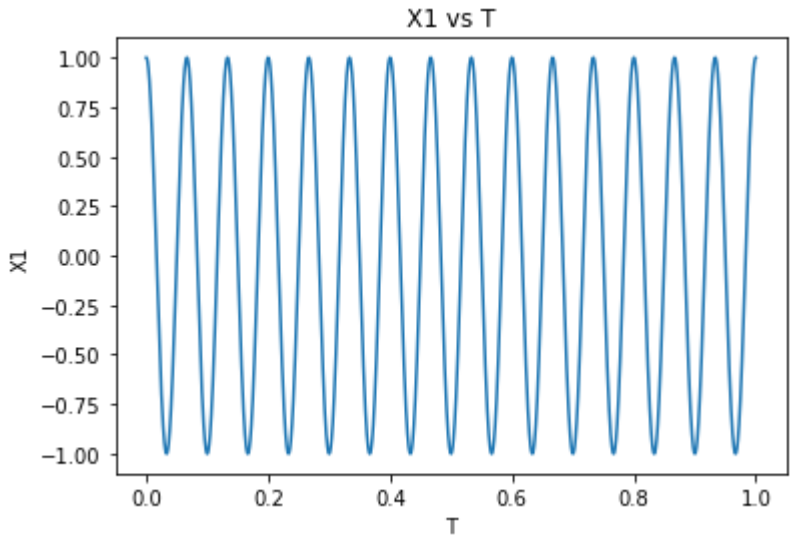
In [115]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from random import uniform
```

1.1 Cosine Wave

In [116]:
```
#Part A
def StudentCosine(T, f, A):
    x = A * np.cos(2 * np.pi * f * T)
    return x
```

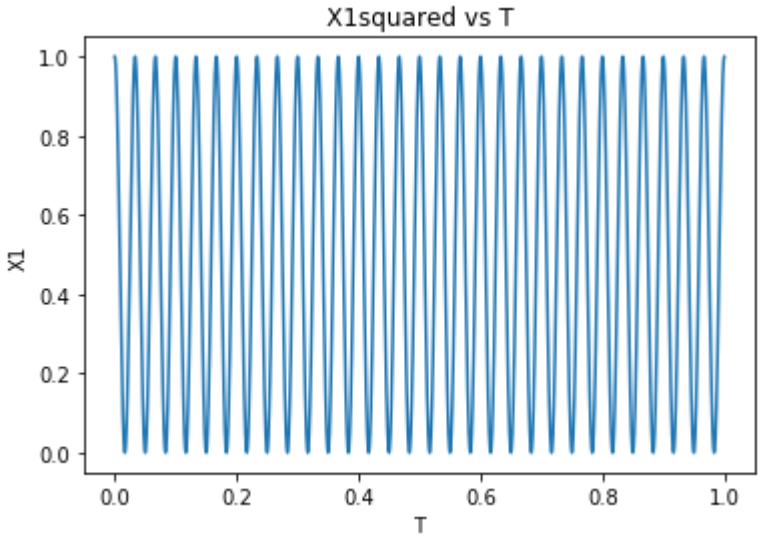
In [117]:
```
#Part B
T = np.arange(0, 1, .0001)
A = 1
f1 = 15
x1 = StudentCosine(T, f1, A)

#Part C
plt.plot(T, x1)
plt.ylabel("X1")
plt.xlabel("T")
plt.title("X1 vs T")
```

Out[117]: Text(0.5, 1.0, 'X1 vs T')



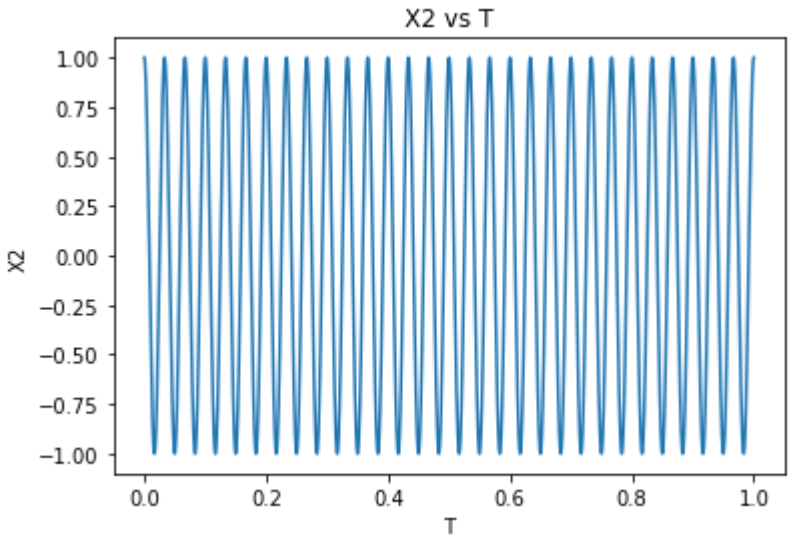
In [118]:
```
x1squared = x1**2;
plt.plot(T, x1squared)
plt.title("X1squared vs T")
plt.ylabel("X1")
plt.xlabel("T")
```

Out[118]: Text(0.5, 0, 'T')



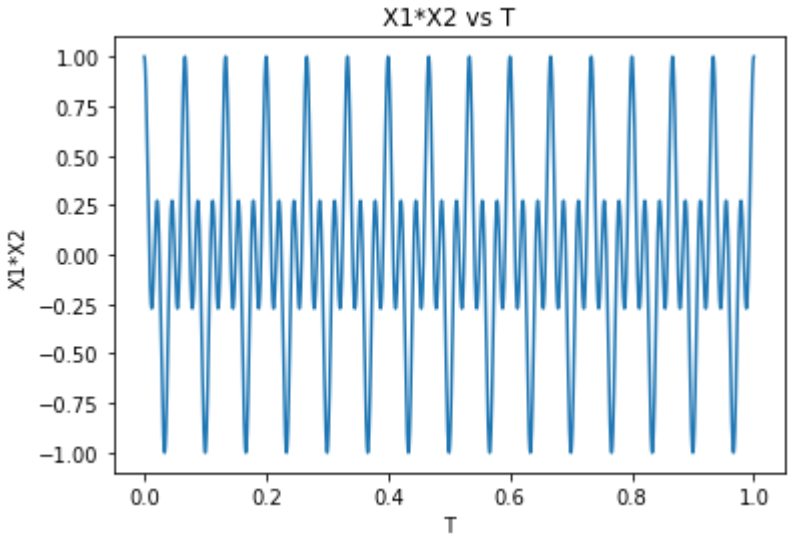
In [119]:
```
#Part D
f2 = 2 * f1
x2 = StudentCosine(T, f2, A)
plt.plot(T, x2)
plt.title("X2 vs T")
plt.ylabel("X2")
plt.xlabel("T")
```

Out[119]: Text(0.5, 0, 'T')



In [120]:
```
x3 = x1 * x2
plt.plot(T, x3)
plt.title("X1*X2 vs T")
plt.ylabel("X1*X2")
plt.xlabel("T")
```

Out[120]: Text(0.5, 0, 'T')



1.2 Matrix Operations

In [121]:
```
#Part A
A = np.mat([[1, 0, 0],
            [0, 1, 0],
            [0, 0, 1]])
B = np.mat([[1, 2, 0],
            [0, 4, 2],
            [3, 0, 0]])
#Part B
x = np.linalg.det(A)
y = np.linalg.det(B)
print(x)
print(y)

#Part C
C = A + B
D = A * B
print(C)
print(D)

#Part D
E = np.linalg.inv(A)
F = np.linalg.inv(B)
print(E)
print(F)

#Part E
eigenvalues, eigenvectors = np.linalg.eig(B)
eigenvaluesdiagonal = np.diag(eigenvalues)
answer = eigenvectors * eigenvaluesdiagonal * np.linalg.inv(eigenvectors)
print(answer)

#Part F
symmetric = np.matrix([[1, 5, 7],
                       [5, 2, 8],
                       [7, 8, 3]])
symmetriceigenvalues, symmetriceigenvectors = np.linalg.eig(symmetric)
smtranspose = symmetriceigenvectors.transpose()
test1 = smtranspose * symmetriceigenvectors
test2 = symmetriceigenvectors * smtranspose
print(test1)
print(test2)
#both return the identity matrix

final1 = symmetriceigenvectors * np.diag(symmetriceigenvalues) * np.linalg.inv
(symmetriceigenvectors)
final2 = symmetriceigenvectors * np.diag(symmetriceigenvalues) * symmetriceige
nvectors.transpose()
print(final1)
print(final2)
#both return the exact original matrix
```

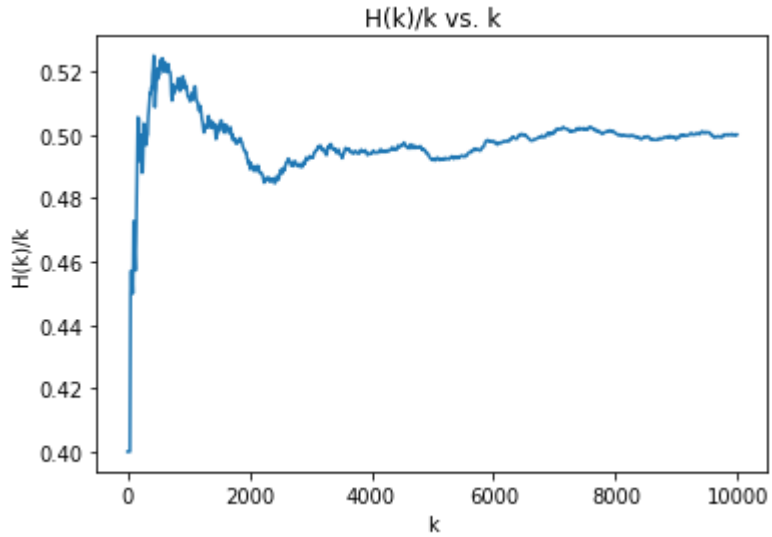
```
1.0
12.0
[[2 2 0]
 [0 5 2]
 [3 0 1]]
[[1 2 0]
 [0 4 2]
 [3 0 0]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[ 0.          0.          0.33333333]
 [ 0.5         0.         -0.16666667]
 [-1.          0.5         0.33333333]]
[[ 1.00000000e+00+3.47967103e-17j  2.00000000e+00+7.02263175e-18j
   4.16333634e-17-2.53749558e-17j]
 [-6.66133815e-16+1.21728781e-17j  4.00000000e+00+4.78327908e-18j
   2.00000000e+00-1.36222738e-16j]
 [ 3.00000000e+00-1.60514045e-17j -4.44089210e-16+3.43614541e-17j
   1.11022302e-16+1.24659540e-16j]]
[[ 1.00000000e+00 -6.59662022e-16  6.71667888e-17]
 [-6.59662022e-16  1.00000000e+00 -5.35612820e-16]
 [ 6.71667888e-17 -5.35612820e-16  1.00000000e+00]]
[[ 1.00000000e+00 -7.25594232e-17  1.13824201e-17]
 [-7.25594232e-17  1.00000000e+00  1.33638875e-17]
 [ 1.13824201e-17  1.33638875e-17  1.00000000e+00]]
[[1. 5. 7.]
 [5. 2. 8.]
 [7. 8. 3.]]
[[1. 5. 7.]
 [5. 2. 8.]
 [7. 8. 3.]]
```

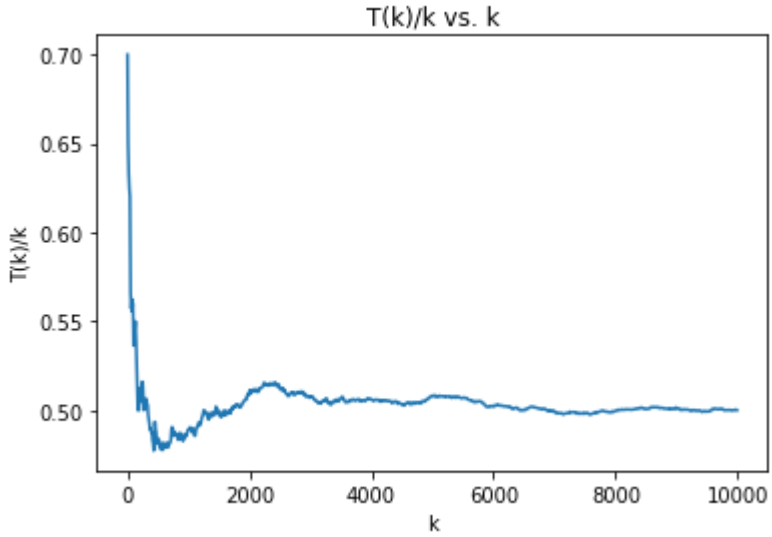
2.1 Fair Coin Toss

In [122]:
```
#Part A
L = []
def FairCoinToss():
    if uniform(0,1)<0.5: return "H"
    else: return "T"

for x in range(0, 10000):
    a = FairCoinToss()
    L.append(a)

#Part B
heads = 0
krange = np.arange(10, 10001, 10)
headtotal = []
for k in range (0, 10000):
    if L[k] == 'H':
        heads = heads + 1
    if k!= 0 and k % 10 == 0:
        percent = heads/k
        headtotal.append(percent)

if L[9999] == "H":
    heads = heads + 1
totalfraction = heads/10000
headtotal.append(totalfraction)

fig2 = plt.figure(2)
plt.subplot(1, 1, 1)
plt.plot(np.arange(0, 10000, 10), headtotal)

plt.title("H(k)/k vs. k")
plt.xlabel("k")
plt.ylabel("H(k)/k")
#Converges to 0.5

#Part C
for x in range(0, 10000):
    a = FairCoinToss()
    L.append(a)
#Part B
tails = 0
krange = np.arange(10, 10001, 10)
tailtotal = []
for k in range (0, 10000):
    if L[k] == 'T':
        tails = tails + 1
    if k!= 0 and k % 10 == 0:
        percent = tails/k
        tailtotal.append(percent)

if L[9999] == "T":
    tails = tails + 1
tailtotalfraction = tails/10000
tailtotal.append(tailtotalfraction)

fig3 = plt.figure(3)
```

```
plt.subplot(1, 1, 1)
plt.plot(np.arange(0, 10000, 10), tailtotal)

plt.title("T(k)/k vs. k")
plt.xlabel("k")
plt.ylabel("T(k)/k")

#this converges to 0.5 as well which is the probability of tails
```

Out[122]: Text(0, 0.5, 'T(k)/k')





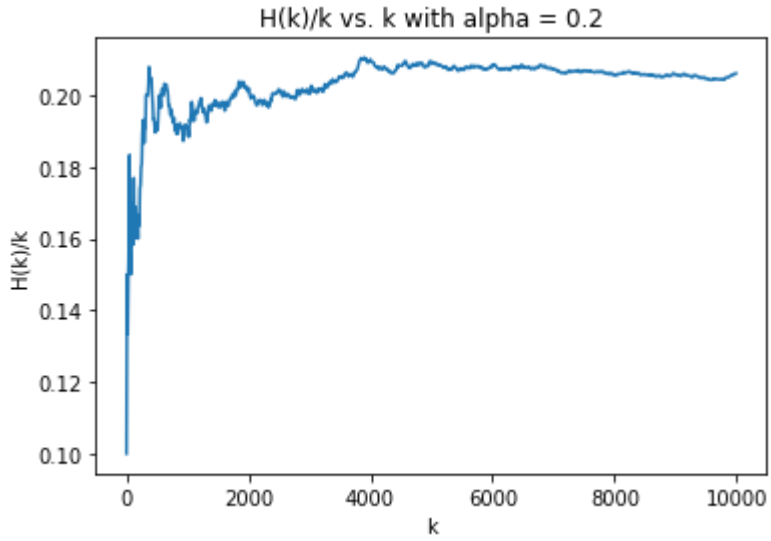
2.2 Biased Coin Toss

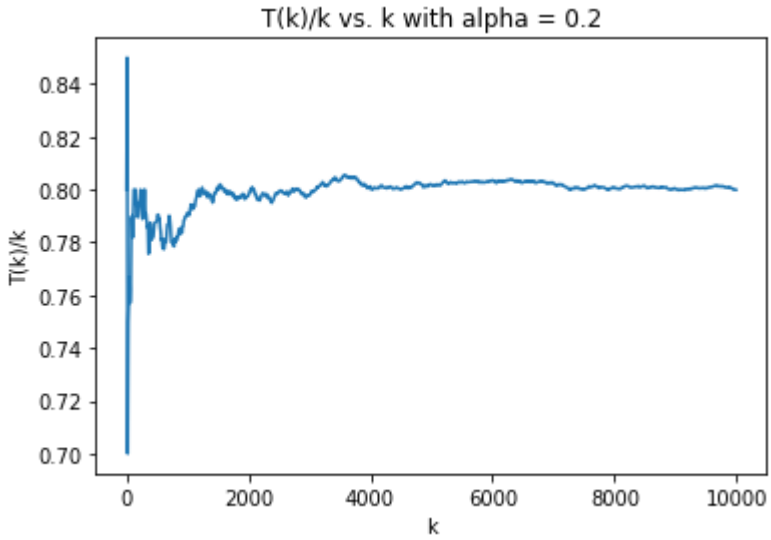
In [123]:
```
#Part A
def BiasedCoinToss(alpha):
    if uniform(0,1) < alpha: return "H"
    else: return "T"

#Part B
L = []
for x in range(0, 10000):
    a = BiasedCoinToss(0.2)
    L.append(a)
#Part B
heads = 0
krange = np.arange(10, 10001, 10)
headtotal = []
for k in range (0, 10000):
    if L[k] == 'H':
        heads = heads + 1
    if k!= 0 and k % 10 == 0:
        percent = heads/k
        headtotal.append(percent)

if L[9999] == "H":
    heads = heads + 1
totalfraction = heads/10000
headtotal.append(totalfraction)

fig4 = plt.figure(4)
plt.subplot(1, 1, 1)
plt.plot(np.arange(0, 10000, 10), headtotal)
plt.title("H(k)/k vs. k with alpha = 0.2")
plt.xlabel("k")
plt.ylabel("H(k)/k")
#this converges to 0.2 (alpha)

L = []
for x in range(0, 10000):
    a = BiasedCoinToss(0.2)
    L.append(a)
#Part B
tails = 0
krange = np.arange(10, 10001, 10)
tailtotal = []
for k in range (0, 10000):
    if L[k] == 'T':
        tails = tails + 1
    if k!= 0 and k % 10 == 0:
        percent = tails/k
        tailtotal.append(percent)

if L[9999] == "T":
    tails = tails + 1
tailtotalfraction = tails/10000
tailtotal.append(tailtotalfraction)

fig5 = plt.figure(5)
plt.subplot(1, 1, 1)
```

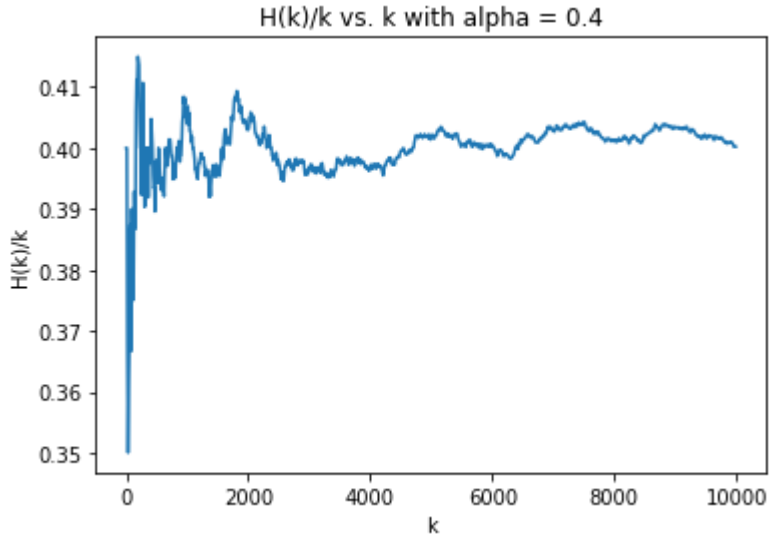
```
plt.plot(np.arange(0, 10000, 10), tailtotal)
plt.title("T(k)/k vs. k with alpha = 0.2")
plt.xlabel("k")
plt.ylabel("T(k)/k")
#this converges to 0.8 (1 - alpha)

#second number
L = []
for x in range(0, 10000):
    a = BiasedCoinToss(0.4)
    L.append(a)
#Part B
heads = 0
krange = np.arange(10, 10001, 10)
headtotal = []
for k in range (0, 10000):
    if L[k] == 'H':
        heads = heads + 1
    if k!= 0 and k % 10 == 0:
        percent = heads/k
        headtotal.append(percent)

if L[9999] == "H":
    heads = heads + 1
totalfraction = heads/10000
headtotal.append(totalfraction)

fig6 = plt.figure(6)
plt.subplot(1, 1, 1)
plt.plot(np.arange(0, 10000, 10), headtotal)
plt.title("H(k)/k vs. k with alpha = 0.4")
plt.xlabel("k")
plt.ylabel("H(k)/k")
#this converges to 0.4 (alpha)

L = []
for x in range(0, 10000):
    a = BiasedCoinToss(0.4)
    L.append(a)
#Part B
tails = 0
krange = np.arange(10, 10001, 10)
tailtotal = []
for k in range (0, 10000):
    if L[k] == 'T':
        tails = tails + 1
    if k!= 0 and k % 10 == 0:
        percent = tails/k
        tailtotal.append(percent)

if L[9999] == "T":
    tails = tails + 1
tailtotalfraction = tails/10000
tailtotal.append(tailtotalfraction)

fig7 = plt.figure(7)
plt.subplot(1, 1, 1)
```

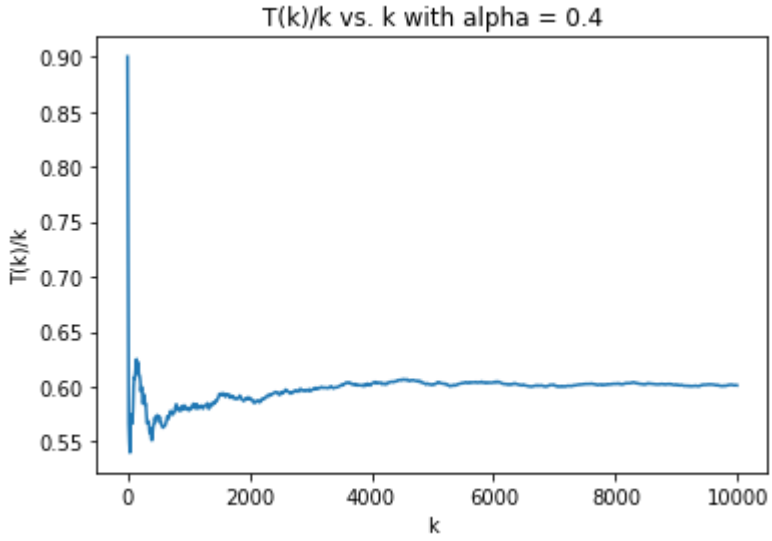
```
plt.plot(np.arange(0, 10000, 10), tailtotal)
plt.title("T(k)/k vs. k with alpha = 0.4")
plt.xlabel("k")
plt.ylabel("T(k)/k")
#this converges to 0.6 (1 - alpha)
```

Out[123]: Text(0, 0.5, 'T(k)/k')









2.3 Eight-Sided Die Rolling

In [2]:
```
#Part A
def EightSidedDieRolling():
    combinations = ['HHH', 'HHT', 'HTH', 'HTT', 'THH', 'THT', 'TTH', 'TTT']
    string = ""
    for x in range(0, 3):
        string = string + FairCoinToss()
    returner = 1
    for y in combinations:
        if string == y:
            return returner
        returner = returner + 1
#Part B
L = []
for x in range(0, 10000):
    a = EightSidedDieRolling()
    L.append(a)

#Part C
ones = 0
krange = np.arange(10, 10001, 10)
totalones = []
for k in range (0, 10000):
    if L[k] == 1:
        ones = ones + 1
    if k!= 0 and k %10 == 0:
        percent = ones/k
        totalones.append(percent)

if L[9999] == 1:
    ones = ones + 1
lastpercent = ones/10000
totalones.append(lastpercent)

fig9 = plt.figure(9)
plt.subplot(1, 1, 1)
plt.plot(np.arange(0, 10000, 10), totalones)

plt.title("1(k)/k vs. k")
plt.xlabel("k")
plt.ylabel("1(k)")
#1(k)/k converges to 1/8 which is 0.125

#Part D
L = []
for x in range(0, 10000):
    a = EightSidedDieRolling()
    L.append(a)

total = 0
krange = np.arange(10, 10001, 10)
sumsofar = []
for k in range (0, 10000):
    total = total + L[k]
    if k!= 0 and k % 10 == 0:
        average = total/k
        sumsofar.append(average)
```

```
lastvalue = L[9999]
total = total + lastvalue
lastsum = total/10000
sumsofar.append(lastsum)

fig10 = plt.figure(10)
plt.subplot(1, 1, 1)
plt.plot(np.arange(0, 10000, 10), sumsofar)

plt.title("average of k values vs. k")
plt.xlabel("k")
plt.ylabel("average of k values")
#this converges to the median between 1 and 8 which is 4.5

#Part E
attempts = []
for x in range (0, 100000):
    valuesleft = [1, 2, 3, 4, 5, 6, 7, 8]
    count = 0
    while len(valuesleft) > 0:
        a = EightSidedDieRolling()
        if (np.isin(a, valuesleft)):
            valuesleft.remove(a)
        count = count + 1
    attempts = np.append(attempts, count)

finalaverage = np.mean(attempts)
print(finalaverage)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-a9b2ca1d7375> in <module>
     13 L = []
     14 for x in range(0, 10000):
---> 15     a = EightSidedDieRolling()
     16     L.append(a)
     17

<ipython-input-2-a9b2ca1d7375> in EightSidedDieRolling()
      4     string = ""
      5     for x in range(0, 3):
----> 6         string = string + FairCoinToss()
      7     returner = 1
      8     for y in combinations:

NameError: name 'FairCoinToss' is not defined
```

In [ ]: