1

a

In [74]:
```
from scipy.stats import binom

f=0.01
i=0.58
while f < 0.346:
    f=binom.pmf(3,4,i)
    i+=0.00001

print(f"Desired fraction of 3 heads: {f:.3f}")
print(f"Probability of getting 1 head: {i:.2f}")
```

```
Desired fraction of 3 heads: 0.346
Probability of getting 1 head: 0.60
```

b

In [75]:
```
from scipy.stats import binom
from math import floor

f=0.01
i=0.58
while f < 0.346:
    f=binom.pmf(3,4,i)
    i+=0.00001

b=binom.stats(4,0.6)
print(f'Proability of heads: {(b[0])} +- {b[1]}')
```

```
Proability of heads: 2.4 +- 0.96
```

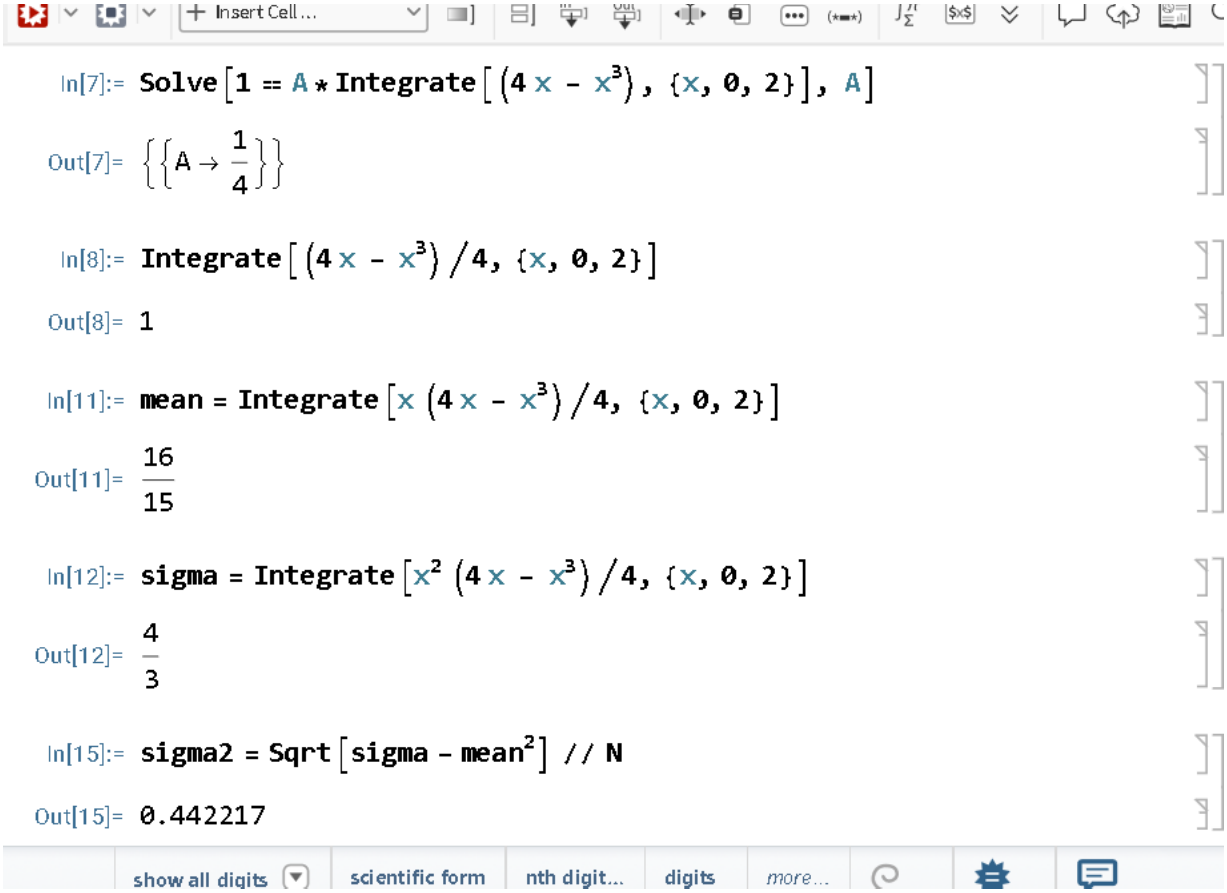
c

In [34]:
```
from scipy.stats import poisson
from scipy.stats import binom
from math import floor

f=0.01
i=0.59
coins=4

while f < 0.346:
    f=binom.pmf(3,4,i)
    i+=1e-6

same=binom.pmf(0,4,i)+ binom.pmf(4,4,i)
print(f'Fraction of all coins getting showing the same side: \n{same:.3f}')
```

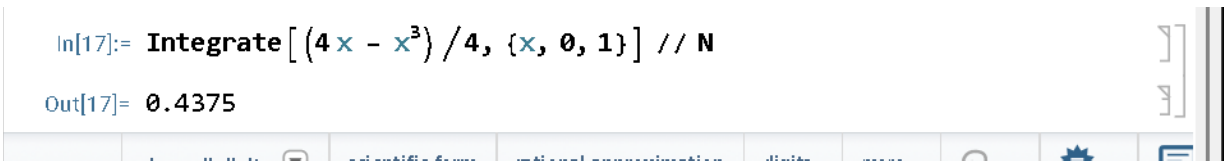
```
Fraction of all coins getting showing the same side:
0.155
```

2

a



b



3

a

```
In [63]: from numpy import exp, mean, std
         from numpy.random import randn

         N = 100000
         x = 0.35 + 0.05 * randn(N)
         y = 1.25 + 0.02 * randn(N)

         B = (4*x**2 + 1) * exp(-2*y**2)

         print(f"B = {mean(B):.4f} +- {std(B):.4f}")
```

```
B = 0.0661 +- 0.0091
```

b

In [4]:
```
from numpy import exp, sqrt

# Given values
x, y = 0.35, 1.25
sigma_x, sigma_y = 0.05, 0.02

# Partial derivatives
dB_dx = 8*x * exp(-2*y**2)
dB_dy = (4*x**2 + 1) * (-4*y) * exp(-2*y**2)

# Original uncertainty
sigma_B = sqrt((dB_dx * sigma_x)**2 + (dB_dy * sigma_y)**2)

# Target uncertainty (25% reduction)
sigma_B_target = 0.75 * sigma_B

# Solve for new sigma_x
sigma_x_new = sqrt((sigma_B_target**2 - (dB_dy * sigma_y)**2) / dB_dx**2)

# Number of measurements and time
n = (sigma_x / sigma_x_new)**2
time = n * sqrt(10)

print(f"Time required: {time:.1f} seconds = {time/60:.2f} minutes")
```

```
Time required: 47.2 seconds = 0.79 minutes
```

4

In [97]:
```
import numpy as np
from scipy.integrate import quad

N = 3000
T = []

for i in range(N):
    x0    = np.random.normal(0.50, 0.02)
    v0    = np.random.normal(2.45, 0.03)
    alpha = np.random.normal(1.15, 0.07)
    beta  = np.random.normal(0.023, 0.02)

    if alpha <= 0 or beta <= 0:
        continue

    try:
        exp0 = np.exp(beta * x0**2)
        arg  = exp0 + beta * v0**2 / alpha
        if arg <= 0:
            continue
        x_final = np.sqrt( np.log(arg) / beta )
        if x_final <= x0:
            continue

        def v(x):
            inner = v0**2 - (alpha/beta) * (np.exp(beta*x**2) - exp0)
            return np.sqrt(max(inner, 0))

        t, err = quad(lambda x: 1/v(x) if v(x)>1e-10 else 1e10, x0, x_final,
                      epsabs=1e-5, epsrel=1e-5, limit=60)

        if err < 0.02 and 0.3 < t < 5:
            T.append(t)

    except:
        continue

if T:
    print(f"Valid: {len(T)} / {N}  ({len(T)/N:.1%})")
    print(f"t = {np.mean(T):.3f} ± {np.std(T):.3f} s")
```

```
Valid: 2646 / 3000  (88.2%)
t = 1.192 ± 0.056 s
```

5

In [33]:
```
from scipy.stats import binom
from numpy import mean, std, array
from pandas import DataFrame
from scipy.special import erf, erfinv

F=array([2.86, 2.81, 2.88, 2.66, 2.75,
         2.80, 2.72, 2.74, 2.84, 2.64,
         2.88, 2.75, 6.85, 2.66, 2.84, 2.61])


n_max=(F.max()-F.mean())/F.std()
n_min=(F.min()-F.mean())/F.std()

reject_max=(1-erf(n_max/sqrt(2)))
reject_min=(1-erf(n_min/sqrt(2)))

print(f"Minimum value: {F.min()} \nChauvenet's value: {reject_min:.6f}")
print("")
print(f"Maximum value: {F.max()} \nChauvenet's value: {reject_max:.6f}")


F=DataFrame(F)
display(F.describe())
```

```
Minimum value: 2.61
Chauvenet's value: 1.318928

Maximum value: 6.85
Chauvenet's value: 0.000114
```

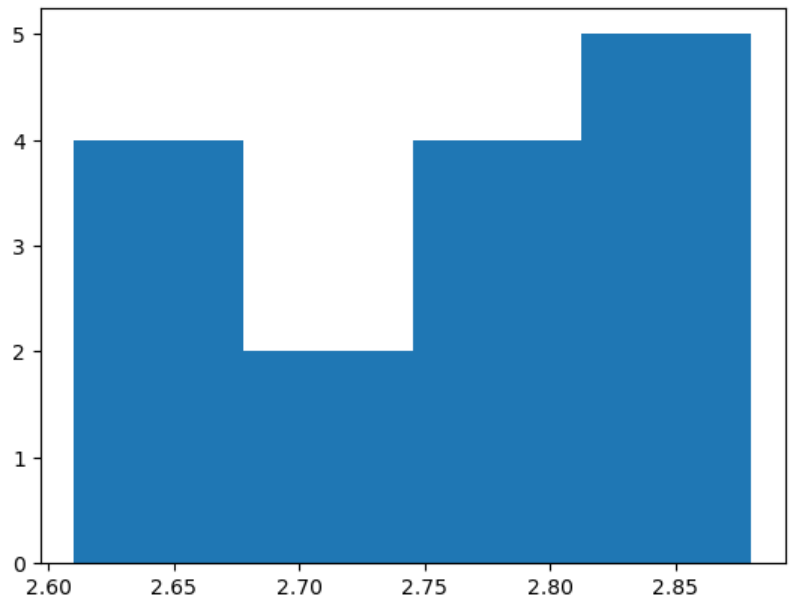
	0
count	16.000000
mean	3.018125
std	1.025568
min	2.610000
25%	2.705000
50%	2.775000
75%	2.845000
max	6.850000

Because Chauvenets's value for 6.85 in the set is so small, we should definitely not include it.

In [32]:
```
from scipy.stats import binom
from numpy import mean, std, array
from pandas import DataFrame
from scipy.special import erf, erfinv

F=array([2.86, 2.81, 2.88, 2.66, 2.75,
         2.80, 2.72, 2.74, 2.84, 2.64,
         2.88, 2.75, 6.85, 2.66, 2.84, 2.61])

F=F[F!=F.max()]

F=DataFrame(F)
display(F.describe())
```

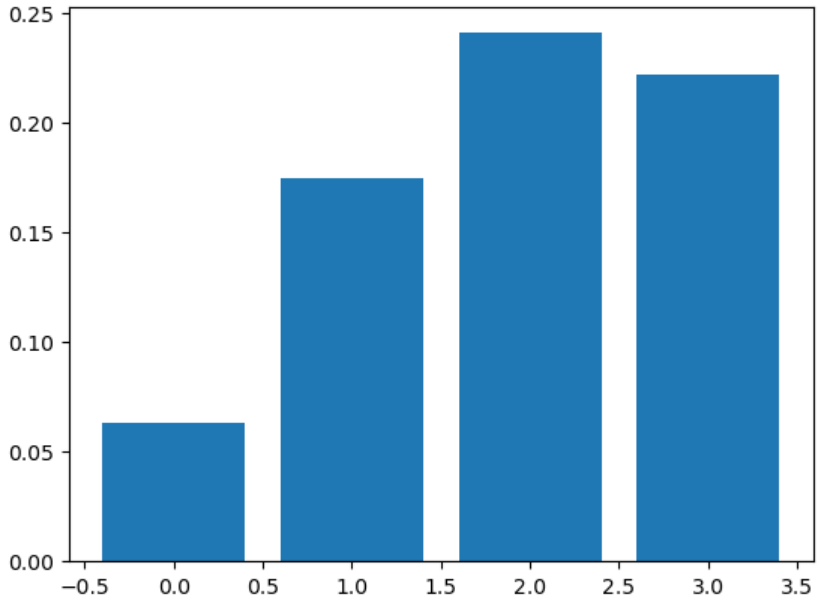
	0
count	15.000000
mean	2.762667
std	0.090512
min	2.610000
25%	2.690000
50%	2.750000
75%	2.840000
max	2.880000

In [44]:
```
from numpy import arange, mean
from scipy.stats import poisson
from matplotlib import pyplot as plt

F=array([2.86, 2.81, 2.88, 2.66, 2.75,
         2.80, 2.72, 2.74, 2.84, 2.64,
         2.88, 2.75, 6.85, 2.66, 2.84, 2.61])

F=F[F!=F.max()]

x = arange(0,max(F)+1)
P=poisson.pmf(x,mean(F))

plt.hist(F, bins=4)

plt.show()
```



Doesnt't really fit a Poisson Distribution

In [45]:
```
from numpy import arange, mean
from scipy.stats import poisson
from matplotlib import pyplot as plt

F=array([2.86, 2.81, 2.88, 2.66, 2.75,
         2.80, 2.72, 2.74, 2.84, 2.64,
         2.88, 2.75, 6.85, 2.66, 2.84, 2.61])

F=F[F!=F.max()]

x = arange(0,max(F)+1)
P=poisson.pmf(x,mean(F))

plt.bar(x,P)

plt.show()
```



But it does is we use a histogram instead?