A Short Introduction to Computer Programming Using Python

Carsten Fuhs and David Weston
(based on earlier documents by
Sergio Gutierrez-Santos, Keith Mannock, and Roger Mitton)
Birkbeck, University of London

v1.4

This document forms part of the pre-course reading for several MSc courses at Birkbeck.
(Last revision: December 4, 2019.)

Important note: If anything is unclear, or you have any sort of problem (e.g., downloading or installing Python on your computer), please send an email to Carsten Fuhs (*carsten@dcs.bbk.ac.uk*).

1 Introduction

This reading material serves as a preparation for an aptitude test for several MSc programmes at Birkbeck, University of London. This test is an online test that one can take via a web browser, and it only assumes content presented in this document. The questions on the test are similar in style to those in Section 5.

1.1 Recommended further reading

This booklet is designed to be self-contained. However, we will point you to several external resources which we recommend if you are interested in further explanations, additional background, and (also interactive) exercises.

We recommend two electronic textbooks in particular: "Python for Everybody – Exploring Data Using Python 3" by Charles R. Severance [1] and "Think Python – How to Think like a Computer Scientist" by Allen Downey [2]. These textbooks have been made available officially for free download. There are also interactive editions that can be used online, which will give you immediate feedback.

1.2 Motivation

Computing increasingly permeates everyday life, from the apps on our phones to the social networks that connect us. At the same time, computing is a general tool that we can use for solving real-world problems. For you to get a computer to do something, you have to tell it what you want it to do. You give it a series of instructions in the form of a *program*. The computer then helps us by carrying out these instructions very fast, much faster and more reliably than a human. In this sense, a program is a description of how to carry out a process automatically.

Often programs that others have written will do the job. However, we don't want to be limited to the programs that other people have created. We want to tailor our program to the needs of the problem that we have at hand. Writing programs is about expressing our ideas for solving a problem clearly and unambiguously. This is a skill increasingly sought also outside of the software industry. The programs that you can write may then help solve problems in areas such as business, data science, engineering, and medicine.

As a pointer to further reading for this part, you can find an extended description of the purpose and use of writing computer programs in [1, Chapter 1] and [2, Chapter 1].

1.3 Programming language

You write a program in a *programming language*. Many programming languages have been devised over the decades. The language we will use in this document is named *Python*. More precisely, we are using version 3 of the Python language. Python hits a sweet spot: it makes writing simple programs easy and is robust enough for a wide uptake in a variety of industries.

Programming languages vary a lot, but an instruction in a typical programming language might contain some English words (such as `while` or `return`), perhaps a mathematical expression (such as `x + 2`) and some punctuation marks used in a special way.

Programs can vary in length from a few lines to thousands of lines. Here is a complete, short program written in Python:

Example

```
print("Given a series of words, each on a separate line,")
print("this program finds the length of the longest word.")
print("Please enter several sentences, one per line.")
print("Finish with a blank line.")
maxi = 0
word = "."
while len(word) > 0:
    word = input()
    if len(word) > maxi:
        maxi = len(word)

if maxi == 0:
    print("There were no words.")
else:
    print("The longest sentence was " + str(maxi) + " characters long.")
```

When you write a program, you have to be very careful to keep to the syntax of the programming language, i.e., to the grammar rules of the language. For example, the above Python program would behave incorrectly if in the 12^{th} line we wrote

```
if maxi = 0:
```

instead of

```
if maxi == 0:
```

or if in the 9^{th} line we omitted the colon in

```
if len(word) > maxi:
```

The computer would refuse to accept the program if we added dots or colons in strange places or if any of the parentheses (or colons) were missing, or if we wrote `length` instead of `len` or even `Len` instead of `len`.

1.4 Input and output

Almost all programs need to communicate with the outside world to be useful. In particular, they need to read input and to remember it. They also need to respond with an answer.

To get the computer to take some input as text and to store it in its memory, we write in Python:

```
word = input()
```

`input()` is a phrase which has a special meaning to the computer. The combination of these instructions means "Take some input which is a sequence of characters and put it into the computer's memory."

`word` by contrast, is a word that I (the programmer) have chosen. I could have used `characters` or `thingy` or `breakfast` or just `s` or almost any word I wanted. (There are some restrictions which I will deal with in the next section.)

Computers can take in all sorts of things as input — numbers, letters, words, records and so on — but, to begin with, we will write programs that generally handle text as sequences (or *strings*) of characters (like "I", "hi", "My mother has 3 cats", or "This is AWESOME!"). We'll also assume that the computer is taking its input from the keyboard, i.e., when the program is executed, you key in one or more words at the keyboard and these are the characters that the computer puts into its memory.

You can imagine the memory of the computer as consisting of lots of little boxes. Programmers can reserve some of these boxes for use by their programs and they refer to these boxes by giving them names.

```
word = input()
```

means "Take a string of characters input using the keyboard and terminated when the user presses RETURN, and then put this string into the box called `word`." When the program runs and this instruction gets executed, the computer will take the words which you type at the keyboard (whatever you like) and will put them into the box which has the name `word`.

Each of these boxes in the computer's memory can hold only one string at a time. If a box is holding, say, "hello", and you put "Good bye!" into it, the "Good bye!" replaces the "hello". In computer parlance these boxes are called *variables* because the content inside the box can vary; you can put an "I" in it to start with and later change it to a "you" and change it again to "That was a hurricane" and so on as often as you want. Your program can have as many variables as you want.

In Python (and many other programming languages), you have to make sure that you have put something into a variable before you read from it, e.g., to print its contents. If you try to take something out of a box into which you have not yet put anything, it is not clear what that "something" should be. This is why Python will complain if you write a program that reads from a variable into which nothing has yet been put.

If you want the computer to display (on the screen) the contents of one of its boxes, you use `print(thingy)` where instead of `thingy` you write the name of the box. For example, we can print the contents of `word` by writing:

```
print(word)
```

If the contents of the box called `word` happened to be "Karl", then when the computer came to execute the instruction `print(word)` the word "Karl" would appear on the screen.