Random Number

Random numbers are useful for a variety of purposes, such as generating data encryption keys, simulating, and modelling complex phenomena and for selecting random samples from larger data sets. They have also been used aesthetically, for example in literature and music, and are of course ever popular for games and gambling.

When **one single random number** is concern, that random number should be **selected from a set of possible values**, where the **selection of every single value in the set is equally probable, i.e., a uniform distribution**. However, when considering a **sequence of random numbers**, the **selection of each random value must be statistically independent of the others**.

Random Number Generator (RNG)

With the advent of computers, programmers recognised the need for a means of introducing randomness into a computer program. However, surprising as it may seem, it is difficult to get a computer to do something by chance.

Random numbers take on a large role in the world of computing. In real life, a fundamental use of random numbers can be seen in **lotteries and lucky draws**, where computer programs are used to **generate random numbers to `decide the winning tickets**.

In the world of computing, random numbers can be used to **generate data encryption keys**, or to **select an independent random sample from the data set**.

A computer follows its instructions blindly and is therefore completely predictable. (A computer that does not follow its instructions in this manner is considered broken.)

There are **two** main approaches to generating random numbers using a computer:

1. **Pseudo-Random Number Generators (PRNGs)**,
2. **True Random Number Generators (TRNGs)**.

The approaches have distinct characteristics, and each has its pros and cons.

True Random Numbers

Random numbers obtained from certain **uncontrollable** physical phenomena.

The physical phenomenon can be very simple, like the little variations in somebody's mouse movements or in the amount of time between keystrokes.

A good physical phenomenon to use is a radioactive source. **The points in time at which a radioactive source decays are completely unpredictable, and they can quite easily be detected and fed into a computer**, avoiding any buffering mechanisms in the operating system.

For most real-life applications, a pseudo-random number is sufficient. For example, a CD player in “random” mode is really playing in pseudo-random mode, with a pattern that is discernible if you listen carefully enough. Using pseudo-random numbers is perfectly acceptable in this case because there’s no quantitative advantage in the degree of randomness.

Can a computer generate a truly random number?

Computers are **deterministic devices** (i.e. given and using the same input, it will follow a fixed set of instructions and therefore produce the same output). For true random numbers to be generated, the device generating the random number has to **produce totally unpredictable and unbiased output from the same input source**. **It is impossible to apply reverse engineering on truly random numbers**. There are devices that generate numbers that claim to be truly random. They rely on unpredictable processes like thermal or atmospheric noise rather than human-defined patterns.

You can program a machine to generate what can be called “random” numbers, but **the machine is always at the mercy of its programming**. **On a completely deterministic machine you cannot generate anything you could really call a random sequence of numbers, because the machine is following the same algorithm to generate them**. Typically, that means it starts with a common ‘seed’ number and then follows a pattern. The results may be sufficiently complex to make the pattern difficult to identify, but because it is

ruled by a carefully defined and consistently repeated algorithm, the numbers it produces are not truly random. They are what we call 'pseudo-random' numbers.

Pseudo-Random Number Generators (PRNGs)

Pseudo Random Number Generator (PRNG) refers to an **algorithm that uses mathematical formulas to produce sequences of random numbers**. PRNGs generate a sequence of numbers approximating the properties of random numbers.

A PRNG starts from an arbitrary **starting state** using a **seed state**. **With a starting point,** a sequence of pseudo-random numbers can be generated in a short time. Additionally, these numbers **can also be reproduced later**.

Hence, the random numbers generated by PRNGs are **deterministic** and **efficient**.

How a fundamental PRNG work?

The Linear Congruential Generator is most common and oldest algorithm for generating pseudo-randomised numbers. The generator is defined by the recurrence relation:

$$X_{n+1} = (aX_n + c) \% m$$

Given that:
X is a sequence of pseudo – random values,
$m > 0$, (m – modulus),
$0 < a < m$ (a - multiplier),
$0 \leq c < m$ (c - increment),
$0 \leq X_0 < m$ (X_0, seed or start value)

By choosing the following values:

$m = 9,\ a = 2,\ c = 0,\ X_0 = 1$

We have, $X_{n+1} = (2X_n + 0) \% 9$

Hence,
$X_0 = \mathbf{1}$
$X_1 = (2 \% 9) = \mathbf{2}$
$X_2 = (4 \% 9) = \mathbf{4}$
$X_3 = (8 \% 9) = \mathbf{8}$
$X_4 = (16 \% 9) = \mathbf{7}$
$X_5 = (14 \% 9) = \mathbf{5}$
$X_6 = (10 \% 9) = \mathbf{1}$
$X_7 = (2 \% 9) = \mathbf{2}$

Characteristics of a PRNG

- **Efficient**: PRNG can produce many numbers in a short time and is advantageous for applications that need many numbers.
- **Deterministic**: A given sequence of numbers can be reproduced later if the starting point in the sequence is known. Determinism is handy if you need to replay the same sequence of numbers again at a later stage.
- **Periodic**: PRNGs are periodic, which means that the sequence will eventually repeat itself. While periodicity is hardly ever a desirable characteristic, modern PRNGs have a period that is so long that it can be ignored for most practical purposes.

Important

- You are required to know how to generate random numbers in their chosen programming language.
- You are likely required to use random numbers in the problems they solve and so, they should also be able to include them in their pseudocode.

Using a random number generator in Pseudocode

It is important to note that any use of a function to generate random numbers **must be clearly explained**.

The following are two commonly used random number generator functions:

1. `RND()`: generates and returns a single random real number between 0 and 1.
2. `RANDOMBETWEEN(min, max)`: generates a random integer between the integers min and max.

`RND()` function

Returns a single random number.

The `RND()` function returns a float value less than 1 but greater than or equal to 0. (i.e. `[0.0, 1.0)`) The value of number determines how `RND()`generates a random number:

To produce random integers **in a given range**, the following expression can be used:

```
   INT((upperbound - lowerbound + 1) * RND() + lowerbound)

// upperbound is the highest number in the range, and
// lowerbound is the lowest number in the range
```

Pseudocode – To generate a random integer value from 1 to 6 (both incl)

```
FUNCTION RANDOMBETWEEN (min: INTEGER, max: INTEGER) RETURNS INTEGER:
     DECLARE rand_no: INTEGER
     rand_no = INT((max - min + 1) * RND() + min)
     RETURN rand_no
ENDFUNCTION
// -----------Main program------------
DECLARE random_no: INTEGER
random_no = RANDOMBETWEEN(1, 6)
```

Generating a Random Number in Python

randint(a, b)	Returns a random integer `N` such that `a <= N <= b. i.e. [a,b]`
random()	Returns a random floating point number in the range `[0.0 to 1.0).`

randrange(start, stop, step)	Return a randomly selected element from `randrange(start, stop, step)`. **Expected output:** Expected output of `randrange(1, 10, 3)` //incl 1, not incl 10 Integer values ∈ `[1, 4, 7]`

```
>>> import random
>>> a = random.random()
>>> b = random.randint(1998, 2019)
>>> c = random.randrange(1, 10, 2)
>>> a
0.9601103464930293
>>> b
2007
>>> c
7
>>>
```

True-Random Number Generators (TRNGs)

In comparison with PRNGs, TRNGs **extract randomness from physical phenomena** and introduce it into a computer. There are many other ways to get true randomness into your computer. A good physical phenomenon is to use is a **radioactive source**, as radioactive source decays are completely unpredictable, and they are easily be detected to have them fed into a computer. Another suitable physical phenomenon is **atmospheric noise**, which can be picked up easily by a normal radio. For instance, the background noise picked up from an office or laboratory can be used to extract randomness, but **consider factors like operating hours, busy hours, background noise** etc., need to be taken into consideration as they **reduce the level of randomness**.

Regardless of which physical phenomenon is used, the process of generating true random numbers involves identifying little, unpredictable changes in the data.

Pros & Cons of TRNGs

The characteristics of TRNGs are quite different from PRNGs. First, TRNGs are generally more inefficient as compared to PRNGs, taking considerably longer time to produce numbers. They are also nondeterministic, meaning that a given sequence of numbers cannot be reproduced, although the same sequence may of course occur several times by chance. TRNGs should not be periodic.

Comparison between PRNG and TRNG

Characteristic	Pseudo-Random Number Generator	True Random Number Generators
Efficiency	Excellent	Poor
Determinism	Deterministic	Nondeterministic
Periodicity	Periodic	Aperiodic

Application of PRNG & TRNG

Application	Most Suitable Generator
Lotteries and Draws	TRNG
Games and Gambling	TRNG
Random Sampling (e.g., drug screening)	TRNG
Simulation and Modelling	PRNG
Security (e.g., generation of data encryption keys)	TRNG

Programming Exercises

Exercise 1

(i) Randomize and print the outcome of a 6 – sides unbiased die and roll this die for n 100 independent trials. Print the outcomes of each trial.

(ii) Obtain the number of trials as an integer input from user and derive the average result where:

```
average = (sum of the outcomes taken from the n trials) ÷ n
```

(iii) The outcome of rolling an unbiased die is tabulated in a frequency table. Display the frequency table of all the possible outcomes taken from the n trials.

Exercise 2

Using Python's built-in function `random()`, create:

(i) `myRandomInt(a,b)` that returns a randomised integer value that is in the range of `{a,…,b}`, where a and b are positive integers and `b > a`.

(ii) myRandomIntHalfOpen(a, b), that returns a randomised integer value that is in the range of `{a,…,b-1}`, where a and b are positive integers and `b > a`.