IAT267 Introduction to Technological Systems

Lecture 8

The Processing language.
Arduino and Processing.

Course Project

- All teams submitted very interesting proposals
- One requirement for the project is to have communication with the computer. A Processing application running on the computer is required.
- Think of ways in which you can have a physical installation which communicates with a Processing application running on your computer.

Milestone 3 – Course Project

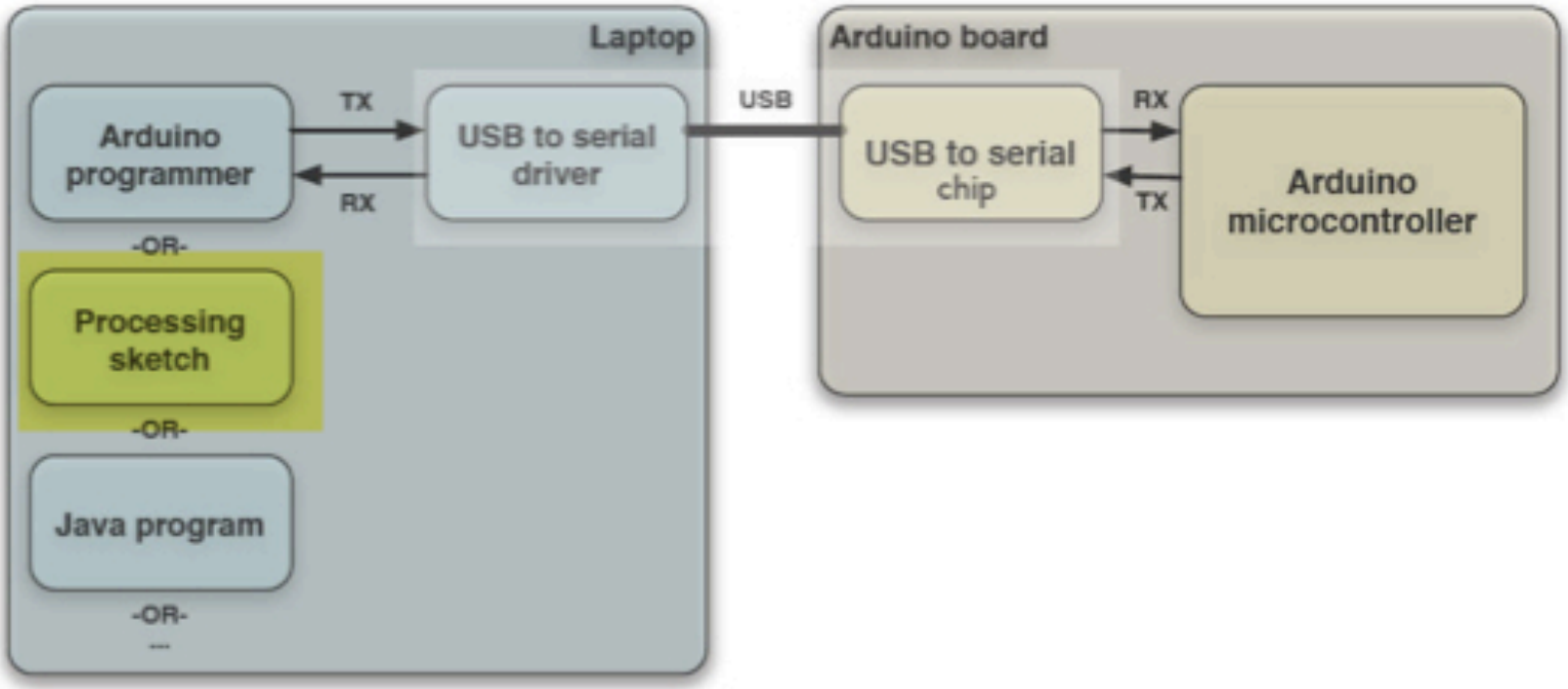
- Posted on webct.

Workshops

- Save your codes from all workshops
- Older code will be re-used in future workshops
- On the exam there will be questions from both lectures and workshops

Processing language (review)

Arduino to Computer

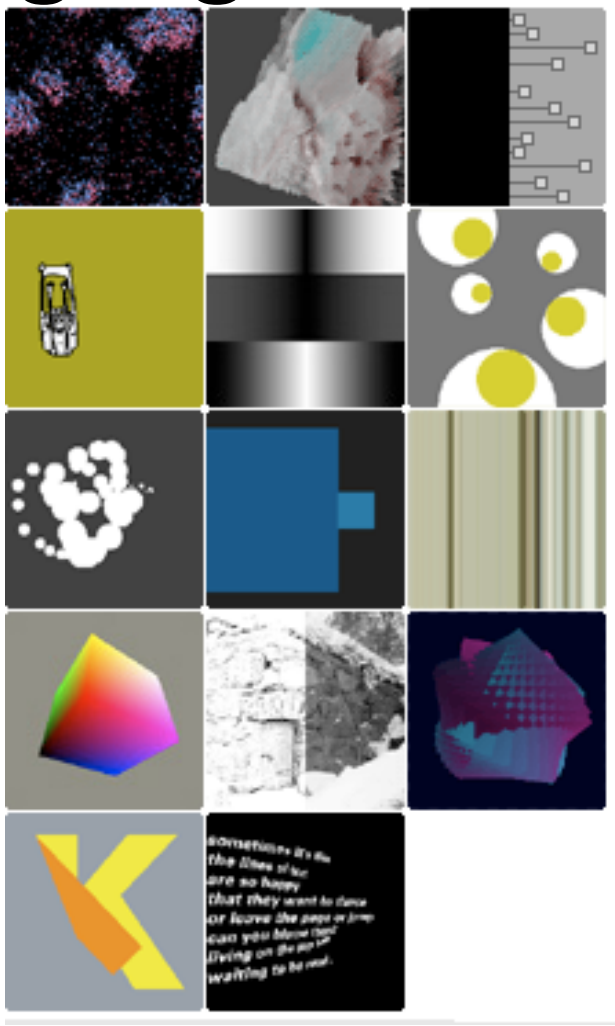


USB is totally optional for Arduino
But it makes things easier

The Processing language

- Processing makes Java programming as fun & easy as Arduino makes microcontroller programming

- Is often used to interface to devices like Arduino
- Most of you have programmed in Processing in IAT265



What is Processing?

- Processing is 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**.

- Processing is open source like Arduino.
- Processing GUI and Arduino GUI are from the same code, which is why they look & act similar.

Processing elements

- The Processing Development Environment

- A collection of functions (also referred to as commands or methods) that make up the “core” programming interface, or API, as well as several libraries that support more advanced features such as drawing with OpenGL, reading XML files, and saving complex imagery in PDF format.

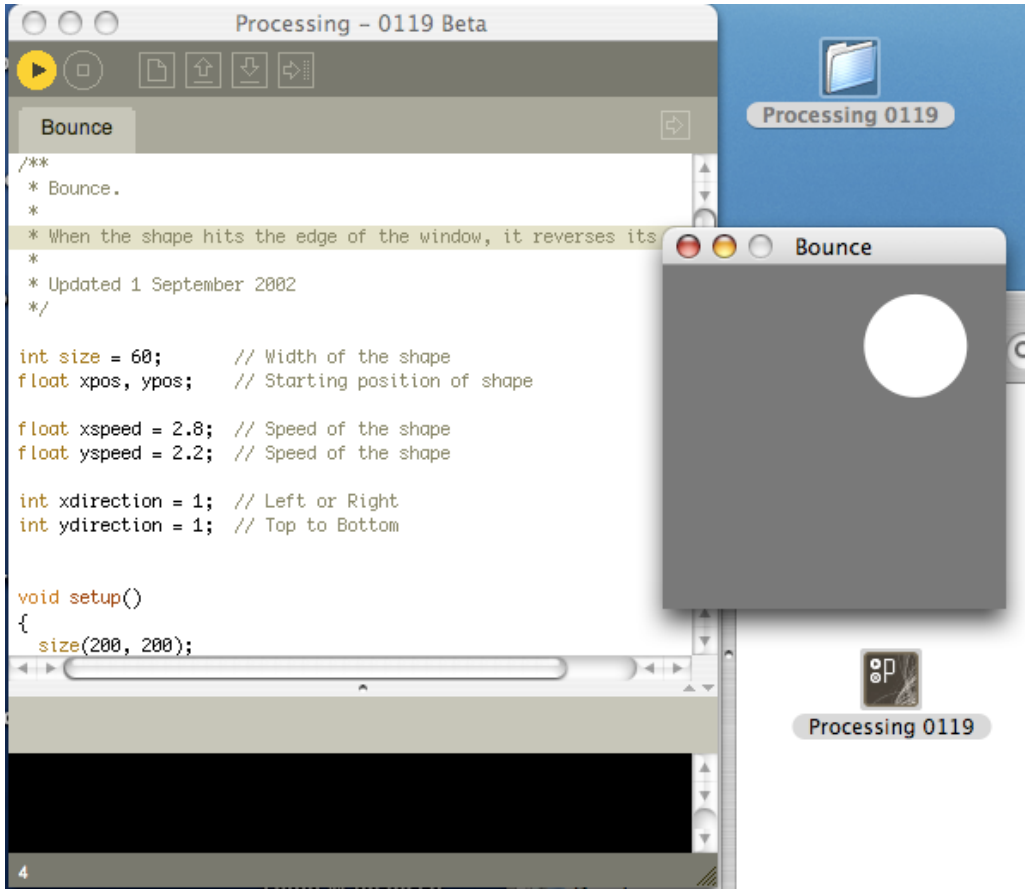
Processing elements

- A language syntax, identical to Java but with a few modifications.

- An active online community, hosted at *http://processing.org.*

Using Processing

- First, install Processing
- Load up “Sketchbook » Examples » Motion » Bounce”
- Press “Run” button
- You just made a Java applet



- Also try Examples » Motion » Collision.

- Notice how “Run” launches a new window containing the sketch.

- The black area at the bottom is a status window, just like in Arduino.

About Processing

- Processing sketches have very similar structure to Arduino sketches
  - setup() – set up sketch, like size, framerate
  - draw() – like loop(), called repeatedly
  - Other functions can exist when using libraries

Processing: Hello World

- The Processing equivalent of a "Hello World" program is simply to draw a line:

  line(15, 25, 70, 90);

- Enter this example and press the Run button, which is an icon that looks like the Play button from any audio or video device.

- Your code will appear in a new window, with a gray background and a black line from coordinate (15, 25) to (70, 90). The (0, 0) coordinate is the upper left-hand corner of the display window.

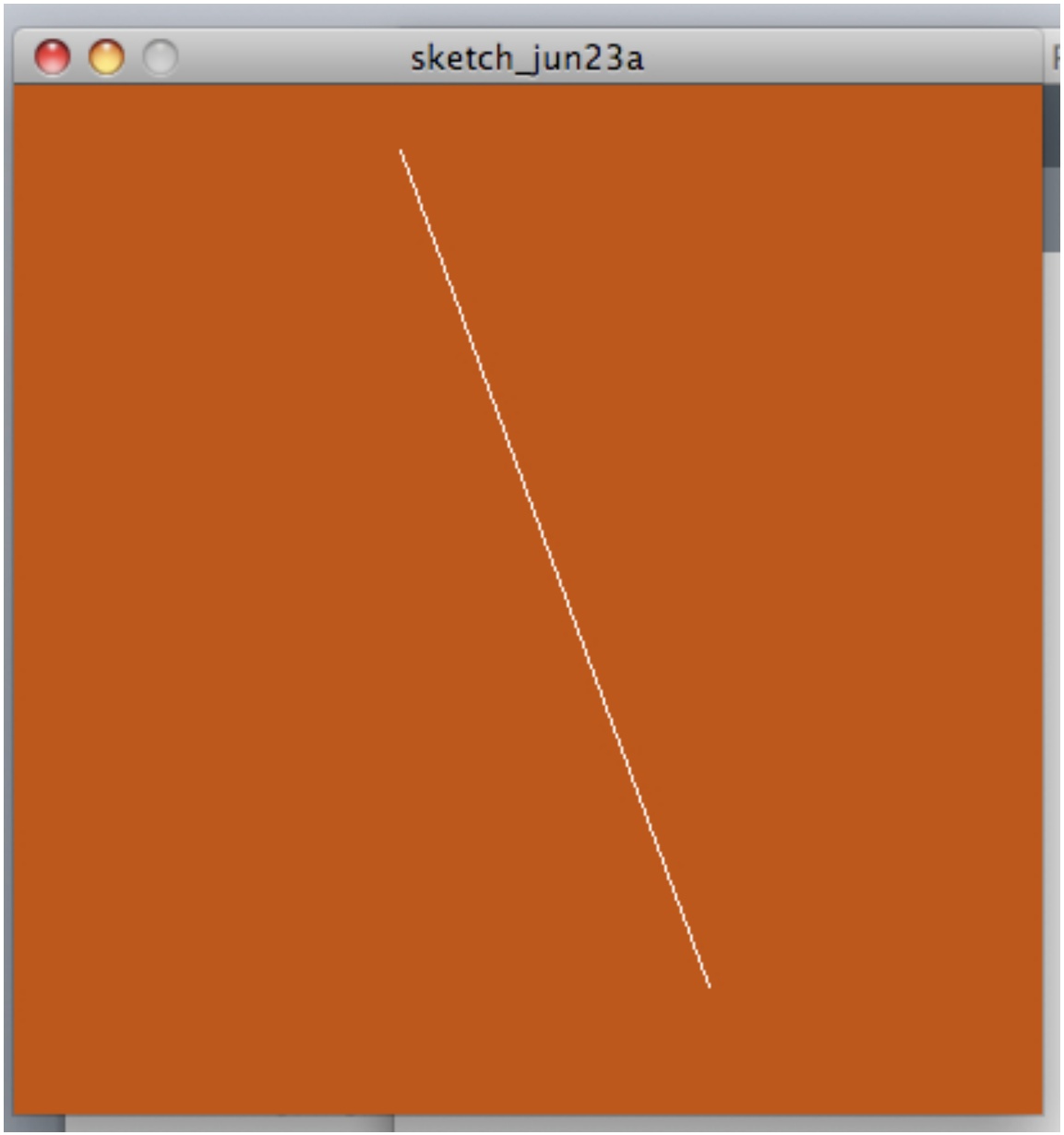
- Building on this program to change the size of the display window and set the background color, type in the code below:

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

- This version sets the window size to 400 x 400 pixels, sets the background to an orange-red, and draws the line in white, by setting the stroke color to 255.

- By default, colors are specified in the range 0 to 255.

- Other variations of the parameters to the stroke() function provide alternate results.


sketch_jun23a

stroke variations

- *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(255, 128, 0, 128); // bright orange with 50% transparency*

drawing functions

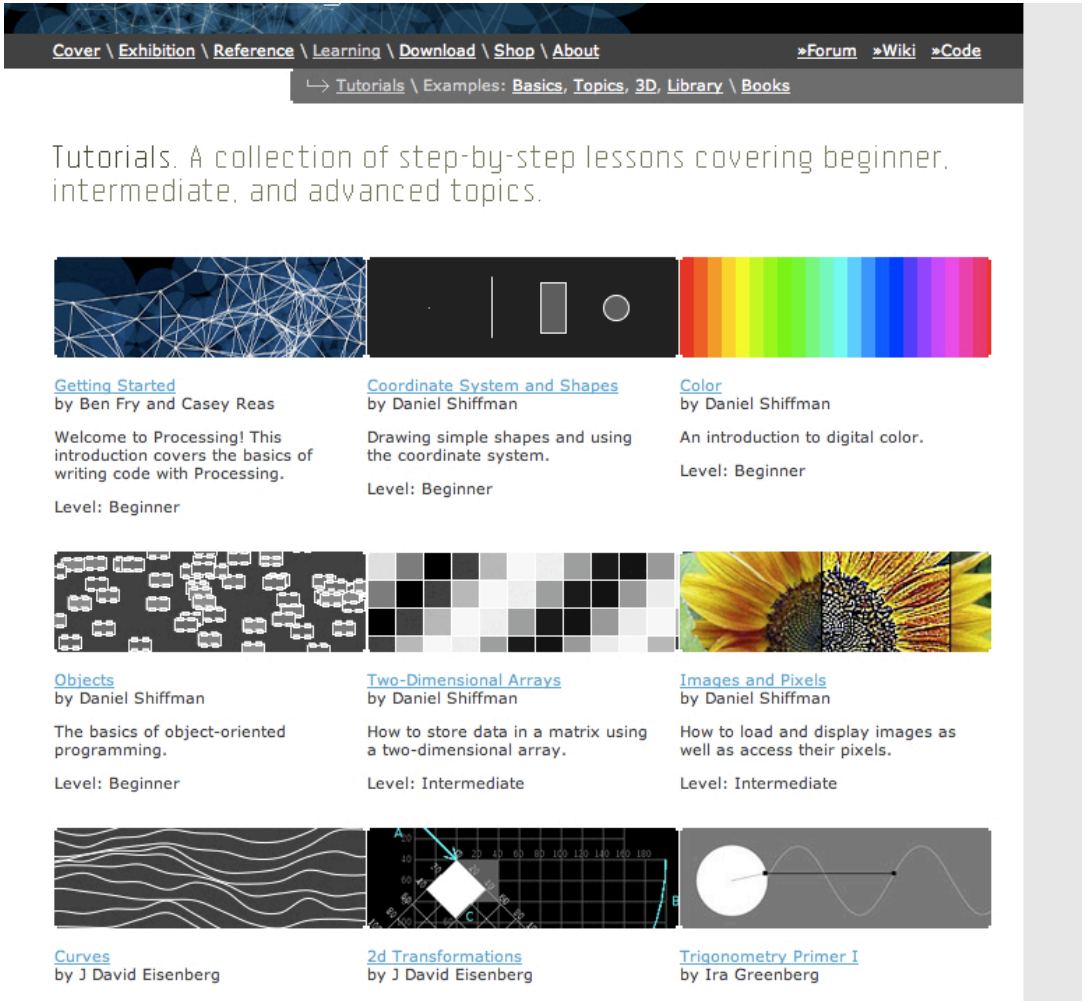
- The same alternatives work for the *fill()* function, which sets the fill color, and the *background()* function, which clears the display window.

- 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.

Learning Processing

- Excellent resource:

Processing website:
http://processing.org/learning/



Processing and Arduino: serial communication

- Processing and Arduino both talk to “serial” devices
- **Only one program per serial port at a time**
- !!!So turn off Arduino’s Serial Monitor when connecting via Processing and vice-versa!!!

Processing: serial library

- Processing has a “Serial” library to talk to Arduino. For example:
  - serial communications
    - port = new Serial(..,“my_port_name”,9600)
    - port.read(), port.write(), etc.
    - serialEvent() { }

Processing Serial Library

- The Processing serial library allows for easily reading and writing data to and from external machines.

- It allows two computers to send and receive data and gives you the flexibility to communicate with custom microcontroller devices, using them as the input or output to Processing programs.

Serial class

Serial
available()
read()
readChar()
readBytes()
readBytesUntil()
readString()
readStringUntil()
buffer()
bufferUntil()
last()
lastChar()
write()
clear()
stop()
list()

Serial events

serialEvent()

Serial class

- Class for sending and receiving data using the serial communication protocol.

- http://processing.org/reference/libraries/serial/Serial.html

- Analyze the methods of the serial library

Common Processing Serial Use

- Four steps:
  - load library
  - set portname
  - open port
  - read/write port

Explained Code Examples

1. Sending data from Arduino to computer through serial comm.

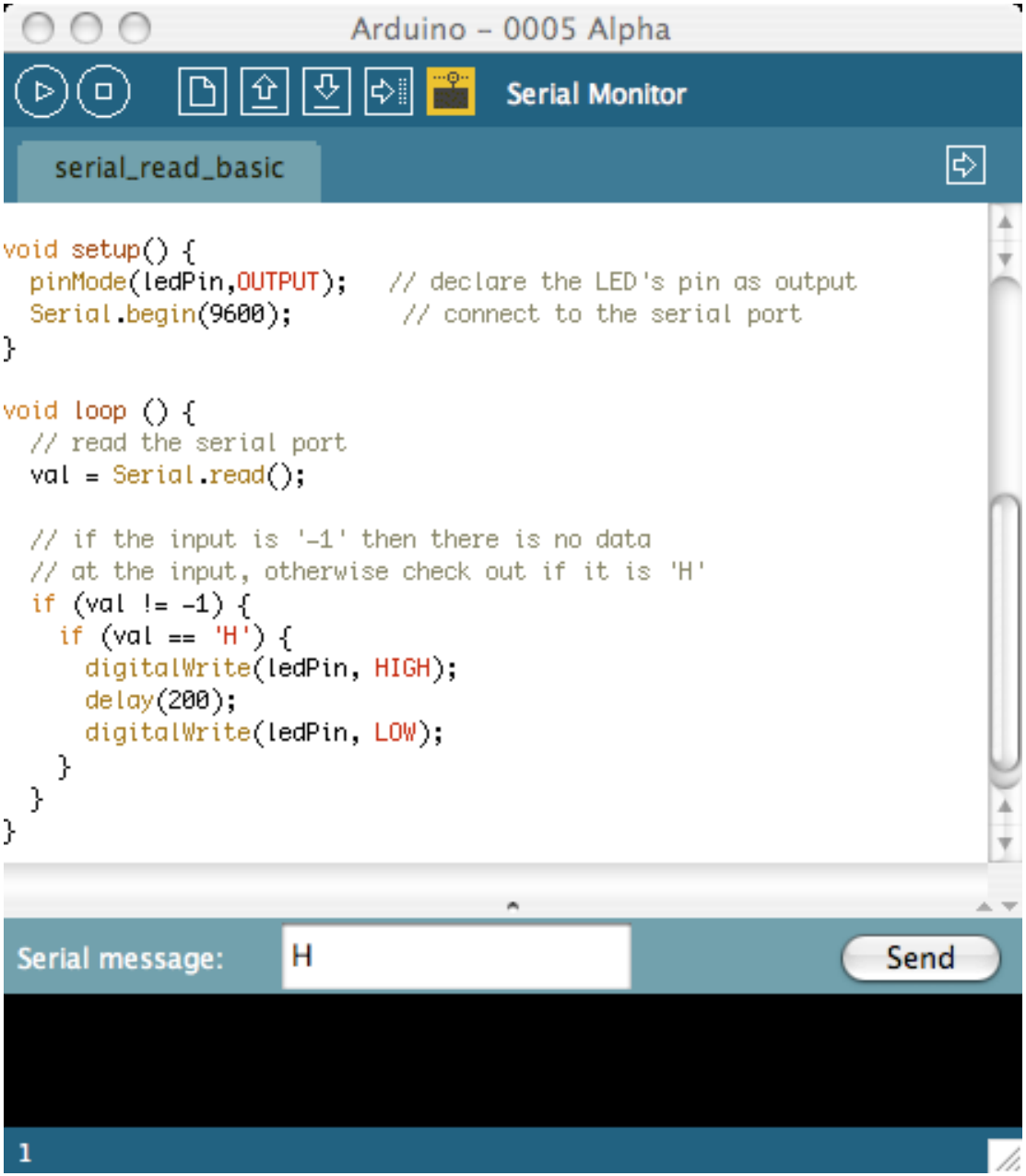
```
/*
 * AnalogInput
 * by DojoDave <http://www.0j0.org>
 *
 * Turns on and off a light emitting diode(LED) connected to digital
 * pin 13. The amount of time the LED will be on and off depends on
 * the value obtained by analogRead(). In the easiest case we connect
 * a potentiometer to analog pin 2.
 *
 * http://www.arduino.cc/en/Tutorial/AnalogInput
 */

int potPin = 0;    // select the input pin for the potentiometer
int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the value coming from the sensor
int interval = 500;

void setup() {
  pinMode(ledPin, OUTPUT);  // declare the ledPin as an OUTPUT
  Serial.begin(9600);
}

void loop() {
  val = analogRead(potPin);    // read the value from the sensor
  //interval = (1024-val);  //reversing the functionality : dark: slow blinking, large values, more delay
  interval=val;     //dark: fast blink, small values, small delay
  Serial.println(interval);

  digitalWrite(ledPin, HIGH);  // turn the ledPin on
  delay(interval);                  // stop the program for some time
  digitalWrite(ledPin, LOW);   // turn the ledPin off
  delay(interval);                  // stop the program for some time
}
```



2. User typing data at the serial port (Serial Monitor) of Arduino software

- If the user types ‘H’ the LED blinks
- Arduino microcontroller acts as a function of received data from the serial port
- Notice the use of Serial.read()

3. Serial.print()

- Can send sensor data from Arduino to computer with Serial.print()
- There are many different variations to suit your needs:

```
int val = 123;
Serial.print(val);      // sends 3 ASCII chars "123"
Serial.print(val,DEC);  // same as above
Serial.print(val,HEX);  // sends 2 ASCII chars "7B"
Serial.print(val,BIN);  // sends 8 ASCII chars "01111011"
Serial.print(val,BYTE); // sends 1 byte, the verbatim value
```

4. Controlling the computer from Arduino

- You write one program on Arduino, one on the computer
- Arduino code:

```
void loop() {
  val = analogRead(analogInput);   // read the value on analog input
  Serial.print(val/4,BYTE);        // print a byte value out
  delay(50);                       // wait a bit to not overload the port
}
```

- Processing code:

```
import processing.serial.*;

Serial myPort;  // The serial port

void setup() {
  String portname = "/dev/tty.usbserial-A3000Xv0";
  myPort = new Serial(this, myPort, 9600);
}

void draw() {
  while (myPort.available() > 0) {
    int inByte = myPort.read();
    println(inByte);
  }
}
```

- The Arduino code reads the data from a sensor and sends it as BYTE to the serial port
- The Processing code reads this data and then does something with it (not shown).

Serial.read()

- For example, if the user types a number to the serial port:
  - To make use of the number value, it needs to be converted from ASCII character to number
  - **val=val - '0'** : this does the conversion to number
  - See ASCII table on the next slide

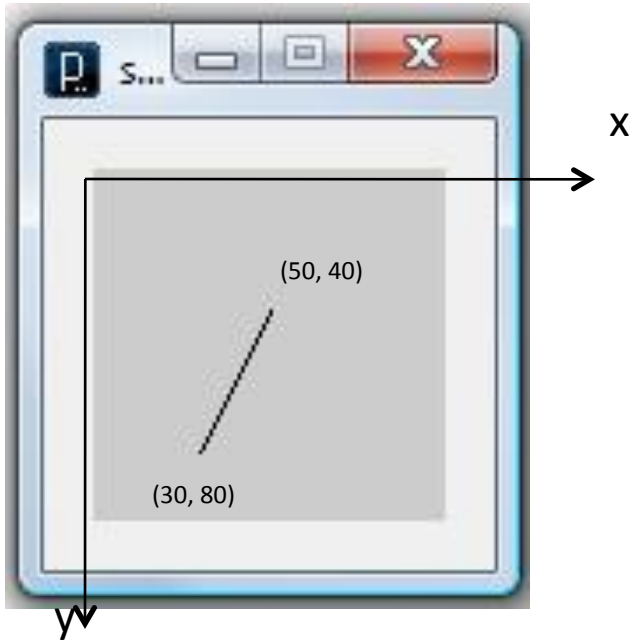
ASCII Chart

Table 7.1
ASCII Chart

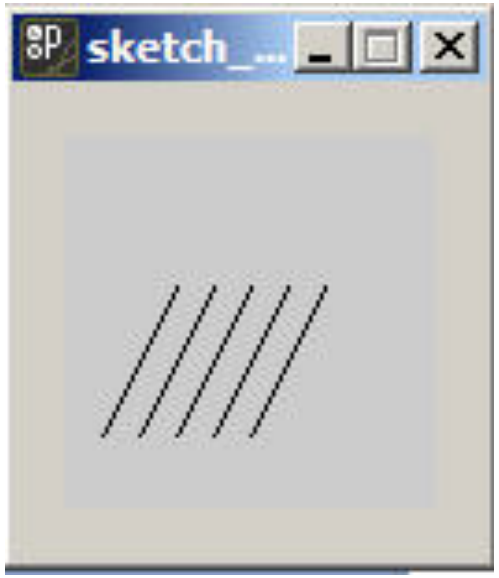
0	NUL	**16**	DLE	**32**	SP	**48**	0	**64**	@	**80**	P	**96**	`	**112**	P
1	SOH	**17**	DC1	**33**	!	**49**	1	**65**	A	**81**	Q	**97**	a	**113**	Q
2	STX	**18**	DC2	**34**	“	**50**	2	**66**	B	**82**	R	**98**	b	**114**	R
3	ETX	**19**	DC3	**35**	#	**51**	3	**67**	C	**83**	S	**99**	c	**115**	S
4	EOT	**20**	DC4	**36**	$	**52**	4	**68**	D	**84**	T	**100**	d	**116**	T
5	ENQ	**21**	NAK	**37**	%	**53**	5	**69**	E	**85**	U	**101**	e	**117**	U
6	ACK	**22**	SYN	**38**	&	**54**	6	**70**	F	**86**	V	**102**	f	**118**	V
7	BEL	**23**	ETB	**39**	‘	**55**	7	**71**	G	**87**	W	**103**	g	**119**	W
8	BS	**24**	CAN	**40**	(	**56**	8	**72**	H	**88**	X	**104**	h	**120**	X
9	HT	**25**	EM	**41**	)	**57**	9	**73**	I	**89**	Y	**105**	i	**121**	Y
10	LF	**26**	SUB	**42**	*	**58**	:	**74**	J	**90**	Z	**106**	j	**122**	Z
11	VT	**27**	ESC	**43**	+	**59**	;	**75**	K	**91**	[	**107**	k	**123**	{
12	FF	**28**	FS	**44**	,	**60**	<	**76**	L	**92**	\	**108**	l	**124**	|
13	CR	**29**	GS	**45**	-	**61**	=	**77**	M	**93**	]	**109**	m	**125**	}
14	SO	**30**	RS	**46**	.	**62**	>	**78**	N	**94**	^	**110**	n	**126**	~
15	SI	**31**	US	**47**	/	**63**	?	**79**	O	**95**	_	**111**	o	**127**	DEL

Position of a line:

- 4 numbers: coordinates of the endpoints
- Origin of the coordinate system: upper-left corner, numbers increase right and down



```
line (10, 80, 30, 40);  //Left line
line (20, 80, 40, 40);
line(30, 80, 50, 40);  //Middle line
line(40, 80, 60, 40);
line (50, 80, 70, 40);  //Right line
```



Visual Attributes of Shapes

- Controlled with code elements that set:
  - Color or gray values
  - Width of lines
  - Quality of rendering

```
background(0); //set the black background
stroke (255); //set line value to white
strokeWeight(5); //set line width to 5 pixels
smooth (); //smooth line edges
line (10, 80, 30, 40);  //Left line
line (20, 80, 40, 40);
line(30, 80, 50, 40);  //Middle line
line(40, 80, 60, 40);
line (50, 80, 70, 40);  //Right line
```

Adding more structure

- To create animation and interactive programs it is required for the program to run continuously
  - setup(): code inside setup() runs once when the program first starts
  - draw(): code inside draw() runs continuously: one image frame is drawn to the display window at the end of each loop

```
int x=0; //set the horizontal position
int y=55; //set the vertical position

void setup()
{
  size(100, 100); //set the window to 100x100 pixels
}

void draw()
{

  background(204);
  line(x,y, x+20, y-40); //left line
  line(x+10, y, x+30, y-40); //middle line
  line(x+20, y, x+40, y-40); //right line

  x=x+1;
  if (x>100)
  {
    x=-40;
  }
}
```

The result: lines moving horizontally

Data from Keyboard and Mouse

- When a program is running continuously: Processing stores data from the input devices:
  - Mouse
  - Keyboard
- This data can be used to affect what is happening in the display window
- See an example on the next slide.

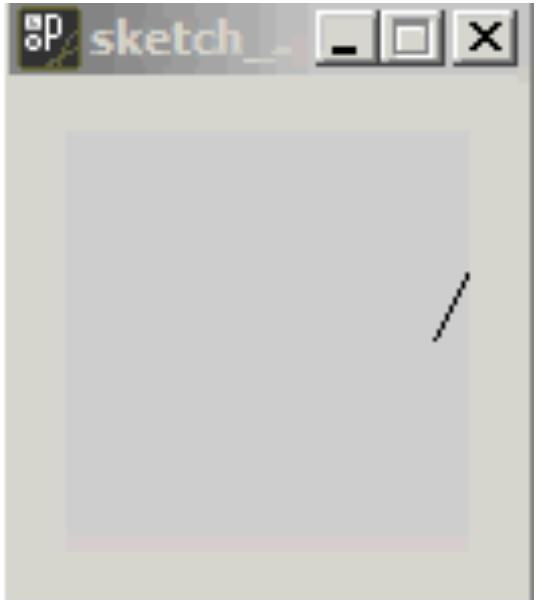
```
void setup()
{
  size(100, 100); //set the window to 100x100 pixels
}

void draw()
{

 background(204);
 //assign the horizontal value of the cursor to x
 float x=mouseX;
 //assign the vertical value of the cursor to y
 float y=mouseY;

 line(x, y, x+20, y-40);
 line(x+10, y, x+30, y-40);
 line(x+20, y, x+40, y-40);
}
```

The result: lines move with the mouse



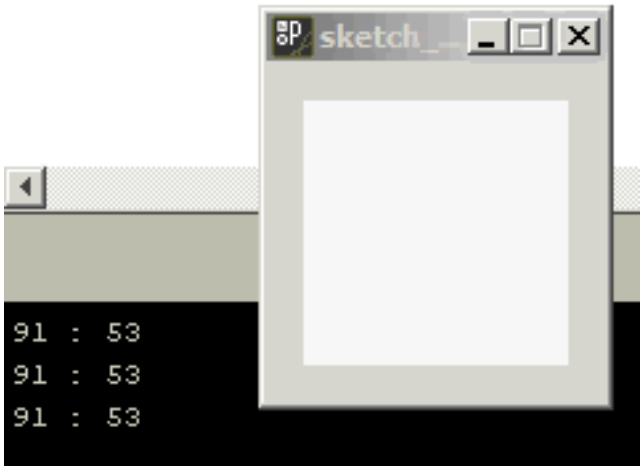
Mouse Data

- The Processing variables ***mouseX*** and ***mouseY*** store the x-coordinate and y-coordinate of the cursor relative to the origin

- Code to see the actual values produced while moving the mouse – on next slide

```
void setup()
{
      size(100, 100); //set the window to 100x100 pixels
       frameRate(12);
}

void draw()
{
    background(245);
    println(mouseX + " : " + mouseY);
}
```



Functions in Processing

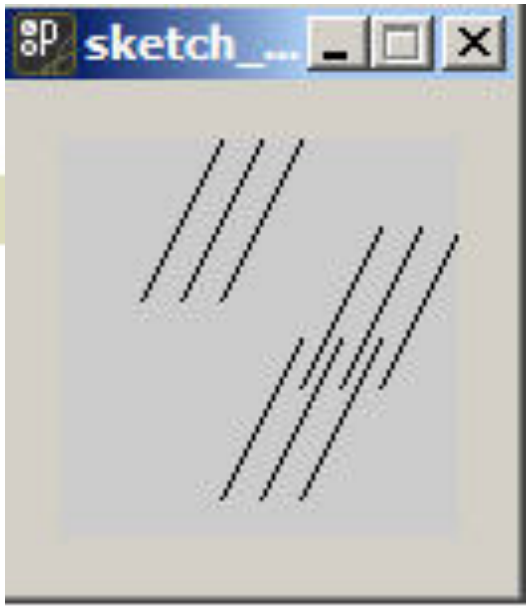
- Function = a set of code that performs a specific task
  - All built-in functions of Processing can be found at:
    http://processing.org/reference/
- Example of function that we implemented:

```
void diagonals (int x, int y)
{
   line(x, y, x+20, y-40);
   line(x+10, y, x+30, y-40);
   line(x+20, y, x+40, y-40);
}
```

```
void setup()
{
  size(100, 100); //set the window to 100x100 pixels
}

void draw()
{
  diagonals(40,90);
  diagonals(60,62);
  diagonals(20,40);
}

void diagonals (int x, int y)
{
        line(x, y, x+20, y-40);
        line(x+10, y, x+30, y-40);
        line(x+20, y, x+40, y-40);
}
```



Array & Loop

- So far, in all examples, each variable stored one data element

- **Array**: can store a list of elements with a single name
  - A ***for*** loop can be used to cycle through each array element in sequence

```
int num=20;
int[] dx = new int[num]; //declare and create an array
int[] dy = new int[num]; //declare and create an array

void setup()
{
  size(100, 100); //set the window to 100x100 pixels
  for (int i=0; i<num; i++)
  {
    dx[i]=i*5;
    dy[i]=12+(i*6);
  }
}

void draw()
{
  background(204);
  for (int i=0; i<num; i++)
  {
    dx[i]=dx[i]+1;
    if (dx[i]>100)
    {
      dx[i]=-100;
    }
   diagonals(dx[i], dy[i]);
  }
}

 void diagonals (int x, int y)
{
        line(x, y, x+20, y-40);
        line(x+10, y, x+30, y-40);
        line(x+20, y, x+40, y-40);
}
```

The result: 20 groups of animated diagonals (3 in each group)

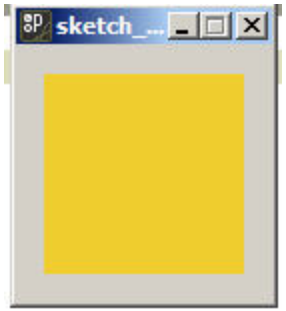
Color in Processing

- The most common way to specify color on the computer is with RGB or RGBA values
  - All the R, G, B, A values go from 0 to 255
- In Processing, colors are defined by the parameters to the background (), fill() and stroke() functions:
  - background(value1, value2, value3)
  - fill(value1, value2, value3)
  - fill(value1, value2, value3, alpha)
  - stroke(value1, value2, value3)
  - stroke(value1, value2, value3, alpha)

color

- value1: red component
- value2: green component
- value3: blue component
- The optional alpha parameter to fill() or stroke () defines the transparency
  - alpha=255(opaque)
  - alpha=0 (entirely transparent: won't be visible)

background(242,204,47);



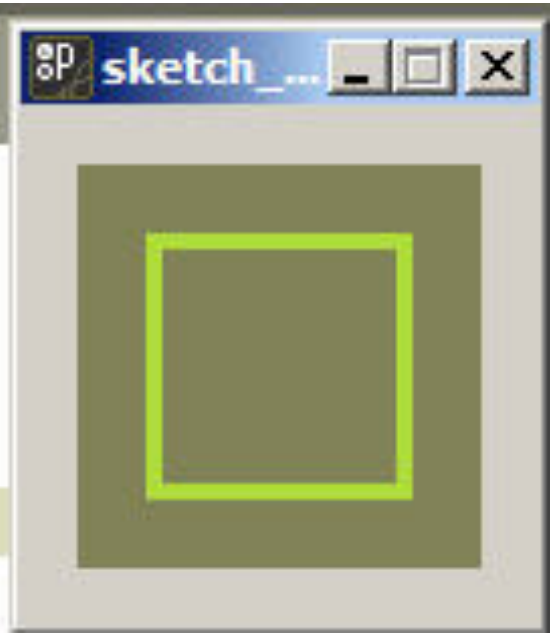
background (174,221,60);

Stroke and Fill

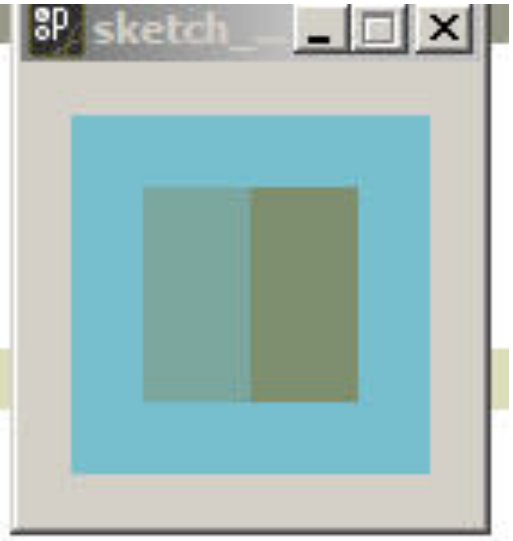
- noStroke(): Disables drawing the stroke (outline)
- noFill(): Disables filling the shape
- *If both **noStroke()** and **noFill()** are called, nothing will be drawn to the screen*
  - Could be useful when you want a shape to disappear

sketch_080303a §

```
background(129,130,87);
noFill();
strokeWeight(4);
stroke(174,221,60);
rect(19,19,62,62);
```




```
background(116,193,206);
noStroke();
fill(129,130,87,102); //more transparent
rect(20,20,30,60);
fill(129,130,87,204); //less transparent
rect(50,20,30,60);
```



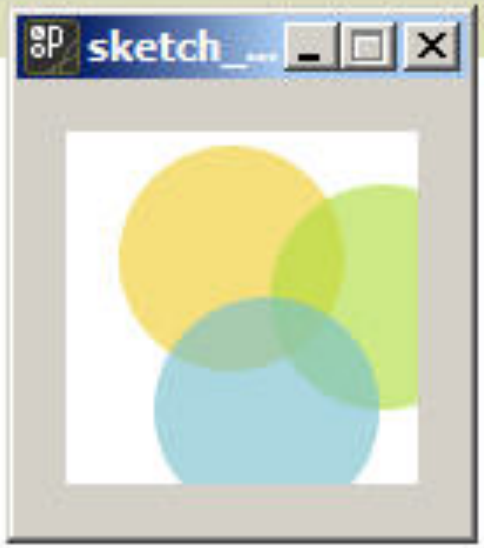
Transparency

- Transparency can be used to create new colors by overlapping shapes.

- The colors originating from overlaps depend on the order in which the shapes are drawn.

```
background(0);
noStroke();
smooth();
fill(242, 204, 47, 160); //yellow
ellipse(47,36,64,64);
fill(174,221,60,160); //green
ellipse(90,47,64,64);
fill(116,193,206,160); //blue
ellipse(57,79,64,64);
```


```
background(255);
noStroke();
smooth();
fill(242, 204, 47, 160); //yellow
ellipse(47,36,64,64);
fill(174,221,60,160); //green
ellipse(90,47,64,64);
fill(116,193,206,160); //blue
ellipse(57,79,64,64);
```



Drawing Shapes

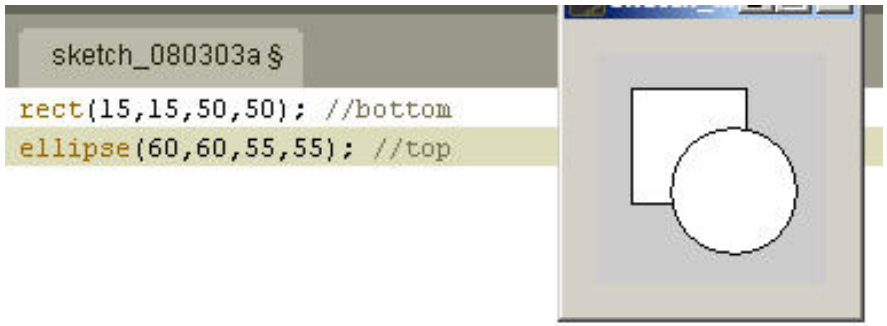
- A point is the simplest visual element and is drawn with the point() function:
  - point(x,y)
- Lines: (we have this in our first example of today):
  - line(x1,y1,x2,y2);
- Triangle: function to draw triangles
  - triangle(x1,y1,x2,y2,x3,y3);

Rectangles and Ellipses

- Drawing rectangles and ellipses works differently than for the previous shapes: the four parameters set the position and size of the shape:
  - rect(x, y, width, height);
    - x, y: specify the location of the upper-left corner, the (width, height) is about the size of the rectangle
  - ellipse(x, y, width, height);
    - The first two parameters set the location of the center of the ellipse, the (width, height) is about the size of its bounding box

Drawing Order

- The order in which the shapes are drawn in the code defines which shapes appear on top of the others in the display window.



Resources

- For the code and examples on the slides:
  - **Processing: a programming handbook for visual designers and artists**/ Casey Reas, Ben Fry/ The MIT Press/ Cambridge, Massachusetts, London, England (this book is available in the Library)
  - **Processing - Creative Coding and Computational Art** / Ira Greenberg
  - Code examples: www.processing.org

More on Processing

- www.processing.org/learning
  - Lots of code examples

Thank you

Questions?