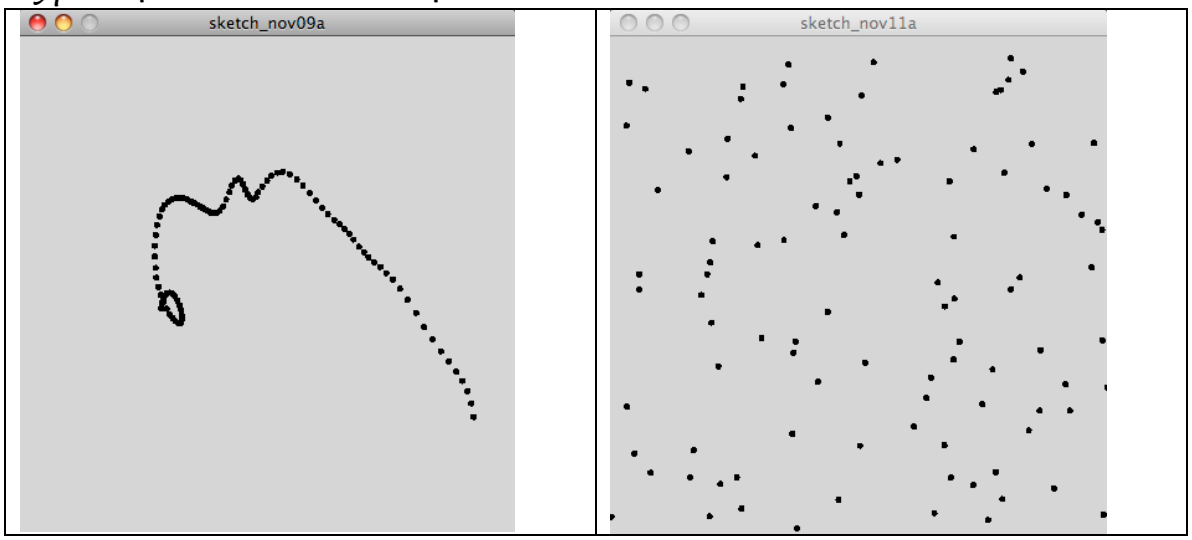
Noise

Random numbers are very useful to our work as programmers. We have used them in a number of ways. In Processing we have a luxury with the **random( )** *function. In case you forgot, the* **random( )** *function returns* **float** *values. There are two signatures for calling* **random( )**:

1. **random(float)** *which returns a* **float** *value between zero and up to but not including the value of the parameter*
2. **random(float, float)** *which returns a float value between the first parameter and up to but not including the value of the second parameter*

In other programming languages (c, c++. Java), the **random** *function behaves very differently. It usually returns a decimal value between zero and up to but not including one. We have to add our own arithmetic to shift the random value into the range we need.*

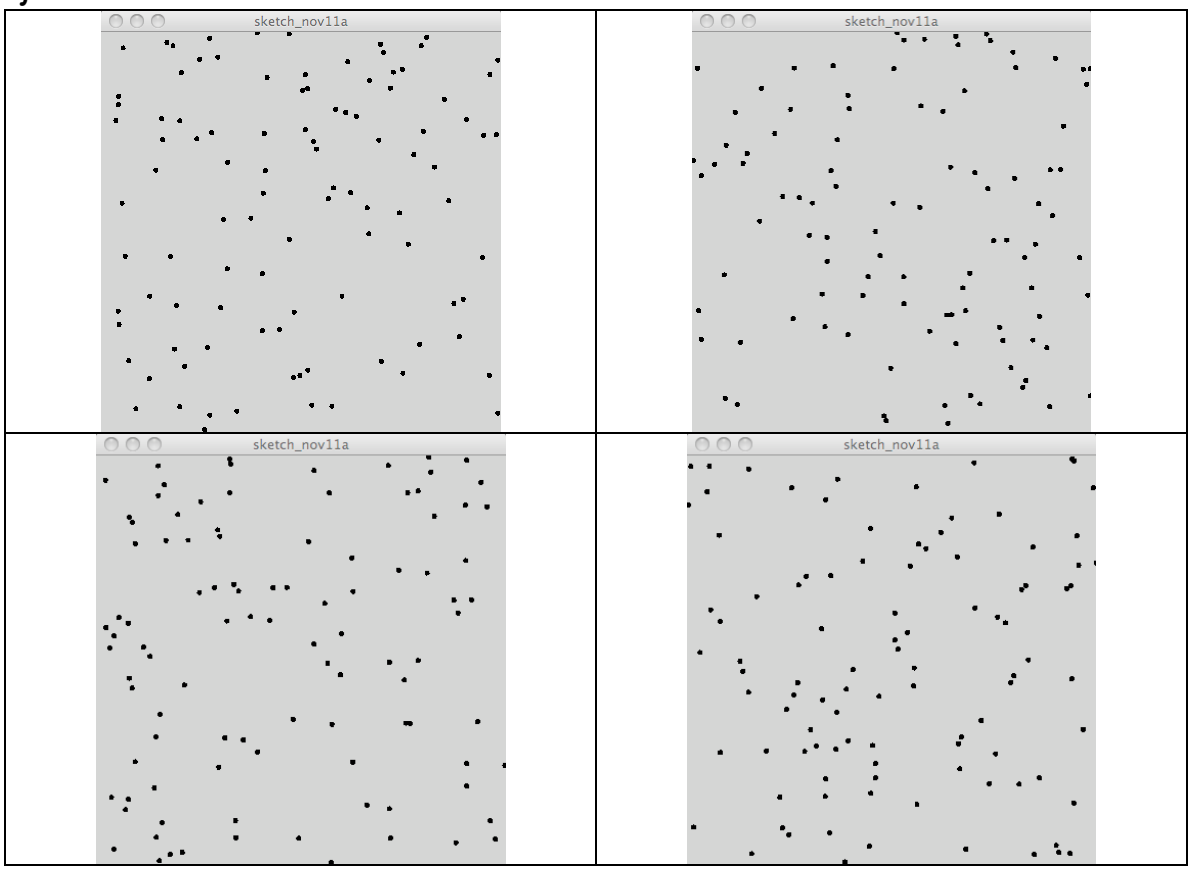
The random values returned by the function have no obvious relationship to the other random values. If they did, we could "cheat" if we used the function in writing a game. But, what if we wanted the next random number to be related to the previous one is some manner? Below are two sets of 100 points using randomly generated values of **x** *and* **y** *by two different types of random value functions:*



By inspection, can you determine which plot was generated by the **random( )** *function that we have used in class?*

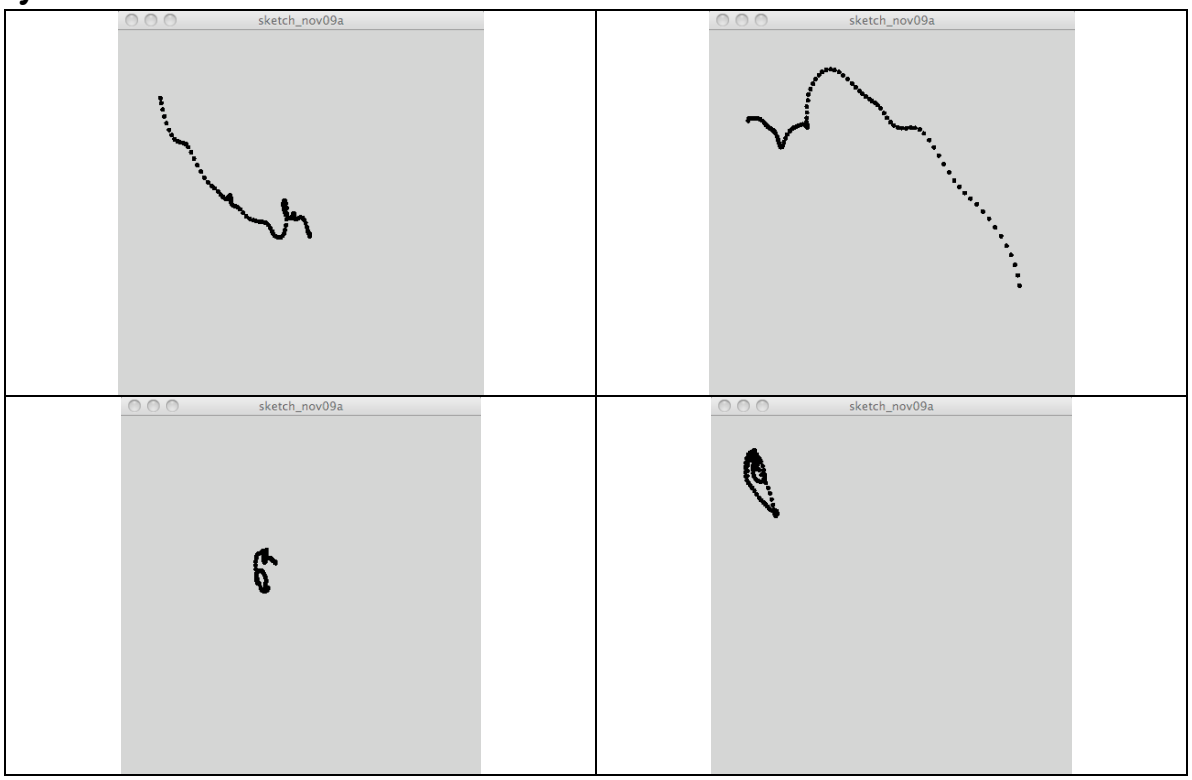
Hopefully you chose the right plot. Here is the code and output for four runs of the code that generated the plot on the right:

```
void setup( )
{
  size( 400, 400 );
  strokeWeight( 4 );
  background(204);
}
void draw()
{
  for( int i = 0; i < 100; i++)
  {
    float x = random( width) ;
    float y = random( height );
    point( x,y );
  }
  noLoop();
}
```



Here is the code and output of four runs for the code that generated the output on the left:

```
float xoff = 0.0;
float yoff = 0.1;
void setup( )
{
  size( 400, 400 );
  strokeWeight( 4 );
  background(204);
}
void draw()
{
  for( int i = 0; i < 100; i++)
  {
     xoff = xoff + .01;
     yoff = yoff + .02;
     float x = noise(xoff) * width;
     float y = noise(yoff) * height;
     point( x,y );
  }
  noLoop();
}
```



The **x** *and* **y** *values for these points were genereated using random values returned by Processing's* **noise()** *function. In these plots we can see that each newly returned value is related in some manner to the previous value.*

Processing's documentation on the noise function gives you some background. The technique (algorithm) used by the function is named after the person, Ken Perlin who developed it. According to Wikipedia, he won an Academy Award for Technical Achievement from the Academy of Motion Picture Arts and Sciences in 1997 for this contribution to the 1982 movie Tron. The technique is often called Perlin noise. It is used today to generate such forms as clouds, water, and land in computer generated images (CGI);

Here is the 100 actual values returned by Processing's **noise( )** *function from a run of the code shown above:*

0.47986597	0.38734418	0.34457532	0.2859778	0.3384362
0.47786325	0.38497344	0.34022632	0.28613523	0.33882996
0.47457564	0.38241827	0.33626893	0.28753534	0.33896476
0.47013262	0.378594	0.33079487	0.28880042	0.33845404
0.4649182	0.3734412	0.3260786	0.29141024	0.3378385
0.4584306	0.36913756	0.3204948	0.29445785	0.33660161
0.45160374	0.36484095	0.31510687	0.29743034	0.334685
0.44444704	0.36187142	0.31108323	0.30148578	0.33243567
0.4371369	0.35876182	0.30673426	0.30505294	0.33039266
0.43020386	0.35682046	0.30391735	0.3093356	0.32892138
0.4227469	0.35456306	0.30071595	0.3136343	0.3276147
0.41610792	0.3529221	0.2976696	0.31714797	0.32655683
0.41012445	0.35262266	0.29562092	0.32103762	0.3257497
0.40498042	0.35166764	0.29294428	0.32395038	0.32526538
0.40107957	0.3516546	0.29133168	0.32705477	0.32507545
0.397246	0.35109547	0.28940663	0.32991102	0.32495114
0.3946676	0.35063747	0.28756848	0.3319153	0.32490528
0.3928898	0.351169	0.28688917	0.33411282	0.32488447
0.3911529	0.3496812	0.285765	0.33558744	0.32490966
0.3889251	0.34799734	0.2858417	0.33721545	0.32492584

Processing's **noise( )** *function returns values between zero and but not including 1.00 So our code must take this value and scale it and shift it into the range we need. In the code above the values needed are between zero and* **width** *or* **height** *so we do not have to shift the returned value. But we do have to scale it. Scaling is done in this code:*

```
float x = noise(xoff) * width;
float y = noise(yoff) * height;
```

The value returned by **noise( )** *is scaled by multiplying it by the* **width** *or* **height** *of the window. Since the returned value is between zero and 100, this code is essentially using the returned value as a percentage of the* **width** *or* **height**.

The value returned by the **noise( )** *function differs for two reasons. Each different run of the program causes a different sequence of values. Here are two runs of a slightly modified program. This program returns only ten values and the parameter for the call is a* **constant**.

```
void draw(  )
{
  for( int i = 0; i < 10; i++)
  {
     println( noise( 0.0 ) );
  }
  noLoop(  );
}
```

run #1	run #2
0.92410743	0.8696755
0.92410743	0.8696755
0.92410743	0.8696755
0.92410743	0.8696755
0.92410743	0.8696755
0.92410743	0.8696755
0.92410743	0.8696755
0.92410743	0.8696755
0.92410743	0.8696755
0.92410743	0.8696755

As you can see, each run generated a different random value but the ten returned values in each run are the same. The difference between the returned values in an individual run is controlled by the parameter of the call. A constant returns the same value.

But what happens when the parameter's value is altered by different amounts. Here is three runs and the returned values of another slightly modified version of the code that uses different parameters for the call of the **noise( )** *function .*

run #1 change = 0.1	**run #2 change = 0.01**	**run #3 change = 0.001**

void draw()
{
float offSet = 0.0;
for(int i = 0; i < 10; i++)
{
offSet = offSet + 0.1;
println(noise(offSet));
 	
}
noLoop();
}
	void draw()
	{
	float offSet = 0.0;
	for(int i = 0; i < 10; i++)
	{
	offSet = offSet + 0.01;
	println(noise(offSet));
	
	}
	noLoop();
	}
		void draw()
		{
		float offSet = 0.0;
		for(int i = 0; i < 10; i++)
		{
		offSet = offSet + 0.001;
		println(noise(offSet));
		
		}
		noLoop();
		}

0.67037326 0.55882114 0.472238 0.39287695 0.30840316 0.3161984 0.2899496 0.24448672 0.18861179 0.15002044	0.20696281 0.20643763 0.20557557 0.2044105 0.20304315 0.20134191 0.19955176 0.19767511 0.1957582 0.19394018	0.32957157 0.32957587 0.3295873 0.3295977 0.32961744 0.32964167 0.32967186 0.3296954 0.32972658 0.3297699

We can see that smaller incremental changes in the parameter variable **offSet** *generate smaller increments in the values returned by the* **noise( )** *function.*

A detailed and in depth look at **noise( )** *is beyond a first look that is being presented in class and in these notes. And, unlike the* **rect( )** *or* **point( )** *functions, where a single reading and use makes us an expert in using them, understanding the behavior of the* **noise( )** *function takes a great deal of thought, study, and time working with it in code. The work should take into account that there are three signatures for calling* **noise( )**. *These values relate to the* **x**, **y**, *and* **z** *coordinates in* **noise space**. *There is another function in Processing that is closely related to the* **noise( )** *function -* **noiseDetail( )**. *This function gives us more control over the values returned by the function.*

Use **noise( )** *and work with it as you see fit, but watch your time.*