Review

- **`setup()`** & **`draw()`**
- The event loop
- **`mouseX`**, **`mouseY`**
- Mouse and Keyboard interactions
- Arcs, curves, bézier curves, custom shapes

- Programming principals
  - Syntax is important
  - Reference manuals are your friend
  - Don't be afraid to try different things

Mouse Interaction

- Built-in predefined variables that hold the mouse X and Y locations
  - current **`mouseX mouseY`**
  - previous (last) **`pmouseX pmouseY`**
  - 0 if mouse is not in window
- Built-in predefined variables that indicate the button state:
  - is the **`mousePressed`** ?
  - which **`mouseButton`** ?
    - LEFT
    - RIGHT
    - CENTER

```
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
}

keyPressed  // a variable: true when a key is currently
   being pressed
```

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

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~		€	
130	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
140	Œ		Ž			‘	’	“	”	•
150	–	—	˜	™	š	›	œ		ž	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Text

`text(theString, x, y);`

- Draws *theString* on the sketch at (x, y)
- A string is represented by “”
- `text(“CS110 is fun!”, width/2, height/2);`

`textSize(size);`

- Sets the current font size

```
random(high);
random(low, high);
```

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

```
print(something);
println(something);
```

Print something to the Processing console.

randomEllipse

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

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

Variables

- A <u>location</u> where data is stored
- A variable name is <u>declared</u> as a specific <u>data type</u>
- Names must begin with a letter, “_” or “$” and can container letters, digits, “_” and “$”

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

Variable Uses

- Refer to a value throughout your program
  - but allow it to be changed
  - As temporary storage for a intermediate computed result
  - To parameterize – instead of hardcoding coordinates
- Special variables (preset variables)
  - **`width, height`**
  - **`mouseX, mouseY, pmouseX, pmouseY`**
- Assigned with a single =
  - known as the assignment operator
  - left side and right side are not equal

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

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!";
```

Data Type Conversion

- Types must match
- If variable types on the two sides of an assignment do not match, one must be converted
  - automatic conversion
  - explicit conversion (casting)

```
float f = 10.0;
int i = 5;

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

Mixing types and Integer Division

- 3*1.5
  - value?
  - type?
- 3/2
- 2/3
- x/y

Images

`save(filename);`
`loadImage(`*`filename`*`);`
- Loads an image from a file in the sketch folder.
- Or in the *data* subfolder.
- 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 (resize)

`imageMode(CORNER);`
- X and Y define the upper left corner
- X2 and Y2 define width and height.

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);
}
```

Conditionals: if-statement

```
if (boolean_expression) {
   statements;
}
```

What does this do?

```
void draw() {
    if (mouseX > 50 && mouseY > 50) {
        ellipse( mouseX, mouseY, 10, 10 );
    }
}
```

Logical Expressions

&& logical conjunction (and)
- both expressions must be true for conjunction to be true

|| logical disjunction (or)
- either expression must be true for disjunction to be true

! logical negation (not)
- true → false, false → true

Relational Expressions

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

Relational Expressions: Examples

```
1. if (true) { … }
2. if (10 > 10) { … }
3. if (10 >= 10) { … }
4. if ('a' == 'a') { … }
5. if ('a' != 'a') { … }
6. if ("Bryn Mawr" != "bryn mawr") { … }
```

Logical Expression Examples

```
1. if ((2 > 1) && (3 > 4)) { … }
2. if (("blah" == "blah") && (1 + 2 == 3)) { … }
3. if (!false) { … }
4. if (!(1 < -1)) { … }
5. if (!(10 < 20) || false ) { … }
6. if (!(10 > 20) && (10 < 20)) { … }
7. if ((true || false) && true) { … }
8. if ((true && false) || true )) { … }
9. …
```

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-else-if-statement

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

What does this do?

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

void draw() {
  if (mouseX < width/2) {
    if (mouseY < height/2) {
      fill(0, 255, 0);
    }
    else {
      fill(0, 0, 255);
    }
  }
  else {
    if (mouseY < height/2) {
      fill(255, 0, 0);
    }
    else {
      fill(255);
    }
  }
  ellipse(mouseX, mouseY, 50, 30);
}
```

And this?

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

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

Does this work better?

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

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

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

Simulated Motion (balldrop)

p = position
v = velocity
a = acceleration

- Constant acceleration (a)
  - assuming small time intervals (t=1)

    $p_{i+1} = p_i + v_i$

    $v_{i+1} = v_i + a$