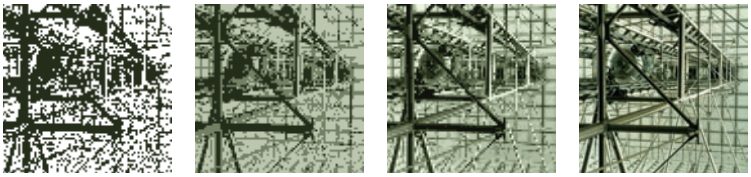
Image 1: Display, Tint

This unit introduces loading and displaying images.

Syntax introduced:

```
PImage, loadImage(), image()
tint(), noTint()
```

Digital photographs are fundamentally different from analog photographs captured on film. Like computer screens, digital photos are rectangular grids of color. The dimensions of digital images are measured in units of pixels. If an image is 320 pixels wide and 240 pixels high, it has 76,800 total pixels. If an image is 1280 pixels wide and 1024 pixels wide, the total number of pixels is an impressive 1,310,720 (1.3 megapixels). Every digital image has a color depth. The color depth refers to the number of bits (p. 669) used to store each pixel. If the color depth of an image is 1, each pixel can be one of two values, for example, black or white. If the color depth is 4, each pixel can be one of 16 values. If the color depth of an image is 8, each pixel can be one of 256 values. Looking at the same image displayed with different color depths reveals how this affects the image's appearance:



1-bit (1 color) 2-bit (4 colors) 4-bit (16 colors) 8-bit (256 colors)

When the Apple Macintosh computer was introduced in 1984, it had a black-and-white screen. Since then, the reproduction of color on screen has rapidly improved. Many contemporary screens have a color depth of 24, which means each pixel can be one of 16,777,216 available colors. This number is typically referred to as "millions of colors."

Digital images are comprised of numbers representing colors. The file format of an image determines how the numbers are ordered in the file. Some file formats store the color data in mathematically complex arrangements to compress the data and reduce the size of the resulting file. A program that loads an image file must know the file format of the image so it can translate the file's data into the expected image. Different types of digital image formats serve specific needs. Processing can load GIF, JPEG, and PNG images, along with some other formats as described in the reference. If you don't already have your images in one of these formats, you can convert other types of digital images to these formats with programs such as GIMP or Adobe Photoshop. Refer to the documentation for these programs if you're unsure how to convert images.

How do you know which image format to use? They all have obscure names that don't help in making this decision, but each format's advantages becomes clear through comparison:

Format	Extension	Color depth	Transparency
GIF	.gif	1-bit to 8-bit	1-bit
JPEG	.jpg	24-bit	None
PNG	.png	1-bit to 24-bit	8-bit

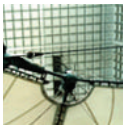
If you are displaying your work on the Internet, image compression becomes an important issue. GIF images are useful for simple graphics with a limited number of colors and transparency. PNG images have similar characteristics but support the full range of colors and transparency. The JPEG format works well for photos, and JPEG files will be smaller than most images saved as PNG. This is because JPEG is a "lossy" format, which means it sacrifices some image quality to reduce file size.

Display

Processing can load images, display them on the screen, and change their size, position, opacity, and tint. There's a data type for images called `PImage`. The same way that integers are stored in variables of the `int` data type and values of `true` and `false` are stored in the `boolean` data type, images are stored in variables of the `PImage` data type. Before displaying an image, it's necessary to first load it with the `loadImage()` function. Be sure to include the file format extension as a part of the name and to put the entire name in quotes (e.g., "*pup.gif*", "*kat.jpg*", "*ignatz.png*"). For the image to load, it must be in the data folder of the current program. Add the image by selecting the "Add File" option in the Sketch menu of the Processing environment. Navigate to the image's location on your computer, select the image's icon or name, and click "Open" to add it to the sketch's data folder. As a shortcut, you can also drag and drop an image to the Processing window. To make sure the image was added, select "Show Sketch Folder" from the Sketch menu. The image will be inside the *data* folder. With the image file in the right place, you can load and then display it with the `image()` function:

```
image(name, x, y)
image(name, x, y, width, height)
```

The parameters for `image()` determine the image to draw and its position and size. The *`name`* parameter must be a `PImage` variable. The *`x`* and *`y`* parameters set the position of the upper-left corner of the image. The image will display at its actual size (in units of pixels), but you can change the size by adding the *`width`* and *`height`* parameters. Be careful to use the correct capitalization when loading images. If the image is *arch.jpg*, trying to load *Arch.jpg* or *arch.JPG* will create an error. Also, avoid the use of spaces in image names, which can cause problems.



10-01

```
PImage img;
// Image must be in the sketch's "data" folder
img = loadImage("arch.jpg");
image(img, 0, 0);
```

10-02

```
PImage img;
// Image must be in the sketch's "data" folder
img = loadImage("arch.jpg");
image(img, 20, 20, 60, 60);
```

Image color, Transparency

Images are colored with the `tint()` function. This function is used the same way as `fill()` and `stroke()`, but it affects only images:

```
tint(gray)
tint(gray, alpha)
tint(value1, value2, value3)
tint(value1, value2, value3, alpha)
tint(color)
```

All images drawn after running `tint()` will be tinted by the color specified in the parameters. This has no permanent effect on the images, and running the `noTint()` function disables the coloration for all images drawn after it is run.

10-03

```
PImage img;
img = loadImage("arch.jpg");
tint(102); // Tint gray
image(img, 0, 0);
noTint(); // Disable tint
image(img, 50, 0);
```

10-04

```
PImage img;
img = loadImage("arch.jpg");
tint(0, 153, 204); // Tint blue
image(img, 0, 0);
noTint(); // Disable tint
image(img, 50, 0);
```

10-05

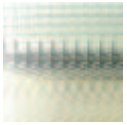
```
color yellow = color(220, 214, 41);
color green = color(110, 164, 32);
color tan = color(180, 177, 132);
PImage img;
img = loadImage("arch.jpg");
tint(yellow);
image(img, 0, 0);
tint(green);
image(img, 33, 0);
tint(tan);
image(img, 66, 0);
```

The parameters for `tint()` follow the color space determined by the `colorMode()` function (remember, the default color mode is RGB, with all values ranging from 0 to 255). If the color mode is changed to HSB or a different range, the tint values should be specified relative to that mode.

To make an image transparent without changing its color, set the tint to white. The value will depend on the current color mode, but the default white value is 255.

10-06

```
PImage img;
img = loadImage("arch.jpg");
background(255);
tint(255, 102);  // Alpha to 102 without changing the tint
image(img, 0, 0, 100, 100);
tint(255, 204, 0, 153);  // Tint to yellow, alpha to 153
image(img, 20, 20, 100, 100);
```



10-07

```
PImage img;
img = loadImage("arch.jpg");
background(255);
tint(255, 51);
// Draw the image 10 times, moving each to the right
for (int i = 0; i < 10; i++) {
  image(img, i*10, 0);
}
```

GIF and PNG images retain their transparency when loaded and displayed in Processing. This allows anything drawn before the image to be visible through the transparent sections of the image. GIF images have only 1-bit transparency, meaning each pixel can only be completely opaque or completely transparent. The PNG format supports 8-bit transparency, meaning there are 256 levels of opacity.

10-08

```
// Loads a GIF image with 1-bit transparency
PImage img;
img = loadImage("archTrans.gif");
background(255);
image(img, 0, 0);
image(img, -20, 0);
```

10-09

```
// Loads a PNG image with 8-bit transparency
PImage img;
img = loadImage("arch.png");
background(255);
image(img, 0, 0);
image(img, -20, 0);
```

Exercises

1. *Draw two images in the display window.*
2. *Draw three images in the display window, each with a different tint.*
3. *Load a GIF or PNG image with transparency and create a collage by layering the image.*