

Python random Module – Generate Random Numbers/Sequences

By Vijaykrishna Ram / December 28, 2019

This article is about the `random` module in Python, which is used to generate pseudo-random numbers for various probabilistic distributions.

Python random Module Methods

1. `seed()`

This initializes a random number generator. To generate a new random sequence, a seed must be set depending on the current system time.

`random.seed()` sets the seed for random number generation.

2. `getstate()`

This returns an object containing the current state of the generator. To restore the state, pass the object to `setstate()`.

3. `setstate(state_obj)`

This restores the state of the generator at the point when `getstate()` was called, by passing the state object.

4. `getrandbits(k)`

This returns a Python integer with `k` random bits. This is useful for methods like `randrange()` to handle arbitrary large ranges for random number generation.

```
>>> import random
>>> random.getrandbits(100) # Get a random integer having 100 bits
802952130840845478288641107953
```

Here is an example to illustrate `getstate()` and `setstate()` methods.

```
import random

random.seed(1)

# Get the state of the generator
state = random.getstate()

print('Generating a random sequence of 3 integers...')
for i in range(3):
    print(random.randint(1, 1000))

# Restore the state to a point before the sequence was generated
random.setstate(state)
print('Generating the same identical sequence of 3 integers...')
for i in range(3):
    print(random.randint(1, 1000))
```

Possible Output:

```
Generating a random sequence of 3 integers...
138
583
868
Generating the same identical sequence of 3 integers...
138
583
868
```

Generate Random Integers

The random module provides some special methods for generating random integers.

1. randrange(start, stop, step)

Returns a randomly selected integer from `range(start, stop, step)`. This raises a `ValueError` if `start > stop`.

2. randint(a, b)

Returns a random integer between `a` and `b` (both inclusive). This also raises a `ValueError` if `a > b`.

Here is an example that illustrates both the above functions.

```
import random

i = 100
j = 20e7

# Generates a random number between i and j
a = random.randrange(i, j)
try:
    b = random.randrange(j, i)
except ValueError:
    print('ValueError on randrange() since start > stop')

c = random.randint(100, 200)
try:
    d = random.randint(200, 100)
except ValueError:
    print('ValueError on randint() since 200 > 100')

print('i =', i, ' and j =', j)
print('randrange() generated number:', a)
print('randint() generated number:', c)
```

Possible Output

```
ValueError on randrange() since start > stop
ValueError on randint() since 200 > 100
i = 100 and j = 200000000.0
randrange() generated number: 143577043
randint() generated number: 170
```

Generating Random floating point numbers

Similar to generating integers, there are functions that generate random floating point sequences.

- `random.random()` -> Returns the next random floating point number between [0.0 to 1.0)
- `random.uniform(a, b)` -> Returns a random floating point N such that $a \leq N \leq b$ if $a \leq b$ and $b \leq N \leq a$ if $b < a$.
- `random.expovariate(lambda)` -> Returns a number corresponding to an exponential distribution.
- `random.gauss(mu, sigma)` -> Returns a number corresponding to a gaussian distribution.

There are similar functions for other distributions, such as Normal Distribution, Gamma Distribution, etc.

An example of generating these floating-point numbers is given below:

```
import random

print('Random number from 0 to 1 :', random.random())
print('Uniform Distribution between [1,5] :', random.uniform(1, 5))
print('Gaussian Distribution with mean = 0 and standard deviation = 1 :')
print('Exponential Distribution with lambda = 0.1 :', random.expovariate(0.1))
print('Normal Distribution with mean = 1 and standard deviation = 2:', random.gauss(1, 2))
```

Possible Output

```
Random number from 0 to 1 : 0.44663645835100585
Uniform Distribution between [1,5] : 3.65657099941547
Gaussian Distribution with mean = 0 and standard deviation = 1 : -2.271
Exponential Distribution with lambda = 0.1 : 12.64275539117617
Normal Distribution with mean = 1 and standard deviation = 2 : 4.259037
```

Random Sequences using the random module

Similar to integers and floating-point sequences, a generic sequence can be a collection of items, like a List / Tuple. The `random` module provides useful functions which can introduce a state of randomness to sequences.

1. `random.shuffle(x)`

This is used to shuffle the sequence in place. A sequence can be any list/tuple containing elements.

Example Code to illustrate shuffling:

```
import random

sequence = [random.randint(0, i) for i in range(10)]

print('Before shuffling', sequence)

random.shuffle(sequence)

print('After shuffling', sequence)
```

Possible Output:

```
Before shuffling [0, 0, 2, 0, 4, 5, 5, 0, 1, 9]
After shuffling [5, 0, 9, 1, 5, 0, 4, 2, 0, 0]
```

2. `random.choice(seq)`

This is a widely used function in practice, wherein you would want to randomly pick up an item from a List/sequence.

```
import random

a = ['one', 'eleven', 'twelve', 'five', 'six', 'ten']

print(a)

for i in range(5):
    print(random.choice(a))
```

Possible Output

```
['one', 'eleven', 'twelve', 'five', 'six', 'ten']
ten
eleven
six
twelve
twelve
```

3. random.sample(population, k)

Returns a random sample from a sequence of length `k`.

```
import random

a = ['one', 'eleven', 'twelve', 'five', 'six', 'ten']

print(a)

for i in range(3):
    b = random.sample(a, 2)
    print('random sample:', b)
```

Possible Output

```
['one', 'eleven', 'twelve', 'five', 'six', 'ten']
```

```
random sample: ['five', 'twelve']  
random sample: ['ten', 'six']  
random sample: ['eleven', 'one']
```

Random Seed

Since pseudorandom generation is based on the previous number, we usually use the system time to make sure that the program gives a new output every time we run it. We thus make use of **seeds**.

Python provides us with `random.seed()` with which we can set a seed to get an initial value. This seed value determines the output of a random number generator, so if it remains the same, the output also remains the same.

```
import random  
  
random.seed(1)  
  
print('Generating a random sequence of 4 numbers...')  
print([random.randint(1, 100) for i in range(5)])  
  
# Reset the seed to 1 again  
random.seed(1)  
  
# We now get the same sequence  
print([random.randint(1, 100) for i in range(5)])
```

Possible Output

```
Generating a random sequence of 4 numbers...  
[18, 73, 98, 9, 33]  
[18, 73, 98, 9, 33]
```

This ensures that we need to be mindful of our seed when dealing with pseudorandom sequences, since the sequence may repeat if the seed is unchanged.

Conclusion

We learned about various methods that Python's random module provides us with, for dealing with Integers, floating-point numbers, and other sequences like Lists, etc. We also saw how the **seed** influences the sequence of the pseudorandom numbers.

References

- [Python random module Documentation](#)
- [JournalDev article on random numbers](#)

[← Previous Post](#)[Next Post →](#)

Recent Posts

[Connected Health: Exploring IoT Solutions for Healthcare Transformation](#)[Maximizing Cost Savings Through Offshore Development: A Comprehensive Guide](#)[Running a Flask Application with Python3: A Step-by-Step Guide](#)[Levenshtein Distance in Python: Troubleshooting Installation Errors on Windows](#)[10 PyJanitor's Miscellaneous Functions for Enhancing Data Cleaning](#)[Random Search in Machine Learning: Hyperparameter Tuning Technique](#)[Flake8: Python's Powerful Code Analysis Tool for Improved Code Quality](#)[Delivery Route Optimization using Python: A Step-by-Step Guide](#)[Understanding Capital Asset Pricing Model \(CAPM\)](#)[Splitting Lists into Sub-Lists in Python: Techniques and Advantages](#)

Favorite Sites

[GoLang Tutorials](#)

[VM-Help](#)

[Linux Tutorials](#)

[MySQL Tutorials](#)

[CodeForGeek](#)

[Mkyong](#)

Copyright © 2024 AskPython · All Rights Reserved
Privacy Policy · Terms and Conditions · Contact · About · Team
AskPython is part of Diyansh IT Services Private Limited