

# Chapter 1

## Data types

In this chapter you will:

- learn about data types
- learn about tuples, lists and dictionaries
- make a 'magic' card trick app.



## Data types

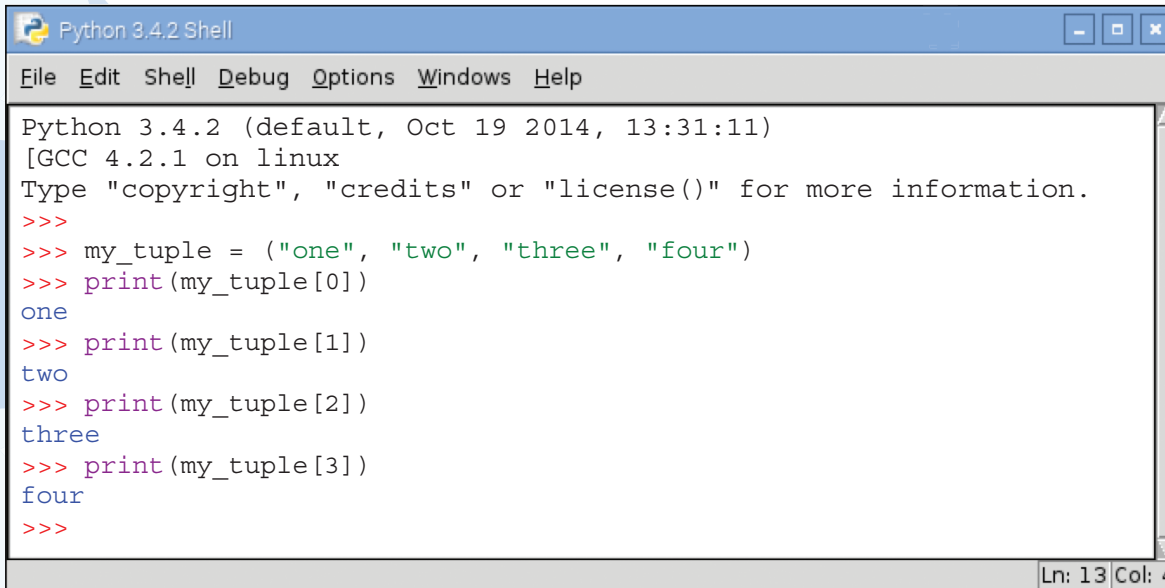
In *Python Basics* you were introduced to **strings** (bits of text), **integers** (whole numbers) and **floats** (numbers with a decimal point). These are examples of **data types**. There are more! In this chapter we are going to look at some new data types: **tuples**, **lists** and **dictionaries**. These are all **containers** that store more than one piece of **data** but they do so in different ways and have their own advantages and disadvantages.

A string could also be included in this group as it stores a whole sequence of letters. We will find that there are several **functions** that we can use on strings that can also be used on tuples, lists and dictionaries.

# Tuples

These are the simplest of our new data types. They can store strings, integers and other data types like this:

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



```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.2.1 on linux
Type "copyright", "credits" or "license()" for more information.
>>>
>>> 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
>>>
```

Each **value** in the tuple has an **index** starting from 0, so `print(my_tuple[1])` produces the **output** `two`.

Unlike the other Python container data types we will see, the contents of a tuple cannot be changed after it has been created.

## Think of a card

A common group of items that could be stored in a tuple is a pack of cards. There are 52 unchanging cards in a standard pack. It is often easier to handle the cards in our apps if we store the two possible characteristics of each card in two tuples:

```
suit = ("clubs", "diamonds", "hearts", "spades")
rank = ("two", "three", "four", "five", "six", "seven", "eight", "nine", "ten",
        "jack", "queen", "king", "ace")
```

The suits are referenced by their indexes starting from zero. “clubs” is stored in the `suit` tuple at index 0 and referenced by `suit[0]`. The rank of “queen” is stored in `rank[10]`.

### ? Quick Quiz 1.1

Which card has the two characteristics `suit[2]` and `rank[5]`?

- 1 The five of diamonds.
- 2 The six of diamonds.
- 3 The six of hearts.
- 4 The seven of hearts.

To select a random suit we can import the `random` **module** and then select a number between 0 and 3 using this code:

```
random.randint(0, 3)
```

and then use this in place of the index in the `suit` tuple like this:

```
my_random_suit = suit[random.randint(0, 3)]
```

## ? Quick Quiz 1.2

What would be the code required to select a random rank?

You are probably familiar with selecting random integers in this way from Coding Club Level 1 books. The random module has a more reliable and convenient method for selecting item values in containers, `choice()`. Instead of using `randint()` to select a random suit from `suit()` we can use this code:

```
my_random_suit = random.choice(suit)
```

This can form the basis of many card-playing games or even the short app in Code Box 1.1. We are now going to work in **script mode**. Copy this code into a new **script** and try running it.

### Code Box 1.1

```
# thinkOfACard.py

import random

# Initialise variables
guess = ""
correct = "n"

# Initialise tuples
suit = ("clubs", "diamonds", "hearts", "spades")
rank = ("two", "three", "four", "five", "six", "seven", "eight",
        "nine", "ten", "jack", "queen", "king", "ace")
```

*(continues on the next page)*

```

# Function
def choose_card():
    # Computer picks a card:
    guess_suit = random.choice(suit)
    guess_rank = random.choice(rank)
    guess = guess_rank + " of " + guess_suit
    return guess

# Start the game
print("Hello, my name is Mighty Persistent.")
print("I have magic powers: I can guess what you are thinking.")
print("Think of a card but do not tell me what it is.\n")

input("Press ENTER when you have thought of a card.")

print("\nYou are thinking of the", choose_card())
correct = input("Am I correct? (y/n)")

while correct != "y":
    print("\nOh, then it must be the", choose_card())
    correct = input("Am I correct? (y/n)")

# Finish
print("\nYay!\nI told you I could guess what you are thinking.")

# Exit nicely
input("\n\nPress the ENTER key to finish.")

```

## Analysis of Code Box 1.1

OK, it is not the most sensible app! Now that we have that out of the way, let's look at 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 tuples

We have to separate the strings in the tuples with commas. Starting a new line between the values in our container data types makes no difference, so we can use this feature to make our tuples more readable.

## The choice () function

```
guess_suit = random.choice(suit)
```

This line of code asks the `choice()` **method** in the `random` module to select a random value from the `suit` tuple. The suit that is chosen is then stored in the **variable** called `guess_suit`.

## Adding strings

```
guess = guess_rank + "of" + guess_suit
```

Remember how strings can be joined with the **+** **operator**.

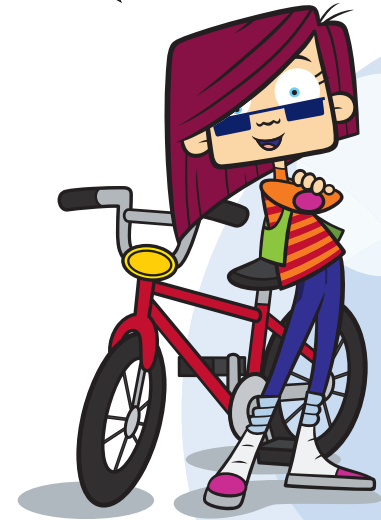
### ? Quick Quiz 1.3

Read this code:

```
guess_rank = "three"  
guess_suit = "diamonds"  
guess = guess_rank + " of " + guess_suit
```

What will the variable `guess` contain?

This app kept my little brother occupied for hours!



## 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 can store in a variable just as we did when we stored `y` or `n` in the variable `correct`.

## While loops

The code in a **while loop** keeps repeating until a certain test is successful. In this case the test requires `correct` to have the value `"y"`.

# Lists

A **list** is another type of container data type. These 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. We can, however, use functions 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 the values are separated by commas.

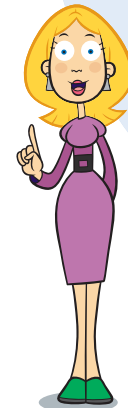
Look at how this works in **interactive mode**:

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

Hmm, the list of strings 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.



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.

### ? Quick Quiz 1.4

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

- 1 A place to store twelve strings consisting of the months in a year (e.g. "March") that we want to use in an application.
- 2 A place to store names of the cards in a player's hand in a card game application.
- 3 A place to store the names of the compass buttons (N, S, E, W, NE, SE, SW and NW) used in a game 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. This time, we call the index a **key**. Keys can be strings, integers, floats or even tuples. Here are two examples:

key                      value

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

or

key                      value

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

How embarrassing, I was confused for a moment here – 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!



No need to be embarrassed! We all forget simple things when focusing on new ideas. This is why it is so important to consolidate your learning by writing and experimenting with code. For this reason it is a good idea to make sure you try as many of the ideas and challenges at the end of the chapters in this book as possible.





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'
```

## Interactive Session

**Dictionaries are unordered.**

Try entering these two lines of code in interactive mode and see what happens when you press return.

```
>>> my_dictionary = {"1":"one", "2":"two", "3":"three", "4":"four"}
>>> print(my_dictionary)
```

Is the output in a logical order?

## What's with the brackets?

When we create a new container variable, Python provides us with a quick way of defining which kind we require by the bracket choice:

- 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 any 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 values can be added 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 the Appendix. 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
```

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

Table 1.1 Some useful functions.

## ? Quick Quiz 1.5

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

- 1 A container to store players, personal best scores achieved in a game app, such as: Jeff: 5400, Leela: 12600, etc.
- 2 A container to store the days in a week.
- 3 A container to store the monthly average temperature data for Manchester in 2013.
- 4 A container to store the names of the students who currently attend the climbing club.

## Chapter summary

In this chapter you have:

- learnt more about data types.
- learnt about the container data types: tuples, lists and dictionaries.
- made a short 'magic' card game app.
- 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 learnt a little bit more before providing any puzzles involving them.)

## Puzzle

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

## Challenge 1

- 1 Add some code to `thinkOfACard.py` so that the computer, *Mighty Persistent*, 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 computer 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.”

## Challenge 2

The `thinkOfACard.py` app behaves oddly if the user types anything other than y or n. Add some code to catch unexpected keyboard entries and handle them a little more elegantly.

There are several ways to do these challenges.

To see some example answers go to [http://www.codingclub.co.uk/book5\\_resources.php](http://www.codingclub.co.uk/book5_resources.php).

## Idea 1

You could improve the `thinkOfACard.py` app by adding a tuple of silly comments that the computer randomly says instead of always saying: "Oh, then it must be the ..." such as:

```
silly_comments = ("I never give up, is it the ",  
                 "I cannot believe I am wrong, it must be the ",  
                 "Please let me try again. Is it the ",  
                 "You must REALLY think about the card! Is it the ")
```

See if you can add something like this to your code.

## Idea 2

Have a look at `guessMyCard.py` in the Chapter1 **Answers** folder and see if it inspires you to come up with some other funny card-based programs. (Note the use of `while True:` to create an infinite loop and the key word `break` to get out of it. This can be a very useful construct in game programming.)