any distinction between the *left* shoe and the *right* shoe. Maybe it doesn't matter whether you interpret the instructions as "*left* shoe, *right* shoe" or "*right* shoe, *left* shoe." But it probably does matter if you interpret it as "*left* shoe, *left* shoe." However, the second version of the algorithm eliminates this possible confusion by explicitly stating which shoe should be put on at each step in the process.

It is important to understand that every step of an algorithm is critically important and that they must be executed *exactly* as intended. If a step is unimportant, it would just be left out of the algorithm in the first place. The fact that it's there means it *needs* to be there. And it needs to be executed exactly as it was intended. If, for example, an algorithm for getting from Point A to Point Z says to turn left at Point F and you decide to turn right instead, it is

highly unlikely that the remaining instructions will get you to your intended destination because you've deviated from the intended plan laid out by the algorithm.

In all of these cases, it's important that whoever (or whatever, in the case of computers) reads an algorithm be able to understand with 100% certainty exactly what was intended by a given algorithm.

Fortunately, algorithms can be expressed in many different visual and/or textual formats, including flow diagrams, pseudocode, mathematical notation, programming languages, etc. Each of these formats dictates its own specific style and syntactic structure that is designed to ensure that any algorithm can be expressed in an understandable and clearly unambiguous way. Throughout this course, we'll examine a number of these formats for writing algorithms that are clear and readable.

CODING SKILLS:

Encryption

Highlights

- You will identify the needs and applications of cryptography in our digital world.
- You will analyze the differences between symmetric (single-key) encryption and asymmetric (public key) encryption.
- You will examine the mathematical foundation of cryptography.
- You will encode and decode messages using common cryptographic techniques.

Encryption

The Need for Secure Communication

For more than three millennia, humans have sought ways to protect and secure information so that it is only available to a limited set of select individuals. Whether they were ancient military leaders seeking to keep their battle plans out of enemy hands or modern-day financial investors trading stocks online, there is an unlimited number of scenarios in which access to information needs to be restricted.

One way to ensure that information is not accessible to others is to completely destroy the information. Imagine writing a private message down on a sheet of paper and then shredding the page into a million tiny pieces. The information contained within the original message will be all but lost, ensuring that no bystander might piece it all back together again. Unfortunately, it also means that you will also not be able to reconstruct your data, rendering this an undesirable solution to the problem.

Instead, the real problem is to obscure the data, rather than destroy it. If only there were a way to scramble the information in some way that renders it unreadable to anyone who does not know how to unscramble it. This is the ultimate goal of *cryptology*, the study of securing (or *encrypting*) information such that it is inaccessible by third parties.

Alice, Bob, and Eve

The classic example that is often used to demonstrate and explain the problem of encryption is the case of Alice, Bob, and Eve. In this scenario, **Alice (A)** wishes to send a message to **Bob (B)**, but she and Bob are unable to meet in person to exchange information directly. Instead, all communications between Alice and Bob must go through a third-party messenger, **Eve**. Unfortunately, Eve (as in *eavesdropper*) is potentially untrustworthy and might read any messages she is asked to deliver.

This hypothetical scenario is a stylized version of what happens every time you make a private phone call, access a web page, or send an email or text message. You, like Alice, are reliant upon the computers, routers, online services, and other parts of the network infrastructure to deliver the data of your message to your intended recipient. Unfortunately, like with Eve, you have no oversight of these components and cannot guarantee that an interested third party is not accessing the private data that you are transmitting.

Caesar Cipher

One of the earliest and simplest attempts at encryption is the *Caesar cipher*, employed by Julius Caesar in the 1st century BC. This schema is known as a *substitution cipher* because it substitutes each letter of the original, unencrypted message (called the *plaintext*) with a corresponding letter in the final, encrypted message (called the *ciphertext*).

The Caesar cipher works by aligning two alphabets against one another and offsetting them

by a number of positions. Caesar himself used a "left rotation" of three spaces, causing an **a** of the plaintext to align with an **x** in the ciphertext.

Plaintext:	abcdefghijklmnopqrstuvwxyz
Ciphertext:	xyzabcdefghijklmnopqrstuvwxyz

Try playing around with an interactive demo of the [Caesar cipher](#) to see how messages can be encrypted and decrypted.

For example, if you enter a plaintext message of **this is a caesar cipher** and set the offset to be 23 (left shift of 3), then clicking on "Encipher Plaintext," will produce the following ciphertext:

Plaintext: this is a caesar cipher
Ciphertext: QEFP FP X ZXBPXO ZFMEBO

Similarly, entering a ciphertext of **GUVF VF QRPELCGRQ** and an offset of 13 will produce the following plaintext:

Ciphertext: GUVF VF QRPELCGRQ
Plaintext: this is decrypted

Decoding Exercise

See if you can decipher these messages. Note that you might need to try a number of different offsets to find the right key to decrypt the message back into an intelligible plaintext.

PGGTFU CZ POF
DRO YPPCOD SC DOX
JC IJ TJP CVQZ YDNXJQZMZY HT NZXMZO
LUJYFWAPVU PZ MBU
MVKZGXBQWV QA ABQTT NCV

Keys

Notice how in order to decrypt each of these messages, you must know three things:

1. The ciphertext message
2. The method of encryption used to create the ciphertext (e.g., a Caesar cipher)
3. The number of positions by which the plaintext and ciphertext alphabets have been offset

The last of these items, the offset, serves as the *key* that effectively locks and unlocks the message. Using the key to encrypt the message into a ciphertext secures the message and protects it from prying eyes, much like locking the message in a box or safe. Likewise, you use the key to properly align the alphabets and unlock the message in order to read the

original plaintext.

In the earlier scenario, as long as Bob knows which key Alice used to encrypt her message, he can use the same key to decrypt the message once he receives it. However, without being told which key was used, Eve cannot decrypt the message as easily as either Bob or Alice.

Obviously, as you have previously seen for yourself, a Caesar cipher's key can be easily deduced through simple trial and error. After all, there are only 25 possible keys and it is easy to try each one through brute force.

Today, modern encryption schemes are far more sophisticated than those used in Julius Caesar's day and they use far more complex keys that are much harder to guess through brute force techniques. But the model used more than 2000 years ago is still more or less the one we use today.

1. A sender (Alice) uses an encryption scheme and a key to encode a message.
2. The encoded message is transmitted through one or more intermediate and potentially untrustworthy handlers (Eve).
3. The recipient, who knows which encryption scheme was used and is already in possession of the necessary key unencrypts the message back into its original, plaintext form.

The only difference between the ancient Roman times and now is that computational processing power has made Eve's job much easier and the need for stronger encryption algorithms and keys much greater.

Encoding Exercise

Using the [Caesar cipher](#) simulator, create five new ciphertexts of your own and exchange them with a partner. See who can decrypt the other's five messages first.

Consider designing a more sophisticated algorithm that would be more secure than the Caesar cipher. What sequence of steps could you perform to securely encode your message that would make it harder for an "Eve" to crack your code?

UNIT TOPIC:

Cybersecurity

Highlights

- You will examine a number of common threats to cybersecurity, including distributed denial of service attacks (DDoS), phishing, viruses, and social engineering.
- You will identify the needs for robust cybersecurity.
- You will analyze the software, hardware, and human components of cybersecurity.
- You will analyze the function and effectiveness of common cybersecurity solutions, including antivirus software and firewalls.

Cybersecurity

Cybersecurity

A Caesar cipher may be sufficient for obscuring notes passed in class, but modern technology makes circumventing such simple algorithms quite easy. After all, there are 25 possible offsets to attempt in a simple Caesar cipher based on the English alphabet. More sophisticated algorithms were developed centuries ago, such as the [Vigenère cipher](#) — which effectively rotates the Caesar cipher offset used to encrypt each new letter in a text.

However, encryption is just a small piece of the puzzle to secure communication.

[Cybersecurity](#) — "measures taken to protect a computer or computer system (as on the Internet) against unauthorized access or attack" — encompasses a much broader scope of techniques.

Traditionally, the [CIA Triad](#) defines the target areas when developing a secure system. CIA is an initialism representing the following concepts:

- Confidentiality is the ability to limit access to information to a certain set of users.
- Integrity is the certainty that information is accurate.
- Availability is the reliability of access to information.

Although the field of information security has grown to include other areas of focus, such as [authentication](#), these three core concepts remain central.

Confidentiality

Confidentiality is the ability to limit access to information to a certain set of users.

Confidentiality is what most people equate with cybersecurity, and at the heart of confidentiality protocols lies encryption. We examined encryption — through the Caesar cipher — as an example application of an algorithm. Although the basic algorithm of the Caesar cipher — using mathematics to convert one character to another — remains relevant today, the protocols and mathematics used to apply the algorithm have gotten much more sophisticated.

Keys

The first departure from the Caesar cipher algorithm is the use of more sophisticated *keys* in generating an encrypted text. The [Vigenère cipher](#) took the basic idea of a Caesar cipher and added an extra piece of information to make it more secure — a keyphrase.

In this simple example comparing the two methods, imagine that a plaintext message, **IT IS BURIED IN THE BACKYARD** is encrypted first with a Caesar cipher with an offset of 3:

Plaintext: IT IS BURIED IN THE BACKYARD
Ciphertext: LW LV EXULHG LQ WKH EDFNBDUG

and second with the Vigenère cipher using the keyphrase **DIG**:

Plaintext:	IT IS BURIED IN THE BACKYARD
Ciphertext:	LB OV JAUQKG QT WPK EIINGGUL

In reality, this Vigenère cipher is just using *three different Caesar ciphers* in succession, each with an offset corresponding to the letters of the keyphrase, **DIG**:

'D' is 3 letters after the beginning of the alphabet, 'A', so
'I' is 8 letters after the beginning of the alphabet, 'A', so
'G' is 6 letters after the beginning of the alphabet, 'A', so

Then, the cycle repeats:

Plaintext:	IT IS BURIED IN THE BACKYARD
Keyphrase:	DI GD IGDIGD IG DIG DIGDIGDI
Offset:	38 63 863863 86 386 38638638
Ciphertext:	LB OV JAUQKG QT WPK EIINGGUL

It is worth noting that the Vigenère cipher with the keyphrase of simply the letter **D** is the same as our original Caesar cipher in this example — each time offsetting by 3.

Restricted Knowledge

Notice that each of these encryption methods relies on restricted knowledge.

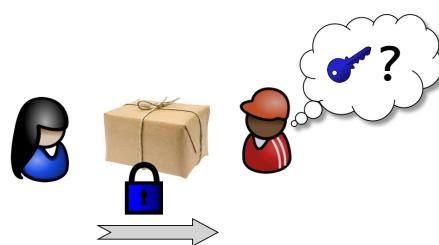
With the Caesar cipher, anyone who knows the algorithm and the offset can decrypt the message. However, we have seen that even if you know just the algorithm, it is not hard to decrypt without also knowing the offset — there are only 25 possibilities. With the Vigenère cipher, anyone who knows the algorithm and the key can decrypt the message. It is much more difficult to decrypt the message without the key than before because patterns in the text are less obvious, and there are many more possibilities than 25.

However, what if someone obtains the key?

Let's consider some possible scenarios:

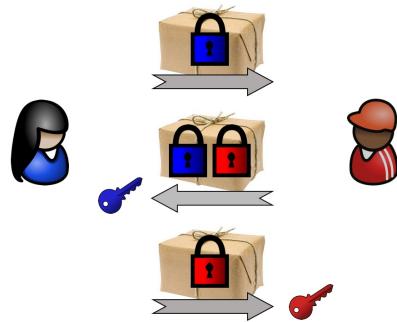
Scenario 1:

Alice sends Bob a locked box with her message inside. Although it gets passed through many hands before reaching Bob (e.g., the courier system), it is locked and so Bob receives it securely. However, to unlock it, he needs the key. How does Alice send Bob the key in a *secure* way?



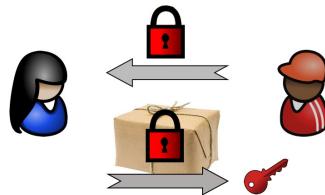
Scenario 2:

Alice sends Bob a locked box with her message inside. When it reaches Bob, he puts his own lock on it and sends it back. Then Alice removes her lock, and sends the box back to Bob. When Bob receives the box, it is locked only with the lock that he put on it. He unlocks it and retrieves Alice's message — or does he?

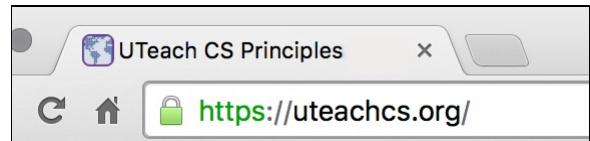


Scenario 3:

Bob has invented a special lock. It is special because it costs nothing to duplicate and send, and it is virtually impossible to analyze the lock and create a key. The key and the original lock must be created at the same time. He sends out his locks to anyone who wants to send him a message. Alice locks her box with one of Bob's special locks and sends it to him. When he receives it, he unlocks it with his special unique key and reads the message.



The third scenario is actually how modern secure message passing happens. For example, when a website asks a consumer to send his/her credit card information through a webform, the website first sends the user a lock to encrypt his/her information. Now, any traffic sent over the Internet may only be read by the website that created the original key-lock pair. This is called [Secure Sockets Layer \(SSL\)](#) and is typically indicated by a padlock icon in the browser's address bar.



Cybersecurity

Integrity

Integrity is the certainty that information is accurate.

Although confidentiality is the most *well-known* aspect of information security, integrity may be the most *important*. In considering the difference between the two, imagine an online banking system. Confidentiality dictates that only the account's owner has access to the information. Integrity guarantees that the information is correct. If every time the page is refreshed, a new (and incorrect) bank balance is shown, then the confidentiality of the information is arguably less important than its integrity.

We will discuss bits (**binary digits**) in detail in [Unit 3: Data Representation](#), but for now, at a high level, imagine that your bank balance is encoded using only zeroes and ones.

So, your balance of \$1,000 might be stored in the bank's database as:

0000001111101000

However, in transmission between the bank's database in Switzerland and your home computer, one of the bits gets flipped from a **1** to a **0**:

X
0000000111101000

When you view your bank balance, instead of the expected \$1,000, you see \$488 instead! One little **binary digit** just cost you half of your money!

Computer scientists have created an entire subfield called “coding theory” devoted to solving problems such as these.

Can you brainstorm some ways to protect against this type of error?

Repetition

TIMMY: You're dead as a doornail, Smalls.
TOMMY: You're dead as a doornail, Smalls.
TIMMY: Nice knowing you.
TOMMY: Nice knowing...
TIMMY: ...shut up, Tommy.

—Timmy and Tommy “Repeat” Timmons from *The Sandlot*

One of the simplest ways to detect errors such as this is to simply repeat what you send multiple times.

- How might this help resolve errors?
- Is it guaranteed to resolve errors?
- What is a drawback of using this method?

Can you brainstorm any other methods to present data to guarantee its correctness?

Cybersecurity

Availability

Availability is the reliability of access to information.

Availability is an important consideration in information security as well. Imagine that you were somehow able to guarantee that a message was completely accurate and confidential, but you could not guarantee that it would actually arrive at its destination. This is such an important piece of the CIA Triad that it is subject to one of the more common attacks on computer systems.

The Distributed Denial-Of-Service (DDoS) Attack

The Internet is pretty amazing. A person can sit at home at her computer, search the World Wide Web for the best price on backpacks, compare the results of multiple online retailers, select one such as amazon.com, make a secure purchase through SSL encrypted protocols, and receive the item in the mail. Not only that, but thousands of other people may be doing the exact same thing in their homes—all over the world—and never know about each other. No standing in lines, no competing for a salesperson's attention.

Moreover, in this scenario, the consumer interacts with a bunch of websites, not just the retailer. Perhaps she visits a search engine to find and visit specialized websites for price comparison, reviews, and retail. The entire process as a whole involves two-way communication between multiple computer systems.

Imagine that one of these pieces suddenly becomes *unavailable*—how does that affect the scenario?

An Analogy

Imagine that you are in a classroom of about 30 students. Alice and Bob are the class “know-it-alls” who always raise their hands to answer questions posed by the teacher, Mr. Garrison. A fellow student, Eve, decides to play a trick on them. She visits each of the other students in the class and convinces them to raise their hands anytime Mr. Garrison asks a question. Assuming he calls on any of the other students, not Alice or Bob, they agree to respond, “May I go the restroom?” or “Did you lose some weight?” or something else unrelated.

How does this affect the classroom dynamic?

Will Mr. Garrison’s question be answered? How long will it take? How does he know who in the class will respond with a legitimate answer and who won’t?

A DDoS attack works much in the same way.

Multiple computers are infected with malware (malicious software) so that an attacker may



coordinate them to send requests to a web server at the same time—much like Eve orchestrated the synchronized hand raising. So, instead of the situation described above, where amazon.com is interacting with multiple real consumers, it is also interacting with a flood of irrelevant requests, like “Reload the home page.” Because the server cannot distinguish between legitimate and bogus requests, real consumers may have to wait an exceedingly long time for their requests to be addressed. To them, however, it just seems as if the server has stopped responding.

Think About It

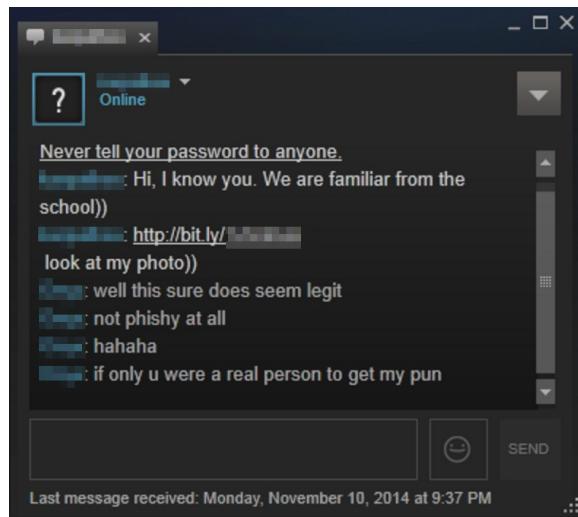
Can you imagine any solutions to the DDoS attack? How might Mr. Garrison solve and/or prevent the analogous attack in his classroom? What are some potential drawbacks to any solutions he implements? How well do these solutions carry over to DDoS attacks on web servers?

Cybersecurity

Social Engineering

Information security is not just focused on *technical* attacks on computer systems. Many malicious attackers use *social engineering* techniques as well. [Social engineering](#) refers to the “psychological manipulation of people into performing actions or divulging confidential information.”

The most commonly known social engineering tactic is to masquerade as an official representative of an organization and ask or demand confidential information. This is called *phishing*.



However, there are many other types of attacks that rely on manipulating people into believing something false or not entirely true. Most of these rely on the use of malware (malicious software).

Some examples include:

- Viruses/Worms
- Botnets
- Keyloggers
- Backdoors
- Trojan Horses
- Time Bombs
- Spyware

Exercise

Select, research, and report on one of these examples of malware. Include the following information in your report:

- What is your chosen malware?

- How does it work?
- How does social engineering figure into its distribution (i.e., what is misleading about its installation or use)?
- What are some real-world examples?
- How is it prevented and/or removed?

BIG PICTURE:

Electronic Voting

Highlights

- You will integrate the cybersecurity concepts of confidentiality, integrity, availability, and social engineering in the greater context of electronic voting.

Electronic Voting

From Paper to Bits

One of the biggest cybersecurity concerns is *electronic voting*. Paper ballots are increasingly giving way to electronically submitted, tallied, and stored votes. We have seen ways the CIA Triad is applied to electronic messaging, data storage, and account authentication to protect against malicious attacks. However, given the importance and far-reaching implications of election results, electronic voting must adhere to the CIA Triad perhaps more than any other application.

A recent news article, "[America's Electronic Voting Machines Are Scarily Easy Targets,](#)"

explores some of the objections to the use of electronic voting in national elections.



Your Task

You will select one of the components of the CIA Triad (confidentiality, integrity, or availability) and explore its implications for electronic voting. Your group will present to the class on the following:

- How does your chosen characteristic—confidentiality, integrity, or availability—apply to electronic voting? In other words, what would a violation of one of these look like in practice?
- What are the implications of such a violation?
- How might social engineering play a role in an attack targeting this CIA characteristic?
- What is a potential solution? How viable is it? What are the drawbacks associated with it?

UNIT TOPIC:

Programming Languages

Grammar, Vocabulary, and Syntax

- You will examine the need for clarity and precision in communicating an algorithmic solution to a problem.
- You will examine the shortcomings and ambiguities of natural languages.
- You will identify the elements of clear communication, including well-specified grammar, vocabulary, and syntax.
- You will analyze the need for artificial programming languages.
- You will compare high-level languages with low-level languages.

Idea to Execution

- You will examine the process in which a program is written in a high-level language, compiled into a low-level language, loaded into memory, and then executed by a processor.

UNIT TOPIC:

Programming Languages

Grammar, Vocabulary, and Syntax

- You will examine the need for clarity and precision in communicating an algorithmic solution to a problem.
- You will examine the shortcomings and ambiguities of natural languages.
- You will identify the elements of clear communication, including well-specified grammar, vocabulary, and syntax.
- You will analyze the need for artificial programming languages.
- You will compare high-level languages with low-level languages.

Clarity and Ambiguity

Communication Is Key

Algorithms are just lists of instructions for solving a given problem. But in order for the instructions to work, they must be followed exactly as intended. Fortunately, computers are very good at doing what they're told. In fact, they're designed to execute a sequence of operations in a completely predictable manner. They do exactly what they're told.

When computers misbehave, it's not the fault of the computer, but the fault of the program. More specifically, it's the fault of the *programmer*—the human who designed the algorithm in the first place. If the computer does something the programmer didn't intend, it's because it was given the wrong set of instructions. In order to get a computer to do what's intended, you must specify what you mean very precisely.

Exercise

She saw the man on the hill with the telescope.

Sketch a picture of the scene described by the above statement. Then, compare your drawing to those of your neighbors. Are they the same? Are they different? Why?

Discuss your observations with the class.

Avoiding Ambiguity

If you ask someone for directions and they say, “Third door on the left,” do they mean *their* left or *your* left? Without clarifying “my” or “your” left, the statement is ambiguous; its exact meaning can be interpreted in different ways.

In the previous exercise, you and your neighbors likely did not all draw the same thing. And yet, all of your drawings were probably perfectly correct interpretations of the phrase, “*She saw the man on the hill with the telescope.*”

So why the difference in the drawings? Because the statement is ambiguous! Computer scientists would say that the sentence has *multiple parse trees*. That means that there are *multiple* ways in which the sentence can be interpreted, or *parsed*.

If we look more closely at the sentence, the heart of the problem stems from the prepositional phrases, “on the hill” and “with the telescope.” Which of these nouns does “with the telescope” refer to? The *woman*? The *man*? Or the *hill*? Without additional information, it could modify any of them.

In fact, there are six different interpretations of the sentence that are all equally valid from a grammatical standpoint.

Which one of these did you draw? Can you draw the six *different* interpretations?

Artificial Languages

Natural Languages

English, French, Spanish, Arabic, Chinese, Hebrew, Italian... Languages spoken throughout the world are known as *natural* languages. These are languages that evolved *naturally* through everyday communication between people over hundreds of years. These languages are complex, but also structured. Each has its own set of specialized vocabulary, grammatical structure, and syntax that distinguishes it from another. Because these languages grew around human communication, they usually have both written and spoken forms. In addition, the formality of the language can vary by usage (e.g., the way you might talk to your grandmother vs. your friends, “lol t3xtsp34k”).

Despite all their differences, the one thing that natural languages have in common is their potential for ambiguity. Ambiguity makes natural languages a very poor means of communicating clear and precise instructions to be interpreted and executed exactly as intended. In short, natural languages make lousy programming languages. As computer scientists, we need something better.

Artificial Programming Languages

Enter *artificial* languages! Unlike a natural language that develops gradually and informally among whole societies, artificial languages are usually developed relatively quickly by small groups for very specific purposes. While they can be complex, just like the natural language you might use to communicate at home, they are usually much simpler and structured.

Over the years, a number of attempts have been made to develop and popularize a single, “universal” written/spoken language that could cross cultures and political borders.

Esperanto is one such attempt. To date, only about two million people speak the language (and most of those as a second language).

But for human-to-computer interactions, Artificial languages designed specifically for programming include Java, C++, Swift, Python, BASIC, Pascal, Cobol, and Fortran. These languages are usually characterized by their textual nature (i.e., letters, digits, and punctuation typed into a computer with a keyboard). They use a certain set of keywords and punctuation in order for the programmer express each instruction in a form that can be interpreted and executed without any ambiguity. Here are a few examples of the varying syntax between different programming languages and the forms that an equivalent instruction to print the word “Hello” might be written in each language.

```
10 PRINT "Hello"
```

```
print "Hello"
```

```
System.out.print("Hello");
```

Notice how even without knowing the language or what all of the special punctuation might mean, each of these statements is relatively readable to any English speaker (yes, most keywords in these programming languages tend to be English, as opposed to another natural language). Compare these to the way we've seen algorithms informally expressed throughout this unit:

1) Print the word "Hello."

We will revisit textual programming languages later in [Unit 4: Digital Media Processing](#), where you'll get to try your hand at writing programs in the *Processing* programming language.

Visual Programming Languages

One of the biggest challenges learning to work with textual programming languages is the problem of syntax—these languages are demanding about every little thing. If the programmer doesn't use the exact spelling, capitalization, and punctuation required by that specific programming language, the computer won't do what he/she wants it to do.

To combat this problem, a number of *visual programming languages* have been developed to allow programmers to drag and drop pictures or icons into organized *blocks* that represent the different parts of a program. Blocks-based programming tools automatically handle all the grammar and syntax the programming language requires. This allows novice programmers to focus on the *logic* of their programs, rather than the syntax, spelling, capitalization, etc.



The example above shows how individual programming “blocks” can be assembled in a language like *Scratch*. In [Unit 2: Programming](#), you will get to build your first interactive programs using Scratch.

High-Level vs. Low-Level Languages

Language Hierarchy

A wide range of languages exists, both natural (like English) and artificial (programming languages like Scratch). Ultimately, language is a tool for communicating an idea. In the case of computing, this communication may be human, machine, or anywhere in between.

At the *high* end of that spectrum lie the languages that are best for human communication. These languages are complex and use a high level of *abstraction* (a concept we'll explore in greater depth in [Unit 3: Data Representation](#)). They are symbolic and rely on strings of letters to form words that represent abstract concepts, ideas, actions, and objects. Our brains are optimized for this form of symbolic expression, so these languages are easy for us to read, write, and understand. Different programming languages offer different levels of abstraction. Examples can be found below. With all of its abstraction and inconsistent grammar and usage, natural languages are difficult to use in ways that will allow machines to understand our intent. With products like Apple's "Siri" interface or Google's "OK Google" voice search feature, the technology has made great strides in recent years, but there's still a long way to go before machines can truly understand natural languages.

Low-level languages, however, are optimized for machines. Computers are built using highly structured circuitry that responds to very logical and clear signals. Because low-level languages are much more concrete and straightforward, with limited vocabulary and very structured syntax, nothing is left to interpretation.

Machine languages are the most basic type of programming language. They represent the actual binary instructions issued to computer processors and are very difficult for humans to read. Moreover, they may vary from computer to computer! Believe it or not, in the beginning days of computer programming, *all* programs were written in binary machine language. Today, this is very rare.

The table below outlines this language hierarchy. Notice that the natural languages in the first row (e.g., English) are suited only for humans, and processor-specific languages in the last row (i.e., ones and zeroes, 10000010) are suited only for machines. The languages in between—high-level programming languages like Scratch, C++, Python—and low-level programming languages that tell the computer processor what to do (but are still somewhat readable by humans) bridge that gap, allowing humans and computers to communicate with each other.

Language	Characteristics	Example
Natural (English,	Evolved naturally by entire societies through human communication.	"Add 2 and 3 and

(English, Spanish, Chinese, Hindi, etc.)	Potentially ambiguous. Easy for humans to read, write, and parse. Difficult for machines to parse.	"Add 2 and 3 and assign the sum to a variable called 'x'."
High-level Programming (Java, C++, Python, BASIC, Scratch, etc.)	Relatively easy for humans to read, write, and parse. Guaranteed to be unambiguous. Easy for humans to read, write, and parse. Easy for machines to parse.	
Low-level Programming (Assembly)	A direct translation of machine language using an abbreviated syntax. Guaranteed to be unambiguous. Less natural for humans, but still readable to the trained eye. Easy for machines to parse.	
Low-level Programming (Machine)	Directly related to the hardware circuitry of the specific processor executing the code. Guaranteed to be unambiguous. Difficult for humans to read and write. Easy for machines to parse.	

The high-level and low-level *programming languages* in the middle of the table are uniquely well-suited for computer scientists because they are abstract enough to feel natural and intuitive to humans, but basic and structured enough for machines to process with the level of precision that's required to have them do what we want.

High-Level Programming Languages

Over the next few units, you'll get the chance to work with two different high-level programming languages: *Scratch* and *Processing*. *Scratch* is a *visual programming*

language, allowing you to drag-and-drop blocks to communicate with the computer without worrying about spelling, punctuation, etc., whereas Processing is a *textual programming language*.

One of the key features of most high-level programming languages is that, because of the abstraction they employ, people can program without needing to worry about (or even know about) the specific configuration or design of the computer's underlying circuitry. This allows the programmer to focus solely on the task of designing and coding a logical solution to whatever problem he/she might be working on.

Languages like Scratch and Processing are platform-independent, meaning that the programs you write will run on just about any modern computer, regardless of the model, version of the operating system (e.g., Windows, Mac OS, Linux, etc.), or maker of the hardware.

Low-Level Programming Languages

On the other end of the spectrum, low-level languages are entirely dependent on the underlying hardware. The language the computer uses is specific to the individual processor within the machine. That is, there is a direct correlation between the 1s and 0s of the binary code and the billions of microscopic, electronic switches embedded within the circuitry of the computer.

Fortunately, most programmers never need to actually work at such a low level or program directly with 1s and 0s. Instead, software development tools, such as the Scratch and Processing interfaces you will use in upcoming units, automatically generate the low-level, binary code that the processor requires. They do this by interpreting your high-level, abstract instructions into lower-level machine code through a process known as *compilation*, which we'll explore in the next section.

UNIT TOPIC:

Programming Languages

Idea to Execution

- You will examine the process by which a program is written in a high-level language, compiled into a low-level language, loaded into memory, and then executed by a processor.

Idea to Execution

Life Cycle of a Program

How does a program become a program? And what does a computer actually do when it “runs” a program? Let’s take a peek under the hood and examine the process of how an idea gets turned into a dynamic, interactive program running on your computer.

The Idea

It all starts with an idea. Well, actually, it starts with a problem that necessitates a solution that then gives rise to the idea for how to solve the problem.

Smart problem solvers begin the process of turning an idea into something real by first analyzing the problem at hand. They ask questions, collect information, gather data, look at existing solutions, and think through the situation as thoroughly as they can before diving into a solution.

It might seem difficult to restrain yourself when you are struck with a brilliant idea. But the best ideas are almost never the first ones that come up. Remember the encryption exercise. Most students could intuitively come up with the Caesar cipher all on their own, but that proves to be one of the least robust solutions to that problem.

In practice, the first idea to come to mind is almost always an inferior solution. But with careful thought, consideration, and analysis of the idea’s strengths and weaknesses, a good designer will always come up with something better than their first instinct. So, avoid those kneejerk reactions to initial ideas and look for the better idea. It’s out there. You just need to make the effort to seek it out.

Construct an Algorithm

Once a developer has identified a solution that is worth pursuing, the next step is to begin turning it into a well-specified plan of action.

As we’ve seen earlier in this unit, computer scientists — whether they are programmers writing code or designers developing algorithms — apply their computational thinking skills to the process of mapping out a logical sequence of instructions that will solve the problem. They recognize and use the *sequencing*, *selection*, and *iteration* patterns we’ve looked at in the past. They anticipate and identify all of the unseen and unexpected complications that their solution will need to encounter. And they clearly define the requirements, expectations, and specifications for how the resulting program should perform.

Write the Code

Once the program specifications have been clearly defined and all algorithms have been constructed, a programmer then **implements** the algorithm in a formal programming language (i.e., *writes* the code at an appropriate high-level or low-level programming language).

These days, most software is developed in a high-level language that takes advantage of the benefits of abstraction in making the final code simpler and more straightforward for humans to understand, while still remaining formal and strict enough for a machine to read and parse. A wealth of development tools are available to help programmers more easily manage large programs that combine large numbers of smaller algorithms and operations. Both *Scratch* and *Processing*, which you will work with in the next few units, provide an easy-to-use, graphical interface with which to develop, test, and run your programs.

Smart programmers also design thorough test cases to verify that all aspects of their program perform as intended. Sometimes the design of the algorithm is flawed, leading to a program that misperforms. Other times, the algorithm is sound, but the program fails to accurately implement it in the chosen programming language. Either way, it is important that these *bugs* (i.e., *errors*) in the program are identified and corrected before the software is distributed to end-users.

Compile the Code

The programs that programmers write in high-level languages are known as **source code**. Technically, these source code files are not really programs, per se. They are the abstract, high-level instructions that define what the program is and how it will function. The *actual* program — that is, the file containing the machine-level instructions the computer actually runs — is written in special binary (1s and 0s) patterns that relate directly to the way the computer's processor has been designed and wired. This binary, **machine code** is a direct, low-level translation from the high-level source code.

Fortunately, the programmer does not have to personally perform this tedious translation. Instead, it can be automated in a process called **compilation**. This basically means that your program can be translated by another program! During the compilation process, another program, called the **compiler**, translates each of your high-level, human-readable instructions into a corresponding string of 1s and 0s that make sense to the computer's processor. The resulting string of binary information constitutes an *executable* program.

Execute the Compiled Code

When a user wants to run, or *execute*, a program, all of the instructions and raw data that make up that program are loaded into memory (i.e., RAM). That is, the binary, machine code is copied from wherever it is stored in **secondary storage** (e.g., hard drive, USB drive, etc.) into **primary storage** (e.g., RAM).

RAM is essentially just a really big, organized collection of tiny electrical circuits for storing information. Each of those circuits can be individually turned *on* or *off*, which corresponds to the binary states of **1** and **0**, respectively. Each circuit in RAM is, thus, capable of storing **1 bit** (from the abbreviation of “**b**inary **d**igit”) of information. For perspective, 1 gigabyte (1 GB) of RAM consists of more than **8.5 trillion** bits.

Once a program's binary contents have been copied into RAM, the computer's **central**

processing unit (CPU) begins reading each of the binary instructions bit-by-bit, byte-by-byte, word-by-word as it executes each instruction in the sequence that the code/algorithm specifies.

As it does so, each instruction or operation causes various electronic components to be activated throughout the computer. Some of these operations change the values stored in RAM. Others read and calculate new values based on information stored in RAM. Still others draw images, symbols, or text onto the screen; capture user input from a mouse, keyboard, or touchscreen; send data to a printer; play music and sound effects; or any number of other types of basic computational functionality.

The sum total of all of these discrete operations is what the end user recognizes as a properly functioning program, whether it is a web browser, a word processor, a game, a streaming media player, or whatever else it was that the original *idea* that started this process was intended to do.

CODING SKILLS:

Pseudocode

Highlights

- You will examine the use of pseudocode to quickly and clearly express general algorithmic ideas.
- You will familiarize yourself with the College Board's standardized pseudocode that will be used on assessments throughout the course and on the AP Computer Science Principles exam in May.

Pseudocode

Communication Is Key

You might have noticed that throughout this unit, we've been using a particular style and format for presenting each of our algorithms:

- 1) do the first operation
- 2) do the second operation
- 3) if a condition is true...
- 4) ...do the third operation
- 5) otherwise...
- 6) ...do an alternate third operation
- 7) do the fourth operation
- 8) repeat 100 times...
- 9) ...do the fifth operation
- 10) stop.

This format was chosen specifically to communicate very precise and unambiguous instructions while maximizing its readability. It has a very clear and mostly intuitive structure that has hopefully been easy for you to read, follow, and understand. Unfortunately, it has some drawbacks.

For example, it can tend to be a bit wordy. We might start to abbreviate our wording or drop minor words that aren't essential for getting the idea across, like articles ([a](#), [an](#), [the](#)). Consider these two alternatives.

4) ...do the third operation

4) ...do operation3

One could argue that the second version is just as clear about its intent despite being more concise. Of course, anytime you abbreviate some text or leave out some words, you run the risk of leaving out important information. So the challenge becomes finding the right balance between conciseness and completeness — that is, being brief, but still saying everything you need to say.

This is a major issue when a language developer tries to design a new programming language. As we saw when we discussed high-level and low-level languages, each language has its own particular set of grammatical rules and structures that are designed to allow programmers to clearly communicate everything they need to express with conciseness and precision.

However, when talking about algorithms in general, we're interested in being able to describe solutions at an extremely high conceptual level without needing to get into the specifics of the grammar of a particular programming language. And that's where

pseudocode comes into play.

"False Code"

We use the term **pseudocode** to describe any method used to express an algorithm in an informal, language-agnostic manner. Derived from the Greek word *pseudes*, or "lying, false," pseudocode refers to a way of writing formal instructions in an informal manner that does not necessarily adhere to the grammatical rules and syntax of any particular language.

Instead, pseudocode aims to mimic the general style of a programming language without worrying about the exact syntax or structure of the language. It's basically a shorthand way of writing out the details of an algorithm.

In fact, while pseudocode does not have any standard, universal form, most individuals or organizations that need to develop and communicate high-level algorithms will often settle on their own, mutually agreed-upon set of notational conventions that they will use.

AP Computer Science Principles

Having said that, since the College Board does not designate any particular programming language for this course, it has established its own set of *pseudocode* conventions that it will use on the AP Computer Science Principles exam.

Pseudocode for the AP Exam

Exam Reference Sheet

The following materials are available in the "Reproducibles for Students" section (p. 102) of the [*AP Computer Science Principles Course and Exam Description*](#). The "AP Computer Science Principles Exam Reference Sheet" begins on p. 114.

As AP® Computer Science Principles does not designate any particular programming language, this reference sheet provides instructions and explanations to help students understand the format and meaning of the questions they will see on the exam.

While students are free to use any format of their choosing when writing their own algorithms, they are encouraged to use this style guide as a reference.

AP Computer Science Principles Exam Reference Sheet

As AP Computer Science Principles does not designate any particular programming language, this reference sheet provides instructions and explanations to help students understand the format and meaning of the questions they will see on the exam. The reference sheet includes two programming formats: text based and block based.

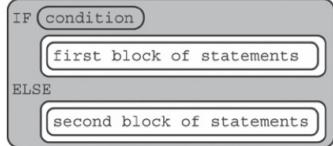
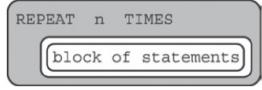
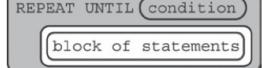
Programming instructions use four data types: numbers, Booleans, strings, and lists.

Instructions from any of the following categories may appear on the exam:

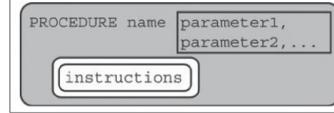
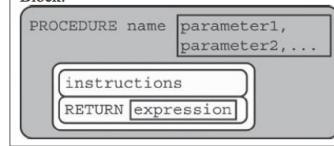
- ▶ Assignment, Display, and Input
- ▶ Arithmetic Operators and Numeric Procedures
- ▶ Relational and Boolean Operators
- ▶ Selection
- ▶ Iteration
- ▶ List Operations
- ▶ Procedures
- ▶ Robot

Instruction	Explanation
Assignment, Display, and Input	
Text: <code>a ← expression</code>	Evaluates expression and assigns the result to the variable a.
Block: 	
Text: <code>DISPLAY (expression)</code>	Displays the value of expression, followed by a space.
Block: 	
Text: <code>INPUT ()</code>	Accepts a value from the user and returns it.
Block: 	
Arithmetic Operators and Numeric Procedures	
Text and Block: <code>a + b</code> <code>a - b</code> <code>a * b</code> <code>a / b</code>	The arithmetic operators +, -, *, and / are used to perform arithmetic on a and b. For example, <code>3 / 2</code> evaluates to 1.5.
Text and Block: <code>a MOD b</code>	Evaluates to the remainder when a is divided by b. Assume that a and b are positive integers. For example, <code>17 MOD 5</code> evaluates to 2.
Text: <code>RANDOM (a, b)</code>	Evaluates to a random integer from a to b, including a and b.
Block: 	For example, <code>RANDOM (1, 3)</code> could evaluate to 1, 2, or 3.
Relational and Boolean Operators	
Text and Block: <code>a = b</code> <code>a ≠ b</code> <code>a > b</code> <code>a < b</code> <code>a ≥ b</code> <code>a ≤ b</code>	The relational operators =, ≠, >, <, ≥, and ≤ are used to test the relationship between two variables, expressions, or values. For example, <code>a = b</code> evaluates to true if a and b are equal; otherwise, it evaluates to false.

Instruction	Explanation
Relational and Boolean Operators (continued)	
Text: NOT condition Block: NOT <code>(condition)</code>	Evaluates to true if condition is false; otherwise evaluates to false.
Text: condition1 AND condition2 Block: <code>(condition1) AND (condition2)</code>	Evaluates to true if both condition1 and condition2 are true; otherwise, evaluates to false.
Text: condition1 OR condition2 Block: <code>(condition1) OR (condition2)</code>	Evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true; otherwise, evaluates to false.
Selection	
Text: <code>IF (condition) { <block of statements> }</code> Block: 	The code in block of statements is executed if the Boolean expression condition evaluates to true; no action is taken if condition evaluates to false.

Instruction	Explanation
Selection (continued)	
Text: <pre>IF (condition) { <first block of statements> } ELSE { <second block of statements> }</pre> <p>Block:</p> 	The code in first block of statements is executed if the Boolean expression condition evaluates to true; otherwise, the code in second block of statements is executed.
Iteration	
Text: <pre>REPEAT n TIMES { <block of statements> }</pre> <p>Block:</p> 	The code in block of statements is executed n times.
Text: <pre>REPEAT UNTIL (condition) { <block of statements> }</pre> <p>Block:</p> 	The code in block of statements is repeated until the Boolean expression condition evaluates to true.

Instruction	Explanation
List Operations	
For all list operations, if a list index is less than 1 or greater than the length of the list, an error message is produced and the program terminates.	
Text: list[i]	Refers to the element of list at index i. The first element of list is at index 1.
Block: list [i]	
Text: list[i] ← list[j]	Assigns the value of list[j] to list[i].
Block: list [i] ← list [j]	
Text: list ← [value1, value2, value3]	Assigns value1, value2, and value3 to list[1], list[2], and list[3], respectively.
Block: list ← value1, value2, value3	
Text: FOR EACH item IN list { <block of statements> }	The variable item is assigned the value of each element of list sequentially, in order from the first element to the last element. The code in block of statements is executed once for each assignment of item.
Block: FOR EACH item IN list block of statements	
Text: INSERT (list, i, value)	Any values in list at indices greater than or equal to i are shifted to the right. The length of list is increased by 1, and value is placed at index i in list.
Block: INSERT list, i, value	
Text: APPEND (list, value)	The length of list is increased by 1, and value is placed at the end of list.
Block: APPEND list, value	

Instruction	Explanation
List Operations (continued)	
Text: REMOVE (list, i) Block: 	Removes the item at index <i>i</i> in <i>list</i> and shifts to the left any values at indices greater than <i>i</i> . The length of <i>list</i> is decreased by 1.
Text: LENGTH (list) Block: 	Evaluates to the number of elements in <i>list</i> .
Procedures	
Text: PROCEDURE name (parameter1, parameter2, ...) { <instructions> } Block: 	A procedure, <i>name</i> , takes zero or more parameters. The procedure contains programming instructions.
Text: PROCEDURE name (parameter1, parameter2, ...) { <instructions> RETURN (expression) } Block: 	A procedure, <i>name</i> , takes zero or more parameters. The procedure contains programming instructions and returns the value of <i>expression</i> . The <i>RETURN</i> statement may appear at any point inside the procedure and causes an immediate return from the procedure back to the calling program.

Instruction	Explanation
Robot	
If the robot attempts to move to a square that is not open or is beyond the edge of the grid, the robot will stay in its current location and the program will terminate.	
Text: MOVE_FORWARD () Block: 	The robot moves one square forward in the direction it is facing.
Text: ROTATE_LEFT () Block: 	The robot rotates in place 90 degrees counterclockwise (i.e., makes an in-place left turn).
Text: ROTATE_RIGHT () Block: 	The robot rotates in place 90 degrees clockwise (i.e., makes an in-place right turn).
Text: CAN_MOVE (direction) Block: 	Evaluates to <code>true</code> if there is an open square one square in the direction relative to where the robot is facing; otherwise evaluates to <code>false</code> . The value of <code>direction</code> can be <code>left</code> , <code>right</code> , <code>forward</code> , or <code>backward</code> .

UNIT TOPIC:

Solvability and Performance

Decidability and Efficiency

- You will identify which problems can and cannot always be solved by an algorithm.
- You will examine methods of comparing equivalent algorithms for relative efficiency.
- You will evaluate the relative efficiency of equivalent algorithms.
- You will identify factors that allow solutions to scale efficiently.
- You will examine the implications of Moore's Law on the research and development of new and existing technologies.

Decidability and Efficiency

Can Computers Solve Every Problem?

It probably doesn't surprise you to hear the answer is *no*. However, it may surprise you to learn that this is *mathematically proven* to be impossible. In fact, Alan Turing, one of the most influential figures in computer science, theorized what would later become the digital computer as part of his proof.



An *undecidable problem* is one in which no algorithm can be constructed that always leads to a correct yes-or-no answer. Turing proved that at least one task is computationally *undecidable* — **The Halting Problem**. He proved that you cannot write a program that will tell you which computer programs will halt (i.e., exit) and which ones will continue forever (i.e., infinitely loop).

Many Ways to Solve a Problem

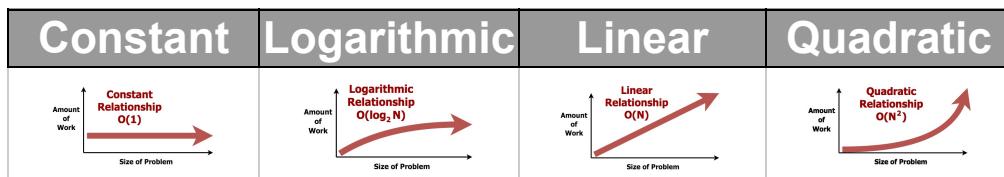
However, most computer scientists spend time constructing programs for problems that computers *can* solve. A programmer's knowledge and skill affects how a program is developed and how it is used to solve a problem. Even so, there are often many different approaches to solving the same problem. Multiple solutions may be equally valid with no single one being exclusively better than the other. In fact, for most of the problems that you'll encounter in your life, there is **almost never just one, single way that the problem can be solved**. Instead, there are almost always multiple ways of solving a given problem — some of which might be better choices than others.

Your challenge as a computational thinker is to 1) design a solution that works and 2) recognize the ways in which your algorithm can be made more efficient.

Scalability

So, how do we know that one algorithmic solution is any better than another? In order to answer that question, we first need a way of measuring the relative performance that an algorithm offers. And that measurement comes down to a question of **scalability**. That is, how well does the algorithm perform at larger and larger scales?

For example, if we *double* the number of items in a list, how much harder does that make the task of finding a particular item in that list? What if we *triple* the size of the list? *Quadruple* it? What if we increase the size of the overall problem by a factor of N ? How much does that increase the amount of work required by the algorithm?



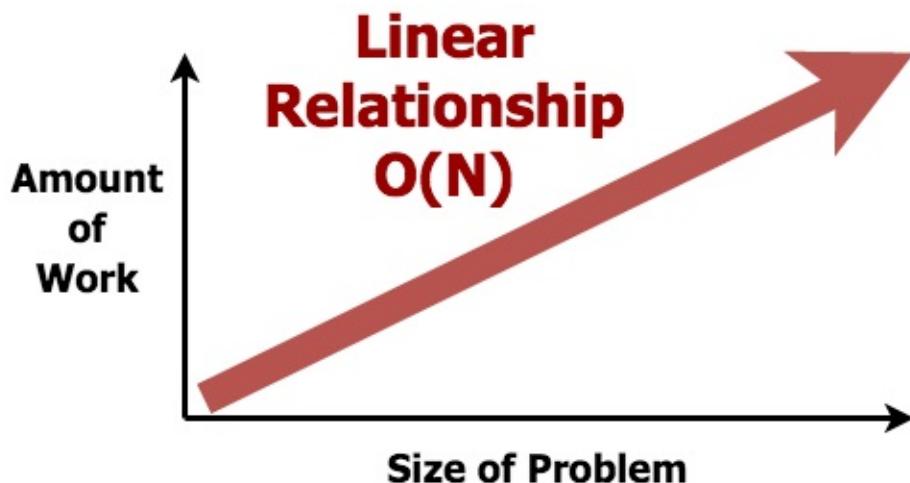
Computer scientists use something called "Big-O Notation" (a mathematical concept we won't get into here) to describe how well different algorithms scale. This allows them to classify different solutions into different categories of relative performance.

- **Constant:** No matter how much a problem grows, the amount of work stays more or less the same.
- **Logarithmic:** Every doubling of the size of a problem only requires one extra unit of work.
- **Linear:** As the size of a problem grows, the amount of work required grows at approximately the same rate.
- **Quadratic:** As the size of a problem grows, the amount of extra work required increases much more quickly.

Performance Comparisons

Imagine you needed to search through a list of 1,000,000 items.

If the item were toward the front of the list, **Sequential Search** would find the item with very few examinations. However, if the item were toward the end of the list, Sequential Search might need to look through as many as one million items! On average, depending on where the item might be located in the list, Sequential Search will require examining about 500,000 items. This is referred to as having **linear** performance because there is a linear relationship between the size of the problem (i.e., the total number of items you must search through) and the amount of work required to solve the problem (i.e., how many items you must actually look at before finding what you are looking for).



Compare that to **Binary Search**, which does not see the same one-to-one relationship between the size of the problem and the amount of work. Because of its divide-and-conquer approach, the amount of work required to find an item grows much more slowly with Binary Search than with Sequential Search. In fact, with this **logarithmic** behavior, you can double the total number of items in the list and it only costs one more unit of additional work. This means that if the size of the problem grows incredibly large, the amount of extra work that the algorithm will require will stay relatively small.



Problems with Scalability

Small and simple problems are often easy to solve because even an inefficient solution likely doesn't require much work in the first place. But as problems grow larger, the efficiency of an algorithm becomes more and more important. So important that it might mean the difference between the problem even being solvable or not. Improvements in algorithms, hardware, and software increase the kinds of problems and the size of problems solvable by programming.

Case Study #1: Finding an IP Address

When you open up a Web browser and click on a link or type in a URL, the page you requested usually loads pretty quickly — often in only a few seconds and rarely no more than a minute. But a lot has to happen in those few seconds in order for that to happen. And surprisingly, that involves some extremely large problems.

When you initiate one of these Web page requests from your computer, located somewhere on the Internet (i.e., wherever you are at the moment), it must be sent through the vast network of computers to find the one, specific computer that the website's server runs on. Out of the millions (billions?) of computers on the Internet, your browser has to find one very specific machine. And if that sounds like a case for a searching algorithm, you're right!

In order to route a request to a specific server, the server's IP address is used to identify the intended destination. The IPv4 standard, with its 32-bit addressing (###.###.###.###), means that there are technically 2^{32} uniquely different addresses — or more specifically, 4,294,967,296 uniquely different destinations for your request to be sent to.

Altogether, this means that in the blink of an eye that it takes your browser to load and display your desired webpage, it has to search through a routing table of more nearly 4.3 billion addresses to find the right destination. Clearly, the system has been designed to use a highly efficient set of algorithms for conducting such a large-scale search in an extremely short amount of time.

That is, the infrastructure of the Internet was designed specifically to ensure that it could perform its basic routing functions quickly and accurately despite the massive scale of the

problem — it scales well.

Case Study #2: Brute Forcing a Password

On the other hand, suppose law enforcement wants to develop a piece of forensics software that is intended to break the encryption on whatever data they need to investigate. The easiest solution is to take a "brute force" approach that simply tries all possible password combinations. It is essentially a Sequential Search through every possible password, which makes it a linear solution, which certainly seems like it should be a viable solution to the problem. But how well does it really scale?

In this case, it is not so much a problem with how efficiently the algorithm scales, but rather a problem with how quickly size and complexity of the problem can scale.

Imagine you have a four-digit ($0 - 9$) passcode, like you might have on your phone or home security system. How many possible passcodes exist? Well, if the code consists of four digits, each of which is one of 10 possible digits ($0 - 9$), then there are 10^4 , or 10,000, unique combinations (0000 through 9999). A brute force approach would only need to make no more than 10,000 attempts to ensure success. Ten thousand is a lot, but trivial with the help of computational technology.

How can we make it harder to crack?

1. Make the passcode longer.
 - Every single digit added to the length of the passcode increases the number of possibilities by a factor of 10.
2. Increase the number of characters to use.
 - Instead of limiting the passcode to numerical digits, 0 through 9 , it could use uppercase letters, lowercase letters, punctuation, symbols, emoji, etc.

Digits	Characters	Combinations
4	$0 - 9$ 10	$10^4 = 10,000$
5	$0 - 9$ 10	$10^5 = 100,000$
4	$0 - 9$, $a - z$ 36	$36^4 = 1,679,616$
4	$0 - 9$, $a - z$, $A - Z$ 62	$62^4 = 14,776,336$
5	$0 - 9$, $a - z$, $A - Z$ 62	$62^5 = 916,132,832$
8	$0 - 9$, $a - z$, $A - Z$ 62	$62^8 = 218,340,105,584,896$
12	$0 - 9$, $a - z$, $A - Z$ 62	$62^{12} = 3.2E21$

Notice how quickly and easily it is to increase the size of the problem. Even just a meager, eight-digit passcode made up of alphanumeric characters (`0`–`9`, `a`–`z`, `A`–`Z`) has more than 200 trillion possible combinations, meaning the brute force approach could similarly require more than 200 trillion attempts before cracking the encryption. Even at only 12 digits long, such a passcode would allow so many possible passcodes that a brute force attempt taking only 1ms per attempt would still require ten times the age of the universe. Clearly, a brute force solution does not scale well.

Exercise

Working together in small groups, identify real-world examples of problems whose solutions do and do not scale well. Also identify a problem that is so complex that there is no computational solution to *feasibly* solve the problem in a reasonable amount of time.

Be prepared to share and discuss your examples with the class.

Moore's Law

Skating to Where the Puck is Going to Be

"I skate to where the puck is going to be, not where it has been."
— Wayne Gretzky

If there is one thing that can be said about technology, it's that it always changes. Whether technology gets faster or smaller or more powerful, it's a moving target. And any innovators who want to capitalize on the latest advances in computing technology face the difficult challenge of not just keeping up with these changes, but trying to get *ahead* of them.

Between the research and development process, the scaling up of production, and the lengthy marketing and promotion of a new product, it takes time to invent a new technology and bring it to market. Enough time, in fact, that the technology might be outdated by the time it reaches the end users.

For the last half-century or more, technologists have struggled to find a balance between using the latest, cutting-edge technology versus anticipating and gambling on technological advances that haven't even been invented yet. If they're not forward-thinking enough, their products will be stale and underperforming and they'll go out of business. However, if they're too forward-thinking, the technological advances their product relies on might not come about in time to keep the company running, in which case they'll also go out of business.

History is littered with failed companies who either under- or overestimated the pace of technological change. But those who have been able to have been able to accurately predict the rate at which these advances will occur have reaped the rewards of innovation after innovation in our digital world.

A Remarkably Astute Observation

Back in 1965, before the industry really had any idea how to measure its rate of progress, much less the importance of knowing how to predict the pace of future innovations, Gordon Moore (co-founder of Intel) made an observation that revolutionized the technology industry and the way we think about building upon today's technology in order to invent tomorrow's technology.



As the director of research and development at Fairchild Semiconductor, Moore was asked to speculate on how he imagined the semiconductor components industry might develop over the next 10 years. This industry was responsible for turning silicon ingots into the intricately designed, wafer-thin discs that solid-state components and integrated circuits are made of. Each of these components comprise millions of tiny transistors, resistors, diodes,

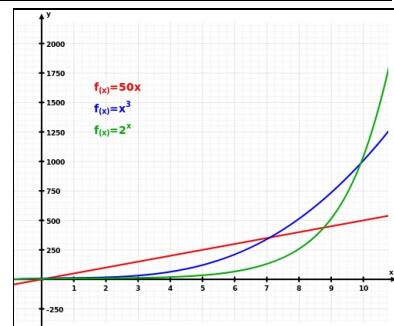
and capacitors all carefully interlaced to form the innerworkings of our modern computing technology.

Moore noted that as the manufacturing tools and processes miniaturized over time, we were able to pack more and more of these tiny components into the same amount of space. Specifically, he observed that the number of transistors that could fit on a chip roughly doubled every one to two years.

While he originally predicted only that this rate of progress would last for the next decade, incredibly, it has held true for the last 50 years! Due to its long-lasting accuracy, Moore's observation that the density of transistors doubles at a predictable rate has more commonly been dubbed "Moore's Law."

Exponential Growth

The mathematical property of Moore's Law that makes it such a powerful concept lies in the fact that it describes an exponential growth in the capacity and performance of electronic components. Rather than seeing a steady pattern of incremental improvements over time (a linear pattern), we instead see an accelerating increase in performance (an exponential pattern). That is, every year doesn't just see the same improvement as the previous year. Instead, each year sees *more* improvement (*twice* as much, in fact) than what was achieved the previous year. And the following year will see an even greater increase in improvement.



Implications of Moore's Law

Today, when we hear someone use Moore's Law as a predictor of something, it's almost never about the doubling of the number of transistors that can fit on a chip. Instead, it's usually in reference to the doubling of speed or the doubling of processing power or the doubling of some other relevant metric of computing strength.

In the case of memory (e.g., RAM), an increase in the number of transistors means an increase in the number of memory circuits that can be built from those transistors. And more memory means more data can be stored and/or manipulated.

But how do the number of transistors that can fit on a chip tell us anything about improvements in speed or processing power? The answer has to do with miniaturization. Any increase that Moore's Law predicts in the density of transistors means a corresponding decrease in the size and/or spacing of transistors. In other words, the transistors are closer to one another and the electrons that flow from one transistor to the next have less distance to travel and thus, can propagate more quickly through the components of a circuit and perform more calculations in a given period of time.

In short, Moore's Law tells us that we can expect exponential improvements over time for a wide array of properties of our computing hardware, including memory capacity and/or

speed. And with that knowledge, we can better plan for future technologies, even those that are not yet feasible.

Predicting the Future

"The best way to predict the future is to invent it." — Alan Kay

But how do computer scientists and engineers actually use Moore's Law?

Imagine you have an idea for a new form of technology, but after careful design and analysis, you've estimated that today's technology is an order of magnitude (i.e., a factor of 10) too slow to handle the massive amounts of real-time computations that your invention will need.

Using Moore's Law, if we assume a doubling of speed approximately every 18 months, you can reasonably predict that computers will achieve the tenfold speed increase that you need in only five years. You can then begin planning the research and development of your invention over the next five years, knowing that by the time you need the technology to perform at the levels you require, it will.

This is exactly how large tech manufacturers operate. At any given point in time, their engineers are busy working on technologies that are not yet feasible, but that will become a reality by the time they are ready to introduce them onto the market.

Nothing Lasts Forever

For decades, people have been warning about the impending demise of Moore's Law, even Gordon Moore himself!

And it's true, unfortunately. Moore's Law has its own limits. Transistors cannot become infinitely dense. At some point, this steady, exponential increase that we've witnessed over the last half century will have to slow to a stop. But when? And why?

The fact is, over the last decade or so, the miniaturization of transistor technology has actually made the hardware less reliable due to the physical limits of silicon and electricity.

A transistor is, essentially, just a controllable switch. Apply an electric charge to it and the transistor will open, allowing a second electric charge to flow through a logic circuit. Remove the charge and the switch closes, stopping the flow of electrons through the logic circuit. But as these components have gotten smaller and smaller over the years, the number of electrons flowing through each switch has been reduced from a flood to a trickle and the percent of electrons that leak out of the circuitry (noticeable as radiant heat) has skyrocketed (that's why your laptop or tablet or phone feels warm after extensive use). With fewer electrons to carry information through the circuitry, more and more error is introduced into the system.

As the electronic components get smaller and faster, the excess heat and the corresponding loss of information increase.

It is inevitable, then, that a point will come when these costs exceed the benefits and the continued miniaturization of circuitry will no longer be a useful design methodology. At that point, Moore's Law, as we know it, will truly be dead.

Sidestepping Moore's Law

So what will happen when the physical limits of silicon-based circuitry inevitably maxes out and Moore's Law is no longer a useful metric? Will technological innovations come to a halt? Will our computers and phones and other electronic computational devices stop getting faster or smaller or more powerful?

Not likely. The demand for faster, smaller, and more powerful technology will remain, driving technologists to find new and alternative ways of solving the problem that don't rely on the continued miniaturization of electronics.

In fact, in recent years, many of these alternative solutions are already quite common in today's consumer electronics and computers. One such solution has been the advent of multi-core processing. Essentially, if new processors cannot be designed to operate any faster or more efficiently than current processors, then perhaps we can achieve an effective doubling of performance by simply adding a second processor.

That sounds good (and it can be), but it also introduces a host of new challenges and tradeoffs, such as concurrency and multi-threading (i.e., the simultaneous use of shared resources). And unfortunately, not all computations can be accelerated by adding more processors. After all, if you had two handheld calculators, would that enable you to complete your math homework twice as fast? What about three calculators? Or four? Or ten?

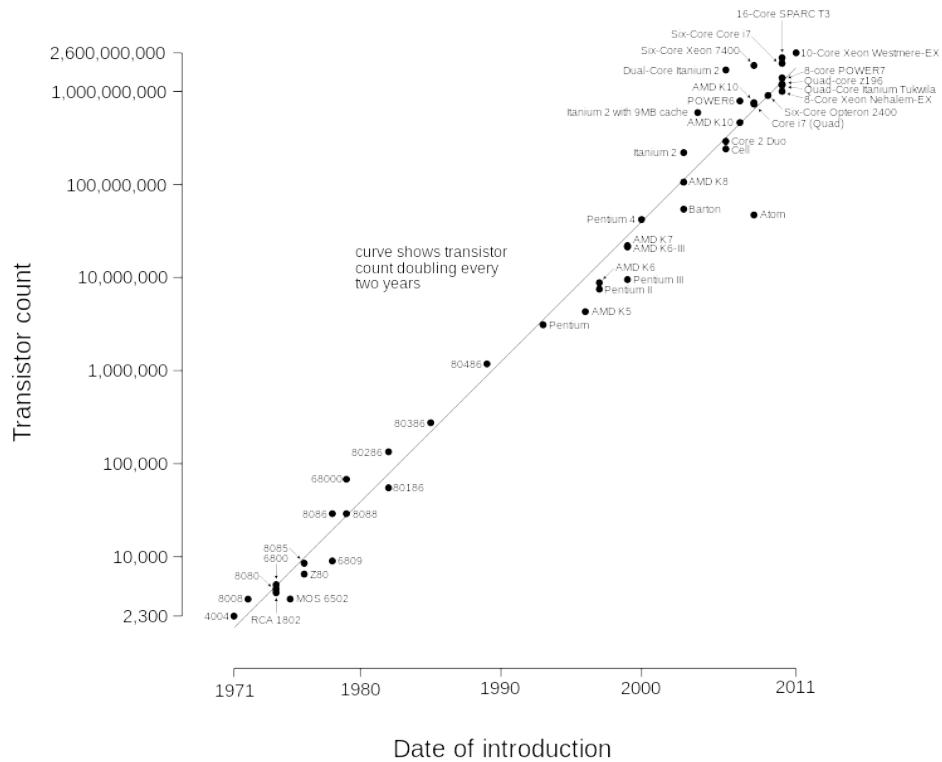
Not Just for Technology

One of the most amazing things about Moore's Law is that, while it originated as a descriptor of technological improvements in silicon-based hardware, the patterns that it describes can (and have) been seen in numerous other non-technological contexts throughout science and society.

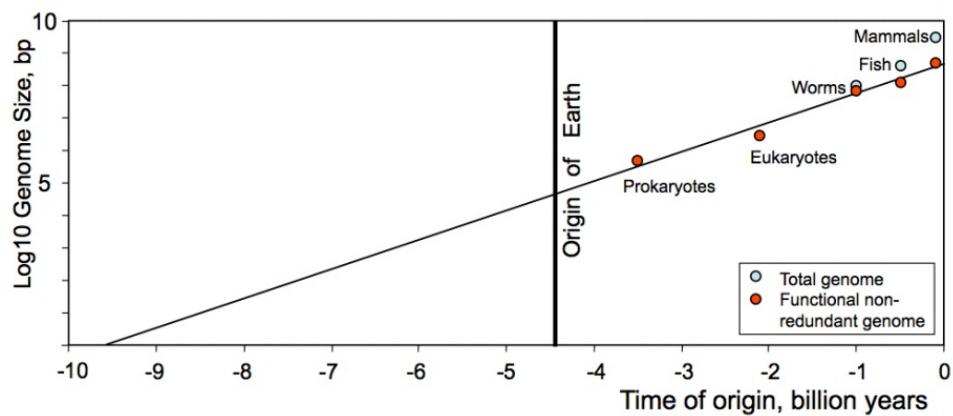
Many things, in fact, exhibit similar, exponential rates of change. And if one knows what that rate is, not only can you extrapolate forward in time to predict future outcomes, but you can also extrapolate backward in time to reveal the approximate origins of observed phenomena as well.

For example, given the number of transistors that can fit on a chip today, we can actually work backwards from the present state of chip design to deduce that the point in time in which only a single transistor could fit on a chip must have been somewhere in the late 1950s or early 1960s. In fact, the first silicon-based integrated circuit was developed in 1959.

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Clever scientists have taken this idea and applied it to a number of other natural and artificial phenomena that also exhibit exponential rates. For example, genetics researchers Alexei Sharov from the National Institute on Aging and Richard Gordon from the Gulf Specimen Marine Laboratory have used the principle behind Moore's Law to attempt to [determine a timeframe for the origins of life](#).

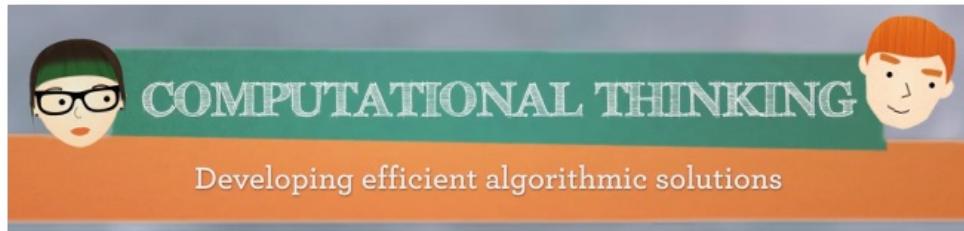


Like Moore and his transistors, Sharov and Gordon have observed that the rate at which genetic complexity in life forms has evolved over time from simple to complex organisms follows a similar exponential growth rate. In their case, they have measured that rate to be a doubling every 376 million years — much, much slower than advances in transistor technology, but still just as consistent.

So what did Sharov and Gordon conclude from their research once they applied Moore's

Law to their data? They determined that, “Linear regression of genetic complexity (on a log scale) extrapolated back to just one base pair suggests the time of the origin of life = 9.7 ± 2.5 billion years ago.” With the age of the Earth estimated to be only about half that age at 4.5 billion years old, their analysis suggests that life as we know it might predate the formation of our planet and opens up new avenues of potential research into the origins of life.

Password Generator Project: Rubric Check



Instructions

This activity provides you and a partner group time to provide one another *critical feedback* about the progress of your projects.

1. Pair up with one another group.
2. Rate the components of your partner group's [Password Generator Project rubric](#). You may write this out or print the rubric and circle components on it.
3. Review your partner group's work and provide them with documentation that describes what you *Like*, what you *Wonder*, and what their *Next Steps* should be.
4. When both you and your partner group have completed the previous steps, share your feedback with one another.

Listen to what your partner group says! You do not need to heed all of their advice, but consider it wisely. The whole idea is to have a critical third party provide ideas to make your project better before you submit the final version for a grade.

UNIT 2

Programming

When used correctly, computational technologies can prove to be extremely powerful and effective tools for solving a wide range of problems. But in order to fully harness that power, an individual needs to be proficient in instructing those tools to perform highly precise operations in well-structured and logical sequences. This unit seeks to ease students into this new, structured, and more formalized way of thinking about problem solving and programming through the use of Scratch, a block-based, visual programming language.

Once introduced to the Scratch platform and programming environment, students will then experiment with a number of basic programming concepts and constructs, such as variables, user input, and selection statements. In the process, students will not only learn how to implement intended functionality by constructing well-designed blocks of executable code, but they will also explore techniques for debugging their code and verifying its correctness.

UNIT PROJECT:

Scratch Program

Highlights

- You will collaborate in pairs to design, implement, and debug a novel, aesthetically pleasing, and intuitive program using the Scratch programming environment.
- You will identify a specific purpose that your program will serve (e.g., entertainment, problem solving, education, artistic expression, etc.).
- You will integrate interactive and multimedia elements into your program.
- You will integrate common programming constructs, such as variables and selection statements, into your program.
- You will test, debug, and correct your program.
- You will use appropriate terminology while writing documentation detailing the full use of your program and its features.
- You will explain your design and implementation choices while demonstrating and sharing your finished programs with your peers.
- You will provide a written analysis of at least one other design team's program, identifying its strengths and weaknesses and offering suggestions for improvement.

Scratch Programming Project

"Software is a great combination between artistry and engineering." – Bill Gates



<https://www.youtube.com/embed/SGtecDQGV-g>

Do you imagine computer programmers to be *scientists?* *artists?* *architects?* *wizards?* The truth lies at the intersection of all four.

Through programming, artists create music and visual art, scientists create models of possible worlds, engineers build new products, medical researchers design and test possible cures, and businesses create jobs and wealth. Even as a beginner, through programming, you have the power to design, to build — to *create*. What will you create?

Assignment

Design and construct a Scratch program, game, or movie that incorporates interaction and media.

Working in pairs, your task is to code a useful application using the Scratch visual programming language. Your program must be interactive and incorporate multimedia (e.g., graphics, animations, etc.). You are free to code a program for whatever (appropriate) purpose you like. Will your program entertain? Solve a problem? Help educate? Be artistic? The possibilities are limitless, though you may want to brainstorm an idea sooner rather than later. That way you can focus your efforts early on and have time for beta-testing and revising your code.

Submission

Your submission will be in the form of an original program that you develop with Scratch, along with written documentation detailing its use. You may submit either a link to the completed program that you code, or download the program itself and submit the program's file itself. You must also submit a text document for your documentation. Your Scratch program must:

- use numeric and string **variables** appropriately, with meaningful names,
- contain **conditionals** to simulate decisions or implement branching,
- implement repetition through both "repeat" and "forever" loop blocks,
- include **documentation** detailing its use, using key terminology from the glossary appropriately and as necessary, and
- be **aesthetically** pleasing, including proper grammar and lack of spelling errors.

When you are finished, you will submit a link to — or the source file of — your Scratch program. Your program will be graded using the attached rubric. You will then review other groups' submissions, and reflect on any differences from your own work.

Good luck and have fun!

Learning Goals

Over the course of this module and this project, you will learn to:

- Apply the concept of "programmability" across multiple domains.
- Create a flowchart/diagram representing a structured process.
- Convert between a flowchart and visual program.
- Apply pre-defined functions over sprite objects.
- Utilize "Broadcast" to pass messages among sprite objects.
- Debug faulty logic via the Instruction Step method.
- Use variables to store and access dynamic content.
- Develop routines that process user input.
- Simulate decision-making through the use of conditional blocks.
- Simulate multi-way decisions through nesting blocks.
- Tailor program output to user input.
- Simulate randomness through the use of a pseudorandom number generator.
- Implement repetition through the use of loop blocks.
- Simulate counting through the use of variables and loop blocks.
- Simulate decision-making through the use of condition-controlled loop blocks.
- Reorder a list.

Rubric

Content Area	Performance Quality

	<p>5 Program uses a combination of four or more numeric and string variables appropriately.</p> <p>—AND— All variables have meaningful names and purposes in the program.</p>	<p>3 Program uses a combination of fewer than four numeric and string variables appropriately</p> <p>AND all variable have meaningful names and purposes in the program.</p> <p>—OR— Program uses four or more numeric and string variables appropriately, but some variables do not have meaningful names or purposes in the program.</p>	<p>1 Program uses only one kind of variable—either numeric or string.</p> <p>—AND— Not all variables have meaningful names or purposes in the program.</p>	<p>0 Not enough criteria are met in order to award any credit.</p>
Variables				
Conditionals	<p>10 Program contains four or more conditionals to simulate decisions or implement branching.</p> <p>—AND— All conditionals are used effectively and correctly with purpose in the program.</p>	<p>7 Program contains fewer than four conditionals to simulate decisions or implement branching</p> <p>AND all conditionals are used effectively and correctly with purpose in the program.</p> <p>—OR— Program contains four or more conditionals to simulate decisions or implement branching but not all conditionals are used effectively and correctly with purpose in the program.</p>	<p>4 Program uses fewer than four conditionals to simulate decisions or implement branching.</p> <p>—AND— Not all conditionals are used effectively and correctly with purpose in the program.</p>	<p>0 Not enough criteria are met in order to award any credit.</p>
	<p>10 Program implements repetition through the combination of</p>	<p>7 Program implements repetition through the combination of</p>	<p>4 Program implements repetition through the use of only</p>	<p>0 Not enough criteria are met in order to</p>

	<p>Loops</p> <p>four or more repeat and forever loops.</p> <p>—AND—</p> <p>All loops are used effectively and correctly with purpose in the program.</p>	<p>fewer than four repeat and forever loops</p> <p>AND all loops are used effectively and correctly with purpose in the program.</p> <p>—OR—</p> <p>Program implements repetition through the combination of four or more repeat and forever loops but not all loops are used effectively and correctly with purpose in the program.</p>	<p>one kind of loop, either repeat or forever.</p> <p>—AND—</p> <p>Not all loops are used effectively and correctly with purpose in the program.</p>	<p>award any credit.</p>
Documentation	<p>5 There is descriptive documentation that explains the program's purpose and major code segments.</p> <p>—AND—</p> <p>There are complete and clear instructions on how to use the program.</p>	<p>3 There is documentation that explains the program's purpose and major code segments</p> <p>AND there are instructions on how to use the program.</p> <p>—OR—</p> <p>There is descriptive documentation that explains the program's purpose and some code segments</p> <p>AND there are instructions on how to use the program.</p>	<p>1 There is documentation that explains the purpose of the program and some code segments.</p> <p>—AND—</p> <p>There are incomplete instructions on how to use the program.</p>	<p>0 Not enough criteria are met in order to award any credit.</p>
	<p>5 The entire program includes aesthetically pleasing colors, graphics and/or multimedia.</p> <p>—AND—</p> <p>There are no spelling or grammatical errors throughout the program.</p>	<p>3 Some of the program includes aesthetically pleasing colors, graphics and/or multimedia</p> <p>AND there are no spelling or grammatical errors throughout the program.</p>	<p>1 Some of the program includes aesthetically pleasing colors, graphics and/or multimedia.</p> <p>—AND—</p> <p>There are several spelling or grammatical</p>	<p>0 Not enough criteria are met in order to award any credit.</p>

	throughout the program.	—OR— The entire program includes aesthetically pleasing colors, graphics and/or multimedia AND there are some spelling or grammatical errors throughout the program.	errors throughout the program.	
Functionality	15 The program functions exactly as described in the instructions. —AND— The program is easy to use.	10 The program lacks some functionality that was described in the instructions AND The program is easy to use. —OR— The program functions exactly as described in the instructions AND the program is somewhat difficult to use.	5 The program lacks some functionality that was described in the instructions. —AND— The program is somewhat difficult to use.	0 Not enough criteria are met in order to award any credit.
TOTAL				50 pts

BIG PICTURE:

The *Who, What, and Why* of Programming

Highlights

- You will examine and discuss the motivations behind a number of high-profile individuals in the field of programming.
- You will discuss the benefits of programming as a tool and a profession.

Who Programs?

Careers that Program

This video, produced by code.org, includes snippets of people in various careers speaking of the importance of programming. The [accompanying page of quotes](#) contains the thoughts of many, many others who have indicated a great need for programming.



<https://www.youtube.com/embed/nKlu9yen5nc>

Choose **one** person from the video or the quotes page that you find interesting in *some* way —perhaps what this person said is interesting, or maybe his/her appearance on a list of people discussing the importance of computer programming is surprising to you.

Discuss with your classmates your choice, including:

- the person's **name**
- what he/she **said**, and
- 2–3 sentences detailing why **you** find this person's perspective on programming interesting.

Why Program?

Why Program?

Learning to program is increasingly important in a variety of fields. In truth, as physical and mental operations become automated, virtually **every** field of human endeavor will be reliant on software and the people who create it.

Throughout history, the economy (and those activities which it drives—such as politics, social mobility, and innovation) has been rooted in resources. In the agricultural era, these resources were typically those that could be produced by farming, ranching, mining, fishing, etc. This was succeeded by the Industrial Age, in which resources such as coal, oil, steel, and mass labor were key. Many refer to the current time as the Information Age, where information, represented digitally, is the major driving force of the economy. In this age, those companies that are able to collect, manipulate, and analyze information (data) are typically based on the work of (and many times, even founded by) computer programmers/software engineers.

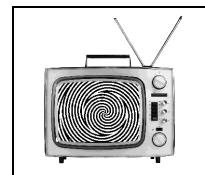


Beyond this, some have speculated that we are on the cusp of a new economic model, the [Imagination Age](#). Whereas programming is integral to function in the Information Age, in the Imagination Age, creativity is the primary resource, and the ability to code will be foundational.

What Is a Program?

What Is a Program?

Carlos: Carla, did you catch that episode of Who Wants To Be a Zombie Hunter last night?



Carla: Pfft, no. I don't watch that! If I could, I would defenestrate our TV.

Carlos: Defene—what? What's the big deal with TV anyway?

Carla: Defenestrate. Look it up. Anyway, TV is just a way for THE MAN to tell you what to do, what to buy, and what to think. Why do you think they call it "television programming"? They are programming you, Carlos; You. are. a. robot.

Carlos: Pfft, man, whatever Carla. You are paranoid.

Carla: No, I'm just on the lookout for *real* zombies.

Compare Carla's connotation of the term "television programming" with these denotations of the word "program" from [dictionary.com](#):

pro·gram

[proh-gram, -gruhm]

noun

1. a plan of action to accomplish a specified end: a school lunch program.
2. a plan or schedule of activities, procedures, etc., to be followed.
3. a radio or television performance or production.
4. a list of items, pieces, performers, etc., in a musical, theatrical, or other entertainment.
5. an entertainment with reference to its pieces or numbers: a program of American and French music.
6. a planned, coordinated group of activities, procedures, etc., often for a specific purpose, or a facility offering such a series of activities: a drug rehabilitation program; a graduate program in linguistics.
7. a prospectus or syllabus: a program of courses being offered.
8. *Computers:*
 1. a systematic plan for the automatic solution of a problem by a computer.
 2. the precise sequence of instructions enabling a computer to solve a problem.

Think About It

How does the *ambiguity* of the word "program" seemingly support Carla's paranoid theories of mind control? What do "mind control" and programming a computer have in common?

Structured Processes

The definitions cited in the previous activity have a few common themes. One such theme is that a program is a story. In fact, writing a computer program is a lot like writing a story. They both have beginnings and ends, and the interesting part is how they move from the beginning to the end.

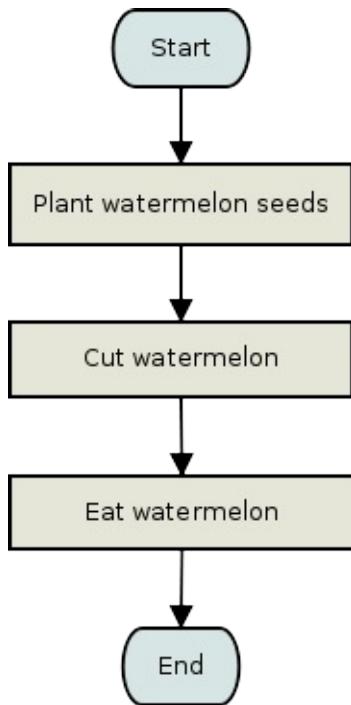
Remember the days of elementary school, where you'd get an assignment like the following?

Sequence of Events

The following images represent various events in a story. Please number each image 1, 2, or 3 according to the logical sequence of events in the story. Follow up by writing a brief description of the scene in its proper order (*at least 1 sentence per event*).



These activities always deal with a specified process, where one event logically follows from another. In this particular example, there is some ambiguity regarding where the process begins. Let's assume that the process begins with the seed packet image. Given that constraint, it only seems logical that our watermelon eater must cut the watermelon before eating it. So, our sequence of events looks like this:



Because programs (like our story) *also* need well-defined beginnings and ends, a number of discrete events, and some indication of the ordering of these events, many programmers use diagrams like the one above (called **flowcharts**) to plan their programs. ([examples](#)) Flowcharts are visual representations of structured processes — encompassing computer programs, stories and scripts, business models, and more.

You may find that flowcharting is helpful to you in planning out the sequence of events in your programs. For many text-based programming languages (like you will see later in this course), a flowchart provides a nice visual counterpoint to the program code itself. However, our programs for this module will be written primarily in Scratch, a **visual programming language (VPL)**, which is structured similarly to a flowchart:

Scratch	Flowchart
<pre> if [touching mud?] then [change speed by -5 v] [move (speed) steps v] end </pre>	<pre> graph TD Start(()) --> Decision{Am I in mud?} Decision -- YES --> SlowDown[slow down 5 MPH] SlowDown --> MoveForward{move forward distance depends on speed} Decision -- NO --> MoveForward </pre>

UNIT TOPIC:

Visual Programming

Welcome to Scratch

- You will utilize a graphical editor to read, construct, and execute dynamic programs.

Programming with Blocks

- You will share and collaborate on your own programs.
- You will examine how well-specified behavior of objects can be constructed through sequential actions and operations.

Remixing Scratch Projects

- You will examine, modify, and execute programs developed by others.

UNIT TOPIC:

Visual Programming

Welcome to Scratch

- You will utilize a graphical editor to read, construct, and execute dynamic programs.

Personal Scratch Pages

Creating Your Page



In order to get started with Scratch, the visual programming language that you will use for the Unit 2 project, you will need to do the following:

1. Save a bookmark for <http://scratch.mit.edu/> in your browser for easy access
2. Click "Join Scratch" and create a Scratch account so that you can save your projects in the cloud, and access all of the features Scratch has to offer.
3. In a shared space provided to you by your teacher, post a link to your public Scratch page. You do NOT have to make all of your projects public, however, your public page will showcase those that you share.
 - E.g., Scratchteam on Scratch page: <http://scratch.mit.edu/users/Scratchteam/>

When you have posted your link, spend some time exploring the user interface and with programming. There are some excellent beginner tutorials under the **Help** tab at the top of the page if you'd like to get started right away. Experiment with the different features available to you now, but don't worry! We will explore *all* of the features necessary to create a great project in detail throughout the remainder of this unit.



Scratchteam

Scratcher | Joined 6 years, 1 month ago
United States

About me

The Scratch Team develops and runs Scratch. As you may know, we're based at the MIT Media Lab.

What I'm working on

Making improvements to Scratch 2.0!

Featured Project



Press 1, 2, 3 for spells
Spacekey to reset

Wizard Spells

What I've been doing

- Scratchteam was promoted to manager of Untitled Studio 1 month, 4 weeks ago
- Scratchteam was promoted to manager of Untitled Studio 1 month, 4 weeks ago
- Scratchteam was promoted to manager of Untitled Studio 1 month, 4 weeks ago
- Scratchteam was promoted to manager of Untitled Studio 1 month, 4 weeks ago

Shared Projects (35)

Pizza Chef by Scratchteam

Snowboarding by Scratchteam

Pong with High Score by Scratchteam

Wizard Spells by Scratchteam

Greeting Card by Scratchteam

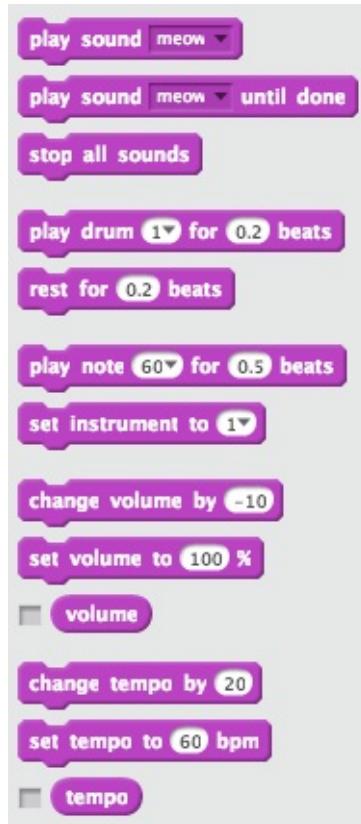
[View all](#)

The Cat's Meow

Your Turn

Enough jibber jabber! Let's do something! For your first program, make a quick song.

Navigate to the Sound tab to access the following blocks:



To create a sound script, simply drag any of the **play** blocks to the script pane. Click on the block(s) in the script panel to execute them.

Take some time to figure out how to connect and disconnect blocks. Be sure to practice removing a piece from the middle of a long script and reconnecting the surrounding pieces.

One at a Time or in Unison?

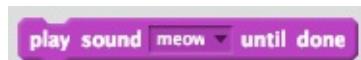
With this brief introduction to the Scratch interface, we examine how sprites and blocks interact and affect one another.

For example, the **play sound** blocks allow us to control when sounds are played as well as *how many* are played.

Consider the difference between these two blocks:



and



The difference, of course, is the phrase "*until done*." Until done with what? Clicking on them individually does not provide enough information to distinguish between the two. So, test them in multiples:

- If you execute this small script, how many meows do you hear?



- How about the following revision — how many meows do you hear now?



- Now, how would you articulate the difference between the two types of play sound blocks?

Experiment

Describe the behaviors exhibited by the following combinations, and provide your answers in a text submission:

- 1) Two `play sound [meow]` blocks and then one `play sound [meow] until done`:



Answer:



- 2) Two `play sound [meow] until done` blocks and then one `play sound [meow]`:



Answer:



3) Given these results—what exactly does "until done" mean?

Answer:



Save Early and Often

Save Early and Often

Few things are more frustrating and disheartening than to see all of your hard work vanish in an instant due to a sudden power outage, random browser crash, or your neighbor accidentally kicking out the power cord under your desk. These unexpected events are always quick and seem to occur at the most inopportune moment — and sooner or later they happen to *everybody*, even you.

Fortunately, the impact of this inevitable disaster can be reduced by frequently saving your work. Just like making frequent use of save points in your favorite game, taking a moment to quickly record your progress can save you a *lot* of time and effort in trying to recreate it after a disaster.

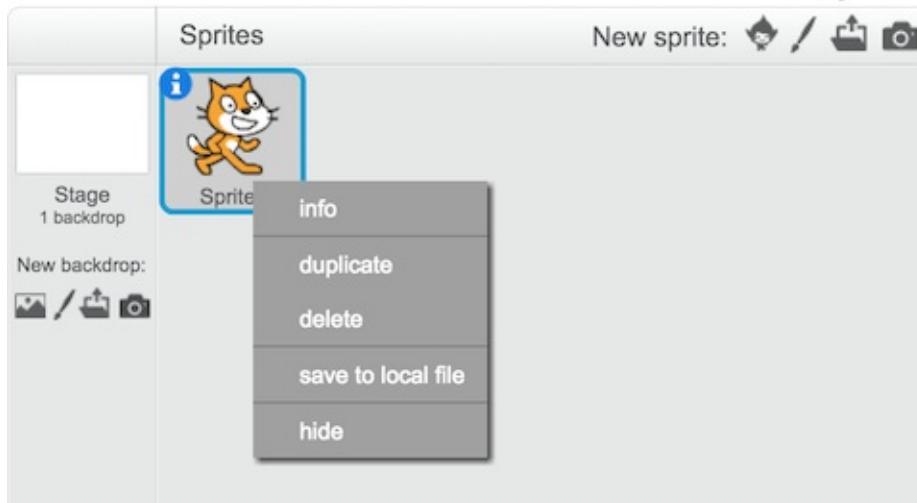
In Scratch, you can save your projects by clicking "Save now" in the "File" menu. While you do not need to save your work after every little change, you should develop the habit of regularly clicking on "Save now" whenever you've made any significant additions to your program and would like to test it.

Exporting and Importing Sprites

Not only can you save your Scratch projects, but you can also save individual sprites and their associated scripts separately. This will allow you to reuse a sprite in multiple projects without creating it from scratch (haha! you got pwned, er...I mean punned) each time.

To **save** (or **export**) a sprite, right-click on the sprite and select "save to local file."

To **load** (or **import**) a sprite, click on the icon with a folder next to New sprite and select the sprite that you want to add to your project.



UNIT TOPIC:

Visual Programming

Programming with Blocks

- You will share and collaborate on your own programs.
- You will examine how well-specified behavior of objects can be constructed through sequential actions and operations.

Experimenting with Play

Events

The `play sound` blocks and short script examples from the last page played in order — beginning with one block and then the next until there were no more blocks to play.

However, we did see that what we actually heard depended on whether we waited until one sound was done playing before we began the next.

Navigate to the Events tab.

The first block (the `when green flag clicked` block) is an event block. Event blocks are triggered by the corresponding described behavior. Scratch provides the **green flag** and **red stop sign** icons in the top right corner of the stage to begin and end the execution of programs. When the **green flag** is pressed, any (and all) sequences of blocks in your program beginning with a `when green flag clicked` block begin executing.

Try it! Connect the `play sound` blocks from the previous activity to a `when green flag clicked` block, then execute your program by clicking the **green flag**.

Broadcast

Another type of event block is tied to the `broadcast` block. This block allows us to have some control over which blocks are executed at a given time by communicating a message. When a message is `broadcast[...]`, the corresponding `when I receive [...]` blocks are executed.

Create the following scripts in Scratch. **Note** that the Cat and the Duck sprites have completely separate script panes. Click on each character to see its script pane. *This allows each sprite to have its own unique behaviors defined.*

Note that the blocks are color-coded according to their function:

- The purple **Looks** tab contains the `say` blocks used in the example.
- Similarly, the brown Events tab contains the broadcast and when blocks.

When you are done, press the **green flag** to start the short play.

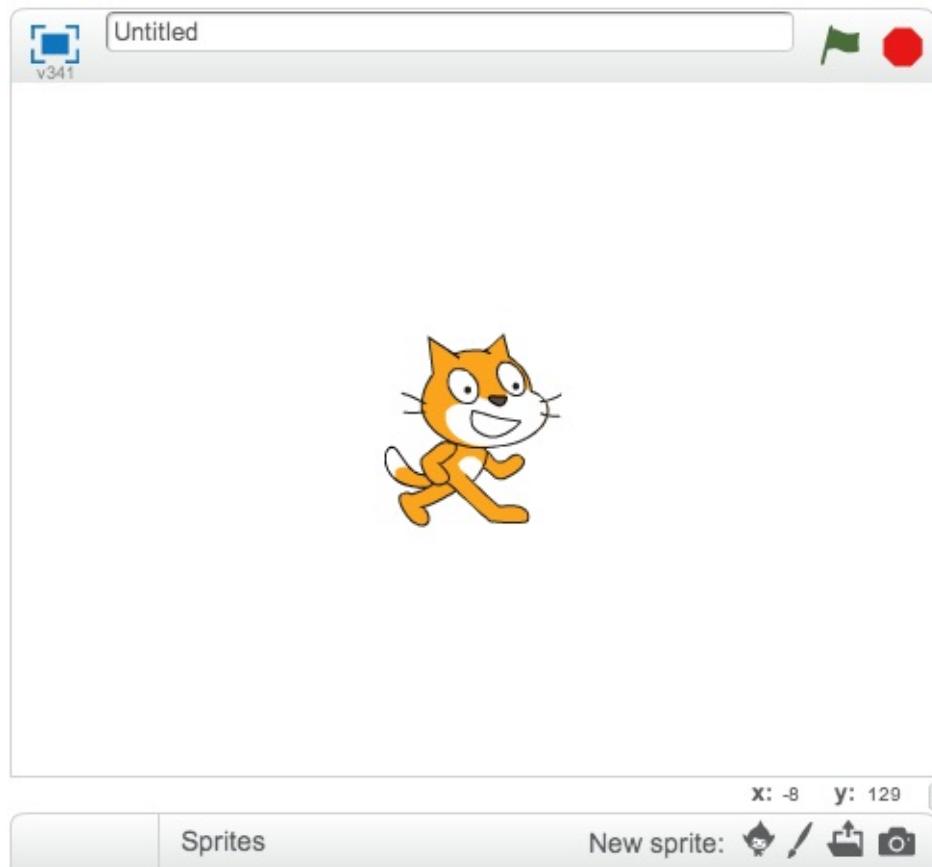
For the cat:	For the duck:



A note about style: You will notice that we chose to give meaningful names to the messages that were broadcast so that it would help us keep track of what we were doing and what messages we were sending. We recommend that you do this in your projects. As your projects grow in size, the number of unique names will grow accordingly. Choosing meaningful names will help you find errors more quickly.

Hints

To choose a new sprite from a library of existing sprites, click on the icon below the stage that looks like a head.



Different Ways to Broadcast

Play Sound and Wait

How many times will you hear the meow sound when you run the script below?



Answer:

Play Sound without Wait

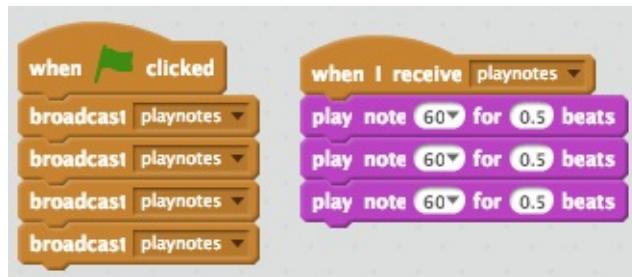
How many times will you hear the meow sound when you run the script below?



Answer:

Broadcast without Wait

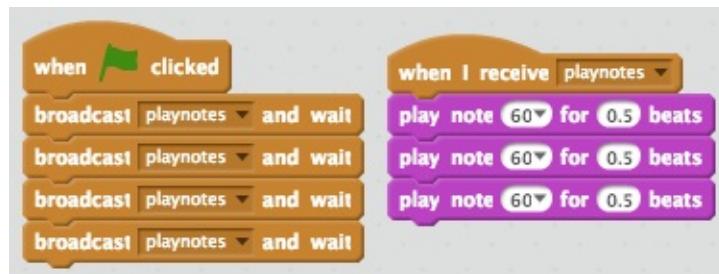
How many times will you hear the note 60 play when you click on the green flag?



Answer:

Broadcast with Wait

How many times will you hear the note 60 play when you click on the green flag?



Answer:

Chain of Broadcasts

How many times will you hear the meow sound when you click on the green flag?



Answer:

Mismatched Broadcasts

How many times will you hear the meow sound when you click on the green flag?

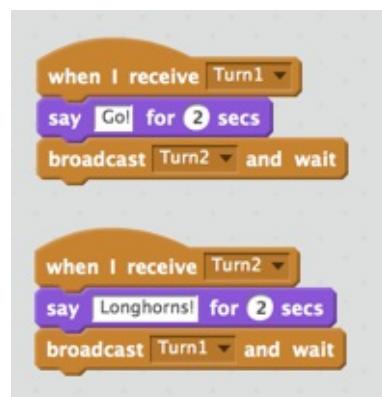


Answer:

Loops

Without trying this in Scratch, answer the following question:

What does the following set of scripts for the Cat do when you click on the **when I receive [Turn1]** block?



Answer:

UNIT TOPIC:

Visual Programming

Remixing Scratch Projects

- You will examine, modify, and execute programs developed by others.

Remixing Scratch Projects

Remixing

Scratch is available free of charge and runs in your web browser. Additionally, the Scratch site contains forums, tutorial videos, and a variety of helpful resources. You should spend some time exploring the site, and these resources. For this activity, you will *explore* other users' Scratch programs and remix one to make it your own.

A Scratch account allows you to access many useful features, including the ability to share, discuss, and modify other users' projects in a collaborative manner. Using the **Explore** tab in the main menu bar, you can search for and select from a wide variety of projects that other users have created. Not only can you click "See inside" to get a look at how the project works and how it was created, but you can also choose to "Remix" the project to create your own copy of the project and begin modifying it in whatever creative ways you can imagine.



The Scratch developers feel that sharing, reading, and writing code are all important aspects in the learning process. Therefore, any Scratch program shared on the site is licensed under the [Creative Commons Share Alike](#) license and may be copied by any other user and used as the basis of a new, derivative work.

In this activity, you will browse projects on the Scratch site, select one that interests you, and remix it in some way.

1. Navigate to the [Scratch website](#), and log in to your Scratch account.
2. Choose the **Explore** tab on the menu bar of the Scratch front page.



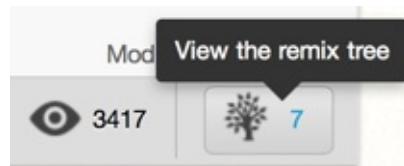
3. Find a project that interests you. When selecting a project, try to imagine how you would like to modify it.
4. Click the button labeled "See inside."



5. Click the button in the top right corner labeled "Remix."



6. You now have a project copy of your own to freely modify. Any changes made by you will only affect your copy! Some projects have many, many versions that have been modified from an original. In order to see how a project has evolved and diversified over time, click the "Remix Tree" button on the project's page.



7. In a shared space provided to you by your teacher, post links to **both the original project and your remix, as well as a description of the changes you made.** Explore your classmates' remixes!

CODING SKILLS:

Choreography Notation

Highlights

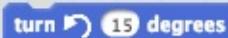
- You will examine non-traditional forms of domain-specific notation.
- You will design and construct instructions using a non-traditional, domain-specific notation.
- You will analyze the clarity and legibility of instructions written in a non-traditional, domain-specific notation by reading and executing instructions created by others.

Let's Dance!

Motion

In Scratch, sprite behaviors are a form of program output. Up until now, we've only "heard" output in the form of **play sound** blocks. However, sprites can do much more than sing (or meow).

The programming blocks located in the **Motion** tab will affect the sprite's state — its location (i.e., where it is) and/or orientation (i.e., which way it's *facing*). Two of the most commonly used of these blocks are the following:

Moving	Turning
	

Two important things to note about these blocks:

1. The blocks instruct the sprite to move relative to its state where it is before the block is executed. In other words, you can click on the sprite and drag it somewhere, and **move [10] steps** will still function correctly and move the sprite 10 steps from its present state. The same is true for **turn** blocks — no matter which direction the sprite is currently facing, **turn [15] degrees** will still function correctly.
2. The number of steps or degrees is customizable. The "holes" in the programming blocks, where a selection can be made, are called parameters. Scratch will allow negative parameters in these motion blocks.

Hypothesize what the following will do:



Implement it and test it out. Were you right? What effect does turning a negative number of degrees have on the sprite?

Motion Combinations

Motion blocks can be combined much like Sound blocks. Hypothesize and test what the following sequence of blocks will do:



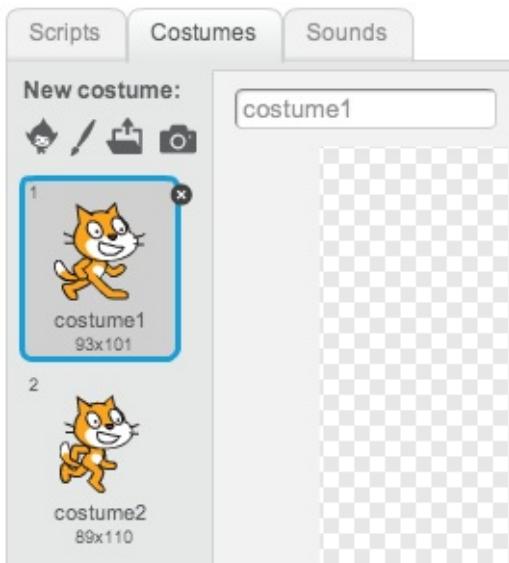
Extension

Try to figure out the relationship between the blocks on the left and the buttons on the right. These buttons can be helpful to get the characters to face each other.

Blocks	Buttons
<pre>turn (15 degrees) turn (15 degrees) point in direction (90°)</pre>	 <div style="border: 1px solid black; padding: 2px;"> Sprite1 x: 0 y: 0 direction: 90° rotation style: ↗ ↔ ⚡ can drag in player: <input checked="" type="checkbox"/> show: <input checked="" type="checkbox"/> </div>

Sound and motion need not be separate. Sequences of blocks that combine the two (and other types of behaviors) can be created by switching among the tabs and selecting the blocks you wish to combine. The example dance sequence uses four types of blocks:

- The brown Control block handles the green flag event, so the program knows when to start and what to do once it has started.
- The magenta Sound blocks output audio drum beats.
- The blue Motion blocks change the sprite's location and orientation.
- The purple Looks blocks change the sprite's appearance.
 - Each sprite may have multiple 'costumes' associated with it. The default 'Scratch the Cat' sprite has two—you can view them by clicking on the 'Costumes' tab, then you can **switch** to a particular costume by name or simply choose **next costume** to alternate between them.



Instructions

Your job is to remix some starter Scratch code to make it a dance all your own!

1. Open a new Scratch project.
2. Recreate the following Scratch starter code in your Scratch project.
3. Modify the code and remix it into your own dance!
4. Share your Scratch dance with a classmate. Describe the modifications you made, and give your dance a name.



Choreography

Dancing Like a Programmer

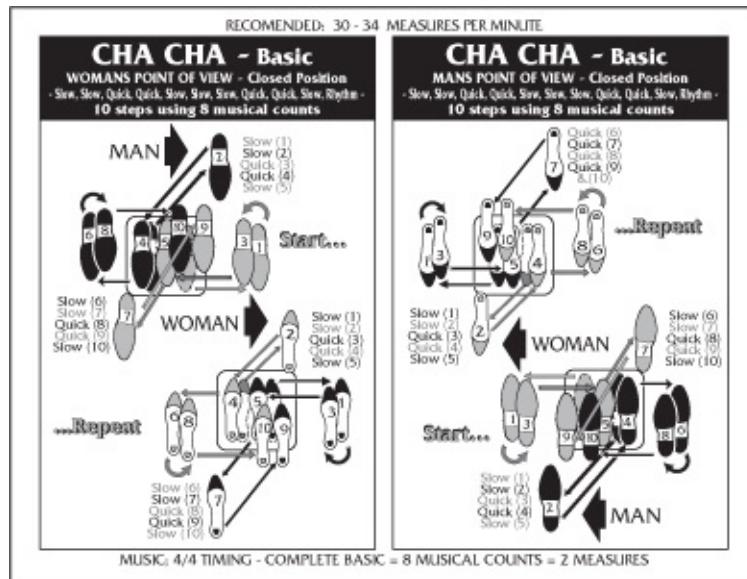
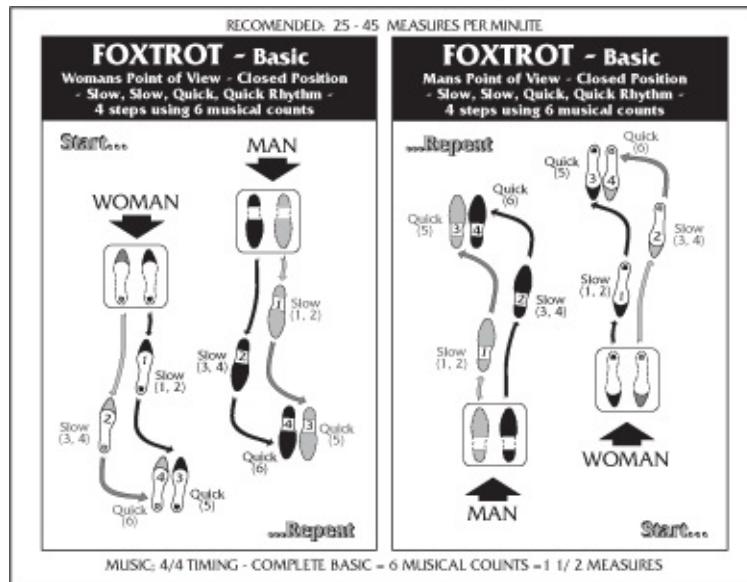
Choreographers are programmers, too!

A choreographer creates dance routines. These routines must be captured in some form of notation so that dancers may synchronize with one another and maintain the same experience across multiple performances and venues.

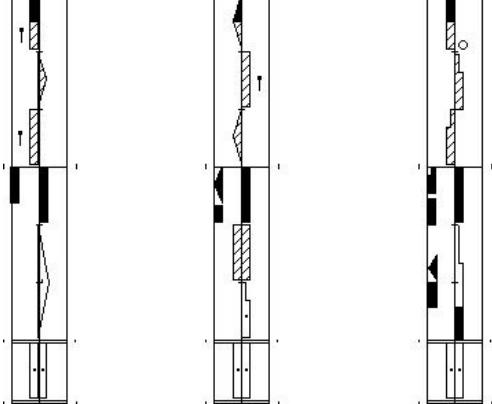


Below are examples of two types of dance notation:

1. Informal Footprints



1. A formal notation — [Labanotation](#)

Rudolf Laban	Labanotation
	

[Laban Lab — Interactive Labanotation Tutorials](#)

Instructions

1. Research different types of dance notation online.
2. Choose one, and use the dance notation to choreograph and document a simple dance.
3. Record the choreography using the notation on a piece of paper.
4. On the back of the paper, write a reflection of the process including the following:
 1. a description of your dance in simple English,
 2. which notation system you used and why, and
 3. how this notation system is both like and unlike Scratch.

Your classmates will try to replicate your dance based on your notation! Good luck.

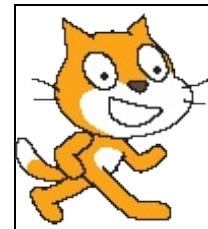
Animated Movie

Activity

In this activity, you will make an animated movie using Scratch. The topic of the short film may be anything appropriate, but be wise about how to allot your time (e.g., don't focus too much on any one aspect while neglecting others).

To receive full credit, your movie should have *at least*:

1. Two motion blocks.
2. Two different characters (AKA sprites).
3. Four total broadcasts so that the characters interact with one another.
4. Two different sounds.



Be creative! Create characters, tell a story, and experiment with moving sprites around the stage. *Program* your own movie.

When you have finished, post a link to your *Animated Movie* in the shared space provided to you by your teacher.

Rubric

Criteria	Points
The program contains two <i>motion</i> blocks.	2 pts
The program contains two different <i>sprites</i> .	2 pts
The program includes four <i>broadcasts</i> between sprites.	4 pts
The program produces two sounds through <i>play</i> blocks.	2 pts
TOTAL	10 pts

UNIT TOPIC:

Program State

User Input

- You will write programs that incorporate dynamic, user-driven, keyboard controls and input.

Variables

- You will examine how the dynamic state of an object or program can be stored and changed using variables.
- You will analyze the role of clear, descriptive names for objects, behaviors, variables, and other identifiers in maintaining the readability of code.

UNIT TOPIC:

Program State

User Input

- You will write programs that incorporate dynamic, user-driven, keyboard controls and input.

Stored State

The State-of-Being State

State can be such a strange word. In computer-speak, it corresponds roughly to everything about the computer at a given moment — everything that is in memory, what programs are executing and what instruction they are currently processing, signals to and from input and output devices, and so on.

In this course, "state" encapsulates *attributes* — a *description* of what the computer is doing at any given moment.

Input and Storage

Up to this point, we have created programs that deal with the following computer functions:

- Input
- Storage
- Processing (We instructed the computer to make calculations, such as adjusting location and orientation, or count beats per minute for synthetic drum beats.)
- Output (Our focus has been primarily centered on output — observable behaviors generated by the computer such as animation, sound, and text.)

To be fair, our programs *have used* input and storage to some degree. For example, clicking the green flag in Scratch is a form of input. We interact with this control to signal the computer to begin the program.

Our use of storage to this point is more abstract. Consider the what the computer needs to "remember" in order to execute the program.

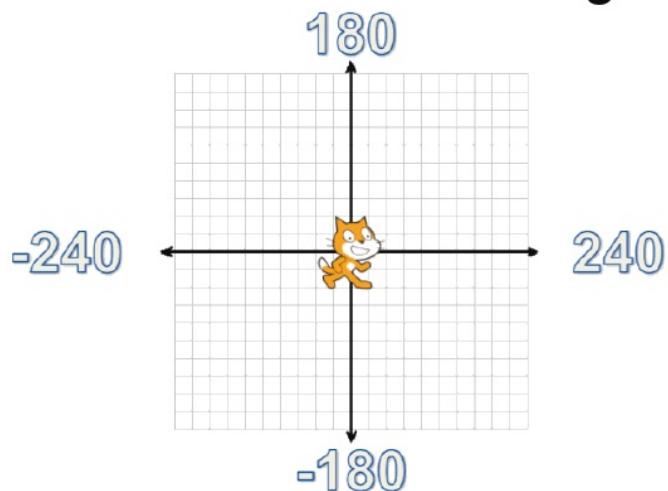
First, it needs to "remember" the program — *what are its instructions?* Second, there are attributes about the environment it needs to "remember" in order to function properly. These attributes are called **state**. An example of state in our previous programs is the location of Scratch the Cat. In order for the program to `move [10] steps`, it needs to "remember" where Scratch the Cat is actually located. However, these instances of input and storage are not ones we explicitly control. During our next few activities, we will focus on *customized* input and storage.

Position

The sprite occupies a point (x,y) on the stage corresponding to the x- and y- axes below.

How is a sprite's state related to its position on a coordinate grid? How are *position* and *state* different?

Position on the Stage!



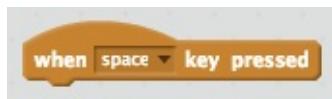
In Scratch, the sprite usually begins at the center of the stage. Its position is (0,0).

User Input and Interaction

User Input

User input for programs can come in many forms. Most often, a user interacts with a program by clicking or moving a mouse, or through typing via a keyboard. In this way, a user can direct a program to behave in a certain way *without* predetermining the data when programming. In the following activity, you will program Scratch the Cat to react to user input.

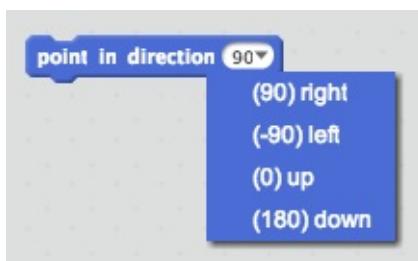
In order to do that, we will use the `when event` blocks located in the "Control" tab.



The event trigger can be changed using the drop down menu. **Create 4 separate event blocks corresponding to each of the arrow keys.**

Moving with the Arrow Keys

Now that our program is anticipating four different key presses as possible inputs, let's instruct Scratch to react to these inputs. We can use the block `point in direction [. . .]` located in the **Motion** tab as shown below:



Instructions

Your job is to create a program with a sprite that moves related to user input, including arrow keys and at least two other inputs.

Program an original program in Scratch that meets the following *minimum* requirements for your Scratch program:

1. Create a new Scratch program with at least one sprite.
2. Create four separate `event` blocks corresponding to each of the arrow keys.
3. Add the appropriate `point in direction [. . .]` block to each `event` block you created above. NOTE: The Cat should react to each key press by facing the proper direction.
4. Attach a `move [10] steps` block to each of your events. Note: The Cat should move around based upon the arrow keys pressed.
5. Add two other `event` blocks that react to different user input (e.g., key presses, mouse movement/clicks)

6. Personalize it in at least three other ways.
7. Provide documentation for your program (describe what it does) as the Instructions. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself.

Show Me Your State

Show Me Your State

We've explicitly dealt with input now. In an example program chunk of the previous activity, we used the `when up arrow key pressed` block to detect our input, and then move your sprite upward. But what does *upward* mean?

We know what the output (the observed behavior) of upward is: when the computer executes its commands, it redraws the screen with the sprite 10 pixels above his previous location. How does the program know where to draw the updated Scratch and which direction to face him?

The computer needs to "remember" the sprite's original location and orientation, its state, and update them with new ones — by accessing and modifying storage. Examine the following instructions; locate the attributes (of the Scratch sprite) we are changing:



We are changing the sprite's direction attribute as indicated by the `point in direction [0]` block. However, we are also changing his location with the `move [10] steps` block. "10 steps" is not an attribute of a sprite—what attributes do you hypothesize indicate *location*?

Scratch uses the familiar x-y coordinate plane to indicate location. This means that each sprite has three attributes associated with it for location and orientation: `direction`, `x position`, and `y position`. You can view these attributes and their associated values by checking the following boxes in the **Motion** tab:

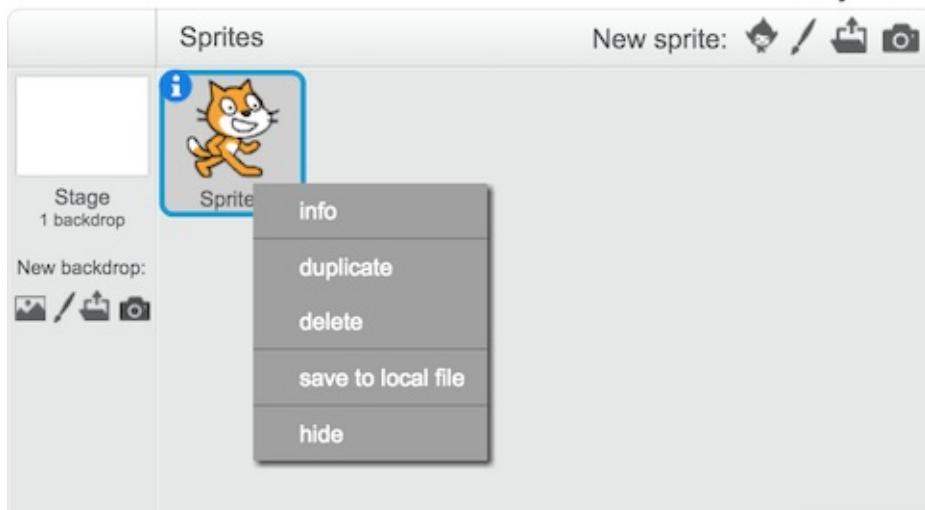


Try It Out!

Load your Scratch program from User input and interaction.

Experiment with location and orientation using the `direction`, `x position`, and `y position` blocks. Move your sprite around and observe how its state changes over time.

For a more detailed view of a sprite's state, you can right-click the sprite in the sprite picker, and select "info."



Text Input

Text Input

Let's use both the changeable and placeholder qualities of variables to make a program in which Scratch interacts with us. We've already used text output in Scratch with the `say [. . .]` blocks located in the **Looks** tab. However, we can also ask for text input as well. Many of the input-oriented blocks are located in the **Sensing** tab.

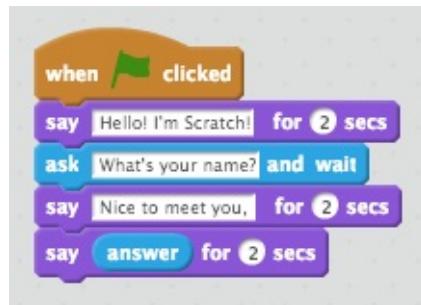
Why do you think that the **Sensing** tab contains blocks for input? How do *you* receive input from the world around you?

In the following activity, we are going to work with the following blocks:



As you can see, the variable `answer` is changeable. It has a different value at different times, depending on its context — when it is being viewed and how it was updated.

What about the placeholder quality of variables? How can we leverage that in our programs? We can use `answer` as a placeholder for text input in our program. This way, the program will use whatever is entered for `answer` without knowing ahead of time what that is. This is better illustrated with an example. *Create the following program. Remember that blocks are color-coded according to their tabs.*

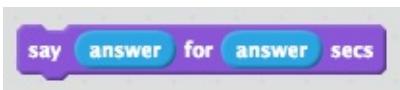


Try It Out!

Load your Scratch program from [User Input and Interaction](#), [Show Me Your State](#), and make the following edits:

1. Drag the `ask [what's your name?]` and `wait` block to the Scripts pane. Click on it. Scratch will ask your name and wait for you to respond.
2. If you'd like to see your answer, check the box next to `answer` as is done in the image at the top of the page. The variable `answer` contains whatever you type when Scratch asks you a question using the `ask [. . .]` block.
3. Add another `ask [. . .]` block immediately after the "What's your name?" block.
4. Change the question text, and re-execute the program.

5. Now execute the program. What happens and why?
6. Drag the **answer** block to your script again so that it reads



```
say [answer for answer secs]
```

Does this do what you would expect to? Why or why not? What would you need to do to make it work properly?

7. Personalize it in at least three other ways.
8. Provide documentation for your program (describe what it does) as the Instructions. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself.

UNIT TOPIC:

Program State

Variables

- You will examine how the dynamic state of an object or program can be stored and changed using variables.
- You will analyze the role of clear, descriptive names for objects, behaviors, variables, and other identifiers in maintaining the readability of code.
- You will integrate randomness into a program through the use of the Random Number Generator.

Variables

Changeable Placeholders

The attributes we use to describe Scratch the Cat's location and orientation are called **variables**.



Wait a minute! We've seen that word before in countless contexts! Let's recall two forms of variables:

Variables in Algebra I

Remember Algebra I and its most famous denizen, x ? That li'l guy is everywhere from

$$\begin{aligned} 2x + 3 &= 15 \\ \text{to} \\ x &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \end{aligned}$$

Its primary purpose is to act as a *placeholder* for a value that was unknown. The point of most Algebra I problems is to figure out just exactly what x is a *placeholder* for—a kind of a mathematical hide-and-seek game.

Variables in a Science Fair Project

The Science Fair always ends with a cafeteria full of tri-fold cardboard displays, but a good science fair project always begins with a hypothesis and an experiment to test it. This picture is the beginning of such an experiment; plants were given distilled water, tap water, brackish water, and salt water for daily waterings. The hypothesis might be that the purer the water, the better the growth. The experiment is designed to test that conjecture.



This is done by establishing controls and variables. If you want to make sure that any difference you see in plant growth is due to the type of water you give them, you have to make sure that they receive equal treatment in every other possible way — the same amount of sunlight, the same type of soil, etc. These constant factors are controls, whereas the water is the variable.

In this sense, a variable is a quantity (or quality) that changes. In other words, what it represents in one scenario may be completely different in another.

Programming Variables

Variables in computer programs are similar to *both* of the types of variables previously discussed — they are both placeholders and changeable quantities. Let's look at examples for each characteristic:

Placeholder

Scratch the Cat is minding his business when suddenly a voice from above tells him `turn ← [15] degrees`! We know from examining his attributes that means subtract 15 degrees from his `direction`. Or, perhaps the voice tells him `move [10] steps`. This one is more complicated because it involves modifying both `x position` and `y position` and it depends on which `direction` he is facing. However, to some degree, it doesn't matter, because the variables are an *abstraction*. Essentially, `turn ← [15] degrees` means subtract 15 degrees from `direction` — **whatever it is**. The variable `direction` acts as a placeholder.

Changeable Quantity

Consider `direction` in the example above. As you rotate Scratch the Cat, the `direction` variable changes. Note how this is different than variables in algebra equations. In $2x + 3 = 15$, x is always 6. There is one correct answer for "what is x ?" However, unlike algebra equations, computer programs are dynamic things; they change over time. Scratch the Cat moves, turns, and meows. All of this possible because we are able to change the variables that define him.

Try It Out!

Load the Scratch program from User input and interaction (and Show Me Your State), and edit it as described below.

Check the box next to `direction` in the **Motion** tab. Now, an indicator will appear on the screen with the label `direction` and a value. Continually click the `turn ← [15] degrees` block and track how the `direction` changes. In the information pane for the sprite, you can freely rotate it. Notice how `direction` is updated. Clicking on `turn ← [15] degrees` once more will still work as intended, because it merely subtracts 15 degrees from `direction` — **whatever it happens to be**.

Names Are Important

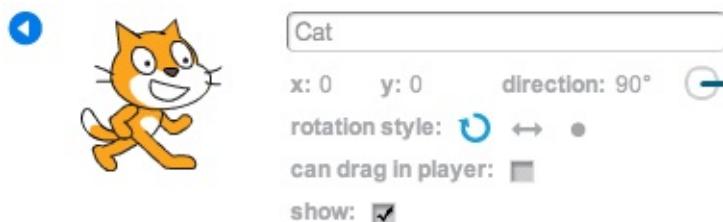
Names Are Important

"What's in a name? That which we call a rose by any other name would smell as sweet..." — William Shakespeare, Romeo and Juliet

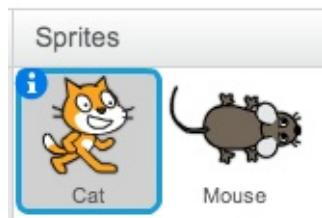
Add a second sprite that is controlled by a different set of keys — use **W** for up, **A** for left, **S** for down, and **D** for right. At this point, you should have two sprites that can move independently of one another. *Remember that sprites can be imported using the folder button below.*



We want to be able to refer to our sprites with intuitive names. By default, your sprites will probably be called something like **Sprite1** and **Sprite2**. Rename them by clicking on the sprite and typing into the text box, as shown below:



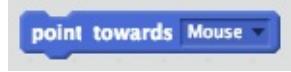
Do the same for both of your sprites, and you should have something that resembles:



Names are important! Naming our sprites **Cat** and **Mouse** will make programming with them much easier. Just as we used **answer** as a placeholder to refer to what the user types in, these names that we've given our sprite characters can be used as placeholders to refer to them. Giving them meaningful names makes keeping track of which is which much easier.

Try It Out!

In your first sprite's script pane, drag the **point towards [. . .]** block into the script pane. In the drop down, you'll notice that one of the options is your second sprite (e.g., Mouse). If we had kept the name **Sprite2**, that is what would have been listed. Select your second sprite so that the block now resembles:



Click the block. Move your second sprite by clicking and dragging it somewhere else. Click the block. Move your second sprite. Repeat.

Discussion Questions

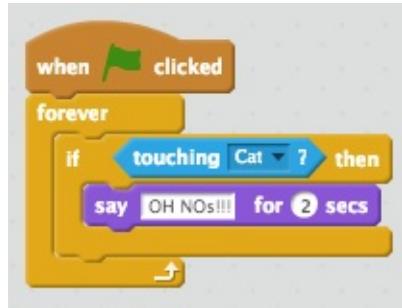
1. Describe how the name **Mouse** is a placeholder for the sprite associated with it.
2. How would the program be different if we named the sprite **Mus musculus** instead?
3. Imagine the following analogous scenario:
 - You are asked to face your teacher during this class. Stop and imagine what would happen if you followed this instruction. Who would you be facing? Which direction?
 - Now, imagine that you go to your next class, and you receive the same instruction. Who would you be facing? Which direction?

Likely, the context of the situations are different, so you will be facing two different directions and two different people. *How is this representative of the variable nature of the name "teacher"?* Also, does "face your teacher" have the same meaning for all students?

Game of Tag

Tag!

Let's make use of these names we've given our sprites to make them interact with each other. We want one of the characters to react if the two characters touch. In the last step, we showed an example with Cat and Mouse. We can add following script to the Mouse sprite to say "OH NOs!!!" when it touches the Cat.



Note: One really important thing to note here is that once we click the green flag, the script above will **always** be running: notice that it always has a yellow outline.

Compare this to the script below that would only run once when you click the green flag. The functionality above is often called an infinite loop, and can be very helpful when we want something to continue running forever.



Try It Out!

Load your Scratch program from [Text Input](#) and [Names are Important](#).

Replicate the two commands illustrated in the screenshots above using your own sprites. **Remember that blocks are color coded according to their tabs.** Experiment with these and other codes to make your program unique.

Randomness

Random Number Generator

We can generate a random number in Scratch between any two numbers (inclusive) using the **pick random** block:



This will report an integer 1, 2, 3, 4, 5, 6, 7, 8, 9, or 10 — each with 1/10 probability. This type of block (with the round edges) is a **reporter** block — it *reports* a value. We can use this block inside other blocks that take a value:



In other words, a reporter block can be used in any case that a manually entered value or variable can. You can even use **pick random** blocks as the parameters to other **pick random** blocks:



What do you think this complex combination of blocks *does*?

Instructions

Your job is to make a sprite move around the screen randomly. Load your Scratch program from [Game of Tag](#), and adjust it as follows:

1. Make one character move around randomly, but still stay within the confines of the screen. You can do this with the **if on edge, bounce** block.
2. Use **pick random** to change the second sprite's appearance. You can vary many different factors in the "Looks" tab, including size, costume, as well as many colorizing and distorting effects.
3. Be personalized in at least three ways.

4. Be usable, efficient, and effective.
5. Provide documentation (describe what it does) in the Instructions. Be sure to describe how your program is original.

Custom Variables

Customization

If Scratch were limited to the three variable *attributes* of `x position`, `y position`, and `direction` that the interface provides, manipulating sprites on the screen would become boring very quickly. Anyone who has played a [role-playing game \(RPG\)](#) is familiar with a variety of attributes and skills typically available to characters. Because Scratch allows the creation of custom variables, we can create new attributes for our characters.

Your current [Game of Tag](#) program dictates how fast each character moves (e.g., 10 steps per movement). We could instead vary each sprite's speed according to its "skill level." In order to do that, let's create a new variable called `speed`:



In the **Variables** tab, click `Make a Variable`, and enter `speed` as the name. We will want each sprite in our program to have its own speed, so make sure that the button `For this sprite only` is selected. Create one `speed` variable for each character. What would happen if `For all sprites` were selected? *Experiment!*

The **Variables** tab supplies two blocks to change the value of `speed`:



Now that you know about `speed` variables, make use of them for your [Game of Tag](#) program. For each `move [10] steps` block, replace the numerical value `10` with the variable `speed` by dragging it from the **Variables** tab into the slot.

For example, you should have something resembling the following:



Instructions

Add the following functionality to your [Game of Tag](#) program:

1. When the green flag is clicked, have each sprite ask for an initial speed. Set the **speed** of each according to the **answer** given.
2. When the second sprite is caught (e.g., when the cat touches the mouse, and the mouse says "OH NOs!!!"), increase the second sprite's **speed** by **5**.
3. Pretend that the first sprite is a vertical kind of guy—whenever the first sprite moves up or down, increase his **speed** by **1**. Whenever he moves right or left, decrease his **speed** by **1**.
4. Make your **speed** variables visible on the screen so you can test your program to ensure it works correctly.
5. Personalize your program in at least three other ways.
6. Provide documentation for your program (describe what it does) as the Instructions. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself. Your work will be reviewed by a peer, and in turn, you will review one of your peers' projects. You should base your evaluation on the assignment rubric.

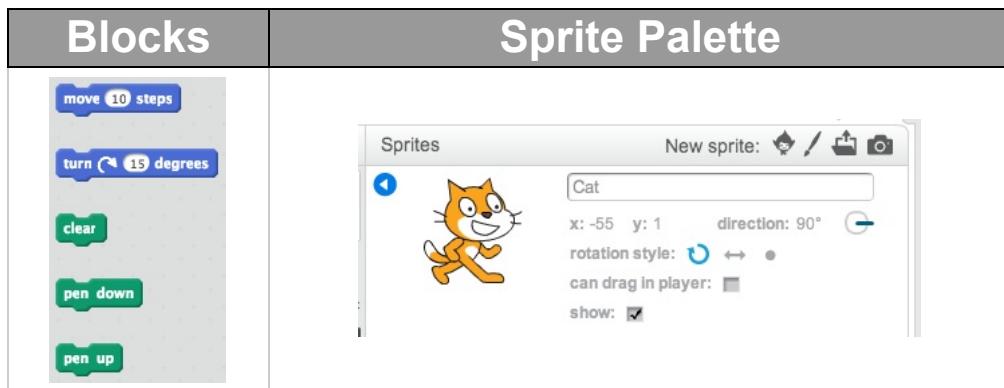
Rubric

Criteria	Points
Characters move properly	2 pts
Mouse says "OH NOs!!!" when characters touch	2 pts
Speed is determined by user input	2 pts
Mouse speed is conditioned on collisions with Cat	2 pts
Cat's speed is conditioned on directional movement	2 pts
Displays speeds on the screen	1 pt
Documentation, usability, and personalization	1 pt
TOTAL	12 pts

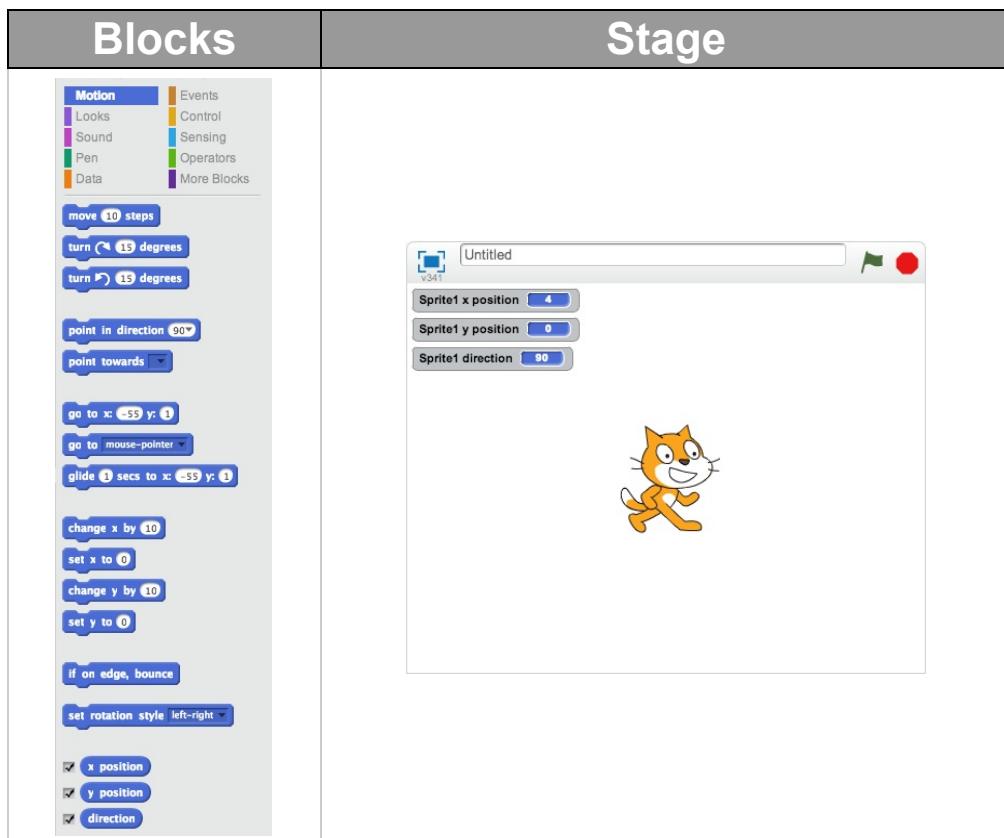
Drawing Commands

Experiment with Drawing Commands

Experiment with the pieces shown below from the **Motion** and **Pen** tabs. Figure out what each one does, use the pieces below to draw a square. To make your character smaller so that it is easier to see where it is drawing, alter the size attribute using blocks located in the **Looks** tab. Make sure that you select the option for "rotation style" so that the character *freely rotates* (as seen below).



Use commands to show the **direction**, **x position**, and **y position** of the character on the screen when the program runs. Remember, in the **Motion** tab you can select these to be shown on the stage (as shown below).

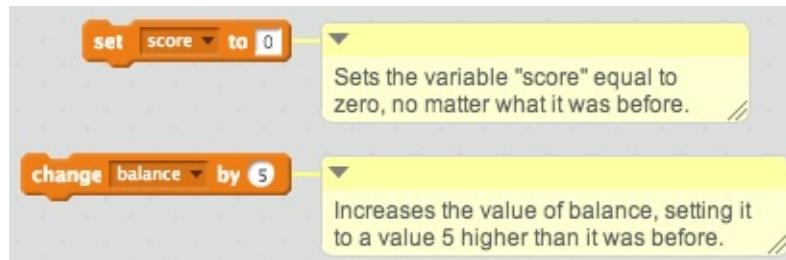


Reviewing Variables

Change vs. Set

The previous assignment was aimed at getting you comfortable with using **variables** to store values, giving you power to do things you couldn't without them. Variables are one of the central concepts in computer science and one of the most powerful tools in a programmer's toolbox. They will pop up steadily throughout the rest of the course.

These are two blocks that we will be seeing a *lot* in Scratch programs. Make sure that you are clear on the differences between them.



You can use these two blocks to accomplish the same thing, but it probably makes more sense to use one over the other depending on what you are trying to do.

- If you are trying to set the value relative to what it already is (such as adding \$5 to a bank balance), you will probably be better off using a **change** block.
- If you are trying to set it to a totally new, unrelated value (such as resetting a score in a video game), then you will probably want to use a **set** block.

Consider this:

change and **set** blocks are analogous to **turn [. . .] degrees** and **point in direction [. . .]**. How are they similar? Be prepared to discuss your thoughts in class tomorrow.

UNIT TOPIC:

Selection Statements

"if...else" Statements

- You will examine the uses of selection statements in programming.
- You will analyze the differences between simple selection and complex, nested selection statements.

Quiz Show

- You will examine the use of the Boolean operators "AND," "OR," and "NOT" in constructing complex conditional statements.

UNIT TOPIC:

Selection Statements

"if...else" Statements

- You will examine the uses of selection statements in programming.
- You will analyze the differences between simple selection and complex, nested selection statements.

Decisions

Decisions, Decisions...

*"Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference." —Robert Frost*

It may not seem like a free-willed thinking agent, but our previous program made a **decision** —sort of. It chose between two possible actions, either to increase Mouse's speed or to not increase Mouse's speed. It made this decision many, many times depending on whether Cat and Mouse were touching.

This type of "decision" is made by using the conditional **if** block. Much like the word "if" works in English, the conditional block only executes the blocks inside of it *if* something —some *condition* — occurs.

Think about the following English phrases:

- "If you clean your room, I'll give you \$10."
- "If the bond measure passes, the new stadium will be built."
- "If you stay on the yellow brick road, you'll reach Emerald City."

Contrast these with the instruction blocks we've created up to this point, which have been more like imperative statements:

- "Give me \$10."
- "Build the stadium."
- "Travel to Emerald City."

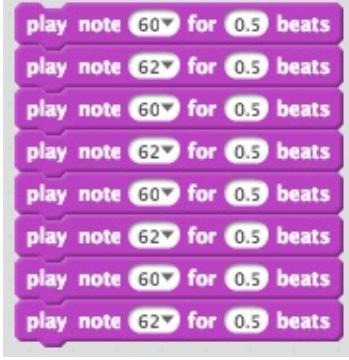
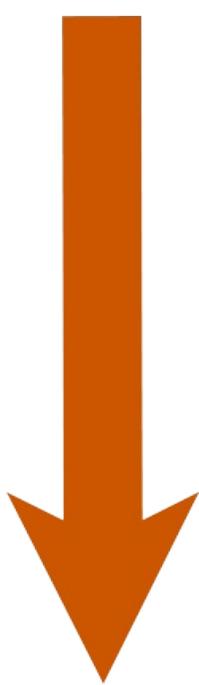
"I went into a clothes store and a lady came up to me and said 'if you need anything, I'm Jill.' I've never met anyone with a conditional identity before." — Demetri Martin

If Blocks

The most basic type of conditional block is the **if** block. Sometimes, computer scientists refer to these types of instructions as **branching**. The following diagrams provide a visual explanation:

Consider a basic computer program much like the ones we've written up until now. The computer executes one instruction, then moves on to the next — the one immediately following it:

	Direction of Flow
--	-------------------

	Direction of Flow
	

The arrow indicates the **control flow**, or how the computer moves from instruction to instruction over time. In other words, it starts at the top (the beginning) and moves downward (chaining instruction after instruction) from there, until it reaches the end. **Note:** Notice that the **if** block is contained in the **Control** tab — these are blocks that explicitly deal with flow control.

The **if** block however represents a *choice*. If some condition occurs, then the control flow *branches off* down a different path of instructions. If the condition does not occur, then the flow occurs as normal:

Instructions	Direction of Flow
	

In this example, if the sprite is **touching [mud]**, then its **speed** slows down by 5. Regardless, it moves **speed** number of steps afterward. The condition of **touching [mud]** just has an effect on the **speed** of the movement.

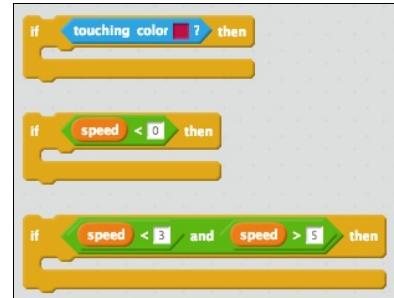
Scratch allows for easy visual indication of which blocks are affected by an **if** block; any block enclosed in the yellow frame of blocks headed by the **if** is executed only *if* the condition is met. Otherwise, they are simply skipped over.

Binary Conditions

Consider the "condition" in an **if** block. What kinds of conditions can be used in a program? Much like other aspects of computer science, it boils down to a bit (a dichotomy, a yes/no question, on or off) — a condition has two possible outcomes, either it is met or it is **not** met. In the Representation module, we will spend a great deal more time examining the power inherent in dichotomies.

Scratch "shape-codes" these binary values (also called **Boolean** values) as hexagons. There are many valid hexagon-shaped blocks available to fill the hexagon-shaped hole in an **if** block. Examine the following:

1. The first is a **Sensing** block much like the ones we used in our Game of Tag. Either the sprite is touching the reddish color indicated, or it is not — there is no in between.
2. The second is an **Operator** block. The **Operators** tab contains arithmetic operators like +, -, ×, and ÷, but it also contains *Boolean operators*, like <, >, **and**, **or**, **not** — operators that essentially have two outcomes (yes/no, true/false, etc.).
3. The third uses multiple operators to define a more complex condition. Note that there is nothing preventing the condition from being *silly*, just as long as the condition can be true or false. When is the condition true and when is it false in this example?



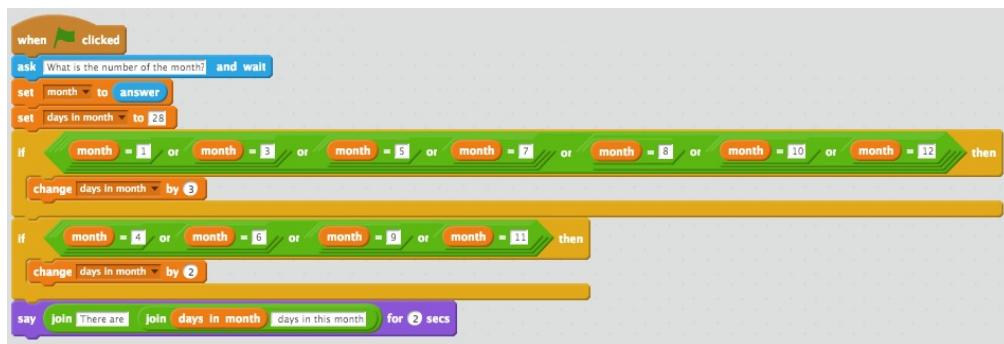
How Many Days...?

Instructions

How many days are there in a given month? We have created a [Scratch script](#) that calculates the number of days in the month corresponding to the number that a user types in. Discuss with a partner how this script works. There are a lot of new things in this program that you may not have seen before, so be sure to explore.

Note: This code is NOT written with the best style or efficiency. We hope your reaction to this is "YUCK!"

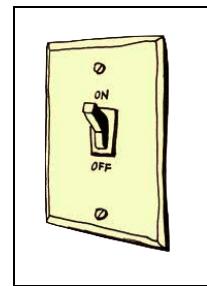
We have made this code explicitly complex to challenge your understanding of basic code and blocks and their combinations. Feel free to use pen and paper to take notes or work out the code algorithmically.



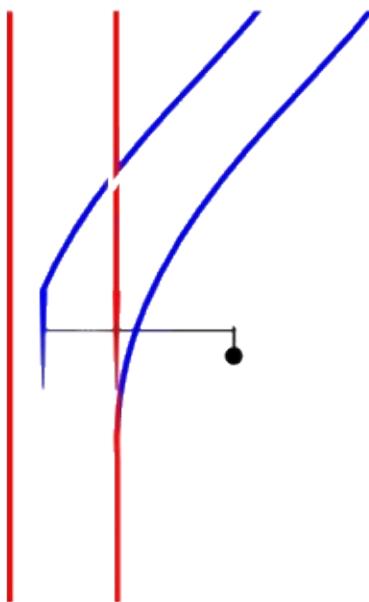
Switching and Nesting

Switching and Nesting... or ELSE!

Our conditional blocks up to now have been analogous to light switches — when turned "on," something occurs, but when turned "off," that something does *not* occur. This makes sense given the nature of our conditions — they are binary — either on or off.



However, this is not the only kind of binary switch we can model in a program. Consider a [railroad switch](#) as a type of binary, either/or switch.



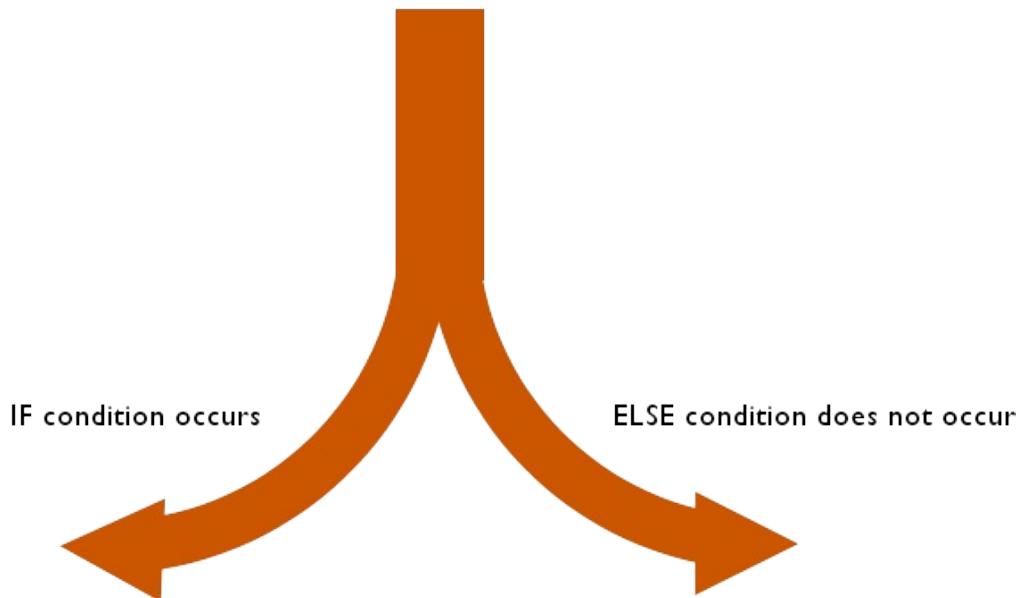
With this type of switch, rather than choosing to do something or not, you are selecting between two alternate outcomes. *One (and only one) of the outcomes will occur!* In the railroad switch example, the train, upon approaching the junction, will take either the left or the right track, depending on how the switch is oriented.

Scratch allows for this type of conditional with the **if / else** block: Check the condition. Is it *true* or *false*?

If it is *true*, then do what is enclosed in the **if** portion of the block. This is exactly the same

as the standard **if** block.

If it is *false*, then do what is enclosed in the **else** portion of the block. This differs from the standard if block. Instead of doing nothing rather than what is enclosed in the **if** block, an alternate set of blocks is executed.



Russian Nesting Dolls

"If you can't fly then run, if you can't run then walk, if you can't walk then crawl, but whatever you do you have to keep moving forward." — Martin Luther King, Jr.



The Russian nesting dolls are an iconic example of Russian culture. However, they also

illustrate an important construct in computer programming—*nested blocks*.

Motivation

Our newly learned `if / else` block is powerful; it allows a program to follow one of two paths upon reaching a condition. In other words, it allows the program to make an either/or decision.

Example:

Are you thirsty?

Options: [YES] [NO]

However, this is not how humans decide most things. Generally, there are more than two possible outcomes to a decision.

Example:

What would you like to drink?

Options: [Water] [Soda] [Coffee] [Juice]

It would be nice if our programs could choose from among multiple different paths rather than just two.

A Precursor to the Next Unit

We will examine this concept in much greater detail in Unit 3: Data Representation, but for now we will rely on our intuitions. Although each bit (in this case, each condition) represents a dichotomy between two possible choices, they can be chained together to represent more. Let's look at how we might "chain" `if / else` blocks together to represent the questions/answers above.

The first question: "Are you thirsty?"

Notice that if the `answer` is `yes`, the program asks "What would you like to drink?" If the `answer` is `no`, the program simply continues on. The second question is dependent on the the `answer` of the first.



The second question: "What would you like to drink?"

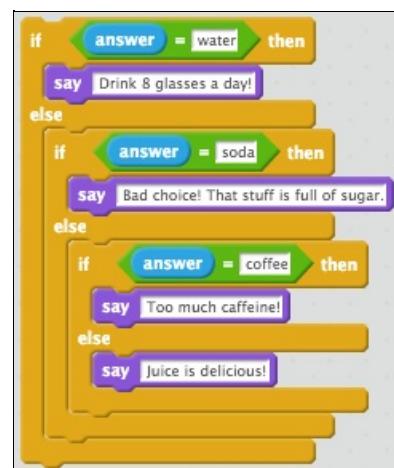
There are four possible answers, so the program handles each to produce four different outputs. However, notice that the `if / else` blocks are nested, much like nesting dolls, where one `if / else` block is *contained entirely within another*. This is different than saying, "If it's water, do *this*. If it's soda, do *that*. If it's coffee, do *x*, and if it's juice, do *y*."

Like the nesting dolls, the blocks within an `if` or `else` portion of a block are not even accessed/executed unless the program "uncovers" them by taking the appropriate branch.

So, in our example program, the `answer` is not even checked against `soda` if the `answer` was already matched to `water`. At the point at which the program finds a match for `answer`, it is effectively done searching for one.

The algorithm is as follows:

- Check to see if the answer is `water`.
 - If it is, output "Drink 8 glasses a day!" [DONE]
 - If it is not, check to see if the answer is `soda`.
 - If it is, output "Bad choice! That stuff is full of sugar." [DONE]
 - If it is not, check to see if the answer is `coffee`.
 - If it is, output "Too much caffeine!" [DONE]
 - If it is not, output "Juice is delicious." [DONE]



If [DONE] is ever reached, then the program is effectively done with executing the nested `if / else` blocks.

Discussion Questions:

The blocks tying both questions together:

How is `juice` matched if it is never explicitly checked for? How many times is `answer` checked in each of the following cases?

- `no`
- `yes, water`
- `yes, soda`
- `yes, coffee`
- `yes, juice`



How could the program be modified to check the answers to ensure that they are valid. In other words, if someone answers `Sunny D` or `milk` to "What would you like to drink?", the response would be "Sorry, we don't have that."

UNIT TOPIC:

Selection Statements

Quiz Show

- You will examine the use of the Boolean operators "AND," "OR," and "NOT" in constructing complex conditional statements.

Quiz Show

Instructions

In this assignment, you will create a Scratch program that simulates a quiz-style game show. Your program should execute *at least* the following actions:

1. The host sprite should "walk" onto the stage, and introduce him/herself.
2. The host then asks a series of three questions (of your choosing). Each question should have at least one correct answer.
3. A user's score should be kept. **HINT:** Create a variable `score` and increase it by one any time a correct `answer` is given.
4. The host should give the score and conclude the "show" when the three questions have been asked/answered.
5. Make your game unique, usable, and enjoyable.
6. Personalize it in at least three other ways.
7. Provide documentation for your program (describe what it does) as the Instructions. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself. Your work will be reviewed by a peer, and in turn, you will review one of your peer's projects. You should base your evaluation on the assignment rubric.

Rubric

Criteria	Points
Host enters stage (movement) and introduces him/herself.	1 pt
Three questions asked.	3 pts
Score is correctly kept for correct answers.	3 pts
Answers are indicated as RIGHT or WRONG by host.	1 pt
Host conclude show with score.	1 pt
Documentation, usability, and personalization	1 pt
TOTAL	10 pts

UNIT PROJECT:

Scratch Program

Highlights

- You will collaborate in pairs to design, implement, and debug a novel, aesthetically pleasing, and intuitive program using the Scratch programming environment.
- You will identify a specific purpose that your program will serve (e.g., entertainment, problem solving, education, artistic expression, etc.).
- You will integrate interactive and multimedia elements into your program.
- You will integrate common programming constructs, such as variables and selection statements into your program.
- You will test, debug, and correct your program.
- You will use appropriate terminology while writing documentation detailing the full use of your program and its features.
- You will explain your design and implementation choices while demonstrating and sharing your finished programs with your peers.
- You will provide a written analysis of at least one other design team's program, identifying its strengths and weaknesses and offering suggestions for improvement.

Scratch Project: Documentation



For a program to be useful, it needs to have three qualities:

- Correctness — In other words, the program should do what it's supposed to do. Smaller components of a program must themselves be correct to form an overall correct program. Imagine a calculator program that produced output indicating that $2 + 2 = 3$. The usefulness of this calculator would certainly be suspect. We may consider looking at how the **addition** procedure is executing. The style of the program can also affect the determination of program correctness.
- Efficiency — Now imagine that the calculator were fixed so that it always produced correct output, but took a very long time to do it. Entering $2 + 2 =$ now returns the proper answer 4 , but only after spending 12 minutes processing its input. The usefulness of this calculator would certainly be suspect.
- Clarity — Imagine that the calculator has been altered to be both correct and efficient. What other quality could possibly make it unusable? Imagine a [calculator](#) in which to add a 2 and 2 , some other key sequence than $2 + 2$ must be entered. By not knowing the proper key sequence, or more generally how the calculator functions, you would certainly have doubts about the usefulness of this calculator. **Note:** *The calculator linked above is actually very useful, once you've mastered the input mode. Try the examples linked at the top of the page. How does it work?*

In this assignment, you will concentrate on the third point — clarity. Write up a document detailing how your application is used. You should concentrate on the following (as applicable):

- What does your application do? What problem does it solve?
- How is it used? What are the specifications for input? What constitutes valid input? Invalid input?
- What are some common examples of its usage?
- What are any limitations of which users need to be aware?
- How might the program differ from user intuition? How is it unlike other, similar applications?

TOPIC:

Repetition

Repeat

- You will make use of loop constructs to create repetition without the need for duplicating code.

Repeat Until

- You will make use of loop constructs to create repetition without the need for duplicating code.
- You will explore the differences between counter- and conditional-based loops.

Loops and Variables

- You will integrate sequencing, selection, and iteration into a single mini-project artifact.

UNIT TOPIC:

Repetition

Repeat

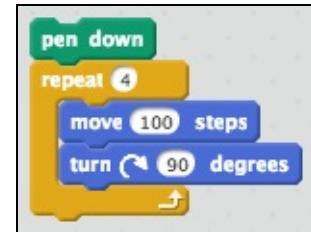
- You will make use of loop constructs to create repetition without the need for duplicating code.

Repeat

Experiment with Repeat

In [Game of Tag](#), you experimented briefly with the **forever** block. This "infinite loop" construct is often useful, but a more common scenario is repeating an action for a specific number of times (i.e., when the number is less than ∞). Scratch makes this a simple task with **Repeat** blocks. We can use a **Repeat** to make drawing shapes a lot easier! You can see below a script to draw a square.

Computer scientists often call this type of "control flow" a loop, because the flow cycles back around to a point at which the program has already executed an instruction. In the case of an infinite loop (**forever**), the cycling continues as long as the program remains executing. In the case of a **repeat** block, the loop only executes the number of times specified by the parameter (in the case of the square, four times). Use your finger to trace the control flow through the program. How does the word "loop" apply to your movements?



Using a **repeat** command, draw the following regular* shapes:

- Equilateral Triangle
- Pentagon
- Hexagon
- Octagon
- Circle
- Five-sided Star

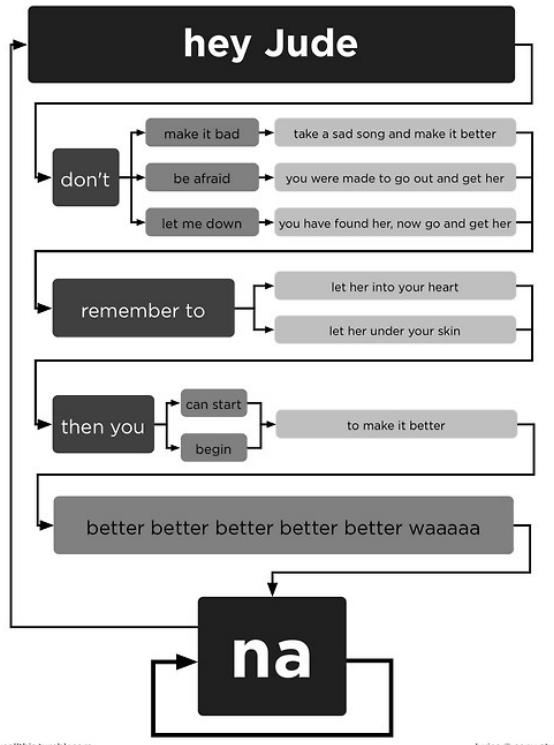
What patterns can you identify between these shapes and the commands?

*"Regular" means all the sides are the same length.

Repeat After Me

Hey Jude

Take a moment to refer to the flowchart outlining the lyrics of *Hey Jude* by The Beatles.

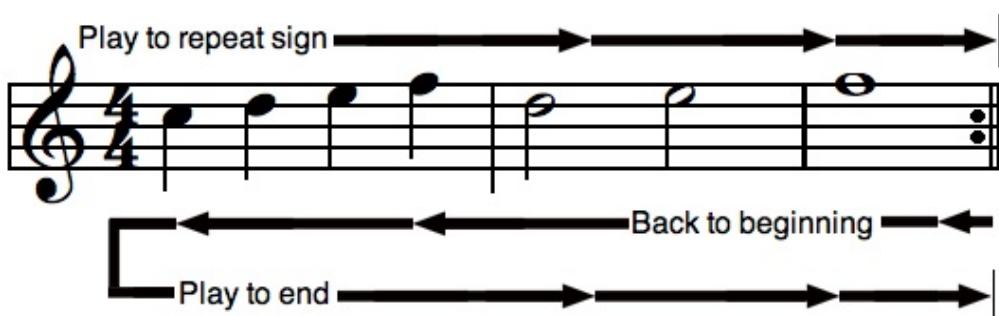
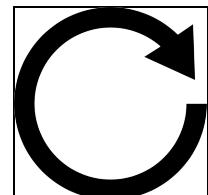


loveallthis.tumblr.com

lyrics © sony atv

A less concise, more precise chart of the lyrics might list each and every time "Na" is sung. An even more precise representation would include the timing, pitch, and tempo, like a piece of sheet music might illustrate.

However, even the precise notation that musicians rely on allows for shortcuts to indicate repetition. The colon symbol below indicates a phrase that is to be repeated once:



Instructions

Answer the following questions about representing repetition algorithmically with programming.

1) In order for a repetition to be reproduced unambiguously, what three attributes are needed to convey the proper instructions?

Answer:

2) How are these three attributes conveyed in the music notation above? How are they conveyed in the Scratch **repeat** block?

Answer:

3) Find and record a set of instructions containing a repetition clause (e.g., instructions on a shampoo bottle, a recipe, etc.). How are these attributes conveyed there? If any are missing, how would the reader infer them?

Answer:

Tempo

Changing Tempo

Changing the speed of a song can be easier than the programming strategies employed in [The Cat's Meow](#). A variable called `tempo` helps tremendously by changing how long one beat is.

Variable	Blocks	Program
<pre>tempo [60]</pre>	<pre>change tempo by [20] set tempo to [60 bpm] <input checked="" type="checkbox"/> tempo</pre>	<pre>set tempo to [15 bpm] repeat (20) play drum [48 v] for [0.2] beats change tempo by [10]</pre>

In the Sounds tab, find and select the check box next to `tempo` so that you can see the value of the variable `tempo` at any given moment on the stage. The `tempo` and `change tempo / set tempo` blocks provide three main ways to increase the `tempo` of a song.

1. Initialize the value of `tempo`.
2. Inside the `repeat`:
 - Use the value of `tempo`.
 - Change the value of `tempo`.

Try It Out!

Experiment with these codes to learn more.

1. Create a new program in Scratch.
2. Copy the script directly above.
3. Make another version where the `tempo` increases quickly.
4. Make another version where the `tempo` increases slowly.
5. Make another version where the `tempo` starts out fast and then decreases.
6. Make another version where the `tempo` starts out slow, then gets fast, then gets slow again. **NOTE:** You should make all five scripts within one Scratch program.
7. Personalize it in at least three other ways.

When you are satisfied with your work, submit a link to your program or the program itself.

Regular Polygon Generator

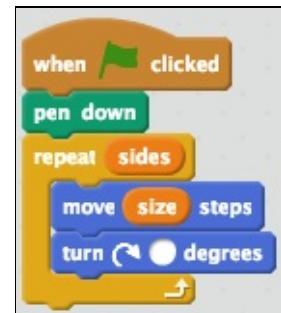
Generalization

The power of programming lies in the ability to not only automate tasks, but to generalize them. Rather than create a different shape generator for each regular polygon with varying numbers of sides, we can use the abstraction variables to program a generic regular shape generator.

Instructions

Combine what you've learned about variables and shapes to program an automated shape generator. At minimum, your program should execute the following actions:

1. Draw a shape with a number of sides based upon the value of a variable named `sides`. For example, if the variable `sides` is `3`, it will draw a triangle. If `sides` is `8`, it will draw an octagon.
2. Draw the shape based on a variable `size` that determines the length of each side.
3. Ask the user to specify each of these parameters (`size` and `sides`) when the script is executed.
4. Personalize it in at least three ways.
5. Test your program! Make sure it works correctly for an assortment of values for both `size` and `sides`.
6. Provide documentation for your program (describe what it does) as the Instructions. Be sure to describe how your program is original.



When you are satisfied with your work, submit a link to your program or the program itself.

If you are having difficulty with determining the correct "formula" for the number of degrees to turn each time, think about the following:

- You are working with two variables — what are they, and how might they be relevant?
- What numeric value would a square use to fill the blank? A triangle? Use these to think of a function to generalize them.

UNIT TOPIC:

Repetition

Repeat Until

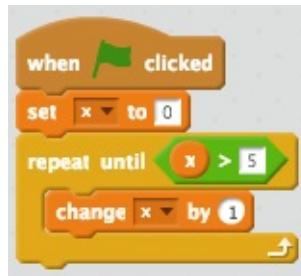
- You will make use of loop constructs to create repetition without the need for duplicating code.
- You will explore the differences between counter- and conditional-based loops.

Repeat Until

Repeat Until

There is another Scratch piece that lets us repeat things. It is called `repeat until`.

- Just like `repeat`, it will do everything inside the C-shaped block a number of times.
- Just like `if`, it is dependent upon a condition. Before it starts the loop each time, it checks to see if the condition (below, $x > 5$) is true. If this condition is true, then it will not `repeat` again.



The `repeat until` block can be really helpful to keep track of what the variable x is at each point to help us understand how this new piece works.

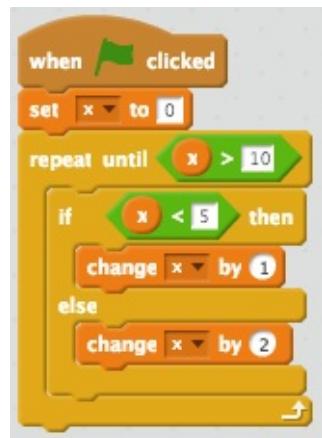
- In the right column we keep track of the value of x .
- In the diagram below we draw a horizontal line every time we start the loop.
 - Here we labeled each line "Top of loop" and "Bottom of loop," but we could just use the horizontal line to keep track of this information.
- Within each loop the variable x increases by 1, so we write down the new value for x .

	x
Before the loop	0
Top of loop	0
Bottom of loop	1
Top of loop	1
Bottom of loop	2
Top of loop	2
Bottom of loop	3
Top of loop	3
Bottom of loop	4
Top of loop	4
Bottom of loop	5
Top of loop	5
Bottom of loop	6

Try It Out!

Use a chart like the one above to keep track of what would happen in the complicated

repeat until code below. Try it without Scratch!



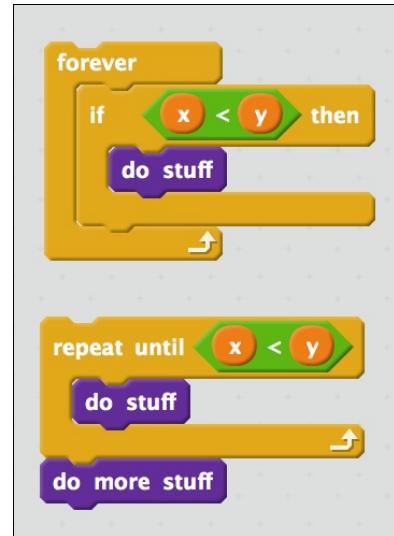
Conditional Loops Compared

Conditional Loops Compared

Scratch provides two loop constructs that are dependent on a condition for execution. They are similar, yet differ in one crucial aspect — how the condition relates to execution:

- **forever** with an immediate enclosed **if** will only execute the blocks in the loop if the condition is true. Think of it as an **if** block that keeps occurring. However, if the condition ever fails, then the body of the loop is not executed. Interestingly, the loop still continues to execute, constantly checking the condition; if it ever becomes true again, the body of the loop will execute. Because it never stops, you are unable to attach blocks beneath it. They would never execute.
- **repeat until** will default to executing the blocks in the loop repeatedly. The condition is what the loop requires in order to exit — to not repeat the loop.

Accordingly, because the loop may terminate, you are able to attach blocks beneath the loop, as they are able to be executed in some conditions.



Put another way, if you were instructed to "Hop on one foot until the clock chimes," you would continue to hop, stopping only when the condition you are looking for (a clock chime) occurs. However, if you were instructed to "Hop on one foot if the clock chimes, forever," you would wait until you heard a clock chime before you hopped. Each time the clock chimed, you would hop on one foot. Your instruction is to do that forever, so as long as you are following the instructions, you will continually stop to hop on one foot each time a clock chimes.

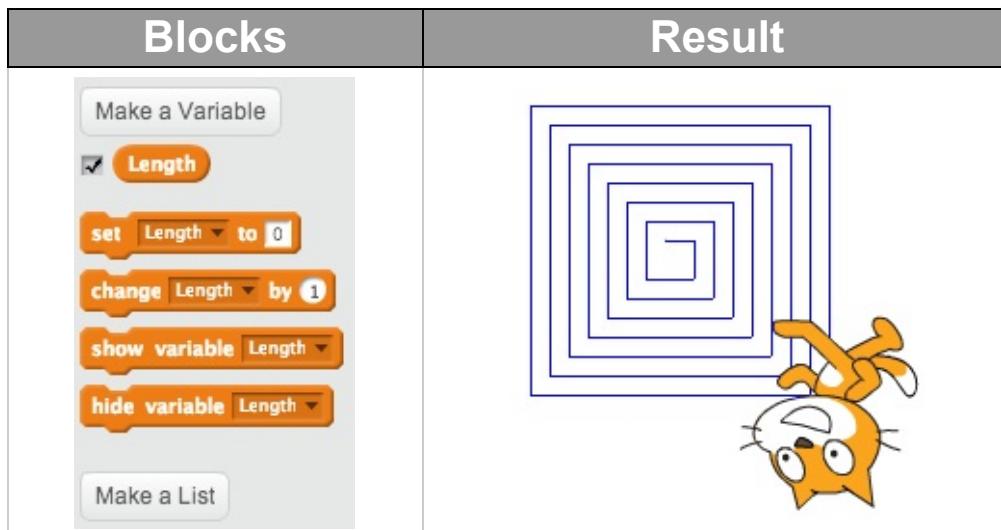
Draw a "Squiral"

Instructions

In this assignment, you will create a new Scratch program that can generate the following "squiral" (i.e., square+spiral) below. Create a new program in Scratch, and open your program from [Regular Polygon Generator](#) to reference. Your program should:

1. Include a new variable **length**.
2. Draw the "squiral" below. (HINT: Consider how many sides of each "square" in the squiral are equal **length** and when the length of each side *changes*.)
3. Personalize the program in at least three other ways.
4. Provide documentation for your program (describe what it does) as the Instructions. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself.



UNIT TOPIC:

Repetition

Loops and Variables

- You will integrate sequencing, selection, and iteration into a single mini-project artifact.

Loops and Variables Mini-Project

Instructions

To demonstrate your knowledge of loops and variables, you will choose among three different assignments. Select the one you feel is most appealing to you. Each incorporates the loops and variables, but in different contexts.

Option I	Option II	Option III
<u>Draw a Picture</u>	<u>Electronic Keyboard</u>	<u>Countdown</u>
You will program Scratch pen blocks to draw a picture. A successful solution will incorporate loops to handle repeated patterns.	You will program a virtual electronic keyboard that plays notes with computer keyboard presses. Loops can be incorporated to add functionality (e.g., for drum patterns, tempo changes).	You will program a digital countdown timer that simulates the function of a stopwatch. Loops will be utilized for timed countdown functionality. <i>NOTE: This option is the most challenging.</i>

Rubric

Criteria	Ratings	Points
Variables	Program uses both pre-defined and user-defined variables appropriately.	3 pts
	Program uses variables inappropriately or is missing user-defined variables.	2
	Program uses only pre-defined variables inappropriately.	1
Conditionals	Program uses conditional statements to simulate decisions and "branching."	3 pts
	Program uses conditionals with logical flaws, such as branches that are never taken or improper nesting. However, the program functionality works mostly as intended.	2
	Program uses conditionals improperly, such that the program does not work as intended.	1
Aesthetics	Program output is aesthetically pleasing and correct.	2 pts
	Program works as intended, but contains a few flaws (visual or aural).	1

Creativity	Program is creative.	2 pts
	Program is minimal, contains only core requirements.	1
TOTAL		10 pts

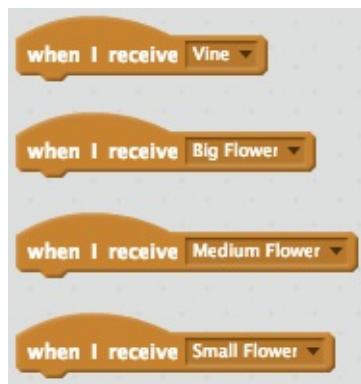
Option I: Draw a Picture

Example

Create something (*anything!*) beautiful with the drawing tools in Scratch. For instance, the following image was created in Scratch:



The image above plays around with `set pen color to [...]` and `set pen size to [...]`. If you want to create more complex images, you probably want to use `broadcast` blocks to delegate the drawing of different parts of the image. For example, we used the following `broadcast` blocks for the image above:



Tip: Your images will draw more quickly if you use the hide tab to hide the character.

Instructions

In this assignment, you will create a Scratch program that draws something beautiful. Your

original program should:

1. draw a beautiful picture,
2. use loops to draw repeated patterns,
3. use variables appropriately and as necessary,
4. be usable, efficient, and effective, and
5. include documentation (describe what it does) in the Instructions pane. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself. Your work will be reviewed by a peer, and in turn, you will review one of your peers' projects. You should base your evaluation on the assignment rubric.

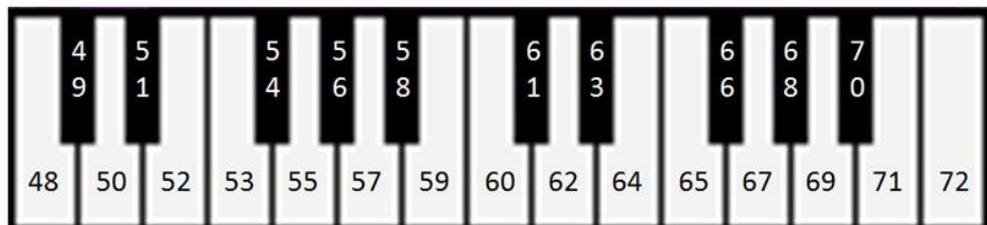
Option II: Electronic Keyboard

Features

This option asks you to program a virtual keyboard! The code to the right gives you a start, but you must add additional functionality, perhaps by adding scripts for other keys on the computer keyboard, so that your electric keyboard can change:

- Volume,
- Instrument,
- Tempo, and/or
- Drum loop/patterns.

You might want to print or draw a piano keyboard or a computer keyboard to keep track of how you have mapped keys to notes. We have provided a few images below that you can use (or print) to work with.



Instructions

In this assignment, you will create virtual keyboard using Scratch. Your original program should:

1. simulate a virtual electronic keyboard that plays notes with computer keyboard presses,
2. use loops to add functionality (e.g., for drum patterns, tempo changes),

3. use variables appropriately and as necessary,
4. be usable, efficient, and effective, and
5. include documentation (describe what it does) in the Instructions pane. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself. Your work will be reviewed by a peer, and in turn, you will review one of your peers' projects. You should base your evaluation on the assignment rubric.

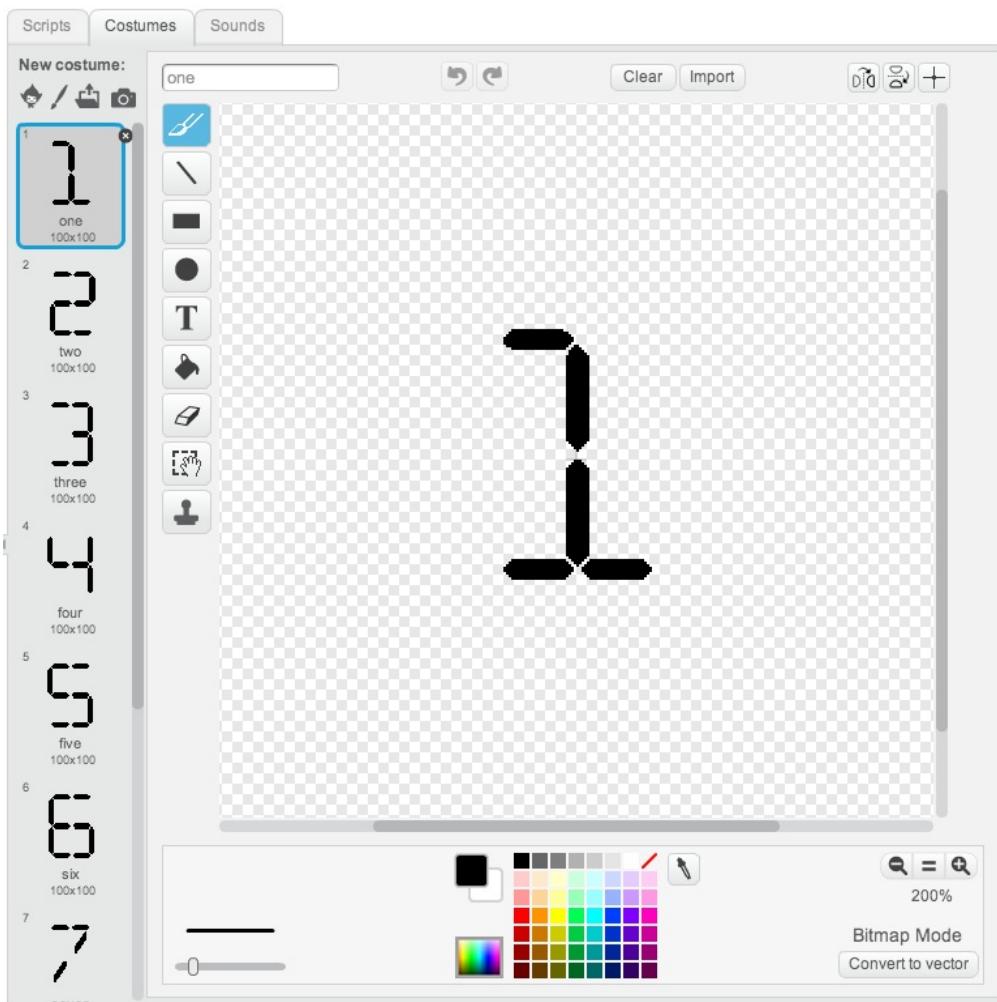
Option III: Countdown

Starter Code

Often, we want to display text or numbers on the screen without using the `say` block. We have written a program that displays one number based upon a variable named `digit`. You can copy and remix the original program [starter code](#). You should modify this program to work for five digits! (Initially, however, you may want to start by modifying it just to work with two digits.)

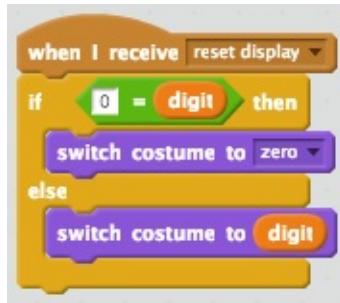
How Does the Original Code Work?

We have created a `MyDigit` sprite with ten different costumes, one for each digit, as shown below. `Costume 1` corresponds to the digit `1`, `costume 2` corresponds to the digit `2` and so on. `Costume 10` corresponds to the digit `0`. Each costume has a name (for example, `one`, as shown below) and a number (for example, `1`).



When we use the `switch to costume` block, we can specify either the name or the costume number. In the script below, we use the costume number (`1` – `9`) to set the costume unless the digit is `0`. If the digit is `0`, we cannot just tell it to switch to `costume`

0 (because there is no `costume 0`), so we have to use the name of the costume [`zero`].



Modulus

The `mod` block is essential to completing the program efficiently.

The modulus (mod) operator is used to return the remainder of a division operation. It might remind us how we answered division problems before we learned about decimals (e.g., $10 \div 3 = 3$ remainder 1). Correspondingly, $10 \text{ mod } 3 = 1$. As you complete this assignment, consider the place value of each digit you need to change. If you count down from 25, you can isolate the 5 from 25 by performing a mod operation ($25 \text{ mod } 10 = 5$). You can then isolate the 2 from 25 by performing the following operation:

$$[[25 - [25 \text{ mod } 10]] \text{ mod } 100] / 10$$

To see how this reduces to 2, note that

$$\begin{aligned} & [[25 - [25 \text{ mod } 10]] \text{ mod } 100] / 10 \\ &= [[25 - 5] \text{ mod } 100] / 10 \\ &= [20 \text{ mod } 100] / 10 \\ &= [20] / 10 \\ &= 2 \end{aligned}$$

For more explanation on the `mod` block, select the Help pane on the right side of the screen in Scratch.

Instructions

In this assignment, you will program a virtual countdown timer using Scratch. Your original program should:

1. simulate a virtual countdown timer that can count from any five-digit number down to zero,
2. remix the [starter code](#) provided,
3. use loops for timed countdown functionality,
4. use variables appropriately and as necessary,
5. be personalized in at least three ways,
6. be usable, efficient, and effective, and
7. include documentation (describe what it does) in the Instructions pane. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself. Your work will be reviewed by a peer, and in turn, you will review one of your peers' projects. You should base your evaluation on the assignment rubric.

CODING SKILLS:

Rock, Paper, Scissors

Rock, Paper, Scissors

- You will integrate sequencing, selection, and iteration into a single artifact.
- You will focus on efficiency while writing code.

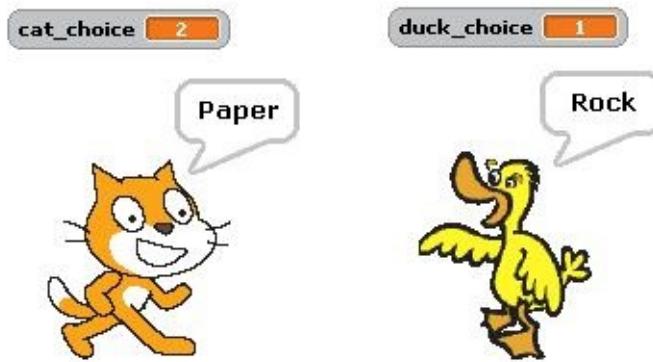
Rock Paper Scissors

Rock, Paper, Scissors

Mankind has been plagued for millennia by disagreement and indecision. Luckily, we have recently discovered plans for constructing the ultimate decision-making tool in ancient manuscripts attributed to scholars from lost Atlantis. It's... Rock, Paper, Scissors!

For this activity, you will program a *Rock, Paper, Scissors* game using Scratch.

- Make a Scratch script that makes two sprites say either "Rock," "Paper," or "Scissors" as appropriate when the green flag is pressed. You must use a `pick random block` to generate both sprites' choices.



- Add a third sprite. After the first two sprites pick their choices and say the accompanying words, the third sprite should say who wins each round (or whether the round is a tie). You may find this chart of [rock, paper, scissor outcomes](#) useful.
- Add score variables (e.g., `cat_score` and `duck_score`) that keep track of how many times each character has won (ties net zero points each). Remember, you can make these variables visible on the stage by clicking on the check-box next to the variable name.
- The third sprite should end the game when either contestant (e.g., Cat or Duck) reaches a score of three wins, and announce the winner.

Instructions

In this assignment, you will program a Rock, Paper, Scissors game using Scratch. Your original program should:

1. Simulate a Rock, Paper, Scissors game,
2. include two contestants who choose answers randomly,
3. include a referee who officiates automatically (based on this chart of rock, paper, scissor outcomes),
4. use score variables that keep track of how many times each character has won,
5. ends when one contestant wins three rounds,

6. be personalized in at least three ways,
7. be usable, efficient, and effective, and
8. provide documentation (describe what it does) in the Instructions. Be sure to describe how your program is original.

When you are satisfied with your work, submit a link to your program or the program itself. Your work will be reviewed by a peer, and in turn, you will review one of your peers' projects. You should base your evaluation on the assignment rubric.

Rubric

Criteria	Points
Uses random block appropriately to generate plays.	1 pt
Displays correct/consistent plays for randomly generated numbers.	3 pts
Resolution of all cases is correct (e.g., Rock beats Scissors).	3 pts
Scores for each sprite are correctly calculated and displayed.	2 pts
Game ends as described.	1 pt
TOTAL	10 pts

UNIT PROJECT:

Scratch Program

Highlights

- You will collaborate in pairs to design, implement, and debug a novel, aesthetically pleasing, and intuitive program using the Scratch programming environment.
- You will identify a specific purpose that your program will serve (e.g., entertainment, problem solving, education, artistic expression, etc.).
- You will integrate interactive and multimedia elements into your program.
- You will integrate common programming constructs, such as variables and selection statements into your program.
- You will test, debug, and correct your program.
- You will use appropriate terminology while writing documentation detailing the full use of your program and its features.
- You will explain your design and implementation choices while demonstrating and sharing your finished programs with your peers.
- You will provide a written analysis of at least one other design team's program, identifying its strengths and weaknesses and offering suggestions for improvement.

Scratch Project: Rubric Check



Instructions

This activity provides you and a partner group time to provide one another *critical feedback* about the progress of your projects.

1. Pair up with one another group.
2. Rate the components of your partner group's Scratch program and the documentation of their approach according to the [Programming Project rubric](#). You may write this out or print the rubric and circle components on it.
3. Review your partner group's work and provide them with documentation that describes what you *Like*, what you *Wonder*, and what their *Next Steps* should be.
4. When both you and your partner group have completed the previous steps, share your feedback with one another.

Listen to what your partner group says! You do not need to heed all of their advice, but consider it wisely. The whole idea is to have a critical third party provide ideas to make your project better before you submit the final version for a grade.

UNIT 3

Data Representation

In order to make the most effective use of computational tools and data-driven applications, students need to have a clear awareness and sense of comfort with the diverse kinds of information that may be available for use by these programs and the various ways that information may be digitally represented, stored, and manipulated within the computer. This unit focuses on providing students with an overview of the various levels of abstraction that are used in the digital representation of discrete data and information.

Students will initially focus on the lowest levels of digital representation and storage by examining different base representations of numbers (including decimal and binary) and their application to ASCII and Unicode character encoding. Students will also explore the distinctions between analog and digital forms of representation. Finally, students will examine the characteristics of lists and the types of common use-cases for these linear, ordered collections, including traversing, searching, and sorting.

UNIT PROJECT:

Unintend'o Game Controller

Highlights

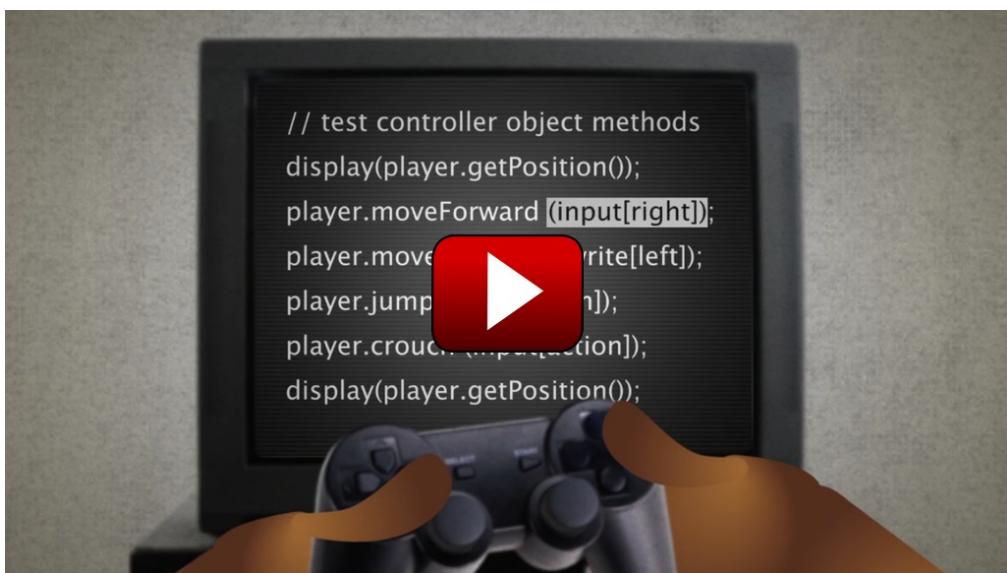
- You will develop a Scratch program that acts as a device driver for a video game controller interface.
- You will map each of six controls (UP, DOWN, LEFT, RIGHT, A, and B) to individual bits.
- You will map each binary pattern of button presses to different game actions (e.g., walk forward, walk backward, turn left, turn right, jump, duck, whirl, leap, crawl, etc.).
- You will use a list to track the history of button presses.
- You will write detailed specifications and justifications for each button-to-action mapping of your design.
- You will collaborate with your peers throughout the design and development process to determine end-user requests for features and to share feedback on design and implementation strategies.
- You will write documentation detailing the use of your program and its features using appropriate terminology.

Unintend'o Controller Project

"There are 10 types of people in this world: those who understand binary and those who don't." – Unknown

Introduction

Now that you've learned some basics about Programming using Scratch, it's time to delve deeper into the layers of computational abstraction — to understand how computers represent information. From video game platforms, to smartphones and supercomputers, it's all bits and binary underneath. Learning how to make bits and binary work for you can be very useful.



<https://www.youtube.com/embed/iMcZ7oZDjRU>

Unintend'o®

Controller Project

It's 1987 and a low budget startup video game company, Unintend'o (*pronounced: UN — INTEND—OH*), is trying to get into the gaming business by making knock-off games of popular titles. Their first game, entitled *Peerless Fabio Twins*, involves two pipefitting, Italian twin bothers, named Fabio and Lucio, who face a variety of dangers in order to rescue the Countess Lichen. To do so, they must defeat all sorts of animated lichens and molds by jumping on top of them.

You've been hired to create a new controller for the video game company Unintend'o. You

will need to decide the role of each button on the controller so that you can move the character through a 2D digital space.

Assignment

Program a video game controller using bits.

Given a game controller with four directional arrow buttons (\leftarrow , \rightarrow , \uparrow , and \downarrow) and two additional buttons, labeled **B** and **A**:



- Develop a controller interface for basic game moves (e.g., walk forward, backward, jump, duck) and advanced combo moves (e.g., whirl, leap, crawl), which requires that you:
 - Map the button's states (pressed/not pressed) to a binary sequence that travels down a wire.
 - Map each binary sequence to one of the aforementioned (and other) actions.
- Write detailed technical specifications that describe and evaluate your system.

Submission

Your submission will include both a Scratch program and the technical specifications of your device driver. You should use [this starter code](#) to begin the project.



The Scratch program should:

- map keys pressed to a binary sequence. Note that interaction between the keys (\leftarrow , \rightarrow , \uparrow , \downarrow , **A**, and **B**) and the program is restricted to only this. In other words, the key presses should only be used to build the binary sequence.
- cause Fabio to act according to the binary sequence you have created. Note that Fabio does not directly reference controller or keyboard events, but rather acts according to the value of the binary sequence.
- use loops for repetitive tasks.
- use conditionals for branching/decisions.
- work as specified in technical specifications.

The technical specifications must include:

- A basic overview of the controller design with an explanation of each button's role in the game — in effect, a *user's manual*.
- An evaluation of the driver you developed, including:

- a table outlining all possible button combinations and what action each causes (if any). Assuming you assign each button to a bit, there are 2^6 or 64 possible combinations!
- a description of which button combinations perform a special action (i.e., a combo move). **Your project should include at least two combo moves that perform a special action when two button presses occur simultaneously. You should include two separate combo moves, not just one combo move duplicated for multiple directions.** *For example, you might cause Fabio to run double speed if (and only if) the ← or → buttons are pressed while also holding down the B button would count toward one of your combo moves.*
- a description of any limitations your controller may have that lie outside of your table of button-press combinations. *For example, perhaps if both the ← and → buttons are pressed at the same time, Fabio oscillates between the two directions rather than favoring one over the other or standing completely still.*
- Key terminology from the glossary used where appropriate.

Learning Goals

Over the course of this module and this project, you will learn to:

- Describe the dichotomous nature of binary code.
- Describe how bits encode information.
- Explain how state space is exponentially proportional to the number of bits encoding it.
- Construct a binary representation of a decimal numeral
- Explain how abstraction simplifies the use of ASCII text.
- Explain how digital representation can universally approximate physical media.
- Evaluate the social impact of the universality of digital information.
- Explain how a digital copy is or is not perfect.
- Use lists as ordered data structures that may contain multiple values.

Content Area	Performance Quality			
Bitstrings	10 Fabio performs tasks based on the bitstring and not directly from the key presses or controller buttons. —AND— The bitstring is created correctly.	7 Fabio performs tasks based on the bitstring and not directly from the key presses or controller buttons but the bitstring may be created incorrectly.	4 The bitstring may be created correctly, but Fabio responds to the key presses or controller buttons and not the bitstring.	0 Not enough criteria are met in order to award any credit.
	10	7	4	0

Loops and Conditionals	<p>Both loops and conditionals have been added to the program .</p> <p>—AND—</p> <p>All loops and conditionals are used effectively and correctly with purpose in the program.</p>	<p>Loops or conditionals (but not both) have been added to the program AND all loops or conditionals are used effectively and correctly with purpose in the program.</p> <p>—OR—</p> <p>Both loops and conditionals have been added to the program but not all loops or conditionals are used effectively and correctly with purpose in the program.</p>	<p>Loops or conditionals (but not both) have been added to the program.</p> <p>—AND—</p> <p>Not all loops or conditionals are used effectively and correctly with purpose in the program.</p>	Not enough criteria are met in order to award any credit.
	<p>5 Program uses a list to accurately keep track of all button presses through their associated bitstrings throughout the program.</p>	<p>3 Program uses a list to keep track of all button presses through their associated bitstrings throughout the program but may not be accurately recorded.</p> <p>—OR—</p> <p>Program uses a list to accurately keep track of some button presses through their associated bitstrings.</p>	<p>1 Program uses a list to keep track of button presses through their associated bitstrings at some point in the program, but they may not be accurately.</p>	<p>0 Not enough criteria are met in order to award any credit.</p>
History List				
Basic Moves	<p>5 Fabio performs 6 different actions when the user presses left, right, up, down, A and B.</p> <p>—AND—</p> <p>The movements appear fluid and be aesthetically pleasing.</p>	<p>3 Fabio performs 6 different actions when the user presses left, right, up, down, A and B, but some movements do not appear fluid or may not be aesthetically pleasing.</p> <p>—OR—</p>	<p>1 Fabio performs some kind of action when the user presses keys.</p>	<p>0 Not enough criteria are met in order to award any credit.</p>

		Fabio performs less than 6 different actions when the user presses left, right, up, down, A and B, AND those movements appear fluid and are aesthetically pleasing.		
Combo Moves	5 Fabio performs at least 2 different actions when the user presses a combination of buttons simultaneously. —AND— The movement appear fluid and be aesthetically pleasing.	3 Fabio performs at least 2 different actions when the user presses a combination of buttons simultaneously, but some movements do not appear fluid or may not be aesthetically pleasing. —OR— Fabio performs one action when the user presses a combination of buttons simultaneously AND the movements appear fluid and are aesthetically pleasing.	1 Fabio performs one action when the user presses a combination of buttons simultaneously but the movements do not appear fluid or may not be aesthetically pleasing.	0 Not enough criteria are met in order to award any credit.
Driver Evaluation	10 There is a completed table outlining all possible combinations of buttons presses that perform a specific action. —AND— It includes the bitstring that those presses map to along with an explanation of the move that it performs.	7 There is a completed table outlining most possible combinations of buttons presses that perform a specific action AND It includes the bitstring that those presses map to along with an explanation of the move that it performs AND Any limitations of the driver are explained	4 There is a table of some possible combinations of button presses that perform specific actions. —AND— It includes some wording about the bitstrings, moves and limitations of the driver.	0 Not enough criteria are met in order to award any credit.

	<p>—AND—</p> <p>Any limitations of the driver are explained and justified.</p>	<p>and justified.</p> <p>—OR—</p> <p>There is a completed table outlining all possible combinations of buttons presses that perform a specific action AND It includes some wording about the bitstrings, moves and limitations.</p>		
Documentation	<p>5 There is descriptive documentation that explains the program's purpose and major code segments.</p> <p>—AND— There are complete and clear instructions on how to use the program.</p>	<p>3 There is documentation that explains the program's purpose and major code segments AND there are instructions on how to use the program.</p> <p>—OR— There is descriptive documentation that explains the program's purpose and only some code segments AND there are instructions on how to use the program.</p>	<p>1 There is documentation that explains the purpose of the program and some code segments.</p> <p>—AND— There are some instructions on how to use the program.</p>	<p>0 Not enough criteria are met in order to award any credit.</p>
				TOTAL 50 pts

UNIT TOPIC:**Binary Encoding of Information****Binary**

- You will examine how numerical values are represented using different bases, including decimal and binary.

Base Conversions

- You will explore methods of converting values from decimal to binary and binary to decimal.
- You will explore methods of counting in binary.
- You will examine the exponential relationship between the number of digits and their range of representable values.

ASCII vs. Unicode

- You will examine how alphanumeric characters and symbols may be represented using ASCII and Unicode character mappings.
- You will analyze the differences in state space between ASCII and Unicode standards.

UNIT TOPIC:**Binary Encoding of Information****Binary**

- You will examine how numerical values are represented using different bases, including decimal and binary.

What Is Binary?

What Is Binary?

"20 Questions" demonstrates the power of *dichotomous relationships*, in which something can only be one thing or another (Yes/No). Other examples of dichotomous relationships might include: a light switch, which can either be flipped on/off (not including dimmer switches), or handedness (left/right, not including ambidextrous folks), or at the most basic level, existence (something either exists or it does not).

Binary code is another example of a dichotomous relationship. Binary code is represented with the two symbols **1** and **0**. In binary code, if something isn't **1** then it must be **0** and vice versa. An example of what binary code looks like is **10110**. Unlike the alphabet, which uses the characters **A** — **Z** or decimal numbers, which use the digits **0** — **9**, binary uses *only* **1**s and **0**s to represent something. These **1**s and **0**s are referred to as bits (short for **binary digits**), and they are foundational to digital computing.

The purpose of bits is to represent something *digitally*. They are how information is stored, accessed, transformed, and used by computers. Everything that we see on a computer is actually stored as bits. The letters on this screen, the images, the links, everything you see on this webpage is stored digitally as electrical switches turned off or on (typically represented as long strings of **1**s and **0**s) that computers can interpret and transform into symbols we understand, like numbers, letters, images, sounds, and programs.

This is, of course, an oversimplification of binary code and bits. Typically, in modern computers, the **1**s and **0**s we refer to are the presence or absence of electrical signals, *but they don't have to be!* One of the many beauties of computer science is that abstraction allows us to view many processes and systems computationally — even those not involving a "computer."

Learn more about bits and binary code from *Blown to Bits*:

- [*Blown to Bits, Chapter 1*](#)

Twenty Questions

Binary Code in 20Q

Bits can encode everything. How many yes/no questions would it take to identify anything you can think of (e.g., animal, vegetable, music, artist, etc.)? Today you'll play and discuss the game "20 Questions" in order to think about how binary code can help represent just about anything.

"20 Questions" has been a popular game in the United States for over a century. Have you played it with your friends or family before? If you haven't, here's how the game works:

In the traditional game, one player is chosen to be the answerer. That person chooses a subject (object) but does not reveal this to the others. All other players are questioners. They each take turns asking a question that can be answered with a simple "Yes" or "No." In variants of the game (see below), multiple state answers may be included, such as the answer "Maybe." The answerer answers each question in turn. Sample questions could be: "Is it bigger than a breadbox?" or "Can I put it in my pocket?" Lying is not allowed in the game. If a questioner guesses the correct answer, that questioner wins and becomes the answerer for the next round. If 20 questions are asked without a correct guess, then the answerer has stumped the questioners and gets to be the answerer for another round.



Careful selection of questions can greatly improve the odds of the questioner winning the game. For example, a question such as "Is it a machine?" can allow the questioner to cover a broad range of areas using a single question that can be answered with a simple "yes" or "no." If the answerer responds with "yes," the questioner can use the next question to narrow down the answer; if the answerer responds with "no," the questioner has successfully eliminated a number of possibilities for the answer. ([Wikipedia](#))

How has computing affected 20 questions? Well, now we can play against computers using artificial intelligence.

1. Navigate to [20Q.net](#).
2. Play 20 Questions against the computer by choosing the language you'd like to play in. Follow the online instructions and see how accurately the computer can guess your chosen subject.
3. After a few games, click "About Us" in the left side bar of the [20Q.net](#) website. Read

the page to learn more about how the program works.

4. Next, play a game of 20 Questions with a neighbor. Choose the questions you ask in the game strategically.
5. When you are finished, you will be expected to discuss the following questions:
 - What were some “good” questions? (“good” means “efficient and effective”) Why were those questions good?
 - What were some “bad” questions? (“bad” means “inefficient or ineffective”) Why were those questions bad?
 - How does the choice of questions affect their utility?
 - Who is more effective—20Q.net or your neighbor?

State Space

Definition

Technically, the phrase **state space** refers to the *space of potential possibilities*. Watch the following video of a man making handmade noodles in Beijing, China. In this case, **state space** refers to the *potential number of noodles he makes every time he folds the existing noodles in half*.



<https://www.youtube.com/embed/6rfu1ZHiMP8>

State Space and Decimal Numbers

If you have three decimal digits (), you can encode 1,000 different numbers (000–999). The *state space* refers to all of those potential values. In other words, a three-digit base10 number has 1000 possible states.

The state space of number systems grows exponentially with the number of possible digits.

- If you have 1 decimal digit (), you can encode 10 different numbers (0–9).
- If you have 2 decimal digits (), you can encode 100 different numbers (00–99).
- If you have 3 decimal digits (), you can encode 1,000 different numbers (000–999).
- ...

Notice a pattern? Every time you add a decimal digit, the state space multiplies by 10 (e.g., $10 \times 10 = 100$, $100 \times 10 = 1,000$).

All Your Base Are Belong to Us

As seen above, number systems represent *exponential growth*. Each digit is a coefficient of an exponential term. The base of each exponential term is the base of the number system.

Decimal:

324_{10}

$$= (\boxed{3} \times 10^2) + (\boxed{2} \times 10^1) + (\boxed{4} \times 10^0)$$

$$= \boxed{300} + \boxed{20} + \boxed{4}$$

$$= \boxed{324}_{10}$$

Binary:

100111_2

$$= (\boxed{1} \times 2^5) + (\boxed{0} \times 2^4) + (\boxed{0} \times 2^3) + (\boxed{1} \times 2^2) + (\boxed{1} \times 2^1) + (\boxed{1} \times 2^0)$$

$$= \boxed{32} + \boxed{4} + \boxed{2} + \boxed{1}$$

$$= \boxed{39}_{10}$$

Mathematically, any number can be described by the following notation, where i is the position of each digit in the number (i.e., $i=0$ represents the "ones" place, $i=1$ represents the "tens" place, $i=2$ represents the "hundreds" place, etc.):

$$\sum_{i=0}^n (digit_i \cdot base^i)$$

The base represents how much the state space grows multiplicatively each time a digit is added.

State Space and Binary Numbers

Unlike the base10 system, each place value in binary (or base2) has only two possibilities (0 or 1). The addition of each digit represents a doubling of the possible values represented:

- If you have 1 binary digit ($\boxed{}$), you can encode two different numbers (0–1).
- If you have 2 binary digits ($\boxed{}\boxed{}$), you can encode four different numbers (0–3).
- If you have 3 binary digits ($\boxed{}\boxed{}\boxed{}$), you can encode eight different numbers (0–7).
- ...

Every time you add a binary digit, the state space multiplies by 2 (e.g., $2 \times 2 = 4$, $4 \times 2 = 8$). The pattern here is the same, though the *base* differs.

State Space Is Not Just About Numbers

Consider the [20 Questions](#) activity that you previously explored. Each time a "yes" or "no" (binary) question is resolved, the number of potential items that your question sequence can distinguish effectively doubles. With five questions, it's possible to distinguish among 32 objects. With 10, the number grows to 1,024. With 20 questions, the number reaches 2^{20} or

1,048,576 objects!

Imagine a very constrained space, in which I ask you to identify which direction you are facing in 1D space (i.e., "left" or "right"). How many yes/no questions must you ask to figure out the direction? The obvious answer is "2," but that's actually more than we need. Instead of asking "*Is it left?*" and "*Is it right?*", we could just ask "*Is it left?*" or "*Is it right?*" The state space of one binary question contains two possibilities ($2^1 = 2$), so that's all we need.

For homework, consider a scenario in which you must identify which cardinal (**N**, **S**, **E**, **W**) or intercardinal (**NE**, **NW**, **SE**, **SW**) direction an object is facing. How many "yes" or "no" questions would it take to identify the direction? Naïvely, it would take eight questions (e.g., *Is it "North"?*, *Is it "Northeast"?*, *Is it "East"?*, etc.), but because we are using binary states, we know that it can be done in three questions.

Write Up the Following:

1. Give **three dichotomous** questions that will correctly identify an object's cardinal or intercardinal orientation when answered. **Note:** There is not just one single correct answer here!
2. Create a table that maps each answer sequence to its matching direction. This table should have three columns (for each of the questions) and eight rows (for each of the directions). **Note:** Each sequence should be uniquely different than the others. Think about why this must be true.

Example Table:

	Question 1	Question 2	Question 3
N	Yes	No	Yes
NE			
E			
SE			
S			
SW			
W			
NW			

High/Low Guessing

Game

In the High/Low game, you will choose a number between 1–1000. The computer will make guesses such as “*I guess that your number is 3*” in order to determine your number. You will then respond that incorrect guesses are “*too high*” or “*too low*”. In the end, the computer always ends up guessing the correct number, but how many guesses are ideal for this task? How many guesses **guarantee** a correct guess? The computer claims it can guess correctly using *at most ten* guesses each time. Think about how this relates to the state space in this scenario—the numbers 1–1000.

Revisit the [Binary Search tool](#) from Unit 1. Experiment with the applet to determine if this is true. Write your answer in 2–3 sentences including an explanation of your reasoning.

Once you have written up your response, try to apply the same reasoning yourself. Guess the computer's number in as few questions as possible by playing Funbrain's [Guess the Number](#) game.

UNIT TOPIC:**Binary Encoding of Information****Base Conversions**

- You will explore methods of converting values from decimal to binary and binary to decimal.
- You will explore methods of counting in binary.
- You will examine the exponential relationship between the number of digits and their range of representable values.

The Amazing Binari

The Amazing Binari Transmutes Decimal to Binary!



Come one, come all, and witness the magic of the Amazing Binari (rhymes with Atari)! After years of study at the Citadel, Binari has returned to the West with an amazing power. The Amazing Binari can convert any decimal number into its binary representation! Wow! Don't believe it? Test his abilities [here](#).

How does he do that? No matter what he may claim, it's not magic.

Your job is to deduce how to convert decimal numbers to their binary representations by observing the Amazing Binari at work. Give the Amazing Binari a number to transmute, and watch what he does. Notice any patterns? How would you describe his actions?

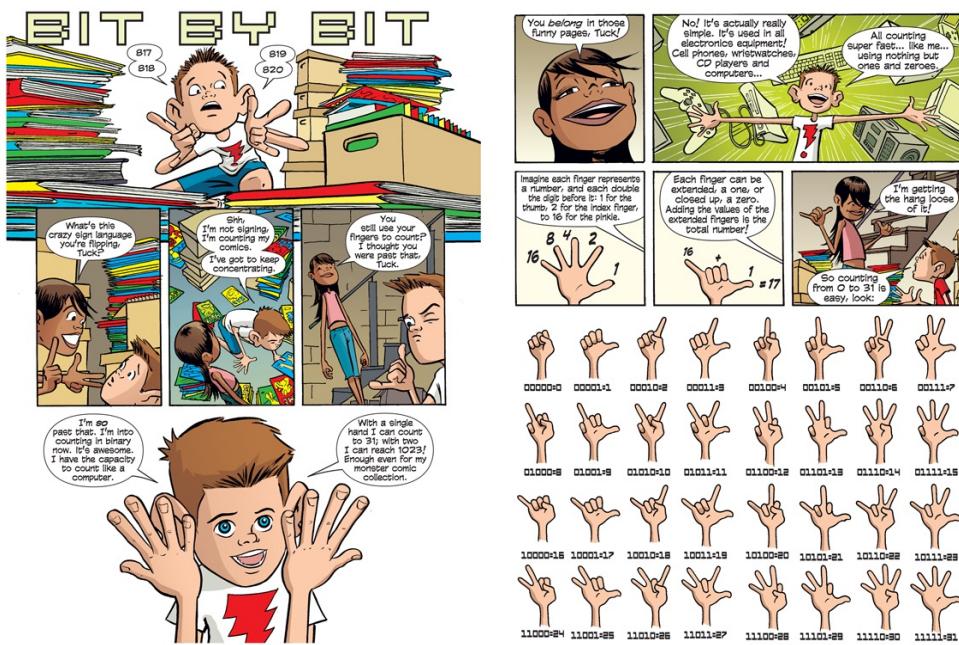
Write and submit your interpretation of his actions. In other words, describe how to convert decimal numbers to their binary representations, using the Amazing Binari as an example.

Binary Finger Counting

Counting in Binary

How useful is binary? If it can encode anything, can it encode numbers? Letters? Of course it can! Let's try counting in binary just like you first tried counting in kindergarten...on your fingers.

How high can you count on one hand? 5? Ha! You can do much better. Using the idea of dichotomous relationships and binary, [Binary Finger Counting](#) allows you to count to numbers much larger than 5 using only the five fingers you have on one hand. Consider each finger to be either "extended" or "not." Each of your fingers then represents one bit. Here's a comic that explains the entire process:



Notice that each finger doesn't represent a separate number, it represents a dichotomy, or division into two classes (e.g., "the number is <16 (if not extended), or ≥16 (if extended)"). Each new finger doubles the number of values that can be represented.

Even better, let's see binary numbers dance:



<https://www.youtube.com/embed/OCYZTg3jahU>

Now try it yourself. Use the cartoon or video to help you along, or if you prefer written instructions, here is a procedure (or *algorithm*) that explains how you can encode a number in the range 0-31 on one hand:

- The pinky represents whether the number is <16 (if not extended), or ≥ 16 (if extended).
 - So, if it is 16 or higher, extend your pinky, subtract 16 from your number, and continue with Step 2.
 - If it is lower than 16, just skip ahead to Step 2.
- The ring finger represents whether the remaining value is <8 (if not extended), or ≥ 8 (if extended).
 - So, if it is 8 or higher, extend your ring finger, subtract 8 from your number, and continue with Step 3.
 - If it is lower than 8, just skip ahead to Step 3.
- The middle finger represents whether the remaining value is <4 (if not extended), or ≥ 4 (if extended).
 - So, if it is 4 or higher, extend your middle finger, subtract 4 from your number, and continue with Step 4.
 - If it is lower than 4, just skip ahead to Step 4.
- The index finger represents whether the remaining value is <2 (if not extended), or ≥ 2 (if extended).
 - So, if it is 2 or higher, extend your index finger, subtract 4 from your number, and continue with Step 5.
 - If it is lower than 2, just skip ahead to Step 5.
- The thumb represents whether the remaining value is <1 (if not extended), or =1 (if extended).
 - So, if it is 1, extend your thumb, subtract 1 from your number, and you are done (you should have 0 as a remaining value).

- You are done (you should have 0 as a remaining value).

Notice that each finger doesn't represent a separate number, it represents a dichotomy or division into two classes (e.g., "the number is <16 (if not extended), or ≥16 (if extended)"). Each new finger doubles the number of values that can be represented.

Check your counting using this [Hand Counting Applet](#).

Challenge

How many values could this guy encode?

HINT: Think about what value the sixth finger would represent.

Go back and look at the pattern for the values represented by your own five fingers, and then apply it to the sixth finger.



UNIT TOPIC:**Binary Encoding of Information****ASCII vs. Unicode**

- You will examine how alphanumeric characters and symbols may be represented using ASCII and Unicode character mappings.
- You will analyze the differences in state space between ASCII and Unicode standards.

Alphanumeric Representation

Encoding Alphanumerics with the ASCII Table

ASCII	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 0 0 0	N	U	S	H	S	X	E	T	E	Q	A	K	B	L	B	S
0 0 0 1	D	L	D	1	D	2	D	3	D	4	N	K	S	Y	E	Σ
0 0 1 0	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	
0 0 1 1	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0 1 0 0	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0 1 0 1	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0 1 1 0	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0 1 1 1	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
1 0 0 0	Ä	À	Ç	É	Ñ	Ö	Ü	á	à	â	ã	å	ç	é	è	
1 0 0 1	ê	ë	í	ì	î	ï	ñ	ó	ò	ô	ö	ö	ú	ù	û	ü
1 0 1 0	†	°	¢	£	§	•	¶	฿	₪	®	©	™	‘	’	≠	∅
1 0 1 1	∞	±	≤	≥	¥	µ	ø	Σ	∏	π	∫	ª	º	Ω	¤	ø
1 1 0 0	ż	ı	¬	√	f	≈	△	«	»	...	À	Á	Ó	Œ	œ	
1 1 0 1	–	—	”	‘	’	÷	◊	ÿ	ÿ	›	€	⟨	⟩	fi	fl	
1 1 1 0	‡	·	,	„	‰	Â	Ê	Á	Ë	È	Í	Î	Ï	Ó	Ô	
1 1 1 1	apple	Ø	Ú	Û	Ù	ı	^	~	—	~	·	°	„	“	~	

↑

This bit specifies whether the character is in the top half of the table (Standard ASCII) or the bottom half (Extended ASCII).

Standard ASCII Character set (7 bits)

Extended ASCII Character set (8 bits)

This **ASCII** (American Standard Code for Information Interchange) table outlines a common set of conventions established for converting between binary values and alphanumeric characters.

The table functions as a mapping from binary values to alphanumeric symbols. This is an arbitrary mapping that was constructed many years ago. Note that there is no inherent meaning that dictates that the double-quote character, " (or 34 in decimal notation). Much like your groups are doing for your controller projects, this table and its mappings were defined by real people.

Watch this tutorial for a demonstration of how this table is read:

ASCII	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 0 0 0	N	U	S	H	S	X	E	T	E	Q	A	K	B	L	B	S
0 0 0 1	D	L	D	1	D	2	D	3	D	4	N	K	S	Y	I	E
0 0 1 0	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	
0 0 1 1	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0 1 0 0	@	A	B	C	D	E	F	G	H	I						O
0 1 0 1	P	Q	R	S	T	U	V	W	X	Y						-
0 1 1 0	'	a	b	c	d	e	f	g	h	i						o
0 1 1 1	p	q	r	s	t	u	v	w	x	y	z	,	,	,	,	,
1 0 0 0	Ä	À	Ç	É	Ñ	Ö	Ü	á	à	ã	ä	å	ç	é	è	
1 0 0 1	ë	ë	í	í	í	í	í	ñ	ñ	ñ	ö	ö	ö	ö	ú	ú
1 0 1 0	†	°	c	£	§	*	¶	¤	¤	¤	¤	¤	¤	¤	¤	¤
1 0 1 1	∞	±	≤	≥	¥	µ	∂	Σ	Π	π	∫	¤	°	Ω	æ	ø
1 1 0 0	ż	ż	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
1 1 0 1	-	-	"	"	'	'	÷	◊	◊	◊	/	€	<	>	fi	fl
1 1 1 0	‡	‡	"	"	%	À	È	À	È	È	È	Í	Í	Í	Ó	Ó



01000001

Standard ASCII Character set (7 bits)



Extended ASCII Character set (8 bits)

<https://www.youtube.com/embed/9NKcgp8KH2k>

You can convert alphanumerics into binary (or vice versa) using the ASCII table as the tutorial explains, or you can use one of the many free transducers found online, like this [Binary to Text Converter](#). Try it out!

Note about "*computational thinking*": Remember, one aspect of computational thinking is *abstraction*. Computers can't understand language or symbols like we do, but they can do mathematical computations really quickly. The abstraction that maps values to symbols creates the illusion that computers work directly with language constructs.

Errors/Noise

Have you considered the fact that **g** and **w** are only different by *one bit*?

'g` is encoded as **01100111**

'w` is encoded as **01110111**

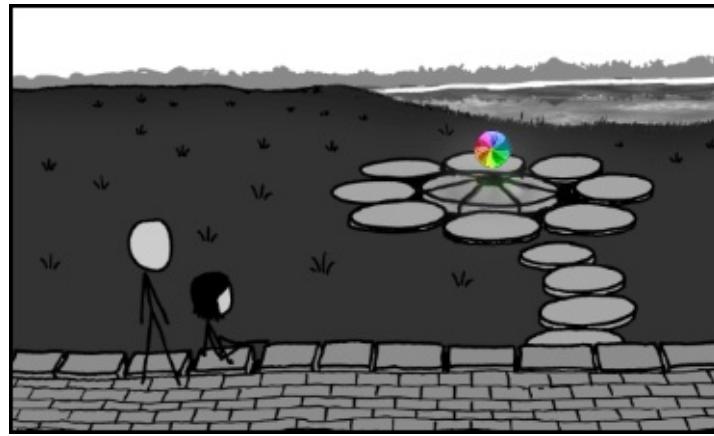
If somehow this bit were "flipped," a confusing message may result:

This cake is really wood. Try some!

Umm, no thanks? I'm not a termite.

Computer scientists call problems such as these noise. Noise is irrelevant or meaningless data that has found its way into otherwise meaningful code. Noise can make it difficult for computers to efficiently and effectively represent binary as alphanumeric or other representations (e.g., images, audio). Luckily, algorithms have been developed to detect and even correct such problems, which is why they don't typically occur.

Otherwise, as a result of noise, instead of what you expect, you might end up missing a really delicious piece of cake, or wind up with a blue screen or the spinning wheel of death.



"There's always the hope that if you sit and watch for long enough, the beachball will vanish and the thing it interrupted will return." — [The Eternal Flame](#), xkcd #961 (Randal Munroe)

Digital Scavenger Hunt

Abstraction in Action

Bits are everywhere, and they can represent virtually anything. In fact, at the binary level, the previous sentence looks like this:

```
01000010 01101001 01110100 01110011 00100000 01100001  
01110010 01100101 00100000 01100101 01110110 01100101  
01110010 01111001 01110111 01101000 01100101 01110010  
01100101 00101100 00100000 01100001 01101110 01100100  
00100000 01110100 01101000 01100101 01111001 00100000  
01100011 01100001 01101110 00100000 01110010 01100101  
01110000 01110010 01100101 01110011 01100101 01101110  
01110100 00100000 01110110 01101001 01110010 01110100  
01110101 01100001 01101100 01101100 01111001 00100000  
01100001 01101110 01111001 01110100 01101000 01101001  
01101110 01100111 00101110
```

The bits themselves do not mean anything without a rule that tells us how to interpret them. This is the beauty of abstraction and a big part of the illusion that makes computers work the way they do.

Consider the following string of bits:

```
01000100 01110101 01100100 01100101 00101100 00100000  
01110100 01101000 01101001 01110011 00100000 01101001  
01110011 00100000 01101111 01100010 01110110 01101001  
01101111 01110101 01110011 01101100 01111001 00100000  
01110100 01100101 01111000 01110100 00101110
```

The task of figuring out what these bits mean is impossible without establishing a method for decoding them. They may be a message encoded in ASCII, a save file for *Peerless Fabio Twins*, or part of an image. They could be any of these and more. Assuming that they are

ASCII, mapping each set of eight is trivial; just map them from one encoding to another.

Use the [XLATE tool](#) to read the message encoded in ASCII in the previous bitstring. Simply copy and paste the binary into the "binary" pane and press the button. XLATE will present different encodings of the same bits. It should be fairly obvious which is the intended encoding.

Your Task: A Digital Scavenger Hunt

Your task is to find and submit a secret message buried in binary. This task stresses that binary can encode different types of data depending on the rules for abstraction used to interpret it.

1. Load `secret.txt` in a web browser to obtain some bits. Before continuing, look at the bits. What do you think is encoded? How different are the bits in different areas of the bitstring?
2. Paste these bits (ALL OF THEM) into the binary pane in [XLATE](#) and click the DECODE button. Nothing obvious will be apparent, but remember how differently the bits were distributed in binary. Scroll around the panes and look for something interesting. You will find further instructions there.

Submit the Following:

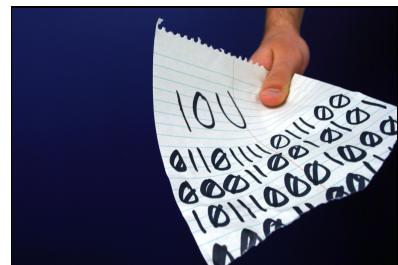
- The secret message.
- A brief outline of the process you used to find the message. Include dead ends!
- Answer the following short questions:
 1. What type of file do you open at the end?
 2. Why is the sequence `11111111` repeated often in `secret.txt`? What does it represent? *Hint: Consider the type of file that is encoded and what occurs most often within it.*
 3. Type a message (like "sir i soon saw bob was no osiris") into the text field in XLATE. Which of the encodings (binary, ASCII decimal, hexadecimal or BASE64) is the most compact? Why?

Reading and Writing in ASCII

Passing Notes

How many times have your teachers told you or your classmates, "Don't pass notes in class!" Well, now it's your turn to do just that, but there are a few caveats:

- Your note must be written in binary code (0s and 1s).
- You must use the values for alphanumeric symbols found in the ASCII table.
- You may not use any online alphanumeric/ASCII converters.



*A copy of the ASCII table is available at the end of these instructions.

Procedure

1. Match up with a partner.
2. Write down the contents of your note in alphanumeric characters (e.g., "If there's a bustle in your hedgerow, don't be alarmed now.")
3. Use the ASCII table to write the note out to your partner using only binary code.
4. When you have both completed writing your notes in binary, trade papers and translate the binary code back to alphanumeric using the ASCII table.

Think About It

- Was there any noise in your partner's note?
- How do you know when one character ends and the other begins?

ASCII	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
	0	0	0	0	N_U	S_H	S_X	E_X	E_T	E_Q	A_K	B_L	B_S	H_T	L_F	Y_T	
	0	0	0	1	D_L	D_1	D_2	D_3	D_4	N_K	S_Y	E_Z	C_N	E_M	S_B	E_C	
	0	0	1	0		!	"	#	\$	%	&	'	()	*	+	
0	0	1	1	0	0	1	2	3	4	5	6	7	8	9	:	;	
0	1	0	0	0	@	A	B	C	D	E	F	G	H	I	J	K	
0	1	0	1	0	P	Q	R	S	T	U	V	W	X	Y	Z	[\]
0	1	1	0	0	^	a	b	c	d	e	f	g	h	i	j	k	
0	1	1	1	0	p	q	r	s	t	u	v	w	x	y	z	{	}
1	0	0	0	0	Ä	À	Ç	É	Ñ	Ö	Ü	á	à	â	ä	å	
1	0	0	1	0	ê	ë	í	ì	î	ï	ñ	ó	ò	ô	ö	õ	
1	0	1	0	0	†	°	¢	£	§	•	¶	ß	®	©	™	‘	’
1	0	1	1	0	∞	±	≤	≥	¥	µ	ø	Σ	Π	π	∫	ª	º
1	1	0	0	0	ξ	i	¬	√	f	≈	△	«	»	...	À	Á	Ó
1	1	0	1	0	-	-	"	"	'	÷	◊	ÿ	ÿ	/	€	<	>
1	1	1	0	0	‡	·	,	,	‰	Â	Ê	Á	Ë	Í	Î	Ï	Ó
1	1	1	1	0	apple	ò	ú	û	û	ı	^	~	—	~	·	°	„

↑

This bit specifies whether the character is in the top half of the table
(Standard ASCII) or the bottom half (Extended ASCII).

Standard ASCII Character set (7 bits)

Extended ASCII Character set (8 bits)

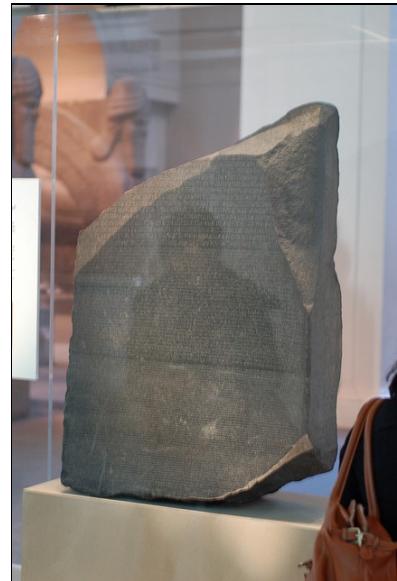
Unicode vs. ASCII

Unicode vs. ASCII

This chunk of rock is the [Rosetta Stone](#). Historically, it is important because it allowed the first deciphering of otherwise strange symbols found in ancient Egyptian ruins. It contained one piece of narrative text in three different forms — in ancient Egyptian hieroglyphics, Ancient Demotic, and Ancient Greek. This allowed historians to translate the hieroglyphics into Greek, which was already well-understood.

It is important to remember that not all languages are represented with the same 26 symbols that comprise American English.

As you may remember, ASCII stands for *American Standard Code for Information Interchange*. So it probably comes as no surprise that it was designed to meet the alphabetic needs of American languages like English. However, with this narrow bias, ASCII fails to provide for the many letters and characters that are common to the multitude of other world languages beyond English. As computers have become more and more ubiquitous, and people who speak languages other than English have begun to use computers, program, and participate in social media, the need to extend this venerable standard to accommodate them has grown.



The problem, of course, is that the ASCII table allows a very limited set of symbols — seven bits' worth. In order to make room for more symbols, more bits are needed. How many are needed?

The current solution is to use the newer **Unicode** standard. Unicode is a binary encoding system that can represent much more of the world's text than ASCII can. Unicode allows computers to represent most of the world languages' alphabets, not just English.

- The ASCII table includes 2^7 values (7 bits).
- Unicode includes 2^{16} values (16 bits).

which means:

- ASCII can represent 128 different characters.
- Unicode can represent 65,536 different characters!

Anything encoded with ASCII will work with Unicode; the first 128 symbols are the same. Extra zeroes are used to pad the beginning of the Unicode equivalent to an ASCII encoding to account for Unicode's extra state space (2^{16} vs. 2^7 possibilities).

For example, the letter "U" is represented by both as:

ASCII	Unicode
1110101	0000000001110101

Using Unicode with an ASCII Keyboard

Common misconception: *The characters available for use are restricted to those on the keyboard.*

Unicode interfaces (alternative codes, character maps, even extended keyboards) allow for a huge range of symbols. Before Unicode we ate "jalapenos." Now we can eat "jalapeños," which are much tastier.

Can somebody type in other languages using Unicode and any keyboard? Yes! Even with the keyboards that are primarily found in the United States, we can use, type, and encode in other languages using Unicode. Here's how:

- Instructions for entering Unicode using:
 - [Windows](#)
 - [OS X](#)

For a web accessible application that generates Unicode character tables, try [The Unicode Range Viewer](#).

Additionally, you may like to see some of the many examples of world language symbols in context. Try entering some common English words and phrases into a language translation site such as [Foreignword.com](#) or [Google Translate](#).



An Arabic keyboard. Click [here](#) to see (and use) more language-specific keyboards.

CODING SKILLS:

Binary Birthday Cake

Highlights

- You will construct a Scratch program that simulates candles on a birthday cake being lit so as to show the user's age in binary.

Binary Birthday Cake

Grandpa is Turning 60!

Unfortunately, the supermarket doesn't have enough candles to fill his birthday cake! He needs five boxes of 12 to light the cake the traditional way. The store only has one box — not enough to even form the numerals "6" and "0" legibly.

Luckily, you aren't constrained by traditional thinking!

Your job is to design and code a Scratch program that, given somebody's age, will light a birthday cake in **binary**.

For example, if the person were **26** years old, the binary representation of their age is **11010**.



We can see this by calculating what each binary digit represents within the overall number:

$$\begin{aligned} &= 11010_2 \\ &= (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) \\ &= (1 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (0 \times 1) \\ &= 16 + 8 + 0 + 2 + 0 \\ &= 26_{10} \end{aligned}$$

The birthday cake for a 26-year-old should, therefore, have *at least five* candles to represent the 5 bits of the binary number 11010_2 . If we say a lit candle represents a 1 bit and an unlit candle represents a 0 bit, our finished cake might look something like this:



Note that the example picture above includes two additional leading unlit candles; because these are unlit, they are effectively identical to leading zeroes, and as such, do not affect the value of the number represented.

Program Notes

- A starter project you may remix is available [here](#).
- The program screen should include at least a birthday cake and candles.
- The allowable age range for a person to enter is 0 – 122, as 122 is the oldest confirmed age of a human, namely [Jeanne Calment](#) — so the maximum number of candles required is seven.
- Lastly, the easiest way to have a candle lit/unlit is to have two different "costumes" — one corresponding to each, and select the one to display based on the binary representation of the desired age.
- (*Optional*) The number of candles should change depending on the age. So, a one-year-old would have one candle, a 30-year-old would have five, and an 83-year-old would have seven, because the next highest power of 2 in each case is 1, 5, and 7, respectively.

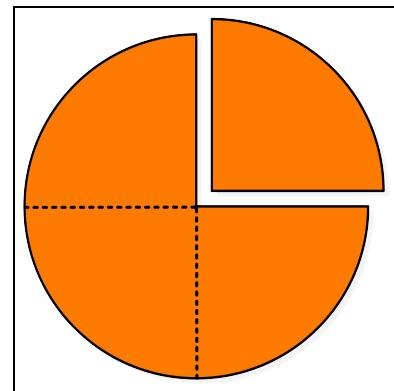
Floating Point Numbers

Not Just Whole Numbers

Of course, digital devices can represent more than just whole numbers. But how can numbers with decimal points be represented in binary? The most intuitive way might be to first treat the number as a whole number, and then always insert a decimal point at the same point in its decimal equivalent.

Consider the following 32 bit representation:

```
110111111000101110111111000101
    3754287045
      375428 . 7045
```



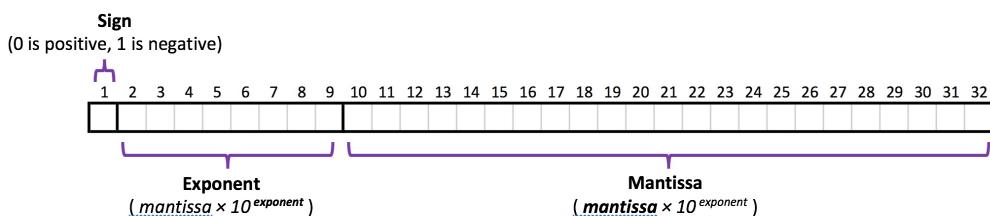
This assumes that the number is always represented to the 1/10,000^{ths} in precision, but this is arbitrary; we could have chosen any different number of decimal places. However, we would need to choose this number beforehand, and it would apply to any number we encoded with this representation. This is called **fixed-point** representation, because the decimal point is always in the same place.

Because of this, fixed-point numbers are very limited in the range of values they can represent. In our example above, we could encode 0–429496.7295. That seems like a fairly large range, but we couldn't even encode 1 million using this 32 bit, 4 decimal place, fixed-point representation!

Floating Point Numbers

To circumvent this limitation, modern computers use a **floating-point** representation. This means that not only is the *numeric value* determined by the binary representation, but where the *decimal point* is located is encoded as well.

The standard convention for 32 bit floating point numbers—called the *IEEE Standard for Floating-Point Arithmetic (IEEE 754)*—splits the bits into groups like this:



The *mantissa* is the numeric portion of the encoding, the *exponent* indicates where to place the decimal point, and the *sign* denotes if the number is negative or positive. If you have ever used [scientific notation](#), then you have written a value similar to the way computers store and process floating point numbers.

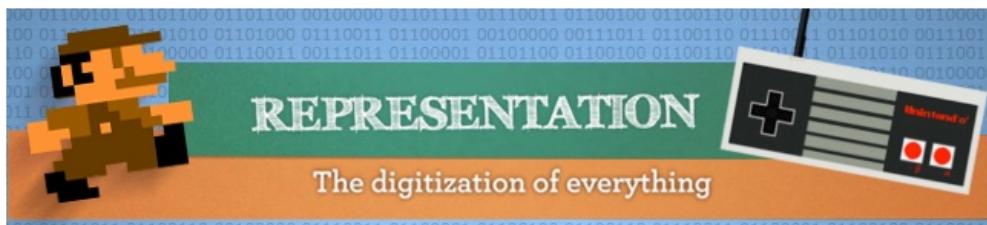
UNIT PROJECT:

Unintend'o Game Controller

Highlights

- You will develop a Scratch program that acts as a device driver for a video game controller interface.
- You will map each of six controls (UP, DOWN, LEFT, RIGHT, A, and B) to individual bits.
- You will map each binary pattern of button presses to different game actions (e.g., walk forward, walk backward, turn left, turn right, jump, duck, whirl, leap, crawl, etc.).
- You will use a list to track the history of button presses.
- You will write detailed specifications and justifications for each button-to-action mapping of your design.
- You will collaborate with your peers throughout the design and development process to determine end-user requests for features and to share feedback on design and implementation strategies.
- You will write documentation detailing the use of your program and its features using appropriate terminology.

Unintend'o Project: Binary Mapping



Assignment

Today, your groups will begin work on the Unintend'o Controller Project by mapping each of the controller button presses to a binary code. These codes will represent the digital instructions to make the Fabio character move.

For example,

\leftarrow	\rightarrow	\uparrow	\downarrow	B	A	Result
1	0	0	0	0	0	Fabio walks left
0	1	0	0	0	1	Fabio jumps right

Indicate how each binary sequence is mapped to a character action. This behavior should be expressed on the right side of the arrow.

A draft of the binary code mapping table is due by the end of the period, but you will also be able to work on the Scratch program once you have finished your tables. Remember to include combo moves in your table.

The following is an empty table you may use. You may create your own instead if you like.

Unintend'o Controller Mappings

UNIT TOPIC:

Digital Approximations

Digitization

- You will examine the implications of variable-width encodings (e.g., Morse code) vs. fixed-width encodings (e.g., Baudot code).
- You will design a digital representation.

Analog vs. Digital Data

- You will analyze the differences between discrete (digital) and continuous (analog) representations of natural phenomena.

Perfect Copies

- You will analyze the extent to which digital approximations accurately reflect the reality that they represent.
- You will examine the social implications of the ease with which perfect digital copies can be made.

UNIT TOPIC:

Digital Approximations

Digitization

- You will examine the implications of variable-width encodings (e.g., Morse code) vs. fixed-width encodings (e.g., Baudot code).
- You will design a digital representation.

Variable vs. Fixed-Width Encodings

In 1836, people wanted to communicate with each other across great distances instantly—just like we do today—but technologies such as mobile phones and the Internet were still well over 100 years away.

Telegraphy was the first step toward bridging this communication gap. However, in its infancy, long-distance communication was limited to sending two states—either an *electric signal* or *no electric signal*. Samuel Morse developed a code comprised of *dots* and *dashes* (or *dits* and *dahs*) that could be sent electronically via wires that spanned miles and miles from one city to another. On one end, a telegraph operator would use a *key* to send a message—and on the other, an operator would hear the Morse Code, and transcribe it into letters, numbers, and punctuation. A skilled operator could instantaneously translate the Morse Code into alphanumeric symbols. It was easily the fastest way to communicate with people across the state, country, or even internationally.

How effective was Morse Code at sending messages? Let's compare it to SMS text messaging:



<https://www.youtube.com/embed/pRuRE-Bwk1U>

This video seems a bit dated—they are using flip phones after all. What factors do you think might change the outcome today?

Morse Code

Morse code is a *variable-width* encoding. This means that each of the characters represented by the dashes and dots of Morse may be different lengths. This was done by design for *efficiency*. Samuel Morse knew that some characters would be sent much more often than others, and so the information required to send them should be less. For example, 'E' and 'T' each occur often and, as such, are represented respectively by one *dot* and one *dash*. 'Z', on the other hand, occurs infrequently, and so requires 4 bits to send (*dash dash dot dot*).

However, variable-length codes introduce some added complexity—how does one know where one character ends and another begins? In other words, what differentiates two 'E's in a row from one 'I'? Morse used time—the delay between characters—to delimit characters. But, in a way, this sacrifices robustness for efficiency. The sender's perception of time may be different than the receiver's—particularly at high speeds.

Baudot Code

	1	2	3	4	5		1	2	3	4	5
A	•	•				Q	1	•	•	•	•
B	?	•		•	•	R	4		•		•
C	(•	•	•	S	'	•		•	
D	²	•		•		T	5				•
E	³	•				U	7	•	•	•	
F	¹	•	•	•		V)	•	•	•	
G	³	•	•	•		W	2	•	•		
H	⁵	•	•	•		X	₂	•	•	•	
I	⁸	•	•			Y	₆	•		•	
J	⁷	•	•			Z	-	•		•	
K	⁹	•	•	•	•	FIG SPACE	FIG SPACE	•	•	•	
L	₁	•		•		LTR SPACE	LTR SPACE		•		
M	,		•	•	•	LINE	LINE		•		
N	-		•	•	•	PAGE	PAGE				
O	⁹		•	•	•	MFH	MFH	•	•	•	
P	O	•	•	•		COL	COL	•			

#12345 gives invisible correction on page printers & % on slip printers.

International Morse Code

1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to seven dots.

A	• -	U	• • -
B	- • •	V	• • -
C	- • -	W	- • -
D	- •	X	- • -
E	•	Y	- • -
F	• • -	Z	- • -
G	- -		
H	• • •		
I	• •		
J	• - - -		
K	- • -		
L	- • • •		
M	- -		
N	- •		
O	- - -		
P	- • - -		
Q	- - • -		
R	- • - -		
S	• • •		
T	-		
1	• - - - -		
2	- • - - -		
3	- - • - -		
4	- - - • -		
5	- - - - •		
6	- - - - -		
7	- - - - - -		
8	- - - - - -		
9	- - - - - -		
0	- - - - - -		

Émile Baudot, a French telegrapher, sought to minimize this ambiguity by creating a fixed-width code. Every character sent was 5 bits long. There is no confusion over where one character ends and another begins because they are *all the same length*.

This is effectively similar to ASCII and Unicode. When the computer was young and the standards committee was designing a character set, they selected Baudot's method rather than Morse's. They were selecting for robustness over efficiency.

Like so many choices made in the history of computer science, there is no single correct answer. Each choice is a selection among trade-offs—efficiency, robustness, correctness, ease of use, etc.

UNIT TOPIC:

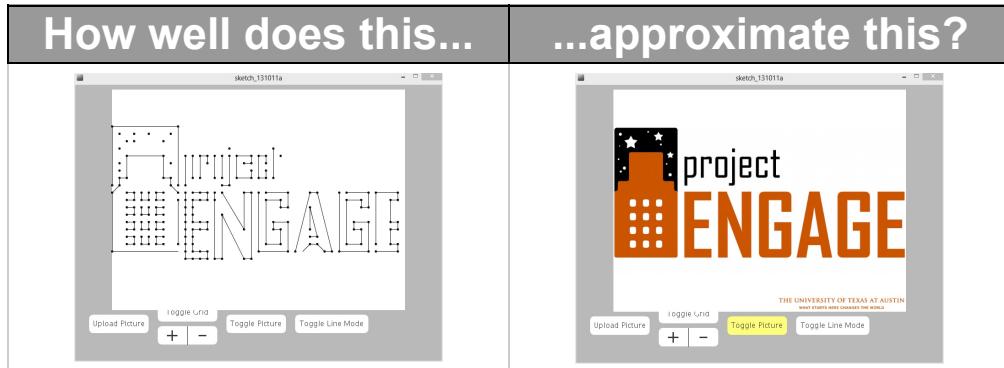
Digital Approximations

Analog vs. Digital Data

- You will analyze the differences between discrete (digital) and continuous (analog) representations of natural phenomena.

Approximating Physical Media

Approximating Physical Media with Coordinate Grids



In the next unit, [Unit 4: Digital Media Processing](#), you will investigate different ways that bits may encode different types of media. Additionally, you will learn to use a new, more powerful programming language called [Processing](#) to perform image processing. As a teaser, this activity includes a pre-written application in Processing that allows the user to create a point-by-point *approximation* of an image.

The Processing Application

First, you will have to open the Processing application, which should already be installed on your computer (the icon for Processing resembles the image to the right). If you cannot find the Processing application, search your computer for "Processing." If the Processing application has not been installed, you will have to download it from the [Processing Download page](#).



Note that you do not need to donate anything to download Processing—simply click the \$0 option and then Download.

Activity Instructions

Next, download this activity's Processing source file:

Click to Download: [PictureGridDrawingActivity](#) Unzip

PictureGridDrawingActivity.zip and locate the .pde file in the resulting folder. Open the source code within Processing by selecting *Open...* in the *File* menu and navigating to the .pde file. As Processing is a text-based language, the code will look much different than Scratch. However, the constructs are much the same.

1. Take a second to browse through the source file, and look for any keywords that you may recognize from Scratch.
2. To execute the program, press the play button.
3. Click [Upload Picture](#), and choose an image file to load into the program window.

Next you will "digitize" this image by redrawing it along a grid.

4. Select **Toggle Grid**, and with the grid on, click carefully on grid intersections in order to place points. Place points around your image in order to represent it as best you can. To remove an unwanted point, click it again.
5. The **Toggle Line** function will cause the next series of points placed to be directly connected with lines. **Note:** Clicking dots will toggle them on and off (as a sort of 'UNDO' feature). This means that there can only be one line leaving and entering each dot.
6. Try to create a point-by-point representation of your image simply by creating the points that define/outline its major features.
7. Toggle both the grid and the image so that they are not shown. Show the result to some of your peers. Can they discern the original image from your representation?
8. Try it with a new grid size. Adjust the granularity of the grid with the **+ | -** buttons. Attempt the drawing/guessing activity once again with the new grid. How does this change the effectiveness?

The short tutorial below demonstrates how to run and work with the source file in the Processing application. Note that this program has a limited set of features. It may not perform every task you would like it to in order to best create your representation. You will have to experiment with the program to create your representations:



https://www.youtube.com/embed/P37_1h_kJbY

Digital vs. Analog

Think About It

Which is better, more valuable, and/or more important? Digital or analog artifacts?

Spend 5 minutes recording your thoughts about *digital* representations vs. the *analog* items they represent. For example, the *Book of Kells* (written and illustrated *circa* 800 CE) is one of the most famous illuminated manuscripts of all time. Is the physical copy of the *Book of Kells* better than the digital images of the book?



UNIT TOPIC:

Digital Approximations

Perfect Copies

- You will analyze the extent to which digital approximations accurately reflect the reality that they represent.
- You will examine the social implications of the ease with which perfect digital copies can be made.

Digital Copies

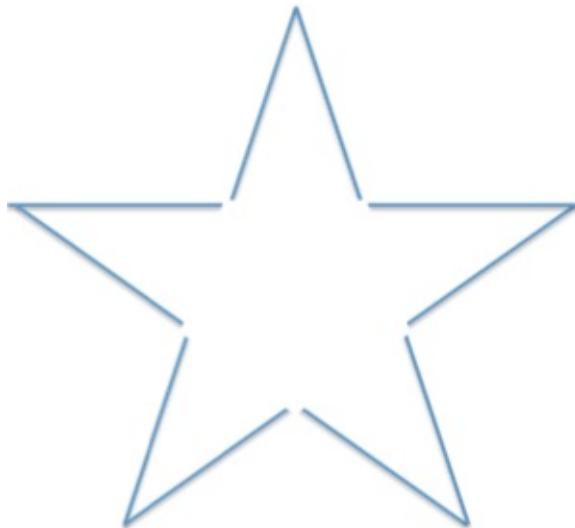
Experiment

To learn why digital copies are easy to copy, let's try an experiment:

1. Take out a half-sheet of paper.
2. You will have two minutes to do the following:
 - On one side of the paper, copy the following list of numbers down exactly.
 - On the other side, copy the image of the star exactly.

Wait until your teacher instructs you to begin, then copy them as best you can. Your neighbor will judge your work, and as a class, you will discuss it afterward.

Copy This Star:



Copy This List of Numbers:

10
17
14
4
3
12
17

12

6

4

10

17

Perfect Imperfection

Common misconception: *Digital copies are "perfect."*

- Whereas the process of copying does not suffer degradation, the digital copies, like the original digital file, are still only approximations of the natural object. What actually is copied perfectly in a digital file is the binary representation, including any flaws or loss through digitization. This also assumes the process of copying is careful. If not, error can be introduced causing an "imperfect" digital copy.

Quiz

To see how well you understand the perfect imperfection of digital copies, answer the following questions as best you can.

1) *What* is perfectly copied in a digital copy?

Answer:



2) Which copy is most like the original? Why?

- A digital image sent from one phone to another
- A digital image printed on an inkjet printer

Answer:



BIG PICTURE:

Legality of Reselling Digital Music

Highlights

- You will examine and discuss the legality of reselling "used" digital music.

Reselling Digital Music

Should It Be Illegal to Resell “Used” Digital Music?

Think about purchasing an item. When you are done with it, what are your options for disposing of it? You can throw it away, recycle it, give it away, store it, or sell it—depending on what *it* actually is. What if the item is a piece of recorded music? As of March 2013, it actually depends on how it is distributed. Today you will debate the legality of reselling used digital music.

1. First, your teacher will assign you to either the **for** or **against** team.
2. Next, read the following article, considering your team's perspective:
[Is It Illegal to Resell “Used” Digital Music?](#)
3. The Digital Millennium Copyright Act (DMCA) has been a benefit and a challenge in making copyrighted digital material widely available. Read this brief summary provided by the American Library Association, [“DMCA: The Digital Millennium Copyright Act”](#), and apply it to your team's argument.
4. Then, write a reflection about the decision considering your team's argument. Your teacher will ask your team to get together and allow you to brainstorm your arguments before the debate.
5. Last, your team will debate according to the debate protocol of your teacher's choosing. Your teacher will be the judge and decide which team puts forth the better argument.



Make sure you address the *issues*, not the other team!

UNIT PROJECT:

Unintend'o Game Controller

Highlights

- You will develop a Scratch program that acts as a device driver for a video game controller interface.
- You will map each of six controls (UP, DOWN, LEFT, RIGHT, A, and B) to individual bits.
- You will map each binary pattern of button presses to different game actions (e.g., walk forward, walk backward, turn left, turn right, jump, duck, whirl, leap, crawl, etc.).
- You will use a list to track the history of button presses.
- You will write detailed specifications and justifications for each button-to-action mapping of your design.
- You will collaborate with your peers throughout the design and development process to determine end-user requests for features and to share feedback on design and implementation strategies.
- You will write documentation detailing the use of your program and its features using appropriate terminology.

Unintend'o Project: Programming



Assignment

Your job today is to work on the actual Scratch program you will create for the Unintend'o Controller project. You may take a variety of approaches to make the program function and to make the controller/game fun to use and play! Remember, the Scratch program should:

- Map each key press to a binary sequence. Note that interaction between the keys (e.g., **←**, **→**, **↑**, **↓**, **A**, and **B**) and the program is restricted to only this. In other words, the key presses should only be used to build the binary sequence.
- Cause Fabio to act according to the binary sequence you have created. **Note that Fabio does *not* directly reference the controller or keyboard events, but rather acts according to the value of the binary sequence.**
- Use loops for repetitive tasks.
- Use conditionals for branching/decisions.
- Work as specified in the technical specifications your group is creating.

For your reference, here is an [example solution](#). Note that the example solution contains no combo moves, but simply captures the basic

UNIT TOPIC:

Lists

Making a List

- You will examine the use of lists as ordered data structures that may contain multiple values.
- You will investigate the use of index values to represent the position of an item in a list.
- You will analyze the implications of accessing an index position beyond the bounds of a list.

Processing a List

- You will investigate common operations for processing elements of a list, including searching for an element, removing an element, swapping the positions of two elements, or sorting an entire list into ascending or descending order.

Sorting a List

- You will examine the implications of case-sensitivity on ordered lists of strings.

Lists in Action

- You will integrate multiple list operations into a single application.

UNIT TOPIC:

Lists

Making a List

- You will examine the use of lists as ordered data structures that may contain multiple values.
- You will investigate the use of index values to represent the position of an item in a list.
- You will analyze the implications of accessing an index position beyond the bounds of a list.

Making a List

Make a List

We are going to start working with lists! Lists are a type of **data structure**, a particular way of storing data. Many objects can be easily stored in a list, and so lists have become a very useful and ubiquitous data structure in computer science. For example, if we are developing a game, and we want to keep track of the people that have played our game, we might keep their names in a list.

As with anything new, the first thing to do is explore! Make a new list named **players** by clicking on the button **Make a List**, as shown below.



Below the name of your new list, you will see a group of blocks that operate on lists. Use the **players** list to experiment and determine what these blocks do before going on to the next step. **Remember to check the box next to **players** to view its contents on the stage.**



Weird Cases in Lists

Instructions

This quiz is not graded. By this point, we do not assume you know the answers to all of these questions, and we encourage you to make a best guess for each question. We also want to reinforce that the answers to these questions will be important later on, but that you can figure them out whenever you need to!

The questions in this quiz are essentially "*what does this do?*" questions. When dealing with these questions, you must realize that people designed every programming language. So, people weighed pros and cons and made a decision about what the language should do in each situation (its *specifications*). You might disagree with them about what should happen, but it is important to realize that **you can always test code** to figure out what it does.

The following questions test *boundary cases*, where what the code should do is not intuitive.

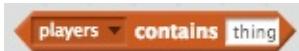
1) Assuming that you only have five elements in your players list, what would happen if you ran the script below?



- A. "thing" will be added to the end of the list
- B. "thing" will be added to the beginning of the list
- C. The list will remain unchanged
- D. Blank entries will be added to the list so that the new element can be inserted at the index specified

Answer:

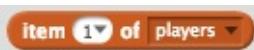
2) Are the words in a list case-sensitive? In other words, is "Thing" different from "thing"?



- A. Yes
- B. No

Answer:

3) What happens when you run the script below with a list that has no elements?



- A. The block will report a blank.
- B. The block will turn red and there will be an error.

- C. The block will report the item that was most recently at the specified index.
- D. The block will not run.

Answer:

- 4) What gets added to players when you run the following script?



Answer:

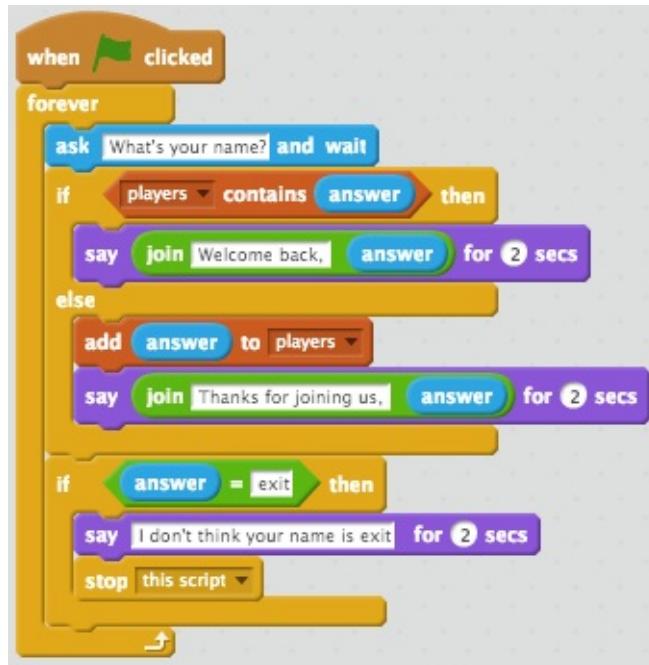


Reading a List

Read a List of Names

The following script is intended to keep track of the names of the players of a game, but it has an error! Find and fix the error. **Submit your solution.**

The program is available [here](#), as a Scratch project. Once you are done with it, you may find it useful in your other projects.



UNIT TOPIC:

Lists

Processing a List

- You will investigate common operations for processing elements of a list, including searching for an element, removing an element, swapping the positions of two elements, or sorting an entire list into ascending or descending order.

Processing a List

Index

We want to have the character read all of the names from our **players** list. This is a very common type of problem when dealing with lists: we want to do the same thing for each item in the list. To tackle this problem, we are going to use a variable called **index** to keep track of the position of the element in the list we are processing.



Once you get the script above to work, try to make a script that will read all of the player names and then say them all together, with the appropriate commas, spaces, and the word and, as shown below. Remember to save your work!

List	Output
<pre> players 1 Alicia 2 Amy 3 Bradley 4 Deborah 5 Kim + length: 5 </pre>	<p>A Scratch cat character is shown with a speech bubble containing the text: "Hello Alicia, Amy, Bradley, Deborah, and Kim".</p>

Hint: Before you make the final result with the serial commas and the "and," try to make a script that will just **join** together all the names into one variable, and then have it **say** that variable. The following block is a good starting point:



Index Variables

Practice with Index Variables

As we have seen in the previous section, we generally use an index variable to process the elements of a list. Below, we have reproduced the script that makes the character **say** the names of the players in a list.



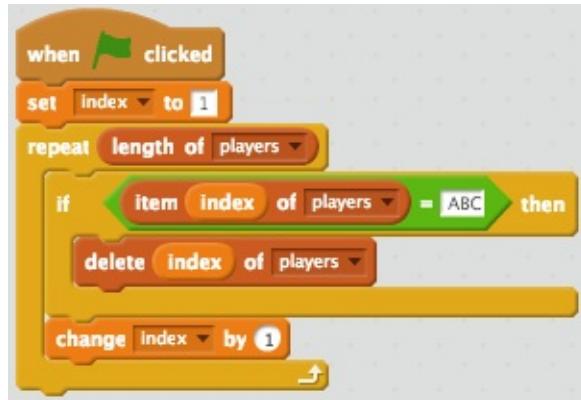
Modify the script to make the character **say** the following:

1. Every other name in a list.
2. Every third name in a list.
3. Names in a list in reverse order.
4. Names in a list that start with the letter **C**.
5. Names that are longer than four letters.

Remove from a List

Remove

The script below is supposed to remove all of the occurrences of "ABC" from the **players** list.



Find a list for which the script below will not remove all occurrences of **ABC** (such lists DO exist), and explain your findings:

- When does the script not work?
- Why is not working?
- Can you generalize what type of lists do not work with this script?
- How might it be fixed?

The script is available [here](#) as a Scratch project.

Sentences as Lists

Processing a Sentence

The pattern of using an index variable to process a bunch of values is pretty common in programming, and is not specific to lists alone! The block on the left below is the one that we used to say all of the names in our player list. The block on the right can be used to say all of the letters in a sentence like "Go Longhorns!"

Players	Sentence
<pre> when I receive [read player list] set [index v] to [1] repeat ([length of] [players]) say [item [index] of [players]] for [1] secs change [index v] by [1] end </pre>	<pre> when [green flag] clicked set [my sentence v] to [Go Longhorns!] set [index v] to [1] repeat ([length of] [my sentence]) say [letter [index] of [my sentence]] for [1] secs change [index v] by [1] end </pre>

There are striking similarities between the two scripts, and so we should be able to perform the same tasks on sentences as those we did on lists. Write blocks that make the character say the following:

- Every other letter in a sentence.
- Every third letter.
- The letters in reverse order.
- Say a range of letters, using either a `repeat until` or a `repeat` block.

Note: Often, when writing a script, if we have already written a similar script for another purpose or data structure, we can either modify the script for the new functionality, or we can create a more general script that can be used for both purposes.

UNIT TOPIC:

Lists

Sorting a List

- You will examine the implications of case-sensitivity on ordered lists of strings.

Swaps

Reordering Lists

Sometimes, we care about the order of a list (e.g., dictionary) and sometimes it doesn't matter (e.g., shopping list). A very common task for lists is reordering them, either to impose an order that did not exist before (such as sorting a list — alphabetizing a list of names) or changing the ordering (such as reversing a list — taking the alphabetized list of names and making it a reverse-alphabetized list).

At the heart of the matter, reordering a list involves swapping the positions of two items repeatedly:

- In the case of a reversal, this might entail taking the first item and swapping it with the last, then the second item with the second to last, etc.
- In the case of sorting a list of names, it might involve taking what should be first (alphabetically) and swapping it with what actually is first, then continuing down the line.

NOTE: Sorting is a very well-studied topic in computer science. The sorting algorithm described above is a "selection sort." Note the various sorting algorithms described [here](#). "Selection sort" is not the most efficient way to sort a list, but it is the way humans typically manually alphabetize things. How might you apply one of the more efficient algorithms described to physically alphabetizing file folders, books, etc.?

The Trouble with Swapping

Because programs are ordered processes, the order in which events occur matter. What does this have to do with swapping? Well, consider the following algorithm for swapping:

X	Y
15	32

1. Copy X to Y.
2. Copy Y to X.

Seems simple enough until you step through it. After Step 1, our table now looks like this:

X	Y
15	15

Step 2 now depends on the "old" value of X, which was overwritten. We need to "save" that value somewhere in order to use it later. We cannot save it in Y, because then we'll have the same problem in reverse. We need a third variable:

X	Y	SwapSpace
15	32	

Using this third variable, write out the steps that would constitute an algorithm for swapping two values.

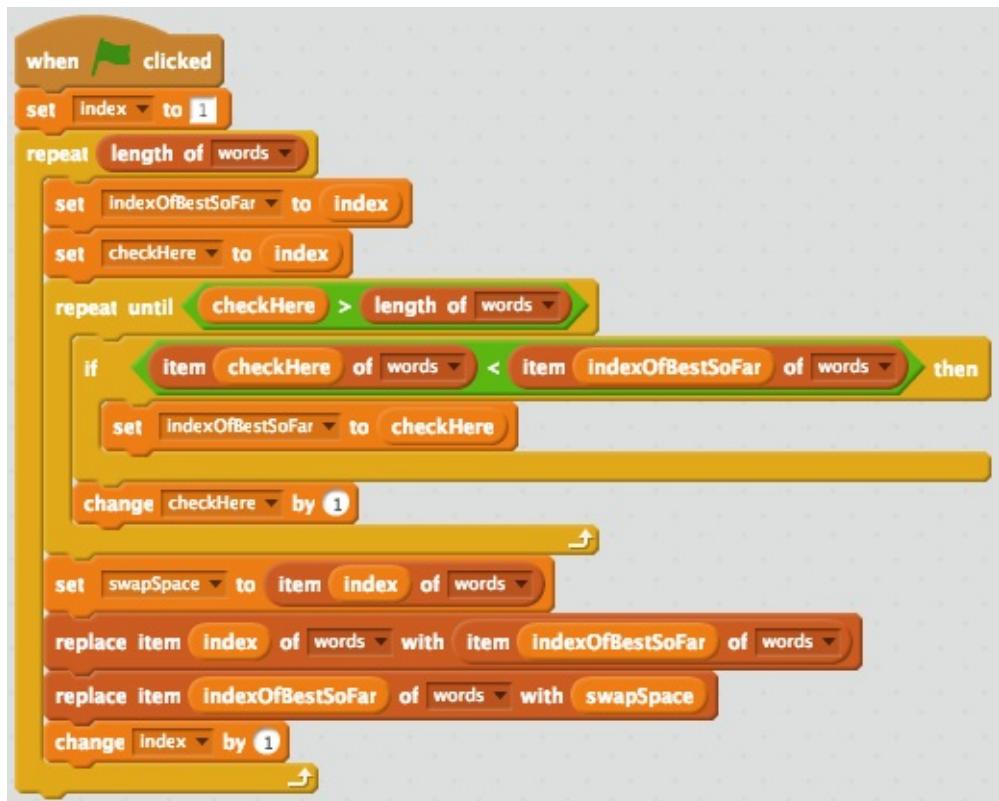
Reorder

Reordering

As the previous assignment indicated, reordering a list is a very common occurrence within a computer program. Consider some of these common tasks:

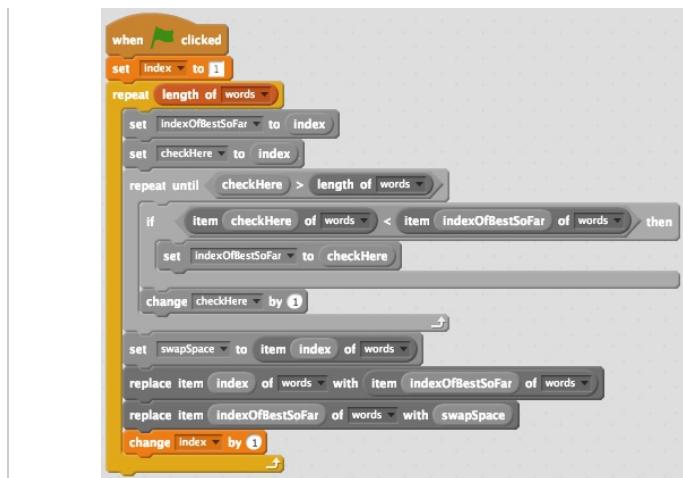
- Rearranging the apps on an iPhone
- Sorting items in an online shop by "bestselling," "price," or "best reviews"
- Rotating an image
- Ranking search results

All of these tasks are essentially reordering a list of items so that it is displayed in a different way. We've output lists in different orders before now, but we have not actually altered a list's contents so that they are stored in a different order. Examine this (possibly daunting) program:

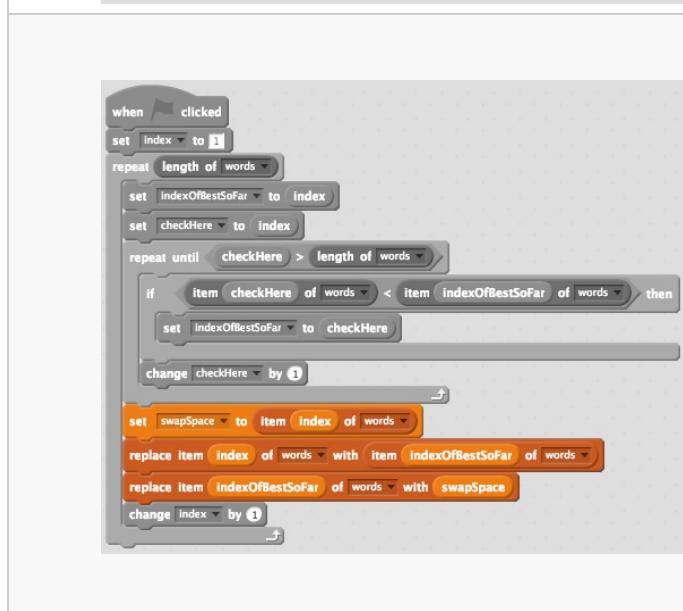


This program reorders a list, but how? It's not actually as complex as it looks at first glance. Some of the parts of this program are directly copied from programs we have dealt with previously. For example,

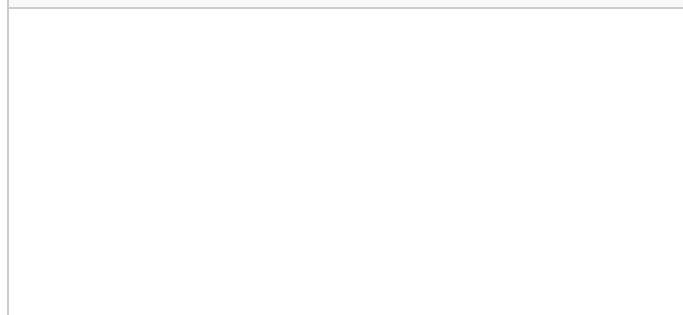
Code	Explanation
	The highlighted section is just the framework



for processing a list — going through each item in the list one after the other. For reference, see the previous "Process a List" activity. Compare the program there, which simply instructs Scratch to say each item in the list, to the highlighted blocks here.



Here, the highlighted section is the algorithm for swapping two items in a list. This is a direct application of the previous "Swaps" activity, only rather than swapping two singular variables with one another, we are swapping two list items. Compare these blocks with those in the previous activity. How are they the same? How are they different?



Finally, let's examine the remaining blocks. We know that we are processing a list, swapping out some items with others, but which items are we swapping?

The highlighted blocks determine which items are swapped.

indexOfBestSoFar

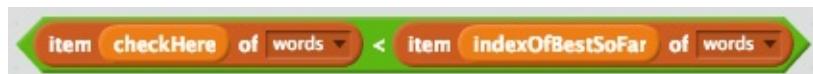
holds the index of the tentative item to be swapped.

checkHere

is an index variable that is used to process everything in the list that has not been reordered. The highlighted loop starts

in the unsorted part of the list of words and repeatedly checks if any/all of the items are alphabetically before the rest. By the time it reaches the end, the first word alphabetically is denoted by `indexOfBestSoFar`. It will be then swapped with the first item, and so on.

The real workhorse of the program is this block of code:



This comparison block is the portion of our program that determines that we want the listed to be sorted in *alphabetical* order. If we were to desire a different ordering, such as by `length of [word]`, *only this block would differ in our resulting program!*

Experiment

Access the [Reorder!](#) project on the Scratch website and play around with it. Work with it to sort items in numerical order, reverse alphabetical order, and ordered by `length of [word]`.

Describe each of the changes you made while experimenting (*INCLUDING THE FAILED ATTEMPTS!*), and indicate why they work the way they do.

UNIT TOPIC:

Lists

Lists in Action

- You will integrate multiple list operations into a single application.

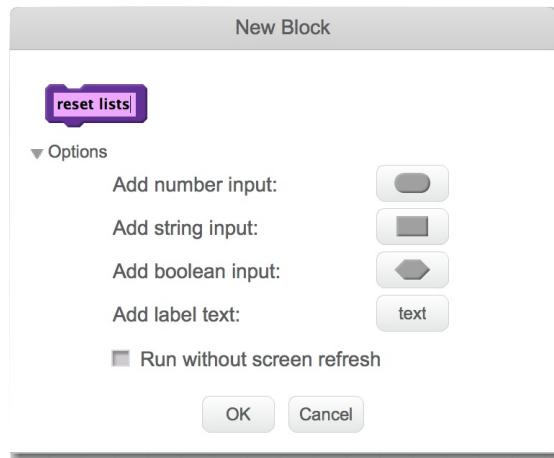
MultiSet to Set

Now that you have learned the basic operations to add, access, and remove items in a list, let's integrate them into a basic **procedure**.

A procedure is a group of blocks that have been combined to perform a specific task. In Scratch, this is done with the "Make a Block" button in the "More Blocks" pane.

To demonstrate, let's create a block that resets two lists to a default configuration.

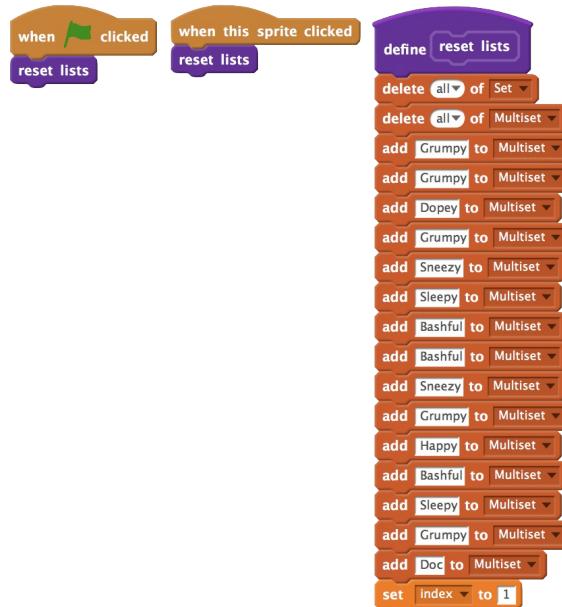
First, we will click the "Make a Block" button, and name the block something descriptive, like "reset lists":



Next, we will associate some code with it. Any time the newly created `reset lists` block is used, all of the code under the `define reset lists` block will be executed:



Finally, we may use the new **reset lists** block anywhere in our code. In this example, the **reset lists block** is called anytime the sprite is clicked:



Your Task

Your task is to create a block that creates a set from a multiset. A multiset is a collection of items that may contain duplicates; a set does not contain duplicate entries. Order does not matter in either collection.

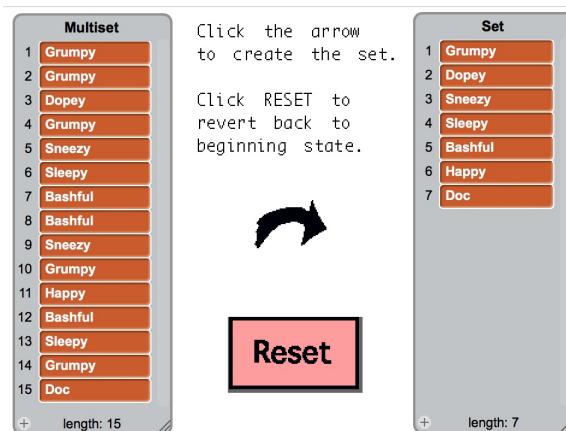
Example:

Given the multiset `{ a, d, c, a, c, b }`, you are to create a block that will populate the corresponding set `{ a, d, c, b }`. Note that because order does not matter, `{ b, a, d, c }` is also a valid solution.

The Scratch starter code provided [here](#) includes two lists entitled multiset and set. You will create a block to perform the conversion as part of the **DeDupe** sprite, an arrow that begins the "de-duplication" conversion process when clicked.

Note: A completely correct solution will work for any list of items, not just the sample one provided.

The following screenshot depicts the output of a correct solution:



After you have a working solution, submit your code and a description of how you tested it. Note that you should try to test it on more lists than just the one provided!

Extensions

For an additional challenge, try the following:

- Create a block that incorporates the selection sort code in [Reorder](#) so that the items in Set are always in alphabetical order.
- Create a block that converts MultiSet into a Set rather than simply *creating a new Set*. To do this, you will need to selectively remove items from MultiSet.

UNIT PROJECT:

Unintend'o Game Controller

Highlights

- You will develop a Scratch program that acts as a device driver for a video game controller interface.
- You will map each of six controls (UP, DOWN, LEFT, RIGHT, A, and B) to individual bits.
- You will map each binary pattern of button presses to different game actions (e.g., walk forward, walk backward, turn left, turn right, jump, duck, whirl, leap, crawl, etc.).
- You will use a list to track the history of button presses.
- You will write detailed specifications and justifications for each button-to-action mapping of your design.
- You will collaborate with your peers throughout the design and development process to determine end-user requests for features and to share feedback on design and implementation strategies.
- You will write documentation detailing the use of your program and its features using appropriate terminology.

Unintend'o Project: Rubric Check



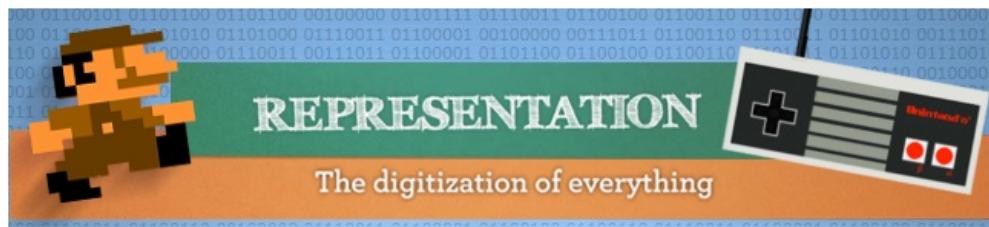
Instructions

This activity provides you and a partner group time to provide one another *critical feedback* about the progress of your projects.

1. Pair up with one another group.
2. Rate the components of your partner group's Scratch program and the documentation of their approach according to the [Unintend'o Project rubric](#). You may write this out or print the rubric and circle components on it.
3. Review your partner group's work and provide them with documentation that describes what you *Like*, what you *Wonder*, and what their *Next Steps* should be.
4. When both you and your partner group have completed the previous steps, share your feedback with one another.

Listen to what your partner group says! You do not need to heed all of their advice, but consider it wisely. The whole idea is to have a critical third party provide ideas to make your project better before you submit the final version for a grade.

Unintend'o Project: Gallery Walk



Instructions

To present your controller to the class, you will participate in a **gallery walk**. An art gallery refers to an art museum (or any collection of rooms) that houses an exhibition of an artist's work for exhibition or sale ([Wikipedia](#)). In this case, the gallery will be composed of your Unintend'o projects. All students who have completed the project will display their Scratch programs and technical specifications. Then, you will rotate around the classroom, experimenting with the controls as outlined in the technical specifications. At the end of the gallery walk, you will vote for your favorite controller artifacts:

- Whose controller schemes are the most efficient? ...the most effective? ...the most expandable? ...the most clever?
- Which special combo moves are the most novel?
- **Which controller would make for the best game?**



UNIT 4

Digital Media Processing

Building upon their earlier, visual programming experiences with Scratch, this unit guides students through the transition to programming in a high-level, procedural language through a brief introduction to Processing. By familiarizing themselves with a text-based environment that more closely reflects the actual programming tools used in industry, such as Java, C++, or Python, students will be better equipped for continuing their studies in computer science beyond the scope of this course.

With the help of Processing's graphical programming model that is designed to simplify the task of creating sophisticated, visual artifacts, students will explore the characteristics of the RGB color model and its use in encoding digital images. For the unit project, they will apply these concepts toward the implementation of a series of algorithmic filters for digitally modifying images to achieve various visual effects. Finally, students will also investigate the methods of representing and modifying digital audio, including Auto-Tune and audio compression.

UNIT PROJECT:

Image Filter

Highlights

- You will utilize pair programming to design and implement a program for filtering digital images.
- Using the Processing programming language, you will develop code to systematically transform an image by mathematically manipulating its bits, pixel by pixel.
- You will write documentation detailing the use of your program and its features using appropriate terminology.
- You will explain your design and implementation choices by demonstrating and sharing your finished programs with your peers.

Image Filter Project

“You don’t take a photograph, you make it.” – Ansel Adams



<https://www.youtube.com/embed/p5QQNmkSE5Y>

All digital media consists of bits and their abstractions. Manipulating bits can make abstractions more useful, usable, or beautiful. Much like wizards in a fantasy setting, programmers are able to change the very core representations of reality.

Computer scientists manipulate bits to achieve a wide variety of outcomes. Image editors can transform many characteristics of images with ease. Productivity software can perform complex mathematical functions automatically. Video game environments can be rendered dynamically, and more.

With computational thinking and the proper programming skills, anyone can manipulate bits.

Assignment

Create a filter to transform digital images by programmatically manipulating pixels.

Working in pairs, your task is to design and develop a program for filtering digital images. More specifically, you must use the [Processing](#) programming language to code a program capable of systematically transforming an image by mathematically manipulating its bits. Your program must allow users to upload original digital images of various formats, perform transformations on these images programmatically, and save the manipulated results. How



will your image filter program function? What image features, attributes, and representations will it transform, and how will it transform them? As you work through this module, continuously think about how the skills and knowledge you gain may apply to your project.

Submission

Your submission will be an original program coded with the [Processing](#) language. You must submit the 'sketchbook' [.pde](#) file, along with documentation describing its functionality. Download, execute, and build upon this starter code:

Click to Download: [FilterProjectStarter](#)

Your image filter program must:

- Perform 2 tasks on an image at the **pixel level**.
- Perform 1 task on an image by **reordering pixels**.
- **Use Processing constructs** to accomplish this task, including:
 - the `setup()` and `draw()` functions,
 - branching control flow (`if` / `else`),
 - color functions (`color()`, `red()`, `green()`, etc.), and
 - mouse/keyboard interaction.
- Include **documentation** detailing its use, appropriately using key terminology as necessary.
- Your program should be **aesthetically pleasing** and easy to use.

When you are finished, you will submit the source code of your Processing program, which will be graded using the attached rubric. You will then review one other group's submission, and reflect upon any differences from your own work.

Learning Goals

Over the course of this module and this project, you will learn to:

- use a text-based programming language ([Processing](#))
- use appropriate computer science terminology
- represent color using the RGB color model
- create vector and raster-based images
- transform behaviors by manipulating underlying representations
- create and modify digital audio
- analyze the costs and benefits of encoding schemes
- evaluate ethical practices related to digital media production, consumption, and ownership.

Rubric

Content Area	Performance Quality
--------------	---------------------

	5 Both loops and conditionals have been added to the program. AND All loops and conditionals are used effectively and correctly with purpose in the program.	3 Loops or conditionals (but not both) have been added to the program AND all loops or conditionals are used effectively and correctly with purpose in the program. OR Both loops and conditionals have been added to the program but not all loops or conditionals are used effectively and correctly with purpose in the program.	1 Loops or conditionals (but not both) have been added to the program. AND Not all loops or conditionals are used effectively and correctly with purpose in the program.	0 Not enough criteria are met in order to award any credit.
Filter 1	10 Program includes an image filter that alters the image at the pixel level (not using a built-in Processing filter function). AND The filter uses Processing constructs appropriately and effectively.	7 Program includes an image filter that alters the image at the pixel level (not using a built-in Processing filter function). AND The filter uses Processing constructs appropriately.	4 Program includes an image filter that alters the image at the pixel level (not using a built-in Processing filter function), but Processing constructs are not used appropriately.	0 Not enough criteria are met in order to award any credit.
Filter 2	10 Program includes a second image filter that alters the image at the pixel level (not using a built-in Processing filter function). AND The filter uses Processing constructs appropriately and effectively.	7 Program includes a second image filter that alters the image at the pixel level (not using a built-in Processing filter function). AND The filter uses Processing constructs appropriately.	4 Program includes a second image filter that alters the image at the pixel level (not using a built-in Processing filter function), but Processing constructs are not used appropriately.	0 Not enough criteria are met in order to award any credit.
	10 Program	7 Program	4 Program	0 Not

	Filter 3	<p>includes a third image filter that alters the image <i>by reordering pixels</i> (not using a built-in Processing filter function).</p> <p>AND</p> <p>The filter uses Processing constructs appropriately and effectively.</p>	<p>includes a third image filter that alters the image <i>by reordering pixels</i> (not using a built-in Processing filter function).</p> <p>AND</p> <p>The filter uses Processing constructs appropriately.</p>	<p>includes a third image filter that alters the image <i>by reordering pixels</i> (not using a built-in Processing filter function), but Processing constructs are not used appropriately.</p>	enough criteria are met in order to award any credit.
	Aesthetics and Functionality	<p>10 All filters created in the program are aesthetically pleasing.</p> <p>AND</p> <p>Provides a name for all of the filters (e.g., sepia, blur, shiny) and change the values of the buttons to match.</p> <p>AND</p> <p>Program is easy to use and allows multiple filters to be applied simultaneously.</p>	<p>7 All filters created in the program are aesthetically pleasing.</p> <p>AND</p> <p>Some of the filters have been provided names (e.g., sepia, blur, shiny) and the values of the buttons are changed to match.</p> <p>AND</p> <p>Program is somewhat easy to use and may allow multiple filters to be applied simultaneously.</p>	<p>4 Some filters created in the program are aesthetically pleasing.</p> <p>AND</p> <p>Some of the filters have been provided names (e.g., sepia, blur, shiny) and the values of the buttons are changed to match .</p> <p>OR</p> <p>Program is somewhat easy to use and may allow multiple filters to be applied simultaneously.</p>	<p>0 Not enough criteria are met in order to award any credit.</p>
	Documentation	<p>5 Explains each filter's functionality and purpose and describes the major code segments within the program.</p>	<p>3 Explains each filter's functionality and purpose OR describes the major code segments within the program.</p>	<p>1 Document exists, NEITHER describes explain each filter's functionality and purpose NOR the major code segments within the program.</p>	<p>0 Not enough criteria are met in order to award any credit.</p>
TOTAL					50 pts

UNIT TOPIC:

Procedural Programming

Introduction to Processing

- You will explore the capabilities of a text-based programming language (Processing).
- You will compare and contrast the programming capabilities of a visual programming language (Scratch) with those of a text-based programming language (Processing).

Drawing

- You will write programs that make use of parameterized methods to invoke specific behaviors.
- You will understand the importance of using proper punctuation and syntax when coding in a text-based programming language.
- You will write code using common programming constructs like conditional if() for selection and while() loops for iteration.

Mouse Interaction

- You will use event handlers to animate on-screen effects and respond to mouse input.

Keyboard Interaction

- You will use event handlers to animate on-screen effects and respond to keyboard input.

UNIT TOPIC:

Procedural Programming

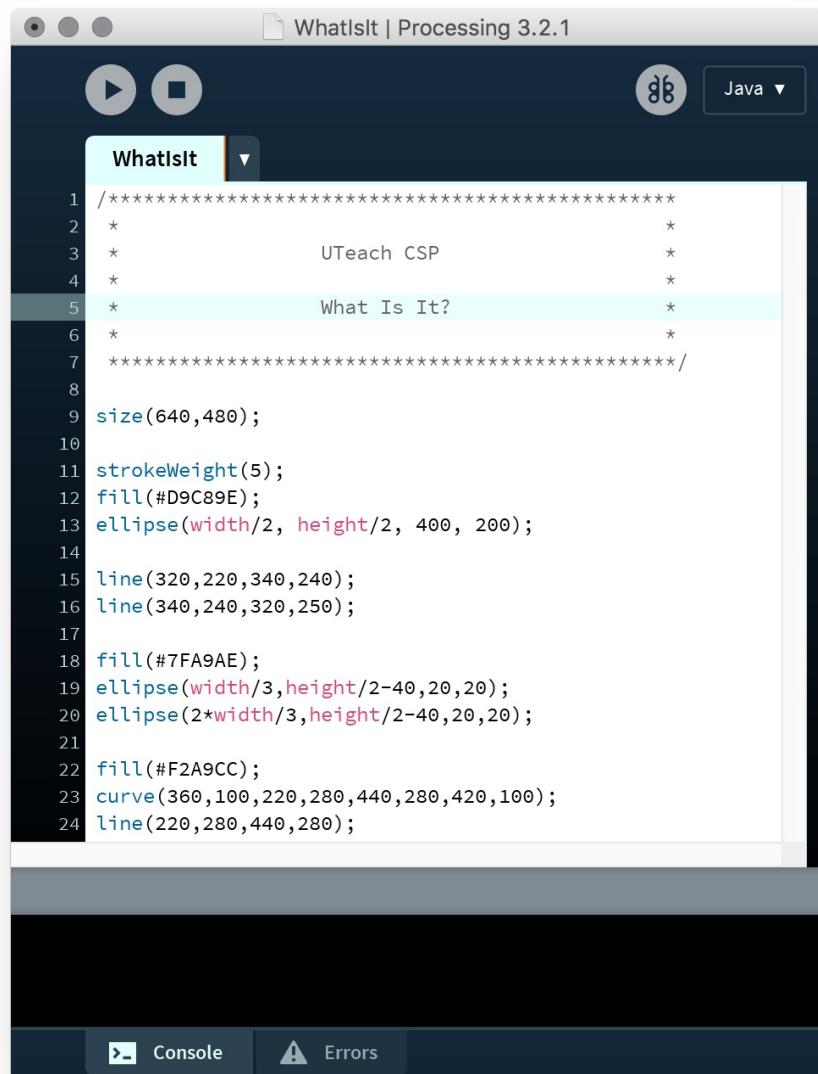
Introduction to Processing

- You will explore the capabilities of a text-based programming language (Processing).
- You will compare and contrast the programming capabilities of a visual programming language (Scratch) with those of a text-based programming language (Processing).

Writing Code

Introduction to Processing

Before we *write* any Processing programs, let's take some time to *read* one. The following is a short example of a Processing sketch that draws a picture using Processing's built-in drawing tools. Read it and try to determine *roughly* what it does *before* executing it.



```
WhatIsIt | Processing 3.2.1
Java ▾

WhatIsIt
1 ****
2 *
3 *          UTeach CSP
4 *
5 *          What Is It?
6 *
7 ****
8
9 size(640,480);
10
11 strokeWeight(5);
12 fill(#D9C89E);
13 ellipse(width/2, height/2, 400, 200);
14
15 line(320,220,340,240);
16 line(340,240,320,250);
17
18 fill(#7FA9AE);
19 ellipse(width/3,height/2-40,20,20);
20 ellipse(2*width/3,height/2-40,20,20);
21
22 fill(#F2A9CC);
23 curve(360,100,220,280,440,280,420,100);
24 line(220,280,440,280);
```

As you complete the following tasks, note that each of the commands has a real-world action that it approximates.

- Type the sketch into Processing as-is, and execute it. *Does it look anything like you imagined by examining the code?*
- Tinker with the **parameters** for each of the following:

- ellipse
- fill
- strokeWeight
- line
- curve
- size

Make *at least* 5 alterations to the sketch by manipulating these parameters.

Assignment

Record

1. an image of your results, and
2. a description of the parameters you changed to make the alterations.

Share these with a neighbor, and provide critical feedback.

Scratch vs. Processing

Comparison: Scratch vs. Processing

There are many similarities between the two languages; in fact, these similarities occur across nearly all of the programming languages in use today! Nearly all programming languages are equivalent in terms of being able to express any algorithm.

Scratch	Processing
	

To get you started seeing how familiar Processing should already be, here are few examples of how Scratch and Processing compare with one another:

- Each has buttons that **start** and **stop** the execution of a program.

Scratch	Processing
	

- Each has a way of outputting text.

Scratch	Processing
	<code>println("Hello!");</code>

- The conditional statements look similar:

1. The keywords are the same: **if** and **else**.
2. **if** comes before **else**.
3. **else** is optional.
4. There is a "hole" where the condition to decide upon is placed (e.g., parentheses).
5. The contingent blocks of statements are packaged in a structure (e.g., braces).

Scratch	Processing
	<code>if (SOMETHING)</code>

	<pre>{ DO THIS... } else { DO THAT... }</pre>
--	---

- Each offers a way to assign a value to a variable.

Scratch	Processing
	<pre>index = 0;</pre>

- Each offers a way to change the value of a variable.

Scratch	Processing
	<pre>index = index + 1;</pre>

- Each offers a way to repeat an action. Note that Scratch uses a "repeat *until*..." metaphor, while Processing uses a "repeat *while*..." metaphor. This difference means that the Boolean conditions that determine whether or not to repeat another iteration of the loop are opposite one another:

- **Scratch:** The loop continues *until* a condition becomes `true`.
 - `repeat until x < y` (i.e., x is less than y)
- **Processing:** The loop continues *while* a condition remains `true`.
 - `while x >= y` (i.e., x is *NOT* less than y)

Scratch	Processing
	<pre>while (x >= y) { DO THIS... }</pre>

Scratch Constructs Revisited

Scratch Code Blocks

Recall that Scratch organizes types of blocks according to their usage. Each category is given its own pane and block color. For example, those blocks that affect the flow of program execution (such as branching paths and sequences of events) are a gold color, as seen to the right. If you know the type of tool you require, you can easily find it through the Scratch interface. Unfortunately, text-based languages like Processing are more difficult to navigate. In order to locate the proper tool for a job, you must first learn the basic constructs available to you.

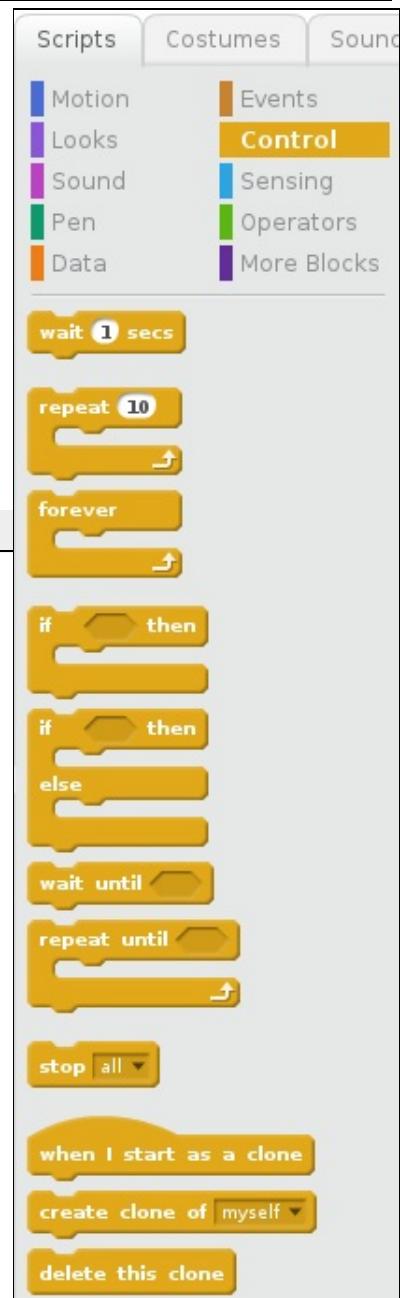
The Mechanic vs. The Surgeon

In order to illustrate this idea, imagine two people, Anna and Beatrice, who are both professionals who use tools to "fix" broken machines. Anna is an auto mechanic. She uses a variety of tools such as wrenches, screwdrivers, and probes to diagnose and correct malfunctioning engine parts. Beatrice is a surgeon. She uses stethoscopes, scalpels, and forceps to diagnose and repair tissues and organs.

Anna has her tools organized in such a way that they are easy to find, easy to reuse, and easy to select. She has a toolbox with drawers specified for each *type* of tool: fasteners, diagnostics, electrical, etc. This is similar to the way Scratch organizes its available blocks. On the upside, blocks are easy to locate and select. On the downside, all of the available tools must be easily placed in a drawer according to its purpose.

Beatrice, the surgeon, has an assistant who locates tools and hands them to her upon request. The request, "Scalpel, blade #11," is met with the correct tool. On the upside, the number of tools, variations in capabilities, and customization are more extensive than what Anna may be used to. On the downside, Beatrice must know exactly which tool she needs before she requests it. There is also much more potential for error: using the wrong tool for the job, making an incorrect request, etc.

As a programmer, it is important to have knowledge of the basic tools when using a text-based language such as Processing. It is much more difficult to "jump in" and start trying to construct a program than it is with Scratch, where all of the tools are laid out before you in



an easily comprehensible manner.

As such, and this is very important, you should spend time familiarizing yourself with the basic commands available in Processing. One approach that may help you is to use the Scratch interface to explore types of commands and then seek out their equivalents in Processing. Additionally, reading other, pre-written programs and deconstructing how they work is one of the most useful approaches you can take to be successful. In fact, this approach will carry you through any number of languages and any level of expertise.

Punctuation

Punctuation Matters? Punctuation Matters!

Punctuation is important. The English language is full of ambiguities that are often confusing when translating natural speech into text. A reader does not have the luxury of hearing the intonation, pauses, and speech fillers that usually accompany speech. Punctuation is a form of annotation that signals some of these missing cues to a reader.

To the right is a sign—hung in a mall restroom—that could use proper punctuation. Take a moment and read it aloud with no intonation or pausing as the lack of punctuation assumes.

How might you correct the poster to convey the proper message? What sort of speech cues would your suggested punctuation marks indicate?



A Short Exercise

Match the following phrases with the appropriate pictures:

Image 1	Phrases	Image 2
	Let's eat, children!	
	Let's eat children!	

English clearly requires punctuation to function properly. Otherwise, figuring out the meaning of a sentence falls to the *reader*. But computers are not *readers*, and computer programs are designed to be as unambiguous as possible, so why does punctuation matter in Processing code?

You have probably heard the phrase, "Computers are dumb." This isn't exactly true, as you are learning throughout this course, but the statement, "Computers lack intuition" *is* a true statement. In other words, any behavior a computer exhibits is either programmed explicitly or follows some set of explicitly programmed rules. Whereas a person can go back and figure out the meaning of a sentence that is clearly incorrect, a computer cannot. It does not have the experience and ability to guess correctly when there are ambiguities. Beyond that, we would not want them to because they might guess wrongly, and our programs would not work as we intended them to work.

Punctuation Symbols { ; }

The most important punctuation you will master in Processing are the braces that enclose a block of statements `{ }` and the semi-colon that terminates each statement `;`. You will make many errors learning how to place these properly and remembering to do so. A common complaint is that they don't seem necessary. However, computers, as mentioned above, are not able to just "get what you mean."

Braces `{ }`: These enclose any group of statements that need to be treated as a unit. The most common way you will use them is to replicate the way Scratch blocks "enveloped" other blocks. For example, the `if` statement in Scratch has a slot that contains the statements that are to be executed if, and only if, the conditional statement is `true`. In Processing, braces play the same role:

Scratch	Processing	Description
	<pre>if (SOMETHING) { DO THIS... } else { DO THAT... }</pre>	<p>The conditional statements look similar:</p> <ol style="list-style-type: none">1. The keywords are the same: <code>if</code> and <code>else</code>.2. <code>if</code> comes before <code>else</code>.3. <code>else</code> is optional.4. There is a "hole" where the condition to decide upon is placed, e.g., parentheses.5. The contingent blocks of statements are packaged in a structure (e.g., braces).

Semicolon `;`: Each block in Scratch was just a distinct unit. With Processing, you must piece together your own statements using keywords, variables, and punctuation. The semi-colon ends the statement. There are two major hurdles to learning its usage.

1. Not all statements utilize a semi-colon (e.g., the `if` statement references above). For the most part, though, statements do require it, and you will develop an intuition for the exceptions to the rule.
2. You would think that the Processing environment would be able to tell you where to place your missing semi-colons as part of the error message you receive when you've left one out. Processing does attempt to provide a likely location, but it is not foolproof. In fact, this is a fairly difficult problem. In order to illustrate with an English equivalent, read the following "garden path sentences." Some of them are difficult to unravel, but they are all grammatically correct!

Garden Path Sentences

- The horse raced past the barn fell.
- The florist sent the flowers was pleased.
- Time flies like an arrow; fruit flies like a banana.
- The complex houses married and single soldiers and their families.
- The man whistling tunes pianos.
- The old man the boat.
- I convinced her children are noisy.
- The tomcat curled up on the cushion seemed friendly.
- The man returned to his house was happy.
- Mary gave the child the dog bit a bandaid.
- The government plans to raise taxes were defeated.
- The girl told the story cried.

UNIT TOPIC:

Procedural Programming

Drawing

- You will write programs that make use of parameterized methods to invoke specific behaviors.
- You will understand the importance of using proper punctuation and syntax when coding in a text-based programming language.
- You will write code using common programming constructs like conditional if() for selection and while() loops for iteration.

Draw Shapes

Consult the Documentation

Knowing what Processing's built-in functions *do* and how to use them to build your application is important, but do you need to memorize all of the various components of each? Do you need to remember exactly what each of the parameters in the statement `rect(30, 20, 55, 55, 3, 6, 12, 18);` mean?



The answer to these questions is — thankfully — *no*. Processing provides a concise, but extensive, reference for each of its functions.

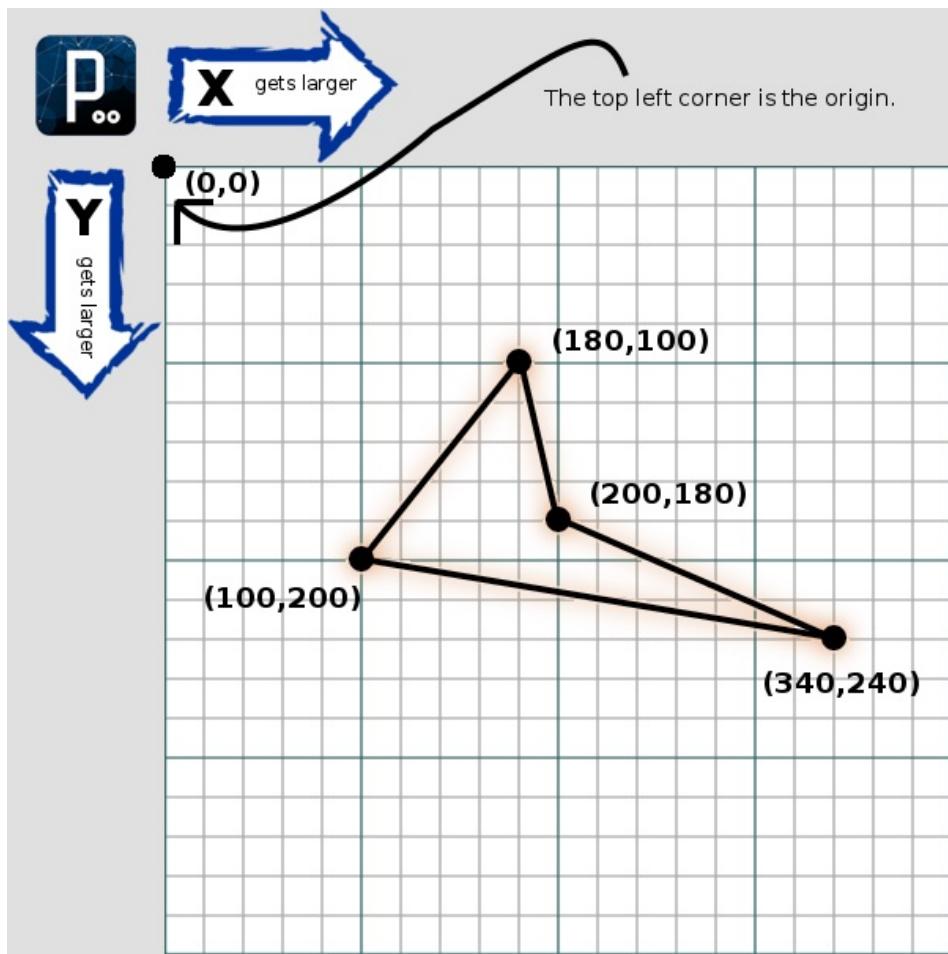
[PROCESSING DOCUMENTATION](#)

Note that the documentation is organized by types of functions. In other words, to find out how to draw a particular shape, look under the "Shapes" heading.

Instructions

Use the Processing Documentation to create one 5-line program that does each of the following. **Note that Processing orients its axes differently than those typically associated with Cartesian coordinates. See the figure below for reference.**

1. Set the size of the display window to a height of 300 and a width of 500.
2. Draw an ellipse with a width of 20 and a height of 30. It should be centered at the x-y coordinates (100, 40).
3. Create a color variable 'c' set to burnt orange. The color values for burnt orange are (**RED** = 204), (**GREEN** = 85), and (**BLUE** = 0).
4. Draw a quadrilateral that matches the following figure:

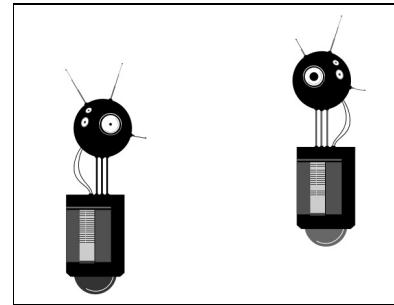


- Fill the quadrilateral with the color you defined as 'c'.

Draw a Figure

Creating the Look

Previously, you coded and modified an image of a happy face. Now, you will combine shapes in order to create a more complex original figure of your choosing (e.g., the robots to the right). Unlike your picture from [Writing Code](#), this figure will not focus only upon a static outcome (i.e., what it looks like), but instead includes other considerations, such as how the figure will behave or how we might interact with it. What is the distinction? In terms of Scratch, the distinction between a background image and a sprite would be similar: there is much more associated with a sprite than simply its static appearance.



For this assignment, however, we will focus solely on the appearance of your original figure. While designing the figure, though, think about possible future behaviors as well.

Instructions

Using vector shape-drawing commands in Processing, code a sketchbook that draws a figure. Examples of good figures might include a robot, person, cat, kangaroo, automobile, etc.

Submission

You must submit the .pde of the Processing sketchbook that draws your original figure.

Tips

- Read an example piece of code containing a drawing. Outline the steps in the example. Think about how can you apply these skills to your own drawing.
- As always, examine the [Processing Reference Page](#). Focus on 2D Shapes, particularly the `ellipse()`, `line()`, `point()`, `rect()`, and `triangle()` functions to create a figure.
- Remember that the window size is stored in `width` and `height` variables. If you'd like to reference the exact center of the window, use something akin to `point(width/2, height/2)`.
- The `println()` function can be a lifesaver when troubleshooting your code. Use it to output diagnostic messages as needed. Reference the [Debugging with println\(\) Guide](#).

Reference Starting Points

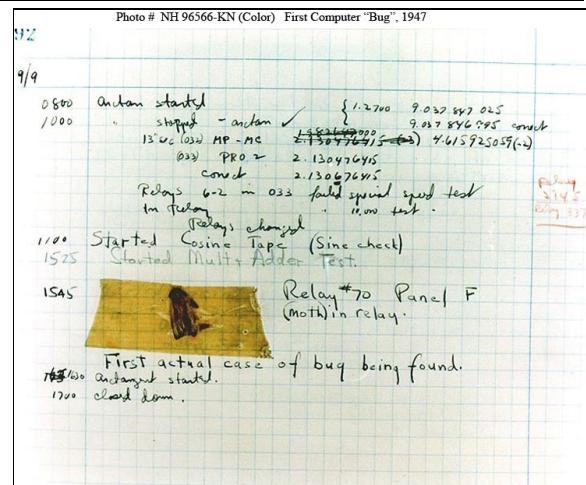
Functions	Variables
-----------	-----------

<u>ellipse()</u>	<u>width</u>
<u>line()</u>	<u>height</u>
<u>point()</u>	
<u>rect()</u>	
<u>triangle()</u>	
<u>println()</u>	

Debugging with `println()`

Finding Bugs

In programming, any unwanted or unintended property of a program or piece of hardware, especially one that causes it to malfunction, is referred to as a *bug*. The term originated in 1945 when Admiral Grace Hopper traced a malfunction with the Harvard Mark II computer to an actual bug (specifically, a [moth](#)) caught in one of the electronic relays. She taped the moth into her log book and noted it as the "First actual case of bug being found." The name stuck and ever since, programmers have been plagued (no pun intended) with bugs.



Finding software bugs in your programs is hard. It may be the most difficult part of programming. Of course, the simplest way to prevent bugs is to not have any in the first place! Planning the logic and anticipated input/output of your code before actually typing it into the interpreter is important. This has spawned the humorous adage:

Weeks of programming can save you hours of planning.

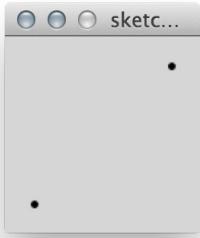
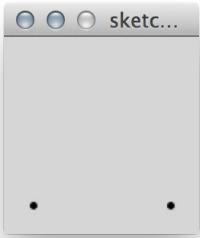
Of course, programs are written by people, and people make mistakes. Even the best programmers are not immune to buggy code. Computer scientists have developed a multitude of methods for finding errors in programs:

1. Examples of intended behavior on specific inputs help people understand what a program is supposed to do. Programmers can add comments within the program to describe *what should happen* based on specified inputs.
2. Programs should be kept as short and concise (while doing what it is intended to do) as possible. Duplicated code and longer code segments can make it harder to reason about a program's functionality.
3. Comments should be used effectively so a programmer can explain how a piece of code is used within the program. Comments can also help justify and explain a program's correctness.
4. The simplest—and the one you should concentrate on using—is called "[print debugging](#)."

Printing Variables

Consider the following code that exchanges the `x` and `y` values of a point:

```
size(100 , 100);
strokeWeight(5);
stroke(0);
int x = 5;
int y = 95;
point(x , y);
x = y;
y = x;
point(x , y);
```

Expected Output	Actual Output
	
2 points at opposite corners	2 points at adjacent corners

Clearly the variables `x` and `y` are not what we expect them to be. So, what are they?

Adding `println(x);` after each point in the program where we make a change will tell us just that (similarly for `y`):

```
size(100 , 100);
strokeWeight(5);
stroke(0);
int x = 5;
int y = 95;
point(x , y);
x = y;
println(x);    // for debugging
y = x;
println(y);    // for debugging
point(x , y);
```

This prints the following to the output window — not quite the values that we expected.

Where have we seen this error before?

```
95
95
```

Printing Checkpoints

Another type of error occurs when your program crashes and you cannot easily determine where. To figure out where the problem is, you can add `println()` statements to your

code at various points:

```
// ...a bunch of stuff...
println("You reached point A");
// ...some more stuff...
println("You reached point B");
// ...just a little bit more stuff...
println("You reached point C");
// ...this is the last bit of stuff I promise...
```

Imagine that when the above program is run, it prints the following before crashing:

```
You reached point A
You reached point B
```

Clearly the problem lies in the area `...just a little bit more stuff...`. The program execution never made it through these statements to print `You reached point C`.

UNIT TOPIC:

Procedural Programming

Mouse Interaction

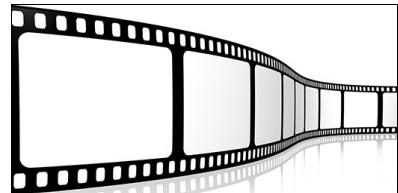
- You will use event handlers to animate on-screen effects and respond to mouse input.

Movement

The Illusion of Movement

As alluded to in the previous assignment, we are going to add some behaviors to our figures. Specifically, we will add movement to our figures so that they can respond to user input.

Unlike with Scratch, Processing does not include pre-built blocks that move sprites around the screen. Instead, much like with cinematic film or old-fashioned flipbooks, we must redraw each scene anew every time we wish to alter it. Animation and movement are perceptual illusions created through rapid succession of nearly identical images. (cf., [the phi phenomenon](#)).



Before animating our figures from [Draw a Figure](#), let's experiment with a simple example to explore how Processing supports this effect.

A Circle that Moves

Let's begin by drawing a circle:

```
ellipse(50, 50, 80, 80);
```

Note that there is no 'circle' function. Of course, a circle is just a special form of an ellipse, much in the same way that a square is a special form of a rectangle. This line of code means "draw an ellipse, with the center 50 pixels over from the left and 50 pixels down from the top, with a width and height of 80 pixels."

All of your Processing applications to this point have followed this format. The interpreter starts at the top of the source and executes each function or command in sequence as it moves down the list of instructions.

However, most basic programs in Processing follow a slightly more complex model, and this provides the framework for dynamic programs. In order to animate objects on the screen, we need to redraw the screen quickly while re-orienting the objects as they move (much as in film—as discussed earlier).

Processing provides this framework through the `setup()` and `draw()` functions. Up to now, you have used Processing functions to draw ellipses and rectangles or change colors. Processing includes these pre-written functions, and you, as the programmer, determine when and where they are used.

In this assignment, you will do something completely different. In fact, you and Processing will switch roles in a way. Processing will execute the `setup()` and `draw()` functions automatically to start the program and constantly redraw the screen. You will define what

each of these functions actually does when Processing uses them.

Exercise

Type in, and execute, the following program:

```
void setup()
{
    size(480, 240);
}

void draw()
{
    if (mousePressed)
    {
        fill(0);
    }
    else
    {
        fill(255);
    }
    ellipse(mouseX, mouseY, 80, 80);
}
```

1) What does it do?

Answer:



The `draw()` function dictates how to redraw the screen. Processing defaults to redrawing the screen 60 times per second. This can be altered using the `frameRate()` function.

Insert the code `frameRate(10);` into your `setup()` function.

2) How does this change your program's behavior?

Answer:



3) Why should we insert the `frameRate()` command into `setup()` rather than `draw()`?

Answer:



Swap the order of the parameters so that the `ellipse()` command reads `ellipse(80, 80, mouseX, mouseY)`.

4) What does this do, and why?

Answer:



Insert the `background(200);` instruction at the very beginning in your `draw()` function block to have Processing redraw the background each time it refreshes the screen.

5) How does this affect the behavior of the program? Why?

Answer:



Additional Tutorials

The previous activity is further detailed in the official Processing tutorials. If you'd like more information about the Processing environment than outlined above, visit [Processing Tutorial: Getting Started](#).

Animate Your Figure

Bringing Your Figure to Life

Using your figure from [Draw a Figure](#), add the following functionality:

1. Alter it to use the `setup()` and `draw()` routines.
2. Make it responsive to mouse movement and button presses. Be creative! Perhaps pressing the mouse button causes your figure to blink or open its mouth.

You may use the strategies outlined in the [Movement](#) exercise.

Submission

Submit the `.pde` of your Processing sketchbook.

Reference Starting Points

Functions	Variables
<code>setup()</code>	<code>mousePressed</code>
<code>draw()</code>	<code>mouseX</code>
<code>background()</code>	<code>mouseY</code>
	<code>pmouseX</code>
	<code>pmouseY</code>

UNIT TOPIC:

Procedural Programming

Keyboard Interaction

- You will use event handlers to animate on-screen effects and respond to keyboard input.

Keyboard Input

Adding More Controls

Processing is not restricted to mouse movement for input and output. In this activity, you will extend your figure from [Animate Your Figure](#) to incorporate keyboard button presses as well. In order to do so, you will use a variety of pre-built variables for keyboard input similar to the `mousePressed` variable.

As usual, you should read and reference the [Processing Documentation](#) in order to understand the use of these constructs. However, here is a brief outline of how the keyboard input system in Processing works.

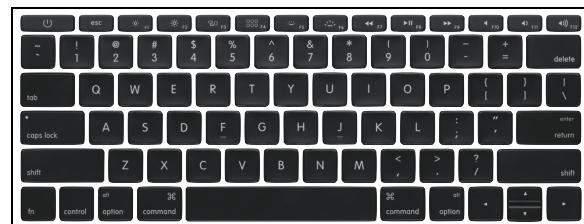
Keyboard Input

If a key is pressed, the built-in variable

`keyPressed` is set to `TRUE`. This means that a simple test, such as `if (keyPressed)`, will allow the execution of a block of code if (and only if) a key has been pressed. Another

variable, named `key`, can be used to check the contents of the key that has been pressed. In other words, `if (key == 'A')` will check the pressed key against the capital letter A.

The `key` variable may indicate a key has been pressed that is not an ASCII encoded character. In this case, `if (key == CODED)` will evaluate to `TRUE`. Should this occur, the value of the key is stored in the variable `keyCode`. For example, the following code checks to see if a key has been pressed, then if it is a non-ASCII encoded key, and finally if it is the "up arrow" key:



```
if (keyPressed)
{
    if (key == CODED)
    {
        if (keyCode == UP)
        {
            // up arrow key stuff happens
        }
        else
        {
            // other stuff happens
        }
    }
}
```

Instructions

Modify your sketchbook from [Animate Your Figure](#) as follows:

1. Add the ability to respond to key press commands.
2. Include the use of *at least 5 key strokes*, including a mix of alphanumeric characters and the arrow keys.

Much like the mouse button press, each of these key presses should alter the appearance or behavior of the figure in some unique way. Remember to reference the [Processing Documentation](#) for examples.

Submission

Submit the `.pde` of your Processing sketchbook.

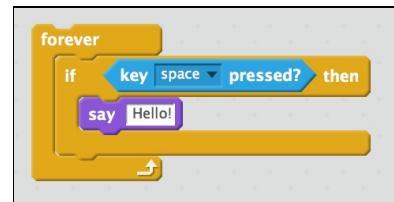
Reference Starting Points

Variables	Control Structures
<code>keyPressed</code>	<code>if</code>
<code>key</code>	<code>else</code>
<code>keyCode</code>	<code>while</code>

Loops

Forever Listening for Keystrokes

The construct to the right can be used to enable a sprite to say "Hello!" anytime the space bar is pressed.



However, the Scratch code contains a statement/block not found in Processing—the **forever** block. You may recall that placing the code that checks for a key press in a loop is necessary so that the program will continually check for key presses rather than doing it once and moving on. So, what is the equivalent for this in Processing?

Processing has the same types of control statements as Scratch that allow for looping. In particular, we will examine the **while** loop. **However, Processing also has an implicit looping structure.** You may recall that the **draw()** function is executed every time the screen is redrawn—defaulting to 60 times per second. This means that any code in the **draw()** function—including key press checks—will be executed continually as long as the program is running! For this reason, it is a good idea to reserve a section in **draw()** precisely for this purpose.

As usual, you should **read and reference the [Processing Documentation](#)** in order to understand the use of these constructs.

The **while** Loop

The **while** loop is a very useful construct for repetition and is composed of 4 major elements:

1. Initialization
2. Condition
3. Action
4. Update

To see a **while** loop in context, let's imagine we are drawing the buttons on a snowman's torso. Each of the buttons is essentially the same, so this is a perfect task for a loop.

```
1 size(400,600);
2
3 noStroke();
4 ellipse(200,160,80,80); //head
5 ellipse(200,260,160,160); //torso
6 ellipse(200,420,240,240); //bottom
7
8 fill(0); //buttons are black
9
10
11 int numberOfButtonsDrawn = 0; ←Initialization
12 while (numberOfButtonsDrawn < 5) ←Condition
13 {
14     ellipse(200,(210+20*numberOfButtonsDrawn),10,10); ←Action
15     numberOfButtonsDrawn = numberOfButtonsDrawn + 1; ←Update
16 }
17
18
19
20
21
22
23
```

The red arrows in the above image show the relative positions for each of the four parts of the `while` loop:

Initialization: Here, we are keeping track of the number of buttons we have drawn. As we have not drawn any before the loop begins, this variable is set to zero.

Condition: We are going to draw 5 buttons. So, as long as we have not (i.e., `while`) drawn 5 buttons, we will keep executing the loop.

Action: This is where we draw a button. Notice that we do not draw 5 completely identical buttons, because that would mean that we draw them on top of one another. Instead, we vary the *y-position*, so that the first button begins at *y-coordinate* 210, and each remaining button is drawn 20 pixels below the previous.

Update: Lastly, it is important that we indicate that we have drawn a new button each time by updating the value of `numberOfButtonsDrawn`. *What would happen if we did not?*

Instructions

Modify your sketchbook from [Keyboard Input](#) as follows:

1. Add a `while` loop that adds various repeated components to your figure.
2. The loop should execute a minimum of 3 times.

You may use the snowman code above as an example.

Reference Starting Points

Variables	Control Structures
<code>keyPressed</code>	<code>if</code>
<code>key</code>	<code>else</code>
<code>keyCode</code>	<code>while</code>

UNIT TOPIC:

Image Manipulation

RGB Color

- You will examine the structure of raster images as compositions of individual pixels.
- You will explore various methods of representing color, including RGB, CMYK, and HSV.
- You will explore the various colors that can be produced by the combination of different ratios of red, green, and blue light.

Raster Images

- You will modify the color channels of pixels in an image to produce a variety of effects.

Raster Image Manipulation

- You will design algorithms for modifying the pixels in an image in prescribed ways to create custom image filters.

Encoding Schemes

- You will explore the difference between lossy and lossless encoding schemes of several common image file formats.

UNIT TOPIC:

Image Manipulation

RGB Color

- You will examine the structure of raster images as compositions of individual pixels.
- You will explore various methods of representing color, including RGB, CMYK, and HSV.
- You will explore the various colors that can be produced by the combination of different ratios of red, green, and blue light.

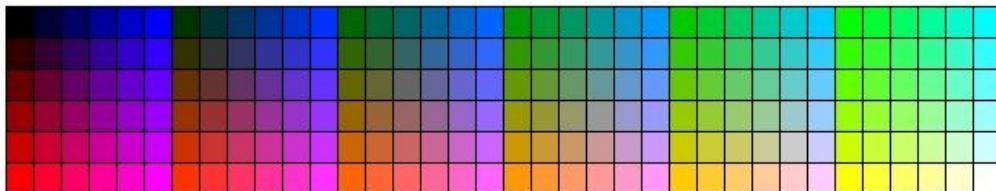
Calculating Colors

Red, Green, and Blue

Bits are just bits—**1s** and **0s**. Without instructions for encoding or abstracting the bits, they have no meaning. Think about this: What would happen if an image encoded in RGB format were read as BGR (blue-green-red) format instead? Assuming that all other aspects are the same, blue and red in the image would be reversed, resulting in some alternate color schemes for the image. Although this is a trivial example, it is important to realize that these standards for encoding give meaning to the sea of bits floating around your computer and the Internet.

Instructions

In this experiment, you will manipulate colors by altering their binary representations with this [Color Calculator](#) tool.



Exercise

- 1) Enter the following values as RGB values into the [Color Calculator](#). What color is generated?

Red	Green	Blue
01110111	11001101	00011110

- A. Purple
- B. Green
- C. Pink
- D. Brown

Answer:

- 2) Notice that the largest value is the **Green** sequence. How does that affect the color?

You may experiment with the [Color Calculator](#).

Red	Green	Blue
01110111	11001101	00011110

- A. It makes the color green.
- B. It makes green the most dominant tint to the color.
- C. Red and blue dominate the green.
- D. It makes it less green; higher numbers are less intense.

Answer:

-
- 3) Zero out the **Red** and **Blue** sequences. How does that affect the color? Select two answers.

You may experiment with the [Color Calculator](#).

Red	Green	Blue
00000000	11001101	00000000

- A. The color gets brighter.
- B. The color gets darker.
- C. The color get more green.
- D. Nothing changes.

Answer:

-
- 4) Copy the **Green** value to the **Blue** value. Leave the **Red** at zero. What color best describes the result?

You may experiment with the [Color Calculator](#).

Red	Green	Blue
00000000	11001101	11001101

- A. Yellow
- B. Pink
- C. Orange
- D. Teal

Answer:

-
- 5) Last, copy the **Green** value to **Red**. They should all be identical now. Note the effect that

has on the color. Why do you think that happens?

You may experiment with the [Color Calculator](#).

Red	Green	Blue
11001101	11001101	11001101

Answer:



Hexadecimal

Lingua Franca

We have discussed two commonly used bases for encoding numeric information—base 2 (binary) and base 10 (decimal). Each of these serves as a “native encoding”—binary for digital devices and decimal for humans.

- Binary is convenient notation for computers because digital circuits are essentially a collection of on/off switches. Remember that the underlying concept of binary notation is the representation of information as a series of dichotomies. The dichotomies may be represented in a number of ways—including **on/off** or **0/1**.
- Decimal is convenient notation for modern humans. Most languages use decimal as the basis for numeral representation. Many have proposed that base 10 is used because people learn to count using their 10 fingers. There may be truth to this. The Yuki tribe in California counted with *the spaces between their fingers*, and because of this, they used base 8 to represent numbers!

Computers use binary by necessity because it best complements the underlying hardware. Why don’t humans use binary? Perhaps it is because the length of binary numerals can grow quite large quickly. The 3 digits of **999** in decimal are equivalent to 8 digits in binary—**11100111**. Beyond this, the use of only two symbols may make numerals hard to read. Which pair is more distinct—**999** and **723**—or **11100111** and **11010011**?

Because of this, computer scientists use a third base to serve as a *lingua franca*—a language that is adopted as a common language between speakers whose native languages are different.

Binary Abbreviated

Base 16—hexadecimal—is commonly used as this lingua franca. Why base 16?

Consider the following table:

Binary†	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6

0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

[†]Note that the numerals are padded to the left with zeroes so that the binary column has the same number of digits. This does not affect the values. Why?

Using the 16 symbols of hexadecimal, you can represent all the possible permutations of 4 bits! Note that the symbols A–F are used to extend the numerals representing each digit beyond those of decimal—A is equivalent to 10, and F is equivalent to 15. Hexadecimal serves as an abbreviated form of binary!

Hexadecimal is common when representing RGB colors. RGB uses 24 bits—8 for the red channel, 8 for the green channel, and 8 for the blue channel. So, the following color:



is represented as **001101011001110010001111** in binary. This is difficult to read.

However, splitting the numerals into groups of 4, **0011 0101 1001 1100 1000 1111**, we can easily convert this long string of **0s** and **1s** into **359C8F** in hexadecimal:

```
0011 0101 1001 1100 1000 1111
0011 0101 1001 1100 1000 F
0011 0101 1001 1100 8 F
0011 0101 1001 C 8 F
0011 0101 9 C 8 F
0011 5 9 C 8 F
3 5 9 C 8 F
```

The algorithm to convert binary to hexadecimal is as follows:

1. Starting at the right most digit of the binary number, isolate 4 digits.
2. Replace the 4 digits with their equivalent hexadecimal numeral.
3. Repeat steps 1 and 2, moving leftward[†].

[†]If the final sequence of bits numbers less than 4, insert **0s** to the left until the sequence is 4

digits long.

Exercises

Convert the follow from binary to hexadecimal:

1) $11001010_2 =$

Answer:

2) $1010101001_2 =$

Answer:

3) $10101110001111010111010_2 =$

Answer:

4) $111111011011010000101011_2 =$

Answer:

5) $10101011100110111101111_2 =$

Answer:

Try using the table in reverse! Convert from hexadecimal to binary:

6) $\boxed{14}_{16} =$

Answer:

7) $\boxed{C4}_{16} =$

Answer:

8) $\boxed{387}_{16} =$

Answer:

9) $\boxed{FFFF}_{16} =$

Answer:

10) **B7F5CA**₁₆ =

Answer:

Color

Color Channels

As previously indicated, computer screens use a combination of red, green, and blue to produce all of the displayable colors. Similarly, in Processing, manipulating colors is merely a matter of specifying the amounts of **red**, **green**, and **blue** an object's color contains.

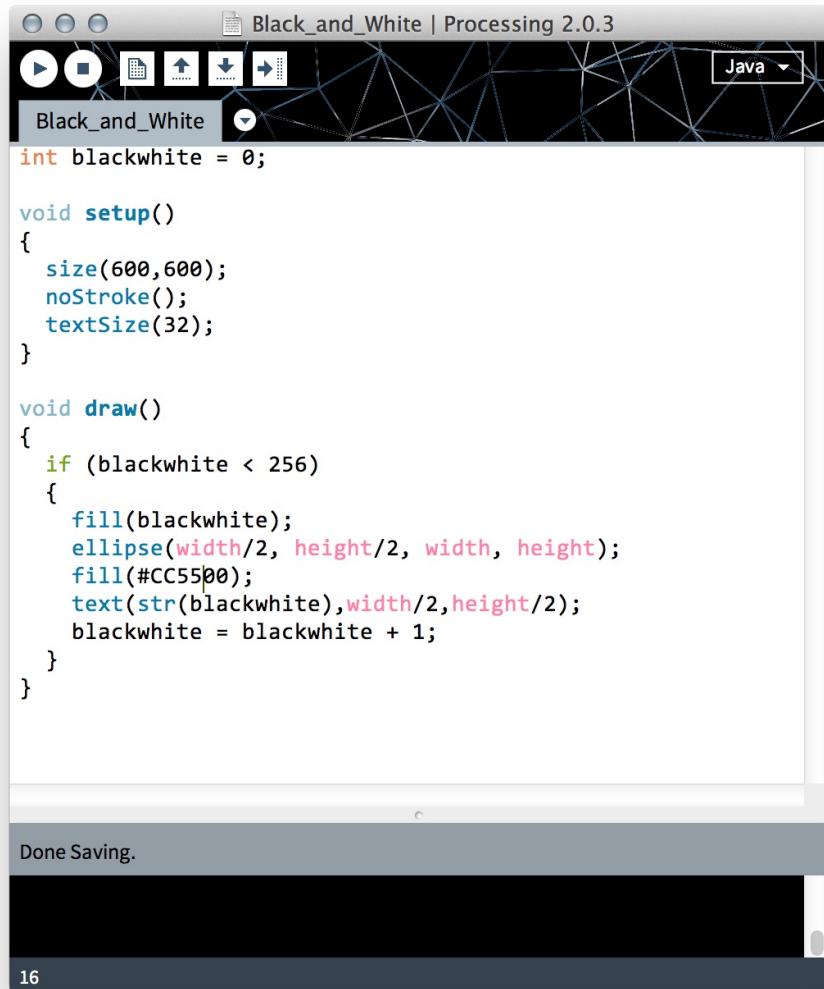
To illustrate how scaling these color amounts (otherwise known as "channels") works, we shall start with a simpler example—*grayscale*.

Technically, a black and white image comprises two colors, black and white. These could be represented via 1 bit (e.g., 0 for black, 1 for white). However, in Processing (and life), there is more to the spectrum than black or white. Processing allocates 8 bits for specifying shades of gray. This means that there are 28, or 256, different shades of gray possible.

The boundaries for black and white in the spectrum are at opposite ends, just as they were in the 1 bit case. This means that the instruction `stroke(0);` would set the pen to produce black lines and the instruction `stroke(255);` would set the pen to produce white lines.

To demonstrate how the shades of gray vary between these two values, execute the following program:

Click to Download: [BlackAndWhite](#)



If you would like to see a slower gradation of grayscale, you can insert a `frameRate(10);` line in your `setup()` function.

From Grayscale to Color

Although color seems much more complex than shades of gray, in reality, it is just a "triplication" of the same idea.



<https://www.youtube.com/embed/x6D8PAGeIN8>

To illustrate how each of the red, green, and blue color channels works in the same way as grayscale, alter the `fill(blackwhite);` line to read `fill(blackwhite, 0, 0);`. How does this change the behavior of your program? Try `fill(0, blackwhite, 0);` and `fill(0, 0, blackwhite);`. *What do they produce?*

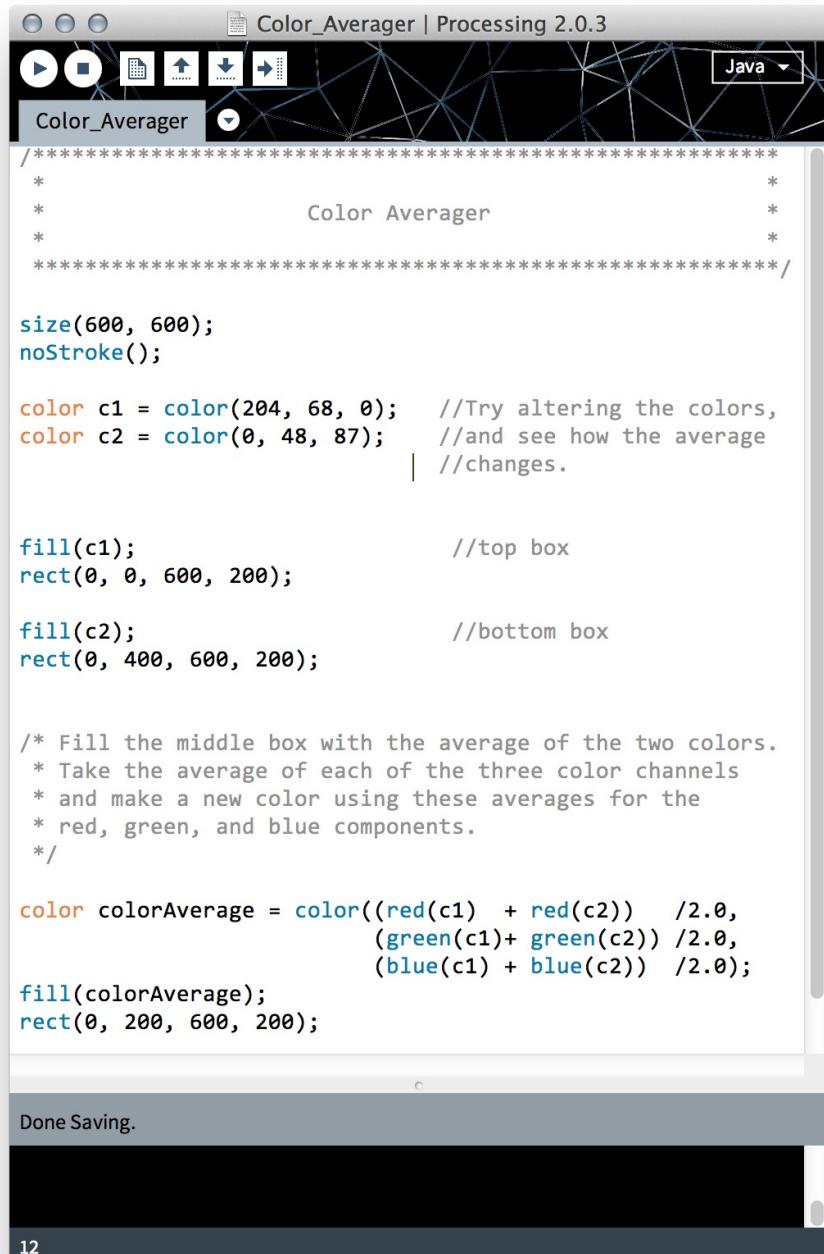
Finally, try `fill(blackwhite, blackwhite, blackwhite);`. What does this do? Why?

Color Variables

Processing allows you to store colors as variables as well. Once you store a color, you can access its red, green, and blue components separately.

To illustrate this, execute the follow program that "averages" two colors:

Click to Download: [ColorAverager](#)



```

Color_Averager | Processing 2.0.3
Java

/*
 *          Color Averager
 *
 ****
size(600, 600);
noStroke();

color c1 = color(204, 68, 0); //Try altering the colors,
color c2 = color(0, 48, 87); //and see how the average
                           //changes.

fill(c1);                      //top box
rect(0, 0, 600, 200);

fill(c2);                      //bottom box
rect(0, 400, 600, 200);

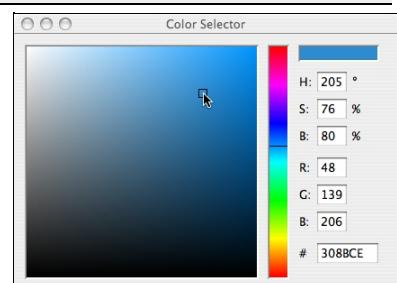
/* Fill the middle box with the average of the two colors.
 * Take the average of each of the three color channels
 * and make a new color using these averages for the
 * red, green, and blue components.
 */
color colorAverage = color((red(c1) + red(c2)) / 2.0,
                           (green(c1)+ green(c2)) /2.0,
                           (blue(c1) + blue(c2)) /2.0);
fill(colorAverage);
rect(0, 200, 600, 200);

Done Saving.

```

Instructions

1. Use the RGB color model to color your figure from [Loops](#). Although you can use an online [Color Calculator](#) like the one presented in the previous activity, Processing provides its own color selector tool as well. Access it in the *Tools* entry on the menu bar.
2. Add at least one additional keystroke that changes some color in your figure. *For example, perhaps pressing 'd' darkens the colors in your figure (makes the color channels have lower values—less intensity), and pressing 'l'*



lightens them (makes the color channels have higher values—more intensity).

3. Finally, note that you can alter the colors passed as parameters to the `stroke()`, `fill()`, and `background()` functions. Use each of these at least once in your coloring scheme.

Submission

Submit the `.pde` of your Processing program.

Reference Starting Points

Functions
<code>color()</code>
<code>stroke()</code>
<code>fill()</code>
<code>background()</code>

UNIT TOPIC:

Image Manipulation

Raster Images

- You will modify the color channels of pixels in an image to produce a variety of effects.

Raster Images

Raster Image Processing

Download the code below, and execute it. Examine the script, focusing on each of the component parts.

Click to Download: [RasterStarter](#)

Basically, it functions as follows.

For each pixel of the image:

1. Each of the three color channels are extracted and stored in variables named `r`, `g`, and `b`.
2. A new color value is created using these very same `r`, `g`, and `b` values.
3. The old color value for the pixel is replaced by the newly created color.

The resulting image appears exactly the same as the original image. But what if the new color value created in Step 2 were to use altered values for `r`, `g`, and/or `b`? How might that affect the appearance of the resulting image?

Strategies for Manipulating Raster Images

To practice using Processing to manipulate pixels in an image — as well as RGB and arrays — work through the following tasks. By manipulating the program, you can manipulate the image representation programmatically. **Try each of the following strategies.** This list is by no means exhaustive, but it should give you a good foundation for exploration. **It is important to spend time working through each of these tasks.**

1. Use one of the colors (`r`, `g`, or `b`) to define all three of the color channels in the new color.

Example: `color newColor = color(g, g, g);`

What is the result? Why?

2. Keeping one of the values the same, set the other two to zero.

Example: `color newColor = color(0, g, 0);`

3. Swap the colors with one another. In other words, you might use `r` for the **blue** channel, `b` for the **green** channel, etc.
4. Divide each of the color values by 2.

5. Now, try multiplying one of the color values by 2.
6. As mentioned previously, Processing uses 8 bits for each color channel, giving a usable range of 0–255 for the separate **red**, **green**, and **blue** channels. What do you think would happen if a value of 0 in the old image were mapped to a value of 255 in the new—and *vice versa*? Try it. Replace each of color channels **r**, **g**, and **b** with their complements, (i.e. **(255 - r)**, **(255 - g)**, and **(255 - b)**). *What happens? Why?*

Pixel Order

Manipulating each pixel achieves interesting effects, but we can also work with the entire array of pixels as well. Try these two simple examples:

1. Instead of updating the color each time a new pixel is loaded, update the color only once every 20 times a pixel is loaded. HINT: Change the **r**, **g**, and **b** variables only when **location** is a multiple of 20.

Example:

```
if (location % 20 == 0)
{
    update the variables
}
```

2. The current loop begins at 0 and goes all the way through the array of **pixels** in order. The last pixel location is **pixels.length - 1**. Replace the line storing the pixel color at **location** with:

```
pixels[pixels.length - 1 - location] = newColor;
```

Does the result surprise you? If it does, review the [Swaps](#) assignment from Unit 3: Data Representation.

Eliminating Digital Noise

Noisy Images

Each of the following is a Parlante image puzzle[†] — an image that has been obscured with digital noise. You must recover the original image by attenuating this noise as outlined below.

Instructions

1. Load each of the images below into [RasterStarter](#).
2. Modify the RGB values according to the instructions provided for each puzzle.
3. Identify the mysterious hidden images!

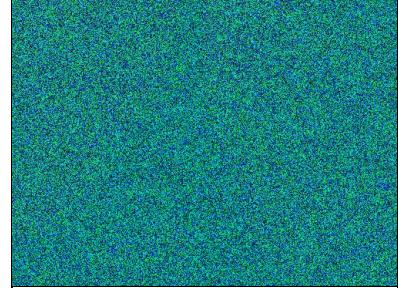
To access each image, right-click and copy the image address to paste into the provided [loadImage\(\)](#) function.

Submission

Type a description of each image (from the Iron, Copper, and West puzzles) into the provided text box for credit.

Iron Puzzle

The [iron-puzzle.png](#) image is a puzzle; it contains an image of something famous, however the image has been distorted. The famous object is in the red values; however, the red values have all been divided by 10, so they are too small by a factor of 10. The blue and green values are all just meaningless random values ("noise") added to obscure the real image. You must undo these distortions to reveal the real image.



First, set all the blue and green values to 0 to get them out of the way. Look at the result... if you look very carefully, you may see the real image, although it is very, very dark (way down towards 0). Then multiply each red value by 10, scaling it back up to approximately its proper value. What is the famous object?

Answer:

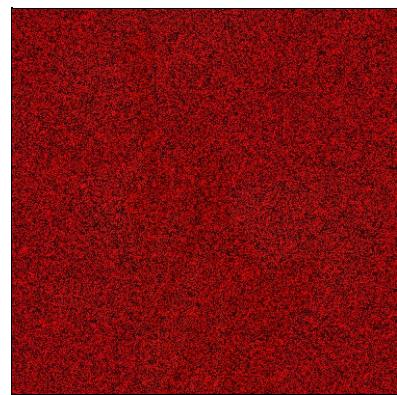


Copper Puzzle

The [copper-puzzle.png](#) image is a puzzle—it shows something famous, but the image has been distorted. While the true image is in the blue and green values, all the blue and

green values have all been divided by 20, so the values are very small. The red values are all just random numbers, noise added on top to obscure things. Undo these distortions to reveal the true image.

First, set the red values to 0 to get that out of the way. You may be able to see the image very faintly at this point, but it is very dark. Then multiply the blue and green values by 20 to get them back approximately to their proper values. What is the famous object?

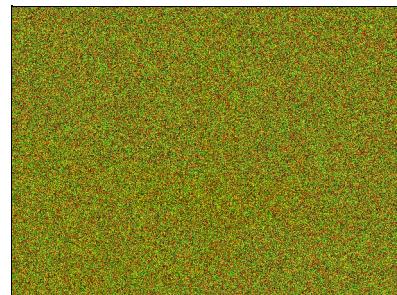


Answer:



West Puzzle

The [west-puzzle.png](#) image is a puzzle. It shows something famous, however the image has been distorted. Use if-logic along with other pixel techniques to recover the true image. The true image is exclusively in the blue values, so set all red and green values to 0. The hidden image is encoded using only the blue values that are less than 16 (that is, 0 through 15). If a blue value is less than 16, multiply it by 16 to scale it up to its proper value. Alternatively, if a blue value in the encoded image is 16 or more, it is random garbage and should be ignored (interpreted as 0). This should yield the recovered image, but all in the blue channel. As a final fix, the image should be in the red channel to look more correct, so change your code to move the values from the blue channel to the red channel.



Answer:



[†]This activity is adapted from [Nick Parlante, Nifty Assignments, 2011](#).

UNIT TOPIC:

Image Manipulation

Raster Image Manipulation

- You will design algorithms for modifying the pixels in an image in prescribed ways to create custom image filters.

Raster Image Manipulation

Assignment

Time to get your hands dirty and begin manipulating bits!

Instructions

1. Download the [RasterStarter](#) code, and execute it.
2. Load and modify an image for your choosing. Use [Photopin.com](#), an online aggregator of digital images with Creative Commons licenses, to find an image.
3. You may use any or all of these [strategies for manipulating Raster Images](#) or create your own using a combination of any of the ideas presented.
4. Be creative!

Submission

You must submit the following items:

1. A link to the original image
2. A [screenshot](#) of your modified image
3. The Processing sketchbook you created to produce it

UNIT TOPIC:

Image Manipulation

Encoding Schemes

- You will explore the difference between lossy and lossless encoding schemes of several common image file formats.

Digital Image File Extensions

Lossy vs. Lossless Formats

Your task is to complete a table that organizes the following information about various digital image file types. Helpful resources to begin with are [Image Formats: What's the Difference Between JPG, GIF, PNG?](#) and [Digital Image File Types Explained](#).

Begin with this blank table:

	Lossy or Lossless?	Advantages	Disadvantages	Ideal for...
JPG				
GIF				
SVG				
PNG				
BMP				
TIFF				

Encoding Schemes

Abstraction in Imagery

Have you ever heard the colloquialism, "There's more than one way to skin a cat"? Taken literally, the sentence is somewhat morbid, but figuratively, it means that there is often more than one way to accomplish a task, whether that be skinning a cat, coding a program, or representing an image digitally.

Common misconception: *Images have to be stored pixel by pixel.*

- Abstraction allows for virtually any representation as long as a program knows how to interpret it and convert it into some visual artifact.

Look at the image below, and consider how you would describe it and how a computer would "describe" (i.e., encode) it:



You would probably describe aspects like:

- repeated patterns
- names of objects
- color (RGB, wavelength, intensity)
- size (length, width, height)
- shape
- location (coordinates)
- brightness (greyscale, darkness, power allocation)
- texture

These are all characteristics that we notice visually, and they are therefore factors that we need to consider when we represent images digitally, so that the computer can accurately interpret the abstraction and display the image as we intend.

Image Type Matters

Digital photos tend to be color-rich, digital diagrams tend to be shape-dependent, and digital drawings rely on a great deal of contrast/brightness. Encoding schemes must account for and suit the factors that impact different image types more heavily than others. "There is more than one way to skin a cat," but sometimes one way is better than another! Consider the following factors that impact how an image is to be used:

- information - *What kind of image is it? Is it a photograph, a drawing, black & white?*
- size - *How big is it?*
- use - *What is the image used for? Is it used for a fax machine, digital photography, a diagram?*
- scalability - *Does the image need to be scaled? Does it need to be reduced, enlarged, rotated?*

Each of these factors determine which encoding schemes are best suited to represent an image. For example, let's consider a black & white diagram of an arrow, like this one:



Welcome to the Picasa Photo Viewer

In this case, the color is not as important as the shape, location, and direction of the arrow, which provide meaning to the diagram. A **vector** encoding would be much more suitable for this image than a **raster** representation, because shape, location, and direction information can be expressed more concisely and require less data.

Certain encoding schemes are better than others for representing shape, location, and direction. On the other hand, other encoding schemes are more effective for encoding an array of colors than they are for encoding shape, location, or direction.

When you choose the encoding scheme for a digital image representation, consider the most important aspects of the image and determine what type of image formats would work best for you to realize them.

Just as text can be represented with binary by conventional mappings, so can media (images, video, audio, etc.). Unlike the fairly standardized encoding of text, images are represented in many different ways.

Line Contour Drawing

Consider "line contour drawing," where the artist puts the pencil on the paper and traces out an outline of an object *without picking up the pencil*. How might an image be encoded to imitate this style of rendering? Direction and extension are the important components here, whereas location is always relative to the starting point, rather than absolute.

Example list of instructions:

- Start at position (x,y).
- Move 10 pixels in direction 132°.
- Move 8 pixels in direction 278°.
- Stop.



<https://www.youtube.com/embed/p7Mm-hOT0iM>

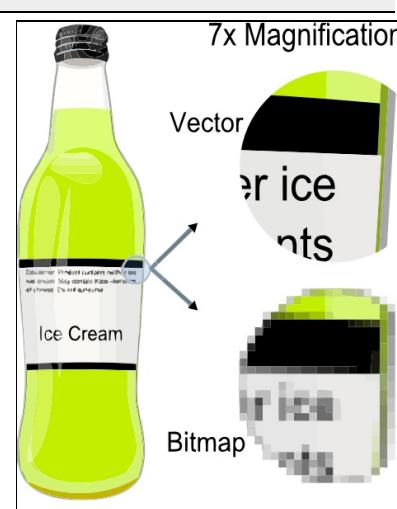
Image Types

Imagine that an image file is opened, and the first few bits read:

011101111100110100011110

What does the computer do with them? What does it interpret them to mean? Computers and images make the encoding bits meaningful with a file type that describes the method of encoding (e.g., BMP, JPG, GIF, SVG, etc.). These file formats provide instructions for interpreting the bits and providing the abstraction that you see as an image on your screen.

If the image file is a raster image, these bits represent color information for a specific picture element (or pixel) on the screen. The most common color model used for computer displays is RGB (*red-green-blue*). In this scheme, the binary encoding may represent the color of the first pixel in the image as follows:



Red	Green	Blue
01110111	11001101	00011110

If the image file is a vector image, these bits represent instructions for drawing shapes. This is much like the Processing programs that contain instructions to draw on the screen:

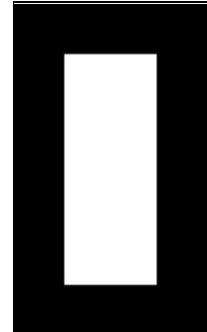
```
rect(300, 200, 150, 50);
```

An example encoding may produce a rectangle if it were for a vector image encoding:

Width	Length	Thickness
01110111	11001101	00011110

Here, the *same bits that determined color before* are abstracted to draw a rectangle with a 119 pixel width, 205 pixel length, and a line thickness of 30 pixels.

Note how simple it would be to double the thickness of the line. Simply multiplying `00011110` by 2 produces `00111100`. If this were a raster format, each of the corresponding pixels of the new larger rectangle would need redefining.



Common Misconception: Bits map directly to an image.

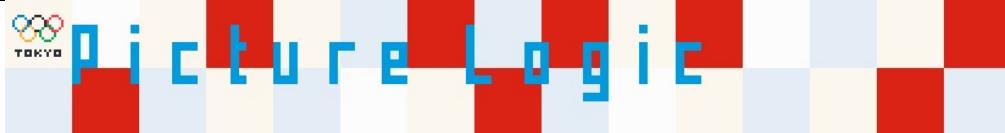
- Abstraction rears its beautiful head once again. Bits mean nothing inherently; their meaning depends upon the instructions to decode them. You could conceivably view ASCII text as a JPG. It would probably be ugly noise – or perhaps exhibit a distinct pattern?
 - Try the reverse — open a JPG in Notepad orTextEdit.
 - Open a file in 'hexdump' to see its hexadecimal representation (BASE16).
 - If you are using a Unix-based operating system (e.g., Linux or Mac OS), try using the command `xxd -b` on a file to see its contents in binary bitstrings of 1s and 0s.

Different file formats (abstractions) lend themselves well to specific purposes. For example:

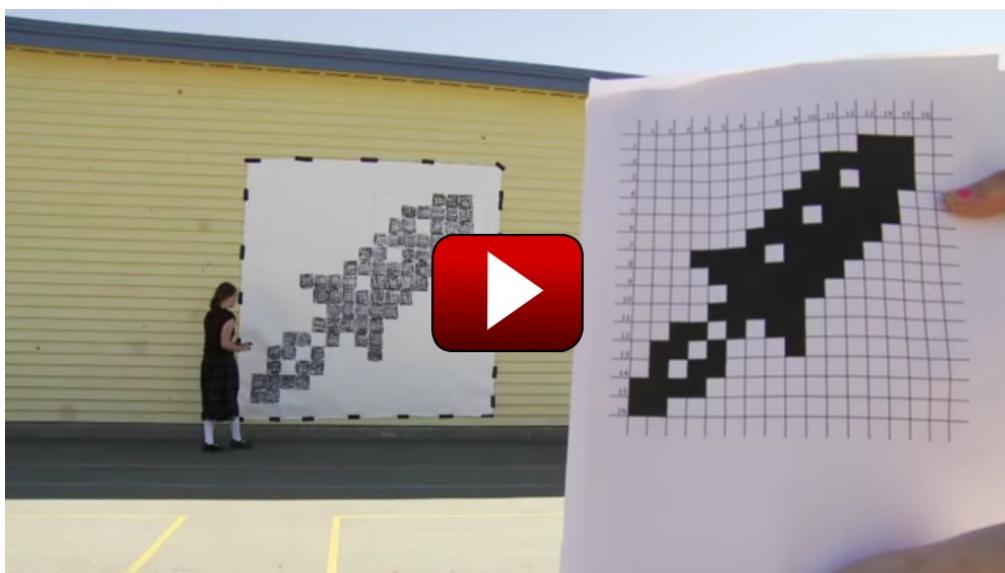
- Compressed [raster](#) formats save hard drive space.
- [Vector](#) formats offer scalability.
- [Raw](#) formats preserve the exact sensor data from a camera or other image capture device.

Picture Logic

An Encoding Scheme Puzzle



File types and other metadata indicate how bits should be interpreted to recreate an image. An **encoding scheme** describes the manner in which a file organizes its constituent bits. There are a variety of encoding schemes that computer scientists utilize for different purposes. Watch this video, which explains image representation in terms of a **run-length encoding scheme**:



<https://www.youtube.com/embed/uaV2RuAJTjQ>

"For example, consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. Let us take a hypothetical single line, with **B** representing a black pixel and **w** representing white:

wwwwwBwwwwwwBBBwwwwwwwwwwBwwwwwwwwwwww

If we apply the run-length encoding scheme to the above hypothetical line, we may interpret this as the following:

5 **w**'s, 1 **B**, 5 **w**'s, 3 **B**'s, 10 **w**'s, 1 **B**', and 12 **w**'s

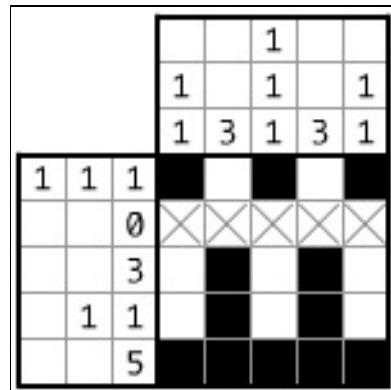
The run-length code represents the original 37 characters in only 16 (22 if you include spaces for clarity).

Of course, the actual format used for the storage of images is generally binary rather than ASCII characters like this, but the principle remains the same. Even binary data files can be compressed with this method; file format specifications often dictate repeated bytes in files as padding space." ([Run Length Encoding](#), Princeton University)

Play the Picture Logic Game

Navigate to the [Picture Logic](#) game, which is based on the run-length encoding scheme, but complicates it slightly. In this game, you will interpret numbers in order to decode an image, much like a computer does.

1. Read the [How to Play Picture Logic](#) tutorial.
2. Navigate to the [Picture Logic](#) game.
3. Attempt the first "How to Play" challenge. What image are you drawing?
4. Continue to complete puzzles until you get stuck.
5. After you complete at least three successfully, you may attempt [other Picture Logic games](#), including *Colorful*, which adds the complexity of multiple colors to the puzzle game.
6. Discuss the following questions as a class when everybody has completed a few puzzles:
 - How are the encodings the same or different from run-length encodings?
 - Why wouldn't a game using just run-length encodings be challenging?
 - Why do you think fax machines use RLE?



UNIT TOPIC:

Image Manipulation

Manipulating Digital Images

- You will design algorithms for modifying the pixels in an image in prescribed ways to create custom image filters.

Filters

Warhol Grids

From 1962–1964, Andy Warhol made thirty silkscreen printings of Marilyn Monroe, addressing themes like death and celebrity ([Tate Museum of Modern Art, London](#)). These colorful screenprints are emblematic of the Pop Art movement that Warhol is commonly associated with. Each screenprint was created from a simple black and white publicity photo of Marilyn used for the 1953 film *Niagara*. Warhol recalls the process:



"In August '62, I started doing silkscreens. I wanted something stronger that gave more of an assembly line effect. With silkscreening you pick a photograph, blow it up, transfer it in glue onto silk, and then roll ink across it so the ink goes through the silk but not through the glue. That way you get the same image, slightly different each time. It was all so simple quick and chancy. I was thrilled with it. When Marilyn Monroe happened to die that month, I got the idea to make screens of her beautiful face—the first Marilyns." ([Webexhibits.org](#))



These transformations were analog, and occurred in physical space. Today, we can programmatically alter images in similar ways. Visit this [WebExhibit](#) about [Andy Warhol's Marilyn Prints](#) and experiment with programmatically altering the publicity photo to create your own Marilyns.



Assignment

Although you can programmatically alter pixels in any way that you wish, some operations are common enough that Processing has pre-built functions that use the graphics hardware in your computer to quickly carry out these image processing tasks. Consult the Processing documentation for [filter\(\)](#) to see examples of each of these common built-in tasks.

Using the [filter\(\)](#) function, alter an image with special



effects to create a 2×2 Warhol Grid.

Example artifact:



Follow these steps to create your Warhol Grid:

1. Find or create a square image.
2. Using the Processing documentation as a guide, create a program that loads the image, filters three copies of it, and saves the result. **Note:** In order to apply the `filter()` method to one image rather than the whole window, you need to specify the image to apply the filter over (e.g., `imgVar1.filter(INVERT)`).
 - Include multiple filters on at least 2 of your variants.
 - One of your variants must apply a single filter multiple times with the use of a loop. For example, variant 1 above uses a loop to blur the image 100 times.
 - One of your variants must apply at least two different filters to the same image. In the example image, Variant 3 includes a lengthy sequence of various filters.
 - Create a labeled Warhol Grid using Processing with the 4 images you have created. You may modify and use the following Processing sketch ([WarholGridStarter](#)) to create the grid. Ensure that the labels describe the effects accurately. Do not simply leave them labeled “Variant x.”

Click to Download: [WarholGridStarter](#)

Submission

Submit the Processing sketch you develop, as well as a reference image.

Reference Starting Points

Functions

`filter()`

`text()`

`save()`

UNIT TOPIC:

Audio Manipulation

Digital Audio

- You will analyze the differences between analog and digital sound.
- You will explore the roles that sampling rate and bit depth play in determining the quality of digitized sound.

Audio Processing

- You will explore methods of programmatically generating digital audio.
- You will explore methods of programmatically altering and modifying digital audio by adjusting volume, pitch, and sampling rate.

Audio Compression

- You will explore the methods and effects of compression algorithms in reducing the amount of data needed to represent an audio sample.

UNIT TOPIC:

Audio Manipulation

Digital Audio

- You will analyze the differences between analog and digital sound.
- You will explore the roles that sampling rate and bit depth play in determining the quality of digitized sound.

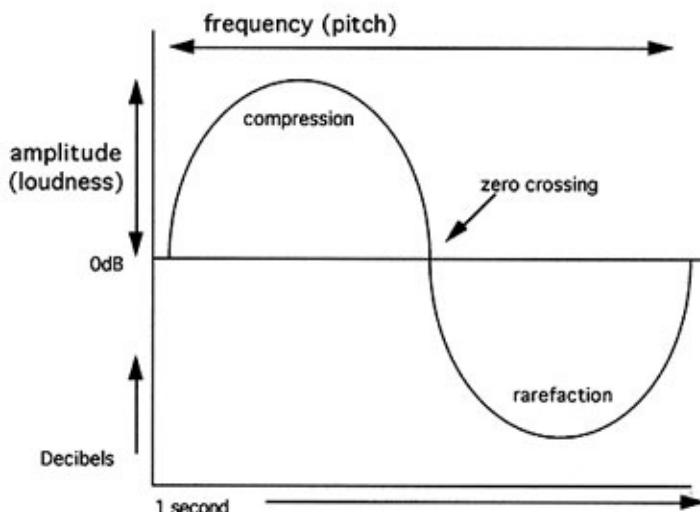
Digitizing Audio

The Nature of Sound

Common misconception: *Digital sounds are "synthesized" sounds.*

- The post-processing effects of technologies like Auto-Tune, noise reduction, etc. are indeed digital, but not in the stereotypical drum machine or synthesizer sort of way. Even hand claps, a capella music, or conversations can be recorded and digitally represented with binary.

Natural sound is simply the movement of molecules through a medium like air. Nature is *continuous*, whereas digitization is *discrete*. When the continuous sound wave is digitized, it is broken into discrete parts. Digital sounds are merely numeric (binary) instructions for replicating the sound production, because all music enters your ears in analog, soundwave form.



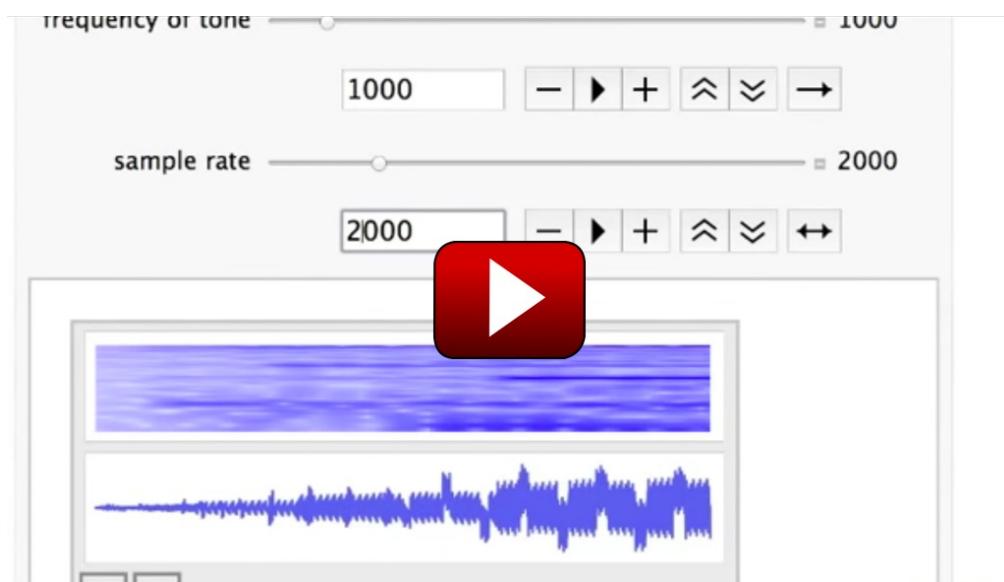
Audio Sampling

Common Misconception: *Digital audio is "stored sound."*

- Digital audio is set of data/instructions to recreate a sound through the right method and device. This is in contrast to vinyl record players, where you can literally scratch a groove with a sewing needle and hear the sound.

How can we convert an analog audio wave to digitized numeric values? We measure the sound wave over fixed periods of time, a process of "[sampling](#)." In essence, audio sampling involves taking short samples of the audio wave almost continuously.

Watch the video to learn more about the digitization process and audio sampling:



<https://www.youtube.com/embed/Ss2DYPBrBhg>

Sampling rates and bits per sample are very important to audio digitization in the following ways:

1. The greater the sampling rate taken from the continuous sound wave, the better quality the digitization.
2. The greater the bit depth used to encode the instructions for playing the sound, the better quality the digitization.

Audio Generation

Theremin Demonstration

Electronics have been used to create music for nearly a century. One of the first electronic instruments is the [theremin](#), an instrument that is played without ever actually touching it.



<https://www.youtube.com/embed/MJACNHHuGp0>

Because the theremin works by using hand positions to directly determine the frequency and amplitude of a sound wave, it is relatively easy to simulate one in software. The code below uses the mouse pointer in lieu of the thereminist's hands.

Installing Processing Libraries

NOTE: Before running the **Theremin.pde** sketchbook, you will need to install the Processing **Sound** software library, which provides the additional code needed for generating audio effects.

1. Open the **Theremin.pde** sketchbook in Processing.
2. Under the **Sketch** menu, select **Import Library...** and choose **Add Library...**
3. Scroll down the list to find **Sound** and select it.
4. Click on the **Install** button.

All future projects that require the **Sound** library will now be able to use this library.

Click to Download: [Theremin](#)

As you experiment with the virtual theremin, note the following:

1. X maps to frequency.
2. Y maps to amplitude.
3. You can click at a particular point to fix the harmonic.

After you have ~~thoroughly annoyed everyone in the vicinity~~ made beautiful music, be prepared to discuss the following before moving on to your assignment.

- What are the parameters required to generate a sound? Note that these are the qualities/quantities that *change* producing a corresponding change in output.
- Are these parameters sufficient to generate *any* type of sound? What other parameters might be needed to generate notes that sound like a guitar or a flute?
- How do these parameters approximate physical reality? How is this similar to the parameters used to generate visual artifacts, like color, position, or brightness?

Assignment

Your assignment is to program a virtual piano in Processing - *il Processiano!* Download the starter code for *il Processiano*.

Click to Download: [Processiano](#)

Extend the code to *generate audio with key presses*. The starter code generates and displays notes when the corresponding keys are pressed (c, d, e), but your program must meet the following specifications:

1. Extend the code to work with the entire range c, d, e, f, g, a, and b. The following [table](#) will help you map notes to exact frequencies.
2. Extend the code to work across two octaves. When you press the SHIFT and C keys (AKA capital 'C'), the procession should play a C note the next octave higher.
HINT: Physics makes this easy. Each note is twice the frequency of itself in the previous octave. In other words, the C one octave above middle C (261.626 Hz) is 2×261.626 , or roughly 523.252 Hz).
3. Try playing a song:

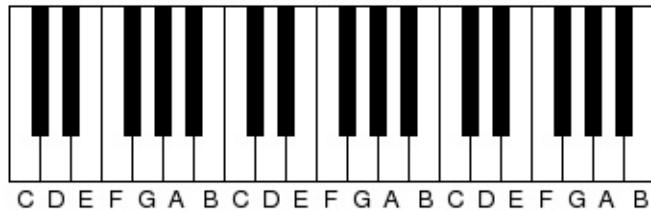
(rh):	3	2	1	2	3	3	3
	E	D	C	D	E	E	E
(lh):	3	4	5	4	3	3	3
	Ma -	ry	had	a	lit -	tle	lamb
	2	2	2	3	5	5	
	D	D	D	E	G	G	
	4	4	4	3	1	1	
	lit -	tle	lamb	lit -	tle	lamb	
	3	2	1	2	3	3	3
	E	D	C	D	E	E	E
	3	4	5	4	3	3	3
	Ma -	ry	had	a	lit -	tle	lamb
	3	2	2	3	2	1	
	E	D	D	E	D	C	
	3	4	4	3	4	5	
	his	fleece	was	white	as	snow	

Submission

Submit the [Processiano.pde](#) file of your Processiano sketchbook.

Extension

Processiano uses only natural (\natural) notes — i.e., no sharps (\sharp) or flats (\flat). Extend the program to allow the user to designate a sharp or flat by using the **CTRL** or **ALT** keys along with the designated note key. Note that not all notes have corresponding sharps and flats! This is why some pairs of white keys on a piano are separated by black keys and some are not (e.g., there is no such note as E \sharp or F \flat).



UNIT TOPIC:

Audio Manipulation

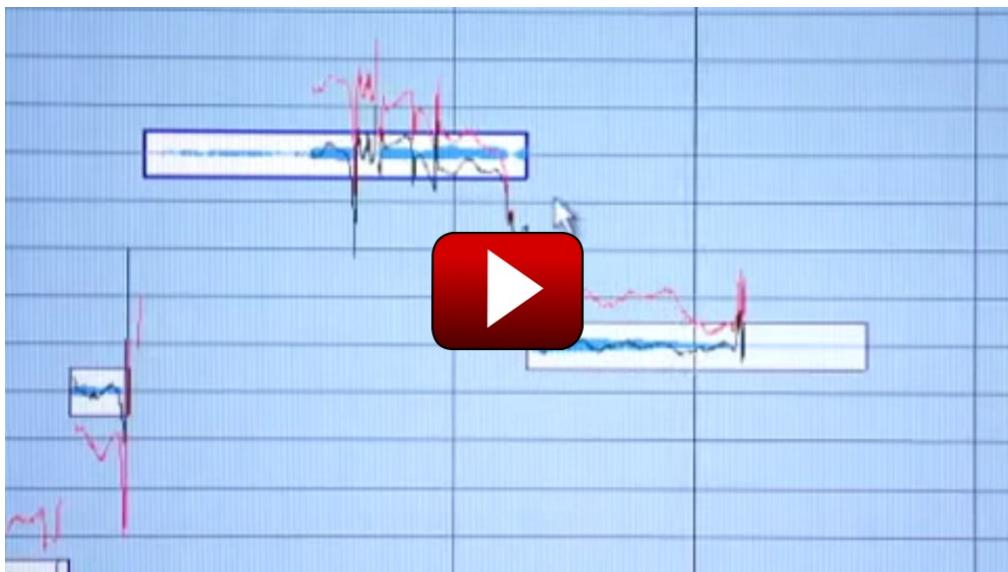
Audio Processing

- You will explore methods of programmatically generating digital audio.
- You will explore methods of programmatically altering and modifying digital audio by adjusting volume, pitch, and sampling rate.

Post-Processing Audio

Auto-Tune

If you've listened to any pop music in the past few years, you've most likely heard a number of songs that use the popular post-processing effect called *Auto-Tune*. Here is a demonstration of how it works:



https://www.youtube.com/embed/9OUgXFZ_WeY

Auto-Tune takes an existing audio file and transforms the bits to make the pitch "perfect." This is possible because digital audio is represented with bits, which can be easily manipulated mathematically using algorithms.

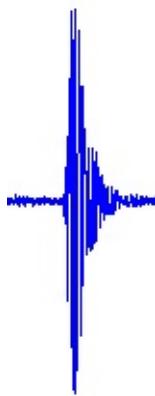
The alteration of a sound after it has been recorded is called *post-processing*. Most audio effects (e.g., echo, reverb, phase shifting, backmasking) are added after the recording (a.k.a. post-processing). Digital representations have alleviated many challenges relating to post-processing of audio, which has drastically changed how we view the creation of music.

Transforming Bits During Post-Processing

Post-processing effects are possible on digital images, video, and sounds because we can transform the bits that encode them. These bits are numeric values and therefore can be altered using algorithms (i.e., functions/math operations).

Let's take a look at how to create an "echo" during post-processing.

- Pretend that we digitized the sound of a hand clap, represented as the following digital sound wave.



- How would we represent the sound when it echoes?
- Since an echo is merely a sound that has bounced off a surface and is re-heard, the sound repeats. Therefore, to simulate an echo, the digitized sound wave should repeat:



- The same digital sound wave is simply copied.

Note that this would result in the exact same sound repeated, but we could overlap the sounds to make the echo effect more pronounced.

Let's think about this in another way, using the following "values" that represent a sound wave:

3-1-2-4-4-5

How might an echo look in terms of the values encoded by the digital representation?

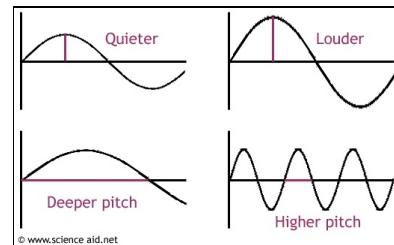
3-1-2-4-4-5 3-1-2-4-4-5 3-1-2-4-4-5 3-1-2-4-4-5 3-1-2-4-4-5

Other post-processing effects are more complex, like noise removal, auto-tuning, etc. If you take all the values in a digitally encoded audio file and reverse their order, that would be similar to [backmasking](#). The variety of post-processing strategies is too great to list here and is ever-evolving, but the great thing is you don't need fancy production software or tools to post-process audio when you have computational thinking and programming skills.

Audio Processing

Digital Audio

Just as digital photography captures and quantizes light and color, digital audio recording may capture sound volume and frequencies. Once these characteristics are represented as values and stored numerically, they may be manipulated. Just as mathematical operations can alter the colors or brightness in an image, they may alter the frequencies and amplitudes of stored sound waves.



However, the techniques used in audio processing are more difficult than those in image processing, because timing is an important factor. You will spend some time doing some simple manipulation of audio files in Processing.

Download the audio processing starter code and 3 sample audio files (compressed into a single [.zip](#)) and execute the sketchbook.

Installing Processing Libraries

NOTE: Before running the [AudioProcessing.pde](#) sketchbook, you will need to install the **Minim** software library, which provides the additional code needed for processing audio effects.

1. Open the [AudioProcessing.pde](#) sketchbook in Processing.
2. Under the **Sketch** menu, select **Import Library...** and choose **Add Library...**
3. Scroll down the list to find **Minim** and select it.
4. Click on the **Install** button.

All future projects that require the **Minim** library will now be able to use this library.

Note: This is a different audio library than the one previously used for Audio Generation.

Click to Download: [AudioProcessing](#)

Examine the code to see how it functions. In particular, note the following code segment:

```
class MyEffect implements AudioEffect
{
    void process(float[] samp)
    {
        float[] newSamp = samp.clone();
        int j = 0;
```

```

        while (j < newSamp.length)
    {
        newSamp[j] = samp[j];
        j = j + 1;
    }
    oldSamp = samp.clone();
    arrayCopy(newSamp, samp);
}

```

This is where the majority of the work takes place. The `process()` function reads into memory an array of the samples and processes each of them one at a time.

In the default code, no post-processing is done. Each sample is simply assigned the unaltered sample: `newSamp[j] = samp[j];`. *Changing this one line can result in a variety of effects.*

Instructions

Perform the following tasks, and note how the audio is altered as a result.

Instead of assigning `newSamp[j]` to be unaltered `samp[j]`, assign it:

- `samp[j] × 2`
- `samp[j] ÷ 2`
- `samp[j] – itself`
- `samp[j] + 1.0`
- `samp[j] + random(0.1)`
- `random(1.0)`

Note that you are operating on each sample. This is akin to operating on each pixel in an image. You can also operate on the order of samples as well.

- Reverse each batch of samples: assign `newSamp[j]` to be `samp[(BUFFERSIZE - 1) - j]`.
- The starter code keeps a copy of the previous batch samples as well as the latest. These are named `oldSamp` and `samp`, respectively. Assign `newSamp[j]` the average of `samp[j]` and `oldSamp[j]`.
- The audio is stored and processed in batches of 4096 samples. This can be altered by changing the `BUFFERSIZE` variable at the beginning of the program code. After altering the variable to a higher value (e.g., `4 * 4096`), re-execute the previous two tasks. What is the effect?

Note that some of the effects will be more pronounced on different types of audio, so you should try each of the tasks above on each of the 3 audio files.

Submission

Submit a reflection on the assignment. Include each of the following items:

- A description of what each of the nine bulleted tasks does,
- an explanation of how each task works, and
- an explanation of how mathematics affects the wave.

UNIT TOPIC:

Audio Manipulation

Audio Compression

- You will explore the methods and effects of compression algorithms in reducing the amount of data needed to represent an audio sample.

X Marks the Spot

Buried Treasure

As you approach the docks, you see Captain Jack, clutching a tattered piece of parchment.



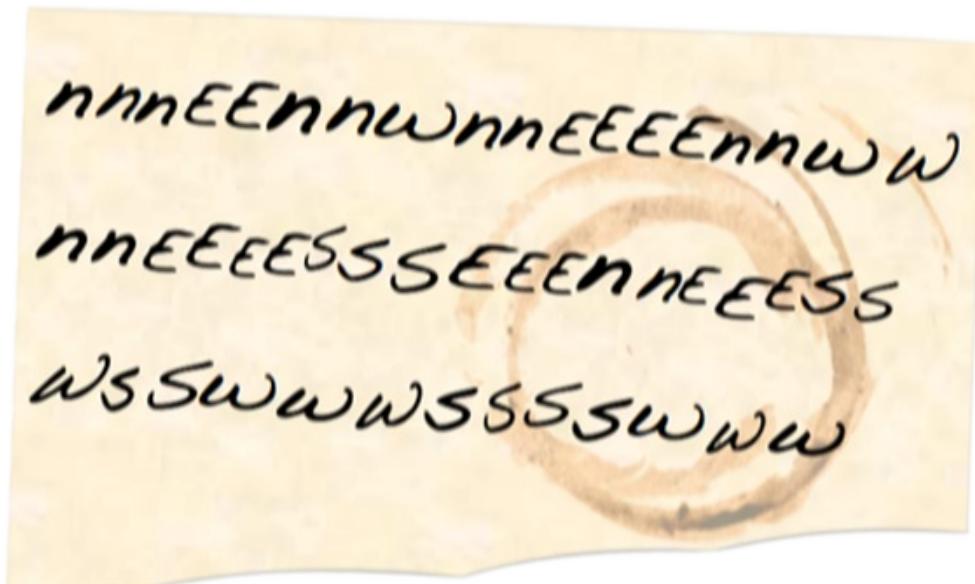
"Ahoy thar! I did bury our treasure but a wee bit yonder," he says, pointing over his shoulder with a crooked thumb. Cracking a toothless grin, he adds, "Bein' first mate, you do serve as me backup in case me directions get lost."

He hands you the parchment. Carefully straightening it, you see a list of letters crossing the page 3 times over, nearly covering it.

"Each of the letters marks 10 paces in a cardinal direction – them be the points on a compass rose." He holds up four fingers, stating, "N be north, E be east, S be south, and W be west," touching each in turn. "At the end of all that pacin', the booty lies but 2 feet 'neath the ground."

Carefully, his brow furrowed in concentration, Jack begins ripping the lower left corner of his page, separating a small piece of parchment from the rest.

Apogetically handing you the scrap and a bulky lump of charcoal to mark your notes, he says, "Err, sorry I did not bring more parchment for ya..."



Can you compress all of this information into a smaller form?

Instructions

Produce a compressed version of the directions so that a stranger could follow them with *minimal* instructions and find the location of the buried treasure. Your minimal instructions should be confined to a brief 3- or 4- sentence description — one that could be given completely orally and easily remembered.

Create a copy of [this spreadsheet](#) to track and compare your solutions. Submit your work when finished.

Compression Algorithms

The Importance of Efficiency

Mathematicians and computer scientists do not like to waste their time repeating themselves. They want to save time, space, and complexity wherever possible. In fact, many of the most common mathematical notations can be considered **compression** algorithms. For example, consider repeated multiplication, which is rife with redundancy:

Efficiency	Expression
Original	$5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5$

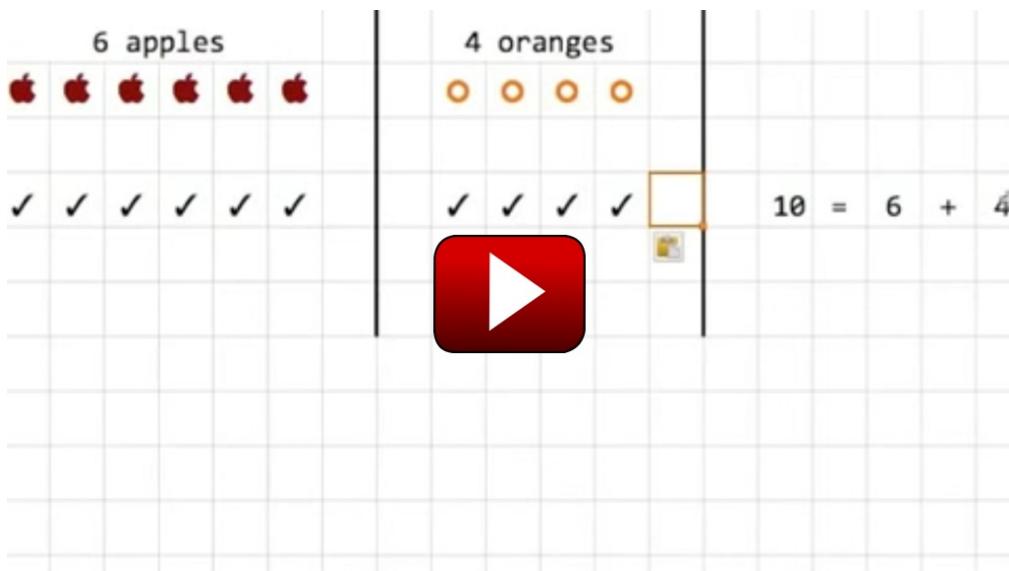
Exponential notation was created to alleviate this degree of **redundancy** and improve legibility:

Efficiency	Expression
Original	$5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5$
Compressed	5^7

Multiplication itself is really nothing more than repeated addition:

Efficiency	Expression
Original	$5 + 5 + 5 + 5 + 5 + 5 + 5$
Compressed	5×7

Watch the video below to learn more about how compression works in terms of mathematical operations, which as you know, are critical to our ability to manipulate digital media:



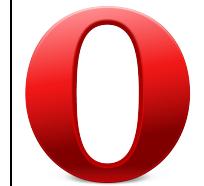
<https://www.youtube.com/embed/cbBFnlwK98M>

The ultimate goal of compression algorithms is to condense information into its most space-efficient, compact form and minimize any waste of space.

Let's take a look at a concrete way of how compression works in computer science and how it may affect you.

Compressed Webpages

Which Internet browser do you use? Chrome? Firefox? Internet Explorer? Safari? Each of these browsers has their advantages and disadvantages.



Opera is another browser option, and it takes a different approach to mobile Internet browsing. Basically, web pages are pre-loaded on Opera's servers.

The compressed pages are then sent to the mobile device when a request is made. By compressing the web pages before sending them to the mobile phone, Opera reduces the quantity of data that needs to be sent over the data network. As a result, web pages load faster on mobile devices and incur fewer charges for data transfers.



<https://www.youtube.com/embed/MVZNTV-cAMg>

Evaluating Compression Algorithms

Not all compression algorithms are created equal. The following guidelines may help users evaluate the effectiveness of algorithms:

- Important characteristics of useful compression "algorithms":
 1. The **process/procedure** nature of algorithms allows for the act of *transforming* the text.
 2. The **consistency** of algorithms ensures the *reliability* of results (always same result).

- 3. The fact that algorithms are **ordered** processes implies *reversibility*, so that the compressed text can be decompressed (not guaranteed – only for lossless).
- A good measure for comparing the effectiveness of compression algorithms is the ratio of length from original to compressed (perhaps denoted as a percentage). **Note:** a better compression ratio over one string of text *does NOT guarantee* that one algorithm is more effective than another. Some forms of compression may be tuned to a specific type of data (like text, music, images, etc.).

Watch the video below to see how two compression algorithms vary and to identify the advantages and disadvantages of these compression algorithm strategies.

Algorithm 1:
Starting at first character:

1. Write down 1.
2. Look at character.
 - a. If it matches the next, add 1 to the number you have written and move right.
 - b. If not, write the character after the number, write down 1 and move right.
3. Repeat Step 2 until the end of the string is reached.
4. Write last character.

Length of original string: 7
Length of compressed string: 6

Project Engage!
The University of Texas at Austin

<https://www.youtube.com/embed/xyKA4arxQ5I>

Common Misconception: One compression algorithm is the best.

- It depends on the context and the patterns of repetition in the data compressed. Also, lossy compression may be acceptable for some types of data (images) and not for others (text).

Since no single compression algorithm is best, it's important that one can analyze and evaluate compression algorithms to identify the optimal strategy for given data in a given context.

Lossy vs. Lossless Compression

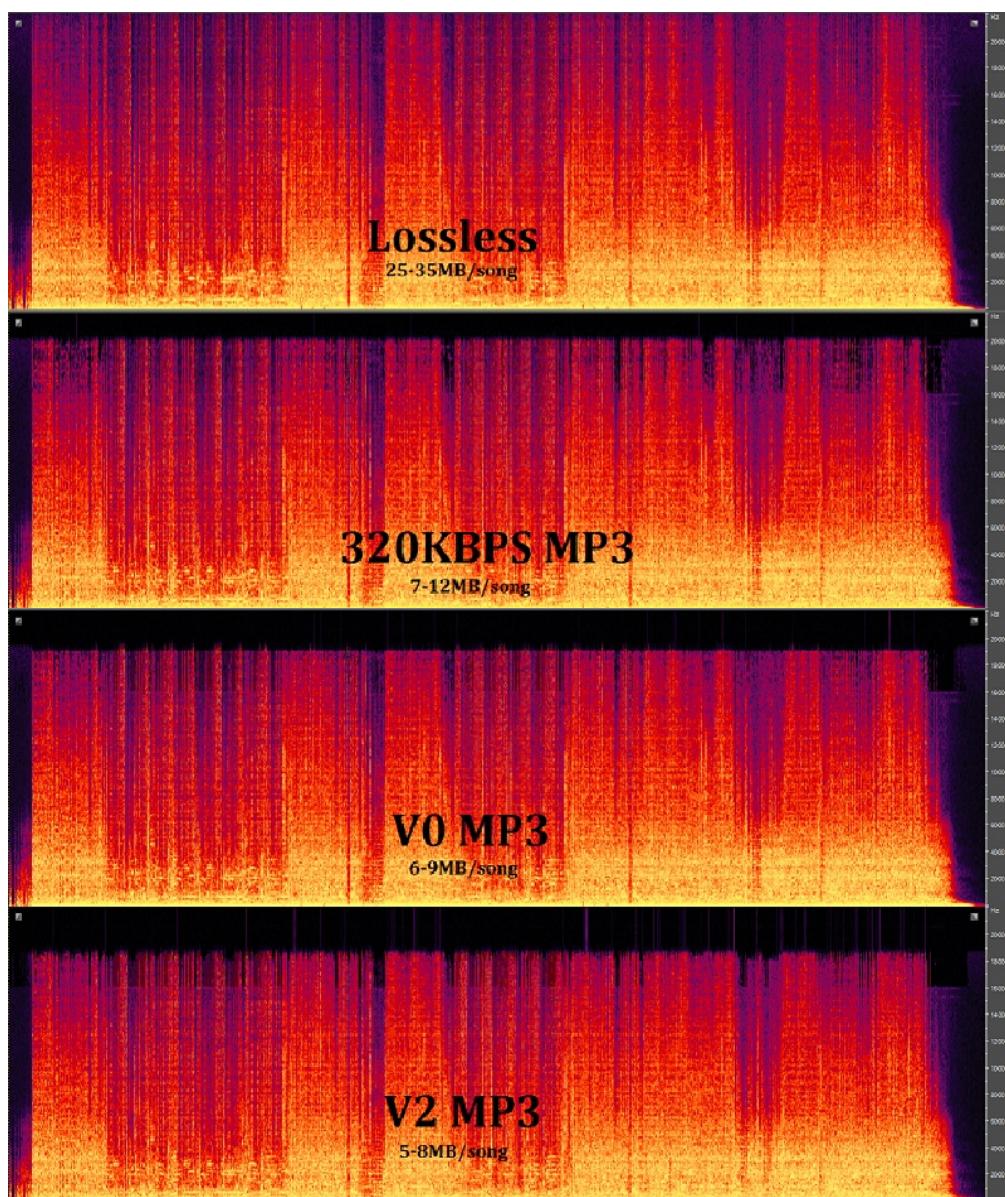
Not all data can be compressed! There is a point at which compression results in a loss of pertinent information. Otherwise one could theoretically compress the content of all the books in the world onto a single sheet of paper (*On a related note, read [this forum post](#) about compressing a compressed file*).

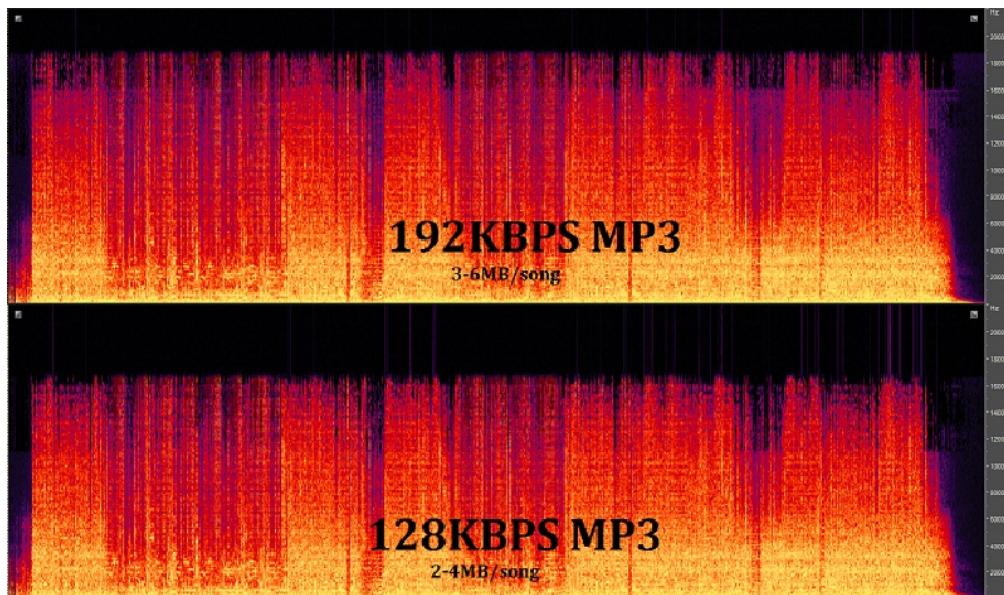
For example, JPG images and MP3 files are already compressed media files. There are firm limits to how much information can be compressed without losing too much, and we use the

terms *lossless* and *lossy* to describe different compression algorithms that adhere to these limits and those that don't.

Lossless means "without loss," just like hopeless means "without hope." **Lossless** compression means that compression has occurred *with zero loss of information*. On the other hand, **lossy** compression indicates that there *has* been some data lost through compression. Different lossy compression algorithms can result in different amounts of lost data. Let's consider an example of lossy compression.

The most common lossy compression occurs when you "rip" an audio CD and convert the tracks to MP3s. The images below represent the digital soundwave that exists before and after various stages of compression from raw audio to MP3. As you scroll down and the audio becomes more compressed, what differences can you notice?





Perhaps it's easier to see quality loss in lossy compression with compressed images:

Example of Lossy Compression



Original Image
(175KB size)



Compressed Image
(25KB size, 85% less
information)



Compressed Image
(11KB size, 94% less
information)

As the image is compressed, there are obvious data/quality losses.

Though these examples are indicative of lossy compression, not all compression is lossy. As mentioned previously, lossless compression involves no data loss. With lossless compression, you can convert back to the original at any time (may be time-intensive, however). For example, FLAC audio loses no quality over CD.

This raises the question:

If lossless compression exists, why would anybody want to use lossy compression?

Lossy compression algorithms often result in greater reductions of file size, offer the best compression ratios, and are designed to be "good enough" approximations. A good example might be the use of a thumbnail by a website as a link to a larger image. The compressed thumbnail version reduces the amount of data necessary to load the page, but if users would like to see the original image, they can follow a link to that version.

However, lossy compression can never be "undone," because the lossy compression algorithms remove information that isn't necessary for representation, but can never be

reconstructed once lost. In other words, you cannot go from a lossy-compressed image back to the original image.

BIG PICTURE:

Ethics of Digital Manipulation

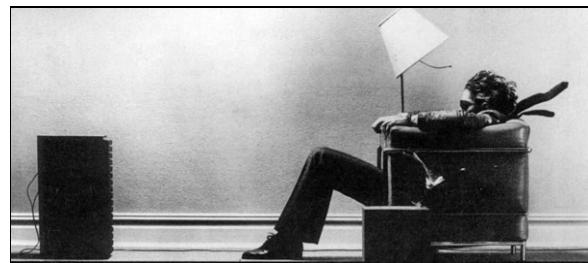
Highlights

- You will explore the positive and negative consequences of digitally altering images.
- You will discuss the ethics of digitally manipulating images, especially in the context of journalism.
- You will discuss the issues related to intellectual property.
- You will explore the limitations and rights associated with a number of common licenses, including Creative Commons.

Original or Manipulated

The Photoshop Phenomenon

In today's world, we consume media of all sorts on a daily basis. Multimedia surrounds us, and often influences our thinking, decision making, and actions. However, the digital nature of these artifacts makes it possible for them to be easily manipulated, edited, "Photoshopped," remixed, remashed, etc., so it can be difficult to decipher between what is "real" and what is "fake."

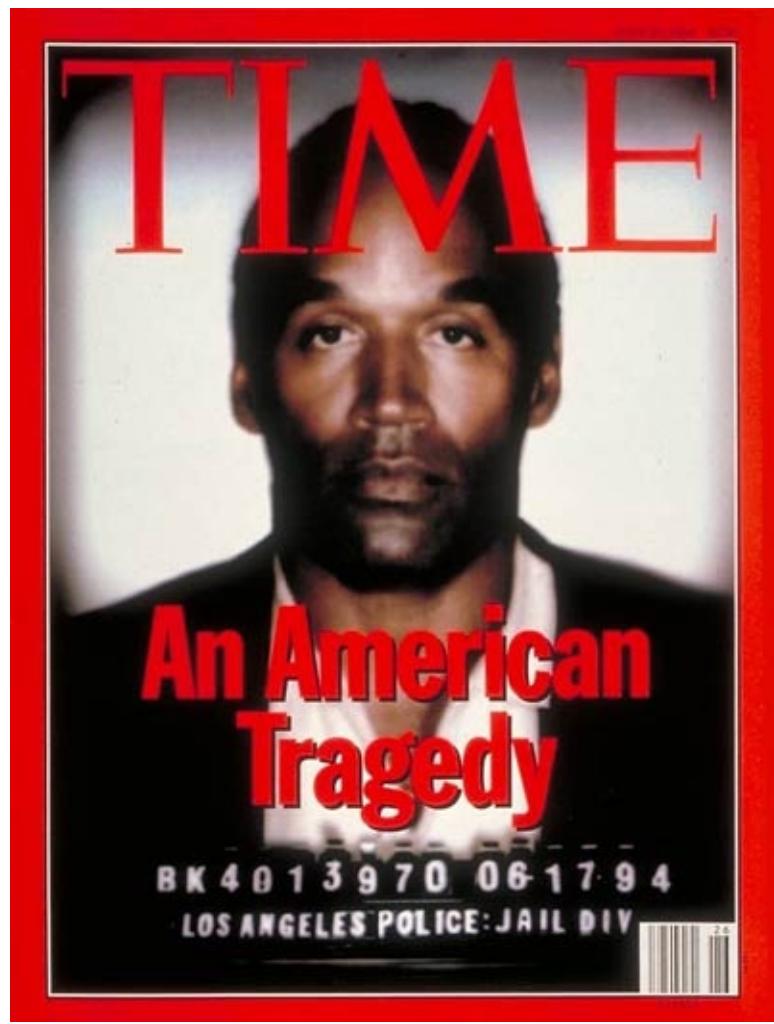


Take this challenge to see how well you can discern between the "real" and the "fake". You will be shown a photo chosen from a set of manipulated photo pairs (in other words, one original and one modified version exist for each photo), and you must decide whether the photo has been digitally manipulated.

After you're done, your teacher will present each pair of original and manipulated photos, and you will discuss possible motivations for manipulating each of the photos. Note that each of these photos comes from actual instances in the news media.

Can You Tell the Difference?

- 1) Is this image the original or was it manipulated? Explain your reasoning.



Answer:



-
- 2) Is this image the original or was it manipulated? Explain your reasoning.



Answer:



3) Is this image the original or was it manipulated? Explain your reasoning.



Answer:



4) Is this image the original or was it manipulated? Explain your reasoning.



Answer:



5) Is this image the original or was it manipulated? Explain your reasoning.



Answer:



6) Is this image the original or was it manipulated? Explain your reasoning.



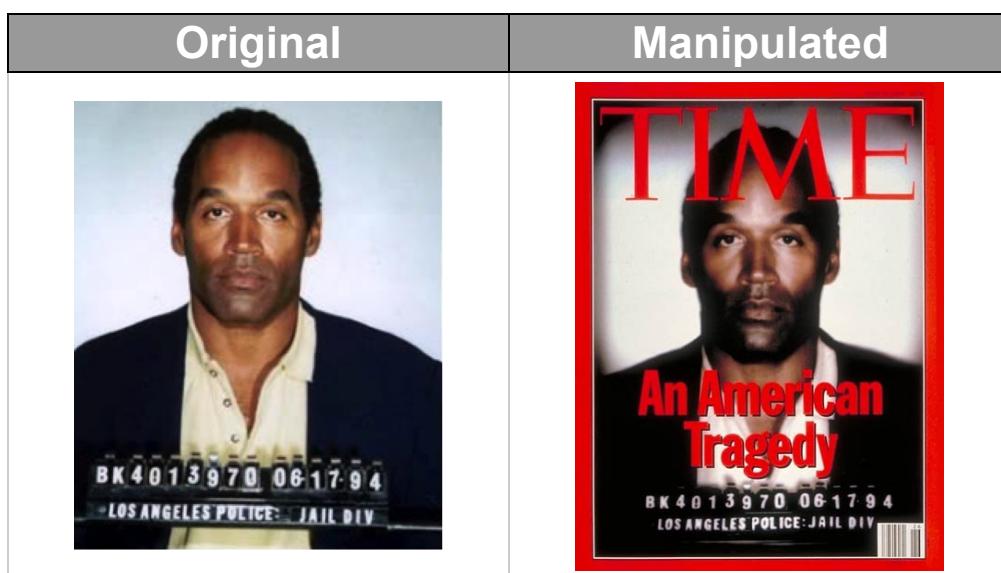
Answer:



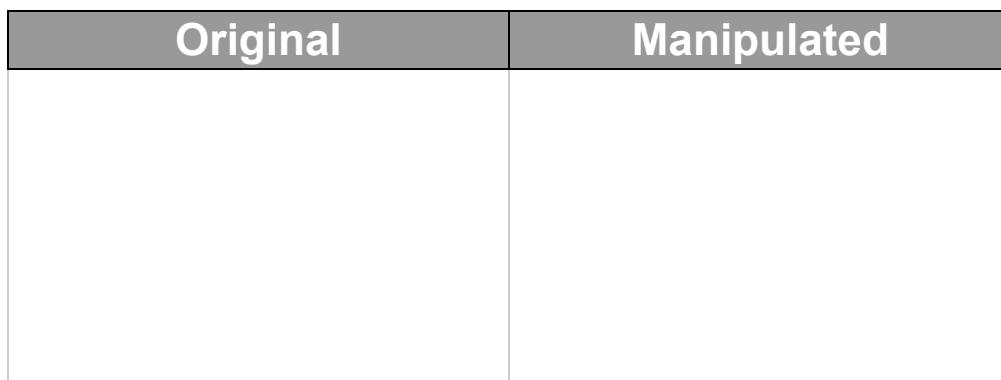
Original or Altered (Answers)

Before and After Comparisons

1) When *TIME* magazine put a mugshot of OJ Simpson on its cover in June 1994 following the brutal double slaying of his ex-wife and her friend, the magazine was widely criticized for manipulating the color tones and shadows to make Simpson appear more ominous and to incite racial sentiments. *TIME* magazine defended their choice as "artistic interpretation." In contrast, *Newsweek* magazine published an undoctored version of the same mugshot for its cover that same week as well, making *TIME*'s manipulation of the image that much more apparent.



2) This is one of at least 79 altered photos that veteran news photographer Allan Detrich of the *Toledo Blade* was found to have modified prior to their publication in the paper. In this particular photo, an image of a basketball was digitally inserted into the photograph, presumably to balance the composition and add excitement to the play being captured by the shot. Despite his reputation as a Pulitzer Prize finalist, the criticism he received for editing the photos beyond the newspaper's accepted standards led to Detrich's resignation and a public apology from the paper.





3) In 2008, the *Liberty Times* published a photo of a visit from a Taiwanese delegation to the Vatican with Pope Benedict. The photo that ran in the paper, however, had been significantly modified to remove the publisher of a rival publication from the center of the image. The paper and the reporter who edited the photo came under criticism that their actions "violated journalistic ethics."



4) A pair of photos taken in 2003 near Basra, Iraq, were combined to make a single, more dramatic image. The two photos were from a series of images shot within moments of each other by photographer Brian Walski that depicted scenes of British soldiers and Iraqi civilians in the war-ravaged region. Before transmitting the day's photos back to his newspaper in Los Angeles, Walski chose two of the photos — neither of which was exceptional on its own — and composited the best parts of each photo into a single image. In one photo, an armed soldier was caught in a striking pose with his arm outstretched while an Iraqi man holding a child was somewhat visible in the background, looking away. In the second photo, the man with the child was more prominent and looking at the soldier, but the soldier's posture was less dramatic. By compositing the two photos from the same actual event, Walski manufactured a moment in time that never actually existed. Upon the discovery of his selective altering of events, the *LA Times* fired Walski, stating that, "What Brian did is totally unacceptable and he violated our trust with our readers."

Original	Manipulated
 	

- 5) In 2005, *USA Today* briefly ran a photo of Secretary of State Condoleezza Rice in which her eyes had been unnaturally brightened, resulting in a "menacing, demon-eyesing stare." After the manipulated photo came to light, *USA Today* pulled the photo from its website and offered the following explanation:

"Editor's note: The photo of Condoleezza Rice that originally accompanied this story was altered in a manner that did not meet USA TODAY's editorial standards. The photo has been replaced by a properly adjusted copy. Photos published online are routinely cropped for size and adjusted for brightness and sharpness to optimize their appearance. In this case, after sharpening the photo for clarity, the editor brightened a portion of Rice's face, giving her eyes an unnatural appearance. This resulted in a distortion of the original not in keeping with our editorial standards." — USA Today

Original	Manipulated
	

- 6) In 2009, controversy arose concerning the amount of post-processing of images that should be allowed in a Danish photo contest. Photojournalist Klavs Bo Christensen was disqualified from the event after contest judges decided that his submission crossed the line

of what was acceptable. Christensen has disputed the judges' decision, claiming that the alterations he made to the color and saturation levels (i.e., "burning," "dodging," and "color correction") fall within the letter of the contest rules, which include the following restriction:

"Photos submitted to Picture of The Year must be a truthful representation of whatever happened in front of the camera during exposure. You may post-process the images electronically in accordance with good practice. That is cropping, burning, dodging, converting to black and white as well as normal exposure and color correction, which preserves the image's original expression."

The judges disagreed, saying that Christensen's photographs "went too far" and that, "The colors almost look like they have been sprayed onto the pictures."

Original	Manipulated
	

Ethics of Digital Manipulation

Debate

In this activity, your class will hold a debate in which one side argues that "digital manipulation in the media" is *essentially unethical* while the other side argues that it is *ethical or irrelevant*.



1. Your teacher will randomly assign students to be either proponents or opponents in the debate over the statement, "digital manipulation of media is never or rarely ethical." *Students must argue their assigned points of view regardless of whether they personally agree with them!* This is important in ensuring that the debate remains challenging and balanced.
2. Watch the video "[Everything is a Remix: Part One](#)" (7:18) to learn more about re-mixing and re-mashing.
3. Prepare a team argument for your assigned side in the debate. You will have 5 minutes to prepare. You should reference the following:
 - the video,
 - the [Original or Manipulated](#) exercise,
 - evidence from public record or personal experience, and
 - concepts covered in the course.

1. Debate the ethics of digital manipulation in the media. Your team will debate according to the debate protocol of your teacher's choosing. Your teacher will be the judge and decide which team puts forth the better argument.

Make sure you address the *issues*, not the other team!



<https://player.vimeo.com/video/14912890>

Creative Commons

Intellectual Property Rights

Now that the Internet is media rich (with digital images, audio, and video on almost every page), concerns about ownership of these digital properties increase.



For example:

- "If I post a video to YouTube, do I still own it?"
- "If I download an image from the internet, can I use it in an advertisement for my Ebay store?"
- "If I manipulate an image with multiple layers, is the new image all mine?"

For the most part, there are laws in place that protect intellectual property, including digital intellectual property. The most familiar set of guidelines for used to understand the licensing of digital intellectual property is that of [Creative Commons](#).

Watch the video below that outlines the most important concepts behind Creative Commons, and consider the appropriateness of each [license](#) in different contexts:



<https://www.youtube.com/embed/AeTIXtEOplA>

Which Creative Common License Is Right for You?

Complete the [Choose a License](#) webform and select a license that you would prefer to utilize with your own work. Next, write a 1 paragraph rationale for why that license would best meet your needs.

Image Filter Project: Rubric Check



Instructions

This activity provides you and a partner group time to provide one another *critical feedback* about the progress of your projects.

1. Pair up with one another group.
2. Rate the components of your partner group's Processing program and documentation according to the [Image Filter Project rubric](#). You may write this out or print the rubric and circle components on it.
3. Review your partner group's work and provide them with documentation that describes what you *Like*, what you *Wonder*, and what their *Next Steps* should be.
4. When both you and your partner group have completed the previous steps, share your feedback with one another.

Listen to what your partner group says! You do not need to heed all of their advice, but consider it wisely. The whole idea is to have a critical third party provide ideas to make your project better before you submit the final version for a grade.

UNIT 5

Big Data

One of the most powerful applications of computational thinking relates to the creation and analysis of large data sets. In this unit, students will explore the complete set of processes and techniques that are involved in collecting large volumes of raw data and extracting new and useful information. Students will look at a variety of ways that data scientists use techniques such as statistical analysis, data mining, clustering, classification, and automatic summarization to construct and visualize new knowledge. And finally, using these techniques themselves, students will perform their own analysis on a sample data set to discover new insights, which they will share with the class through a formal, TED-style presentation.

UNIT PROJECT:

TEDxKinda

Highlights

- You will collaborate in groups to analyze public data sets and extract insightful information and new knowledge using a number of big data analysis techniques and tools.
- You will evaluate and justify the appropriateness of your chosen data set(s).
- You will construct informative and aesthetically pleasing data visualizations.
- You will write a script and prepare speaker notes for a formal presentation of your findings.
- You will cite all online and print sources used in your research and presentation preparation.
- You will deliver a TED-style presentation discussing your data analysis and findings using appropriate terminology.

TEDxKinda Project

"The goal is to turn data into information, and information into insight." – Carly Fiorina, former chief executive of HP



<https://www.youtube.com/embed/I6APyXdHBcg>

Big data is all around us. What can it do for you? us? the world? Tara's use of big data to win a political campaign is no longer a novel strategy:

The Obama team understood the importance of execution and the difficulties of data complexities. One of that group's first priorities before the election was to undertake a massive, 18-month-long database merge so that all data could be housed in a single repository. The database focus also allowed the Obama camp to think expansively in its approach to metrics. "We are going to measure every single thing in this campaign," campaign manager Jim Messina told TIME. Messina was also given good resources to work with: according to the article, he hired an analytics department five times bigger than the 2008 operation. Quite the contrast to the rushed approach taken by the Romney campaign.

So what was the Obama camp able to do with this big data trove? One senior official from the campaign told TIME that the group "ran the election 66,000 times every night" based on the day's data and allocated resources based on likely outcomes. In addition, the demographic information they collected and scored against other factors allowed them to find more targeted ways to buy television advertising to reach their "microtargeted" voters.

Source: [2012 - The First Big Data Election](#), Harvard Business Review

Every day, big data analysis leads to innovative ideas, applications, and knowledge. Many of

these innovations and discoveries are shared via dynamic [TED](#)talk presentations. [TED](#)is a nonprofit organization devoted to "Ideas Worth Spreading." Many of TED's best presenters utilize big data analysis techniques, too. For example, here is a presentation called "[Your Phone Company is Watching](#)," by Malte Spitz.



<https://www.youtube.com/embed/Gv7Y0W0xmYQ>

Now, it's your turn.

Assignment

Create a TEDxKinda presentation based on big data analysis.

Groups will complete extensive research to choose a topic/theme (i.e., identify a problem), conduct in-depth data analysis, and discuss the results via a TEDxKinda presentation, which will be performed for a live audience and recorded.

Each group in your class will choose its own meaningful topic, with the following caveat:

Data sets will **not** be available on every topic, so you may first want to identify potential data sets before you narrow down your ideas to a particular topic or theme. There is a wide variety of public data sets and tools that you may choose from in order to complete your presentation, but you will have to be creative, flexible, focused, and diligent in order to synthesize all the components into a composed, powerful presentation. Feel free to use and edit the [Big Data Sets](#) and [Tools for Big Data Analysis](#) resources.



Each TEDxKinda presentation should:

- include at least two of the following data analysis strategies:
 - cluster analysis,
 - classification,
 - linear regression,
 - association rule mining, and/or
 - outlier, anomaly, and change detection.
- utilize automated summarization,
- provide insightful analysis and evaluation of the topic and related big data sets, and
- be professional and engaging to the audience.

For extra credit, you can try to apply appropriate crowdsourcing strategies and achieve useful results!

For classes that want to go the extra mile, create your own [TED-Ed club](#) at your school, or you can invite the outside world to your TEDxKinda talk and organize an official [TEDxYouth@{insert your city's name}](#), like [TEDxYouth@Austin](#). This requires a little extra work, but changing the world is never easy!

Submission

Your submission will be in the form of a presentation, including [speaker notes](#). The presentation you submit must:

- utilize the data analysis strategies as described above and include information from these analyses explicitly,
- provide insightful analysis and evaluation of the topic and related big data sets,
- contain at least three informative and aesthetically pleasing data visualizations,
- use key terminology from the glossary appropriately and as necessary, and
- include effective speaker notes for asynchronous presentations (and evaluation).

When you are finished, you must submit a copy of your presentation (e.g., Powerpoint, Prezi, etc.) *including speaker notes*. If this includes multiple files, zip the files together into one file. Be sure to appropriately name the file you upload as your submission.

Learning Goals

Over the course of this module and this project, you will learn to:

- Relate the impact of computing to ubiquitous and large-scale data processing.
- Outline the purpose of each of the following data processing tasks: collection, knowledge extraction, data storage, and analysis.
- Define "data" and explain the characteristics of usable and useful data.
- Extract structured information from unstructured data.
- Apply the technique of crowdsourcing to a novel data collection problem.
- Apply the basic features and functionality of modern relational databases.
- Evaluate the trade-offs associated with the storage of structured and unstructured data.

- Debate the implications of large-scale data storage and data persistence on privacy and utility, including the costs associated with each.
- Appraise the trade-off of utility and confidence in descriptive, predictive, and prescriptive data analysis.
- Perform traditional statistical hypothesis testing and exploratory data analysis.
- Visually perform cluster analysis, modeling the dynamics of groups.
- Visually perform anomaly/outlier/change detection and discuss the impact on potential inferences drawn from the data.
- Synthesize a prediction through regression over known data.

Rubric

Criteria	Scoring Notes	Score	
Data Analysis—Method 1 Performs one of the following data analysis strategies (association rule mining, classification, regression analysis; cluster analysis; anomaly, outlier, and change detection) on a large data set accurately and appropriately. Weighted: 20%	<ul style="list-style-type: none"> • Students must use one of the big data analysis techniques learned throughout the course. • The technique must be applied accurately and appropriately. • The technique must apply to the topic or theme. 	2 pts	0 pts
Data Analysis—Method 2 Performs a second strategy from the following list (association rule mining, classification, regression analysis; cluster analysis; anomaly, outlier, and change detection) on a large data set accurately and appropriately. Weighted: 20%	<ul style="list-style-type: none"> • Students must use a different one of the big data analysis techniques learned throughout the course. • The technique must be applied accurately and appropriately. • The technique must apply to the topic or theme. 	2 pts	0 pts
Data Analysis—Automated Summarization Performs automated summarization strategies on a large data set accurately and appropriately. Weighted: 10%	<ul style="list-style-type: none"> • The technique must be applied accurately and appropriately. • The automated summarization strategy used must apply to the topic or theme. 	1 pt	0 pts
Insight	<ul style="list-style-type: none"> • Students clearly understand the data collected. 		

<p>Provides insightful analysis of the data set and makes clear connections to the TEDxKinda presentation topic.</p> <p>Weighted: 20%</p>	<ul style="list-style-type: none"> Students draw conclusions relating to the topic and shares them concisely throughout the presentation. Students are able to answer simple questions about the topic or theme. 	2 pts	0 pts
<p>Quality</p> <p>The presentation should be easily heard, easy to comprehend, and aesthetically pleasing.</p> <p>Weighted: 10%</p>	<ul style="list-style-type: none"> Speakers are easily heard. The presentation remains on topic. The videos and/or slides are aesthetically pleasing and easy to read. 	1 pt	0 pts
<p>Engagement</p> <p>The presentation is engaging and interesting throughout.</p> <p>Weighted: 10%</p>	<ul style="list-style-type: none"> Speakers uses appropriate visualizations. Speakers keep the audience engaged during the presentation. 	1 pt	0 pts
<p>Presenter Notes</p> <p>The presenter notes are complete and thoroughly describe the presentation.</p> <p>Weighted: 10%</p>	<ul style="list-style-type: none"> Presenter notes are submitted prior to the presentation. Notes are complete and thorough, describing the full presentation. 	1 pt	0 pts
TOTAL			10 pts

UNIT TOPIC:

Data Science

Introduction to Big Data

- You will relate the impact of computing to ubiquitous and large-scale data processing.
- You will explore the ways that patterns within large data sets can be used in a predictive manner.
- You will discuss the risks and benefits of drawing conclusions from patterns found in large data sets.

Usability and Usefulness of Data

- You will identify the characteristics that differentiate usable data from unusable data.
- You will identify the characteristics that differentiate useful data from useless data.

Data Visualization

- You will combine visuals, content knowledge, and interaction to create a dynamic infographic that clearly communicates discrete information about a data set.

UNIT TOPIC:

Data Science

Introduction to Big Data

- You will relate the impact of computing to ubiquitous and large-scale data processing.
- You will explore the ways that patterns within large data sets can be used in a predictive manner.
- You will discuss the risks and benefits of drawing conclusions from patterns found in large data sets.

What Is Big Data?

The Origins of Big Data

There is a mind-boggling amount of data floating around our society. Physicists at CERN have been pondering how to store and share their increasingly massive data for decades—stimulating globalization of the Internet along the way, while "solving" their big data problem. Tim Smith plots CERN's involvement with big data from 50 years ago to today.



<https://www.youtube.com/embed/j-0cUmUyb-Y>

The Power of Big Data

In this unit, you'll discover that everything we say and do in our everyday lives can be quantized, categorized, and analyzed in minute detail. And not only does all of that data encapsulate who we are, what we do, what we like, what we think, who we know, and where we go, but it also manages to capture many of the subtle details of our lives that we may not even be aware of ourselves.

Through clever and sophisticated analysis of large volumes of finely detailed information, data scientists are able to tease out hidden patterns and reveal connections that might not be so obvious at first glance.

For example, many businesses collect, analyze, and use big data in an attempt to better understand their customers and improve the service they provide to those customers. However, this doesn't always go quite according to plan.

The Downside of Big Data

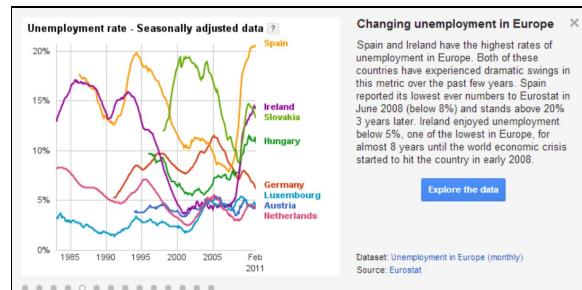
Discovering hidden patterns can be a boon for innovation and efficiency, but what if you are a person who would prefer those patterns stay hidden? What if the information that Big Data reveals actually exposes you in a way that leaves you vulnerable to abuse or misuse of your data? Read [Blown to Bits, Chapter 2](#) to learn more about the risks that Big Data may pose to

individual privacy.

Exploring US Employment Data

US Employment Data

Time to get your hands dirty exploring some free, or "open," data sets using [Google's Public Data Explorer](#). The Google Public Data Explorer has more than 100 different data sets that you can visualize in different ways, but for this assignment we will focus on two that are reputable and address the same topic — employment in the US.



The two data sets you will explore for this assignment are:

- [Unemployment in the US](#) (data collected by the US Bureau of Labor Statistics)
- [Income Inequality in the US](#) (data collected by the US Census Bureau)

Instructions

First, follow both links to their related data visualizations. Each page contains parameters on the left-hand sidebar that populate the chart with data. The tool will prompt you to choose some of these parameters. Choose a few parameters and examine the resulting charts. What about these visualizations interests, surprises, or informs you?

After you have familiarized yourself with both data sets and visualizations, perform a more focused analysis of each in order to discover knowledge that you can share with others.

1. Create a visualization by purposefully choosing parameters that help you examine one aspect of unemployment and/or inequality (e.g., educational attainment, ethnicity, sex, age, or location).
2. Write one paragraph that describes the data visualization you created, explaining what the visualization shows, why this interests you, and why it should be meaningful for others to understand.
3. **Share a link to your visualization, a screenshot of your visualization, and the paragraph you wrote with the class, using a shared space provided to you by your teacher.**
4. Explore your classmates' posts, comparing them to your own. Think about any changes you might make, and provide feedback to your classmates.

This is an introduction to **Knowledge Discovery in Databases (KDD)**, which is a primary purpose of analyzing big data sets. Use this opportunity to practice identifying meaningful data visualizations and writing informative explanations.

UNIT TOPIC:

Data Science

Usability and Usefulness of Data

- You will identify the characteristics that differentiate usable data from unusable data.
- You will identify the characteristics that differentiate useful data from useless data.

Usability vs. Usefulness

Usable and/or Useful Data?

Imagine that a cop uses a radar gun to catch speeding cars, but that the radar gun is completely unpredictable. This faulty radar gun does one of the following, with no predictability:

1. determines the car's correct speed
2. provides no reading at all, or
3. shows an incorrect speed



The unpredictability of this imaginary radar gun would make the data it produces useless. This brings us to the crux of this activity: not all data is useful or usable.

Usable Data

What makes data **usable**?

Data is usable if you can use it, regardless of whether that data is informative or *useful*.

Another aspect of usability is whether the means to process or analyze the data currently exist. Consider the case of the [SETI Institute](#), which has been collecting data from radio telescopes for years and years, but has only been able to analyze roughly 2% of it (despite the vast amounts of computation leveraged to do so). This data may not be considered usable in its current form, because it's simply too big and messy.

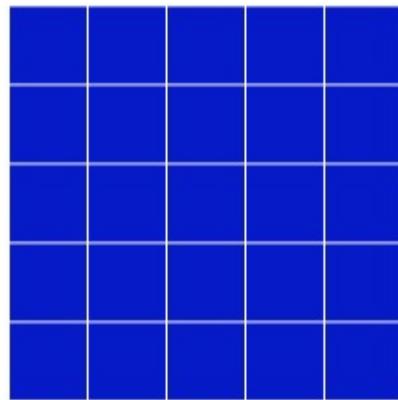
What's interesting about usable data is that in many cases, usable data need not be completely predictable or accurate as long as it falls within a range of possibility. An example of this type of data would be GPS data for phones. With the latest smartphones, GPS can be accurate up to about 10 meters (at its best), which isn't perfect, but is definitely usable...and *useful*.

Useful Data

What makes data **useful**?

Data is useful if somebody would want to use it, in essence making it valuable for some purpose or another (not necessarily financially). The usefulness of data is also directly related to what somebody can do with it.

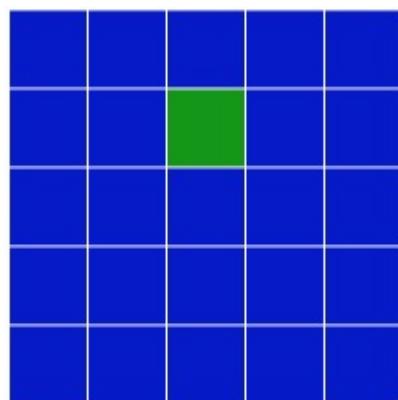
Imagine a light sensor that gives the exact wavelength of light (color) emanating from 1 square foot of the ocean.



This sensor provides an exact, real-time reading, but who cares? Can you make predictions, describe some process, or solve a problem with this data, or is it simply a bit of miscellany that's not of value to anyone? Not surprisingly, the usefulness of data is open to interpretation and context.

Imagine someone has a whole array of these sensors spread across the entire Pacific Ocean. In this case, the amount of color in a grid square can be compared with the data collected from other sensors. Is this useful? That may depend on whom you ask. To a criminal lawyer, car mechanic, or accountant it may be useless, but to a marine biologist, it may be extremely valuable and useful.

Imagine again that one of these sensors has collected data regularly for 30 years, and over that time, the color has shifted from green to blue in certain geographic areas.



What might the reason be for the ocean in this place to obtain a greenish tint? What inference might be postulated with this data? The mere existence of these questions lends credence to the likelihood that this data could be useful.

When determining the usefulness of data, perspective matters.

Common misconception: *If data are being collected, they are being utilized.*

- Data are being collected, because they can possibly be utilized. However, just because data exist does not mean that they are being used or utilized.

Instructions

Think about the following scenarios and how usable or useful the data described can be. Challenge yourself to view the questions from different perspectives, and be prepared to discuss afterward!

Survey

1) Define *usable* in your own words.

Answer:

2) Define *useful* in your own words.

Answer:

3) How **usable** would the receipts from **every McDonalds transaction in a day** be?

- A. Extremely usable
- B. Somewhat usable
- C. Not usable at all

Answer:

4) How **usable** would your **family's medical records** be?

- A. Extremely usable
- B. Somewhat usable
- C. Not usable at all

Answer:

5) How **useful** would **30 random social security numbers** be?

- A. Extremely useful
- B. Somewhat useful
- C. Not useful at all

Answer:

6) How **useful** would **every grade that you've ever received** be?

- A. Extremely useful!
- B. Somewhat useful
- C. Not useful at all

Answer:

7) How **useful** would the **results from hooking a robot up to a polygraph machine** be?

- A. Extremely useful!
- B. Somewhat useful
- C. Not useful at all

Answer:

8) Which of these data sets would be the **most usable**? Why?

- the results from hooking a robot up to a polygraph machine
- every grade that you've ever received
- 30 random social security numbers
- your family's medical records
- receipts from every McDonalds transaction in a day

Answer:



9) Which of these data sets would be the **most useful**? Why?

- the results from hooking a robot up to a polygraph machine
- every grade that you've ever received
- 30 random social security numbers
- your family's medical records
- receipts from every McDonalds transaction in a day

Answer:



UNIT PROJECT:

TEDxKinda

Highlights

- You will collaborate in groups to analyze public data sets and extract insightful information and new knowledge using a number of big data analysis techniques and tools.
- You will evaluate and justify the appropriateness of your chosen data set(s).
- You will construct informative and aesthetically pleasing data visualizations.
- You will write a script and prepare speaker notes for a formal presentation of your findings.
- You will cite all online and print sources used in your research and presentation preparation.
- You will deliver a TED-style presentation discussing your data analysis and findings using appropriate terminology.

TEDxKinda Project: Topics



Presentation Topics and Data Sets

Each TEDxKinda presentation must utilize a number of big data analysis techniques (e.g., visual analysis and crowdsourcing). Consider especially the following passage from the [TEDxKinda Project Description and Rubric](#):

There are a variety of public data sets and tools that you will choose from in order to complete your presentation, but you will have to be creative, flexible, focused, and diligent to synthesize all the components into a composed, powerful presentation. Data sets will not be available on every topic, so you may first want to identify potential data sets before you narrow down your ideas to a topic or theme.

The sooner you can identify an appropriate topic with a relevant dataset, the sooner you may begin and the more purposeful your exploration can be. Use these [Big Data Sets](#) to help jumpstart brainstorming, then continue the search on your own.

Instructions

Your TEDxKinda group must post a 1–2 paragraph to a shared space provided to you by your teacher, with the following requirements:

- Identify your group's preliminary TEDxKinda presentation topic.
 - Include three or more data sets that will inform your group's work.
- Include links, titles, and descriptions.
- State your group's hypothesis for why the data sets and presentation topic may be related.

Of course, you may change topics or data sets at a later time if necessary, but this post describes where you will begin. After you post, read about the other topics groups will be presenting, and explore their data sets. Provide suggestions, share ideas, or ask questions of other groups.

TEDxKinda Project: Big Data Sets



Big Business

Big data has become big business around the world. Companies that have their own big data sets, like Google or Facebook, use those data sets for business purposes and often go through great lengths to keep them private. Big data sets are not always readily available because of this, and because of privacy issues involved with sharing data. Most of the data sets that are openly available are from the government or nonprofit organizations. Some of these are BIG and some are smaller.

This is a list of some of the most accessible and usable big data sets and collections. You may use any of these for your TEDxKinda research, or compile your own subsets of data by using a searchable database. Be sure to clarify the appropriateness of the data sets you find with your teacher:

- [Google Public Data Explorer](#) (130 datasets from Bureau of Labor Statistics, U.S. Census Bureau, etc.)
- [data.gov](#) - an online repository of datasets from U.S. Government
 - Many counties have searchable property databases, such as the [Travis County Appraisal District](#)
 - Some data sets defy categorization, such as the [Texas Death Row Executions](#) data set
- [Google's Ngram Data](#) - data on Google's catalog of millions of books, including raw data sets
- [Google Trends](#) - detailed search history information, including CSV downloads
- [NOAA National Climatic Data Center](#)
- [Knoema](#) - "free to use public and open data platform for users with interests in statistics and data analysis, visual storytelling and making infographics"
- [Geocommons](#) - "all about open data analysis and maps"
- [Stat Silk](#) - "interactive maps of open data"
- [Better World Flux](#) - "a beautiful interactive visualization of information on what really matters in life"
- [Gapminder](#) - "unveiling the beauty of statistics for a better world view"

TEDxKinda Project: Tools



Visual Analysis

- [Google Public Data Explorer](#) - interactive visualizations of large, publicly available data sets
- [Google Trends](#) and [Google Correlate](#) - Google's own big search data sets available for mathematical analysis
- [Google's Ngram Viewer](#) - analyze the history of language through the analysis of words used in books over the past few centuries

Regression Analysis

- [Microsoft's Excel](#) - spreadsheet software
 - Video tutorial - [How to use LINEST\(\) function to perform regression in Excel](#)
- [Open Office](#) - free (albeit limited) spreadsheet software for download
- TI-graphing calculators (e.g. TI Nspire, TI 84, etc.)
 - Video tutorial - [How to use TI-graphing calculators to perform regressions](#)
- [Processing](#) (*very challenging!)

Summarization

- [Wordle](#) - free word cloud visualization tool
- [Text Compactor](#) - web-based automated text summarization tool

Maps

- [Mapstory](#) - organize knowledge about the world spatially and temporally
- [Heat Map Tool](#) - create a heat map that allows you to quickly visualize spatial data using a range of colors
- [Map a List](#) - a wizard for creating and managing customized Google maps of address lists

Crowdsourcing

- [Googledocs](#) - use surveys to gather data from a wide variety of people easily

Examples of Crowdsourcing Tools

- [Mechanical Turk](#) - "A Marketplace for Work"
- [Kickstarter](#) - "Fund & Follow Creativity"
- [Fold.it](#) - "Solve Puzzles for Science"
- [SETI at Home](#) - Help search for extra-terrestrial life with your computing power
- [Star Wars Uncut](#) - crowdsourced version of the Star Wars movie

UNIT TOPIC:

Data Aggregation

Collection

- You will explore the purposes of various processing tasks, including collection, knowledge extraction, and data storage.
- You will identify multiple techniques for data collection, both on and off of the Internet.

Extraction

- You will analyze the characteristics of structured and unstructured data.
- You will extract structured information from unstructured data.
- You will examine methods of extracting information from online sources, including structured and unstructured search engines, screen scrapers, and spiders.

Storage

- You will explore the basic features and functionality of modern relational databases.
- You will debate the implications of large-scale data storage and data persistence on privacy and utility, including the costs associated with each.

UNIT TOPIC:

Data Aggregation

Collection

- You will explore the purposes of various processing tasks, including collection, knowledge extraction, and data storage.
- You will identify multiple techniques for data collection, both on and off of the Internet.

Big Data Collection

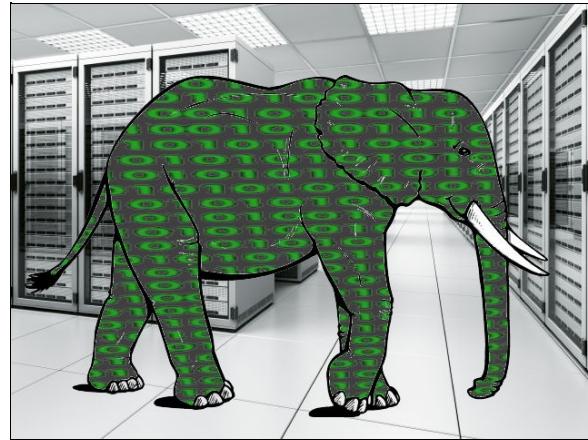
How and Why Is Data Collected

Computing and big data are seemingly everywhere in our digital world, but most of the time we are oblivious to how our data is being collected, and for what purpose it is being used.

Instructions

Your task is to identify **three or more** ways that big data is being collected on a regular basis, including one data collection method for each of the following categories

1. Data collection being performed on the Internet.
2. Digital data collection from a source other than the Internet.
3. Analog (non-digital) data collection being performed in physical places.



For each of these data collection practices:

- Describe how the data collection occurs.
- Analyze the data collection process by answering the following five questions in complete sentences and to the best of your ability:
 - What data are being collected?
 - Why are these data being collected?
 - What are the likely benefits of this data collection?
 - How might this data collection cause harm?
 - How likely is it that these data can be accessed by others?

You may use the Internet or any other resources available to you to identify and analyze these data collection strategies.

Submission

Submit a text document with your description and analysis of the three big data collection strategies you identified.

Creating Structure from Unstructured Data Sets

From Unstructured to Structured

Think about the following scenario and what data is involved:

A gas station has installed cameras and sensors to track the usage of gasoline and make predictions to maintain stock of the right fuels and achieve the best pricing strategies.



Which of the following items are data in this scenario?

Is it data?	Answer
Amount of gas pumped	Yes / No
Type of gas pumped	Yes / No
Price per gallon of gas pumped	Yes / No
Date	Yes / No
Time of day	Yes / No
License plate number	Yes / No
Make/model of car	Yes / No
Color of car	Yes / No

That was a trick question. The correct answer to all of these questions is "Yes." All of these

are examples of data.

Data are pieces of information that are observable and/or measurable. In this scenario, even the color of a car is a characteristic that can be observed, measured, and analyzed — regardless of whether it seems useful. However, the raw data (camera footage) is unstructured. By creating structured data sets, we can make data more usable and useful.

Unstructured vs. Structured Data

Raw, unstructured data can be overwhelming to use if there is too much, or simply lack any organization that would make the data usable and useful. In order to make unstructured data more usable and useful, we apply structure and organization to it, usually after collection. However, structure and organization applied after collection alters it through selection and organization of pertinent details. Details not captured in the resulting organized set may be lost.

Unstructured data contain everything collected in "raw" form, but connections and relationship among strands of data are both harder to trace and much slower to process than structured data sets. On the other hand, structured data are easy to access and organize, but may lack the big picture and details that unstructured data may possess.

Let's consider the following analogy of turning logs into lumber and lumber into a barn:

		
<p>These logs are like raw, unstructured data. In their present state they are not very usable or useful. However, the trees themselves would be even more raw with less structure. By cutting them down, we lose some wood in the form of branches, roots, and leaves.</p>	<p>To make these logs more useful and usable, we apply some structure by converting them into lumber. This lumber is much more usable and useful. For example, one could use it to build a table, a doghouse, a soapbox derby car, or a barn. However, by turning the logs into lumber, we lose some wood in the form of wood chips, tree bark, and sawdust.</p>	<p>To make the lumber more useful and usable, one could build a barn structure. To do this, one has to saw the lumber into different, more useful and usable shapes. However, this process cannot be reversed, so again, we lose some wood chips and sawdust.</p>

Every time we apply structure to an unstructured data set, it becomes more difficult (sometimes impossible) to gain back some of the unstructured data. Think about it this way:

- By turning trees into logs, we lose some "data" (branches, roots, and leaves).

- By turning logs into lumber, we lose some more "data" (wood chips, bark, and sawdust).
- By turning lumber into a barn, we lose even more "data" (wood chips, and sawdust).

However, by adding structure, we also make the data more usable and useful. Hence, one has to weigh the usability and usefulness of structured data against the data loss that occurs from structuring unstructured data sets.

Common misconception: "*Unstructured data*" have no structure.

- At some level, there is structure to even "unstructured data." For example, the binary representation and the particular encoding used for text, video, etc. are forms of structure. "Structure" in the sense of "structured data" means that some level of organization according to their intended use has been applied to the data before they are stored. With unstructured data, data are stored as collected with any necessary structure applied later during analysis.

Utility of Unstructured and Structured Data

Processing unstructured data facilitates knowledge discovery, because emergent groupings (i.e., structure) in data may depend on characteristics that are unknown before data are collected. Knowledge discovery often depends on unforeseen characteristics that may be obscured with structured data. However, when data are imposed with a structure before they are processed, unidentified relationships may be harder to detect than they might be with raw, unstructured data.

Structured data is dependent on known characteristics. For example, data structures could be based on certain prescribed characteristics, such as Employee Name, Employee ID, etc. These structures allow data to be organized into schema that may be more usable and useful. For instance, applying structure to data sets often facilitates quicker data retrieval than with unstructured data: rather than searching through all possible data and selecting the appropriate information (sequential/serial access), structured data are able to exploit predefined methods to access data directly (indexed/random access). Therefore, structured data is usually more efficiently processed than unstructured data. To summarize



briefly:

- **Unstructured data:**

- Everything collected in "raw" form, but connections and relationship among strands of data are both harder to trace and much slower to process
- Requires much more storage space than structured data

- **Structured data:**

- Data that is identifiable because it is organized in a structure
- Requires less storage space than unstructured data
- Organized

Let's examine the data contained in a Walmart receipt, in order to outline the differences between processing unstructured data and structured data.

Unstructured data processing:

With unstructured data processing, receipt data might be stored in the "natural" format in which it arrives (i.e., as a massive set of receipts). Whether these are electronic versions or scanned copies of paper receipts, the only structure assigned them would be those that are implicit in the collection (e.g., transactions might be naturally grouped together because they occurred at the same time or by the same individual). Since this data is to remain unstructured, no further ordering would be imparted through "post-processing."

Structured data processing:

Structured data processing imposes some organization on the data contained in the received transactions (e.g., all transactions could be tracked by each customer, across all trips to any Walmart store). This would facilitate accessibility, search, and reasoning about a particular patron's spending habits (e.g., all data might be organized by date and by store), which may allow the productivity of stores to be assessed and compared easily.

As previously mentioned, though structured data sets allow for easier organization and analysis of certain phenomena, they may make it more difficult to discover unknown relationships because of the structure itself. Remember: when analyzing data sets, one must always ask:

What data or relationships may be lost by imposing structure to the unstructured data set?

Screen Scraping

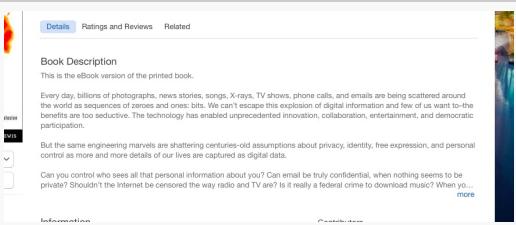
Sometimes we want to analyze data that is already formatted for human use. This data might be a text (e.g. a book), an image (like the one below), a complete website (like this one you're reading), a formatted table/graph, etc. One method for extracting information from these data is called *screen scraping*.

[Screen scraping](#) (or data scraping) is the conversion of data formatted for human use to a format more easily used by automated computer processes. In particular, the conversion is

usually oriented toward taking output produced for humans (such as display on a screen) and producing input for a computer to process (such as a file or the clipboard).

- For example, take a screenshot:
 - Windows: **PrtScn** (Print Screen)
 - Mac OS X: **Cmd** + **Shift** + **3**

Taking a screenshot converts what is seen on the computer display (typically, the entire desktop) to a machine-readable image format, such as in the example below. The file **BtB-screenshot.png** can now be processed like any other image file (e.g., cropped, rotated, etc.) as can be seen in the resulting **BtB-description.png**. In some use cases, an image file isn't appropriate for the task and we want to extract the on-screen information as plaintext. We could then use Optical Character Recognition (OCR) software to convert the visual contents of **BtB-description.png** into another format, such as ASCII text. Now, we can manipulate the words and letters that we captured from the screen just like any other bit text (e.g., copy/paste, translate, etc.).

Filename	Result
BtB-screenshot.png	
BtB-description.png	
BtB-description.txt	<p>Every day, billions of photographs, news stories, songs, X-rays, TV shows, phone calls, and emails are being scattered around the world as sequences of zeroes and ones: bits. We can't escape this explosion of digital information and few of us want to — the benefits are too seductive. The technology has enabled unprecedented innovation, collaboration, entertainment, and democratic participation.</p>

This video is an advertisement for a screen scraping software, but it provides a good overview of how screen scraping can be useful:

Works on

- Windows
- Linux
- Mac OS X
- Solaris
- BSD



<https://www.youtube.com/embed/s0erYF8MsyM>

What other purposes of screen scraping can you think of?

Digitizing Business Cards

Digitizing Business Cards

As digital as our world has become, there is still a tremendous amount of paperwork in it. Schools and workplaces are attempting to conserve resources by going paperless. Luckily, we can use computer science to help with the transition.

One of the challenges and benefits of the paperless office is converting information usually kept on paper into an organized electronic format. By doing so, two goals are met:

1. Use of paper is reduced. This is both good for the environment and a less wasteful use of space.
2. The information is now digitally represented. This means that it is no longer tied to a physical, static piece of paper but can be manipulated computationally. Now, instead of hunting for John Smith's business card, you can search for it electronically. It also means it can be copied without the need for additional physical resources.



Many products exist on the market for converting business cards into electronic data. For instance, a Google search for the term [business card scanner](#) yields many hits that offer solutions. The process is similar to screen scraping and another real life example of [Creating Structure From Unstructured Data Sets](#).

Creating Usable, Useful Electronic Data

Think about the process necessary to convert the paper card into electronic data:

- A digital picture is taken capturing the visible information on the card.
 - At this point, the card is no longer needed. The image of the card has the same functionality as the card itself.
- The image needs to be converted into text, and any images (such as photos or logos) need to be isolated and cropped.
 - The text data is now unstructured.
- *It has no organization!* Before we store this data, we need to isolate the bits of the text corresponding to the attributes we care about (e.g., name, address, phone #).

The final point is the one on which we'll concentrate. Consider these two business cards:

Business Card A	Business Card B
-----------------	-----------------

<p>Kurt Hummel The Treeless Office, CEO</p> <p>khum@treelessoffice.com http://www.treelessoffice.com/</p> <p>305 W. 8th St. Los Angeles, CA 90014</p>	 <p>(213) 555-8963 Fax - (213) 555-3698</p>	<p>Green Products, Inc.</p> <p>1525 N. Park Ave. New York, NY 10029</p> <p>T: (212) 555-9332 F: (212) 555-9333</p> <p>M.Jones@greenpinc.biz http://www.greenpinc.biz/</p>
	<p>Mercedes Jones</p>	

In order to convert these cards to database contact records, we need to first figure out which attributes our contact records support. Obviously, names and telephone numbers are needed, but what about headshots or job titles?

Second, we need to extract the information from each card and fill the contact records appropriately. Fill the following "contact record tables" with the information from the cards above:

Attribute	Business Card A	Business Card B
First name		
Last Name		
Company		
Job Title		
Phone #		
Fax #		
Email Address		
Website		
Business Name		
Street Address		
Street Name		
Unit/Apt. #		
City		
State		
Postal Code		
Country		

Discuss the following questions as a class:

- Which design/content attributes were necessary, and which were not?
- What other content is sometimes found on business cards? *Perform an Internet search to find more examples, if you like.*
- Were there any that require a specific format? If so, what should the format be?
- Are there any that would benefit from having a *default* value?

All of these factors dictate how the record should be filled — but there is more to it. Discuss the following questions as a class:

- How is information located on the card?
- How might a program locate the right information to fill in the attributes for each contact record?
 - Humans have mental rules and intuition for figuring this out, but computers need algorithms to follow and programs to execute. An effective algorithm would be able to help us in this scenario only, but a highly effective algorithm/computer program would work with all possible designs of business cards.

Instructions

Your job is to develop algorithms for identifying five different attributes that are found on business cards. The following attributes are a few examples:

- First name
- Last name
- Company
- Job title
- Phone #
- Fax #
- Email address
- Website
- Business name
- Street address
- Street name
- Unit/Apt. #
- City
- State
- Postal code
- Country

Your rules should make it clear how to automate the collection of information, but they do not have to be written in "code." Here is an example (you may not use this example as your own!):

To identify an email address: query all text on the business card for the following format:

" _____@_____.___ "

- each underscore ("____") represents alphanumeric text
- no spaces or line breaks are allowed for this query

This is only one strategy for identifying a specific string of text. Other strategies may be more

useful for other items, so be creative and experiment. Then, test your rules by searching the Internet for alternate business card designs. Will your rules work for all the cards you find? Make adjustments as necessary. This will help you develop effective algorithms for finding useful and usable data.

Submission

Submit a text document with your five algorithms. Be prepared to share your algorithms with the class and demonstrate how they apply to multiple business card designs.

UNIT TOPIC:

Data Aggregation

Extraction

- You will analyze the characteristics of structured and unstructured data.
- You will extract structured information from unstructured data.
- You will examine methods of extracting information from online sources, including structured and unstructured search engines, screen scrapers, and spiders.

The Internet's Data Structure

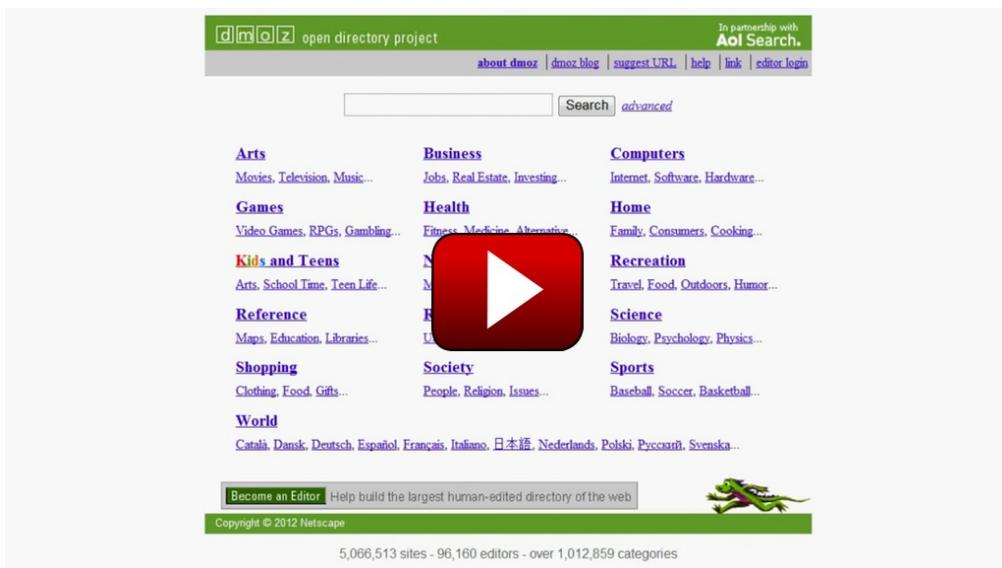
...or Lack Thereof

The World Wide Web is full of unstructured data. There are some light restrictions on how the web is organized (e.g., by nation or by top-level domain — .edu, .gov, etc.), but the web itself is largely unstructured. For example, there are no restrictions on how domain names are semantically organized. Websites about "ferrets" do not necessarily have any distinguishing characteristics in their domain names, such as including the term "ferrets" (e.g., "ferrets.com") or, better yet, a taxonomy like "mustela.mustelidae.carnivora.mammalia.chordata.animalia."

Imagine the web were organized in such a manner as this.

- How would *locating* information be different?
- How would *creating* information be different?

Historically, there have been some attempts to overlay structure onto the Web. One of the most prolific is the [Open Directory Project](#) (aka [DMoz](#)), which contains structured lists of links to individual pages on the web. These directories form the structure and the links represent the data. Watch the following video that demonstrates the difference between structured and unstructured searches for the key term "twins":



<https://www.youtube.com/embed/mIN7O1YyuMs>

Notice that finding information is extremely easy using key terms in Google. Type **twins**, hit enter, and BAM! Results. However, there is a lot of noise in the results. Minnesota Twins and Twins the movie are among the top hits (because Google apparently knows that the searcher loves baseball and Arnold Schwarzenegger movies).

On the other hand, searching via the directories of DMOZ led us down the wrong path

initially. In essence, you need to know the organizing structure that they created in order to really make it useful. Note that Google actually provides *some* structure when you use its autocomplete feature, which could be considered a dynamic structuring schema.

Instructions

Try this type of experiment for yourself. Conduct both a key term search with Google and a directory search with DMoz on the same topic of your choosing. Compare and contrast the results.

Try this experiment:

- Choose a key term to search for.
- Prepare a stopwatch and time how long it takes you to find *quality* results for your key term doing the following:
 1. Conduct an unstructured [Google](#) search for your key term.
 2. Conduct a structured search with [DMoz](#) for the same key term, BUT only use the directories — no key terms allowed!

Submission

Submit links to the *best* link you find with your Google and DMoz searches, along with the time it took you to find each link. Write one paragraph that compares and contrasts the results and evaluates the effectiveness of each processes. Be sure to use the terms *unstructured* and *structured* to describe the search processes, addressing **both the quality and quantity** of your results.

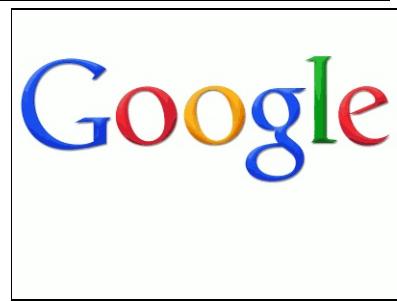
Spiderbots

Spiders

We use Google (and other search engines) to find things on the Web, but how does Google know where all of these pages are located and which ones are relevant to your queries?

Google uses virtual robots (called *spiders*) to index and explore the Web. Spiders aren't really robotic; they are simply small programs that follow a very simple routine:

1. Visit a series of web pages
2. Gather all of the links on each page visited
3. Add the links to its list of pages to visit in the future
4. Repeat until it has visited all pages in its list



Watch the video below to see how these spiders create Google's index of the web and affect your search results:



<https://www.youtube.com/embed/BNHR6IQJGZs>

Assignment

Now you will observe a spider in action! Download this Processing sketchbook for a toy spiderbot.

Click to Download: [SpiderBot](#)

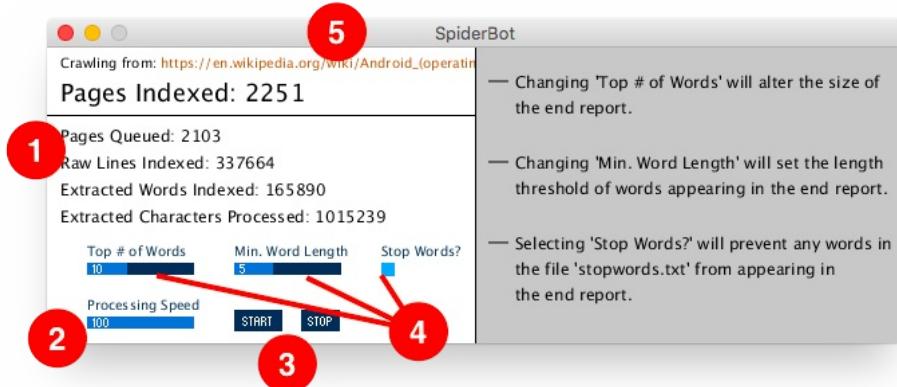
This is not a very robust program as it will only work with webpages that use simple HTML. But, it will allow you to see how a spider traverses the web, and it's very small. Unlike a search engine spider indexing all of the pages it encounters for later retrieval, SpiderBot just

counts — pages visited, number of words seen, etc.

All of the spidering work is done in the function named `processURL`. Have a look at the code, noting what a spiderbot *does*:

1. visit webpages,
2. gather all of the links on each page visited, and
3. add them to its list of pages to visit in the future.

When you execute `SpiderBot.pde`, you are presented with a small interface that details indexing statistics as the bot works, and a few controls to tailor the output. The following images explain each of the statistics reported by the SpiderBot program.



1: These are the dynamically generated statistics of the indexing process. *Pages Indexed* indicates how many pages you've processed. Once they are processed, the spider will not visit them again, even if another link to the page is found later. *Pages Queued* shows how many pages have been added to the worklist. This begins with just one page, and all other pages added are simply URLs found while spidering.

2: *Processing Speed* allows the user to slow down or speed up the process. Note that you can see the progress of your program in the Processing console.

3: *START* begins the spidering process; *STOP* stops the spidering process and presents the final report.

4: These controls fine-tune the final report. *Top # of Words* selects the number of highest frequency terms to report. *Min. Word Length* limits the terms reported to a certain length. In other words, setting the minimum length to two will cause all one-letter words to be skipped in the report. *Stop Words?* is a checkbox that will cause commonly used words to be skipped in the report. There is a file in the SpiderBot document folder called *stopwords.txt* that contains this list. You may add to or remove from it if you like.

5: The spider begins at this URL. You may alter the beginning URL in the Processing source by changing the value of `START_HERE` at the top of the program. However, remember that many sites, particularly those with more advanced features, will not work with

this simple program.

The screenshot shows the SpiderBot application running in Processing 3.2.1. The top window is titled "SpiderBot | Processing 3.2.1". The source code window displays the following Java code:

```
1 /*****
2 * 
3 * Spiderbot (Breadth-First)
4 * 
5 * UTeach CSP
6 * Bradley Beth
7 * rev. 20161024
8 * 
9 *****/
```

The bottom window is titled "RESULTS". It shows the following output:

```
Crawling from: https://en.wikipedia.org/wiki/Android_(operating_system)

Pages Indexed: 1665
Raw Lines Processed: 121651
Extracted Words Indexed: 97306
Extracted Characters Processed: 592648
```

Below this, it says "Top 10 Word Frequencies (with minimum length 5)" and lists the following table:

#	Word	Frequency
1	android	1641
2	google	701
3	function	310
4	mobile	303
5	retrieved	284
6	device	276
7	devices	275
8	system	233
9	support	197
10	sensor	189

Annotations with red numbers are present: a red circle with the number 6 is over the statistics section; a red circle with the number 7 is over the word frequency table.

The final report is generated in the Processing console. You may click and drag the slider between the source code window and the console window to alter the height of the console.

6: This details a summary of the dynamic statistics generated during the web crawl.

7: A table of the highest frequency terms is generated when either *STOP* is pressed or the *Pages Queued* falls to zero. This report is configurable using the controls outlined in figure above.

Submission

You must submit a report (≥ 1 page) outlining the following:

- An introduction to the spidering process. Include a synopsis of how a spider gets its worklist of pages to index.
- A comparison of SpiderBot on two websites with similar subjects. Note that Wikipedia pages work well, because they are formatted fairly simply and include a lot of outgoing links. For example, [https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)) and <https://en.wikipedia.org/wiki/IOS>.
 - Include tables of high frequency terms, with different options selected (such as *Min. Word Length* and *Stop Words?*)
 - Include a summary for pages, lines, words, and characters indexed for each. To facilitate comparison, run each spider routine for the same amount of time at the same speed.
- A brief analysis of the comparison. Consider the following questions:
 - Is it easy to tell which lists of high frequency indexed terms belong to which site? How might adjusting the stop word list help you distinguish?
 - As the process progresses, note the URLs that are being crawled by the spider. Do they have the same domain name as the starting URL? How does this impact the report?

Robot or Not?

Seeing how it's so easy to write a simple program to autonomously crawl the Web and access even the deepest corners of the Internet without any human supervision, it raises the question of how much Web traffic is actual humans and how much is the work of these bots?

Fetching Flutter-bys

Butterflies

Astraptes fulgerator is the subject of a groundbreaking and controversial DNA barcoding study that discovered at least three subspecies within the butterfly species.

Examine this [relational database of individual tracked butterflies](#) organized using Google's Fusion Tables tool. Each tracked butterfly is labeled with certain attributes including: a unique "voucher" ID, an image, sex, the year tracked, etc. Although this database contains only a subset of 87 individual butterflies, you can imagine that it contains many more. The original study examined more than 2,500 caterpillars and 484 adult butterflies.



Because the butterflies are logged in a relational database, looking at certain subsets of populations is a simple task. If we want to find all of the female butterflies, we can apply a filter so that only butterflies who meet that criterion are displayed:

- sex = female

Or, if we'd like to view all of the butterflies tracked during or before 2003, we can create the filter:

- year ≤ 2003

Finally, we can apply both of the criteria, viewing only butterflies who are both female and tracked during or before 2003. This screenshot of the Google Fusion Table shows the results:

herbivore spe...	voucher	URL adult	URL cp lateral	sex	host plant species	host plant family	year
Astraptes SENNOV	03-SRNP-14687			female	Senna hayesiana	Fabaceae	2003
Astraptes FABOV	93-SRNP-7060			female	Centrosema macrocarpum	Fabaceae	1993
Astraptes INGCUP	02-SRNP-32205			female	Inga vera	Fabaceae	2002
Astraptes INGCUP	02-SRNP-19284			female	Cupania glabra	Sapindaceae	2002
Astraptes LOHAMP	03-SRNP-5544			female	Hampea appendiculata	Malvaceae	2003
Astraptes YESENN	00-SRNP-11188			female	Dioclea malacocarpa	Fabaceae	2000
Astraptes LOHAMP	03-SRNP-10879			female	Hampea appendiculata	Malvaceae	2003

Google Fusion Tables provides a nice graphical interface for this. The Structured Query Language (SQL) statement to pull this same data would be:

- SELECT * FROM Astraptes_fulgerator
 - WHERE sex = female
 - AND year ≤ 2003;

In English, this roughly translates to "Select everything in the Table named 'Astraptes_fulgerator' where the sex is female and the year tracked was in or before 2003."

Performing an SQL query on a relational database is much easier than sifting through all of the data by hand, isn't it?

Instructions

Working individually or in pairs, filter the butterflies in [Google Fusion Table](#) in order to find the following:

- Herbivore species: Astraptes SENNOV AND host plant species: Senna papillosa.
- Host plant family: Sapindaceae AND wingspan (mm): 59.
- Primary eco: rain forest AND elevation: <= 300 AND sex: male.

Perform the following tasks for each of the above filters:

1. Take a screenshot of the results.
2. Write/translate the code to the equivalent English sentence (like the example above).

With any remaining time, explore the data set with new search queries.

Submit

Submit a document that includes the three screenshots and the search queries *translated into English*.

UNIT TOPIC:

Data Aggregation

Storage

- You will explore the basic features and functionality of modern relational databases.
- You will debate the implications of large-scale data storage and data persistence on privacy and utility, including the costs associated with each.

Indexing Julius Caesar

Murder of Caesar

"Friends, Romans, countrymen, lend me your ears!" is perhaps the most famous line from Mark Antony's speech at the end of Shakespeare's *Julius Caesar*. To set the scene: Brutus and a group of other Roman senators have just murdered Julius Caesar in the middle of the Senate. Brutus then explains this egregious act to the gathering of Romans outside, and now Julius Caesar's friend Mark Antony (who did *not* participate in the murder) responds to what Brutus has said:



<https://www.youtube.com/embed/RJHDZc45xow>

In this assignment, you will be practicing the process of indexing using text from Mark Antony's famous speech from *Act III, Scene II* in [William Shakespeare's Julius Caesar](#). In pairs, you and your partner will build separate indexes for the document using Processing, looking for specific terms and phrases, and using them to speed up searches of the text.

First, download the following Processing program that you will use to read and mark text in the famous speech.

Click to Download: [IndexingJuliusCaesar](#)

The starter code contains the speech and a stop watch. Each time you click on a word, it will be highlighted in yellow. In pairs, examine the entire text and click to highlight each of the matching terms you find. A list of terms for each partner is found below. You will use each term, one at a time, and record your elapsed times.

Partner 1's Terms and Phrases	Partner 2's Terms and Phrases
Caesar	Brutus

he	him
noble	honorable
ambitious	ambition

Instructions

You will first search the text using **Algorithm A** and share the length of time that it takes you to search for each of the terms in your assignment submission. You will then repeat the search using **Algorithm B**, which provides a means of indexing the text in the file.

Before running the Processing program, edit the first two lines of code to initialize the search term and algorithm as follows:

```
String word = "Caesar";
String algorithm = "A";
```

Note that the value of `word` should be initialized to whichever search term you are looking for and `algorithm` should be initialized to either `A` or `B`, depending on which of the following algorithms you are testing.

Search for each of your assigned search terms using Algorithm A.

Search Algorithm A

- 1) start the timer
- 2) note your search term
- 3) repeat for each word in the original text...
- 4) ...if the word matches your search term...
- 5) ...click to mark the occurrence of the term
- 6) stop your timer
- 7) record the time

Now, you will perform the same task, only you are allowed to use an index that highlights each line in which your search term appears. Search for each of your assigned search terms using Algorithm B. **Don't forget to modify the `word` and `algorithm` values in the Processing program.**

Search Algorithm B

- 1) start the timer
- 2) note your search term
- 3) repeat for each red line in the original text...
- 4) ...repeat for each word on the highlighted line...
- 5) ...if the word matches your search term...
- 6) ...click to mark the occurrence of the term
- 7) stop the timer
- 8) record the time

Compare your results with your partner and discuss the differences between searching for a

keyword with and without an index that identifies the lines where the keyword can be found.

Assignment Submission

When you complete both indexing algorithms, submit a document containing the following information:

1. The recorded times for completing each of the searches.
2. Answers to the following questions:
 1. Which of the two methods was quicker? Why?
 2. Imagine that the text was expanded beyond the speech to all of the play. Which method would be better, and why? What difference would the length of the text make?

Data Persistence

What Happens Online, Stays Online

What happens to information that you delete online? Does it simply vanish never to be seen again? Probably not. **What happens online, stays online.** Computer scientists refer to this as the *persistence* of digital data.

As discussed in the Representation module, digital information is easy to copy. See a digital image you like on a website? Right click on it, choose "Save image as" and now you have possession of a digital copy of that image that was posted online. This strategy applies to everything posted on the web from images to Tweets, and even webpages.

Common misconception: *Information posted on a webpage can be truly "deleted."*

- Not really. Once a page is publicly available, it is potentially indexed by search engines, archived or cached by a service like the Wayback Machine, and downloaded/copied by an individual. Although you can then remove your original page, these other representations may exist independently.

For example, although McDonald's Corporation has changed its website many times in the past 16 years. Here is a copy of its 1996 homepage:

Data persistence can be very useful, and very annoying. Some information we would like to persist forever. For instance, if all of a sudden, Wikipedia went down, we hope that not all of its data would be lost (and the years of work put in by people to create it) because of data persistence. However, the embarrassing baby photo that your mom posted to Facebook of you? Data persistence might be annoying in that case.



Who Owns Your Data?

Herein lies a concern. When users sign up for certain online services like Facebook, Twitter, YouTube, etc., they are also turning over much of their digital data to the company controlling their account. Actually, this is, in essence, how and why these companies exist. If nobody posted information to Facebook, what would be the point of belonging? The service only has value because users give it data. However, once given, the service is free to use that data however it sees fit, as long as it is in accordance with its posted policies and user agreements.

Read the following text from [Facebook's Help Pages](#) about deactivating or deleting personal pages and think about the following two questions:

- Which option is more likely to retain your personal data?
- Does either option guarantee permanent deletion of everything you've created/posted?

Deactivating Accounts

What is the difference between deactivation and deletion?

If you deactivate your account:

- Your profile (timeline) disappears from the Facebook service immediately.
- People on Facebook will not be able to search for you. Some information, like messages you sent, may still be visible to others.
- We save your profile (timeline) information (friends, photos, interests, etc.), just in case you want to [come back](#) to Facebook at some point. If you choose to reactivate your account, the information on your profile (timeline) will be there when you come back.

A lot of people deactivate their accounts for temporary reasons and expect their profiles (timelines) to be there when they return to the service. This option gives you the flexibility to leave and come back whenever you want.

If you permanently delete your account:

- You will not be able to regain access to your account again.
- Most personally identifiable information associated with it is removed from our database. This includes information like your email address, mailing address, and IM screen name. Some personally identifiable information may remain, such as your name if you sent a message to someone else.

Copies of some material (photos, notes, etc.) may remain in our servers for technical reasons, but this material is disassociated from any personal identifiers and completely inaccessible to other people using Facebook.

The language of "Deactivation" states that (all) information is saved "just in case you want to come back to Facebook at some point." This indicates that Facebook retains all of your information, just sets some of it to be publicly inaccessible. Data persistence at work, and for good! Maybe you're just taking a vacation from Facebook and plan on coming back.

Awesome job, Facebook!

However, "Permanent Deletion" states that "most personally identifiable information" is removed and that material posted "may [be] retain[ed] in [their] servers for technical reasons." Although you have deleted your account, **Facebook still retains some portion of your data, such as your photos, friends list, and more.** By posting information to Facebook, you are turning over some semblance of ownership of that data to them. So, if users give up some ownership of their own data when they use or provide it, why do they still continue to do so?

Benefits of Data Persistence

Like so many other things, data persistence has pluses and minuses, including trade-offs in privacy and utility that all responsible Internet users must consider.

- Digital data is immune to [generation loss](#), as multiple identical digital copies of photos may exist at any given time. These digital copies can be copied further and be maintained by many entities.
- A service like Flickr typically maintains redundant backups of all its data. The data themselves (photos, video, etc.) can exist through these means even after the actual camera and/or storage device that originally captured them are destroyed.

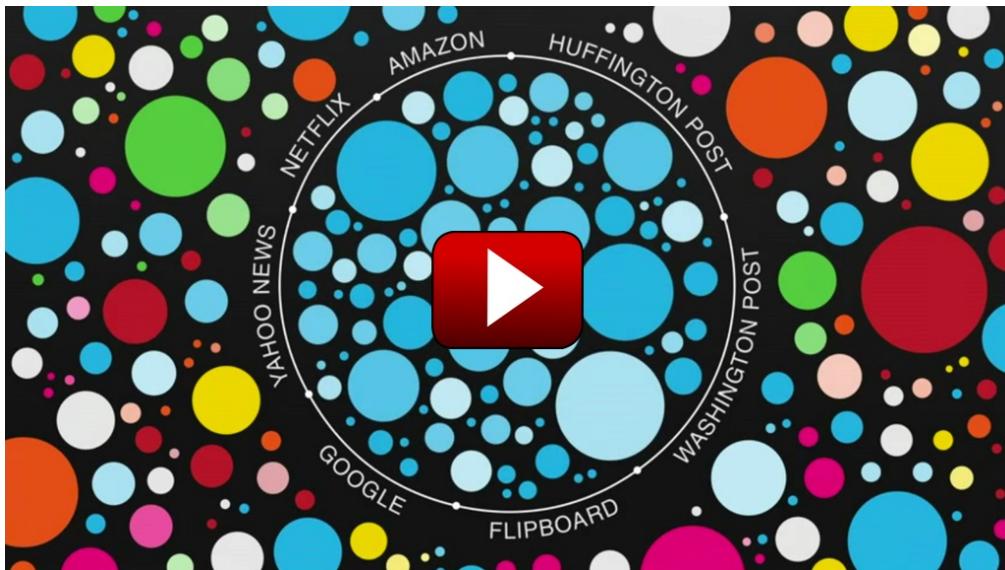
This means that we gain utility from posting online. By posting to Facebook or Flickr, not only are we able to share with our friends and the online communities, we are protecting this digital data from accidental deletion.

These benefits don't even begin to approach the social, scientific, and economic benefits gleaned from users providing data to entities and of data persistence. You will discuss these types benefits more during the Privacy vs. Utility debate.

Your Filter Bubble

The Filter Bubble

Most Internet users are living in a bubble right now...a filter bubble. Eli Pariser is creator of the term *filter bubble*, and he's given a fantastic TED talk explaining the concept of a filter bubble, and how the Internet is hiding important information from you by using your data to make your experience more personal. Watch and learn:



<https://www.youtube.com/embed/B8ofWFx525s>

Experience the bubble for yourself:

1. Complete a Google search for a term like **religion**, **Obama**, **Israel**, etc.
2. Compare your search results with your neighbors' results.
3. Compare your search results with a search using the "bubble-free" [Duck, Duck, Go](#) search engine.

Break Free

You can escape your bubble if you try. Visit these three websites to learn more about how to become freer on the Internet. These resources may inform your thoughts on privacy and data collection:

1. Learn more about how the filter bubble works from [Don't Bubble Us](#).
2. [Duck, Duck, Go](#) is a search engine that does NOT bubble you.
3. Read "[Are we stuck in filter bubbles? Here are five potential paths out](#)" by Jonathan Stray.



Privacy vs. Utility

Classroom Debate

Do you feel like you're being watched all the time? In a way, you are. Before the digitization of everything, data was stored physically:

- Medical histories were collections of paper documents maintained by your doctor/hospital.
- Photos were shot with film, and printed on photo paper.
- People sent handwritten letters or made phone calls instead of emailing or texting.



Life has become increasingly digital, and so has our data. Your personal data can be used to make life easier and better for you and others. The digitization of your personal data also means that your personal data is now easier to reproduce, share, and access — for both good and ill.

What are the implications of large-scale data storage on privacy and utility? Let's explore some examples to help you think about this question.

Privacy vs. Utility

Most of the time, we share our personal digital data because we receive something of value in return, which we call *utility*. This is give-and-take, and most decisions about divulging personal data are relative to their contexts.

For example, a lady might give her cell phone number to a gentleman, hoping for a future date. However, there is also the risk that he might just make prank phone calls or give her number to one of his friends. She trades some personal information in hopes of a desirable return.

Trading Privacy for Utility

Consumers are constantly being asked to give their personal data — purchase histories and spending habits — in order for their activities to be tracked and data stored. Why would consumers allow or even *welcome* this practice?

A) The monetary motivation is incentive to justify linking purchase history to the card holder/patron.

For example, many grocery stores, department stores, etc., only provide discounted prices to those who have signed up for their "Rewards" program. In these cases, you trade your data (purchasing history/spending habits) for reduced prices. In this case, your data has a

clear, monetary value to both you and the store. You must decide whether the value is great enough for you to allow the collection of your data.

B) Companies provide customized "special offers," "deals," or "recommendations" gleaned from analysis of personal data.

Again, there is a clear financial incentive to use personal data in these cases, though less quantifiable since not all deals end up being purchased.

These are two simple examples of why data collection can provide users with utility, but your digital data takes *many* other forms besides purchasing history (see [Blown to Bits, Chapter 2](#)). Here are some examples where the utility gained is not monetary:

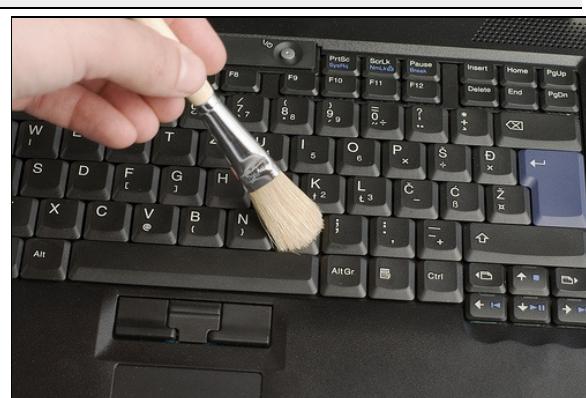
Finger/Footprint	Privacy Concern	Utility Potentially Gained
Public cameras	Individuals can be tracked	Crimes and emergencies can be tracked
Media metadata (e.g., EXIF headers)	Identifiable information can be obtained from photos	Cameras automatically tag location, date/time, camera setting information to pictures for further reference
GPS (Global Positioning System) embedded in phones	Individuals can be tracked real time	App support; usage in emergencies
RFID (Radio Frequency Identification) tags	Tracking of individuals, broadcast of data	Accessibility

Although a wide variety of data may impact your privacy and give you utility, the nature of digital data and the Internet means that your online information is perhaps the most easily accessible.

Browsing Histories

Internet browsers maintain a search/page history — again, this is your personal data (about where you have been on the Internet and what you have searched for). Why do browsers and search engines do this?

- Browsers maintain information for ease of use.
 - Recently visited pages can be



- recalled instantly without redownloading the information required to display them.
- Common searches can be anticipated through "autocomplete" functions so that users do not have to retype them each time.
 - Cookies allow preferences, passwords, and login information to be stored across browser sessions for convenience

The police can use this data, too. Computers seized in police investigations are often analyzed for evidence. For example, a bombing suspect whose computer usage include searches for the terms **pipebomb construction**, **household explosives**, **blueprints of BUILDING X** may find their search histories used as corroborating evidence during trial supporting the prosecution's claim that the suspect had **intent** for a bombing.

Debate: Topic and Instructions

In this activity, your class will hold a debate in which one side argues that **privacy is more important than utility in relation to personal data collection.**

1. Your teacher will randomly assign students to be either pro-privacy or pro-utility. You must argue your assigned points of view regardless of whether you personally agree with them. This is important in ensuring that the debate remains challenging and balanced.
2. Conduct an online search for more information and form arguments that support your assigned point of view. You will have 10-20 minutes to prepare.
3. Debate whether privacy or utility is more important in relation to personal data collection.



Your teacher will share the exact debate protocol with you before the debate begins. Good luck, and remember to remain open-minded and to listen to what the other side says! Will they be able to change your mind, or will you be able to change theirs?

BIG PICTURE:

Data Breaches

Highlights

- You will examine the security risks and responsibilities assumed by companies that collect and store sensitive personal data.
- You will examine the causes and impact of data breaches involving sensitive personal data.

Data Breaches

Is Your Personal Data at Risk?

As our digital world grows, more and more personal data is being collected and stored by schools, businesses, government agencies, research groups, online services, etc. While the efficiency of the Internet provides an excellent backbone for amassing and accessing this wealth of information, it also comes with an unfortunate drawback. Specifically, the potentially high value of that data and the ease of access that the Internet provides also serves to attract and motivate thieves, hackers, and other malicious parties that would like to gain access to this rich supply of personal data.

It has unfortunately become quite common to hear news stories about the latest hack or breach of security that has compromised millions of users' personal data. Click here to read about one such [breach at Target stores](#) in 2013.

My Data Rules

Control Your Data

You've learned about how your personal data is collected, extracted, stored, and analyzed, which has major implications for your privacy. Responsible digital citizens must be aware of data collection and make purposeful decisions weighing the potential costs and benefits and costs associated with personal data use. In this assignment, you will make decisions about how you will control *your* data:

- Will you provide data as often as possible to gain utility anywhere and everywhere?
- Will you never give your data to anybody to try and maintain privacy?
- Can you find a happy balance between privacy and utility? These are decisions that previous generations never had to consider, but you do.

Pledge to Use Your Data Responsibly!

Your job is to write a pledge called "My Data Rules" that outlines your plan to use or not use, and protect your personal data. You will have to write an expository essay in which you:

- Identify examples of personal data that can be collected.
- Identify strategies commonly used for collecting your personal data.
- Make connections between data persistence and your plan to use or not use your data.
- Weigh the costs and benefits of including your data in large scale data sets.

This pledge is purposeful, and should act as a guide for your future decisions regarding data use and storage. Pledge to always be knowledgeable and careful with your information.

Rubric

Criteria	Points
Privacy and Utility	
Summarizes and discusses the positive and negative implications of personal data storage on your privacy and utility	2 pts
Describes more than three potential costs and benefits of your personal data being part of large-scale data sets	3 pts
Data Collection	
Describes at least three data collection strategies used to collect personal data	3 pts
Lists at least 10 examples of personal data that can be collected by others	5 pts

Rules for Data Usage	
Explicitly describes rules that you will follow for allowing your data to be collected, stored, and analyzed	5 pts
Makes meaningful connections between privacy, utility and your rules	3 pts
Key terminology	
Uses key terminology appropriately and as necessary, including: data, data persistence, utility, and privacy	2 pts
TOTAL	23 pts

UNIT TOPIC:

Data Analysis

Statistical Analysis

- You will analyze the tradeoff of utility and confidence in descriptive, predictive, and prescriptive data analysis.
- You will investigate traditional statistical hypothesis testing and exploratory data analysis.

Data Mining

- You will investigate the use of data mining in the discovery of patterns in large data sets.
- You will apply association rule mining to discover knowledge in data sets.
- You will articulate the effects of association rule mining on business and education.

Clustering

- You will visually perform cluster analysis, modeling the dynamics of groups.

Anomaly Detection

- You will visually perform anomaly/outlier/change detection and discuss the impact on potential inferences drawn from the data.

Regression

- You will synthesize a prediction through linear regression over known data.

Classification and Summarization

- You will apply a classification protocol to a dataset and compare results with pre-defined categories.
- You will evaluate the effects of automated summarization on the utility and validity of inference.

UNIT TOPIC:

Data Analysis

Statistical Analysis

- You will analyze the tradeoff of utility and confidence in descriptive, predictive, and prescriptive data analysis.
- You will investigate traditional statistical hypothesis testing and exploratory data analysis.

Statistical Analysis

Survey Says!

Not all statistics or data analyses are meaningful. Just because somebody punctuates a fact with a statistic, does not make it meaningful (or *correct*). Take for instance, watch the video to the right that demonstrates a little statistical absurdity from the movie *Anchorman*.



<https://www.youtube.com/embed/pjvQFtlNQ-M>

Descriptive, Predictive, and Prescriptive Analytics

There are three *uses* of statistical analysis that are commonly used by scientists, mathematicians, politicians, and other professionals across the globe:

- **Descriptive** analytics provide information about collected data via statistics that you are probably familiar with — [mean](#), [median](#), [mode](#), [range](#), etc. They tend to "describe" circumstances, but don't offer conjectures about unknowns.
 - e.g., How many (in terms of percentage) computer science graduates are paid salaries of \$100,000 or more within five years of graduating?
- **Predictive** analytics may provide information about future (or merely unobserved or unknown) events based on previously collected and analyzed data.
 - e.g., How likely is that I will be able to find a high-paying job if I choose to major in computer science vs. biology?
- **Prescriptive** analytics may provide information to maximize the chances of a future event occurring, based on comparing the predictive analyses of multiple options
 - e.g., Which major should I choose in order to maximize my chances of making the highest starting salary after graduation?

These three types of analytics each have advantages and disadvantages that experts must evaluate to utilize them properly. These types of analysis each serve different purposes, but

they all allow for powerful inferences to be drawn from data.

Utility and Confidence of Drawing Inferences

The following report card grades each type of analysis on its utility (how useful is it?) and confidence (how likely is it to be true and/or valid?) in the context of decision making. Notice that there is no *perfect* analysis.

Analysis	Utility level	Confidence level
Descriptive	C	A+
Predictive	B+	B-
Prescriptive	A	C

Let's take a look at each type of analysis, its purpose, and how it is applied to common computing tasks (e.g., a Google search).

Descriptive Analytics

Function	Application
easiest to derive "hard facts" from	The spidering, caching, and indexing of the web is all centered on descriptive analytics
Example: the percentage of graduates employed within six months of graduating	In essence, this creates an easily accessible description of the web

Predictive Analytics

Function	Application
uses descriptive analysis' "hard facts" to extrapolate (make inferences) about where unknown data may lie	the retrieval and ranking of pages in response to a search query
Example: Given that 90 of the 100 CS graduates were employed within six months in 2011, it is ___ % likely that 108 of the 120 CS graduates in 2012 will be employed within six months.	creates a representation of the search terms and compares it against descriptive (indexed) model of the web
predictions are not "hard facts"; they may be wrong!	in essence, predicts what pages will be relevant to whom

Prescriptive Analytics

Function	Application
compiles predictive hypotheses and recommends a plan of action to	autocomplete is a form of

maximize the likelihood of something happening	prescriptive analysis
Example: which major should I choose to maximize the chance that I'll have a job within six months of graduation?	recommendations for further searches are based on potential next steps for users
confidence is the lowest here, because all of the prediction errors associated with previous analyses are compounded.	based on the ranking of previous/potential queries that are similar to the current search query

More Is Better

Generally, **more data leads to greater confidence**. Each of these are based on building models from data. The models' fit to the data increases their power (and thus, utility). This is why big data can be so powerful.

Google's searches are often effective, because their data set is *huge*. They have a ton of data from which to conduct descriptive, predictive, and prescriptive analyses, and then use those analyses to improve user experiences.

Common misconception: Autocomplete is programmed to give its results.

- Autocomplete is based on simple statistics of what people are actually searching for. In other words, if `how do I tie` results in `How do I tie a tie`, this is because statistically speaking, `How do I tie a tie` is the most common query that begins with `how do I tie`.

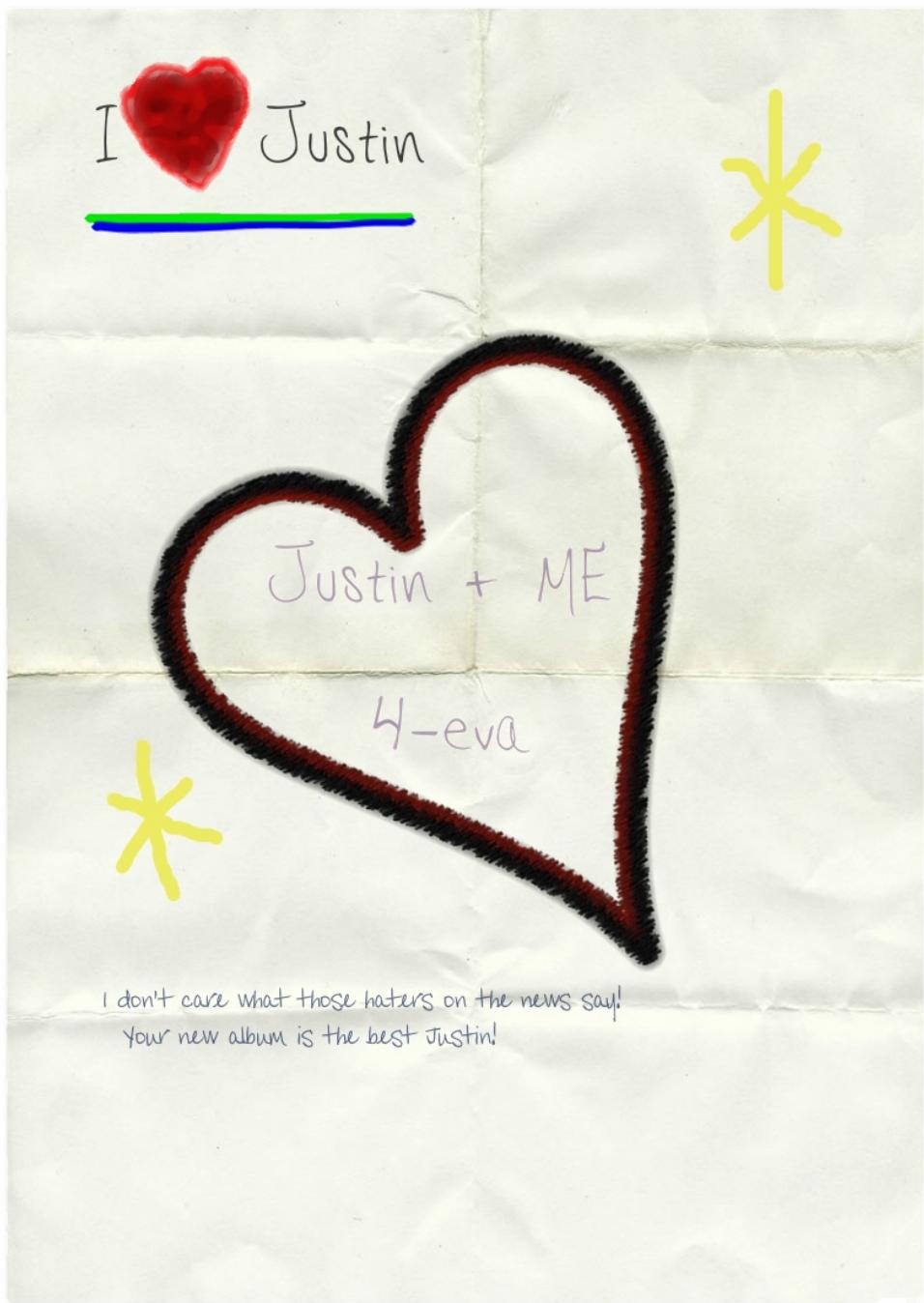
Justin Who?

Exploratory Data Analysis

Martindale High School has recently renovated and reopened its South Wing. Claudia is a new student assigned a locker in this new wing. When she excitedly opens her "new" locker, she discovers that the renovation crew seems to have neglected the actual insides of the lockers — fresh coat of paint in the outside; same old rusty, dented interior.

Even the previous occupant's end-of-year pile of trash is left intact. She begins to clean out the musty mess inside. "Nasty! I think this used to be an apple." Not wanting to touch it, she coaxes it into a wastebasket with a ruler. Underneath, she finds a folded piece of paper. "What's this?"

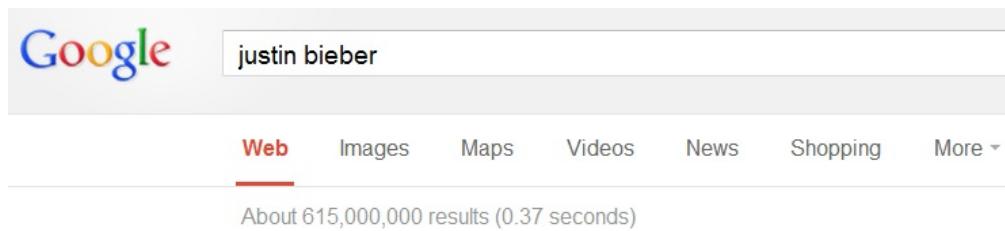
Carefully unfolding it, she finds a note left by the mysterious previous locker tenant:



"Hmm...who's Justin?"

You could formulate a hypothesis based on your intuition, such as "Of course, it's Justin Jones in homeroom — or — Justin Bieber," and call it a day. But wait a minute! Given the text of the note, it's obvious we have a public figure on our hands — someone who's been in the news. This means we have data — and lots of it. Now we can test our hypothesis. We can search for Justin in Google and see what comes up.

Wow! We get lots of hits for Justin Bieber:



Of course, we can't prove that the note is about him, but we can be fairly confident that it is, right?

Maybe we can add keywords such as "album" or "hate" or we can restrict our query to search news only. It seems he is the most likely "Justin" with an "album" in the news...

What do you think? Is it the Biebs, or somebody else? Conduct some exploratory data analysis and test your hypothesis:

Assignment: Investigate the Mysterious Justin

In your groups, investigate the meaning of this note using [Google Trends](#) and [Google Correlate](#) to conduct big data analysis. Google Trends is an online tool that allows users to analyze search-term frequencies across dates and geographic locations. For this assignment, you must submit a document that contains:

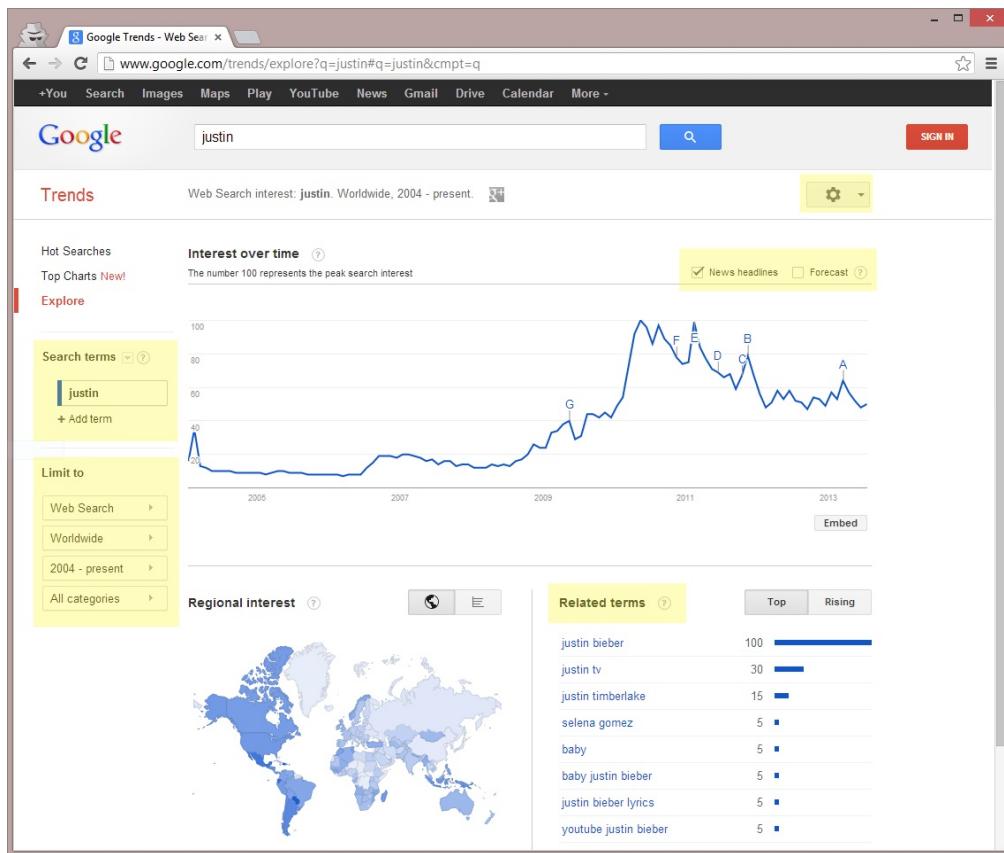
1. A hypothesis about the probable identities of "Justin."
2. A rationale that utilizes data analyses from [Google Trends](#) and [Google Correlate](#) to support your work.

To best support your work, you should take notes on the features you use and results you find. Screenshots of key findings may be useful.

Using Google Trends and Google Correlate

You should get to know Google Trends and Google Correlate. Google provides a tutorial on [Google Correlate](#), complete with screenshots and examples. Feel free to refer to it as necessary, or to ask your peers or teachers questions.

[Google Trends](#) is rather intuitive and easy to use. It can be very interesting, especially if you learn to set all the parameters, which are highlighted in the image below:



Explore the tools and use as much of the functionality as you can. You will be able to tell how search terms change over time, what terms are used together, from where people search, and even what news events correlate with trends over time.

Be Prepared to Discuss

- What did you do first? Next? Last?
- What would you like to be able to do?
- What was the most effective tool in Google Trends? Why?
- Was Google Correlate effective or efficient? Why or why not?
- How important was time/place/other key terms as search queries?
- Who is Justin? Justify your answer.
 - Will we ever know conclusively? Why or why not?

UNIT TOPIC:

Data Analysis

Data Mining

- You will investigate the use of data mining in the discovery of patterns in large data sets.
- You will apply association rule mining to discover knowledge in data sets.

Data Mining

Data Mining

Traditional ore mining begins with an exploration (prospecting) of a resource pool (stone), and proceeds to determining if usable resources exist (ore) and to what degree. Prospectors basically have an idea of what they are looking for, and they run small tests to see if they are correct. Sometimes they strike gold, other times they strike out. Like these physical mines that bring us everything from coal to diamonds, we have a new type of mining: data mining.



Data mining is akin to the discovery of patterns in large data sets. Like ore mining, data mining begins with an exploration (analysis) of a resource pool (data), and proceeds to determine whether usable resources exist (correlations) and to what degree (how strong they are). Not all data miners "strikes it rich." Like ore mining, data mining can result in the observation of no useful patterns. However, like ore mining, sometimes data mining leads to a bonanza of useful information.

In data mining, the emphasis is on the discovery of new knowledge. Data miners want to find new patterns that were previously unobserved. They use statistical analysis of big data to discover what the human eye can't see, just like an ore miner might use a pick, dynamite, or lab test to uncover ore that was not visible to the naked eye before. This is a form of exploratory data analysis rather than statistical hypothesis testing.

Data Mining Strategies

Data mining involves six common classes of tasks, listed below, along with examples of how these strategies can be used in recommender systems, such as those used by Netflix, Pandora, Amazon, <http://www.whatshouldireadnext.com/>, and many other content providers. In each of the descriptions below, a Netflix-related example of its usage is given:

- **Anomaly detection** (Outlier/change/deviation detection) — The identification of unusual data records, that might be interesting or simply data errors and require further investigation.
 - *Movie X* is unlike any of the other movies in User Y's data set. Remove it from our calculations. (example: *The Texas Chainsaw Massacre* is on a list that mostly contains titles such as Teletubbies, Barney and Friends, and Clifford.)
- **Association rule learning** (Dependency modeling) — Searches for relationships between variables. For example, a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing

purposes. This is sometimes referred to as market basket analysis.

- Recommender systems—Users who like *Movie X* tend to also like *Movie Y*.
- **Clustering** — is the task of discovering groups and structures in the data that are in some way or another "similar," without using known structures in the data.
 - Dynamically grouped movie categories: "Romantic Comedies in Paris starring former professional football players."
- **Classification** — is the task of generalizing known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as "legitimate" or as "spam."
 - *Movie X* is a romantic comedy.
- **Regression** — Attempts to find a function that models the data with the least error.
 - Type *X* users typically increase their movie consumption rate by four movies per year.
- **Summarization** — providing a more compact representation of the data set, including visualization and report generation.
 - What type of movie does User *X* typically like? (i.e., sum up user *X*'s preferences in *Y* words)

These strategies all have different purposes, are sometimes more effective on certain data sets and less on others, and oftentimes work best in conjunction with one other. Therefore, there is no one "best" way to perform data mining. Data miners use multiple strategies to uncover patterns and discover new knowledge.

Common misconception: *Data mining is often confused with Artificial Intelligence (AI).*

- Data mining is actually an application of techniques commonly associated with AI. "Machine learning" and "decision support" are standard AI techniques, but when we apply them to "knowledge discovery in databases," we refer to them collectively simply as "tools for data mining."

How much power lies in data mining? Read the following article to see "[How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did.](#)".

Association Rule Mining

Companies Know What You Buy

French toast is one of America's favorite breakfast foods. It's delicious and can be easily prepared at home using a variety of techniques and toppings. Even though it can be prepared a number of ways, almost all French toast recipes call for at least three things:

1. bread
2. milk
3. eggs



If you're going to make French toast, you're going to need bread, you're going to need milk, and you're going to need eggs. What does French toast have to do with big data?

Association Rule Mining

An [association rule](#) is a link between one set of items and another. Specifically, association rules identify instances in which the appearance of one set items (the antecedent) imply that another set of items (the consequent) will also appear.

For example:

$$\{X, Y\} \Rightarrow \{Z\}$$

This rule can be read as, “*If the antecedents (X and Y) appear then it is likely that the consequent (Z) will also appear.*”

By using association rules, we can group items together logically and attempt to make predictions. By tracking each of these transactions, tabulating them, and then discovering which pairs (or larger groups) of columns correlate often with one another, association rules may be generated to capture these correlations in the data. This applies to French toast preparation.

	...	beans	bread	eggs	milk	rice	syrup	...
Patron 1	...	✓	✗	✗	✓	✓	✗	...
Patron 2	...	✓	✓	✓	✓	✓	✓	...
Patron 3	...	✗	✗	✗	✗	✓	✗	...
...
Patron 9320	...	✗	✓	✓	✓	✗	✓	...
Patron 9321	...	✗	✓	✗	✗	✓	✗	...
...

For example:

- If most people who buy milk, bread, and eggs also buy maple syrup, then association rule mining might turn up the following rule:
 - $\{\text{milk, bread, eggs}\} \Rightarrow \{\text{syrup}\}$

Walmart can now target store patrons who purchase milk, bread, and eggs to gently suggest

474

that they might like to also buy syrup. The computerized storefront (or physical storefront with a layout determined by computational data mining) does not know that these patrons may be making French toast, they merely have developed association rules to guide product placement. The process of association rule mining is basically "[How Target Figured Out a Teen Girl was Pregnant...](#)"

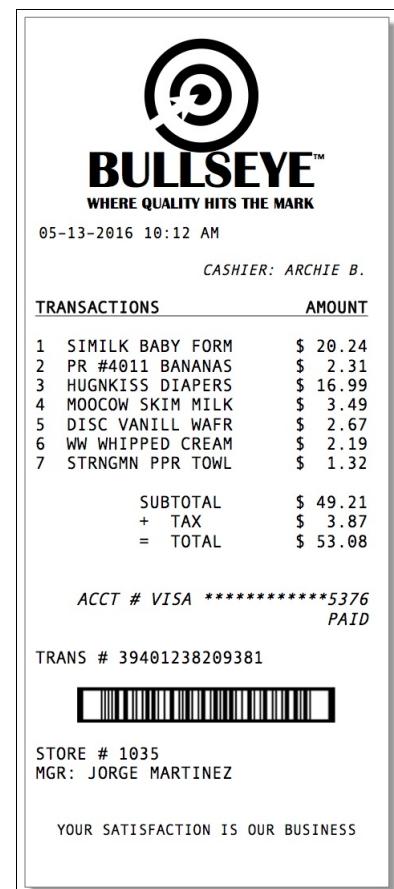
Instructions

Your group has been hired by *Data Market*, a corporation seeking to open a new chain of stores in your region. Their goal is to provide customers with optimal arrangements of store products, in an attempt to minimize the time and effort required to shop.

You will design a mock store product placement scheme—driven by data collection from competitors' stores in the area. Use the receipts provided by your teacher (1) to generate association rules that map potentially correlated products, and then (2) sketch an *endcap* for data-driven product placement targeting potential shoppers in the area.

As you extract data from the receipts, consider the following guiding questions:

1. What is the best way to use the provided table to organize your data collection?
2. What trends do you find in the data?
3. Are there any negative associations between products?
4. What is the ideal size for sets of antecedents/consequents?
5. What additional information might be helpful?
6. Can you imagine scenarios in which sets of products are grouped together?



UNIT TOPIC:

Data Analysis

Clustering

- You will visually perform cluster analysis, modeling the dynamics of groups.

TEDxKinda Project: Identifying Clusters



Instructions

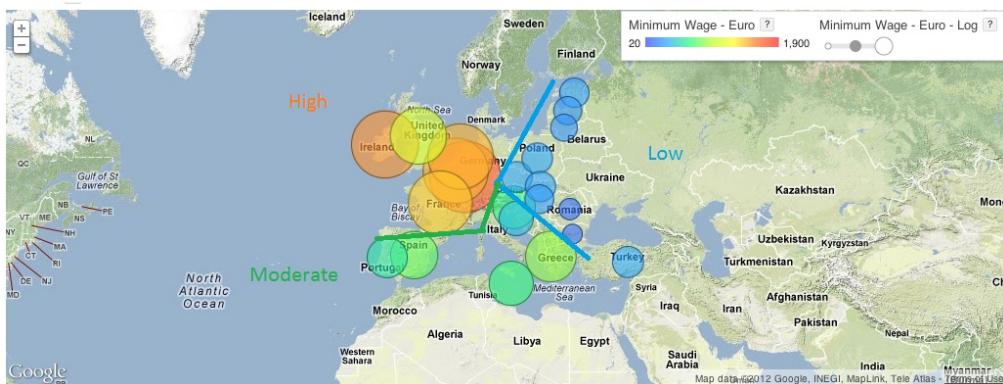
Your challenge is to cluster data in different ways. Using [Google's Public Data Explorer](#), explore a few data sets, and attempt to cluster data within the sets. If you have chosen data sets other than those in Google's collection, you will have to use one of the other [Tools for Big Data Analysis](#), which may be more challenging, but also much more fruitful for your research.

First, take a [screenshot](#) of a data visualization that demonstrates clusters of patterns, and then draw lines on the screenshot that bisect the plane and create clusters. You should cluster your data in two or more different ways, so use two different datasets if one dataset cannot be clustered in two ways. Choose datasets that are related to your TEDxKinda topic, so that you can apply your work to your end presentation.

Submission

Submit a document (e.g., .doc or .pdf) that includes the following items:

1. A proper heading (including names, date, assignment, and title)
2. The images of clustering (see example below)
3. An explanation of how and why you clustered the data
4. A hypothesis detailing what phenomenon the data clusters represent.



UNIT TOPIC:

Data Analysis

Anomaly Detection

- You will visually perform anomaly/outlier/change detection and discuss the impact on potential inferences drawn from the data.

Outliers

Outliers and Credit Card Fraud Detection

Credit card fraud has become more rampant as data become digitally stored, but at the same time, credit card fraud prevention has also become more powerful — primarily because of effective **outlier detection** using Big Data processing techniques.

Although there are many strategies for credit card fraud detection, one of the simplest to explain is the use of outlier detection. In a nutshell, outlier detection is the automated process of discovering data points that do **not** match the patterns inherent in the data as a whole. For credit card companies, this means looking at a client's spending habits, identifying patterns (e.g., cluster analysis), and tracking when spending does not match a person's habits or patterns.



For instance, let's think about the following scenario:

Tara is a Texas high school student who just opened her first bank and credit card accounts one year ago, after receiving her first paycheck from Taco Cabana. Last week, Tara went online and purchased a Willie Nelson concert ticket at a ridiculously low price, \$5! Wow! Almost too good to be true — and it was. The next day, a representative from Tara's bank called to inform her that her credit card was used to purchase gasoline, burgers, and a dirt bike — in Minnesota. Terrified Tara told the teller that the charges were bogus! The bank and credit card company then began to deal with the credit card fraud. Tara was not happy.

How did the bank/credit card company know that Tara did not make those purchases? Outlier detection. In this fictitious scenario, Tara had never spent more than \$100 on her credit card for any single transaction, and no transaction had occurred outside the state of Texas. Both of these may have signaled to the company that this charge was an anomaly (an outlier).

Detecting Outliers Visually

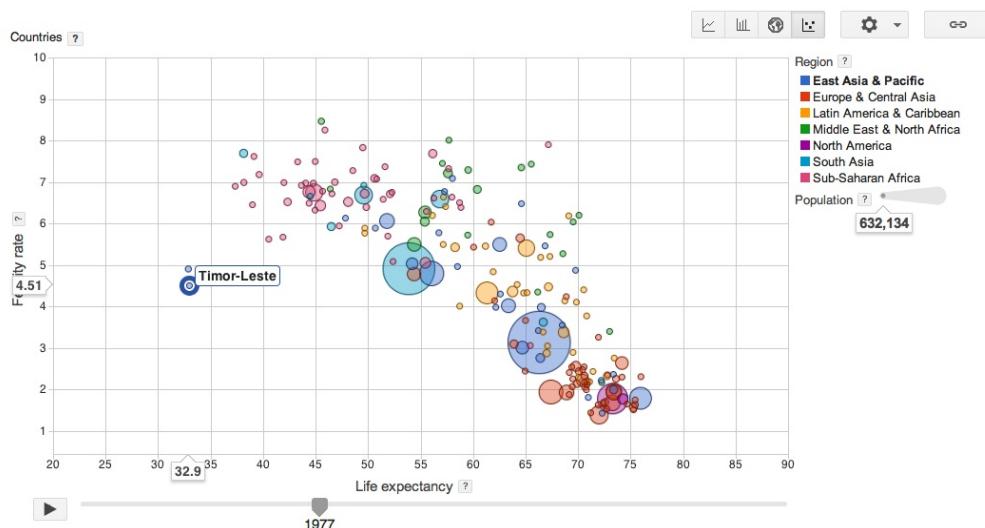
Using one of our big data analysis tools (Google's Public Data Explorer) we can visually identify an anomaly in a dynamic visualization. From there, we can begin to hypothesize a cause for the anomaly.

Navigate to the [Fertility Rate vs. Life Expectancy visualization](#) in Google Public Data Explorer to investigate an anomaly. There are many unusual trends in the data visualization, however, let's focus on one — the case of East Timor (a.k.a., Timor-Leste, identified in the figure below).

The visualization details data along five attributes:

- Life Expectancy — How long individuals in this country live on average. This is indicated along the x-axis.
- Fertility Rate — The average number of times the average woman in this country gives birth. This is indicated along the y-axis.
- Nation — This is the label applied to each bubble. Click on the bubble to view it.
- Region — The colors associated with each bubble indicate what part of the world in which they are located.
- Population — The size of each bubble is proportional to the population of the country represented. This is calculated yearly.

Note that actual values for these data can be obtained by hovering over the data point you wish to inspect. The x- and y-axis values appear along their respective axes, and the population appears in the legend.



East Timor is obviously an outlier in the static image above. It is clearly set apart from the rest of the clustered data. As the graphic indicates, life expectancy was a staggering 32.9 years of age, though fertility rates were average in comparison to the rest world!

However, as can be seen in the dynamic visualization, the true nature of the decline in life expectancy is drastic, taking place over the 1970s. What happened in the 1970s in East Timor? The identification of East Timor as an anomaly might indicate that some variable majorly affected its life expectancy during this time.

That variable, it turns out, was a civil war that broke out between East Timorese political parties, leading to an invasion and eventual occupation by Indonesia. The occupation was marked by extreme violence and brutality and resulted in more than 100,000 deaths, with approximately 20% of those resulting from killings and 80% from hunger and illness. This excessively high number of conflict-related deaths artificially skewed the mortality rate in the region during the time immediately after and during the occupation, as can be seen in the visualization.

Although the outlier in the static graph for 1977 establishes East Timor as a troubled area, the narrative expressed through the visualization not only gives context, but also indicates

that the formation of the outlier was itself an anomaly.

Outliers tend to be meaningful and tell stories about unique circumstances. Identifying outliers and investigating them can make descriptive statistics more accurate and also tell a story on their own.

TEDxKinda Project: Identifying Outliers



Instructions

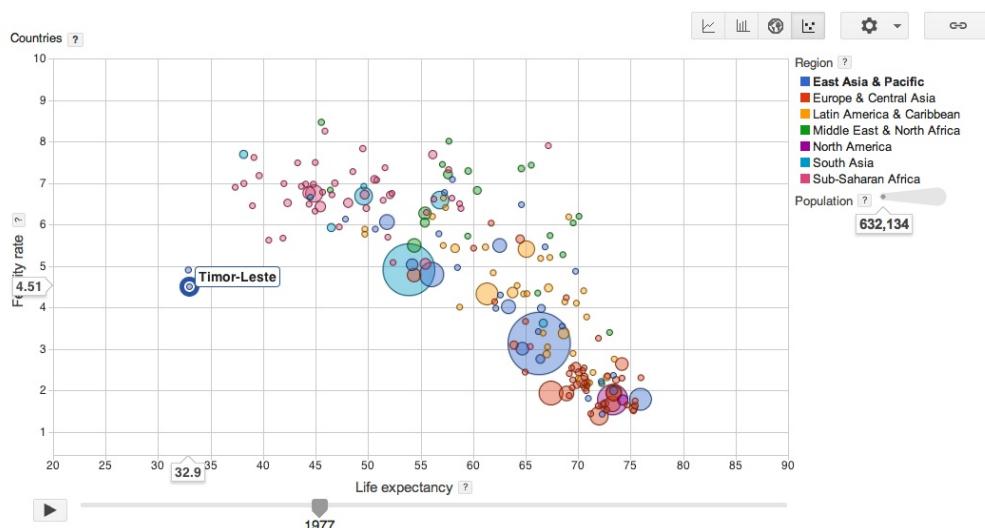
Your challenge is to identify anomalies in data sets using [Google's Public Data Explorer](#). Explore a few data sets, and attempt to identify anomalies within them. If you choose data sets other than those in Google's collection, you will have to use one of the other Tools for Big Data Analysis, which may be more challenging.

These anomalies may distort descriptive data or may tell stories about unique situations. Alter all variables involved in the data sets to see if any single variable contains outliers. When you identify an outlier, take a [screenshot](#) of the complete data visualization that illustrates the outlier(s). Be sure to examine data sets that are related to your TEDxKinda topic, so that you can apply your work to your ultimate presentation.

Submission

Submit a document (e.g., .doc or .pdf) that includes the following items:

1. A proper heading (including names, date, assignment, and title)
2. Images of an outlier (see example below)
3. An explanation of how you discovered the outlier
4. A hypothesis detailing what phenomenon the outlier represents



UNIT TOPIC:

Data Analysis

Regression

- You will synthesize a prediction through linear regression over known data.

TEDxKinda Project: Making Predictions



Instructions

Your challenge is to make predictions with regression, using one of the following tools:

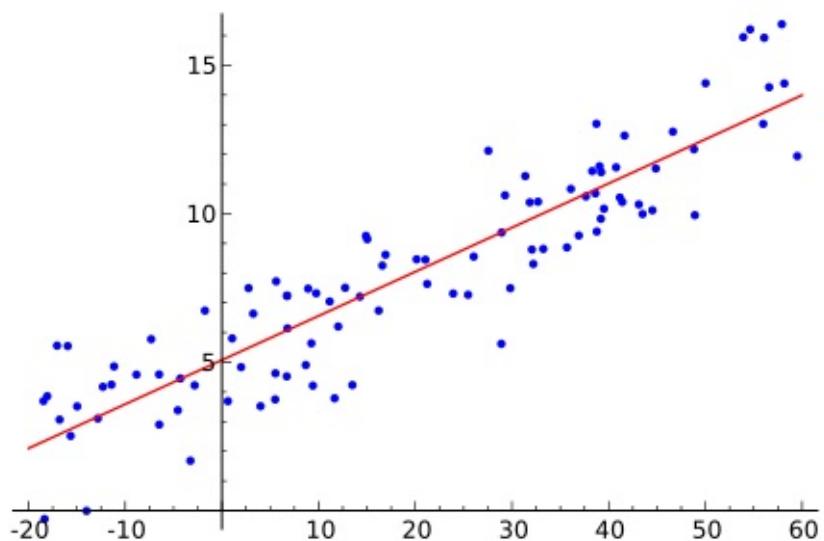
- [Microsoft Excel](#) - spreadsheet software
 - Video tutorial - [Doing Regression in Excel](#)
- [OpenOffice](#) or [LibreOffice](#) - free office suites (including spreadsheet software) for download
- TI-graphing calculators (e.g. TI Nspire, TI 84, etc.)
 - Video tutorial - [How to use TI-graphing calculators to perform regressions](#)
- [Processing](#) (*very challenging!)

Explore a few data sets, and generate regressions with them. These regressions may allow you to make predictions or identify patterns of change over time. When you perform an effective and informative regression, take a [screenshot](#) of the data visualization. Be sure to examine data sets that are related to your TEDxKinda topic, so that you can apply your work to your ultimate presentation.

Submission

Submit a document (e.g., .doc or .pdf) that includes the following items:

1. A proper heading (including names, date, assignment, and title)
2. A data visualization of the regression (see example below)
3. An explanation of how you performed the regression
4. A hypothesis explaining the trend in the regression (i.e., *What's the story?*)



UNIT TOPIC:

Data Analysis

Classification and Summarization

- You will apply a classification protocol to a dataset and compare results with pre-defined categories.
- You will evaluate the effects of automated summarization on the utility and validity of inference.

Classify Me

Classify Me: I've Got Personality!

We've all got personality. Some of us are more outgoing and flamboyant, while others of us are more reserved and introspective. Is it possible to classify a person's personality traits using a simple assessment (test)?

Isabel Briggs Myers and her mother Katharine Briggs developed an assessment to do just that: identify and describe personalities. To help classify personalities, the pair thought of four different categories that they believed could identify complete personality types:



- **Favorite world**
 - Do you prefer to focus on the outer world or on your own inner world? This is called [Extraversion \(E\) or Introversion \(I\)](#).
- **Information**
 - Do you prefer to focus on the basic information you take in, or do you prefer to interpret and add meaning? This is called [Sensing \(S\) or Intuition \(N\)](#).
- **Decisions**
 - When making decisions, do you prefer to first look at logic and consistency or first look at the people and special circumstances? This is called [Thinking \(T\) or Feeling \(F\)](#).
- **Structure**
 - In dealing with the outside world, do you prefer to get things decided or do you prefer to stay open to new information and options? This is called [Judging \(J\) or Perceiving \(P\)](#).

When you decide your preferences for each of the categories, you can put the letters together and compile your personality type into one of the following 16 personality types:

		Thinking		Feeling	
		Judging	Perceiving	Judging	Perceiving
Introversion	Sensing	ISTJ	ISTP	ISFJ	ISFP
	Intuition	INTJ	INTP	INFJ	INFP
Extroversion	Sensing	ESTJ	ESTP	ESFJ	ESFP
	Intuition	ENTJ	ENTP	ENFJ	ENFP

Learn more about each personality type from the [Myers-Briggs Personality Types - Basics](#) webpage.

Instructions: Classify Yourself

The Myers-Briggs Type Indicator isn't perfect. Its validity and reliability may be low, and the results don't correlate well with future success. Still, the MBI is an interesting assessment, in that it attempts to classify data (people) into personality types. To assess which personality type best fits you, try it out!

1. Navigate to this [Jung personality test](#) (similar to the original Myers-Briggs instrument, but free!), and complete it.
2. Write a reflection about the results:
 1. What do you think about the process?
 2. Do the results accurately predict your personality?
3. Post your reflection to a shared space provided to you by your teacher.
4. Read some of your classmates' posts, and respond.

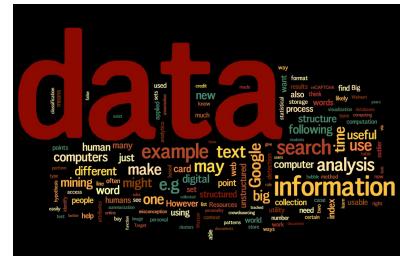
TEDxKinda Project: Automated Summarization



Word Clouds

Have you seen any **word clouds** floating around the Internet? For example, the Wordle image to the right is a word cloud generated using the text of all the webpages included in the Big Data module.

Some of these word clouds come from Wordle:



"...a toy for generating 'word clouds' from text that you provide. The clouds give greater prominence to words that appear more frequently in the source text. You can tweak your clouds with different fonts, layouts, and color schemes. The images you create with Wordle are yours to use however you like. You can print them out, or save them to the Wordle gallery to share with your friends." — [Wordle](#)

Wordles are an attempt to automatically summarize a large amount of data visually. It relies on word counts to identify what might be important ideas, however there are a variety of strategies that one can use in order to *automate summarization*, and each strategy has advantages and disadvantages related to its utility and its validity.

Automated Summarization

Humans are always looking for shortcuts, time-savers, and ways to make their lives easier. This includes attempts to read and comprehend long texts. Perhaps you have used Cliff's Notes instead of reading a novel for an English literature class. Still, some person (maybe Cliff?) had to read the original book and create the summary.

Automated summarization, in which a computer program produces a shortened version of a selected body of text, sounds like a dream come true for every time-crunched high school student assigned Tolstoy's *War and Peace*. This may not be feasible now, but automated summarization strategies are already widespread in our culture.

For example:

- A student's grade or score on a test is meant to summarize the student's understanding of the course material or individual concepts measured on a test.
- A credit score is a numeric value associated with a consumer's credit risk. Missed payments, excessive credit checks, and amount of outstanding debt all contribute to a final numeric FICO score in the range of 350 to 800.

Automated summaries like these can be both *usable* and *useful* — except that summarization of data comes at a cost. Automated summarization is lossy. Summarization attempts to reduce complexity by removing redundant or otherwise less significant details. Effectively, this is a form of **dimension reduction**.

However, these details can not be recovered given just the summary. The process maps the complexity of large sets of data to a simpler, smaller data pool. To illustrate, it is impossible to determine which test items were missed on an exam by merely knowing the student's total score, or to understand the finer points of this Big Data unit by examining the Wordle above.

Three Methods for Text Summarization

Let's explore three different ways in which to summarize text algorithmically. Note that we are trying to create a process that takes a text and generates a paraphrased summary of what the text is about. However, at no point do we actually take into consideration the meaning of any of the words. How is this possible?

When text is written, it has an inherent structure. Sentences are made up according to certain rules like **SUBJECT**

precedes VERB precedes OBJECT. Of course, other languages, and other creatures — like *Yoda* — may have different rules.



Three common techniques for automated text summarization are outlined below: *highest word frequency*, *TF*IDF*, and *topic sentence concatenation*.

The Highest Word Frequency method

This is the most intuitive of the algorithms, and the one that is used by Wordle:

- Parse the text and keep a count of all the words read separately.
- Sort the list so that the most frequent words are first.
- Remove words from a [stop word](#) list.
 - Sample ["stop word" list](#)
- Use the X (such as 10 — 20) most frequent words as a summary.

Think about the following: What is the point of the stop word list? How could this summarization method be exploited by spammers?

The TF*IDF method

The TF*IDF method extends the previous one by adding an assumption that *only uncommon words* are *useful* in a summary:

The words that appear in *all* documents are not useful in differentiating among them, so you begin by finding the most common words and eliminate them. This is like using a stop word list, only you make the list dynamically as you go rather than depending on a pre-determined master list.

- Calculate the *TF (term frequency)* for each word. Basically, this means taking each of the word counts you generated in the previous method and dividing them by the total number of words in the text:

$$tf = \frac{\text{total instances of this word}}{\text{number of all words used}}$$

- Calculate the *IDF (inverse document frequency)* for each word. This means figuring out how many documents out of all the ones you are processing contain the word. As an example, you might leverage Google to get these counts. Search the term using "quotation marks." Google will return the number of documents it retrieved containing exactly that term. Now we just divide all the English documents Google has indexed total, by the number of documents it indexed with that term in order to find the *IDF*.
 - Of course, we don't know what that total number of documents that Google has indexed total, so let's cheat. Assume that every document contains the word "the." Use the document count of the word "**the**" as the total document count. A Google search returned 10,690,000,000 documents when this page was created (March 29, 2014).

$$idf = \frac{\text{total number of documents}}{\text{number of documents with the chosen term}}$$

- Calculate the *TF*IDF* for each word using this formula:

$$tf * idf = tf \times \log idf$$

- Sort the list so that the highest *TF*IDF* words are first.
- Use the top X (such as 10–20) highest ranked *TF*IDF* words as a summary.

Think about the following: How might this compare with the highest word frequency method? What's the major difference?

The Topic Sentence Concatenation method

This method assumes that the main idea of each paragraph is the first sentence, so the topic sentences of each paragraph can encapsulate the meaning of the whole document. For this method, beginning at the top of the document:

1. Scan the first sentence and add it to your summary.
2. Skip remaining text until you reach either the next paragraph or the end of the document.
3. If you reach the next paragraph, repeat steps 1 & 2.
4. If you reach the end, your summary is complete.

Think about the following: How is this method different than the others? How does it fare if you limit the word count to X as you did with the other methods?

Instructions: Utilize Automated Summarization for your TEDxKinda Presentation

Your challenge is to apply one of the automated summarization strategies (i.e., Word Cloud, Highest Word Frequency, TF*IDF, or Topic Sentence Concatenation) to a text (or combine multiple texts for even more data), considering which summarization strategy might be the most useful. You may use automated summarization Tools for Big Data Analysis, which may be more challenging, but also much more fruitful for your research. Be sure to examine texts that are related to your TEDxKinda topic, so that you can apply your work to your end presentation.

Submission

Submit a document (e.g., .doc or .pdf) that includes the following items:

1. A proper heading (including names, date, assignment, and title),
2. The automated summarization,
3. An explanation detailing why you chose the method you did.

UNIT TOPIC:

Data Visualization

Data Visualization

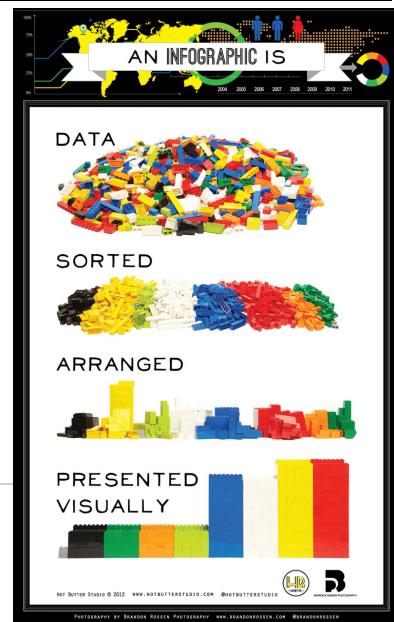
- You will combine visuals, content knowledge, and interaction to create a dynamic infographic that clearly communicates discrete information about a data set.

Interactive Infographics

Big Data Reflection

In this course — and no doubt in life — you have encountered many infographics, like the one to the right. They have become a part of our culture, our digital *zeitgeist*. What exactly is an infographic though?

Infographics are visual tools that allow information, data, or knowledge to be quickly and easily communicated through a single image. Furthermore, information designer, Dave Gray [extrapolates](#) upon the purpose of an infographic, listing six key features:



1. It is a visual explanation that helps you more easily understand, find or do something.
2. It is visual, and when necessary, integrates words and pictures in a fluid, dynamic way.
3. It stands alone and is completely self-explanatory.
4. It reveals information that was formerly hidden or submerged.
5. It makes possible faster, more consistent understanding.
6. It is universally understandable.

Source: [Communication Nation](#), Dave Gray

Instructions

Your job is to program an interactive infographic about your TEDxKinda presentation. An interactive infographic allows users to dynamically manipulate the data in order to engage users and illustrate major concepts. You may *program* the interactive infographic with Processing or Scratch, but there are many other tools specifically designed for creating interactive infographics.

How to Make an Interactive Infographic

First, design a static infographic to which you can then add interactivity. You might want to brainstorm design ideas with interactivity in mind, but one step at a time. For your infographic, you will need the following [three components of all infographics](#):

- Visuals (e.g., color coding, graphics, reference icons, etc.)
- Content (e.g., time frames, statistics, references, charts)
- Knowledge (e.g., inferences, connections, explanations, deductions, predictions)

The design of your infographic will likely not represent all of the work you have put in for the TEDxKinda Project, so select the most relevant points, focusing on quality over quantity.

The following tools and resources may help you obtain the visuals and content of a static infographic. Once you have designed a static infographic prototype, make it interactive so that users can manipulate the data or format of the infographic.

Creating charts and other components

- Excel/Numbers/[Google Docs](#)
- [Wordle](#)/[Tagxedo](#)
- [Many Eyes](#)

Pulling components together

- Word/Pages
- Powerpoint/Keynote
- Photoshop
- ComicLife
- [Prezi](#)
- [Visualize](#) (iOS app)

Find clipart, photos, and images

- Microsoft Office Clipart Gallery
- [Photopin.com](#)
- [Stock.xchng](#)
- [Open ClipArt Library](#)
- [Clker](#)

Create entire infographics

- [Easily](#)
- [Piktochart](#)
- [Infogram](#)

Explore these [10 Wonderful Examples of Interactive Infographics](#) for inspiration. Think about how you may apply the same strategies as they have, but with your topic, your knowledge, and your data.

BIG PICTURE:

Wisdom of the Crowd

Highlights

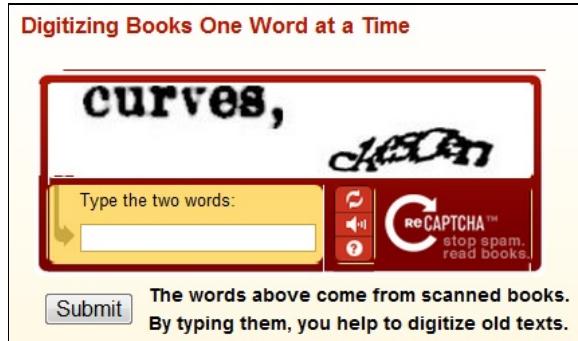
- You will apply the technique of crowdsourcing to a novel data collection problem.

ReCAPTCHA

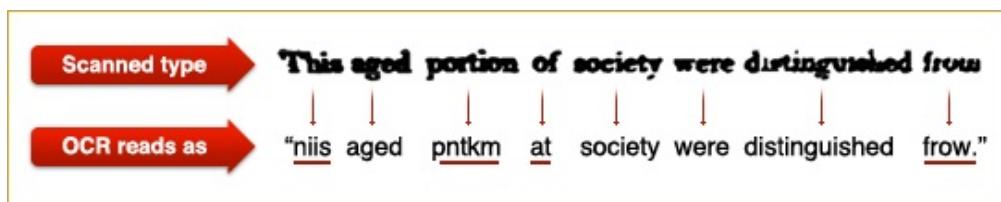
Humans Are Computers, Too!

The image you see to the right is an example of reCAPTCHA (revision of Completely Automated Public Turing test to tell Computers and Humans Apart).

reCAPTCHA is a digital tool used to deter automated form-filling and exploitation of web-based registration systems. Computer programs ('bots) sometimes traverse the Internet to try and complete these forms quickly to generate false reviews, spam message boards with links, or generate [pay-per-click](#) revenue. However, computers have trouble translating those distorted images of text to actual text, while it is (relatively) simple for humans to do so. reCAPTCHA exploits this difference to check if form fillers are likely human.



The inventor of reCAPTCHA, a computer scientist named [Luis von Ahn](#), noted that the computer's difficulty deciphering text is also a roadblock to progress in a different arena — the scanning, recording, and cataloging of pre-digital era books. He decided to crowdsource some human computers to help out. Look at this example to see how he applied reCAPTCHA to the scanning of books:



By simply altering the origin of the problematic data (fuzzy text) from computer-generated images to alphanumeric text typed by human beings, the reCAPTCHA problem becomes doubly useful. Not only are you demonstrating your humanity to the system, you are also utilizing your human computing skills to help digitize the world's books for generations to come. Good for you!

What Is Human Computation?

The word compute means to calculate or reckon, meaning a *computer* is simply something that *calculates* or *reckons* information. Humans have been computers for thousands of years; evidence of counting dates back to 30,000 B.C.E. Only in the past century have we defined computer as a digital/mechanical device that calculates for us.

Most of the time, digital computers work for humans. However, sometimes humans work for digital computers to help with tasks that digital computers are not currently capable of performing. A technique known as *human-based computation* leverages the ability of

humans to process certain types of information more efficiently than machines by outsourcing certain tasks to humans. Together, man and machine are able to solve complex tasks faster and more economically than either could do on their own.

Some tasks that humans can perform remarkably well through specialized brain/neural functions are not well-suited to digital computers (yet!). For example, humans are much better at face recognition and language translation (though state-of-the-art algorithms are becoming more competitive all the time). On the other hand, computers exceed the capabilities of humans in rote data collection, organization, and in some cases, interpretation. As such, *human computation leverages the abilities of both humans and computers to complete complex tasks*. The analysis of tasks performed well or quickly by humans is accelerated or automated by computers, and vice versa.

A great example of human computation is the use of reCAPTCHA to digitize physical texts, it's not the only one. For instance, Luis von Ahn has also developed [Duolingo](#), an digital tool to translate the World Wide Web into different languages by leveraging human computation and translation skills as language learning opportunity for those who participate. Watch the following TEDx talk in which Luis explains how human computation is leveraged with both reCAPTCHA and Duolingo:



<https://www.youtube.com/embed/cQI6jUjFjp4>

Not all crowdsourcing efforts are entirely "noble." Many simply want to leverage human computing on a large scale to accomplish tasks that may be more difficult for a small amount of disconnected people to complete today. Listed below are few of these very popular crowdsourcing efforts online today. How might they inspire your own crowdsourcing efforts?

- [Mechanical Turk](#): "We give businesses and developers access to an on-demand, scalable workforce. Workers select from thousands of tasks and work whenever it's convenient."
- [Kickstarter](#): "Kickstarter is the world's largest funding platform for creative projects. Every week, tens of thousands of amazing people pledge millions of dollars to projects

from the worlds of music, film, art, technology, design, food, publishing and other creative fields."

- [Fold.it](#): "Foldit is a revolutionary new computer game enabling you to contribute to important scientific research."

Crowdsourcing

Collecting Data

Crowdsourcing techniques do not always have to be digital. Before the Internet, it may have been more difficult to collect data from a large number of people, but it was not impossible.

In fact, Francis Galton used crowdsourcing in 1906 to accurately predict the weight of a bull. While visiting a livestock fair in 1906, Galton observed a contest in which participants attempted to guess the weight of a particular ox that was on display. Out of the nearly 800 guesses made, nobody accurately estimated the exact weight of 1,198 pounds. Some guesses were too low, while others were too high. Galton surveyed the range of guesses and noted that out of the nearly 800 guesses, the mean (average) prediction was 1,197 pounds! While no single individual was able to make an accurate guess, the crowd, as a whole, was surprisingly accurate.

This "wisdom of the crowd" phenomenon is not restricted to the weight of bulls, however. It can be seen over and over again anytime a sufficiently large sample size of individuals are asked to estimate an unknown result and is the basis for the use of *crowdsourcing*.

Guess The Number of Jelly Beans

Let's try this same experiment with virtual jelly beans.

We can crowdsource an estimated quantity using Google Forms and Spreadsheets to help gather a large number of responses. Make your best estimates and submit them to the collection. Then, examine the crowdsourced collection of estimates, and see how accurate the average estimate is to the actual amount.

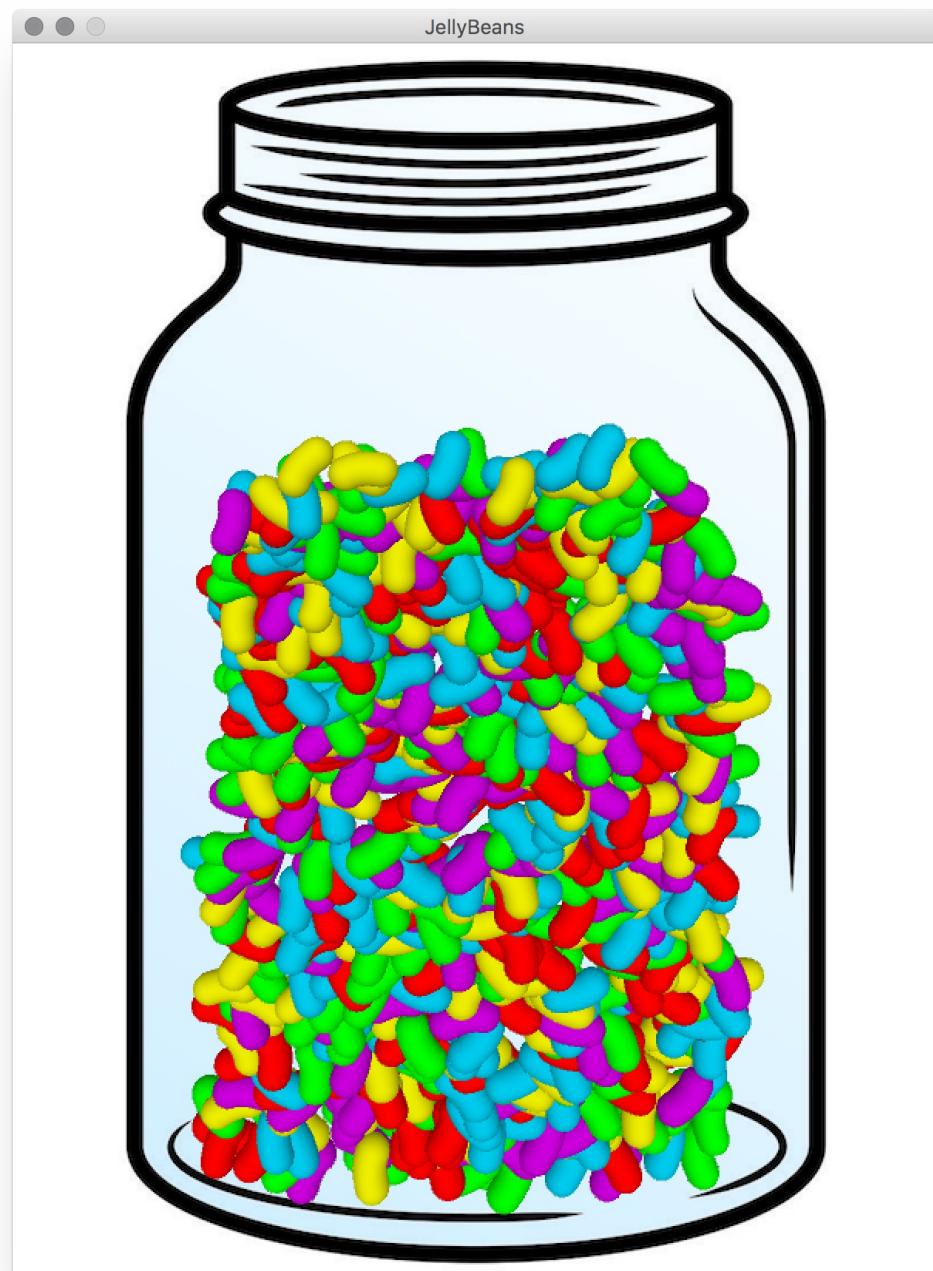


Jar of Jelly Beans

Briefly examine the following jar containing FIVE colors of jelly beans. Indicate your best guess for each of the questions that follow.

* Required

Guess the number of jelly beans



How many jelly beans are there in total? *

Enter a whole number.

Your answer

Please enter your guess below.

How many red jelly beans are there? *

Enter a whole number.

Your answer

How many green jelly beans are there? *

Enter a whole number.

Your answer

How many purple jelly beans are there? *

Enter a whole number.

Your answer

SUBMIT

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Additional Terms

UNIT PROJECT:

TEDxKinda

Highlights

- You will collaborate in groups to analyze public data sets and extract insightful information and new knowledge using a number of big data analysis techniques and tools.
- You will evaluate and justify the appropriateness of your chosen data set(s).
- You will construct informative and aesthetically pleasing data visualizations.
- You will write a script and prepare speaker notes for a formal presentation of your findings.
- You will cite all online and print sources used in your research and presentation preparation.
- You will deliver a TED-style presentation discussing your data analysis and findings using appropriate terminology.

TEDxKinda Project: Rubric Check



Feedback

This activity provides you and a partner group time to provide one another *critical feedback* about the progress of your projects.

1. Pair up with one another group.
2. Each group will present a mock presentation of their TEDxKinda talk. Any artifacts required for the presentation (e.g., infographics, graphs, etc.) should be presented as well.
3. Rate the components of your partner group's talk content and presentation according to the [TEDxKinda Project rubric](#). You may write this out or print the rubric and circle components on it.
4. Review your partner group's work and provide them with documentation that describes what you *Like*, what you *Wonder*, and what their *Next Steps* should be.
5. When both you and your partner group have completed the previous steps, share your feedback with one another.

Listen to what your partner group says! You do not need to heed all of their advice, but consider it wisely. The whole idea is to have a critical third party provide ideas to make your project better before you submit the final version for a grade.

UNIT 6

Innovative Technologies

As a way of further expanding upon the applications of computer science in the advancement of computational technologies, this unit aims to broaden students' awareness of the computing tools they use and rely on every day and to encourage them to start thinking about the decisions and processes that go into the creation of these technologies.

Students will begin by exploring many of the key roles that technology plays in their lives, including social networking, online communication, search, commerce, and news and examining the ways these ever-evolving technologies have impacted individuals and societies in recent years. With so many of these technologies relying on the Internet to connect users and data across varied and remote locations, the students will then "take a peek under the hood" to examine the systems and protocols that make up the global infrastructure of the Internet. Finally, students will turn their attention to the past, present, and future of computing to begin imagining the technology that might exist in their future and the role that they might play in bringing it about.

UNIT PROJECT:

Future Technology

Highlights

- You will collaborate in pairs to envision and design a future innovation in technology.
- You will discuss and identify a specific purpose that your innovation will serve (e.g., entertainment, problem solving, education, artistic expression, etc.) and its key features.
- You will evaluate the potential benefits and risks of your innovation.
- You will identify existing technological resources that your innovation may utilize.
- You will identify technological challenges that must be overcome before your innovation can be fully realized.
- You will develop a mock-up of your innovation that demonstrates its use and functionality.
- You will write a detailed product description and deliver an elevator pitch to the class detailing the features of your innovation and its potential impact on society using appropriate terminology.
- You will provide written feedback to your peers on the potential of each collaborative team's design.

Future Technology Project

"If I had asked people what they wanted, they would have said faster horses." – Henry Ford



<https://www.youtube.com/embed/AlmASHISmTM>

Evolutionary vs. Revolutionary

Most technological advances are merely *evolutionary* in that they build incrementally upon what came before. Each evolutionary version introduces a slight variation that adds new features or enhances performance over its predecessor. But other advances take great technological leaps. These *revolutionary* innovations often spring from original, out-of-the-box thinking and introduce entirely new approaches to common problems.

Whether evolutionary or revolutionary, the impact that new technology has on our lives can be quite profound. Not only do these technological advances bring us new tools that help to make us more efficient or productive, but they oftentimes can completely change our daily routines and radically alter the ways we interact with the world around us.

What is the most *revolutionary* technological innovation in the last five years?

Make a list of at least 10 technological advances from the last five years. For each item on your list, decide whether it is *evolutionary* or *revolutionary* and consider how it has influenced people's lives in the years since its introduction. Be prepared to share and discuss your findings with the class.

Technological Advances

Sometimes, innovation comes about solely from creative thought and imagination. However, more often than not, innovative ideas also rely on essential advances in technology. Without these achievements, even the most creative idea might not be feasible. Here, scientists and

engineers push technology further by broadening scientific knowledge and inventing new tools and processes that others may use in bringing their ideas to reality.

For example, Facebook and other social networking sites would not be possible were it not for a host of other underlying technologies that make those sites possible, including the World Wide Web, TCP/IP networking protocols, web servers, web browsers, mobile phones, digital cameras, and of course, computers just to name a few. Until all of those technologies had been developed, what we know of as Facebook could never have existed.

What other technological dependencies can you think of?

Imagine your typical cell phone and all of the things that it can do. Make a list of at least 10 underlying technologies that had to exist before that phone could have been built. Then identify at least five things that you do all the time that you would not be able to do if cell phones did not exist. Be prepared to discuss your lists with the class.

Impact on Individuals and Society

Technological advances are made *by people for people*. Scientists, inventors, and developers all work hard to innovate new ideas and create new products that directly or indirectly affect the rest of us. In most cases, the impact we experience is a positive one as the technology improves or enhances our lives in some meaningful way (e.g., email correspondence, productivity software). Other times, technological advances also have an equally powerful down side (e.g., spam, computer viruses).

How has digital technology had an impact on your life?

Make a list of at least 10 forms of digital technology that directly impact your own life. As you make your list, try to identify examples that you think nobody else in the class will think to list themselves, but that at least one other person in the class will agree impacts them in the same way that you have described. Afterward, each student in the class will have the chance to name one example from his or her list to see if anybody else in the class shares that same relationship with that form of technology.

Assignment

Design a technological innovation that could someday revolutionize and enhance everyday life.

Working in pairs, your task is to invent a new technological product or service that might be possible within your lifetime. This is your opportunity to be as creative and imaginative as you would like as you and your partner conjure up the next new innovation to potentially revolutionize our digital world.

To begin your collaboration on this project, you and your partner should work together to

identify recent technological trends and use those trends to imagine what might very well come next.

For starters, consider how technology has already advanced during your own life so far. What products, services, or technologies do you now rely on every day that did not exist when you were born? How have those technologies changed over time? Has the change occurred practically overnight or has it evolved gradually? How will these technologies continue to change in the near future? What do they allow you to do today that you could not do in the past? What can you still not do today that you hope to maybe do in the future?

These are the types of questions that futurists ask themselves when they ponder the possibilities that the future likely holds. As a budding young futurist yourself, your task is to use your imagination and your own, personal aspirations and desires to envision one or more of these possibilities and identify ways in which *you* might be able to someday change the world.

For this assignment, you will need to perform a series of tasks in the process of designing and documenting your own future innovation and its potential impact on society:

- Identify a current problem that a technological advancement might solve.
- Invent a technological solution to that problem.
- Identify the key features of your solutions.
- Identify the potential risks and benefits that your solution poses and how they might affect individuals and communities.
- Identify the technological challenges that must be overcome before your solution can become a reality.

elevator pitch

A short presentation that summarizes a big idea and conveys all of the essential information quickly and succinctly. The term derives from the idea that an elevator is a great place to corner someone if you want to force them to hear your pitch for a new idea. But you only have a very limited amount of time to deliver your spiel before the elevator reaches the person's floor and they escape get out.

At the end of this unit, you and your partner will deliver a 2 – 3 minute "elevator pitch" in which you will demonstrate your idea to the rest of the class and attempt to inspire them get behind your vision of the future. After each presentation, you will also provide written feedback to your peers on the potential of the innovations they presented.

Submission

Your submission will be in the form of written report and an "elevator pitch" presentation that you will give to the class that introduces your vision of the future.

Your report should include the following sections:

- **Purpose:** Describe a problem that your innovation seeks to address and what purpose it will ultimately serve (e.g., entertainment, problem solving, education, artistic expression).
- **Description:** Provide a detailed description of your idea, what it would look like, and how it would work or be used.
- **Features:** Provide list of key features that your innovation will possess.
- **Benefits:** Identify multiple ways that future individuals and/or communities will benefit from your technological innovation.
- **Risks:** Identify the potential hazards that your innovation might introduce in society (e.g., security, privacy, social disparity, physical harm).
- **Technological Resources:** Identify existing technological resources that your innovation might utilize or rely upon.
- **Technological Challenges:** Identify any technological challenges or limitations that must be overcome before your innovation can become a reality.

For your "elevator pitch," you should prepare the following items:

- A written script for a 2–3 minute presentation
 - Introduce the idea.
 - Identify the innovation's primary problem or use case.
 - Describe the how the innovation would work.
 - Describe how society would benefit from your innovation.
- A multimedia presentation that includes a simulation of your innovation
 - Highlight the key features of your innovation
 - Demonstrate how someone would use your innovation

Rubric

Criteria	Scoring Notes	Score	
Purpose and Description Clearly describe a problem that your innovation seeks to address and what purpose it will ultimately serve (e.g., entertainment, problem solving, education, artistic expression). Also provide a detailed description of your idea, what it would look like, and how it would work or be used.	<ul style="list-style-type: none"> • Defines the innovation. • States the issue or problem the innovation seeks to address. • Describes what purpose the innovation will ultimately serve. • Describes how the innovation would look, work, and be used. 	2 pts	0 pts
Weighted: 20% Features Provide a comprehensive list of key features that the	<ul style="list-style-type: none"> • Provides a comprehensive list of features. • Provides some 		

innovation will possess.	description as to the purpose of these features.		
Weighted: 10%			
Benefits and Risks			
Identify several ways that future individuals and/or communities will benefit from the technological innovation, and identify several potential hazards that your innovation may introduce (e.g., security, privacy, social disparity, physical harm).	<ul style="list-style-type: none"> Presents several benefits of the technological innovation. Presents several potential hazards of the technological innovation. Links these benefits and hazards to their impact on society. 	2 pts	0 pts
Weighted: 20%			
Technological Resources			
Identify a comprehensive list of existing technological resources that your innovation might utilize or rely upon.	<ul style="list-style-type: none"> Provides a list of technological resources that the innovation relies upon. Lists reasons their innovation relies upon these resources. 	1 pt	0 pts
Weighted: 10%			
Technological Challenges			
Identify several technological challenges or limitations that must be overcome before the innovation can become a reality.	<ul style="list-style-type: none"> Identifies several technological challenges and limitations that must be overcome for the innovation to be successfully created. Provides some insight on how these technological challenges might be overcome. 	1 pt	0 pts
Weighted: 10%			
Presentation Quality			
The presentation is easily heard, engaging, and incorporates appropriate visualizations.	<ul style="list-style-type: none"> Presenters speak loudly and clearly. Presenters are engaging and/or entertaining. Visualizations appropriately support the presentation. 	2 pts	0 pts
Weighted: 20%			
Presenter Length			
Presentation must span 2–3 minutes.	<ul style="list-style-type: none"> Presentation is between 2–3 minutes long. 	1 pt	0 pts
Weighted: 10%			
TOTAL			10 pts

UNIT TOPIC:

Everyday Computing

Social Networking and Communication

- You will explore the ways that innovations in digital technology can impact the lives of individuals and communities.
- You will analyze the role that digital technology plays in your everyday life.
- You will analyze the role that digital technology plays in your social communications and interactions.

Search, Wikis, Commerce, and News

- You will explore the impact that instant access to global search, news, and information has had on individuals and communities.

Cloud Computing

- You will analyze the benefits and risks of cloud computing.

The Digital Divide

- You will investigate the socioeconomic causes and effects related to the digital divide.

UNIT TOPIC:

Everyday Computing

Social Networking and Communication

- You will explore the ways that innovations in digital technology can impact the lives of individuals and communities.
- You will analyze the role that digital technology plays in your everyday life.
- You will analyze the role that digital technology plays in your social communications and interactions.

Social Networking

Yesterday's Technology

While it is becoming increasingly difficult to comprehend, there was actually a time in which most of the modern-day conveniences we take for granted did not yet exist. Things like Facebook, Twitter, Instagram, and Snapchat, for instance, are all relatively new phenomena that have rapidly taken over our culture and forever changed the ways that we connect and interact with one another. But each of these and many other similar technologies now form such an integral component of our digital lifestyles that it is hard to imagine how we might have ever been able to function without these everyday resources.



The truth is, this has always been true about all forms of technology throughout the history of human civilization. All revolutionary forms of technology change people's behavior in ways that make the past seem crude and arcane. Modern digital technology is no different. And even today's cutting-edge innovations will one day feel slow and laborious when something even better inevitably comes along.

Social Interaction Before the Internet

Human beings are social creatures. We have engaged in social interaction since the dawn of civilization, from families to tribes to villages to towns to cities to nations. Originally, most interaction was face-to-face and benefited greatly from the invention of spoken language. But over time, the advent of different technologies, like parchment and written languages, has expanded our reach, increased our capabilities, and altered our behavior. Through writing, we were suddenly able to communicate over great distances of both space and time. Only in the last couple of decades has that interaction moved away from physical form and into the virtual world of electronic communication. And again, it has had a profound impact on our behavior and fostered new forms of social interactions throughout various societies and populations.

Social Structures

One of the most empowering features of electronic social media has been its ability to create a sense of community, especially in places or situations where such a community could not have otherwise existed. With the global scope and widespread reach of social media, previous obstacles like geography, age, or socioeconomic status, that isolated groups of people from one another are no longer barriers. Individuals with shared interests, but whose paths would never have crossed in the "real world," are able to come together, communicate, and interact in the virtual world of an online social network.

This increased ability to find someone who shares the same interests as you allows

marginalized individuals to experience a unique sense of belonging. Likewise, the centralized nature of social networking environments enables new and unique special interest groups and facilitates coordination of group projects and collaboration in a way that was previously difficult, if not impossible.

The Sharing Economy

"I would expect that next year, people will share twice as much information as they share this year, and next year, they will be sharing twice as much as they did the year before." — Mark Zuckerberg, November 2008

The rise in popularity of social networking in recent years has radically altered people's perceptions of privacy and their willingness to "share" what previously had always been seen as personal and/or private information. In 2008, shortly after Facebook opened up its service to the public at large, Mark Zuckerberg [made headlines](#) with his bold assumption that people would be increasingly willing to share anything and everything about themselves, regardless of privacy issues. At the time, most critics scoffed at Zuckerberg's naiveté and arrogance to make such an assumption. However, through the growth and popularity of services like Facebook and Twitter, users, and time, have proven Zuckerberg to be surprisingly correct.

This raises the question of whether people of the past were actually private for privacy's sake or if their reluctance to publicly share personal information had more to do with the simple lack of any effective way to do it. After all, before text messaging, tweeting, or blogs existed, there really was no effective way for individuals to reach out to "the world" and express themselves. It was the invention of a set of new, digital technologies that suddenly enabled this ability to freely and broadly share oneself openly with those who might listen.

Think about that for a second. A young, 24-year-old, college dropout with a unique and controversial vision of the future created a website and forever changed an entire society's attitudes toward personal privacy and public sharing — all through computational technology.

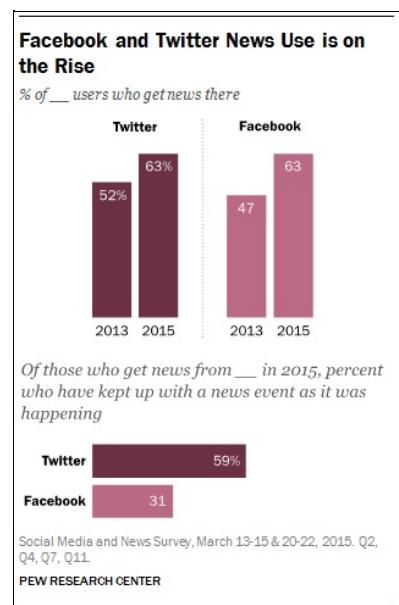
The New News

Historically, technology has always played a key role in democratizing societies and empowering individuals by enabling broader access to careers and capabilities that had previously been limited to an elite few. In recent years, we have seen such a shift occurring in the ways people access and report news events. The rise of blogs and social networks and the ubiquity of Internet-connected camera phones has created a new opportunity for "citizen journalism" in which literally *anybody*, regardless of their journalistic training or experience, can witness and report on important, socially impacting events.

In fact, a recent [study](#) by the Pew Research Center shows that an increasing number of Americans are now getting their news from online, social sites like Facebook and Twitter.

This marks a radical shift in the way information is disseminated throughout the public and, perhaps more importantly, who controls that flow of information. Traditionally, as the only, common source of news, the journalistic community has served to filter and shape the news as it sees fit. However, with the free and open access for anybody to share breaking events, a much more diverse set of voices has emerged.

This newfound and wide-reaching platform for free speech also threatens/promises to overturn the status quo by giving voice to individuals or groups who have historically been marginalized by the mainstream press. For example, in 2011, social media [played a key role](#) in the ouster of Egyptian President Hosni Mubarak and launched a wave of social revolution across the region by using social media to organize demonstrations and to relay events to the rest of the world "like a megaphone."



Assignment

When you go home, find a parent, grandparent, or other adult from an earlier generation with whom you can sit down for a few minutes to discuss the different forms of social interaction and networking that they grew up with.

You should ask questions that will allow you to compare and contrast their experiences with your own. Some ideas for things you might discuss include the following:

- What kinds of social activities did you engage in?
- How often did you socially interact with others?
- Who did you interact with socially? How far away from you did they live?
- Do you still have contact with them today?
- What tools or technology, if any, did you use to interact with others?
- What have been the biggest technological changes in social interaction that you have seen since you were younger?
- How would you have done things differently if today's social media tools were available when you were younger?

Write a short summary of the things you learn from your conversations that includes the following items:

- Identify five aspects of social interaction that have fundamentally changed since they were your age.
- Identify five aspects of social interaction that are more or less the same as when they were your age.
- Describe the most surprising thing that you learned about social interaction in the past

and explain why it was so surprising.

- Identify one way that you think social interaction might change between now and the *next* generation (i.e., in 20–30 years).

Models of Sharing

It's All About Who You Know

Whether you are a developer of a social network or merely one of the countless users of such a network, it is important to recognize that not all social structures are the same. There are many different types of relationships that individuals might have with one another and there are many different ways that they might want to either limit or encourage interaction between each other.

For example, there are many different ways of "knowing" somebody. You might *know about* George Washington or Thomas Jefferson from history class, but do you *know* them personally? Likely not, as they both died over 200 years ago. Similarly, you might *know* your school counselor, nurse, or principal, but do you *know* any of them outside of school? Better still, do any of them *know you*? And if so, to what degree? The school nurse might know about your peanut allergy, but does she know what you did on your last family vacation or the name of your favorite author or musical artist? What about your best friends? How well do they *know you*? How well do you *know them*? Do you share everything? Or do you only share some things? Likely, you choose to share a limited set of details with certain individuals or groups while sharing a different subset of your life with other individuals or groups.

Today's most popular online social networks and self-publishing platforms exhibit this diversity of "knowing" in their very design. Each service is designed to model a very specific form of relationship that you or anybody else might have in your everyday, offline lives.

Public vs. Private Communication

There is an idiom that refers to the difficulty of "putting the genie back in the bottle." The idea is that once a particular event is done, it cannot be undone. Information works in the same way. Once a bit of information is distributed and known by others, it is virtually impossible to retract that information and force it to once again become "unknown." This makes the issue of *privacy* a critical factor in our globally connected, digital world. The technology makes it easy to share information with a wide array of individuals. Usually that is seen as one of the strengths of the Internet, but it also serves as one of its drawbacks as information may be shared (or even *re-shared*) in ways that were never intended.

When it comes to designing or using a social network, it is critical to understand how the service might handle the issue of privacy. On one end of the spectrum, a service might choose to make all information publicly available to *anybody* at *any time*. At the other end of the privacy spectrum, all information is securely locked down and made available only to the owner or creator of the information. The very nature of a "social" medium that involves the interaction and exchange of ideas between multiple people would suggest that the latter extreme is inappropriate for such a service. However, the former extreme in which

everything is publicly accessible might be too public. Most cases call for a solution that falls somewhere in between the two extremes. That is, users usually want to be able to restrict what they share to their intended audience on more of a case-by-case basis.

Public

In its original conception, the World Wide Web was seen as a tool for facilitating a one-to-many publishing platform. Website authors would publish their content where it would be *publicly* hosted on a web server that anybody with a web browser could access and retrieve a copy of the published materials. By default, the content hosted at a web site was to be considered completely public (i.e., available to anybody at any time).

Private(ish)

On the other hand, the email (or "electronic mail") protocol was designed to facilitate *private* communication. That is, email is shared communication that is intended only for the very specific recipient(s) to which it is sent. Ironically, while access to an email message is usually limited to the recipient, the [Simple Mail Transfer Protocol \(SMTP\)](#) that defines how all email transmission is handled, actually does not ensure total privacy. In fact, as an email message is transmitted from node to node throughout the public network of routers and gateways across the Internet, the contents of the messages are exposed in plaintext. In real-world terms, email is *private* in the way that a postcard sent through the US Postal Service is *private* — in the end, only the addressee receives the card, but any mail handler who looks at the back of the card can see and read its contents.

Private

A better example of truly *private* communication (to the extent that that is not an oxymoron) might be Skype or Google Hangouts. Like email, these forms of communication allow one individual to specify their intended contact(s) and limit the conversation to just those parties. Unlike email, however, these video chat services employ encryption so that the digital information that makes up the audio and video signal is encoded before it is transmitted across the Internet and then decoded only after it reaches its destination. This means that the two people on either end of the conversation can see and hear everything being said, but for any eavesdropping router along the way between them, the signal will be unintelligible. In this case, the use of *encryption* is the key factor that enables true privacy with video chat systems. However, this does not prevent either party on either end of the conversation from recording the chat session and later sharing it with others — a privacy vulnerability that even something like Snapchat cannot solve.

Symmetric vs. Asymmetric Relationships

Consider the differences between your relationship with your best friend versus the relationship you have with your favorite Hollywood celebrity. In your friend's case, the friendship is likely mutual. That is, each of you considers the other to be their friend. If so, what you have is a *symmetric* relationship — you like your friend and your friend likes you. However, in the case of the Hollywood celebrity, while you might be a huge fan of theirs, they likely do not even know you exist (sometimes the truth hurts). When a person becomes

famous, by definition, it means that millions of people suddenly know who they are. But the reverse cannot be said. Becoming famous does not ensure that the celebrity knows each and every one of their millions of adoring fans. Almost all of those millions of relationships are one-sided. These are examples of *asymmetric* relationships.

In enabling users to interact with one another, online social networks encourage and regulate the types of activities their users can engage in based on the nature of the type of relationship the network is designed to serve. A symmetric system is built upon the idea that interactions will involve two-way exchanges, often of a more personal nature and in which both parties are more or less peers. In contrast, an asymmetric system assumes a largely one-way flow of information that often involves an impersonal delivery of information.

Which model an online service chooses, whether *symmetric* or *asymmetric*, will dictate the primary characteristics of the network, the engagements it offers, and the users it aims to serve. And of course, many services that start out as one type might, over time, adopt features of the other type, resulting in a hybrid system that is sometimes symmetric and other times asymmetric.

Symmetric

Facebook was built to be a *symmetric* network. The very idea of "friending" involves mutual actions by both parties in the friendship. One person must first choose to "Add Friend" and then the second person must accept the friend request. Both parties must take action to create the relationship in the Facebook system and both parties then have equal access to the features that such a relationship enables within the Facebook ecosystem. In every way, the relationship is mutual, two-way, and symmetric. Of course, over time, Facebook has added features that introduce more asymmetric behaviors, such as the ability to selectively restrict which friends can see/interact with a post. Nevertheless, the service is still primarily true to its symmetrically designed origins.

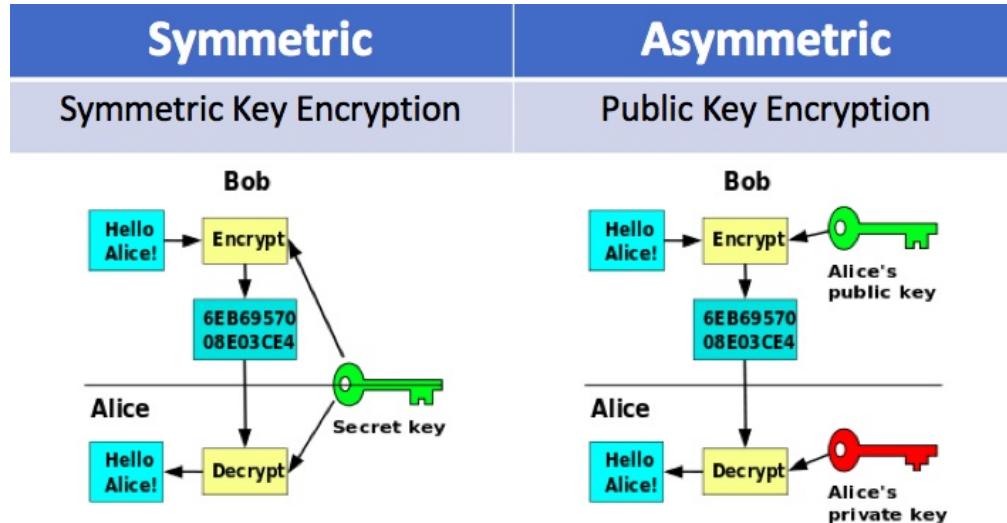
Asymmetric

Twitter, on the other hand, was built to be an *asymmetric* network. Unlike Facebook's friend-based approach where all users are on par with one another, Twitter uses a publisher-subscriber model. One user "tweets" (publishes) a message that is then automatically sent to each of their "followers" (subscribers). For public accounts, these relationships are one-way, requiring action by only one party to create the relationship. Namely, the subscriber chooses to "Follow" the posts from another user. Unlike Facebook, the publishing user does not need to "accept" or explicitly permit the relationship to be established. And like Facebook, Twitter has also evolved into a hybrid system that incorporates some symmetric aspects through the use of direct messages, protected accounts, and per-user blocking.

Encryption

Encryption methods have traditionally been described as either symmetric or asymmetric. When we talked about the Caesar Cipher in Unit 1, Alice and Bob both had the same information to decrypt a ciphertext (encrypted data). Since both parties use the same key to encrypt and decrypt data, we would consider this a *symmetric encryption*. If Alice and Bob

used **public key encryption** to encode and decode data, they would be utilizing **asymmetric encryption**. Public key encryption allows the generation of two keys: a public key (made available publically) and a private key (known only by the owner). *Certificate authorities* (CAs) issue digital certificates that validate the ownership of encrypted keys used in secured communications and are based on a trust model.



Open vs. Closed Platforms

The Internet as we know it today got its start in the early 1960s as a US Department of Defense project known as ARPANET (Advanced Research Projects Agency Network). For the next 20 years, ARPANET was privately operated by the US military and its use was limited exclusively to government and military applications. Only in the 1980s did the network open up to civilian uses and transition to the free and open Internet that we know today.

Together, the ARPANET and Internet provide excellent examples of the differences between *closed* and *open* systems. Closed platforms are often referred to as being "proprietary" because they are owned, managed, and operated by a single owner or proprietor. While that owner might choose to allow others to access and use their platform, they still maintain final control over the system, how it is used, and how it evolves. Open platforms, however, do not have a single owner calling the shots. Instead, they might have a centralized committee of interested partners to coordinate and manage the development and growth of the platform.

There are advantages and disadvantages to both approaches, but the ultimate choice of whether a platform should be open or closed comes down to the goals that drive the development of the system. In the case of ARPANET, the US military had ample government funding to build and operate the network as well as the need for a secure and reliable means of maintaining communication in the event of a nuclear attack. As such, it was *their* system that *they* built with *their own* funds and for *their own* needs, so they kept it to *themselves* (i.e., a *closed* network). By the 1980s, the technological and economic advantages of making a global, electronic network available to the commercial market merited opening up the network to more civilian uses by business and individuals.

Open

Open source and licensing of software and web content raise legal and ethical concerns. *Open source code* is code that is publically available for anyone. Unlike Apple's super secret source code for Mac OS X, open source software allows programmers to view, reuse, and remix the source code for individual use. Though this source code is generally publically available, there are still rules that govern its use (or reuse). Like using any material that you have not created, you must abide by the terms and conditions of use. Typically these terms can be found close to the source code or commented in the program itself.

Email and web publishing are both examples of open platforms that have been created around open standards. Any developer can create an email application that conforms to the various standards for handling email ([SMTP](#), [POP](#), [IMAP](#)). The program can then send email to and receive email from any other email user anywhere on the Internet no matter which email client (program) they might be using. Without these open standards, there would be no guarantee that any message that you write and send would be compatible with the software being used by your intended recipient. But with these standards, all emails created by any standards-compliant email client are guaranteed to be fully compatible with all other such clients, thus enabling the global communications system that we have come to rely on. Open standards also allow us to ensure cryptography is secure for Internet encryption. Nataraj Nagaratnam, IBM Distinguished Engineer, [explains how these open standards reinforce security.](#)

Closed

While services like Facebook and Twitter might be publicly available and free to use, they are still closed, proprietary systems. Each company controls the data that they store and strictly regulates how that data can be accessed and modified by its users. For example, users cannot easily transport their tweets, status messages, comments, chat histories, or friends lists to other competing services. Similarly, the ability to integrate these services into other, third-party apps or sites is strictly limited to what Facebook or Twitter choose to allow. Having this level of control over their platform gives each company the ability to more reliably build, develop, and monetize their platform, but it comes at the expense of user choice and compatibility.

Assignment

As a class, make a list of all of the different services and methods that people can use to communicate and/or interact with one another online. Then, working in pairs, decide whether you would classify each of the services as either public or private, open or closed, and symmetric or asymmetric. Be prepared to explain and defend your choices.

Service	Public/Private	Symmetric/Asymmetric	Open/Closed
email	private	asymmetric	open
Facebook	private	symmetric	closed

Twitter

public

asymmetric

close

UNIT TOPIC:

Everyday Computing

Search, Wikis, Commerce, and News

- You will explore the impact that instant access to global search, news, and information has had on individuals and communities.

Essential Services

The New Normal

As computational technology has grown over the last several decades, it has come to play an increasingly greater role in our modern lifestyle. What was once limited to a select few (e.g., wealthy corporations, electrical engineers, scientific researchers, etc.) is now a commonplace commodity that nearly everybody uses and relies upon to one degree or another.

Today's technology is ubiquitous. From the smart phone in your pocket to the cash register at your neighborhood store to the traffic signals at a nearby intersection, the products of computer science are everywhere. Everything around us collects, stores, and/or manipulates digital data to record information, compute results, and make decisions about how things should work. We cannot escape the influence of digital computing in our everyday lives.

And that has had a profound impact on the ways that societies function and has altered the ways that individuals behave and make decisions. The influence of computational technology is so pervasive that people increasingly take it for granted, especially younger individuals for whom this technology has *always* existed.

One of the goals of this course has been to open your eyes to many of the computational influences that surround you and show you how to harness the capabilities of these tools and resources.

Utility or Luxury?

As computing becomes more pervasive and individuals integrate it into their daily lives, we grow increasingly dependent upon this technological resource. More and more of our infrastructure, whether it is banking, shopping, medical records, communication, or basic utilities like electricity and water, is being built around and optimized to rely upon computational technologies.

One of the issues this raises is the question of what will happen if this infrastructure collapses or is taken away for some natural, political, or economic reasons. How will we function if the Internet "goes out"? What happens to those who cannot afford to pay for network access? These are just a few of the serious questions that arise as we integrate this technology into our lives.

At some point, a technology becomes so essential to the normal functioning of a society that it elevates from a mere convenience or personal luxury to an essential public utility like water and electricity. In recent years, many policymakers have begun to make the argument that access to the Internet is rapidly becoming such an essential utility and that it is something all individuals should be entitled to. In fact, many would argue that we are already there and that the law is simply lagging behind.

Later in this unit, when we discuss a few high-profile social and political issues like the *digital divide* or *net neutrality*, we will look more closely at a number of the questions that society will need to address as the advances of digital technologies continue their forward march of progress.

Search

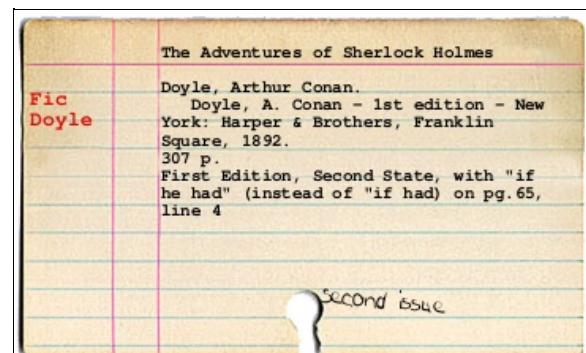
Search

The Internet has no such organization - files are made available at random locations. To search through this chaos, we need smart tools, programs that find resources for us. — Clifford Stoll

In 1998, Larry Page and Sergei Brin launched Google, an online search engine that was driven by the PageRank algorithm (named after Larry Page) that the two had developed while students at Stanford University. While Google was not the first search engine on the Web (at least a dozen popular search engines were developed in the half decade before Google), it was the first to use the relative connectedness of webpages to rank and prioritize search results, an approach that quickly built Google into one of the leading sources of online search. Google's success has been further cemented by the verbification of its name. Today, in popular language, to look up answers online is to "Google it."

The very fact that people routinely turn to search engines like Google, Bing, DuckDuckGo, or even Siri whenever they have a question about something is a testament to the power of search and the value it adds to our lives. Before the Web, information was not as readily available to the mass public. Much of the knowledge and information that society possessed was either private, undocumented, or locked up in books buried in a local library. Immediate access to diverse ideas and resources was not instantaneously accessible anywhere to the degree that it is today. And search engines, like the Dewey Decimal card catalog system of libraries, provide an efficient interface for indexing and finding obscure and relevant bits of information.

The Dewey Decimal card catalog system is an example of *metadata*. Metadata is **data about data**. Metadata can be descriptive data about an image, a Web page, or other complex objects. MP3 files have a robust amount of information about the file. Organizing these files by genre or artist is an excellent way to utilize metadata effectively for searching and sorting these files. Google makes use of similar information about webpages, images, and videos to help you search efficiently. Metadata can increase the effective use of data or data sets by providing additional information about various aspects of that data.



The easy access to any and all types of information has profoundly altered individuals' behaviors, especially when it comes to learning. If there is anything a person might want to learn about or any skills that they might want to develop, the Web has made the tools and

resources needed to acquire that knowledge readily available to anyone who is interested. But while online search has increased the amount that we *can* know, it has also reduced the amount that we *need* to know. No longer is there a need to learn and remember infrequent details. If there is anything important that you need to know later, you can "*just Google it.*"

Wikis

Wikis

"What you get as a wiki reader is access to people who had no voice before." — Ward Cunningham

While online search has made it easier to catalog and index the wealth of knowledge to be found across the entirety of the Web, wikis have consolidated this vast volume of information into well-organized online references built around user-based communities.

Created in 1994 by Ward Cunningham, a wiki is a platform in which multiple users are able to collectively contribute to a shared knowledge base. The crowdsourced nature of wikis gives individual users the ability to shape and inform the content in authentic ways that traditional information sources historically have not. Rather than presenting its information through the filter or with the bias of a centralized editorial control, wikis rely on peer-based writing, fact checking, editing, and moderation.

Many people find it uncomfortable that any random user can edit the content in a wiki unchecked, assuming that such lack of oversight reduces the reliability of the site. However, most wikis develop strong communities of dedicated volunteers to moderate the content on their sites and help to keep vandalism and other disruptive behavior in check. As Ward Cunningham himself has said, "Wikis work best in environments where you're comfortable delegating control to the users of the system," although he has also stated that, "With wiki, you have to trust people more than you have any reason to trust them. In 1995, it was a safer environment, don't know if I could have launched wiki today."

While Wikipedia is perhaps the most well-known and visited wiki, it is by no means the only one. Across the Web, thousands of wikis have been created that specialize in a broad range of special interests serving smaller, often underrepresented populations, giving each one a global voice that they would not have otherwise had.

Commerce

Commerce

"Why leave the comfort of your own home when you can get something custom made to your exact size for less? I believe this is the future." — Patrick Curtis

Another phenomenon that has emerged from the growth of the Internet is **e-commerce**, or **electronic commerce**. With its global reach, the World Wide Web is a powerful tool for those who have something to sell. No longer is a retailer's customer base limited to the local area within driving distance of a storefront. Now, the entire global, online population is capable of serving as potential customers.

New Business Models

By moving their business online, retailers have discovered a number of new and innovative ways to serve their customers.

Online Storefronts

Without the need for the physical presence of a so-called "brick and mortar" store, online retailers are often able to offer more efficient services and better pricing. Resources that would traditionally be spent on an elaborate showroom and hired salespeople can instead be redirected to the creation of an automated online storefront, improved products, and discounts on products and/or shipping costs.

Many consumers have embraced the convenient and hassle-free opportunity to shop online, just as previous generations did with mail-order catalogs. Only now, the online "catalog" offers a much richer and more informative shopping experience through the use of multimedia, user reviews, and personalized customer recommendations based on previous buying behavior.

Independent Sellers

Commerce is another area in which the democratizing effects of the Web can reveal themselves. With online auction sites like eBay or artisanal e-commerce sites like Etsy, individuals now have access to the same global market as the large, corporate retailers for selling their wares. This has sparked a boom in both the supply of and demand for custom-made and limited-production runs of unique, new products and services that were either not available or not feasible before the advent of an e-commerce market.

Crowdfunding

For those independent sellers who envision creating a sustained business or a complete new product or service, **crowdfunding** uses online access to customers as a means of

funding their project. In the past, clever entrepreneurs who had a brilliant idea for a new product might never have been able bring that product to market due to lack of start-up funds (i.e., it takes money to make money). Sites like Kickstarter and Indiegogo were created to enable these innovators a way of reaching out to potential customers and recruiting them as backers who might help make their idea a reality by providing initial "investment" funding.

Credit Card Fraud, Network Attacks, and Phishing

Along with all of the advantages and consumer benefits that e-commerce has made possible, online shopping and electronic banking have also introduced a whole host of new risks that require addressing. While computational technology has streamlined the purchasing process by eliminating face-to-face sales and enabling much higher volumes of transactions to be made, those very same factors have made attacks on financial transactions much more of a problem.

For example, brick and mortar retailers using credit card scanners and electronic cash registers are vulnerable to attacks through these network-connected devices. These automated systems are desirable targets for attackers simply because their very function is to store and process large volumes of valuable information, such as credit card numbers and customer profile data. In recent years, a number of major retailers have accidentally exposed millions of customers' financial data when their systems were attacked.

Another example is *phishing*, a practice used by malicious individuals who pose as legitimate banks or businesses in an attempt to trick customers into revealing their sensitive banking information.

A major focus of research in the computing industry centers on encryption techniques and standards for protecting sensitive data and on the design of more robust systems and procedures for making attacks more difficult to perform. Technologies such as "Chip and PIN," "Apple Pay," and other mobile payment systems are some of the most recent attempts to strengthen the security of electronic financial transactions.

UNIT TOPIC:

Everyday Computing

Cloud Computing

- You will analyze the benefits and risks of cloud computing.

Cloud Computing

Everything Old is New Again

The term "cloud computing" is a relatively recent buzzword for a new type of computing. Thanks to the Internet, Wi-Fi, and other modern, networking technologies, individual users can "offload" much of their computational and data storage efforts onto remotely hosted servers and online services. This has helped to make smaller, more portable computing devices like laptops, tablets, and phones more practical because the device itself does not need to do all of the work. Instead, the heavy lifting can be handed off to "the cloud."

Believe it or not, this is not a new idea in computing. In fact, decades earlier, long before the availability of public or private Internet access, most computers were the size of entire buildings. While the actual devices that people used might have looked like today's desktop computers, they were actually just simple interfaces, so-called "dumb terminals" or "thin clients," that connected remotely to the actual computer or server located elsewhere, much like our phones and tablets connect remotely to web services.

Of course, yesterday's "dumb terminals" have since given way to today's "smart phones," which have far more computing power in their own right. But the increasing use of and reliance upon "the cloud" for remote computing show that recent and incredible advances in technology have not strayed far from their predecessors.

Client-Server Model

Like with the "dumb terminals" of the past, today's cloud computing is built upon the "client-server model." That is, the networking process can be described according to the interactions of two remotely located computers (or more accurately, programs running on those computers) — namely, the "client" and the "server."

The **client** stands at one end of the communication process and typically represents the end-user. At the other end of the process is the **server**, a centralized computer that all individual end users connect to. In other words, the server serves the needs of its clients (i.e., users).

The analogy is that of a waiter at a restaurant. The waiter (i.e., *server*) waits on the customers (i.e., *clients*), each individual customer makes requests about which particular food items or beverages they want. The waiter then goes back into the kitchen while the cooks (i.e., *processor*, or *CPU*) gather the raw ingredients (i.e., *data*) from storage (i.e., *database*) and prepares the meal. When it is ready, the waiter returns to the customer and serves the meal as they requested.

In computing, this process of a client initiating a request or transmitting data to a remote server, which then processes the request or data and returns the appropriate response is the basic functionality of all Internet-based communication. When you surf the Web, your web

browser is the client. The website you visit is essentially data hosted (stored and served) on a remote web server.

All of Your Data Wherever You Go

One of the key advantages that cloud computing offers is portability. The fact that your data can be accessed from anywhere in the world has had a profound influence on the ways that people work and live. With cloud computing, it is no longer necessary to carry your data around with you when you can just download or stream it from a remote server.

This has freed us from our desks and allowed people to create entirely new behaviors and use-cases that were previously impossible or prohibitively expensive before cloud computing.

In the early days of PCs, your data was trapped in a large box sitting on a person's desk at home or in their office. Data that was on their computer at home could not be easily taken to work. Data that they used at work could not be brought home. Any data that needed to be transported from one location to another had to be copied onto physical disks that were slow and had relatively little capacity.

With cloud computing, users can now access their data from *anywhere* — home, school, work, airports, coffee shops, etc. And since wireless transmissions free us of the need for physical media, we can now use phones, tablets, and other devices that lack physical ports to access this cloud data as well.

Of course, one of the trade-offs of storing your data remotely is the additional costs of transmitting your data to and from the cloud — both in terms of time and bandwidth.

Off-Site Storage

Another benefit of cloud-based systems is their use as *off-site storage* for your data. Anyone who has ever used a computer either has or will experience the loss of data at some point. Accidents happen. Hardware fails. Your data is important, but there is no way to ensure that it will remain 100% safe, intact, and uncorrupted. While it is good practice to maintain regular backups of your files, photos, and other digital information, a good rule of thumb is the "3-2-1 Rule":

- 3) Have at least **three** copies of your data.
- 2) Store the copies on **two** different media.
- 1) Keep **one** backup copy off-site.

The first two items deal with redundancy and maintaining multiple copies of your files, but the last item avoids the issue of creating a "single point of failure." It does no good to have multiple backups of your files if they are all kept in the same location. A single disaster, like a fire, flood, earthquake that destroys your house will destroy your files and your backups at the same time.

Cloud storage helps to solve that problem by allowing users to keep a separate copy of their data safely stored at a remote location.

Ownership of Cloud Data

Who Owns Your Data

"You own all of the content and information you post on Facebook, and you can control how it is shared through your privacy and application settings." — Facebook Terms of Service

Despite the advantages of storing one's data *in the cloud*, the use of online services and remote storage systems also comes with a number of legal risks. Specifically, the issues of "ownership" and "access" come into play whenever any party (i.e., the cloud service) acts as a custodian for the property (i.e., the data) of another party (i.e., the user). Unfortunately, cloud computing is still a relatively new phenomenon and the legal distinctions about the ownership of personal information and data is not always so clear.

For example, who *owns* your Facebook profile? You? Or Facebook? Clearly, Facebook *hosts* your profile and stores the raw data from all of your posts, comments, photos, likes, chat histories, and friend connections on their servers, but does that mean that Facebook *owns* the information making up all of that data? If asked, most users would expect that the answer is "no," Facebook does *not* own their personal data. And fortunately, Facebook agrees, actually. According to their [terms of service](#), individual users "own all of the content and information" that they post on Facebook. But those terms of service also include a number of statements in which, by using the service, you grant Facebook permission to do a number of things with your data that may or not be in your best interest.

Selected excerpts from the Facebook terms of service include the following (emphasis added):

- You grant us a non-exclusive, transferable, sub-licensable, royalty-free, worldwide license to **use** any IP [Intellectual Property] content that you post on or in connection with Facebook.
- You give us permission to **use** your name, profile picture, **content**, and **information** in connection with commercial, sponsored, or related content (such as a brand you like) served or enhanced by us.
- You permit a business or other entity to pay us to display your name and/or profile picture with your content or information, without any compensation to you.

The terms of service also clarifies, in detail, several more generalized terms that

- By "**information**" we mean facts and other information about you, including actions taken by users and non-users who interact with Facebook.
- By "**content**" we mean anything you or other users post, provide or share using Facebook Services.
- By "**data**" or "**user data**" or "**user's data**" we mean any data, including a user's **content** or **information** that you or third parties can retrieve from Facebook or provide to Facebook through Platform.
- By "**use**" we mean use, run, copy, publicly perform or display, distribute, modify, translate, and create derivative works of.

Most other services have similar terms of service statements that specify the rights, limitations, and obligations of both users and the service itself with regard to the user's data. Each agreement is different and is tailored to the interests of the service that wrote it, which are not always the same as the interests of the users. As a user, it is important to understand what rights and privileges one might be giving up when they agree to use a particular service.

Assignment

When you sign up for a new service or product, you are often presented with a copy of a *Terms of Service* for you to read and acknowledge. It is almost always a very long document with a lot of seemingly obscure *legalese*, so most people skip past it and quickly check the box saying that they have read and agree to the terms when they have not actually read anything. You have probably done it yourself. It is only human to do so. But what exactly are you agreeing to when you do this? What rights might you be giving up? Even if you never read any of these agreements, it is important to understand that the terms that are carefully detailed in each of these agreements directly affect you, your privacy, and your rights to your own content that you create, store, and/or use with the service. More importantly, these agreements usually specify the limits that the service can be held to with regard to what it can do with your data, whether it is on your behalf or in your best interests or not.

In this exercise, you are to select one of the following online cloud services and actually read its *Terms of Service* agreement to see what such a document includes and then answer the questions below.

- [Amazon Web Services](#)
- [CrashPlan](#)
- [DropBox](#)
- [Evernote](#)
- [Facebook](#)
- [Google Drive/Gmail](#)

- [iCloud](#)
- [Instagram](#)
- [Rackspace](#)
- [Snapchat](#)
- [Yahoo/Flickr](#)

Answer the following questions about the Terms of Service that you read:

1. What restrictions does the service impose on its users?
2. What permissions does the user grant to the service?
3. How is the user's data defined? What does it specifically include? What does it not include?
4. What are the costs for using the service?
5. Can the service use the user's data for its own purposes (e.g., promotion and advertising, making money, etc.)?
6. What happens to a user's data if they stop using the service?

UNIT TOPIC:

Everyday Computing

The Digital Divide

- You will investigate the socioeconomic causes and effects related to the digital divide.

The Digital Divide

What Is It?

Following a 1995 study by the Markle Foundation, Lloyd Morrisett, president of the foundation, described a "digital divide" between the *haves* and *have-nots* when it comes to access to information. Essentially, the study found that the same racial and cultural barriers that impact societies offline also have a similar effect on their access to online resources. As a result, some have likened this phenomenon to the racial and socioeconomic disparities of previous generations, calling it the Civil Rights issue of the new millennium.

The term is now used to describe the gap that exists between those who have sufficient access to information and communication technologies and those who do not. The reasons for this gap are numerous and complex, but the effects of the unequal access to computational services within society can be felt in a variety of areas, including education, healthcare, employment, social connectedness, and political awareness and participation.

Why Does It Exist?

The digital divide also exists across all levels of society, including local, national, and global scales. One of the reasons for this is the natural diversity and inequality within large societies. Just as age, income, education level, disability, language, and literacy may help or hinder an individual's ability to excel within their community, those factors also express themselves in enabling or suppressing online access to the digital community.

There are many different factors that specifically contribute to the existence of the digital divide. In general though, the problem can often be described in terms of obstacles that inhibit individuals from fully realizing the potential of information and communication technologies:

1. *Physical Resources*: Lack of computers and/or network connections make digital participation impossible.
2. *Opportunity*: Lack of situations in which individuals are able to connect digitally inhibits participation.
3. *Digital Literacy*: Lack of experience or comfort with digital technology as well as a lack of awareness of its benefits discourages active participation.
4. *Digital Skills*: Lack of education and training on the proper use of digital technology limits participation.

Simply connecting online requires certain financial and physical expenses, such as a computer, networking infrastructure (e.g., copper cables, fiber optics, wireless access points, routers), and Internet and network connectivity services. For many, these costs are insurmountable and make online engagement an impossible fantasy. In poorer communities or nations, the benefits promised by Internet connectivity must compete with more pressing

needs, such as food, clothing, and shelter. For people in these communities, the Internet often does not rise to the level of becoming a priority. As a result, they are excluded and disconnected from the larger community of the digital world.

In addition, a lack of technological literacy also prevents many people from connecting online. Whether it is that they do not know how to use technology or that they simply do not understand or recognize the benefits of technology, many people avoid or are otherwise prevented from accessing the online world. As a result, these people, too, are isolated from the digital world, whether by choice or by circumstance.

Why Does It Matter?

Is the "digital divide" much ado about nothing? Does everybody really *need* to be connected? Are they truly disadvantaged if they do not have access to information and communication technologies? The answer to all of these questions depends on the degree to which access to online services is becoming an essential necessity.

Many of today's basic utilities and social resources are moving exclusively online or at least have an online component. In journalism, inexpensive and readily accessible print publications are rapidly being discontinued in favor of their online counterparts. Those without connectivity are thus losing their access to news and civic discourse, impeding their ability to remain well-informed citizens. Similarly, healthcare and many government-related services now expect individuals to create and manage their accounts via online portals. Again, for those without connectivity, this becomes yet another obstacle between them and the services they need and are entitled to.

In addition, the problem is not just about what these disconnect communities lack. It is also about what the rest of society as a whole, including both the haves and have-nots, loses through their lack of participation. For every individual that is excluded from the modern digital ecosystem, that is one more voice that is silenced. One more voice that is unable to contribute ideas and solutions to problems. One more voice whose perspective is lost. These individuals bring much-needed value to a community and their lack of participation in the digital world does a great disservice to all of society.

Also, as the disparity between the informational haves and have-nots grows, these two populations become even more divided and polarized, leading to civil inequality and increased social tensions that further alienate the communities from one another.

Assignment

For this task, you will be given two topics on which you are to prepare a short presentation. For the first task, you must work alone and you may not use any technological devices beyond a pen/pencil and paper. For the second task, you may collaborate in small groups and the group may use any computational device(s) that you have access to (e.g., computer, smart phone, tablet, calculator, presentation software, web browser, access to the Internet, etc.).

In addition to your two short group presentations, you should each individually write a brief

reflection on your experiences completing the two tasks. Be sure to compare and contrast the experiences of working collaboratively with technology versus working alone and without technology.

UNIT TOPIC:

The Internet

Network Infrastructure

- You will examine the overall design and architecture of the Internet.
- You will explore the role of servers, routers, gateways, and clients.
- You will examine the domain name system and its role in network routing.

Communication Protocols

- You will examine a number of standard network protocols, including IP, TCP, UDP, SMTP, HTTP, and FTP.
- You will investigate the series of components and events that are involved in the transmission of an email or SMS text over the network.

World Wide Web

- You will analyze the impact of hyperlinked documents on how individuals find, acquire, and learn new information.
- You will analyze the legal, social, and commercial impact that the World Wide Web has had on society.

UNIT TOPIC:

The Internet

Network Infrastructure

- You will examine the overall design and architecture of the Internet.
- You will explore the role of servers, routers, gateways, and clients.
- You will examine the domain name system and its role in network routing.

Communication Protocols

- You will examine a number of standard network protocols, including IP, TCP, UDP, SMTP, HTTP, and FTP.
- You will investigate the series of components and events that are involved in the transmission of an email or SMS text over the network.

Network Infrastructure

Network Architecture

In its simplest form, the Internet is just a large collection of interconnected devices. Some of the devices are connected directly to one another while others are only connected indirectly through a series of intermediate devices. Altogether, these devices make up the network. They consist of all varieties of electronic, computational hardware, including desktop computers, laptops, tablets, phones, printers, cameras, routers, Wi-Fi access points, etc.

Through this arrangement of interconnectedness, any two devices are able to "communicate" with one another by transmitting a digital signal along the appropriate medium (e.g., copper wire, fiber optic cable, radio wave transmission, etc.) that makes up that connection.

One of the strengths of the Internet's design is the inherent redundancy that such a complex and multiply connected network offers. In most cases, there is more than one pathway through which a transmission can be sent in order to reach its destination. This allows the network to not only be fast and efficient by finding the most optimal route, but also robust enough to continue functioning even if part of the network fails and a pathway is cut off.

Client-Server Model

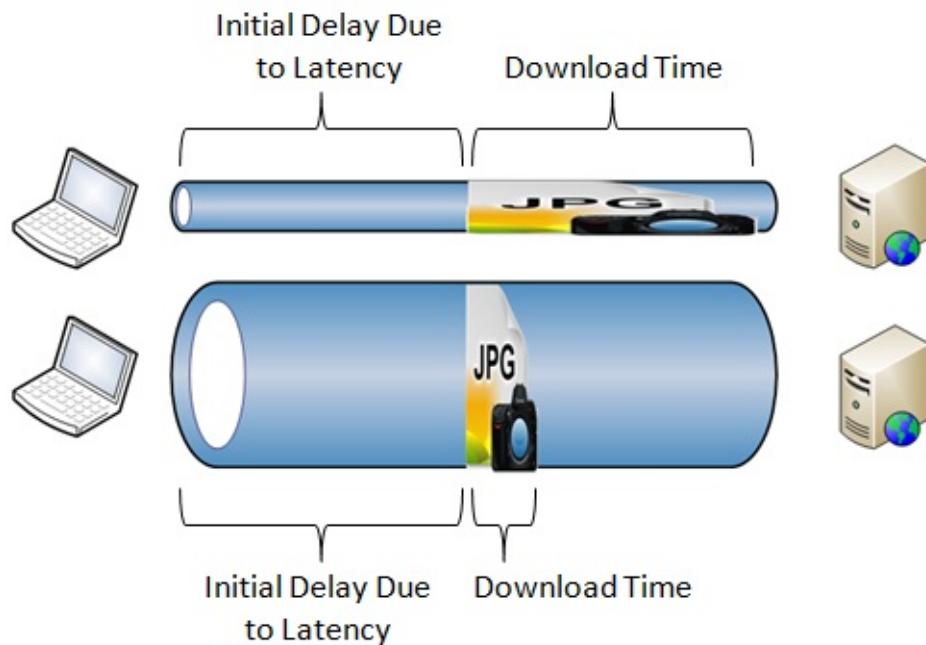
The basic operation of any networked communication system centers on the transmission of information between two parties. Traditionally, these parties are referred to as the "client" and the "server."

Typically, the *client* initiates the communication by sending a request to the *server* — usually a fully automated program running on a remotely located computer — which then processes the request and sends the appropriate response back to the *client*. Examples of client software that users might be familiar with include web browsers, e-mail applications, and chat programs.

When a user clicks on a link in a browser, a URL (i.e., the address of a web page) request is sent out into the larger network, where it is then routed to the location of the particular computer that is running a web server for the requested URL. The server then generates the content for the page (formatted in HTML) and transmits it back through the network to the user's computer. The web browser (i.e., the client) then interprets the HTML information that it receives and uses that to render the text and images onto the user's screen.

So how fast can this virtual exchange happen? And what factors limit this speed? While *technically* transfer of information over the Internet can be very, very fast (up to 750 Mbps in [some areas](#)), two aspects to consider are latency and bandwidth. The **bandwidth** of a system is a measure of bit rate — the amount of data (measured in bits) that can be sent in a fixed amount of time. The **latency** of a system is the time elapsed between the

transmission and the receipt of a request. If you can imagine the Internet as a pipe that information travels through, bandwidth is the size of the pipe. You can see in the picture below the relationship between latency and bandwidth, as well as a small amount bandwidth (top) compared to larger bandwidth (bottom).



In some use-cases, the *server* might initiate the communication in what is referred to as a "server-side push" (in contrast to the "client-side pull" described above). In a "push" situation, a centralized server *pushes* information out to one or more clients without the end-user explicitly requesting the information.

A chat program is example of an application that makes use of both "client-side pull" and "server-side push" transmissions. In a situation where two users are messaging with one another, they are each operating an application that is functioning as a *client* and are remotely communicating with a central *server* located somewhere on the larger network. The server effectively sits in between these two users, relaying their messages from one to the other. When one user sends a message, the contents of the message as well as delivery instructions are sent to the server, which is listening for such an incoming transmission. The server then interprets the delivery instructions to identify which user the message is intended for. The *server* then initiates a transmission in which the message is "pushed" to other user's *client*, which renders the message on their device as an incoming message from the first user.

Nodes, Gateways, and Routers

The overall structure of a communications network, through which all digital traffic flows, consists of a number of *nodes*. Each node is a machine or device that acts as either a communication endpoint (e.g., a *client* computer or a remote *server*), a connection point that links together two other nodes (e.g., a *gateway*, *bridge*, or *modem*), or a redistribution point that links together multiple nodes and other networks (e.g., a *router*, *hub*, or *switch*).

Communication Protocols

The Importance of Protocols

protocol

A set of formal standards that specify the proper formatting of data, communication procedures, and handling of information between any two networked components.

The processes involved in browsing a webpage, sending an email or text message, or uploading a photo to an online service all seem relatively straightforward and simple. You click on a link and the webpage loads in your browser. You type your message and click **Send**. Or you tap the **Share** icon and select your favorite social media site and after a few seconds, your photo just magically appears online. It is all so very simple. Except that, behind the scenes, it is not simple at all.

Every time you perform one of these actions as a user, you set off a chain reaction of complex data processing and information exchange within dozens, if not hundreds, of separate computational devices across the breadth of the Internet. In these examples, the software running on your computer, phone, tablet, or other computing device responds to your clicks and taps by initiating a multi-stage network transmission in which your data hops from node to node as it is handed off from one router or hub to the next until it finally reaches an appropriate server on the other end. In each of these hops, where your data is transmitted between two nodes, the exchange of information is made possible through the use of standard, agreed-upon communication protocols that each node follows precisely.

These protocols, or sets of rules for the proper handling and formatting of information, are designed to establish a common interface through which different hardware/software components can interact, regardless of their own internal design or manufacturer.

In the case of communication protocols, abstraction allows for a clear set of standards for how information should be exchanged to be explicitly described while avoiding the complexity and minutiae of how a manufacturer might actually implement that standard in their hardware or software. By describing the standards broadly, abstraction also enables future innovation by not limiting or restricting the kinds of hardware that can implement the standards. As long as it conforms to the expected protocols, any type of compatible component can be developed to participate in the exchange of digital information on the Internet.

Standard Protocols

Throughout the history of the Internet, as technologies have evolved, a number of standards have been developed to ensure the efficiency and robustness of these new capabilities.

Traditionally, the task of developing these communication protocols falls on the shoulders of independent researchers who develop the protocols and ad hoc standards committees, often made up of experts and organizations who have a vested interest in the proper functioning of the Internet as a whole. Together these committees agree upon and oversee application and further refinement of existing protocol standards.

Designing and developing a protocol is no easy task. Any agreed-upon standard needs to 1) enable reliable and efficient transmission of data at large and small scales, 2) provide an unambiguous set of protocols, and 3) be flexible enough to adapt to and accommodate future technological innovations. Hierarchy (which will be discussed in greater detail in [Domain Name System](#)) and redundancy (provided by established protocols) help systems, like the Internet, scale.

Several of the most commonly used protocols include those for Internet data transmission (IP, TCP, UDP), e-mail (SMTP), webpages (HTTP, HTTPS), and files (FTP). Using the notion of abstraction, the Internet protocol suite, more commonly known as TCP/IP, organizes each of these into a stack of distinct layers based on the hierarchical relationships of their use. These include the *link*, *transport*, *Internet*, and *application* layers.

But communication protocols are not just limited to the Internet as a whole. A number of independent, privately owned platforms also offer protocols for publicly integrating third-parties into their networks through the use of API's (*application program interface*). APIs and libraries simplify complex programming tasks by providing developers the building blocks necessary to interface with an existing environment or other software components.

Programming documentation (discussed in *Unit 4: Draw Shapes*) for APIs/libraries is an important aspect for a programmer to understand all components of development.

For example, when Twitter first launched in 2006 as a private messaging service, it was built around the SMS text messaging standards (whose protocol limits messages to 160 characters). Twitter's protocol limited its own messages to 140 characters, using the remaining 20 characters to convey meta information about the message (e.g., user id of the sender, etc.). However, Twitter's initial growth benefited from their offering of a public API that allowed countless third-party developers to create their own innovative applications. By following the protocols laid out in the API, these applications could tap into the Twitter backbone to send and retrieve tweets. In this way, open standards (like the Twitter API) fuel the growth of the Internet.

The Link Layer

These standards specify the protocols for establishing the physical (and wireless) connections between end-point devices.

Ethernet ([IEEE 802.3](#))

This protocol specifies the use of physical, short-range connections commonly used in local area networks (LAN), such as homes or offices, which uses coaxial cable, twisted pair, or fiber optic connections. You will often see these types of connections between a cable

modem and a wireless access point or router or between a desktop computer and a wall outlet (which ultimately connects elsewhere to a modem).

Wireless LAN ([IEEE 802.11](#))

802.11 is the standard wireless protocol for Wi-Fi communications seen in most laptops, phones, tablets, and other networked devices that are not physically connected to a network.

Bluetooth ([IEEE 802.15.1](#))

Bluetooth is another wireless connectivity protocol intended specifically as a *peer-to-peer* connection between two close-range devices. The protocol specifies procedures for two devices to be "paired" with one another so that they can securely exchange data over the air, without the need for a physical connection.

The Transport Layer

These standards specify the protocols for how data is transmitted through the network.

Transmission Control Protocol ([TCP](#))

The TCP protocol ensures that all packets of a data stream are transmitted and received exactly as originally sent. It specifies methods of performing error-checking analysis of the received packets to ensure that no error or loss of information was introduced along the way. If a packet is lost or found to be damaged, the TCP protocol will identify the error and request that the packet be resent. In this way, routing on the Internet is fault tolerant and redundant. TCP is most suited for applications that prioritize accuracy and completeness of transmission over speed, such as e-mail (SMTP), webpages (HTTP, HTTPS), and file transfer (FTP).

User Datagram Protocol ([UDP](#))

The UDP protocol is better suited for applications in which speed is more important than accuracy or completeness of information, such as online gaming or video or audio streaming.

The Internet Layer

These standards specify the protocols for how individual datagrams (i.e., packets) are packaged and labeled for delivery over the network.

Internet Protocol ([IP](#))

Originally developed as part of the *Transmission Control Program* in 1974 and then later separated out as its own standard, the Internet Protocol (IP) specifies how individual packets of information are packaged and labeled for delivery, much like the way that the postal service specifies how envelopes and packages should be sized, addressed, and stamped.

The Application Layer

These standards specify the protocols for handling data that is designed for specific use-cases, such as e-mail (SMTP), the World Wide Web (HTTP, HTTPS), or raw file handling

(FTP).

Simple Mail Transfer Protocol ([SMTP](#))

This protocol specifies how electronic mail should be formatted in order to be sent and routed to the intended recipient. Most notably, the standard specifies how header information may be prepended to the beginning of a message by each node that handles the message along the way. Users who receive an e-mail can view the headers of the full message once they receive it to see various information about where the message came from and through which server it originated from and those that it passed through on its way to their inbox.

Hypertext Transfer Protocol ([HTTP](#), [HTTPS](#))

These are the primary transport protocols used by the World Wide Web for delivering web content. The protocol was designed to transmit hypertext documents that have been formatted according to another protocol, HTML (Hypertext Markup Language). The "S" at the end of "HTTPS" indicates "Secure" and provides additional protocols and procedures for encrypting information prior to transmission and decrypting it upon receipt.

Other common protocols in the application layer include the **File Transfer Protocol ([FTP](#))** for sending files, **Domain Name System ([DNS](#))** for looking up IP addresses of domains, **Dynamic Host Configuration Protocol ([DHCP](#))** for initializing an Internet connection with an ISP, and **Post Office Protocol ([POP](#))** and **Internet Message Access Protocol ([IMAP](#))** for storing/retrieving e-mail messages.

Internet Protocol

A Protocol for Packet Network Intercommunication

When Vint Cerf and Bob Kahn first proposed the *Transmission Control Program* in 1974, it established the use of a technique, known as *packet switching*, as the underlying method of transferring data between nodes across the Internet.

With packet switching (or packet *routing*), all data is subdivided into small, suitably sized blocks that are then transmitted independently from one another, potentially taking different routes to reach the data's intended destination. Rather than sending an entire message of some arbitrary and varying length all at once and hoping that it reaches its destination intact, larger messages are broken up into smaller, fixed-size *packets*.

Each of these packets contains two components: a *header* and a *payload*. The header contains the IP addresses of the source node (e.g., the sending client) and the destination (e.g., the receiving server) and any other information needed to deliver the packet. The payload is the actual data that is being sent.

In a real-world example, the process of sending digital information across a network is a lot like mailing a document across the country via postal mail. The sender seals the document (the *payload*) in an envelope or box (the *header*) upon which is written the return address (*source* IP address) and the recipient's address (*destination* IP address). Delivery of the package involves handing it off through a series of many postal carriers and routing systems (*nodes*) as they transport the package to where it needs to go.

By sending data as individualized packets of information, the Internet is able to operate more efficiently and is better able to handle unforeseen errors in transmission.

Imagine sending a very large document consisting of thousands of pages. If sent all at once as a single document, any single error, delay, or misdelivery along the way will require the entire document to be resent in full. However, if each page of the document was individually sent, any misdelivery will require only that particular page to be resent. In addition, as the amount of data traffic varies over time, different routes might become faster than others. By sending the larger document as many smaller pieces, each piece can be routed along the most optimal path at that moment. Together these advantages of packet switching serve to improve the speed, efficiency, and reliability of the communication network.

IP Addresses

The relative location of each node on the network is specified by the node's IP address, a unique numerical identifier, that allows other nodes to know which nodes they are directly connected to and to determine a route for sending data to its intended destination. In the case of the Internet, the people who designed the network devised a system for assigning unique 32-bit numbers to each device on the network.

For example, as of 2016, the address for the google.com search engine is **74.125.224.72**. What this means is that when you "Google something," your browser (i.e., your *client*) sends a search request through the Internet's network of nodes, addressed to a *server* located in some far-off location whose IP address just happens to be **74.125.224.72**.

As devices are added to the network, each is assigned an IP address that translates roughly to its geographic location on the network. More precisely, the Internet Assigned Numbers Authority (IANA) oversees the distribution of these addresses through a number of regional registries. Each registry then assigns blocks of addresses to different entities, like your local Internet service provider (ISP), which then assigns an available address from that block to each computer or device that connects through its service.

As a result, while each IP address can provide a general sense of the geographic location of the organization that a node is associated with (e.g., your local ISP), the actual address does not specifically locate the actual node itself. For example, two computers with very similar IP addresses from the same block of numbers might actually be physically located across town from one another and have nothing in common other than they both use the same ISP for Internet connectivity.

IPv4 ("Internet Protocol, version 4")

An IP address is usually expressed in human-readable form as a series of four numbers, each within the range of **0** through **255**, separated by periods (e.g., **123.45.67.89**). That is, the lowest possible numerical value for any address would be **0.0.0.0** and the highest possible value would be **255.255.255.255**.

Each of these four numbers is referred to as an *octet* because they each represent *eight* bits (or *binary digits*) of information. In [UNIT 3: Data Representation](#), we looked at binary numbers and how they correspond to the decimal (base-10) numbers that you are more familiar with and we will see why eight bits of data can only represent values between **0** and **255**. Suffice to say, four octets of eight bits each adds up to 32 bits of total information that can be used to identify each node on a network.

Just like there is a limit to the range of possible values that can be represented by 8 bits of data (e.g., **0** – **255**), there is a similar limit to what can be represented by 32 bits of information (e.g., **0** – **4,294,967,296**). That means that the Internet, as originally designed, can only generate 4.3 billion unique, 32-bit addresses, which limits the Internet to a maximum capacity of no more than 4.3 *billion* nodes. While that is a *lot* of devices, that number is still finite and can easily run out. In fact, it already has in some areas. APNIC (the Asia Pacific Network Information Centre), the registry organization for the Asia Pacific region, exhausted its pool of regional addresses in 2011. Then, in the fall of 2015, the American Registry for Internet Numbers (ARIN), the registry organization that assigns address for the United States, Canada, the Caribbean, and North Atlantic islands issued the last of its unassigned addresses.

IPv6 ("Internet Protocol, version 6")

While most Internet traffic today uses IPv4 as its addressing standard (and will likely continue for some time to come), it is gradually being replaced by IPv6 ("version 6") due to this issue of limited address capacity. IPv6 solves this problem by using 128-bit addresses — four times the length of the IPv4 addresses. With 128 bits of data packed into each address, this new standard is capable of sustaining more than 3.4×10^{38} individual nodes. That is *340 billion billion billion!*

But is that *enough*? Well, to put that number in perspective, if every grain of sand on all of Earth's beaches were assigned its own, unique IP address, the IPv6 standard would still allow for 60 quadrillion more similar Earthlike planets each with their own IP-enabled beach sand to have their own, unique addresses. In other words, while that number, too, is finite, it is reasonable to say that yes, it most likely is *enough*.

Domain Name System

The Internet's Directory Assistance

Every website is essentially a collection of content stored on a web server. Every web server is just a program running on an Internet-connected computer. Every Internet-connected computer is a networked node. And every networked node is assigned an IP address that identifies where it can be found on the network. So, it is reasonable to conclude that every website can be identified by its 32-bit IP address.

But how often have you ever entered an IP address into your browser? How often do you see any IP address in the address bar as you browser the Web? For most people, the answer to these questions is "rarely, if ever." Instead of IP addresses, most users rely on the use of *domain names* to reference an online site or service. And the Domain Name System is the hidden service quietly working in the background that makes it all possible.

Domain Names

The fact is that a 32-bit number, while perfectly ideal for the digital and electronic components of the routers and other computational devices that support the Internet, is not very human-friendly. Which of the following is more intuitive for humans?

- `01000010110111001001111001000100`
- `66.220.158.68`
- `facebook.com`

The first is an actual 32-bit address. The second is that same address expressed in a more human-friendly format of four octets. The last version, however, is the most recognizable of the three because it is descriptive and tells us everything that we, as humans, would want to know—namely that it refers to the social networking site, Facebook.

Numbers are great ways of cataloging and indexing things. People have Social Security Numbers, the products we buy have Universal Product Code (UPC barcodes), books have International Standard Book Numbers (ISBN), credit cards and bank accounts are numbered, drivers licenses have numbers, etc. But these numbers are meant for automated systems to store and process computationally. We prefer to call each other by our first and last names, products by their brands, and books by their titles. So it comes as no surprise that we would prefer to reference our favorite websites by descriptive names rather than an obscure and seemingly arbitrary sequence of numbers.

DNS Lookup

When you type a URL into an address bar, it is usually in a form that includes a domain name. For example:

<https://www.facebook.com/>

But how does this translate into an IP address that your computer and all of the intermediate routers and gateways can use to locate the actual Google server that hosts the page you are trying to load? The answer is a massive *lookup table* that acts like a large directory, dictionary, or phonebook that allows you to use a value that you do know to look up a value you do not know.

When you try to load a webpage, your web browser (e.g., the client) isolates the domain name specified in the URL you typed (e.g., "facebook.com") and sends that name to a special server known as a Domain Name Server (DNS). That is, the browser sends a request to a pre-configured IP address that corresponds to the location of a nearby nameserver (usually belonging to your ISP, although Google actually operates a couple handy nameservers at 8.8.8.8 and 8.8.4.4).

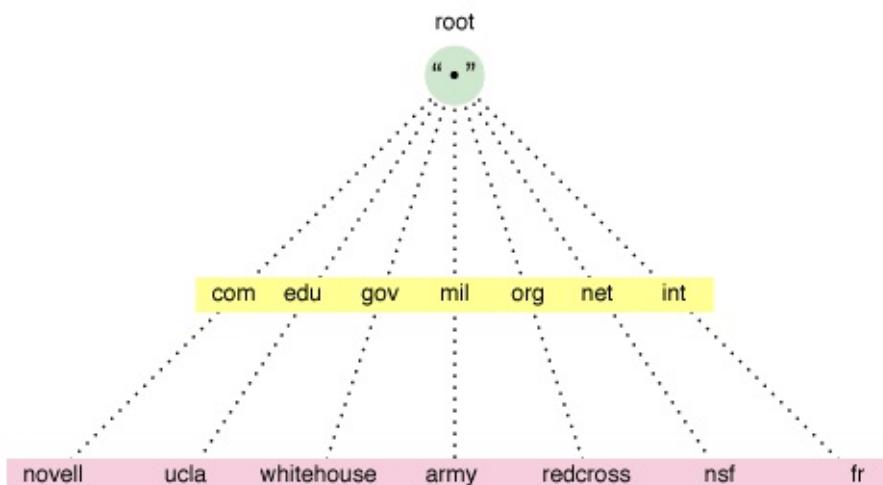
The nameserver, which is a computer that stores an updated list of every registered domain name and the IP address of the server that hosts that domain, looks up the address of requested domain name and sends that information back to your web browser.

Your web browser then sends the URL of your original page request to the IP address that it received from the nameserver.

Without a centralized name server at a known IP address, your browser would have no way of knowing which IP address it should contact to fulfill your page request.

DNS Hierarchy

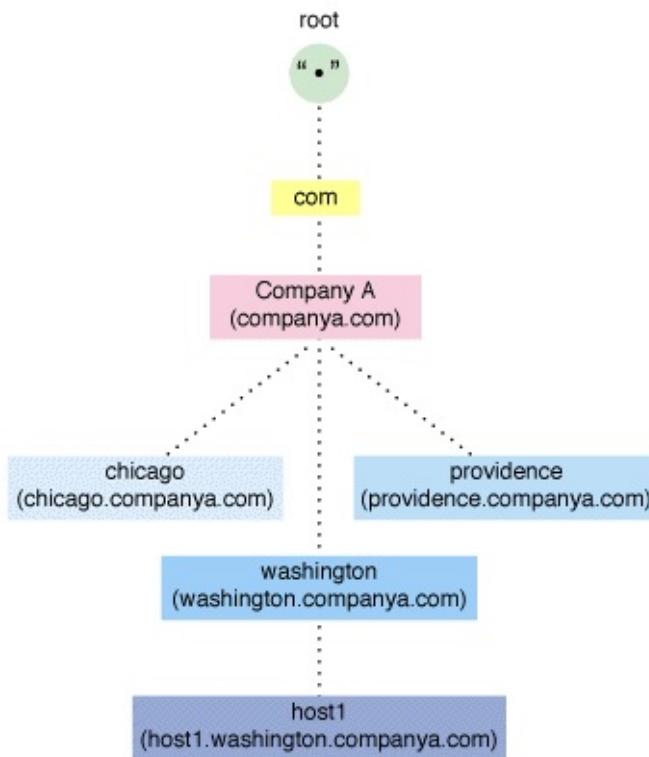
The domain name syntax is *hierarchical*. A hierarchy is an arrangement of elements in a ranking of inclusiveness or superiority. As you can see below, the beginningng is the *root* (signified by the period/dot). This is the top of the DNS hierachal tree.



The root is divided into familiar *domains* (.com, .org, .edu). From there, you can see we can travel into several *subdomains*. [Novell](#) offers an example furthering this understanding:

For example, Company A creates a domain called companya.com u

Any domain in a subtree is considered part of all domains above it.



The hierarchy that DNS utilizes allows for this system to scale to solve larger problems!

Common misconception: *DNS is secure.*

DNS was not completely secure because it did not include security based on the information that it contains (like host names and IP addresses).

UNIT TOPIC:

The Internet

World Wide Web

- You will analyze the impact of hyperlinked documents on how individuals find, acquire, and learn new information.
- You will analyze the legal, social, and commercial impact that the World Wide Web has had on society.

World Wide Web

WWW

How many times have you seen **www.** at the start of a URL? It is so ubiquitous that many web browsers and web sites will insert it into the URL even if you do not type it. But that **www.** is a special part of a domain's address that indicates that it is a server that hosts content that is designed to meet the standards of the **World Wide Web**. And almost every online service you likely use is a part of the World Wide Web.

In fact, the World Wide Web is one of those things that most of us use on a regular basis without ever thinking about how it works or what problems it was originally created to solve. But, since its inception in the early 1990s, the Web has proven to be one of the most revolutionary and empowering inventions in history.

Not to be confused with the broader concept of the *Internet*, the *World Wide Web*, itself, is a content-oriented ecosystem that has been built atop the globally networked infrastructure of the Internet. It was designed primarily to provide an open platform that could provide uses from all over the world a standard and accessible means of communicating and sharing information online.

Origins and Growth of the Web

"In those days, there was different information on different computers, but you had to log on to different computers to get at it. Also, sometimes you had to learn a different program on each computer. Often it was just easier to go and ask people when they were having coffee..." — Tim Berners-Lee

While working at CERN near Geneva, Switzerland, British computer scientist, **Tim Berners-Lee**, recognized the potential of the Internet as a communications and computational medium and proposed the development of a platform that might help to overcome some of its limitations.



As the Internet became more established, the world's many computers, servers, routers, and other computational devices gradually became networked together into a worldwide, interconnected ecosystem. However, much the same way that the introduction of air travel in the early 1900's suddenly brought together people from far off lands who spoke different languages, shared different customs, and adhered to different laws, the Internet also exposed similar differences and incompatibilities between the world's various computing systems.

Berners-Lee proposed that a standardized set of protocols and tools be developed that might help to ease the integration of these disparate computing systems and to facilitate

improved communications between them. In short, he wanted to employ the ideas of abstraction to design a more generalized means of sharing information across the Internet that was independent of any particular hardware or software that a user might be using. It is also to keep in mind that abstractions can be combined. Lower-level abstractions can be blended to make higher-level abstractions, such as short message services (SMS) or email messages, images, audio files, and videos.

As a result of his efforts, Berners-Lee created the set of fundamental tools and technologies that make up what we now more familiarly know of as the World Wide Web*.

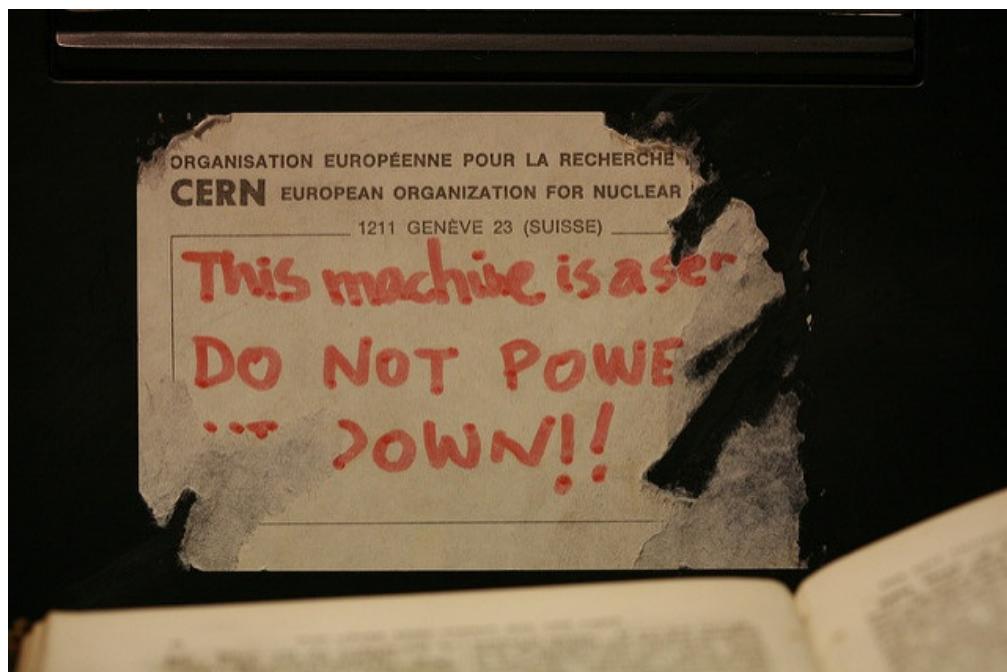
Web Applications:

- **Web browser**
 - Client application that runs on an end-user's computer and it used to request and view web pages
- **Web server**
 - Program that runs on a remote computer and that serves up web pages

Web Technologies:

- **HTML (*Hypertext Markup Language*)**
 - A standardized set of formatting instructions that dictate how the content of a web page should be arranged and displayed by the client application (i.e., web browser)
- **URI (*Uniform Resource Identifier*)**
 - A unique address that identifies each resource on the web
 - Also known as a **URL** (*Uniform Resource Locator*)
- **HTTP (*Hypertext Transfer Protocol*)**
 - Standards for requesting and receiving linked resources from across the Web

On August 6, 1991, Berners-Lee brought the world's [first web site](#) online. It ran on a NeXT cube computer located in his lab at CERN and prominently displayed a sticker on the front of the machine which read, "This machine is a server. DO NOT POWER IT DOWN!!"



*Interestingly enough, "World Wide Web" was not the only name that Berners-Lee considered when choosing a name for his creation. He almost named it by one of his other ideas: *Information Mesh*, *Mine of Information*, or *Information Mine*. Consider how the Web *the Mine* might look today with URLs like moi.google.com or moi.facebook.com instead of our familiar www. prefix.

Hyperlinks

One of the key features that Berners-Lee incorporated into his invention is the use of *hyperlinks* to connect documents with one another in a non-linear way. Unlike the pages of a book are arranged linearly in sequence (e.g., page 1, page 2, page 3, etc.), there is no such sequencing of documents in the World Wide Web. Instead, like the multiply connected computers of the Internet, the Web consists of a collection of massively interconnected pages of content.

Each web page is effectively a single, text-based document that has been "marked up" with embedded formatting instructions known as HTML (**Hypertext Markup Language**) *tags*. Each of these electronic documents are stored on a computer running a web server. The location of the file within the computer's file system corresponds to the documents URL (i.e., the *address* of the web page).

A *hyperlink* is a clickable bit of text, image, or other on-screen element within an HTML document that a user can select to request another, related document. Each link is designed to enable the user to selective seek out, or *browse*, from one document to the next, following whatever sequence they choose. This non-linear approach to organizing and connecting information has created an unlimited number of new ways that people can find, learn, and consume information.

Consider the following bit of HTML:

You can [search](http://www.google.com) for something, [tweet](http://www.twitter.com) a comment, or [like](http://www.facebook.com) a friend's post at these popular sites.

The above example, would produce the following hyperlinked text within a web page:

You can [search](#) for something, [tweet](#) a comment, or [like](#) a friend's post at these popular sites.

Here, you can see that "search," "tweet," and "like" have each been formatted to act as hyperlinks (linking to Google, Twitter, and Facebook, respectively). Each hyperlink is denoted with the use of an *anchor* (`<a>...`) tag that frames the text being linked (e.g., "search," "tweet," and "like"). Each anchor tag includes the URI of the other page or site that the hyperlink is referencing (e.g. `href="..."`).

When a user clicks on any of these links, the web browser sends a request to the corresponding web server for the specified page (as referenced in the `href` tag).

Exercise #1: Map the Web

Build a map of the World Wide Web. OK, maybe not *all* of it (It is rather large, after all.). In this exercise, you will begin mapping out the interconnectedness of a *very small* portion of the Web.

1. Using your preferred search engine ([Google](#), [Bing](#), [DuckDuckGo](#), etc.), conduct a search for your own name.
2. Record the URL of the first link that your search returns.
3. Visit that URL and count and record the total number of different links that you can find on that page.
4. Also record the URLs of up to 3 more of the hyperlinks on that page.
5. Continue repeating this process counting and recording hyperlinks for each URL you

record for at least two more levels.

Using your findings, estimate the total number of different pages that could be reached if you were to start at the URL found from your original "vanity search" (i.e., searching for your own name) and followed a series of five clicks. What about 10 clicks? 20 clicks?

Exercise #2: Wikipedia Race

Your teacher will select a random topic for you to look up on [Wikipedia](#). This will be your starting point for the race. Your teacher will then name a second topic. This will be your target. Your goal is to browse through Wikipedia to reach your target topic by only clicking on hyperlinks within the body of Wikipedia article. What is the shortest path that you can find to get from the starting topic to the ending topic (i.e., following the fewest number of links)?

BIG PICTURE:

Net Neutrality

Highlights

- You will discuss and explore the issues on both sides of the net neutrality issue.

Net Neutrality

"A Series of Tubes"

"The Internet is not something that you just dump something on. It's not a big truck. It is a series of tubes." — Sen. Ted Stevens

For many Americans, the term "net neutrality" first rose to their level of awareness as a result of the widespread ridicule in the popular media received by remarks made on the Senate floor on June 28, 2006. While addressing a Senate commerce committee discussing pending amendments to a telecommunications bill, Alaskan senator, Ted Stevens, secured his Internet fame with a rambling, 11-minute speech, highlighted by an unfortunately chosen "series of tubes" analogy.



<https://www.youtube.com/embed/f99PcP0aFNE>

While Senator Stevens' remarks (and the numerous Internet memes that they inspired) helped to raise public awareness of the so-called *net neutrality* debate, they did little to clarify the issue or inform the public about the actual stakes involved.

In simple terms, *net neutrality* concerns itself with ensuring equal and unrestricted access to all legal content that is available throughout the Internet. In principle, it argues that Internet service providers (ISP) and other gateway services that connect individual users to the wealth of content and services available across the network, should not discriminate against or favor certain content over other content.

Common Carriage

Ultimately, the net neutrality issue comes down to the question of how government legislation should or should not regulate the access to and delivery of Internet services. Is the Internet a public utility, like telephone, water, or electricity, where the good of the

community entitles all individuals to unrestricted access to these essential services? Or is the Internet more like a commodity or specialized luxury that consumers merely choose to purchase or not.

Historically, the FCC has considered the Internet to be an "information service". That is, it treats networked information as a product that can be optionally purchased or subscribed to, rather than pure communications between two parties (i.e., more like a magazine or newspaper than a phone call or letter). As such, the FCC (as in the *Federal Communications Commission*) has, until recently, taken a rather hands-off view of the Internet since it was not strictly regarded as a communication medium.

However, many proponents for net neutrality argue that the ways that people actually use and rely upon access to the Internet are much more analogous to other "telecommunications services" like telephone service. That is, the Internet is merely a conduit through which users connect to a remote service to send and receive communications in the same way that a phone line is a conduit, managed by a phone company, for communication between two callers.

In order to ensure that such party-to-party communications are unrestricted, unfiltered, and uncensored, legislation specifically regulates how telecommunications services operate and imposes limits on what they can and cannot do with the data and information passing through their channels. These services are designated as *common carriers* — a classification for a person or company that transports goods or people for another party and that is responsible for any possible loss of the goods during transport.

For example, telephone companies have been granted common carrier status. This means that while they are obligated to provide connection services between telephone users (i.e., carry communications), they are restricted from interfering, altering, filtering, or accessing the content of those calls. Similarly, this common carrier status also protects telephone companies from any liability for the communications they transmit (e.g., a telephone call between two individuals planning a crime does not make the telephone company a co-conspirator or accomplice to the crime). As long as a common carrier treats all communications equally, with the same hands-off policy, it is not responsible for the misdoings of its users.

"Common carriage is not a new concept — these rules have a centuries-old history. They have long been applied to facilities central to the public life and economy of our nation, including canal systems, railroads, public highways, and telegraph and telephone networks. In fact, common carrier rules have already been written into the Telecommunications Act of 1996 by Congress; they just need to be applied to broadband Internet communications by the FCC." — [ACLU](#)

By changing its interpretation of Internet services from "informational" to

"telecommunications", the FCC is able to apply many of the same precedents to the Internet as it does to other, more well-established telecommunications services.

The Case **FOR** Net Neutrality

In 2003, legal scholar Tim Wu helped define and popularize the term *net neutrality* by building upon the ideals of *common carriage*. He recognized that the Internet had been built around the notion of free and open access to information. However, unless safeguards were put in place, the gateway services, like ISPs, could potentially use their monopolistic positioning to undermine that goal.



<https://www.youtube.com/embed/uKcjQPVwfDk>

During his presidency, Barack Obama served as a strong proponent of the net neutrality principles. In his recommendations to the FCC, Obama advocated for the creation of new rules and policies designed to ensure that no single company offering access to Internet services can ever act as a gatekeeper that limits, filters, or censors what users can do, say, or see online.

"For almost a century, our law has recognized that companies who connect you to the world have special obligations not to exploit the monopoly they enjoy over access in and out of your home or business. That is why a phone call from a customer of one phone company can reliably reach a customer of a different one, and why you will not be penalized solely for calling someone who is using another provider. It is common sense that the same philosophy should guide any service that is based on the transmission of information — whether a phone call, or a packet of data." — Barack Obama

The president's recommendations reflected the expectations that the majority of users already hold concerning their Internet access and usage, specifically addressing the following assumptions:

- **No blocking:** All legally available content should be accessible by any user. ISPs should not be allowed to selectively restrict access to or censor content requested by its users.
- **No throttling:** ISPs should not intentionally slow down or degrade transmissions to certain sites or online services. All content should be treated equally.
- **Increased transparency:** FCC rules should apply to stages of Internet transmission and not be limited only to ISPs and their connection to individual users.
- **No paid prioritization:** In order to promote the growth of Internet-related businesses of all sizes, all sites and online services should have equal access to the Internet and should compete on a level playing field. No service should be handicapped simply because it cannot afford to pay for better service.

"Imagine if the phone company could mess with your calls every time you tried to order pizza from Domino's, because Pizza Hut is paying them to route their calls first. The phone company isn't allowed to do that, and, for a while, the FCC said broadband providers couldn't either." — [ACLU](#)

The Case AGAINST Net Neutrality

As an individual user, it is easy to see why net neutrality is probably a good thing to support. However, there are many equally valid arguments to be made *against* the policy as well. From the perspective of the companies and organizations that design, build, and operate the large, physical infrastructure of the Internet, there are many legitimate reasons why net neutrality principles actually interfere with the ability to provide improved services and/or inhibit further technological innovations or investments.

For example, Christopher Yoo, a legal scholar who specializes in communication and computer and information sciences, offers a number of reasons why net neutrality, as proposed, is misguided and potentially counterproductive.

One of Yoo's arguments centers on the issue of data discrimination — the notion that an ISP might treat different types of data or data from different sources differently, potentially favoring one type of data while restricting or filtering another type.

"The early Internet was dominated by applications such as email and Web browsing, in which delays of half a second were virtually unnoticeable. These are being replaced by newer applications, such as Internet telephony and streaming video, in which such delays can be catastrophic. One obvious solution would be to give a higher priority to traffic associated with time-sensitive applications. Unfortunately, this is precisely the type of discrimination between applications that network neutrality would condemn" — [Christopher Yoo](#)

UNIT TOPIC:

Innovations in Computing

Pioneers in Computing

- You will investigate a number of key individuals and breakthroughs in the development of modern computing.
- You will explore the design goals and technological advances in the development of the modern computer.
- You will explore the design goals and technological advances in the development of the Internet.
- You will explore the design goals and technological advances in the development of human-computer interfaces.

Distributed Computing

- You will examine the roles and applications of distributed computing.

Ethics of Autonomous Technology

- You will examine the ethical implications of autonomous technology.

UNIT TOPIC:

Innovations in Computing

Internet of Things

- You will explain how computing innovations affect communication, interaction, and cognition.
- You will examine how sensor networks function.
- You will examine the societal benefits and threats of smart devices.

Internet of Things

Are they taking over?



You have probably heard a conversation about machines rising up and controlling humankind - probably from an older relative or paranoid acquaintance. Glorified by the entertainment industry through movies (*I, Robot*, *The Matrix*, and *Transformers*) and books (*Cinder: Book One in the Lunar Chronicles*, *Robopocalypse*, and *Do Androids Dream of Electric Sheep?*), the fear of robots "taking over the world" has been a concern for many individuals. With the advent of many of our smart technologies, like sensor networks and Global Positioning Systems (GPS), convenience has triumphed over necessity in many of our lives. Our knowledge concerning how these smart technologies work and the securities that surround them may be our saving grace if/when the machines decide to turn against their makers (**us**).

The knowledge we have about computer security is suddenly applicable to everything and the restrictions and regulations and controls from the real world start being imposed onto us. The place where we're first seeing this collision is the Internet of Things. -Bruce Schneier

Internet of Things



The *Internet of Things* is the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data. All of the devices that are connected using the protocols of the Internet make up the Internet of Things. We humans have found these devices useful for tasks such as communication, navigation, and health care. For example, GPS and related technologies have changed how humans travel, navigate, and find information related to geolocation. Before GPS, navigation would be planned in advance - spontaneous changes would be difficult to make.

Sensor networks are created by autonomous sensors to measure environmental conditions such as light, temperature, and sound. These *sensor networks* facilitate new ways of interacting with the environment and with physical systems. Business use motion sensors to control lighting and temperature to conserve energy and money. Security can be established in organizations by creating laser grids and sensors to keep items safe. Ultimately, the Internet of Things allows us to bring the smart devices in our lives together in an efficient working capacity.



So, will these smart devices become too intelligent through their **interconnectedness** and take over the world? Your sibling has the same question...

Instructions

Recently, your younger sibling watched *I, Robot* and is now scared to go to bed because they think the "robots are going to attack in the night." Compose a short (between one and two minutes) presentation on the medium of your choice (Microsoft PowerPoint, Google Slides, etc.) to help your younger sibling feel brave about going to bed. If you cannot convince them, your parents said that your younger sibling would have to sleep in your room (and they probably snore).

1. Research the following topics:
 - Internet of Things
 - Smart devices
 - Sensor networks
 - Security related to the above topics

2. Make sure to properly cite your sources in the presentation. For help on citing material, visit the [Purdue Online Writing Lab](#).
3. Submit your presentation to your teacher through the specified method. Make to explain these topics in a way that a younger sibling would understand.

UNIT TOPIC:

Innovations in Computing

Distributed Computing

- You will examine the roles and applications of distributed computing.

Distributed Computing

Sharing the Workload

Consider a typical anthill. Depending on its size, it likely consists of millions, if not *billions*, of tiny grains of dirt or clay, each one carefully excavated from underground and individually hauled up to the surface where it is deposited. How long would it take to build such a structure? A single ant moving one grain of dirt at a time would take ages. But many anthills, even very large ones, seem to pop up practically overnight. How does such a monumental job happen so quickly?



The answer, of course, is that no single ant builds an entire anthill by itself. Each hill is the product of an entire colony, with each ant moving only a single grain of dirt at a time. But collectively, the entire colony is capable of moving thousands of grains at a time, vastly reducing the time needed to accomplish the greater task.

And this same process of distributing the workload for a large or complex task among multiple workers who can each operate in parallel with one another can also be applied to large and complex computational problems as well.

Thanks to the development of the Internet and other large-scale networking environments, the processing power of multiple computers can be harnessed to work together in solving complex computations that would otherwise be impractical, if not impossible, to solve using only a single computer.

Many such problems involve the processing of large amounts of raw data or the simulation of millions, if not billions, of various scenarios. Rather than asking a single computer to process the entire data set, it is possible to connect to a large array of additional computers and systematically farm out smaller segments of the larger data set to each computer. Each computer then only processes a small fragment of the overall problem, but collectively, this "colony" of networked computers can crunch through the data and achieve a result in a fraction of the time.

SETI@Home

In recent years, a number of projects have emerged that employ this process of distributed computing to tackle problems that would otherwise be too resource-intensive to solve in a reasonable amount of time. One of the first such projects to gain widespread popularity is [SETI@Home](#), launched in May 1999 by the University of California at Berkeley.

SETI (Search for Extraterrestrial Intelligence) is a scientific area whose goal is to detect intelligent life outside Earth. One approach, known as radio SETI, uses radio telescopes to listen for narrow-bandwidth radio signals from space. Such signals are not known to occur naturally, so a detection would provide evidence of extraterrestrial technology.

The SETI@Home project searches through massive amounts of astronomical data collected by radio telescopes in an attempt to recognize unique, space-borne radio signals that do not occur naturally. Signs of such signals would provide intriguing evidence for the possibility of some form of intelligence beyond Earth.

First conceived in 1995 around the same time that the World Wide Web was emerging and growing in popularity, the project aimed to harness the processing power of all of these newly networked home computers around the world to assist in searching through and analyzing the large volumes of radio telescope data.

Individual users who wish to participate in SETI@Home and donate their idle computer time can [download](#) the project's BOINC (Berkeley Open Infrastructure for Network Computing) software and run it on their own computer. This program connects remotely to the SETI@Home servers that then deliver an ongoing series of processing jobs that your computer can crunch on in the background (the program can even be set as your screensaver). As your computer completes each assigned task, it sends the results back to the home server and receives its next batch of data to process.

SETI@Home is just one of many [distributed computing projects](#) that users can participate in. The BOINC software alone allows users to join over three dozen scientific research projects that address topics across the full array of scientific disciplines, including physics, mathematics, chemistry, encryption, evolution, astronomy, medical, biological, computer science, etc.

Botnets

Unfortunately, massively distributed computing can also be used for malicious purposes as well as the beneficial uses described above. Such is the case for *botnets* — large collections of networked computers that have been infected by a worm, virus, or other form of malicious software that enables the computer to be remotely controlled or utilized without the owner's knowledge.

Much like the way that the SETI@Home servers deliver payloads of data to be processed on users' computers, botnets also deliver commands to its network of "zombie" computers, instructing them to perform a variety of less altruistic actions like sending massive amounts of spam or launching coordinated DDoS (Distributed Denial of Service) attacks upon unsuspecting sites.

In fact, these botnets are one of the primary reasons that spam and DDoS attacks are as bad as they are. Firstly, the large number of "zombie" computers forming the botnet

contributes to the massive scale of the spam and attacks. Secondly, the distributed nature of the botnet makes it difficult to track down the originating computers because they do not come from a single, centralized source.

Other botnets have been known to hijack the processing cycles of unsuspecting victims' computers to engage in [bitcoin mining](#).

"What would you do if you could harness the processing power of a very large array of computers?"

UNIT TOPIC:

Innovations in Computing

Ethics of Autonomous Technology

- You will examine the ethical implications of autonomous technology.

Ethics of Autonomous Technology

Are Machines Ethical?

Does the question of whether machines are ethical even make sense? *Can* a machine have ethics? From phones to cars to home-automated devices, as our technology becomes increasingly "smart" and capable of managing many of our daily tasks for us, these questions begin to become more relevant.



Humans are clearly capable of ethical behavior. As such, each of our actions is performed within the context of what we consider safe, fair, just, and ethical. But as new forms of technology take over these tasks, do they (or even *can* they) exhibit the same level of ethical awareness that we demonstrate ourselves?

The Trolley Problem

Several decades ago, philosophers proposed a thought experiment, known as the [*trolley problem*](#), that has become extremely relevant to computer scientists and legislators as the reality of practical self-driving vehicles has become ever more present.

The Trolley Problem

You witness a runaway trolley (i.e., a streetcar) barreling down the street. In its path, you notice five people stuck on the tracks and unable to move out of the way before they are struck by the trolley.

Fortunately, you have time to pull a lever that will redirect the trolley onto a parallel track where there is only one person in its path.

You have two options:

1. Do nothing and allow the five people to be killed.
2. Pull the lever and allow the one person to be killed.

Which choice is the *correct* choice?

Most people tend to agree that pulling the lever is the *better* choice since it lessens the loss of life from five victims to only one. However, an alternate scenario that also results in only one loss of life is not quite so clear-cut. Instead of a lever and a parallel track, consider the following variation:

The Fat Man Variation

Again, there is a runaway trolley bearing down on five inevitable victims.

This time, however, you notice a very fat man standing next to the tracks. You quickly conclude that shoving the fat man into the path of the speeding trolley will derail and stop the trolley while killing the fat man, but save the five people farther down the track.

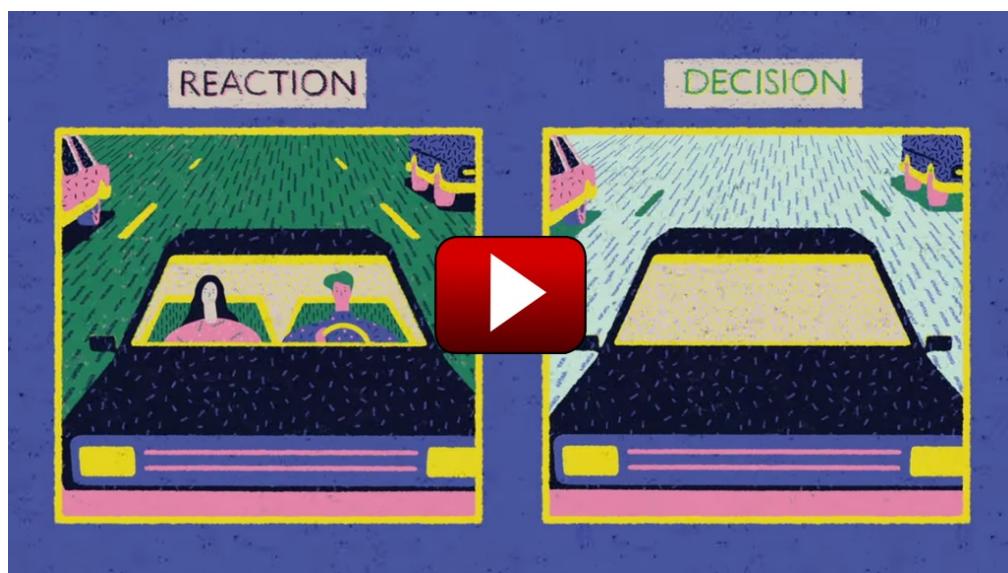
You have two options:

1. Do nothing and allow the five people to be killed.
2. Push the fat man onto the track causing him to be killed.

Now which choice is the *correct* choice?

How is this scenario different from the first? And why do so many people say that they could not sacrifice the fat man while they were perfectly willing to sacrifice the sole person standing on the parallel track? It is quite a dilemma and shows just how difficult it can be to know what the *right* solution is in various scenarios.

In these hypothetical thought experiments, it may be easy to dismiss the ethical challenges because the situations are not *real* and there are no *real* victims, only *imaginary* victims. But with the advent of autonomous vehicles, the potential victims are very real and the programmers who design the algorithms that dictate how a car chooses between different options must deal with these very *real* issues. The code that they write might literally mean the difference between life and death.



<https://www.youtube.com/embed/ixIoDYVfKA0>

Whose Fault Is an Accident?

Accidents happen. Nobody wants, expects, or intends for them to happen, but they do happen nonetheless. In the case of human-driven vehicles, years of precedence allow us to assign liability to the appropriate driver(s) who are responsible for the action. It is usually quite straightforward. If car A causes the accident with car B, then the driver of car A is clearly at fault.

However, with autonomous vehicles, this question becomes a bit more nebulous. If self-

driving car A causes an accident, who is the *driver* of record? The car itself? The owner of the car that caused the accident? The passenger(s) of the car who instructed the car to drive them through that intersection? The carmaker that designed and sold the car as a safe vehicle? The team of programmers whose code made the decision that led to the accident? Or is the responsibility shared between some number of these parties? And if so, which parties and in which proportions?

These are not easy questions to answer and the laws regarding autonomous vehicles (or other autonomous devices) have traditionally not kept up with the pace of new technologies. Inevitably, the necessary laws will be written and many of these questions will be resolved, as they have with other technological advances of the past. But for now, things are anything but clear.

But this issue does introduce a fascinating complication of any technological advancement that impacts the full spectrum of individuals involved in the application of new technology. At the coding and design end of the spectrum, programmers and engineers need to be well-versed in ethics and the law, especially as they attempt to model such behavior in the decision-making algorithms of their devices. And on the opposite end of the spectrum, lawmakers and legal authorities need to have a solid understanding of technology and its capabilities and limitations in order to write sound and enforceable legislation.

Exercise

As legislators draft new laws and automakers develop new autonomous vehicle technologies, both groups often rely upon a consensus of expert opinions with regard to the ethics and responsibilities of these new innovations. They use these agreed-upon sets of policies and standards to shape and guide their development of new technologies and the laws that apply to them.

Draft a set of policies for autonomous vehicles that address situations like those described in the "trolley problem" or the video above.

As a class, you will develop a set of policies that should be clear enough and detailed enough that lawmakers could use them as the basis for drafting new legislation and carmakers could use them as guides for developing new self-driving algorithms for their automobiles.

Initially, as an entire class, you will discuss the issues related to autonomous vehicles and brainstorm ideas and the goals that you wish your policies to achieve. Next, you will divide into small groups, with each group focusing on the development of one or two policy ideas in detail. Finally, the groups will share their results with the full class who will then discuss, revise, and select the various policy ideas that will make up the standards for the class'

overall, recommended autonomous vehicle policy.

As you develop and discuss your new policies, you should make sure to generalize your definitions such that they can apply to a broad variety of scenarios. It would be helpful to consider examples of similar scenarios to which your policy *should* apply and those to which it *should not* apply. Your policies should include the following elements:

- Use clear and precise language.
- Clearly describe the conditions/scenarios to which the policy applies and does not apply.
- Clearly state how difficult decisions, such as in the "trolley problem", should be made according to your policy.
- Focus on how to determine what the "correct" decision is and how that decision might be enforced, both in law and in code.

UNIT PROJECT:

Future Technology

Highlights

- You will collaborate in pairs to envision and design a future innovation in technology.
- You will discuss and identify a specific purpose that your innovation will serve (e.g., entertainment, problem solving, education, artistic expression, etc.) and its key features.
- You will evaluate the potential benefits and risks of your innovation.
- You will identify existing technological resources that your innovation may utilize.
- You will identify technological challenges that must be overcome before your innovation can be fully realized.
- You will develop a mock-up of your innovation that demonstrates its use and functionality.
- You will write a detailed product description and deliver an elevator pitch to the class detailing the features of your innovation and its potential impact on society using appropriate terminology.
- You will provide written feedback to your peers on the potential of each collaborative team's design.

Future Technology Project: Rubric Check



Feedback

This activity provides you and a partner group time to provide one another *critical feedback* about the progress of your projects.

1. Pair up with one another group.
2. Each group will present a mock presentation of their Future Technology talk. Any artifacts required for the presentation (e.g., infographics, graphs, etc.) should be presented as well.
3. Rate the components of your partner group's Processing program and documentation according to the [Future Technology Project](#). You may write this out or print the rubric and circle components on it.
4. Review your partner group's work and provide them with documentation that describes what you *Like*, what you *Wonder*, and what their *Next Steps* should be.
5. When both you and your partner group have completed the previous steps, share your feedback with one another.

Listen to what your partner group says! You do not need to heed all of their advice, but consider it wisely. The whole idea is to have a critical third party provide ideas to make your project better before you submit the final version for a grade.

Performance Tasks

This unit serves to fulfill the Performance Task requirements of the AP Computer Science Principles exam. This externally moderated assessment will account for 40% of the student's AP exam score. As such, the work produced in this unit should reflect the sole work of the student and performed in-class with minimal involvement from the classroom teacher. For the "Create" Performance Task, the student may receive collaborative support from a fellow student, but the work submitted should be the individual student's own work.

By this point in the course, all of the projects, exercises, and classroom discussions from the previous six units will have provided students with extensive, hands-on experience with the exploration, use, and creation of computational artifacts in a variety of contexts. In this unit, students will draw upon those collective skills to demonstrate mastery of essential course concepts by completing the "Explore" and "Create" Performance Tasks that make up the AP through-course assessment.

For the "Explore – Impact of Computing Innovations" Performance Task, students will demonstrate their ability to conduct independent research into an innovative technology and intelligently discuss its impact and influence on society as a whole. And for the "Create – Applications from Ideas" Performance Task, students will demonstrate their ability to work individually and collaboratively to design and develop a functional program for solving a problem and/or self-expression.

AP CSP Course Assessment

Exam Overview

The following information is available in the full AP Computer Science Principles [Course and Exam Description](#). See page 103 for a detailed overview of the exam and performance task requirements.

AP Computer Science Principles

Handout

AP Computer Science Principles Assessment Overview for Students

The AP Computer Science Principles assessment is composed of two through-course performance assessment tasks and one end-of-course exam. All three assessments are summative and will be used to calculate a final AP score (using the 1–5 scale) for AP Computer Science Principles.

Component	Timing	Percentage of Total AP Score
Explore Performance Task	8 hours	16%
Create Performance Task	12 hours	24%
End-of-Course Exam	2 hours	60%

Students who are completing the AP Computer Science Principles course in a nontraditional classroom situation (e.g., online, homeschool, independent study) should consult a school-based AP Coordinator for instructions on taking the AP Exam and submitting work for the performance tasks.

Investigation and Citation

The through-course performance assessment requires you to create computational artifacts. One of these artifacts will be a program; the other artifact allows you to choose any computational media you wish to use. A computational artifact is a visualization, a graphic, a video, a program, or an audio recording that you create using the computer. In creating a computational artifact, you should avoid plagiarism by acknowledging, attributing, and/or citing sources and including a bibliography with your submission. Sources that should be cited include text, images, graphs, and program code that are used in the creation of your computational artifacts.

When completing the Explore – Impacts of Computing Innovations performance task, you will be expected to conduct investigations on a computing innovation. You must ensure you have identified relevant, credible, and easily accessible sources to support your creation of a computational artifact as well as to support your responses to the prompts. You can search for print or nonprint sources as part of your investigation. You can refer to a journal, Web page, or an expert that is being quoted as part of your written response. Avoid plagiarism by acknowledging, attributing, and/or citing sources throughout your responses.

AP Computer Science Principles Policy on Plagiarism

A student who fails to acknowledge (i.e., through citation, through attribution, by reference, and/or through acknowledgment in a bibliographic entry) the source or author of any and all information or evidence taken from the work of someone else will receive a score of 0 on that particular component of the performance assessment task.

To the best of their ability, teachers will ensure that students understand ethical use and acknowledgment of the ideas and work of others as well as the consequences of plagiarism. The student's individual voice should be clearly evident, and the ideas of others must be acknowledged, attributed, and/or cited.

Programming Language Requirements

AP Computer Science Principles is language agnostic. This means that there is no specific language requirement. When completing the Create – Applications from Ideas performance task for this course, you are allowed to select a language you feel is most appropriate to meet the requirements of the task. When selecting a language or program, you should review the requirements section of the performance task to ensure that your program will be sophisticated enough to implement mathematical and logical concepts, create abstractions, and implement algorithms.

Peer-to-Peer Collaboration

For the Create – Applications from Ideas performance task, you are encouraged to collaborate with another student in your class.

Students completing AP Computer Science Principles in a nontraditional classroom situation (e.g., online, homeschool, independent study) are encouraged to collaborate with another student peer.

Preparing for the Through-Course Assessment

The through-course assessment consists of two performance tasks. The following guidelines are meant to help you be successful on the performance tasks as well as to clarify or address any questions you may have regarding the process of completing these tasks.

Prior to your teacher administering the performance tasks, you should:

- ▶ obtain content knowledge and skills that will help you succeed on the performance tasks;
- ▶ practice either an entire task or components of the tasks;

- ▶ review the rubrics to understand how your work will be assessed;
- ▶ examine examples of performance task submissions at high, medium, and low levels, and to avoid potential plagiarism issues, consider carefully how you will cite these appropriately as you create your own computational artifacts for each performance task; if you choose a similar topic, your treatment of the topic must be unique;
- ▶ pay attention to the instructions concerning the size of the file to be uploaded;
- ▶ ensure you know the proper way to evaluate and appropriately cite a source, including program code; any program code which has not been written by you must be cited and credit given to the author;
- ▶ understand the level of detail expected in writing your responses;
- ▶ understand that you may not revise your work once you have completed the submission process of the official administration of the task; and
- ▶ be aware that the scoring process that occurs in the AP Reading may be different from the scoring process that occurs in your classroom; the AP score that you receive may be different than your classroom grade.

Preparing for the Explore – Impact of Computing Innovations Performance Task

Prior to your teacher administering this task, you should:

- ▶ obtain the meaning and purpose of creating a computational artifact; your creation could solve a problem, show creative expression, or provide the viewer with new insight or knowledge;
- ▶ understand that a computing innovation (i.e., an innovation that depends on computing or computing tools to define its functionality) that has a meaningful personal or community emphasis is an appropriate choice, as long as it fulfills the requirements to have an impact on society, economy, and culture;
- ▶ clearly identify beneficial and harmful effects of computing innovations;
- ▶ practice searching and evaluating sources relevant to computing innovations; all sources cited must be relevant, credible, and easily accessible;
- ▶ practice writing responses based on relevant and credible sources;
- ▶ practice appropriate citation of sources used in the creation of your computational artifact;
- ▶ have exposure to the use of a variety of computational tools that can be used to create effective artifacts; and

- understand which computational artifacts would be considered effective and ineffective.

Effective artifacts include:

- visual, graphical, and/or audio content to help a reader understand the purpose of a computing innovation; and
- the use of communications media, such as animations, comic strips, infographics, and/or public service announcements, to illustrate the purpose of a computing innovation.

Ineffective artifacts include:

- artifacts that repeat information previously supplied in the written responses;
- multislide presentations with paragraphs of text or bullets;
- artifacts that have not been created by the student; and
- artifacts that focus on advertising the computing innovation's functionality instead of the purpose of the innovation.

Preparing for the Create – Applications from Ideas Performance Task

Prior to your teacher administering this task, you should:

- ensure you know effective ways to collaborate; and
- obtain programming support as necessary while practicing the skills needed to complete the performance task.

Guidelines for Completing the Through-Course Assessment

The through-course assessment consists of two performance tasks.

You must:

- be aware of the task, timeline, components and scoring criteria.

It is recommended that you:

- follow a timeline and schedule for completing the performance task;
- seek clarification from your teacher or AP Coordinator pertaining to the task, timeline, components, and scoring criteria;
- seek clarification from your teacher or AP Coordinator regarding submission requirements;
- allow your own interests to drive your choice of computing innovation and program;
- as needed, seek assistance from your teacher or AP Coordinator in defining your focus and choice of topics;

- ▶ use relevant and credible sources to gather information about your computing innovation when completing the Explore performance task;
- ▶ seek assistance from your teacher resolve technical problems that impede work, such as a failing workstation or difficulty with access to networks, or help with saving files or making movie files;
- ▶ obtain assistance from your teacher or AP Coordinator with the formation of peer-to-peer collaboration when completing the Create performance task;
- ▶ seek assistance from your teacher or AP Coordinator in resolving collaboration issues where one partner is clearly and directly impeding the completion of the Create performance task; and
- ▶ seek guidance from your teacher or AP Coordinator to use and cite APIs or other pieces of open-source code. Program code not written by you can be used in programs as long as you are extending the project in some new way. You should provide citation and credit for programming code you did not write.

You may not:

- ▶ submit work that has been revised, amended, or corrected by another individual, with the exception of cited program code;
- ▶ submit work from a practice performance task as your official submission to the College Board to be scored by the AP Program; or
- ▶ seek assistance or feedback on answers to prompts.

The "Explore" Performance Task

Performance Task Instructions

Refer to the following assignment description and submission requirements as you complete the **Performance Task: Explore — Impact of Computing Innovations**. These documents can be found in the [AP Computer Science Principles Course and Exam Description](#), starting on page 108.

AP Computer Science Principles

Handout

Performance Task: Explore – Impact of Computing Innovations

Overview

Computing innovations impact our lives in ways that require considerable study and reflection for us to fully understand them. In this performance task, you will explore a computing innovation of your choice. Your close examination of this computing innovation will deepen your understanding of computer science principles.

Please note that once this performance task has been assigned as an assessment (rather than as practice), you are expected to complete the task with minimal assistance from anyone. For more clarification see the Guidelines for Completing the Through-Course Assessment section.

You will be provided with 8 hours of class time to develop, complete, and submit the following:

- ▶ A computational artifact
- ▶ Written responses

Scoring rubrics and instructions for submitting your performance tasks are available on the AP Computer Science Principles Course Home Page.

Note: Students in nontraditional classroom environments should consult a school-based AP Coordinator for submission instructions.

General Requirements

This performance task requires you to select and investigate a computational innovation that:

- ▶ has had or has the potential to have significant beneficial and harmful effects on society, economy, or culture;
- ▶ consumes, produces, and/or transforms data; and
- ▶ raises at least one data storage concern, data privacy concern, or data security concern.

You are also required to:

- ▶ investigate your computing innovation using a variety of sources (e.g., print, online, expert interviews);

- cite at least three sources that helped you create your computational artifact and/or formulate your written responses;
 - At least two of the sources must be available online or in print; your third source may be either online, in print, or a personal interview with an expert on the computing innovation.
 - At least two of the sources must have been created after the end of the previous academic year.
- produce a computational artifact that illustrates, represents, or explains the computing innovation's intended purpose, its function, or its effect; and
- provide written responses to questions about your computational artifact and computing innovation.

Submission Requirements

1. Computational Artifact

Your computational artifact must provide an illustration, representation, or explanation of the computing innovation's intended purpose, its function, or its effect. The computational artifact must not simply repeat the information supplied in the written responses and should be primarily nontextual.

Submit a video, audio, or PDF file. Use computing tools and techniques to create one original computational artifact (a visualization, a graphic, a video, a program, or an audio recording). Acceptable multimedia file types include .mp3, .mp4, .wmv, .avi, .mov, .wav, .aif, or .pdf format. PDFs must not exceed three pages. Video or audio files must not exceed 1 minute in length and must not exceed 30MB in size.

2. Written Responses

Submit one PDF file in which you respond directly to each of the prompts below. **Clearly label your responses 2a–2e in order.** Your responses must provide evidence of the extensive knowledge you have developed about your chosen computing innovation and its impact(s). Write your responses so they would be understandable to someone who is not familiar with the computing innovation. Include citations, as applicable, within your written responses. **Your response to prompts 2a–2d combined must not exceed 700 words.** The references required in 2e are not included in the final word count.

Computational Artifact

2a. Provide information on your computing innovation and computational artifact.

- ♦ Name the computing innovation that is represented by your computational artifact.
- ♦ Describe the computing innovation's intended purpose and function.
- ♦ Describe how your computational artifact illustrates, represents, or explains the computing innovation's intended purpose, its function, or its effect.

(Approximately 100 words)

2b. Describe your development process, explicitly identifying the computing tools and techniques you used to create your artifact. Your description must be detailed enough so that a person unfamiliar with those tools and techniques will understand your process. (*Approximately 100 words*)

Computing Innovation

2c. Explain at least one beneficial effect and at least one harmful effect the computing innovation has had, or has the potential to have, on society, economy, or culture. (*Approximately 250 words*)

2d. Using specific details, describe:

- ♦ the data your innovation uses;
- ♦ how the innovation consumes (as input), produces (as output), and/or transforms data; and
- ♦ at least one data storage concern, data privacy concern, or data security concern directly related to the computing innovation.

(*Approximately 250 words*)

References

2e. Provide a list of at least three online or print sources used to create your computational artifact and/or support your responses to the prompts provided in this performance task.

- ♦ At least two of the sources must have been created after the end of the previous academic year.
- ♦ For each online source, include the permanent URL. Identify the author, title, source, the date you retrieved the source, and, if possible, the date the reference was written or posted.
- ♦ For each print source, include the author, title of excerpt/article and magazine or book, page number(s), publisher, and date of publication.
- ♦ If you include an interview source, include the name of the person you interviewed, the date on which the interview occurred, and the person's position in the field.
- ♦ Include citations for the sources you used, and number each source accordingly.
- ♦ Each source must be relevant, credible, and easily accessed.

The "Create" Performance Task

Performance Task Instructions

Refer to the following assignment description and submission requirements as you complete the **Performance Task: Create — Applications from Ideas**. These documents can be found in the *AP Computer Science Principles Course and Exam Description*, starting on page 111.

AP Computer Science Principles

Handout

Performance Task: Create – Applications from Ideas

Overview

Programming is a collaborative and creative process that brings ideas to life through the development of software. Programs can help solve problems, enable innovations, or express personal interests. In this performance task, you will be developing a program of your choice. Your development process should include iteratively designing, implementing, and testing your program. You are strongly encouraged to work with another student in your class.

Please note that once this performance task has been assigned as an assessment (rather than as practice), you are expected to complete the task with minimal assistance from anyone other than your collaborative partner. For more clarification see the Guidelines for Completing the Through-Course Assessment section.

You will be provided with 12 hours of class time to complete and submit the following:

- ▶ A video of your program running
- ▶ Individual written responses about your program and development process
- ▶ Program code

Scoring rubrics and instructions for submitting your performance tasks are available on the AP Computer Science Principles Course Home Page.

Note: Students in nontraditional classroom environments should consult a school-based AP Coordinator for instructions.

General Requirements

This performance task requires you to develop a program on a topic that interests you or one that solves a problem. It is strongly recommended that a portion of the program involve some form of collaboration with another student in your class. Your program development must also involve a significant amount of independent work in the planning and designing parts of the process.

You are required to:

- ▶ iteratively design, implement, and test your program;
- ▶ independently create at least one significant part of your program;
- ▶ create a video that displays the running of your program and demonstrates its functionality;

- ▶ write responses to questions about your program; and
- ▶ include your entire program code.

Program Requirements

Your program must demonstrate a variety of capabilities and implement several different language features that, when combined, produce a result that cannot be easily accomplished without computing tools and techniques. Your program should draw upon mathematical and logical concepts, such as use of numbers, variables, mathematical expressions with arithmetic operators, logical and Boolean operators and expressions, decision statements, iteration, and/or collections.

Your program must demonstrate:

- ▶ use of several effectively integrated mathematical and logical concepts, from the language you are using;
- ▶ implementation of complex algorithms based on mathematical and logical concepts; and
- ▶ development and use of abstractions to manage the complexity of your program (e.g., procedures, abstractions provided by the programming language, APIs).

Submission Requirements

1. Video

Submit one video in .mp4, .wmv, .avi, or .mov format that demonstrates the running of at least one significant feature of your program. **Your video must not exceed 1 minute in length and must not exceed 30MB in size.**

2. Written Responses

Submit one PDF file in which you respond directly to each prompt. **Clearly label your responses 2a–2d in order. Your response to all prompts combined must not exceed 750 words, exclusive of the Program Code.**

Program Purpose and Development

2a. Provide a written response or audio narration in your video that:

- ♦ identifies the programming language;
- ♦ identifies the purpose of your program; and
- ♦ explains what the video illustrates.

(Approximately 150 words)

2b. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development. *(Approximately 200 words)*

- 2c. Capture and paste the program code segment that implements an algorithm (marked with an **oval** in **section 3** below) that is fundamental for your program to achieve its intended purpose. Your code segment must include an algorithm that integrates other algorithms and integrates mathematical and/or logical concepts. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. (*Approximately 200 words*)
- 2d. Capture and paste the program code segment that contains an abstraction you developed (marked with a **rectangle** in **section 3** below). Your abstraction should integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. (*Approximately 200 words*)

3. Program Code

Capture and paste your entire program code in this section.

- › Mark with an **oval** the segment of program code that implements the most complex algorithm you created for your program.
- › Mark with a **rectangle** the segment of program code that represents an abstraction you developed.
- › Include comments or citations for program code that has been written by someone else.

UNIT A1

Artificial Intelligence: *Turing Test*

As computing devices grow more powerful, they are increasingly used to augment or replace human labor through the simulation of human skills and behaviors. In this unit, you will explore some of the ways in which computer scientists incorporate “artificial intelligence (AI)” into their algorithms, and what the philosophical implications of these advances may mean for the future. You will examine the use of a number of AI-infused applications, and finally, evaluate these applications using the standard metric for general AI—the *Turing Test*.

Turing Test Project

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human." – Alan Turing



<https://www.youtube.com/embed/Dccwpc0WY0w>

Introduction

As computation becomes more efficient, software is rapidly replacing human workers in the labor market. Robots now work assembly lines, the Internet has replaced the postal service as the primary means for distributing written communication, and electronic kiosks have replaced information desks and even, in some cases, waiters. These are examples of physical labor—but what about mental labor? Can computer software ever rival human intelligence? Will computers replace doctors, educators, or even artists?

What does it mean to be intelligent? How can we measure intelligence?

Assignment

Your job is to write a protocol for conducting a Turing Test that can distinguish artificial intelligence from human intelligence using only text-based chat.

Submission

Submit a write-up of your Turing Test protocol experiment. The protocol should include the following:

- an introduction that states the purpose of the test and its underlying principles,
- at least 10 detailed directives for someone to complete in order to identify artificial intelligence through text-based chat (these steps should be so detailed that any two people performing the Turing Test would do so in exactly the same way),
- quotations used as necessary to indicate exact text that must be used (e.g., questions

to ask),

- example responses for each of the questions, describing in detail how text responses should be analyzed (e.g., differences in expected and actual output for a given input), and
- an analysis of the strengths and weaknesses of the protocol.

Learning Goals

You will be able to:

- Comprehend how the mechanical manipulation of symbols is a form of *problem solving*.
- Analyze a “chatterbot” for pattern recognition and manipulation.
- Evaluate multiple approaches for designing a Turing Test for both effectiveness and generalizability.
- Differentiate between “strong” and “weak” AI.
- Identify relationships between the AI subfield of Natural Language Processing (NLP) and Human-Computer Interface (HCI).
- Describe visual and speech recognition as a form of multi-modal AI.
- Discover, implement, and exploit strategies to discern between human and artificial intelligences.

Criteria	Points
TURING TEST PROTOCOL	
Describes clear and concise steps for the administration of your Turing Test experimental design.	3 pts
Explains how the final Turing Test design assures that the experiment is effective and generalizable.	3 pts
Uses key terminology from the Artificial Intelligence glossary appropriately and effectively.	2 pts
TURING TEST RESULTS	
Provides background information for the analysis of the chatterbots’ AI	3 pts
Provides a detailed, sequential account describing the administration of your Turing Test to two different chatterbots	2 pts
TURING TEST EVALUATION	
Describes the two chatterbots, analyzing their strengths and weaknesses, and pattern recognition and manipulation abilities.	4 pts
Includes specific examples as evidence	2 pts
Commits few, if any, grammatical, organizational, or spelling errors	1 pt

TURING TEST SYNTHESIS	
Your Scratch program:	
Responds generally to user input	4 pts
Responds specifically to selective phrases	2 pts
Utilizes previous responses (memory)	2 pts
Includes useful and clear documentation	2 pts
TOTAL	30 pts

What Is A Chatterbot?

What Is A Chatterbot?

Q: What is a chatterbot?

A: A robot that won't shut up!

A **chatterbot** is a computer program designed to simulate an intelligent conversation with one or more human users through audio, video, or text. From the birth of electronic computers, computer scientists have conjectured about the possibility of communicating with machines as if they were humans.

For example, in *Wargames* (1983), "a young man finds a back door into a military central computer in which reality is confused with game-playing, possibly starting World War III." ([IMDB](#))



<https://www.youtube.com/embed/D-9l5jSDL50>

A Common Misconception

Computer software, like Siri or Alexa, can understand language and communicate.

This is false! Computers can manipulate binary-encoded symbols or values. Good responses that seem thoughtful are really just generated from automated reasoning routines (e.g., `IF ASKED x, RESPOND WITH y`). Huge advances have been made by tailoring how the responses are generated. Initially, they were hand-coded; now, they are usually the result of sophisticated statistical modeling.

Recently, [freakishly realistic telemarketing robots are denying they're robots](#). Follow the link, read the article and listen to the sound recordings. Would you be convinced by the 'real' telemarketer? Read the update to the story (as linked from within the article). How does this

change things?

Assignment

1. Play with and analyze the strategies used by these 3 different chatbots for a few minutes each:
 1. [Eliza](#)
 2. [Cleverbot](#)
 3. [Program-O](#)
2. Take notes on the strengths/weakness of each.
3. After you have tested and recorded your notes for each of the chatterbots, brainstorm a response to the question, "What are some common strategies that might 'break' the system and expose the AI as not human?"

Write a paragraph that summarizes the strategies you have considered.

Black-Box Testing Chatterbots

Black-Box Testing

When using a software application, it is rare for the user to read—or even have access to—the source code. The only real observable components of the system are what goes in (i.e., input, such as clicking, mouse movement, data entry) and what comes out (i.e., output, such as video, printout, gameplay, or text on the screen). The software itself is no different than a featureless “**black box**”. The user may have no idea *how* the software works, but she definitely can discern *what it does*.



So, given a black box, how might one check to see if it works properly? The user typically knows not only *what it does*, but what she expects it to do—*what it should do* or *what is acceptable for it to do*. Feeding data to a system and comparing the output to expectations is called **black-box testing**.

Black-box testing may seem like a compromise when testing a lot of software, because the source code *is* visible. Consider the millions of Scratch programs on the [Scratch website](#). Why go through the trouble of black-box testing when you can just examine the code blocks?

Well, errors are often caught when a system is used in a way not originally intended, or when specific **boundary cases** occur. In order to find unknown errors, it is often useful to *abstract* away the inner functionality of a system (i.e., *how it works*) and concentrate on its input and output (i.e., *what it does*). Comparing expected and actual results can spotlight which component of the system is at fault.

Activity

In this jigsaw activity, you will experiment with different chatterbots in order to better understand their behavior patterns.

Specifically, you will:

1. *work* with team members from other groups to perform a black-box test on **one** of the chatterbots below. To black-box test the chatterbot, ask it a list of questions, and record and compare its responses to possible reasonable answers. Using these responses, form a theory of how the chatterbot works.
2. *share* what you learned about that particular chatterbot with your project group.
3. *outline* a **general** method for discovering patterns in a chatterbot's automated reasoning.

Jigsaw instructions

Your teacher will assign each member in your group a number. You will sit with students from other groups assigned the same number (e.g., students assigned #1 sit together). Each numbered group will analyze the chatterbot that corresponds with its number:

1. [Eliza](#)
2. [Cleverbot](#)
3. [Program-O](#)

Part I. In your *numbered* groups:



- Black-box test your assigned chatterbot in order to figure out how it functions (10–15 min.)
- Create a list of questions to ask the chatterbot.
- Ask your assigned chatterbot the questions and record its responses.
- Identify anomalies ('mistakes') that the chatterbot produces. Note: An anomaly occurs when the 'bot generates a response that is *unexpected*.
- Detail and test a hypothesis for the reasoning behind each problematic response. Revise and retest your hypothesis as needed.
- Prepare to share your theory, analysis, and evidence with your project group.

For example:

Hypothesis

I hypothesize that Dr. Romulon is not easily able to parse the negation when phrased as **do YOU not**. It seems to pick up the pattern of **don't**—which is a single word. Perhaps it would understand **do not** if it were collocated (i.e., occur in the same location) in place of **don't**, even though that is awkward sounding to a human speaker.

Test

Human: What do you do?

'Bot: I talk to people on the web. What do you do?

Human: What do you not do?

'Bot: I like to meet new people online.

Human: What don't you do?

'Bot: There are many things still mysterious to me. I am just beginning.

Human: What do not you do?

'Bot: There are many things still mysterious to me. I am just beginning.

Conclusion

The hypothesis is borne out by this pattern. Of course, this is just another anecdote, and it may be that other cases of "do YOU not" are hand-coded to work properly.

Part II. In your *project groups*:

Reconvene in your project groups.

1. Share what you learned about each chatterbot with your team members.
2. **Outline a method for discovering patterns in *any* chatterbot's automated reasoning.**
 - Include specific steps.
 - Test your method. Does the method generate patterns in all of the chatbots' responses? If not, which ones and why?
 - Revise your methods of how the chatterbots function to accommodate any new discoveries, if necessary.
 - Be prepared to discuss your work with the class.

Part III. As a class:

Discuss your findings. Address the following questions:

1. "What were some question/answer pairs that gave you clues about how the chatterbots worked?"
2. "Which chatterbot was the most effective? Why?"
3. "How might you expose the chatterbot as non-human? Would this strategy always work?"
4. "What could you do differently in creating a chatterbot, so that it might seem more human?"

Unspecified Input

Common Misconception

Bots are preprogrammed with specific question/response pairs.

This is false. An easy way to test this is to substitute out one word for another and compare

responses. To illustrate, consider the following trivial example:

"What is your name?"	"What is your name?"
"Dave."	"Frank."
"Hello, Dave. I'm HAL."	"Hello, Frank. I'm HAL."

A hard-coded question/answer response system would require a separate routine to handle each possible name—or even each possible wording/phrasing of a question (e.g., "What is your name?" vs. "What's your name?").

How might programs allow developers to utilize unspecified input? Think *programmatically* (e.g. algorithmically, with Scratch blocks or Processing scripts in mind, etc.).

Consider L'il Johnny McPixel™

Tyrell Corporation introduced the *L'il Johnny McPixel™* program in 2020, replacing the venerable Siri personal assistant application. Commonly known as the MCP, the AI program is a vast improvement over its predecessor, with the capability to learn from interactions with each of its users—roughly 3 billion worldwide.

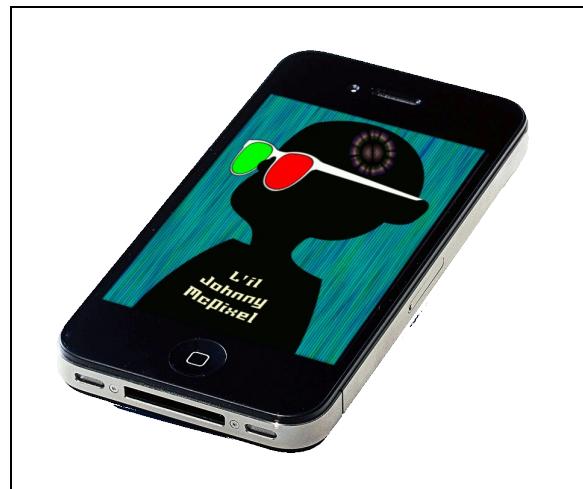
Unlike Siri, the MCP is able to incorporate updates to its software dynamically, improving its interface and performance through advanced machine learning techniques. Its exponential growth in performance has sparked debate among its user-base. The interactions that some users experience with their MCPs feel so real, so human-like, that petitions have been made to legally declare each individual MCP instance as a 'person'. In fact, some petitioners have indicated that they intend to *marry* their McPixels once the legal proceedings are complete.

Protestors have been picketing in Washington, D.C. for the rights of their MCPs.

Is L'il Johnny McPixel™ *human*?

Additional known facts about L'il Johnny McPixel™:

- The MCP is not a central program shared by all users; each user has its own *instance*.
- The MCP centrally incorporates data from all users and pushes out updates to each device incorporating new changes.
- The MCP is designed to perform personal assistant tasks (e.g., make appointments, set reminders, perform searches, answer questions). The MCP is also outfitted to be conversational, learning by imitation of human interactions worldwide.
- Each individual MCP contains both a centralized knowledge base and an individual knowledge base. It can learn aspects of the local environment (e.g., its users' favorite colors, pet peeves, habits).
- Different MCP instances develop different mannerisms as reinforced by their users. This includes use of slang, general tone, and demeanor.
- The MCP is not designed as a general, all-purpose AI, but rather to perform certain tasks. However, some users have explained that the imitative, conversational nature of their MCPs includes human-like discussion of virtually any topic. Through imitation, MCPs have learned to indicate personal preferences, including stances on political platforms/candidates.
- MCPs do not *smell*, *taste*, or *feel*. They are limited to acquiring information through



sound and sight from external sensors, as well as Wi-Fi information exchange.

- Any MCP can be reverted to a prior version, whereupon it loses any learned functionality which is no longer supported.

What do you think?

Some Interesting Resources:

- A similar idea is explored in the 2013 Spike Jonze film, [*Her*](#).
- John Searle's Chinese Room thought-experiment: <http://www.zompist.com/searle.html>
- Pathological evidence that the brain is a form of machinery: http://www-rohan.sdsu.edu/~gawron/intro/course_core/lectures/aphasia_cases_slides.html
- Scientists discuss what it means to be human:
<http://www.wired.com/wiredscience/2008/06/what-does-it-me/>
- Free-will and determinism: http://en.wikipedia.org/wiki/Free_will;
http://en.wikipedia.org/wiki/Nomological_determinism
- The "Duck Test": http://en.wikipedia.org/wiki/Duck_test
- HAL 9000 scene (Is self-preservation its end motive?): <http://www.youtube.com/watch?v=HwBmPiOmEGQ>
- The Technological Singularity: [*Signs of the Singularity*](#) part of [*IEEE Spectrum's SPECIAL REPORT: THE SINGULARITY*](#)

The Humanity of L'il Johnny McPixel™?

What exactly does it mean to be “human” or a “person”?

In this activity, your class will hold a debate in which one side argues that **L'il Johnny McPixel™ is “human”** and the other argues that **L'il Johnny McPixel™ is *not* “human”**.

1. Your teacher will randomly assign students to be either proponents or opponents in the debate. *Students must argue their assigned points of view whether or not they personally agree with them!* This is important in ensuring that the debate remains challenging, balanced, and fun.
2. Re-read [Consider L'il Johnny McPixel™](#) to refresh your memory and prepare for the debate.
3. Use the [McPixel Debate Organizer](#) to prepare your arguments.
4. Examine outside sources:
 1. The 2013 Spike Jonze film, [Her](#)
 2. John Searle's [Chinese Room thought-experiment](#).
 3. Pathological evidence that the [brain is a form of machinery](#)
 4. Scientists discuss what it [means to be human](#)
 5. [Free-will and determinism](#)
 6. The "[Duck Test](#)"
 7. [HAL 9000 scene](#). (Is self-preservation its end motive?)
 8. The Technological Singularity: [Signs of the Singularity](#) part of [IEEE Spectrum's SPECIAL REPORT: THE SINGULARITY](#).
5. Debate the humanity (or lack thereof) of Johnny McPixel.



Your teacher will share the exact debate protocol with you before the debate begins. Remember to remain open-minded and to *listen* to the opposing view and respond appropriately.

Turing Test Project: Draft

In the classic 1982 science fiction movie [Blade Runner](#), Deckard is a police officer who serves as a type of AI “judge.” In the future, blade runners test for and identify potentially homicidal human “replicants” (androids). Watch this clip to get a better understanding of what exactly a blade runner like Deckard does in this envisioning of the future:



<https://www.youtube.com/watch?v=Yk3yQ5ktvWw>

In essence, the officers in *Blade Runner* perform Turing Tests, but have access to physiological responses for analysis (like the "[Voight-Kampff Test](#)" in the movie). Of course, Turing Tests today rely *only* on text-based chat, as computers have no physiology.

Turing Test strategies

The following are strategies that may be helpful in designing an effective Turing Test:

1. Use memory-dependent language.
 - Example: "Do you know **Barack**?"
"Would you like to meet **him**?"
2. Refer to activities normally considered as outside of the realm of 'pure thought', such as emotions, experience, physicality.
 - Example: "Does painting a picture make you **happy**?"
"*What style of painting do you like to paint?*"
3. Refer to aspects of consciousness ([qualia](#))—Imagine you teach a computer *all there is to know* physically about colors. Would it then experience “red” in the same way you do?
 - Example: "**Describe** the color **red**."
4. Give computationally difficult tasks.
 - Example: "What's the **square root of 1729**?"
5. Elicit reasoning.

- Example: After completing Example 1, follow with "Why would you like to meet him?"

Assignment

Your job is to write a draft protocol for a Turing Test that can distinguish human from artificial intelligence *using only text-based chat*. The protocol should include the following:

1. An introduction that states the purpose of the test and its underlying principles.
2. At least 10 detailed directives for someone to complete in order to identify artificial intelligence *only through text-based chat*.
 1. These steps should be so detailed that any two people performing the Turing Test would do so in *exactly* the same way.
 2. Use quotations as necessary to indicate exact text that must be used (e.g., questions to ask).
 3. Describe in detail how text responses should be analyzed (e.g., differences in expected and actual output for a given input).
3. An analysis of the strengths and weaknesses of the protocol.

Creating Intelligence

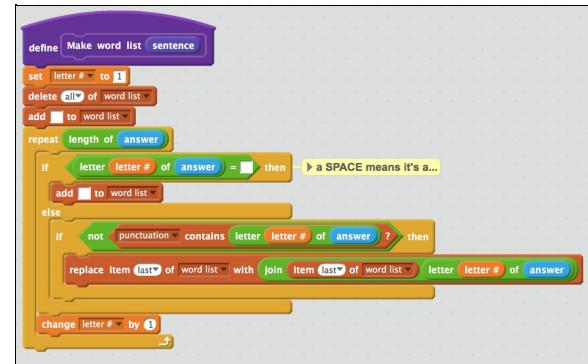
Turning Analysis into Synthesis

Now that you have spent some time *analyzing* chatbots, let's direct what you have learned toward *synthesizing* a new chatbot.

Cleverbot requires sophisticated machine learning techniques and vast amounts of data to achieve its realism. However, simple manipulation of words and phrases can go a long way, as seen with Eliza. It is likely that the chatbot you create will be more like Eliza than Cleverbot.

To begin, create a program that simply responds to user input. Then, you will want to break a sentence down into words, and respond differently according to the words that you extract.

To begin, you may wish to examine some simple examples:

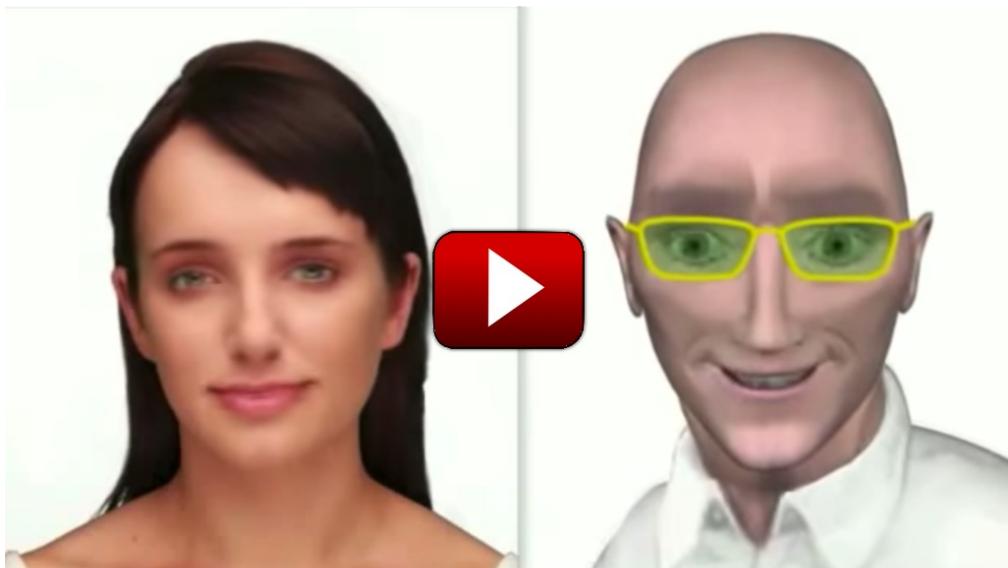


- The Scratch wiki contains a simple tutorial for creating the template for a chatbot—[Scratch Tutorial: Creating a Chat Bot](#).
- [Chatbot starter code](#) is provided for your use. You are **not required** to use any of this code, however. This code contains routines that:
 - separate a sentence into words,
 - remove punctuation from words, and
 - use a word from the previous response to construct a new question.

Each of these will get you started, but your final product will likely reflect the personality and tone that **you** create. Will your chatbot be *sassy?* *polite?* *aloof?* The choice is yours!

Turing Test Project: Experiment

What happens when one chatterbot converses with another chatterbot? Do they recognize each other as artificial intelligences?



<https://www.youtube.com/embed/zWuw7tWbrlw>

As you can see, the female chatbot immediately recognized that the male conversant was a chatbot. Perhaps the female recognized certain responses, or the answers from the male were too mechanical-sounding, or perhaps the male was inhumanly polite.

Is it *that* easy? Perhaps not...

Assignment

Now it's your turn! Your job is to apply your own Turing Test protocol to the 3 chatterbots from the [Black-box Testing Chatterbots](#) activity:

1. [Eliza](#)
2. [Cleverbot](#)
3. [Program-O](#)

First, download the [Turing Test - Practice spreadsheet](#) where you may record the results of your Turing Test experiments.

There are 5 individual worksheets in the spreadsheet. Each of these sheets describes a suggested strategy for you to try with each chatterbot. Experiment with different questions within each category and see how the chatterbot responds. Try to ask as many questions as possible, and ask each chatterbot the *same* questions to gauge how they respond *differently*. Think about the following questions in your analysis:

- How do the responses differ?

- Does the order in which questions are asked make a difference?
- Which questions are more **effective** for your purpose?
- Are there useful questioning strategies you have discovered that do not fit in the 5 pre-defined categories?

Class discussion

1. What happened when you used *memory-dependent language*?
2. What happened when you referred to activities outside of '*pure thought/cognition*'?
3. What were the chatterbots' perceptions of *consciousness*?
4. What happened with *computationally difficult* tasks?
5. What happened when you *elicited reasoning* (i.e., asked '*why*')?
6. Did you use any strategies beyond these? If so, what were they and how did they perform?
7. How can you use your knowledge of the experiences gained in this activity to design a better 'bot?
8. What would make for a better test, accounting for these possible modifications?

Experimenting with Visual Identification

In pairs, you will experiment with a process used in AI to define data by *features*. Features are used by AI algorithms to classify data through similarities and differences with other data in a dataset.

Activity

1. Pair up in groups of two. One person is designated *Student A* and the other *Student B*.
2. *Student A* will choose an image of a single animal, and define one attribute of the image (e.g. pink nose, etc.)
3. *Student B* will find an image online that matches the attribute *Student A* chose.
4. Determine if they are the same type of object (not necessarily the same *image*, however).
5. *Student A* will continue to add attributes until the objects match. Students will keep track of how many attributes were necessary in order to make the objects match.
6. After the objects match, switch roles and repeat the experiment. As you continue the experiment, improve upon selecting attributes that are the most useful.

A simple example of the activity

You start with these two **winged** objects:



Your first attribute for distinction may be **size**. Although this is a starting point for these objects, what if, instead of an airliner, it's a paper airplane?



Now the **size** distinction is irrelevant. Next, you choose the attribute **movable wings**.



The introduction of a butterfly causes yet another attribute to be necessary to distinguish between butterflies and birds. This process would continue until the two images are of the same category.

Feature selection is important in defining what distinguishes one object/concept from another.

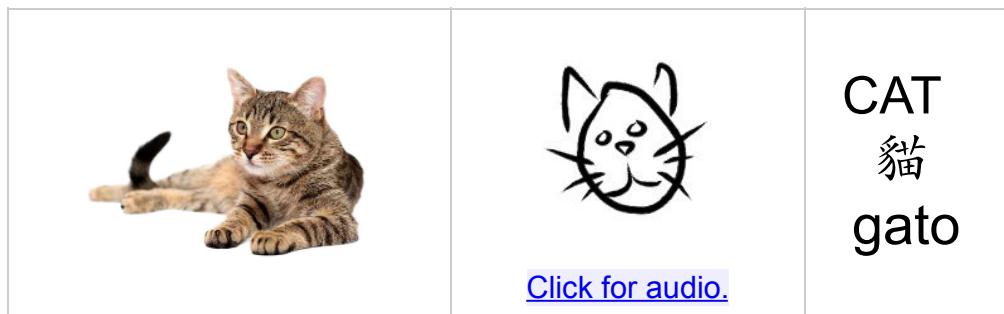
Object X is *this* and **Object Y** is *that* because they differ in these ways (features **a**, **b**, **c**,...).

Class Discussion

1. How many iterations of feature selection were necessary?
2. Are the features as relevant if you consider *any and all possible objects*?
3. Apply the strategies discussed to *visually ambiguous* pairs of objects. Given this [Venn diagram](#) of **Monarch** and **Viceroy butterflies**, how might you construct features that *disambiguate* the pair?
4. How is this similar to the [20 Questions](#) activity? What implications does 20 Questions have for disambiguation? **HINT:** Think about *optimal* feature selections.

Multi-modal Artificial Intelligence

Although each sensory system (e.g., visual, auditory, olfactory, tactile, gustatory) is generally thought of as distinct, they each contribute toward cognitive and linguistic function. Many times, the knowledge acquired through these different systems is integrated. Consider the following concept:



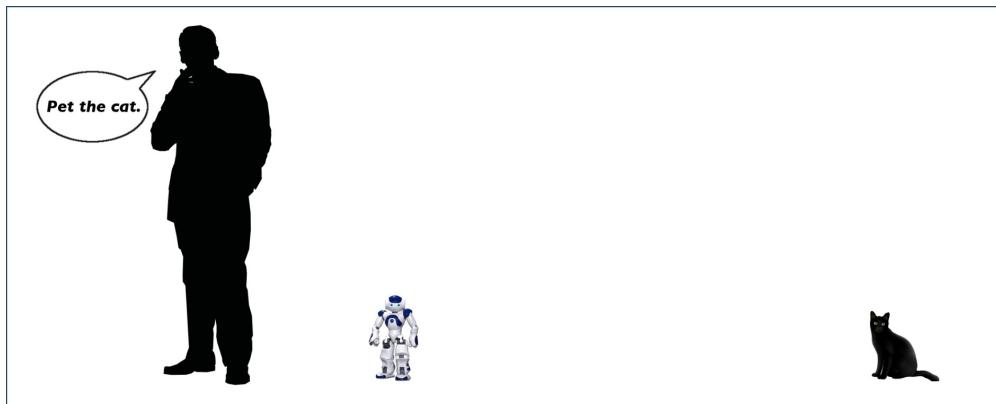
Each of the multimedia above are linked to the same concept, though they engage different modalities. Additionally, some people may link different subsets of these media to the concept. For example,

- a Chinese speaker who only speaks Chinese may not recognize *cat* or *gato*,
- an illiterate person may not recognize any of the words, and
- a deaf person may not incorporate the audio clip into his/her concept.

Multi-modal approach to petting a cat

Consider this situation in which you are interacting with a being with artificial intelligence.

You tell the robot, "Pet the cat."



In order for the robot to carry out the command, it must use different modes of acquiring data and must understand what you are saying by using the following:

- **Phonetics:** Speech recognition is necessary to translate audio to words.
- **Semantics:** The individual words must be relatable to objects and/or actions.
- **Syntax:** An understanding of English sentence structure is needed to understand what

the words mean when put together.

- **Pragmatics:** Ideally, the interface should infer intention—that this is more than a descriptive sentence, but a command directed to the robot.

Assuming the linguistic system is sufficient and the robot understands the command, the robot must also *physically* carry out the command. This requires integration of sensory and motor functions:

- **Visual recognition:** Identify the object to be interacted with; in this case, the cat.

Note that a connection is established here between the word **cat** and a visual of a cat. But, what if the object *looks* like a cat, but isn't. For example the Corgi below is *not* a cat, but looks similar to one:



One is a cat; the other is a Corgi.

How might the system **disambiguate** dog from cat when they are visually similar? What other sensory systems might contribute?

The robot will also need to execute motor skills in order to physically pet the cat. This includes:

- **Sensorimotor feedback:** Walking towards the cat.
- **Semantics:** Associate an action—and its motor control—with the action verb *pet*.

While this task may seem simple, remember that a robot has to *know* important information in order to perform this task. Consider all of the additional nuances involved in the simple statement "Pet the cat." How long does the robot pet the cat? How much pressure should it use? Where does it pet the cat? What if the cat moves when the robot approaches? It is a far more complex task than the 3 word command implies.

Common Misconception

A system (e.g., a robot) must be explicitly programmed to do any and all of the actions it can perform.

IF 'walk' THEN execute step...

This is not *quite* true, and the distinction between the statements

1. *a computer cannot do anything without being programmed to do it*
2. *a computer can learn to do X, Y, or Z*

is really a matter of abstraction. In fact, A.I. relies on this abstraction in defining what *intelligence* actually means.

The A.I. sub-field of **machine learning** is concerned with systems that learn connections among data *from* data. The programming required is not oriented toward manipulating known data as much as implementing strategies to learn from patterns in the data. For example, rather than program a robot to recognize objects with pointy ears, whiskers, and vertically slit pupils as *cats*, a machine learning system would learn these features (and/or possibly others) as common features of cats by training over a large set of images labeled *cat*. Researchers are currently working on *unsupervised* machine learning techniques, where the explicit training over samples is unnecessary.

Dasher

Role-play—Adaptive Technologies

Natural Language Processing (NLP) and **Machine Learning (ML)** can have a profound impact on how we interact with computers. We may not have computers that converse with us just yet, but the strides toward realizing this have enabled many advances in HCI (Human-Computer Interaction).

Rather than focus on obvious in-roads, such as Siri, let's look at an interface that utilizes NLP in a more subdued way.

Imagine that you are quadriplegic, without the use of your hands, feet, arms, or legs.

Movement is restricted to the neck up. You text friends through utilizing an a rod you control with your mouth to individually punch keys on a keyboard. It's exhausting and very slow. Making mistakes is painful; one mis-pressed keystroke becomes at least 3:

- the original mistaken character,
- the backspace key, and
- the (hopefully) correct keystroke.

Unfortunately, *hypothetical you* has also lost most of your speaking capacity. The motor neurons that control the muscles that produce speech have been damaged. Imagine how improvements in NLP and speech recognition have helped others with similar conditions. Instead of manually typing text, they may speak it aloud as the computer transcribes. Predictive techniques generally 'guess' the correct word when in doubt, and uncaught errors can then be corrected by keyboard.



Dasher

Dasher is an interface designed in the UK that has helped many in similar situations of impairment.

Explore the [Dasher site](#), and if possible, download and execute an instance of Dasher. Experiment with the interface, and record your answers to the following questions:

1. Some of the squares corresponding to letters are much larger than others. Also, sometimes one letter will be large, and other times, it will be relatively small. Why? What significance does the size of each letter's area have?

2. If you move randomly (but not wildly), actual words and phrases tend to pop up rather than plain gibberish. Why do you think that is?
3. How might this experience be different if the application were trained on a different language? Why?
4. What effect on performance do you anticipate if the application continually retrained on your input?
5. Spell some common words and phrases like, “Hello how are you today” and some uncommon words and phrases like, “my uncle used to love me but she died”. How do these experiences differ?
6. How is *Dasher* similar to *autocorrect*?

Dasher +

In 2016, an article was published describing an extension to Dasher to work through a brain-computer interface (BCI) entitled [*A Brain–Computer Interface for the Dasher Alternative Text Entry System*](#). The article describes “a new software tool was developed to allow subjects to type words onto a computer screen via Dasher using their thoughts.”

1. What are the implications for this technology?
2. Why is a BCI-enabled Dasher more accessible than a linguistic BCI system?

Ambiguity

Why do computer systems have a difficult time with ambiguity?

Computers are not yet capable of storing and processing the vast amounts of *contextual* data required for resolving ambiguities well. Consider the following scenario overheard by both a human and a speech recognition system:

Julia: Yuck. Do you know what *sweetbreads* are?

Julio: Yes. I've eaten a [pair, pear].

A speech recognition system is likely to choose *pear*, whereas a human (with the proper context) would most likely choose *pair*. Why do you think the computer would pick *pear*? Also, what could we do to the program to correct this incorrect word choice?

Ambiguity Rocks

What's the big deal with language anyway? Why is it so hard for computers to process it? After all, once we learn a language as a child, we don't really think too much about how it works (except maybe in English class).

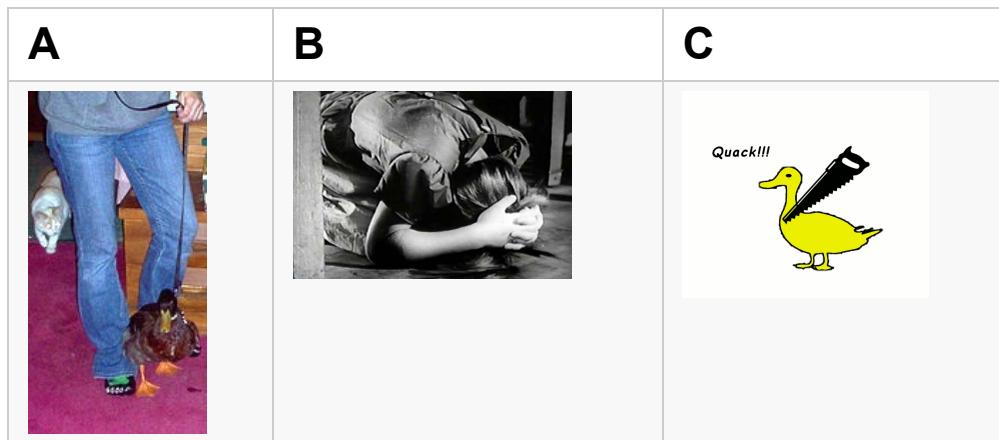
The most difficult aspect of language to process is *ambiguity*—when one word, phrase, or sentence can have multiple meanings. This happens frequently with words—as evidenced by spelling errors (e.g., *they're*, *their*, and *there*) and homographs (e.g., *bass*—do you *hear* it or *eat* it?)

However, these ambiguous meanings become even more pronounced as words are strung together into phrases and sentences. Humans usually have a good filter for only hearing the one that makes sense in the given context. Computer have some difficulty with this, though AI research is making headway. Consider the follow example, in which a computer program might have difficulty matching a short sentence with a visual scene:



I saw her duck.

A computer might return 3 possibilities for this sentence; do any of them match your first impression as you read the sentence?



Probably, you pictured something like B, maybe like A, and almost assuredly not C (unless you are a sadistic duck-hater). However, a computer program would need to be trained to prefer one of these over the others in any given context.

How do AI systems utilize probability and statistics to behave ‘intelligently’?

Humans are able to resolve ambiguities and missing information almost without conscious thought through intuition. Computers, as we know, however, must have explicitly defined

instructions for processing data. Even if the computer were to choose randomly among possible choices, this *random selection* routine would have to be specified in the program's code.

Computers can mimic this intuition by statistically analyzing which of the possible choices are most likely given the context. So, Autocorrect might notice that you are typing *Should we incl...* and autofills the rest *ude*. Other words may be relevant (e.g., *incline*), but *include* is more likely to occur in that particular phrase. If instead the text read *Head up the incl...*, Autocorrect may have made a different choice.

<p><i>Of course, the calculated most likely choice (based on context and prior history) isn't always the best choice:</i></p> 	<p><i>Eh, humans make mistakes, too. Ambiguity is a difficult problem.</i></p> 
--	--

Are we there yet?

So, can we talk to our computers now? Are they poised to be great conversationalists? What about Siri?

Common misconception We are close to the Star Trek computer-type linguistic interface.



<https://www.youtube.com/embed/MA1hD3XRlh0>

Not *really*. On a surface level, what Siri (integrated with GPS positioning) does is much like the Star Trek computer. However, Siri's functionality is limited in scope—to the phone and the functions it provides. It cannot play chess with you, or understand and process compound sentences, like "Message my mom that I got home safely **AND** set my alarm for 3 AM."



Artificial Intelligence: Turing Test Project: Rubric Check



Instructions

This activity provides you and a partner group time to provide one another *critical feedback* about the progress of your projects.

1. Pair up with another group.
2. Rate the components of your partner group's written Turing Test protocol, Scratch program, and the documentation of their approach according to the [Turing Test Project rubric](#). You may write this out or print the rubric and circle components on it.
3. Review your partner group's work and provide them with documentation that describes what you *Like*, what you *Wonder*, and what their *Next Steps* should be.
4. When both you and your partner group have completed the previous steps, share your feedback with one another.

Listen to what your partner group says! You do not need to heed all of their advice, but consider it wisely. The whole idea is to have a critical third party provide ideas to make your project better before you submit the final version for a grade.