```
begin
    import Pkg
    # activate the shared project environment
    Pkg.activate(Base.current_project())
    using Omega, Distributions, UnicodePlots, OmegaExamples
end
```

```
  Activating project at `~/Documents/GitHub/Omega.jl/OmegaExamples`
```

```
Entities should not be multiplied without necessity. - William of Ockham
```

In learning and perceiving we are fitting models to the data of experience. Typically our hypothesis space will span models varying greatly in complexity: some models will have many more free parameters or degrees of freedom than others. Under traditional approaches to model fitting we adjust each model's parameters until it fits best, then choose the best-fitting model; a model with strictly more free parameters will tend to be preferred regardless of whether it actually comes closer to describing the true processes that generated the data. It then often generalizes less well – this is called over fitting.

But this is not the way the mind works. Humans assess models with a natural eye for complexity, balancing fit to the data with model complexity in subtle ways that will not inevitably prefer the most complex model. Instead we often seem to judge models using Occam's razor: we choose the least complex hypothesis that fits the data well. In doing so we avoid over-fitting our data in order to support successful generalizations and predictions.

Occam's Razor

An elegant and powerful form of Occam's razor arises in the context of Bayesian inference, known as the Bayesian Occam's razor. Bayesian Occam's razor refers to the fact that "more complex" hypotheses about the data are penalized automatically in conditional inference. In many formulations of Occam's razor, complexity is measured syntactically: for instance, it may be the description length of the hypothesis in some representation language, or a count of the number of free parameters used to specify the hypothesis. Syntactic forms of Occam's razor have difficulty justifying the complexity measure on principled, non-arbitrary grounds. They also leave unspecified exactly how the weight of a complexity penalty should trade off with a measure of fit to the data. Fit is intrinsically a semantic notion, a matter of correspondence between the model's predictions and our observations of the world. When complexity is measured syntactically and fit is measured semantically, they are intrinsically incommensurable and the trade-off between them will always be to some extent arbitrary.

In the Bayesian Occam's razor, both complexity and fit are measured semantically. The semantic notion of complexity is a measure of flexibility: a hypothesis that is flexible enough to generate many different sets of observations is more complex, and will tend to receive lower posterior probability than a less flexible hypothesis that explains the same data. Because more complex hypotheses can generate a greater variety of data sets, they must necessarily assign a lower probability to each one. When we condition on some data, all else being equal, the posterior distribution over the hypotheses will favor the simpler ones because they have the tightest fit to the observations.

From the standpoint of a probabilistic programming language, the Bayesian Occam's razor is essentially inescapable. We do not judge models based on their best fitting behavior but rather on their average behavior. No fitting per se occurs during conditional inference. Instead, we draw conditional samples from each model representing the model's likely ways of generating the data. A model that tends to fit the data well on average – to produce relatively many generative histories with that are consistent with the data – will do better than a model that can fit better for certain parameter settings but worse on average.

The Law of Conservation of Belief

It is convenient to emphasize an aspect of probabilistic modeling that seems deceptively trivial, but comes up repeatedly when thinking about inference. In Bayesian statistics we think of probabilities as being degrees of belief. Our generative model reflects world knowledge and the probabilities that we assign to the possible sampled values reflect how strongly we believe in each possibility. The laws of probability theory ensure that our beliefs remain consistent as we reason.

A consequence of belief maintenance is known as the Law of Conservation of Belief. Here are two equivalent formulations of this principle:

- Sampling from a distribution selects exactly one possibility (in doing so it implicitly rejects all other possible values).

- The total probability mass of a distribution must sum to 1. That is, we only have a single unit of belief to spread around.

The latter formulation leads to a common metaphor in discussing generative models: We can usefully think of belief as a "currency" that is "spent" by the probabilistic choices required to construct a sample. Since each choice requires "spending" some currency, an outcome that requires more choices to construct it will generally be more costly, i.e. less probable.

It is this conservation of belief that gives rise to the Bayesian Occam's razor. A hypothesis that spends its probability on many alternatives that don't explain the current data will have less probability for the alternatives that do, and will hence do less well overall than a hypothesis which only entertains options that fit the current data. We next examine a special case where this tradeoff plays out clearly, the size principle, then come back to the more general cases.

The Size Principle

A simple case of Bayes Occam's razor comes from the size principle (Tenenbaum and Griffiths, 2001): Of hypotheses which generate data uniformly, the one with smallest extension that is still consistent with the data is the most probable.

The following program demonstrates the size principle with a very simple model. Here we have two hypothesized sets: Big has 6 elements and Small has 3 elements. The generative model chooses one of the hypotheses at random and samples some number of symbols from it uniformly. We then wish to infer the hypothesis given observed elements.

hypothesis_to_dist (generic function with 1 method)

```
function hypothesis_to_dist(hyp, ω, i)
    if hyp== "Big"
        arr  = ["a", "b", "c", "d", "e", "f"]
        return arr[(i~Categorical(ones(6).* (1/6)))(ω)]
    else
        arr  = ["a", "b", "c"]
        return arr[(i~Categorical(ones(3).*(1/3)))(ω)]
    end
end
```

hypothesis (generic function with 1 method)

```
hypothesis(ω) = (@~ Bernoulli())(ω) ? "Big" : "Small"
```

rand_hyp (generic function with 1 method)

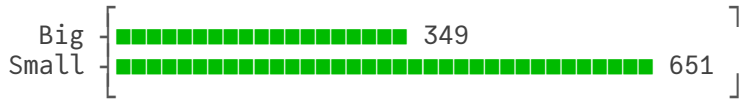
```
rand_hyp(ω, data) =
    map(i -> hypothesis_to_dist(hypothesis(ω), ω, i), 1:length(data))
```

data = ["a"]

```
data = ["a"]
```

post (generic function with 1 method)

```
post(data) = hypothesis |ᶜ pw(==, Variable(ω -> rand_hyp(ω, data)), data)
```




```
viz(randsample(post(data), 1000))
```

With a single observed `a`, we already favor hypothesis `Small`. What happens when we increase the amount of observed data? Consider the learning trajectory:

hyp_post (generic function with 1 method)

```
hyp_post(data) =
    (ω -> (hypothesis(ω) == "Big")) |ᶜ pw(==, Variable(ω -> rand_hyp(ω, data)), data)
```

```
full_data =  ["a", "b", "a", "b", "b", "a", "b"]
```

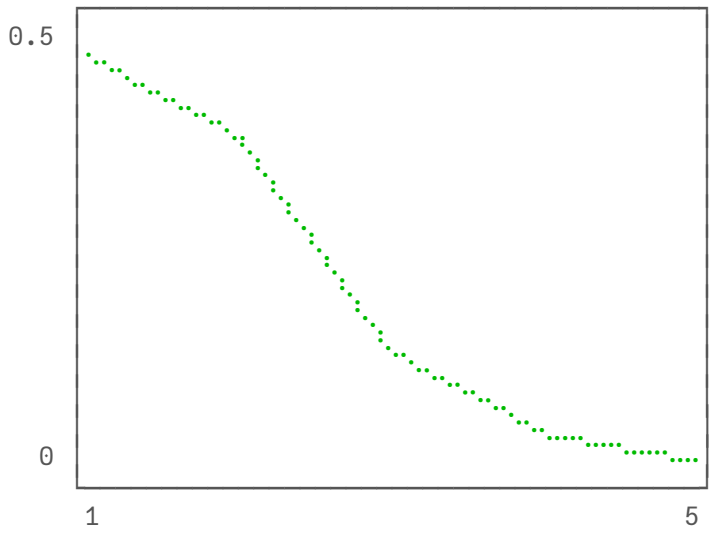
```
1 full_data = ["a", "b", "a", "b", "b", "a", "b"]
```

```
data_sizes =  [0, 1, 3, 5, 7]
```

```
1 data_sizes = [0, 1, 3, 5, 7]
```

```
prob_big (generic function with 1 method)
```

```
1 prob_big(size, data) = mean(randsample(hyp_post(data[1:size]), 100))
```




```
1 lineplot(map(i -> prob_big(i, full_data), data_sizes))
```

As the number of data points increases, the hypothesis `Small` rapidly comes to dominate the posterior distribution. Why is this happening? Observations are distributed uniformly over the hypothesized set, the law of conservation of belief and the symmetry between observations imply that the probability of a draw from `Big` is $\frac{1}{6}$, while the probability of a draw from `Small` is $\frac{1}{3}$. Thus, by the product rule of probabilities, the probability of drawing a set of N observations from `Big` is $(\frac{1}{6})^N$, while the probability of drawing a set of observations from `Small` is $(\frac{1}{3})^N$. The later probability decreases much more slowly than the former as the number of observations increases. Using Bayes' rule, the posterior distribution over hypotheses is given by:

$$P(hypothesis \mid observations) \propto P(observations \mid hypothesis)P(hypothesis)$$

Because our hypotheses are equally probable a priori, this simplifies to:

$$P(hypothesis \mid observations) \propto P(observations \mid hypothesis)$$

So we see that the posterior distribution over hypotheses, in this case, is just the normalized likelihood $P(observations \mid hypothesis)$. The likelihood ratio, $\frac{P(observations|Big)}{P(observations|Small)} = (\frac{1}{2})^N$, determines how quickly the simpler hypothesis `Small` comes to dominate the posterior.

One way to understand the Size Principle is that probabilistic inference takes into account *implicit negative evidence*. More flexible hypotheses could have generated more observations. Thus if those hypotheses were the true hypotheses we would expect to see a greater variety of observations. If the data does not contain them, this is a form of negative evidence against those hypotheses. Importantly,

the Size Principle tells us that the prior distribution on hypotheses does not have to penalize complexity. The complexity of the hypothesis itself will lead to its being disfavored in the posterior distribution.

The size principle is related to an influential proposal in linguistics known as the *subset principle*. Intuitively, the subset principle suggests that when two grammars both account for the same data, the grammar that generates a smaller language should be preferred. (The name “subset principle” was originally introduced by Bob Berwick to refer to a slightly different result.)

Generalizing the Size Principle: Bayes Occam's Razor

In our example above we have illustrated Bayes Occam's razor with examples based strictly on the "size" of the hypotheses involved, however, the principle is more general. The Bayesian Occam's razor says that all else being equal the hypothesis that assigns the highest likelihood to the data will dominate the posterior. Because of the law of conservation of belief, assigning higher likelihood to the observed data requires assigning lower likelihood to other possible data. Consider the following example:

```
hypothesis_to_dist_ (generic function with 1 method)
```

```
function hypothesis_to_dist_(hyp, ω, i)
    arr  = ["a", "b", "c", "d"]
    if hyp== "A"
        return arr[(i~Categorical([0.375, 0.375, 0.125, 0.125]))(ω)]
    else
        return arr[(i~Categorical([0.25, 0.25, 0.25, 0.25]))(ω)]
    end
end
```

```
obs_data =  ["a", "b", "a", "b", "c", "d", "b", "b"]
```

```
obs_data = ["a", "b", "a", "b", "c", "d", "b", "b"]
```

```
hypothesis_ (generic function with 1 method)
```

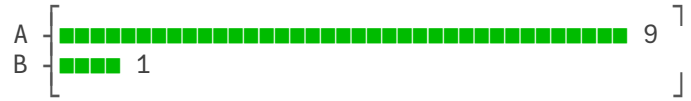
```
hypothesis_(ω) = (@~ Bernoulli())(ω) ? "A" : "B"
```

```
rand_hyp_ (generic function with 1 method)
```

```
rand_hyp_(ω, data) =
    map(i -> hypothesis_to_dist_(hypothesis_(ω), ω, i), 1:length(data))
```

```
posterior (generic function with 1 method)
```

```
posterior(data) = hypothesis_ |ᶜ pw(==, Variable(ω -> rand_hyp_(ω, data)), data)
```




```
viz(randsample(posterior(obs_data), 10))
```

In this example, unlike the size principle cases above, both hypotheses lead to the same possible observed values. However, hypothesis A is skewed toward examples a and b – while it can produce c or d, it is less likely to do so. In this sense hypothesis A is less flexible than hypothesis B. The data set we conditioned on also has exemplars of all the elements in the support of the two hypotheses. However, because there are more exemplars of elements favoured by hypothesis A, this hypothesis is favoured in the posterior. The Bayesian Occam's razor emerges naturally here.

These examples suggest another way to understand Bayes Occam's razor: the posterior distribution will favour hypotheses for which the data set is simpler in the sense that it is more "easily generated." Here more "easily" generated means generated with higher probability. We will see a more striking example of this for compositional models at the end of this section of the tutorial.

Model selection with the Bayesian Occam's Razor

The law of conservation of belief turns most clearly into Occam's Razor when we consider models with more internal structure: some continuous or discrete parameters that at different settings determine how likely the model is to produce data that look more or less like our observations. To select among models we simply need to describe each model as a probabilistic program, and also to write a higher-level program that generates these hypotheses.

Example: Fair or unfair coin?

In a previous chapter, we considered learning about the weight of a coin and noted that a simple prior on weights seemed unable to capture our more discrete intuition – that we first decide if the coin is fair or not, and only then worry about its weight. This example shows how our inferences about coin flipping can be explained in terms of model selection guided by the Bayesian Occam's razor. Imagine a coin that you take out of a freshly unwrapped roll of quarters straight from the bank. Almost surely this coin is fair... But how does that sense change when you see more or less anomalous sequences of flips? We can simultaneously ask if the coin is fair, and what is its weight.

```
observed_data =  ["h", "h", "t", "h", "t", "h", "h", "h", "t", "h"]
```

```
observed_data = ["h", "h", "t", "h", "t", "h", "h", "h", "t", "h"] # fair coin
# observed_data = repeat(["h"], 10) #? suspicious coincidence, probability of H = 0.5
# observed_data = repeat(["h"], 15) # probably unfair - probablility of h is near 1
# observed_data = repeat(["h"], 20) # definitely unfair - probablility of h is near 1
# observed_data = randsample(ifelseₚ((@~ Bernoulli(0.85)), "h", "t") # unfair coin, probability of H = 0.85
```

```
fair_prior = 0.999
```

```
fair_prior = 0.999
```

```
pseudo_counts =  (α = 1, β = 1)
```

```
pseudo_counts = (α = 1, β = 1)
```

```
fair = 4076790675392942329@Distributions.Bernoulli{Float64}(p=0.999)
```

```
fair = @~ Bernoulli(fair_prior)
```

```
coin_weight = ifelseₚ(4076790675392942329@Distributions.Bernoulli{Float64}(p=0.999), 0.5, -23046766285408
```

```
coin_weight = ifelse.(fair, 0.5, @~ Beta(pseudo_counts...))
```


```
viz(randsample(coin_weight, 1000) .- 0.5)
```

```
evidence (generic function with 1 method)
```

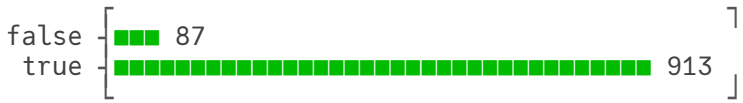
```
evidence(data) =
    pw(==, manynth(Bernoulli(coin_weight), 1:length(data)), (data .== "h"))
```

```
posterior_ (generic function with 1 method)
```

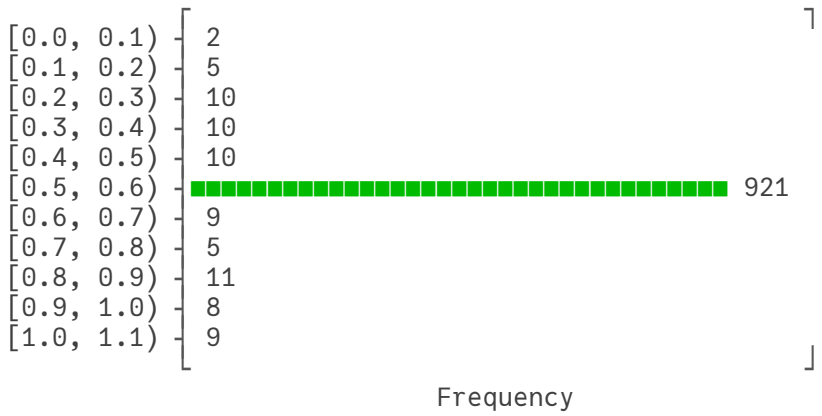
```
posterior_(data) = (@joint fair coin_weight) |ᶜ evidence(data)
```

```
post_ =
 [(fair = true, coin_weight = 0.5), (fair = true, coin_weight = 0.5), (fair = true, coin_weight
```

```
post_ = randsample(posterior_(observed_data), 1000, alg = MH)
```




```
viz(map(p -> p.fair, post_))
```




```
viz(map(p -> p.coin_weight, post_))
```

Try some of the observation sets that we've commented out above and see if the inferences accord with your intuitions.

Now let's look at the learning trajectories for this model:

```
true_coin =
ifelseₚ(17941605632947827 61@Distributions.Bernoulli{Float64}(p=0.9), Base.RefValue{String}(
```

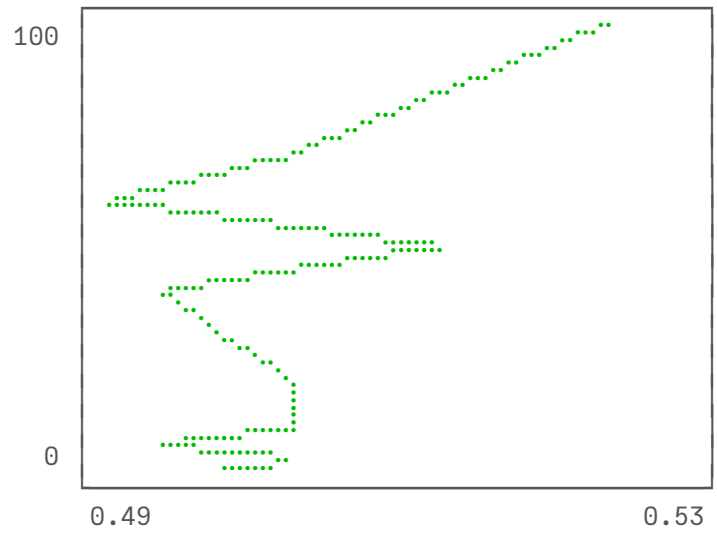
```
true_coin = ifelse.((@~ Bernoulli(0.9)), "h", "t")
```

```
data_sizes_ =  [0, 1, 3, 6, 10, 20, 30, 40, 50, 60, 70, 100]
```

```
data_sizes_ = [0,1,3,6,10,20,30,40,50,60,70,100]
```

```
predictions =
 [0.498508, 0.501334, 0.502992, 0.494735, 0.503457, 0.503192, 0.498945, 0.494977, 0.512883, 0
```

```
predictions =
    mean.(map(d -> getfield.(randsample(posterior_(randsample(true_coin, d)), 1000,
    alg = MH), :coin_weight), data_sizes_))
```




```
lineplot(predictions, data_sizes_)
```

In general (though not on every run) the learning trajectory stays near 0.5 initially—favouring the simpler hypothesis that the coin is fair—then switches fairly abruptly to near 0.9 — as it infers that it is an unfair coin and likely has high weight. Here the Bayesian Occam's Razor penalizes the hypothesis with the flexibility to learn any coin weight until the data overwhelmingly favour it.

The Effect of Unused Parameters

When statisticians suggest methods for model selection, they often include a penalty for the *number* of parameters. This seems like a worrying policy from the point of view of a probabilistic program: we could always introduce parameters that are not used, and therefore have no effect on the program. For instance, we could change the above coin flipping example so that it draws the potential unfair coin weight even in the model which gives a fair coin:

```
unfair_weight = -2455216454684399565@Distributions.Beta{Float64}(α=1.0, β=1.0)
```

```
unfair_weight = @~ Beta(pseudo_counts...)
```

```
coin_weight_ =
ifelse_p(4076790675392942329@Distributions.Bernoulli{Float64}(p=0.999), 0.5, -24552164546843
```

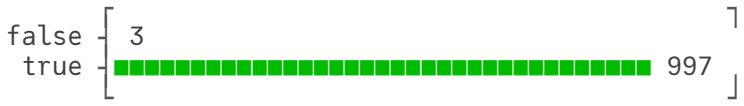
```
coin_weight_ = ifelse.(fair, 0.5, unfair_weight)
```

```
results =
  Conditional(#15 (generic function with 1 method), ==_p(2245772204142488467@#1, true))
```

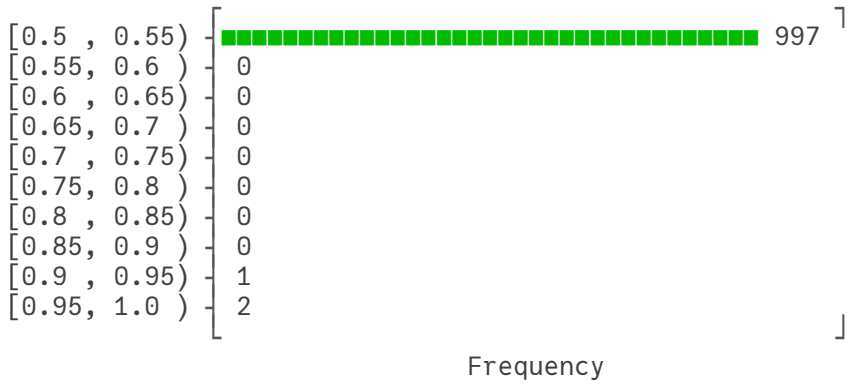
```
results = (@joint fair coin_weight_) |ᶜ ((@~ Bernoulli(coin_weight_)) .== true)
```

```
results_samples =
  [(fair = true, coin_weight_ = 0.5), (fair = true, coin_weight_ = 0.5), (fair = true, coin_weigh
```

```
results_samples = randsample(results, 1000)
```




```
viz(map(p -> p.fair, results_samples))
```




```
viz(map(p -> p.coin_weight_, results_samples))
```

The two models now have the same number of free parameters (the unfair coin weight), but we will still favor the simpler hypothesis, as we did above. Why? The Bayesian Occam's razor penalizes models not for having more parameters (or longer code) but for too much flexibility – being able to fit too many other potential observations. Unused parameters (or parameters with very little effect) don't increase this flexibility, and hence aren't penalized. The Bayesian Occam's razor only penalizes complexity that matters for prediction, and only to the extent that it matters.

Example: Curve Fitting

This example shows how the Bayesian Occam's Razor can be used to select the right order of a polynomial fit.

```
make_poly (generic function with 1 method)
```

```
make_poly(as) = var -> sum(map(x -> x[2] * (var ^ (x[1] - 1)), enumerate(as)))
```

```
coeffs =  Mv(1:4, Distributions.Normal{Float64}(μ=0.0, σ=2.0))
```

```
coeffs = manynth(Normal(0, 2), 1:4)
```

```
order = -8991556983631779698@Distributions.DiscreteUniform(a=1, b=4)
```

```
order = @~ DiscreteUniform(1, 4)
```

```
f (generic function with 1 method)
```

```
f(ω) = make_poly(coeffs(ω)[1 : order(ω)])
```

```
fn (generic function with 1 method)
```

```
fn(i, ω, x) = (i ~ Normal(f(ω)(x), 2))(ω)
```

```
obs_fn (generic function with 1 method)
```

```
obs_fn(data) = ω -> all(map(d -> fn(d[1], ω, d[2].x) == d[2].y, enumerate(data)))
```

```
ret = #25 (generic function with 1 method)
```

```
ret = @joint order coeffs
```

```
obs_data_ =
  [(x = -4, y = 69.7664), (x = -3, y = 36.6359), (x = -2, y = 19.9524), (x = -1, y = 4.81949), (
```

```
obs_data_ = [
    (x = - 4, y = 69.76636938284166),
    (x = -3, y = 36.63586217969598),
    (x = -2, y = 19.95244368751754),
    (x = -1, y = 4.819485497724985),
    (x = 0, y = 4.027631414787425),
    (x = 1, y = 3.755022418210824),
    (x = 2, y = 6.557548104903805),
    (x = 3, y = 23.922485493795072),
    (x = 4, y = 50.69924692420815)
]
```

```
p_ =
  Conditional(#25 (generic function with 1 method), #1 (generic function with 1 method))
```

```
p_ = ret |ᶜ obs_fn(obs_data_)
```

```
  [(order = 2, coeffs = [0.368245, -0.591138, -0.344739, -0.319122]), (order = 2, coeffs = [0.36
```

```
randsample(p_, 1000, alg = MH)
```

Try the above code using a different data set generated from the same function:

```
size = -4:4
```

```
size = -4:4
```

```
data_ =
  [(-4, -69.4599), (-3, -36.6821), (-2, -17.5781), (-1, -8.31146), (0, -5.04598), (1, -3.94535)
```

```
data_ = randsample(ω -> map(x -> (x, (@~ Normal(f(ω)(x), 2))(ω)), size))
```

You can also try making the data set smaller, or generate data from a different order polynomial. How much data does it take tend to believe the polynomial is third order?