

## **Лабораторная работа №4 «Использование библиотек Python для работы со случайными числами»**

Порядок выполнения практической работы:

1. Ознакомиться с возможностями модуля `special` библиотеки SciPy для генерации случайных чисел и их последовательностей.  
*Рекомендуемые источники:*
  - приложение1;
  - [https://skillbox.ru/media/code/gotovimsya\\_k\\_sobesedovaniyu\\_sluchaynye\\_chisla\\_v\\_python/](https://skillbox.ru/media/code/gotovimsya_k_sobesedovaniyu_sluchaynye_chisla_v_python/)
  - <https://python-scripts.com/random-data>
  - <https://ps.readthedocs.io/ru/latest/random.html>
2. Изучить примеры генерации случайных чисел средствами библиотеки `random` и модуля `numpy.random` из библиотеки `numpy`. Реализовать приведенные примеры (см. рис. 1- 4) на языке программирования Python, изменив параметры вызываемых операторов (например, диапазоны случайных чисел).
3. Оставить и реализовать на Python задачу случайного выбора объекта из множества (аналогично задаче о шарах или монетах, представленные на рис. 5,6).
4. Сформировать отчет, включающий распечатку программного кода на языке Python с распечаткой результатов и соответствующими комментариями (относительно применяемых параметров процедур и функций, а также решаемой задачи) .

### Приложение 1<sup>1</sup>.

Случайные числа (в контексте программирования) — это искусственно полученная последовательность чисел из определённого диапазона, которая подчиняется одному из законов распределения случайной величины.

Случайные числа и их последовательности используются в математических моделях различных сфер науки и техники: в экономике (страхование, логистика), физике (движение частиц), микроэлектронике, криптографии и др.

Существует два основных способа получения псевдослучайных чисел на Python: с помощью «родной» библиотеки [random](#) и с помощью модуля [numpy.random](#) из библиотеки numpy (см. рис. 1, 2)

```
import random

# Случайное число между 0 и 1 (исключая 1)
random.random()

# Случайное число между 1 и 2, равномерное распределение
random.uniform(1,2)

# Случайное число, нормальное распределение
# 1 и 2 здесь уже не границы диапазона, а параметры  $\mu$  и  $\sigma$ 
random.gauss(1,2)
```

Рис1.

```
from numpy.random import default_rng

rng = default_rng()

vals = rng.standard_normal(10)
```

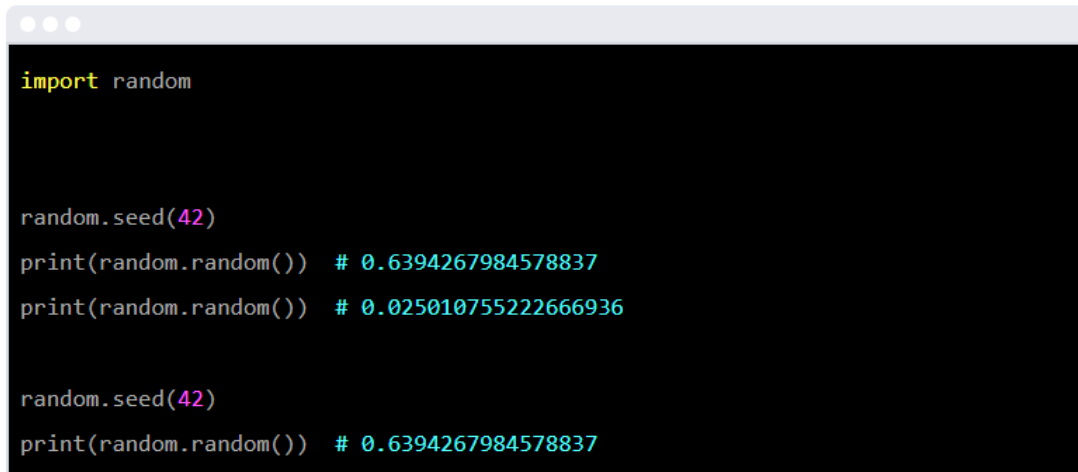
В первой строке мы импортировали `default_rng` — это «генератор генераторов» случайных массивов из модуля `numpy.random`. Во второй — создали экземпляр такого генератора и присвоили ему имя `rng`. В третьей использовали его метод `standard_normal`, чтобы получить numpy-массив из 10 случайных чисел, и записали массив в переменную `vals`.

---

<sup>1</sup> Все приведенные примеры проверены в IDE Microsoft Visual Studio, см. рис. 7, 8.

Рис. 2.

Истинно случайную последовательность повторить невозможно. Но для повторения псевдослучайных чисел в обеих основных библиотеках — `random` и `numpy.random` есть функция `seed()`, которая отвечает за инициализацию («посев») последовательности (см. рис. 3):.



```
import random

random.seed(42)
print(random.random()) # 0.6394267984578837
print(random.random()) # 0.02501075522666936

random.seed(42)
print(random.random()) # 0.6394267984578837
```

Рис. 3.

Передавая аргумент 42 в функцию `seed()`, мы указываем конкретное место в псевдослучайной последовательности, поэтому команда `random.random()` в третьей и последней строках выдаёт одинаковое число — оно идёт первым после точки, помеченной как `seed(42)`.

В `seed()` можно передать целые и дробные числа, а также строки и кортежи. Если оставить скобки пустыми, то в качестве аргумента `seed()` возьмёт текущее системное время.

Аналогичная функция есть в модуле `numpy.random` (см. рис. 4):

```
import numpy as np

np.random.seed(42)
print(np.random.rand()) # 0.3745401188473625
print(np.random.rand()) # 0.9507143064099162

np.random.seed(42)
print(np.random.rand()) # 0.3745401188473625
```

Рис. 4.

Например, код для численной проверки ответа к задачке «Какова вероятность вытащить зелёный шар из мешка, в котором 1 зелёный и 4 красных шара» представлен на рис. 5. (Ответ  $\frac{1}{5} = 0,2$ ).

Иными словами, если 100 раз вынимать шар из мешка, возвращая его обратно, количество выпадения зелёных шаров должно приближаться к 20. Вариант кода для проверки:

```
import random

green_ball_count = 0 # счётчик зелёных шаров

for i in range(0,100):
    new_ball = random.choice(['green', 'red', 'red', 'red', 'red'])
    if new_ball == 'green':
        green_ball_count = green_ball_count + 1

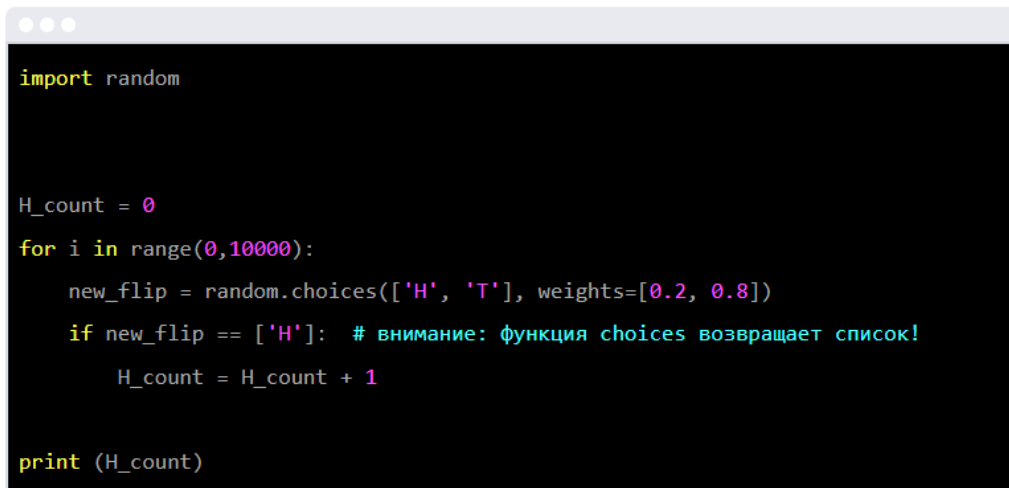
print (green_ball_count)
```

Функция `random.choice()` случайным образом выбирает значение из заданного диапазона — списка из одного «green» и четырёх «red». Код выведет количество зелёных шаров после 100 попыток.

Рис. 5

Другой вариант: предположим, у нас есть так называемая «нечестная» монетка, где орёл (H, «heads») и решка (T, «tails») выпадают не с вероятностью  $\frac{1}{2}$ , как положено, а по-другому: орёл с вероятностью  $p(H) = 0,2$ , а решка, соответственно,  $p(T) = 0,8$ .

Тогда код для проверки будет выглядеть примерно так (см. рис. 6):



```
import random

H_count = 0
for i in range(0,10000):
    new_flip = random.choices(['H', 'T'], weights=[0.2, 0.8])
    if new_flip == ['H']: # внимание: функция choices возвращает список!
        H_count = H_count + 1

print (H_count)
```

Здесь используется другая функция, `choices`, в которую вместе со списком значений можно в параметре `weights` передавать вероятности их выпадения.

Код выведет количество выпавших орлов после 10 000 бросков.

---

Рис. 6

```
import random
from numpy.random import default_rng
import numpy as np
print('generaciya sluchainih chisel i posledovatelnostey sredstvami modulya RANDOM:\n')
print(random.random())
print(random.uniform(1,2))
print(random.gauss(1,2))
rng=default_rng()
vals = rng.standard_normal(10)
print('massiv iz 10ti sluchainih chisel:\n', vals)
print('povtor psevdosluchainih posledovatelnostey:\n')
random.seed(42)
print(random.random())
print(random.random())
random.seed(42)
print(random.random())

print('\ngeneraciya sluchainih chisel i posledovatelnostey sredstvami modulya NUMPY.RANDOM:\n')
np.random.seed(42)
print(np.random.rand())
print(np.random.rand())
np.random.seed(42)
print(np.random.rand())

print('\nzadacha o raznocvetnih sharah:')
green_ball_count = 0 # счётчик зелёных шаров

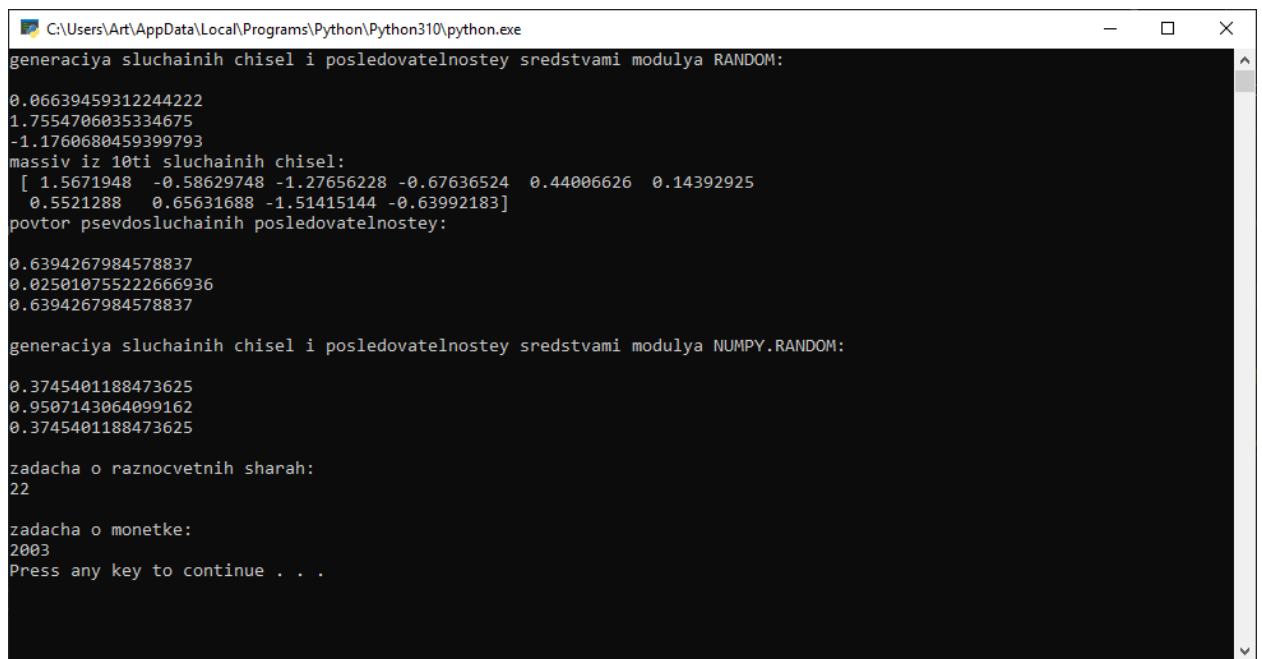
for i in range(0,100):
    new_ball = random.choice(['green', 'red', 'red', 'red', 'red'])
    if new_ball == 'green':
        green_ball_count = green_ball_count + 1

print (green_ball_count)

print('\nzadacha o monetke:')
H_count = 0
for i in range(0,10000):
    new_flip = random.choices(['H', 'T'], weights=[0.2, 0.8])
    if new_flip == ['H']: # внимание: функция choices возвращает список!
        H_count = H_count + 1

print (H_count)
```

Рис. 7. Листинг, полученный в ходе проверки программного кода, приведенного в примерах, в IDE Microsoft Visual Studio.



```
C:\Users\Art\AppData\Local\Programs\Python\Python310\python.exe
generaciya sluchainih chisel i posledovatelnostey sredstvami modulya RANDOM:
0.06639459312244222
1.7554706035334675
-1.1760680459399793
massiv iz 10ti sluchainih chisel:
[ 1.5671948 -0.58629748 -1.27656228 -0.67636524  0.44006626  0.14392925
  0.5521288  0.65631688 -1.51415144 -0.63992183]
povtor psevdosluchainih posledovatelnostey:
0.6394267984578837
0.025010755222666936
0.6394267984578837

generaciya sluchainih chisel i posledovatelnostey sredstvami modulya NUMPY.RANDOM:
0.3745401188473625
0.9507143064099162
0.3745401188473625

zadacha o raznocvetnih sharah:
22

zadacha o monetke:
2003
Press any key to continue . . .
```

Рис. 8. Результаты работы программы.