1 **Processing: Understanding Images & Pixels**

CST112

2 **What is Processing?**

- ▶Processing is a language that was (still is being) developed by Ben Fry (UCLA) and Casey Reas (MIT/Broad Institute)
- ▶It is a "shell" built around Java programming language
  - ▶Simplifies programming of visual programs
  - ▶Yields immediate graphical output
- ▶Very suitable language for beginners learning how program

3 **Programming Approaches (Page 1)**

- ▶Traditional command line output:
  1. Write your code as text
  2. Code produces output on the command line
  3. Enter text on the command line to interact with the program

4 **Programming Approaches (Page 2)**

- ▶Learning programming with Processing:
  1. Write your code as text
  2. Code produces visuals in a window
  3. User interacts with those visuals via the mouse (and more ...)

5 **Coordinate Systems**

- ▶In geometry, a system which uses numbers (the coordinates) to determine position of a point or other geometric element
- ▶Order of coordinates is significant; sometimes are identified by a letter, as in the "*x*-coordinate"
- ▶Use of coordinate systems allow problems in geometry to be translated into problems about numbers and vice versa

6 **Number Line**

- ▶Simplest example of a coordinate system is identification of points on a line with real numbers using number line
- ▶In this system, an arbitrary point *O* (the origin) is chosen on a given line (usually zero)
- ▶Coordinate of a point *P* is defined as the signed distance from *O* to *P*, taken as a positive or negative value

7 **Cartesian Coordinate System**

- ▶A Cartesian coordinate system is a *2-dimensional* system specifying a point in a plane by a *pair* of numerical coordinates
  - ▶Coordinates are the *signed* distances from the point to *two* fixed perpendicular directed lines, measured in the same unit of length

- ▶Each reference line is called a coordinate axis or just axis of the system, and the point where they meet is its origin
  - ▶The x-axis and the y-axis
- ▶

9 **Pixels**

- ▶A pixel (or picture element) is a single "point" of light on a computer screen
  - ▶Smallest addressable screen element of picture that can be controlled
- ▶Each pixel has its own address—corresponds to its *coordinates*
- ▶Arranged normally in a two-dimensional grid, and often are represented using dots or squares

11 **Pixel Coordinate System**

- ▶The pixel coordinate system defines the origin as the upper-left corner of the computer screen
  - ▶E.g. the x- (horizontal) and y- (vertical) coordinate values zero (0, 0) both are at the upper-left corner
- ▶Pixel centers are offset by (0.5, 0.5) from integer locations

14 **Draw a Point (Page 1)**

- ▶To draw a point (a dot) that fills exactly one pixel in a pixel coordinate system:
  - ▶The x-coordinate is specified as the first value
  - ▶The y-coordinate is specified as the second value
- ▶For example:
  - ▶Draw the point at (20, 20)

16 **Draw a Point (Page 2)**

- ▶The point "function" in Processing is used to draw a point inside the "output window"
  - ▶The default window size is 100 by 100 pixels
- ▶Format:

  point(*x-coord*, *y-coord*);

17 **Draw a Point (Page 3)**

- ▶Example:

  point(20, 20);
  - ▶The two "arguments" (20, 20) are the x-coordinate and y-coordinate
  - ▶All Processing (and Java) statements end with a semicolon (;)

18 **Draw a Line (Page 1)**

- ▶To draw a line in a pixel coordinate system:
  - ▶The start of the line (point *A*) is specified as one set of coordinates
  - ▶The end the line (point *B*) is specified as a second set of coordinates
- ▶For example:

▶Start the line at (10, 25)—point *A*
▶End the line at (75, 100)—point *B*

20 **Draw a Line (Page 2)**

▶The line function in Processing is used to draw a line inside the output window
▶Format:
line(*x-PointA*, *y-PointA*, *x-PointB*, *y-PointB*);

21 **Draw a Line (Page 3)**

▶Example:
line(10, 25, 75, 100);
▶First two arguments (10, 25) are the x- and y-coordinates of point *A*
▶Last two arguments (75, 100) are x- and y-coordinates of point *B*

22 **Draw a Rectangle (Page 1)**

▶To draw a rectangle specify:
▶The x- and y-coordinates of its upper-left corner
▶Its width and height
▶For example:
▶Specify the upper left corner (20, 10)—its x- and y-coordinates
▶Specify its width and height (60, 40)

24 **Draw a Rectangle (Page 2)**

▶The rect function in Processing is used to draw a rectangle inside the output window
▶Format:
rect(*x-UpperLeft*, *y-UpperLeft*, *width*, *height*);

25 **Draw a Rectangle (Page 3)**

▶Example:
rect(20, 10, 60, 40);
▶First two arguments (20, 10) are x- and y-coordinates of the upper-left corner
▶Last two arguments (60, 40) are width and height

26 **The rectMode Function (Page 1)**

▶There are two other formats for drawing rectangles:
▶The x- and y-coordinates specify the *centerpoint* of the rectangle in addition to the width and height
rect(*x-Center*, *y-Center*, *width*, *height*);
▶The *two* sets of x- and y-coordinates specify the *upper-left* and *lower-right* corners of the rectangle
rect(*x-UpperLeft*, *y-UpperLeft*, *x-LowerRight*, *y-LowerRight*);

27 **The rectMode Function (Page 2)**

▶Select the format by assigning one of three values using the rectMode function prior

to calling the rect function:

rectMode(CORNER); — rect function arguments specify upper-left corner, width, and height of the rectangle (the *default*)

rectMode(CENTER); — rect arguments specify the centerpoint, width, and height of the rectangle

rectMode(CORNERS); — rect arguments specify upper-left and lower-right corners of the rectangle

29 **The rectMode Function (Page 3)**

▶To specify the centerpoint, width, and height:

```
rectMode(CENTER);
rect(50, 30, 60, 40);
```

31 **The rectMode Function (Page 4)**

▶To specify the upper-left and lower-right corners:

```
rectMode(CORNERS);
rect(20, 10, 80, 50);
```

32 **Draw a Square (Page 1)**

▶To draw a square specify:

- ▶The x- and y-coordinates of its upper-left corner
- ▶Its width (same as its height which is not specified)

▶For example:

- ▶Specify the upper left corner (20, 10)—its x- and y-coordinates
- ▶Specify its width (40)

34 **Draw a Square (Page 2)**

▶The square function in Processing is used to draw a square inside the output window

▶Format:

square(*x-UpperLeft*, *y-UpperLeft*, *width*);

35 **Draw a Square (Page 3)**

▶Example:

```
square(20, 10, 40);
```

- ▶First two arguments (20, 10) are x- and y- coordinates of the upper-left corner
- ▶Last argument (40) is the width

36 **Draw a Square (Page 4)**

▶The rectMode function also may be applied to squares prior to calling the square function but only to change the origin to centerpoint:

rectMode(CORNER); — square function arguments specify upper-left corner and width of the square (the *default*)

rectMode(CENTER); — square function arguments specify the centerpoint and width of the square

~~rectMode(CORNERS);~~ *does not apply*

37 **Draw an Ellipse (Page 1)**

- To draw an ellipse specify:
  - The x- and y-coordinates of its centerpoint
  - Its width and height
- For example:
  - Specify the centerpoint (65, 50)—its x- and y-coordinates
  - Specify its width and height (50, 60)

39 **Draw an Ellipse (Page 2)**

- The ellipse function in Processing is used to draw an ellipse inside the output window
- Format:

  ellipse(*x-CoordCenter*, *y-CoordCenter*, *width*, *height*);

40 **Draw an Ellipse (Page 3)**

- Example:

  ellipse(65, 50, 50, 60);
  - First two arguments (65, 50) are x- and y-coordinates of the centerpoint
  - Last two arguments (50, 60) are the width and height

41 **The ellipseMode Function (Page 1)**

- There are two other formats for drawing ellipses:
  - The x- and y- coordinates specify the upper-left corner of the bounding box in which the ellipse is drawn

    ellipse(*x-UpperLeft*, *y-UpperLeft*, *width*, *height*);
  - The two sets of x- and y-coordinates specify the upper-left and lower-right corners of the bounding box in which the ellipse is drawn

    ellipse(*x-UpperLeft*, *y-UpperLeft*, *x-LowerRight*, *y-LowerRight*);

42 **The ellipseMode Function (Page 2)**

- Select the format by assigning one of three values using the ellipseMode function prior to calling the ellipse function:

  ellipseMode(CENTER); — ellipse function arguments specify the centerpoint, width, and height of the ellipse (the *default*)

  ellipseMode(CORNER); — ellipse function arguments specify upper-left corner , width, and height of the ellipse

  ellipseMode(CORNERS); — ellipse function arguments specify upper-left and lower-right corners of the ellipse

44 **The ellipseMode Function (Page 3)**

- To specify the upper-left corner, width, and height:

  ellipseMode(CORNER);

ellipse(40, 20, 50, 60);

46 **The ellipseMode Function (Page 4)**

▶To specify the upper-left and lower-right corners:
ellipseMode(CORNERS);
ellipse(40, 20, 90, 80);

47 **Draw a Circle (Page 1)**

▶To draw a circle specify:
▶The x- and y-coordinates of its centerpoint
▶Its diameter (the width and height are the same but not specified)
▶For example:
▶Specify the centerpoint (60, 50)—its x- and y-coordinates
▶Specify its diameter (50)

49 **Draw a Circle (Page 2)**

▶The circle function in Processing is used to draw a circle inside the output window
▶Format:
circle(*x-CoordCenter*, *y-CoordCenter*, *diameter*);

50 **Draw a Circle (Page 3)**

▶Example:
circle(60, 50, 50);
▶First two arguments (60, 50) are x- and y-coordinates of the centerpoint
▶Last argument (50) is the diameter

51 **Draw a Circle (Page 4)**

▶The ellipseMode function also may be applied to circles prior to calling the circle function but only to change the origin to upper-left corner:
ellipseMode(CENTER); — circle function arguments specify centerpoint and diameter of the circle (the *default*)
ellipseMode(CORNER); — circle function arguments specify upper-left corner and diameter of the circle
~~ellipseMode(CORNERS);~~ *does not apply*

55 **Greyscale Color**

▶Greyscale is a shade of gray ranging from black (all) to white (none)
▶Measured as a numeric value from zero (0) which is black to 255 which is white:
▶Black (0) = 0% brightness (aka 100% greyscale)
▶White (255) = 100% brightness (aka 0% greyscale)
▶All other values above zero and below 255 are some shade of gray

56 **Bits and Bytes (Page 1)**

▶A bit (literally meaning a binary digit) is the basic unit of information and storage in

computing (memory and storage)

- In computer memory, a bit is a "switch" that can be either:
  - "On"—represented by the digit 1 in binary
  - "Off"—represented by the digit 0 in binary

57 **Bits and Bytes (Page 2)**

- Since bits only are able to represent two pieces of information, they usually are *grouped* into larger organizational units
- The smallest "meaningful" unit of computer memory is a byte
  - A byte consists of eight bits
  - Used in binary to represent all numerical values from zero (0) to 255

58 **Bits and Bytes (Page 3)**

- The numbers between zero (0) and 255 represented as eight-digit binary numbers:
  - 0—00000000
  - 1—00000001
  - 2—00000010
  - 3—00000011

  ...
  - 253—11111101
  - 254—11111110
  - 255—11111111

59 **The background Function (Page 1)**

- Sets the background grayscale (or color) of the output window where the shapes are drawn
- Format:

  background(*int*);
  - The *int* argument is an integer in the range 0–255 that represents black, or white, or a shade of gray

60 **The background Function (Page 2)**

- Example:

  background(255);
  - Background will be white (255)

61 **The fill Function (Page 1**

- Sets the interior fill grayscale (or color) of a shape such as a rectangle or ellipse
  - The default fill grayscale is white
- Format:

  fill(*int*);
  - The *int* argument is an integer in the range 0–255 that represents black, or white, or a shade of gray

62 **The fill Function (Page 2)**

▶Example:

fill(127);

▶Interior fill will be gray (127)

63 **The stroke Function (Page 1)**

▶The stroke function sets the outer border (outline) grayscale (or color) of a shape such as a rectangle or ellipse

▶Lines and points only have a stoke—no fill

▶The default stoke color is black

▶Format:

stroke(*int*);

▶The *int* argument is an integer in the range 0–255 that represents black, or white, or a shade of gray

64 **The stroke Function (Page 2)**

▶Example:

stroke(200);

▶Stroke (outer border, line, or point) will be light gray (200)

65 **The noFill Function**

▶The noFill function draws a shape (e.g. rectangle, ellipse, etc.) with an outline border, but *no filled interior*

▶Format:

noFill();

66 **The noStroke Function**

▶The noStroke function draws a shape (e.g. rectangle, ellipse, etc.) with a filled interior, but *no outline border*

▶Format:

noStroke();

69 **RGB Color (Page 1)**

▶RGB (Red—Green—Blue) color describes "colored light" viewed coming from its source, e.g. the colors of a video display (monitor)

▶Called an additive color system, because light is *added* from the primary colors (red, green and blue) to make new colors

70 **RGB Color (Page 2)**

▶Values for red, green, and blue are usually specified using a scale from zero (0) to 255

▶Use three bytes, one byte each (per pixel) for the red, green and blue values, in the range 0–255

▶Higher numbers mean more of each color of light; lower numbers mean less of

the color of light

- ▶This method provides for over 16 million colors

71 **RGB Color (Page 3)**

- ▶Examples:
  - ▶Secondary colors:
    - ▶Yellow = Red(255) + Green(255) + Blue(0)
    - ▶Cyan = Red(0) + Green(255) + Blue(255)
    - ▶Magenta = Red(255) + Green(0) + Blue(255)
  - ▶All (or none) colors:
    - ▶White = Red(255) + Green(255) + Blue(255)
    - ▶Black = Red(0) + Green(0) + Blue(0)

72 **RGB Color (Page 4)**

- ▶You can experiment with RGB by selecting the "Color Selector" from the Processing "Tools" menu
- ▶Also try websites with RGB color selector tools

73 **The background Function (Revisited—Page 1)**

- ▶An alternative version of the background function sets the background *color* of the window
- ▶Format:

  background(*redInt*, *greenInt*, *blueInt*);
  - ▶The arguments are integers in the range 0–255 that represent amount of red, green and blue in the color

74 **The background Function (Revisited—Page 2)**

- ▶Example:

  background(255, 255, 0);
  - ▶Background color will be yellow (255, 255, 0)

75 **The fill Function (Revisited—Page 1)**

- ▶An alternative version of the fill function sets the interior fill *color* of a shape
- ▶Format:

  fill(*redInt*, *greenInt*, *blueInt*);
  - ▶The arguments are integers in the range 0–255 that represent amount of red, green and blue in the color

76 **The fill Function (Revisited—Page 2)**

- ▶Example:

  fill(0, 255, 255);

  circle(50, 50, 100);
  - ▶Fill color will be cyan (0, 255, 255)

77 **The stroke Function (Revisited—Page 1)**

- ▶An alternative version of the stroke function sets the outline border *color* of a shape, or the *color* of a line or point
- ▶Format:

  stroke(*redInt*, *greenInt*, *blueInt*);
  - ▶The arguments are integers in the range 0–255 that represent amount of red, green and blue in the color

78 **The stroke Function (Revisited—Page 2)**

- ▶Example:

  stroke(255, 0, 255);

  circle(50, 50, 100);
  - ▶Stroke color will be magenta (255, 0, 255)

82 **Color Transparency**

- ▶The term color transparency used at its simplest means there is "full transparency", e.g. completely invisible
- ▶More complex is "partial transparency", where an illusion is achieved that the graphic is partially transparent in the same way as colored glass

83 **The fill Function (Revisited Again—Page 1)**

- ▶An alternative version of the fill function sets the interior fill *color transparency* of a shape
- ▶Format:

  fill(*redInt*, *greenInt*, *blueInt*, *transparencyInt*);
  - ▶Fourth argument is an integer that represents amount of opacity
  - ▶Opacity values range:
    - ▶From zero (0), completely transparent (0% opaque)
    - ▶To 255, no transparency (100% opaque)

84 **The fill Function (Revisited Again—Page 2)**

- ▶Example:

  fill(0, 255, 255, 127);
  - ▶Fill will be cyan with 50% opacity (0, 255, 255, 127)

86 **The colorMode Function (Page 1)**

- ▶The colorMode sets:
  - ▶*Color mode* for the way color data is interpreted by Processing, either RGB which is the default or HSB (Hue/Saturation/Brightness)
  - ▶*Customized ranges* rather than the default zero (0) to 255

87 **The colorMode Function (Page 2)**

- ▶Formats:

  colorMode(RGB/HSB, *rangeForAll*);
  - ▶Color mode is either RGB or HSB

▶The *rangeForAll* specifies the high end for all elements as the same
colorMode(RGB/HSB, *redRange*, *greenRange*, *blueRange*, *transparencyRange*);
▶Sets high end for each element separately

88 **The colorMode Function (Page 3)**

▶Examples:
colorMode(RGB, 100);
▶Uses *RGB mode* with ranges from zero (0) to 100 (percentages)
colorMode(RGB, 255, 255, 255, 100);
▶Uses *RGB mode* with RGB ranges as default, but sets the transparency from zero (0) to 100 (as percentages)

91 **The colorMode Function (Page 3)**

▶Examples (*con*.):
colorMode(HSB, 100);
▶Uses *HSB mode* (Hue/Saturation/Brightness) ranges from zero (0) to 99 (percentages)
▶Hue by default is measured in degrees (0-359)
▶Describing colors using hue, saturation and brightness (light) (HSL) is a convenient way to organize differences in colors as perceived by people; human perception sees colors in these ways and not as triplets of numbers