Basic Input and Output

CSE 120 Winter 2018

Instructor: Justin Hsia

Teaching Assistants: Anupam Gupta, Cheng Ni, Eugene Oh, Sam Wolfson, Sophie Tian, Teagan Horkan

How the Facebook algorithm update will change your news feed

"Zuckerberg has said his goal for this year is to fix Facebook, whether by protecting against foreign interference and abuse or by making users feel better about how they spend time on Facebook. So, the company is set to try to have users see fewer posts from publishers, businesses and celebrities, and more from friends and family.

"Facebook says it will start prioritising news sources deemed trustworthy in the US and then internationally. It says it has surveyed a 'diverse and representative sample' of US users and next week it will begin testing prioritising the news sources deemed trustworthy."

- http://metro.co.uk/2018/01/20/facebook-algorithm-update-will-change-news-feed-7245129/

Administrivia

- Assignments:
  - Reading Check 3 due tomorrow *before lab* (1/25)
  - Jumping Monster due Friday (1/26)
    - harder assignment, will take time!
- "Big Idea" this week: The Internet
- Upcoming: Creativity Project, Midterm (2/5)
  - Vote on Piazza for Midterm Review Session?

Lecture Outline

- **Other Useful Processing Tools**
- User Input and Output
  - Mouse (input)
  - Keyboard (input)
  - Text (output)

System Variables

- Special variables that hold values related to the state of the program, often related to user input
  - You don't need to declare these variables
  - These variables will update automatically as the program runs
  - Colored **pink/magenta-ish** in the Processing environment

- We've used some of these already:
  - `mouseX`, `mouseY`, `width`, `height`

mouse position

canvas size

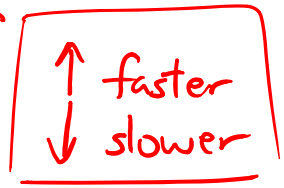
- We'll see more today

Drawing and Frames

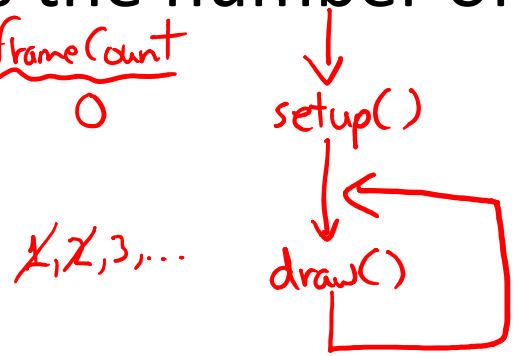
- Control and track how frequently **`draw`**`()` runs
  - Each time **`draw`**`()` runs, it is called a new *frame*

- `frameRate()` changes the desired number of frame updates there are *per second*
  - Larger argument is faster
  - Default is `frameRate(60)`

frame Rate (30) refreshes half as frequently



- System variable `frameCount` returns the number of frames since the start of the program
  - Starts at 0 in **`setup`**`()`



Drawing and Frames

- Control and track how frequently **`draw`**( ) runs
  - Each time **`draw`**( ) runs, it is called a new *frame*

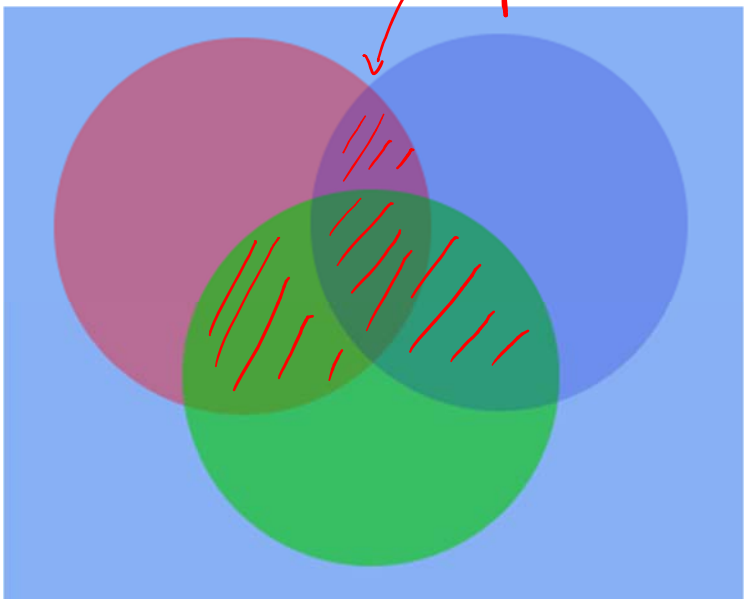
- `noLoop`( ) stops **`draw`**( ) from being continuously executed
  - Can restart using `loop`( )

Transparency/Opacity

- You can add a 4th argument to a color!
  - This also applies to the `fill()` and `stroke()` functions
- This argument also takes an integer between 0–255
  - 0 is fully transparent (invisible)
  - 255 is fully opaque (the default)

```
size(400, 320);
noStroke();
background(136, 177, 245);

fill(255, 0, 0, 100);
ellipse(132, 120, 200, 200);

fill(0, 200, 0, 150);
ellipse(200, 200, 200, 200);

fill(0, 0, 200, 50);
ellipse(268, 118, 200, 200);
```

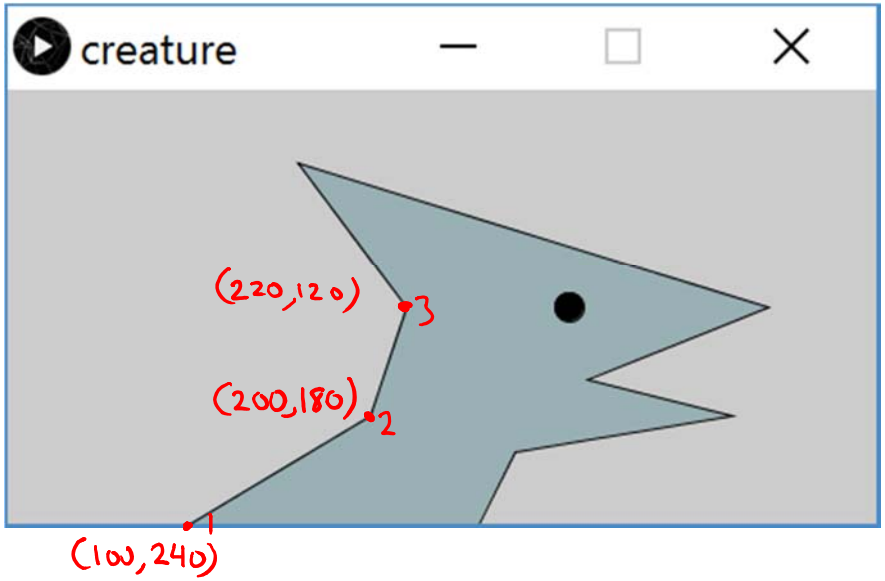
slightly transparent



Custom Shapes

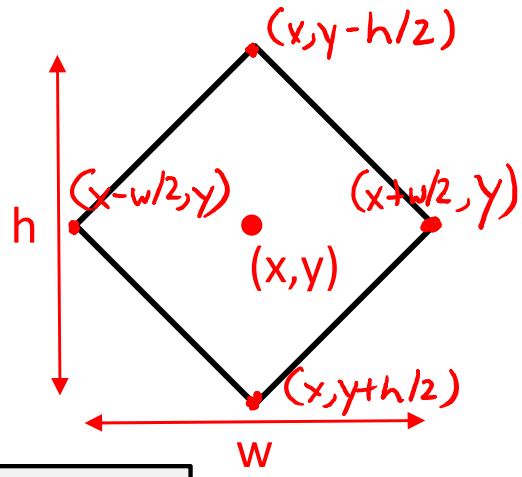
- Define vertices between `beginShape()` and `endShape()`
  - If planning to reuse, best to create in a separate function

so can use like rect(), ellipse(), etc.



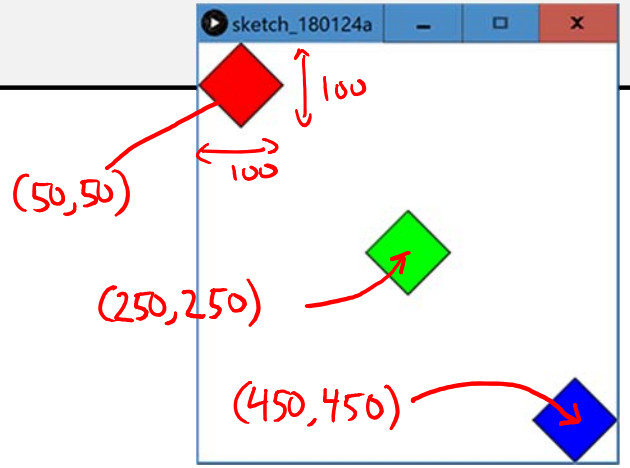

```
creature
1  size(480,240);
2
3  fill(153, 176, 180);
4  beginShape();
5  vertex(100, 240);
6  vertex(200, 180);
7  vertex(220, 120);
8  vertex(160,  40);
9  vertex(420, 120);
10 vertex(320, 160);
11 vertex(400, 180);
12 vertex(280, 200);
13 vertex(260, 240);
14 endShape();
15
16 fill(0);
17 ellipse(310, 120, 16, 16);
```

Functions Practice: Diamond



- Fill in the code to produce:

```
void diamond( float x, float y, float w, float h ) {
   beginShape();
      vertex( x , y-h/2 );
      vertex( x+w/2 , y );
      vertex( x , y+h/2 );
      vertex( x-w/2 , y );
      vertex( x , y-h/2 );
endShape();
}
```



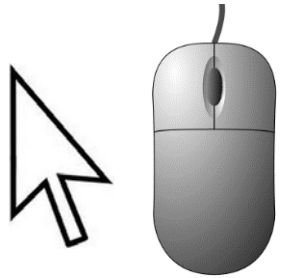

```
void draw() {
    fill(255,0,0);
    diamond(50,50,100,100);
    fill(0,255,0);
    diamond(250,250,100,100);
    fill(0,0,255);
    diamond(450,450,100,100);
}
```

Lecture Outline

- Other Useful Processing Tricks
- **User Input and Output ***
  - Mouse
  - Keyboard
  - Text

* We will look at a subset of the available Processing commands. For a full list, see the Processing Reference.

The Mouse



- System variables:
  - `mouseX` – x-coordinate of mouse in current frame
  - `mouseY` – y-coordinate of mouse in current frame
  - `pmouseX` – x-coordinate of mouse in previous frame
  - `pmouseY` – y-coordinate of mouse in previous frame
  - `mousePressed` – is a button currently being pressed?
- Built-in functions:
  - `mousePressed()` – called very time a button is pressed
  - `mouseReleased()` – called every time a button is released

current frame

previous frame

boolean → true / false

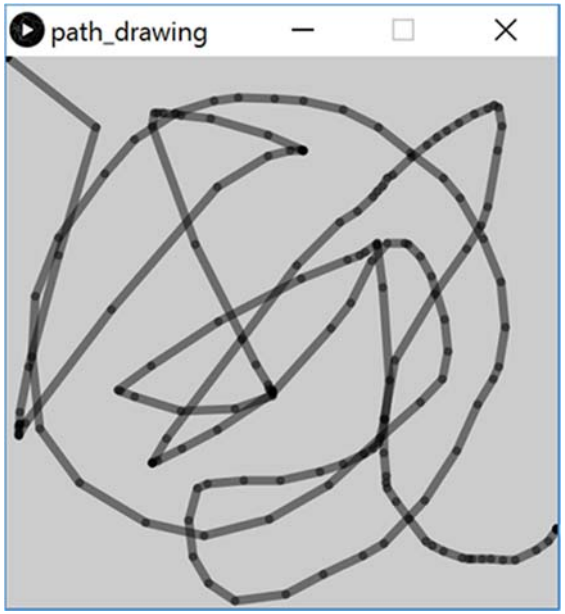
can be confusing. You should only be using one of these.

Example: Path Drawing

- Last lecture we wrote a *dot*-drawing program

ellipse(mouseX, mouseY, 10, 10);

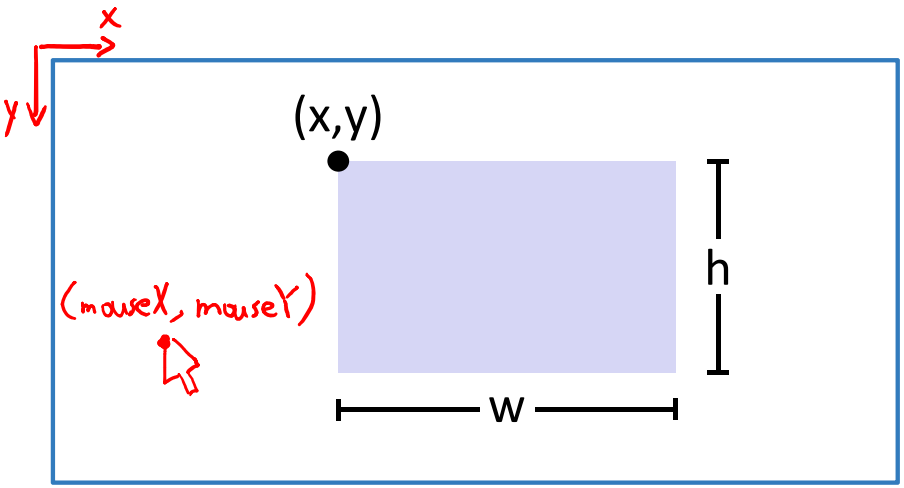
- We can additionally use `pmouseX` and `pmouseY` to create a *path*-drawing program

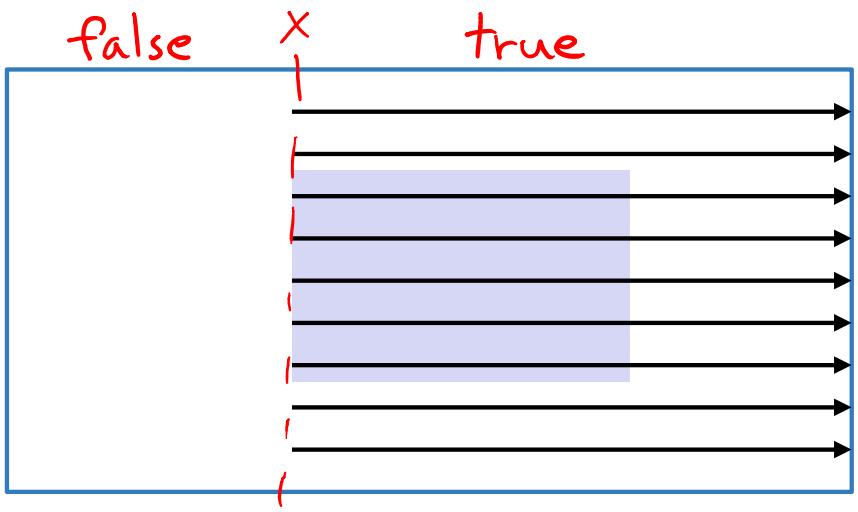



```
7  void setup() {
8    size(500,500);          // set drawing canvas size
9    strokeWeight(8);        // thicker lines
10   stroke(0,0,0, 120);     // black line with some transparency
11   frameRate(30);          // slow down the frame rate
12 }
13
14 void draw() {
15   line(mouseX, mouseY, pmouseX, pmouseY);
16 }
```

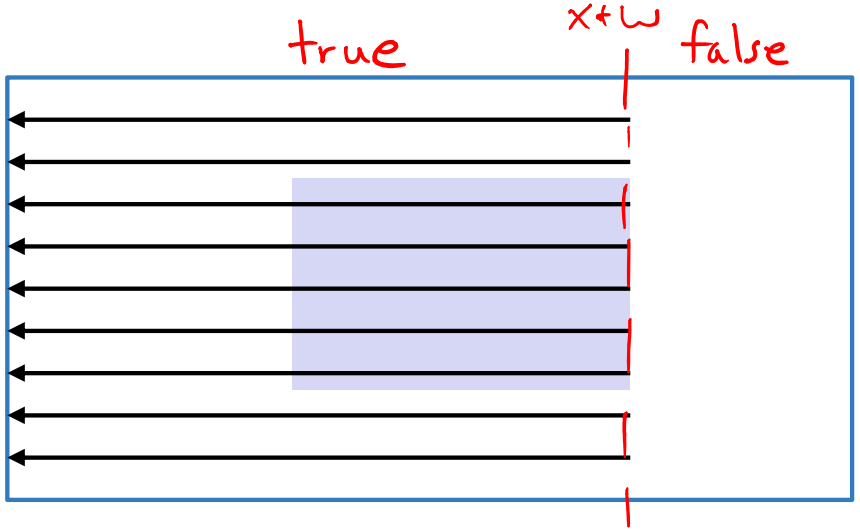
← drawing the path your mouse takes

Hovering Over a Rectangle



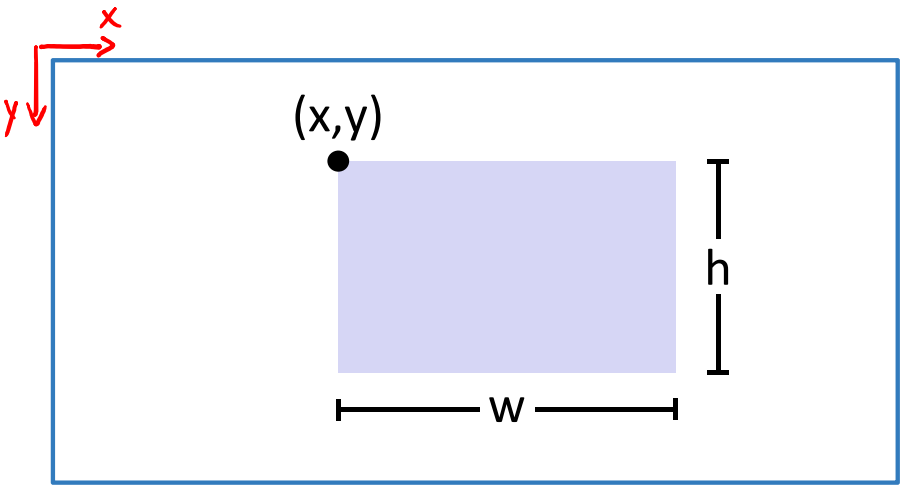


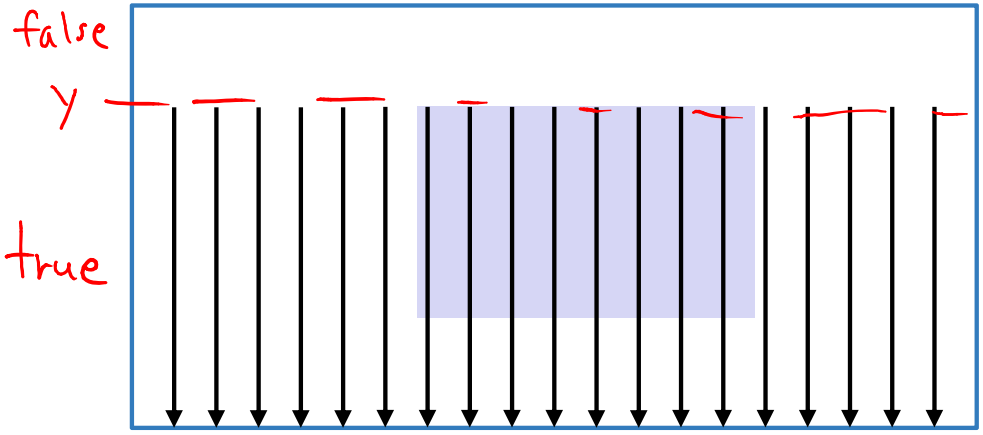

```
if(mouseX >= x)
```



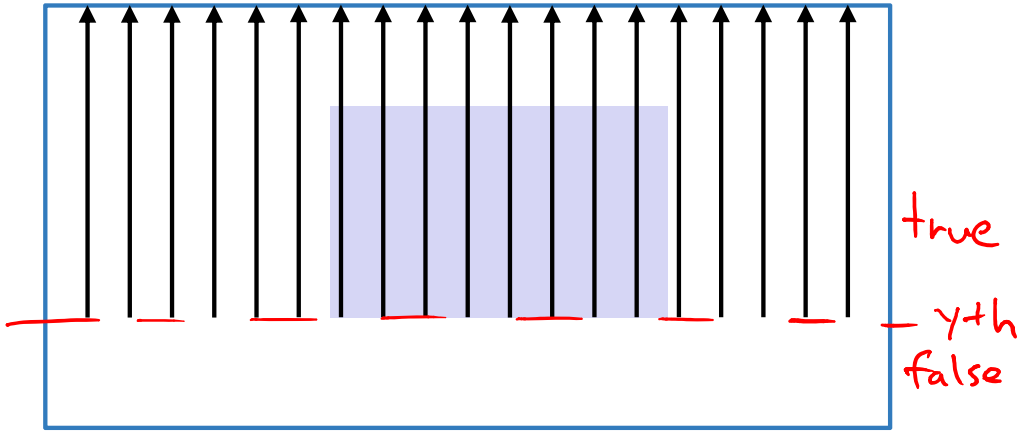

```
if(mouseX <= x + w)
```

Hovering Over a Rectangle



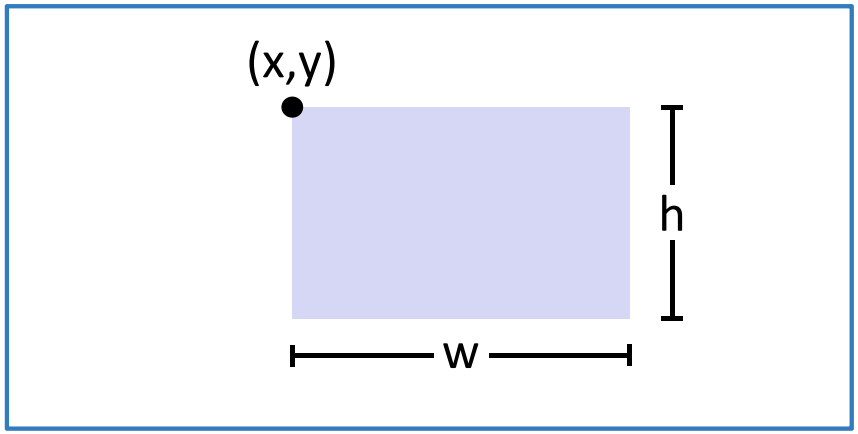


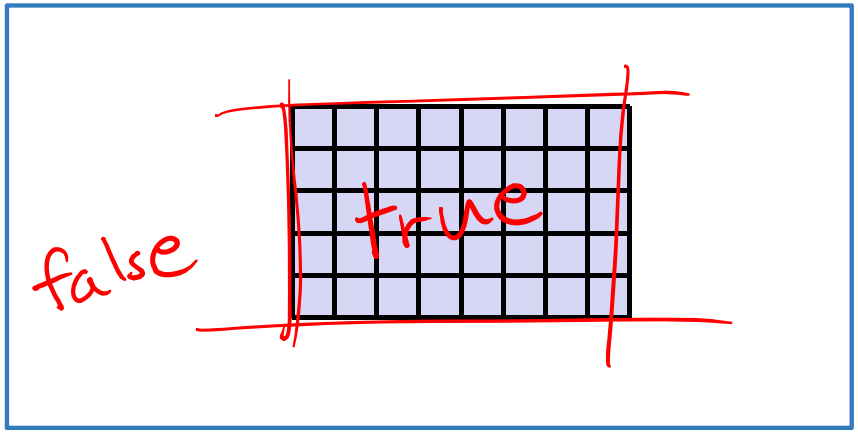

```
if(mouseY >= y)
```




```
if(mouseY <= y + h)
```

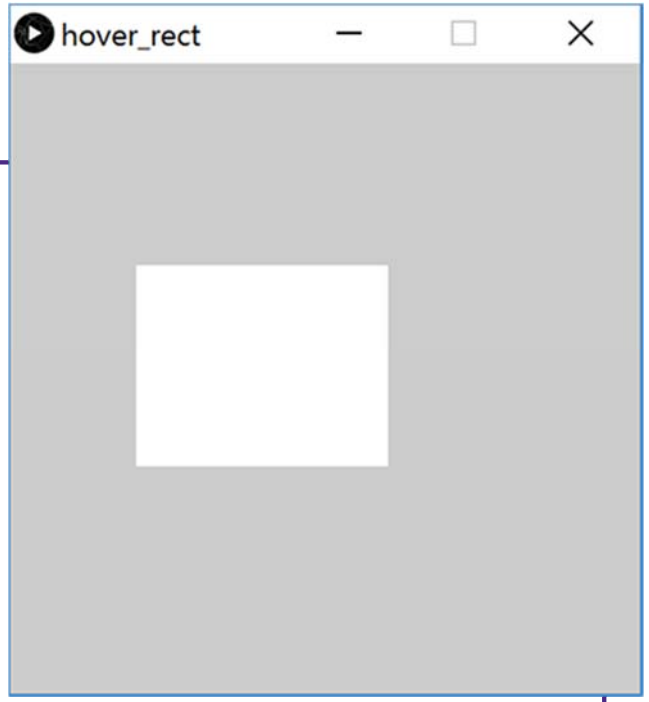
Hovering Over a Rectangle






```
if( (mouseX >= x)     &&
    (mouseX <= x + w) &&
    (mouseY >= y)     &&
    (mouseY <= y + h) )
```

Hovering Over a Rectangle




```
int x = 100;     // x-position of upper-left corner
int y = 160;     // y-position of upper-left corner
int w = 200;     // width of rectangle
int h = 160;     // height of rectangle

void setup() {
  size(500,500);  // set drawing canvas size
  noStroke();     // no shape outlines
}

void draw() {
  background(204);  // clear the canvas

  if ( (mouseX >= x) && (mouseX <= x+w) && (mouseY >= y) && (mouseY <= y+h) ) {
    fill(0);        // black is mouse is hovering over
  } else {
    fill(255);      // white otherwise
  }

  rect(x, y, w, h); // draw the rectangle
}
```

The Keyboard

- System variables:
  - `key` – stores the ASCII value of the last key press
  - `keyCode` – stores codes for non-ASCII keys (*e.g.* UP, LEFT)
  - `keyPressed` – is any key currently being pressed?

again, recommended to only use one or the other for each program

- Built-in functions:
  - `keyPressed()` – called every time a key is pressed

- New datatype: `char`
  - Stores a single character (really just a number)
  - Should be surrounded by *single* quotes
  - *e.g.* `char letter = 'a';` ← actually the ASCII value for 'a'

Example: What does this code do?

```
1  int position = 0;
2
3  void setup() {
4    size(400, 100);
5    noStroke();
6    background(0);
7    fill(0);
8  }
9
10 void draw() {
11   ellipse(position, 40, 40, 40);
12 }
13
14 void keyPressed() {
15   if(key == 'g'){
16     fill(0, 255, 0);
17   }
18
19   if(key == 'y') {
20     fill(255, 255, 0);
21   }
22
23   if(key == 'm') {
24     fill(255, 0, 255);
25   }
26
27   position = position + 50; // position+=50;
28 }
```

draws a circle at (position, 40) every frame

runs any time a key is pressed

these are chars

change fill to green

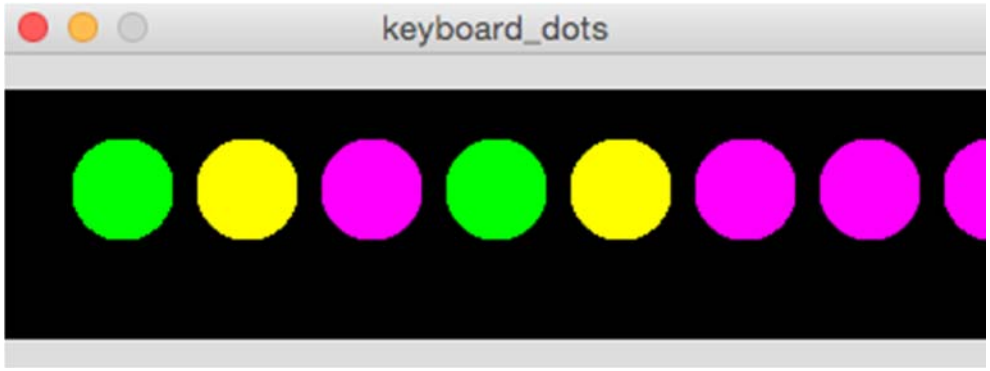
fill to yellow

fill to magenta

draw next circle at new position

this executes no matter which key is pressed!

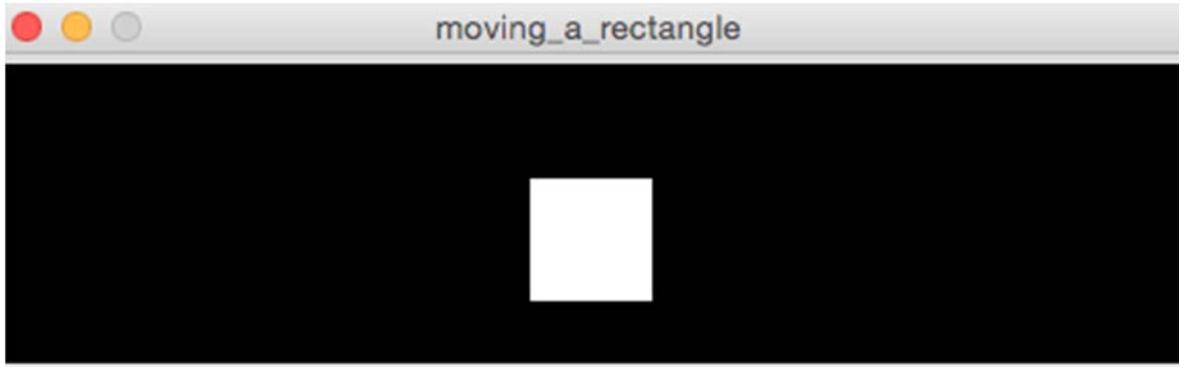
Example: Keyboard Dots



keyboard_dots

```
int position = 0;

void setup() {
  size(400, 100);
  noStroke();
  background(0);
  fill(0);
}

void draw() {
  ellipse(position, 40, 40, 40);
}

void keyPressed() {
  if(key == 'g'){
    fill(0, 255, 0);
  }

  if(key == 'y') {
    fill(255, 255, 0);
  }

  if(key == 'm') {
    fill(255, 0, 255);
  }

  position = position + 50; // position+=50;
}
```

Example: Moving a Rectangle



- **Note:** non-character keys, such as the arrow keys (UP, DOWN, LEFT, RIGHT) are *coded* keys

```
11    if(keyPressed) {
12      if(key == CODED) {
13        if(keyCode == LEFT) {
14          x = x - 1;
15        }
```

Example: Moving a Rectangle

moving_a_rectangle

```
1  int x = 215;
2
3  void setup() {
4    size(480, 120);
5  }
6
7  void draw() {
8    background(0);
9    rect(x, 45, 50, 50);
10
11   if(keyPressed) {
12     if(key == CODED) {
13       if(keyCode == LEFT) {
14         x = x - 1;
15       }
16
17       if(keyCode == RIGHT) {
18         x = x + 1;
19       }
20     }
21   }
22 }
```

Text Output

char 'h' ← 1 character only
vs.
String "hello" ← many characters in sequence

- `println(yourText);`
  - Prints `yourText` to the *console*, which is the black area below your Processing code
  - Useful for debugging, particularly your portfolio
- `text(yourText, x, y);`
  - Prints `yourText` on the drawing canvas, starting with the *bottom-left* corner at coordinate `(x,y)`
  - Change the size of your text using `textSize(size);`
- `yourText` should be between *double* quotes
  - We will talk more about the datatype `String` later

Example: Displaying Typed Keys

display_letters

```
1  void setup() {
2    size(120, 120);
3    textSize(64);
4    textAlign(CENTER);
5  }
6
7  void draw() {
8    background(0);
9    text(key, 60, 80);
10 }
```

Looking Forward

- Next week is the Creativity Assignment
  - In pairs, you will be asked to brainstorm TWO Processing projects *of your choice*
  - You will implement and submit ONE of your two projects
  - The point is to use the tools available to you to make something fun and creative!
  - Planning document due Tuesday (1/30)
  - Actual programs due next Friday (2/2)

- Portfolio Update 1 is due Wednesday (1/31)
  - Taijitu, Logo Design, Lego Family, Animal Functions
  - Ask your TAs for assistance if you encounter problems!