

Chapter 1

Data types

In this chapter you will:

- learn about data types
- learn about tuples, lists and dictionaries
- make a version of MyMagic8Ball that is much shorter than the one from *Python Basics*.

Data types

In *Python Basics* you learned about strings (bits of text), integers (whole numbers) and floats (numbers with a decimal point). These are examples of **data types**. There are more! In this chapter we will look at some new data types: tuples, lists and dictionaries. These new data types are all called **container** data types because they store more than one piece of data. For example, they can store several strings. They do so in different ways and have their own advantages and disadvantages.

A **string** is rather like a container because it stores a whole sequence of letters or numbers (or a mixture of both). In *Python Basics* we learned that there are several functions we can use on strings. We can also use many of these functions on tuples, lists and dictionaries.



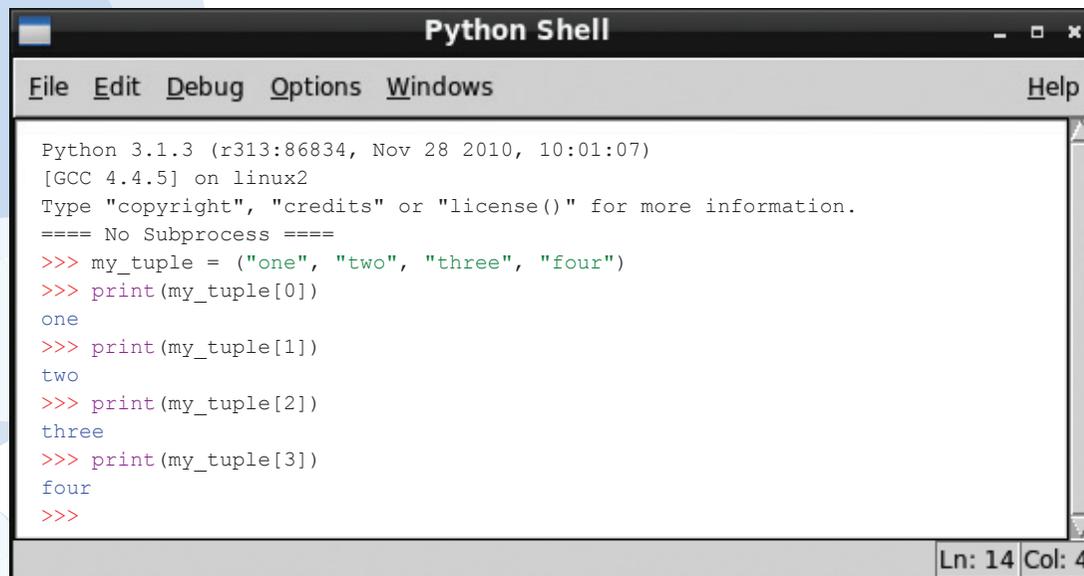
Tuples

A **tuple** is the simplest of our new data types. They can store strings, integers and other data types. Here is an example of a tuple that stores four strings, each separated by a comma:

```
my_tuple = ("one", "two", "three", "four")
```

Each **value** in a tuple is separated by a comma. Unlike variables, we cannot change what is stored in a given tuple.

Each value in the tuple has an **index** starting from 0. So, `print(my_tuple[1])` for the example above produces the **output** `two`. Look at how this works in Figure 1.1.



```
Python Shell
File Edit Debug Options Windows Help
Python 3.1.3 (r313:86834, Nov 28 2010, 10:01:07)
[GCC 4.4.5] on linux2
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> my_tuple = ("one", "two", "three", "four")
>>> print(my_tuple[0])
one
>>> print(my_tuple[1])
two
>>> print(my_tuple[2])
three
>>> print(my_tuple[3])
four
>>>
```

Figure 1.1 A tuple.

MyMagic8Ball

In *Python Basics* we wrote a small application called MyMagic8Ball that used the `random` module and the functions `print()`, `input()` and `randint()`. Here is the code:

Code Box 1.1

```
# My Magic 8 Ball

import random

# write answers
ans1="Go for it!"
ans2="No way, Jose!"
ans3="I'm not sure. Ask me again."
ans4="Fear of the unknown is what imprisons us."
ans5="It would be madness to do that!"
ans6="Only you can save mankind!"
ans7="Makes no difference to me, do or don't - whatever."
ans8="Yes, I think on balance that is the right choice."

print("Welcome to MyMagic8Ball.")

# get the user's question
question = input("Ask me for advice then press ENTER to shake me.\n")
```

(continues on the next page)

```
print("shaking ...\\n" * 4)

# use the randint() function to select the correct answer
choice=random.randint(1, 8)
if choice==1:
    answer=ans1
elif choice==2:
    answer=ans2
elif choice==3:
    answer=ans3
elif choice==4:
    answer=ans4
elif choice==5:
    answer=ans5
elif choice==6:
    answer=ans6
elif choice==7:
    answer=ans7
else:
    answer=ans8

# print the answer to the screen
print(answer)

input("\\n\\nPress the RETURN key to finish.")
```

Now see how much easier and shorter the code is if we include a tuple:

Code Box 1.2

```
# My Magic 8 Ball
import random

# put answers in a tuple
answers = (
    "Go for it!",
    "No way, Jose!",
    "I'm not sure. Ask me again.",
    "Fear of the unknown is what imprisons us.",
    "It would be madness to do that!",
    "Only you can save mankind!",
    "Makes no difference to me, do or don't - whatever.",
    "Yes, I think on balance that is the right choice."
)

print("Welcome to MyMagic8Ball.")

# get the user's question
question = input("Ask me for advice then press ENTER to shake me.\n")

print("shaking ...\n" * 4)

# use the randint() function to select the correct answer
choice = random.randint(0, 7)
```

(continues on the next page)

```
# print the answer to the screen
print(answers[choice])

# exit nicely
input("\n\nPress the RETURN key to finish.")
```

Analysis of Code Box 1.2

If it is a while since you read *Python Basics*, you might find it useful to type this code into IDLE and think about it line by line. Here is what it does.

The import statement

We are going to use a **function** from Python's random module so we need to import it.

The tuple

We have to separate the strings in the tuple `answers` with commas. Starting a new line after each comma makes the code much easier to read.

The input() function

The `input()` function listens to the keyboard entry and waits for the **return** key to be pressed. It then returns the keyboard input as a string, which we store in the **variable** `question`.

```
question = input("Ask me for advice then press ENTER to shake me.\n")
```

variable name to access
the keyboard input

string that is printed out,
giving instructions to the user

The randint() function

```
choice = random.randint(0, 7)
```

This function means that the `randint()` **method** in the `random` module must select a random number from 0 to 7. This number is then stored in the variable called `choice`. (A method is a function in a class.)

Finishing off

```
print(answers[choice])
```

This uses the random number `choice` as the index in the `answers` tuple. This line selects the string that was randomly chosen from the tuple and prints it.

Experiment

The two scripts are available from the Coding Club website (www.codingclub.co.uk). Try them both out and check that they do the same thing.

Are you a bit confused about when to use round brackets and when to use square brackets? Basically when we create a tuple we wrap its contents in round brackets. Whenever we call an indexed value from the tuple, we put the index (its position in the list) in square brackets.



Lists

A **list** is another type of container. They are very similar to tuples except that they can be altered. Think of tuples as quick, memory-efficient lists that cannot be altered by other code. We cannot insert or delete items in tuples with our programs. There are however functions to allow us to insert or delete items in lists. Lists are written like this:

```
my_list = ["one", "two", "three", "four"]
```

Just as with tuples, each value in the list has an index starting from 0 and each value is separated by a comma.

Look at how this works in **interactive mode**:

```
>>> my_list = ["one", "two", "three", "four"]
>>> my_list[2]
'three'
>>> my_tuple = ("one", "two", "three", "four")
>>> my_tuple[2]
'three'
>>>
```

You can see that both a list and a tuple provide the same output. So, when would we use a list instead of a tuple? We would choose a list rather than a tuple if we want our program to add, remove or change an item within the list.

Hmm, the list is surrounded by square brackets this time.



Do you remember that interactive mode in Python means using the Python shell rather than saving and running a file? It is very useful for running little experiments.



? Quick Quiz 1.1

For each of the following say which is the best choice, a list or a tuple:

- 1 A place to store seven strings consisting of the days of the week (e.g. "Monday") that we want to use in an application.
- 2 A place to store the full names of members of the Coding Club in an application we use to keep track of who is still a club member.
- 3 A place to store the ten integer values (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9) of the keys used to make a calculator app.

Dictionaries

The last of our container data types is a **dictionary**. Dictionaries take a slightly different form. In dictionaries we supply our own indexes. Here, we call the index a **key**. Keys can be strings, integers, floats or even tuples. Here is an example:

```
my_dictionary = {1:"cat", 2:"dog", 3:"horse", 4:"fish"}
```

Diagram: Arrows point from the word "key" to the numbers 1, 2, 3, and 4. Arrows point from the word "value" to the strings "cat", "dog", "horse", and "fish".

or

```
my_dictionary = {"1":"cat", "2":"dog", "3":"horse", "4":"fish"}
```

Diagram: Arrows point from the word "key" to the strings "1", "2", "3", and "4". Arrows point from the word "value" to the strings "cat", "dog", "horse", and "fish".

Silly me, I was confused for a moment here as I had forgotten that strings always appear in speech marks and numbers do not. So 1 is an integer but "1" is a number stored as a string!



Look at how this works in interactive mode:

```
>>> my_dictionary = {1:"one", 2:"two", 3:"three", 4:"four"}
>>> my_dictionary[2]
'two'
>>> my_dictionary = {"1":"one", "2":"two", "3":"three", "4":"four"}
>>> my_dictionary["2"]
'two'
```

You might have noticed that dictionaries require a different structure within the brackets to assign keys to the values. They use a colon ':' to separate the value from its key.

What's with the brackets?

When we create a new container, Python provides us with a quick way of defining which kind we require by our choice of brackets.

- If you want a tuple – wrap it in round brackets.
- If you want a list – use square brackets.
- If it's a dictionary you are after – use curly brackets.

Delving Deeper

What's the difference?

Strings, tuples and lists are all indexed **ordered containers**; the values are automatically given an index based on the order in which they were input. Dictionaries have keys that *you* provide and the key-value pairs are *not* stored in a particular order. Strings and tuples have their content set at creation and cannot be changed by a program directly. Lists and dictionaries are containers in which the values can be added to and changed in a variety of ways.

It is also possible to create empty containers like this:

```
my_string = ""
my_tuple = ()
my_list = []
my_dictionary = {}
```

Useful functions

Table 1.1 provides a list of useful functions you can use on strings, tuples, lists and dictionaries. You can also find it in Appendix 1. The table assumes the following containers have been created:

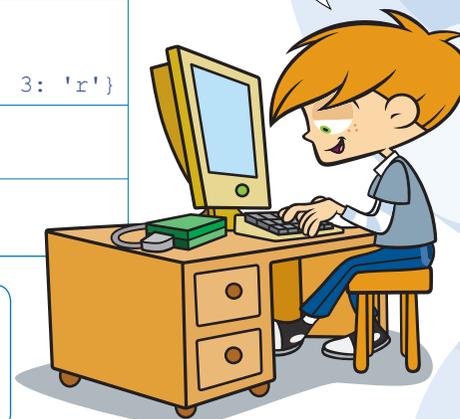
```
>>> s = "bar" # a string
>>> t = ("b", "a", "r") # a tuple
>>> l = ["b", "a", "r"] # a list
>>> d = {1:"b", 2:"a", 3:"r"} # a dictionary
```

Function	Strings	Tuples	Lists	Dictionaries
print all	<pre>>>> print(s) bar</pre>	<pre>>>> print(t) ('b', 'a', 'r')</pre>	<pre>>>> print(l) ['b', 'a', 'r']</pre>	<pre>>>> print(d) {1: 'b', 2: 'a', 3: 'r'}</pre>
print element	<pre>>>> print(s[2]) r</pre>	<pre>>>> print(t[2]) r</pre>	<pre>>>> print(l[2]) r</pre>	<pre>>>> print(d[2]) a</pre>
combine	<pre>>>> a=s+"f" >>> a 'barf'</pre>	<pre>>>> a=t+"f," >>> a ('b', 'a', 'r', 'f')</pre>	<pre>>>> a=l+["f"] >>> a ['b', 'a', 'r', 'f']</pre>	
add an element	Strings cannot be altered.	Tuples cannot be altered.	<pre>>>> l.append("f") >>> l ['b', 'a', 'r', 'f']</pre>	<pre>>>> d[4]="f" >>> d[4] 'f'</pre>
sort	Strings cannot be altered.	Tuples cannot be altered.	<pre>>>> l.sort() >>> l ['a', 'b', 'r']</pre>	<pre>>>> sorted(d) ['1', '2', '3'] >>> sorted(d.values()) ['a', 'b', 'r']</pre>
delete an element	Strings cannot be altered.	Tuples cannot be altered.	<pre>>>> del l[1] >>> l ['b', 'r']</pre>	<pre>>>> del d[1] >>> i {2:'a', 3:'r'}</pre>
replace element	Strings cannot be altered.	Tuples cannot be altered.	<pre>>>> l[0]="c" >>> l ['c', 'a', 'r']</pre>	<pre>>>> d[1]="c" >>> print(d) {1: 'c', 2: 'a', 3: 'r'}</pre>
find	<pre>>>> i.find("b") 0</pre>	<pre>>>> t.index("b") 0</pre>	<pre>>>> l.index("b") 0</pre>	
get length	<pre>>>> len(s) 3</pre>	<pre>>>> len(t) 3</pre>	<pre>>>> len(l) 3</pre>	<pre>>>> len(d) 3</pre>

Table 1.1 Some useful functions.

```
s = "bar" # a string
t = ("b", "a", "r") # a tuple
l = ["b", "a", "r"] # a list
d = {1:"b", 2:"a", 3:"r"} # a dictionary
```

This table could be very helpful when I write my own applications!



? Quick Quiz 1.2

For each of the following say whether to choose a tuple, a list, or a dictionary:

- 1 A container to store the personal best times achieved by club swimmers in the 100m freestyle such as: Mark: 65.34s, Freya: 68.04s, etc.
- 2 A container to store the months of the year.
- 3 A container to store the monthly rainfall data for London in 2012.
- 4 A container to store the names of the students who currently attend the chess club.

Chapter summary

In this chapter you have:

- learned more about data types
- learned about tuples, lists and dictionaries
- made a shorter version of MyMagic8Ball
- seen some of the different functions that can and cannot be used with the new data types.

We will explore these new data types further in this book. Here are just a few ideas that will help you refresh your coding skills from *Python Basics*. (As dictionaries are the hardest to use, we will wait until you have learned a little bit more before providing any puzzles involving them.)



Puzzle

Write a new version of MyMagic8Ball using a list instead of a tuple. It should work in exactly the same way if you get it right because lists can do everything tuples can and more.

Challenge

This is a challenge from *Python Basics* so although you may be a bit rusty you should be able to manage it. Hopefully it brings back happy memories for you.

- 1 Add some code to `myMagic8Ball12.py` (Code Box 1.2) so that the Magic8Ball says “Hi” and asks for the user’s name at the start of the game.
- 2 It should then store the input in a variable such as `user_name`
- 3 Change the code so that the Magic8Ball talks to the user using their name. At the end for example, it could say: “Thanks for playing, [Name]. Please press the RETURN key to finish.”

There are several ways to do this.

To see one answer go to www.codingclub.co.uk/book2_resources.php

Idea

Change the Magic8Ball game into a fortune cookie game. You could call it `myFortuneCookie.py`

You are destined to become
a famous computer scientist
one day!

