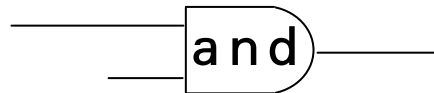


Mathematics

for the Digital Age



Programming

in Python

>>> Second Edition:
with Python 3

Maria Litvin

Phillips Academy, Andover, Massachusetts

Gary Litvin

Skylight Software, Inc.

Skylight Publishing
Andover, Massachusetts

Skylight Publishing
9 Bartlet Street, Suite 70
Andover, MA 01810

web: <http://www.skylit.com>
e-mail: sales@skylit.com
support@skylit.com

**Copyright © 2010 by Maria Litvin, Gary Litvin, and
Skylight Publishing**

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the authors and Skylight Publishing.

Library of Congress Control Number: 2009913596

ISBN 978-0-9824775-8-8

The names of commercially available software and products mentioned in this book are used for identification purposes only and may be trademarks or registered trademarks owned by corporations and other commercial entities. Skylight Publishing and the authors have no affiliation with and disclaim any sponsorship or endorsement by any of these products' manufacturers or trademarks' owners.

1 2 3 4 5 6 7 8 9 10 15 14 13 12 11 10

Printed in the United States of America

14 Matrices, Sets, and Dictionaries

14.1 Prologue

A table is a common way of presenting and analyzing data. For example, a teacher's gradebook may have one row for each student and one column for each test (Figure 14-1). This way the teacher can conveniently see the scores for each assignment and follow the progress of each student.

	Test 1	Test 2	Final
Abbot, David	94	87	85
Brower, Janet	85	90	93
Costello, Emily	90	92	90
Cote, Adam	78	72	65

Figure 14-1. A table in a gradebook

A company manager might arrange sales data by year and by quarter to compare the quarterly figures for each year as well as quarterly fluctuations from year to year (Figure 14-2). And so on — tables are ubiquitous.

	2003	2004	2005	2006
1st Qtr	3,512	3,002	3,623	3,216
2nd Qtr	4,720	4,500	4,295	4,080
3rd Qtr	3,827	3,391	3,994	3,511
4th Qtr	5,008	4,856	4,659	4,388
Total	17,067	15,749	16,571	15,195

Figure 14-2. A table of sales data by year and quarter

In mathematics, a rectangular table of numbers is called a *matrix*. Matrices are important in many branches of mathematics, especially in *linear algebra*, which deals with linear transformations of vectors and solving systems of linear equations.

In this chapter we will learn how to represent tables and matrices in Python programs.

As we saw in Chapter 1, a *set* represents a collection of elements with no duplicate values. Python’s built-in function `set` converts a given “sequence” (list, string, tuple, etc.) into a set. A set in Python is implemented as a special data structure (called a *hash table*) that makes searching for a particular element very efficient.

A *dictionary* is another Python structure. A dictionary defines a mapping from a set of *keys* to a set of *values*. All the keys in a dictionary are different from each other. A dictionary associates one value with each key. A dictionary is, therefore, another way to define a function whose domain is a finite set. In this chapter, you will learn how to work with Python sets and dictionaries.

14.2 Tables and Matrices

There is no special structure in Python for representing a two-dimensional table. Typically each row of a table is represented as a list, and the whole table is represented as a list of lists, its rows. For example,

```
1 2 3
4 5 6
```

can be defined as:

```
>>> m = [[1, 2, 3], [4, 5, 6]]
```

Actually, Python lets you split lists between lines, so you can write

```
>>> m = [[1, 2, 3],
         [4, 5, 6]]
```

`m[0]` is the first row of the table, `m[1]` is the second row, and so on. `m[r]` refers to the row with the index `r` (the $(r+1)$ -th row of the table — recall that in Python indices start from 0). The elements in the row with the index `r` are `m[r][0]`, `m[r][1]`, `m[r][2]`, and so on. In the above example, the value of the element `m[0][2]` is 3.

If m is a table defined as a list of its rows, $m[r][c]$ refers to the element in the row with the index r and the column with the index c . $\text{len}(m)$ gives the number of rows; $\text{len}(m[r])$ gives the number of elements in the r -th row.

For a rectangular table m in which all rows are the same length, use $\text{len}(m[0])$ for the number of columns.



The mathematical term *matrix* usually refers to a rectangular table that contains numbers. Each element of a matrix is numbered by two subscripts: the row number and the column number. For example:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

The numbers in the matrix are called *elements* or *entries*. In math, the subscripts usually start from 1; you have to translate them into Python indices (which start from 0) when you represent a matrix in a program.

Example 1

Write a Python function that returns the sum of all the elements of a given matrix.

Solution

```
def sumElements(m):
    sumMrc = 0
    r = 0
    while r < len(m):
        c = 0
        while c < len(m[r]):
            sumMrc += m[r][c]
            c += 1
        r += 1
    return sumMrc
```

Or:

```
def sumElements(m):
    sumMrc = 0
    for row in m:
        for x in row:
            sumMrc += x
    return sumMrc
```

Or, even shorter:

```
def sumElements(m):
    sumMrc = 0
    for row in m:
        sumMrc += sum(row)
    return sumMrc
```

Or, using a list comprehension:

```
def sumElements(m):
    return sum([sum(row) for row in m])
```



In math, a list of n numbers is called an n -dimensional vector. For example, a three-dimensional vector (x, y, z) can represent the x, y, z coordinates of a point in space. Let $\vec{x} = (x_1, x_2, \dots, x_n)$ and $\vec{y} = (y_1, y_2, \dots, y_n)$ be two n -dimensional vectors. Their sum $\vec{x} + \vec{y}$, is defined as a new vector $(x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$.

Example 2

Write a Python function that returns the sum of two given vectors (represented as lists of the same length).

Solution

```
def sumVectors(x, y):
    v = len(x)*[0] # create a list of the same length as x,
                  #   filled with 0s
    i = 0
    while i < len(x):
        v[i] = x[i] + y[i]
        i += 1
    return v
```

Note that we have to create a vector of the required length before we can store the resulting values in it.



A table in Python can hold any types of objects. For example, you can represent a chess position as an 8 by 8 table in which each element either holds a chess piece or None.

You need to create a table before you can store values in it.

Example 3

Write a Python function that creates and returns a table with a given numbers of rows and columns, filled with None values.

Solution

```
def buildTable(nRows, nCols):
    table = []
    for r in range(nRows):
        table.append(nCols*[None])
    return table
```

We start with an empty list of rows `table` and append a new row to it `nRows` times. `nCols*[None]` creates one row: a list of `nCols` elements, all set to None.



In the above example, you might find it wasteful to repeat `nCols*[None]` several times. You might be tempted to take a shortcut: create a row once, then just append it to the table `nRows` times. Like this:

```
def buildTable(nRows, nCols):
    # buggy code!
    table = []
    row = nCols*[None]
    for r in xrange(nRows):
        table.append(row)
    return table
```

At first, everything seems OK:

```
>>> t = buildTable(2, 3)
>>> t
[[None, None, None], [None, None, None]]
```

But try this:

```
>>> t[0][0] = 'Some'
>>> t
```

What do you get?

```
[['Some', None, None], ['Some', None, None]]
```

This is because the rows in your table actually refer to the same list!

Exercises

1. Let $\vec{x} = (x_1, x_2, \dots, x_n)$ and $\vec{y} = (y_1, y_2, \dots, y_n)$ be two n -dimensional vectors. Their *dot product*, denoted $\vec{x} \cdot \vec{y}$, is defined as $\vec{x} \cdot \vec{y} = x_1y_1 + x_2y_2 + x_3y_3 + \dots + x_ny_n$. Write and test a Python function that returns the dot product of two given vectors (represented as lists of the same length). ✓
2. Write and test a Python function that returns the sum of the elements on the main diagonal (upper left to lower right) of a square matrix (represented as a list of lists). (In linear algebra, this value is called the *trace* of the matrix.)

3. Write and test a Python function that takes a matrix and prints it neatly, as a rectangular table with aligned right-justified columns. Assume that all the entries in the matrix are positive integers with no more than two digits. ✓
4. ■ Write and test a Python program that reads a matrix of integers from a text file. Each line in the file holds integer values for the corresponding row of the matrix (separated by spaces or tabs). ≦ Hint: For testing, use the function from Question 3 to display the resulting matrix. ≧
5. ■ An n by n matrix defines a *linear transformation* (function) on n -dimensional vectors. If $A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$ and $\vec{x} = (x_1, x_2, \dots, x_n)$, then $A \cdot \vec{x}$ is a new vector $\vec{y} = (y_1, y_2, \dots, y_n)$, such that $y_i = a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n$. In other words, y_i is the dot product of the i -th row of the matrix and \vec{x} . Write and test a Python function that takes an n by n matrix A and an n -dimensional vector \vec{x} and returns the vector $A \cdot \vec{x}$. ✓
6. ♦ Suppose we have two n by n matrices:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}$$

The matrix C in which $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}$ is called the *product* of A and B and denoted as $A \cdot B$ or simply AB . c_{ij} is the dot product of the i -th row in A and the j -th column in B . Write and test a Python function that takes two square matrices of the same size and returns their product. ≦ Hint: don't forget to create the resulting matrix before you put values into it. ≧

- 7.♦ Write a program that prompts the user to enter an odd positive integer n , between 3 and 19, and prints out a magic square of size n . (A magic square holds consecutive numbers from 1 to n^2 , so that the sum of the numbers in each row, in each column, and in each of the two diagonals is the same.)

There is a simple method for generating a magic square when its size is an odd number. We start with 1 in the middle of the top row. We try to place the next consecutive number by shifting from the current position diagonally up and to the right by one. If we get out of bounds above the top row, we “wrap around” and take the corresponding spot in the bottom row. Likewise, if we get out of bounds on the right, we wrap around and take the corresponding spot in the left column. If the spot where we want to place the number is already filled, we instead shift down by one from the current position. For example:

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

14.3 Sets

In Python, values between braces, separated by commas, define a set. For example:

```
>>> nums = {1, 2, 3}
```

Python’s built-in function `set` takes a “sequence” (such as a list, a tuple, a string, or another set) and builds a set of all the different elements from that sequence. For example:

```
>>> nums = set([1, 2, 3, 2, 1])
>>> nums
{1, 2, 3}
```

set() creates an empty set. {} is an empty dictionary -- see Section 14.4.

You might be wondering: Why do we need sets if we already have lists? There are two reasons. First, the implementation of a set in Python uses a special structure, called a *hash table*, which allows you to find any element quickly and to add and remove elements quickly, regardless of the size of the set. Second, a set cannot have duplicates, so if you want to get rid of duplicate values in a list, you can simply construct a set from that list.

If you are looking for a particular value in a list, you pretty much have to scan through the list to find that value (or to become convinced that the value is not in the list). You can try to keep the list in order (for example, a list of names in alphabetical order), which can make searching faster. But then inserting elements in the middle takes more time. A hash table helps to optimize the time for both searching and adding new elements.

↓ We don't want to go into the details of hashing here. The main idea is to store each element in or near a location that is calculated from the value of that element itself. For example, all valid zip codes that start with 417 can be held in a relatively short list (called a *bucket*) attached to the 417-th element in a list of buckets. To find whether 41705 is a valid zip code, we go straight to the 417-th bucket, then look for 41705 only in that bucket. The number of buckets in this example will be 1000 (from 000 to 999). Some of the buckets in a hash table may be empty, so some space may be wasted, but if we need fast performance, we don't mind wasting a little space.

The elements in a set are not stored in any particular order, and they do not have indices.

We usually just want to know whether a given value is in the set. We can tell that by using the `in` operator:

```
if x in s:  
    ...
```

or

```
if x not in s:  
    ...
```

For example:

```
if 'Ben' in names:
    print 'Hello, Ben'
```

Occasionally we might need to use a `for` loop to access all the elements in a set, but the `for` loop will produce the elements in an unpredictable order. For example:

```
>>> names = {'Ana', 'Ben', 'Clair'}
>>> for name in names:
        print(name, end = ' ')
```

Ben Ana Clair

The choice function cannot be applied directly to a set — you need to convert the set into a list first.

For example:

```
from random import choice
name = choice(list(names))
```

You can apply the built-in functions `len`, `sum`, `max`, and `min` to sets. `len` returns the number of elements in the set. `sum` returns the sum of all elements (assuming that the `+` operator can be applied to them). `max` and `min` return the largest and the smallest element, respectively, assuming all the elements are comparable to each other.



Python has intersection, union, and difference operators for sets. The symbols for them are `&`, `|`, and `-`, respectively. For example:

```
>>> s1 = {1, 2, 3, 4}
>>> s2 = {1, 3, 5, 7}
>>> s1 & s2
{1, 3}
>>> s1 | s2
{1, 2, 3, 4, 5, 7}
>>> s1 - s2
{2, 4}
```

(`s1-s2` is a set of all the elements from `s1` that are not in `s2`.)

Python also has the *symmetric difference* operator `^`. `s1^s2` is a set of all the elements that are either in `s1` or in `s2`, but not in both. For example:

```
>>> s1 = {1, 2, 3, 4}
>>> s2 = {1, 3, 5, 7}
>>> s1 ^ s2
{2, 4, 5, 7}
```

Note that the same symbols `&`, `|`, and `^` are used for the bitwise and, or, and xor operators on integers (see Chapter 7) as for the intersection, union, and symmetric difference of sets. This is not a coincidence, of course: recall the correspondence between the operators on sets and the logical operators (see Chapter 6).



A set in Python also has the methods `add`, `update`, `discard`, `remove`, and `issubset`. For example, `s.add(x)` adds `x` to the set `s`. `s.discard(x)` removes `x` from `s` if `x` is in `s` (and does nothing if it isn't). The set methods are summarized in Appendix D.

Exercises

1. Write a one-line function `allDifferent(word)` that returns `True` if all the letters in `word` are different. $\hat{=}$ Hint: use a set. $\hat{=}$ ✓
2. Can a Python set hold tuples? Lists? Other sets? Explore and explain the results.
3. Does the `choice` function (from the `random` module) work for sets? Explore and explain the result.
4. Draw a venn diagram for the symmetric difference of two sets.
5. Write an expression equivalent to `s1^s2` using the `&`, `|`, and `-` operators for sets. ✓
6. Pretending that the `&` operator for sets and the `intersection` method do not exist, write your own function `intersection(s1, s2)` that returns the intersection of the sets `s1` and `s2`. ✓
7. Write a function `primes(n)` that returns a set of all the prime numbers that do not exceed `n`. For example, `primes(10)` should return `{2, 3, 5, 7}`. Your function should build the resulting set gradually. Start with an empty set. For all integers `k` from 2 to `n`, if `k` is not divisible by any of the primes in the set, add `k` to the set.

- 8.▪ You can create your own hash table as a list of, say, 16 buckets. Each bucket is a list, too. Start with a hash table in which all the buckets are empty. In Python every “hashable” object (such as a number, a string, etc.) has a method `__hash__()`, which returns an integer. For example:

```
>>> 'Amy'.__hash__()
871006000
```

`x.__hash__() % 16` gives you a number in the range from 0 to 15. You can use that number as the index of the bucket in which `x` belongs. If `x` is not in that bucket yet, you can add it to the bucket. Write a function `myOwnSet(lst)` that takes a list of values `lst`, puts them into your hash table, constructed as described above, and returns that table.

- 9.♦ In the `myOwnSet` function described in Question 8, make the number of buckets in the hash table a parameter passed to the function. Create an empty table with 30 buckets, then put into it about 100 words from a typical text file or list of words. Find the most populated bucket and print out the number of words in it. ⚡ Hint: write a *helper* function `add(s, word)` that adds `word` to `myOwnSet s`. ⚡

14.4 Dictionaries

A *dictionary* in Python establishes a correspondence between a set of *keys* and a set of *values*. Only one value corresponds to each key. (In some other programming languages, such a structure is called a *map*). In mathematical terms, a dictionary defines a function on a set of keys. In practical terms, a dictionary lets you quickly look up a value (an object, a data record, a text segment) associated with a given key. In a database of taxpayers, for example, the key may be a taxpayer’s social security number, and his or her record may be the associated value. In the zip code lookup program, the key may be a zip code, and the name of the city or town for that zip code will be the associated value. The set of keys is implemented as a hash table.

You can create a dictionary with initial entries by listing the *key : value* pairs within curly braces. For example:

```
coins = {'Quarter' : 25, 'Dime' : 10,
        'Nickel' : 5, 'Penny' : 1}
```

Another example:

```
states = {
    'AL' : 'Alabama',
    'AK' : 'Alaska',
    # ...
    # ...
    'WY' : 'Wyoming'
}
```

Sometimes a short dictionary is used to name the fields of a data record for convenience. For example:

```
book = {'title' : 'Green Eggs and Ham',
        'author' : 'Dr. Seuss',
        'isbn' : '0545002850'}
```

`d = {}` sets `d` to an empty dictionary.

Once a dictionary is defined, you can refer to its values using a key as an “index.” For example:

```
>>> book['author']
'Dr. Seuss'
>>> book['isbn']
'0545002850'
```

You can modify the value associated with a key and add new key-value pairs using similar syntax:

```
>>> book['isbn'] = '9780583324205'
>>> book['price'] = 7.99
>>> book['price']
7.99
```

Note that in the above examples the keys are literal strings, not names of variables.

If `d` is a dictionary, `len(d)` returns the number of key-value pairs in `d`. `d.keys()` returns a set (more precisely a `dict_keys` type object) of all the keys in `d`. Use `k` in `d` to test whether the key `k` is defined in the dictionary `d`. The statement `del d[k]` removes the key `k` and the associated value from the dictionary `d`.

The dictionary methods are summarized in Appendix D.



We said earlier that a dictionary associates a single value with each key. However, nothing prevents us from making that value a list or a tuple, if necessary. For example:

```
spanishEnglish = {
    'abajo' : ['down', 'below', 'downstairs'],
    'presto' : ['quick', 'prompt', 'ready', 'soon']
}
...
translations = spanishEnglish['presto']
print(translations)
```

The output will be:

```
[quick, prompt, ready, soon]
```

Exercises

1. Define a dictionary that holds several name-password pairs (with the name used as the key). Write and test a function that takes such a dictionary and a (name, password) pair (represented as a tuple) and returns `True` if the name is in the dictionary and the password matches the one in the dictionary.
2. A tune is described by its title, band, and duration in seconds. Define a dictionary for “Alligator” by The Nationals, 4:05. ✓
3. Suppose in a dictionary all values associated with keys are different. Write a function `reverseDictionary(d)` that takes such a dictionary and returns a “reverse” dictionary in which each value becomes a key and its key becomes the associated value. For example, `reverseDictionary({1: 'A', 2: 'B'})` should return `{'A':1, 'B':2}`.

4. ■ Write a program that quizzes a student on the capitals of the 50 U.S. states. Create a text file that lists all the states and their capitals. The program reads the file line by line and puts all the state-capital pairs into a dictionary. The program then extracts a set of all the states from the dictionary, and ten times presents a randomly chosen state to the student (without repetition). The program matches each answer (case blind) against the state capital in the dictionary and keeps track of the number of correct answers. At the end, the program displays the number of correct answers.

5. ♦ Define a dictionary that can serve as an index for a book. Each key is a word; the associated value is a list of the page numbers of all the pages on which that word occurs. Write and test a function `addEntry(d, word, page)` that takes such a dictionary `d` and adds `page` to the list of page numbers associated with `word`. If `word` is already in the dictionary and `page` is in its list, then `page` should not be added. If `word` is not yet in the dictionary, then `addEntry` should create a new entry for `word` with `page` in its list.

14.5 Review

Terms and notation introduced in this chapter:

<i>Matrix</i>	<i>Hashing</i>	$\vec{x} \cdot \vec{y}$
<i>Element or entry in a matrix</i>	<i>Hash table</i>	
<i>Vector</i>	<i>Symmetric difference</i>	$C = AB$
<i>Dot product</i>	<i>Map</i>	
<i>Product of matrices</i>	<i>Dictionary</i>	

Some of the Python features mentioned in this chapter:

```
m = [[1, 2, 3],
      [4, 5, 6]]
nRows = len(m)
nCols = len(m[0])
x = m[i][j]

table = []
for r in range(nRows):
    table.append(nCols*[None])

s = {'A', 'B', 'C'}
if x in s:
    ...
if x not in s:
    ...
emptySet = set()
s1 & s2
s1 | s2
s1 - s2
s1 ^ s2

d = {'K1':'A', 'K2':'B', 'K3':'C'}
x = d[k]
d[k] = x
del d[k]
if k in d:
    ...
allKeys = d.keys()
```