Module 03

Input / Output
We can write more interesting programs when we can exchange information with the outside world.
Problem with the outside world: there’s a lot of it.
Use the Sketch Folder as a gateway to the outside world.
Use Sketch → Add File... to make a file available to your sketch, or drop the file into the sketch folder directly.

Any files created by the sketch will be left in the sketch folder.
1. Reading and writing images
PImage loadImage(String filename) {
    ...
}

A built-in function that takes the name of a file as a String parameter, finds that file in your sketch folder, and tries to import it as an image. Returns an object of type PImage.
High-level PImage operations

PImage img;

void setup()
{
  size( 800, 800 );
  img = loadImage( "some_image.jpg" );
}

void draw()
{
  background( 255 );

  imageMode( CORNER );
  noTint();
  image( img, 0, 0 );
  image( img, width - img.width, height - img.height );

  tint( 255, 120, 120 );
  imageMode( CENTER );
  image( img, width/2, height/2, 250, 250 );
}
High-level PImage operations

PImage img;

void setup()
{
    size(800, 800);
    img = loadImage("some_image.jpg");
}

void draw()
{
    background(255);

    imageMode(CORNER);
    noTint();
    image(img, 0, 0);
    image(img, width - img.width, height - img.height);

    tint(255, 120, 120);
    imageMode(CENTER);
    image(img, width/2, height/2, 250, 250);
}
High-level PImage operations

PImage img;

void setup()
{
  size( 800, 800 );
  img = loadImage( "some_image.jpg" );
}

void draw()
{
  background( 255 );

  imageMode( CORNER );
  noTint();
  image( img, 0, 0 );
  image( img, width - img.width, height - img.height );

  tint( 255, 120, 120 );
  imageMode( CENTER );
  image( img, width/2, height/2, 250, 250 );
}
High-level PImage operations

PImage img;

void setup()
{
  size( 800, 800 );
  img = loadImage( "some_image.jpg" );
}

void draw()
{
  background( 255 );

  imageMode( CORNER );
  noTint();
  image( img, 0, 0 );
  image( img, width - img.width, height - img.height );

  tint( 255, 120, 120 );
  imageMode( CENTER );
  image( img, width/2, height/2, 250, 250 );
}
High-level PImage operations

PImage img;

void setup()
{
    size( 800, 800 );
    img = loadImage( "some_image.jpg" );
}

void draw()
{
    background( 255 );

    imageMode( CORNER );
    noTint();
    image( img, 0, 0 );
    image( img, width - img.width, height - img.height );

    tint( 255, 120, 120 );
    imageMode( CENTER );
    image( img, width/2, height/2, 250, 250 );
}
High-level PImage operations

PImage img;

void setup()
{
    size( 800, 800 );
    img = loadImage( "some_image.jpg" );
}

void draw()
{
    background( 255 );

    imageMode( CORNER );
    noTint();
    image( img, 0, 0 );
    image( img, width - img.width, height - img.height );

    tint( 255, 120, 120 );
    imageMode( CENTER );
    image( img, width/2, height/2, 250, 250 );
}

Apply a colour wash to all images.
High-level PImage operations

PImage img;

void setup()
{
    size( 800, 800 );
    img = loadImage( "some_image.jpg" );
}

void draw()
{
    background( 255 );
    noTint();
    image( img, 0, 0 );
    image( img, width - img.width, height - img.height );

    tint( 255, 120, 120 );
    imageMode( CENTER );
    image( img, width/2, height/2, 250, 250 );
}

Change the anchor point of the image.
Image no-nos

PImage img = loadImage( "some_image.jpg" );

void setup()
{
    size( 800, 800 );
    ...
}

Don’t try to load the image in the global variable declaration. This will usually fail.
Image no-nos

void draw()
{
    PImage img = loadImage( "some_image.jpg" );
    image( img, 0, 0 );
}

Don’t load images in draw(). This won’t break the program, but it will work much harder than necessary. Load the image once in setup().
PImage img;

void setup()
{
    img = loadImage("some_image.jpg");
}

void draw()
{
    image(img, 0, 0);
    image(img, width - img.width, height - img.height);
    image(img, width/2, height/2, 250, 250);
}
Standard image idiom

Global variable to hold image.

```java
void setup()
{
    PImage img;
    img = loadImage("some_image.jpg");
}

void draw()
{
    background(0);
    image( img, 0, 0 );
    image( img, width - img.width, height - img.height );
    image( img, width/2, height/2, 250, 250 );
}
```
**Standard image idiom**

```java
PImage img;  // Global variable to hold image.

void setup()
{

  img = loadImage("some_image.jpg");  // Load image in setup().
}

void draw()
{

  image( img, 0, 0 );
  image( img, width - img.width, height - img.height );

  image( img, width/2, height/2, 250, 250 );
}
```
Standard image idiom

Global variable to hold image.

```cpp
PImage img;

void setup()
{
  img = loadImage("some_image.jpg");
}

void draw()
{
  image(img, 0, 0);
  image(img, width - img.width, height - img.height);
  image(img, width/2, height/2, 250, 250);
}
```

Load image in `setup()`.

Use image in `draw()`.
You can also copy a region out of a source image, and scale it to any rectangle in the sketch window.

```cpp
copy( img, sx, sy, sw, sh, dx, dy, dw, dh );
```
You can also copy a *region* out of a source image, and scale it to any rectangle in the sketch window.

```c
    copy(img, sx, sy, sw, sh, dx, dy, dw, dh);
```

The source image to copy pixels from
You can also copy a *region* out of a source image, and scale it to any rectangle in the sketch window.

```plaintext
copy( img, sx, sy, sw, sh, dx, dy, dw, dh );
```

A rectangle of pixels in the source image. Just like the arguments in a call to `rect()`
You can also copy a region out of a source image, and scale it to any rectangle in the sketch window.

```
copy( img, sx, sy, sw, sh, dx, dy, dw, dh );
```

A rectangle of pixels in the sketch window. Again, just like a call to `rect()`
copy( img, sx, sy, sw, sh, dx, dy, dw, dh );
Writing images

Several ways to do this. Easiest is to take a screenshot.

```java
void save(String filename) { ... }
```

Save the contents of the sketch window to an image with the given file name.

```java
void saveFrame() { ... }
void saveFrame(String name_template) { ... }
```

Same as above, but include a counter in the saved file name. Useful for animations.
void keyPressed()
{
    if (key == 's') {
        save( "screen.png" );
    }
}

2. Reading and writing illustrations
Raster image: represented using a grid of pixels.

Vector illustration: represented using geometric paths.
Raster image: represented using a grid of pixels.
JPG, PNG, GIF, BMP, TIFF, ...

Vector illustration: represented using geometric paths.
PDF, EPS, AI, SVG, ...
Images

```
loadImage()
PImage
image()
```
Images

loadImage()
PImage
image()

Illustrations

loadShape()
PShape
shape()
PShape tiger;

void setup()
{
  size( 500, 500 );
  tiger = loadShape( "tiger.svg" );
}

void draw()
{
  shape( tiger, 0, 0 );
}
The **PShape** class has a `disableStyle()` method that forces the SVG to be drawn with the current fill and stroke settings.

```java
void draw()
{
    background( 255 );
    if( keyPressed ) {
        tiger.disableStyle();
        fill( 255, 0, 0 );
        noStroke();
    } else {
        tiger.enableStyle();
    }
    shape( tiger, 0, 0 );
}
```
Writing illustrations

Processing can export any drawing to PDF or SVG (PDF is nicer). But the functionality isn’t built-in—you need to request it.

```java
import processing.pdf.*;
```
Writing illustrations

Processing can export any drawing to PDF or SVG (PDF is nicer). But the functionality isn’t built-in—you need to request it.

```
import processing.pdf.*;
```

“Import directive”: make all the functionality in the named library available in this sketch.
Use `beginRecord()` and `endRecord()` to copy all drawing commands into an external file.

```java
import processing.pdf.*;

void setup()
{
    beginRecord( PDF, "output.pdf" );
    // Draw something here
    endRecord();
}
```
boolean recording = false;

void draw()
{
  if( recording ) {
    beginRecord( PDF, "output.pdf" );
  }

  // Draw as usual

  if( recording ) {
    endRecord();
    recording = false;
  }
}

void keyPressed()
{
  if( key == 's' ) {
    recording = true;
  }
}
3. Reading and writing text
Marley was dead: to begin with. There is no doubt whatever about that. The register of his burial was signed by the clergyman, the clerk, the undertaker, and the chief mourner. Scrooge signed it; and Scrooge's name was good upon 'Change, for anything he chose to put his hand to. Old Marley was as dead as a door-nail.

Mind! I don't mean to say that I know, of my own knowledge, what there is particularly dead about a door-nail. I might have been inclined, myself, to regard a coffin-nail as the deadliest piece of ironmongery in the trade. But the wisdom of our ancestors is in the simile; and my unhallowed hands shall not disturb it, or the Country's done for. You will therefore permit me to repeat, emphatically, that Marley was as dead as a door-nail.

Scrooge knew he was dead? Of course he did. How could it be otherwise? Scrooge and he were partners for I don't know how many years. Scrooge was his sole executor, his sole administrator, his sole assign, his sole residuary legatee, his sole friend and sole mourner. And even Scrooge was not so dreadful a cutpurse as to ask him any questions, either unanswerable or impertinent.

The mention of Marley's funeral brings me back to the point I started from. There is no doubt that Marley was dead. This must be distinctly understood, or nothing wonderful can come of the story I am going to relate. If we were not perfectly convinced that Hamlet's Father died before the play began, there would be nothing more remarkable in his taking a stroll at night, in an easterly wind, upon his own ramparts, than that there would be in any other middle-aged gentleman rashly turning out after dark in a breezy spot -- say Saint Paul's Churchyard for instance -- literally to astonish his son's weak mind.

Scrooge never painted out Old Marley's name. There it stood, years afterwards, above the warehouse door: Scrooge and Marley. The firm was known as Scrooge and Marley. Sometimes people new to the business called Scrooge Scrooge, and sometimes Marley, but he answered to both names: it was all the same to him.

Oh! But he would fain have kept his outward self with hearty alms; and he would fain have kept it warm and dry for himself; and would have kept both warm and dry for himself and others. But between you and me, he couldn't manage it. He couldn't make his eyes red, or his thin lips blue, and spoke out shrewdly in his grating voice. A frosty time was on his head, and on his eyebrows, and his wiry chin. He carried his own low temperature always along with him; he had his own private winter, and had no sort of warm thought to spare to any other. He was Shackleton's captain in the air; he was heaped up in the snow; he was buried in the ice; he was in every exposure that could make him more of himself, in the only one that could have made him less of himself. He was the frost over the place; he was the frost in the place; he was the frost of the place.

External heat and cold had little influence on Scrooge. No warmth could warm, no wintry weather chill him. No wind that blew was bitterer than he, no falling snow was more intent upon its purpose, no pelting rain less open to entreaty. Foul weather didn't know where to have him. The heaviest rain, and snow, and hail, and sleet, could boast of the advantage over him in one respect. It was never so thick and steady as his own rime and ice.

Scrooge never painted out Old Marley's name. There it stood, years afterwards, above the warehouse door: Scrooge and Marley. The firm was known as Scrooge and Marley. Sometimes people new to the business called Scrooge Scrooge, and sometimes Marley, but he answered to both names: it was all the same to him.

Plain text is the "default" mode of information storage and communication. Being able to work with text gives us access to large amounts of real-world data.
Hi Craig,

---

Hi Craig,
<table>
<thead>
<tr>
<th>Date</th>
<th>Low temperature</th>
<th>High temperature</th>
<th>Precipitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-Jan-14</td>
<td>-15.6</td>
<td>-8.9</td>
<td>0.1</td>
</tr>
<tr>
<td>02-Jan-14</td>
<td>-17.7</td>
<td>-15.1</td>
<td>0.1</td>
</tr>
<tr>
<td>03-Jan-14</td>
<td>-23.4</td>
<td>-13.1</td>
<td>0</td>
</tr>
<tr>
<td>04-Jan-14</td>
<td>-12.7</td>
<td>-2.5</td>
<td>0</td>
</tr>
<tr>
<td>05-Jan-14</td>
<td>-3.7</td>
<td>-1.2</td>
<td>19.1</td>
</tr>
<tr>
<td>06-Jan-14</td>
<td>-19.6</td>
<td>-2.1</td>
<td>7.7</td>
</tr>
<tr>
<td>07-Jan-14</td>
<td>-26.1</td>
<td>-18.7</td>
<td>1.5</td>
</tr>
<tr>
<td>08-Jan-14</td>
<td>-19.1</td>
<td>-11.1</td>
<td>0</td>
</tr>
<tr>
<td>09-Jan-14</td>
<td>-22.2</td>
<td>-8.3</td>
<td>0</td>
</tr>
<tr>
<td>10-Jan-14</td>
<td>-8.3</td>
<td>2.4</td>
<td>0</td>
</tr>
<tr>
<td>11-Jan-14</td>
<td>0.3</td>
<td>5.4</td>
<td>26.4</td>
</tr>
<tr>
<td>12-Jan-14</td>
<td>-0.8</td>
<td>1.3</td>
<td>0</td>
</tr>
<tr>
<td>13-Jan-14</td>
<td>0.4</td>
<td>5.8</td>
<td>0.2</td>
</tr>
<tr>
<td>14-Jan-14</td>
<td>-2.5</td>
<td>3.3</td>
<td>0</td>
</tr>
<tr>
<td>15-Jan-14</td>
<td>-8.5</td>
<td>-0.4</td>
<td>1.4</td>
</tr>
<tr>
<td>16-Jan-14</td>
<td>-8.7</td>
<td>-4</td>
<td>2.7</td>
</tr>
<tr>
<td>17-Jan-14</td>
<td>-8</td>
<td>-0.3</td>
<td>3.9</td>
</tr>
<tr>
<td>18-Jan-14</td>
<td>-10.1</td>
<td>-4.6</td>
<td>1.7</td>
</tr>
<tr>
<td>19-Jan-14</td>
<td>-10.7</td>
<td>-3.4</td>
<td>1.4</td>
</tr>
</tbody>
</table>
Reading text

Reading text from a file can be quite painful in many programming languages. Processing keeps it simple:

```java
String[] loadStrings(String filename) { ... }
```

Load a text file from the sketch folder. Break it up into lines and return an array of Strings, one per line.
PROCESSING  P R AA1 S EH0 S IH0 NG
PROCESSION  P R AH0 S EH1 SH AH0 N
PROCESSION(1)  P R OW0 S EH1 SH AH0 N
PROCESSIONAL  P R AH0 S EH1 SH AH0 N AH0 L
PROCESSIONAL(1)  P R OW0 S EH1 SH AH0 N AH0 L
PROCESSIONS  P R OW0 S EH1 SH AH0 N Z

dict.txt
void setup()
{
    String[] lines = loadStrings( "dict.txt" );
    printArray( lines );
}
void setup()
{
    String[] lines = loadStrings( "dict.txt" );
    printArray( lines );
}
Breaking up long lines

A line in a file may contain lots of individual chunks of data separated by whitespace. We’d like to break lines into words, just as we broke files into lines.

String[] splitTokens(String line) { ... }

Turn a line of text into an array of “words” (any non-whitespace characters separated by whitespace).

(Note that join() can reassemble individual strings into a single result.)
String s = " Marley was dead: to begin with. ";
String[] toks = splitTokens(s);
printArray(toks);
String s = "Marley was dead: to begin with. ";
String[] toks = splitTokens( s );
printArray( toks );

[0] "Marley"
[1] "was"
[2] "dead:"
[3] "to"
[4] "begin"
[5] "with."
Writing text

We know we can use `println()` to send any text to the console.

A similar mechanism allows us to create objects that stand in for text files. Sending those objects `println()` messages puts text into the file.

```java
PrintWriter createWriter(String filename) { ... }
```

Create an object that can output text to a file.
Idiom for writing text

```java
PrintWriter pw = createWriter( "output.txt" );

pw.println( "Hello" );
pw.println( mouseX );
pw.println( PI );
pw.println( "THE END" );

pw.flush();
pw.close();
```
Idiom for writing text

```java
PrintWriter pw = createWriter("output.txt");
pw.println("Hello");
pw.println(mouseX);
pw.println(PI);
pw.println("THE END");
pw.flush();
pw.close();
```

Create an object to write to.
Idiom for writing text

```java
PrintWriter pw = createWriter( "output.txt" );

pw.println( "Hello" );
pw.println( mouseX );
pw.println( PI );
pw.println( "THE END" );

pw.flush();
pw.close();
```

Send some text to the writer object.
Idiom for writing text

```java
PrintWriter pw = createWriter( "output.txt" );

pw.println( "Hello" );
pw.println( mouseX );
pw.println( PI );
pw.println( "THE END" );

pw.flush();
pw.close();
```

Send the data out to permanent storage and close the file.
Reasons to write text

Logging: Create a permanent record of the behaviour of the program to review later.

Persistence: Store information about the program’s state in an external file so that the sketch can restart with that state later.

Workflow: create text output that can be read by another program for further processing.