PAT 204/504 (Fall 2024)

Creative Coding

Lecture 2: Processing Basics

Instructor: Hao-Wen Dong

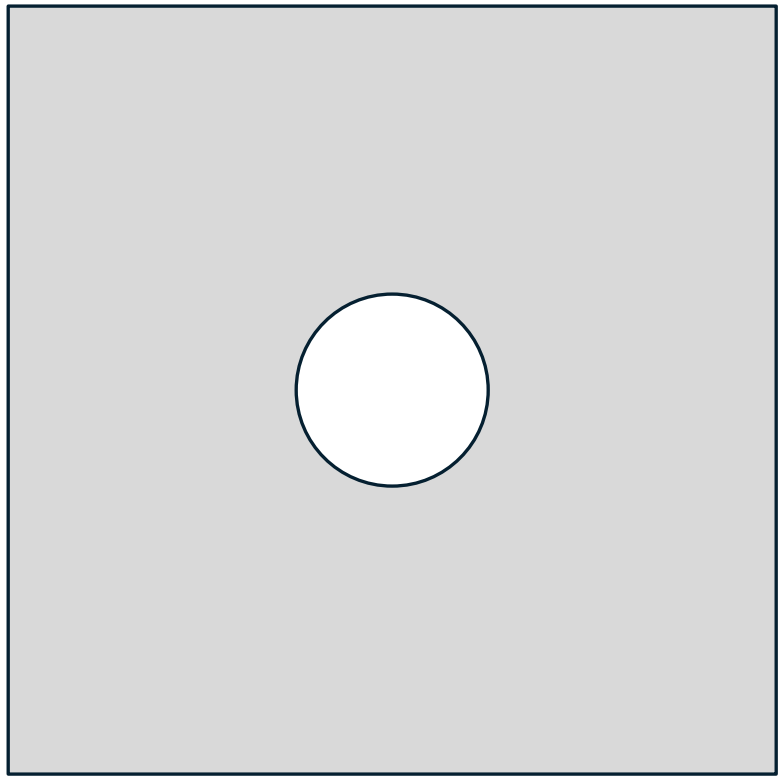
SCHOOL OF MUSIC, THEATRE & DANCE
PERFORMING ARTS TECHNOLOGY
UNIVERSITY OF MICHIGAN

Course Website

- **Main website**: hermandong.com/teaching/pat204_504_fall2024
  - Syllabus, schedule, lecture slides, code examples, etc.
- **Piazza**: Announcements, Q&A
- **Gradescope**: Assignment submission, grading, regrade requests
- **Canvas**: Recordings

(Recap) Your First Processing Sketch

```
size(400, 400);
circle(200, 200, 100);
```



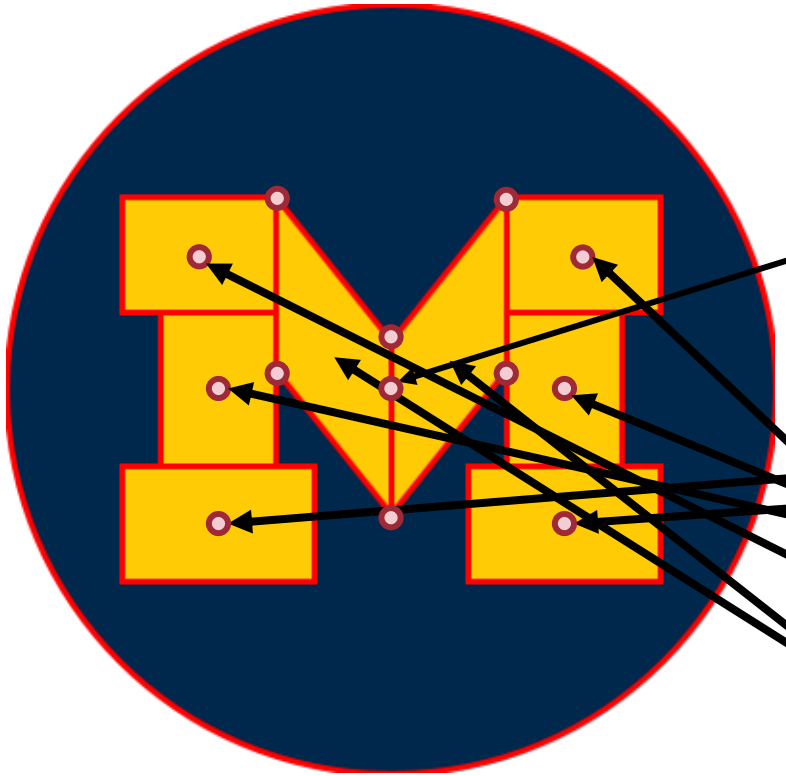
(Recap) More Shapes

- Circle: **`circle(x, y, diameter)`**
- Ellipse: **`ellipse(x, y, width, height)`**
- Square: **`square(x, y, width)`**
- Rectangle: **`rect(x, y, width, height)`**
- Point: **`point(x, y)`**
- Line: **`line(x1, y1, x2, y2)`**
- Triangle: **`triangle(x1, y1, x2, y2, x3, y3)`**
- Quadrilateral: **`quad(x1, y1, x2, y2, x3, y3, x4, y4)`**

(Recap) 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

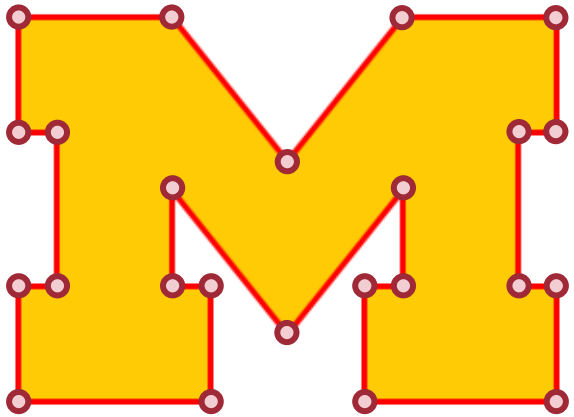
```
noStroke()
```

```
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()`**
  - 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);
}
```

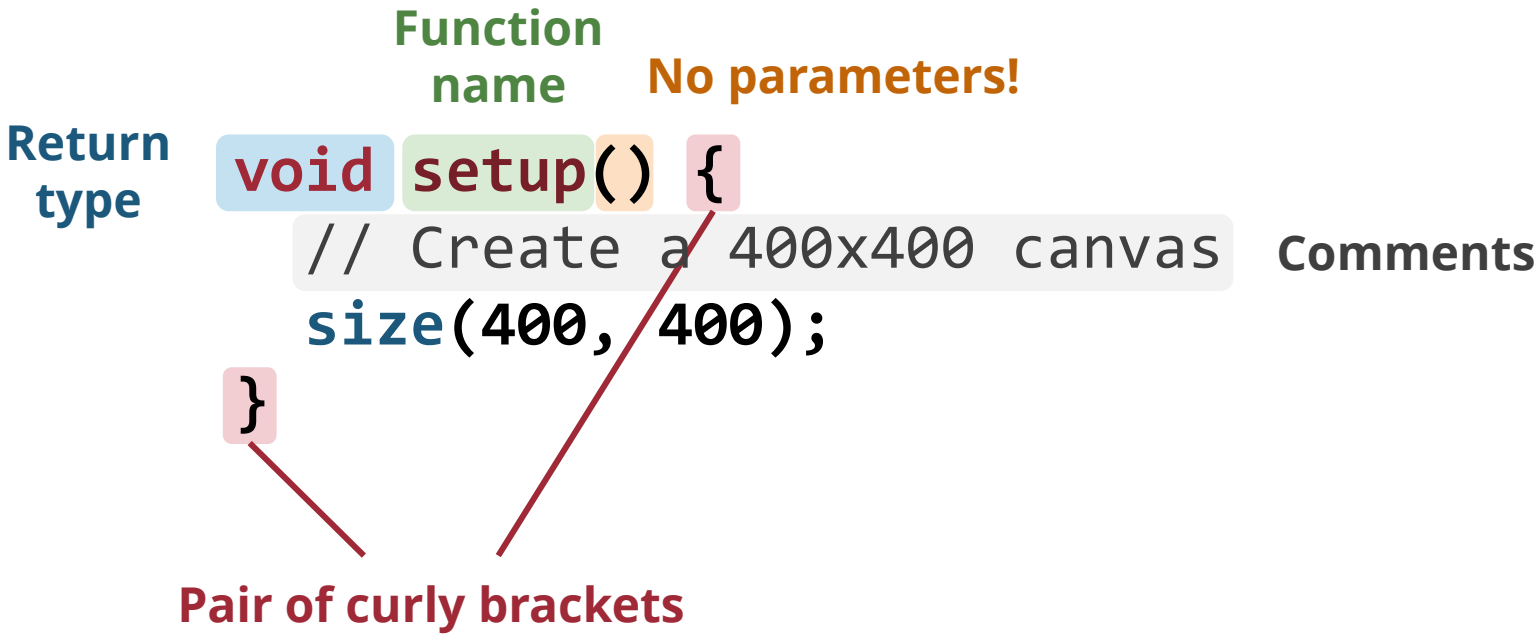
```
void draw() {
  // Set the background color to white
  background(255);

  // 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);
}
```

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!
  - For example, taking screenshots for your assignments!
  - Resetting the whole canvas in an interactive session!

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

Built-in Mouse Functions

	Execute when
• `mousePressed()`	the mouse is *pressed*
• `mouseReleased()`	the mouse is *released*
• `mouseClicked()`	the mouse is *clicked*
• `mouseMoved()`	the mouse is *being moved but not pressed*
• `mouseDragged()`	the mouse is *being moved and pressed*

Built-in Keyboard Functions

	Execute when
• `keyPressed()`	a key is *pressed*
• `keyReleased()`	a key is *released*
• `keyTyped()`	a key is *pressed* (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

```
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`

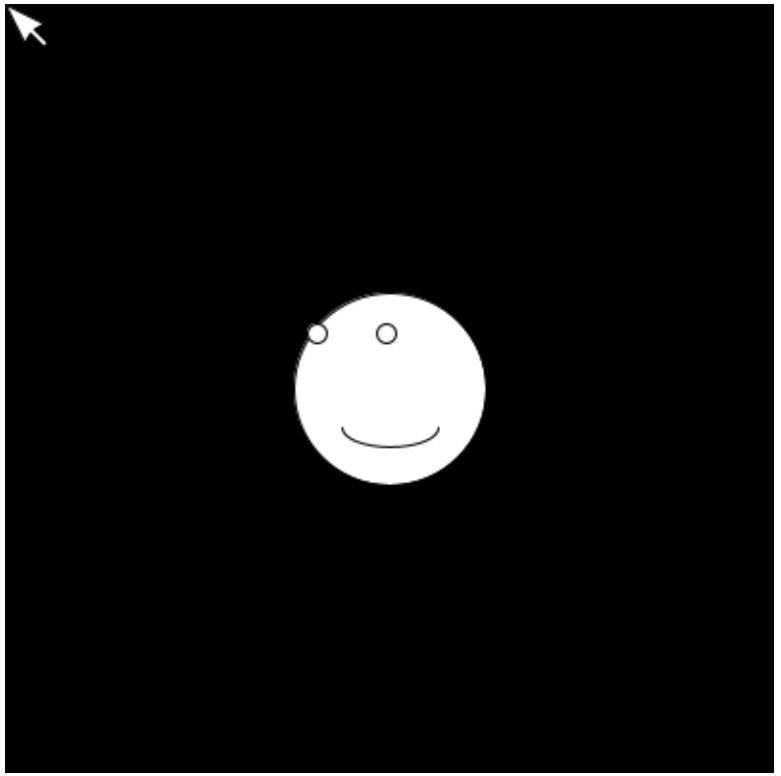
camel case

Built-in Global Variables

- `width`, `height`
- `mouseX`, `mouseY`
- `mousePressed`, `keyPressed`
- `key`, `keycode`
- `rectMode(CENTER)`

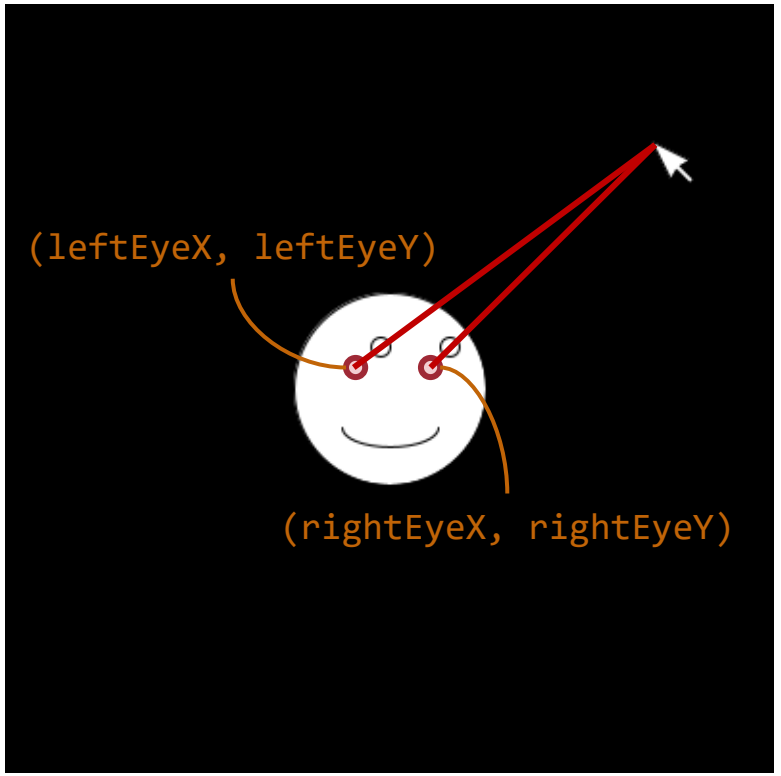
Exercise: ~~Creepy~~ Eyes

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



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
);
```

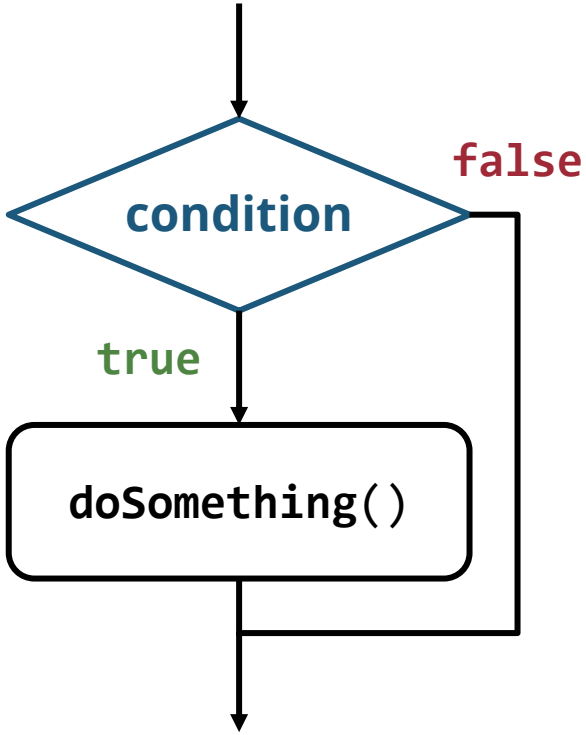


Conditionals

Conditionals – if Statement

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

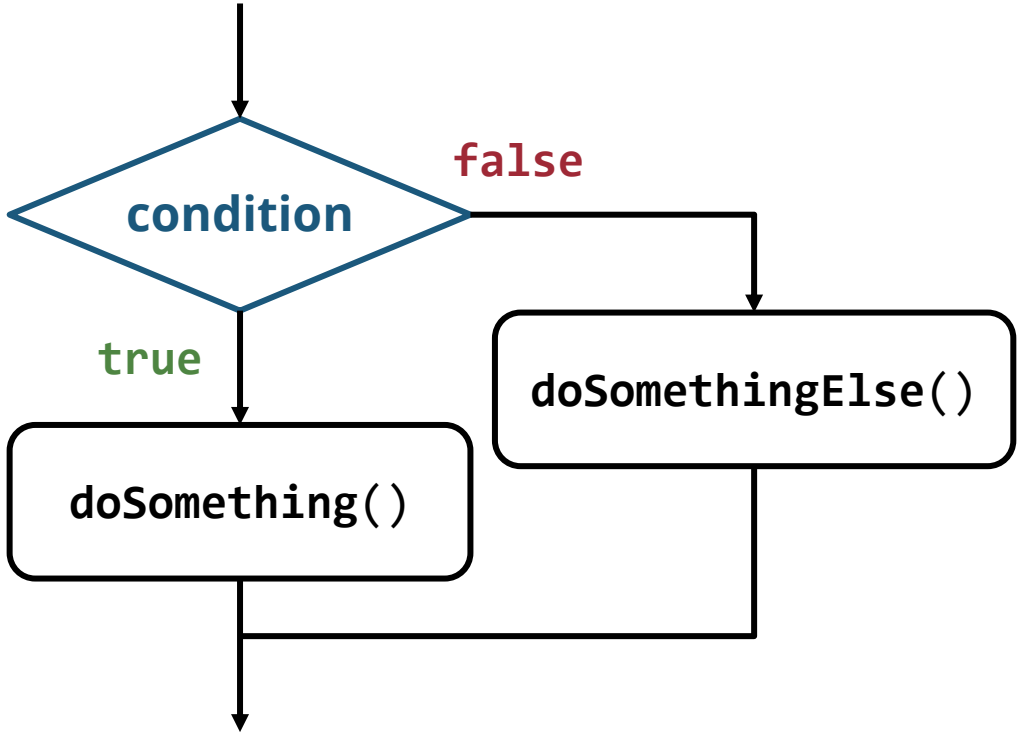
```
if (condition) {
  doSomething();
}
```



Conditionals – if-else Statement

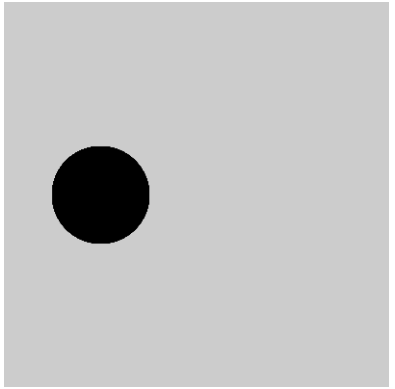
- Control the program flow based on a **condition**

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

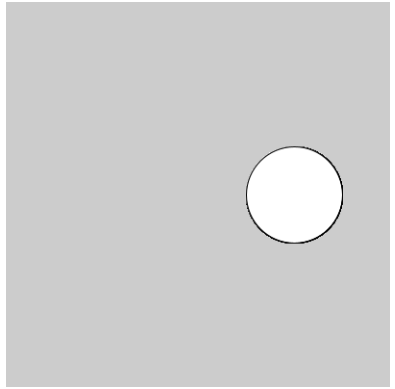


Example: if-else Statement

```
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

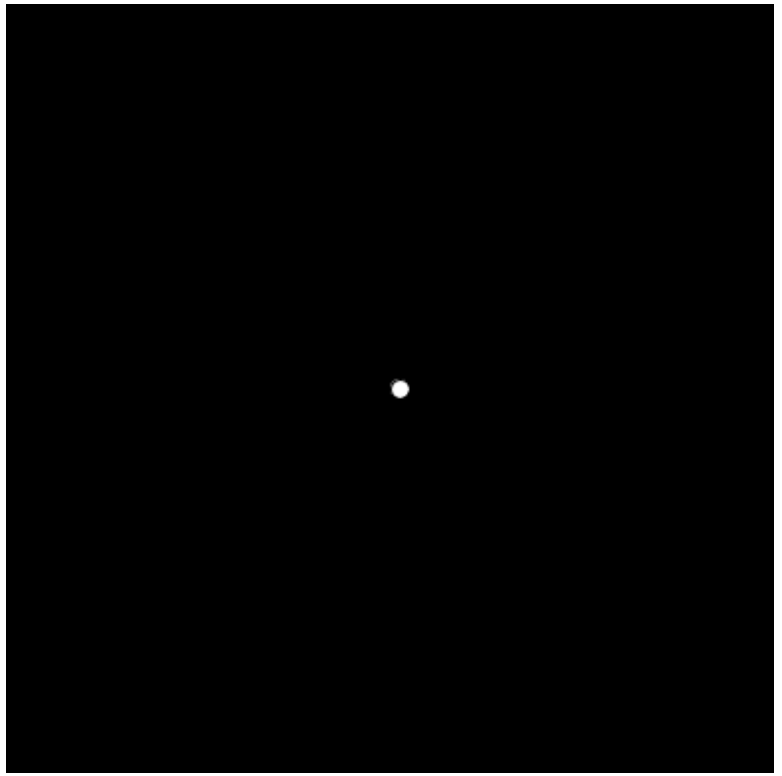
- `>` greater than
- `<` less than

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

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

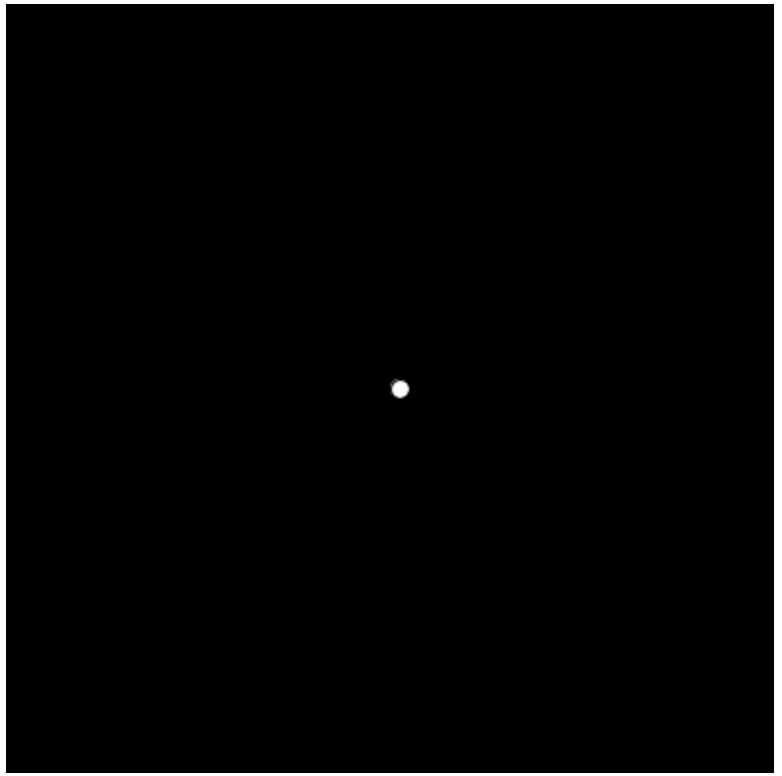
Exercise: Bouncing Ball

- 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);
}
```



Homework 1: Bouncing Hello World

- Instructions will be released on Gradescope
- You need to find the function for **text rendering**
  - The documentation is your friend!
  - https://processing.org/reference
- You need to figure out how to calculate the **height and width of the text box**
  - There'll be many friendly hints in the instructions 😊
- Due at **11:59pm ET** on **September 6**
- Late submissions: **1 point deducted per day**

Saving an Animation

- `saveFrame("frames/###.png")`
- Create GIF/MP4 via "**Tools > Movie Maker**"

Logical Expressions

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

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

Keyboard Controls

More about Keyboard Controls

- What does the following function do?

```
void keyPressed() {
  if (key == ' ') {
    saveFrame("screenshot.png");
  }
}
```

Key & KeyCode

- `Key` stores the most recent key used (either pressed or released)
- For special (non-ASCII) keys, `KeyCode` is used
  - UP, DOWN, LEFT, RIGHT
  - ALT, CONTROL, SHIFT
  - You don’t need `KeyCode` for BACKSPACE, TAB, ENTER, RETURN, ESC, and DELETE

KeyCode

- What does the following function do?

```
void keyPressed() {
  if (key == CODED) {
    if (keyCode == UP) {
      noLoop();
    }
  }
}
```

KeyCode

- How about this?

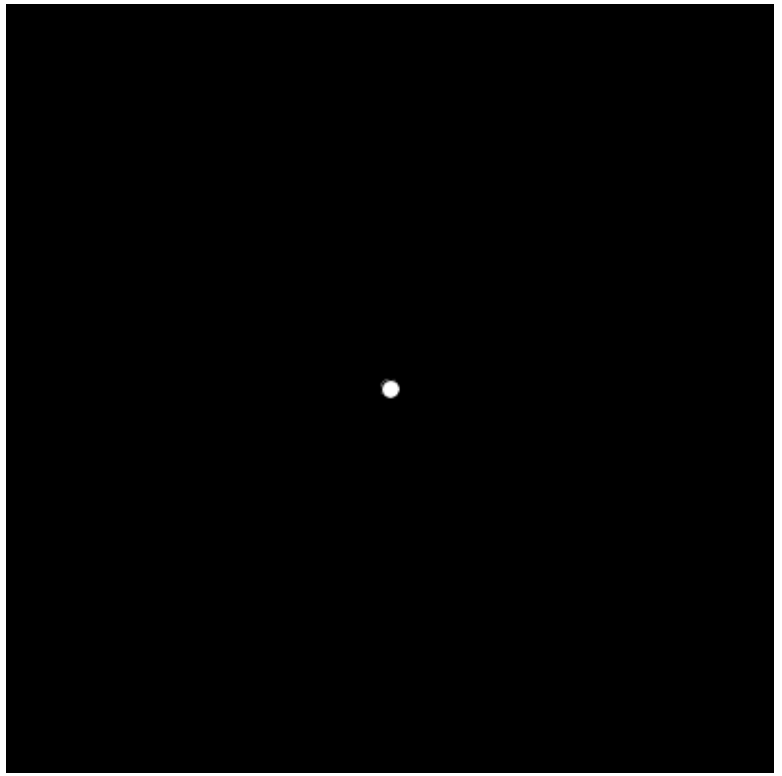
```
void keyPressed() {
  if (key == CODED) {
    if (keyCode == BACKSPACE) {
      noLoop();
    }
  }
}
```

Conditionals – if-else-if Statement

```
if (condition) {
  doSomething();
} else if (condition2) {
  doSomethingElse();
} else {
  doYetSomethingElse();
}
```

Example: Use Arrow Keys to Control a Ball

```
float x = 200;
float y = 200;
float step = 10;

void keyPressed() {
  if (key == CODED) {
    if (keyCode == LEFT) {
      x = x - step;
    } else if (keyCode == RIGHT) {
      x = x + step;
    } else if (keyCode == UP) {
      y = y - step;
    } else if (keyCode == DOWN) {
      y = y + step;
    }
  }
}
```



Switch Statement

```
switch(expression) {
  case value1:
    doSomething();
    break;

  case value2:
    doSomethingElse();
    break;

  case value2:
    doYetSomethingElse();
    break;
}
```

Switch Statement

```
if (keyCode == LEFT) {
  x = x - step;
} else if (keyCode == RIGHT) {
  x = x + step;
} else if (keyCode == UP) {
  y = y - step;
} else if (keyCode == DOWN) {
  y = y + step;
}
```

```
switch(keyCode) {
  case LEFT:
    x = x - step;
    break;

  case RIGHT:
    x = x + step;
    break;

  case UP:
    y = y - step;
    break;

  case DOWN:
    y = y + step;
    break;
}
```