# Learning to Generate Random Numbers with SAS: A Practical Guide with Examples

Authored by

**Mohammed looti**

October 31, 2025

Generating random data is a foundational necessity in statistical computing, crucial for tasks ranging from simulation and modeling to creating test datasets and performing advanced sampling techniques. Within the SAS environment, the primary mechanism for generating these values is the **rand()** function.

This powerful function enables users to draw numbers from various statistical distributions, ensuring that data generation is robust and aligned with specific analytical requirements. The following comprehensive guide explores three practical applications, demonstrating how to use the **rand()** function effectively to generate single values, create variables with multiple observations, and populate several independent variables within a dataset.

## Understanding Pseudorandom Number Generation in SAS

Before implementing the code examples, it is essential to understand the nature of the values being generated. Computers cannot generate truly random numbers; instead, they rely on algorithms to produce sequences of numbers that exhibit properties statistically indistinguishable from genuine randomness. These are known as **pseudorandom** numbers.

The sequence of pseudorandom numbers is entirely determined by an initial value, referred to as the **seed**. If the same seed is used, the generator will produce the exact same sequence of numbers every time. This determinism is actually a feature, not a bug, especially in scientific research where reproducibility is paramount. Understanding how to manage this seed is key to controlling the random number generation process in SAS.

The **rand()** function itself is highly flexible. It requires specifying the distribution type (e.g., "integer," "normal," "uniform") followed by the parameters appropriate for that distribution (e.g., minimum and maximum for an integer, or mean and standard deviation for a normal distribution). Mastery of this function allows analysts to simulate complex data scenarios accurately.

## Controlling Randomness and Ensuring Reproducibility (The CALL STREAMINIT Statement)

For any statistical analysis involving random numbers to be valid and verifiable, the process must be reproducible. In SAS, this critical step is managed by the **call streaminit()** function, which is designed to set the seed for the random number generator.

The syntax is straightforward: `call streaminit(seed_value)`. The `seed_value` is an integer that initializes the sequence. If you omit the **call streaminit()** statement, SAS will typically use a default seed based on the system clock's current time, resulting in a different random sequence every time the code is executed. While this might be suitable for non-critical simulations, it guarantees non-reproducibility.

We strongly recommend always including **call streaminit()** when generating random data. This practice ensures that anyone running your code, including yourself at a later date, will obtain the identical set of random values, thus validating your simulation or sampling methodology. Changing the seed value is the only way to obtain a different, independent stream of pseudorandom numbers while maintaining control over the process.

## Example 1: Generating a Single Random Integer Using RAND

The simplest application of the random number function is generating a single data point. This example demonstrates how to create a dataset containing just one observation, where the variable `x` holds a random integer bounded between 1 and 10, inclusive.

The process is executed within a **DATA step**, which is fundamental to data manipulation in SAS. We utilize `call streaminit(1)` to set the seed to 1, guaranteeing that the result will be consistent across all executions. The **rand("integer", 1, 10)** syntax explicitly requests a random integer from the specified range.

Note the inclusion of the **output** statement immediately following the assignment of the random value. In a standard DATA step, observations are written to the dataset only when the step completes, but here, we must explicitly use **output** to force the creation of the single observation before the step naturally terminates.

```
/*create dataset with variable that contain random value*/
data my_data;
call streaminit(1); /*make this example reproducible*/
x = rand("integer", 1, 10);
output;
run;

/*view dataset*/
proc print data=my_data;
```

| Obs | x |
|---|---|
| 1 | 9 |

Due to the specific seed value of 1, the generated random number between 1 and 10 consistently turns out to be **9** in this execution. If the **call streaminit(1)** statement were removed, the resulting value would change each time the code is run, which is often undesirable for testing or documentation purposes. If variability is desired, simply omit the **call streaminit()** function entirely,

allowing SAS to select a time-based seed, or manually change the seed number before each execution.

## Example 2: Creating a Variable with Multiple Random Observations

More often than a single value, analysts need to generate an entire series of random data points for a variable. This is accomplished by leveraging the iterative power of the DATA step in conjunction with a **DO loop**. This next example demonstrates how to generate a variable, `x`, containing 10 distinct random integer values, where each value is between 1 and 20.

The **DO i = 1 to 10** loop dictates that the code block within the loop will execute ten times. Crucially, the **output** statement is nested inside the loop. Each time the loop iterates, a new random value is generated by `x = rand("integer", 1, 20)`, and the **output** statement writes that observation to the dataset `my_data` before the next iteration begins. This results in ten observations being created.

We use `call streaminit(10)` to initialize the random stream. It is important to remember that the number passed to `streaminit` (10 in this case) is just the starting seed; it is not related to the number of observations (also 10) generated by the DO loop.

```
/*create dataset with variable that contain random value*/
data my_data;
call streaminit(10);
do i = 1 to 10;
x = rand("integer", 1, 20);
output;
end;
run;

/*view dataset*/
proc print data=my_data;
```

| Obs | i | x |
|---|---|---|
| 1 | 1 | 20 |
| 2 | 2 | 4 |
| 3 | 3 | 7 |
| 4 | 4 | 16 |
| 5 | 5 | 19 |
| 6 | 6 | 20 |
| 7 | 7 | 4 |
| 8 | 8 | 8 |
| 9 | 9 | 19 |
| 10 | 10 | 19 |

Inspecting the resulting dataset confirms that the variable `x` contains ten values, and each one is a random integer within the specified inclusive range of 1 to 20. This structure is ideal for creating simulated input data where a variable needs to be populated with stochastic values based on a uniform discrete distribution.

## Example 3: Generating Multiple Independent Random Variables

In complex simulations, it is frequently necessary to generate multiple variables concurrently, each following potentially different distributions or ranges. The third example illustrates how to generate two independent variables, `x` and `y`, both within the same **DATA step** and **DO loop** structure.

We maintain the same setup as Example 2 (using a loop to generate 10 observations and setting the seed with `call streaminit(10)`). However, within the loop, we now define two separate calls to the **rand()** function. The first generates `x` as an integer between 1 and 20, while the second generates `y` as an integer between 50 and 100.

Crucially, because both calls to the rand function utilize the same initialized stream (set by `streaminit(10)`), they draw values sequentially from that single DATA step sequence. For example, the first number generated (for `x`) uses the first number in the sequence, and the second number generated (for `y`) uses the second number in the sequence, and so on for all 10 iterations. This ensures that the two generated variables, `x` and `y`, are drawing from the same underlying source, maintaining the overall reproducibility of the dataset.

**/*create dataset with variable that contain random value*/**
**data my_data;**
**call streaminit(10);**

```
do i = 1 to 10;
x = rand("integer", 1, 20);
y = rand("integer", 50, 100);
output;
end;
run;

/*view dataset*/
proc print data=my_data;
```

| Obs | i | x | y |
|---|---|---|---|
| 1 | 1 | 20 | 59 |
| 2 | 2 | 7 | 88 |
| 3 | 3 | 19 | 98 |
| 4 | 4 | 4 | 68 |
| 5 | 5 | 19 | 97 |
| 6 | 6 | 15 | 56 |
| 7 | 7 | 19 | 98 |
| 8 | 8 | 18 | 64 |
| 9 | 9 | 14 | 76 |
| 10 | 10 | 16 | 67 |

The printed output clearly shows that the **x** variable contains 10 random integers exclusively drawn from the range , while the **y** variable contains 10 random integers exclusively drawn from the range . This method is highly effective for generating multivariate simulated data where the variables have distinct scales or distributional requirements.

## Advanced Applications of the RAND Function and Different Distributions

While the examples above focus on the discrete "integer" distribution, the DATA step **rand()** function is capable of generating numbers from a wide array of continuous and discrete probability distributions, making it a versatile tool for complex simulations such as Monte Carlo analysis.

To use these other distributions, you simply replace the string parameter "integer" with the desired distribution name and provide the necessary parameters:

**Uniform:** `rand("uniform")` generates a value between 0 and 1.

**Normal:** `rand("normal")` generates a value from a standard normal distribution (mean 0, standard deviation 1).
**Exponential:** `rand("exponential", lambda)` generates a value from an exponential distribution, requiring the rate parameter lambda.
**Beta, Gamma, Poisson, and others:** A full range of distributions is supported, requiring their specific shape and scale parameters.

When generating continuous variables, such as those from the Normal or Uniform distributions, the output values will be floating-point numbers rather than integers. This flexibility allows SAS programmers to model almost any real-world random phenomenon accurately within their simulated datasets. Always consult the official SAS documentation for the precise parameters required for each distribution supported by the **rand()** function.

## Further Resources for SAS Programming

Generating random numbers is often just the initial step in a larger analytical workflow. To deepen your expertise in data management and statistical modeling within the SAS environment, consider exploring related tutorials that cover common programming tasks.

These resources will help you transition from simple data generation to complex analysis, ensuring you can efficiently manipulate, transform, and analyze the simulated or real-world data you generate using tools like the **rand()** function.