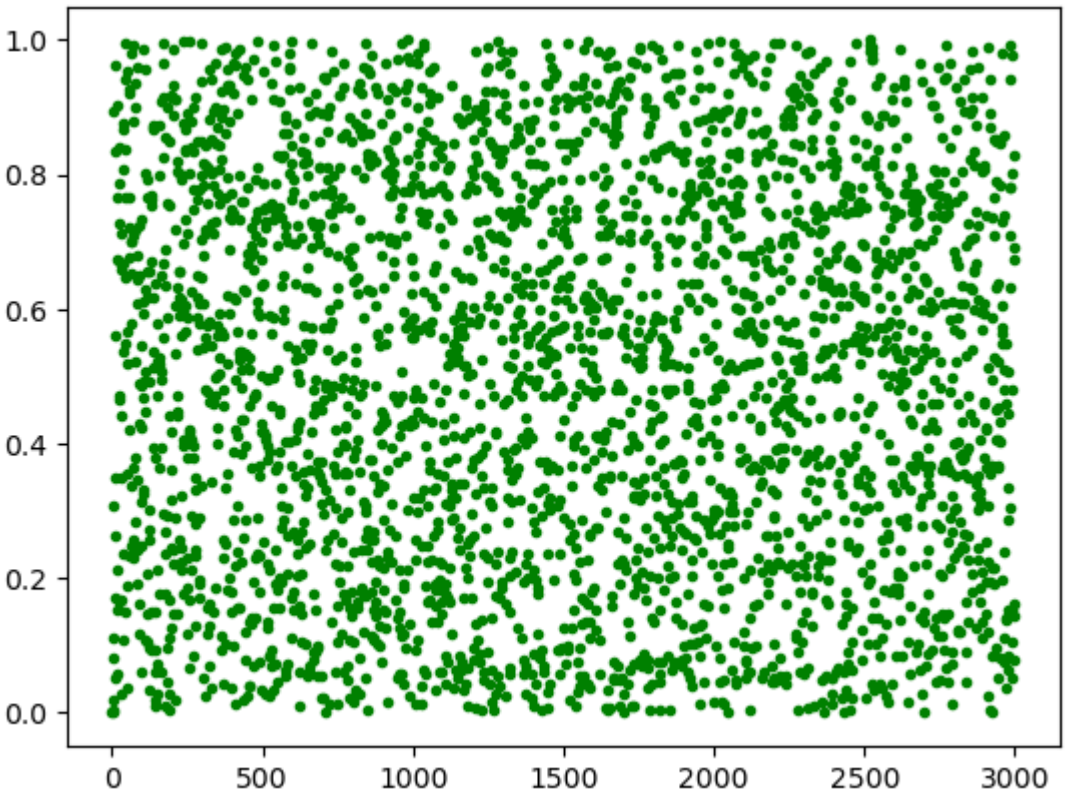
Simulation of random variables using a linear congruence generator.

Using Fortran's congruence generator (IBM)

In [3]:
```
alea=[];
alea.append(50);
for i in range(0,50):
    alea.append((2**16+3)*alea[i]%(2**32));
alea_norm=[alea[i]/(2**32) for i in range(0,50)];
print(alea_norm)
#print(alea)
#20 then 3000
```

```
[1.1641532182693481e-08, 0.0007629743777215481, 0.004577741492539644, 0.020599679
555743933, 0.0823984039016068, 0.3089933074079454, 0.11237420933321118, 0.8933054
893277586, 0.34846505196765065, 0.051040907856076956, 0.17005997942760587, 0.5609
917058609426, 0.8354104203172028, 0.9635371691547334, 0.26252923207357526, 0.9033
408700488508, 0.057282131630927324, 0.21362495934590697, 0.7662105713970959, 0.67
46387942694128, 0.15193762304261327, 0.8398765898309648, 0.6718209316022694, 0.47
203628113493323, 0.7858293023891747, 0.4666492841206491, 0.7274319832213223, 0.16
47483422420919, 0.4416022044606507, 0.16687814658507705, 0.026849039364606142, 0.
6591909169219434, 0.7135041472502053, 0.34830663120374084, 0.6683024619705975, 0.
8750550909899175, 0.23560838820412755, 0.5381545103155077, 0.10845156805589795, 0
.8073188154958189, 0.8678487804718316, 0.9412233433686197, 0.836701035965234, 0.5
491961254738271, 0.7648674291558564, 0.6464394456706941, 0.9948298116214573, 0.15
102385869249701, 0.952674847561866, 0.35683435713872313]
```

In [4]:
```
import matplotlib.pyplot as plt
plt.plot(range(0,3000),alea_norm,'.',color='g')
plt.show()
```



In [6]:
```
%matplotlib
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
alea1=[alea_norm[3*i] for i in range(0,1000)];
alea2=[alea_norm[3*i+1] for i in range(0,1000)];
alea3=[alea_norm[3*i+2] for i in range(0,1000)];
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(alea1, alea2, alea3, c='r', marker='o');
ax.set_xlabel('alea1')
ax.set_ylabel('alea2')
ax.set_zlabel('alea3')

plt.show()
```

```
Using matplotlib backend: MacOSX
```

In [7]:
```
import numpy.random as npr
a=npr.rand()
print('random value:',a)
print('random value: {:.7f}'.format(a))
npr.seed(2025)
a=npr.rand()
b=npr.rand()
c=npr.rand()
print(a,b,c)
npr.seed(2025)
a=npr.rand()
b=npr.rand()
c=npr.rand()
print(a,b,c)
```

```
npr.seed() # it uses the current time as the seed value
a=npr.rand()
b=npr.rand()
c=npr.rand()
print(a,b,c)
```

```
random value: 0.32762316486049825
random value: 0.3276232
0.1354881636779618 0.887851702730378 0.9326056398865025
0.1354881636779618 0.887851702730378 0.9326056398865025
0.3097299374127067 0.4158611883324189 0.968085811840334
```

In [8]:
```
import numpy.random as npr
import numpy as np
m=200
outcome=[]
proba=[0, 0.1, 0.6, 0.3]
values=[1,7,8]
cumprob=np.cumsum(proba)
coin=npr.rand(m)
for i in range(0,m):
    k=sum((coin[i]>cumprob)*1)-1
    outcome.append(values[k])
print(outcome)
import matplotlib.pyplot as plt
plt.hist(outcome,density=True)
plt.show()
```

```
[7, 8, 8, 7, 8, 7, 8, 8, 8, 7, 8, 8, 7, 8, 7, 7, 7, 7, 7, 7, 7, 7, 7, 1, 8, 7, 7,
7, 8, 1, 1, 8, 7, 8, 7, 8, 7, 8, 7, 8, 8, 7, 7, 7, 7, 7, 8, 8, 8, 1, 8, 7, 1, 7,
1, 7, 7, 1, 8, 1, 7, 7, 7, 7, 7, 8, 7, 8, 7, 8, 7, 7, 7, 7, 8, 7, 7, 7, 1, 8, 8,
7, 1, 7, 8, 7, 8, 7, 7, 8, 7, 8, 8, 8, 7, 7, 7, 7, 8, 7, 7, 7, 8, 8, 7, 8, 8, 8,
8, 1, 7, 7, 8, 1, 7, 7, 7, 7, 7, 8, 7, 1, 7, 8, 7, 7, 7, 7, 1, 8, 7, 7, 7, 8, 7,
7, 8, 7, 8, 8, 7, 8, 7, 8, 8, 7, 1, 8, 8, 7, 8, 7, 7, 7, 7, 8, 7, 7, 7, 7, 8, 1,
8, 7, 8, 7, 7, 1, 8, 1, 8, 7, 1, 7, 7, 8, 7, 7, 7, 8, 7, 7, 8, 1, 7, 7, 1, 7, 7,
7, 7, 7, 8, 8, 7, 7, 7, 7, 7, 7]
```

In [9]:
```
p=0.2;
print((npr.rand()<=p))
print((npr.rand()<=p)*1)
```

```
True
0
```

In [10]:
```
p=0.2;
n=10;
a=(npr.rand(n)<p)*1
binom=sum(a)
print(a)
print(binom)
```

```
[1 0 0 0 0 0 1 1 0 1]
4
```

In [11]:
```
n=10;
lambda0=1;
var=-np.log(npr.rand(n))/lambda0;
print(var)
```

```
[1.66633425 1.52111289 1.28888837 1.37214109 0.54077533 0.3734536
 1.18128997 1.42418584 0.85980459 0.21336129]
```

In [13]:
```
n=100;
p=0.1;
var=np.floor(np.log(npr.rand(n))/np.log(1-p))+1;
print(var)
```

```
[13. 19. 10. 22. 21. 20.  4.  4. 25.  6.  3. 12.  3. 13.  4. 16. 33.  4.
  4.  2.  7. 12.  3.  3. 29. 20.  6. 42. 18.  5.  7. 11.  5.  3.  5.  9.
 27.  7. 13.  2.  6.  3.  9. 12.  2.  3. 31.  3.  5. 17.  7.  8. 11.  1.
 12.  6. 18.  6. 11.  7.  3.  8.  4.  7. 17.  9.  3.  7. 10. 24.  7. 27.
  3.  7.  8.  3.  5.  9.  4. 13. 14.  3. 10.  8.  6. 21. 16.  1.  3.  1.
  6. 15. 30. 15.  4.  3. 21. 12. 22.  3.]
```

In [16]:
```
n=10;
mu=0;
sigma=1;
print(npr.normal(mu,sigma,n))
```

```
[-0.93397138  1.58991113 -1.07445361 -0.4209596   2.06633613 -1.91859686
  0.10345337 -0.12858846  0.67393238 -0.76958424]
```