Unidad 15

Matemáticas: Aleatoriedad

Elementos que se introducen en esta Unidad:
`random(), randomSeed(), noise(), noiseSeed()`

Lo aleatorio tiene una fundamental importancia en nuestra historia, especialmente en el arte moderno. Con acciones como arrastrar objetos, romperlos, y demás, los artistas generar una pieza con cierto nivel de valor aleatorio. Por ejemplo, una técnica muy usada de este estilo es llenar pequeñas bolsas con pintura y ubicarlas en un lienzo. Posteriormente, se lanzan dardos que rompen estas bolsas y dejan caer la pintura de una forma completamente inesperada, y con resultados de texturas muy interesantes. Es así, como se presenta un contraste muy elevado entre una estructura rígida y el completo caos.

-Valores Inesperados

La función `random()` permite devolver un valor completamente aleatorio e inesperado de un rango especificado por parámetros:

```
random(alto)
random(bajo, alto)
```

La función regresa un valor aleatorio decimal (`float`) desde el 0 hasta el parámetro *alto*. Otra forma de ingresar parámetros a la función es a través de dos parámetros en lugar de uno solo. El valor *bajo* será el primer valor del rango, y el valor *alto* será el último. Eso significa que si ingresamos el valor 3 primero, y luego el valor 5, nos devolverá un valor aleatorio decimal entre, por supuesto, 3 y 5. Además, acepta perfectamente valores negativos.
El valor siempre será del tipo `float`. Si se deseara obtener un valor aleatorio pero entero (`int`), deberemos recurrir a la función `int()` para convertirlo.

```
float f = random(5.2);        //Asigna a  f un valor decimal entre 0 y 5.2
int i = random(5.2);          //ERROR! No se puede asignar un valor aleatorio a
                              //una variable int
int j = int(random(5.2));     //Asigna a j un valor entero entre 0 to 5
```

Ya que, como queda dicho, los valores que devuelve valores impredecibles, cada vez que el programa se ejecuta obtendremos diferentes resultados. Este número puede ser utilizado para controlar algún aspecto del programa.

```
smooth();
strokeWeight(10);
stroke(0, 130);
line(0, random(100), 100, random(100));
line(0, random(100), 100, random(100));
line(0, random(100), 100, random(100));
line(0, random(100), 100, random(100));
line(0, random(100), 100, random(100));
```

Si utilizamos la versión de `random()` que acepta dos parámetros, tendremos mas control sobre algunas propiedades.

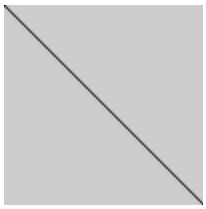
```
smooth();
strokeWeight(20);
stroke(0, 230);
float r = random(5, 45);
stroke(r * 5.6, 230);
```

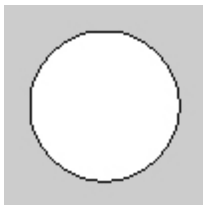
```
line(0, r, 100, random(55, 95));
r = random(5, 45);
stroke(r * 5.6, 230);
line(0, r, 100, random(55, 95));
r = random(5, 45);
stroke(r * 5.6, 230);
line(0, r, 100, random(55, 95));
```

Podemos, además, crear combinaciones. Por ejemplo, podemos utilizar un ciclo `FOR` para crear texturas muy interesantes y sacar mayor provecho al la aleatoriedad.

```
background(0);
smooth();
stroke(255, 60);
for (int i = 0; i < 100; i++) {
      float r = random(10);
      strokeWeight(r);
      float offset = r * 5.0;
      line(i-20, 100, i+offset, 0);
}
```

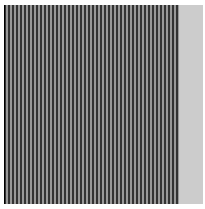
Los valores aleatorios determinan el flujo de un programa. Por lo tanto, podemos implementarlos junto con alguna expresión relacional, por ejemplo un IF. En el siguiente ejemplo, se evalúa si el valor es mayor o menor que 50, y dibuja una figura diferente dependiendo el caso:

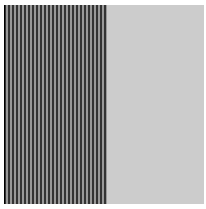




```
float r = random(100);
if (r < 50.0) {
      line(0, 0, 100, 100);
} else {
      ellipse(50, 50, 75, 75);
}
```

También podemos utilizarlo para limitar la frecuencia con la que se dibuja alguna figura:





```
int num = int(random(50)) + 1;
for (int i = 0; i < num; i++) {
      line(i * 2, 0, i * 2, 100);
}
```

A veces deseamos conseguir un valor aleatorio, pero queremos forzar a que siempre sea el mismo. Para eso se utiliza la función `randomSeed()`. Esta es la manera en la que produciremos dichos números:

```
randomSeed(valor)
```

El parámetro *valor* tiene que ser del tipo entero (`int`). Utilizar el mismo número producirá siempre el mismo orden de valores. Es decir, mantendremos un valor aleatorio, pero forzaremos a que siempre se trate del mismo valor.

```
int s = 6;                    //Valor del Seed
background(0);
smooth();
stroke(255, 60);
randomSeed(s);          //Produce el mismo número cada vez
for (int i = 0; i < 100; i++) {
```

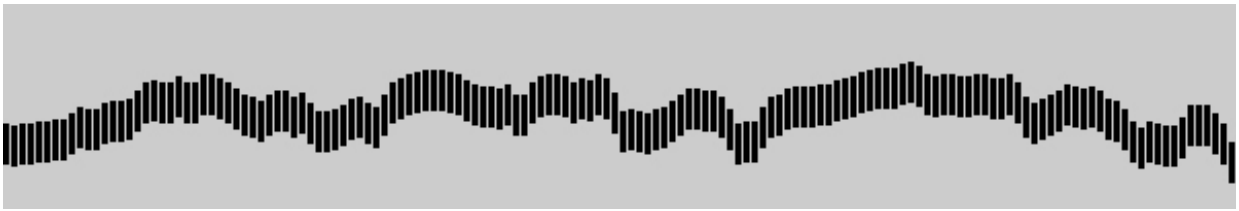
```
            float r = random(10);
            strokeWeight(r);
            float offset = r * 5;
            line(i-20, 100, i+offset, 0);
      }
```

-Noise

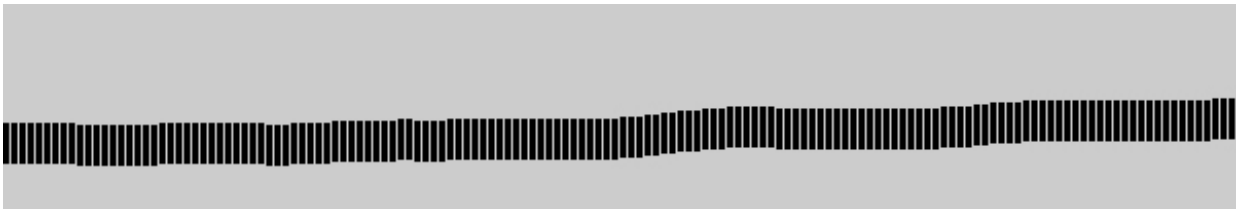
La función `noise()` nos permite tener un amplio control en los valores aleatorios que creamos. Principalmente, se utiliza para crear movimientos orgánicos. Hay tres versiones de la función, acepta un parámetro, dos parámetros e incluso tres parámetros:

```
noise(x)
noise(x, y)
noise(x, y, z)
```

La versión de un solo parámetro permite crear una simple secuencia de números aleatorios. Por ende, más cantidad de parámetros permiten crear mas dimensiones de aleatoriedad. Dos parámetros pueden ser usados para crear texturas en dos dimensiones, mientras que tres parámetros pueden utilizarse para crear figuras en tres dimensiones o movimientos de animación variados. Independientemente del valor ingresado, siempre se devuelve un valor entre 0.0 y 1.0. Si se desea otra clase de valor, se puede recurrir a alguna de las operaciones aritméticas ya vistas.
Los números generados por esta función, pueden ser utilizados para producir leves cambios dentro de un mismo rango. Por ejemplo, movimientos muy cortos pero aleatorios de la misma figura (que sumados a valores seno y/o coseno producirán una variedad de movimientos orgánicos). Por lo general, los mejores valores para trabajar son los que van de entre 0.005 y 0.3, pero depende de que es lo que se quiera conseguir. Además, posee en adición la función `noiseSeed()` la cual trabaja exactamente igual que `randomSeed()`. En el siguiente ejemplo se incorporar, también, una variable *inc*, la cual se utiliza para controlar las diferencias entre valores aleatorios:



`inc = 0.1`

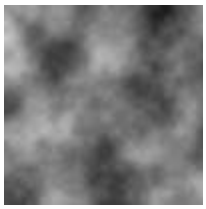


`inc = 0.01`

```
size(600, 100);
float v = 0.0;
float inc = 0.1;
noStroke();
fill(0);
noiseSeed(0);
for (int i = 0; i < width; i = i+4) {
      float n = noise(v) * 70.0;
      rect(i, 10 + n, 3, 20);
```

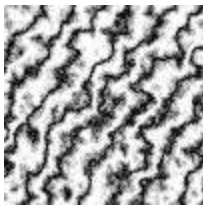
```
      v = v + inc;
}
```

Si se utiliza el segundo parámetro, se nos da la posibilidad de crear texturas en dos dimensiones. En el siguiente ejemplo, se utiliza junto con variaciones de la variable *inc* y un ciclo FOR pueden utilizarse para crear sistemas muy interesantes:



```
float xnoise = 0.0;
float ynoise = 0.0;
float inc = 0.04;
for (int y = 0; y < height; y++) {
      for (int x = 0; x < width; x++) {
            float gray = noise(xnoise, ynoise) * 255;
            stroke(gray);
            point(x, y);
            xnoise = xnoise + inc;
      }
      xnoise = 0;
      ynoise = ynoise + inc;
}
```

Ahora bien, las combinaciones con las estructuras ya vistas y el efecto de la aleatoriedad del tipo *noise* son ilimitadas y de mucha variedad. Un efecto interesante es combinarlo con un valor de seno para generar una textura con turbulencia.



```
float power = 3;  //Intensidad de la turbulencia
float d = 8;      //Densidad de la turbulencia
noStroke();
for (int y = 0; y < height; y++) {
      for (int x = 0; x < width; x++) {
            float total = 0.0;
            for (float i = d; i >= 1; i = i/2.0) {
                  total += noise(x/d, y/d) * d;
            }
            float turbulence = 128.0 * total / d;
            float base = (x * 0.2) + (y * 0.12);
            float offset = base + (power * turbulence /
                                    256.0);
            float gray = abs(sin(offset)) * 256.0;
            stroke(gray);
            point(x, y);
      }
}
```