In [1]:
```
using PyPlot
using LinearAlgebra
using Statistics
using Measurements
```

```
ArgumentError: Package Measurements not found in current path:
- Run `import Pkg; Pkg.add("Measurements")` to install the Measuremen
ts package.


Stacktrace:
 [1] require(::Module, ::Symbol) at ./loading.jl:892
 [2] top-level scope at In[1]:4
```

A few normalization methods

In [2]:
```
normalize_p_pmf(py)  = normalize_p_pmf!(copy(py))

normalize_p_pmf!(py) = normalize!(py, 1)

normalize_p_pdf(py, y) = normalize_p_pdf!(copy(py), y)

function normalize_p_pdf!(py::Vector{T}, y) where T<:Real
    npy = length(py)
    Δy  = diff(collect(y))
    @assert npy == length(y)
    @assert all(py .>= 0)
    @assert all(Δy .>= 0)
    c = zero(T)
    for i in 1:npy-1
        mn, mx = extrema((py[i],py[i+1]))
        c += (mn + (mx - mn)/2) * Δy[i]
    end
    py ./= c
    return py
end


normalize_p_max(py) = normalize_p_max!(copy(py))

function normalize_p_max!(py)
    mx   = maximum(py)
    py ./= mx
    return py
end
```

Out[2]:
```
normalize_p_max! (generic function with 1 method)
```

Binomial/Beta simulation, density

In [3]:
```
binomial_loglike(y,n,θ) = y*log(θ) + (n-y)*log(1-θ)

bernoulli_rand(θ, sz...) = rand(sz...) .< θ

binomial_rand(n,θ) = sum(bernoulli_rand(θ,n))

function binomial_rand(n,θ, sz...)
    rtn = fill(-1,sz)
    for i in eachindex(rtn)
        rtn[i] = binomial_rand(n,θ)
    end
    rtn
end

beta_logdensity(θ, φ1, φ2) = (φ1-1) * log(θ) + (φ2-1) * log(1-θ)

function beta_rand(α::Int,β::Int)
    X = sum(randn(2α).^2)
    Y = sum(randn(2β).^2)
    X/(X+Y)
end

function beta_rand(α::Int,β::Int, sz...)
    rtn = fill(-1.0,sz)
    for i in eachindex(rtn)
        rtn[i] = beta_rand(α::Int,β::Int)
    end
    rtn
end
```

Out[3]:
```
beta_rand (generic function with 2 methods)
```

Some examples of the above methods

In [4]:
```
n, θ, y = 100, 0.4, 5
α, β = 2, 4
@show binomial_loglike(y, n, θ)
@show beta_logdensity(θ,α,β)
@show bernoulli_rand(θ)
@show binomial_rand(n, θ)
@show beta_rand(α,β);
```

```
binomial_loglike(y, n, θ) = -53.10988791713989
beta_logdensity(θ, α, β) = -2.448767603172127
bernoulli_rand(θ) = false
binomial_rand(n, θ) = 36
beta_rand(α, β) = 0.49916881215009856
```

Here is how to use the extensions which fill an array of specified size with iid simulations

In [5]:
```
bernoulli_rand(θ, 5, 7)
```

Out[5]:
```
5×7 BitArray{2}:
 1  0  1  0  0  0  0
 1  0  1  1  1  0  1
 0  0  0  0  0  0  0
 1  1  1  0  1  0  1
 0  0  0  0  1  1  1
```

In [6]:
```
binomial_rand(n, θ, 5,7)
```

Out[6]:
```
5×7 Array{Int64,2}:
 48  43  40  29  45  34  43
 46  37  29  39  38  43  44
 38  39  38  30  43  41  35
 41  36  38  30  42  40  43
 44  38  48  34  41  54  30
```

In [7]:
```
beta_rand(α, β, 5, 7)
```

Out[7]:
```
5×7 Array{Float64,2}:
 0.225531  0.298874  0.388462   0.140666  0.460167  0.140939  0.46632
6
 0.254171  0.424164  0.212461   0.399877  0.192179  0.20534   0.10362
5
 0.462627  0.625682  0.272553   0.433069  0.542355  0.389744  0.10934
4
 0.444699  0.107508  0.448424   0.573618  0.635168  0.116177  0.34823
9
 0.201883  0.129178  0.0526285  0.588435  0.252433  0.2035    0.50891
9
```

Quiz (for 05-21-2020)

Question 1:

Find $E(1/(x+1))$ where $x \sim Beta(4, 5)$.

In [8]:
```
#📌####### code up the solution here (note: you can use simulations)
alpha, beta = 4, 5
N = Int(1e6)
samples = beta_rand(alpha, beta, N)
mean(1 ./ (samples .+ 1))
```

Out[8]:
```
0.7005915044233021
```

Question 2:

Find $P(e^x > 3/2)$ where $x \sim Beta(4, 5)$.

In [9]:
```
#📌####### code up the solution here (note: you can use simulations)
alpha, beta = 4, 5
N = Int(1e6)
samples = beta_rand(alpha, beta, N)
 mean(exp.(samples) .> 3/2)
```

Out[9]:
```
0.580636
```

Question 3:

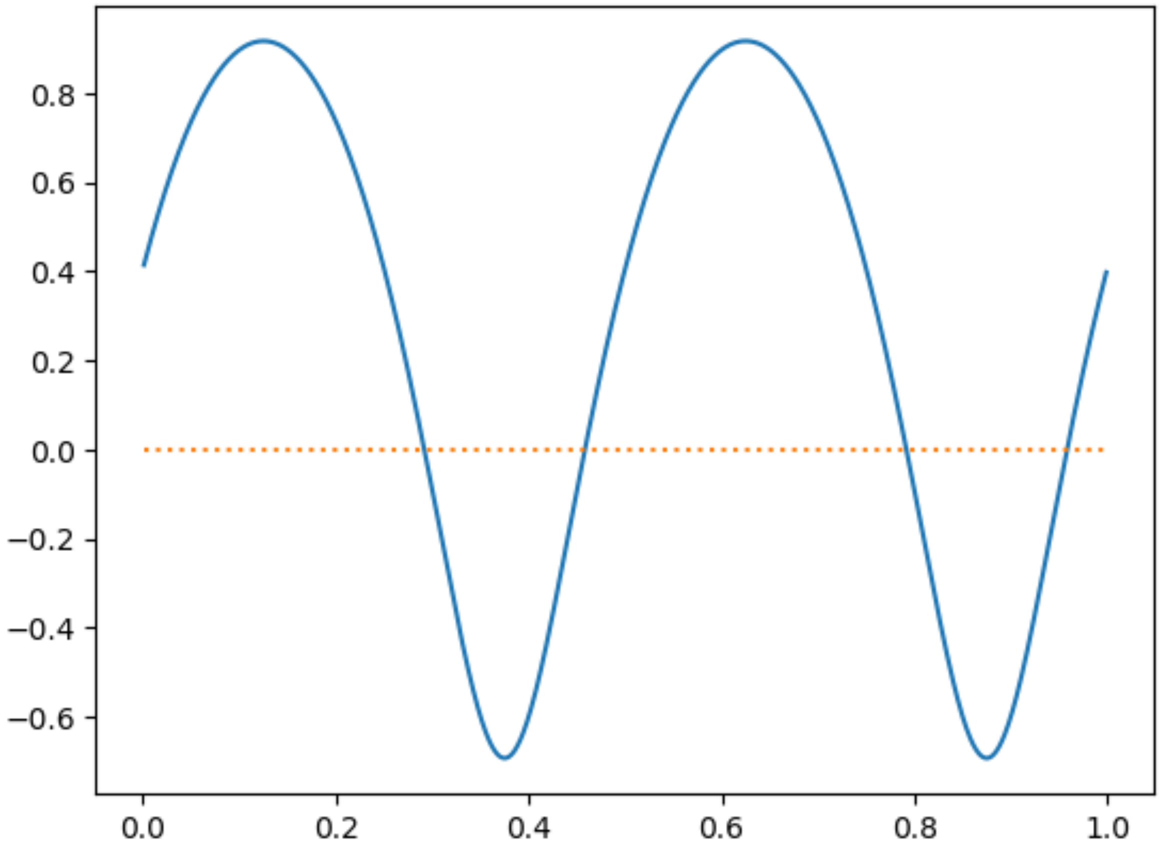
Suppose $y|\theta \sim Binomial(n = 10, \theta)$ and θ has prior log density given as follows (up to an additive constant)

In [10]:
```
quiz_logprior(θ) =  log(1.5 + sin(4π*θ))
```

Out[10]:
```
quiz_logprior (generic function with 1 method)
```

Here is a plot of `quiz_logprior(θ)` on a grid of θ values

In [11]:
```
quiz_θgrid = range(0,1,length=1000)[2:end-1]
plot(quiz_θgrid, quiz_logprior.(quiz_θgrid))
plot(quiz_θgrid, quiz_θgrid .* 0, ":")
```



Out[11]:
```
1-element Array{PyCall.PyObject,1}:
 PyObject <matplotlib.lines.Line2D object at 0x144b26510>
```

Now given an observation $y = 8$ find the posterior expected value of θ, i.e. find $E(\theta|y)$.

In [12]:
```
#📌####### code up the solution here (note: you can use simulations)
log_postrior(theta) = quiz_logprior(theta) + binomial_loglike(8, 10, t
N = Int(1e6)
thetas = range(0, 1, length=N)
unnormalized_pdf = exp.(log_postrior.(thetas))
sum(unnormalized_pdf .* thetas) ./ sum(unnormalized_pdf);
```

Homework (due 05-27-2020)

This problem explores using a Beta prior for Binomial data.

The data for this exercise are the following win/losses for your favoriate team over the course of three seasons.

In [13]:
```
season1_wins_losses = (5,2)
season2_wins_losses = (7,2)
season3_wins_losses = (10,3)
```

Out[13]:
```
(10, 3)
```

So, e.g., in season 2 the team won 7 games and lost 2.

Suppose your team wins each game with probability $\theta \in (0, 1)$, fixed over all seasons. Also suppose the games are independent so the number of wins is $Binomial(n, \theta)$ where n is the number of games played. No one knows the true value of θ but we are going to use Bayes and the historical wins/losses to quantify likely θ values (with a posterior distribution on θ). Later in the notebook you will use this posterior distribution to investigate bets on your favoriate team in season 4 when Vegas posts odds for your team to win games in season 4.

Remark: As your working this homework be sure to notice how natural the updating rules (from prior to posterior) are for the Beta hyper-parameters as one iteratively collects Binomial data.

Lets start by setting the prior parameters for $p(\theta) = Beta(\theta|1, 1)$

In [14]:
```
prior_betaφ    = (1,1)
```

Out[14]:
```
(1, 1)
```

We can simulate from $p(\theta)$ to get an idea of the ensemble of likely θ values quantified by the prior.

In [15]:
```
some_possibleθ = beta_rand(prior_betaφ[1], prior_betaφ[2], 5,5)
```

Out[15]:
```
5×5 Array{Float64,2}:
 0.319073   0.824605   0.989302  0.855969  0.299627
 0.255125   0.0942245  0.189526  0.333576  0.845661
 0.246579   0.347732   0.81091   0.3759    0.493388
 0.0539639  0.371759   0.661396  0.3067    0.103425
 0.63206    0.493589   0.286387  0.872071  0.900583
```

Lets convert these θ values from win probabilities to odds (rounded to be out of 100)

In [16]:
```
map(θ -> "$(round(Int,100*θ)):$(round(Int,100*(1-θ)))", some_possibleθ
```

Out[16]:
```
5×5 Array{String,2}:
 "32:68"  "82:18"  "99:1"   "86:14"  "30:70"
 "26:74"  "9:91"   "19:81"  "33:67"  "85:15"
 "25:75"  "35:65"  "81:19"  "38:62"  "49:51"
 "5:95"   "37:63"  "66:34"  "31:69"  "10:90"
 "63:37"  "49:51"  "29:71"  "87:13"  "90:10"
```

Using the fact that the Binomial likelihood function and the Beta density are conjugate we just need to update the Beta parameters for the posteior distributions after eash season

In [17]:
```
season1_betaφ   = prior_betaφ    .+ season1_wins_losses
season12_betaφ  = season1_betaφ  .+ season2_wins_losses
season123_betaφ = season12_betaφ .+ season3_wins_losses
```

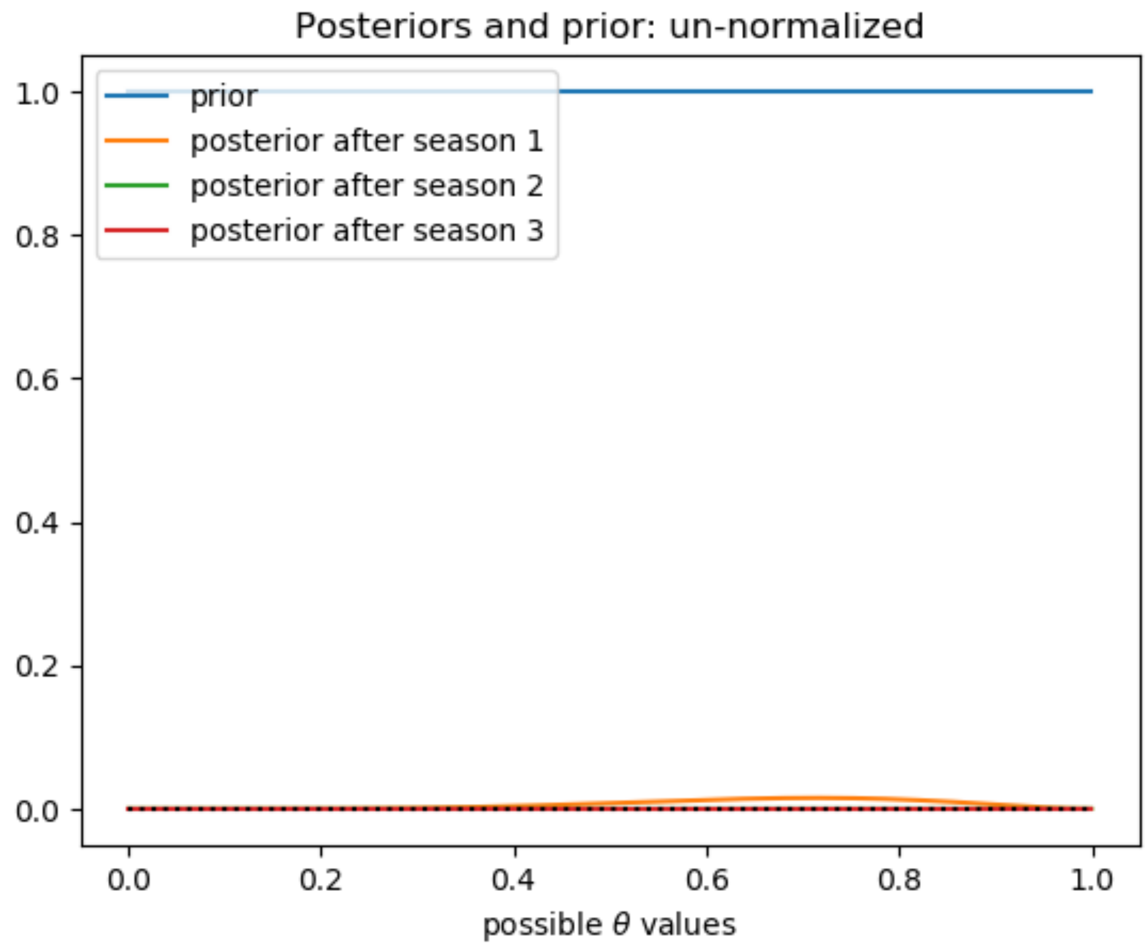
Out[17]:
```
(23, 8)
```

Here are some plots of the prior and posterior after each season with different normalizations

In [18]:
```
θs = range(0,1,length=1000)[2:end-1]

prior_pθ  = exp.(beta_logdensity.(θs, prior_betaφ[1], prior_betaφ[2]))
season1_pθ▯y = exp.(beta_logdensity.(θs, season1_betaφ[1], season1_bet
season12_pθ▯y = exp.(beta_logdensity.(θs, season12_betaφ[1], season12_
season123_pθ▯y = exp.(beta_logdensity.(θs, season123_betaφ[1], season1
```

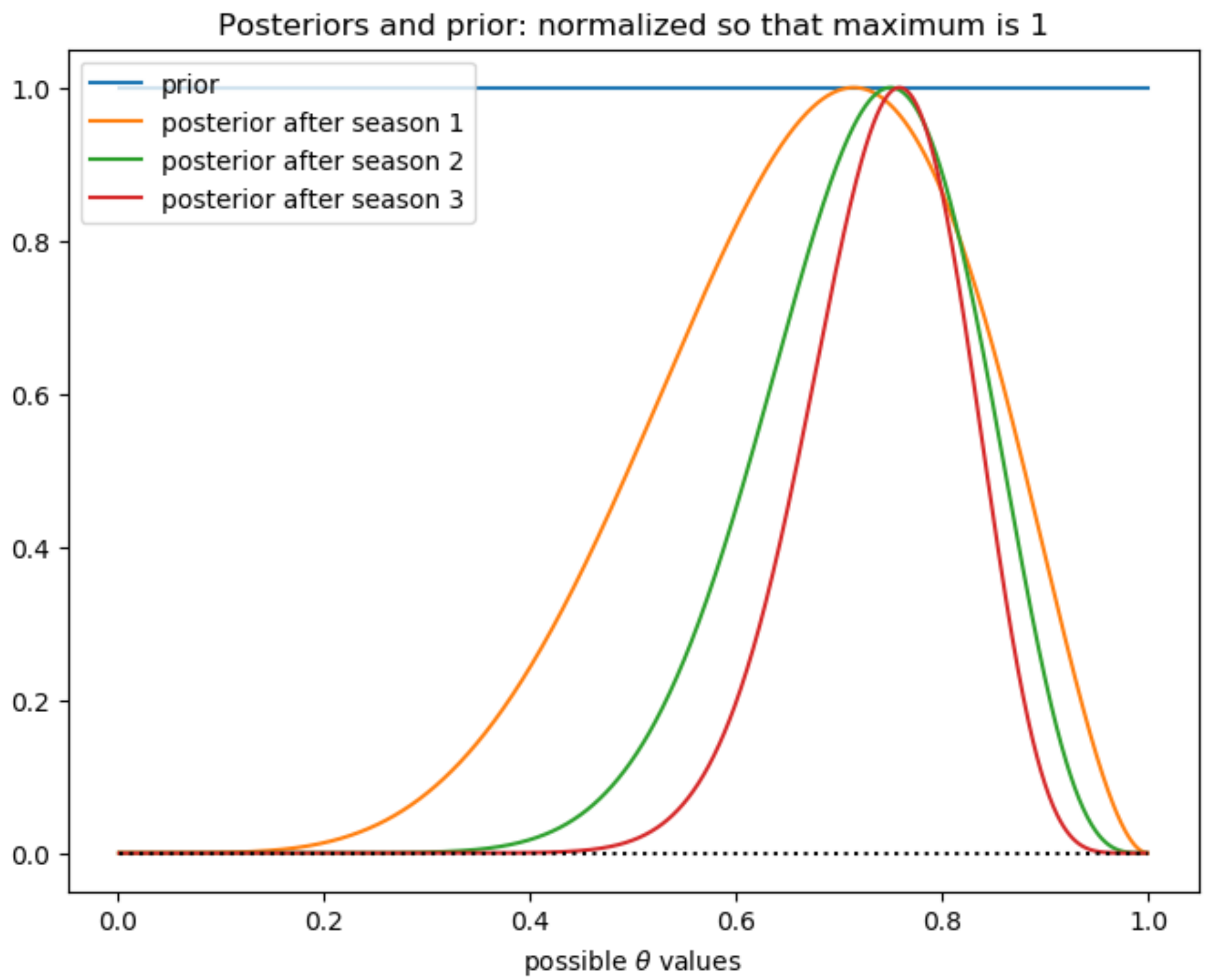
Out[18]:
```
998-element Array{Float64,1}:
 1.0151135630152371e-66
 4.227920907029572e-60
 3.141114837350033e-56
 1.7485667504413563e-53
 2.353128412100501e-51
 1.2899506311741365e-49
 3.8048665298032425e-48
 7.130108467748212e-47
 9.448925205931944e-46
 9.52741060756556e-45
 7.700847569239747e-44
 5.185711159200399e-43
 2.995663285477602e-42
 ⋮
 2.7659665424411634e-14
 1.5381632059414946e-14
 8.070849642047382e-15
 3.947050551942457e-15
 1.7695023716940644e-15
 7.104669974744994e-16
 2.4691249106690277e-16
 7.045167590027248e-17
 1.510527224854546e-17
 2.0613658160918504e-18
 1.2334009936070436e-19
 9.850828308805955e-22
```

In [19]:
```
fig, ax = subplots(1)
ax.plot(θs, prior_pθ, label="prior")
ax.plot(θs, season1_pθ▯y, label="posterior after season 1")
ax.plot(θs, season12_pθ▯y, label="posterior after season 2")
ax.plot(θs, season123_pθ▯y, label="posterior after season 3")
ax.plot(θs, 0 .* θs,"k:")
ax.set_xlabel(L"possible $\theta$ values")
ax.set_title("Posteriors and prior: un-normalized")
ax.legend(loc=2)
```



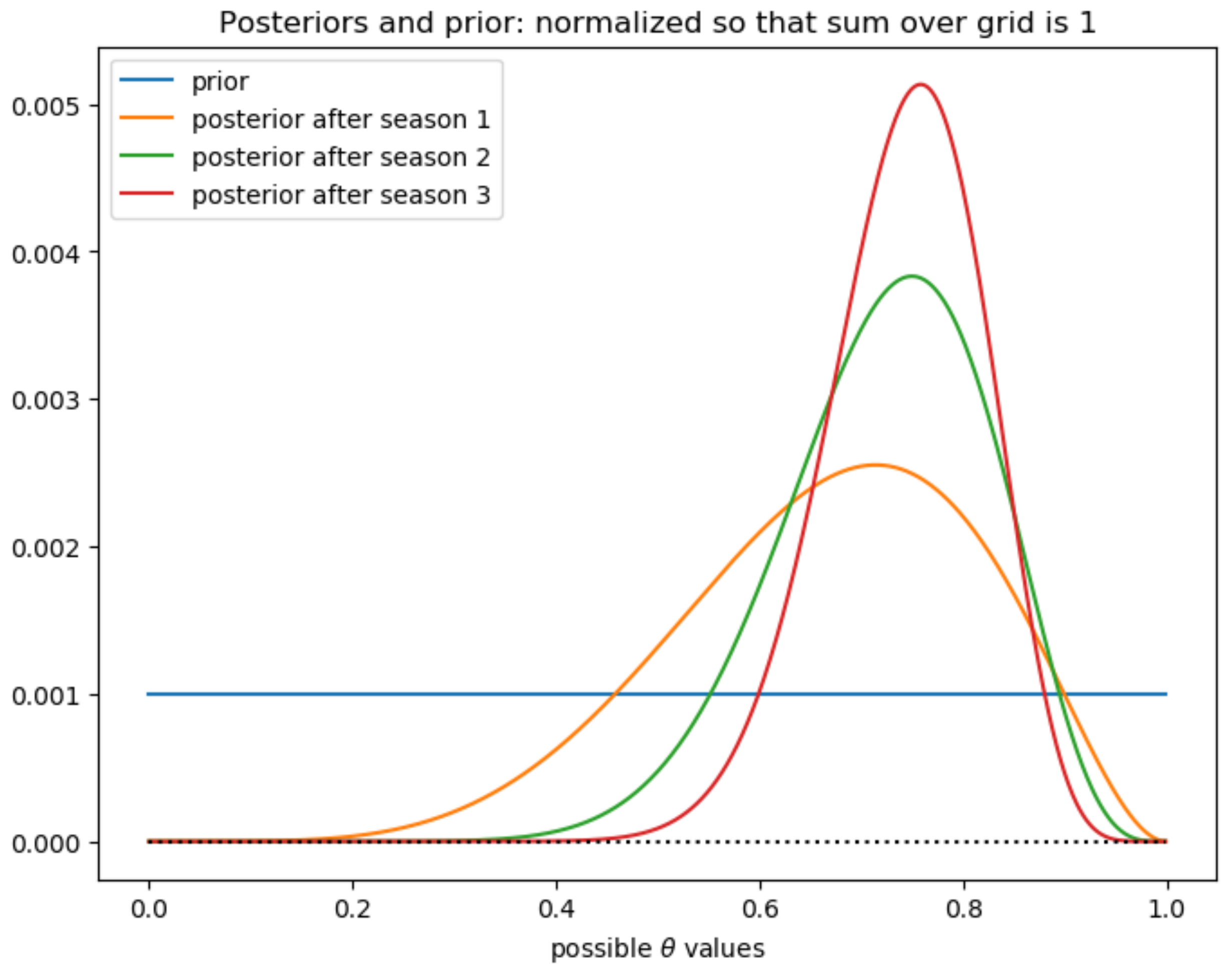
Out[19]:
```
PyObject <matplotlib.legend.Legend object at 0x144e14fd0>
```

In [20]:
```
normalize_p_max!(prior_pθ)
normalize_p_max!(season1_pθ▯y)
normalize_p_max!(season12_pθ▯y)
normalize_p_max!(season123_pθ▯y)

fig, ax = subplots(1,figsize=(8,6))
ax.plot(θs, prior_pθ, label="prior")
ax.plot(θs, season1_pθ▯y, label="posterior after season 1")
ax.plot(θs, season12_pθ▯y, label="posterior after season 2")
ax.plot(θs, season123_pθ▯y, label="posterior after season 3")
ax.plot(θs, 0 .* θs,"k:")
ax.set_xlabel(L"possible $\theta$ values")
ax.set_title("Posteriors and prior: normalized so that maximum is 1")
ax.legend(loc=2)
```



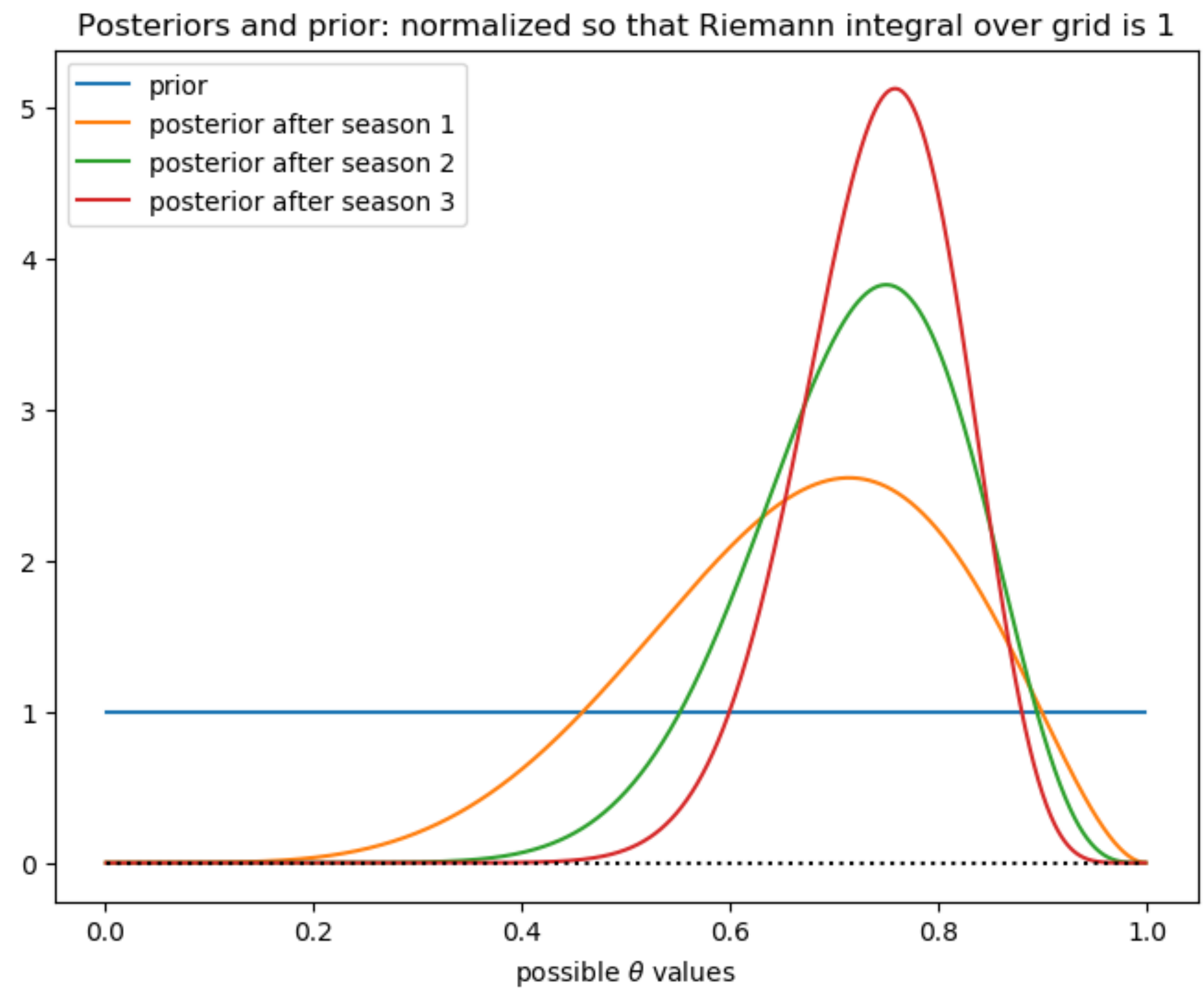
Out[20]: PyObject <matplotlib.legend.Legend object at 0x146434ed0>

In [21]:
```
normalize_p_pmf!(prior_pθ)
normalize_p_pmf!(season1_pθ∣y)
normalize_p_pmf!(season12_pθ∣y)
normalize_p_pmf!(season123_pθ∣y)

fig, ax = subplots(1,figsize=(8,6))
ax.plot(θs, prior_pθ, label="prior")
ax.plot(θs, season1_pθ∣y, label="posterior after season 1")
ax.plot(θs, season12_pθ∣y, label="posterior after season 2")
ax.plot(θs, season123_pθ∣y, label="posterior after season 3")
ax.plot(θs, 0 .* θs,"k:")
ax.set_xlabel(L"possible $\theta$ values")
ax.set_title("Posteriors and prior: normalized so that sum over grid i
ax.legend(loc=2)
```



Out[21]:
```
PyObject <matplotlib.legend.Legend object at 0x1460e7c50>
```

In [22]:
```
normalize_p_pdf!(prior_pθ,θs)
normalize_p_pdf!(season1_pθ▯y,θs)
normalize_p_pdf!(season12_pθ▯y,θs)
normalize_p_pdf!(season123_pθ▯y,θs)

fig, ax = subplots(1,figsize=(8,6))
ax.plot(θs, prior_pθ, label="prior")
ax.plot(θs, season1_pθ▯y, label="posterior after season 1")
ax.plot(θs, season12_pθ▯y, label="posterior after season 2")
ax.plot(θs, season123_pθ▯y, label="posterior after season 3")
ax.plot(θs, 0 .* θs,"k:")
ax.set_xlabel(L"possible $\theta$ values")
ax.set_title("Posteriors and prior: normalized so that Riemann integra
ax.legend(loc=2)
```



Out[22]: PyObject <matplotlib.legend.Legend object at 0x1468a5f50>

Betting on season 4 games with Vegas odds

Suppose that Vegas has put 8:1 odds for your favoriate team to win in each game of season 4. Which means you can either bet for your team to win or to lose on any particular game.

If you bet that your team will win, then Vegas pays out $\$1/8$ for every $\$1$ bet placed. If, instead you bet for your team to lose, Vegas pays out $\$8$ for every $\$1$ bet placed.

Question 1:

Suppose your planning to put $\$100$ on your teams first game of season 4 but are not sure if you should bet for them to win or to lose.

Use the Beta posterior based on the win/lose records of the previous seasons to simulate possible θ values from the posterior. For each of these θ possibilities simulate the outcome of the first game of the season and, based on who wins, determine your winnings when betting $\$100$ to win or lose. Finally use these simulations to find your expected winnings (which is negative if you lose the bet) for each of the two betting options.

In [53]:
```
########

N = 100000
some_possibleθ = beta_rand(season123_betaφ[1], season123_betaφ[2], N)
result = zeros(N);
```

In [54]:
```
for i = 1:N
    result[i] = bernoulli_rand(some_possibleθ[i])
end
```

In [83]:
```

#get the possibility of win and loss
p_win = mean(result .== 1)
```

Out[83]: 0.73956

In [84]:
```
p_loss = 1 - p_win
```

Out[84]: 0.26044

In [85]:
```
# get the expected value of bet win
p_win * 1/8*100 + p_loss * (-1) * 100
```

Out[85]: -16.799500000000002

In [86]:
```
# get the expected value of bet loss
p_win * (-1)*100 + p_loss * 8 * 100
```

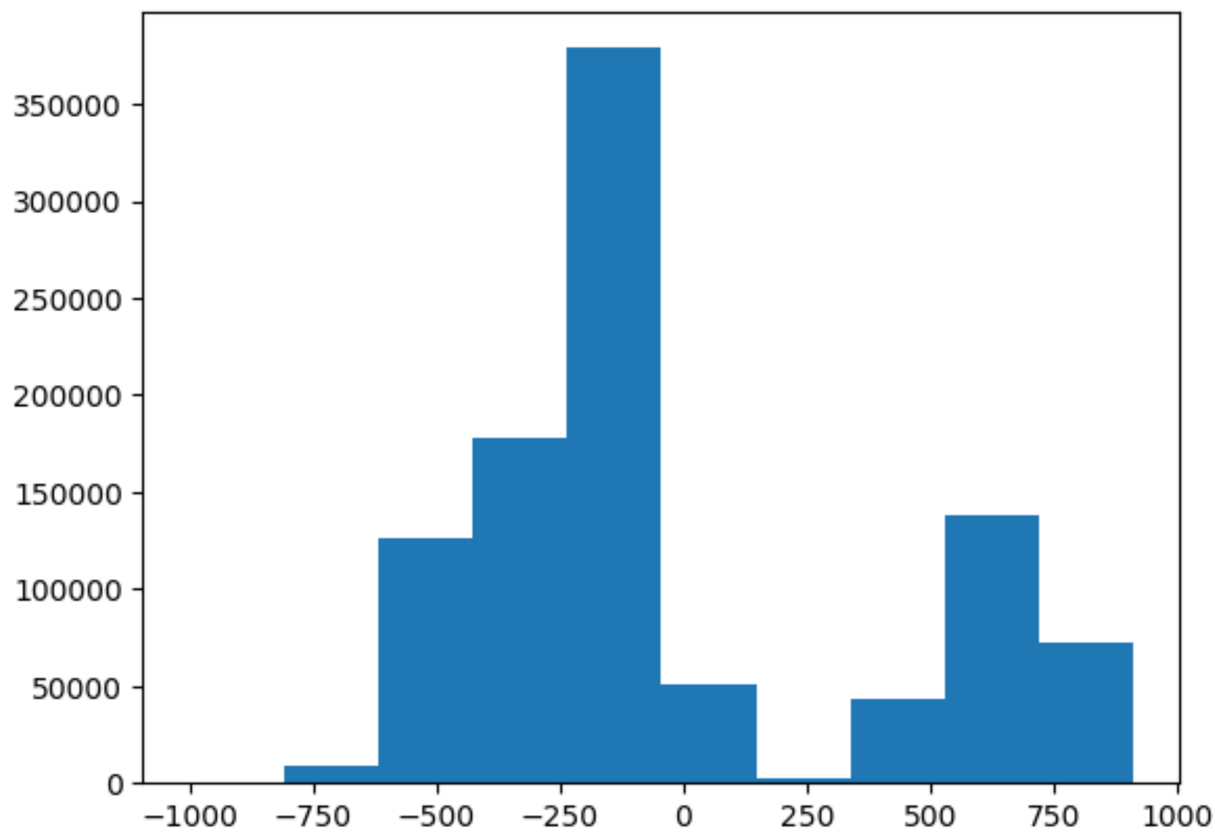
Out[86]: 134.39600000000002

Question 2:

Now suppose that in season 4 your team will play a total of 10 games. Suppose further that you have decided to bet $100 on each game. The first 9 games your going to bet on your team to win. The last game of the season your going to bet that your team doesn't win.

Simulate the your total winnings from season 4 (i.e. the sum of winnings from all 10 games) based on this betting strategy. Make a histrogram of these simulated values.

In [80]:
```
########
function total_winnings_for_question2(win_loss_10)
    winnings = zeros(size(win_loss_10))
    tmp = view(winnings, :, 1:9)
    tmp[win_loss_10[:, 1:9] .== 1] .+= 1/8 * 100
    tmp[win_loss_10[:, 1:9] .== 0] .-= 100
    tmp = view(winnings, :, 10)
    tmp[win_loss_10[:, 10] .== 1] .-= 100
    tmp[win_loss_10[:, 10] .== 0] .+= 8 * 100
    return sum(winnings, dims=2)
end
N = 1000000
alpha, beta = season123_betaφ
theta_samples_10 = beta_rand(alpha, beta, N, 10)
win_loss_samples_10 = bernoulli_rand.(theta_samples_10)
winnings = total_winnings_for_question2(win_loss_samples_10)
hist(winnings);
```



Find the probability you lose money in season 4.

In [81]:
```
######## probability_loss_money
mean(winnings .< 0)
```

Out[81]: 0.692283

Find the expected amount of money you make in season 4.

In [82]:
```
######## expected_money
mean(winnings)
```

Out[82]: -17.3918125