PAT 204/504 (Fall 2024)

Creative Coding

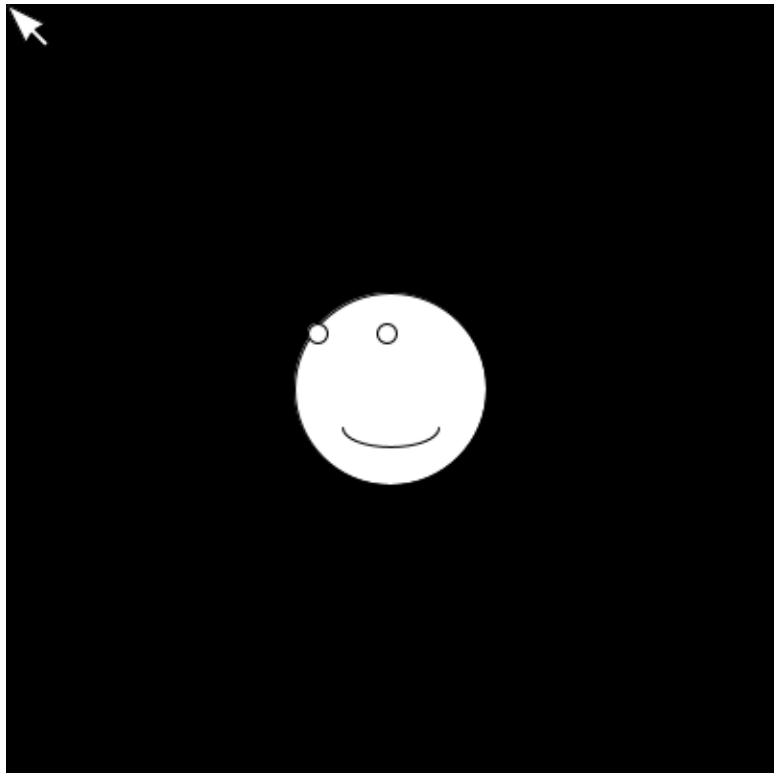
Lecture 3: Randomness & Keyboard Controls

Instructor: Hao-Wen Dong

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

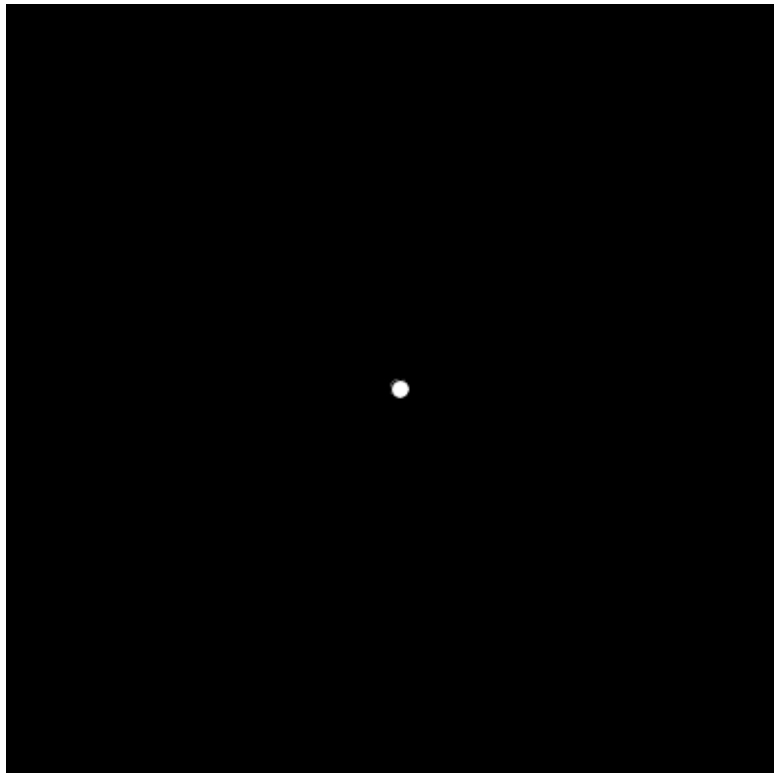
(Recap) ~~Creepy~~ Eyes

```
float leftEyeX = 180, leftEyeY = 190;
float rightEyeX = 220, rightEyeY = 190;
float scale = 0.1;

float leftDeltaX, leftDeltaY;
float rightDeltaX, rightDeltaY;

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

void draw() {
  background(0);

  // Draw the face
  circle(200, 200, 100);

  // Draw the smile
  arc(200, 220, 50, 20, 0, PI);

  // Calculate the positions 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);
}
```



(Recap) Bouncing Ball

```
float ballSize = 10;  // Size of the ball
float x;  // Current x-position of the ball
float speedX = 5;  // Current speed of the ball
boolean saveFrames = false;

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

  // Initialize the ball position
  x = width / 2;
}


void draw() {
  // Create a black background
  background(0);

  // Check if the ball hits the left/right border
  if (x > width - ballSize / 2) {
    speedX = -speedX;
  } else if (x < ballSize / 2) {
    speedX = -speedX;
  }

  // Move the ball
  x += speedX;

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

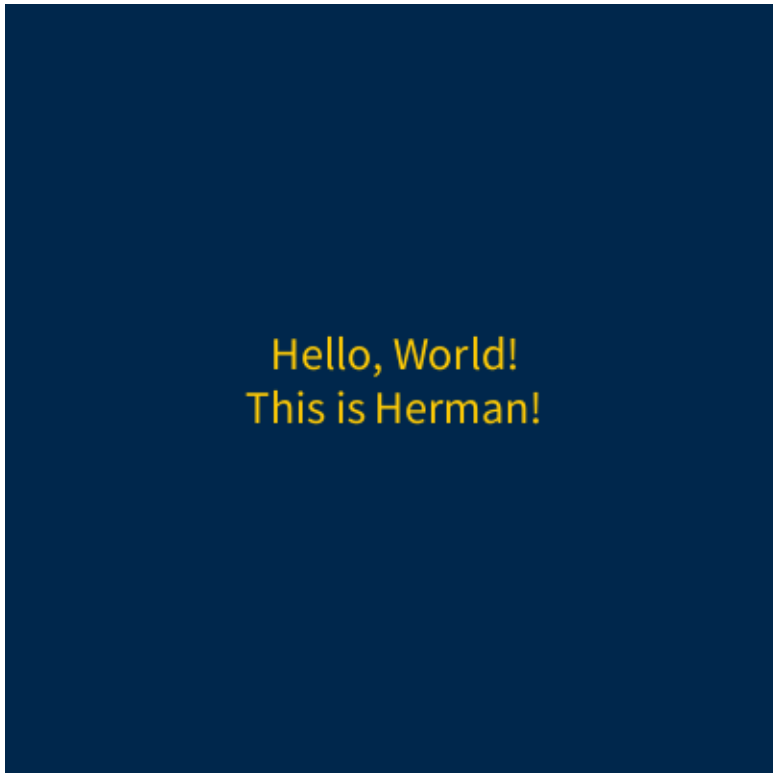


Homework 1: Bouncing Hello World

- 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**

Bonus: Controlling Bouncing Hello World

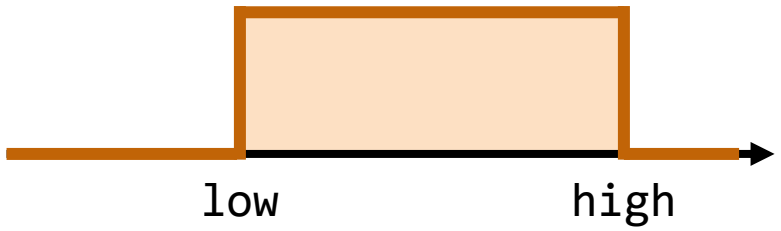
- Let's add some ***controls*** and ***randomness*** at the same time!
- Make it
  - **move toward a *random* direction** at a preset speed when the program starts
  - when the mouse is clicked, **move toward the direction where the cursor is at** a preset speed

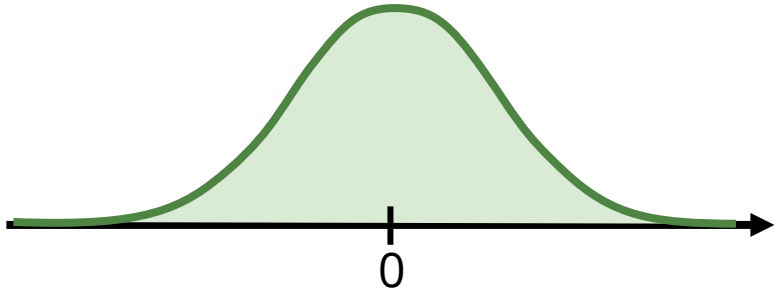


Randomness

Randomness

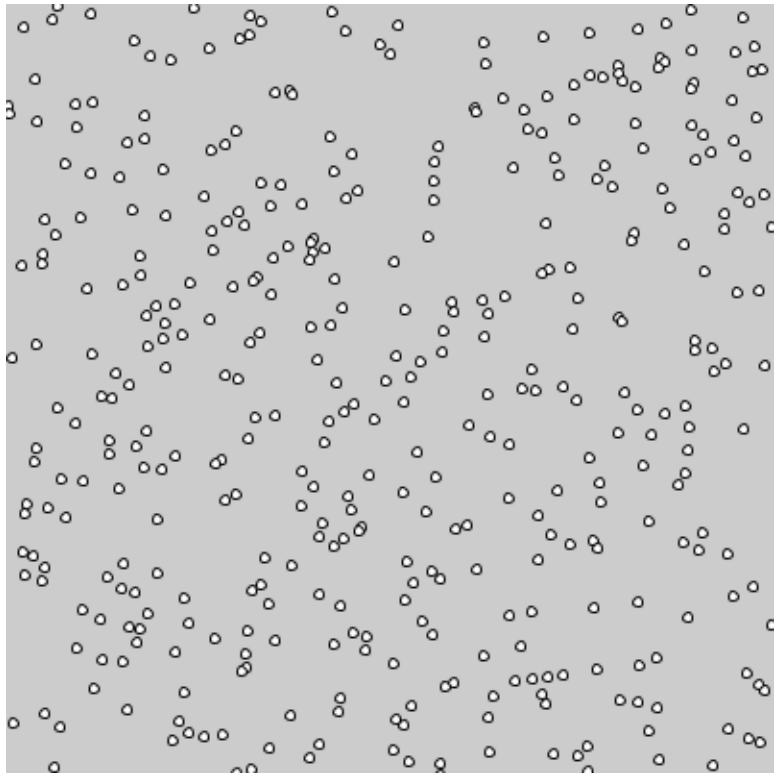
- `random(high)` Generate a random number in $U[0, high]$
- `random(low, high)` Generate a random number in $U[low, high]$
- `randomGaussian()` Generate a random number in $N[0, 1]$





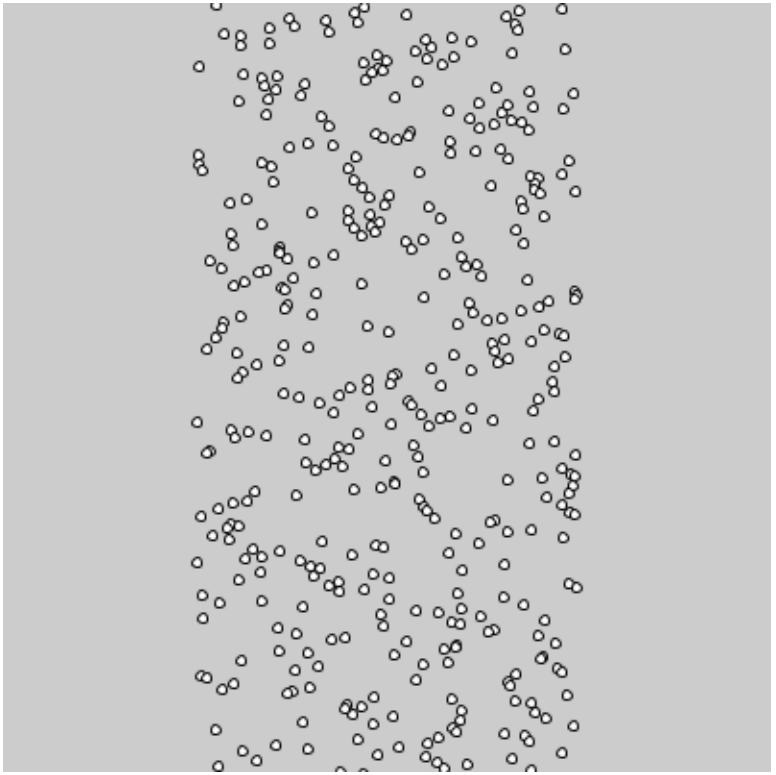
random()

```
float x, y;

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

void draw() {
  // Generate a random number
  x = random(400);

  // Gradually increase the y-position
  y = y + 1;

  // Draw a circle
  circle(x, y, 5);
}
```

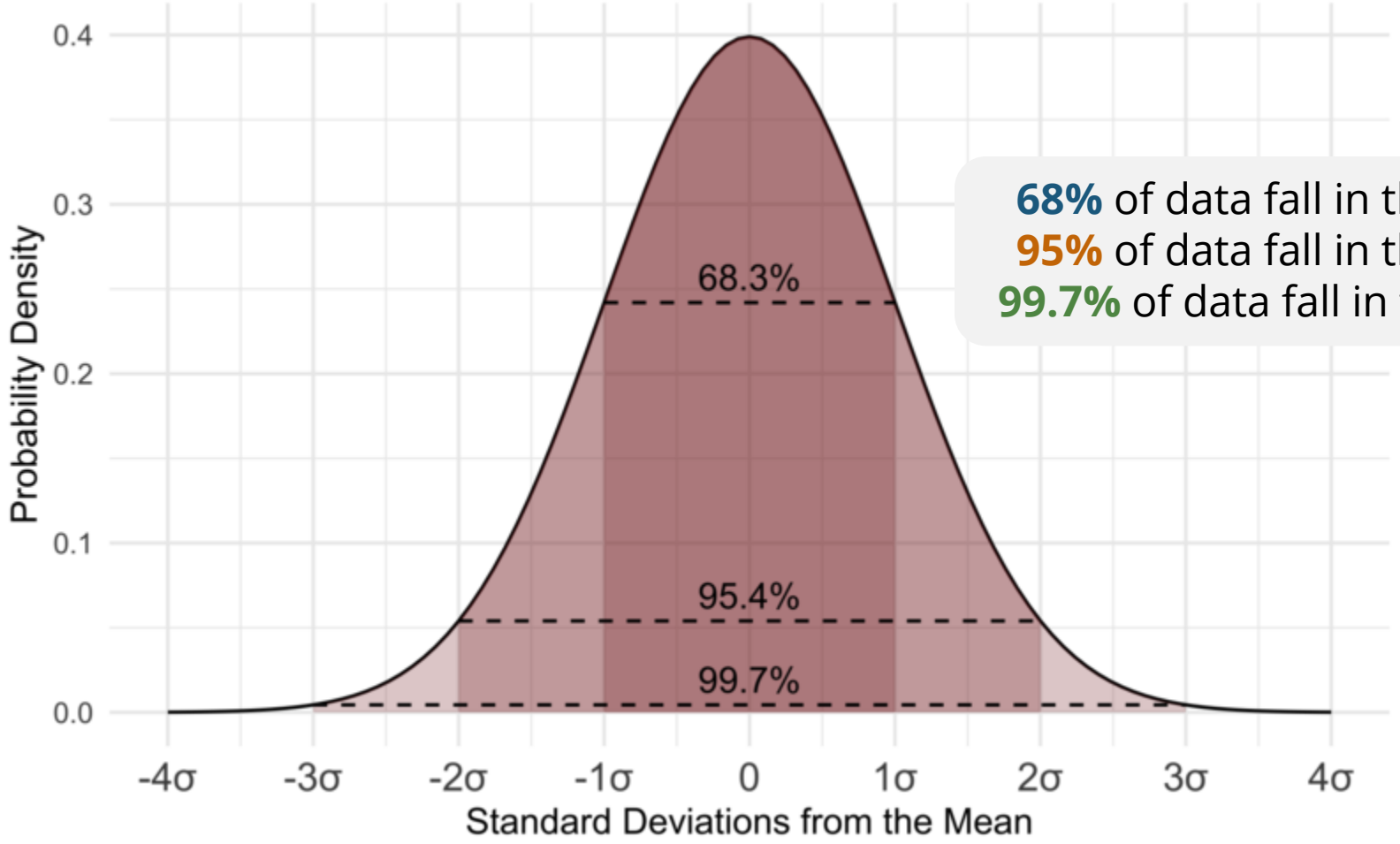


random()

```
float x, y;

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

void draw() {
  // Generate a random number
  x = random(100, 300);

  // Gradually increase the y-position
  y = y + 1;

  // Draw a circle
  circle(x, y, 5);
}
```



Gaussian Distribution & the 68–95–99.7 Rule



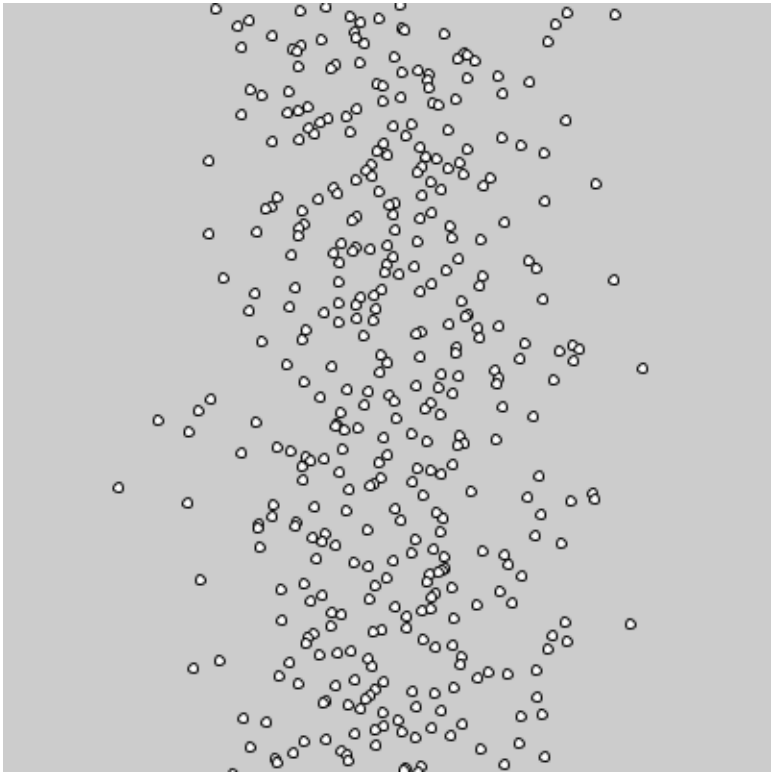
68% of data fall in the **±1σ** range
95% of data fall in the **±2σ** range
99.7% of data fall in the **±3σ** range

randomGaussian()

```
float x, y;

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

void draw() {
  // Generate a random number
  x = randomGaussian() * 50 + 200;

  // Gradually increase the y-position
  y = y + 1;

  // Draw a circle
  circle(x, y, 5);
}
```

$\sigma = 50$

Centered at 200

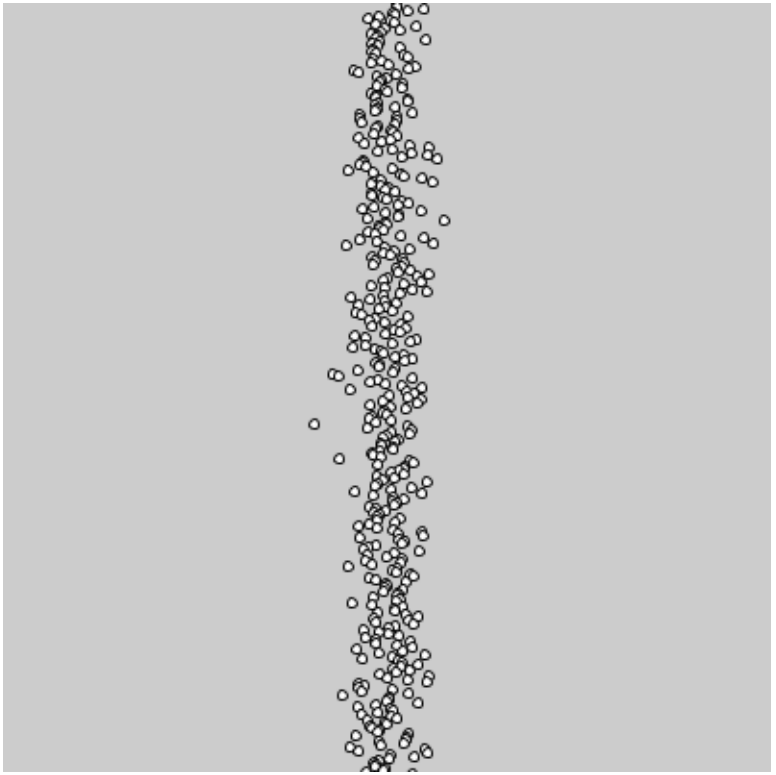


randomGaussian()

```
float x, y;

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

void draw() {
  // Generate a random number
  x = randomGaussian() * 10 + 200;

  // Gradually increase the y-position
  y = y + 1;

  // Draw a circle
  circle(x, y, 5);
}
```

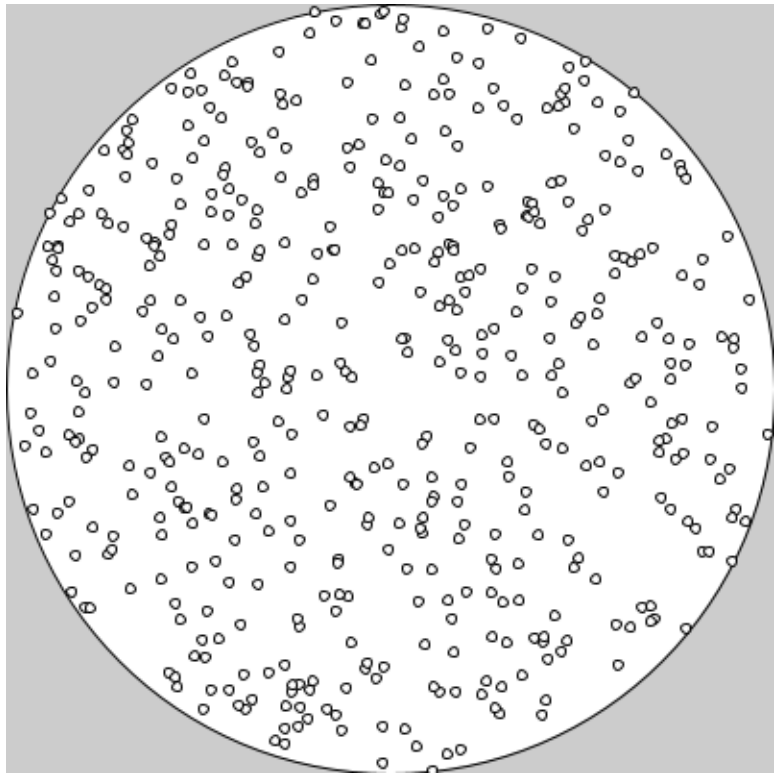
$\sigma = 10$

Centered at 200



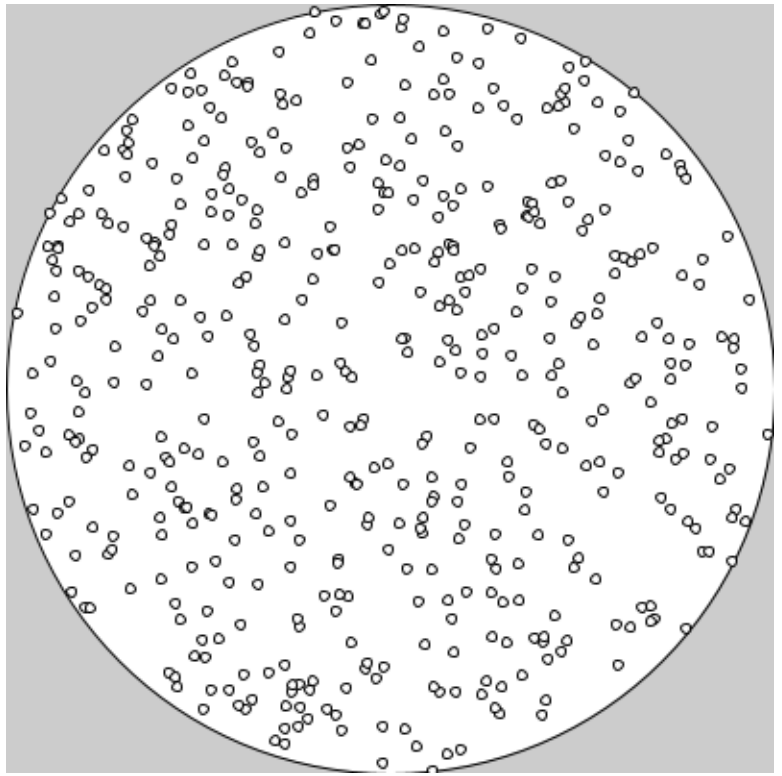
Exercise: Random Points in a Circle

- How to randomly **sample a point in a circle**?
- Two strategies:
  - Sample randomly and **keep only those fall in the circle**
  - Sample points that **always fall in the circle**

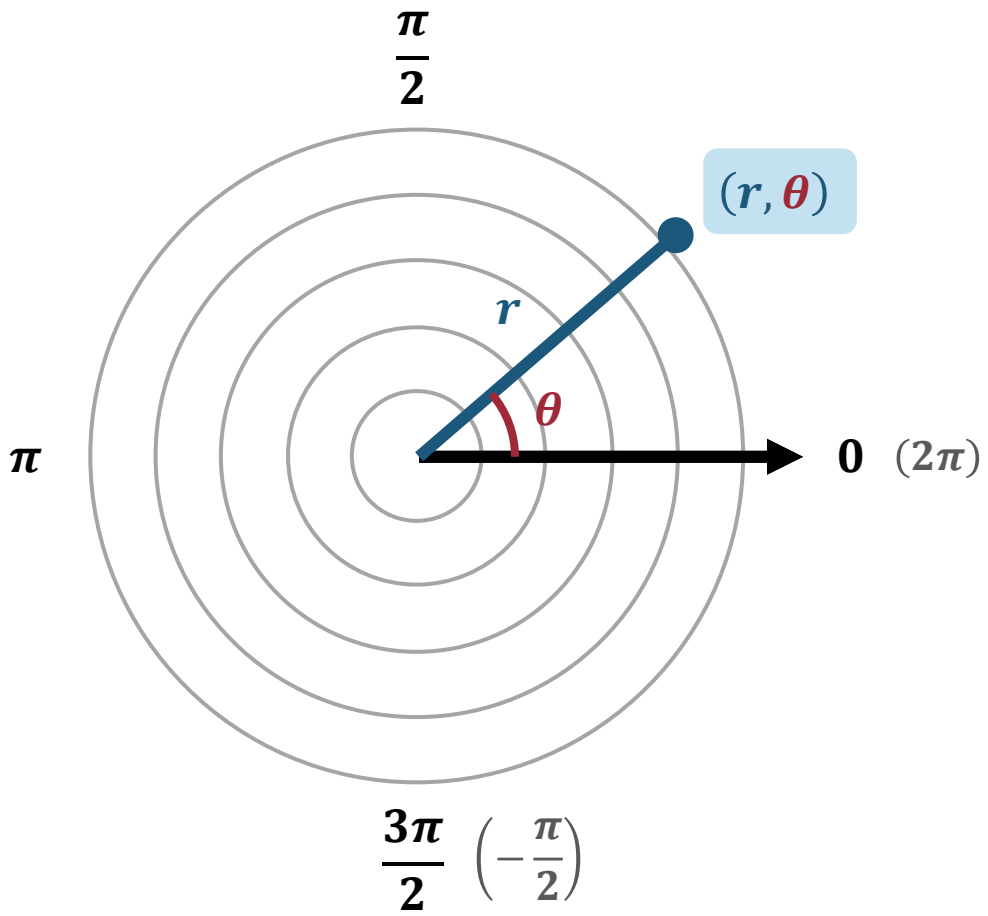


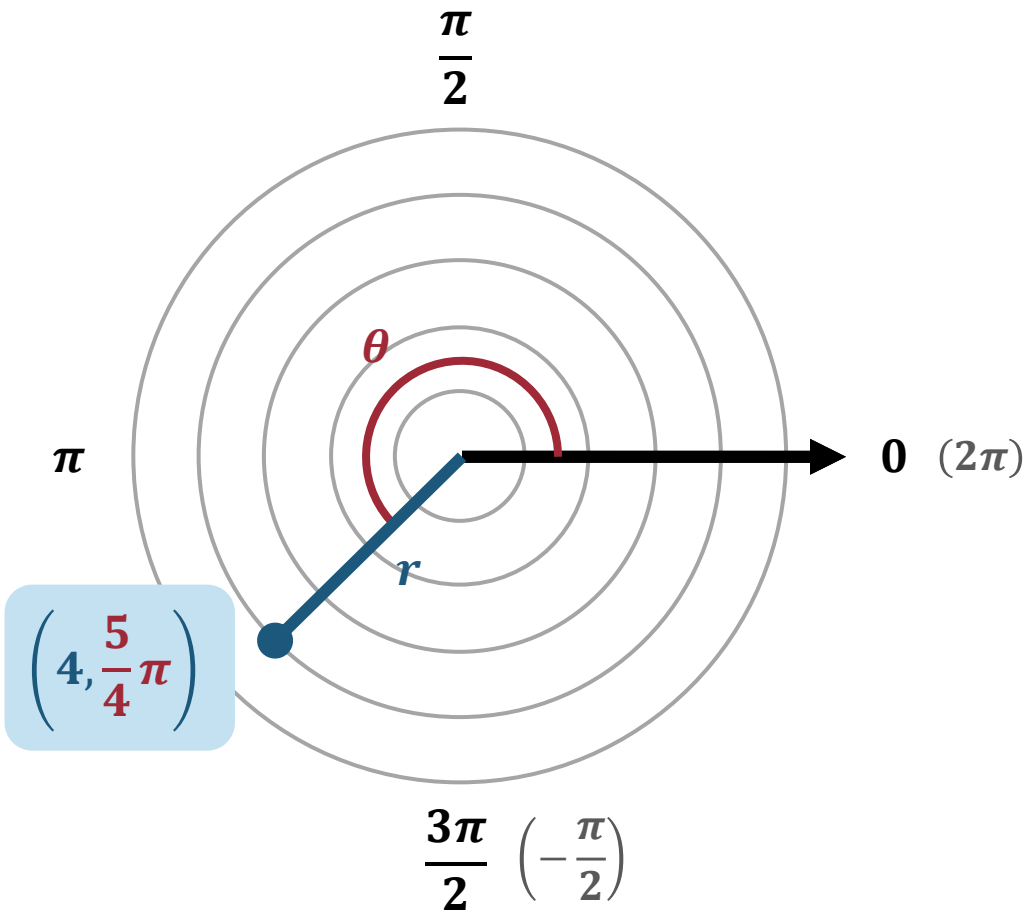
Exercise: Random Points in a Circle (Rejection Sampling)

```
// Calculate the x- and y-positions
x = random(400);
y = random(400);

// Check if the point lies in the circle
if (dist(200, 200, x, y) < 200) {
  // Draw a circle
  circle(x, y, 5);
}
```

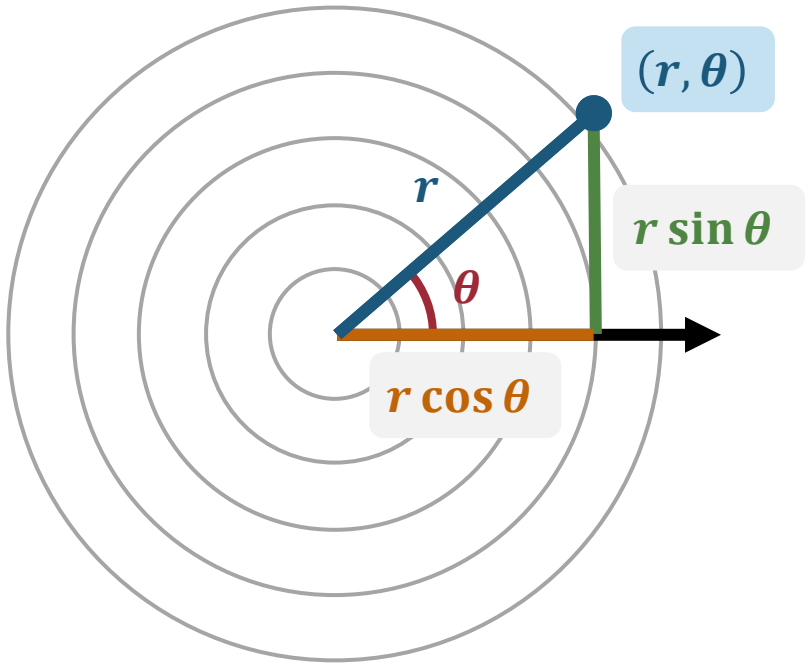


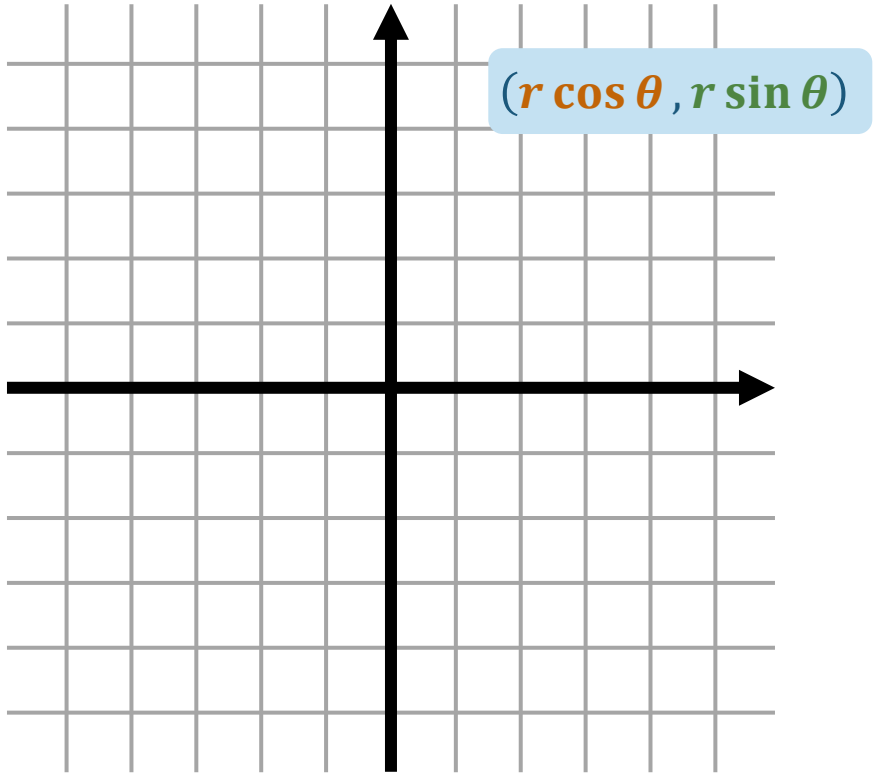
Polar Coordinate





Conversion: Polar → Cartesian

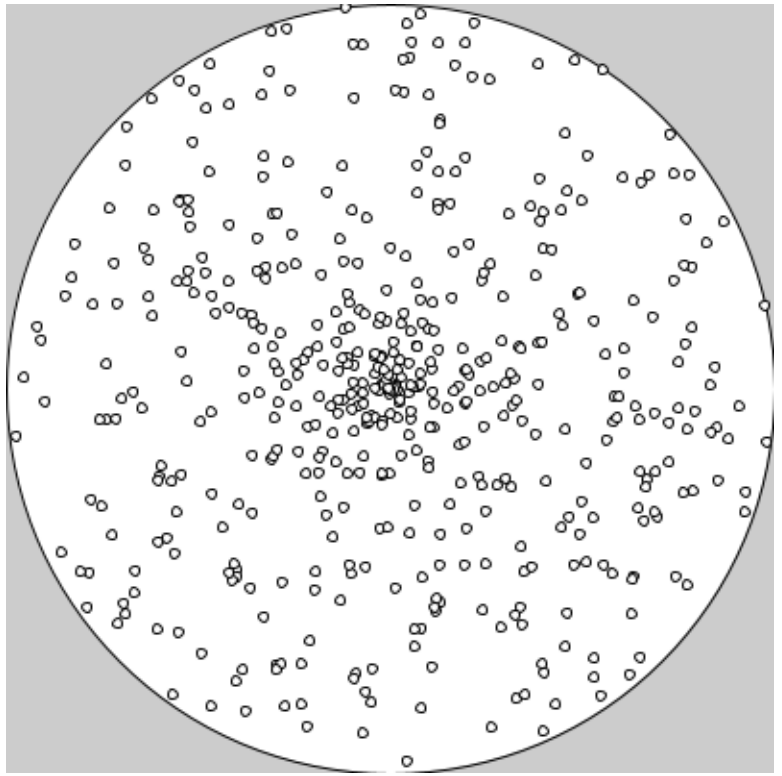




$$(r, \theta) \rightarrow (r \cos \theta , r \sin \theta)$$

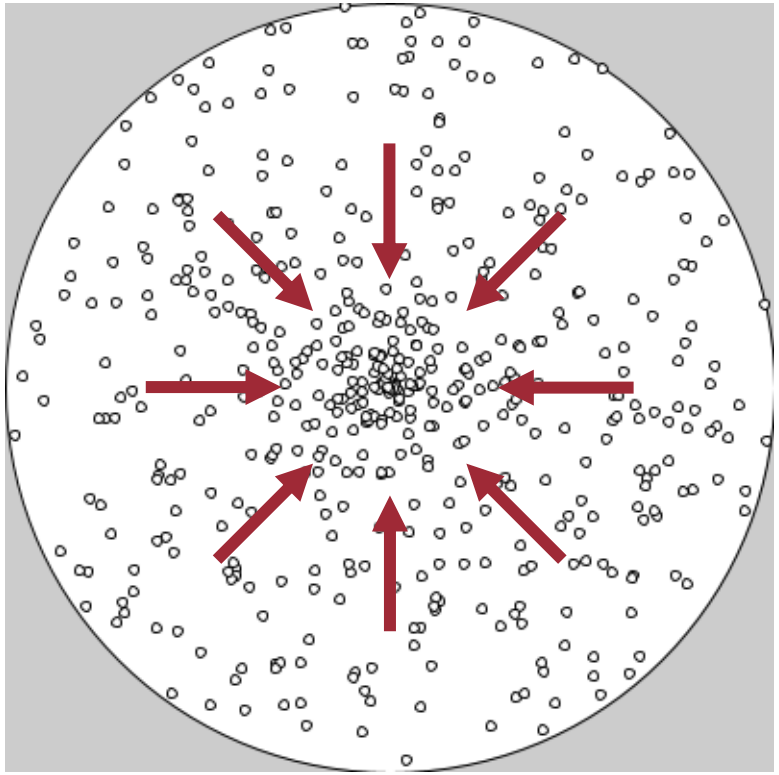
Exercise: Random Points in a Circle (Polar Coordinate Sampling)

```
// Generate a random radius and angle
r = random(200);
theta = random(0, TWO_PI);

// Calculate the x- and y-positions
x = 200 + r * cos(theta);
y = 200 + r * sin(theta);

// Draw a circle
circle(x, y, 5);
```

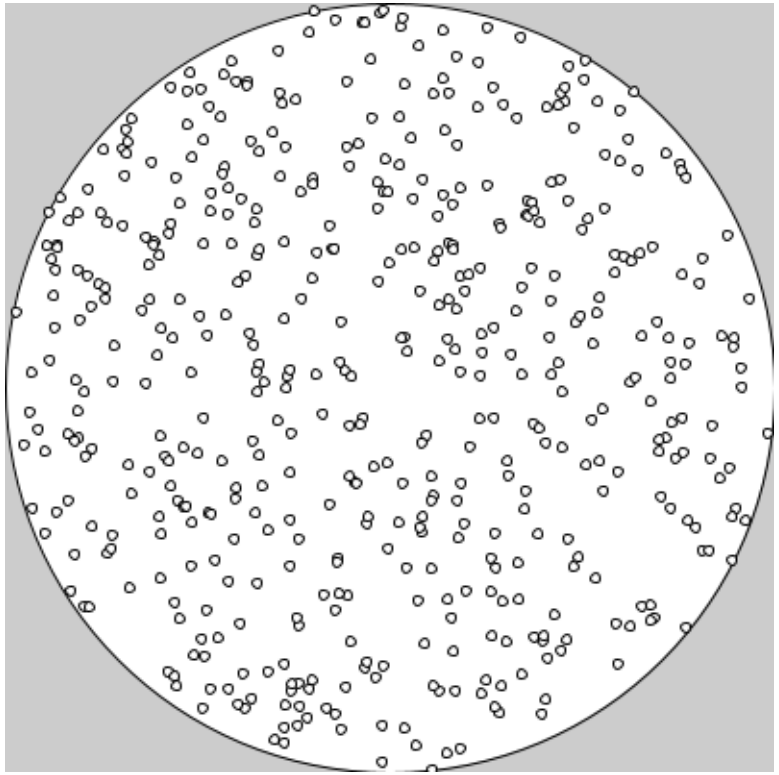


Naïve Approach vs Rejection Sampling

Naïve Approach

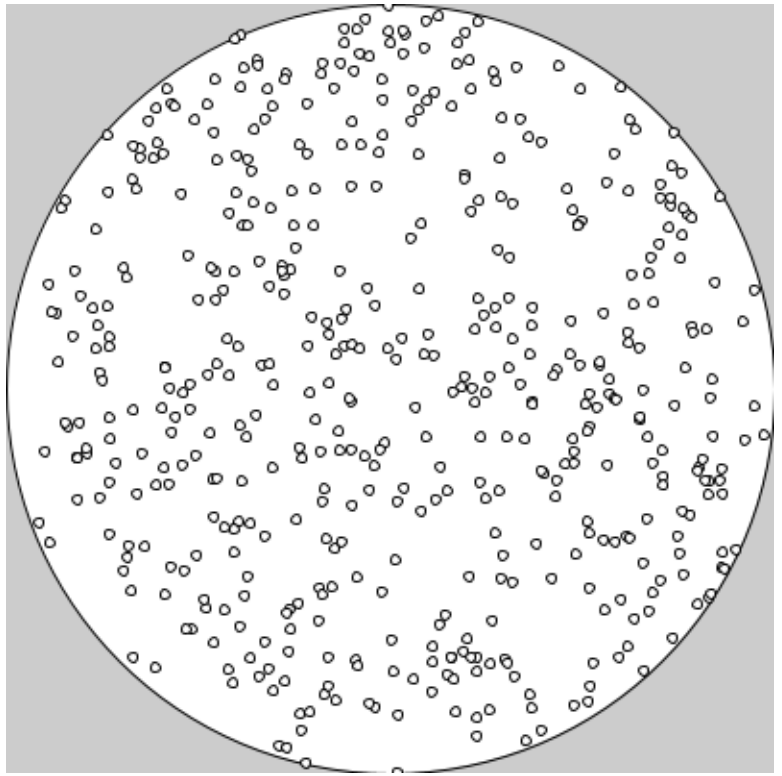


Rejection Sampling



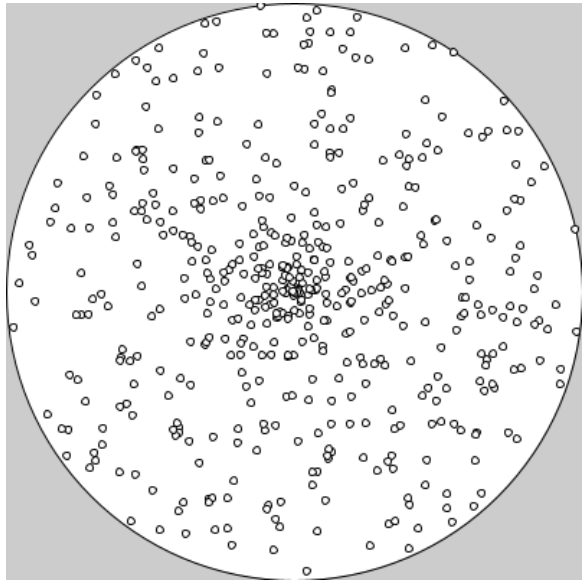
Advanced Method: Inversion Transform Sampling

```
// Generate a random radius and angle
r = 200 * sqrt(random(1));
theta = random(0, TWO_PI);

// Calculate the x- and y-positions
x = 200 + r * cos(theta);
y = 200 + r * sin(theta);

// Draw a circle
circle(x, y, 5);
```



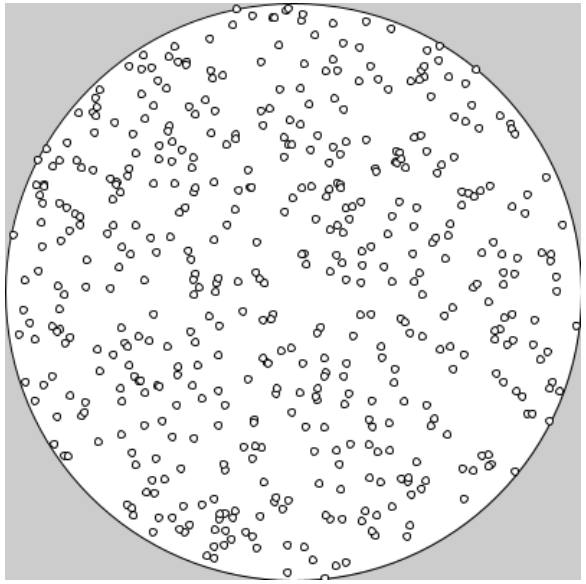
Three Approaches

Naïve Approach



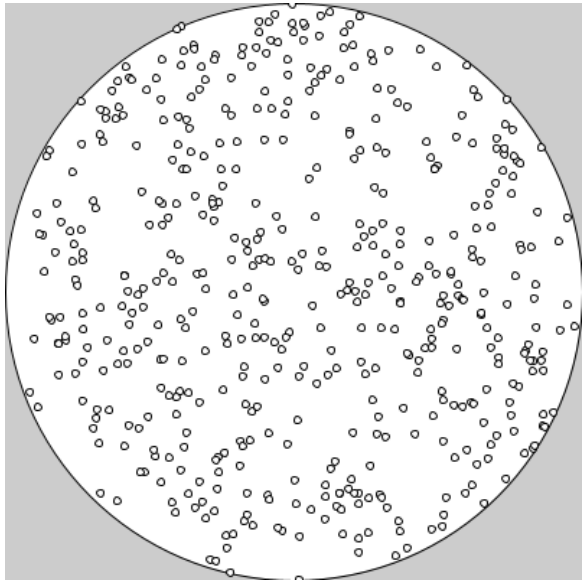
Not uniformly sampled
(denser near the center)

Rejection Sampling



Does not always produce a point at each step

Inverse Transform Sampling



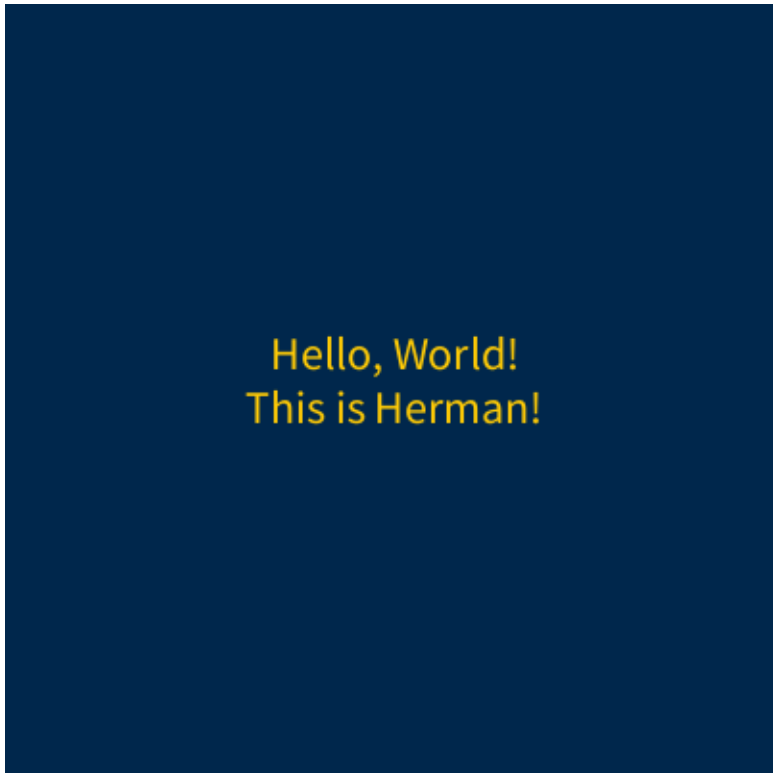
Require some math
(not always tractable)

What if I want the same random numbers every time?

- `random()` is not truly random, but ***random enough*** for most use cases
- Technically, we call these algorithms *pseudorandom* algorithms
- Pseudorandom algorithms take a **random seed** as input, which controls the sequence of random numbers it generates
- **`randomSeed(seed)`** set the random seed!
  - If you set the random seed to a fixed number, you'll get the exact same sequence of random numbers when calling `random()` and `randomGaussian()`
  - Any integer works → **`randomSeed(0)`** or **`randomSeed(1000)`** or **`randomSeed(1024)`**

Bonus: Controlling Bouncing Hello World

- Let's add some ***controls*** and ***randomness*** at the same time!
- Make it
  - move toward a ***random*** **direction** at a preset speed when the program starts
  - when the mouse is clicked, move toward **the direction where the cursor is at** a preset speed



They All Work, but Which is Better?

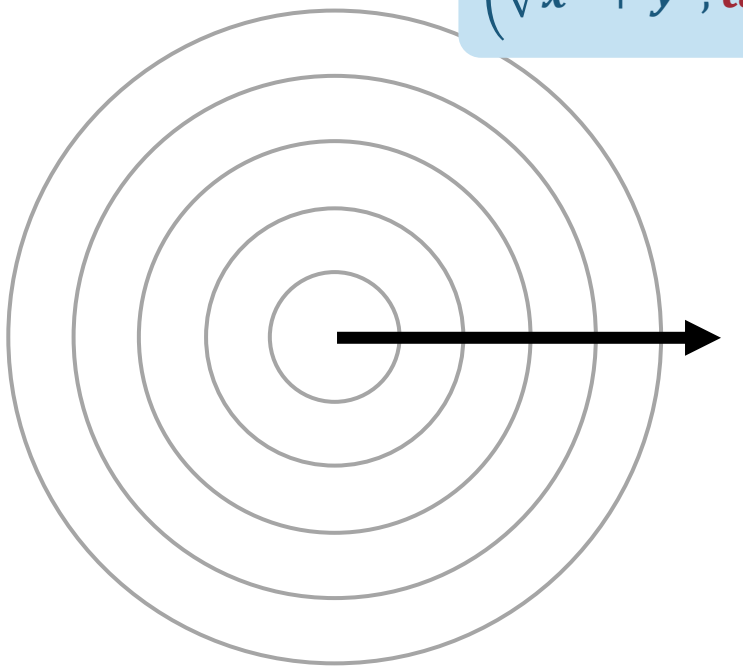
```
speedX = speed * random(0, 1);
speedY = speed * random(0, 1);
```

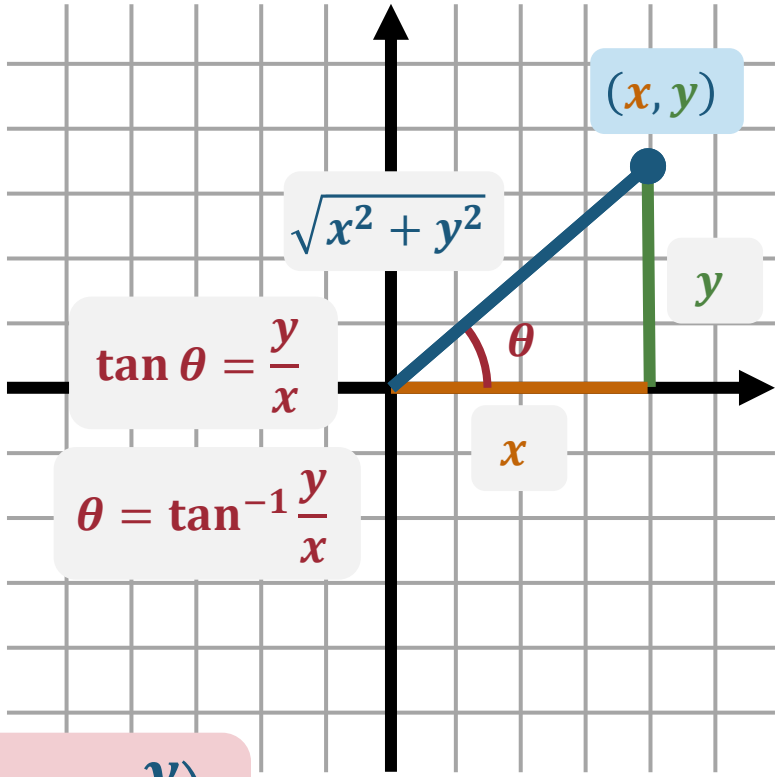
```
a = random(0, 1);
speedX = speed * a;
speedY = speed * sqrt(1 - a * a);
```

```
theta = random(0, TWO_PI);
speedX = speed * cos(theta);
speedY = speed * sin(theta);
```

Conversion: Cartesian → Polar

$$\left(\sqrt{x^2+y^2}, \tan^{-1}\frac{y}{x}\right)$$

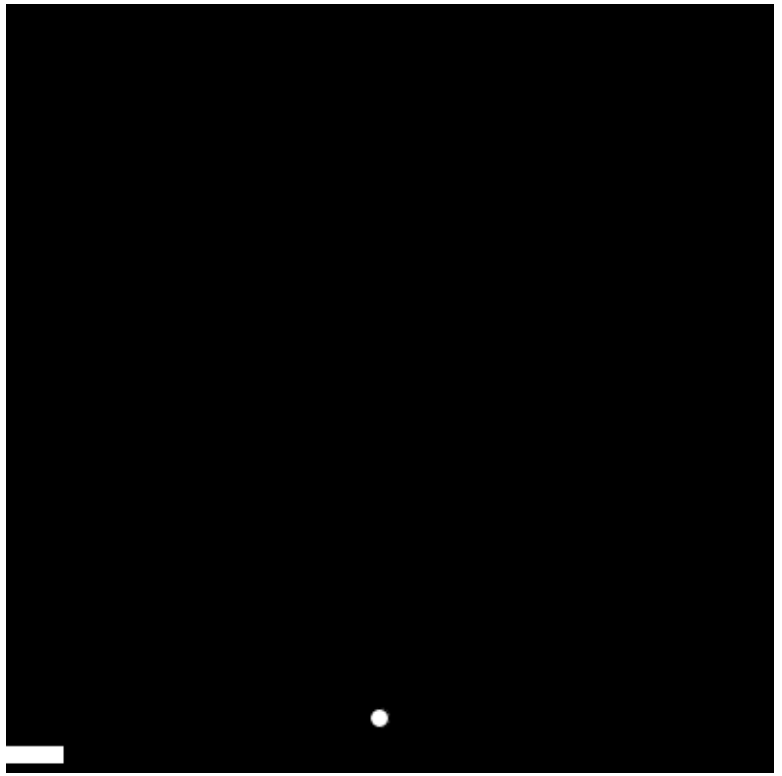




$$(x, y) \rightarrow \left(\sqrt{x^2+y^2}, \tan^{-1}\frac{y}{x}\right)$$

Homework 2: Paddle Ball Game

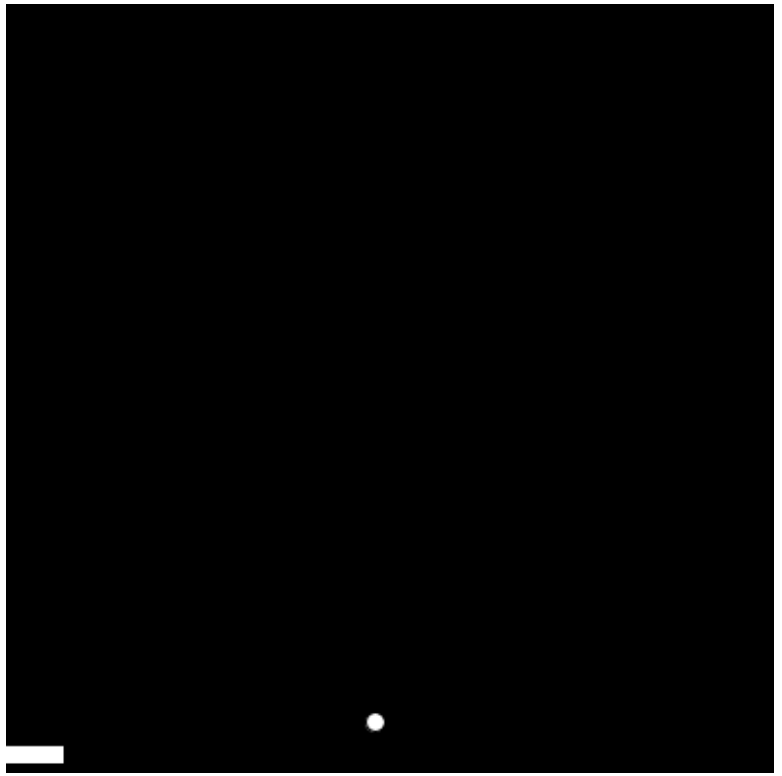
- Instructions will be released on Gradescope
- Features
  - Use the mouse to control the paddle bar
  - Show “GAME OVER!” when the paddle bar does not catch the ball
  - Click the mouse to restart the game
    - You’ll implement an `init()` function that will be called when the game starts or restarts
- Due at **11:59pm ET** on **September 13**
- Late submissions: **1 point deducted per day**



Bonus: Sticky & Shrinking Paddle Ball Game

- To make the game more fun and challenging
  - **Shrink the paddle bar size** after each successful catch
    - Until the bar reaches a reasonable minimum width
  - Apply **stickiness between the paddle and the ball**
    - A fixed ratio of the bar velocity will be added to the ball
    - Use `pmouseX` and `mouseX` to calculate the paddle velocity

The value of `mouseX` in the previous frame



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();
    }
  }
}
```

Disable looping the draw() call

Controlling `draw()`

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

keyCode

- How about this?

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

BACKSPACE is *not* coded

keyCode

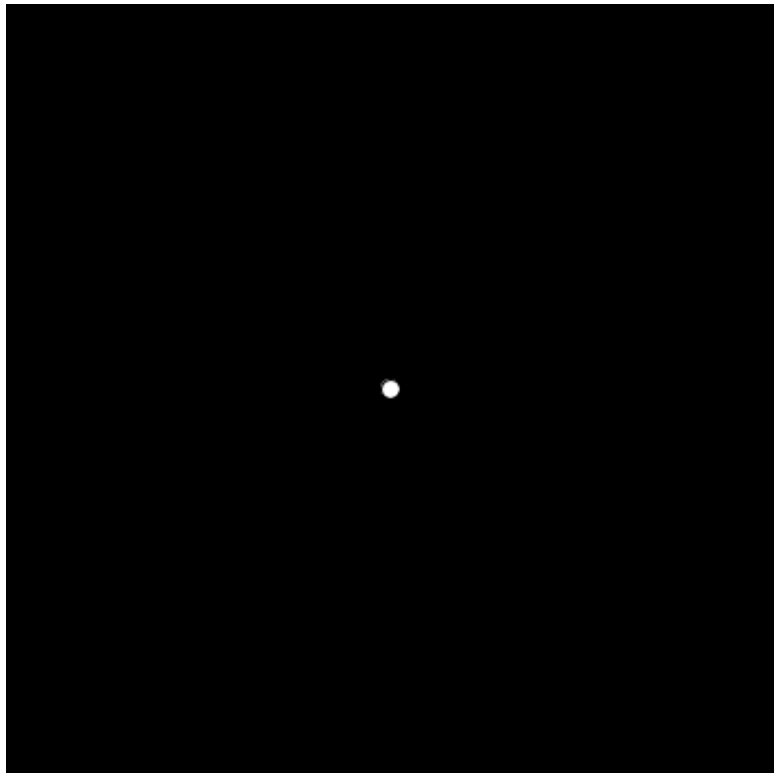
- You don’t need keyCode for BACKSPACE, TAB, ENTER, RETURN, ESC, and DELETE

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

Exercise: Use the Arrow Keys to Control a Ball

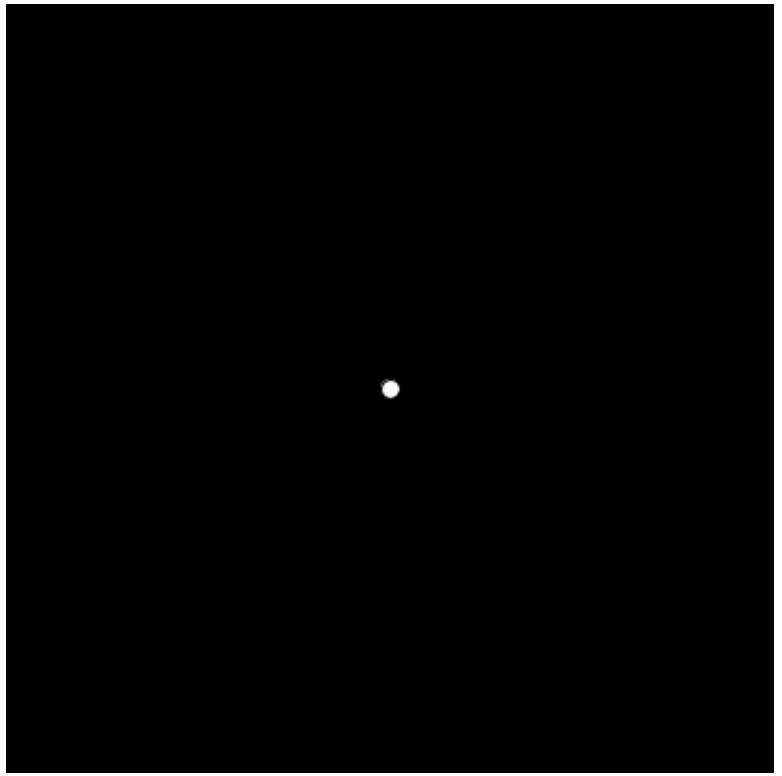
- Create a simple interface where you can move the ball around using the arrow keys
- **Hints**:
  - What **variables** do we need?
  - You'll need `keyCode` for the arrow keys

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



Exercise: Use the Arrow Keys to Control a Ball

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

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



Conditionals – if-else-if Statement

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

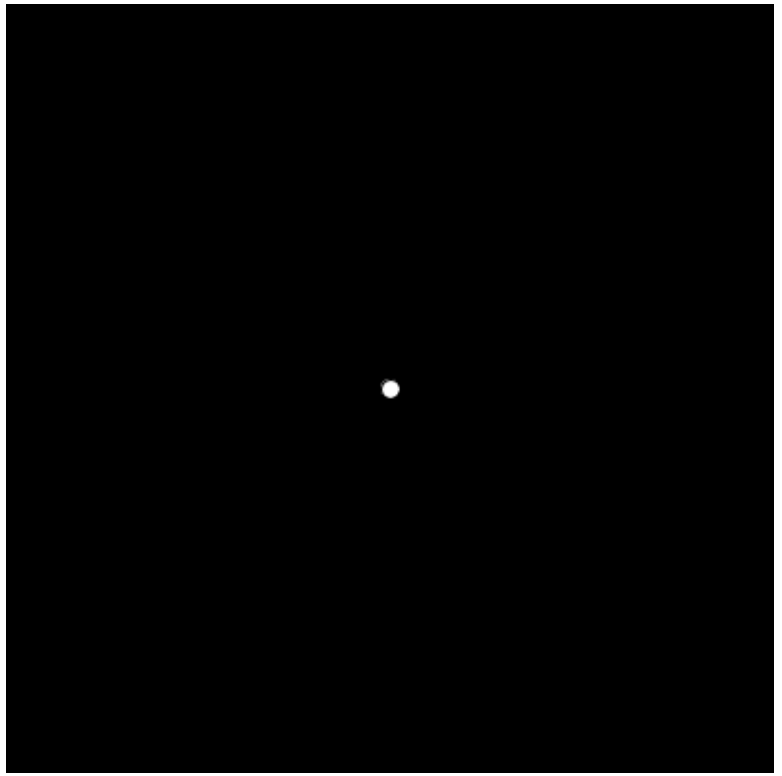
Conditionals – if-else-if Statement

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

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

Exercise: Use the Arrow Keys to Control a Ball

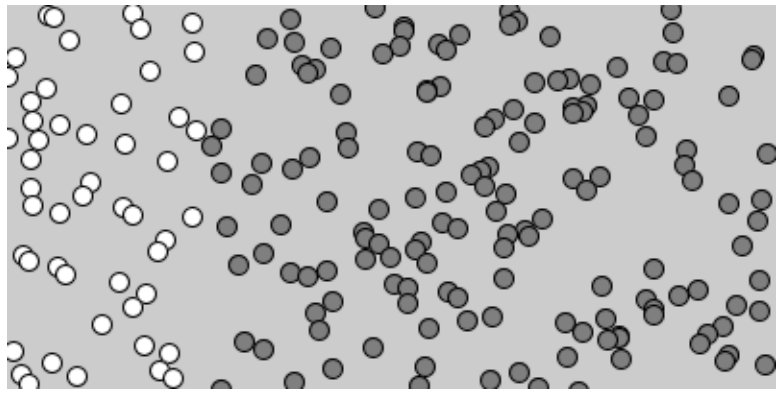
```
float x = 200;
float y = 200;
float step = 10;

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;
    }
  }
}
```



else if versus if

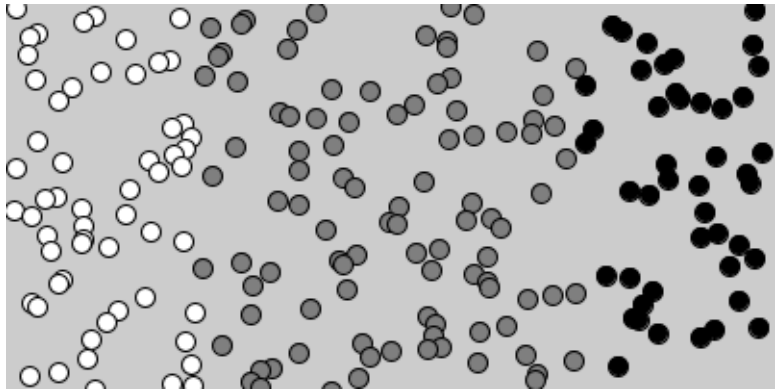
```
if (mouseX > 300) {
  fill(0);
}
if (mouseX < 100) {
  fill(255);
} else {
  fill(127);
}
```

mouseX ≥ 100



```
if (mouseX > 300) {
  fill(0);
} else if (mouseX < 100) {
  fill(255);
} else {
  fill(127);
}
```

100 ≤ mouseX ≤ 300

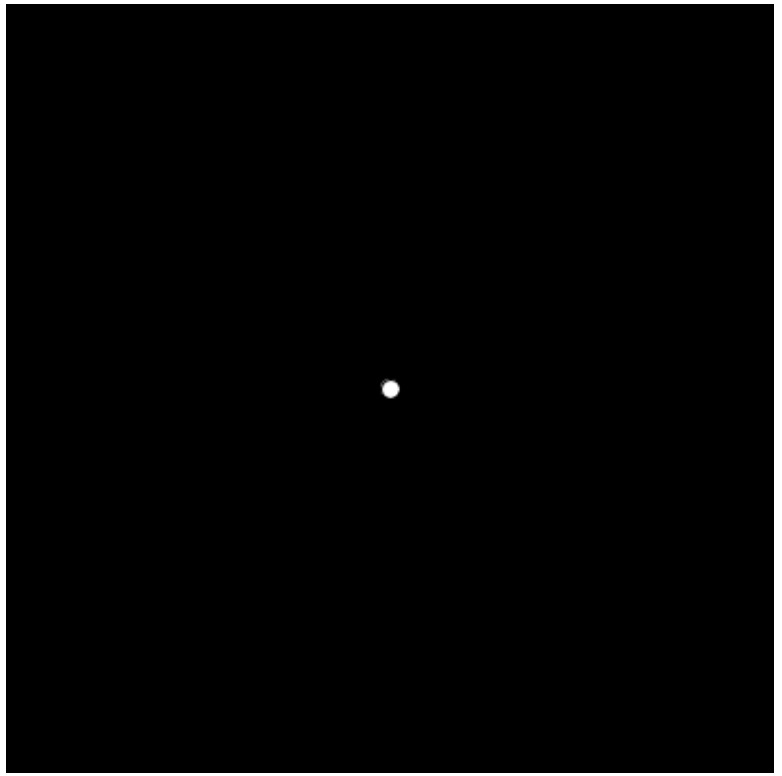


`switch` Statement

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

  case value2:
    doSomethingElse();
    break;

  case value3:
    doYetSomethingElse();
    break;
}
```

Exercise: Rewrite this program using `switch`

```
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

```
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;
}
```

Combining Multiple Conditions in `switch`

```
switch(expression) {
  case value1:
  case value2:
  case value3:
    doSomethingElse();
    break;

  case value4:
    doYetSomethingElse();
    break;
}
```