PAT 204/504 (Fall 2025)

# Creative Coding

## Lecture 2: Processing Basics

Instructor: Hao-Wen Dong

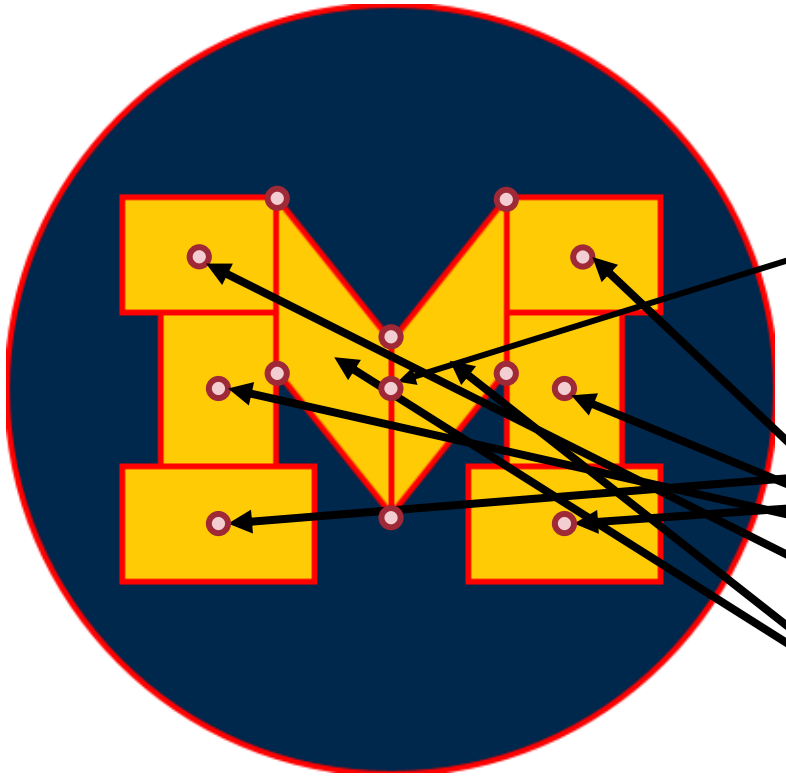
UNIVERSITY OF MICHIGAN

# More on Shapes

# Can you recreate the Block M in Processing?

- Michigan **Blue**:  **#00274C**
- Michigan **Maize**:  **#FFCB05**

# My Version



```
void setup() {
  // Create a 400x400 canvas
  size(400, 400);
}

void draw() {
  // Set the background color to white
  background(255);

  // Draw the shapes without outlines
  noStroke();

  // Draw the blue circle at the back
  fill(#00274C);
  circle(200, 200, 400);

  // Set the anchor point of rectangles to the center
  rectMode(CENTER);

  // Set up the yellow text color
  fill(#FFCB05);

  // Draw the feet
  rect(110, 270, 100, 60);
  rect(290, 270, 100, 60);

  // Draw the columns
  rect(110, 210, 60, 150);
  rect(290, 210, 60, 150);

  // Draw the caps
  rect(100, 130, 80, 60);
  rect(300, 130, 80, 60);

  // Draw the "V"
  quad(140, 100, 140, 190, 200, 265, 200, 175);
  quad(260, 100, 260, 190, 200, 265, 200, 175);
}
```

# Strokes

- `noStroke()` Disable outlines
- `strokeWeight(weight)` Set outline weight
- `stroke(color)` Set outline color

# Strokes

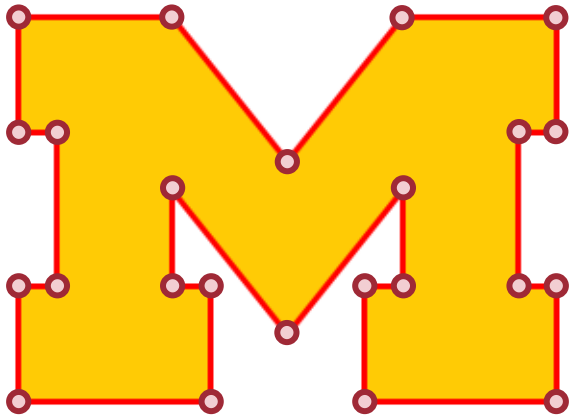
```
noStroke()
```

```
strokeWeight(3)
stroke(#FF0000)
```

# Custom Shapes

```
// Draw a custom shape
beginShape();
vertex(60, 100);
vertex(60, 160);
vertex(80, 160);
vertex(80, 240);
vertex(60, 240);
vertex(60, 300);
vertex(160, 300);
vertex(160, 240);
vertex(140, 240);
vertex(140, 190);
vertex(200, 265);
vertex(260, 190);
vertex(260, 240);
vertex(240, 240);
vertex(240, 300);
vertex(340, 300);
vertex(340, 240);
vertex(320, 240);
vertex(320, 160);
vertex(340, 160);
vertex(340, 100);
vertex(260, 100);
vertex(200, 175);
vertex(140, 100);
endShape(CLOSE);
```

# Custom Shapes



```
// Draw a custom shape
beginShape();
vertex(60, 100);
vertex(60, 160);
vertex(80, 160);
vertex(80, 240);
vertex(60, 240);
vertex(60, 300);
vertex(160, 300);
vertex(160, 240);
vertex(140, 240);
vertex(140, 190);
vertex(200, 265);
vertex(260, 190);
vertex(260, 240);
vertex(240, 240);
vertex(240, 300);
vertex(340, 300);
vertex(340, 240);
vertex(320, 240);
vertex(320, 160);
vertex(340, 160);
vertex(340, 100);
vertex(260, 100);
vertex(200, 175);
vertex(140, 100);
endShape(CLOSE);
```

# endShape(CLOSE) vs endShape()

```
// Draw a custom shape
beginShape();
vertex(60, 100);
vertex(60, 160);
vertex(80, 160);
vertex(80, 240);
vertex(60, 240);
vertex(60, 300);
vertex(160, 300);
vertex(160, 240);
vertex(140, 240);
vertex(140, 190);
vertex(200, 265);
vertex(260, 190);
vertex(260, 240);
vertex(240, 240);
vertex(240, 300);
vertex(340, 300);
vertex(340, 240);
vertex(320, 240);
vertex(320, 160);
vertex(340, 160);
vertex(340, 100);
vertex(260, 100);
vertex(200, 175);
vertex(140, 100);
endShape();
```

# My Version

```
void setup() {
  // Create a 400x400 canvas
  size(400, 400);
}
```

```
void draw() {
  // Set the background color to white
  background(255);

  // Draw the shapes without outlines
  noStroke();

  // Draw the blue circle at the back
  fill(#00274C);
  circle(200, 200, 400);

  // Set the anchor point of rectangles to the center
  rectMode(CENTER);

  // Set up the yellow text color
  fill(#FFCB05);

  // Draw the feet
  rect(110, 270, 100, 60);
  rect(290, 270, 100, 60);

  // Draw the columns
  rect(110, 210, 60, 150);
  rect(290, 210, 60, 150);

  // Draw the caps
  rect(100, 130, 80, 60);
  rect(300, 130, 80, 60);

  // Draw the "V"
  quad(140, 100, 140, 190, 200, 265, 200, 175);
  quad(260, 100, 260, 190, 200, 265, 200, 175);
}
```

# setup() & draw()

# setup() & draw()

- **setup()**
  - Run only **once** when the program starts!
- **draw()**
  - Run **every frame** (default fps is 60)

frames per second

```
void setup() {
  // Create a 400x400 canvas
  size(400, 400);
}
```

```
void draw() {
  // Set the background color to white
  background(255);

  // Draw the shapes without outlines
  noStroke();

  // Draw the blue circle at the back
  fill(#00274C);
  circle(200, 200, 400);

  // Set the anchor point of rectangles to the center
  rectMode(CENTER);

  // Set up the yellow text color
  fill(#FFCB05);

  // Draw the feet
  rect(110, 270, 100, 60);
  rect(290, 270, 100, 60);

  // Draw the columns
  rect(110, 210, 60, 150);
  rect(290, 210, 60, 150);

  // Draw the caps
  rect(100, 130, 80, 60);
  rect(300, 130, 80, 60);

  // Draw the "V"
  quad(140, 100, 140, 190, 200, 265, 200, 175);
  quad(260, 100, 260, 190, 200, 265, 200, 175);
}
```

# setup() & draw(): Examples

```
void setup() {
  size(400, 400);
}

void draw() {
  background(50);
  circle(mouseX, mouseY, 10);
}
```

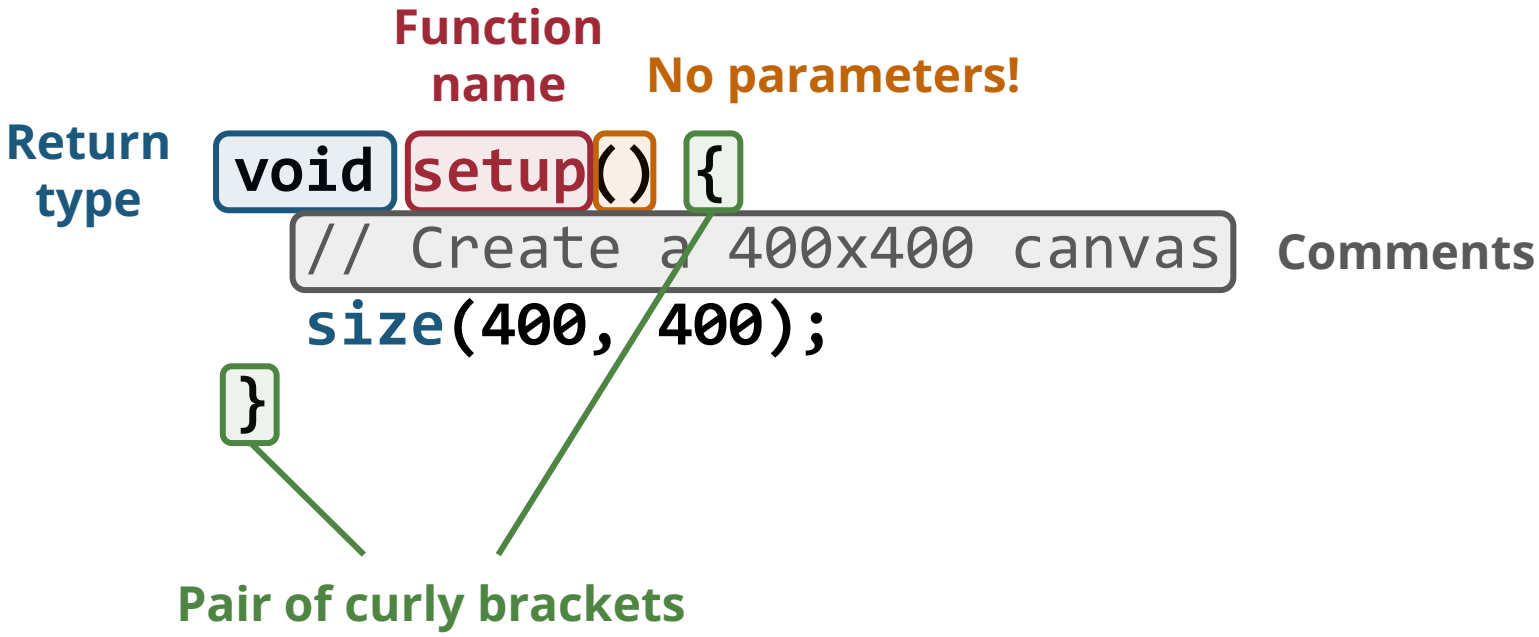
```
void setup() {
  size(400, 400);
  background(50);
}

void draw() {
  circle(mouseX, mouseY, 10);
}
```

# Controlling `draw()`

- Add to **`setup()`**
  - **`noLoop()`** Disable looping the `draw()` call
  - **`loop()`** Enable looping the `draw()` call
  - **`frameRate(fps)`** Set the frame rate of looping `draw()` calls

# Functions

# Defining a Function



# Why Functions?

- **Modularity**
  - Breakdown code into several **self-contained** modules
- **Reusability**
  - **Reuse a routine elsewhere** instead of rewriting it again!
- **Readability**
  - `dist(x1, x1, x2, y2)` vs. `sqrt((x1 – x2) * (x1 – x2) + (y1 –y2) * (y1 – y2))`
- **Scoping**
  - Avoid variable name pollution → we'll get to this later

# `mouseClicked()` Function

- Execute whenever the mouse in clicked
- Can be handy in many ways!
  - Resetting the whole canvas in an interactive session!
  - Taking screenshots for your assignments!

```
void mouseClicked() {
  saveFrame("screenshot.png");
}
```

# Built-in Mouse Functions

| Function | Called when |
| --- | --- |
| `mousePressed()` | mouse is *pressed* |
| `mouseReleased()` | mouse is *released* |
| `mouseClicked()` | mouse is *clicked* (pressed & released) |
| `mouseMoved()` | mouse is *being moved* but *not pressed* |
| `mouseDragged()` | mouse is *being moved* and *pressed* |

# Built-in Mouse Functions: Example

```
void setup() {
  size(400, 400);
  background(50);
}

void draw(){
}
```

Draw by pressing and dragging the mouse

```
void mouseDragged() {
  noStroke();
  circle(mouseX, mouseY, 10);
}
```

Clear everything when the mouse is clicked

```
void mouseClicked() {
  background(50);
}
```

# Built-in Keyboard Functions

| Function | Called when |
| --- | --- |
| `keyPressed()` | a key is *pressed* |
| `keyReleased()` | a key is *released* |
| `keyTyped()` | a key is *clicked* (called repeatedly if held down) |

# Variables

# Define a Variable

- Declare → Initialize → Use

**Declare**

```
float x;
```

**Initialize**

```
void setup() {
  size(400, 400);
  x = 200;
}
```

**Use**

```
void draw() {
  circle(x, 200, 100);
}
```

# Declare & Initialize

- Declare + Initialize → Use

**Declare + Initialize**

Data type

```
float x = 200;
```

**Use**

```
void setup() {
  size(400, 400);
}

void draw() {
  circle(x, 200, 100);
}
```

# Data Types

- **`int`** `0, 1, 2, ..., -1, -2, ...`
- **`float`** `0.0, 1.5, 3.14159, -2.71828`
- **`boolean`** `true, false`

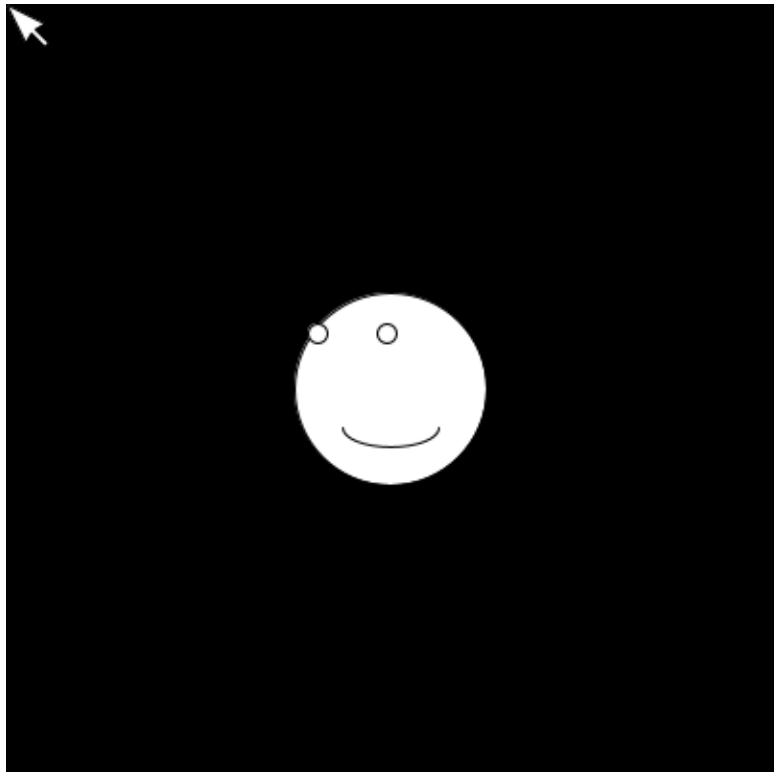
- Examples
  - `int count = 0`
  - `float x = 10.5`
  - `boolean isGameOver = false`

# Built-in Global Variables

- Predefined variables that you can use right away
- Examples:
  - `width`, `height`
  - `mouseX`, `mouseY`
  - `mousePressed`, `keyPressed`
  - `key`, `keycode`
  - `rectMode(CENTER)`

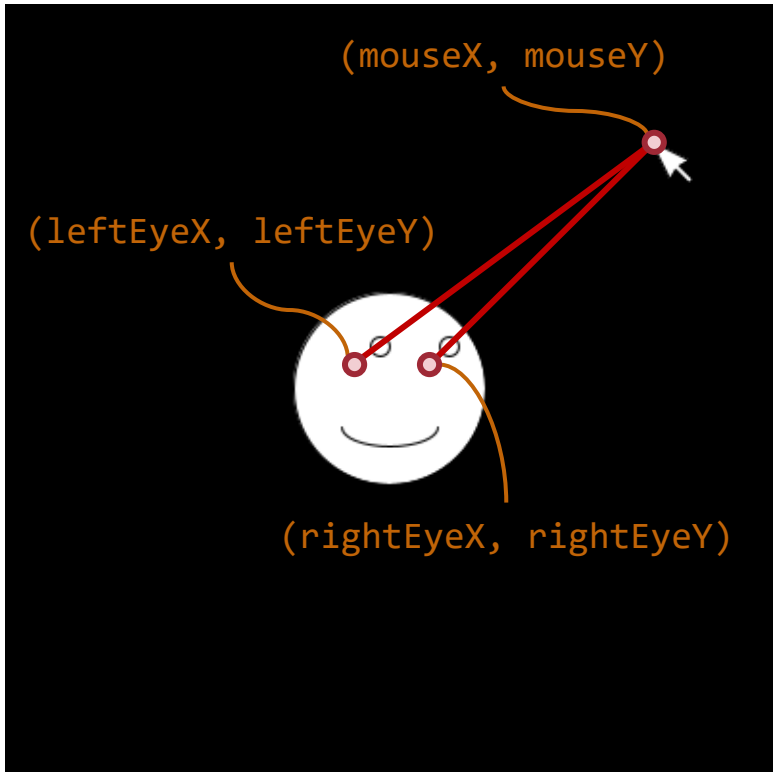
# 🧐 **Exercise**: ~~Creepy~~ Eyes

- Make a simple face where the eyes will **stare at the direction where the mouse is**!
- Hints
  - Start with drawing a **static face**
  - Use `arc()` to get the smile
    - `arc(200, 220, 50, 20, 0, PI)`
  - Use **mouseX** & **mouseY** to find where the mouse is



# 🧐 **Exercise**: ~~Creepy~~ Eyes

```
// Calculate the position of the eyes
leftDeltaX = (mouseX – leftEyeX) * scale;
leftDeltaY = (mouseY – leftEyeY) * scale;
rightDeltaX = (mouseX – rightEyeX) * scale;
rightDeltaY = (mouseY – rightEyeY) * scale;

// Draw the eyes
circle(leftEyeX + leftDeltaX, leftEyeY + leftDeltaY, 10);
circle(rightEyeX + rightDeltaX, rightEyeY + rightDeltaY, 10);
```

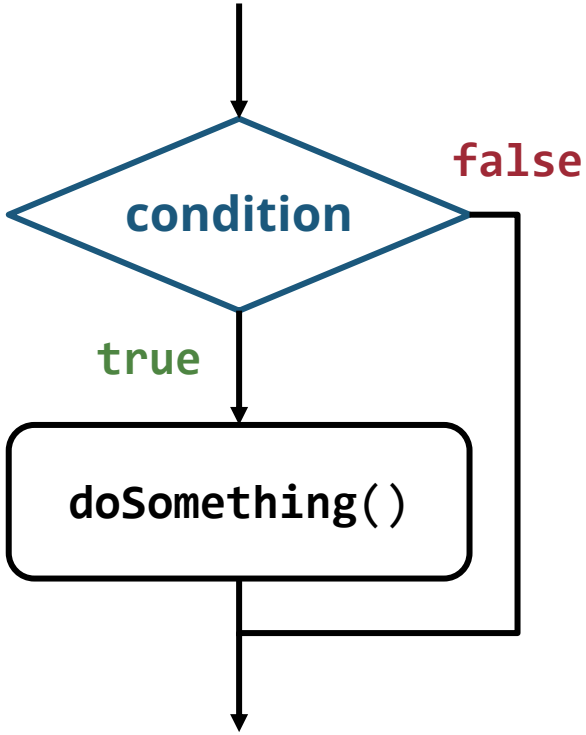


# Conditionals

# `if` Statement

- Control the program flow based on a **condition**

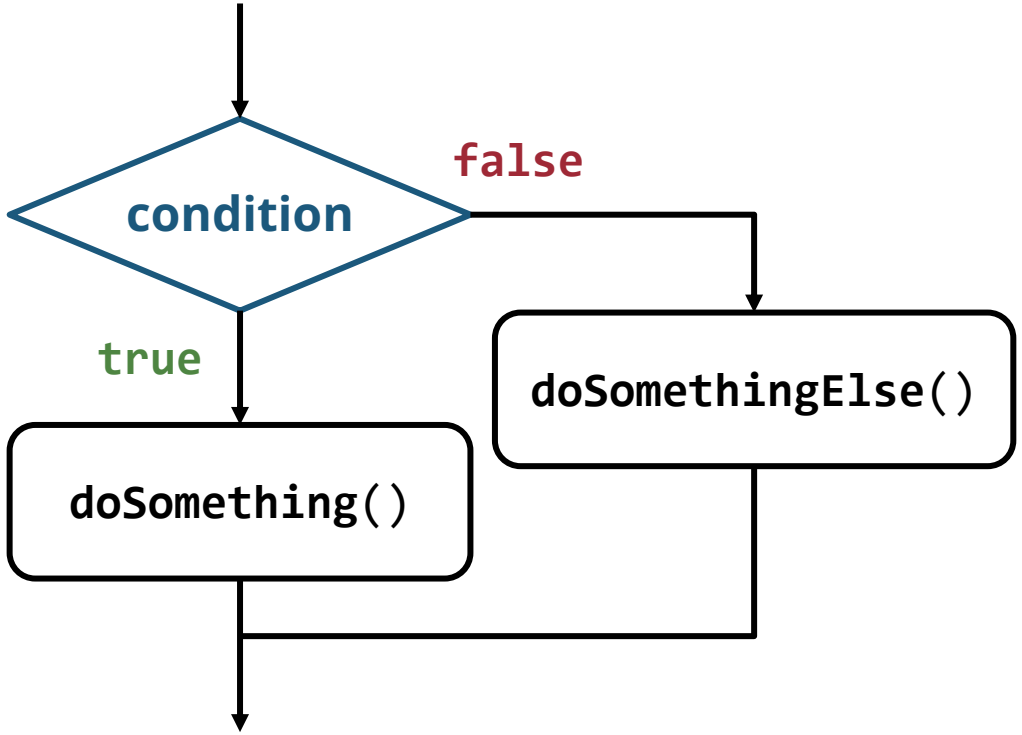
```
if (condition) {
  doSomething();
}
```



# `if-else` Statement

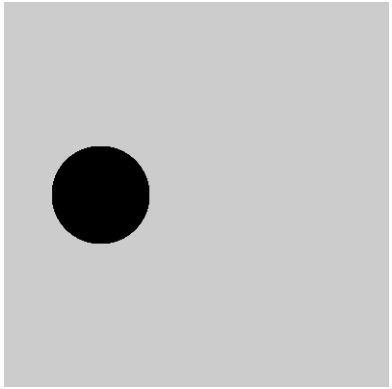
- Control the program flow based on a **condition**

```
if (condition) {
  doSomething();
} else {
  doSomethingElse();
}
```

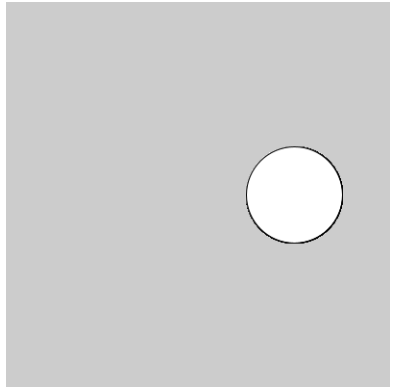


# `if-else` Statement: Examples

```
float x = 100;

void setup() {
  size(400, 400);
}

void draw() {
  if (x < 200) {
    fill(0);
  } else {
    fill(255);
  }
  circle(x, 200, 100);
}
```



```
float x = 300;

void setup() {
  size(400, 400);
}

void draw() {
  if (x < 200) {
    fill(0);
  } else {
    fill(255);
  }
  circle(x, 200, 100);
}
```



# Relational Operators

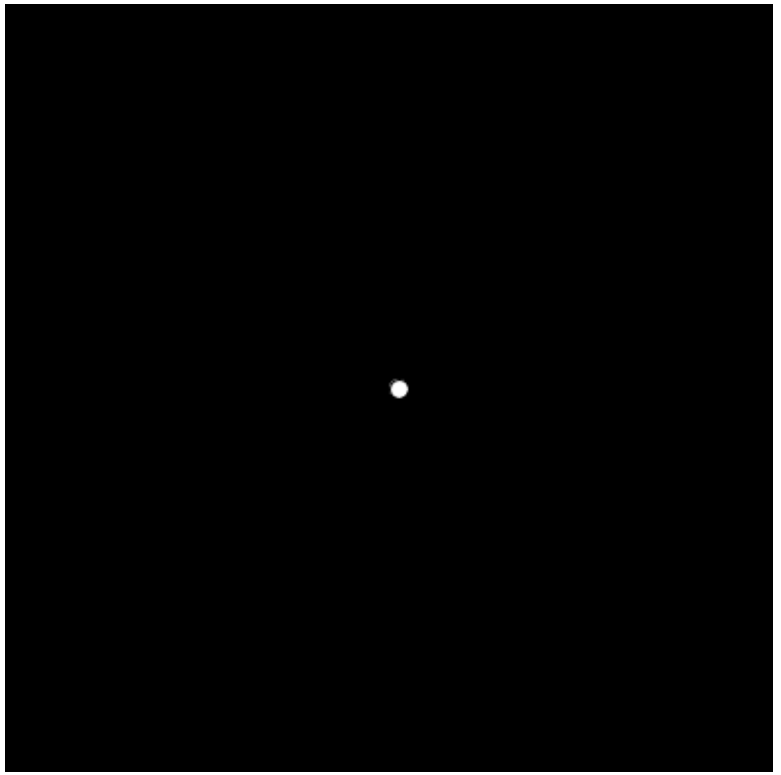
- `>` greater than
- `<` less than

- `>=` greater than or equal to
- `<=` less than or equal to

- `==` equal to
- `!=` not equal to

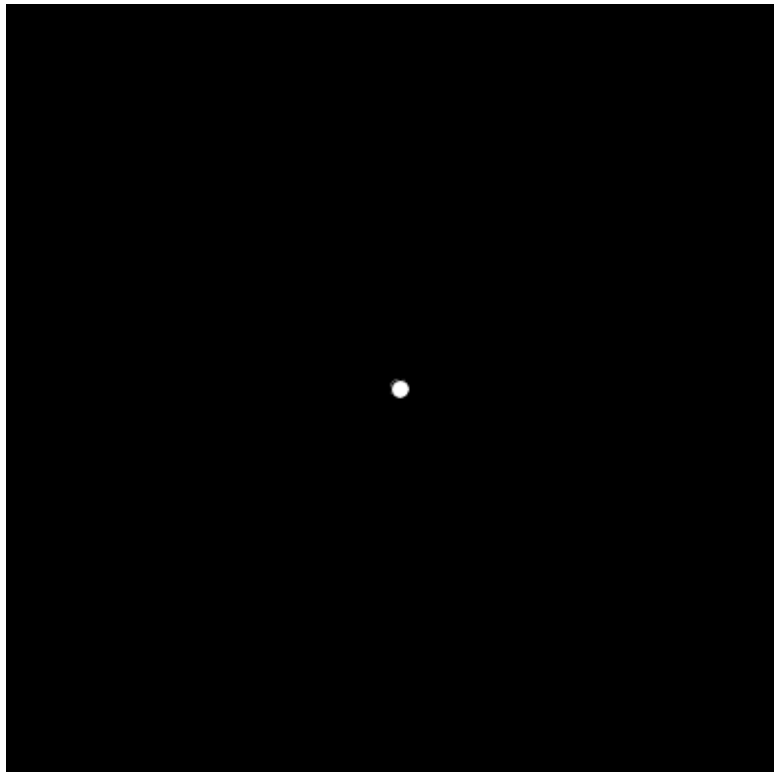
# 🧐 **Exercise**: Bouncing Ball

- The ball bounces back when it hit the walls
- Think about
  - What **variables** do I need?
  - How do I **check if the ball has hit the wall**?



# First Step: Never-stopping Ball

```
// Current x-position of the ball
float x = 200;

// Current speed of the ball
float speedX = 5;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  // Move the ball
  x = x + speedX;

  // Draw the ball
  circle(x, 200, 10);
}
```



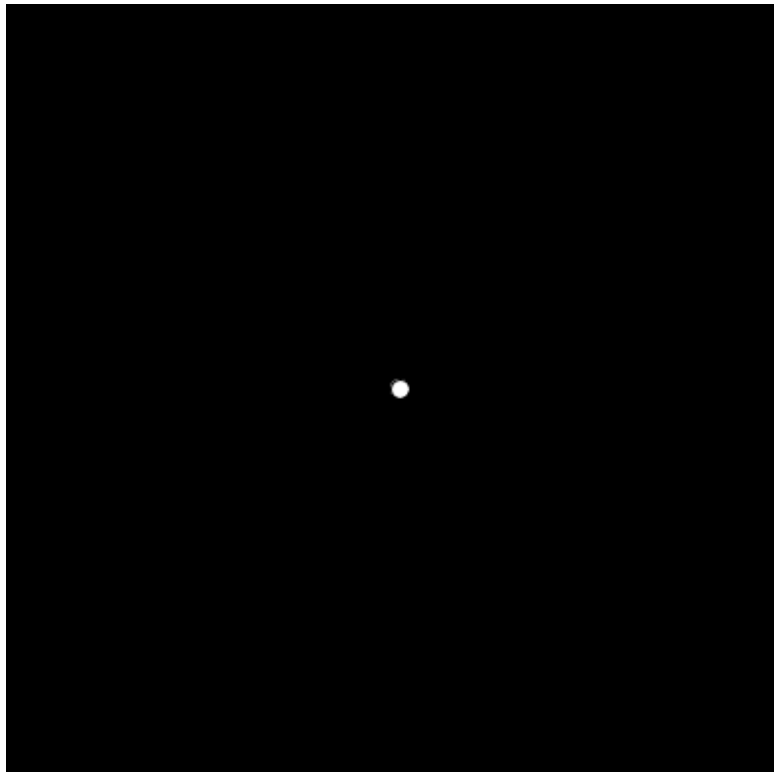
# 🧐 Exercise: Bouncing Ball

```
float ballSize = 10;
float x;
float speedX = 5;

void setup() {
  size(400, 400);
  x = width / 2;
}

void draw() {
  background(0);

  if (x > width - ballSize / 2) {
    speedX = -speedX;
  } else if (x < ballSize / 2) {
    speedX = -speedX;
  }

  x += speedX;
  circle(x, 200, ballSize);
}
```

**Initialize ball position**

**Check if the ball has hit the left/right border**

**Move & draw the ball**



# 🔥 **HW 1**: Bouncing Hello World

- Instructions will be sent by **emails** and released on the **course website**
- You'll explore functions for **text rendering**
- You'll need to figure out how to calculate the **height and width of the text box**
- **The documentation is your friend!**
  - processing.org/reference

# Saving an Animation

- `saveFrame("frames/###.png")`
- Create GIF/MP4 via "**Tools > Movie Maker**"
  - To install Movie Maker
    - Click "Tools > Manage Tools"
    - Click "Movie Maker II | Movie Maker for Processing 4"
    - Click "Install"
- 🐛 There's a bug in the latest release of Movie Maker for Windows
  - Download the ffmpeg executable here
  - Copy "**ffmpeg.exe**" (in the "bin/" folder) to "C:\Users\[USERNAME]\Documents\Processing\tools\MovieMaker\tool"

# Logical Expressions

- `&&` AND
- `||` OR
- `!` NOT

- Examples
  - `(x > 200) && (y > 200)`
  - `(x > 200) || (y > 200)`
  - `!isGameOver`

# Recap

# setup() & draw(): Examples

```
void setup() {
  size(400, 400);
}

void draw() {
  background(50);
  circle(mouseX, mouseY, 10);
}
```

```
void setup() {
  size(400, 400);
  background(50);
}

void draw() {
  circle(mouseX, mouseY, 10);
}
```

# Built-in Mouse Functions: Example

```
void setup() {
  size(400, 400);
  background(50);
}

void draw(){
}
```

Draw by pressing and dragging the mouse

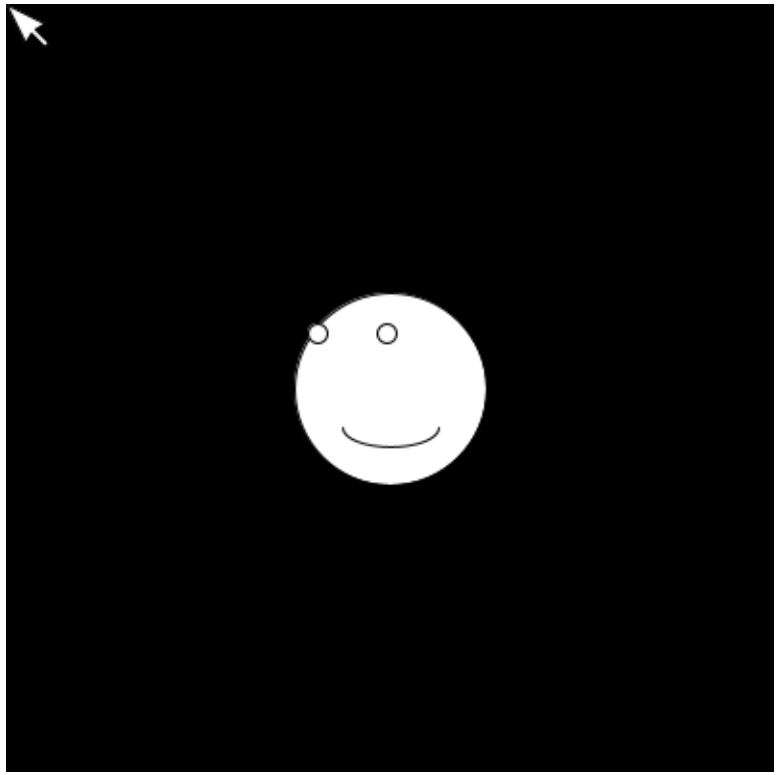
```
void mouseDragged() {
  noStroke();
  circle(mouseX, mouseY, 10);
}
```

Clear everything when the mouse is clicked

```
void mouseClicked() {
  background(50);
}
```

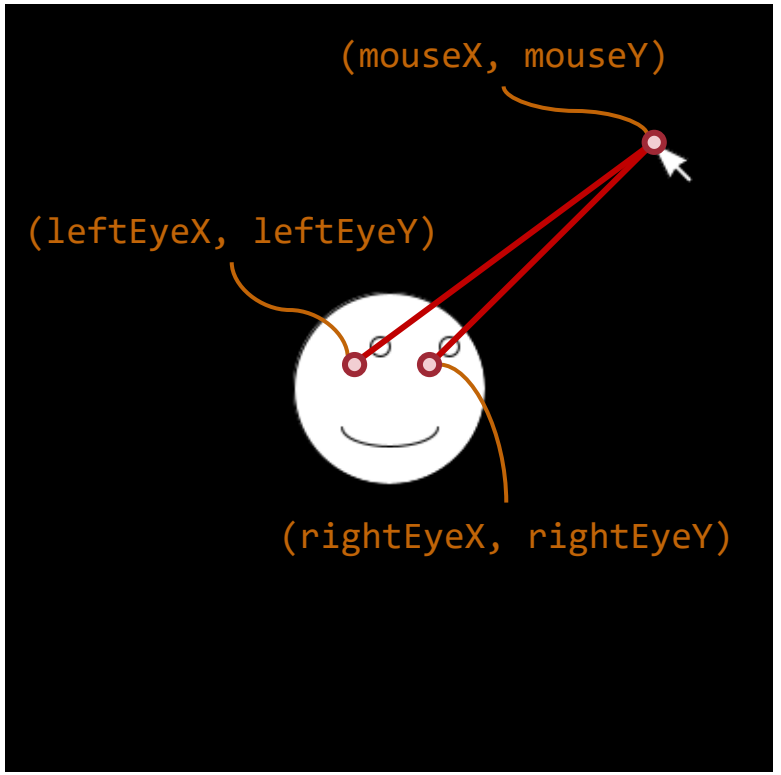
# 🧐 **Exercise**: ~~Creepy~~ Eyes

- Make a simple face where the eyes will **stare at the direction where the mouse is**!
- Hints
  - Start with drawing a **static face**
  - Use `arc()` to get the smile
    - `arc(200, 220, 50, 20, 0, PI)`
  - Use **mouseX** & **mouseY** to find where the mouse is



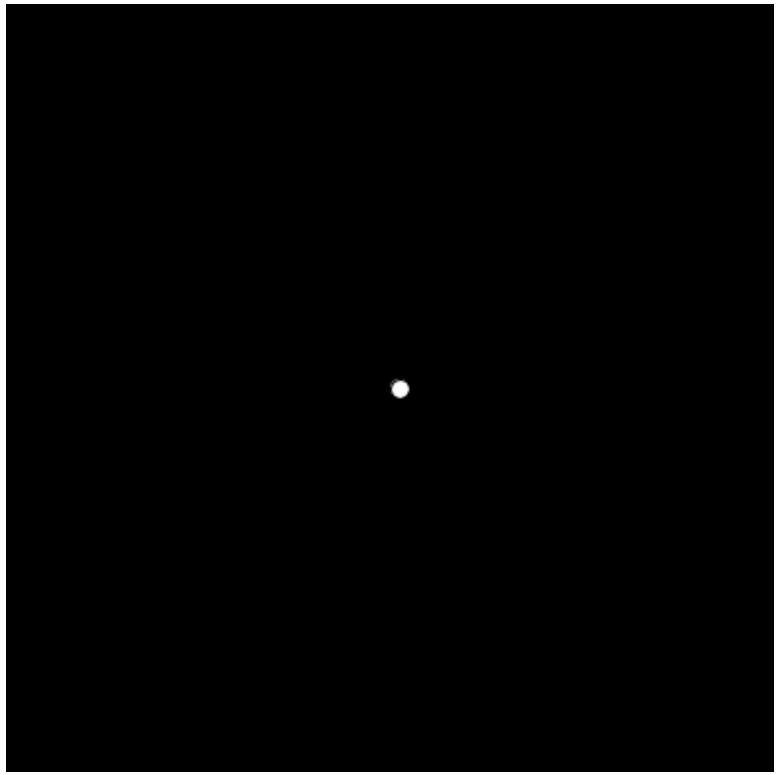
# 🧐 **Exercise**: ~~Creepy~~ Eyes

```
// Calculate the position of the eyes
leftDeltaX = (mouseX – leftEyeX) * scale;
leftDeltaY = (mouseY – leftEyeY) * scale;
rightDeltaX = (mouseX – rightEyeX) * scale;
rightDeltaY = (mouseY – rightEyeY) * scale;

// Draw the eyes
circle(leftEyeX + leftDeltaX, leftEyeY + leftDeltaY, 10);
circle(rightEyeX + rightDeltaX, rightEyeY + rightDeltaY, 10);
```



# First Step: Never-stopping Ball

```
// Current x-position of the ball
float x = 200;

// Current speed of the ball
float speedX = 5;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  // Move the ball
  x = x + speedX;

  // Draw the ball
  circle(x, 200, 10);
}
```



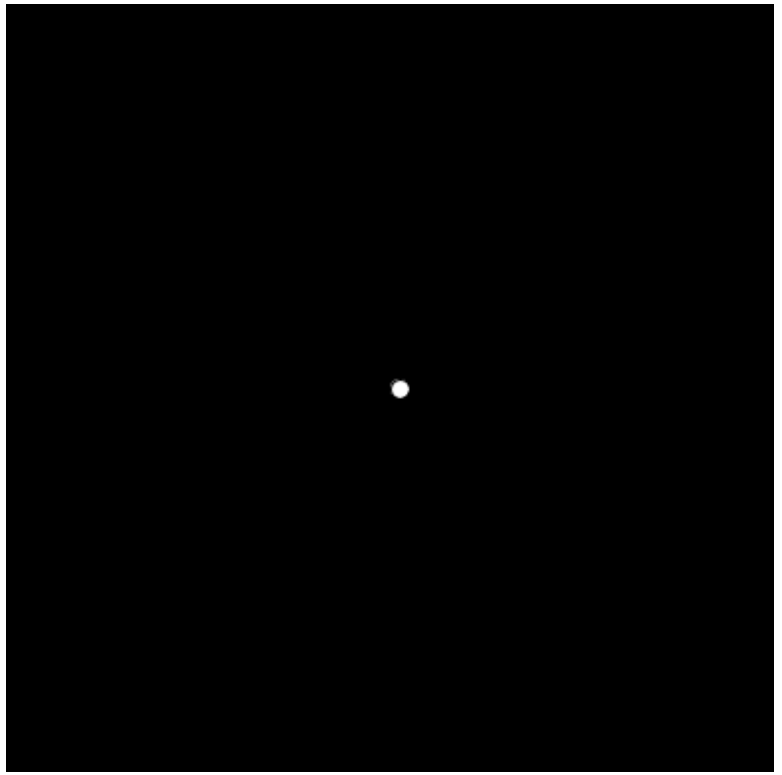
# 🧐 Exercise: Bouncing Ball

```
float ballSize = 10;
float x;
float speedX = 5;

void setup() {
  size(400, 400);
  x = width / 2;
}

void draw() {
  background(0);

  if (x > width - ballSize / 2) {
    speedX = -speedX;
  } else if (x < ballSize / 2) {
    speedX = -speedX;
  }

  x += speedX;
  circle(x, 200, ballSize);
}
```

**Initialize ball position**

**Check if the ball has hit the left/right border**

**Move & draw the ball**



# 🔥 **HW 1**: Bouncing Hello World

- Instructions will be sent by **emails** and released on the **course website**
- You'll explore functions for **text rendering**
- You'll need to figure out how to calculate the **height and width of the text box**
- **The documentation is your friend!**
  - processing.org/reference

## Next Lecture

# Keyboard Controls & Randomness