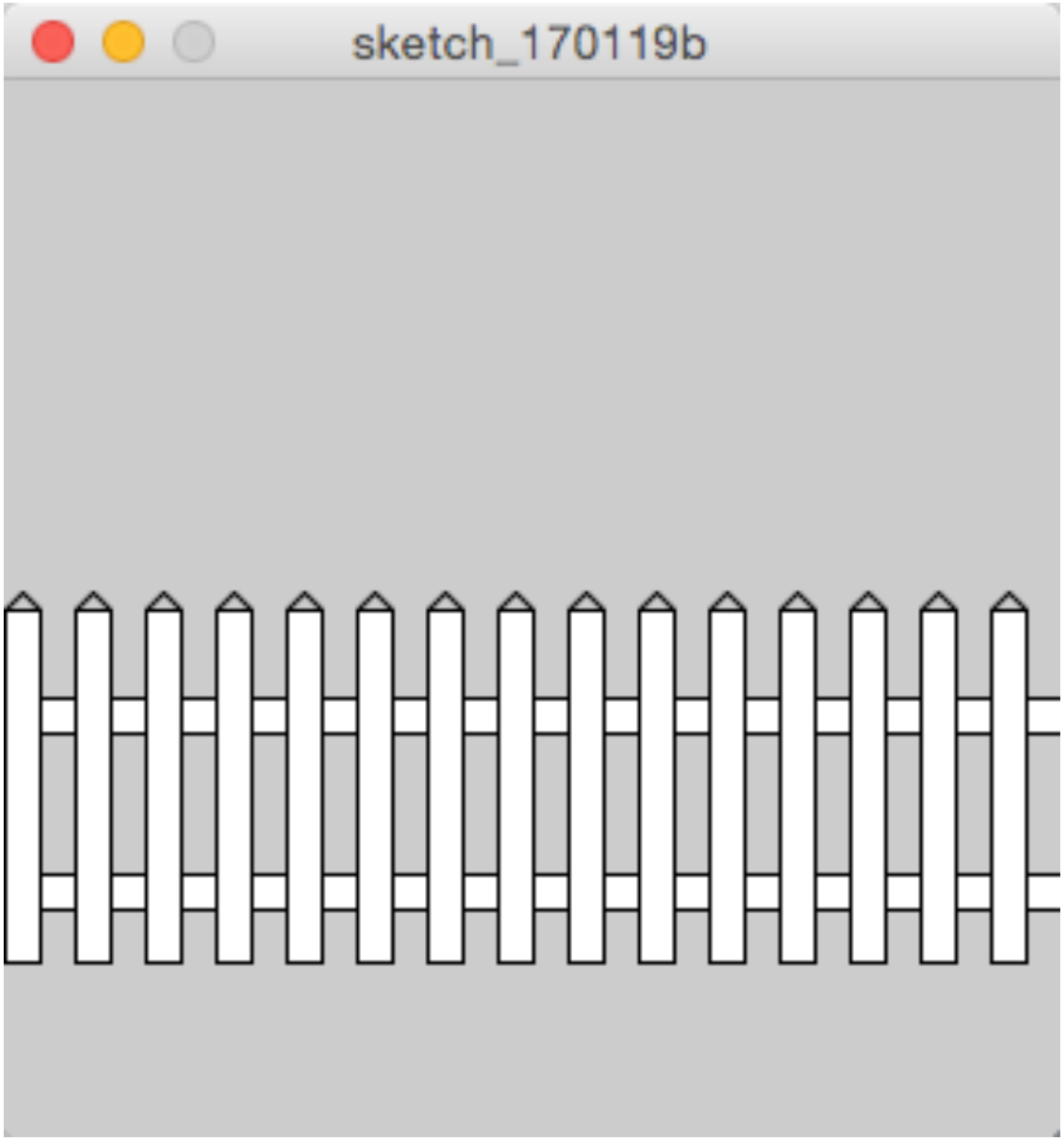
Chapter 6

Loops


sketch_170119b

```
// move to the center of the left edge
translate(0,height/2);
int picketWidth = 10;
int picketHeight = 100;

// draw the top rail, as wide as a single picket
rect(0,picketHeight/4, width, picketWidth);

// draw the bottom rail
rect(0,picketHeight*3/4, width, picketWidth);

// draw one picket
rect(0,0,picketWidth,picketHeight);
line(0,0,picketWidth/2,-picketWidth/2);
line(picketWidth,0, picketWidth/2,-picketWidth/2);

// move over and draw another picket
translate(picketWidth*2, 0);
rect(0,0,picketWidth,picketHeight);
line(0,0,picketWidth/2,-picketWidth/2);
line(picketWidth,0, picketWidth/2,-picketWidth/2);
```

```
// move to the center of the left edge
translate(0,height/2);
int picketWidth = 10;
int picketHeight = 100;
. . .
// move over and draw another picket
translate(picketWidth*2, 0);
rect(0,0,picketWidth,picketHeight);
line(0,0,picketWidth/2,-picketWidth/2);
line(picketWidth,0, picketWidth/2,-picketWidth/2);

// move over and draw another picket
translate(picketWidth*2, 0);
rect(0,0,picketWidth,picketHeight);
line(0,0,picketWidth/2,-picketWidth/2);
line(picketWidth,0, picketWidth/2,-picketWidth/2);
// repeat as necessary
```

```
// move to the center of the left edge
translate(0,height/2);
int picketWidth = 10;
int picketHeight = 100;
. . .
// looking ahead - doing it with a loop
int picketPosition = 0;
while( picketPosition < width) {
 rect(0,0,picketWidth,picketHeight);
 line(0,0,picketWidth/2,-picketWidth/2);
 line(picketWidth,0, picketWidth/2,-picketWidth/2);
 translate(picketWidth*2, 0);
 picketPosition = picketPosition + picketWidth*2;
}
```

`while` Statement

- Repeat some action as long as a specified condition is true

  while(<boolean expression>)

      <substatement>

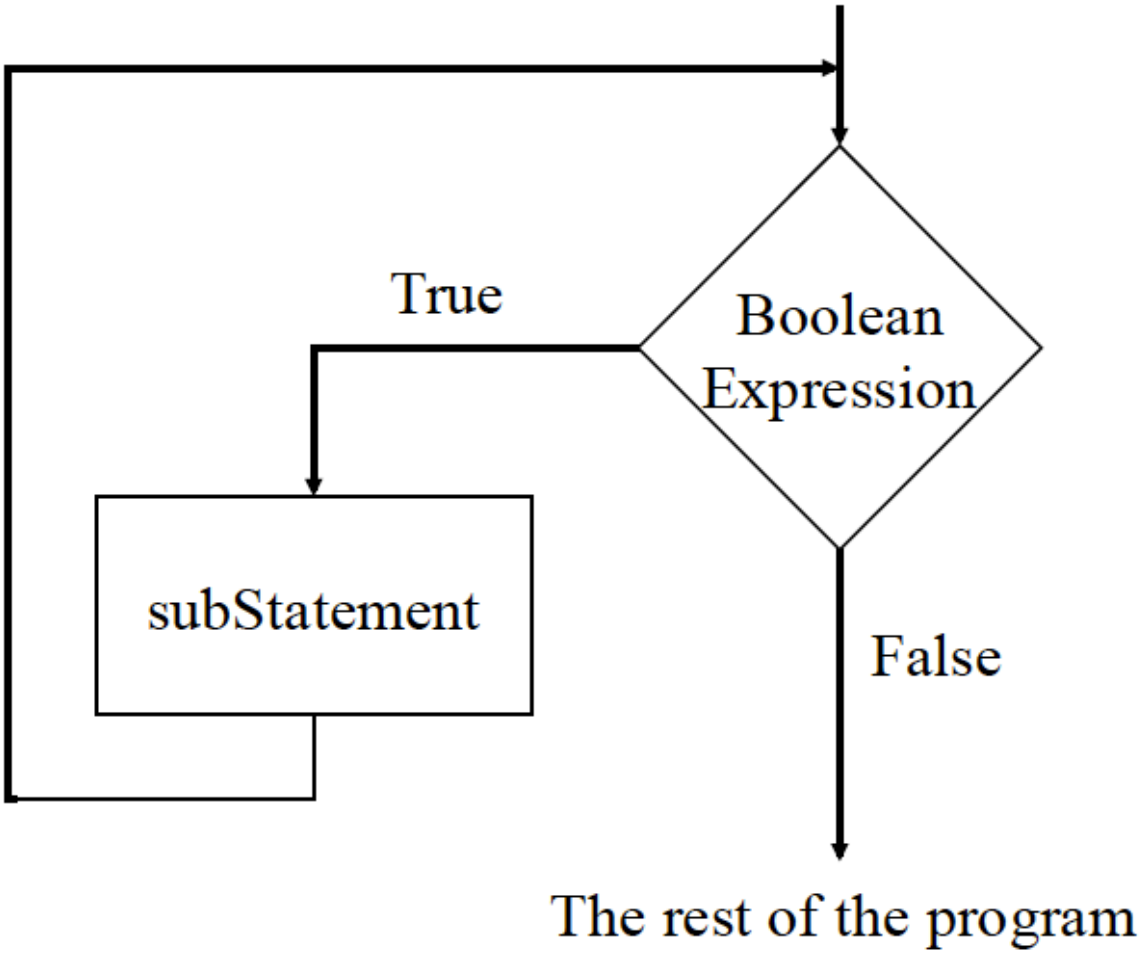
- Repeatedly execute <substatement> while

  <boolean expression> is true

  – May not execute <statement> at all

  – May never stop executing <statement> (must update something)

`while` Statement Flowchart



```
// basicWhileLoop
void setup() {
   size(400,400);
   noFill();
}
void draw() {
   background(255);
   int x = 0;
   while (x <= width) {
      ellipse(mouseX,mouseY,x,x);
      x = x + 20;
   }
}
```

```
void setup() {
   size(400,400);
}
int x = 0;
void draw() {
   background(255);
   while (x <= width) {
      rect(x,x,20,20);
      x = x + 20;
   }
   ellipse(mouseX, mouseY, 20, 20);
}
```

What is displayed after roughly 5 seconds?

A. Only a circle following the mouse.

B. A circle following the mouse and a diagonal line of squares.

C. Nothing.

D. Only a diagonal line of squares.

E. A diagonal line of squares and a trail of circles where the mouse has been.

```
int yPos = 1;
while (yPos != 50) {
    rect(xPos, yPos, 20, 40);
    yPos = yPos + 10;
}
```

How many times does this loop execute?

A. 0

B. 4

C. 5

D. 50

E. more than 50

Infinite Loops

- Occurs when loop condition (the boolean expression) is always true
- Will cause processing to hang
- You most likely will have to force quit processing if you encounter an infinite loop
- Save all of your work before trying out loops
- What is like an infinite loop that we see all the time?

```
// linesAppearingOneAtATimeNoLoop
int y = 0;
void setup() {
    size(200,200);
    frameRate(5);
    background(0);
}
void draw() {
    stroke(255);
    line(0,y,width,y);
    y = y + 10;
}
```

```
// linesAppearingOneAtATime
int endY;
void setup() {
    size(200,200);
    frameRate(5);
    endY = 0;
}
void draw() {
    background(0);
    int y = 0;
    while (y < endY) {
        stroke(255);
        line(0,y,width,y);
        y = y + 10;
    }
    endY = endY + 10;
    ellipse(mouseX, mouseY, 40, 40);
}
```

```
// linesAppearingOneAtATime
int endY;
void setup() {
    size(200,200);
    endY = 0;
}
void draw() {
    background(0);
    int y = 0;
    while (y < endY) {
        stroke(255);
        line(0,y,width,y);
        y = y + 10;
    }
    if (frameCount%10 == 0) {
        endY = endY + 10;
    }
    ellipse(mouseX, mouseY, 40, 40);
}
```

```
int endY, increment = 10;
void setup() {
   size(200,200);
   endY = 0;
}
void draw() {
   background(0);
   int y = 0;
   while (y < endY) {
      stroke(255);
      line(0,y,width,y);
      y = y + 10;
   }
   if (frameCount%10 == 0) {
      endY = endY + increment;
      if (endY > height || endY < 0)
         increment = -increment;
   }
   ellipse(mouseX, mouseY, 40, 40);
}
```

A. Lines appear starting at the top and then begin to disappear when the display is full, again from the top.

B. Lines appear starting at the top and then all disappear when the display is full, and repeat.

C. Lines appear starting at the top and then begin to disappear when the display is full, but starting from the bottom.

D. Lines appear starting at the top, filling the display just as before. No change once the display is full of lines.

```
int endY, increment = 10;
void setup() {
   size(200,200);
   endY = 0;
}
void draw() {
   background(0);
   int y = 0;
   while (y < endY) {
      stroke(255);
      line(0,y,width,y);
      y = y + 10;
   }
   if (frameCount%10 == 0) {
      endY = endY + increment;
   }
   if (endY > height || endY < 0)
      increment = -increment;
   ellipse(mouseX, mouseY, 40, 40);
}
```

A. Lines appear starting at the top and then begin to disappear when the display is full, again from the top.

B. Lines appear starting at the top and then all disappear when the display is full, and repeat.

C. Lines appear starting at the top and then begin to disappear when the display is full, but starting from the bottom.

D. Lines appear starting at the top, filling the display just as before. No change once the display is full of lines.

```
size(200,200);
stroke(255,255,0);
fill(100,0,100);
translate(width/2,height/2);
int numPetals = 16;
int petalLength = 60, petalWidth = petalLength/3;
float angle = 2*PI/numPetals;
int i = 0;
while ( i < numPetals ) {
   ellipse(0,petalLength/2,petalWidth,petalLength);
   rotate(angle);
   i = i + 1;
}
```

```
size(200,200);
stroke(255,255,0);
fill(100,0,100);
translate(width/2,height/2);
smooth();
int numPetals = 1;
int petalLength = 60, petalWidth = petalLength/3;
float angle = 2*PI/numPetals;
int i = 0;
while ( i < numPetals ) {
    ellipse(0,petalLength/2,petalWidth,petalLength);
    rotate(angle);
    i = i + 1;
}
```

What does this variation draw?

A. Nothing

B. One petal hanging down from the center of the display.

C. One petal pointing up from the center of the display.

D. One petal pointing to the right.

E. One petal pointing to the left.

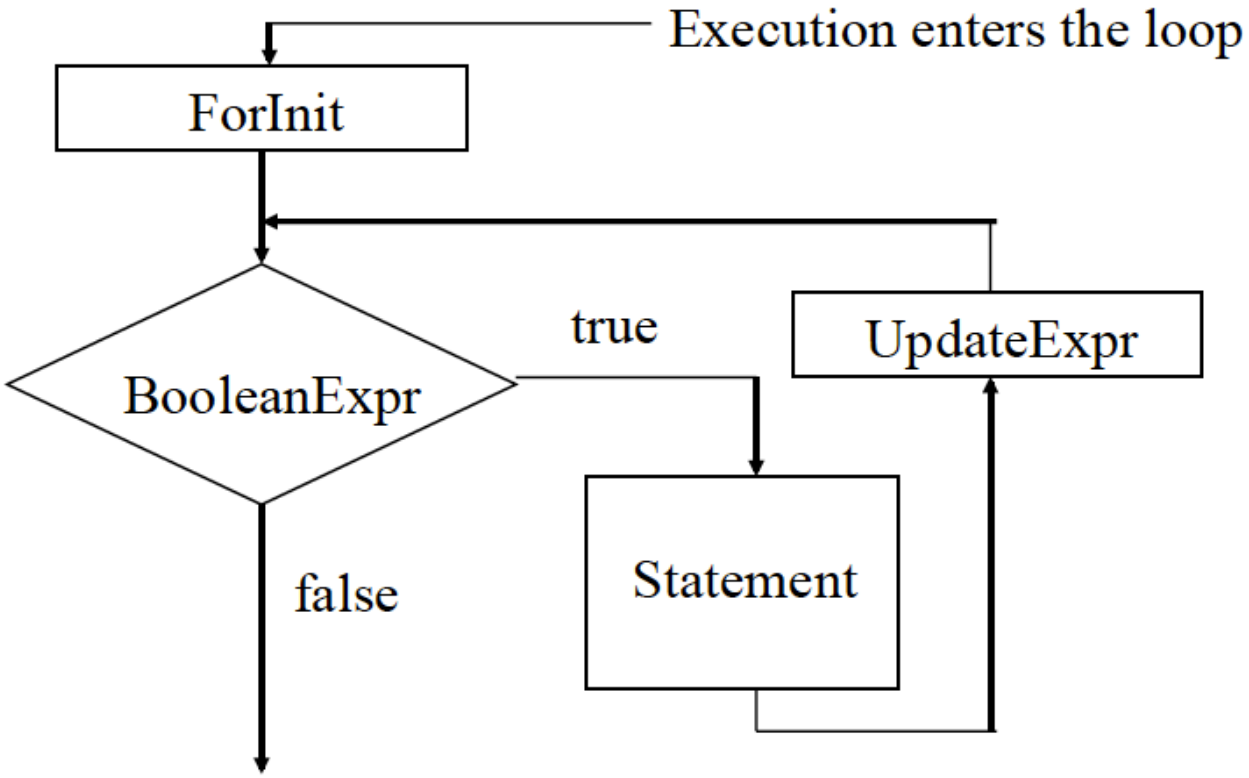
```
size(200,200);
stroke(255,255,0);
fill(100,0,100);
translate(width/2,height/2);
smooth();
int numPetals = 1;
int petalLength = 60, petalWidth = petalLength/3;
float angle = 2*PI/numPetals;
int i = 0;
while ( i < numPetals ) {
    ellipse(0,petalLength/2,petalWidth,petalLength);
    rotate(angle);
    i = i + 1;
}
```

```
// linesAppearingOneAtATime
int endY;
void setup() {
   size(200,200);
   endY = 0;
}
void draw() {
   background(0);
   int y = 0;
   while (y < endY) {
      stroke(255);
      line(0,y,width,y);
      y = y + 10;
   }
   if (frameCount%10 == 0) {
      endY = endY + 10;
   }
   ellipse(mouseX, mouseY, 40, 40);
}
```

Write a program that shows a ball moving across the screen, bouncing off the edges. When the user clicks, the ball should stop moving. When the user clicks again, the ball should resume movement.

The `for` Statement

`for` (ForInit; BooleanExpr; UpdateExpr)
    Statement



Simple `for` Statements

```
// print numbers 0 to 9
for(int value = 0; value <= 9; value = value+1) {
   println(value);
}

// draw vertical lines
for (int xPos = 0; xPos < width; xPos = xPos+xDist) {
   line(xPos,0, xPos, height);
}
```

```
...
int numPetals = 16;
int petalLength = 60, petalWidth = petalLength/3;
float angle = 2*PI/numPetals;
for(int i = 0; i < numPetals; i = i + 1 ) {
   ellipse(0,petalLength/2,petalWidth,petalLength);
   rotate(angle);
}
```

```
// linesAppearingOneAtATime
...
for(int y = 0; y < endY; y = y + 10) {
   stroke(255);
   line(0,y,width,y);
}
if (frameCount%10 == 0) {
   endY = endY + 10;
}
ellipse(mouseX, mouseY, 40, 40);
```

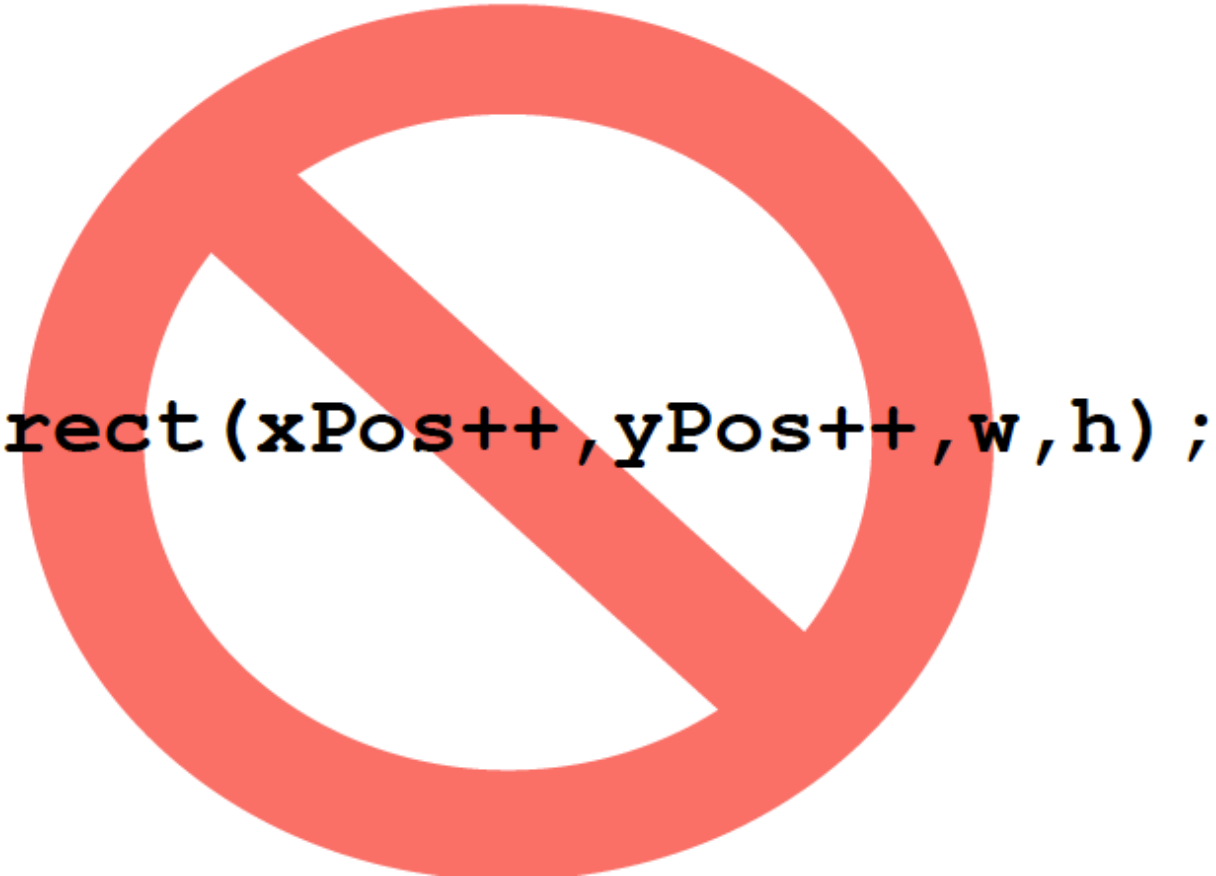
```
size(500, 500);
for(int i = 1; i <= 100; i++) {
   ellipse(random(width), random(height),
   random(width/10), random(height/10));
}
```

Use for or while loops?

- If you know how many times: use for
- Otherwise: use while
- for and while technically interchangeable

The Shortcuts

- `i++, ++i, x+=value, x-=value, x*=value, x/=value`
- avoid using x++ or ++x as a parameter to a function



Create a static sketch that fills a dark blue sky with a thousand stars.

More useful functions

- min(x, y) returns the smaller of x and y
- max(x, y) returns the larger of x and y
- constrain(a, x, y) returns
  - a if a is between x and y
  - x if a is less than x
  - y if a is greater than y

```
void setup() { size(400,400); }
void draw() {
   background(255);
   for(int xy = 0;_____; xy = xy + 10) {
      rect(xy, xy, 10, 10);
   }
}
```

What goes in the blank so the program does what is being demonstrated?

A. `xy < min(mouseX, mouseY)`

B. `xy > min(mouseX, mouseY)`

C. `xy < max(mouseX, mouseY)`

D. `xy > max(mouseX, mouseY)`

E. `xy < constrain(xy, mouseX, mouseY)`

Create a grid of squares, each randomly colored, using a for loop within a for loop.

```
size(500,500);
noStroke();
int squareSize = 20;
int numSquares = width/squareSize;
for(int i = 0; i < numSquares; i++) {
   for(int j = 0; j < numSquares; j++) {
      fill(i*255/numSquares,j*255/numSquares,0);
      rect(j*width/numSquares, i*height/numSquares,
      width/numSquares, height/numSquares);
   }
}
```

```
int squareSize = 20;
int numSquares;
void setup() {
   size(500,500);
   numSquares = width/squareSize;
   noStroke();
}
void draw() {
   for(int i = 0; i < numSquares; i++) {
      for(int j = 0; j < numSquares; j++) {
         fill(i*255/numSquares,j*255/numSquares,0);
         rect(j*width/numSquares, i*height/numSquares,
         width/numSquares, height/numSquares);
      }
   }
}
```

```
int squareSize = 20, numSquares;
void setup() {
   size(500,500);
   numSquares = width/squareSize;
   noStroke();
}
void draw() {
   for(int i = 0; ____________; i++) {
      for(int j = 0; _________________; j++) {
         fill(i*255/numSquares,j*255/numSquares,0);
         rect(j*width/numSquares, i*height/numSquares,
         width/numSquares, height/numSquares);
      }
   }
}
```

Which choice makes the filled area more or less follow the mouse?

A. `i < mouseY j < mouseX`

B. `i < mouseX j < mouseY`

C. `i < mouseY/squareSize j < mouseX/squareSize`

D. `i < mouseX/squareSize j < mouseY/squareSize`

Modify the previous program so that as long as the mouse is pressed the squares are filled with random colors and whenever the mouse is not pressed it behaves as before.

```
int squareSize = 20, numSquares;
void setup() {
  size(500,500);
  numSquares = width/squareSize;
  noStroke();
}


void draw() {
  background(255);
  for(int i = 0; i < mouseY/squareSize; i++) {
    for(int j = 0; j < mouseX/squareSize; j++) {
      fill(i*255/numSquares,j*255/numSquares,0);
      rect(j*width/numSquares, i*height/numSquares,
      width/numSquares, height/numSquares);
    }
  }
}
```

Some other variations. Make the squares on the diagonal ALWAYS white but the others change based upon the mouse position (pressed or not pressed).