

0.3. THE FUNCTION



The notation

$$f : D \rightarrow F$$

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 $\{A, B, \dots, Z\}$.

^aSome imaginary historians have conjectured that Caesar's assassination 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

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

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
```

multiple

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.

mul. prod
積

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

factor
約數

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

筆算
12
3

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 $(6, 2)$.

plain
italic
bold
正体
斜体
太字

(1, 12)
(12, 1)

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:

順

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 2).

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.

with great power comes great responsibility

or
(1, 2) → 8
(2, 2) → 4

∴ 12 =

專用語

quote
誤引用

造

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

F to the power of D

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": *← 文字*

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 $\text{id}_D : D \rightarrow D$ called the identity function for D , defined by

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

for every $d \in D$.

g follows f
g after f

0.3.5 Composition of functions

合成

g circle f
g on f

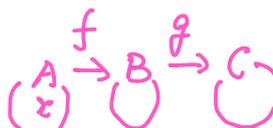
The operation functional composition combines two functions to get a new function. We will later define matrix multiplication in terms of functional composition. Given two functions $f : A \rightarrow B$ and $g : B \rightarrow C$, the function $g \circ f$, called the composition of g and f , is a function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

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

belongs

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.



合法

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 $g \circ f(x) = (x + 1)^2$.

y square

Example 0.3.11: Define the function

$$f : \{A, B, C, \dots, Z\} \rightarrow \{0, 1, 2, \dots, 25\}$$

by

/: 剰余 ⇒ $A \mapsto 0, B \mapsto 1, C \mapsto 2, \dots, Z \mapsto 25$

Define the function g as follows. The domain and co-domain of g are both the set $\{0, 1, 2, \dots, 25\}$, and $g(x) = (x + 3) \bmod 26$. For a third function h , the domain is $\{0, \dots, 25\}$ and the co-domain is $\{A, \dots, Z\}$, 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 5) being composed.

直観
直感

illustrate
イラスト

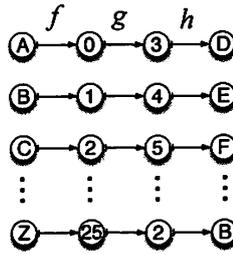


Figure 1: This figure represents the composition of the functions f, g, h . Each function is represented by arrows from circles representing its domain to circles representing its co-domain. The composition of the three functions is represented by following three arrows.

結合性

0.3.6 Associativity of function composition

合成則結合則

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 .

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

□

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

逆函數

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 5): f is a function from $\{A, \dots, Z\}$ to $\{0, \dots, 25\}$ and h is a function from $\{0, \dots, 25\}$ to $\{A, \dots, Z\}$. Each one reverses the effect of the other. That is, $h \circ f$ is the identity function on $\{A, \dots, Z\}$, and $f \circ h$ is the identity function on $\{0, \dots, 25\}$. 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

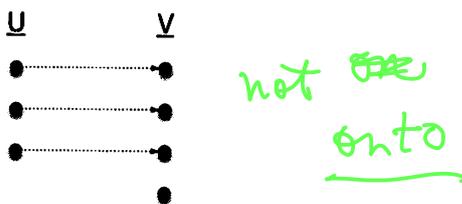


Figure 2: A function $f : U \rightarrow V$ is depicted that is not onto, because the fourth element of the co-domain is not the image under f of any element

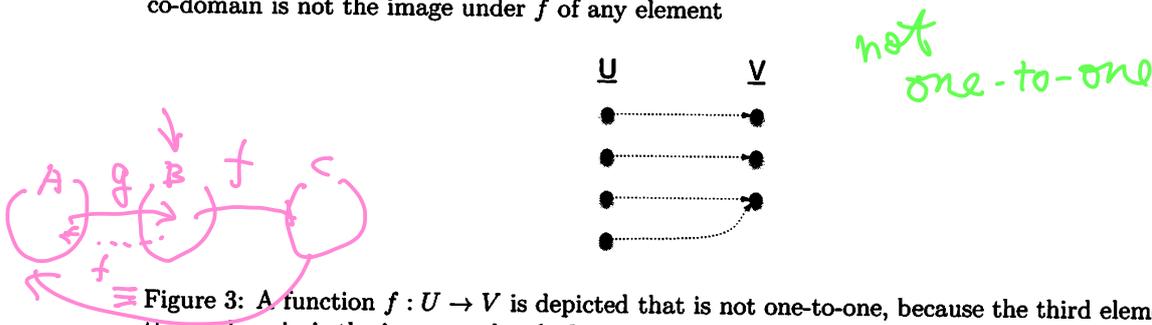


Figure 3: A function $f : U \rightarrow V$ is depicted that is not one-to-one, because the third element of the co-domain is the image under f of more than one element.

- $f \circ g$ 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 $f : D \rightarrow F$. 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 3). Since a prime 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. \square

Theorem 0.3.18 (Function Invertibility Theorem): A function is invertible if 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 , $f(g(\hat{y})) = \hat{y}$. \square



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

Proof

Let $f : U \rightarrow V$ 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 $g_1(v) = g_2(v)$. \square

0.3.8 Invertibility of the composition of invertible functions

In Example 0.3.11 (Page 5), 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 $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$.



\times **Problem 0.3.21:** Prove Lemma 0.3.20.

\Rightarrow **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.

\square

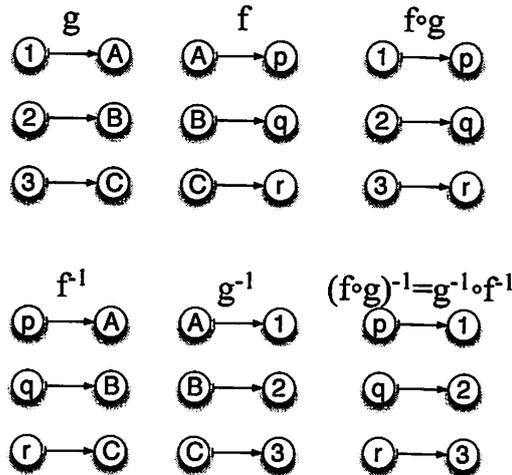


Figure 4: The top part of this figure shows two invertible functions f and g , and their composition $f \circ g$. Note that the composition $f \circ g$ is invertible. This illustrates Lemma 0.3.20. The bottom part of this figure shows g^{-1} , f^{-1} and $(f \circ g)^{-1}$. Note that $(f \circ g)^{-1}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

確率論

0.4 Probability

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

Random Number (<http://xkcd.com/221/>)

「やあ」

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.)

統計学

分布

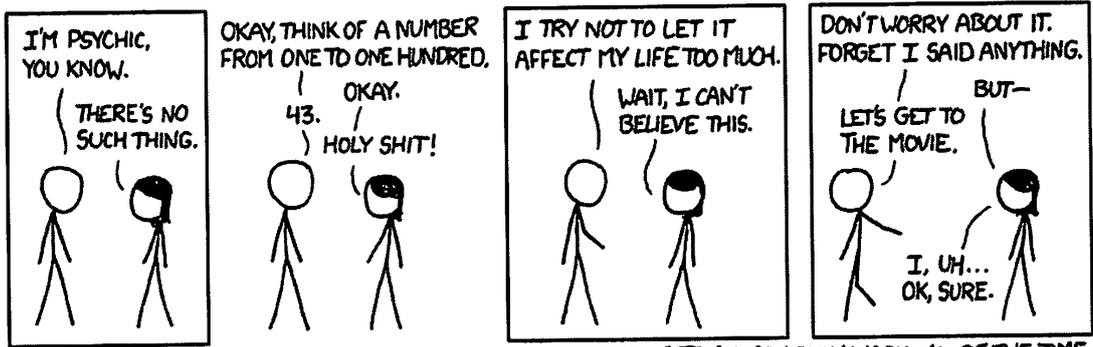
0.4.1 Probability distributions

離散

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.

尤度
ゆうど

抽象化



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: $\text{Pr}(\text{heads}) = \text{Pr}(\text{tails})$. Since we require the sum to be 1, $\text{Pr}(\text{heads}) = 1/2$ and $\text{Pr}(\text{tails}) = 1/2$. 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 $\text{Pr}(1) = \text{Pr}(2) = \dots = \text{Pr}(6)$. 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 $\Omega = \{A, B, C, \dots, Z\}$, 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 10), 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 $f : \{1, 2, 3, 4, 5, 6\} \rightarrow \{0, 1\}$ 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 10)), 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 10), each of these possibilities 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 $\{0: 1/2, 1: 1/2\}$.