

Introduction to Processing

Mithun Paul, CS345, Fall 2017

Why Processing?

Sample Homework problem.

<https://docs.google.com/document/u/1/d/1glKDj4YgYwIIYRN0uePYP7aG2UWMvjLIJ8mwVKMStZg/pub>

<https://www2.cs.arizona.edu/classes/cs345/fall17/prog1.pdf>

Solution to HW problem

What is in code?

What is in file1.in?

What is processing?

- A simple programming environment that was created to make it easier to **develop visually oriented applications** with an emphasis on animation and providing users with instant feedback through interaction

WHAT IS PROCESSING?

- Processing is a scripting language that allows you to write simple graphics programs using an IDE.
-
- Processing is an Imperative, Procedural, Object-Oriented programming language
-

Resources

- Processing web site:
<http://www.processing.org/>
- • Reference:
<http://www.processing.org/reference/index.html>

Linear motion:

- <http://www.processing.org/learning/topics/linear.html>
- Sequential animation:
<http://www.processing.org/learning/topics/sequential.html>

Processing consists of:

- **The Processing Development Environment (PDE).**
 - This is the software that runs when you double-click the Processing icon.
 - The PDE is an Integrated Development Environment (IDE) with a minimalist set of features designed as a simple introduction to programming or for testing one-off ideas.
- A language **syntax, identical to Java** but with a few modifications.

Sketching with Processing

- A Processing program is called a *sketch*
 - The idea is to make Java-style programming feel more like scripting, and adopt the process of scripting to quickly write code
 - Sketches are stored in the *sketchbook*, a folder that's used as the default location for saving all of your projects.
- Advanced programmers need not use the PDE, and may instead choose to use its libraries with the Java environment of choice.

SIMPLE PROCESSING PROGRAMS

Static sketch

- A program written as a list of statements (like the previous examples) is called a *static* mode sketch.
- In static mode, a series of functions are used to perform tasks or create a single image without any animation or interaction.
- Interactive programs are drawn as a series of frames, which you can create by adding functions titled `setup()` and `draw()` as shown in the code below.
 - These are built-in functions that are called automatically

MY FIRST PROCESSING SKETCH

SETUP and DRAW

```
void setup()
{
    size(400, 400);
    stroke(255);
    background(192, 64, 0);
}

void draw()
{
    line(150, 25, mouseX, mouseY);
}
```

- The **setup()** block runs once, and the draw() block runs repeatedly.
- As such, setup() can be used for any initialization; in this case, setting the screen size, making the background orange, and setting the stroke color to white.
- The **draw()** block is used to handle animation. The size() function must always be the first line inside setup().


- Created by using functions `setup()` and `draw()`
- Draws a series of frames
- Example:

```
void setup() {  
  size(400, 400)  
  stroke(255);  
  background(192, 64, 0);  
}
```



Called once

```
void draw() {  
  line(150, 25, mouseX, mouseY);  
}
```



Called repeatedly
(default: 60 times/sec)

- How can you change the sketch so that it draws just a single line that follows the mouse?

QN) WHAT DOES FUNCTION SIZE() DO?

What parameters does it expect?

Reference:

<http://www.processing.org/reference/index.html>

https://processing.org/reference/size_.html

More about size()

- Can only be called once
- Cannot be used to resize window
- Sets width and height variables for the window
- Select renderer with optional third argument
 - P2D
 - P3D
 - PDF
- Set window to maximum display size with
`size(displayWidth, displayHeight);`

Stroke

- `stroke(255);` // sets the stroke color to white
- `stroke(255, 255, 255);` // identical to the line above
- `stroke(255, 128, 0);` // bright orange (red 255, green 128, blue 0)
- `stroke(#FF8000);` // bright orange as a web color
- `stroke(255, 128, 0, 128);` // bright orange with 50% transparency
- The same alternatives work for the `fill()` function, which sets the fill color, and the `background()` function, which clears the display window.
- Like all Processing functions that affect drawing properties, the fill and stroke colors affect all geometry drawn to the screen until the next fill and stroke functions.

THINGS TO REMEMBER

To change the size of the display window and set the background color, type in the code below:

```
size(400, 400);  
background(192, 64, 0);  
stroke(255);  
line(150, 25, 270, 350);
```

- To draw just a single line that follows the mouse, move the `background()` function to the `draw()` function, which will clear the display window (filling it with orange) each time `draw()` runs.

```
void setup() {  
    size(400, 400);  
    stroke(255);  
}
```

```
void draw() {  
    background(192, 64, 0);  
    line(150, 25, mouseX,  
mouseY);  
}
```

MY FIRST PROCESSING SKETCH

Create a basic canvas, and draw a line with white stroke.

QN) WHAT DOES FUNCTION LINE() DO?

What parameters does it expect?

Another look at Solution to HW problem

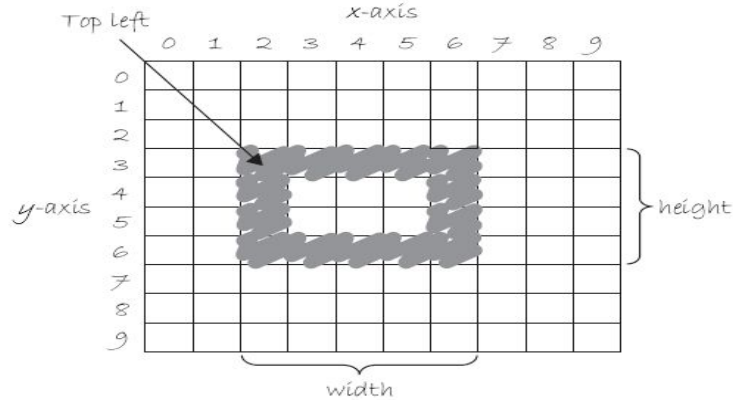
Setup and draw in code?

test2.in

cmd+Click = show function usages

Drawing Complex Images

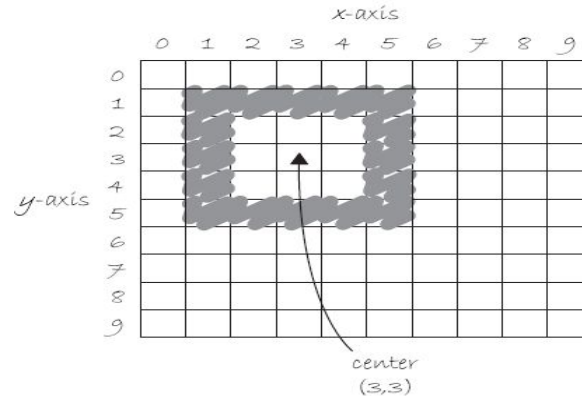
rect(), rectMode()



width height
rect (2,3,5,4);
top left top left
x y

Default:
rectMode(CORNER)

rectMode(CENTER)



rectMode (CENTER);
rect (3,3,5,5);
center center
x y
height
width


```
rect(30, 20, 55, 55);
```

1. Draws a rectangle to the screen.
2. Four-sided shape with every angle at ninety degrees.
3. first two parameters set the location of the upper-left corner,
4. the third sets the width, and
5. the fourth sets the height.

https://www.processing.org/reference/rect_.html

`rectMode (CENTER) ;`

https://www.processing.org/reference/rectMode_.html

1. Modifies the location from which rectangles are drawn by changing the way in which parameters given to `rect()` are interpreted
2. `rectMode (CENTER)` interprets the first two parameters of `rect()` as the shape's center point, while the third and fourth parameters are its width and height.
3.

```
rectMode (CENTER); // Set rectMode to CENTER  
fill(100); // Set fill to gray  
rect(50, 50, 30, 30); // Draw gray rect using CENTER  
mode
```
4. If you want the center of the rectangle to be in the center of the canvas what do you do?

THINGS TO REMEMBER

The (0, 0) coordinate is the upper left-hand corner of the display window.

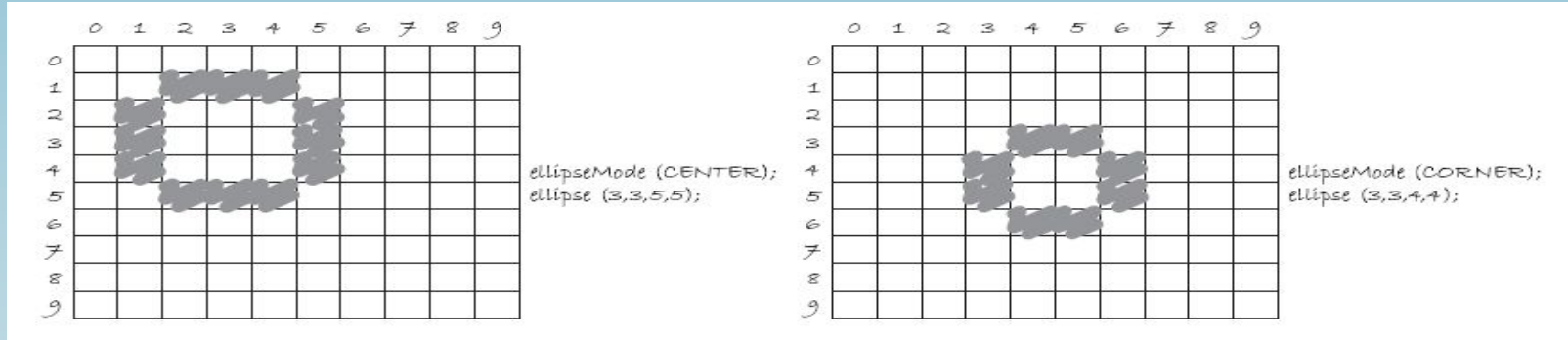
X Axis: From top Left Corner: Left to Right ----->

Y Axis: From top Left Corner: Top to Bottom

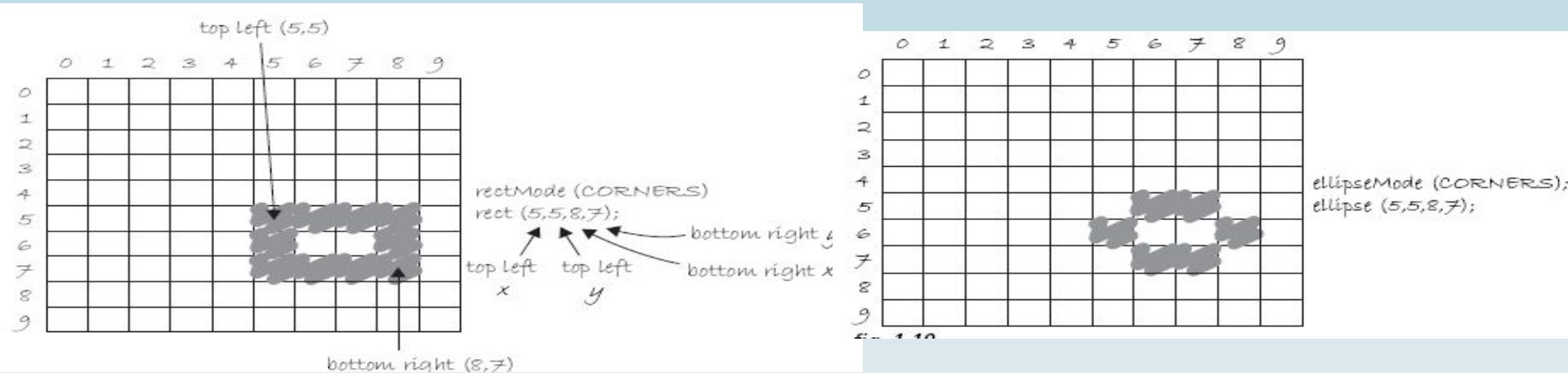
```
rect(10, 100, 50, 50);
```

```
rect(100, 10, 30, 30);
```

ellipse(), ellipseMode()



rectMode(CORNERS), ellipseMode(CORNERS)



```
ellipse (56, 46, 55, 55) ;
```

The first two parameters set the location,

The third and fourth parameters set the shape's width and height.

An ellipse with equal width and height is a circle

triangle(), arc(), quad(), curve()

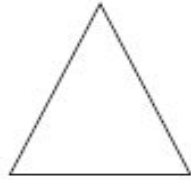
triangle()

arc()

quad()

curve()

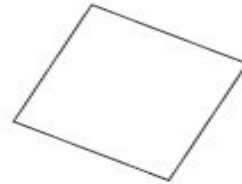
In Processing, every shape has a **stroke()** or a **fill()** or both. The **stroke()** is the outline of the shape, and the **fill()** is the interior of that shape.



Triangle



Arc

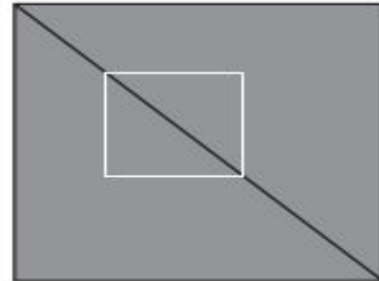


Quad



Curve

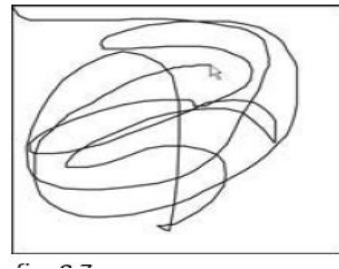
```
background(150);  
stroke(0);  
line(0,0,100,100);  
stroke(255);  
noFill();  
rect(25,25,50,50);
```



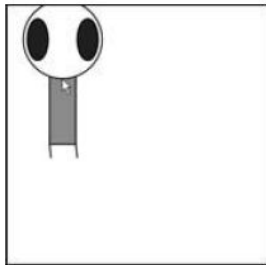
Drawing a continuous line

```
void setup() {  
  size(200,200);  
  background(255);  
  smooth();  
}  
  
void draw() {  
  stroke(0);  
  line(pmouseX,pmouseY,mouseX,mouseY);  
}
```

Draw a line from previous mouse location to current mouse location.



The function ***smooth()*** enables “anti-aliasing” which smooths the edges of the shapes. ***no smooth()*** disables anti-aliasing.



frameRate(), which requires an integer between 1 and 60, enforces the speed at which *Processing* will cycle through ***draw()***.

frameRate(30), for example, means 30 frames per second, a traditional speed for computer animation.

Bouncing ball

```
int x = 0;
int speed = 1;

void setup() {
  size(200,200);
  smooth();
}

void draw() {
  background(255);

  x = x + speed;

  if ((x > width) || (x < 0)) {
    speed = speed * -1;
  }

  // Display circle at x location
  stroke(0);
  fill(175);
  ellipse(x,100,32,32);
}
```

Add the current speed to the x location.

If the object reaches either edge, multiply speed by -1 to turn it around.

Class Exercise 1

1. Draw a white circle with diameter 100 pixels in the center of a red window the size of the display.

INTERACTIVE EVENTS



Active Sketch

- Most programs will employ active mode, which use the `setup()` and `draw()` blocks.
- More advanced mouse handling can also be introduced; for instance, the `mousePressed()` function will be called whenever the mouse is pressed.

```
void setup() {  
    size(400, 400);  
    stroke(255);  
}
```

```
void draw() {  
    line(150, 25, mouseX, mouseY);  
}
```

```
void mousePressed() {  
    background(192, 64, 0);  
}
```

mousePressed() function

- Called whenever the mouse is pressed
- Example

```
void setup() {  
  size(400, 400);  
  stroke(255);  
}  
void draw() {  
  line(150, 25, mouseX, mouseY);  
}  
void mousePressed() {  
  background(192, 64, 0);  
}
```

Class Exercise 2

1. Draw a white circle with diameter 100 pixels in the center of a red window the size of the display.
2. Modify your sketch so that the circle changes color when you press the mouse.

Exporting to an Application

- You can create an application for MacOS, Linux, or Windows by using
File → Export Application

FOR HOMEWORKS

- We will provide GUI layer
- Attach GUI +Your Logic
- Fancy
- Ready to go when out of the box.
- Modular code

Attach GUI +Your Logic

```
else if (nextButton.mouseOver()) {
```

```
//   Your logic
```

```
}
```


MORE TOOLS FOR YOU

Variables

- variables provide a way to save information within your sketch and use it to control the position, size, shape, etc of what you are drawing
- variables have a data type, a name and a value
- valid data types are:
 - int — for storing integers (whole numbers)
 - float — for storing floating point (real) numbers
 - boolean — for storing true or false values
 - char — for storing single characters
 - String — for storing multiple (strings of) characters
- example:

```
int x1 = 10;
```

```
int y1 = 10;
```

```
int x2 = 20;
```

```
int y2 = 20;
```

```
line( x1, y1, x2, y2 );
```

Looping

- loops are used for doing things repeatedly
- there are two basic types of loops:
 - for loops
 - while loops
- loops are handy for animation, because you typically want to display things repeatedly when you are doing animation
- looping is a type of:
 - repetition (required element of imperative programming)
 - iteration (same thing as repetition)

for loops

- for loops repeat things for a fixed number of times
- syntax:

```
for ( init; test; update ) {  
    statements  
}
```

- example:

```
int x = 10;  
int y1 = 10;  
int y2 = 20;  
for ( int i=0; i<10; i++ ) {  
    line( x, y1, x, y2 );  
    x = x + 10;  
}
```

while loops

- while loops repeat things as long as a condition holds true
- syntax:

```
while ( expression ) {  
    statements  
}
```

- example:

```
int x = 10;  
int y1 = 30;  
int y2 = 40;  
while ( x < width ) {  
    line( x, y1, x, y2 );  
    x = x + 10;  
}
```

Standard Processing Program

1. Setup any variables or classes you are going to use.
2. Use setup() function to specify things to do once, when the sketch first opens
3. Use draw() function to specify things to do repeatedly
 - use `frameRate()` function to specify how often things should be repeated in `draw()`;
 - default frame-rate is 60 (60 frames per second)
 - NOTE: call to `frameRate()` should be done inside `setup()` function
4. Declare and event-listeners that you are going to use.
5. Declare any custom made functions you are going to use.
6. Declare any classes that you are going to use.

Animation

Basic animation involves the following steps:

1. Drawing initial frame - perhaps in `setup()`.
2. Waiting some amount of time (e.g., 1/60th of a second)
 - Processing does that automatically
3. Erasing the screen.
 - Usually be reapplying the background (draw does this automatically).
4. Drawing the next frame.
5. Repeating steps 2-4, until you are ready to stop animating.

There are two basic ways to implement animation:

6. Drawing your own shapes, text, etc.
7. Displaying a GIF or other image file

Vector Animation (drawing shapes)

From <http://www.processing.org/learning/topics/linear.html>

```
float a = 100;
void setup() {
  size( 640, 200 );
  stroke( 255 );
}
void draw() {
  background( 51 );
  a = a - 0.5;
  if ( a < 0 ) {
    a = height;
  }
  line( 0, a, width, a );
}
```


Bitmap Animation (using pictures)

<http://www.processing.org/learning/topics/sequential.html>

```
int numFrames = 4; // The number of frames in the animation
int frame = 0;
PImage[ ] images = new PImage[numFrames];
    void setup() {
        size( 200, 200 );
        frameRate( 30 );
        images[0] = loadImage("PT_anim0000.gif");
        images[1] = loadImage("PT_anim0001.gif");
        images[2] = loadImage("PT_anim0002.gif");
        images[3] = loadImage("PT_anim0003.gif");
    }
    void draw() {
        frame = ( frame + 1 ) % numFrames; // Use % to cycle through frames
        image( images[frame], 50, 50 );
    }
```

Movement and Animation

```
int xPos = 0;
int yPos = 50;

....

void draw() {
    xPos = (xPos + 2) % width;
    frame = ( frame + 1 ) % numFrames; // Use % to cycle through frames
    image( images[frame], xPos, yPos );
}

...

void keyPressed() {
    if (key == CODED) {
        if (keyCode == UP) {
            yPos = yPos - 2;
        } else if (keyCode == DOWN) {
            yPos = yPos + 2;
        }
    }
}
}
```

More Mouse Interaction

- `mouseX` and `mouseY`
 - indicate (x, y) location of mouse pointer
- `mouseClicked()`
 - handles behavior when user clicks mouse button (press and release)
- `mouseMoved()`
 - handles behavior when user moves mouse (moves it without pressing button)
- `mouseDragged()`
 - handles behavior when user drags mouse (moves it with button pressed)
- `mouseButton`
 - indicates which button was pressed, on a multi-button mouse (on a Mac, use Cntl-click for left mouse button, Alt-click for middle mouse button and Apple-click for right mouse button)

Example 1 (mouse location)

```
void setup() {  
    size( 200, 200 );  
}
```

```
void draw() {  
    background( #cccccc );  
    // What happens if you remove the line above?  
    fill( #000099 );  
    rect( mouseX, mouseY, 20, 20 );  
}
```

Example 2 (mouseMoved)

```
void setup() {  
    size( 200, 200 );  
}  
void draw() {  
    background( #cccccc );  
    fill( #990000 );  
    rect( mouseX, mouseY, 20, 20 );  
}  
void mouseMoved() {  
    fill( #000099 );  
    rect( mouseX, mouseY, 20, 20 );  
}  
/* how does this behave differently from the mouse location example?  
*/
```

Example 3 (mouseDragged)

```
void setup() {  
    size( 200, 200 );  
}  
void draw() {  
    background( #cccccc );  
    fill( #990000 );  
    rect( mouseX, mouseY, 20, 20 );  
}  
void mouseMoved() {  
    fill( #000099 );  
    rect( mouseX, mouseY, 20, 20 );  
}  
void mouseDragged() {  
    fill( #009900 );  
    rect( mouseX, mouseY, 20, 20 );  
}
```

Example #4 (mouseClicked)

```
int r = 0;
int g = 0;
int b = 0;
void setup() {
    size( 200, 200 );
}
void draw() {
    background( #ffffff );
    fill( r, g, b );
    rect( 50, 50, 20, 20 );
}
void mouseClicked() {
    r = r + 51;
    if ( r > 255 ) {
        r = 0;
        g = g + 51;
        if ( g > 255 ) {
            g = 0;
            b = b + 51;
            if ( b > 255 ) {
                b = 0;
            }
        }
    }
}
```

Example #5 (mouseButton)

```
void setup() {  
    size( 200, 200 );  
}  
void draw() {  
    background( #cccccc );  
    rect( mouseX, mouseY, 20, 20 );  
}  
void mousePressed() {  
    if ( mouseButton == LEFT ) {  
        fill( #990000 );  
    }  
    else if ( mouseButton == CENTER ) {  
        fill( #009900 );  
    }  
    else if ( mouseButton == RIGHT ) { // Ctrl-click on mac  
        fill( #000099 );  
    }  
}
```


Other useful functions

System Variables

width —Width (in pixels) of sketch window.

height —Height (in pixels) of sketch window.

frameCount —Number of frames processed.

frameRate —Rate that frames are processed (per second).

screen.width —Width (in pixels) of entire screen.

screen.height —Height (in pixels) of entire screen.

key —Most recent key pressed on the keyboard.

keyCode —Numeric code for key pressed on keyboard.

keyPressed —True or false? Is a key pressed?

```
void setup() {  
  size(200,200);  
  frameRate(30);  
}
```

```
void draw() {  
  background(100);  
  stroke(255);  
  fill(frameCount/2);  
  rectMode(CENTER);  
  rect(width/2,height/2,mouseX+10,mouseY+10);  
}
```

frameCount is used to color a rectangle.

```
void keyPressed() {  
  println(key);  
}
```

The rectangle will always be in the middle of the window if it is located at (width/2, height/2).

Exporting and distributing your work

- One of the most significant features of the Processing environment is its ability to bundle your sketch into an applet or application with just one click
- Select File → Export to package your current sketch as an applet

Creating images from your work

- Images are saved with the `saveFrame()` function.
- Adding `saveFrame()` at the end of `draw()` will produce a numbered sequence of TIFF-format images of the program's output, named *screen-0001.tif*, *screen-0002.tif*, and so on.
- A new file will be saved each time `draw()` runs
- You can also specify your own name and file type for the file to be saved with a function like
 - `saveFrame("output.png")`
- To do the same for a numbered sequence, use `#` (hash marks) where the numbers should be placed:
 - `saveFrame("output-####.png");`

Examples and references

- The examples can be accessed from the File → Examples menu
- To see how a function works, select its name, and then right-click and choose Find in Reference from the pop-up menu

The Data Folder

- `loadImage()` and `loadStrings()` functions use the data folder that is a subdirectory inside the sketch folder.
- When you export an application, the data folder gets bundled with it.
- Use Sketch → Add File to add a file to the data folder.

Libraries add new features

- A *library* is a collection of code in a specified format that makes it easy to use within Processing.
- One example is the PDF Export library. This library makes it possible to write PDF files directly from Processing.
To use the PDF library in a project, choose Sketch → Import Library → PDF Export. This will add the following line to the top of the sketch:

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

- Now that the PDF library is imported, you may use it to create a file. For instance, the following line of code creates a new PDF file named *lines.pdf* that you can draw to.

```
beginRecord(PDF, "line.pdf");
```