Homework 5

also posted on Brightspace

due next Wed (Oct 12) in class

Homework 5 will be accepted without any late penalty so long as it is turned in before Thu Oct 13 @11:59pm

`Homework5.pdf` (written description)

`Homework5.ipynb`

`BoysBW.jpg`

Homework 6

Homework 6 will be posted this week,
and it will be due in **two weeks**, on Wed Oct 26
(of course, you do not need to work on it over fall break)

it will be a little bit longer since you have a few more regular class days to work on it

Homework 6 (Q1)

Homework 6 will need to be done using a .py file and created/edited/debugged using PyCharm or another IDE

Q1 asks for screen shots as indicated

(a) while editing the code

(b) while running the full code

(c) while running a cell

(d) while debugging (break points and stepping)

more on PyCharm

debugging Jupyter Notebooks in PyCharm

"Step Out"

use "Step Out" if you accidentally "Step In"
to a function you did not write

"Run to Cursor"

lines with multiple evaluations

download from Brightspace

`Random.ipynb`

Homework 6 (Q2)

- **Q2** asks you to take a set of data like this:

```
data = np.array([68, 47, 63, 76, 44, 64, 81,
     66, 106, 68, 72, 72, 46, 75, 49, 84, 88])
```

and produce a "stem and leaf plot"

you should Google "stem and leaf plot"

```
 4 | 4 6 7 9
 5 |
 6 | 3 4 6 8 8
 7 | 2 2 5 6
 8 | 1 4 8
 9 |
10 | 6
```

an old "low-tech" form of data visualization

Homework 6 (Q2)

- **Q2** asks you to take a set of data like this:

```
data = np.array([68, 47, 63, 76, 44, 64, 81,
      66, 106, 68, 72, 72, 46, 75, 49, 84, 88])
```

and produce a "stem and leaf plot"

you should Google "stem and leaf plot"

```
 4 | 4 6 7 9
 5 |
 6 | 3 4 6 8 8
 7 | 2 2 5 6
 8 | 1 4 8
 9 |
10 | 6
```

an old "low-tech" form of data visualization

Homework 6 (Q2)

- **Q2** asks you to take a set of data like this:

```
data = np.array([68, 47, 63, 76, 44, 64, 81,
      66, 106, 68, 72, 72, 46, 75, 49, 84, 88])
```

and produce a "stem and leaf plot"

you should Google "stem and leaf plot"

```
 4 | 4 6 7 9
 5 |
 6 | 3 4 6 8 8
 7 | 2 2 5 6
 8 | 1 4 8
 9 |
10 | 6
```

an old "low-tech" form of data visualization

Homework 6 (Q2)

- **Q2** asks you to take a set of data like this:

```
data = np.array([68, 47, 63, 76, 44, 64, 81,
      66, 106, 68, 72, 72, 46, 75, 49, 84, 88])
```

and produce a "stem and leaf plot"

you should Google "stem and leaf plot"

```
 4 | 4 6 7 9
 5 |
 6 | 3 4 6 8 8
 7 | 2 2 5 6
 8 | 1 4 8
 9 |
10 | 6
```

an old "low-tech" form of data visualization

Homework 6 (Q2)

- **Q2** asks you to take a set of data like this:

```
data = np.array([68, 47, 63, 76, 44, 64, 81,
     66, 106, 68, 72, 72, 46, 75, 49, 84, 88])
```

and produce a "stem and leaf plot"

you should Google "stem and leaf plot"

```
 4 | 4 6 7 9
 5 |
 6 | 3 4 6 8 8
 7 | 2 2 5 6
 8 | 1 4 8
 9 |
10 | 6
```

an old "low-tech" form of data visualization

Homework 6 (Q2)

- **Q2** asks you to take a set of data like this:

```
data = np.array([68, 47, 63, 76, 44, 64, 81,
        66, 106, 68, 72, 72, 46, 75, 49, 84, 88])
```

and produce a "stem and leaf plot"

you should Google "stem and leaf plot"

```
 4 | 4 6 7 9
 5 |
 6 | 3 4 6 8 8
 7 | 2 2 5 6
 8 | 1 4 8
 9 |
10 | 6
```

note that the "leaves" are ordered (smallest to largest)

an old "low-tech" form of data visualization

Homework 6 (Q2)

- **Q2** asks you to take a set of data like this:

```
data = np.array([68, 47, 63, 76, 44, 64, 81,
      66, 106, 68, 72, 72, 46, 75, 49, 84, 88])
```

and produce a "stem and leaf plot"

you should Google "stem and leaf plot"

```
 4 | 4 6 7 9
 5 |
 6 | 3 4 6 8 8
 7 | 2 2 5 6
 8 | 1 4 8
 9 |
10 | 6
```

(1) function that is passed the data and creates a data structure that holds the stem and leaf plot and returns it

(2) function that is passed the data structure holding the stem and leaf plot and prints it out

an old "low-tech" form of data visualization

Homework 6 Hints

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                 49, 84, 88])
```

```
stems      leaves
   4  |  4 6 7 9
   5  |
   6  |  3 4 6 8 8
   7  |  2 2 5 6
   8  |  1 4 8
   9  |
  10  |  6
```

let's start with what you might want to end up with
(useful way to solve a programming problem)

an "algorithm" is a sequence of steps to solve a problem
here, you generate the steps by hand, then instantiate in code

https://en.wikipedia.org/wiki/Algorithm

Homework 6 Hints

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                 49, 84, 88])
```

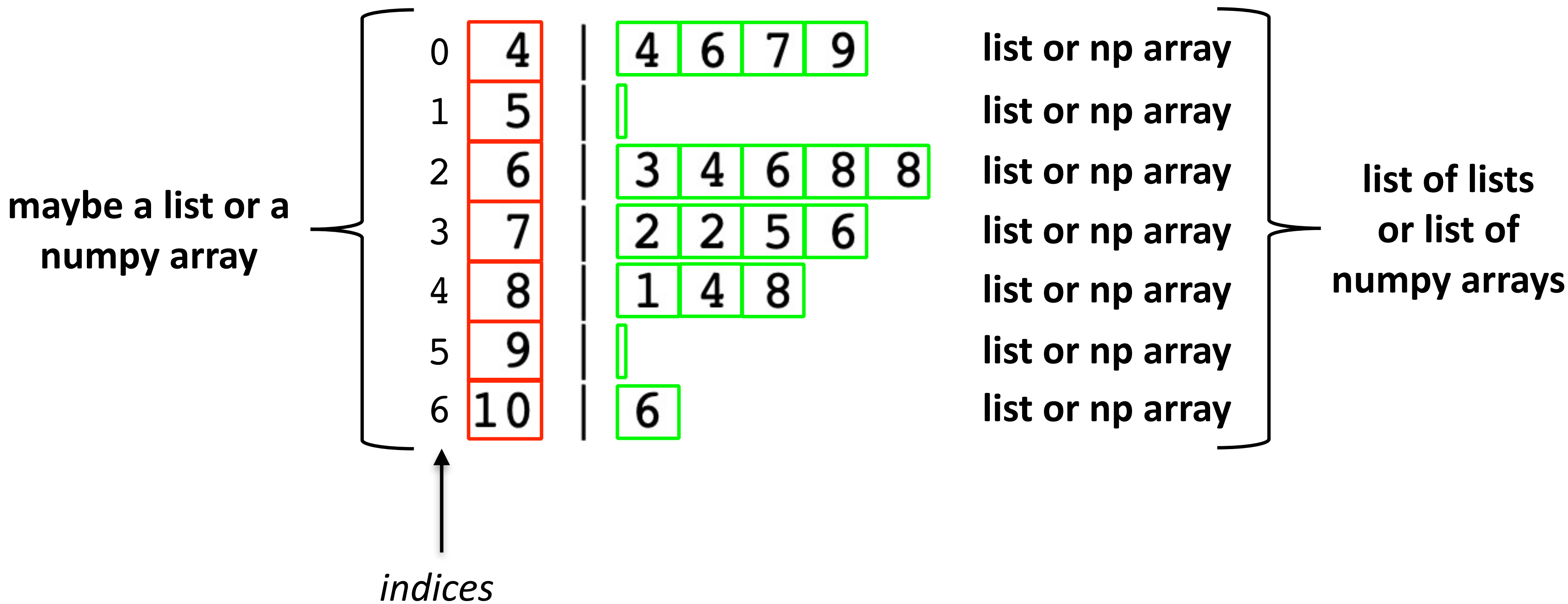
maybe a list or a numpy array

0	4	4 6 7 9
1	5	
2	6	3 4 6 8 8
3	7	2 2 5 6
4	8	1 4 8
5	9	
6	10	6

indices

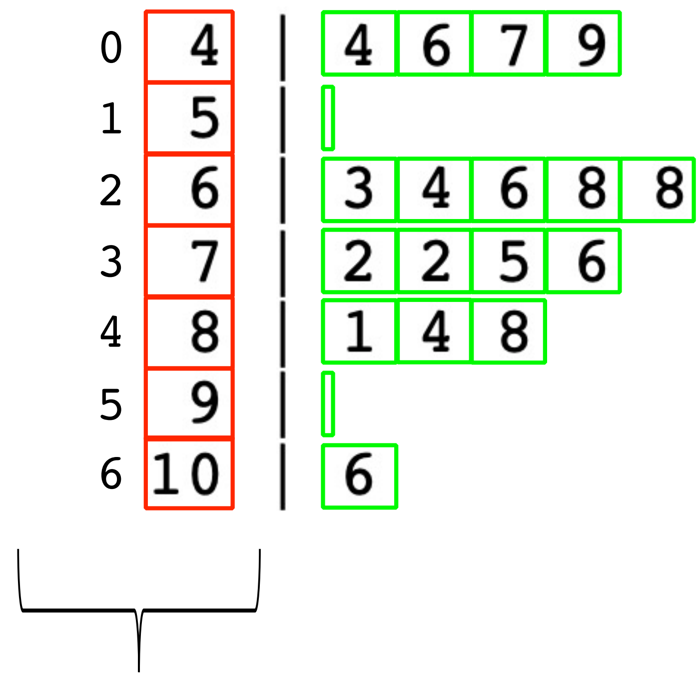
Homework 6 Hints

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                49, 84, 88])
```



Homework 6 Hints

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                 49, 84, 88])
```

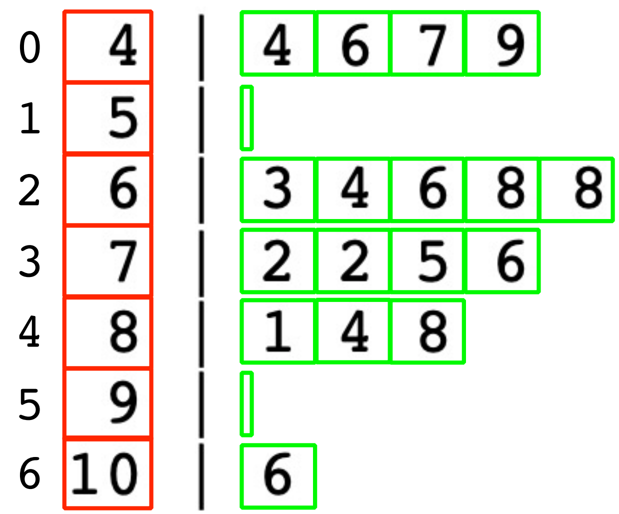


[4, 5, 6, 7, 8, 9, 10]

stems

Homework 6 Hints

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                 49, 84, 88])
```



[4, 5, 6, 7, 8, 9, 10]

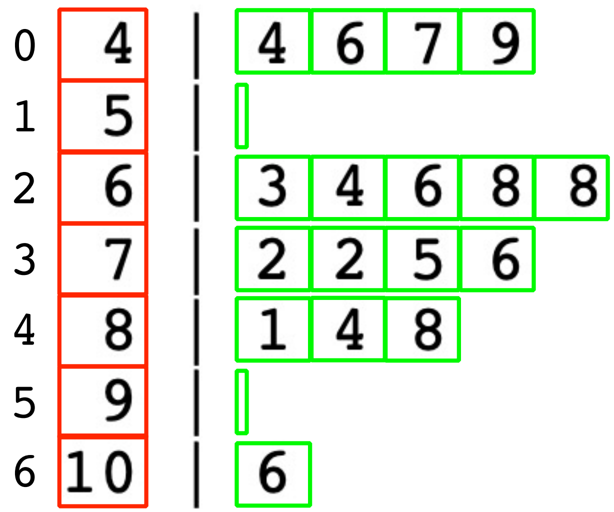
stems

[[4, 6, 7, 9], [], [3, 4, 6, 8, 8], [2, 2, 5, 6], [1, 4, 8], [], [6]]

leaves

Homework 6 Hints

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                 49, 84, 88])
```



[4, 5, 6, 7, 8, 9, 10]

stems

[[4, 6, 7, 9], [], [3, 4, 6, 8, 8], [2, 2, 5, 6], [1, 4, 8], [], [6]]

leaves

maybe a list

Homework 6 Hints

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                 49, 84, 88])
```

0	4	|	4 6 7 9
1	5	|	
2	6	|	3 4 6 8 8
3	7	|	2 2 5 6
4	8	|	1 4 8
5	9	|	
6	10	|	6

[4, 5, 6, 7, 8, 9, 10]

stems

[[4, 6, 7, 9], [], [3, 4, 6, 8, 8], [2, 2, 5, 6], [1, 4, 8], [], [6]]

leaves

maybe a list

obviously, this doesn't tell you how to fill the stem and leaves

Homework 6 Hints

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                 49, 84, 88])
```

stems	leaves
4	4 6 7 9
5	
6	3 4 6 8 8
7	2 2 5 6
8	1 4 8
9	
10	6

note that the leaves are in order, so we will want to fill the leaves list of lists in order

so, sort the data from smallest to largest

Homework 6 Hints

–23.678758, –12.45, –3.4, 4.43, 5.5, 5.678, 16.87, 24.7, 56.8

Stem	Leaf	
-2	4	-30 - -21
-1	2	-20 - -11
-0	3	-10 - -1
0	4 6 6	0 - 9
1	7	10 - 19
2	5	20 - 29
3		30 - 39
4		40 - 49
5	7	50 - 59

Key: $-2 \mid 4 = -24$

make sure you notice what happens when there are negative numbers and decimals in the data (see Wikipedia page https://en.wikipedia.org/wiki/Stem-and-leaf_display)

Random Numbers

Hellekalek, P. (1998). Good random number generators are (not so) easy to find. *Mathematics and Computers in Simulation*, 46, 485-505.

different ways of generating random numbers in Python

1) using the `random` module in base Python - NOT RECOMMENDED

2) `random` module in numpy, using **seed/state-based approach**
   - what has been used for many years
   - what you find across the web
   - what most existing Python code uses
   - not "pythonic" in style
   - does not work with parallel code
   - frozen, considered legacy
   - but it is what I will start with

3) `random` module in numpy, using **generator object approach**
   - what we will talk about later
   - recommended for code going forward

`numpy`, `scipy` use generator objects, `scikit-learn` uses seed/state approach, `keras/tensorflow` use their own random number generators see `Random.ipynb`

random numbers in Python

one random number generator (building block) in Python is:

Mersenne twister

developed in 1997

period of $2^{19937}-1$

has good statistical properties

reasonably fast

used in Python, Matlab

http://en.wikipedia.org/wiki/Mersenne_twister

but even this is not appropriate for cryptographic applications
despite its statistical properties, it's possible to "crack" it

random numbers in Python (numpy)

```
import numpy.random as R

# using Matlab style prng

# sample from uniform distribution
a = R.rand()
b = R.rand(3,4)

# sample from normal (mean 0, stdev 1)
c = R.randn()
d = R.randn(6,2)
```

random numbers in Python (numpy)

```
import numpy.random as R

# using Python/numpy style prng

# sample from uniform distribution
a = R.uniform()
```

tuple

```
b = R.uniform(size=(3,4))

# sample from normal (mean 0, stdev 1)
c = R.normal()
```

tuple

```
d = R.normal(size=(6,2))
```

seeding

needs to be an integer

```
R.seed(1243134)
print(R.uniform(), R.uniform(), R.uniform())
```

```
R.seed(1243134)
print(R.uniform(), R.uniform(), R.uniform())
```

produce the same random numbers as above

```
R.seed(8237234)
print(R.uniform(), R.uniform(), R.uniform())
```

produce different random numbers

Python vs. Matlab (at start)

Python

```
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.5793889964866393
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.28131924752039117
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.4491634596890828
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.8091308303813897
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.9150216172826635
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.4101784869762677
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.5160396404747076
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.290813502963848
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.3187345040516294
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.3072158336233559
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.12939408577526568
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.37874319118786026
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.8573560924779818
(base) Tom-MacBook-Pro-2020:ForClass palmerit$
```

Python starts with a different seed each time it is run (seeds with system time)

Python vs. Matlab (at start)

Matlab

```
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$
```

Matlab starts with the same seed each time it is run

seeding

```
seed = 2754332
R.seed(seed)
a = R.uniform(size=(5, 4))
```

Best Practices

- for every application in psychology and neuroscience (and scientific computing in general) you should **ALWAYS** seed and **SAVE** the seed (reproduce your results)
- only seed **ONCE**; seeding multiple times can actually disrupt the statistical properties of a PRNG

what could the seed be?

- some function of the subject and session number (ensure that every subject/session has a unique random order)

```
seed = 98433*subject + 413*session + 279
R.seed(seed)
```

- a fixed number (if doing a long simulation)

```
seed = 5433234
R.seed(seed)
```

- date/time but save the date/time (in a file) and label as seed

```
import time
seed = time.time_ns()%(2**32 - 1)
R.seed(seed)
```

returns nanoseconds since January 1970

this is the max seed value

returns remainder

what does it mean to generate a random number

- it makes no sense to say you are "generating a random number" without specifying <u>what probability distribution</u> that random number is <u>randomly sampled from</u>

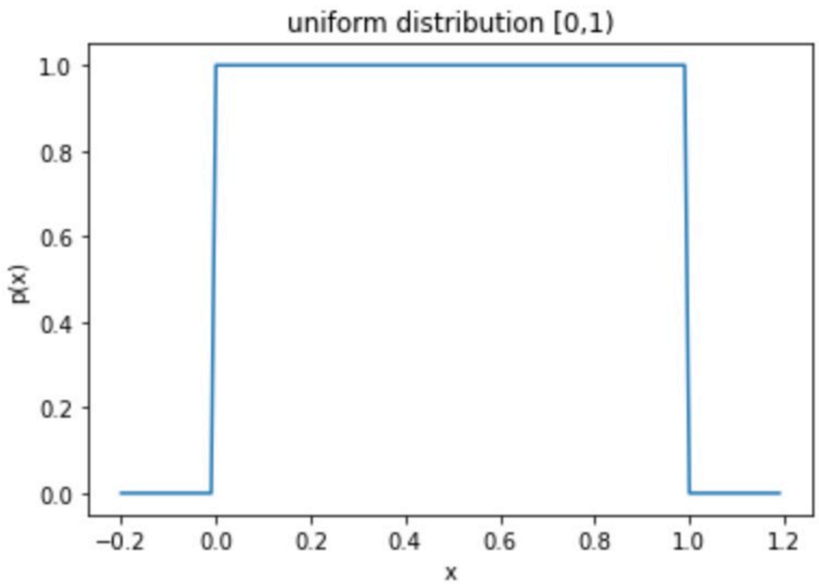
  numpy `R.uniform()` draws a random number from a continuous uniform distribution on `[0,1)`

  all (representable) real numbers on the interval `[0,1)` are <u>equally likely</u>

what does it mean to generate a random number

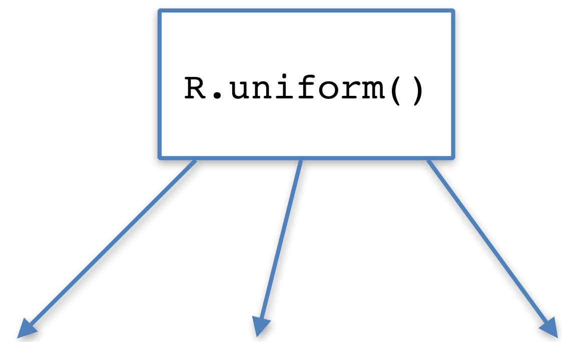
standard uniform distribution

probability density function



standard uniform PRNG

random number generator



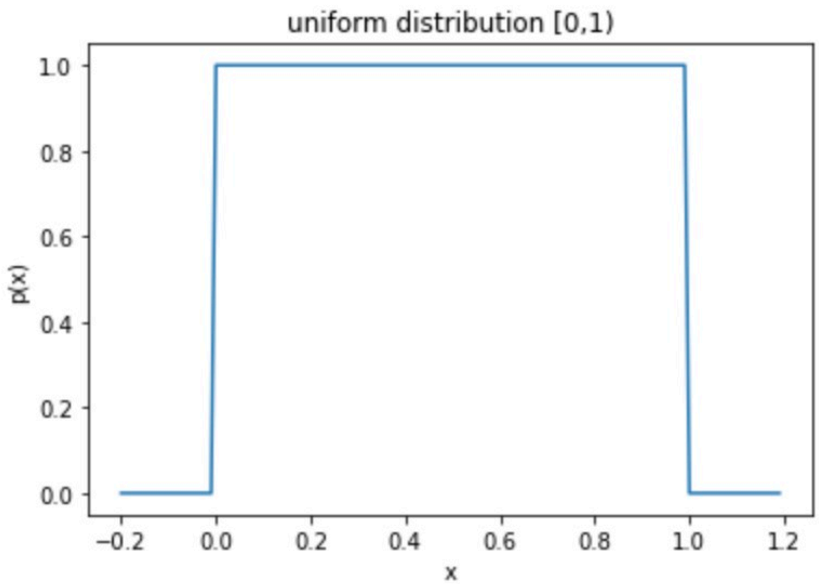
.42345234 .51832049 .10999311

how would you show that these are distributed as a uniform distribution?

what does it mean to generate a random number

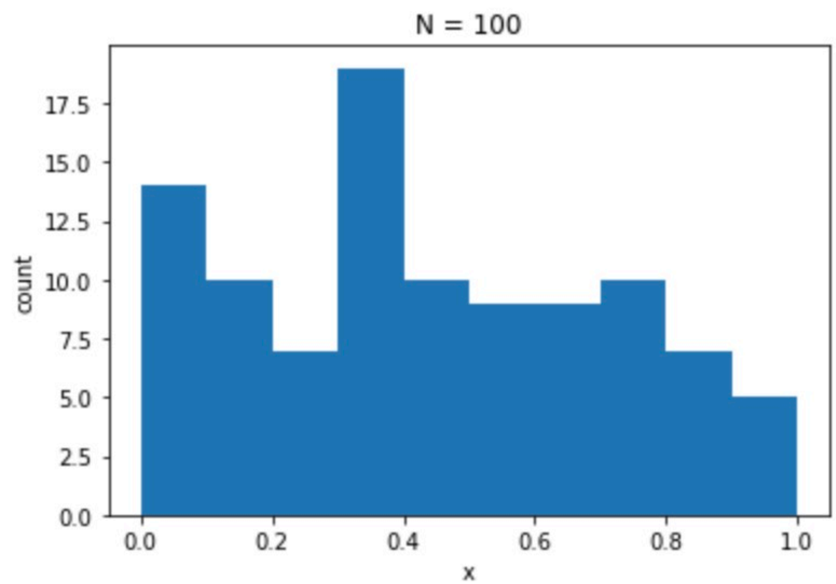
standard uniform distribution

probability density function



standard uniform PRNG

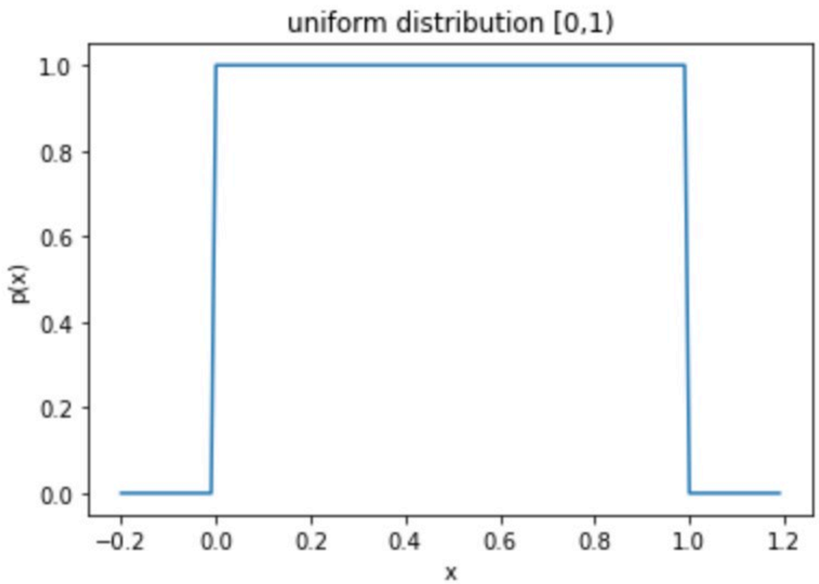
random number generator



what does it mean to generate a random number

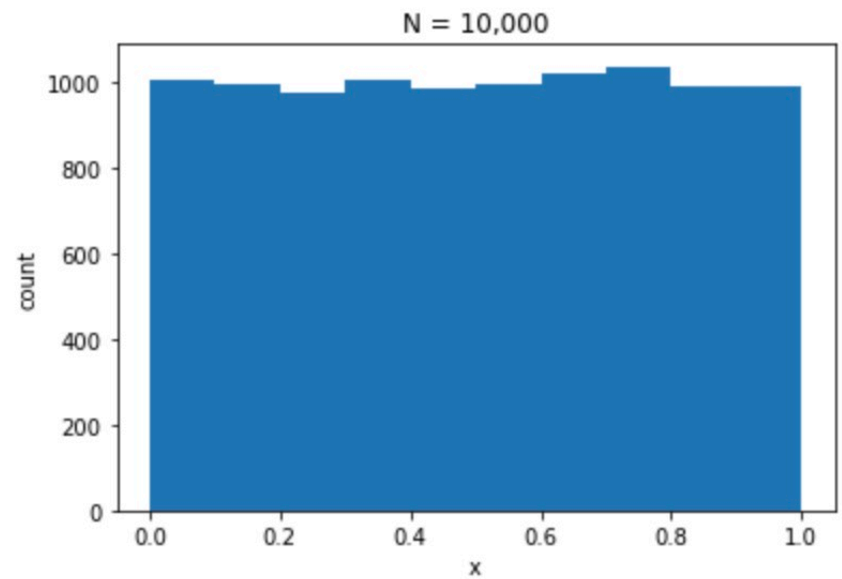
standard uniform distribution

probability density function



standard uniform PRNG

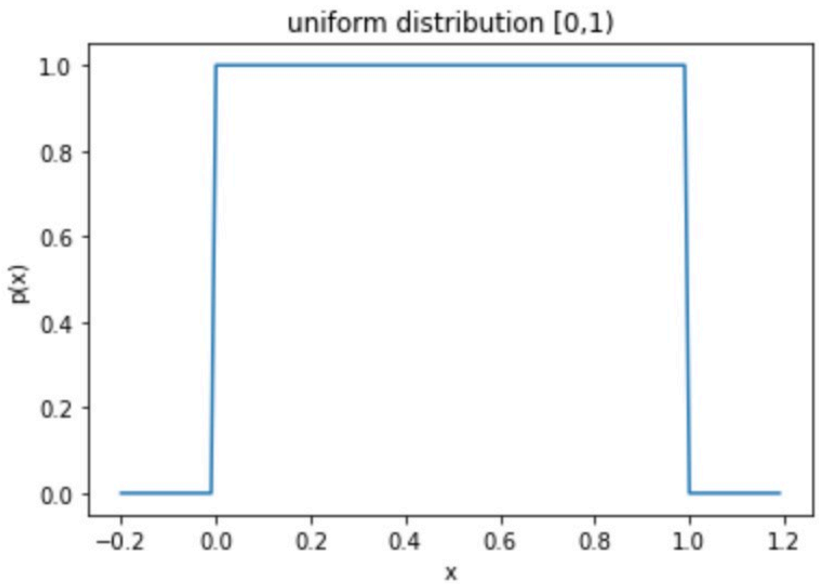
random number generator



what does it mean to generate a random number

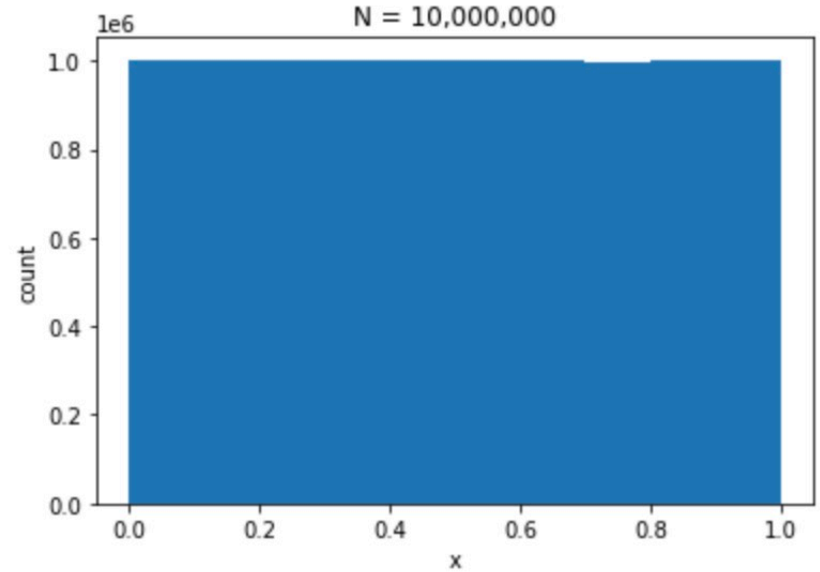
standard uniform distribution

probability density function



standard uniform PRNG

random number generator



(histograms in Python)

```
N = 10000000
rnd = R.uniform(size=(N,))


(h, b) = np.histogram(rnd, bins=10,
                      range=(0,1))


plt.bar(b[:len(b)-1], h, width=.10,
        align='edge')
plt.xlabel('x')
plt.ylabel('count')
plt.title(f'rand()\nN = {N:,}')
plt.show()
```

building block for all other random numbers

`R.uniform()` is the building block for **every other** random number generator

"every other" in what sense?

there are mathematical and computational techniques for using `R.uniform()` to generate random samples from any probability density function / probability mass function

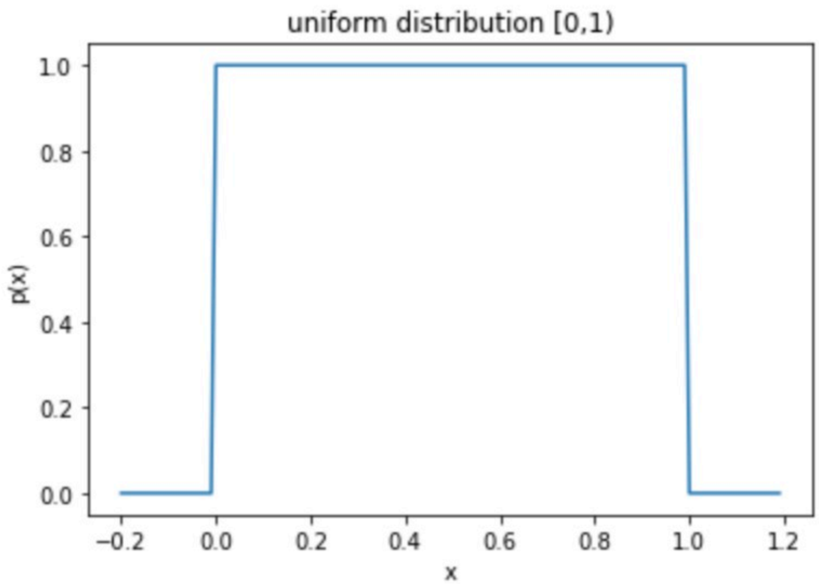
e.g.,

- transformation methods
- rejection sampling
- Markov Chain Monte Carlo (MCMC)

building block for all other random numbers

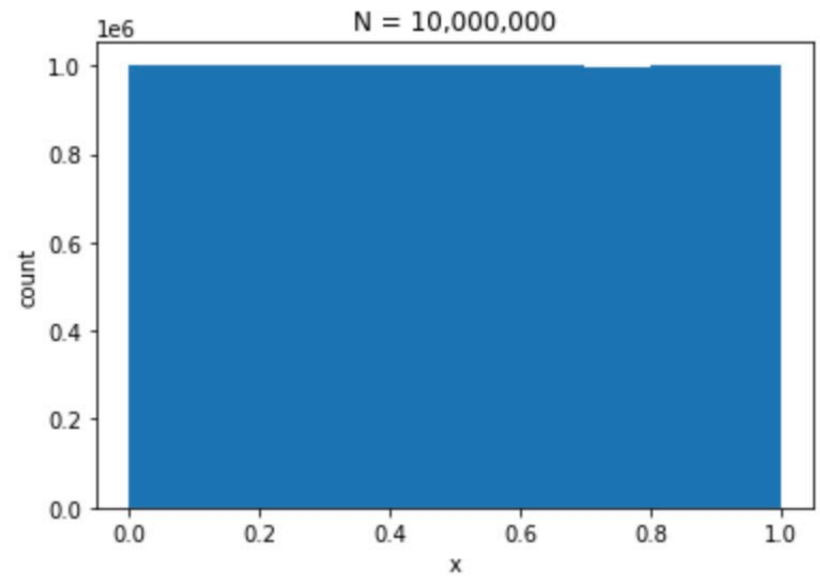
standard uniform distribution

probability density function



standard uniform PRNG

random number generator



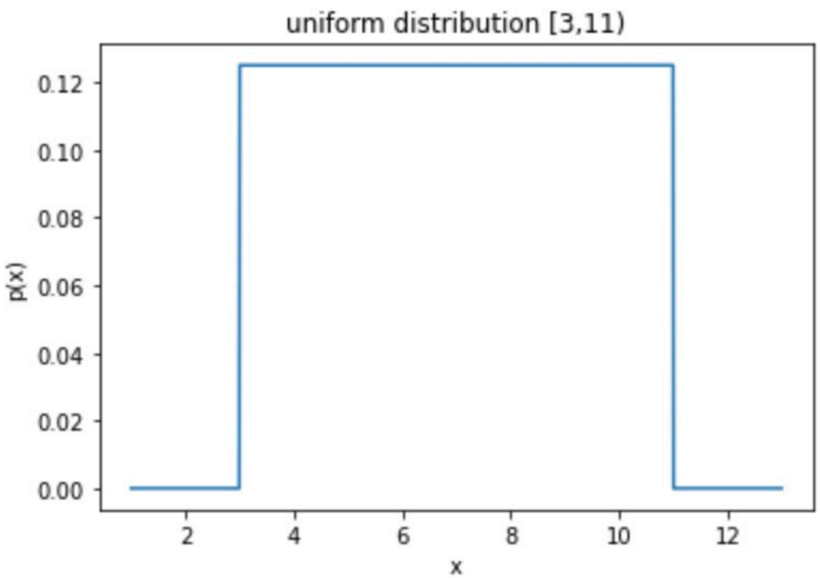
`R.uniform()` (standard uniform PRNG) is used to generate random numbers from any other distribution

continuous or discrete
(normal, Poisson, beta, log-normal, etc.)

general uniform distribution

uniform distribution [a,b)

probability density function



uniform PRNG [a,b)

random number generator

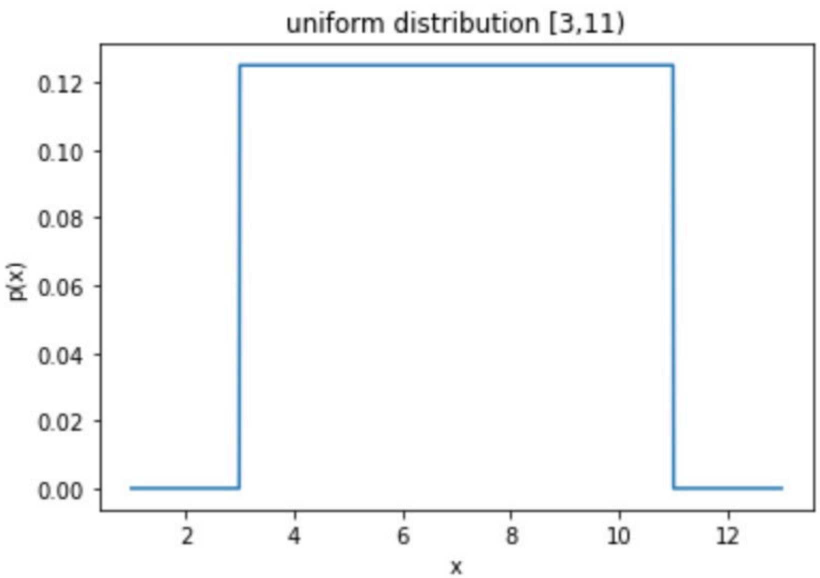
```
(b-a)*R.uniform()
```

continuous random number [0,(b-a))

general uniform distribution

uniform distribution [a,b)

probability density function



uniform PRNG [a,b)

random number generator

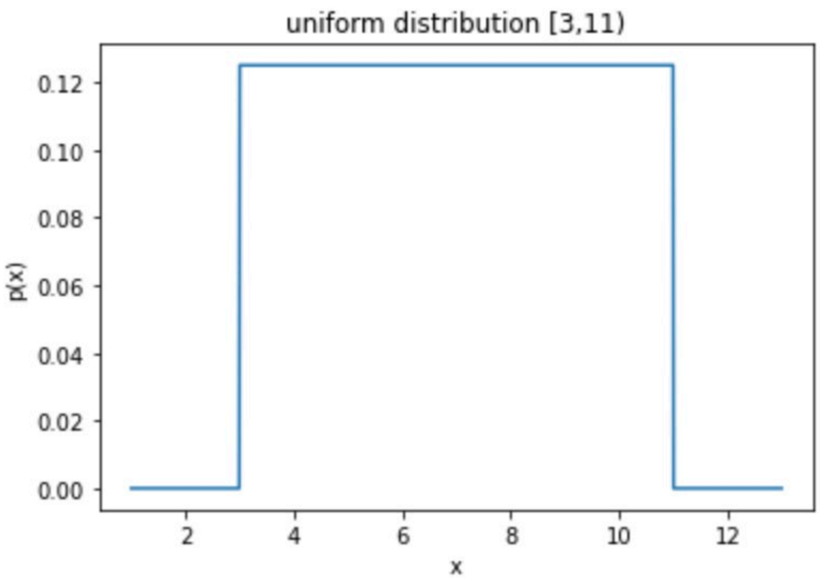
```
a + (b-a)*R.uniform()
```

continuous random number [a,b)

general uniform distribution

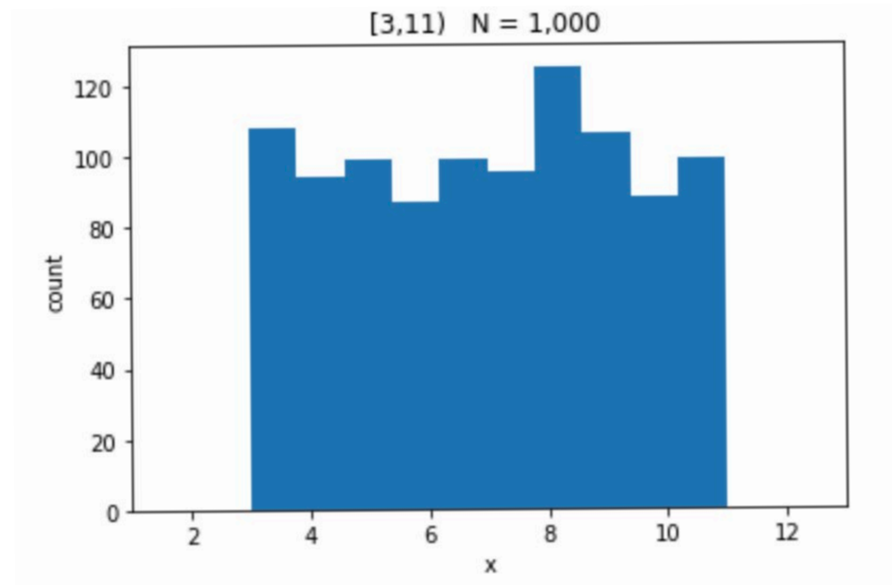
uniform distribution [a,b)

probability density function



uniform PRNG [a,b)

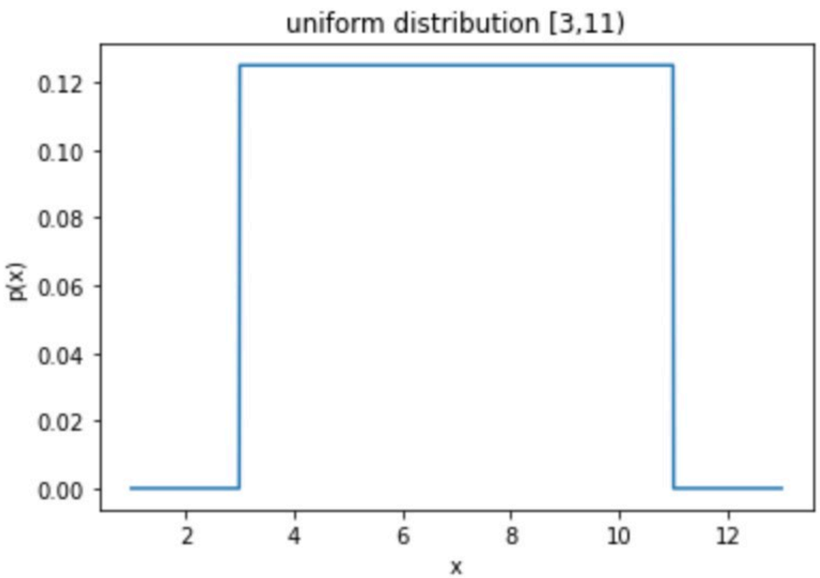
random number generator



general uniform distribution

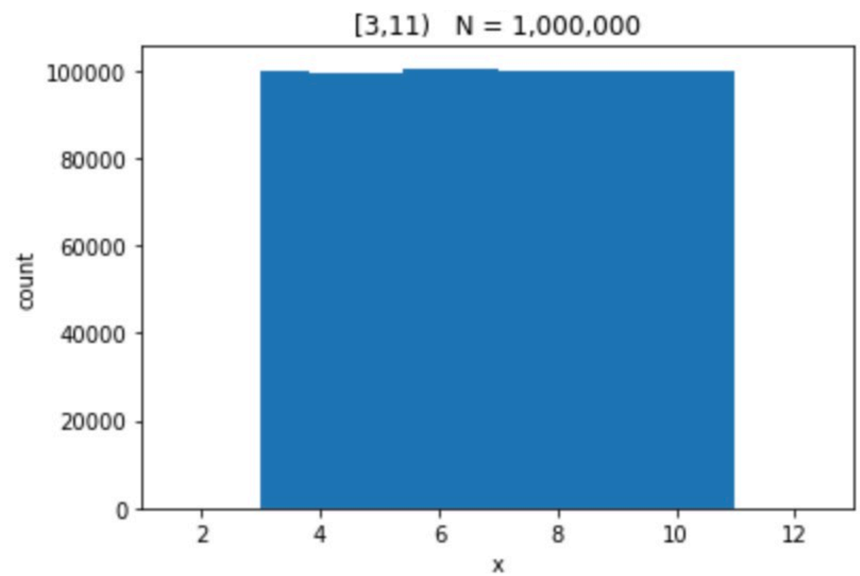
uniform distribution [a,b)

probability density function



uniform PRNG [a,b)

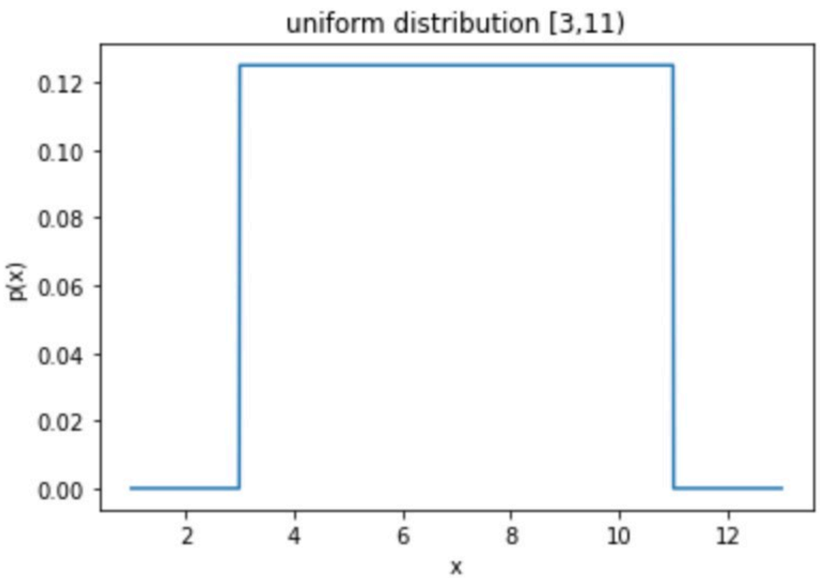
random number generator



general uniform distribution

uniform distribution [a,b)

probability density function



uniform PRNG [a,b)

random number generator

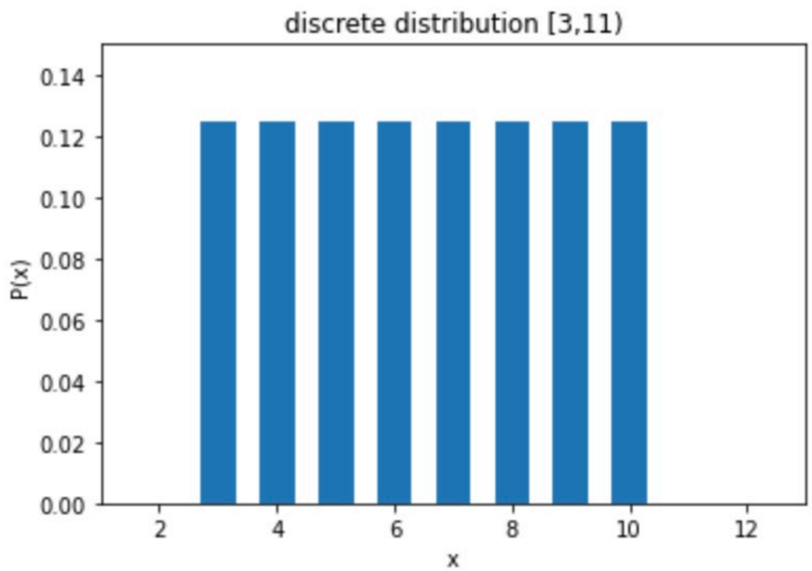
```
a + (b-a)*R.uniform()
```

continuous random number [a,b)

discrete uniform distribution

discrete uniform distribution [a,b)

probability mass function



discrete uniform PRNG [a,b)

random number generator

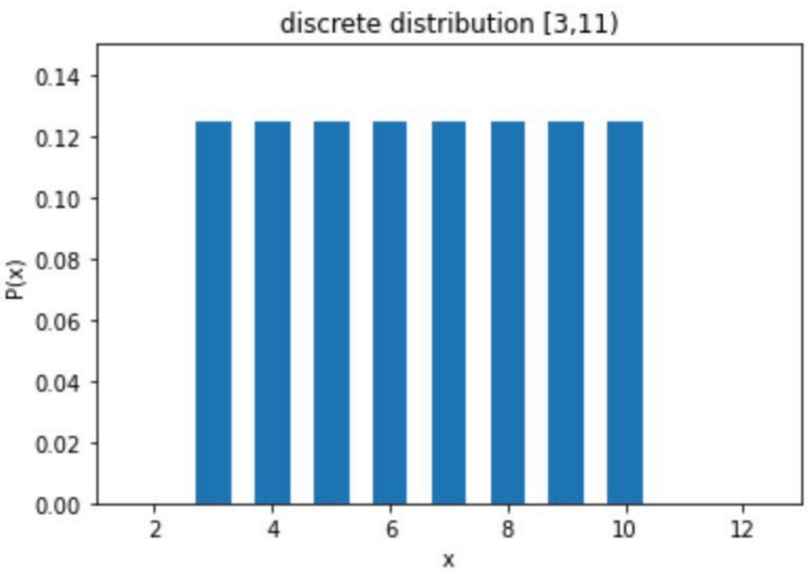
```
np.floor(a + (b-a)*R.uniform())
```

discrete random number [a,b)

discrete uniform distribution

discrete uniform distribution [a,b)

probability mass function



discrete uniform PRNG [a,b)

random number generator

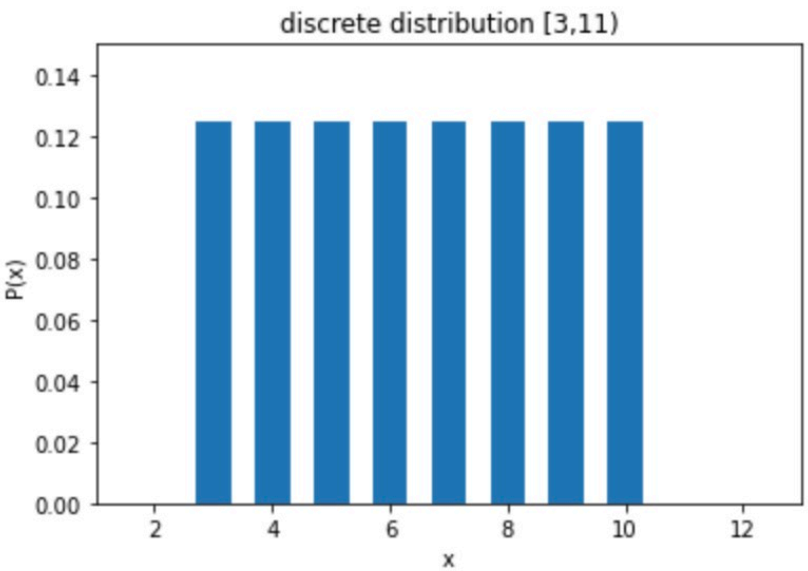
```
R.randint(a, b)
```

discrete random number [a,b)

discrete uniform distribution

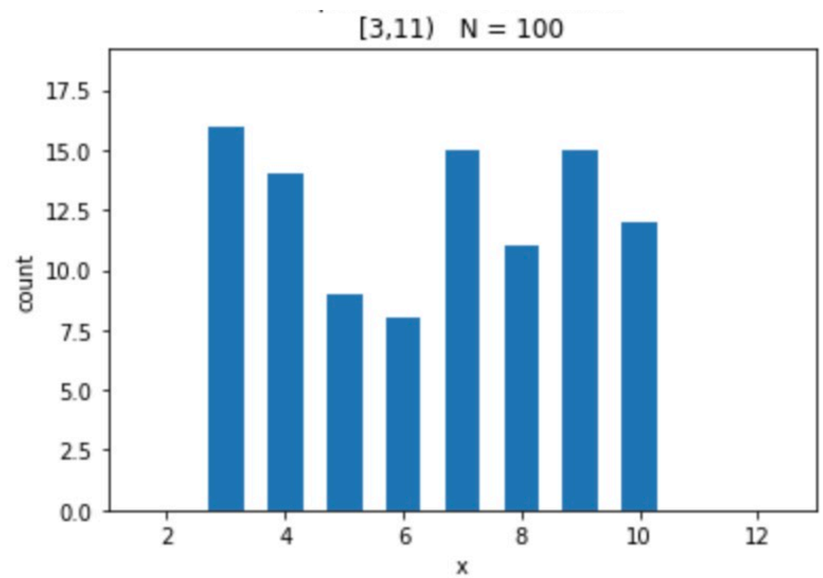
discrete uniform distribution [a,b)

probability mass function



discrete uniform PRNG [a,b)

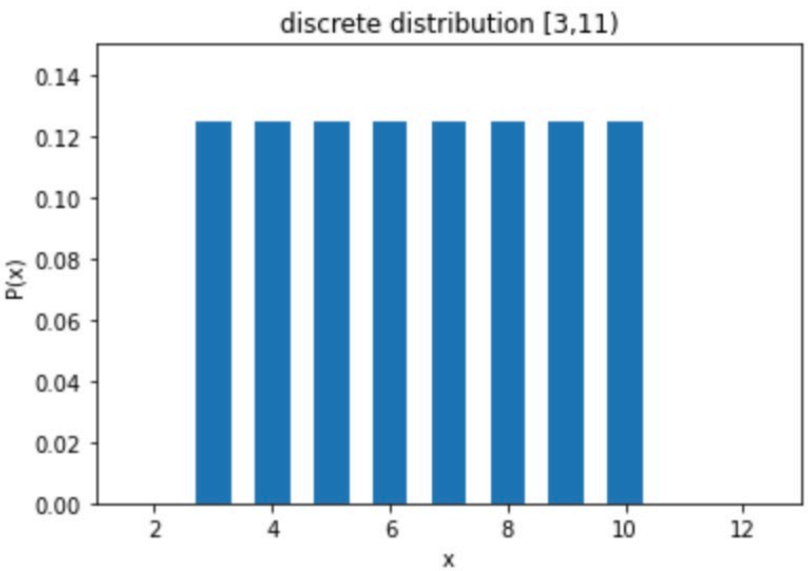
random number generator



discrete uniform distribution

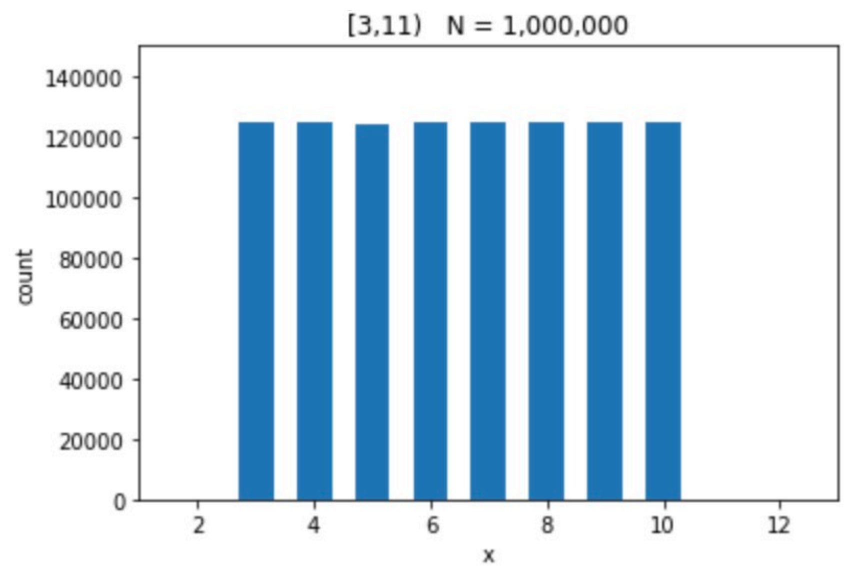
discrete uniform distribution [a,b)

probability mass function



discrete uniform PRNG [a,b)

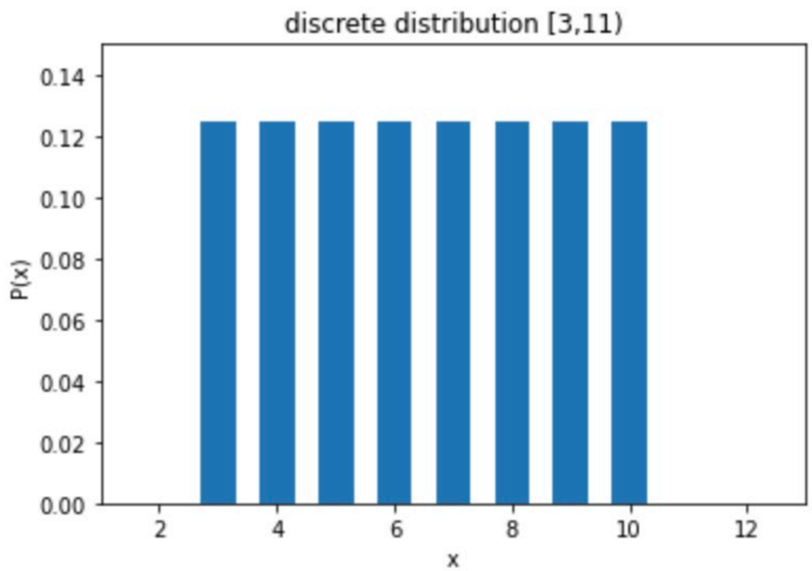
random number generator



discrete uniform distribution

discrete uniform distribution [a,b)

probability mass function



~~discrete uniform PRNG [a,b)~~

random number generator

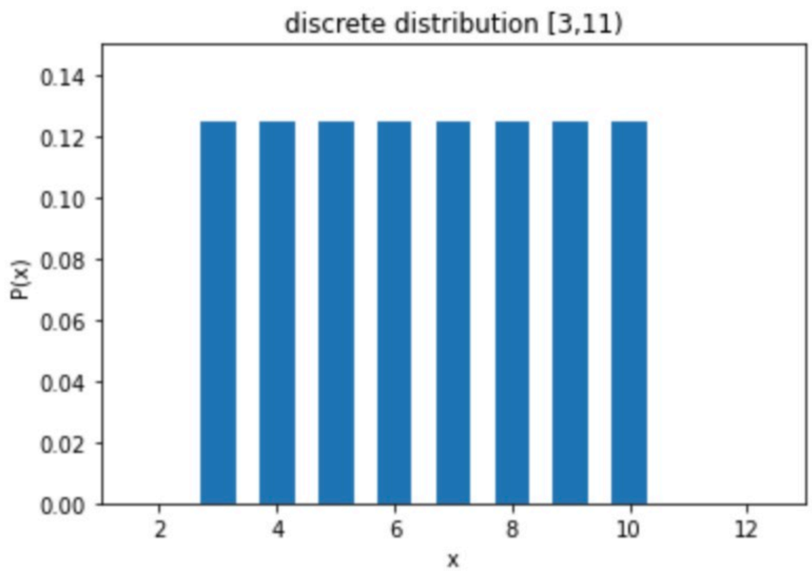
```
np.round(a + (b-a)*R.uniform())
```

contrast with this (which does not work)
but you can find online

discrete uniform distribution

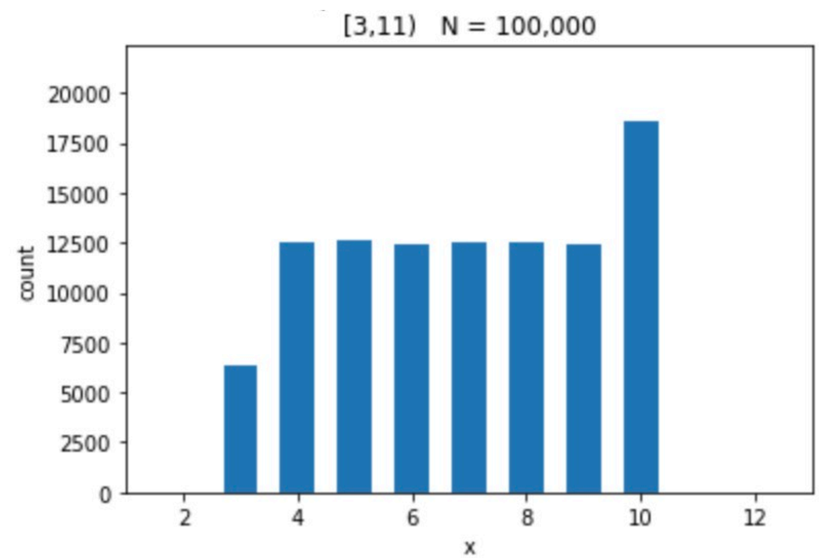
discrete uniform distribution [a,b)

probability mass function



~~discrete uniform PRNG [a,b)~~

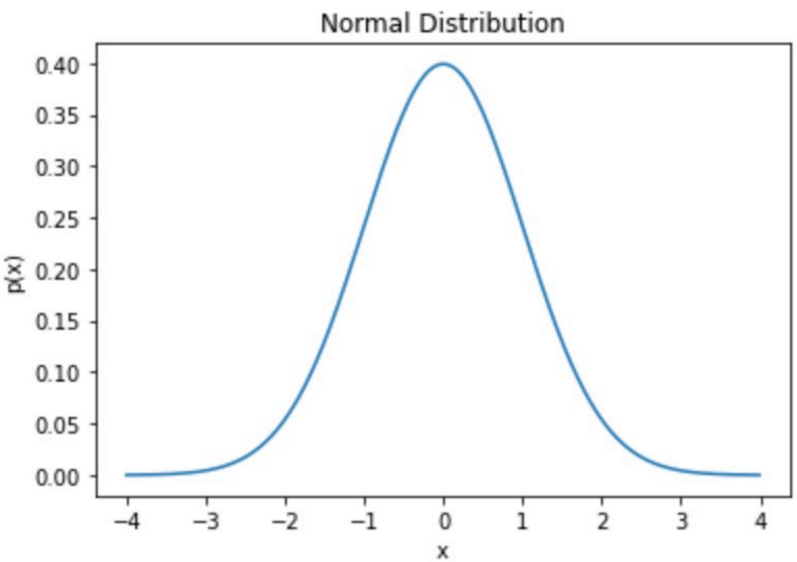
random number generator



standard normal (Gaussian) distribution

normal distribution (mean 0, stdev 1)

probability density function



normal distribution (mean 0, stdev 1)

random number generator

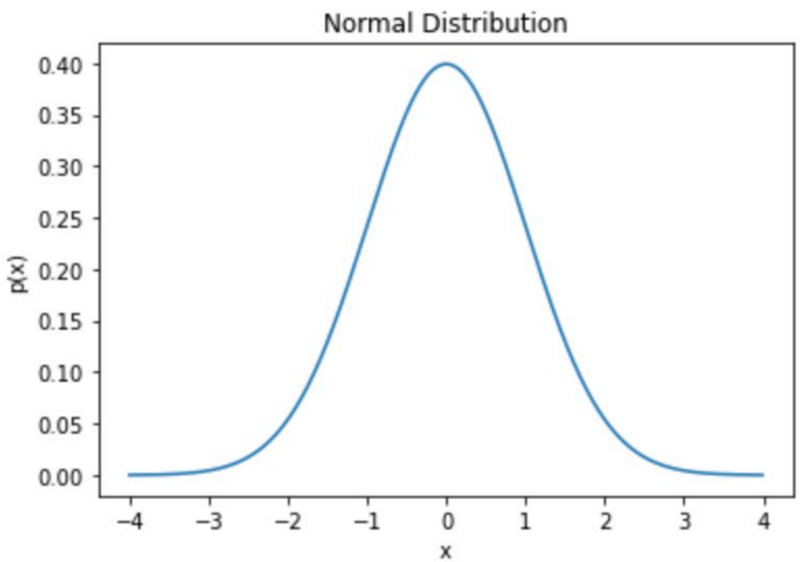
https://en.wikipedia.org/wiki/Box–Muller_transform

```
R.normal()
```

standard normal (Gaussian) distribution

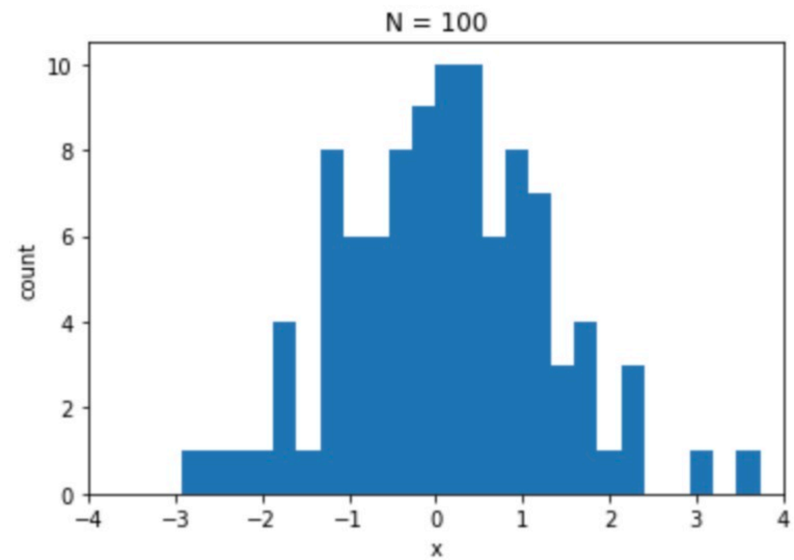
normal distribution (mean 0, stdev 1)

probability density function



normal distribution (mean 0, stdev 1)

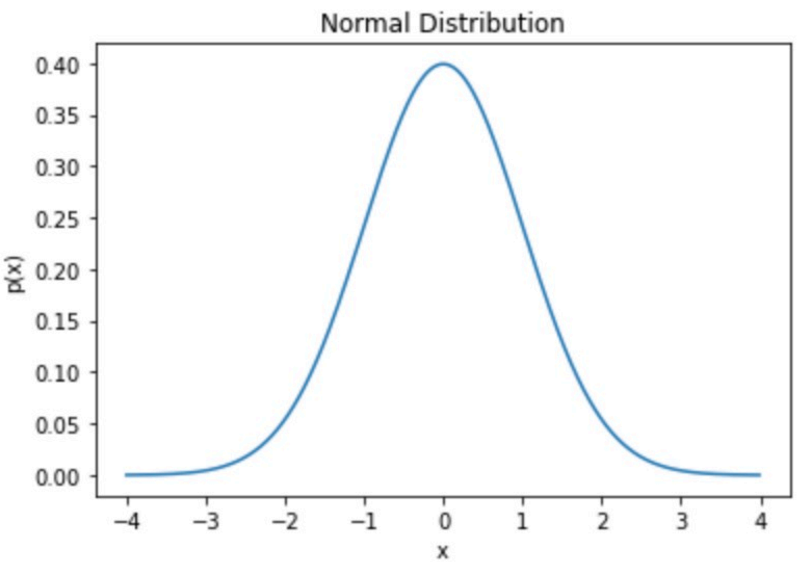
random number generator



standard normal (Gaussian) distribution

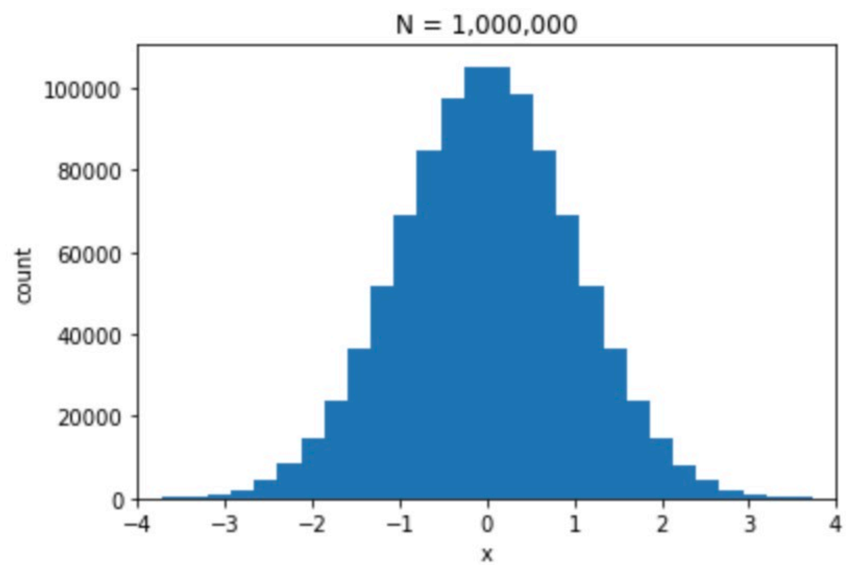
normal distribution (mean 0, stdev 1)

probability density function



normal distribution (mean 0, stdev 1)

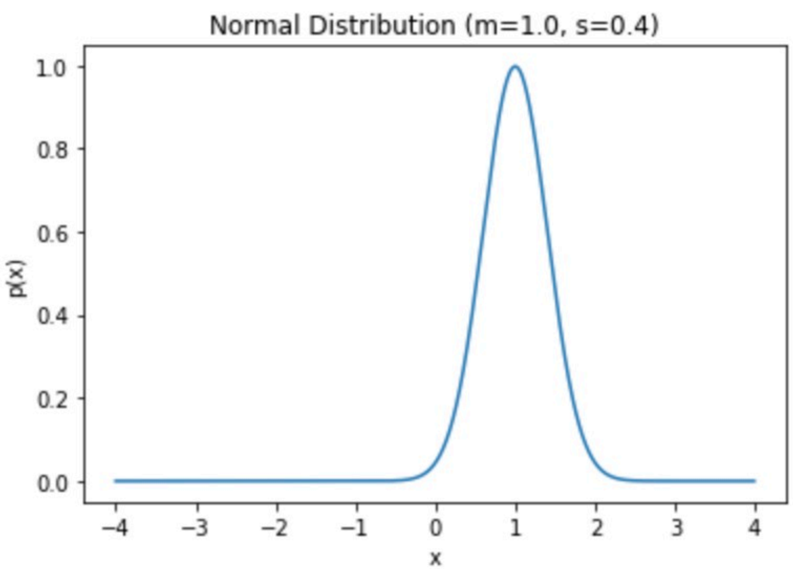
random number generator



normal (Gaussian) distribution

normal distribution (mean μ, stdev σ)

probability density function



normal distribution (mean μ, stdev σ)

random number generator

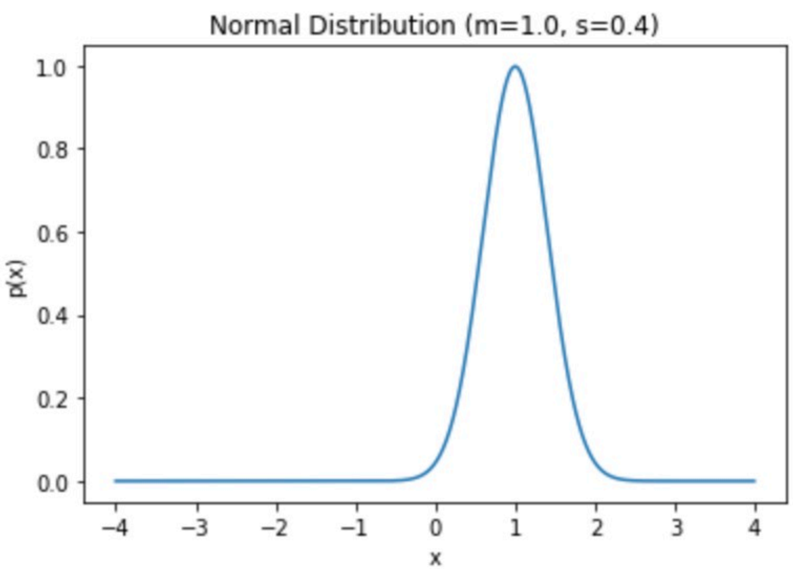
```
m + s*R.normal()
```

other probability distributions also have means and stdevs, but the normal distribution is the only one you can do this kind of operation with

normal (Gaussian) distribution

normal distribution (mean μ, stdev σ)

probability density function



normal distribution (mean μ, stdev σ)

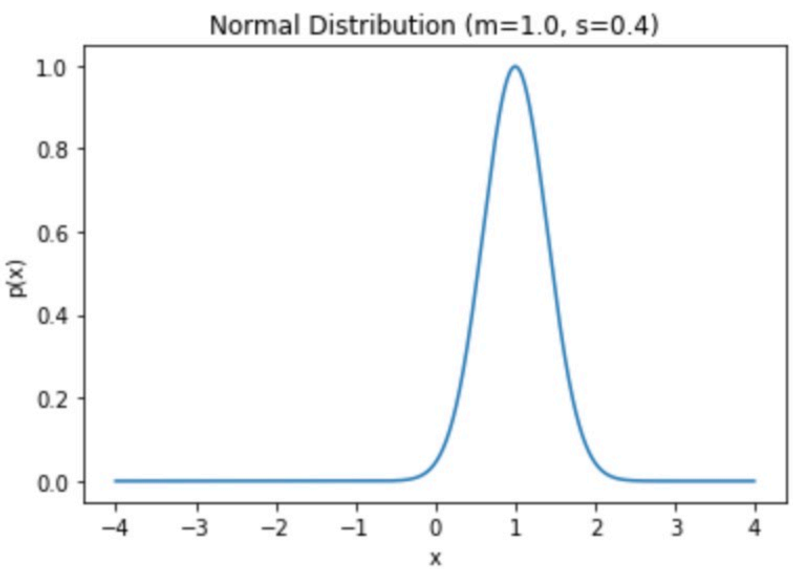
random number generator

```
R.normal(m, s)
```

normal (Gaussian) distribution

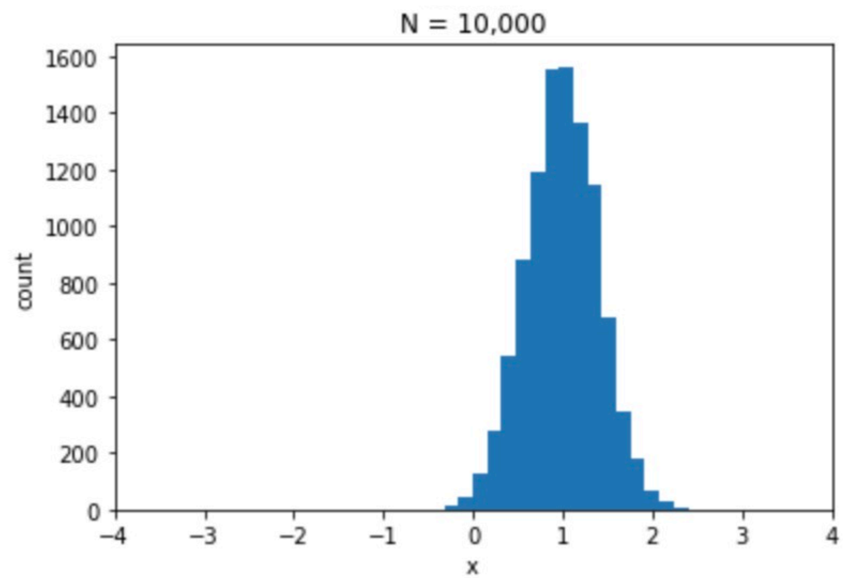
normal distribution (mean μ, stdev σ)

probability density function



normal distribution (mean μ, stdev σ)

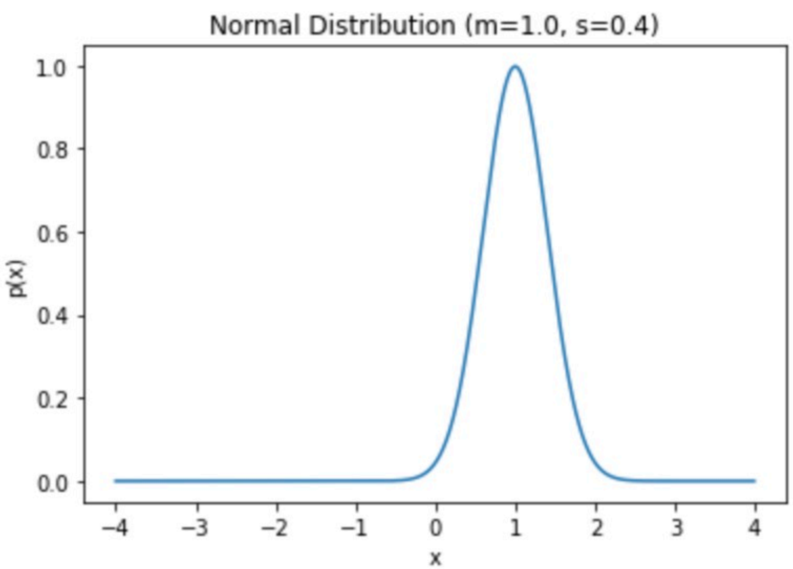
random number generator



normal (Gaussian) distribution

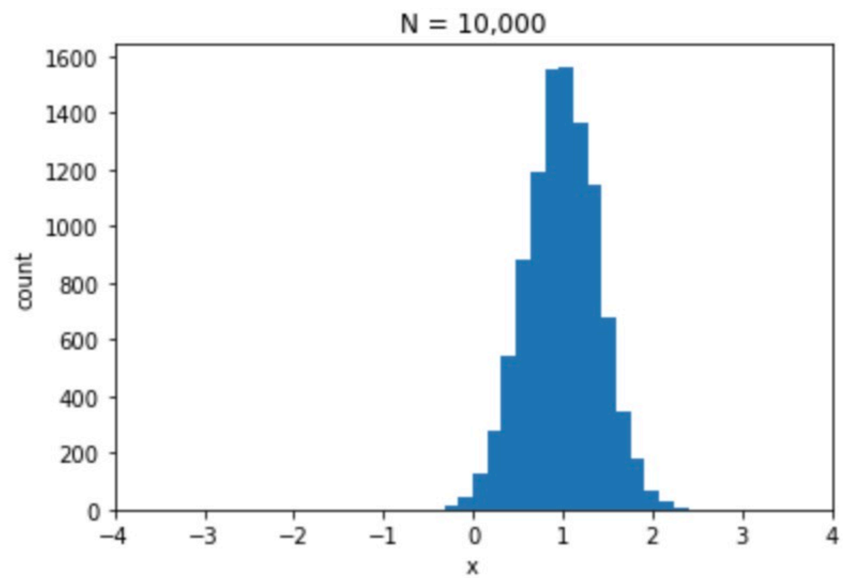
normal distribution (mean μ, stdev σ)

probability density function



normal distribution (mean μ, stdev σ)

random number generator



random numbers from normal distributions are often used to simulate (or inject) "noise"

- change the luminance of individual pixels in an image by some small random amount
- move an x,y location of an object in a display some random amount
- add some variability to level of activation of a unit in a neural network
- these kinds of noise in physical processes often (not always) approximate a normal

Central Limit Theorem : sum (or average) of random numbers from <u>any distribution</u> approach a normal in the limit

simulating Central Limit Theorem

see `Random.ipynb`