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.
<table>
<thead>
<tr>
<th>Disk Inventory X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Library</td>
</tr>
<tr>
<td>Documents</td>
</tr>
<tr>
<td>Desktop</td>
</tr>
<tr>
<td>Music</td>
</tr>
<tr>
<td>Videos</td>
</tr>
<tr>
<td>Photos</td>
</tr>
<tr>
<td>Music Library</td>
</tr>
<tr>
<td>Music Videos</td>
</tr>
<tr>
<td>Library Backup</td>
</tr>
<tr>
<td>Downloads</td>
</tr>
<tr>
<td>Documents</td>
</tr>
<tr>
<td>Pictures</td>
</tr>
<tr>
<td>Single Drive</td>
</tr>
<tr>
<td>System-archived</td>
</tr>
<tr>
<td>Android</td>
</tr>
<tr>
<td>System-archived.tar.gz</td>
</tr>
<tr>
<td>System-archived.tgz</td>
</tr>
<tr>
<td>System-archived.c cp</td>
</tr>
<tr>
<td>System-archived.a c</td>
</tr>
<tr>
<td>System-archived.a b</td>
</tr>
<tr>
<td>System-archived.a</td>
</tr>
<tr>
<td>System-archived.a b</td>
</tr>
<tr>
<td>System-archived.a c</td>
</tr>
<tr>
<td>System-archived.a b</td>
</tr>
<tr>
<td>System-archived.a c</td>
</tr>
<tr>
<td>System-archived.a b</td>
</tr>
<tr>
<td>System-archived.a c</td>
</tr>
<tr>
<td>System-archived.a b</td>
</tr>
<tr>
<td>System-archived.a c</td>
</tr>
<tr>
<td>System-archived.a b</td>
</tr>
<tr>
<td>System-archived.a c</td>
</tr>
<tr>
<td>System-archived.a b</td>
</tr>
<tr>
<td>System-archived.a c</td>
</tr>
<tr>
<td>System-archived.a b</td>
</tr>
</tbody>
</table>
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
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

```java
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 );
}
```

Draw the image at the given coordinates, scaled.
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 );

  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 );
}
**Image no-nos**

```java
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().
Standard image idiom

PImage img;

void setup()
{
  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

PImage img;  // Global variable to hold image.

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

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.

```java
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.

```c
    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.

```
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.

```
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()

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.

```
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 clergymen, the clerk, the undertaker, and the chief mourner. Scrooge signed it: and Scrooge's name was good upon 'Change, for anything.

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 deadest piece of ironmongery in the trade. But the wisdom of our ancestors had picked out the particular nail they knew to be the deadest, for it had been laid upon thening 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 he was dead, there is no possibility of ourRev.

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

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 he was dead, there is no possibility of our

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

Oh! But he was a tight-fisted hand at the grindstone, Scrooge! a squeezing, wrenching, grasping, scraping, clutching, covetous, old sinner! Hard and sharp as flint, from which no steel had ever struck out generous fire; secret, and self-contained, cool as ice, and as motionless.

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. He neither shrank from, nor sought to avoid, the sharper ights, nor the more impotent 

But what did Scrooge care? It was the very thing he liked. To edge his way along the crowded paths of life, warning all human sympathy to keep its distance, was what the knowing ones call "nuts".

Once upon a time -- of all the good days in the year, on Christmas Eve -- old Scrooge sat busy in his counting-house. It was cold, bleak, biting weather: foggy withal: and he could hear the people in the court outside go wheezing up and down, beating the temperature.

The door of Scrooge's counting-house was open that he might keep his eye upon his clerk, who in a dismal little cell beyond, a sort of tank, was copying letters. Scrooge had a very small fire, but

"A merry Christmas, uncle! God save you!" cried a cheerful voice. It was the voice of Scrooge's nephew, who came upon him so quickly that this was the first intimation he had of his approach.

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.
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
PROCESSOR  P R AA1 S EH2 S ER0
PROCESSOR'S  P R AA1 S EH2 S ER0 Z
PROCESSORS  P R AA1 S EH2 S ER0 Z
PROCH  P R AA1 K
PROCHASKA  P R AH0 HH AA1 S K AH0
PROCHAZKA  P R AH0 HH AA1 Z K AH0
PROCHNOW  P R AA1 N AW0
PROCIDA  P R OW0 CH IY1 D AH0
PROCK  P R AA1 K
PROCKTER  P R AA1 K T ER0
PROCLAIM  P R OW0 K L EY1 M
PROCLAIMED  P R OW0 K L EY1 M D
PROCLAIMING  P R OW0 K L EY1 M IH0 NG
PROCLAIMS  P R OW0 K L EY1 M Z
PROCLAMATION  P R AA2 K L AH0 M EY1 SH AH0 N
PROCLAMATIONS  P R AA2 K L AH0 M EY1 SH AH0 N Z
PROCLIVITIES  P R OW0 K L IH1 V AH0 T IH0 Z
PROCLIVITY  P R OW0 K L IH1 V AH0 T IH0
PROCONSUL  P R OW0 K AA1 N S AH0 L
<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>
</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.
void setup()
{
  String[] lines = loadStrings("dict.txt");
  printArray(lines);
}

[0] "PROCESSING  P R AA1 S EH0 S IH0 NG"
[1] "PROCESSION  P R AH0 S EH1 SH AH0 N"
[2] "PROCESSION(1)  P R OW0 S EH1 SH AH0 N"
[3] "PROCESSIONAL  P R AH0 S EH1 SH AH0 N AH0 L"
[4] "PROCESSIONAL(1)  P R OW0 S EH1 SH AH0 N AH0 L"
[5] "PROCESSIONS  P R OW0 S EH1 SH AH0 N Z"
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.

```java
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);

[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

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

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

Send some text to the writer object.

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();
```

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.