

PAT 204/504 (Fall 2026)

Creative Coding

Lecture 2: Processing Basics

Instructor: Hao-Wen (Herman) Dong

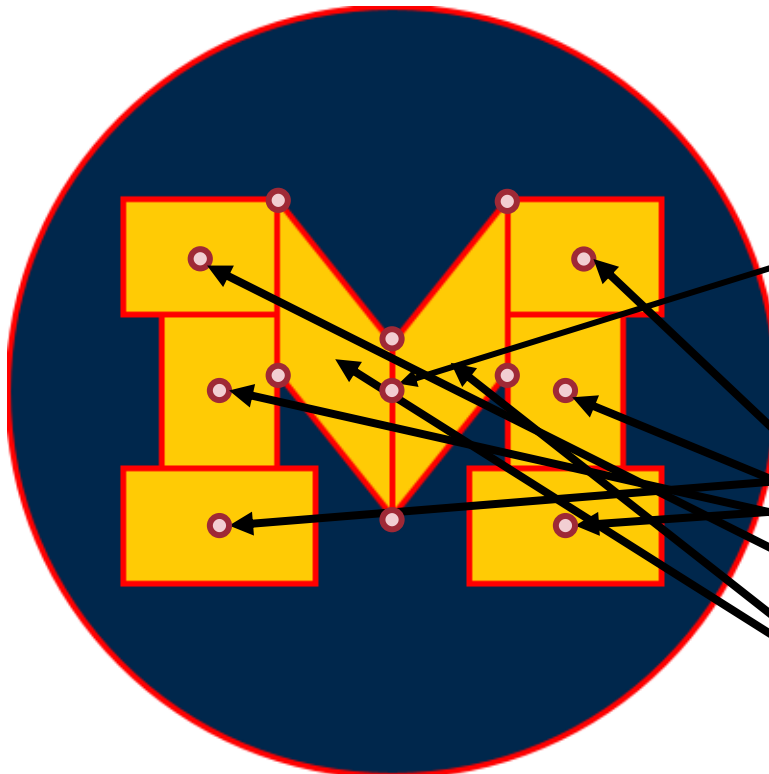
🤔 Exercise: Block M

- Michigan **Blue**: **#00274C**
- Michigan **Maize**: **#FFCB05**
- Functions you'll find useful:
 - `size(width, height)`
 - `background(color)`
 - `noStroke()`
 - `fill(color)`
 - `circle(x, y, diameter)`
 - `rect(x1, y1, x2, y2, x3, y3, x4, y4)`
 - `rectMode(mode) → rectMode(CENTER)`





Exercise: Block M



```
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);  
}
```

More on Shapes

| Strokes

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

| Strokes



`noStroke()`



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

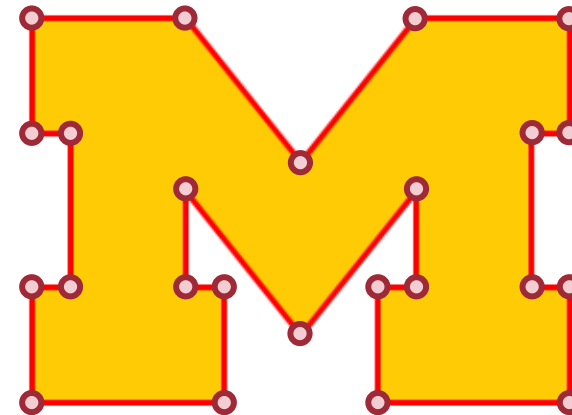
Custom Shapes

```
// 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()



```
beginShape();  
vertex(60, 100);  
...  
vertex(140, 100);  
endShape(CLOSE);
```



```
beginShape();  
vertex(60, 100);  
...  
vertex(140, 100);  
endShape();
```



Exercise: Block M



```
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

Return type Function name No parameters!

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

Comments

Pair of curly brackets

The diagram illustrates the components of a C++ function definition. The code is: `void setup() {
 // Create a 400x400 canvas
 size(400, 400);
}`. Annotations include: 'Return type' pointing to 'void'; 'Function name' pointing to 'setup'; 'No parameters!' pointing to '()'; 'Comments' pointing to the line '// Create a 400x400 canvas'; and 'Pair of curly brackets' pointing to the opening '{' and closing '}' braces.

| 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 is 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);  
}
```

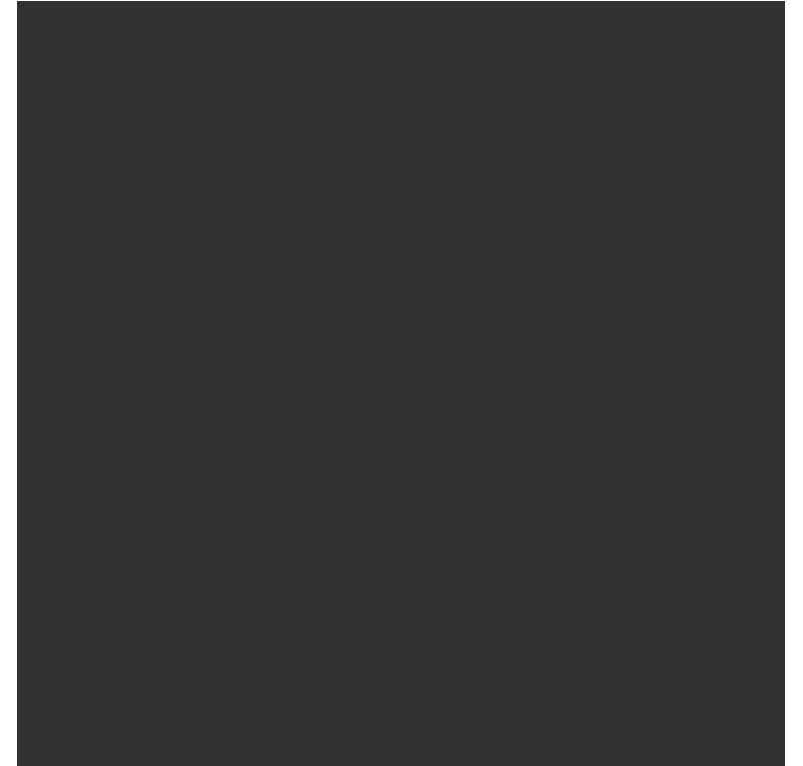
```
void draw(){  
}
```

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
<code>keyPressed()</code>	a key is <i>pressed</i>
<code>keyReleased()</code>	a key is <i>released</i>
<code>keyTyped()</code>	a key is <i>clicked</i> (called repeatedly if held down)

Variables

| Working with a Variable

- Declare → Initialize → Use

Declare → `float x;`

Initialize →

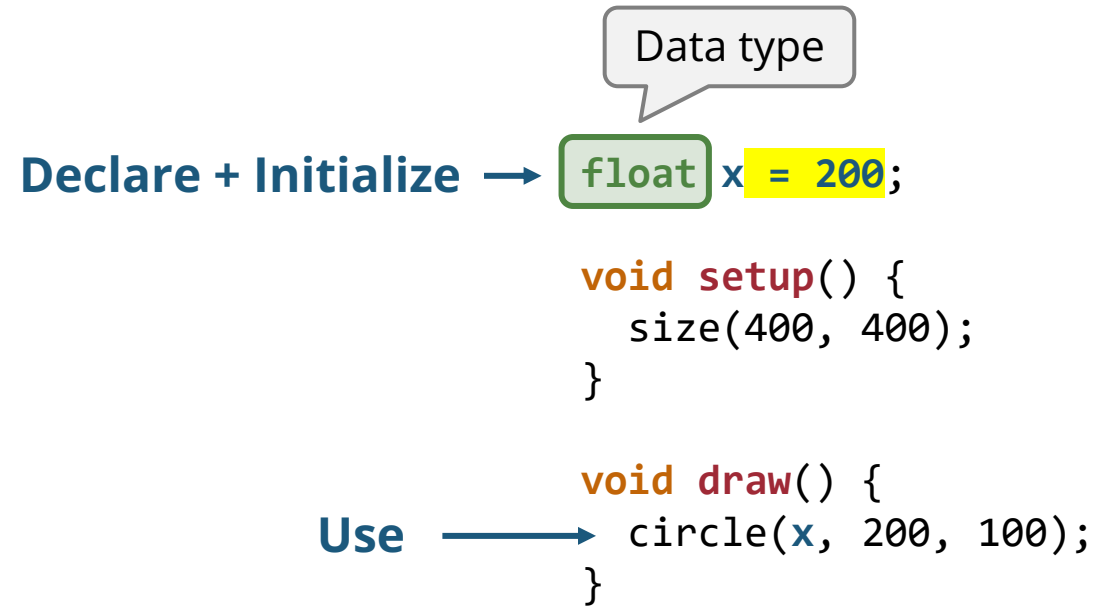
```
void setup() {  
    size(400, 400);  
    x = 200;  
}
```

Use →

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

Declare & Initialize

- Declare + Initialize → Use



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

camelCase

camelCase vs. PascalCase vs. snake_case

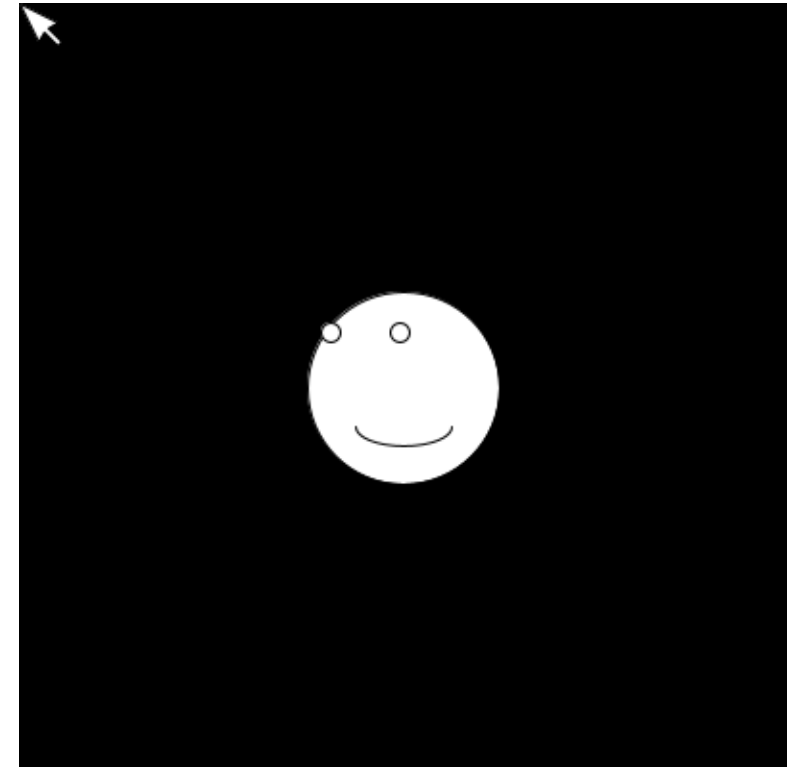
- **Camel case** (camelCase)
 - All lowercase for the first word, uppercase first letters for subsequent words
 - Conventionally used for **variables**, **fields**, and **methods**
- **Pascal case** (PascalCase)
 - Uppercase first letters for all words
 - Conventionally used for **classes**, **interfaces**, and **enums**
- **Snake case** (snake_case)
 - All lowercase letters, words separated by underscores
 - Only used as SCREAMING_SNAKE_CASE for **constants**
- Follow the conventions for **readability!**

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

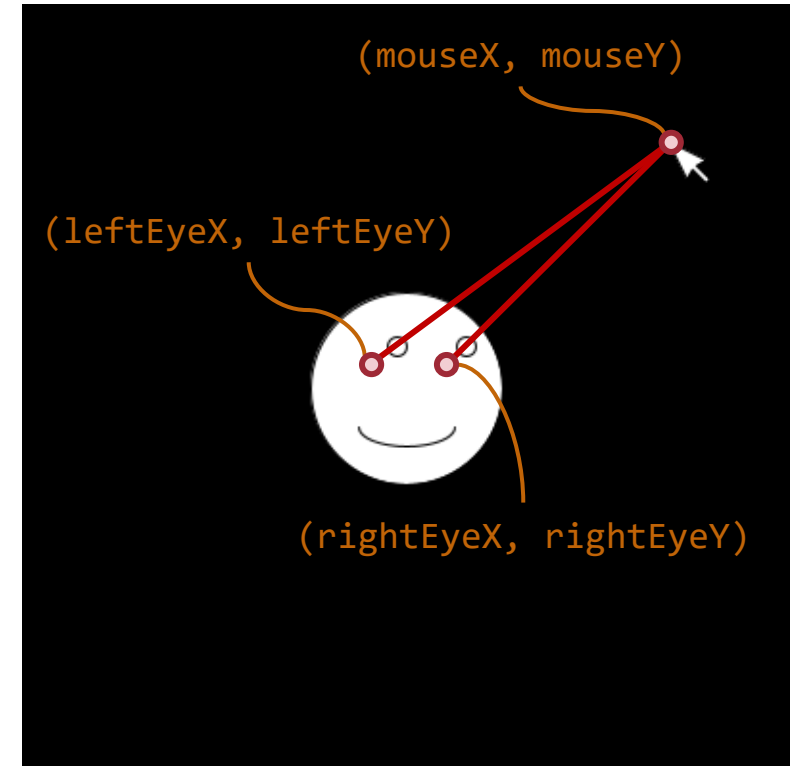
- Make a simple face where the eyes will **stare at the direction where the mouse is!**
- Hints
 - Start with drawing a **static face**
 - Use **arc()** to get the smile
 - `arc(200, 220, 50, 20, 0, PI)`
 - Use **mouseX** & **mouseY** to find where the mouse is



👁️ 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);
```

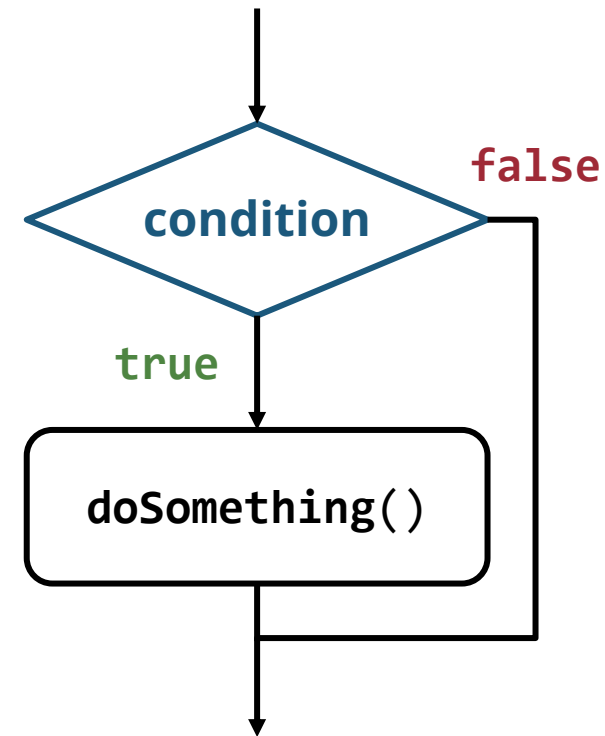


Conditionals

| if Statement

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

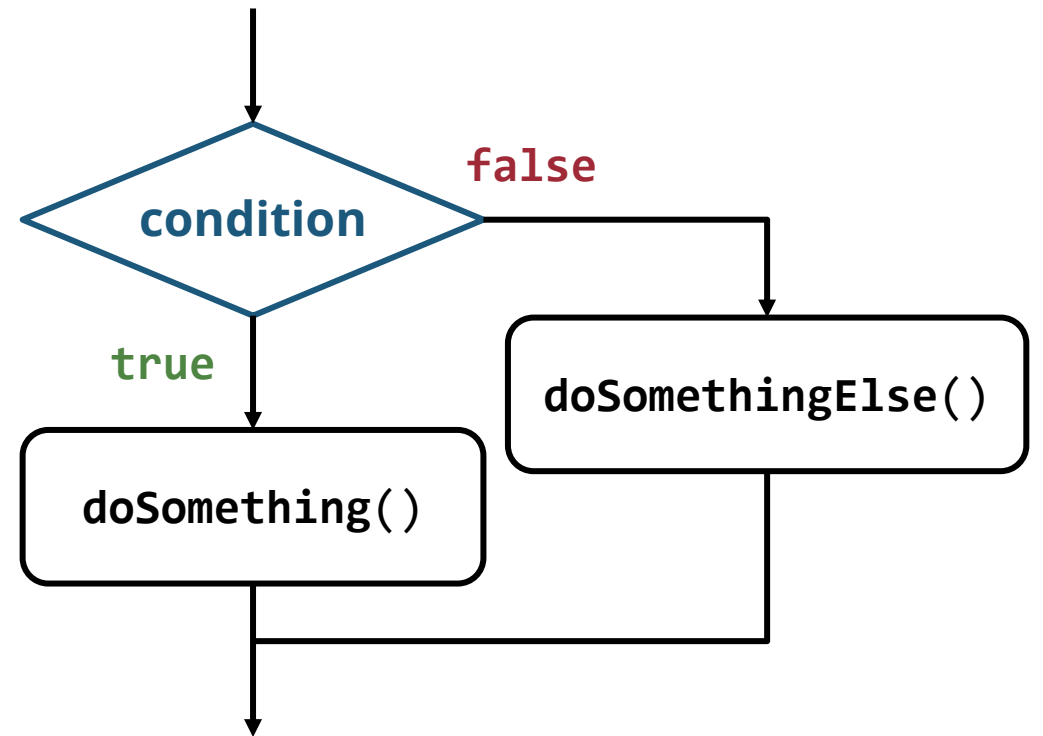
```
if (condition) {  
    doSomething();  
}
```



| if-else Statement

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

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

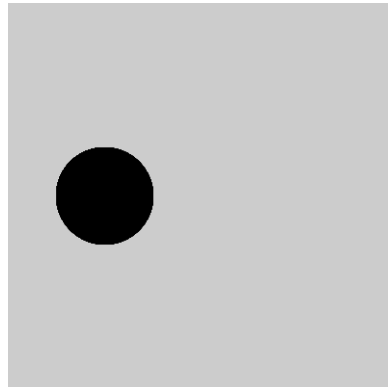


| if-else Statement: Examples

```
float x = 100;
```

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

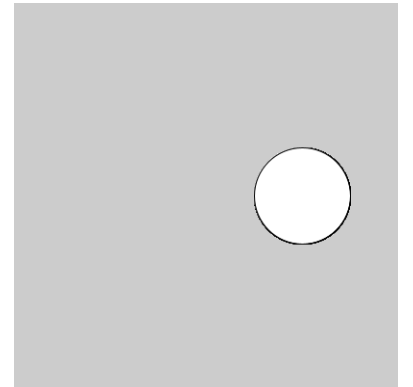
```
void draw() {  
  if (x < 200) {  
    fill(0);  
  } else {  
    fill(255);  
  }  
  circle(x, 200, 100);  
}
```



```
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

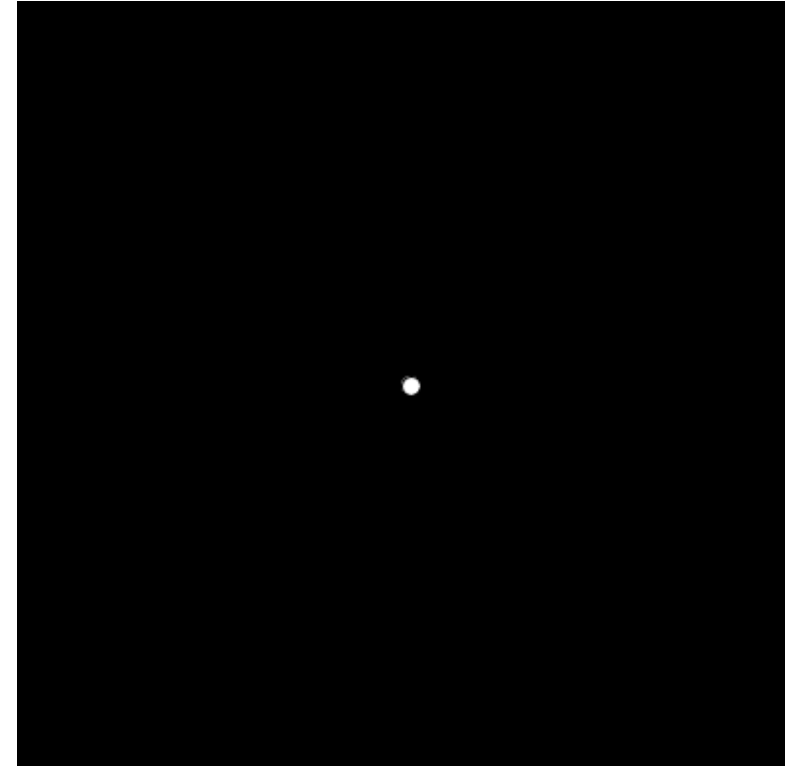
| Logical Expressions

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

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

🤔 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;

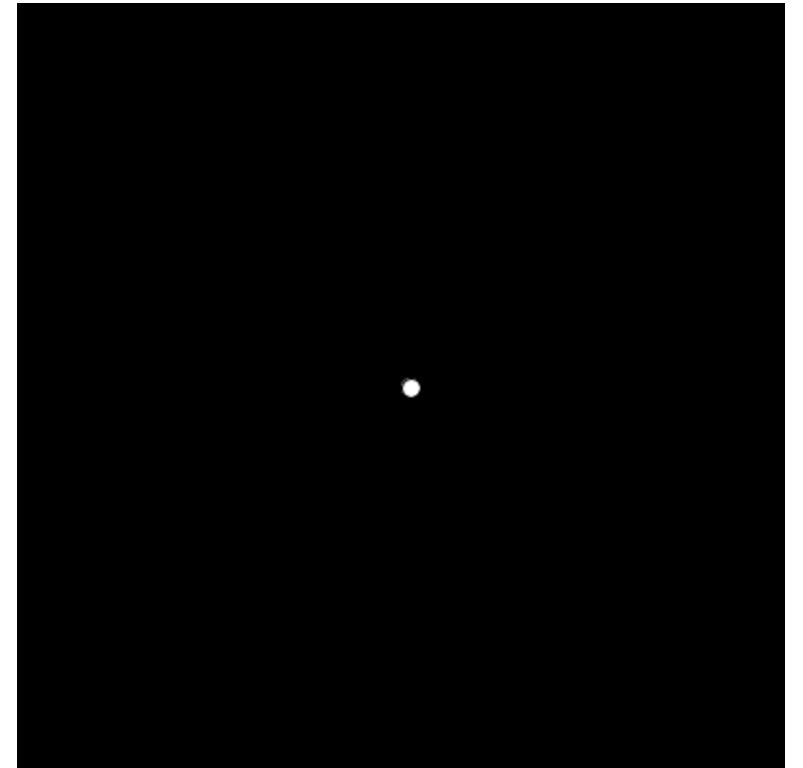
// Current speed of the ball
float speedX = 5;

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

Initialize ball position

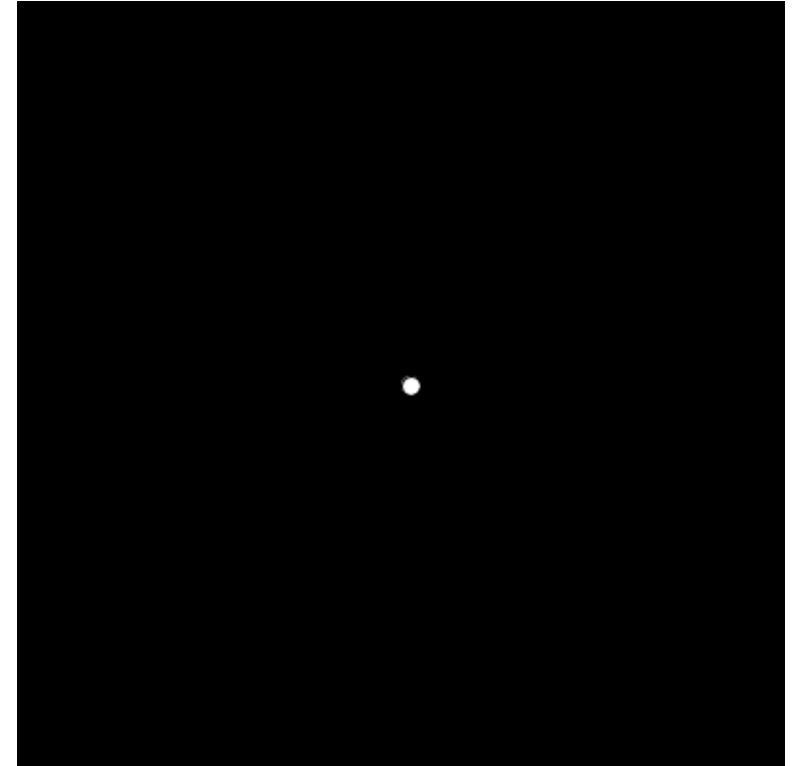
```
float ballSize = 10;  
float x;  
float speedX = 5;  
  
void setup() {  
  size(400, 400);  
  x = width / 2;  
}
```

Check if the ball has hit
the left/right border

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

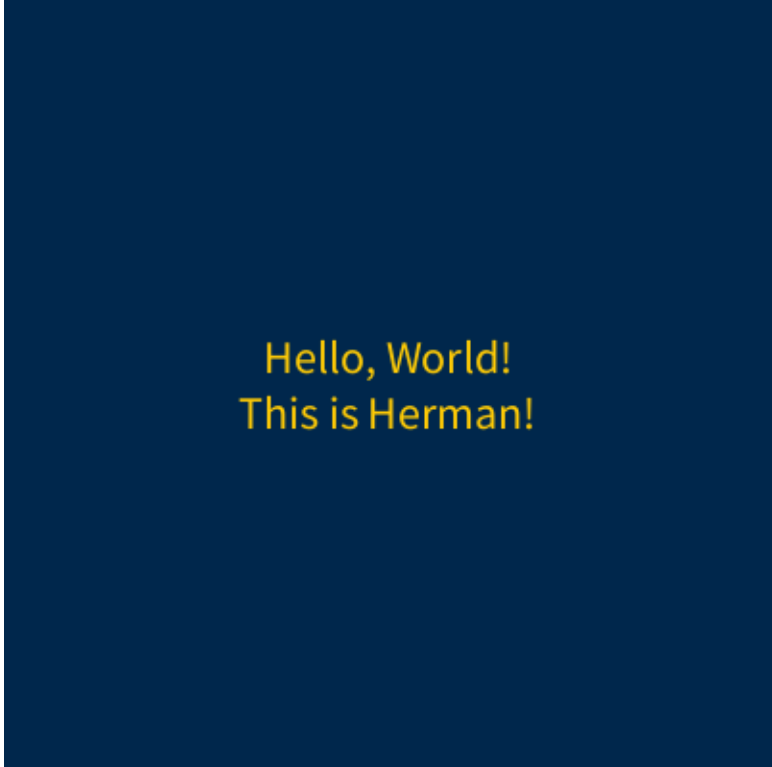
Move & draw the ball

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



🔥 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



Hello, World!
This is Herman!

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"

Recap

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);  
}
```



Built-in Mouse Functions: Example

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

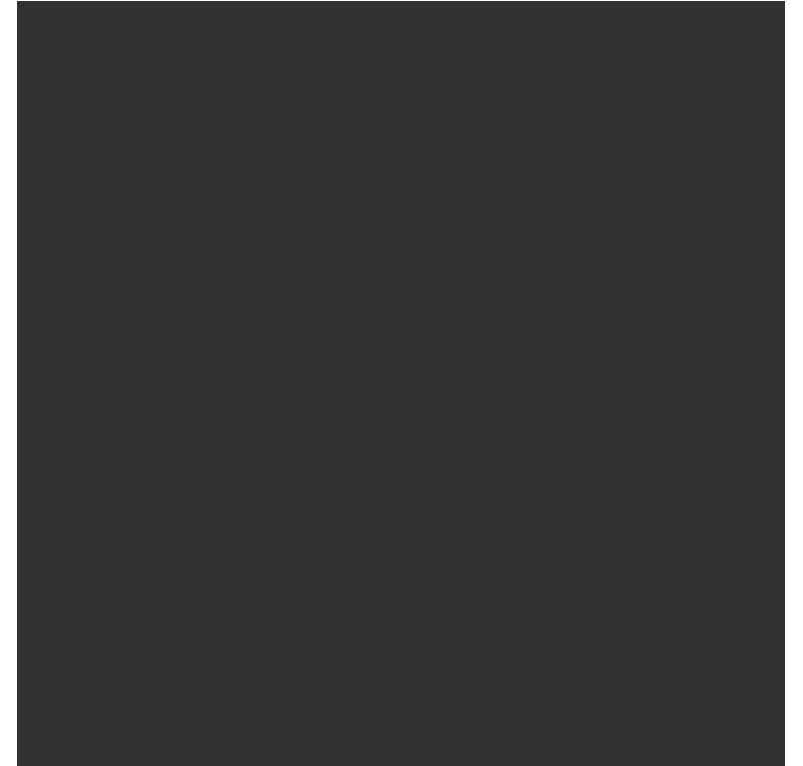
```
void draw(){  
}
```

Draw by pressing and
dragging the mouse

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

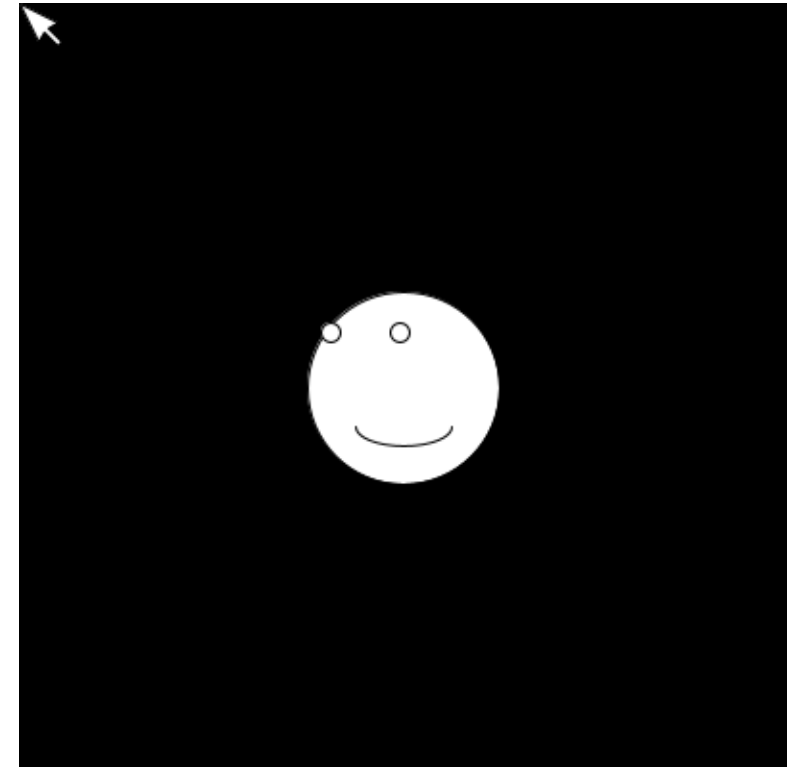
Clear everything when
the mouse is clicked

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



👁️ Exercise: Creepy Eyes

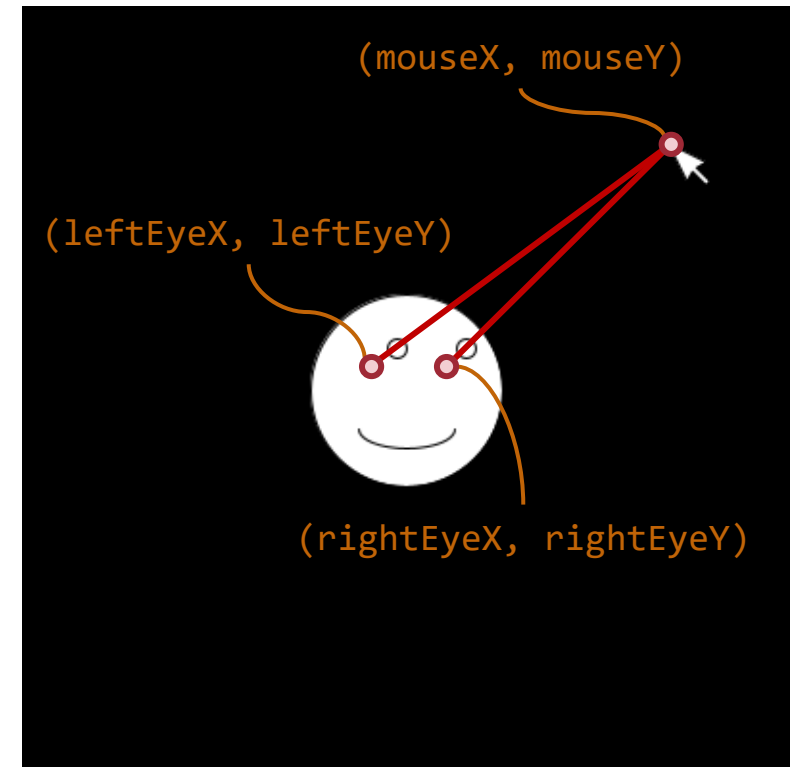
- Make a simple face where the eyes will **stare at the direction where the mouse is!**
- Hints
 - Start with drawing a **static face**
 - Use **arc()** to get the smile
 - `arc(200, 220, 50, 20, 0, PI)`
 - Use **mouseX** & **mouseY** to find where the mouse is



👁️ 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;

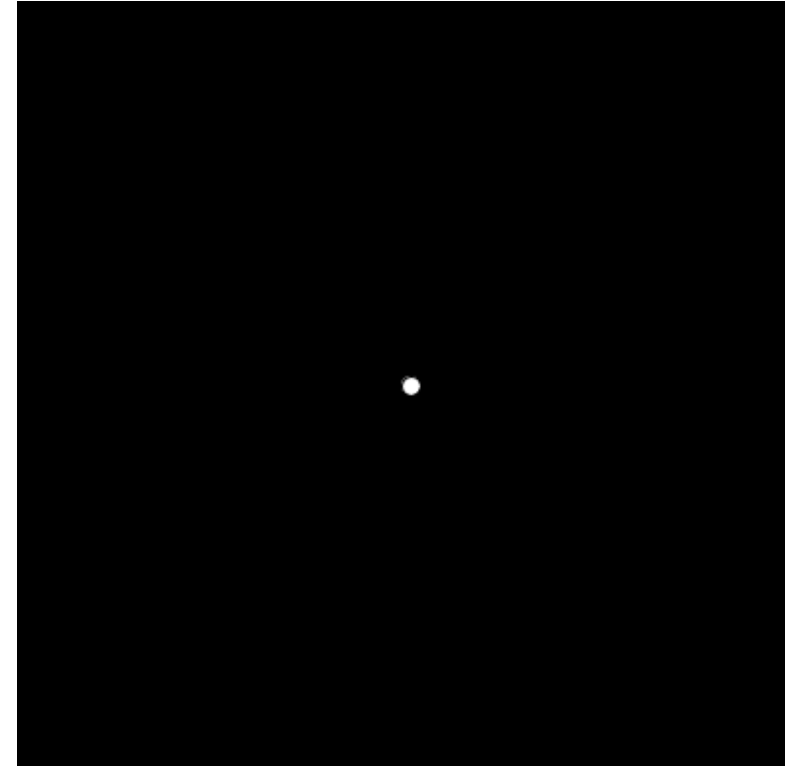
// Current speed of the ball
float speedX = 5;

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

Initialize ball position

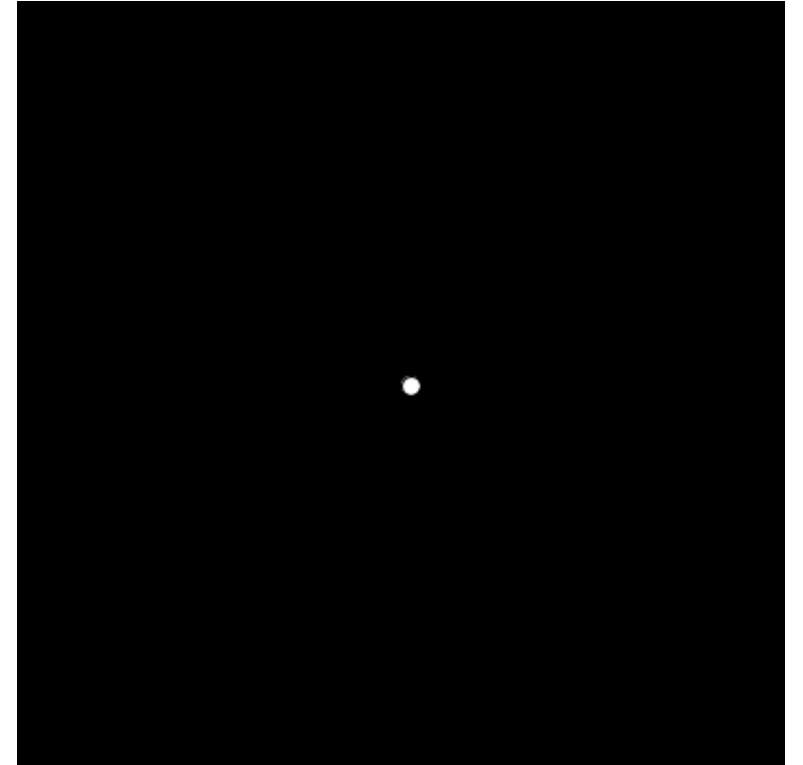
```
float ballSize = 10;  
float x;  
float speedX = 5;  
  
void setup() {  
  size(400, 400);  
  x = width / 2;  
}
```

Check if the ball has hit
the left/right border

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

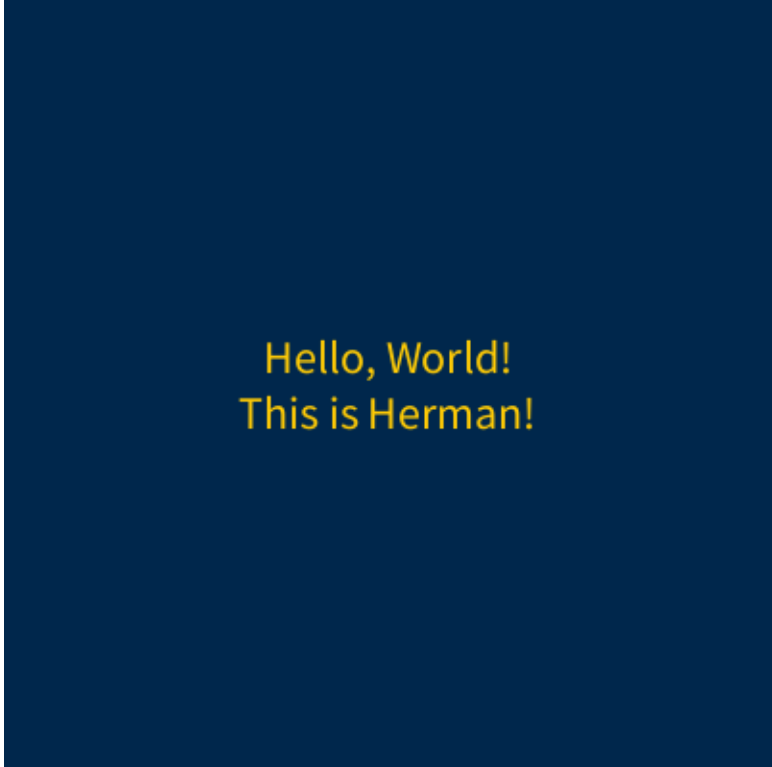
Move & draw the ball

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



🔥 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



Hello, World!
This is Herman!

Next Lecture

Keyboard Controls & Randomness

