Processing Basics

CS 110
Eric Eaton
Software

Processing

– Already installed in the CS Lab
– Also available for your own computer @ www.processing.org
– Processing == Java

Book

Primitive 2D Shapes

- point
- line
- triangle
- rect (rectangle)
- quad (quadrilateral, four-sided polygon)
- ellipse
- arc (section of an ellipse)
- curve (Catmull-Rom spline)
- bezier (Bezier curve)
Language (API). The Processing Language has been designed to facilitate the creation of sophisticated visual and conceptual structures.

**Structure**
- [1 (array access)]
- = (assign)
- catch
- class
- , (comma)
- // (comment)
- {} (curly braces)
- delay()
- /* */ (doc comment)
- . (dot)

**Shape**
- PShape
- 2D Primitives
- arc()
- ellipse()
- line()
- point()
- quad()
- rect()
- triangle()

**Color**
- Setting
  - background()
  - colorMode()
  - fill()
  - noFill()
  - noStroke()
  - stroke()
Anatomy of a Function Call

- Function name
- Parentheses
- Arguments
- Statement terminator

`line(10, 10, 50, 80);`
Coordinate System

(0, 0)

+y

+x
Pixels
Processing Canvas

\texttt{size( width, height );}

Set the size of the canvas.

\texttt{background( [0..255] );}

Set the background grayscale color.
Drawing Primitives

point( x, y );

line( x1, y1, x2, y2 );

triangle( x1, y1, x2, y2, x3, y3 );

quad( x1, y1, x2, y2, x3, y3, x4, y4 );

rect( x, y width, height );

ellipse( x, y, width, height );
smooth() vs. noSmooth()
Colors

Composed of four elements:

1. Red
2. Green
3. Blue
4. Alpha (Transparency)
Why 0 .. 255?
Shape Formatting

1. Fill color
2. Line thickness
3. Line color

*These are properties of your paintbrush, not of the object you are painting.*
Fill Color

fill(gray);
fill(gray, alpha);
fill(red, green, blue);
fill(red, green, blue, alpha);

noFill();
Stroke (Line) Color

stroke(gray);
stroke(gray, alpha);
stroke(red, green, blue);
stroke(red, green, blue, alpha);

noStroke();
strokeCap()  

smooth();  
strokeWeight(12.0);  
strokeCap(ROUND);  
line(20, 30, 80, 30);  
strokeCap(SQUARE);  
line(20, 50, 80, 50);  
strokeCap(PROJECT);  
line(20, 70, 80, 70);

strokeWeight()  

smooth();  
strokeWeight(1);    // Default  
line(20, 20, 80, 20);  
strokeWeight(4);    // Thicker  
line(20, 40, 80, 40);  
strokeWeight(10);   // Beastly  
line(20, 70, 80, 70);

http://processing.org/reference/strokeCap_.html  
http://processing.org/reference/strokeWeight_.html
ellipseMode

```
ellipseMode(CENTER);
ellipse(35, 35, 50, 50);
ellipseMode(CORNER);
fill(102);
ellipse(35, 35, 50, 50);
```

rectMode

```
rectMode(CENTER);
rect(35, 35, 50, 50);
rectMode(CORNER);
fill(102);
rect(35, 35, 50, 50);
```

http://processing.org/reference/ellipseMode_.html
http://processing.org/reference/rectMode_.html
random(high);
random(low, high);

Generate a random number in the range
low (or 0) to high

mouseX
mouseY

Built-in predefined variables that hold the
current mouse X and Y locations

print( something );
println( something );

Print something to the Processing console.
void setup()
{
    // Called once when program starts
}

void draw()
{
    /* Called repeatedly
        while program runs */
}
randomEllipse

void setup()
{
  size(300, 300);
  smooth();
}

void draw()
{
  fill(random(255), random(255), random(255));
  ellipse(mouseX, mouseY, 30, 30);
}
Controlling draw()

`frameRate(fps);`
Sets number of frames displayed per second.
i.e. the number of times `draw()` is called per second. Default = 60.

`noLoop();`
Stops continuously calling `draw()`.

`loop();`
Resumes calling `draw()`.
void mousePressed() {
    // Called when the mouse is pressed
}

void mouseReleased() {
    // Called when the mouse is released
}

void mouseClicked() {
    // Called when the mouse is pressed and released
    // at the same mouse position
}

void mouseMoved() {
    // Called while the mouse is being moved
    // with the mouse button released
}

void mouseDragged() {
    // Called while the mouse is being moved
    // with the mouse button pressed
}
void keyPressed() {
    // Called each time a key is pressed
}

void keyReleased() {
    // Called each time a key is released
}

void keyTyped() {
    // Called when a key is pressed
    // Called repeatedly if the key is held down
}
keyCode vs. key

key
  - A built-in variable that holds the character that was just typed at the keyboard

keyCode
  - A built-in variable that holds the code for the keyboard key that was touched

All built-in keyboard interaction functions …
  • Set keyCode to the integer that codes for the keyboard key
  • Set key to the character typed
  • All keyboard keys have a keyCode value
  • Not all have a key value
## ASCII - American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>!</td>
<td>&quot;</td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>'</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>~</td>
<td>€</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>,</td>
<td>f</td>
<td>”</td>
<td>“</td>
<td>†</td>
<td>‡</td>
<td>^</td>
<td>%</td>
<td>Š</td>
<td>¢</td>
</tr>
<tr>
<td>140</td>
<td>Æ</td>
<td>Ž</td>
<td>Ÿ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>i</td>
<td>c</td>
<td>£</td>
<td>¶</td>
<td>¥</td>
<td>!</td>
</tr>
<tr>
<td>170</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>§</td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>©</td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More Graphics

arc(…)
curve (…)
bézier(…)
shape(…)


Arcs

arc( x, y, width, height, start, stop );

An arc is a section of an ellipse

x, y, width, height
location and size of the ellipse
start, stop
arc bounding angles (in radians)
Arcs

\texttt{arc( x, y, width, height, start, stop );}
Spline Curves

curve( x1, y1, x2, y2, x3, y3, x4, y4 );

Spline: A smooth line drawn through a series of points
A curve is a Catmull-Rom (cubic Hermite) spline
defined by four points

x2, y2 and x3, y3
beginning/end points of visual part of curve
x1, y1 and x4, y4
control points that define curve curvature
Spline Curves

curve( x1, y1, x2, y2, x3, y3, x4, y4 );
Bézier Curves

`bezier( x1, y1, cx1, cy1, cx2, cy2, x2, y2 );`

A smooth curve defined by two anchor points and two control points

\[ x2, \ y2 \ and \ x2, \ y2 \]
anchor points of bézier curve
\[ cx1, \ cy1 \ and \ cx2, \ cy2 \]
control points that define curvature
Bézier Curves

bezier( x1, y1, cx1, cy1, cx2, cy2, x2, y2 );

start : (260.0, 91.0)
ctrl start : (192.0, 297.0)
ctrl end : (388.0, 130.0)
end : (389.0, 301.0)
Custom Shapes

• Composed of a series of vertexes (points)
• Vertexes may or may not be connected with lines
• Lines may join at vertexes in a variety of manners
• Lines may be straight, curves, or bézier splines
• Shape may be closed or open
Custom Shapes

beginShape( [option] );

vertex( x, y );

curveVertex( x, y );

bezierVertex( cx1, cy1, cx2, cy2, x, y );

endShape( [CLOSE] );
strokeJoin()

noFill();
smooth();
strokeWeight(10.0);
strokeJoin(MITER);
beginShape();
vertex(35, 20);
vertex(65, 50);
vertex(35, 80);
endShape();

noFill();
smooth();
strokeWeight(10.0);
strokeJoin(BEVEL);
beginShape();
vertex(35, 20);
vertex(65, 50);
vertex(35, 80);
endShape();

noFill();
smooth();
strokeWeight(10.0);
strokeJoin(ROUND);
beginShape();
vertex(35, 20);
vertex(65, 50);
vertex(35, 80);
endShape();
More Color

colorMode(RGB or HSB);

RGB: (red, green, blue)

HSB:
  hue
    • “pure color”
  saturation
    • “intensity”
  brightness
    • “lightness”
## Decimal vs. Binary vs. Hexadecimal

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00000000</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>00000010</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>00000011</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>00000100</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>00000101</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>00000110</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>00000111</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>00001000</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>00001001</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>00001010</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>00001011</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>00001100</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>00001101</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>00001110</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>00001111</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>00010000</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>00010001</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>00010010</td>
</tr>
</tbody>
</table>
Images

`loadImage(filename);`
- Loads an image from a file in the data folder in sketch folder.
  - Must be assigned to a variable of type PImage.

`image(img, X, Y, [X2, Y2]);`
- Draws the image `img` on the canvas at X, Y
  - Optionally fits image into box X,Y and X2,Y2

`imageMode(CORNER);`
- X2 and Y2 define width and height.

`imageMode(CORNERS);`
- X2 and Y2 define opposite corner.
PImage img;

void setup()
{
  size(500, 400);
  img = loadImage("natura-morta.jpg");
  image(img, 50, 40);
}
Example Sketches...

– LadyBug1
– Monster1
– Ndebele
– Penguin1
– SouthParkCharacter1
– Sushi
– GiorgioMorandi
OpenProcessing

http://www.openprocessing.org/

– Bryn Mawr and SMU student sketches
Dropbox

• https://www.dropbox.com/
Processing.JS

• A Javascript implementation of Processing
• Runs in any modern web browser
  – Does not run well in IE8 and under
• Most of Processing is implemented
  – Images are processed slowly
  – No file IO
• http://processingjs.org
Studio Sketchpad

• Collaboratively edit, run and chat about a Processing.js program

• http://sketchpad.cc/