np.random.seed(1234) # for reproducibility?

Random类的基值seed是什么意思？  
比如：  
Random ran1=new Random();  
Random ran2=new Random(12345);  
这两个有什么区别啊？运行结果没区别啊！

生成随机数

Random random = new Random();

Random random = new Random(100);//指定种子数100

random调用不同的方法，获得随机数。

如果2个Random对象使用相同的种子（比如都是100），并且以相同的顺序调用相同的函数，那它们返回值完全相同。如下面代码中两个Random对象的输出完全相同

import java.util.\*;

class TestRandom {

public static void main(String[] args) {

Random random1 = new Random(100);

System.out.println(random1.nextInt());

System.out.println(random1.nextFloat());

System.out.println(random1.nextBoolean());

Random random2 = new Random(100);

System.out.println(random2.nextInt());

System.out.println(random2.nextFloat());

System.out.println(random2.nextBoolean());

}

}

====================================================================

一般计算机的随机数都是[伪随机数](http://baike.baidu.com/view/1127.htm" \t "_blank)，以一个真随机数（种子）作为初始条件，然后用一定的算法不停迭代产生随机数，下面介绍两种方法： 

　　一般种子可以以当前的系统时间，这是完全随机的 

　　。 

　　算法1：[平方](http://baike.baidu.com/view/33276.htm" \t "_blank)取中法。 

　　1）将种子设为X0，并mod 10000得到4位数 

　　2)将它平方得到一个8位数（不足8位时前面补0） 

　　3）取中间的4位数可得到下一个4位随机数X1 

　　4）重复1-3步，即可产生多个随机数 

　　这个算法的一个主要缺点是最终它会退化成0，不能继续产生随机数。 

　　算法2：线性同余法 

　　1）将种子设为X0， 

　　2）用一个算法X(n+1)=(a\*X(n)+b) mod c产生X(n+1) 

　　一般将c取得很大，可产生0到c-1之间的伪随机数 

该算法的一个缺点是会出现循环。

np.random.seed(0) makes the random numbers predictable

>>> numpy.random.seed(0) ; numpy.random.rand(4)

array([ 0.55, 0.72, 0.6 , 0.54])

>>> numpy.random.seed(0) ; numpy.random.rand(4)

array([ 0.55, 0.72, 0.6 , 0.54])

With the seed reset, the same numbers will appear every time.

If the random seed is not reset, different numbers appear with every invocation:

>>> numpy.random.rand(4)

array([ 0.42, 0.65, 0.44, 0.89])

>>> numpy.random.rand(4)

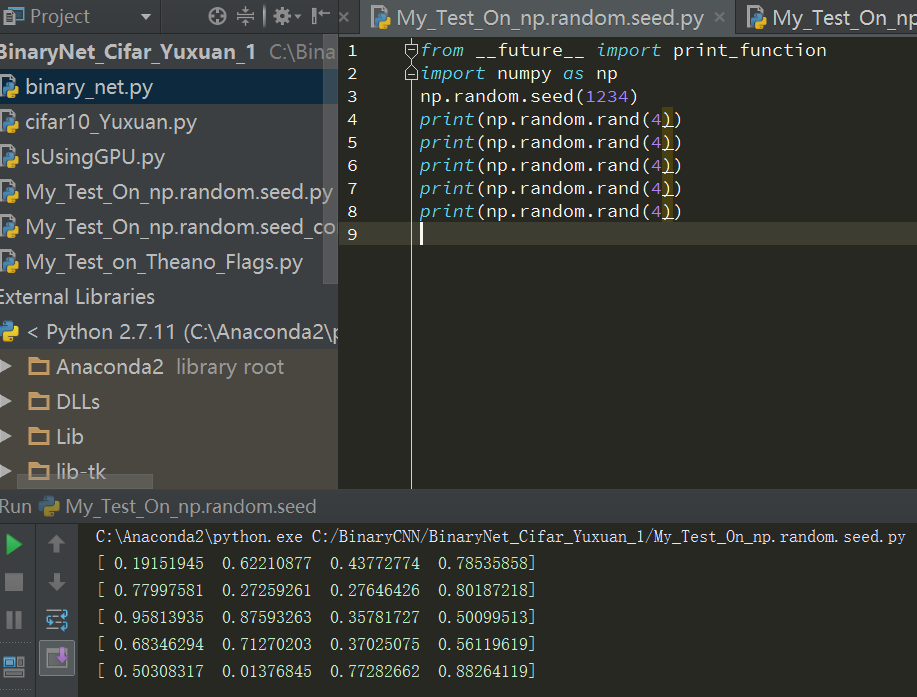
array([ 0.96, 0.38, 0.79, 0.53])

(pseudo-)random numbers work by starting with a number (the seed), multiplying it by a large number, then taking modulo of that product. The resulting number is then used as the seed to generate the next "random" number. When you set the seed, it does the same thing every time, giving you the same numbers.

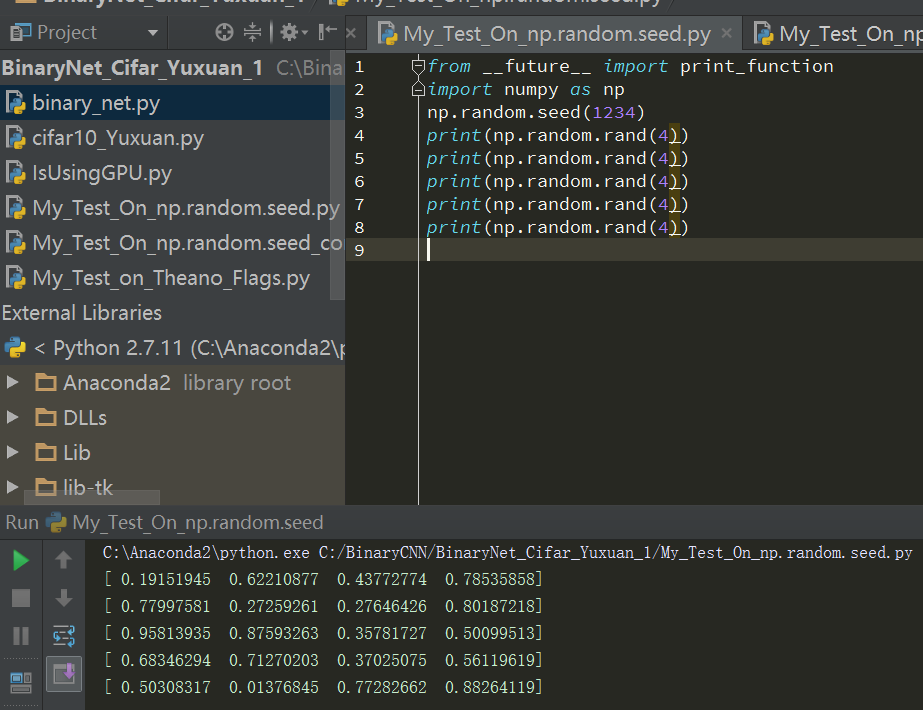
If you want seemingly random numbers, do not set the seed. If you have code that uses random numbers that you want to debug, however, it can be very helpful to set the seed before each run so that the code does the same thing every time you run it.

这个seed的作用是让每次产生的随机数都是一样的。

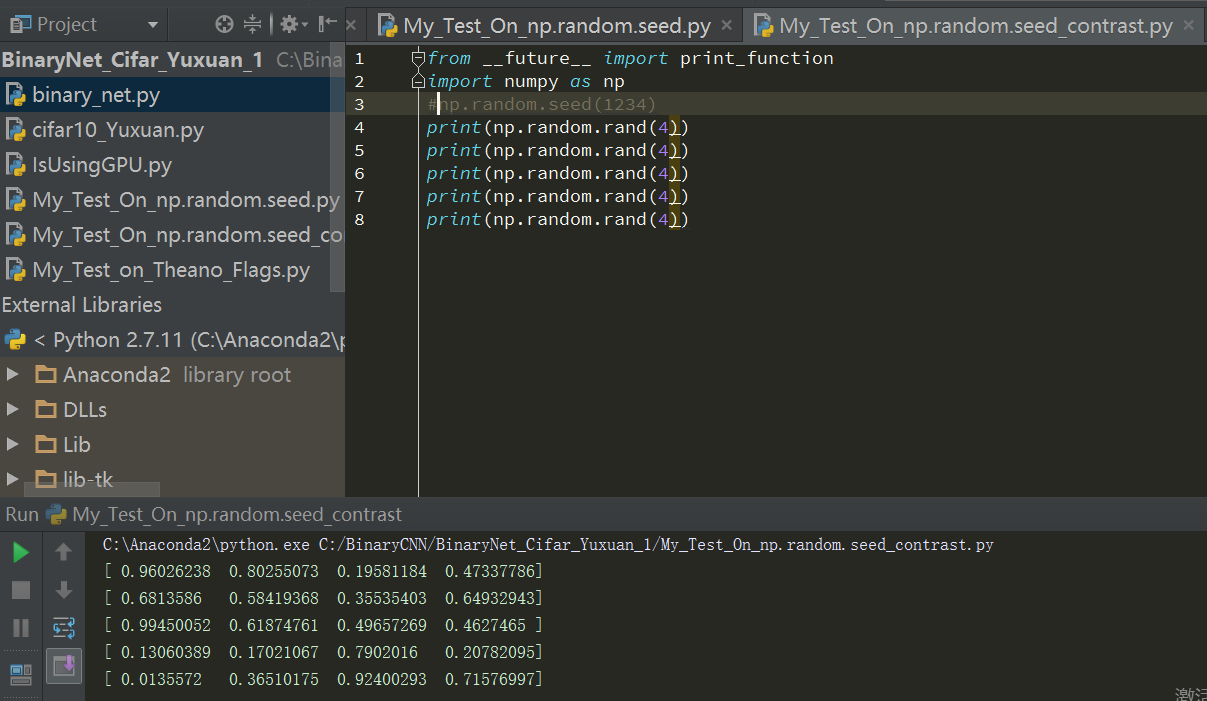
Seed第一次

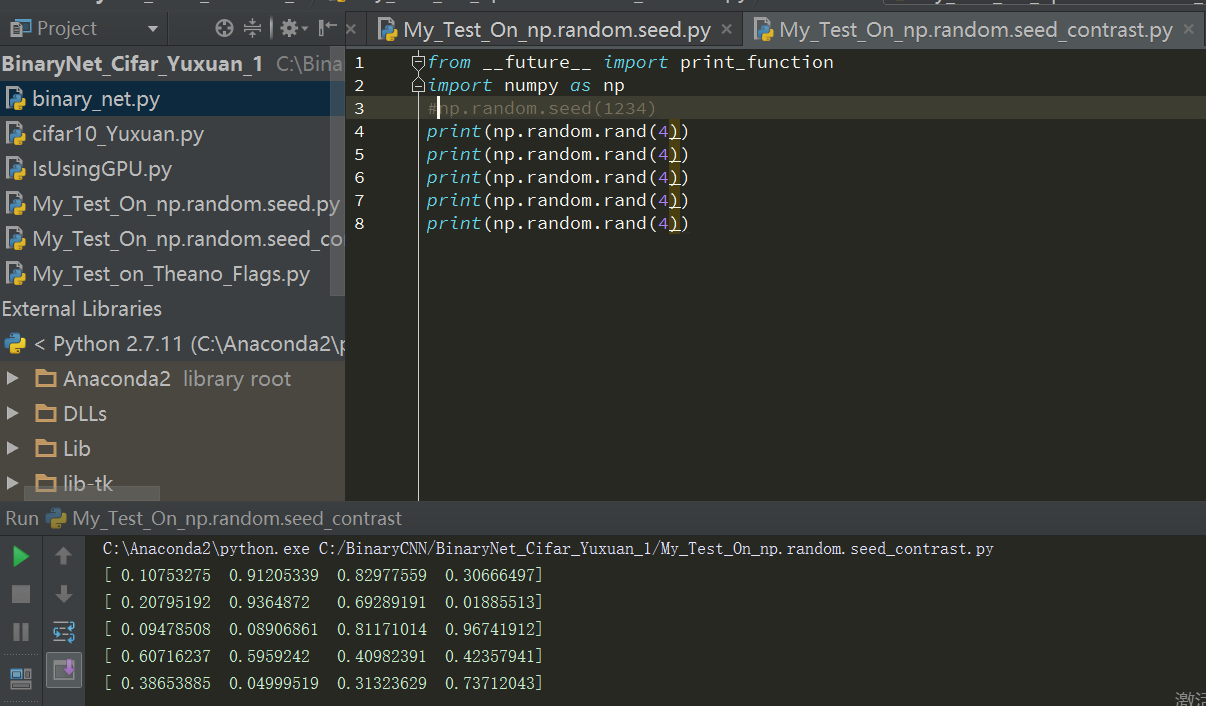


Seed第二次



没有seed第一次



没有seed第二次

所以说是“for reproductivity”

As noted, numpy.random.seed(0) sets the random seed to 0, so the pseudo random numbers you get from random will start from the same point. This can be good for debuging in some cases. HOWEVER, after some reading, this seems to be the wrong way to go at it, if you have threads because it is not thread safe.

from [differences-between-numpy-random-and-random-random-in-python](http://stackoverflow.com/questions/7029993/differences-between-numpy-random-and-random-random-in-python):

For numpy.random.seed(), the main difficulty is that it is not thread-safe - that is, it's not safe to use if you have many different threads of execution, because it's not guaranteed to work if two different threads are executing the function at the same time. If you're not using threads, and if you can reasonably expect that you won't need to rewrite your program this way in the future, numpy.random.seed() should be fine for testing purposes. If there's any reason to suspect that you may need threads in the future, it's much safer in the long run to do as suggested, and to make a local instance of the numpy.random.Random class. As far as I can tell, random.random.seed() is thread-safe (or at least, I haven't found any evidence to the contrary).

example of how to go about this:

from numpy.random import RandomState

prng = RandomState()

print prng.permutation(10)

prng = RandomState()

print prng.permutation(10)

prng = RandomState(42)

print prng.permutation(10)

prng = RandomState(42)

print prng.permutation(10)

may give:

[3 0 4 6 8 2 1 9 7 5]

[1 6 9 0 2 7 8 3 5 4]

[8 1 5 0 7 2 9 4 3 6]

[8 1 5 0 7 2 9 4 3 6]

Lastly, note that there might be cases where initializing to 0 (as opposed to a seed that has not all bits 0) may result to non-uniform distributions for some few first iterations because of the way xor works, but this depends on the algorithm, and is beyond my current worries and the scope of this question.