Last week...
What's out here? and here? and here?
Let’s add Direct Manipulation to the 10 Print sketch:

• **In** `draw()`, **use** `translate()` **inside a geometric context.**

• **User a** `mouseDragged()` **hook to control the amount of translation (by setting global variables).**
Let’s add Direct Manipulation to the 10 Print sketch:

• In `draw()`, use `translate()` inside a geometric context.

• Use a `mouseDragged()` hook to control the amount of translation (by setting global variables).

But there’s no more pattern to see!
Workaround: just draw more of the pattern.

```java
for ( int row = 0; row < 20; ++row ) {
    for ( int col = 0; col < 20; ++col ) {
        float x = col * 40;
        float y = row * 40;

        if ( random(1) < bias ) {
            // Draw a line from NW to SE
            line( x, y, x + 40, y + 40 );
        } else {
            // Draw a line from NE to SW
            line( x + 40, y, x, y + 40 );
        }
    }
}
```
Workaround: just draw more of the pattern.

```c
for ( int row = -20; row < 40; ++row ) {
    for ( int col = -20; col < 40; ++col ) {
        float x = col * 40;
        float y = row * 40;

        if ( random(1) < bias ) {
            // Draw a line from NW to SE
            line( x, y, x + 40, y + 40 );
        } else {
            // Draw a line from NE to SW
            line( x + 40, y, x, y + 40 );
        }
    }
}
```
Workaround: just draw more of the pattern.

```java
for ( int row = -20; row < 40; ++row ) {
    for ( int col = -20; col < 40; ++col ) {
        float x = col * 40;
        float y = row * 40;

        if ( random(1) < bias ) {
            // Draw a line from NW to SE
            line( x, y, x + 40, y + 40 );
        } else {
            // Draw a line from NE to SW
            line( x + 40, y, x, y + 40 );
        }
    }
}
```

But this is inefficient, inelegant, and limited.
What we really want is a permanent way to associate pseudorandom values with points in space.

Let’s examine a simpler 1D problem.
Drawing a graph of random values

beginShape();
for( int x = 0; x < 600; ++x ) {
    vertex( x, random( 0, height ) );
}
endShape();
What's out here? What about here?
Graphing a mathematical function

```c
float myFunc( float x )
{
    float y = sin( x / 50.0 );
    return map( y, -1, 1, 0, height );
}
```
Graphing a mathematical function

```cpp
float myFunc( float x )
{
    float y = sin( x / 50.0 );
    return map( y, -1, 1, 0, height );
}
```

```cpp
... beginShape();
for( int x = 0; x < 600; ++x ) {
    vertex( x, myFunc( x ) );
}
endShape();
```
float dx;

...  
beginShape();  
for( int x = 0; x < 600; ++x ) {  
    vertex( x + dx, myFunc( x ) );  
}  
endShape();
float dx;

...  
beginShape();
for( int x = 0; x < 600; ++x ) {
   vertex( x, myFunc( x - dx ) );
}
endShape();
Unpredictable: can’t guess what the graph will look like.

Repeatable: A given $x$ will always give the same result.
Unpredictable: can’t guess what the graph will look like.

Repeatable: A given $x$ will always give the same result.

Can we create a function that’s repeatable and unpredictable?
float noise( float x ) { ... }

Return a “random” value between 0 and 1. The return value is always the same for a given input value x.
Note: `noise()` is a *smooth* function. If you zoom in enough, it changes slowly.
float noise( float x, float y ) { ... }

Return a “random” value between 0 and 1. The return value is always the same for given input values x and y.
float noise( float x, float y ) { ... }

Return a "random" value between 0 and 1. The return value is always the same for given input values x and y.

float noise( float x, float y, float z ) { ... }
Visualizing 2D noise

```cpp
void draw()
{
    for( int y = 0; y < height; ++y ) {
        for( int x = 0; x < width; ++x ) {
            float ns = 255 * noise( x, y );
            set( x, y, color( ns ) );
        }
    }
}
```
Direct manipulation of a grid

Direct manipulation of an infinite grid is challenging because we want to draw only as much of the grid as we need.
Goals

• Be able to write short sketches that use the `noise()` function.

• Understand how `noise()` works in 1D and 2D.

• Understand the difference between `random()` and `noise()`.