Chapter 1
Python Basics

Abstract This chapter explains basic programming concepts. After an overview of common Python distributions, we show how to use Python as a simple calculator. As a first step toward programming, variables and expressions are introduced. The arithmetic series and Fibonacci numbers illustrate the concepts of iteration and branching. We conclude this chapter with a program for the computation of a planet's orbital velocity around the Sun, using constants and functions from libraries and giving a small glimpse at objects in Python.

1.1 Using Python

There is quite a variety of Python installations. Depending on the operating system of your computer, you might have some basic Python preinstalled. Typically, this is the case on Linux computers. However, you might find it rather cumbersome to use, especially if you are not well experienced with writing source code in elementary text editors and executing the code on the command line. What is more, installing additional packages typically requires administrative privileges. If you work, for example, in a computer lab it is likely that you do not have the necessary access rights. Apart from that, Python version 2.x (that is major version 2 with some subversion x) is still in use, while this book is based on version 3.x.

Especially as a beginner, you will probably find it convenient to work with a a GUI (graphical user interface). Two popular choices for Python programming are Spyder and Jupyter. Spyder (www.spyder-ide.org) is a classical IDE (integrated development environment) which allows you to edit code, execute it and view the output in different frames. Jupyter (jupyter.org) can be operated via an arbitrary web browser. It allows you to run an interactive Python session with input and output cells (basically, just like the console-based `ipython`). Apart from input cells for typing Python source code, there are so-called markdown cells for writing headers and explanatory text. This allows you to use formatting similar to elementary HTML for webpages. A valuable feature is the incorporation of LaTeX to display mathematical expressions. The examples in this book can be downloaded as Jupyter notebooks and Python source code in zipped archives from uhh.de/phy-hs-pybook.

© Springer Nature Switzerland AG 2021

W. Schmidt and M. Völschow, *Numerical Python in Astronomy and Astrophysics*,
Undergraduate Lecture Notes in Physics,
https://doi.org/10.1007/978-3-030-70347-9_1

Since it depends on your personal preferences which software suits you best, we do not presume a particular GUI or Python distribution here. If you choose to work with Spyder or Jupyter, online documentation and tutorials will help you to install the software and to get started (browse the official documentation under docs.spyder-ide.org and jupyter-notebook.readthedocs.io/en/stable). For a comprehensive guideline, see also [1, appendices A and B]. A powerful all-in-one solution is Anaconda, a Python distribution and package manager that can be installed under Windows, macOS or Linux by any user (see docs.anaconda.com for more information). Anaconda provides a largely autonomous environment with all required components and libraries on a per-user basis. Of course, this comes at the cost of large resource consumption (in particular, watch your available disk space).

As a first step, check if you can run the traditional "Hello, World!" example with your favorite Python installation. Being astronomers, we use a slightly modified version:

```
1 print("Hello, Universe!")
```

In this book Python source code is listed in frames with lines numbered on the left (in the above example, there is just one line). Although these line numbers are not part of the source code (that's why they are shown outside of the frame), they are useful for referring to particular parts of a code example. You might be able to display line numbers in your code editor (in Jupyter notebooks, for example, line numbering can be switched on and off in the View menu), but you should not confuse these numbers with the line numbers used in this book. Usually we will continue the numbering over several frames if the displayed pieces of code are related to each other, but we also frequently reset line numbers to 1 when a new program starts or a new idea is introduced. Whenever you encounter a code line with number 1 it should alert you: at this point something new begins.

After executing the print statement in line 1 above, you should see somewhere on your screen the output[1]

```
Hello, Universe!
```

The quotes in the source code are not shown in the output. They are used to signify that the enclosed characters form a string. As you might have guessed, the `print()` function puts the string specified in parentheses on the screen (more precisely, in a window or frame that is used by Python for output).[2]

[1] How to execute Python code depends on the software you are using (consult the documentation). In a notebook, for example, all you need to do is to simultaneously press the enter and shift keys of your keyboard in the cell containing the code.

[2] Enclosing the string in parentheses is obligatory in Python 3. You may find versions of "Hello, World!" without parentheses on the web, which work only with Python 2.

1.2 Understanding Expressions and Assignments

Apart from printing messages on the screen, which is not particularly exciting by itself, Python can be used as a scientific calculator. Let us begin right away with an example from astronomy. Suppose we want to calculate the velocity at which Earth is moving along its orbit around the Sun. For simplicity, we treat the orbit as circular (in fact, it is elliptical with a small eccentricity of 0.017). From the laws of circular motion it follows that we can simply calculate the velocity as the circumference $2\pi r$ of the orbit divided by the period P, which happens to be one year for Earth. After having looked up the value of π, the orbital radius r (i.e. the distance to the Sun) in km, and the length of a year in seconds,[3] we type

```
1 2*3.14159*1.496e8/3.156e7
```

and, once evaluated by Python, we obtain

```
29.783388086185045
```

for the orbital velocity in km/s. Line 1 is an example for a Python expression consisting of literal numbers and the arithmetic operators `*` and `/` for multiplication and division, respectively. The factor of two in the formula for the circumference is simply written as the integer `2`, while the number π is approximately expressed in fixed-point decimal notation as `3.14159`.[4] The radius $r = 1.496 \times 10^8$ km is expressed as `1.496e8`, which is a so-called floating point literal . The character `e` followed by an integer indicates the exponent of the leading digit in the decimal system. In this case, `e8` corresponds to the factor 10^8. Negative exponents are indicated by a minus sign after `e`. For example, 10^{-3} can be expressed as `1.0e-3` or just `1e-3` (inserting `+` for positive exponents is optional).

Of course, there is much more to Python than evaluating literal expressions like a calculator. To get an idea how this works, we turn the example shown above into a little Python program:

```
1 radius = 1.496e8 # orbital radius in km
2 period = 3.156e7 # orbital period in s
3
4 # calculate orbital velocity
5 velocity = 2*3.14159*radius/period
```

Lines 1, 2, and 5 are examples for assignments. Each assignment binds the value of the expression on the right-hand side of the equality sign `=` to a name on the left-hand side. A value with a name that can be used to refer to that value is in essence what is

[3] Strictly speaking, the time needed by Earth to complete one revolution around the Sun is the *sidereal year*, which has about 365.256 d. One day has 86400 s.

[4] In many programming languages, integers such as `2` are treated differently than floating point numbers. For example, using `2.0` instead of the integer `2` in a division might produce a different result. In Python 3, it is usually not necessary to make this distinction. Alas, Python 2 behaves differently in this respect.

called a variable in Python.[5] In line 5, the variables `radius` and `period` are used to compute the orbital velocity of Earth from the formula

$$v = \frac{2\pi r}{P} \tag{1.1}$$

and the result is in turn assigned to the variable `velocity`. Any text between the hash character # and the end of a line is not Python code but a comment explaining the code to someone other than the programmer (once in a while, however, even programmers might be grateful for being reminded in comments about code details in long and complex programs). For example, the comments in line 1 and 2 provide some information about the physical meaning (radius and period of an orbit) and specify the units that are used (km and s). Line 4 comments on what is going on in the following line.

Now, if you execute the code listed above, you might find it surprising that there is no output whatsoever. Actually, the orbital velocity is computed by Python, but assignments do not produce output. To see the value of the velocity, we can append the print statement

```
6 print(velocity)
```

to the program (since this line depends on code lines 1–5 above, we continue the numbering and will keep doing so until an entirely new program starts), which results in the output[6]

```
29.783388086185045
```

This is the same value we obtained with the calculator example at the beginning of this section.

However, we can do a lot better than that by using further Python features. First of all, it is good practice to print the result of a program much in the same way as, hopefully, you would write the result of a handwritten calculation: It should be stated that the result is a velocity in units of km/s. This can be achieved quite easily by using string literals as in the very first example in Sect. 1.1:

```
7 print("orbital velocity =", velocity, "km/s")
```

producing the output

```
orbital velocity = 29.783388086185045 km/s
```

[5] The concept of a variable in Python is different from variables in programming languages such as C, where variables have a fixed data type and can be declared without assigning a value. Basically, a variable in C is a placeholder in memory whose size is determined by the data type. Python variables are objects that are much more versatile.

[6] In interactive Python, just writing the variable name in the final line of a cell would also result in its value being displayed in the output.

It is important to distinguish between the word 'velocity' appearing in the string `"orbital velocity ="` on the one hand and the variable `velocity` separated by commas on the other hand. In the output produced by **`print`** the two strings are concatenated with the value of the variable. Using such a print statement may seem overly complicated because we know, of course, that the program computes the orbital velocity and, since the radius is given in km and the period in seconds, the resulting velocity will be in units of km/s. However, the meaning of a numerical value without any additional information might not be obvious at all in complex, real-world programs producing a multitude of results. For this reason, we shall adhere to the practice of precisely outputting results throughout this book. The simpler version shown in line 6 may come in useful if a program does not work as expected and you want to check intermediate results.

An important issue in numerical computations is the precision of the result. By default, Python displays a floating point value with machine precision (i.e. the maximal precision that is supported by the way a computer stores numbers in memory and performs operations on them). However, not all of the digits are necessarily significant. In our example, we computed the orbital velocity from parameters (the radius and the period) with only four significant digits, corresponding to a relative error of the order 10^{-4}. Although Python performs arithmetical operations with machine precision, the inaccuracy of our data introduces a much larger error. Consequently, it is pointless to display the result with machine precision. Insignificant digits can be discarded in the output by appropriately formatting the value:

```
8 print("orbital velocity = {:5.2f} km/s".format(velocity))
```

Let us see how this works:

- The method **`format`**`()` inserts the value of the variable in parentheses into the preceding string (mind the dot in between). You will learn more about methods in Sect. 1.4.
- The placeholder `{:5.2f}` controls where and in which format the value of the variable `velocity` is inserted. The format specifier `5.2f` after the colon `:` indicates that the value is to be displayed in fixed-point notation with 5 digits altogether (including the decimal point) and 2 digits after the decimal point. The colon before the format specifier is actually not superfluous. It is needed if several variables are formatted in one print statement (examples will follow later).

Indeed, the output now reads

```
orbital velocity = 29.78 km/s
```

Optionally, the total number of digits in the formatting command can be omitted. Python will then just fill in the leading digits before the decimal point (try it; also change the figures in the command and try to understand what happens). A fixed number of digits can be useful, for example, when printing tabulated data.

As the term variable indicates, the value of a variable can be changed in subsequent lines of the program by assigning a new value. For example, you might want to

calculate the orbital velocity of a hypothetical planet at ten times the distance of the Earth from the Sun, i.e. $r = 1.496 \times 10^9$ km. To that end, we could start with the assignment `radius=1.496e9`. Alternatively, we can make use of the of the current value based on the assignment in line 1 and do the following:

```
9  radius = 10*radius
10 print("new orbital radius = {:.3e} km".format(radius))
```

Although an assignment may appear as the equivalent of a mathematical equality, it is of crucial importance to understand that it is not. Transcribing line 9 into the algebraic equation $r = 10r$ is nonsense because one would obtain $1 = 10$ after dividing through r, which is obviously a contradiction. Keep in mind:

> The assignment operator = in Python means *set to*, not *is equal to*.

The code in line 9 thus encompasses three steps:

(a) Take the value currently assigned to `radius`,
(b) multiply this value by ten
(c) and reassign the result to `radius`.

Checking this with the print statement in line 10, we find that the new value of the variable `radius` is indeed 10 times larger than the original value from line 1:

```
new orbital radius = 1.496e+09 km
```

The radius is displayed in exponential notation with three digits after the decimal point, which is enabled by the formatting type `e` in place of `f` in the placeholder `{:.3e}` for the radius (check what happens if you use type `f` in line 10). You must also be aware that repeatedly executing the assignment `radius = 10*radius` in interactive Python increases the radius again and again by a factor of 10, which is possibly not what you might want. However, repeated operation on the same variable is done on purpose in iterative constructions called loops (see Sect. 1.3).

After having defined a new radius, it would not be correct to go straight to the computation of the orbital velocity since the period of the orbit changes, too. The relation between period and radius is given by Kepler's third law of planetary motion, which will be covered in more detail in Sect. 2.2. For a planet in a circular orbit around the Sun, this relation can be expressed as[7]

$$P^2 = \frac{4\pi^2}{GM} r^3 , \tag{1.2}$$

[7]Here it is assumed that the mass of the planet is negligible compared to the mass of the Sun. For the general formulation of Kepler's third law see Sect. 2.2.

where M=1.989 $\times$ 10^{30} kg is the mass of the Sun and G=6.674 $\times$ 10^{-11} N kg^{-2} m^2 is the gravitational constant. To calculate P for given r, we rewrite this equation in the form

$$P = 2\pi\,(GM)^{-1/2}\,r^{3/2}.$$

This formula can be easily turned into Python code by using the exponentiation operator `**` for calculating the power of an expression:

```
11 # calculate period in s from radius in km (Kepler's third law)
12 period = 2*3.14159 * (6.674e-11*1.989e30)**(-1/2) * \
13          (1e3*radius)**(3/2)
14 # print period in yr
15 print("new orbital period = {:.1f} yr".format(period/3.156e7))
16
17 velocity = 2*3.14159*radius/period
18 print("new orbital velocity = {:.2f} km/s".format(velocity))
```

The results are

```
new orbital period = 31.6 yr
new orbital velocity = 9.42 km/s
```

Hence, it would take more than thirty years for the planet to complete its orbit around the Sun, as its orbital velocity is only about one third of Earth's velocity. Actually, these parameters are quite close to those of the planet Saturn in the solar system. The backslash character `\` in line 12 is used to continue an expression that does not fit into a single line in the following line (there is no limitation on the length of a line in Python, but code can become cumbersome to read if too much is squeezed into a single line). An important lesson taught by the code listed above is that you always need to be aware of physical units when performing numerical calculations. Since the radius is specified in km, we obtain the orbital velocity in km/s. However, the mass of the Sun and the gravitational constants in the expression for the orbital period in lines 12–13 are defined in SI units. For the units to be compatible, we need to convert the radius from km to m. This is the reason for the factor 10^3 in the expression `(1e3*radius)**(3/2)`. Of course, this does not change the value of the variable `radius` itself. To avoid confusion, it is stated in the comment in line 11 which units are assumed. Another unit conversion is applied when the resulting period is printed in units of a year in line 15, where the expression `period/3.156e7` is evaluated and inserted into the output string via **`format`**. As you may recall from the beginning of this section, a year has 3.156 $\times$ 10^7 s.

Wrong unit conversion is a common source of error, which may have severe consequences. A famous example is the loss of NASA's Mars Climate Orbiter due to the inconsistent use of metric and imperial units in the software of the spacecraft.[8] As a result, more than $100 million were quite literally burned on Mars. It is therefore extremely important to be clear about the units of all physical quantities in a

[8] See mars.jpl.nasa.gov/msp98/orbiter/.

program. Apart from the simple, but hardly foolproof approach of using explicit conversion factors and indicating units in comments, you will learn different strategies for ensuring the consistency of units in this book.

1.3 Control Structures

The computation of the orbital velocity of Earth in the previous section is a very simple example for the implementation of a numerical algorithm in Python.[9] It involves the following steps:

1. Initialisation of all data needed to perform the following computation.
2. An exactly defined sequence of computational rules (usually based on mathematical formulas), unambiguously producing a result in a finite number of steps given the input from step 1.
3. Output of the result.

In our example, the definition of the variables `radius` and `period` provides the input, the expression for the orbital velocity is a computational rule, and the result assigned to the variable `velocity` is printed as output.

A common generalization of this simple scheme is the repeated execution of the same computational rule in a sequence of steps, where the outcome of one step is used as input for the next step. This is called iteration and will be explained in the remainder of this section. The independent application of the same operations to multiple elements of data is important when working with arrays, which will be introduced in Chap. 2.

Iteration requires a control structure for repeating the execution of a block of statements a given number of times or until a certain condition is met and the iteration terminates. Such a structure is called a loop. For example, let us consider the problem of summing up the first 100 natural numbers (this is a special case of an arithmetic series, in which each term differs by the previous one by a constant):

$$s_n \equiv \sum_{k=1}^{n} k = 1 + 2 + 3 + \ldots + n\,. \tag{1.3}$$

[9] The term algorithm derives from the astronomer and mathematician al-Khwarizmi whose name was transcribed to *Algoritmi* in Latin (cf. [2] if you are interested in the historical background). al-Khwarizmi worked at the *House of Wisdom*, a famous library in Bagdad in the early 9th century. Not only was he the founder of the branch of mathematics that became later known as algebra, he also introduced the decimal system including the digit 0 in a book which was preserved until the modern era only in a Latin translation under the title *Algoritmi de numero Indorum* (this refers to the origin of the number zero in India). The digit 0 is quintessential to the binary system used on all modern computers.