EECS 491 Assignment 1

William Nourse wrn13

In [1]:
```
from scipy import stats # probability functions
import matplotlib.pyplot as plt # plotting utility
import matplotlib.tri as tri
import math
import numpy as np # other math
# display graphs with nicer Jupyter formatting (not with 'Out[]'
%matplotlib inline
```

Q1. Basic probability (10 pts)

1.1. Prove (5 pts)

$$p(x, y|z) = p(x|z)p(y|x, z)$$

Given:

$$p(x|z) = \frac{p(x, z)}{p(z)}$$

$$p(y|x, z) = \frac{p(y, x, z)}{p(x, z)} = \frac{p(x, y, z)}{p(x, z)}$$

Then:

$$p(x|z)p(y|x, z) = \frac{p(x, z)}{p(z)} \frac{p(x, y, z)}{p(x, z)}$$

$$p(x|z)p(y|x, z) = \frac{p(x, y, z)}{p(z)}$$

$$p(x|z)p(y|x, z) = p(x, y|z)$$

1.2. Prove (5 pts)

$$p(x|y, z) = \frac{p(y|x, z)p(x|z)}{p(y|z)}$$

Given:

$$p(x|z) = \frac{p(x,z)}{p(z)}$$
$$p(y|z) = \frac{p(y,z)}{p(z)}$$
$$p(y|x,z) = \frac{p(y,x,z)}{p(x,z)} = \frac{p(x,y,z)}{p(x,z)}$$
$$p(x|y,z) = \frac{p(x,y,z)}{p(y,z)}$$

Then:

$$\frac{p(y|x,z)p(x|z)}{p(y|z)} = \frac{p(x,y,z)p(x,z)p(z)}{p(x,z)p(z)p(y,z)}$$
$$\frac{p(y|x,z)p(x|z)}{p(y|z)} = \frac{p(x,y,z)}{p(y,z)}$$
$$\frac{p(y|x,z)p(x|z)}{p(y|z)} = p(x|y,z)$$

Q2. Independence (10 pts)

2.1 Show that independence is not transitive, i.e. $a \perp b \wedge b \perp c \nRightarrow a \perp c$. Define a joint probability distribution $p(a,b,c)$ for which the previous expression holds and provide an interpretation. (5 pts)

Let $\mathcal{P}$ be the set of hospital ids for patients with pneumonia, a be the ids for males with pneumonia, c be the ids for females with pneumonia, and b be the ids for children with pneumonia.

$\mathcal{P} = \{1,2,3,4,5,6,7,8,9,10\}, a = \{1,3,5,7,9\}, b = \{5,6,7,8\}, c = \{2$

Then

$$p(a,b) = p(b,c) = \frac{2}{10}, p(a)p(b) = p(b)p(c) = \frac{5}{10} * \frac{4}{10} = \frac{20}{100} = \frac{2}{10}$$

So a and b are independent, and b and c are independent.

$$p(a,c) = 0, p(a)p(c) = \frac{5}{10} * \frac{5}{10} = \frac{1}{4} \neq 0$$

So a and c are not independent. Knowing whether an id corresponds to a child does not indicate whether that patient is male or female, however knowing if a patient is male also indicates if they are female.

2.2 Show that conditional independence does not imply marginal independence, i.e. $a \perp b|c \nRightarrow a \perp b$. Again provide an example. (5 pts)

We have two opaque candy dispensers, one with infinite red candies and one with infinite red candies and blue candies. Each dispenser gives one candy at a time. One of the dispensers is chosen at random. If a is the probability that the first candy given is red, and b is the probability that the second candy given is red, these probabilities are not independent since depending on the first candy we can infer which machine we are pulling from. However if we know which dispenser we are pulling from (c), then a and b are no longer independent.

Q3. Inspector Clouseau re-revisited (20 pts)

3.1 Write a program to evaluate $p(B|K)$ in Example 1.3 in Barber. Write your code and choose your data representations so that it is easy to use it to solve the remaining questions. Show that it correctly computes the value in the example. (5 pts)

In [2]:
```
# Murderer = True, not murderer = False
dist = [[0,0,0.3], # B = False, M = False
        [0,1,0.2],  # B = False, M = True
        [1,0,0.6],  # B = True, M = False
        [1,1,0.1]]  # B = True, M = True

def getVal(dist,b,m): # get the probability value in the distrib
    val = 0
    for item in dist:
        if (item[0] == b) and (item[1] == m):
            val = item[2]
    return val

def getTrue(dist,var):
    if var:
        b = 0
        m = 1
    else:
        b = 1
        m = 0
    val = getVal(dist,b,m)
    return val

def clouseau(dist,param): # get the probability that the desired
    if param:
        var = 1
        other = 0
    else:
        var = 0
        other = 1
    pVar = getTrue(dist,var) # prior probability that var is Tru
    pOther = getTrue(dist,other) # prior probability that other
    pBothTrue = dist[3][2]
    numerator = pVar*(pBothTrue*pOther + pVar*(1-pOther))
    denominator = dist[2][2]*(dist[3][2]*dist[1][2] + dist[2][2]
    denominator = denominator + (1-dist[2][2])*(dist[1][2]*dist[
    probVar = numerator/denominator
    return probVar

print(clouseau(dist,0))
```

```
0.7281553398058251
```

3.2 Define a different distribution for $p(K|M, B)$. Your new distribution should result in the outcome that $p(B|K)$ is either < 0.1 or > 0.9, i.e. reasonably strong evidence. Use the original values of $p(B)$ and $p(M)$ from the example. Provide (invent) a reasonable justification for the value of each entry in $p(K|M, B)$. (5 pts)

Processing math: 100%

In [3]:
```
distNew = [[0,0,0.05], # B = False, M = False
       [0,1,0.2],  # B = False, M = True
       [1,0,0.6],  # B = True, M = False
       [1,1,0.05]]  # B = True, M = Tru

print(clouseau(distNew,0))
```

```
0.9018404907975459
```

3.3 Derive the equation for $p(M|K)$. (5 pts)

Using b for the two states of B and m for the two states of M,

$$p(M|K) = \sum_b p(b, M|K)$$

$$p(M|K) = \sum_b \frac{p(b, M, K)}{p(K)}$$

$$p(M|K) = \frac{\sum_b p(K|b, M)p(b, M)}{\sum_{b,m} p(K|b, m), p(b, m)}$$

$$p(M|K) = \frac{p(M)\sum_b p(K|b, m)p(b)}{\sum_m p(m)\sum_b p(K|b, m)p(b)}$$

3.4 Calculate it's value for both the original $p(K|M, B)$ and the one you defined yourself. Is it possible to provide a summary of the main factors that contributed to the value? Why/Why not? Explain. (5 pts)

In [4]:
```
probMaidOld = clouseau(dist,1)
probMaidNew = clouseau(distNew,1)
print("Using Barber's distribution, the probability that the Mai
print("Using my distribution, the probability that the Maid was
```

```
Using Barber's distribution, the probability that the Maid was
the murderer is  0.06796116504854369
Using my distribution, the probability that the Maid was the mu
rderer is  0.06748466257668713
```

The main factor that contributes to this value is the original prior, as that reduces the magnitude of the whole expression and remained unchanged between distributions.

Q4. Biased views (20 pts)

4.1 Write a program that calculates the posterior distribution of the θ (probability of heads) from the Binomial distribution given y heads out of n trials. Feel free to use a package where the necessary distributions are defined as primitives. (5 pts)

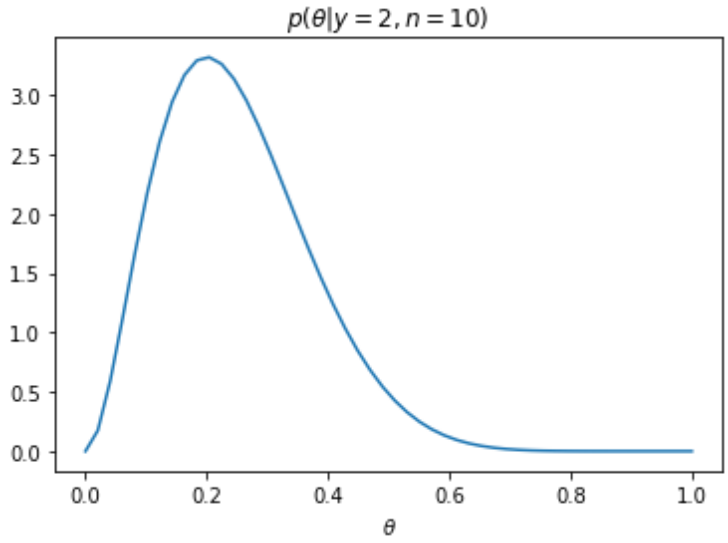
Using Bayes' rule,

$$p(\theta|y,n) = \frac{p(y|\theta,n)p(\theta|n)}{p(y|n)},$$

and if we also assume to have no information for the prior $p(\theta)$, then

$$p(\theta|y,n) \propto p(y|\theta,n) \cdot 1.$$

In [5]:
```
def calcBinomialPosterior(y,n,theta):
    # liklihood is the binomial distribution
    # we're using an uninformative prior p(theta)=1
    posterior = stats.binom.pmf(y,n,theta)*(n+1)

    return posterior

# Let's test it just to make sure the output looks reasonable
y = 2
n = 10
theta = np.linspace(0,1)

fig = plt.figure()
plt.plot(theta,calcBinomialPosterior(y,n,theta))
plt.title(r'$p(\theta|y=2,n=10)$')
plt.xlabel(r'$\theta$')
plt.show()
```



Processing math: 100%

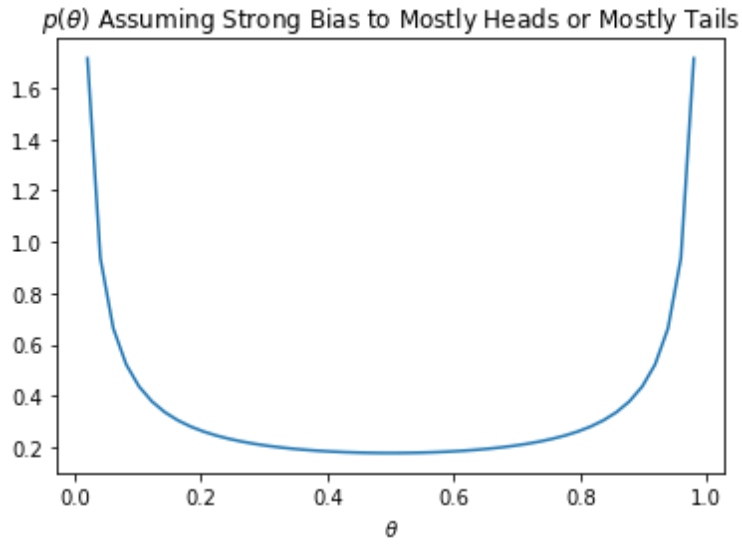
4.2 Imagine three different views on the coin bias:

- "I believe strongly that the coin is biased to either mostly heads or mostly tails."
- "I believe strongly that the coin is unbiased".
- "I don't know anything about the bias of the coin."

Define and plot prior distributions that expresses each of these beliefs. Provide a brief explanation. (5 pts)

If we assume that the coin is mostly biased to heads or tails, then the shape of $p(\theta)$ should have peaks at $\theta = 0$ and $\theta = 1$ with a steep valley in between. This is because we're assuming the bias is probably at either extreme, but it could be somewhere in the middle. For this and the later priors, I am using the β-distribution due to its varied expression using only two parameters.

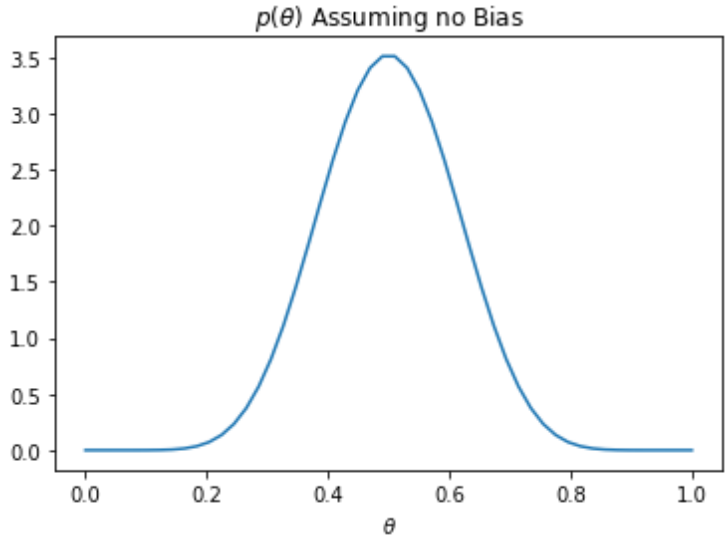
In [6]:
```
# biased to either mostly heads or mostly tails
fig1 = plt.figure()
beta1 = stats.beta(0.1,0.1)
plt.plot(theta,beta1.pdf(theta))
plt.title(r'$p(\theta)$ Assuming Strong Bias to Mostly Heads or M
plt.xlabel(r'$\theta$')
plt.show()
```



If we assume that the coin is unbiased, then that corresponds to a value of 0.5 for θ. There should still be some probability on either side of 0.5 because we aren't 100% sure, but the peak should be pretty steep.

Processing math: 100%

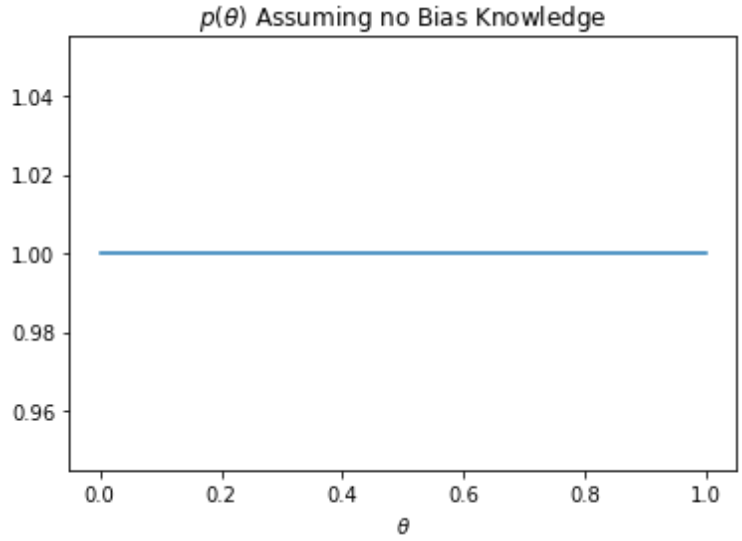
In [7]:
```
# no bias
fig2 = plt.figure()
beta2 = stats.beta(10,10)
plt.plot(theta,beta2.pdf(theta))
plt.xlabel(r'$\theta$')
plt.title(r'$p(\theta)$ Assuming no Bias')
plt.show()
```



If we have no prior knowledge, we have to weight all possible biases equally. The only way that this can be done is with a horizontal line.

Processing math: 100%

In [8]:
```
# no information
fig3 = plt.figure()
beta3 = stats.beta(1,1)
plt.plot(theta,beta3.pdf(theta))
plt.title(r'$p(\theta)$ Assuming no Bias Knowledge')
plt.xlabel(r'$\theta$')
plt.show()
```

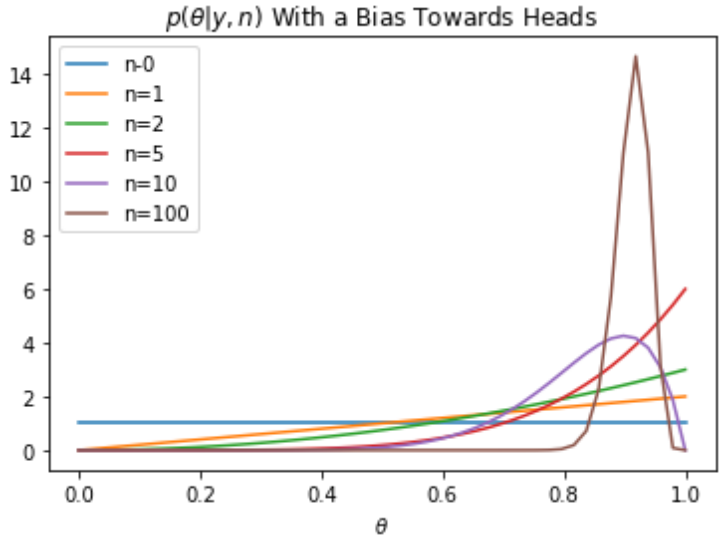


4.3 Perform Bernoulli trials where one of these views is correct. Show how the posterior distribution of θ changes for each view for n=0, 1, 2, 5, 10, and 100. Each view should have its own plot, but with the curves of the posterior after different numbers of trials overlayed. (5 pts)

In [9]:
```
# Stuff which will be useful
def flipCoins(theta,n):
    vals = stats.bernoulli.rvs(theta, size=n)
    y = np.count_nonzero(vals)

    return y

flips = [0,1,2,5,10,100]
```

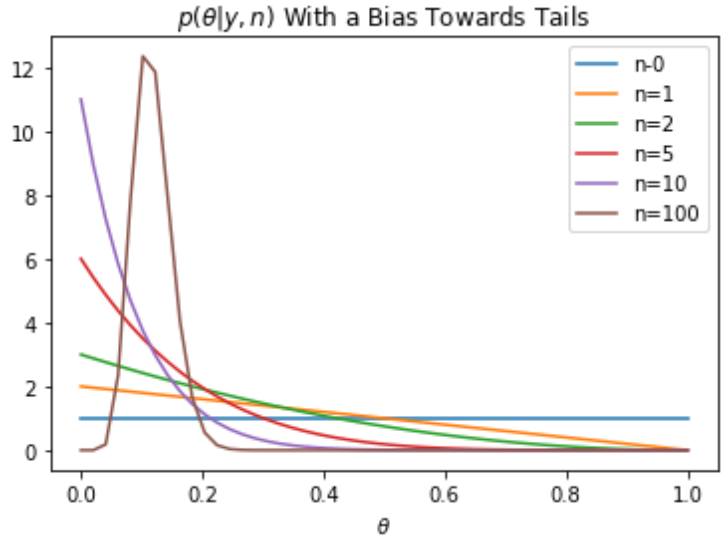
Processing math: 100%

In [10]:
```
# Heavily bias 'heads'
bias = 0.9

theta = np.linspace(0,1)
fig, ax = plt.subplots(1,1)
for i in flips:
    y = flipCoins(bias,i)
    plt.plot(theta,calcBinomialPosterior(y,i,theta))
    plt.title(r'$p(\theta|y,n)$ With a Bias Towards Heads')
plt.legend(['n-0','n=1','n=2','n=5','n=10','n=100'])
plt.xlabel(r'$\theta$')
plt.show()
```



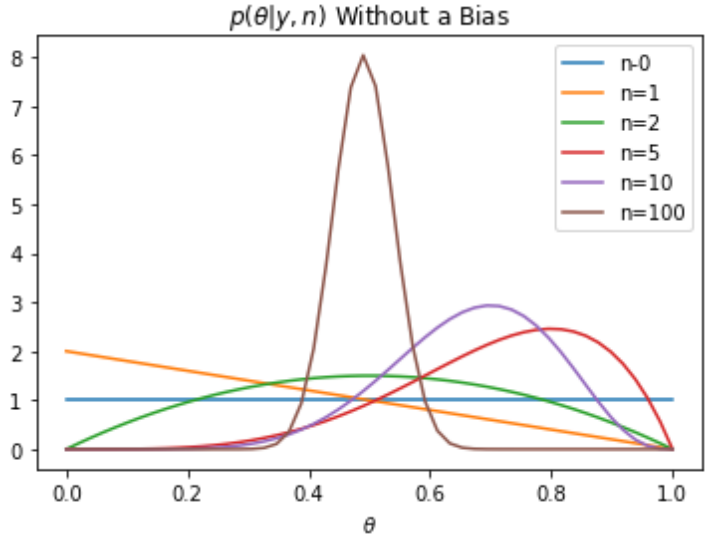
Processing math: 100%

In [11]:
```
# Heavily bias 'tails'
bias = 0.1

theta = np.linspace(0,1)
fig, ax = plt.subplots(1,1)
for i in flips:
    y = flipCoins(bias,i)
    plt.plot(theta,calcBinomialPosterior(y,i,theta))
    plt.title(r'$p(\theta|y,n)$ With a Bias Towards Tails')
plt.legend(['n-0','n=1','n=2','n=5','n=10','n=100'])
plt.xlabel(r'$\theta$')
plt.show()
```



Processing math: 100%

In [12]:
```
# Evenly biased
bias = 0.5

theta = np.linspace(0,1)
fig, ax = plt.subplots(1,1)
for i in flips:
    y = flipCoins(bias,i)
    plt.plot(theta,calcBinomialPosterior(y,i,theta))
    plt.title(r'$p(\theta|y,n)$ Without a Bias')
plt.legend(['n-0','n=1','n=2','n=5','n=10','n=100'])
plt.xlabel(r'$\theta$')
plt.show()
```



4.4 Is it possible that each view will always arrive at an accurate estimate of θ? How might you determine which view is most consistent with the data after n trials? (5 pts)

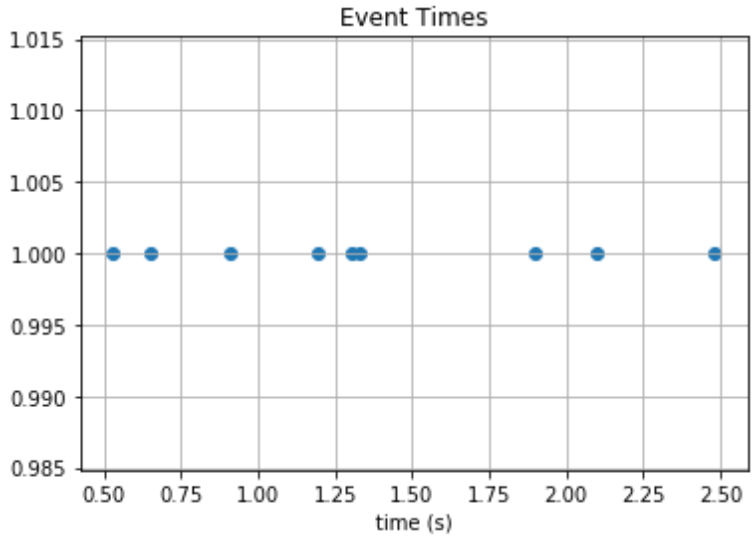
With enough samples each view should end up with an accurate estimate of θ. Of course the random values could be abnormal (all heads) depending on the seed, but we are assuming representative outputs. Also our posterior will take a lot longer to converge if the prior is less representative.

I think the way to confirm which is the most consistent is to take the mean-squared error between the "view" prior and the posterior, and whichever has the lowest error is probably the most consistent with the data.

Q5. Inference using the Poisson distribution (20 pts)

Suppose you observe for 3 seconds and detect a series of events that occur at the following times (in seconds): 0.53, 0.65, 0.91, 1.19, 1.30, 1.33, 1.90, 2.01, 2.48.

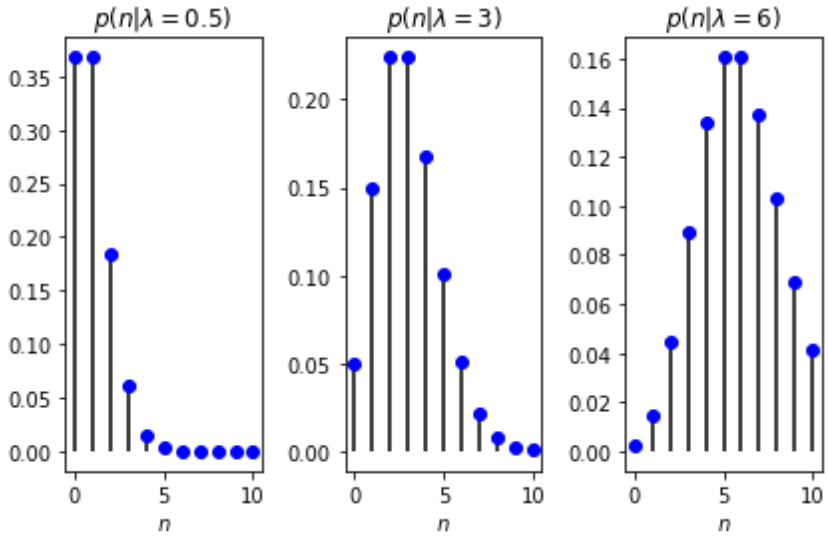
In [13]:
```
times = [0.53,0.65,0.91,1.19,1.3,1.33,1.9,2.10,2.48]
plt.figure()
plt.scatter(times,np.ones(len(times)))
plt.title('Event Times')
plt.xlabel('time (s)')
plt.grid()
```



5.1 Model the rate at which the events are produced using a Poisson distribution where λ is the number of events n observed per unit time (1 second). Show the likelihood equation and plot it for three different values of λ: less, about equal, and greater than what you estimate (intuitively) from the data. (5 pts)

$$p(n|\lambda) = \frac{\lambda^n e^{-\lambda}}{n!}$$

In [14]:
```
def poissonDist(n,rate):
    dist = stats.poisson.pmf(n,rate)
    return dist

# There are 9 events in 3 seconds, so lambda ~3 Hz intuitively
rate = [1,3,6]
n = [0,1,2,3,4,5,6,7,8,9,10]

plt.figure()
plt.subplot(1,3,1)
plt.plot(n,poissonDist(n,rate[0]),'bo')
plt.vlines(n,0,poissonDist(n,rate[0]))
plt.title(r'$p(n|\lambda = 0.5)$')
plt.xlabel(r'$n$')
plt.subplot(1,3,2)
plt.plot(n,poissonDist(n,rate[1]),'bo')
plt.vlines(n,0,poissonDist(n,rate[1]))
plt.title(r'$p(n|\lambda = 3)$')
plt.xlabel(r'$n$')
plt.subplot(1,3,3)
plt.plot(n,poissonDist(n,rate[2]),'bo')
plt.vlines(n,0,poissonDist(n,rate[2]))
plt.title(r'$p(n|\lambda = 6)$')
plt.xlabel(r'$n$')
plt.tight_layout()
```



5.2 Derive the posterior distribution of λ assuming a Gamma prior (usually defined with parameters α and β). The posterior should have the form $p(\lambda|n, T, \alpha, \beta)$ where T is the total duration of the observation period and n is the number of events observed within that period. (5 pts)

Processing math: 100%

Given:

$$p(n|\lambda, T) = \frac{(\lambda T)^n}{n!} e^{-\lambda T}$$

$$p(\lambda, T|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} (\lambda T)^{\alpha-1} e^{-\beta(\lambda T)}$$

Then:

$$p(\lambda|n, T, \alpha, \beta) = \frac{p(n|\lambda, T)p(\lambda, T|\alpha, \beta)}{p(n|T, \alpha, \beta)}$$

$$p(\lambda|n, T, \alpha, \beta) \propto (\lambda T)^n e^{-\lambda T} (\lambda T)^{\alpha-1} e^{-\beta\lambda T}$$

$$p(\lambda|n, T, \alpha, \beta) \propto (\lambda T)^{n+\alpha-1} e^{-(1+\beta)\lambda T}$$

5.3 Show that the Gamma distribution is a *conjugate prior* for the Poisson distribution, i.e. it is also a Gamma distribution, but defined by parameters α' and β' that are functions of the prior and likelihood parameters. (5 pts)

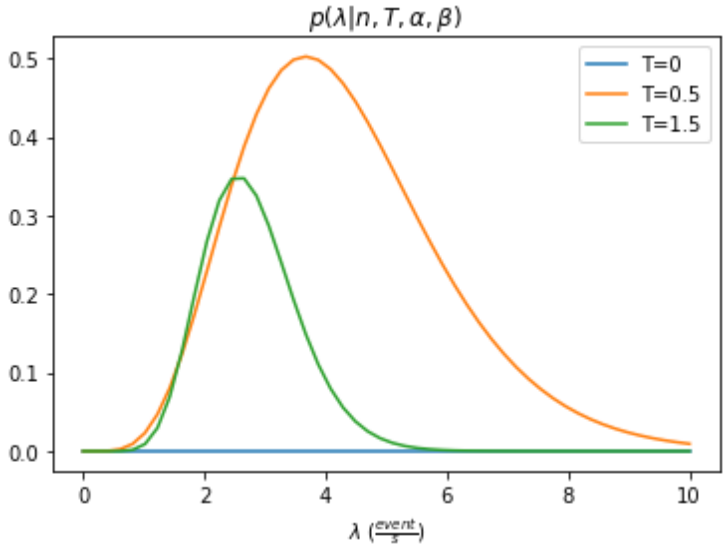
$$p(\lambda \mid n, T, \alpha, \beta) \propto (\lambda T)^{n+\alpha-1} e^{-(1+\beta)\lambda T}$$

$$X = \lambda T, \alpha' = n + \alpha, \beta' = \beta + 1$$

$$Gamma(\alpha', \beta') \propto X^{\alpha'-1} e^{-\beta' X}$$

5.4 Plot the posterior distribution for the data above at times T = 0, 0.5, and 1.5. Overlay the curves on a single plot. Comment how it is possible for your beliefs to change even though no new events have been observed. (5 pts)

In [15]:
```
def poissonPosterior(n,T,alpha,beta,lam):
    alphaP = n+alpha
    betaP = beta+1
    posterior = stats.gamma.pdf(lam*T,alphaP,scale=(1/betaP))
    return posterior

# These values were chosen because they generate a distribution
alpha = 6.5
beta = 2
T = [0,0.5,1.5]
nT = [0,0,6]
lam = np.linspace(0,10)
plt.figure()
plt.plot(lam,poissonPosterior(nT[0],T[0],alpha,beta,lam))
plt.plot(lam,poissonPosterior(nT[1],T[1],alpha,beta,lam))
plt.plot(lam,poissonPosterior(nT[2],T[2],alpha,beta,lam))
plt.legend(['T=0','T=0.5','T=1.5'])
plt.xlabel(r'$\lambda$ $(\frac{event}{s})$')
plt.title(r'$p(\lambda|n,T,\alpha,\beta)$')
plt
plt.show()
```



Our belief is able to update even without any new events because our posterior, $$p(\lambda|n,T,\alpha,\beta) \propto (\lambda T)^{n+\alpha-1}e^{-(1+\beta)\lambda T},$$ is dependent on both T and n. So even when n hasn't increased, T still changes the shape of the posterior distribution. This makes intuitive sense, because the fact that no new events have happened between two sample times should influence what the possible average rate of events.

Processing math: 100%

Q6. Exploration (40 pts)

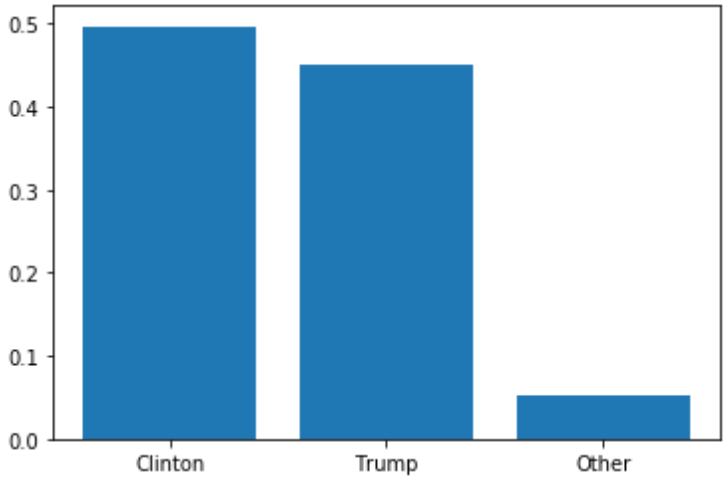
In these problems, you are meant to do creative exploration. Define and explore:

6.1 Polling in the 2016 Presidential Election (Discrete Inference) (20 pts)

A common concern expressed during election periods is the reliability of polls. Polls have a difficult job, in that they need to provide an accurate estimate of what a large population will choose, with out being able to sample a decent proportion of the voting population. As an example, in the 2016 election most poll-houses polled somewhere on the order of 1000 likely voters, which is less than 1% of the 137 million people that actually voted.

As fewer people respond to polls, their accuracy starts to decrease. Let's demonstrate this using the final poll data from the 2016 election (obtained from https://www.realclearpolitics.com/epolls/2016/president/us/general_election_trump_vs_clinton_vs_johnson_vs_stein-5952.html (https://www.realclearpolitics.com/epolls/2016/president/us/general_election_trump_vs_clinton_vs_johnson_vs_stein-5952.html)). First we'll model candidate likelihood using the multinomial distribution, which is a generalisation of the binomial distribution. For the sake of brevity, we will write this as follows, $$p(y_1,...,y_k) = Multinomial(p_1,...,p_k)$$ with y_k representing the k_th candidate, and p_k representing their chance of winning.

In [16]:
```
def multinomialSamples(numTrials):
    # True Probabilities from Polling Data
    truthCandidates = [0.482, # Clinton
                       0.461, # Trump
                       0.057] # Other

    multiVal = stats.multinomial.rvs(numTrials,truthCandidates)

    return multiVal

# Example to make sure truth model looks correct
trials = 600 # number of trials
x = [0,1,2]
multiVal = multinomialSamples(trials)
plt.figure()
plt.bar(x,multiVal)
plt.xticks(x,('Clinton','Trump','Other'))
plt.show()
```



With the our likelihood in the form of a Multinomial distribution, we'll choose the Dirichlet distribution as our prior, written for the sake of brevity as $$p(p_1,...,p_k) = Dirichlet(\alpha_1,...,\alpha_k).$$ We choose this prior because the Dirichlet distribution of distributions is a conjugate prior for the Multinomial distribution. Picking an uninformative value of $\alpha=1$ for all prior α's, our posterior becomes $$p(p_1,...,p_k|y_1,...,y_k) = Dirichlet(\alpha_1',...,\alpha_k')$$ $$\alpha_i' = \alpha_i + y_i = 1 + y_i.$$

Since the Dirichlet distribution is multidimensional, we need to project it in order to be visualize our posterior. Since we have three options (Clinton, Trump, or Other) we will project the distributions onto a triangular 2-simplex.

In [17]:
```
# NOTE: The Code contained in this cell isn't made by me. I need
# Distribution, and I couldn't find any standard python packages

# This code, which plots the Dirichlet on a 2-simplex, was creat
# http://blog.bogatron.net/blog/2014/02/02/visualizing-dirichlet

from functools import reduce

corners = np.array([[0, 0], [1, 0], [0.5, 0.75**0.5]])
triangle = tri.Triangulation(corners[:, 0], corners[:, 1])
# Mid-points of triangle sides opposite of each corner
midpoints = [(corners[(i + 1) % 3] + corners[(i + 2) % 3]) / 2.0
             for i in range(3)]
def xy2bc(xy, tol=1.e-3):
    '''Converts 2D Cartesian coordinates to barycentric.'''
    s = [(corners[i] - midpoints[i]).dot(xy - midpoints[i]) / 0.
         for i in range(3)]
    return np.clip(s, tol, 1.0 - tol)

def draw_dirichlet_contours(dist, nlevels=200, subdiv=8, **kwarg
    refiner = tri.UniformTriRefiner(triangle)
    trimesh = refiner.refine_triangulation(subdiv=subdiv)
    pvals = [dist.pdf(xy2bc(xy)) for xy in zip(trimesh.x, trimes

    plt.tricontourf(trimesh, pvals, nlevels,cmap='gray', **kwarg
    plt.axis('equal')
    plt.xlim(0, 1)
    plt.ylim(0, 0.75**0.5)
    plt.axis('off')

class Dirichlet(object):
    def __init__(self, alpha):
        from math import gamma
        from operator import mul
        self._alpha = np.array(alpha)
        self._coef = gamma(np.sum(self._alpha)) / \
                     reduce(mul, [gamma(a) for a in self._alpha]
    def pdf(self, x):
        '''Returns pdf value for `x`.'''
        from operator import mul
        return self._coef * reduce(mul, [xx ** (aa - 1)
                                         for (xx, aa)in zip(x, s

# Let's give an example plot
draw_dirichlet_contours(Dirichlet([5,5,5]))
plt.title('Distribution with equal likelihoods')
plt.show()
```

Processing math: 100%

In [18]:
```
# Let's plot the Dirichlet posterior over differing amounts of s

numTrials = [6,60,600]

multiVal0 = multinomialSamples(numTrials[0])
multiVal1 = multinomialSamples(numTrials[0])
multiVal2 = multinomialSamples(numTrials[0])
multiVal3 = multinomialSamples(numTrials[0])

plt.figure(figsize=(20,20))
plt.subplot(3,4,1)
draw_dirichlet_contours(Dirichlet(1+10*multiVal0))
plt.vlines(0.5,0,1,colors='r')
plt.title(r'n=6')
plt.subplot(3,4,2)
draw_dirichlet_contours(Dirichlet(1+10*multiVal1))
plt.vlines(0.5,0,1,colors='r')
plt.title(r'n=6')
plt.subplot(3,4,3)
draw_dirichlet_contours(Dirichlet(1+10*multiVal2))
plt.vlines(0.5,0,1,colors='r')
plt.title(r'n=6')
plt.subplot(3,4,4)
draw_dirichlet_contours(Dirichlet(1+10*multiVal3))
plt.vlines(0.5,0,1,colors='r')
plt.title(r'n=6')

multiVal0 = multinomialSamples(numTrials[1])
multiVal1 = multinomialSamples(numTrials[1])
multiVal2 = multinomialSamples(numTrials[1])
multiVal3 = multinomialSamples(numTrials[1])

plt.figure(figsize=(20,20))
plt.subplot(3,4,5)
draw_dirichlet_contours(Dirichlet(1+10*multiVal0))
plt.vlines(0.5,0,1,colors='r')
plt.title(r'n=60')
plt.subplot(3,4,6)
draw_dirichlet_contours(Dirichlet(1+10*multiVal1))
plt.vlines(0.5,0,1,colors='r')
plt.title(r'n=60')
plt.subplot(3,4,7)
draw_dirichlet_contours(Dirichlet(1+10*multiVal2))
plt.vlines(0.5,0,1,colors='r')
plt.title(r'n=60')
plt.subplot(3,4,8)
draw_dirichlet_contours(Dirichlet(1+10*multiVal3))
plt.vlines(0.5,0,1,colors='r')
plt.title(r'n=60')

multiVal0 = multinomialSamples(numTrials[2])
```

In the above series of plots, each row represents a different number of samples, and every individual posterior is based on a new random sample. The red line corresponds to the midline of the 2-simplex. If the distribution is centered to the left of the line it is weighted for Hillary Clinton, for Donald Trump if to the right, and the further up corresponds to the larger weighting of the third-party candidates.

As can be seen, with few samples the location of the posterior can vary wildly. As the number of samples increases, the posterior converges to a fairly stable location. However even with larger numbers of samples, the location does still vary. This demonstrates the problem with polls in general, and continues to be seen in the news coverage of elections. The voting body in the USA is very large, and it would take a prohibitively large number of samples to accurately characterize the actual outcome. This leads to the discrepancies between the various polling organizations.

6.2 The Kalman Estimator (Continuous Inference) (20 pts)

One of the key issues in robotics is that of state estimation. Robots observe the world around them sensors, and while ideally these are extremely precise and accurate, the reality is that sensors always have some level of inaccuracy and noise in their measurements. One standard way to incorporate sensors and refine our belief of their accuracy over time is to use the Kalman filter, which estimates the joint probability distribution of the real-world quantity over time.

We are going to assume that all noise is gaussian. Assuming all forms of sensor noise are gaussian makes the Kalman filter straightforward to implement. This is because the conjugate prior of a gaussian likelihood is a new gaussian, with its mean and standard deviation transformed. Specifically, when we perform a bayesian update our new mean μ and squared standard deviation σ^2 will be

$$\mu' = \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2+\sigma_2^2}$$
$$\sigma^{2'} = \frac{1}{\frac{1}{\sigma_1^2}+\frac{1}{\sigma_2^2}}.$$

We will also need to make a prediction of our next state. This prediction will become our prior at the next sensory update, and is formed as follows:

$$\mu' = \mu_1 + \mu_2$$ $$\sigma^{2'} = \sigma_1^2 + \sigma_2^2.$$

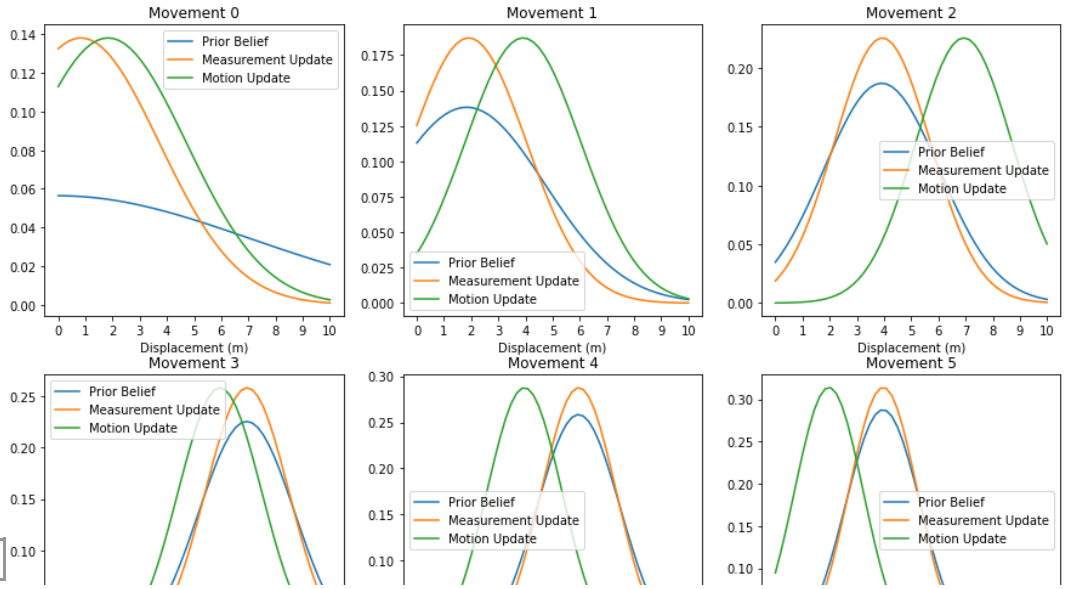
In the example presented here, we're going to look at a robot moving along a straight line. We have two sensors observing the world, one measuring global displacement in meters, and the other measuring velocity in meters per second. New information comes in at a rate of 1 Hz, and at every sample time we will take our prior belief, update our prior based on the current position measurement, and predict our next position based on our current velocity and a motion model.

In [48]:
```
# function to simplify calling scipy with our squared standard d
def gauss(mu, stdSq, x):
    std = np.sqrt(stdSq)
    val = stats.norm.pdf(x,loc=mu,scale=std)
    return val
```

In [20]:
```
# measurement-based update
def update(mu1, stdSq1, mu2, stdSq2):
    mu_prime = (mu1*stdSq2 + mu2*stdSq1)/(stdSq1+stdSq2)
    stdSq_prime = 1/(1/stdSq1 + 1/stdSq2)
    return [mu_prime, stdSq_prime]
```

In [21]:
```
def predict(mu1, stdSq1, mu2, stdSq2):
    mu_predict = mu1 + mu2
    stdSq_predict = stdSq1 + stdSq2

    return [mu_predict, stdSq_predict]
```

In [47]:
```
# measured locations and velocities
measurements = [1,2,4,7,6,4,2,1,0]
motions = [1,2,3,-1,-2,-2,-1,-1,1]

# initial parameters
measurement_stdSq = 10
motion_stdSq = 1
mu = 0
stdSq = 50

x_axis = np.linspace(0,10)
plt.figure(figsize=(15,15))

for i in range(len(measurements)-1):
    plt.subplot(3,3,i+1)
    plt.plot(x_axis,gauss(mu,stdSq,x_axis))
    # measurement update, with uncertainty
    mu, stdSq = update(mu, stdSq, measurements[i], measurement_s
    plt.plot(x_axis,gauss(mu,stdSq,x_axis))
    # motion update, with uncertainty
    mu, sig = predict(mu, sig, motions[i], motion_std)
    plt.plot(x_axis,gauss(mu,stdSq,x_axis))
    plt.legend(['Prior Belief','Measurement Update','Motion Upda
    plt.xticks([0,1,2,3,4,5,6,7,8,9,10])
    plt.xlabel('Displacement (m)')
    plt.title('Movement %i'%i)

plt.subplot(3,3,9)
plt.plot(x_axis,gauss(mu,stdSq,x_axis))
plt.xticks([0,1,2,3,4,5,6,7,8,9,10])
plt.title('Final Location')
plt.vlines(mu,0,0.35)

plt.show()
```



Processing math: 100%

The above series of plots shows our belief changing over time. When we start we know relatively little, and by the final update our belief has converged to be confident in our final state, as shown by the vertical line at zero.