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

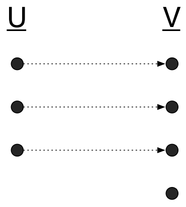


Figure 2: A function $f : U \to V$ is depicted that is not onto, because the fourth element of the co-domain is not the image under f of any element

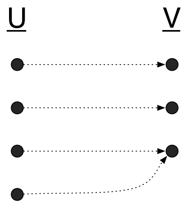


Figure 3: A function $f : U \to V$ is depicted that is not one-to-one, because the third element of the co-domain is the image under f of more than one element.

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 $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$.

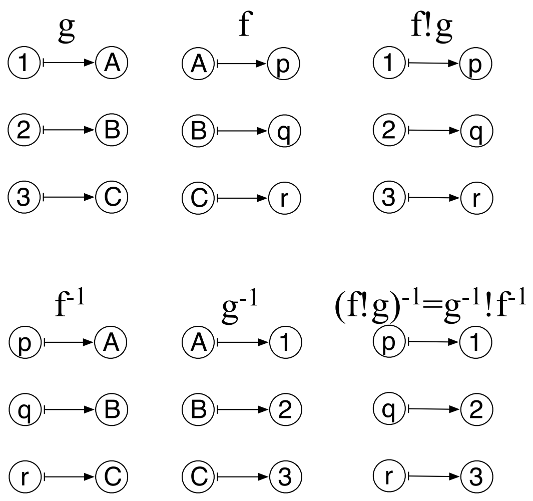


Figure 4: The top part of this figure shows two invertible functions f and g, and their composition $f \circ g$. Note that the composition $f \circ g$ is invertible. This illustrates Lemma 0.3.20. The bottom part of this figure shows g^{-1}, f^{-1} and $(f \circ g)^{-1}$. Note that $(f \circ g)^{-1}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

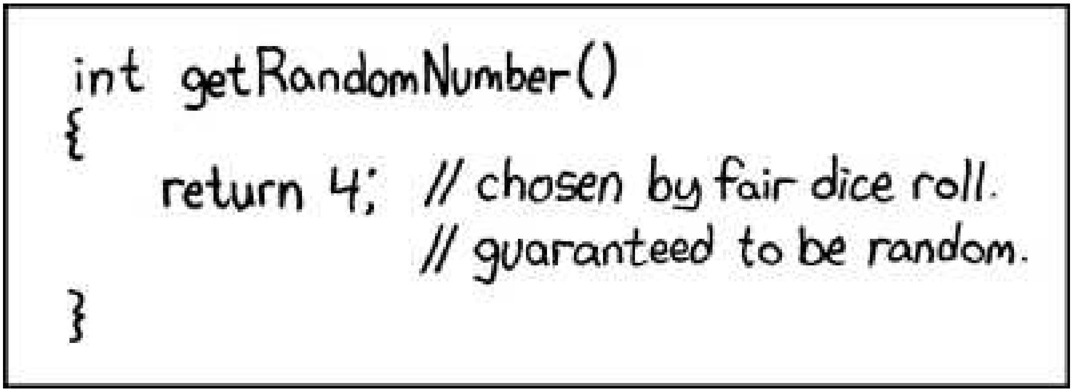
Problem 0.3.21: Prove Lemma 0.3.20.

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



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

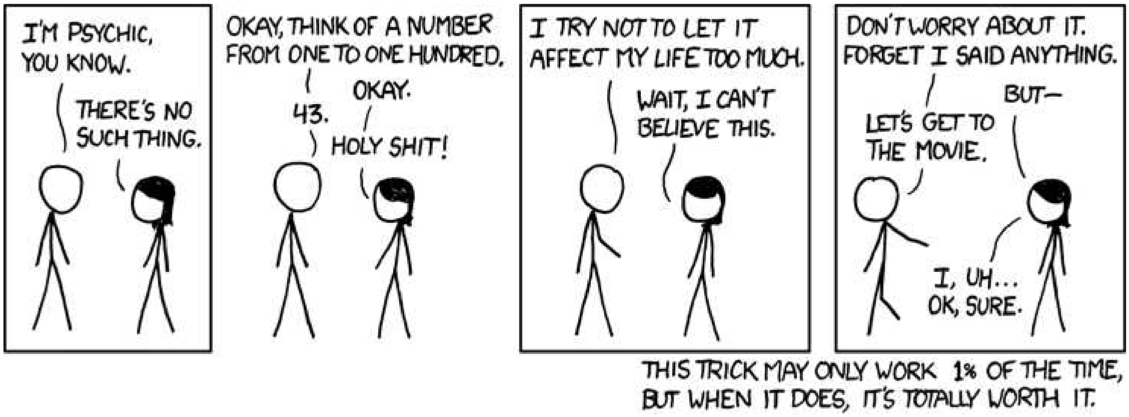
Random Number (`http://xkcd.com/221/`)

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.


I'M PSYCHIC, YOU KNOW.
THERE'S NO SUCH THING.
OKAY, THINK OF A NUMBER FROM ONE TO ONE HUNDRED.
OKAY.
43.
HOLY SHIT!
I TRY NOT TO LET IT AFFECT MY LIFE TOO MUCH.
WAIT, I CAN'T BELIEVE THIS.
DON'T WORRY ABOUT IT. FORGET I SAID ANYTHING.
BUT—
LET'S GET TO THE MOVIE.
I, UH... OK, SURE.
THIS TRICK MAY ONLY WORK 1% OF THE TIME, BUT WHEN IT DOES, IT'S TOTALLY WORTH 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

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

by

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

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.}
```

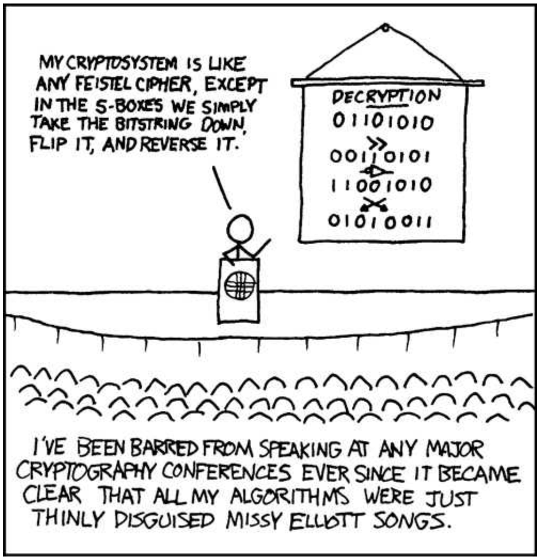
The previous example illustrates that, if the function is invertible, the probabilities are preserved: the probabilities of the various outputs match the probabilities of the inputs. It follows that, if the input is chosen according to a uniform distribution, the distribution of the output is also uniform.

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


MY CRYPTOSYSTEM IS LIKE ANY FEISTEL CIPHER, EXCEPT IN THE S-BOXES WE SIMPLY TAKE THE BITSTRING DOWN, FLIP IT, AND REVERSE IT.
DECRYPTION
01101010
00110101
11001010
01010011
I'VE BEEN BARRED FROM SPEAKING AT ANY MAJOR CRYPTOGRAPHY CONFERENCES EVER SINCE IT BECAME CLEAR THAT ALL MY ALGORITHMS WERE JUST THINLY DISGUISED MISSY ELLIOTT SONGS.

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 $\spadesuit$ 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	$\clubsuit$	0
0	$\heartsuit$	1
1	$\clubsuit$	1
1	$\heartsuit$	0

0.4.5 Perfect secrecy and invertible functions

Consider the functions

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

and

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

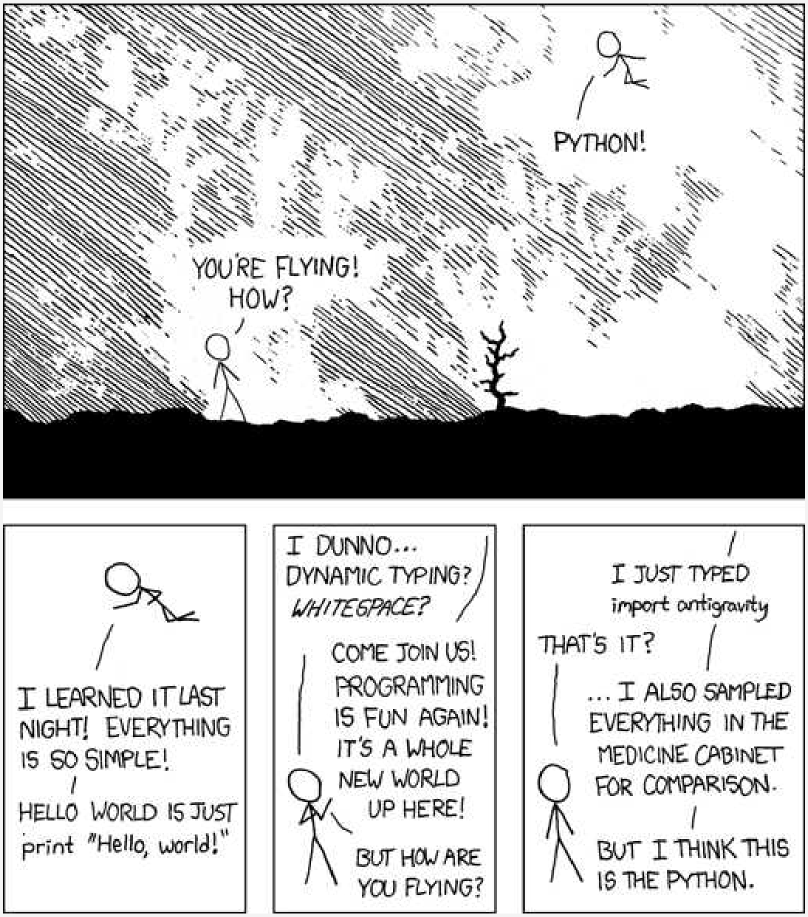
defined by

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

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

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.*

0.5 *Lab: Introduction to Python—sets, lists, dictionaries, and comprehensions*


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Python `http://xkcd.com/353/`

We will be writing all our code in Python (Version 3.x). In writing Python code, we emphasize the use of *comprehensions*, which allow one to express computations over the elements of a set, list, or dictionary without a traditional for-loop. Use of comprehensions leads to more compact and more readable code, code that more clearly expresses the mathematical idea behind the computation being expressed. Comprehensions might be new to even some readers who are familiar with Python, and we encourage those readers to at least skim the material on this topic.

To start Python, simply open a console (also called a shell or a terminal or, under Windows, a "Command Prompt" or "MS-DOS Prompt"), and type `python3` (or perhaps just `python`) to the console (or shell or terminal or Command Prompt) and hit the `Enter` key. After a few lines telling you what version you are using (e.g., Python 3.4.1), you should see `>>>` followed by a space. This is the *prompt*; it indicates that Python is waiting for you to type something. When you type an expression and hit the `Enter` key, Python evaluates the expression and prints the result, and then prints another prompt. To get out of this environment, type `quit()` and `Enter`, or `Control-D`. To interrupt Python when it is running too long, type `Control-C`.

This environment is sometimes called a *REPL*, an acronym for "read-eval-print loop." It reads what you type, evaluates it, and prints the result if any. In this assignment, you will interact with Python primarily through the REPL. In each task, you are asked to come up with an expression of a certain form.

There are two other ways to run Python code. You can import a *module* from within the REPL, and you can run a Python script from the command line (outside the REPL). We will discuss modules and importing in the next lab assignment. This will be an important part of your interaction with Python.

0.5.1 *Simple expressions*

Arithmetic and numbers

You can use Python as a calculator for carrying out arithmetic computations. The binary operators +, *, -, / work as you would expect. To take the negative of a number, use - as a unary operator (as in -9). Exponentiation is represented by the binary operator **, and truncating integer division is //. Finding the remainder when one integer is divided by another (modulo) is done using the % operator. As usual, ** has precedence over * and / and //, which have precedence over + and -, and parentheses can be used for grouping.

To get Python to carry out a calculation, type the expression and press the Enter/Return key:

```
>>> 44+11*4-6/11.
87.454545454545454
>>>
```

Python prints the answer and then prints the prompt again.

Task 0.5.1: Use Python to find the number of minutes in a week.

Task 0.5.2: Use Python to find the remainder of 2304811 divided by 47 without using the modulo operator %. (Hint: Use //.)

Python uses a traditional programming notation for scientific notation. The notation

`6.022e23` denotes the value 6.02×10^{23}, and `6.626e-34` denotes the value 6.626×10^{-34}. As we will discover, since Python uses limited-precision arithmetic, there are round-off errors:

```
>>> 1e16 + 1
1e16
```

Strings

A string is a series of characters that starts and ends with a single-quote mark. Enter a string, and Python will repeat it back to you:

```
>>> 'This sentence is false.'
'This sentence is false.'
```

You can also use double-quote marks; this is useful if your string itself contains a single-quote mark:

```
>>> "So's this one."
"So's this one."
```

Python is doing what it usually does: it *evaluates* (finds the value of) the expression it is given and prints the value. The value of a string is just the string itself.

Comparisons and conditions and Booleans

You can compare values (strings and numbers, for example) using the operators ==, < , >, <=, >=, and !=. (The operator != is inequality.)

```
>>> 5 == 4
False
>>> 4 == 4
True
```

The value of such a comparison is a Boolean value (True or False). An expression whose value is a boolean is called a Boolean expression.

Boolean operators such as `and` and `or` and `not` can be used to form more complicated Boolean expressions.

```
>> True and False
False
>>> True and not (5 == 4)
True
```

Task 0.5.3: Enter a Boolean expression to test whether the sum of 673 and 909 is divisible by 3.

0.5.2 *Assignment statements*

The following is a *statement*, not an expression. Python executes it but produces neither an error message nor a value.

```
>>> mynum = 4+1
```

The result is that henceforth the variable `mynum` is bound to the value 5. Consequently, when Python evaluates the expression consisting solely of `mynum`, the resulting value is 5. We say therefore that the value of `mynum` is 5.

A bit of terminology: the variable being assigned to is called the *left-hand side* of an assignment, and the expression whose value is assigned is called the *right-hand side*.

A variable name must start with a letter and must exclude certain special symbols such as the dot (period). The underscore _ is allowed in a variable name. A variable can be bound to a value of any type. You can rebind `mynum` to a string:

```
>>> mynum = 'Brown'
```

This binding lasts until you assign some other value to `mynum` or until you end your Python session. It is called a *top-level* binding. We will encounter cases of binding variables to values where the bindings are temporary.

It is important to remember (and second nature to most experienced programmers) that an assignment statement binds a variable to the *value* of an expression, not to the expression itself. Python first evaluates the right-hand side and only then assigns the resulting value to the left-hand side. This is the behavior of most programming languages.

Consider the following assignments.

```
>>> x = 5+4
>>> y = 2 * x
>>> y
18
>>> x = 12
>>> y
18
```

In the second assignment, `y` is assigned the value of the expression `2 * x`. The value of that expression is 9, so `y` is bound to 18. In the third assignment, `x` is bound to 12. This does not change the fact that `y` is bound to 18.

0.5.3 ***Conditional expressions***

There is a syntax for conditional expressions:

$\langle expression \rangle$ `if` $\langle condition \rangle$ `else` $\langle expression \rangle$