Thriving in Our Digital World: AP

Developers (http://uteachcs.org)

Thriving in Our Digital World: AP has been developed by The UTeach Institute through a grant from the National Science Foundation (award #1543014).

Course Overview

Thriving in Our Digital World: AP has been designed as a year-long high school course that fully addresses the seven “Big Ideas” of computer science and six “Computational Thinking Practices”, as specified by the College Board’s AP Computer Science Principles curriculum framework.

The lessons and materials used throughout this course incorporate Project-Based Learning (PBL), a pedagogical approach that actively engages students in the educational process, improves retention, and develops problem solving, critical thinking, and group communication skills. Through this collaborative, learner-centric approach, students are encouraged to explore the advantages and societal impact of computational technology while developing their own programming and computational thinking skills.

Sample Course Materials

This document contains a brief sampling of the types of materials available throughout this course. These materials are representative of a typical unit and are intended to provide an example of the general structure and content for each unit, including the following elements:

- Unit Overview for Teachers
  - Support documentation for each unit includes a schedule outlining the overall sequencing of unit topics and lessons, a general description that highlights the unit’s main ideas and projects, and a detailed listing of learning objectives and their corresponding standards from the AP Computer Science Principles Curriculum Framework.

- Student Content
  - Selected examples of student materials are provided here that demonstrate the type of exercises and instructional content included in each unit. These full set of materials will be available for online and/or printed use, as the teacher sees fit.

While not included here, each unit will also include detailed rubrics for projects and formal assessments that are modeled after the AP Computer Science Principles written exam and performance task requirements.
Unit 5: Digital Media Processing

Unit Schedule

<table>
<thead>
<tr>
<th>Day</th>
<th>Topic</th>
<th>Lessons</th>
<th>Learning Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>5A</td>
<td>Unit Project</td>
<td>Image Filter Project</td>
<td>1.2.2, 1.2.3, 1.2.4, 1.3.1, 5.1.2, 5.1.3</td>
</tr>
<tr>
<td>5B</td>
<td>Procedural Programming</td>
<td>Introduction to Processing</td>
<td>1.2.2, 1.2.3, 1.3.1, 2.2.2, 4.1.1, 4.1.2, 5.1.2, 5.3.1, 5.4.1</td>
</tr>
<tr>
<td>5C</td>
<td>Drawing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5D</td>
<td>Mouse Interaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5E</td>
<td>Keyboard Interaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5F</td>
<td>Big Picture</td>
<td>Ethics of Digital Manipulation</td>
<td>7.3.1</td>
</tr>
<tr>
<td>5G</td>
<td>Image Manipulation</td>
<td>RGB Color</td>
<td>1.2.2, 1.2.3, 1.2.4, 1.3.1, 2.2.2, 4.1.1, 4.1.2, 5.1.2, 5.3.1, 5.4.1</td>
</tr>
<tr>
<td>5H</td>
<td>Raster Images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5I</td>
<td>Raster Image Manipulation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5J</td>
<td>Encoding Schemes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5K</td>
<td>Manipulating Digital Images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5L</td>
<td>Unit Project</td>
<td>Image Filter Project (cont.)</td>
<td>1.2.2, 1.2.3, 1.2.4, 1.3.1, 5.1.2, 5.1.3</td>
</tr>
<tr>
<td>5M</td>
<td>Audio Manipulation</td>
<td>Digital Audio</td>
<td>1.3.1, 3.3.1, 4.1.1, 4.1.2, 5.1.2, 7.3.1</td>
</tr>
<tr>
<td>5N</td>
<td>Audio Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5O</td>
<td>Audio Compression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5P</td>
<td>Big Picture</td>
<td>Intellectual Property Rights</td>
<td>7.3.1</td>
</tr>
<tr>
<td>5Q</td>
<td>Unit Project</td>
<td>Image Filter Project (cont.)</td>
<td>1.2.2, 1.2.3, 1.2.4, 1.3.1, 5.1.2, 5.1.3</td>
</tr>
<tr>
<td>5S</td>
<td>Unit Exam</td>
<td>Unit 5 Exam</td>
<td>1.2.2, 1.2.3, 1.2.4, 1.3.1, 2.2.2, 3.3.1, 4.1.1, 4.1.2, 5.1.1, 5.1.2, 5.1.3, 5.3.1, 5.4.1, 7.3.1</td>
</tr>
<tr>
<td>5T</td>
<td>Unit Project</td>
<td>Project Presentations</td>
<td>1.2.2, 1.2.3, 1.2.4, 1.3.1, 5.1.2, 5.1.3</td>
</tr>
</tbody>
</table>

Unit Description

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 an 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 Coverage**

- BI: 1, 2, 3, 4, 5, 7
- LO: 1.2.2, 1.2.3, 1.2.4, 1.3.1, 2.2.2, 3.3.1, 4.1.1, 4.1.2, 5.1.1, 5.1.2, 5.1.3, 5.3.1, 5.4.1, 7.3.1
- CTP: P2, P3, P4, P5, P6

**Unit Topics**

- **Procedural Programming** [1.2.2, 1.2.3, 1.3.1, 2.2.2, 4.1.1, 4.1.2, 5.1.2, 5.3.1, 5.4.1] (P2, P3, P4, P5)
  - Students will explore the capabilities of a text-based programming language (Processing).
  - Students will compare and contrast the programming capabilities of a visual programming language (Scratch) with those of a text-based programming language (Processing).
  - Students will write programs that make use of parameterized methods to invoke specific behaviors.
  - Students will understand the importance of using proper punctuation and syntax when coding in a text-based programming language.
  - Students will use event handlers to animate on-screen effects and respond to mouse and keyboard input.
  - Students will write code using common programming constructs like conditional if() for selection and while() loops for iteration.
- **Image Manipulation** [1.2.2, 1.2.3, 1.2.4, 1.3.1, 2.2.2, 4.1.1, 4.1.2, 5.1.2, 5.3.1, 5.4.1] (P2, P3, P4, P5, P6)
  - Students will examine the structure of raster images as compositions of individual pixels.
  - Students will explore various methods of representing color, including RGB, CMYK, and HSV.
  - Students will explore the various colors that can be produced by the combination of different ratios of red, green, and blue light.
  - Students will modify the color channels of pixels in an image to produce a variety of effects.
  - Students will design algorithms for modifying the pixels in an image in prescribed ways to create custom image filters.
  - Students will explore the difference between lossy and lossless encoding schemes of several common image file formats.
- **Audio Manipulation** [1.3.1, 3.3.1, 4.1.1, 4.1.2, 5.1.2, 7.3.1] (P2, P4, P5)
  - Students will analyze the differences between analog and digital sound.
Thriving in Our Digital World: AP

Unit 5: Digital Media Processing

APPLICATION

- Students will explore the roles that sampling rate and bit depth play in determining the quality of digitized sound.
- Students will explore methods of programatically generating digital audio.
- Students will explore methods of programatically altering and modifying digital audio by adjusting volume, pitch, and sampling rate.
- Students will explore the methods and effects of compression algorithms in reducing the amount of data needed to represent an audio sample.
  - Big Picture [7.3.1] (P4)
    - Students will explore the positive and negative consequences of digitally altering images.
    - Students will discuss the ethics of digitally manipulating images, especially in the context of journalism.
    - Students will discuss the issues related to intellectual property.
    - Students will explore the limitations and rights associated with a number of common licenses, including Creative Commons.

Unit Project [CR1b] [CR1f] [CR2a] [CR2e]
  - Image Filter Project
    - Students will utilize pair programming to design and implement a program for filtering digital images. [1.2.2, 1.2.3, 1.2.4, 5.1.2, 5.1.3] (P2, P6)
    - Using the Processing programming language, students will develop code to systematically transform an image by mathematically manipulating its bits, pixel by pixel. [1.2.2, 1.2.3, 1.3.1, 5.1.2] (P2)
    - Students will write documentation detailing the use of their program and its features using appropriate terminology. [5.1.2] (P2)
    - Students will explain their design and implementation choices by demonstrating and sharing their finished programs with their peers. [1.2.4, 5.1.3] (P6)

Unit Readings
  - Blown to Bits (Abelson, Ledeen, Lewis). Chapter 6: Balance Toppled – Who Owns the Bits?

Unit Assessments
  - Minor exercises addressing specific unit topics and objectives
  - Formally assessed, multiple-choice test addressing unit objectives (single- and multiple-select questions)
  - Rubric-assessed, individual and/or collaborative unit project demonstrating mastery of unit objectives

Unit Objectives
  - Big Idea 1: Creativity
Unit 5: Digital Media Processing

- Create a computational artifact using computing tools and innovative, non-traditional techniques to solve a problem. [1.2.2A, 1.2.2B] (P2)
- Create a computational artifact by combining and modifying existing artifacts to show personal expression of ideas. [1.2.3A, 1.2.3C] (P2)
- Use computational tools to create or modify a computational artifact with enhanced detail and precision. [1.2.3B] (P2)
- Use appropriate interpersonal skills, communication, and group decision-making to create an enhanced, collaborative computational artifact. [1.2.4C, 1.2.4D] (P6)
- Create a collaborative computational artifact that reflects the diverse talents and personal ideas of all group members. [1.2.4E, 1.2.4F] (P6)
- Identify how the creation of digital effects, images, audio, video, and animations has transformed industries. [1.3.1A] (P2)
- Use computing tools to create digital audio and music by synthesizing, sampling, recording, layering, and/or looping sounds. [1.3.1B] (P2)
- Use computing tools to create digital images by generating pixel patterns, manipulating digital images, or combining images. [1.3.1C] (P2)
- Use appropriate software and image editing tools to create digital effects and animations. [1.3.1D] (P2)
- Use computing tools to enable creative exploration of digital images and/or sounds. [1.3.1E] (P2)

- Big Idea 2: Abstraction
  - Develop software using multiple levels of abstraction, including constants, expressions, statements, procedures, and libraries, to more effectively apply available resources and tools to solve problems. [2.2.2A, 2.2.2B] (P3)

- Big Idea 3: Data and Information
  - Analyze the different trade-offs between lossy and lossless compression techniques for storing and transmitting data. [3.3.1C, 3.3.1D, 3.3.1E] (P4)

- Big Idea 4: Algorithms
  - Develop an algorithm using sequencing, selection, and iteration. [4.1.1A, 4.1.1B, 4.1.1C, 4.1.1D] (P2)
  - Develop an algorithm that uses or combines existing, standard algorithms to ensure correctness of the resulting solution. [4.1.1E, 4.1.1F, 4.1.1G] (P2)
  - Explain how natural language, pseudocode, and visual and textual programming languages can all be used to express an algorithm. [4.1.2A, 4.1.2H] (P5)
  - Explain how different languages are better suited than others for expressing algorithms in specific problem domains. [4.1.2D, 4.1.2E] (P5)

- Big Idea 5: Programming
  - Identify ways that advances in computing have enabled creativity in other fields. [5.1.1F] (P2)
  - Develop a large, correct program using an iterative process that incrementally combines tested program components. [5.1.2A, 5.1.2B, 5.1.2C] (P2)
<table>
<thead>
<tr>
<th>Unit 5: Digital Media Processing</th>
<th>APPLICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>○ Provide documentation about program components, such as blocks and procedures, to maintain correct programs when working individually or collaboratively with other programmers. [5.1.2D, 5.1.2E, 5.1.2F] (P2)</td>
<td></td>
</tr>
<tr>
<td>○ Develop a program using appropriate knowledge of and skill in the development process, including designing, implementing, testing, debugging, and maintaining programs. [5.1.2I, 5.1.2J] (P2)</td>
<td></td>
</tr>
<tr>
<td>○ Use collaboration to facilitate multiple perspectives in developing ideas for solving problems by programming. [5.1.3B] (P6)</td>
<td></td>
</tr>
<tr>
<td>○ Use abstraction to create named, parameterized, and reusable blocks of programming in order to reduce the complexity of writing and maintaining a program. [5.3.1A, 5.3.1B, 5.3.1C, 5.3.1D] (P3)</td>
<td></td>
</tr>
<tr>
<td>○ Use parameterization to generalize specific solutions and allow a single function to be used in place of duplicated code. [5.3.1E, 5.3.1F, 5.3.1G] (P3)</td>
<td></td>
</tr>
<tr>
<td>○ Use well-documented application program interfaces (APIs) and libraries to connect software components and to simplify complex programming. [5.3.1M, 5.3.1N, 5.3.1O] (P3)</td>
<td></td>
</tr>
<tr>
<td>○ Use good programming style, such as meaningful names for variables and procedures, shorter code blocks, and non-duplicated code, in order to improve the determination of program correctness. [5.4.1A, 5.4.1B, 5.4.1C, 5.4.1D] (P4)</td>
<td></td>
</tr>
<tr>
<td>○ Debug a program by locating and correcting errors. [5.4.1E] (P4)</td>
<td></td>
</tr>
<tr>
<td>○ Describe the functionality of a program at a high level in terms of what it does and how a user interacts with it and provide examples of intended behavior on specific inputs in order to find program errors. [5.4.1F, 5.4.1G, 5.4.1L, 5.4.1M, 5.4.1N] (P4)</td>
<td></td>
</tr>
<tr>
<td>○ Use visual displays (or different modalities) of program state to help in finding errors. [5.4.1H] (P4)</td>
<td></td>
</tr>
<tr>
<td>○ Justify a program’s correctness by explaining how it meets its specifications. [5.4.1I, 5.4.1J] (P4)</td>
<td></td>
</tr>
<tr>
<td>○ Demonstrate the correctness of a program by demonstrating correctness of its components, including code blocks and procedures. [5.4.1K] (P4)</td>
<td></td>
</tr>
</tbody>
</table>

**Big Idea 7: Global Impact**

○ Analyze the legal and ethical concerns raised by innovations, access, and censorship of digital content. [7.3.1A, 7.3.1B, 7.3.1C, 7.3.1D, 7.3.1E] (P4)

○ Analyze the intellectual property and copyright concerns with digital information, audio, video, and textual content. [7.3.1N, 7.3.1O, 7.3.1P] (P4)
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 programatically manipulating bits.
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 Objectives**

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.
<table>
<thead>
<tr>
<th>Criteria</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manipulates digital images at the PIXEL LEVEL in multiple ways.</td>
<td>5 pts</td>
</tr>
<tr>
<td>Manipulates digital images by RE-ORDERING PIXELS.</td>
<td>5 pts</td>
</tr>
<tr>
<td>Project includes DOCUMENTATION detailing its use. The accompanying</td>
<td>5 pts</td>
</tr>
<tr>
<td>program and documentation align. Key terminology is used as necessary.</td>
<td></td>
</tr>
<tr>
<td>The image filter results are AESTHETICALLY PLEASING.</td>
<td>5 pts</td>
</tr>
<tr>
<td>Uses PROCESSING CONSTRUCTS appropriately and effectively: LOOPS</td>
<td>2 pts</td>
</tr>
<tr>
<td>Uses PROCESSING CONSTRUCTS appropriately and effectively: SYNTAX</td>
<td>2 pts</td>
</tr>
<tr>
<td>Uses PROCESSING CONSTRUCTS appropriately and effectively: CONDITIONALS</td>
<td>2 pts</td>
</tr>
<tr>
<td>Uses PROCESSING CONSTRUCTS appropriately and effectively: COLOR</td>
<td>1 pts</td>
</tr>
<tr>
<td>FUNCTIONS</td>
<td></td>
</tr>
<tr>
<td>Uses PROCESSING CONSTRUCTS appropriately and effectively: CORRECTNESS</td>
<td>3 pts</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>30 pts</strong></td>
</tr>
</tbody>
</table>
5B: Scratch vs. Processing

Comparison: Scratch vs. Processing
There are many similarities among the two languages; in fact, these similarities occur across nearly all of the programming languages in use today!

<table>
<thead>
<tr>
<th>Scratch</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Scratch Logo" /></td>
<td><img src="image2.png" alt="Processing Logo" /></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Scratch</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="Scratch Start/Stop" /></td>
<td><img src="image4.png" alt="Processing Play/Pause" /></td>
</tr>
</tbody>
</table>

- Each has a way of outputting text.

<table>
<thead>
<tr>
<th>Scratch</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5.png" alt="Scratch Say" /></td>
<td><code>println(&quot;Hello!&quot;);</code></td>
</tr>
</tbody>
</table>

- 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 vs. Processing

<table>
<thead>
<tr>
<th>Scratch</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="example" alt="Scratch Code" /></td>
<td><img src="example" alt="Processing Code" /></td>
</tr>
</tbody>
</table>

- Each offers a way to assign a value to a variable.

<table>
<thead>
<tr>
<th>Scratch</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="example" alt="Set Index to 0" /></td>
<td><img src="example" alt="Set Index to 0" /></td>
</tr>
</tbody>
</table>

- Each offers a way to change the value of a variable.

<table>
<thead>
<tr>
<th>Scratch</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="example" alt="Change Index by 1" /></td>
<td><img src="example" alt="Change Index by 1" /></td>
</tr>
</tbody>
</table>

- 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 only means that the Boolean condition that determines whether to repeat another iteration of the loop is backwards between the two languages.

  - **Scratch:** The loop continues *until* a condition becomes **true**.
    - ![Repeat Until Mouse X < Mouse Y](example) (i.e., x is less than y)
  - **Processing:** The loop continues *while* a condition remains **true**.
    - ![While X >= Y](example) (i.e., x is *NOT* less than y)

<table>
<thead>
<tr>
<th>Scratch</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="example" alt="Repeat Until Mouse X &lt; Mouse Y" /></td>
<td><img src="example" alt="While X &gt;= Y" /></td>
</tr>
</tbody>
</table>

---

© 2016 The University of Texas at Austin
5C: 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-line programs that do 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. Draw an ellipse with a width of 20 and a height of 30. It should be centered at the x-y coordinates (100, 40).
2. Create a color variable 'c' set to burnt orange. The color values for burnt orange are (RED = 204), (GREEN = 85), and (BLUE = 0).
3. Draw a quadrilateral that matches the following figure:
5D: 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 and 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, 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:

```java
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:

```java
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(255);` 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:**


© 2016 The University of Texas at Austin
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.
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:
**5F: Ethics of Digital Manipulation**

**Debate**

In this activity, your class will hold a debate in which one side argues that "digital manipulation is never or rarely ethical" while the other side argues that "digital manipulation is always or usually ethical."

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. Conduct an online search for more information about the topic (the ethics of digital manipulation). You will have 5-10 minutes to prepare.

4. Debate the ethics of digital manipulation.

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?
5G: 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.

![Color Palette](image)

**Exercise**

1) Enter the following values as RGB values into the [Color Calculator](#). What color is generated?

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110111</td>
<td>11001101</td>
<td>00011110</td>
</tr>
</tbody>
</table>

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](#).*

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110111</td>
<td>11001101</td>
<td>00011110</td>
</tr>
</tbody>
</table>
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.*

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>11001101</td>
<td>00000000</td>
</tr>
</tbody>
</table>

A. The color gets brighter.
B. The color gets darker.
C. The color gets 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.*

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>11001101</td>
<td>11001101</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001101</td>
<td>11001101</td>
<td>11001101</td>
</tr>
</tbody>
</table>

Answer:
**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:

![Image](image.png)

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:

![Arrow Diagram](https://via.placeholder.com/150)

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.

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:

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110111</td>
<td>11001101</td>
<td>00011110</td>
</tr>
</tbody>
</table>

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:

```javascript
rect(300, 200, 150, 50);
```
An example encoding may produce a rectangle if it were for a vector image encoding:

<table>
<thead>
<tr>
<th>Width</th>
<th>Length</th>
<th>Thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110111</td>
<td>11001101</td>
<td>00011110</td>
</tr>
</tbody>
</table>

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 or TextEdit.
  - 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.
**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](https://www.tate.org.uk)). 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](https://www.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 it, and saves the result. **Note:** The `filter()` function works over the entire display window. As such, you will need to create 3 images separately.
   - Include multiple filters on at least 2 of your variants. For example, variant 1 above uses a loop to blur the image 100 times. 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 (TwoByTwo) to create the grid. Ensure that the labels describe the effects accurately. Do not simply leave them labeled "Variant x".

Click to Download: TwoByTwo

Submission
Submit the Processing sketch you develop.

Reference Starting Points

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter()</td>
</tr>
<tr>
<td>text()</td>
</tr>
<tr>
<td>save()</td>
</tr>
</tbody>
</table>

© 2016 The University of Texas at Austin
5M: 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.

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 Third-party Libraries

**NOTE:** Before running the Theremin.pde sketchbook (or several of the later projects that also involve audio generation), you will need to install Minim, a third-party software library that provides the Processing application additional code 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 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.

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, f, g, a, and b). 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 (261.625565300598634 Hz) is 2 x 261.625565300598634, or 523.251130601197269 Hz.

3. Try playing a song:

![Music.png](https://example.com/Music.png)

**Submission**

Submit the [Processiano.pde](#) file of your Processiano sketchbook.
Extensions

The synthesized audio signal in Processiano is generated as a sine wave. Alter the program to use a different geometry. The available choices are: PulseWave, SawWave, SineWave, SquareWave, TriangleWave.

Processiano uses only natural (♮) notes — i.e., no sharps (#) or flats (♭). 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♯ or F♭).
**5O: 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:

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>$5 \times 5 \times 5 \times 5 \times 5 \times 5$</td>
</tr>
</tbody>
</table>

Exponential notation was created to alleviate this degree of *redundancy* and improve legibility:

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>$5 \times 5 \times 5 \times 5 \times 5 \times 5$</td>
</tr>
<tr>
<td>Compressed</td>
<td>$5^7$</td>
</tr>
</tbody>
</table>

Multiplication itself is really nothing more than repeated addition:

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>$5 + 5 + 5 + 5 + 5 + 5 + 5$</td>
</tr>
<tr>
<td>Compressed</td>
<td>$5 \times 7$</td>
</tr>
</tbody>
</table>

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:
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? 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.

**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.

**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](image)

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.

In the next activity (X Marks the Spot!) you will design and implement your own compression algorithm. Will it be lossless or lossy? That all depends on the goal.

© 2016 The University of Texas at Austin
5P: The Legality of Remixes

Remixes and Mashups

"There are no original ideas. There are only original people." – Barbara Grizzuti

Just as digitization facilitated a boom in post-processing, it has also facilitated a boom in remixes and mashups while at the same time sparking new ethical concerns and debates about ownership, creativity, and originality. Watch this 12-minute and think about how this affects you.

After you watch the video, prepare the following responses for a class-wide discussion:

- Summarize one idea/scenario the video discusses.
- Reflect upon how that scenario affects your/our lives.
- Pose a question to your peers.