Chapter 0

The Function (and other mathematical and computational preliminaries)

> Later generations will regard
> Mengenlehre [set theory] as a disease
> from which one has recovered.
> attributed to Poincáre

The basic mathematical concepts that inform our study of vectors and matrices are sets, sequences (lists), functions, and probability theory.

This chapter also includes an introduction to Python, the programming language we use to (i) model the mathematical objects of interest, (ii) write computational procedures, and (iii) carry out data analyses.

0.1 Set terminology and notation

The reader is likely to be familiar with the idea of a *set*, a collection of mathematical objects in which each object is considered to occur at most once. The objects belonging to a set are its *elements*. We use curly braces to indicate a set specified by explicitly enumerating its elements. For example, $\{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$ is the set of suits in a traditional deck of cards. The order in which elements are listed is not significant; a set imposes no order among its elements.

The symbol $\in$ is used to indicate that an object belongs to a set (equivalently, that the set *contains* the object). For example, $\heartsuit \in \{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$.

One set S_1 is *contained in* another set S_2 (written $S_1 \subseteq S_2$) if every element of S_1 belongs to S_2. Two sets are equal if they contain exactly the same elements. A convenient way to prove that two sets are equal consists of two steps: (1) prove the first set is contained in the second, and (2) prove the second is contained in the first.

A set can be infinite. In Chapter 1, we discuss the set $\mathbb{R}$, which consists of all real numbers, and the set $\mathbb{C}$, which consists of all complex numbers.

If a set S is not infinite, we use $|S|$ to denote its *cardinality*, the number of elements it contains. For example, the set of suits has cardinality 4.

0.2 Cartesian product

One from column A, one from column B.

The *Cartesian product* of two sets A and B is the set of all pairs (a, b) where $a \in A$ and $b \in B$.

Example 0.2.1: For the sets $A = \{1, 2, 3\}$ and $B = \{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$, the Cartesian product is

$$\{(1, \heartsuit), (2, \heartsuit), (3, \heartsuit), (1, \spadesuit), (2, \spadesuit), (3, \spadesuit), (1, \clubsuit), (2, \clubsuit), (3, \clubsuit), (1, \diamondsuit), (2, \diamondsuit), (3, \diamondsuit)\}$$

Quiz 0.2.2: What is the cardinality of $A \times B$ in Example 0.2.1 (Page 2)?

Answer

$|A \times B| = 12$.

Proposition 0.2.3: For finite sets A and B, $|A \times B| = |A| \cdot |B|$.

Quiz 0.2.4: What is the cardinality of $\{1, 2, 3, \ldots, 10, J, Q, K\} \times \{\heartsuit, \spadesuit, \clubsuit, \diamondsuit\}$?

Answer

We use Proposition 0.2.3. The cardinality of the first set is 13, and the cardinality of the second set is 4, so the cardinality of the Cartesian product is $13 \cdot 4$, which is 52.

The Cartesian product is named for René Descartes, whom we shall discuss in Chapter 6.

0.3 The function

Mathematicians never die—they just lose function.

Loosely speaking, a function is a rule that, for each element in some set D of possible inputs, assigns a possible output. The output is said to be the *image* of the input under the function

and the input is a *pre-image* of the output. The set D of possible inputs is called the *domain* of the function.

Formally, a *function* is a (possibly infinite) set of pairs (a, b) no two of which share the same first entry.

Example 0.3.1: The doubling function with domain $\{1, 2, 3, \ldots\}$ is

$$\{(1, 2), (2, 4), (3, 6), (4, 8), \ldots\}$$

The domain can itself consist of pairs of numbers.

Example 0.3.2: The multiplication function with domain $\{1, 2, 3, \ldots\} \times \{1, 2, 3, \ldots\}$ looks something like this:

$$\{((1, 1), 1), ((1, 2), 2), \ldots, ((2, 1), 2), ((2, 2), 4), ((2, 3), 6), \ldots\}$$

For a function named f, the image of q under f is denoted by $f(q)$. If $r = f(q)$, we say that *q maps to r under f*. The notation for "q maps to r" is $q \mapsto r$. (This notation omits specifying the function; it is useful when there is no ambiguity about which function is intended.)

It is convenient when specifying a function to specify a *co-domain* for the function. The co-domain is a set from which the function's output values are chosen. Note that one has some leeway in choosing the co-domain since not all of its members need be outputs.

The notation

$$f : D \longrightarrow F$$

means that f is a function whose domain is the set D and whose *co-domain* (the set of possible outputs) is the set F. (More briefly: "a function from D to F", or "a function that maps D to F.")

Example 0.3.3: Caesar was said to have used a cryptosystem in which each letter was replaced with the one three steps forward in the alphabet (wrapping around for X,Y, and Z).[a] Thus the plaintext MATRIX would be encrypted as the cyphertext PDWULA. The function that maps each plaintext letter to its cyphertext replacement could be written as

$$A \mapsto D, B \mapsto E, C \mapsto F, D \mapsto G, W \mapsto Z, X \mapsto A, Y \mapsto B, Z \mapsto C$$

This function's domain and co-domain are both the alphabet $\{A, B, \ldots, Z\}$.

[a] Some imaginary historians have conjectured that Caesar's assasination can be attributed to his use of such a weak cryptosystem.

Example 0.3.4: The cosine function, *cos*, maps from the set of real numbers (indicated by $\mathbb{R}$)

to the set of real numbers. We would therefore write

$$\cos : \mathbb{R} \longrightarrow \mathbb{R}$$

Of course, the outputs of the cos function do not include all real numbers, only those between -1 and 1.

The *image* of a function f is the set of images of all domain elements. That is, the image of f is the set of elements of the co-domain that actually occur as outputs. For example, the image of Caesar's encryption function is the entire alphabet, and the image of the cosine function is the set of numbers between -1 and 1.

Example 0.3.5: Consider the function *prod* that takes as input a pair of integers greater than 1 and outputs their product. The domain (set of inputs) is the set of pairs of integers greater than 1. We choose to define the co-domain to be the set of all integers greater than 1. The image of the function, however, is the set of *composite* integers since no domain element maps to a prime number.

0.3.1 Functions versus procedures, versus computational problems

There are two other concepts that are closely related to functions and that enter into our story, and we must take some care to distinguish them.

- A *procedure* is a precise description of a computation; it accepts *inputs* (called *arguments*) and produces an output (called the *return value*).

  Example 0.3.6: This example illustrates the Python syntax for defining procedures:

  ```
  def mul(p,q): return p*q
  ```

  In the hope of avoiding confusion, we diverge from the common practice of referring to procedures as "functions".

- A *computational problem* is an input-output specification that a procedure might be required to satisfy.

  Example 0.3.7:
  - *input:* a pair (p, q) of integers greater than 1
  - *output:* the product pq

  Example 0.3.8:
  - *input:* an integer m greater than 1
  - *output:* a pair (p, q) of integers whose product is m

How do these concepts differ from one another?

- Unlike a procedure, a function or computational problem does not give us any idea how to compute the output from the input. There are often many different procedures that satisfy the same input-output specification or that implement the same function. For integer multiplication, there is ordinary long multiplication (you learned this in elementary school), the Karatsuba algorithm (used by Python for long-integer multiplication), the faster Schönhage-Strassen algorithm (which uses the Fast Fourier Transform, discussed in Chapter 10), and the even faster Fürer algorithm, which was discovered in 2007.
- Sometimes the same procedure can be used for different functions. For example, the Python procedure `mul` can be used for multiplying negative integers and numbers that are not integers.
- Unlike a function, a computational problem need not specify a unique output for every input; for Example 0.3.8 (Page 4), if the input is 12, the output could be $(2, 6)$ or $(3, 4)$ or $(4, 3)$ or $(6, 2)$.

0.3.2 The two computational problems related to a function

All the king's horses and all the king's men
Couldn't put Humpty together again.

Although *function* and *computational problem* are defined differently, they are clearly related. For each function f, there is a corresponding computational problem:

The forward problem: Given an element a of f's domain, compute $f(a)$, the image of a under f.

Example 0.3.7 (Page 4) is the computational problem that corresponds in this sense to the function defined in Example 0.3.2 (Page 3).

However, there is another computational problem associated with a function:

The backward problem: Given an element r of the co-domain of the function, compute any pre-image (or report that none exists).

How very different are these two computational problems? Suppose there is a procedure $P(x)$ for computing the image under f of any element of the domain. An obvious procedure for computing the pre-image of r is to iterate through each of the domain elements q, and, one by one, apply the procedure $P(x)$ on q to see if the output matches r.

This approach seems ridiculously profligate—even if the domain is finite, it might be so large that the time required for solving the pre-image problem would be much more than that for $P(x)$—and yet there is no better approach that works for all functions.

Indeed, consider Example 0.3.7 (Page 4) (integer multiplication) and Example 0.3.8 (Page 4) (integer factoring). The fact that integer multiplication is computationally easy while integer factoring is computationally difficult is in fact the basis for the security of the RSA cryptosystem, which is at the heart of secure commerce over the world-wide web.

And yet, as we will see in this book, finding pre-images can be quite useful. What is one to do?

In this context, the generality of the concept of function is also a weakness. To misquote *Spiderman*,

With great generality comes great computational difficulty.

This principle suggests that we consider the pre-image problem not for arbitrary functions but for specific families of functions. Yet here too there is a risk. If the family of functions is too restrictive, the existence of fast procedures for solving the pre-image problem will have no relevance to real-world problems. We must navigate between the Scylla of computational intractability and the Charybdis of inapplicability.

In linear algebra, we will discover a sweet spot. The family of *linear functions*, which are introduced in Chapter 4, manage to model enough of the world to be immensely useful. At the same time, the pre-image problem can be solved for such functions.

0.3.3 Notation for the set of functions with given domain and co-domain

For sets D and F, we use the notation F^D to denote all functions from D to F. For example, the set of functions from the set W of words to the set $\mathbb{R}$ of real numbers is denoted $\mathbb{R}^W$.

This notation derives from a mathematical "pun":

Fact 0.3.9: For any finite sets D and F, $|D^F| = |D|^{|F|}$.

0.3.4 Identity function

For any domain D, there is a function $\mathrm{id}_D : D \longrightarrow D$ called the *identity function* for D, defined by

$$\mathrm{id}_D(d) = d$$

for every $d \in D$.

0.3.5 Composition of functions

The operation *functional composition* combines two functions to get a new function. We will later define matrix multiplication in terms of functional composition. Given two functions $f : A \longrightarrow B$ and $g : B \longrightarrow C$, the function $g \circ f$, called the composition of g and f, is a function whose domain is A and its co-domain is C. It is defined by the rule

$$(g \circ f)(x) = g(f(x))$$

for every $x \in A$.

If the image of f is not contained in the domain of g then $g \circ f$ is not a legal expression.

Example 0.3.10: Say the domain and co-domains of f and g are $\mathbb{R}$, and $f(x) = x + 1$ and $g(y) = y^2$. Then $g \circ f(x) = (x + 1)^2$.

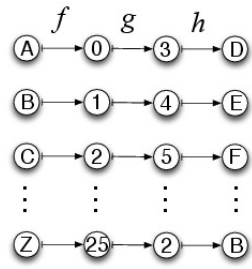


Figure 1: This figure represents the composition of the functions f, g, h. Each function is represented by arrows from circles representing its domain to circles representing its co-domain. The composition of the three functions is represented by following three arrows.

Example 0.3.11: Define the function

$$f : \{A, B, C, \ldots, Z\} \longrightarrow \{0, 1, 2, \ldots, 25\}$$

by

$$A \mapsto 0, B \mapsto 1, C \mapsto 2, \cdots, Z \mapsto 25$$

Define the function g as follows. The domain and co-domain of g are both the set $\{0, 1, 2, \ldots, 25\}$, and $g(x) = (x + 3) \bmod 26$. For a third function h, the domain is $\{0, ...25\}$ and the co-domain is $\{A, ..., Z\}$, and $0 \mapsto A$, $1 \mapsto B$, etc. Then $h \circ (g \circ f)$ is a function that implements the Caesar cypher as described in Example 0.3.3 (Page 3).

For building intuition, we can use a diagram to represent composition of functions with finite domains and co-domains. Figure 1 depicts the three functions of Example 0.3.11 (Page 7) being composed.

0.3.6 Associativity of function composition

Next we show that composition of functions is *associative*:

Proposition 0.3.12 (Associativity of composition): For functions f, g, h,

$$h \circ (g \circ f) = (h \circ g) \circ f$$

if the compositions are legal.

Proof

Let x be any member of the domain of f.

$$\begin{aligned}(h \circ (g \circ f))(x) &= h((g \circ f)(x)) \text{ by definition of } h \circ (g \circ f)) \\ &= h(g(f(x)) \text{ by definition of } g \circ f \\ &= (h \circ g)(f(x)) \text{ by definition of } h \circ g \\ &= ((h \circ g) \circ f)(x) \text{ by definition of } (h \circ g) \circ f\end{aligned}$$

□

Associativity means that parentheses are unnecessary in composition expression: since $h \circ (g \circ f)$ is the same as $(h \circ g) \circ f$, we can write either of them as simply $h \circ g \circ f$.

0.3.7 Functional inverse

Let us take the perspective of a lieutenant of Caesar who has received a cyphertext: PDWULA. To obtain the plaintext, the lieutenant must find for each letter in the cyphertext the letter that maps to it under the encryption function (the function of Example 0.3.3 (Page 3)). That is, he must find the letter that maps to P (namely M), the letter that maps to D (namely A), and so on. In doing so, he can be seen to be applying *another* function to each of the letters of the cyphertext, specifically the function that reverses the effect of the encryption function. This function is said to be the *functional inverse* of the encryption function.

For another example, consider the functions f and h in Example 0.3.11 (Page 7): f is a function from $\{A, \ldots, Z\}$ to $\{0, \ldots, 25\}$ and h is a function from $\{0, \ldots, 25\}$ to $\{A, \ldots, Z\}$. Each one reverses the effect of the other. That is, $h \circ f$ is the identity function on $\{A, \ldots, Z\}$, and $f \circ h$ is the identity function on $\{0, \ldots, 25\}$. We say that h is the functional inverse of f. There is no reason for privileging f, however; f is the functional inverse of h as well.

In general,

Definition 0.3.13: We say that functions f and g are *functional inverses* of each other if

- $f \circ g$ is defined and is the identity function on the domain of g, and
- $g \circ f$ is defined and is the identity function on the domain of f.

Not every function has an inverse. A function that has an inverse is said to be *invertible*. Examples of noninvertible functions are shown in Figures 2 and 3

Definition 0.3.14: Consider a function $f : D \longrightarrow F$. We say that f is *one-to-one* if for every $x, y \in D$, $f(x) = f(y)$ implies $x = y$. We say that f is *onto* if, for every $z \in F$, there exists $x \in D$ such that $f(x) = z$.

Example 0.3.15: Consider the function *prod* defined in Example 0.3.5 (Page 4). Since a prime

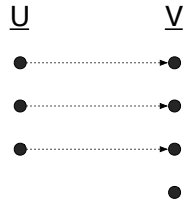


Figure 2: A function $f : U \to V$ is depicted that is not onto, because the fourth element of the co-domain is not the image under f of any element

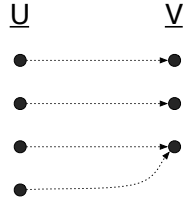


Figure 3: A function $f : U \to V$ is depicted that is not one-to-one, because the third element of the co-domain is the image under f of more than one element.

number has no pre-image, this function is not onto. Since there are multiple pairs of integers, e.g. $(2, 3)$ and $(3, 2)$, that map to the same integer, the function is also not one-to-one.

Lemma 0.3.16: An invertible function is one-to-one.

Proof

Suppose f is not one-to-one, and let x_1 and x_2 be distinct elements of the domain such that $f(x_1) = f(x_2)$. Let $y = f(x_1)$. Assume for a contradiction that f is invertible. The definition of inverse implies that $f^{-1}(y) = x_1$ and also $f^{-1}(y) = x_2$, but both cannot be true. □

Lemma 0.3.17: An invertible function is onto.

Proof

Suppose f is not onto, and let $\hat{y}$ be an element of the co-domain such that $\hat{y}$ is not the image of any domain element. Assume for a contradiction that f is invertible. Then $\hat{y}$ has

an image $\hat{x}$ under f^{-1}. The definition of inverse implies that $f(\hat{x}) = \hat{y}$, a contradiction. □

Theorem 0.3.18 (Function Invertibility Theorem): A function is invertible iff it is one-to-one and onto.

Proof

Lemmas 0.3.16 and 0.3.17 show that an invertible function is one-to-one and onto. Suppose conversely that f is a function that is one-to-one and onto. We define a function g whose domain is the co-domain of f as follows:

> For each element $\hat{y}$ of the co-domain of f, since f is onto, f's domain contains some element $\hat{x}$ for which $f(\hat{x}) = \hat{y}$; we define $g(\hat{y}) = \hat{x}$.

We claim that $g \circ f$ is the identity function on f's domain. Let $\hat{x}$ be any element of f's domain, and let $\hat{y} = f(\hat{x})$. Because f is one-to-one, $\hat{x}$ is the only element of f's domain whose image under f is $\hat{y}$, so $g(\hat{y}) = \hat{x}$. This shows $g \circ f$ is the identity function.

We also claim that $f \circ g$ is the identity function on g's domain. Let $\hat{y}$ be any element of g's domain. By the definition of g, $f(g(\hat{y})) = \hat{y}$. □

Lemma 0.3.19: Every function has at most one functional inverse.

Proof

Let $f : U \to V$ be an invertible function. Suppose that g_1 and g_2 are inverses of f. We show that, for every element $v \in V$, $g_1(v) = g_2(v)$, so g_1 and g_2 are the same function.

Let $v \in V$ be any element of the co-domain of f. Since f is onto (by Lemma 0.3.17), there is some element $u \in U$ such that $v = f(u)$. By definition of inverse, $g_1(v) = u$ and $g_2(v) = u$. Thus $g_1(v) = g_2(v)$. □

0.3.8 Invertibility of the composition of invertible functions

In Example 0.3.11 (Page 7), we saw that the composition of three functions is a function that implements the Caesar cypher. The three functions being composed are all invertible, and the result of composition is also invertible. This is not a coincidence:

Lemma 0.3.20: If f and g are invertible functions and $f \circ g$ exists then $f \circ g$ is invertible and $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$.

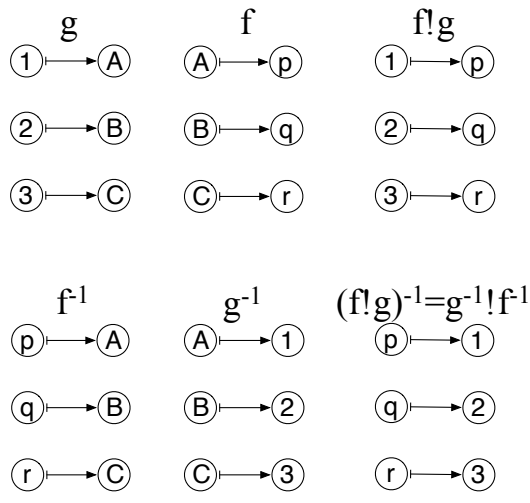


Figure 4: The top part of this figure shows two invertible functions f and g, and their composition $f \circ g$. Note that the composition $f \circ g$ is invertible. This illustrates Lemma 0.3.20. The bottom part of this figure shows g^{-1}, f^{-1} and $(f \circ g)^{-1}$. Note that $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$. This illustrates Lemma 0.3.20.

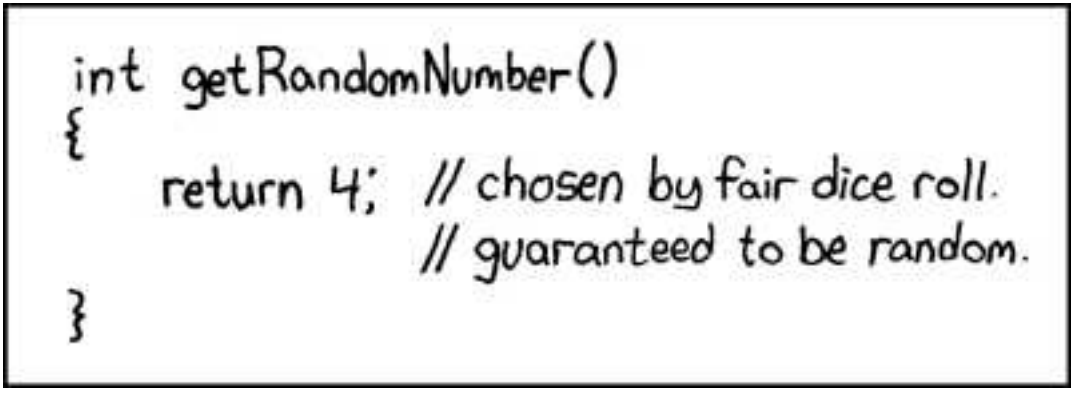
Problem 0.3.21: Prove Lemma 0.3.20.

Problem 0.3.22: Use diagrams like those of Figures 1, 2, and 3 to specify functions g and f that are a counterexample to the following:

False Assertion 0.3.23: Suppose that f and g are functions and $f \circ g$ is invertible. Then f and g are invertible.

□

0.4 Probability



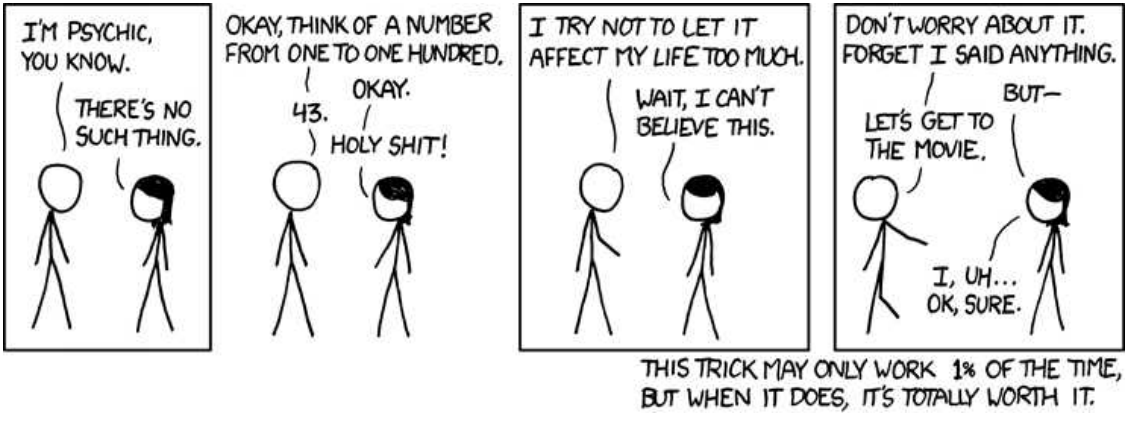
Random Number (`http://xkcd.com/221/`)

One important use of vectors and matrices arises in probability. For example, this is how they arise in Google's PageRank method. We will therefore study very rudimentary probability theory in this course.

In probability theory, nothing ever happens—probability theory is just about what *could* happen, and how likely it is to happen. Probability theory is a calculus of probabilities. It is used to make predictions about a hypothetical experiment. (Once something actually happens, you use *statistics* to figure out what it means.)

0.4.1 Probability distributions

A function $\Pr(\cdot)$ from a finite domain Ω to the set $\mathbb{R}^+$ of nonnegative reals is a *(discrete) probability distribution* if $\sum_{\omega\in\Omega}\Pr(\omega) = 1$. We refer to the elements of the domain as *outcomes*. The image of an outcome under $\Pr(\cdot)$ is called the *probability* of the outcome. The probabilities are supposed to be proportional to the *relative likelihoods* of outcomes. Here I use the term *likelihood* to mean the common-sense notion, and *probability* to mean the mathematical abstraction of it.



Psychic, `http://xkcd.com/628/`

Uniform distributions

For the simplest examples, all the outcomes are equally likely, so they are all assigned the same probabilities. In such a case, we say that the probability distribution is *uniform*.

Example 0.4.1: To model the flipping of a single coin, $\Omega = \{\text{heads}, \text{tails}\}$. We assume that the two outcomes are equally likely, so we assign them the same probability: $\Pr(\text{heads}) = \Pr(\text{tails})$. Since we require the sum to be 1, $\Pr(\text{heads}) = 1/2$ and $\Pr(\text{tails}) = 1/2$. In Python, we would write the probability distribution as

```
>>> Pr = {'heads':1/2, 'tails':1/2}
```

Example 0.4.2: To model the roll of a single die, $\Omega = \{1, 2, 3, 4, 5, 6\}$, and $\Pr(1) = \Pr(2) = \cdots = \Pr(6)$. Since the probabilities of the six outcomes must sum to 1, each of these probabilities must be $1/6$. In Python,

```
>>> Pr = {1:1/6, 2:1/6, 3:1/6, 4:1/6, 5:1/6, 6:1/6}
```

Example 0.4.3: To model the flipping of two coins, a penny and a nickel, $\Omega = \{HH, HT, TH, TT\}$, and each of the outcomes has the same probability, 1/4. In Python,

```
>>> Pr = {('H', 'H'):1/4, ('H', 'T'):1/4, ('T','H'):1/4, ('T','T'):1/4}
```

Nonuniform distributions

In more complicated situations, different outcomes have different probabilities.

Example 0.4.4: Let $\Omega = \{A, B, C, \ldots, Z\}$, and let's assign probabilities according to how likely you are to draw each letter at the beginning of a Scrabble game. Here is the number of tiles with each letter in Scrabble:

A	9	B	2	C	2	D	4
E	12	F	2	G	3	H	2
I	9	J	1	K	1	L	1
M	2	N	6	O	8	P	2
Q	1	R	6	S	4	T	6
U	4	V	2	W	2	X	1
Y	2	Z	1				

The likelihood of drawing an R is twice that of drawing a G, thrice that of drawing a C, and six times that of drawing a Z. We need to assign probabilities that are proportional to these

likelihoods. We must have some number c such that, for each letter, the probability of drawing that letter should be c times the number of copies of that letter.

$$\Pr[\text{drawing letter X}] = c \cdot \text{number of copies of letter X}$$

Summing over all letters, we get

$$1 = c \cdot \text{total number of tiles}$$

Since the total number of tiles is 95, we define $c = 1/95$. The probability of drawing an E is therefore 12/95, which is about .126. The probability of drawing an A is 9/95, and so on. In Python, the probability distribution is

```
{'A':9/95, 'B':2/95, 'C':2/95, 'D':4/95, 'E':12/95, 'F':2/95,
          'G':3/95, 'H':2/95, 'I':9/95, 'J':1/95, 'K':1/95,  'L':1/95,
          'M':2/95, 'N':6/95, 'O':8/95, 'P':2/95, 'Q':1/95,  'R':6/95,
          'S':4/95, 'T':6/95, 'U':4/95, 'V':2/95, 'W':2/95,  'X':1/95,
          'Y':2/95, 'Z':1/95}
```

0.4.2 Events, and adding probabilities

In Example 0.4.4 (Page 13), what is the probability of drawing a *vowel* from the bag?

A set of outcomes is called an *event*. For example, the event of drawing a vowel is represented by the set $\{A, E, I, O, U\}$.

Principle 0.4.5 (Fundamental Principle of Probability Theory): The probability of an event is the sum of probabilities of the outcomes making up the event.

According to this principle, the probability of a vowel is

$$9/95 + 12/95 + 9/95 + 8/95 + 4/95$$

which is $42/95$.

0.4.3 Applying a function to a random input

Now we think about applying a function to a random input. Since the input to the function is random, the output should also be considered random. Given the probability distribution of the input and a specification of the function, we can use probability theory to derive the probability distribution of the output.

Example 0.4.6: Define the function $f : \{1, 2, 3, 4, 5, 6\} \longrightarrow \{0, 1\}$ by

$$f(x) = \begin{cases} 0 & \text{if } x \text{ is even} \\ 1 & \text{if } x \text{ is odd} \end{cases}$$

Consider the experiment in which we roll a single die (as in Example 0.4.2 (Page 13)), yielding one of the numbers in $\{1, 2, 3, 4, 5, 6\}$, and then we apply $f(\cdot)$ to that number, yielding either a 0 or a 1. What is the probability function for the outcome of this experiment?

The outcome of the experiment is 0 if the rolled die shows 2, 4, or 6. As discussed in Example 0.4.2 (Page 13), each of these possibilies has probability $1/6$. By the Fundamental Principle of Probability Theory, therefore, the output of the function is 0 with probability $1/6 + 1/6 + 1/6$, which is $1/2$. Similarly, the output of the function is 1 with probability $1/2$. Thus the probability distribution of the output of the function is `{0: 1/2., 1:1/2.}`.

Quiz 0.4.7: Consider the flipping of a penny and a nickel, described in Example 0.4.3 (Page 13). The outcome is a pair (x, y) where each of x and y is `'H'` or `'T'` (heads or tails). Define the function

$$f : \{\text{('H', 'H') ('H', 'T'), ('T','H'), ('T','T')}\}$$

by

$$f((x, y)) = \text{the number of H's represented}$$

Give the probability distribution for the output of the function.

Answer

`{0: 1/4., 1:1/2., 2:1/4.}`

Example 0.4.8 (Caesar plays Scrabble): Recall that the function f defined in Example 0.3.11 (Page 7) maps A to 0, B to 1, and so on. Consider the experiment in which f is applied to a letter selected randomly according to the probability distribution described in Example 0.4.4 (Page 13). What is the probability distribution of the output?

Because f is an invertible function, there is one and only one input for which the output is 0, namely A. Thus the probability of the output being 0 is exactly the same as the probability of the input being A, namely $9/95$.. Similarly, for each of the integers 0 through 25 comprising the co-domain of f, there is exactly one letter that maps to that integer, so the probability of that integer equals the probability of that letter. The probability distribution is thus

```
{0:9/95., 1:2/95., 2:2/95., 3:4/95., 4:12/95., 5:2/95.,
          6:3/95., 7:2/95., 8:9/95., 9:1/95., 10:1/95.,  11:1/95.,
          12:2/95., 13:6/95., 14:8/95., 15:2/95., 16:1/95.,  17:6/95.,
          18:4/95., 19:6/95., 20:4/95., 21:2/95., 22:2/95.,  23:1/95.,
          24:2/95., 25:1/95.}
```

The previous example illustrates that, if the function is invertible, the probabilities are preserved: the probabilities of the various outputs match the probabilities of the inputs. It follows that, if the input is chosen according to a uniform distribution, the distribution of the output is

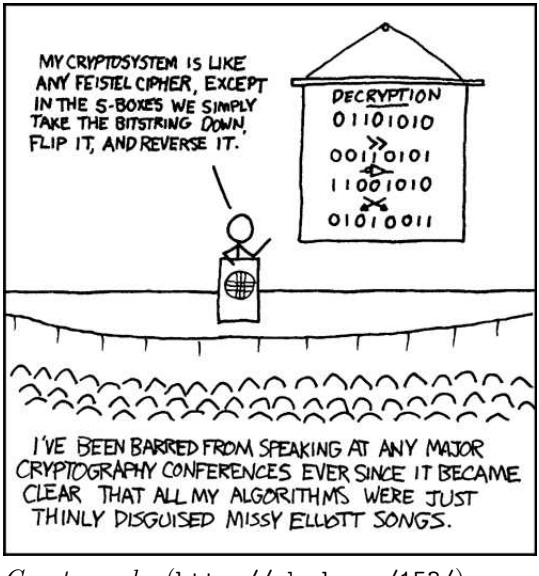
also uniform.

Example 0.4.9: In Caesar's Cyphersystem, one encrypts a letter by advancing it three positions. Of course, the number k of positions by which to advance need not be three; it can be any integer from 0 to 25. We refer to k as the *key*. Suppose we select the key k according to the uniform distribution on $\{0, 1, \ldots, 25\}$, and use it to encrypt the letter P. Let $w : \{0, 1, \ldots, 25\} \longrightarrow \{A, B, \ldots, Z\}$ be the the function mapping the key to the cyphertext:

$$\begin{aligned} w(k) &= h(f(P) + k \bmod 26) \\ &= h(15 + k \bmod 26) \end{aligned}$$

The function $w(\cdot)$ is invertible. The input is chosen according to the uniform distribution, so the distribution of the output is also uniform. Thus when the key is chosen randomly, the cyphertext is equally likely to be any of the twenty-six letters.

0.4.4 Perfect secrecy



Cryptography (`http://xkcd.com/153/`)

We apply the idea of Example 0.4.9 (Page 16) to some even simpler cryptosystems. A cryptosystem must satisfy two obvious requirements:

- the intended recipient of an encrypted message must be able to decrypt it, and
- someone for whom the message was not intended should *not* be able to decrypt it.

The first requirement is straightforward. As for the second, we must dispense with a misconception about security of cryptosystems. The idea that one can keep information secure by

not revealing the *method* by which it was secured is often called, disparagingly, *security through obscurity*. This approach was critiqued in 1881 by a professor of German, Jean-Guillame-Hubert-Victor-François-Alexandre-August Kerckhoffs von Niewenhof, known as August Kerckhoffs. The *Kerckhoffs Doctrine* is that *the security of a cryptosystem should depend only on the secrecy of the key used, not on the secrecy of the system itself.*

There is an encryption method that meets Kerchoffs' stringent requirement. It is utterly unbreakable if used correctly.[1] Suppose Alice and Bob work for the British military. Bob is the commander of some troops stationed in Boston harbor. Alice is the admiral, stationed several miles away. At a certain moment, Alice must convey a one-bit message p (the plaintext) to Bob: whether to attack by land or by sea (0=land, 1=sea). Their plan, agreed upon in advance, is that Alice will encrypt the message, obtaining a one-bit *cyphertext* c, and send the cyphertext c to Bob by hanging one or two lanterns (say, one lantern = 0, two lanterns = 1). They are aware that the fate of a colony might depend on the secrecy of their communication. (As it happens, a rebel, Eve, knows of the plan and will be observing.)

Let's go back in time. Alice and Bob are consulting with their cryptography expert, who suggests the following scheme:

Bad Scheme: Alice and Bob randomly choose k from $\{\clubsuit, \heartsuit, \spadesuit\}$ according to the uniform probability function $(\text{pr}(\clubsuit) = 1/3, \text{pr}(\heartsuit) = 1/3, \text{pr}(\spadesuit) = 1/3)$. Alice and Bob must both know k but must keep it secret. It is the *key*.
When it is time for Alice to use the key to encrypt her plaintext message p, obtaining the cyphertext c, she refers to the following table:

p	k	c
0	$\clubsuit$	0
0	$\heartsuit$	1
0	$\spadesuit$	1
1	$\clubsuit$	1
1	$\heartsuit$	0
1	$\spadesuit$	0

The good news is that this cryptosystem satisfies the first requirement of cryptosystems: it will enable Bob, who knows the key k and receives the cyphertext c, to determine the plaintext p. No two rows of the table have the same k-value and c-value.

The bad news is that this scheme leaks information to Eve. Suppose the message turns out to be 0. In this case, $c = 0$ if $k = \clubsuit$ (which happens with probability 1/3), and $c = 1$ if $k = \heartsuit$ or $k = \spadesuit$ (which, by the Fundamental Principle of Probability Theory, happens with probability 2/3). Thus in this case $c = 1$ is twice as likely as $c = 0$. Now suppose the message turns out to be 1. In this case, a similar analysis shows that $c = 0$ is twice as likely as $c = 1$.

Therefore, when Eve sees the cyphertext c, she learns something about the plaintext p. Learning c doesn't allow Eve to determine the value of p with certainty, but she can revise her estimate of the chance that $p = 0$. For example, suppose that, before seeing c, Eve believed $p = 0$ and $p = 1$ were equally likely. If she sees $c = 1$ then she can infer that $p = 0$ is twice as likely as $p = 1$. The exact calculation depends on Bayes' Rule, which is beyond the scope of this analysis

[1] For an historically significant occurence of the former Soviet Union failing to use it correctly, look up VENONA.

but is quite simple.

Confronted with this argument, the cryptographer changes the scheme simply by removing ♠ as a possible value for p.

Good Scheme: Alice and Bob randomly choose k from $\{\clubsuit, \heartsuit\}$ according to the uniform probability function $(\text{pr}(\clubsuit) = 1/2, \text{pr}(\heartsuit) = 1/2)$
When it is time for Alice to encrypt her plaintext message p, obtaining the cyphertext c, she uses the following table:

p	k	c
0	♣	0
0	♡	1
1	♣	1
1	♡	0

0.4.5 Perfect secrecy and invertible functions

Consider the functions

$$f_0 : \{\clubsuit, \heartsuit\} \longrightarrow \{0, 1\}$$

and

$$f_1 : \{\clubsuit, \heartsuit\} \longrightarrow \{0, 1\}$$

defined by

$$f_0(x) = \text{encryption of 0 when the key is } x$$

$$f_1(x) = \text{encryption of 1 when the key is } x$$

Each of these functions is invertible. Consequently, for each function, if the input x is chosen uniformly at random, the output will also be distributed according to the uniform distribution. This in turn means that the probability distribution of the output does not depend on whether 0 or 1 is being encrypted, so knowing the output gives Eve no information about which is being encrypted. We say the scheme achieves *perfect secrecy*.