

PAT 204/504 (Fall 2024)

Creative Coding

Lecture 4: Loops & Recursion

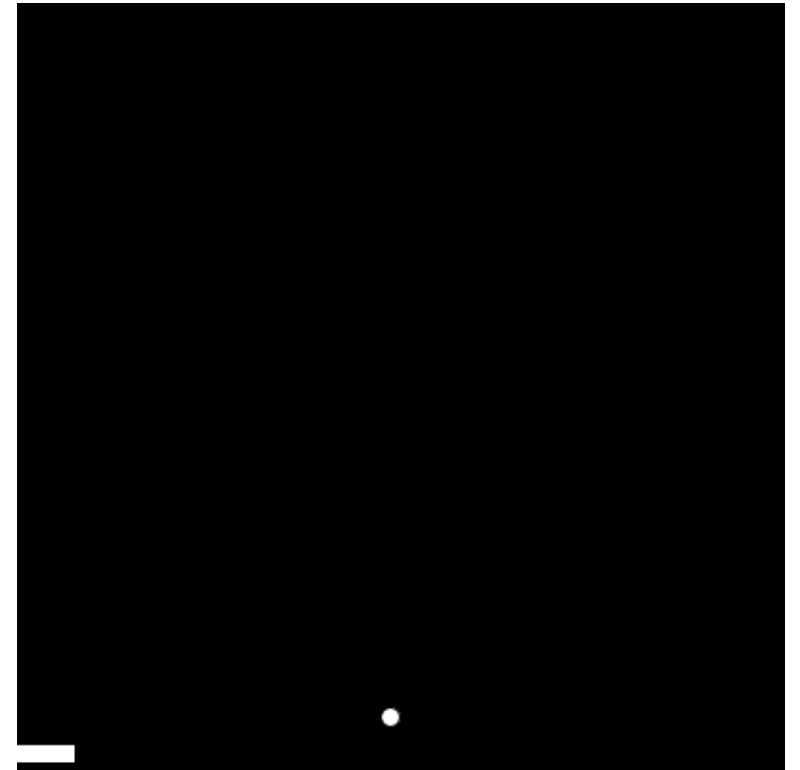
Instructor: Hao-Wen Dong



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

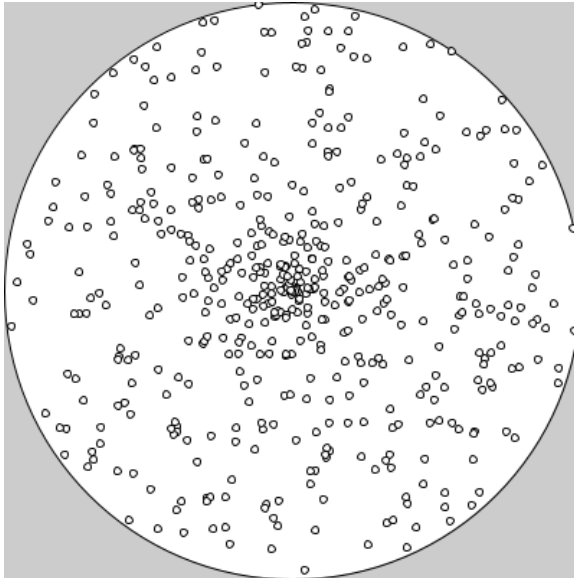
Homework 2: Paddle Ball Game

- 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**



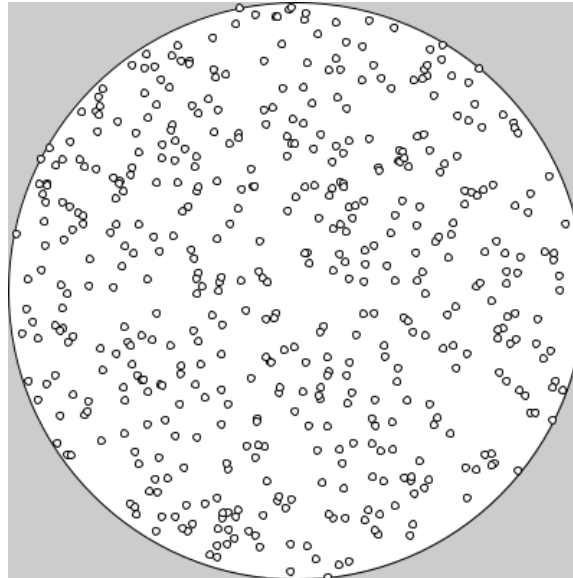
(Recap) Three Approaches

Naïve Approach



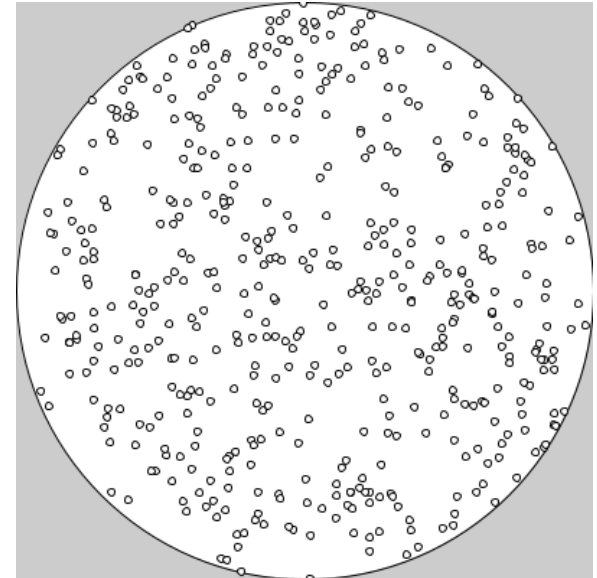
Not uniformly sampled
(denser near the center)

Rejection Sampling



Does not always produce
a point at each step

Inverse Transform Sampling

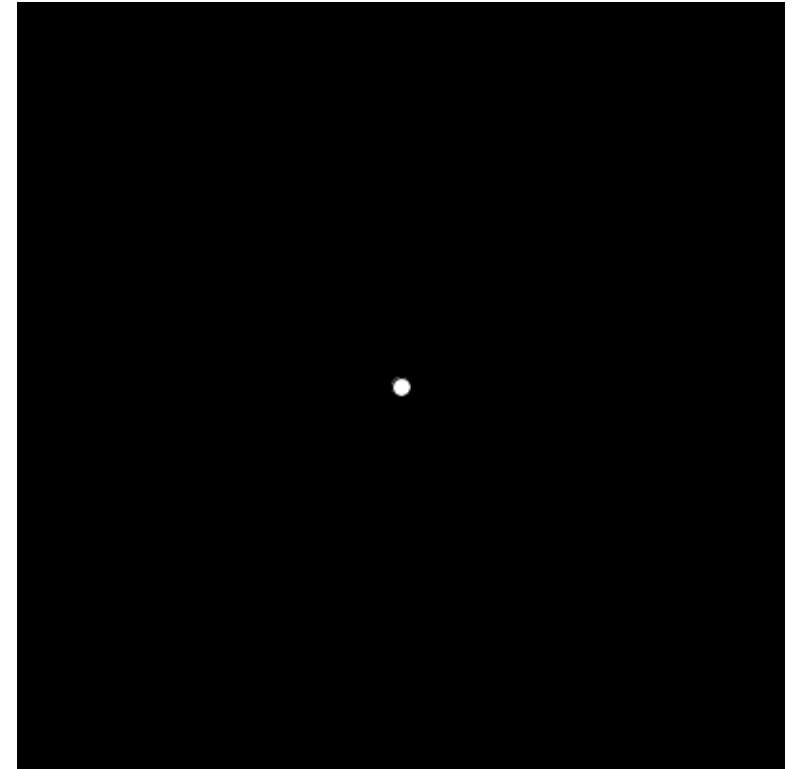


Require some math
(not always tractable)

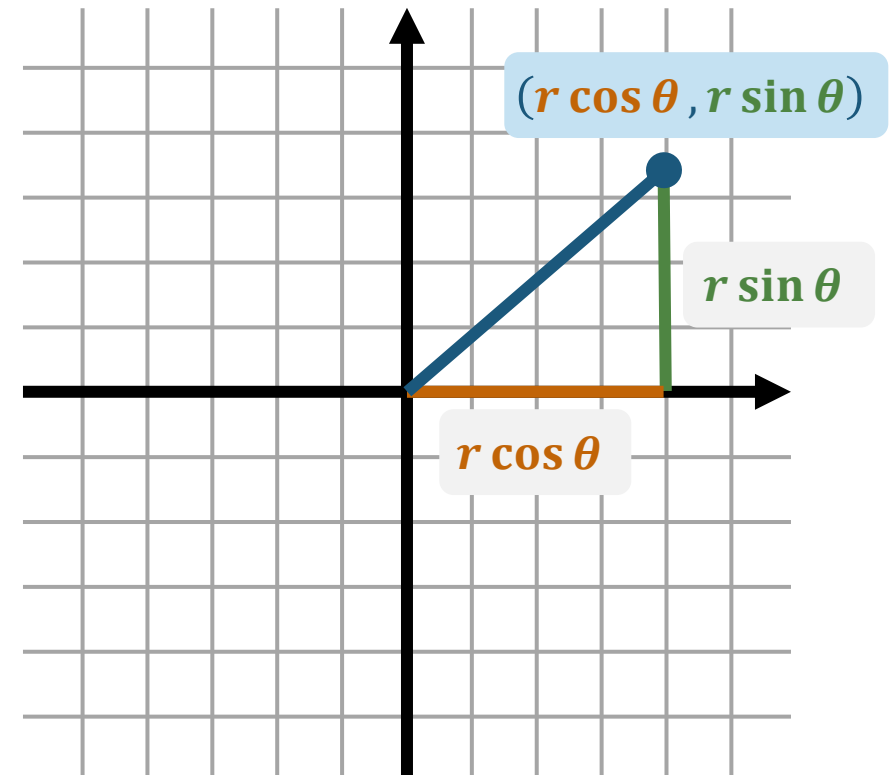
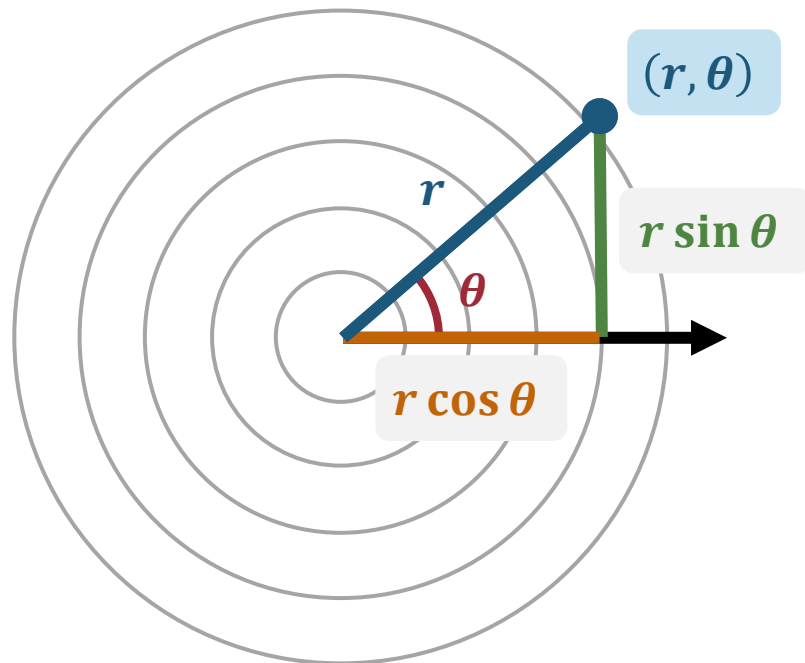
(Recap) Exercise: Use the Arrow Keys to Control a Ball

```
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;
    }
  }
}
```



(Recap) Conversion: Polar \rightarrow Cartesian

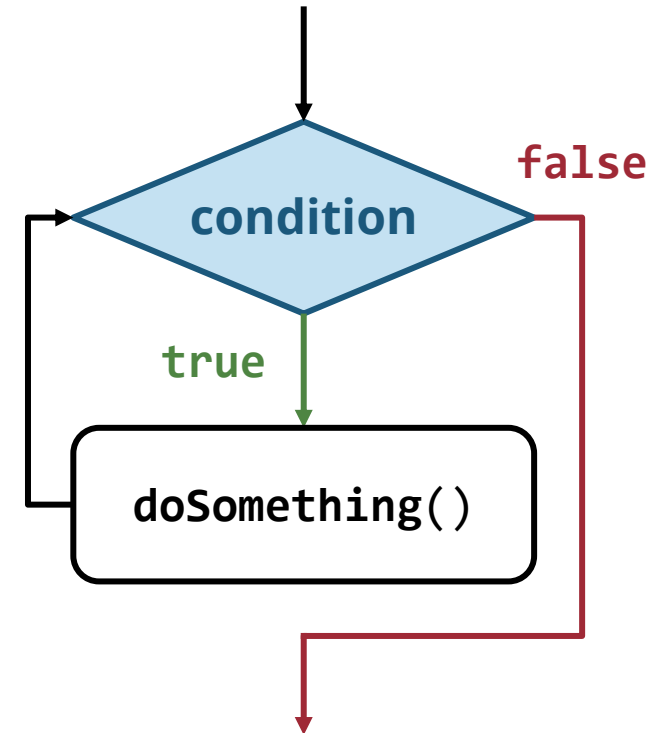


$$(r, \theta) \rightarrow (r \cos \theta, r \sin \theta)$$

Loops

while Loop

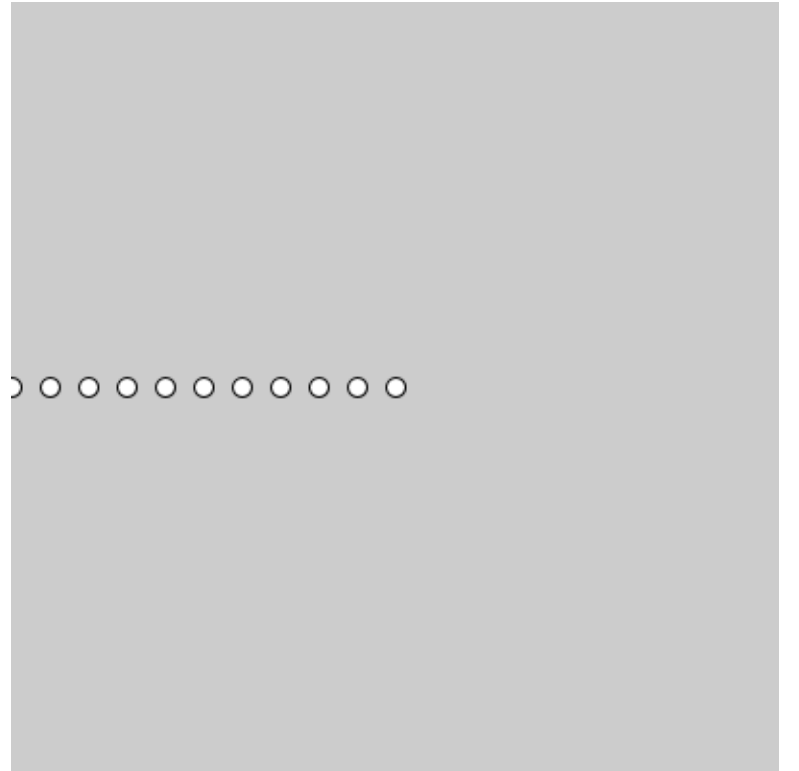
```
while (condition) {  
    doSomething();  
}
```



Example: Series of Circles

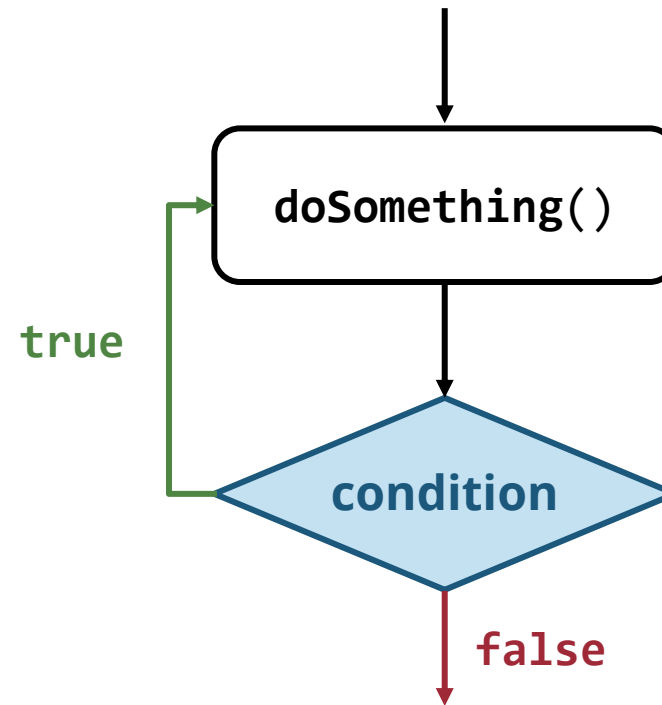
```
// Initialize x
x = 0;

// Draw the circles
while (x <= 200) {
    circle(x, 200, 10);
    x += 20;
}
```



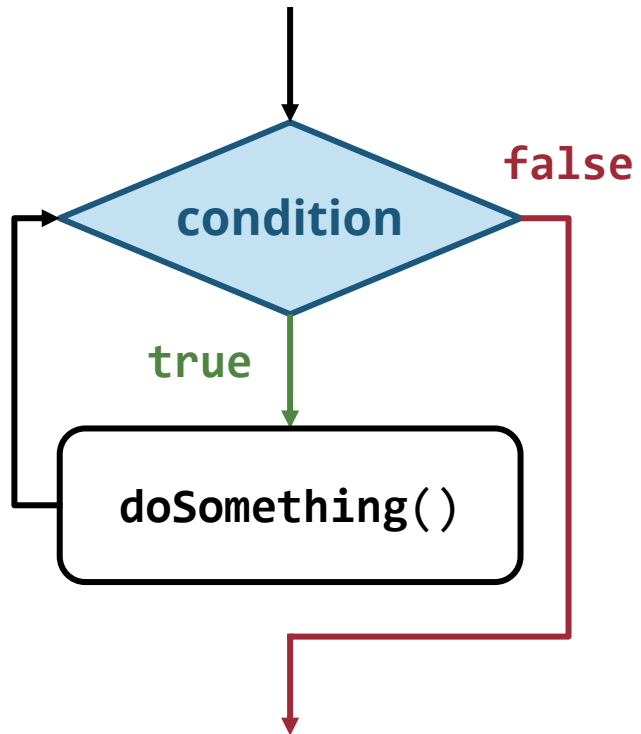
do-while Loop

```
do {  
    doSomething();  
}  
while (condition);
```

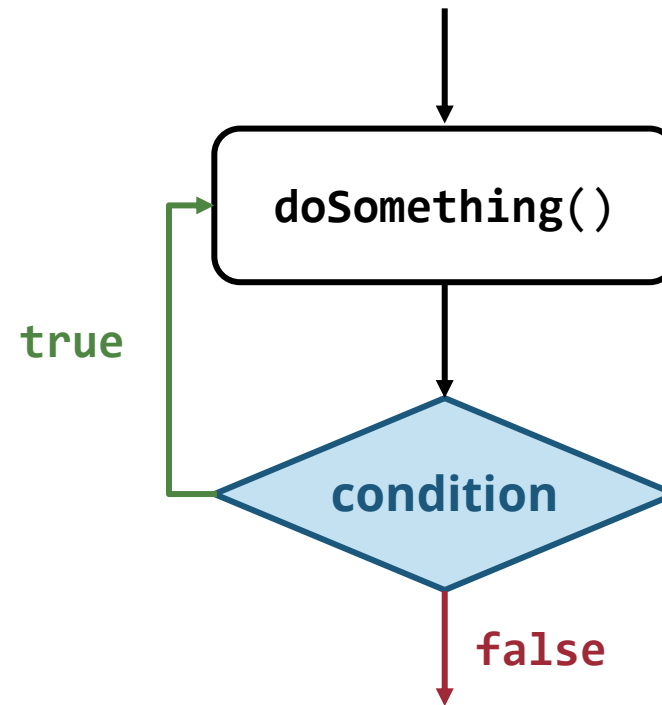


while vs do-while Loops

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



```
do {  
    doSomething();  
}  
while (condition);
```



break – Exiting a Loop in the Middle

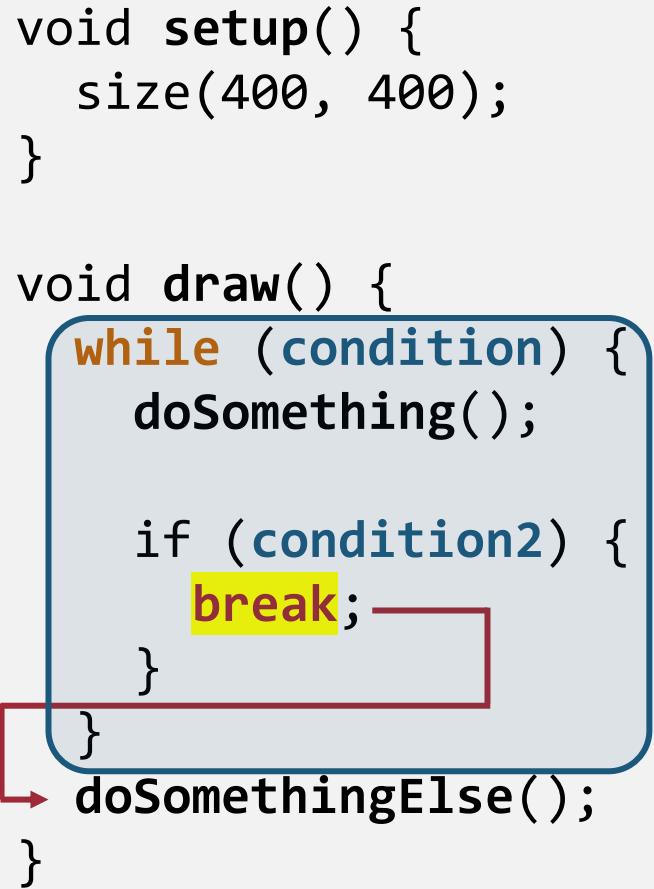
```
while (condition) {  
    doSomething();  
  
    if (condition2) {  
        break;  
    }  
}
```

Common Control Commands

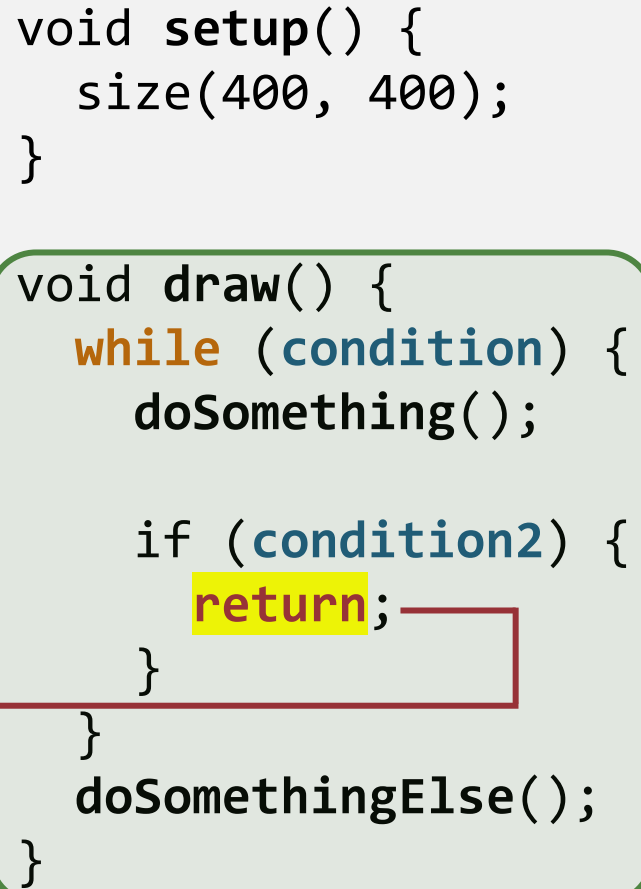
- **break** Exit the current `while`, `for` or `switch` block
- **return** Exit the current function, e.g., `draw()`
- **exit()** Stop the whole program
- **noLoop()** Disable looping the `draw()` function

break vs return vs exit()

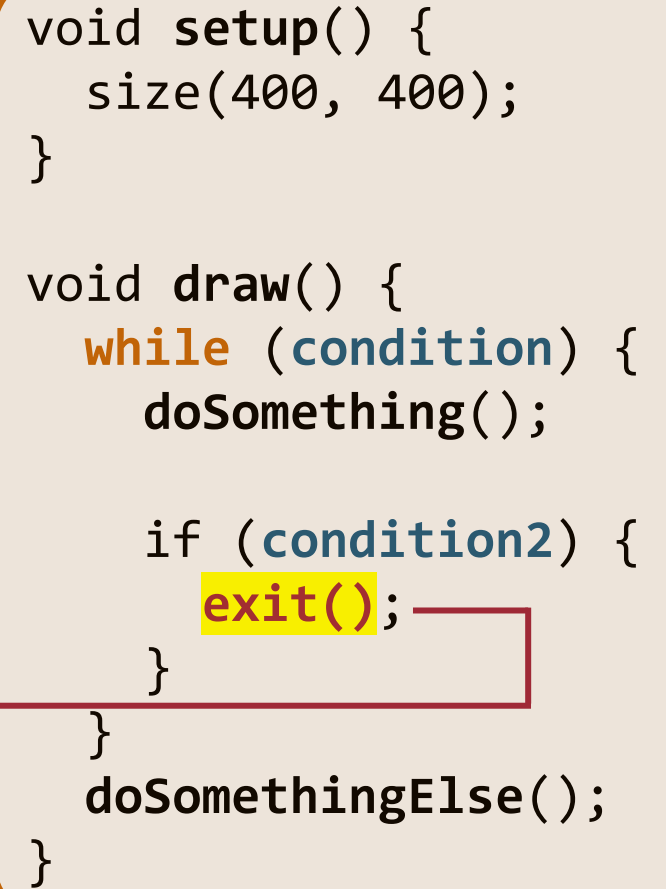
```
void setup() {  
    size(400, 400);  
}  
  
void draw() {  
    while (condition) {  
        doSomething();  
  
        if (condition2) {  
            break;  
        }  
    }  
    doSomethingElse();  
}
```

A blue rounded rectangle highlights the while loop in the code. A red arrow starts from the 'break;' statement, goes right, then down, then left, and finally down to the closing brace of the while loop, indicating that the loop is terminated immediately.

```
void setup() {  
    size(400, 400);  
}  
  
void draw() {  
    while (condition) {  
        doSomething();  
  
        if (condition2) {  
            return;  
        }  
    }  
    doSomethingElse();  
}
```

A green rounded rectangle highlights the while loop in the code. A red arrow starts from the 'return;' statement, goes right, then down, then left, and finally down to the closing brace of the draw() function, indicating that the function returns immediately.

```
void setup() {  
    size(400, 400);  
}  
  
void draw() {  
    while (condition) {  
        doSomething();  
  
        if (condition2) {  
            exit();  
        }  
    }  
    doSomethingElse();  
}
```

An orange rounded rectangle highlights the while loop in the code. A red arrow starts from the 'exit();' statement, goes right, then down, then left, and finally down to the closing brace of the draw() function, indicating that the program terminates immediately.

return vs noLoop() in draw()

- With **return**, draw() exits right away but still loops
- With **noLoop()**, draw() runs until the end but stops looping afterwards

Suppose `isGameOver=true`

```
void draw() {  
  if (isGameOver) {  
    doSomething();  
    return;  
  }  
doSomethingElse();  
}
```

```
void draw() {  
  if (isGameOver) {  
    doSomething();  
    noLoop();  
  }  
  doSomething();  
}
```

while(true) Loop

```
while (true) {  
    doSomething();  
  
    if (condition) {  
        break;  
    }  
}
```

equivalent

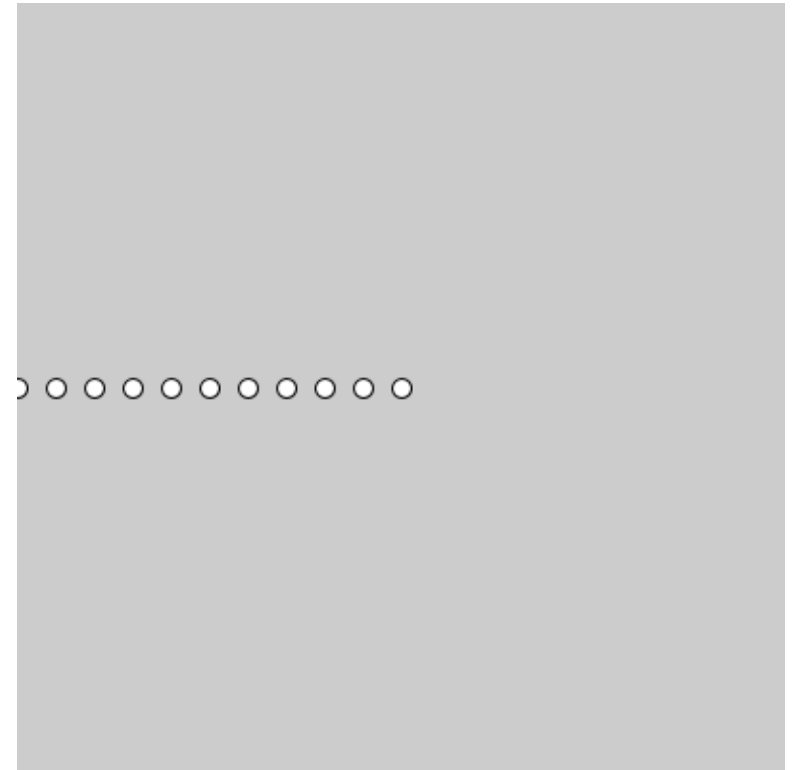


```
do {  
    doSomething();  
}  
while (condition);
```

Example: Series of Circles

```
// Initialize x
float x = 0;

// Draw the circles
while (x <= 200) {
    circle(x, 200, 10);
    x = x + 20;
}
```



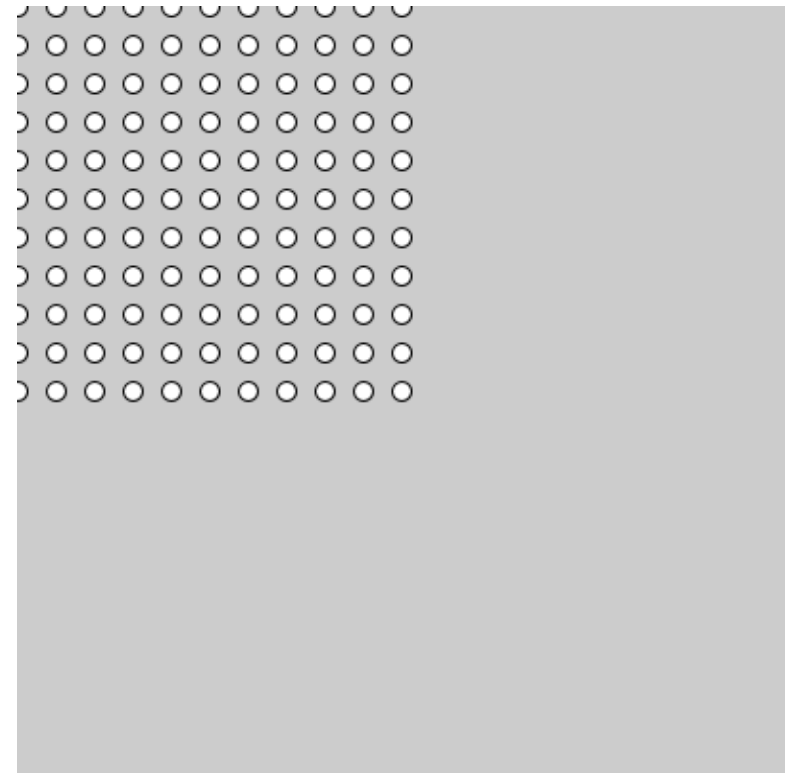
Exercise: Grid of Circles

- Draw a grid of circles using `while` loops
- You'll need *two while loops*
 - One inside the other, i.e., *nested loops*

```
float x = 0, y = 0;

while (x <= 200) {
    y = 0;
    while (y <= 200) {
        circle(x, y, 10);
        y += 20;
    }
    x += 20;
}
```

Don't forget to reset y to 0



for Loop

```
// Initialize x
```

```
float x = 0;
```

```
// Draw the circles
```

```
while (x <= 200) {  
    circle(x, 200, 10);
```

```
    x = x + 20;
```

```
}
```

Initialization

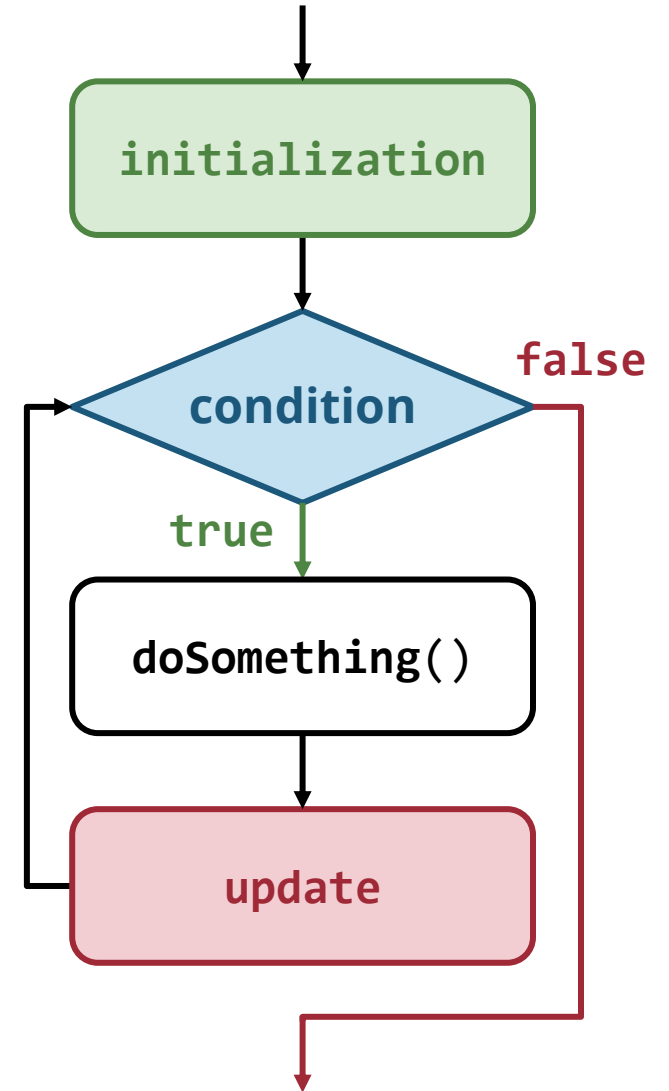
Condition

Update

```
for (float x = 0; x <= 200; x = x + 20) {  
    circle(x, 200, 10);  
}
```

for Loop

```
for (initialization; condition; update) {  
    doSomething();  
}
```



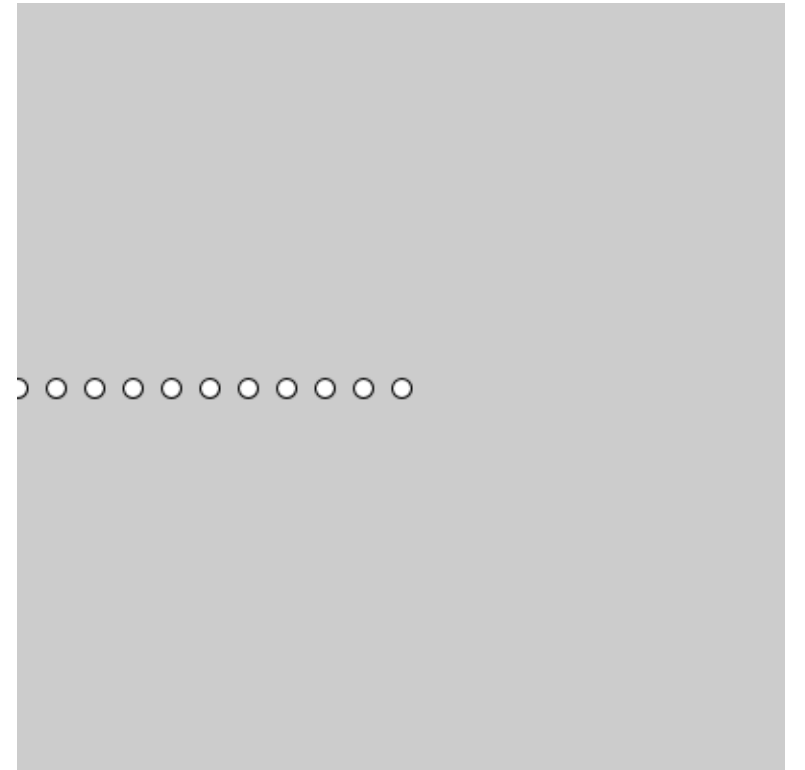
Example: Series of Circles

```
for (float x = 0; x <= 200; x = x + 20) {  
    circle(x, 200, 10);  
}
```

x-position 0 20 40 60 80 100 120 140 160 180 200

```
for (int i = 0; i <= 10; i = i + 1) {  
    circle(i * 20, 200, 10);  
}
```

circle index 0 1 2 3 4 5 6 7 8 9 10

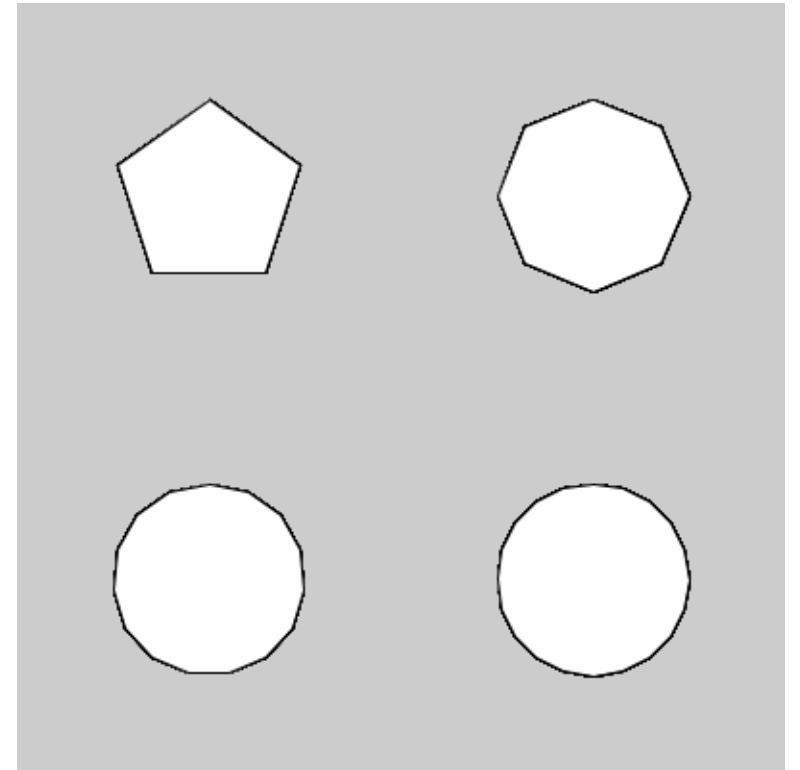
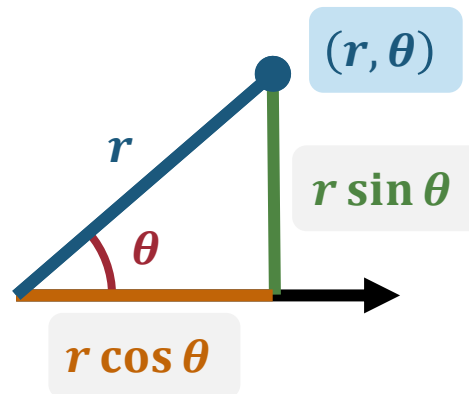


Exercise: Regular Polygons

- Write a function for drawing regular polygons

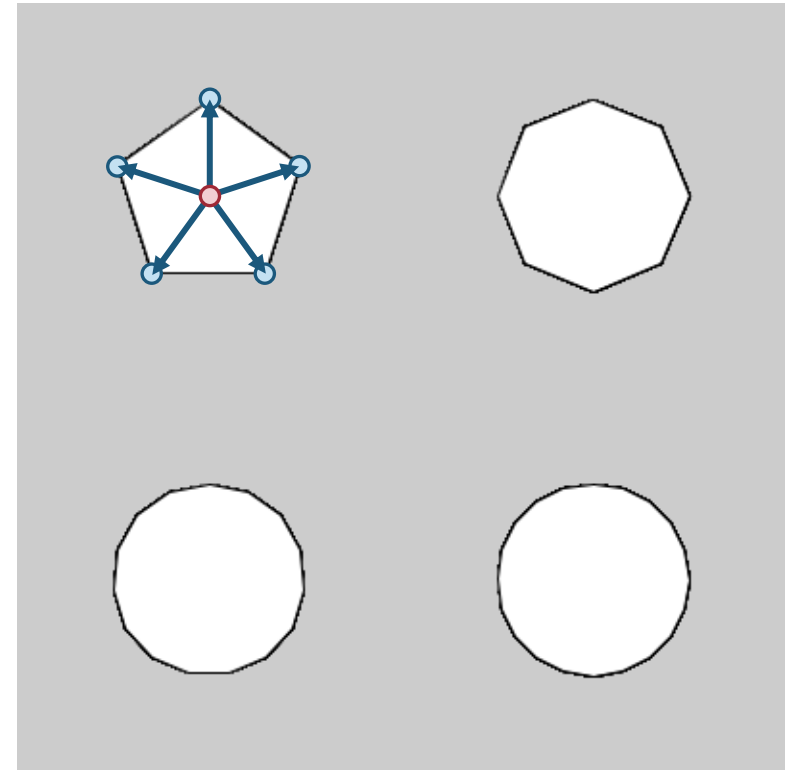
```
void polygon(float x, float y, float radius, int n) {  
    beginShape();  
    for (???) {  
        ???  
    }  
    endShape(CLOSE);  
}
```

- Hints:** Use **polar coordinate!**



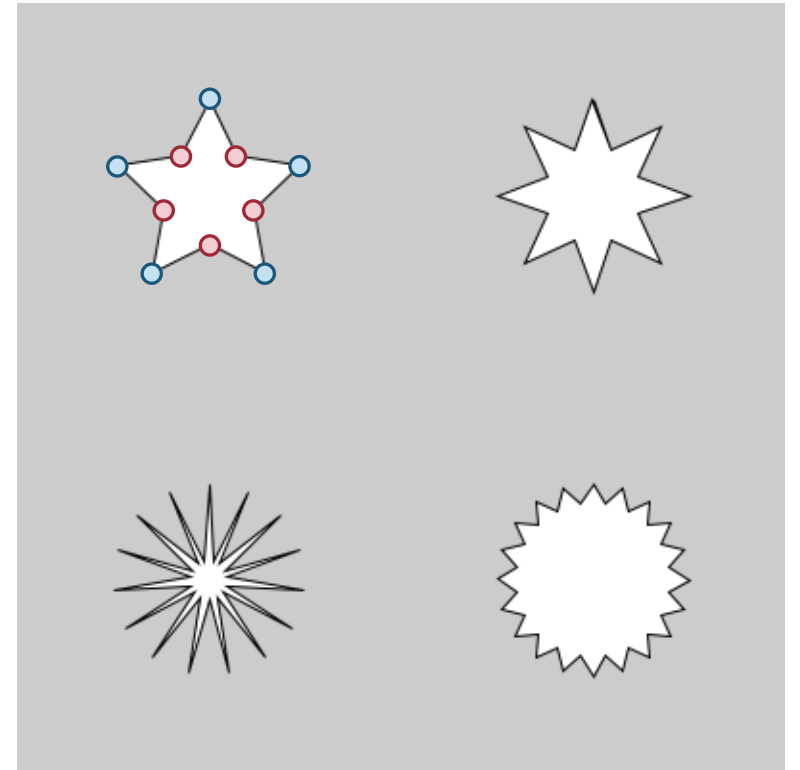
Exercise: Regular Polygons

```
void polygon(float x, float y, float radius, int n) {  
    float vertexX, vertexY;  
    beginShape();  
    for (float a = 0; a < TWO_PI; a += TWO_PI / n) {  
        vertexX = x + radius * cos(a - HALF_PI);  
        vertexY = y + radius * sin(a - HALF_PI);  
        vertex(vertexX, vertexY);  
    }  
    endShape(CLOSE);  
}
```



Example: Stars

```
void star(float x, float y, float r1, float r2, int n) {  
    float vertexX, vertexY;  
    float angle = TWO_PI / n ;  
    beginShape();  
    for (float a = 0; a < TWO_PI; a += angle) {  
        vertexX = x + radius1 * cos(a - HALF_PI);  
        vertexY = y + radius1 * sin(a - HALF_PI);  
        vertex(vertexX, vertexY);  
        vertexX = x + radius2 * cos(a + angle / 2 - HALF_PI);  
        vertexY = y + radius2 * sin(a + angle / 2 - HALF_PI);  
        vertex(vertexX, vertexY);  
    }  
    endShape(CLOSE);  
}
```



Handy Operators: +=, -=, *=, /=, ++, --

```
x = x + 1
```

```
x += 1
```

```
x++
```

```
x = x * 2
```

```
x *= 2
```

```
x = x - 1
```

```
x -= 1
```

```
x--
```

```
x = x / 2
```

```
x /= 2
```

```
for (int i = 0; i <= 10; i = i + 1) {  
    circle(i * 20, 200, 10);  
}
```

```
for (int i = 0; i <= 10; i += 1) {  
    circle(i * 20, 200, 10);  
}
```

```
for (int i = 0; i <= 10; i++) {  
    circle(i * 20, 200, 10);  
}
```

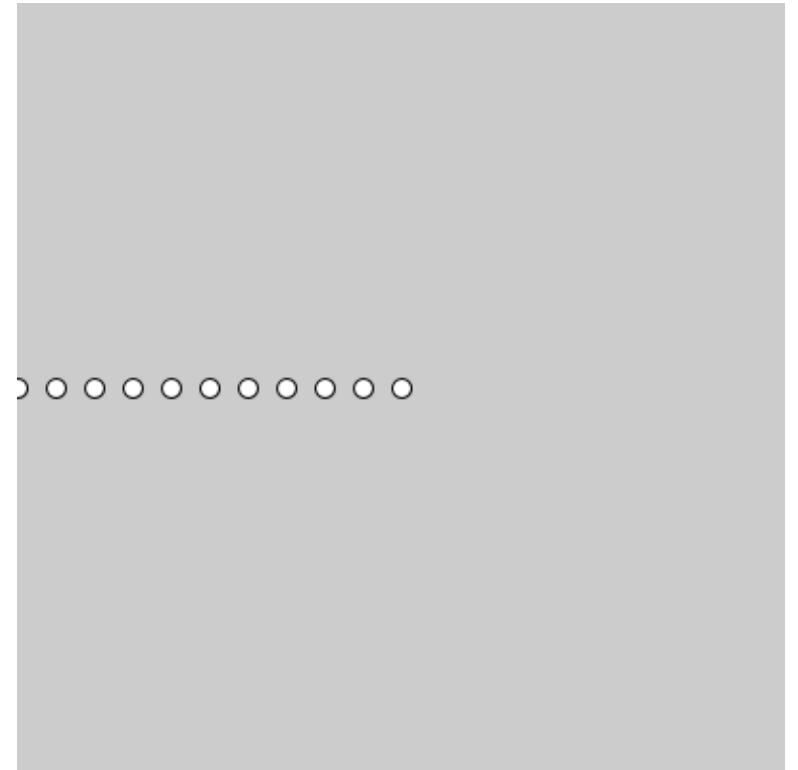

Reverse Looping

```
for (int i = 0; i <= 10; i++) {  
    circle(i * 20, 200, 10);  
}
```

0 1 2 3 4 5 6 7 8 9 10

```
for (int i = 10; i >= 0; i--) {  
    circle(i * 20, 200, 10);  
}
```

10 9 8 7 6 5 4 3 2 1 0



Modulo Operator: %

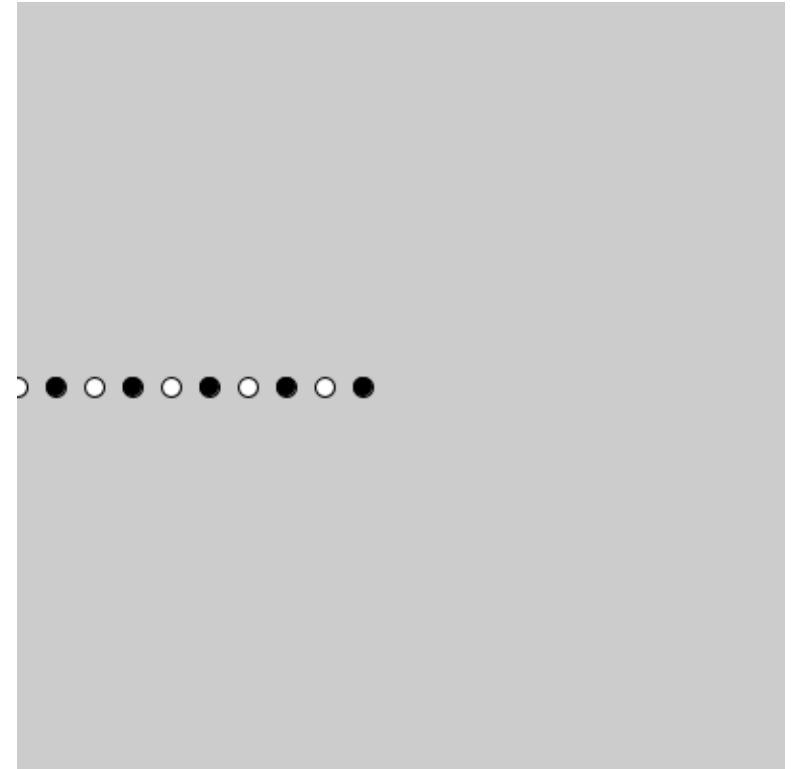
- The modulo operator returns the remainder
 - a / b Quotient
 - $a \% b$ Remainder
- For integer division: $a = b * (a / b) + (a \% b)$

Modulo Operator: %

```
for (int i = 0; i <= 10; i++) {  
    if (i % 2 == 0) {  
        fill(255);  
    } else {  
        fill(0);  
    }  
  
    circle(i * 20, 200, 10);  
}
```

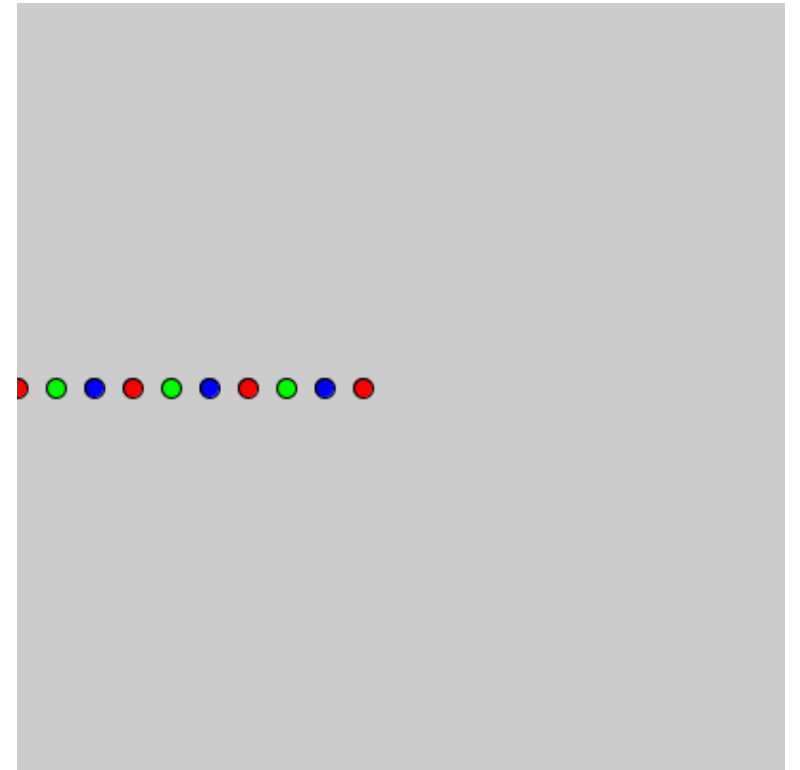
Even numbers

Odd numbers



Modulo Operator: %

```
for (int i = 0; i <= 10; i++) {  
    if (i % 3 == 0) {  
        fill(255, 0, 0);  
    } else if (i % 3 == 1) {  
        fill(0, 255, 0);  
    } else {  
        fill(0, 0, 255);  
    }  
  
    circle(i * 20, 200, 10);  
}
```



Infinite loops: `while(true)` and `for(;;)`

```
while (true) {  
    doSomething();  
  
    if (condition) {  
        break;  
    }  
}
```

equivalent
↔

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

When is an **Infinite Loop** Useful?

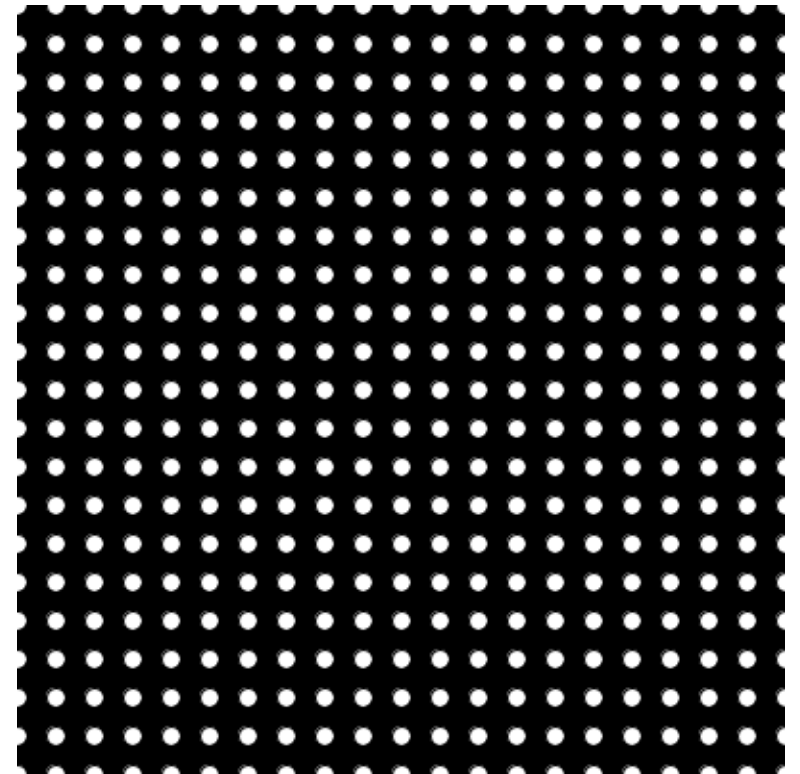
- `while(true)` and `for(;;)` are particularly useful if
 - we need **multiple complex conditions**
 - the **condition needs some computation**

```
while (true) {  
    doSomething();  
  
    if (condition1) break;  
  
    doSomethingElse();  
  
    if (condition2) break;  
  
    doYetSomethingElse();  
}
```

Exercise: Grid of Circles

- Draw a grid of circles using for loops
- Again, you'll need two nested for loops

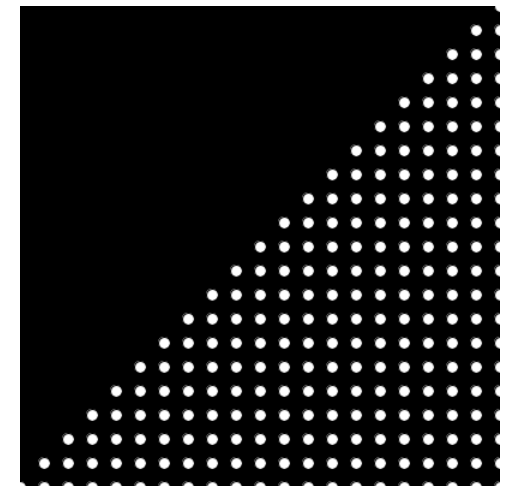
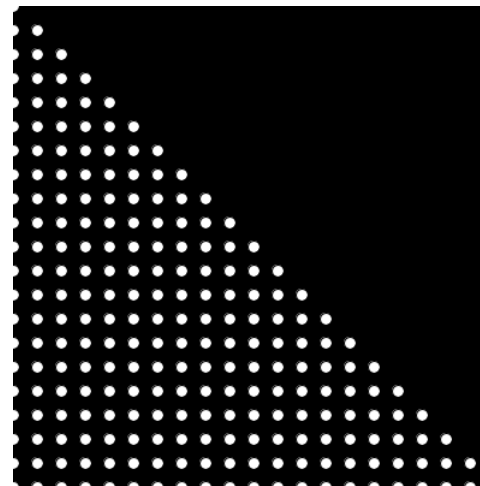
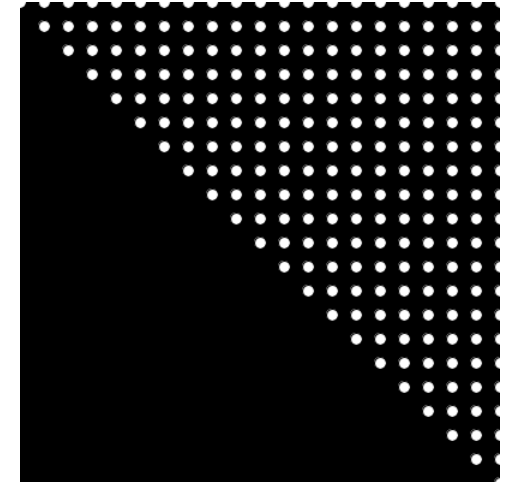
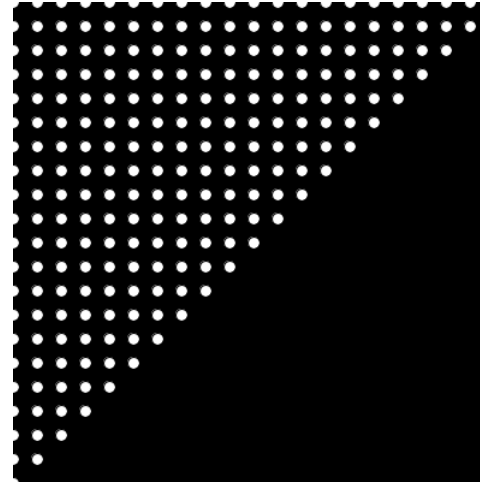
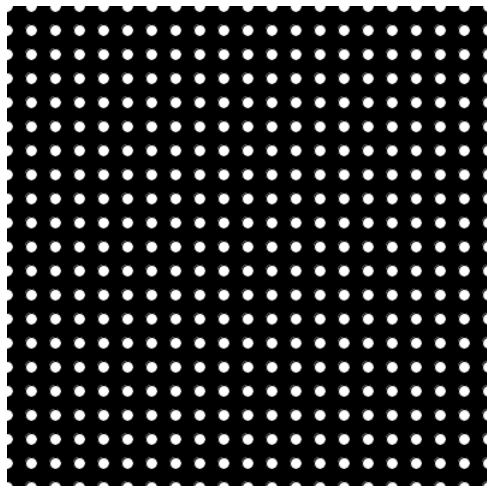
```
for (int i = 0; i <= 20; i++) {  
    for (int j = 0; j <= 20; j++) {  
        circle(i * 20, j * 20, 10);  
    }  
}
```



Example: Half Grid of Circles

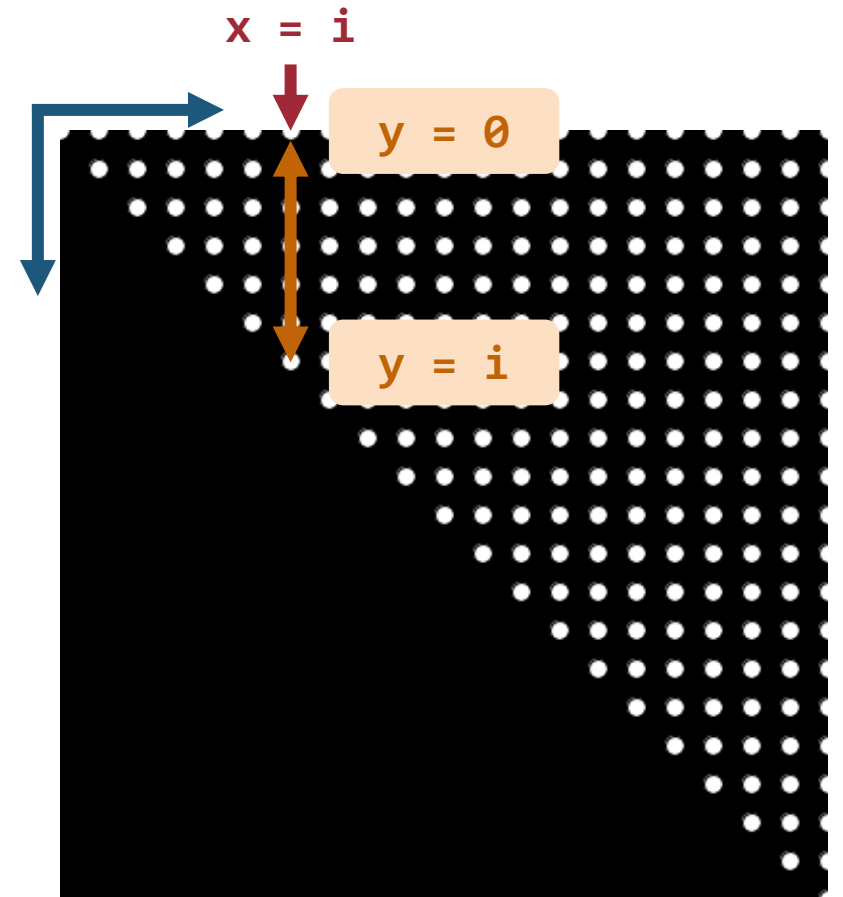
- Draw a half grid of circles using a **nested for loop**

```
for (int i = 0; i <= 20; i++) {  
    for (int j = 0; j <= 20; j++) {  
        circle(i * 20, j * 20, 10);  
    }  
}
```



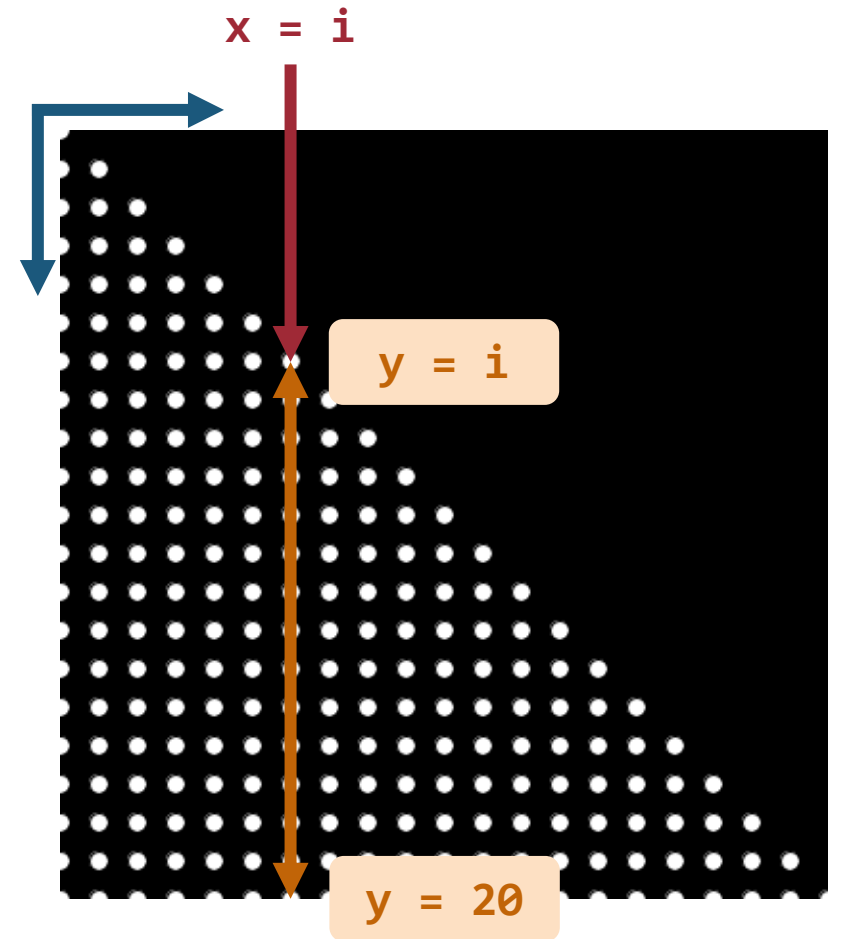
Example: Half Grid of Circles

```
for (int i = 0; i <= 20; i++) {  
  for (int j = 0; j <= i; j++) {  
    circle(i * 20, j * 20, 10);  
  }  
}
```



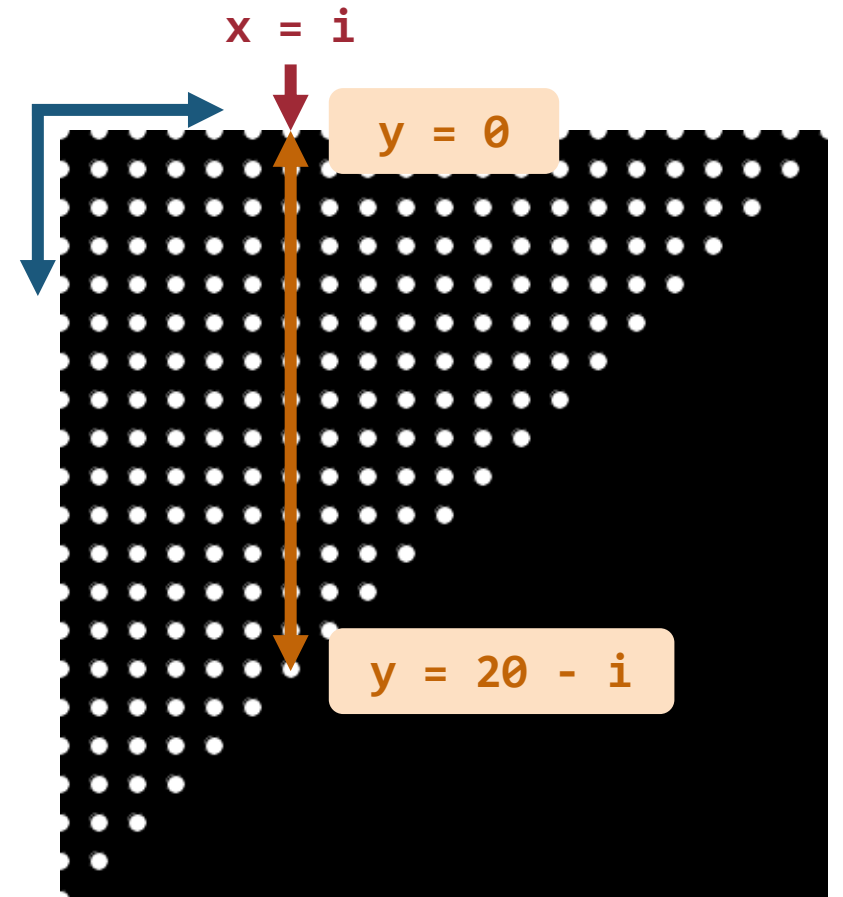
Example: Half Grid of Circles

```
for (int i = 0; i <= 20; i++) {  
  for (int j = i; j <= 20; j++) {  
    circle(i * 20, j * 20, 10);  
  }  
}
```



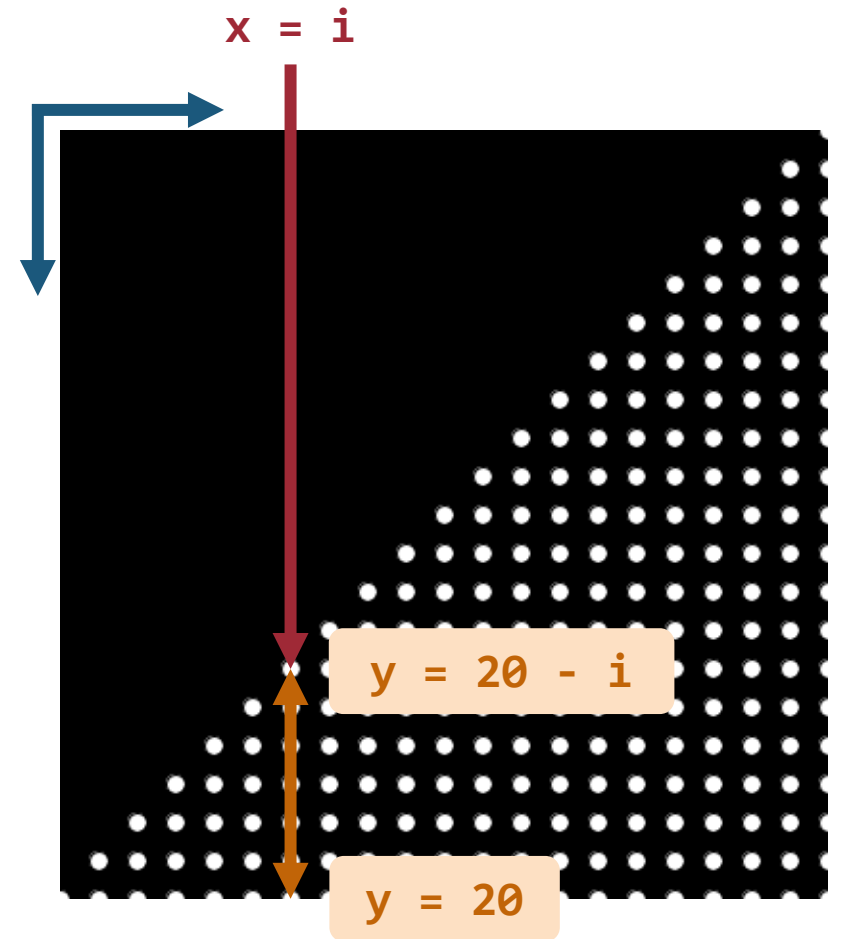
Example: Half Grid of Circles

```
for (int i = 0; i <= 20; i++) {  
  for (int j = 0; j <= 20 - i; j++) {  
    circle(i * 20, j * 20, 10);  
  }  
}
```



Example: Half Grid of Circles

```
for (int i = 0; i <= 20; i++) {  
  for (int j = 20 - i; j <= 20; j++) {  
    circle(i * 20, j * 20, 10);  
  }  
}
```



Variable Scope

```
int i = 1;

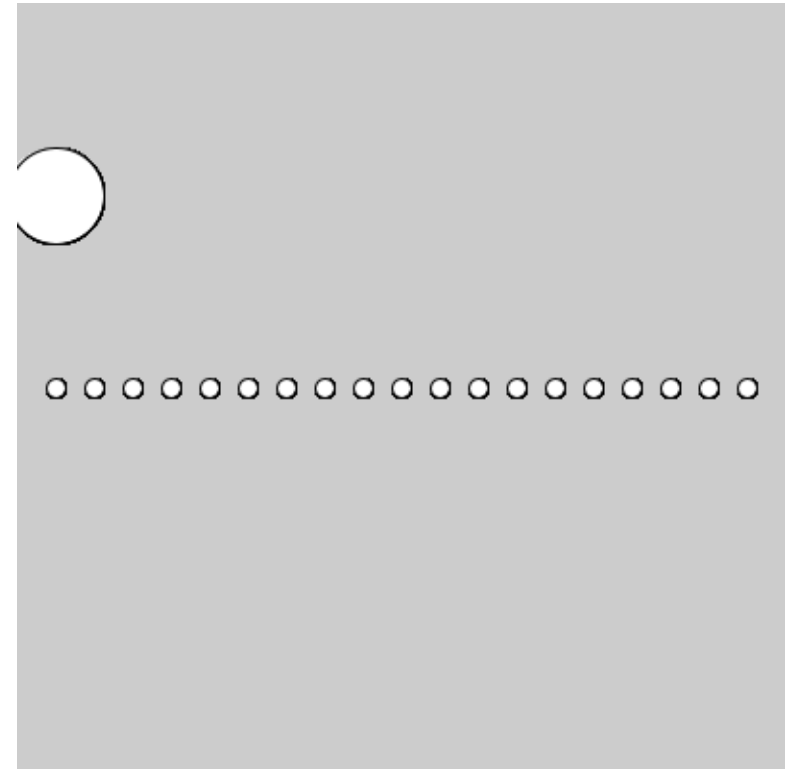
void setup() {
  size(400, 400);
}

void draw() {
  for (int i = 1; i < 20; i++) {
    circle(i * 20, 200, 10);
  }
  circle(i * 20, 100, 50);
}
```

Local variables will be deleted when we exit the block

~~New local variable i~~

~~After the for loop, i is 20~~



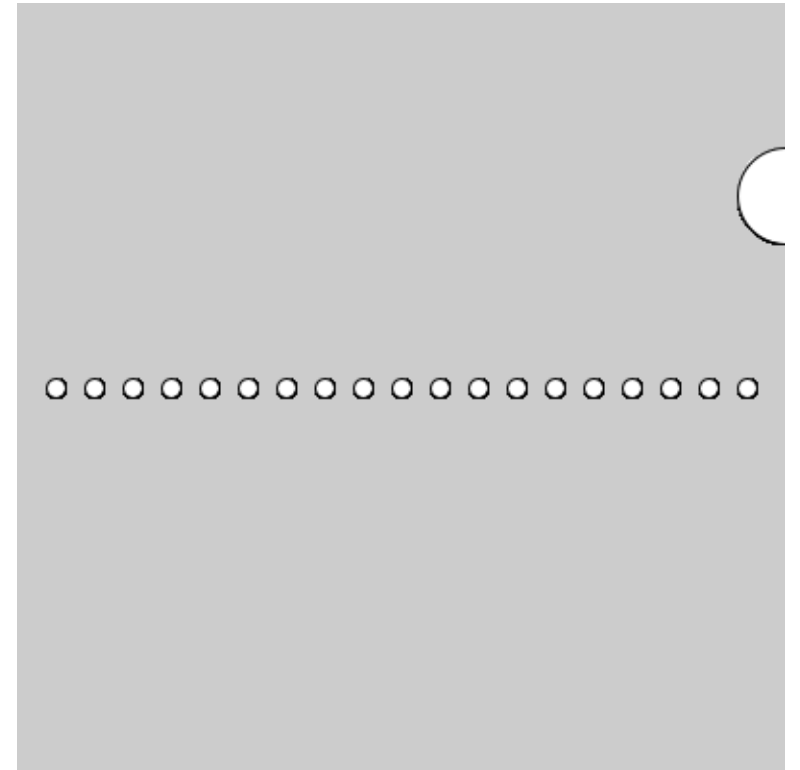
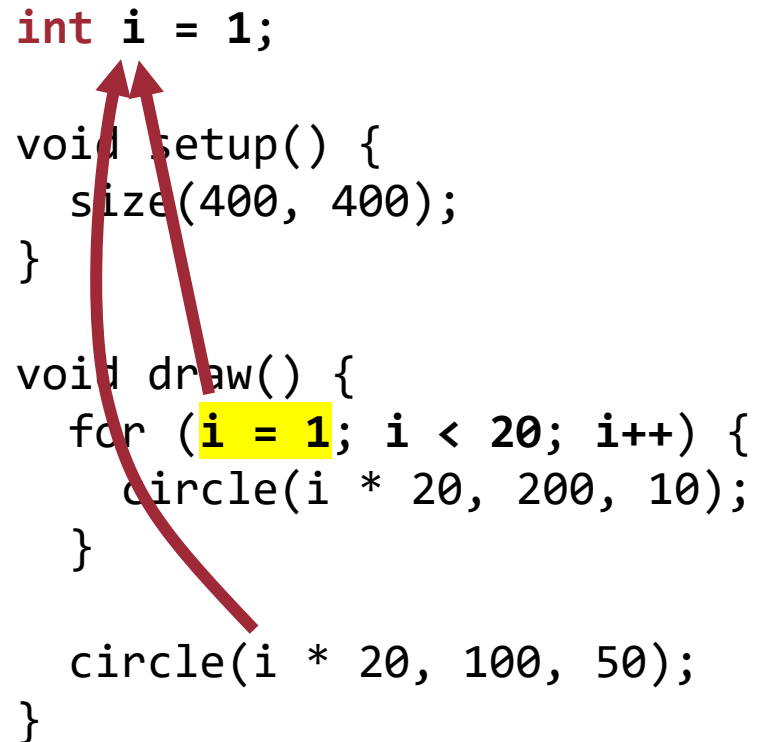
Variable Scope

```
int i = 1;

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

void draw() {
  for (i = 1; i < 20; i++) {
    circle(i * 20, 200, 10);
  }

  circle(i * 20, 100, 50);
}
```

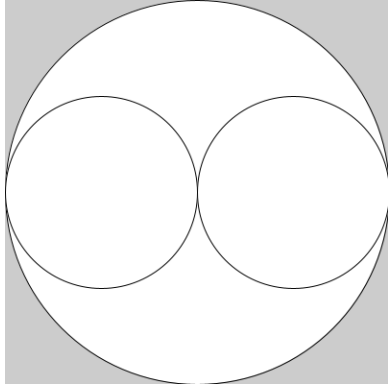


Recursion

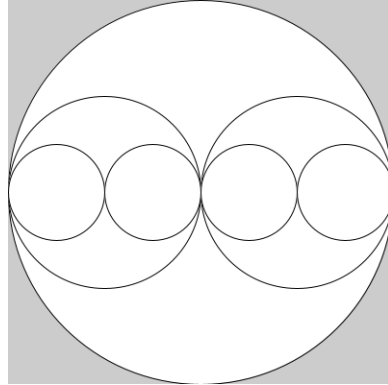
Recursion

- Recursively calling a function

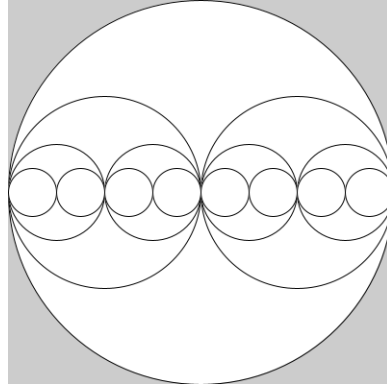
Level = 1



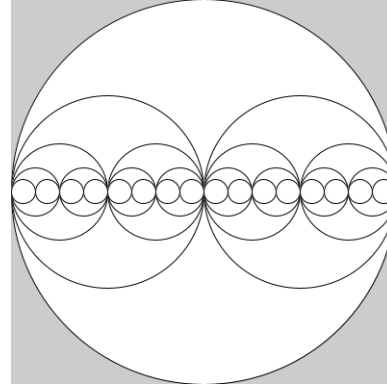
Level = 2



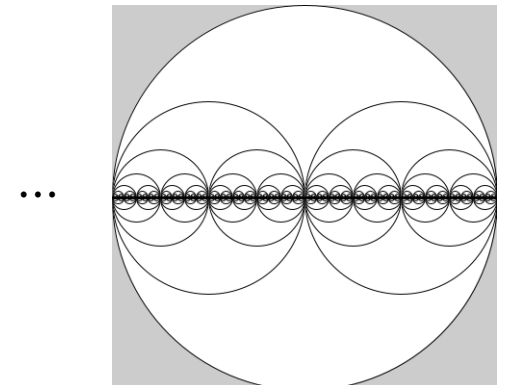
Level = 3



Level = 4

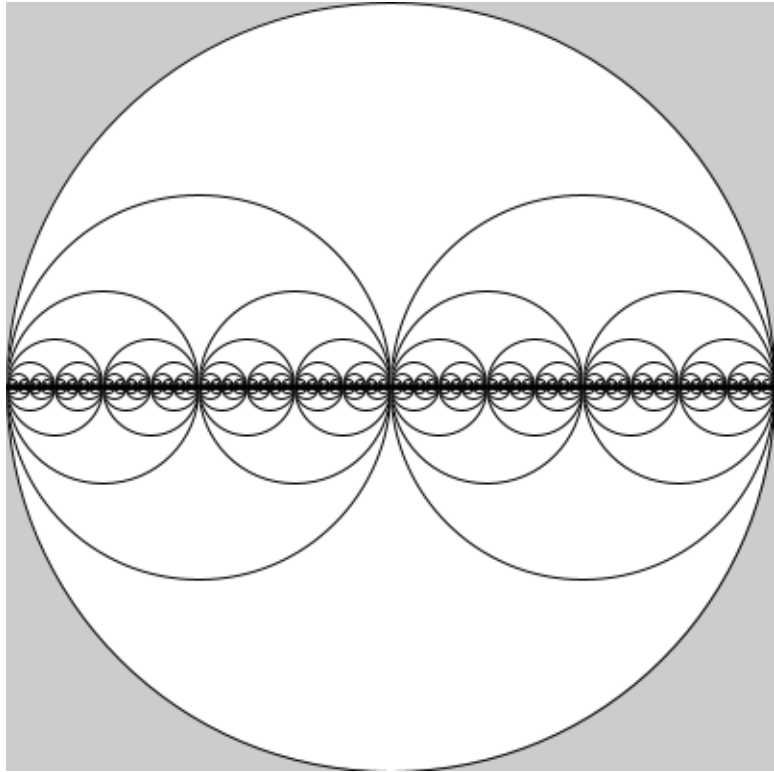


Level $\rightarrow \infty$



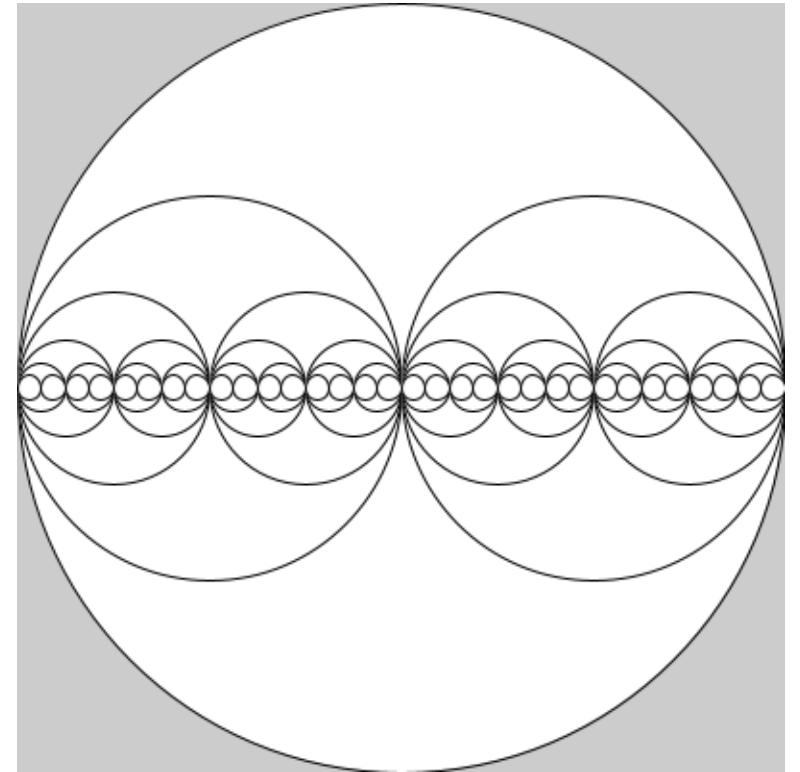
Example: Recursive Circles

```
void drawCircles(float x, float y, float w) {  
    if (w < 1) return; → Stop condition  
    circle(x - w / 4, y, w / 2);  
    drawCircles(x - w / 4, y, w / 2);  
  
    circle(x + w / 4, y, w / 2);  
    drawCircles(x + w / 4, y, w / 2);  
}  
  
void draw() {  
    circle(200, 200, 400);  
    drawCircles(200, 200, w);  
}
```



Example: Recursive Circles to a Certain Level

```
void drawCircles(float x, float y, float w, int l) {  
    if (l < 1) return;  
    if (w < 1) return;  
    circle(x - w / 4, y, w / 2);  
    drawCircles(x - w / 4, y, w / 2, l - 1);  
  
    circle(x + w / 4, y, w / 2);  
    drawCircles(x + w / 4, y, w / 2, l - 1);  
}  
  
void draw() {  
    circle(200, 200, 400);  
    drawCircles(200, 200, w, 6);  
}
```



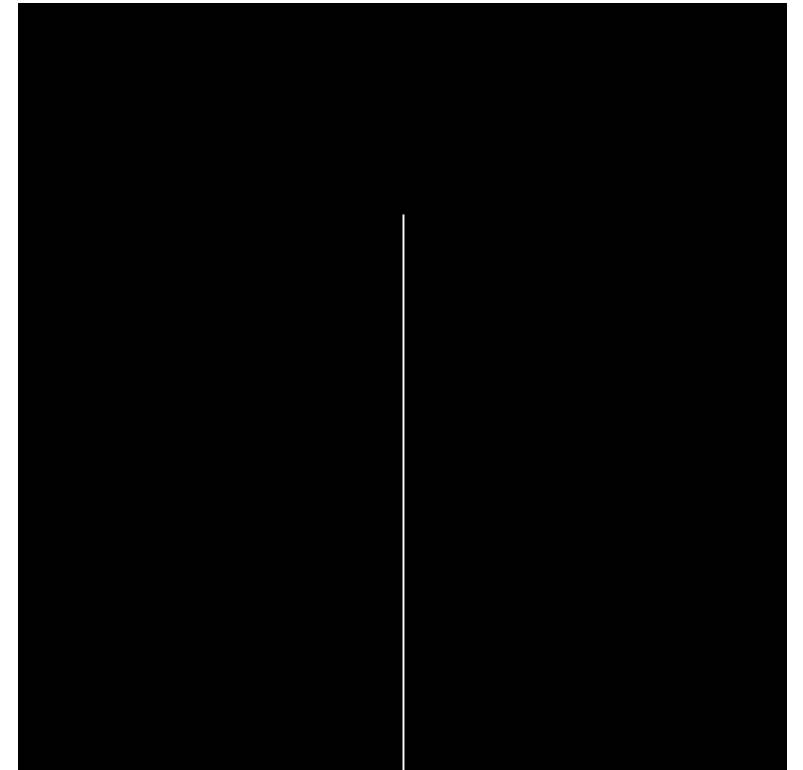
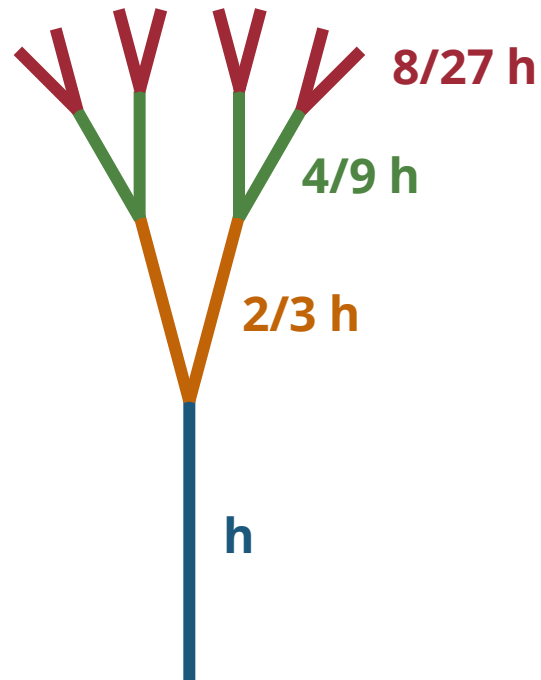
Loops vs Recursion

- **Any recursive function can be written using a loop**, and vice versa!
- Loops are usually faster and use less memory
- Recursive functions sometimes are easier to write
- When to use which?

	# of inner routines	Endpoint
<code>while</code> loop	unknown	unknown
<code>for</code> loop	known	known
Recursion	known	unknown

Example: Recursive Tree

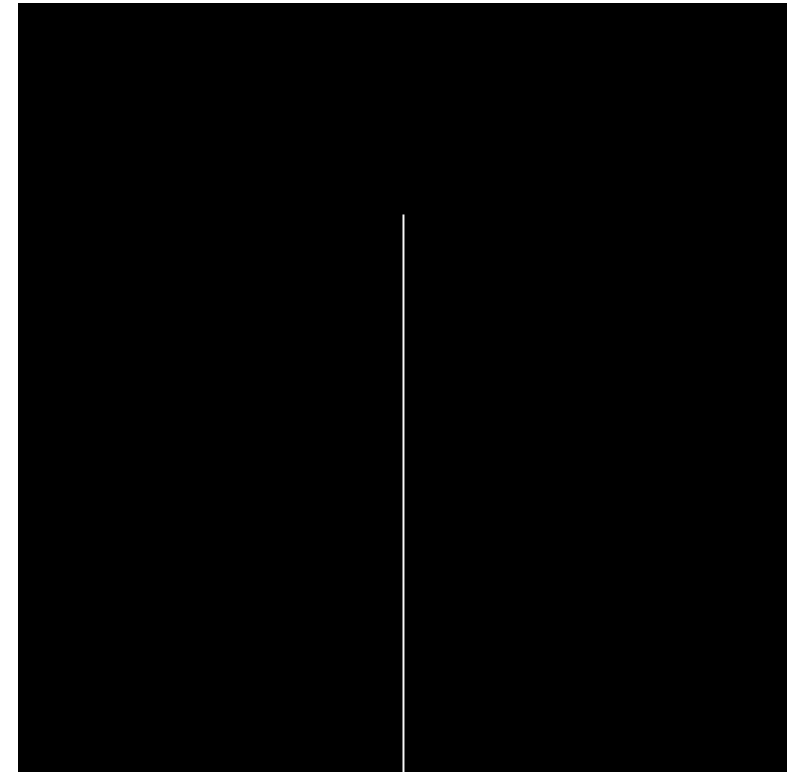
- Symmetric branches of $2/3$ length of its root
 - One branch is rotated counterclockwise for a fixed angle
 - The other branch is rotated clockwise for a fixed angle



Example: Recursive Tree

```
void branch(float h) {  
    if (h < 2) break;  
    // Right branch  
    pushMatrix();  
    rotate(theta);  
    line(0, 0, 0, -h * scale);  
    translate(0, -h * scale);  
    branch(h * scale);  
    popMatrix();  
  
    // Left branch  
    pushMatrix();  
    rotate(-theta);  
    line(0, 0, 0, -h * scale);  
    translate(0, -h * scale);  
    branch(h * scale);  
    popMatrix();  
}
```

Stop condition



Matrix Transforms

- `pushMatrix()` Push the current transformation matrix to the stack
- `popMatrix()` Pop the latest transformation matrix off the stack
- `resetMatrix()` Reset to identity matrix

- `translate(x, y)`
- `rotate(angle)`
- `scale(s)`
- `scale(x, y)`

Example: Recursive Tree

```
void branch(float h) {  
    if (h < 2) break;  
  
    // Right branch  
    pushMatrix();  
    rotate(theta);  
    line(0, 0, 0, -h * scale);  
    translate(0, -h * scale);  
    branch(h * scale);  
    popMatrix();  
  
    // Left branch  
    pushMatrix();  
    rotate(-theta);  
    line(0, 0, 0, -h * scale);  
    translate(0, -h * scale);  
    branch(h * scale);  
    popMatrix();  
}
```

Matrix
stack

