Notebook

January 28, 2019

Local date & time is : 01/28/2019 19:27:49 PST

In [3]:
```
# Answer to Part a
a = (4276 + 1948) / 12195

# Answer to Part b
b = 1948 / (4276 + 1948)

# Answer to Part c
c = 1948 / (1948 + 2291)


a, b, c
```

Out[3]:
```
(0.5103731037310373, 0.3129820051413882, 0.4595423448926634)
```

Because 5 is such a small number compared to 12195, even if the draw is without replacement, it doens not strongly affect the overal probability of the next draw. Hence, the previous draw doesn't strongly influence the next draw, and we can say the draws are almost independent.

In [4]:
```
# Answer to Part e

#partition into all male and all female
#all five are male
male = ((3680 + 2291) / 12195)**5

#all five are female
female = ((4276 + 1948) / 12195)**5

#add together
male + female
```

Out[4]:
```
0.06276906109271552
```

In [7]:
```
# Answer to Part f

# P(all graduate degree | all same gender)
# all female
f = (1948 / (1948 + 4276))**5

# all male
m = (2291 / (2291 + 3680))**5

# P(all five same gender) - P(all graduate degree | all same gender)
(male + female) - (m + f)
```

Out[7]: 0.05145024107290824

In [8]:
```
# 10 choose 2
special.comb(100, 50)
```

Out[8]: 1.0089134454556415e+29

In [9]:
```
def chance_of_heads(n, k):
    """Returns the chance of k heads in n tosses of a fair coin"""
    n = float(n)
    return special.comb(n, k) / (2**n)
```

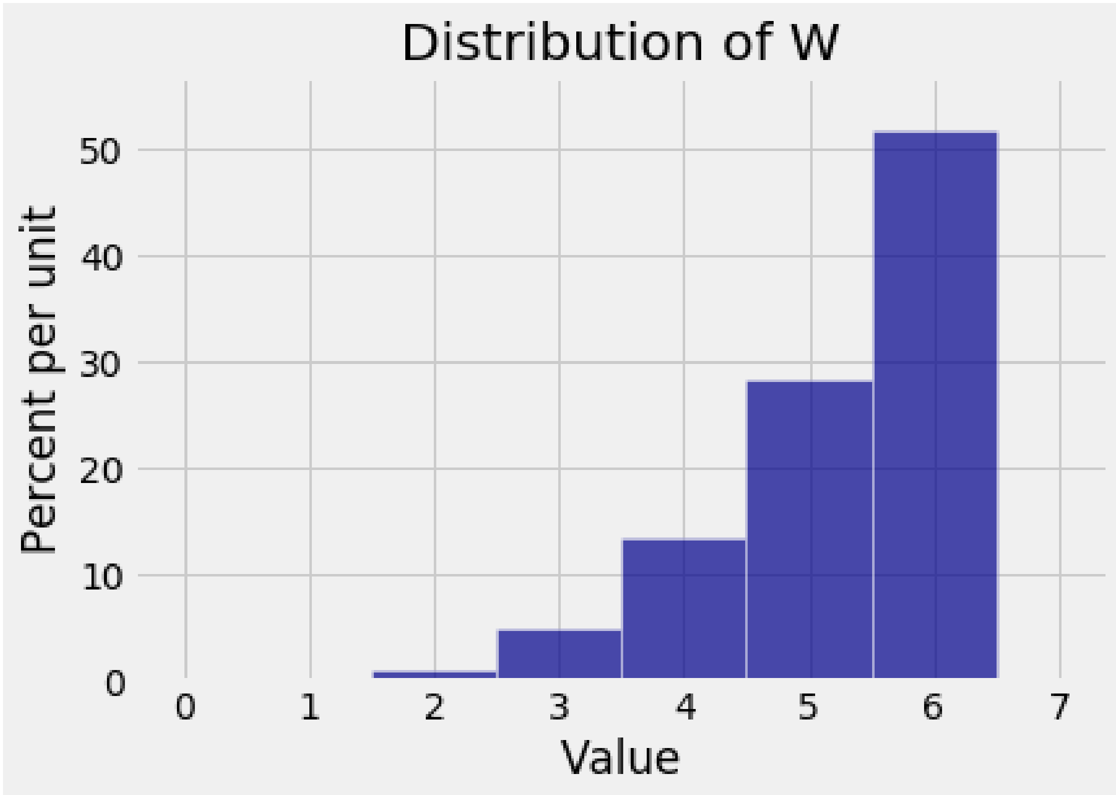
In [11]:
```
(1 - (chance_of_heads(20, 10)))**8
```

Out[11]: 0.21212249859944513

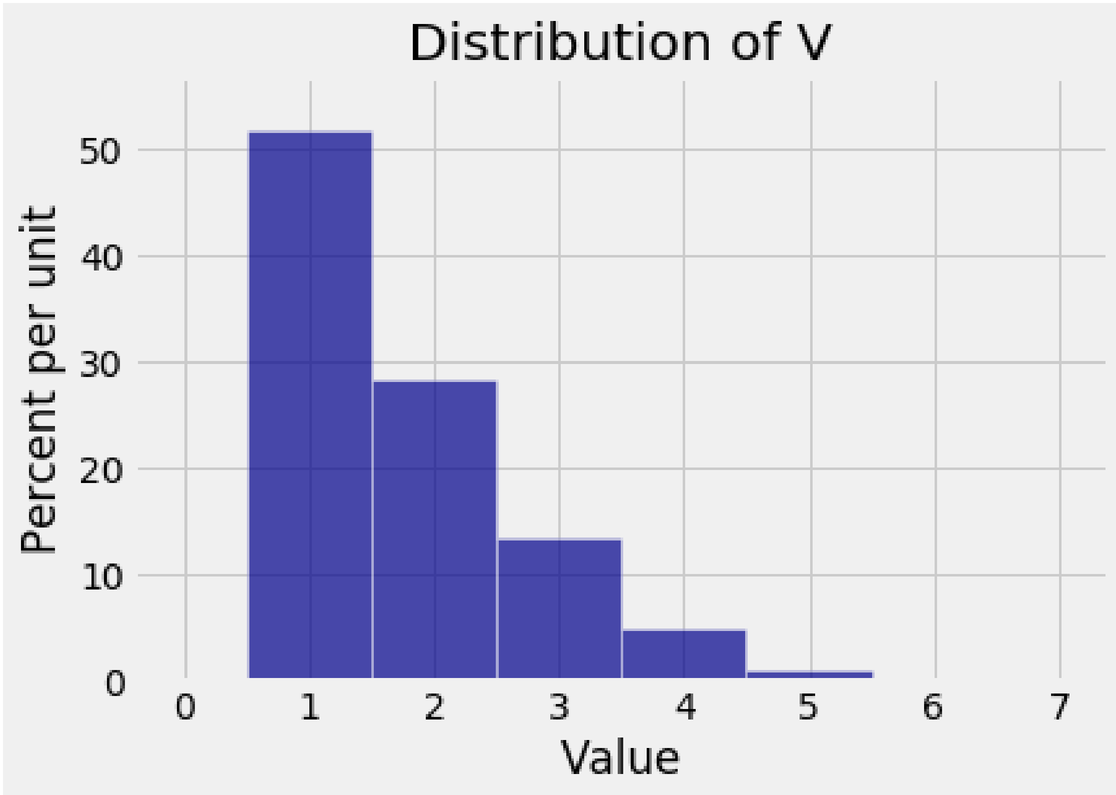
In [14]:
```
def exact_chance(m):
    """Returns P(m heads in 2m tosses)"""
    return chance_of_heads(2*m, m)

exact = chance_and_approx.apply(exact_chance, 'Heads' )         # array of exact chances
approx = 1 / ((np.pi * chance_and_approx[1])**(1/2))         # array of approximations

chance_and_approx = chance_and_approx.with_column('Exact Chance', exact,
                                                 'Approximation', approx)
chance_and_approx
```

Out[14]:
```
Tosses | Heads | Exact Chance | Approximation
50     | 25    | 0.112275     | 0.112838
100    | 50    | 0.0795892    | 0.0797885
... Omitting 2 lines ...
300    | 150   | 0.0460275    | 0.0460659
350    | 175   | 0.0426183    | 0.0426487
400    | 200   | 0.0398693    | 0.0398942
```

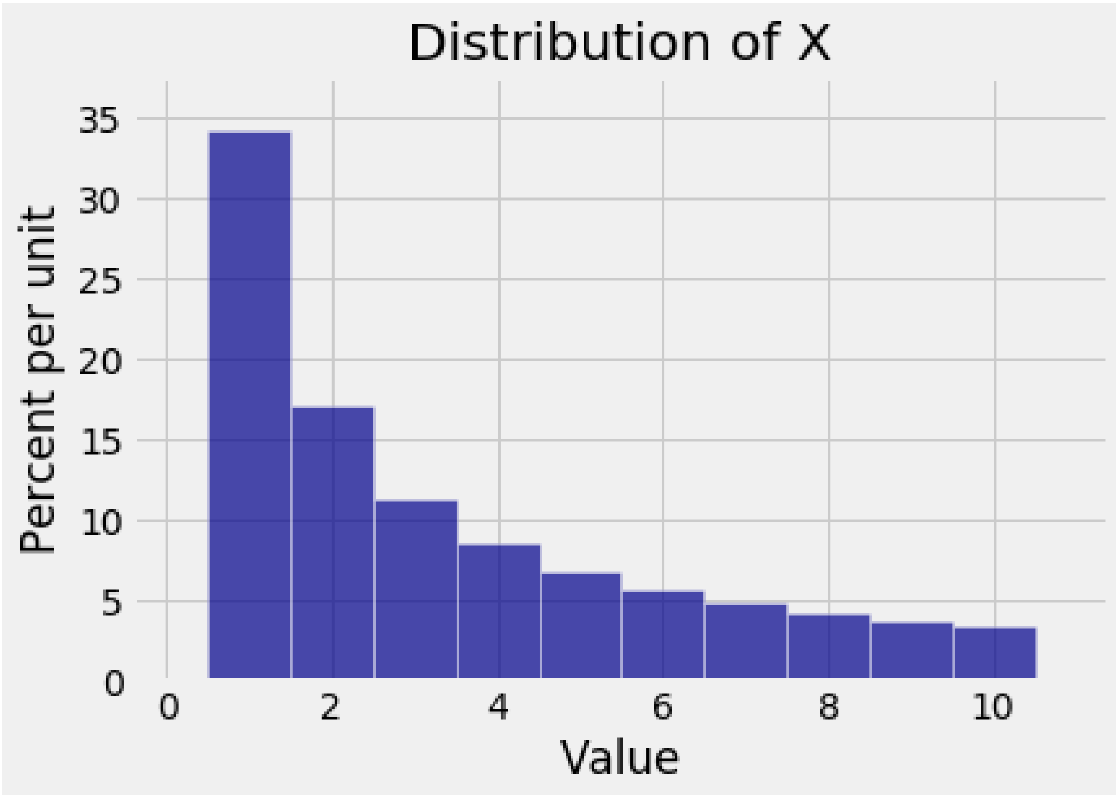
In [57]:
```
n = 4
N = 6
k = np.arange(1, N+1)

# array consisting of P(W=k)
probs_W = (k/N)**n - ((k-1) / N)**n

dist_W = Table().values(k).probabilities(probs_W)
Plot(dist_W)
plt.title('Distribution of W');
```




```
In [58]: probs_V = ((N - k + 1) / N)**n - ((N - k) / N)**n

         dist_V = Table().values(k).probabilities(probs_V)
         Plot(dist_V)
         plt.title('Distribution of V');
```




```
In [59]: k = np.arange(1, 11)    # array of possible values of R
         p = 1/(np.sum(1 / k)) * 1 / k                # array of probabilities of those values
         dist_R = Table().values(k).probabilities(p)
         Plot(dist_R)
         plt.title('Distribution of X');
```




```
In [60]: def joint_probability(x, y):
             """Returns P(R = x, S = y)"""
             return p.item(x - 1) * p.item(y - 1)

         joint_dist = Table().values('R', k, 'S', k).probability_function(joint_probability)
         joint_dist
```

```
Out[60]:            R=1       R=2       R=3       R=4       R=5       R=6       R=7  \
         S=10  0.011657  0.005828  0.003886  0.002914  0.002331  0.001943  0.001665
         S=9   0.012952  0.006476  0.004317  0.003238  0.002590  0.002159  0.001850
         ... Omitting 16 lines ...
         S=3   0.004857  0.004317  0.003886
         S=2   0.007285  0.006476  0.005828
         S=1   0.014571  0.012952  0.011657
```

```
In [61]: def indicator(i, j):
             return abs(i - j) == 2

         joint_dist.event(indicator, 'R', 'S')
```

```
P(Event) = 0.15024019918368425
```

```
Out[61]:              R=1          R=2          R=3          R=4          R=5          R=6  \
         S=10
         S=9
```

```
... Omitting 16 lines ...
S=3
S=2
S=1
```