STAT221Week03

Conditional Probability, Random Variables, Loops and Conditionals

Monte Carlo Methods

Probability (continued)

Recap on probability

An **experiment** is an activity or procedure that produces distinct or well-defined **outcomes**. The set of such outcomes is called the **sample space** of the experiment. The sample space is usually denoted with the symbol Ω.

An **event** is a subset of the sample space.

Probability is a function that assigns numbers in the range 0 to 1 to events

$$P : \text{set of events } \rightarrow [0,1]$$

which satisfies the following **axioms**:

1. For any event A, $0 \leq P(A) \leq 1$.
2. If Ω is the sample space, $P(\Omega) = 1$.
3. If A and B are disjoint (i.e., $A \cap B = \emptyset$), then $P(A \cup B) = P(A) + P(B)$.
4. If $A_1, A_2, \ldots$ is an infinite sequence of pair-wise disjoint events (i.e., $A_i \cap A_j = \emptyset$ when $i = j$), then

$$\begin{array}{rcl} \underbrace{P\left(\bigcup_{i=1}^{\infty} A_i\right)} & = & \underbrace{\sum_{i=1}^{\infty} P(A_i)} \\ A_1 \cup A_2 \cup A_3 \ldots & = & P(A_1) + P(A_2) + P(A_3) + \ldots \end{array}$$

Property 1

$P(A) = 1 - P(A^c)$, where $A^c = \Omega \setminus A$

jsMath

Property 2

For any two events A, B,

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

The idea in Property 2 generalises to the "**inclusion-exclusion**" formula.

Let $A_1, A_2, \ldots, A_n$ be any n events. Then,

$$\begin{aligned} P\left(\bigcup_{i=1}^{n} A_i\right) &= \sum_{i=1}^{n} P(A_i) - \sum_{i<j} P(A_i \cap A_j) \\ &\quad + \sum_{a<j<k} P(A_i \cap A_j \cap A_k) + \ldots + (-1)^{n+1} P(A_1 \cap A_2 \cap \ldots \cap A_n) \end{aligned}$$

In words, we take all the possible intersections of one, two, three, ..., n events and let the signs alternate.

Question

Does the inclusion-exclusion formula agree with the extended Axiom 3: If $A_1, A_2, \ldots, A_n$ are pair-wise disjoint events then $P\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} P(A_i)$

The domain of the probability function

What exactly is stipulated by the axioms about the domain of the probability function?

The domain should be a sigma field (σ-field) or sigma algebra (σ-algebra), denoted $\sigma(\Omega)$ or $\mathcal{F}$ such that:

1. $\Omega \in \mathcal{F}$
2. $A \in (F) \Rightarrow A^c \in \mathcal{F}$
3. $A_1, A_2, \ldots \in \mathcal{F} \Rightarrow \cup A_i \in \mathcal{F}$

Thus the domain of the probability is not just any old set of events (recall events are subsets of Ω), but rather a set of events that form a $sigma$-field that contains the sample space Ω, is closed under complementation and countable union.

$(\Omega, \mathcal{F}(\Omega), P)$ is called a **probability space** or **probability triple**.

Example

Let $\Omega = \{H, T\}$. What σ-fields could we have?

$\mathcal{F}(\Omega) = \{\{H, T\}, \emptyset, \{H\}, \{T\}\}$ is the finest σ-field.

$\mathcal{F}'(\Omega) = \{\{H, T\}, \emptyset\}$ is a trivial σ-field.

Example

Let $\Omega = \{\omega_1, \ldots, \omega_n\}$.

$\mathcal{F}(\Omega) = 2^{\Omega}$, the set of all subsets of Ω, also known as the power set of Ω.

$|2^{\Omega}| = 2^n$.

Independence

Two events A and B are independent if $P(A \cap B) = P(A)P(B)$.

Intuitively, A and B are independent if the occurrence of A has no influence on the probability of the occurrence of B (and vice versa).

Example

Flip a fair coin twice. Event A is the event "The first flip is 'H'"; event B is the event "The second flip is 'H'".

jsMath

$P(A) = \frac{1}{2}$, $P(B) = \frac{1}{2}$

Because the flips are independent (what happens on the first flip does not influence the second flip),

$P(A \cap B) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$.

Example

We can *generalise* this by saying that we will flip a coin with an unknown probability parameter $\theta \in [0, 1]$. We flip this coin twice and the coin is made so that for any flip, $P(\text{'H'}) = \theta$, $P(\text{'T'}) = 1 - \theta$.

Take the same events as before: event A is the event "The first flip is 'H'"; event B is the event "The second flip is 'H'".

Because the flips are independent,

$P(A \cap B) = \theta \times \theta = \theta^2$.

If we take event C as the event "The second flip is 'T'", then

$P(A \cap C) = \theta \times (1 - \theta)$.

Example

Roll a fair die twice. The face of the die is enumerated 1, 2, 3, 4, 5, 6.

Event A is the event "The first roll is 5"; event B is the event "The second roll is 1".

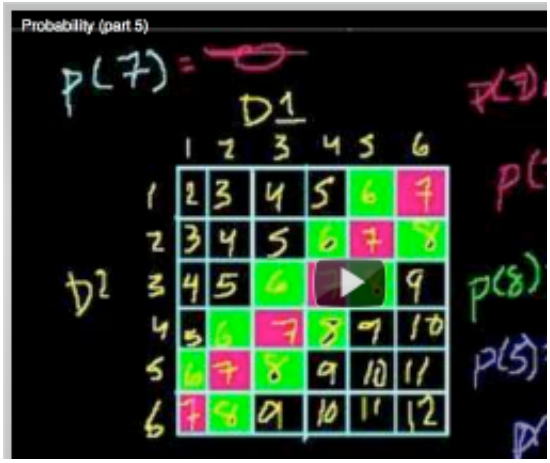
$P(A) = \frac{1}{6}$, $P(B) = \frac{1}{6}$

If the two rolls are independent,

$P(A \cap B) = \frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$

You try at home

Suppose you roll two fair dice independently. What is the probability of getting the sum of the outcomes to be seven? Solution: Please do this at home or with earphones! Watch the Khan Academy movie about probability and two dice.

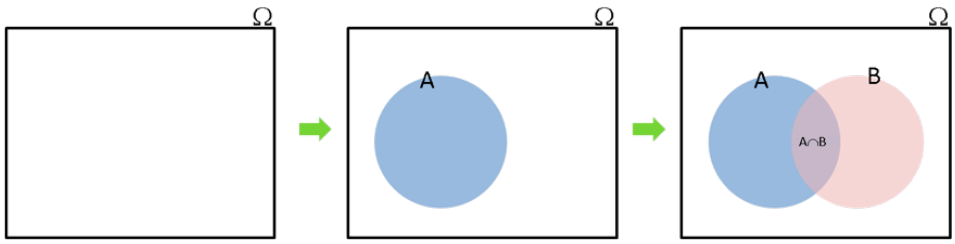


Conditional probability

Suppose that we are told that the event A with $P(A) > 0$ occurs and we are interested in whether another event B will now occur. The sample space has shrunk from Ω to A. The probability that event B will occur **given that A has already occurred** is defined by

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

jsMath



We can understand this by noting that

- Only the outcomes in B that also belong to A can possibly now occur, and
- Since the new sample space is A, we have to divide by $P(A)$ to make

$$P(A|A) = \frac{P(A \cap A)}{P(A)} = \frac{P(A)}{P(A)} = 1$$

If the two events A and B are independent then

$$P(B|A) = \frac{P(B \cap A)}{P(A)} = \frac{P(B)P(A)}{P(A)} = P(B)$$

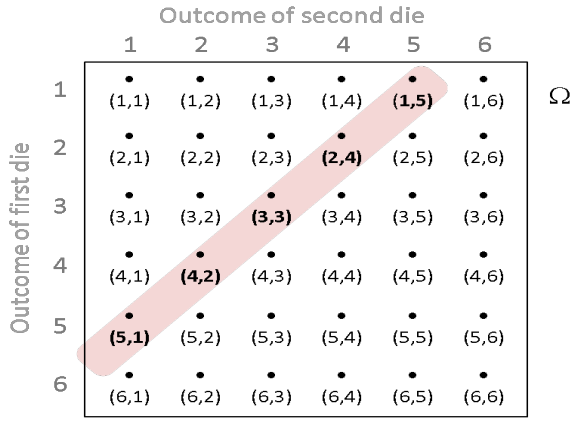
which makes sense - we said that if two events are independent, then the occurrence of A has no influence on the probability of the occurrence of B.

Example

Roll two fair dice.

Event A is the event "The sum of the dice is 6"; event B is the event "The first die is 2".

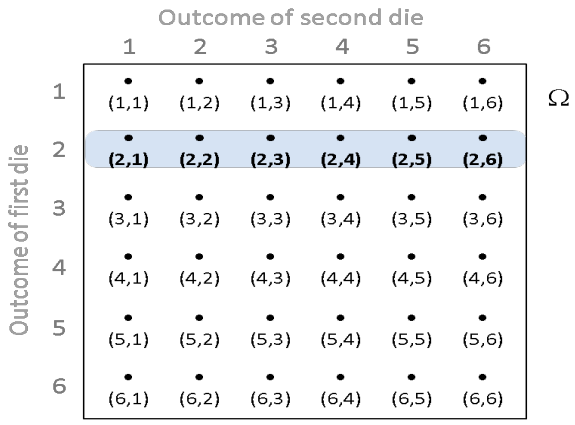
How many ways can we get a 6 out of two dice?



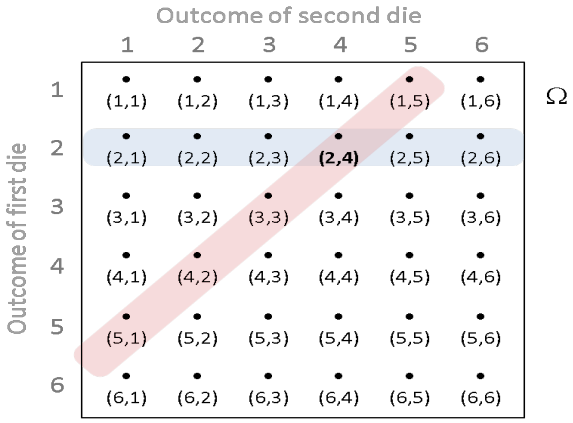
$A = \{(1,5), (2,4), (3,3), (4,2), (5,1)\}$

$P(A) = \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} = \frac{5}{36}$

jsMath



$B = \{(2,1),(2,2),(2,3),(2,4),(2,5),(2,6)\}$

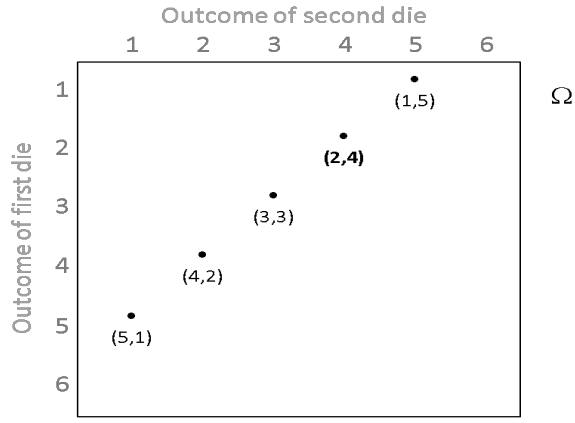


$B \cap A = \{(2,4)\}$

$P(B \cap A) = \frac{1}{36}$

$P(B|A) = \frac{P(B \cap A)}{P(A)} = \frac{\frac{1}{36}}{\frac{5}{36}} = \frac{1}{5}$

Look at this result in terms of what we said about the sample space shrinking to A.



Bayes Theorem

We just saw that $P(B|A)$, the conditional probability that event B will occur **given that A has already occurred** is

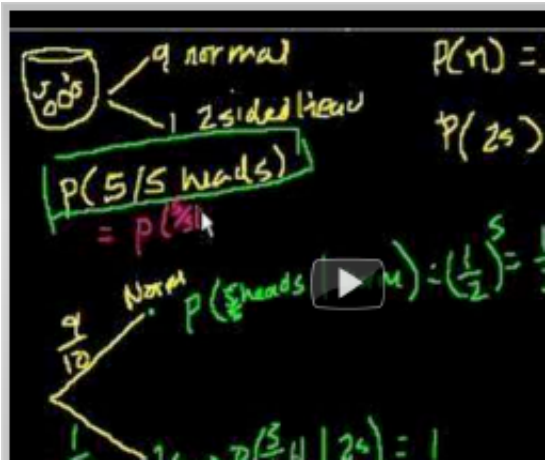
jsMath

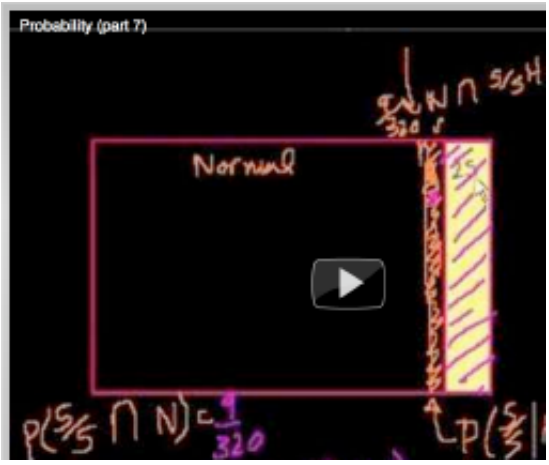
defined by $P(B \cap A)/P(A)$. By using the fact that $B \cap A = A \cap B$ and reapplying the definition of conditional probability to $P(A|B) = P(A \cap B)/P(B)$, we get the so-called Bayes theorem.

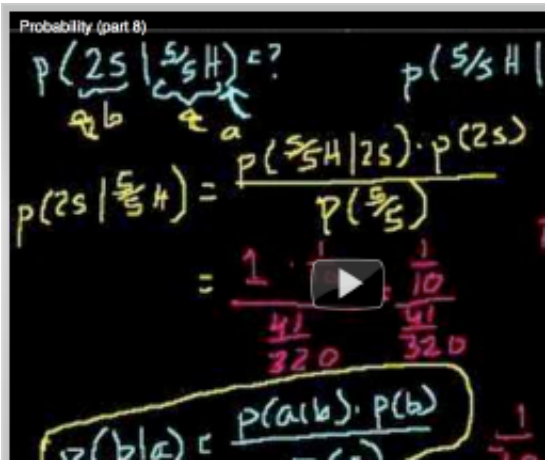
$$\boxed{P(B|A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A \cap B)}{P(A)} = \frac{P(A|B)P(B)}{P(A)}}$$

You try at home

Suppose we have a bag of ten coins. Nine of the ten coins are fair but one of the coins has heads on both sides. What is the probability of getting five heads in a row if I picked a coin at random from the bag and flipped it five times? If I obtained five heads in a row by choosing a coin out of the bag at random and flipping it five times, then what is the probability that I have picked the two-headed coin?
Solution: Please do this at home or with earphones! Watch the Khan Academy movies about applications of conditional probability and Bayes theorem to this bag of 10 coins.







You try

The next cell uses a function called `randint` which we will talk about more later in the course. For this week we'll just use `randint` as a computerised way of rolling a die: every time we call `randint(1,6)` we will get some integer number from 1 to 6, we won't be able to predict in advance what we will get, and the probability of each of the numbers 1, 2, 3, 4, 5, 6 is equal. Here we use randint to simulate the experiment of tossing two dice. The sample space Ω is all 36 possible ordered pairs $(1,1), \ldots (6,6)$. We print out the results for each die. Try evaluating the cell several times and notice how the numbers you get differ each time.

```
randint?
```

```
die1 = randint(1,6)
die2 = randint(1,6)
print "(die 1, die2) is (", die1, ", ", die2, ")"
```

jsMath

Random Variables

A **random variable** is a mapping from the sample space Ω to the set of real numbers $\mathbb{R}$. In other words, it is a numerical value determined by the outcome of the experiment.

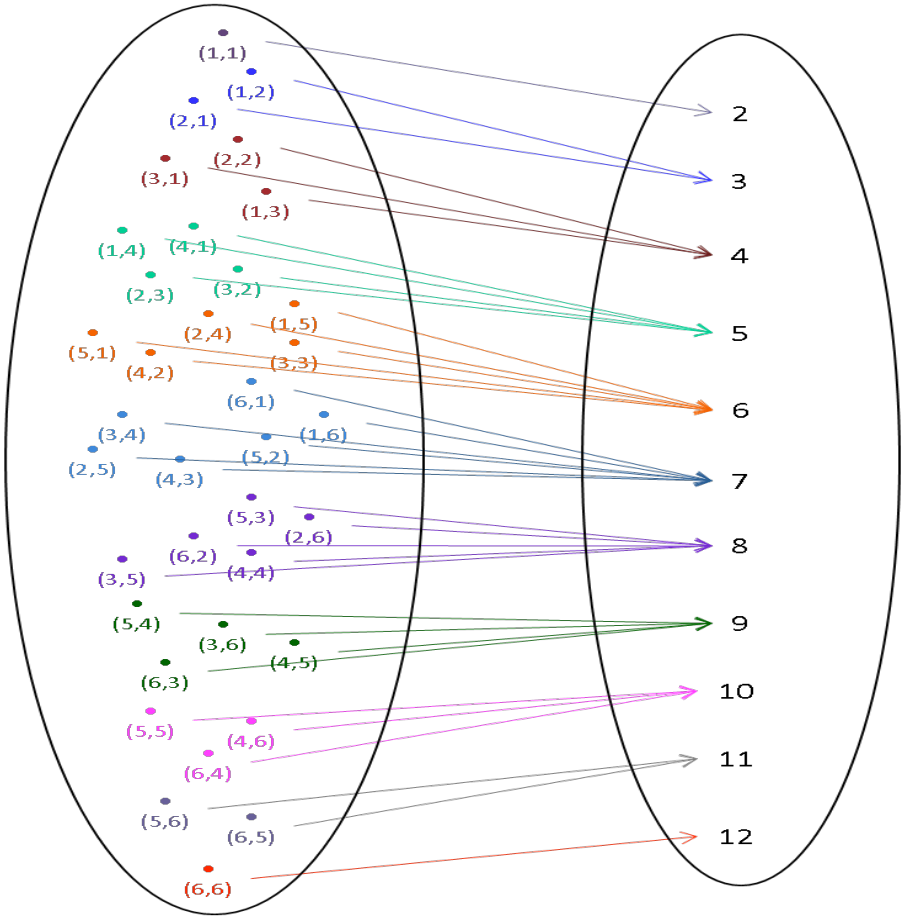
This is not as complicated as it sounds: let's look at a simple example:

Example

Roll two fair dice.

The sample space is the set of 36 ordered pairs $\Omega = \{(1,1),(1,2),\ldots,(2,1),(2,2),\ldots,(1,6),\ldots,(6,6)\}$

Let random variable X be the sum of the two numbers that appear, $X : \Omega \to \mathbb{R}$.



For example, $X(\{(6,6)\}) = 12$

$P(X = 12) = P(\{(6,6)\})$

And, $X(\{(3,2)\}) = 5$

Formal definition of a random variable

jsMath

Let $(\Omega, \mathcal{F}, P)$ be some probability triple. Then a random variable, say X, is a function from the sample space Ω to the set of real numbers $\mathbb{R}$

$$X : \Omega \to \mathbb{R}$$

such that for every $x \in \mathbb{R}$, the inverse image of the half-open interval $(-\infty, x]$ is an element of the collection of events $\mathcal{F}$, i.e.,

for every $x \in \mathbb{R}$,

$$X^{[-1]}\left((-\infty, x]\right) := \{\omega : X(\omega) \leq x\} \in \mathcal{F}$$

Discrete random variable

A random variable is said to be **discrete** when it can take a countable sequence of values (a finite set is countable). The three examples below are discrete random variables.

Probability of a random variable

Finally, we assign probability to a random variable X as follows:

$$P(X \leq x) = P(X^{[-1]}\left((-\infty, x]\right) := P\left(\{\omega : X(\omega) \leq x\}\right)$$

Distribution Function

The distribution function (DF) or cumulative distribution function (CDF) of any RV X, denoted by F is:

$$F(x) := P(X \leq x) = P\left(\{\omega : X(\omega) \leq x\}\right), \text{for any } x \in \mathbb{R}$$

Example - Sum of Two Dice

In our example above (tossing two die and taking X as the sum of the numbers shown) we said that $X\left((3,2)\right) = 5$, but (3,2) is not the only outcome that X maps to 5: $X^{[-1]}\left(5\right) = \{(1,4),(2,3),(3,2),(4,1)\}$

$$\begin{aligned} P(X=5) &= P\left(\{\omega : X(\omega) = 5\}\right) \\ &= P\left(X^{[-1]}\left(5\right)\right) \\ &= P(\{(1,4),(2,3),(3,2),(4,1)\}) \end{aligned}$$

Example - Pick a Fruit at Random

Remember our "well-mixed" fruit bowl containing 3 apples, 2 oranges, 1 lemon? If our experiment is to take a piece of fruit from the bowl and the outcome is the kind of fruit we take, then we saw that $\Omega = \{\text{apple}, \text{orange}, \text{lemon}\}$.

Define a random variable Y to give each kind of fruit a numerical value: $Y(\text{apple}) = 1$, $Y(\text{orange}) = 0$, $Y(\text{lemon}) = 0$.

Example - Flip Until Heads

Flip a fair coin until a 'H' appears. Let X be the number of times we have to flip the coin until the first 'H'.

$\Omega = \{\text{H}, \text{TH}, \text{TTH}, \ldots, \text{TTTTTTTTTH}, \ldots\}$

$X(\text{H}) = 0,\ X(\text{TH}) = 1, X(\text{TTH}) = 2, \ldots$

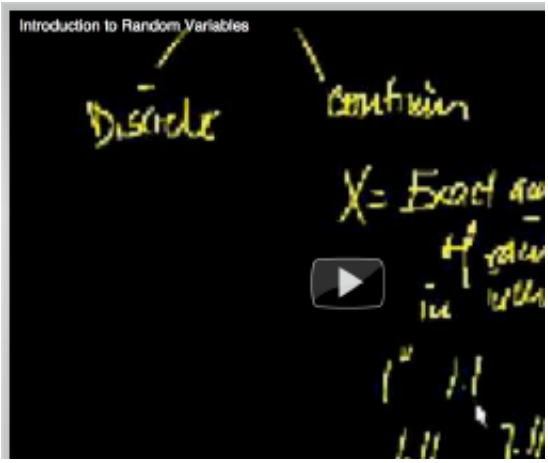
You try at home

Consider the example above of 'Pick a Fruit at Random'. We defined a random variable Y there as $Y(\text{apple}) = 1$, $Y(\text{orange}) = 0, Y(\text{lemon}) = 0$. Using step by step arguments as done in the example of 'Sum of Two Dice' above, find the following probabilities for our random variable Y:

jsMath

$$\begin{aligned} P(Y=0) &= P(\{\omega : Y(\omega) = \quad\}) \\ &= P\left(Y^{[-1]}(\quad)\right) \\ &= P(\{\quad , \quad\}) \end{aligned}$$

Please do this at home or with earphones! Watch the Khan Academy movie about random variables.



When we introduced the subject of probability, we said that many famous people had become interested in it from the point of view of gambling. Games of dice are one of the earliest forms of gambling (probably deriving from an even earlier game which involved throwing animal 'ankle' bones or *astragali*). Galileo was one of those who wrote about dice, including an important piece which explained what many experienced gamblers had sensed but had not been able to formalise - the difference between the probability of throwing a 9 and the probability of throwing a 10 with two dice. You should be able to see why this is from our map above. If you are interested you can read a translation (Galileo wrote in Latin) of Galileo's Sorpa le Scoperte Dei Dadi. This is also printed in a nice book, *Games, Gods and Gambling* by F.N. David (originally published 1962, newer editions now available).

You try

Example 1: fruit bowl experiments

We are going to use Sage with the fruit bowl example, and the random variable X to give each kind of fruit a numerical value: $X(\text{apple}) = 1, X(\text{orange}) = X(\text{lemon}) = 0$ This is a *discrete* random variable because it can only take a finite number of discrete values (in this case, either 1 or 0).

We have made our own random variable map object in Sage called `RV`. As with the Sage probability maps we looked at last week, it is based on a map or dictionary. We specify the sample the `samplespace` and `probabilities` and the random variable map, `MapX`, from the `samplespace` to the values taken by the random variable X).

```
# the sample space set as a list of outcomes
samplespace = ['apple', 'orange', 'lemon']
# the corresponding list of outcome probabilities
probabilities = [3/6, 2/6, 1/6]
# list of image values corresponding to the list of outcomes
# taken by the random variable X (1 if we pick an apple and 0 otherwise)
mapX = [1, 0, 0]
print "defined samplespace, probabilities, and mapX"
```

To make an `RV`, we can specify the lists for the sample space, the probabilities, and the random variable values associated with each outcome. Since there are three different lists here, we can make things clearer by actually saying what each list is. The `RV` we create in the cell below is going to be called X.

jsMath

```
X = RV(sspace=samplespace, probs=probabilities, values=mapX) # this random
variable will be called X
X # disclose the representation of the random variable X
```

(You don't have to worry about how RV works: it is our 'home-made' class for you to try out.)

We can get probabilities using the syntax `X.P(x)` to find $P(X = x)$. 'Syntax' is a way of saying how you instruct the computer to do what you want: the syntax you use for the computer maybe different to the way that you'd write the expressions in your lecture notes.

```
X.P(1)     # find the probability that X is 1
```

```
X.explainLastCalc()     # print out the values used in the calculation of
the probability that X = 1
```

You have seen that different random variables can be defined on the same **probability space**, i.e., the sample space and the associated probability map, depending on how the outcomes are mapped to real values taken by the random variable. Usually there is some good experimental or mathematical reason for the particular random variable (i.e., event-to-value-mappings) that we use. In the experiment we just did we could have been an experimenter particularly interested in citrus fruit but not concerned with what particular kind of citrus it is.

On the other hand, what if we want to differentiate between each fruit in the sample space? Then we could give each fruit-outcome a different value.

```
# the sample space set as a list of outcomes
samplespace = ['apple', 'orange', 'lemon']
# the corresponding list of outcome probabilities
probabilities = [3/6, 2/6, 1/6]
# list of image values corresponding to the list of outcomes
# map for another random variable is 1, 2, 3 if the fruit we pick is apple,
orange or lemon, respectively.
mapZ = [1, 2, 3]
print "defined sample space, probabilities, and mapZ"
```

```
Z = RV(sspace=samplespace, probs=probabilities, values=mapZ) # this random
variable will be called Z
Z # disclose the representation of the random variable Z
```

```
Z.P(1), Z.P(2), Z.P(3) # find the probability that Z=1, Z=2, Z=3
```

```

```

Example 2: Coin toss experiments

An experiment that is used a lot in examples is the coin toss experiment. If we toss a fair coin once, the sample space is either head (denoted here as H) or tail (T). The probability of a head is a half and the probability of tail is a half.

We can do the probability map for this as one of our ProbyMap objects.

```

```

jsMath

```
samplespace = ['H', 'T']     # sample space is the result of one coin toss
probabilities = [1/2,1/2]     # probabilities for a fair coin
probMapCoinToss = ProbyMap(sspace = samplespace, probs=probabilities)
# disclose the representation of the probability map from a one-coin toss
sample space to the probabilities
probMapCoinToss
```

Let's have a random variable called `oneHead` that takes the value 1 if we get one head, 0 otherwise and simulate this with an `RV` object.

.

```
mapOneHead = [1, 0]     # map for a random variable is 1 if the result is a
head, 0 if it is a tail
oneHead = RV(sspace=samplespace, probs=probabilities, values=mapOneHead) #
this random variable will be called OneHead
oneHead     # disclose the representation of the random variable OneHead
```

```
oneHead.P(1)     # find the probability that the random variable oneHead = 1
```

One toss is not very interesting. What if we we have a sample space that is the possible outcomes of two **independent** tosses of the same coin?

```
# we can 'square' our probability map from for one coin toss to get the map
for two tosses of the coin
probMapTwoCoinTosses = probMapCoinToss^2
probMapTwoCoinTosses     # disclose the map from the events for two coin
tosses to the probabilities (the probability map)
```

Tossing the coin twice and looking at the results of each toss in order is a different experiment to tossing the coin once.

We have a different set of possible outcomes, {HH, HT, TH, TT}. Note that the *order* matters: getting a head in the first toss and a tail in the second (HT) is a different event to getting a tail in the first toss and a head in the second (TH).

We can define a different random variable on this set of outcomes. Let's take an experimenter who is particularly interested in the number of heads we get in two tosses.

```
mapHeadsInTwoTosses = [2, 1, 0 ,1] # map for a random variable is the number
of heads in two tosses
headsInTwoTosses=RV(probmap=probMapTwoCoinTosses,
values=mapHeadsInTwoTosses) # this random variable will be called
HeadsInTwoTosses
headsInTwoTosses # disclose the representation of the random variable
```

As you can see, two different events in the sample space, a head and then a tail (HT) and a tail and then a head (TH) both give this random variable a value of 1. The event HH gives it a value 2 (2 heads) and the event TT gives it a value 0 (no heads).

Now we can try the probabilities.

```
headsInTwoTosses.P(0) # probability that headsInTwoTosses = 0
```

jsMath

```
headsInTwoTosses.P(1) # probability that headsInTwoTosses = 1
```

```
headsInTwoTosses.P(2) # probability that headsInTwoTosses = 2
```

```

```

The indicator function

The indicator function of an event $A \in \mathcal{F}$, denoted $\mathbf{1}_A$, is defined as follows:

$$\mathbf{1}_A(\omega) := \begin{cases} 1 & \text{if} \quad \omega \in A \\ 0 & \text{if} \quad \omega \notin A \end{cases}$$

The indicator function $\mathbf{1}_A$ is really an RV.

Example

"Will it rain tomorrow in the Southern Alps?" can be formulated as the RV given by the indicator function of the event "rain drops fall on the Southern Alps tomorrow". Can you imagine what the ω's in the sample space Ω can be?

Probability Mass Function

Recall that a **discrete** RV X takes on at most countably many values in $\mathbb{R}$.

The **probability mass function** (PMF) f of a discrete RV X is :

$$f(x) := P(X = x) = P\left(\{\omega : X(\omega) = x\}\right)$$

Bernoulli random variable

The Bernoulli RV is a θ-parameterised family of $\mathbf{1}_A$.

Take an event A. The parameter θ (pronounced 'theta) denotes the probability that "A occurs", i.e., $P(A) = \theta$.

The indicator function $\mathbf{1}_A$ of "A occurs" is the $Bernoulli(\theta)$ RV.

Model [$Bernoulli(\theta)$]

Given a parameter $\theta \in [0, 1]$, the probability mass function (PMF) for the $Bernoulli(\theta)$ RV X is:

$$f(x;\theta) = \theta^x(1-\theta)^{1-x}\mathbf{1}_{\{0,1\}}(x) = \begin{cases} \theta & \text{if x=1,} \\ 1-\theta & \text{if x=0,} \\ 0 & \text{otherwise} \end{cases}$$

and its DF is:

$$F(x;\theta) = \begin{cases} 1 & \text{if } 1 \le x, \\ 1-\theta & \text{if } 0 \le x < 1, \\ 0 & \text{otherwise} \end{cases}$$

We emphasise the dependence of the probabilities on the parameter θ by specifying it following the semicolon in the argument for f and F and by subscripting the probabilities, i.e. $P_\theta(X = 1) = \theta$ and $P_\theta(X = 0) = 1 - \theta$.

For loops

jsMath

'**For loops**' are a very useful way that most computer programming languages provide to allow us to repeat the same action. In Sage, you can use a for loop on a list to just go through the elements in the list, doing something with each element in turn.

The SAGE syntax for a 'for loop' is:

1. Start with the keyword `for`
2. Then we use a kind of temporary variable which is going to take each value in the list, one by one, in order. In the example below we use the name `x` for this variable and write `for x in myList`.
3. After we have specified what we are looping through, we end the line with a colon `:` before continuing to the next line.
4. Now, we are ready to write the *body* of the for loop. In the body, we put whatever we want to actually do with each value as we loop through. Remember when we defined a function, and the function body was indented? It's the same here: the body is a *block* of code which is indented (the Sage standard indentation for block is 4 spaces).
5. Indicate to Sage when your for loop has ended by moving the beginning of the line following the for loop back so that is it aligned with the `for` which started the whole loop.

The cell below gives a very simple example using a list.

```
# a simple routine to print every number in a list
myList = [1, 3, 6, 4, 2]
for x in myList:            # this statement will just go through the list
in order.
                            # Note the :
                            # each element in turn is assigned to the
variable x
    print x                 # body of the for loop (just one line in this
case)
```

If we wanted to do this for any list, we could write a simple function that takes any list we give it and puts it through a for loop, like the function below.

```
def myPrintListFunc(ll):                                    # start defining the
function
    '''A simple function to print a given list.'''      # the docstring for
the function
    for x in ll:                                            # body of the
function starts
        print x                                             # body of loop is
indented again
```

Notice that we start indenting with 4 spaces when we write the function body. Then, when we have the for loop inside the function body, we just indent the body of the for loop again.

Sage needs the indentation to help it to know what is in a function, or a for loop, and what is outside, but indentation also helps us as programmers to be able to look at a piece of code and easily see what is going on.

Let's try our function on another list.

```
anotherList = [20, 24, 46]
myPrintListFunc(anotherList)
```

jsMath

We have just programmed a basic for loop with a list. We can do much more than this with for loops, but this covers the essentials that we need to know. The important thing to remember is that whenever you have a list, you can easily write a for loop to go through each element in turn and do something with it.

You try

Example 3: For loops

Try first assigning the value 0 to a variable named `mySum` and then making yourself a list (you pick what it is called and what values it contains) and then looping through the list, adding each element in the list to `mySum`. When the loop has finished, the value of `mySum` will be the *accumulated* value of all the elements in the list.

What about defining a function to accumulate the values in a list? Remember to give your function a good name, and include the docstring to tell everyone what it does.

Try out your function!

A for loop can be used on more than just a list. For example, try a loop with the set S we make below - you can try making a loop to print the elements in the set one by one, as we did above, or to accumulate them, or do anything else you think is sensible ....

```
S=set([5,10,15, 70])     # make the set S
S             # display the set S
```

Loop over the set and do something.

You can even use a for loop on a string like "thisisastring", but this is not as useful for us as being able to use a for loop on a list or set.

We can use the `range` function we met last week to make a for loop that will do something a specified number of times. Remind yourself about the `range` function:

```
counter = range(10)    # reminder about range
counter
```

jsMath

Now let's use the counter idea do a specified number of rolls of the die that we can simulate with `randint(1,6)`. Notice that here the actual value of the elements in the list is not being used directly: the list is being used like a counter to make sure that we do something a specified number of times.

```
for c in range(10):
    dieResult = randint(1,6)
    print "Number on die is ", dieResult
```

```

```

Conditional statements

A **conditional statement** is also known as an *if* statement or *if-else* statement. And it's basically as simple as that: *if* [some condition] is true *then* do [something]. To make it more powerful, we can also say *else* (ie, if not), then do [a different something].

The if statement syntax is a way of putting into a programming language a way of thinking that we use every day: E.g. "if it is raining, I'll take the bus to uni; else I'll walk".

You'll notice that when we have an if statement, what we do depends on whether the condition we specify is true or false (i.e. is *conditional on* this expression). This is why if statements are called *conditional* statements.

When we say "depends on whether the condition we specify is true or false", in SAGE terms we are talking about whether the expression we specify as our condition evaluates to the Boolean true or the Boolean false. Remember those Boolean values, `true` and `false`, that we talked about in Lab 1?

The SAGE syntax for a conditional statement including if and else clauses is explained below:

1. Start with the keyword `if`
2. Immediately after keyword `if` we specify the condition that determines what happens next. The condition therefore has to be the kind of *truth statement* we talked about in Lab 1, an expression which SAGE can *evaluate* as either Boolean `true` or Boolean `false`. The examples below will make this a bit clearer.
3. After we have specified the condition, we end the line with a colon `:` before continuing to the next line.
4. Now, immediately below the line that starts `if`, we give the code that says what happens if the condition we specified is true. This block of code is indented so that SAGE can recognise that it should do everything in that block if the condition is true. If the condition is false, SAGE will not go into this indented block to execute any of this code but instead look for the next line which is aligned with the original `if`, i.e. the next line which is *not* part of the if-block.
5. In some situations we only want do something if the condition is true, and there is nothing else we want to do if the condition is false. In this situaion all we need is the if-block of code described above, the part that is executed only if the condition is true. Then we can carry on with the rest of the program. An everyday example would be "if it is cold I will wear gloves".
6. In some other situations, however, we also want to specify something to happen when the condition is false - else-block. We specify this by following the if-block with the keyword `else`, aligned with the `if` so that SAGE knows this is where it jumps to when the if condition is false. The keyword `else` is also followed by a colon `:`
7. Immediately below the line that starts `else`, we say what happens if the condition we specified is false. Again, this block of code (the else-block) is indented so that SAGE can recognise that it should do everything in that block if the condition is false.

Note that SAGE will *either* execute the code in the if-block *or* will execute the code in the else-block. What happens, i.e. which block gets executed, depends on whether the condition evaluates to true or false.

Let's set up a nice simple conditional statement and see what happens. We want take two variables, named x and jsMath

print something out only if x is greater than y. The condition is 'x > y', and you can see that this evaluates to either true or false, depending on the values of x and y.

```
x = 50                              # assign the value 50 to the variable
called x
y = 40                              # assign the value 40 to the variable
called y
if x > y:                           # does the expression 'x > y' evaluate to
true or false
    print "x is greater than y"     # this indented code is only executed if
the condition is true
print "This is the end of my first conditional statement"  # this code is
not indented
```

We can *nest* conditional statements so that one whole conditional statement is in the if-block (or else-block) of another one.

```
x = 50                              # assign a value to a variable called x
y = 40                              # assign a value to a variable called y
z = 60                              # assign a value to a variable called z
if x > y:                           # does the expression 'x > y' evaluate to
true or false
    print "x is greater than y"     # this indented code is only executed if
the condition is true
    if z > x:                       # nested if does the expression 'z > x'
evaluate to true or false
        print "and z is greater than z"    # only executed if both
conditions are true

print "This is the end of my nested conditional statement"  # this code is
not indented
```

You try

Example 4: Conditional statements

The cell above only did something if the condition was true. Now let's try if and else. This is a more complicated example and you might want to come back to it to make sure you understand what is going on. Try assigning different values to the variable `myAge` and see what happens when you evaluate the cell.

```
myAge = 22;             # assign the value 50 to the variable called x
myResult = ''           # myResult is an empty string waiting to be filled in
if myAge > 40:
    print "------- code in the if block is being executed"
    myResult = "You are old enough to know better . . ."
else:
    print "------- code in the else block is being executed"
    myResult = "You are young enough to dream . . ."
print "------- we are now back in the main flow of the program"
print myResult          # the value of myResult will depend on what happened
above
```

jsMath

We could also define a function which uses if and else. Let us define a function called myMaxFunc which takes two variables x and y as parameters. Note how we indent once for the body of the function, and again when we want to indicate what is in the if-block and else-block.

```
def myMaxFunc(x, y):
    '''A function to return the maximum of x and y.'''  # a one-line
docstring

    if x > y:                # use an if-else statement to test if x
greater than y
        retvalue = x         # if-block code is indented again
    else:
        retvalue = y         # else-block code is indented again
```

SAGE language note: There is of course a perfectly good max() function in SAGE which does the same thing - this is just a convenient example of the use of if and else.

Now we try our function out with some variables. Try using different values for firstNumber and secondNumber to test it.

```
firstNumber = 20
secondNumber = 40
myMaxFunc(firstNumber, secondNumber)
```

Finally, lets look at something to bring together for loops and conditional statements. We have seen how we can use a for loop to simulate throwing a die a specified number of times. We could add a conditional statement to count how many times we get a certain number on the die. Try altering the values of `resultOfInterest` and `numberOfThrows` and see what happens. Note that being able to find and alter values in your code easily is part of the benefits of using variable names.

```
resultOfInterest = 5  # specify what number we will count occurrences of
countNumber = 0       # to accumulate the count of the occurrences
numberOfThrows = 100  # how many throws to simulate
for c in range(numberOfThrows):    # loop numberOfThrows times
    dieNumber = randint(1,6)
    if dieNumber == resultOfInterest:
        countNumber = countNumber + 1        # increment for each occurrence
print "You got the number", resultOfInterest, countNumber, "times"
```

To get even fancier, we could set up a map and count the occurences of every number 1, 2, ... 6. **Make sure that you understand what we are doing here**. We are using a dictionary with (key, value) pairs to associate a count with each number that could come up on the die (number on die, count of number on die). Notice that we do not have to use the conditional statement (if ...) because we can access the *value* by using the *key* that is a particular result of a roll of the die.

Try altering the `numberOfThrows` and see what happens.

jsMath

```
countMap = {1:0, 2:0, 3:0, 4:0, 5:0, 6:0}      # start with a dictionary
with all count values 0
numberOfThrows = 100                            # number of throws to make
for c in range(numberOfThrows):                 # loop numberOfThrows times
    dieNumber = randint(1,6)
    countMap[dieNumber] = countMap[dieNumber]+1

countMap                                        # disclose the final countMap
```

Earlier, we looked at the probability of the event $B|A$ when we toss two dice and A is the event that the sum of the two numbers on the dice is 6 and B is the event that the first die is a 2. We worked out that $P(B|A) = \frac{1}{5}$. See if you can do another version of the code we have above to simulate throwing two dice a specified number of times, and use two *nested* conditional statements to accumulate a count of how many times the sum of the two numbers is 6 and how many out of those times the first die is a 2. When the loop has finished, you could print out or disclose the proportion

$$\frac{\text{number of times sum is 6 and first die is 2}}{\text{number of times sum is 6}}$$

```

```

Example 5: More coin toss experiments

If you have time, try the three-coin-toss and four-coin-toss examples below. These continue from Example 2 above.

Another new experiment: the outcomes of three tosses of the coin.

```
samplespace = ['H', 'T']     # sample space is the result of one coin toss
probabilities = [1/2,1/2]     # probabilities for a fair coin
probMapCoinToss = ProbyMap(sspace = samplespace, probs=probabilities)
# disclose the representation of the probability map from a one-coin toss
sample space to the probabilities
probMapCoinToss
probMapThreeCoinTosses = probMapCoinToss^3     # 'cubing' the probability
map from for one coin toss to get the map for three tosses of the coin
probMapThreeCoinTosses
```

The random variable we define is the number of heads we get in three tosses.

```
mapHeadsInThreeTosses = [2, 1, 3, 1, 1, 0, 2, 2] # map for a random variable
is the number of heads in three tosses
X=RV(probmap=probMapThreeCoinTosses, values=mapHeadsInThreeTosses) # this
random variable will be called X (easier than headsInThreeTosses!)
X # disclose the representation of the random variable X
```

```
X.P(3)     # find the probability that X = 3 (X is the number of heads in
three tosses of the coin)
```

If you are not bored yet, try a four-tosses experiment where we define a random variable which takes value 1 when the outcome includes exactly two heads, and 0 otherwise.

```
probMapFourCoinTosses = probMapCoinToss^4     # the probability map for 4
independent tosses of the coin
probMapFourCoinTosses
```

jsMath

```
mapExactlyTwoHeadsInFourTosses = [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
1, 0] # map for a r.v. which takes value 1 if two heads, 0 otherwise
Y=RV(probmap=probMapFourCoinTosses, values=mapExactlyTwoHeadsInFourTosses) #
this random variable will be called Y
Y # disclose the representation of the random variable Y
```

```
Y.P(0) # find the probability that we don't get exactly 2 heads in four
```

```
Y.explainLastCalc() # see the calculation
```

```

```

```
%hide
```

```
%hide
```

jsMath