Review

- Commenting your code
- Random numbers and printing messages
- mouseX, mouseY
- setup() & draw()
- frameRate(), loop(), noLoop()
- Mouse and Keyboard interaction
- Arcs, curves, bézier curves, custom shapes
- Example Sketches

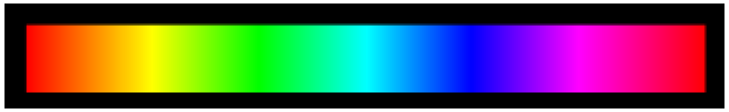
More Color

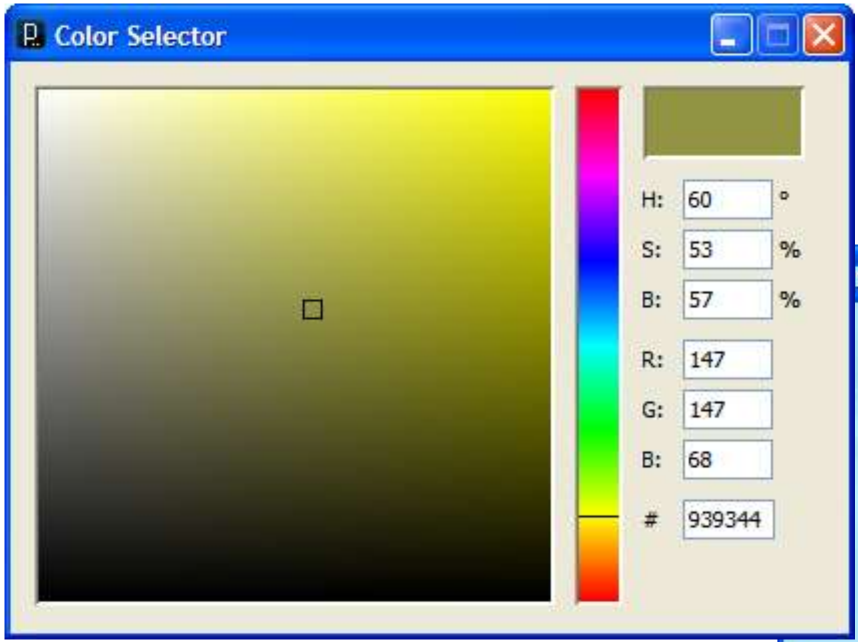
`colorMode(`*`RGB`*` or `*`HSB`*`);`

RGB: (red, green, blue)

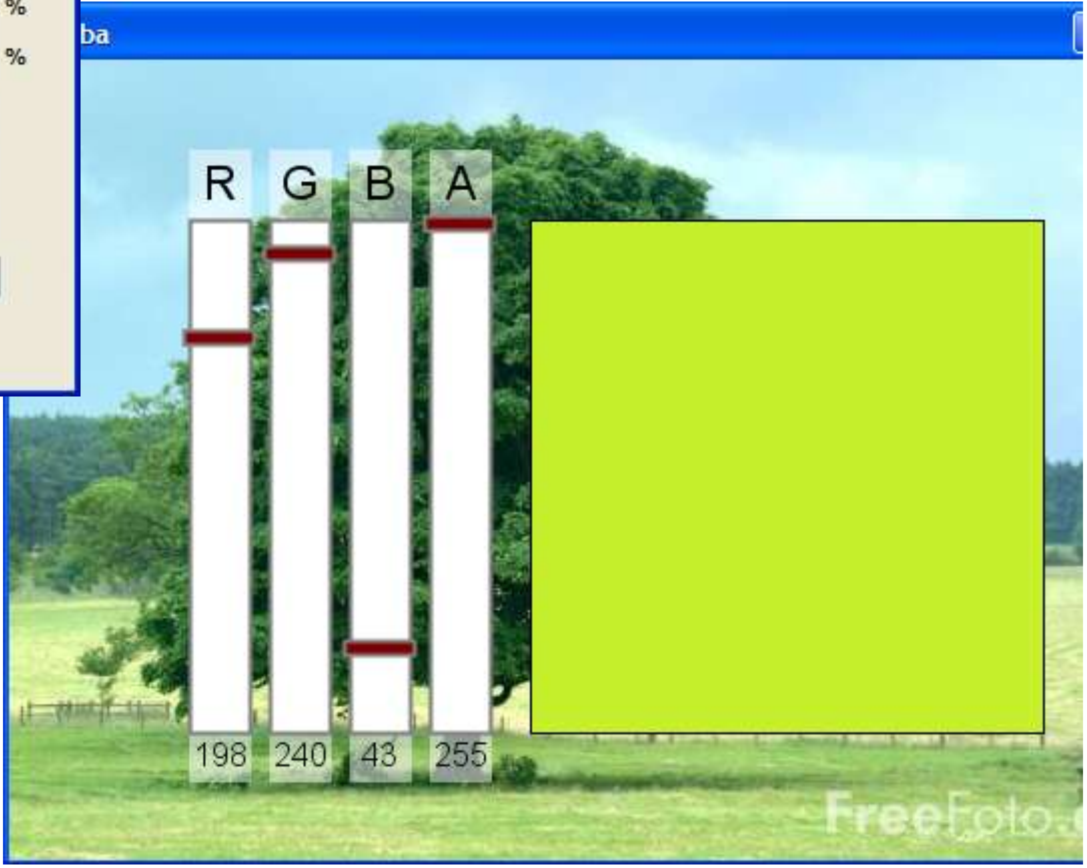
HSB:

- hue
  - “pure color”
- saturation
  - “intensity”
- brightness
  - “lightness”





H: 60 °
S: 53 %
B: 57 %
R: 147
G: 147
B: 68
939344



Decimal vs. Binary vs. Hexadecimal

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

counter.pde

Variables

- A *name* to which data can be assigned
- A <u>variable</u> name is <u>declared</u> as a specific <u>data type</u>
- Variables must begin with a letter, “_” or “$”
- Variables can container letters, digits, “_” and “$”
- Syntax:

type name [= expression];

```
int i;
int j = 12;
boolean bReady = true;
float fSize = 10.0;
color _red = color(255,0,0);
String name123 = “Fred”;
PImage img;
```

Primitive Data Types

Type	Range	Default	Bytes
boolean	{ true, false }	false	?
byte	{ 0..255 }	0	1
int	{ -2,147,483,648 .. 2,147,483,647 }	0	4
long	{ -9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807 }	0	8
float	{ -3.40282347E+38 .. 3.40282347E+38 }	0.0	4
double	*much larger/smaller*	0.0	8
color	{ #00000000 .. #FFFFFFFF }	*black*	4
char	*a single character* 'a', 'b', …	'\u0000'	2

Variables

Draws a line from last mouse position to current.

Variables used to store last mouse position

```
// Variables that store the last mouse pressed position.
int lastX;
int lastY;

void setup() {
  size(500, 300);
}

void draw() { /* must exist */ }

// Draw a line from the last mouse position
// to the current position.
void mousePressed() {
  line(lastX, lastY, mouseX, mouseY);
  lastX = mouseX;
  lastY = mouseY;
}
```

variables1.pde

Variables

Orbit mouse with two shapes.

Variables used for temporary calculated values.

```
// Mouse orbiter
float angle;                       // Orbit angle state variable

void setup() {
  size(500, 300);
  background(255);
}

void draw() {
  background(255);
  fill(0, 0, 255);
  angle = angle + 0.3;             // Increment angle
  float dX = 30.0*cos(angle);      // Mouse position offset
  float dY = 30.0*sin(angle);      // Draw two orbiting shapes
  ellipse(mouseX + dX, mouseY + dY, 5, 5);
  ellipse(mouseX - dX, mouseY - dY, 5, 5);
}
```

variables2.pde

Data Type Conversion

- Variables of some types can be converted to other types.
- Type conversion function names are the types to which data will be converted

```
// binary(…), boolean(…), byte(…),
// char(…), float(…), str(…)

float f = 10.0;
int i;

//i = f;                    // Throws a runtime error
i = int(f);

println( char(65) );       // Prints the character 'A'
```

Other "things" …

Type	Range	Default	Bytes
String	a series of chars in quotes “abc”	null	?
PImage	an image	null	?
PFont	a font for rendering text	null	?
…			

```
String message = "Hello World!";
```

Images

`PImage img;`

- Declares a variable to hold an image

`img = 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.

Image Example

```
imageExample
    └ imageExample.pde
    └ data
        └ natura-morta.jpg
```

```
PImage img;

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

imageExample.pde

Expressions

- Series of data values, variables and/or sub-expressions, related by operators and function calls, and grouped by parentheses.

- Expressions are automatically evaluated and replaced by the final evaluated value.

- Expressions can be assigned to variables using “=“
  - Expression is always on right
  - Variable name is always on left

variable_name **=** *expression;*

Operators

Symbols that operate on one or two sub-expressions.

Infix, prefix, or postfix

- Mathematical (`+`, `-`, `*`, `/`, …)
  - Perform standard mathematical operations (PEMDAS)
- Relational (`<`, `>`, `==`, `!=`, …)
  - Test relationship between related expressions.
  - Always returns a boolean value (true or false).
- Logical (`&&`, `||`, `!`)
  - Logical conjunction (and), disjunction (or), negation (not).
  - Always returns a boolean value (true or false).

Mathematical Operators

`+, -, *, /`	and …
`i ++;`	*equivalent to* `i = i + 1;`
`i += 2;`	*equivalent to* `i = i + 2;`
`i --;`	*equivalent to* `i = i - 1;`
`i -= 3;`	*equivalent to* `i = i - 3;`
`i *= 2;`	*equivalent to* `i = i * 2;`
`i /= 4;`	*equivalent to* `i = i / 4;`
`i % 3;`	the remainder after i is divided by 3 (modulo)

Examples:

```
1 + 2
slope = (y2 - y1) / (x2 - x1);
i++
```

Relational Operators

<	less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equivalent
!=	is not equivalent

Examples:

```
true
10 >= 10
'A' != 'A'
```

Logical Operators

`&&` logical conjunction (and)

both expressions must evaluate to 'true' for conjunction to evaluate to 'true'

`||` logical disjunction (or)

either expression must evaluate to 'true' for disjunction to evaluate to 'true'

`!` logical negation (not)

!true $\rightarrow$ false, !false $\rightarrow$ true

Examples:

```
true && true
true || false
!false
```

Some Built-in Mathematical Functions

```
sin(x), cos(x), tan(x), asin(x), …
abs(x), exp(x), pow(x, y), log(x), sqrt(x), …
max(x1, x2), min(x1, x2), floor(x), ceil(x), …
```

`dist(x1, y1, x2, y2)` -> distance between two points

`norm(value, low, high)` -> normalizes a value to [0-1]

… and many more, all of which can be included in an expression.

Evaluating Expressions

```
1 + 2
pow(sin(x),2) + pow(cos(x),2) == 1.0
max(1, 2, 3) >= 2
floor(2.9) == ceil(1.8)
```

Conditionals: if-statements

```
if ( boolean_expression_1 ) {

        //statements;

}
```

```
// What does this do?
void draw() {
        if ( mouseY < 50 ) {
                println("the sky");
        } else {
                println("the ground");
        }
}
```

Conditionals: if-else-statement

```
if ( boolean_expression ) {

   //statements executed when boolean_expression is true;

} else {

   //statements executed when boolean_expression is false;

}
```

```
// What does this do?
void draw() {
   if ( mouseY < 50 ) {
        println("the sky");
   } else {
        println("the ground");
   }
}
```

Conditionals: if-statements

```
if ( boolean_expression_1 ) {
      //statements;
} else if ( boolean_expression_2 ) {
      //statements;
} else if ( boolean_expression_3 ) {
      //statements;
} else {
      //statements;
}
```

Optional

Conditionals: If-statement examples

```
if (j < i) { … }

if (true) { … }

if (keyCode == 38) { … }

if (mouseX > 250 && mouseY > 250) { … }

if (speed > 100.0 && bMoving == false) { … }

if (speed > 100.0 && !bMoving) { … }

if (x < 10 || x > 20) { … }
```

What will this do?

```
void setup() {
  size(500,500);
  smooth();
  ellipseMode(CENTER);
}

void draw() {

  if ( mouseX < 250 && mouseY < 250 )
  {
    stroke(255, 0, 0);
    fill(0, 255, 0);
  }
  else if ( mouseX < 250 && mouseY >= 250 )
  {
    stroke(255, 0, 0);
    fill(0, 0, 255);
  }
  else if ( mouseX >= 250 && mouseY < 250 )
  {
    stroke(0, 0, 255);
    fill(255, 0, 0);
  }
  else
  {
    stroke(0, 0, 255);
    fill(255);
  }
  ellipse(mouseX, mouseY, 50, 30);
}
```

if1.pde

```
void setup() {
  size( 500, 500 );
  smooth();
}

void draw() {

  if ( mouseX > 100 )
  {
    background( 255, 0, 0 );
  }
  else if ( mouseX > 200 )
  {
    background( 0, 0, 255 );
  }

}
```

What does this do?

if2.pde

What does this do?

```
void setup() {
  size( 500, 500 );
  smooth();
}

void draw() {

  if ( mouseX > 200 )
  {
    background( 255, 0, 0 );
  }

  if ( mouseX > 100 )
  {
    background( 0, 0, 255 );
  }

}
```

if3.pde

Conditionals: switch-statement

- Works like an if-statement, only …
  - Expression returns any value (not limited to a boolean)
  - The first option (case) with an equivalent value is executed.
- Convenient for large numbers of value tests.

```
switch( expression ) {
  case label1:          // label1 equals expression
     statements;
     break;
  case label2:          // label2 equals expression
     statements;
     break;
  default:              // Nothing matches
     statements;
}
```

What does this do?

```
void setup() {
  size(500, 500);
  smooth();
}
void draw() {}

void keyPressed() {
  switch(key)
  {
    case 'a':
    case 'A':
      println("Turning left");
      break;
    case 's':
    case 'S':
      println("Turning right");
      break;
  }
}
```

switch1.pde

```
int positionX = 250;
int positionY = 250;
int deltaX = 0;
int deltaY = 0;

void setup() {
  size(500, 500);
  smooth();
}

void draw() {
  background(255);

  // Increment position and clip value
  positionX = positionX + deltaX;
  positionY = positionY + deltaY;

  // Clip values
  if (positionX < 0)      positionX = 0;
  if (positionX > width)  positionX = width;
  if (positionY < 0)      positionY = 0;
  if (positionY > height) positionY = height;

  // Draw ellipse
  ellipse(positionX, positionY, 50, 50);
}
```

```
void keyPressed() {
    // Change direction based on key code
    switch (keyCode) {
    case 37:
      deltaX = -2;
      deltaY = 0;
      break;
    case 39:
      deltaX = 2;
      deltaY = 0;
      break;
    case 38:
      deltaY = -2;
      deltaX = 0;
      break;
    case 40:
      deltaY = 2;
      deltaX = 0;
      break;
    case 32:
      deltaX = 0;
      deltaY = 0;
      break;
  }
}
```

switch2.pde
switch3.pde

Note the distinction between state (keyPressed) and behavior (draw).

The Walker

```
// The Walker

boolean walking = false;     // true if walking
boolean walkPose = false;    // current walk pose
float speed = 2.0;           // walk speed
float cX = 100.0;            // current walker location
float cY = 100.0;

void setup() {
  size(500, 500);
  smooth();
}
```

Continued …

What will this do?

```
void draw() {
  background(255);

  // Test current keyPressed
  if (keyPressed == true) {
    switch( keyCode ) {
      case UP:
        walking = true;
        cY -= speed;
        break;
      case DOWN:
        walking = true;
        cY += speed;
        break;
      case LEFT:
        walking = true;
        cX -= speed;
        break;
      case RIGHT:
        walking = true;
        cX += speed;
        break;
      default:
        walking = false;
    }
  } else {
    walking = false;
  }
```

```
  // Draw the walker
  fill(200);
  stroke(0);
  line(cX, cY, cX, cY+20);            // body
  ellipse(cX, cY, 10, 10);            // head
  if (walkPose == true) {
    line(cX-10, cY+10, cX+10, cY+10);// arms pose 1
    line(cX, cY+20, cX-10, cY+30);   // legs pose 1
    line(cX, cY+20, cX+10, cY+30);
  } else {
    line(cX-10, cY+5, cX+10, cY+15); // arms pose 2
    line(cX, cY+20, cX-5, cY+30);    // legs pose 2
    line(cX, cY+20, cX+5, cY+30);
  }

  // If walking, change walk pose.
  if (walking == true) {
    walkPose = !walkPose;
  }
}
```