pRNGs

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

– John Von Neumann (1951)

pRNGs

A pseudo random number generator (**pRNG**) is a function that generates one number on each invocation. Good pRNGs produce sequences of number that pass many statistical tests showing **there are no concerning** patterns and correlations among the values. This property alone would be sufficient to call them *random number generators*.

We will call this function *Random*(), its return value is usually u.

Random() produces a $\mathfrak{Real}$ number in the open interval $(0, 1)$.

In practice, not all languages and pRNG libraries have a *Random*() equivalent. The best do.

Worst case: write your own wrapper function to guarantee the $\mathfrak{Real}$ $(0, 1)$ result.

The minimal return type for *Random*() is an IEEE-754-1985 "`double`".

Seeds

Before calling *Random*() in programming environments, the wise developer **seeds** the pRNG's underlying state machine with a value, typically a non-negative integer (although libraries can vary in the extent of different acceptable seed types).

There is no common convention for the name of the "seeding" function, you'll have to look it up for your environment.

The seed value dictates the sequence of values generated by subsequent calls to *Random*(); at least until it is re-seeded.

Seeds

Why do we say **pseudo**-RNG?

While

0.3230	0.540	0.764	0.920	0.70	0.93	0.430	0.3164
0.222	0.601	0.693	0.696	0.917	0.823	0.880	0.096
0.570	0.638	0.930	0.1027	0.780	0.3278	0.916	0.065

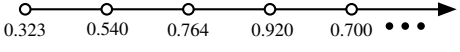
may look like a sequence of random numbers, if you can reproduce them at any time using a seed value 423 in Python3's standard `random` module — they **are decidedly not** random.

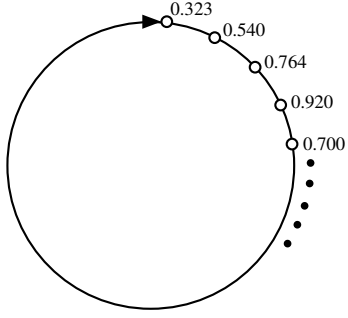
ρ — the pRNG's period

Random() can be called an unlimited number of times, but since we are working with finite digital machines, the values themselves cannot be unlimited.
Eventually, pRNGs will **"loop" around to the first value produced** (after seeding) and then you will notice the sequence will repeat itself.

The values from *Random*() **don't run in a line** with an arrow on the end, instead they **run in a circle**, eventually overlapping the starting point.

NOT A LINE . . .





IT'S A CIRCLE

Example: C++

```
#include <cstdlib>
#include <iostream>

using namespace std;

// man pages:  srand48(3) and drand48(3)
int main( int argc, char* argv[] )
{
    srand48( 423 );
    /***
     * BEWARE: drand48 returns values in [0,1), not (0,1)
     */
    cout << drand48() << " "
         << drand48() << " "
         << drand48() << endl;
    return 0;
}
```

Example: Java

```
import java.util.Random;

public class example {
    public static void main ( String args[] )
    {
        Random prng = new Random( 423 );
        // BEWARE: nextDouble() returns values in [0,1),
        // NOT (0,1)
        System.out.format( "%f %f %f\n",
                prng.nextDouble(),
                prng.nextDouble(), prng.nextDouble() );
        System.exit(0);
    }
}
```

Example: Python 3

```
#!/usr/bin/env python3
import random

# create pRNG object with seed as parameter
R = random.Random(423)

# BEWARE! .random() is a [0,1) value, NOT (0,1)
print( R.random(), R.random(), R.random() )
```