Python for Econometrics and Operations Research

Sander Gribling   Pieter Kleer   Johan van Leeuwaarden

Sven Polak

ii

Table of contents

Chapter 1

Welcome

1.1 Welcome

Welcome to the online "book" that serves as an introduction to the programming language Python, that you will see in various courses throughout the Econometrics and Operations Research bachelor program.

Before we jump into coding with Python, we will start by discussing what programming is at the most basic level and motivating why we are learning how to code in Python in the first place.

1.2 What is a programming language?

Without getting into a complicated details, a programming language is a way to communicate to a computer via written text in a way that the computer can understand you so that you can instruct it to do various operations. This is very different to how we often usually interact with a computer, which often involves pointing and clicking on different buttons and menus with your mouse.

Knowing how to program is a very useful skill because you can automate repetitive tasks that would otherwise take you a very long time if you had to them "by hand" (i.e. by clicking things with your mouse). For example, suppose you work in a hotel in a city and you need to check how much your competitors are charging for rooms on different days so that you can adjust prices to stay competitive. Every day you have to go to all the different websites of the competing hotels and take note of the prices in an Excel sheet. With programming, what you could do instead is write code that tells the computer to automatically visit those websites every day, record the hotel room prices, and put them into a dataset for you. This is a process called *web scraping* and can be done with Python. This is just one example of the many ways programming languages

can automate repetitive tasks.

When humans speak to each other and someone makes a grammar mistake, it usually isn't a big deal. We usually know what they mean. But if you make a "syntax error", i.e, grammar mistake in a programming language, it won't understand what you mean. The computer will throw an error. What is worse still is a "semantic error" which is when the computer runs your code without an error but does something you didn't want it to do. Therefore we need to be very careful when writing in a programming language.

1.3   Why Python?

There are many different programming languages out there: C, C++, C#, Java, JavaScript, R, Julia, Stata, MATLAB, Fortran, Ruby, Perl, Rust, Go, Lua, Swift - the list goes on. So why should we learn Python over these other alternatives?

The best programming language depends on the task you want to accomplish. Are you building a website, writing computer software, creating a game, or analyzing data? While many languages could perform all of these tasks, some languages excel in some of them. In this course our goal is to learn basic programming techniques required for data science, and Python is by far the most popular programming language for this task. But it's not only useful for that. It is also often used in web development, creating desktop applications and games, and for scientific computations. It is therefore a very versatile programming language that can complete a very wide range of tasks.

Python is also completely free and open source and can run on all common operating systems. This means you can share your code with anyone and they will be able to run it, no matter what computer they are on or where they are in the world.

There is also a very large active community that creates packages to do a wide-range of operations, keeping Python up to date with the latest developments. For example, excellent community help is available at Stackoverflow, so if you Google how to do something in Python most likely that question has already been answered on Stackoverflow. Funnily enough, a key skill to develop with programming is how to formulate your question into Google to land on the right Stackoverflow page. More recently, "large-language" models like ChatGPT have become a very useful resource for Python. ChatGPT can write excellent Python code and also explains all the steps it takes, so we encourage you to use it as a tool to help you when you are stuck. You should keep in mind though that throughout the bachelor program, you will not always be allowed to use tools like ChatGPT.

These days employers are increasingly looking to hire people with programming skills. Knowing how to program in Python - one of the most commonly used languages by companies - is therefore a very valuable addition to your CV.

Chapter 2

Installation

In this chapter we will learn how to install Python and run our very first command.

2.1 Installing Anaconda

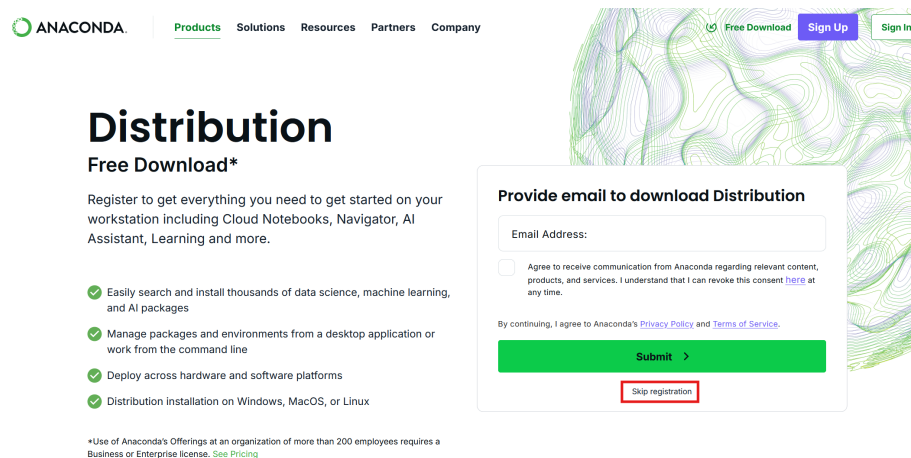The easiest way to install Python is by installing Anaconda. You can do this by visiting https://www.anaconda.com/download.

You should see this page:

Figure 2.1: Anaconda Download Page

You should click the "Skip registration" button (although feel free to register if

you like). You will then see the following page:
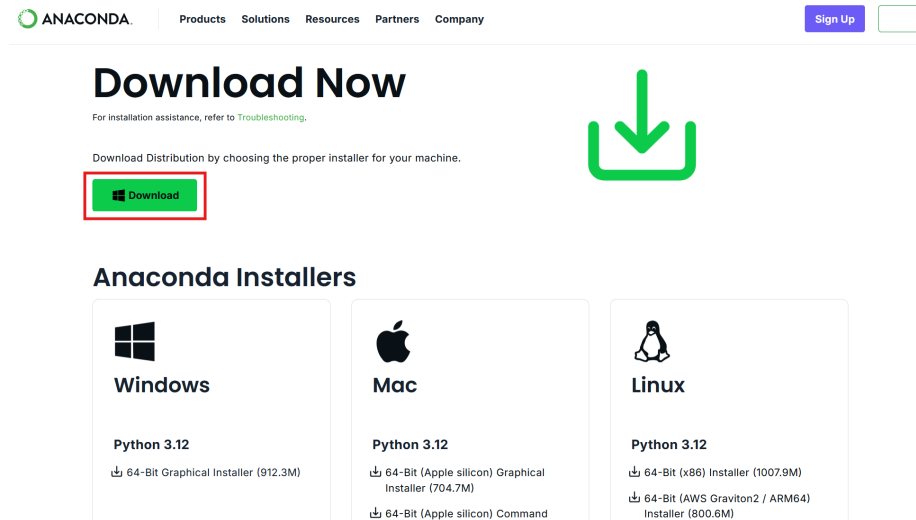
Figure 2.2: Anaconda Download Page

You should then click on the "Download" button. Mac users will see a Mac logo instead.

After downloading the file, click on it to install it. Follow the installation wizard and keep all the default options during installation.

After installation you will see a number of new applications on your computer. You can see all applications that were installed using *Anaconda Navigator*.
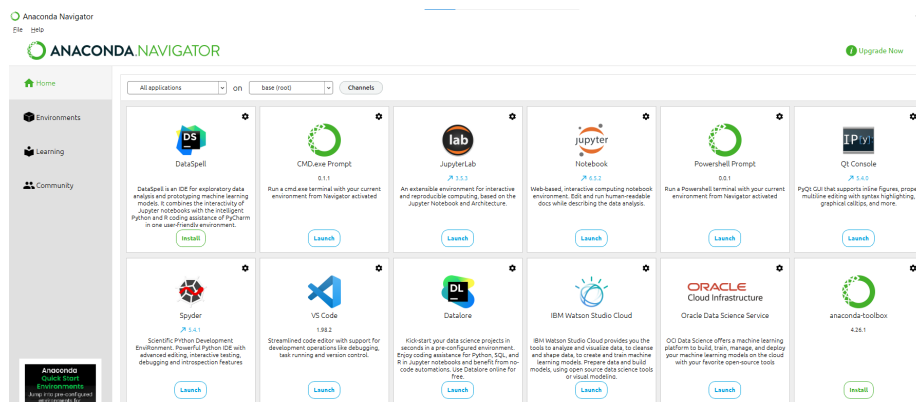

Figure 2.3: Anaconda Navigator

We highlight two applications:

- *Jupyter Notebook.* This is a web application that allows you to write a notebook (like a report) with text and Python code snippets with output. We will learn how to use this application later on.
- *Spyder/VS Code.* These are computer applications that allows you to write Python scripts and execute them to see the output. Such an application is called an Integrated Desktop Environment (IDE).

2.2 Jupyter Notebook

You can open the Jupyter Notebook application either by pressing 'Launch' in the Anaconda Navigator, or you can search for the application directly on your (university) computer via the Start menu.

The application will open as a tab in a web browser, and you should then see a list of folders.

Figure 2.4: Jupyter Notebook application

You can navigate to a folder and then create a new notebook by clicking on 'New' in the top-right and then selecting 'Python 3 (ipykernel)' under Notebook.
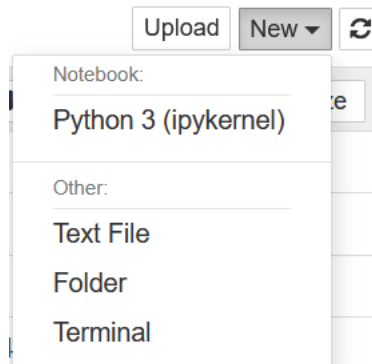
Figure 2.5: Creating notebook

The new notebook will open in another tab and is stored in the folder in which you created it, typically under the name 'Untitled.ipynb'. You can change the name of the file either in the folder in which you stored it, or via `File -> Rename` in the top-left corner. You should see the empty notebook as below.
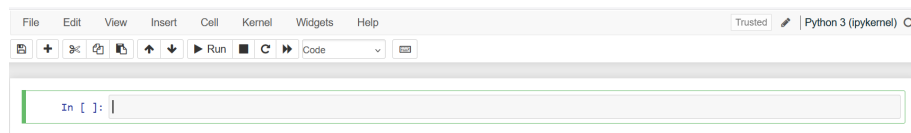
Figure 2.6: New notebook

In the bar you can type Python code. Let us execute our first code, which is a simple calculation $1+1$! To find $1+1$ in Python, we can use the command `1+1`, similar to how we would do it in Excel or in the Google search engine. Let's try this out. Type 1+1 in the code bar and click 'Run'. We will see the output 2 on the next line next to a red Out [1]:
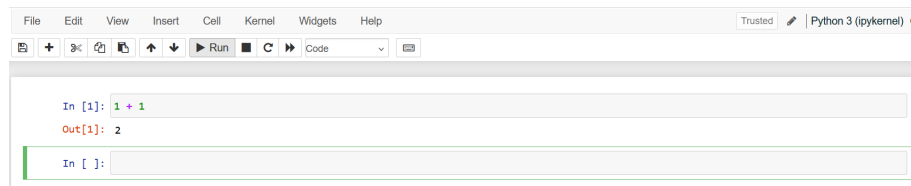
Figure 2.7: First Python code

The red Out [1] means this is the output from the first line of input (after the green In [1]). If we continue typing code in the second bar, it will be called In [2] and its output Out [2].

2.3   Code Snippets in This Book

In this book, we won't always show screenshots like we did above. Instead we will show code snippets in boxes like this:

```
1 + 1
```

```
2
```

The part that is code will be in color and there will be a small clipboard icon on the right which you can use to copy the code to paste into your own Notebook to be able to experiment with it yourself. The output from the code will always be in a separate gray box below it (without a clipboard icon).

Chapter 3

Some (math) basics

In this chapter we will learn how to use Python as a calculator. In Chapter 2 we already saw how to calculate `1 + 1`. We will now go through some different operations. We will also learn about *functions* and their *arguments* along the way, which we will be be using again and again throughout the rest of this course.

3.1 Arithmetic operations

We start with the most basic arithmetic operations: Addition, subtraction, multiplication and division are given by the standard `+`, `-`, `*` and `/` operators that you would use in other programs like Excel. For example, addition:

```
2 + 3
```

```
5
```

Subtraction:

```
5 - 3
```

```
2
```

Multiplication:

```
2 * 3
```

```
6
```

Division:

```
3 / 2
```

```
1.5
```

It is also possible to do multiple operations at the same time using parentheses. For example, suppose we wanted to calculate:

$$\frac{2+4}{4\times 2} = \frac{6}{8} = 0.75$$

We can calculate this in Python as follows:

```
(2 + 4) / (4 * 2)
```

```
0.75
```

With the `**` operator (two stars) we can raise a number to the power of another number. For example, $2^3 = 2\times 2\times 2 = 8$ can be computed as

```
2 ** 3
```

```
8
```

Be very careful **not** to use `^` for exponentiation. This actually does a very different thing in Python that we won't have any use for in this book.

> 💡 Exercise 3.1
>
> Compute the following expressions using the operator $+, -, *, /$ and $**$:
>   1.
>   2.
>   3.

> 🔥 Solution
>
>   1.
>   2.
>   3.

3.2   Variables

In Python we can assign single values to *variables* and then work with and manipulate those variables.

Assigning a single value to a variable is very straightforward. We put the name we want to give to the variable on the left, then use the `=` symbol as the *assignment operator*, and put the value to the right of the `=`. The `=` operator binds a value (on the right-hand size of `=`) to a name (on the left-hand side of `=`).

To see this at work, let's set $x = 2$ and $y = 3$ and calculate $x + y$:

```
x = 2
y = 3
x + y
```

```
5
```

When we assign $x = 2$, in our code, the value is not fixed forever. We can assign a new value to x. For example, we can assign the number 6 to x instead. The sum of x, which is 6, and y, which is 3 is now 9:

```
x = 6
x + y
```

```
9
```

Finally, you cannot set $x = 2$ with the command `2 = x`. That will result in an error. The name must be on the left of = and the value must be on the right of =.

> 💡 **Exercise 3.2**
>
> Define variables a, b, c with values $19, 3, 7$, respectively and compute the expressions
>   1. $a \cdot b \cdot c$
>   2.
>   3.

> 🔥 **Solution**
>
>   1.
>   2.
>   3.

3.3 Lists

We can also store multiple variables in one object, a so-called *list*. A list with values is created by writing down a sequence of numbers, separated by commas, in between two brackets [and].

```
z = [3, 9, 1, 7]
z
```

```
[3, 9, 1, 7]
```

We can also create lists with fractional numbers.

```
z = [3.1, 9, 1.9, 7]
z
```

```
[3.1, 9, 1.9, 7]
```

To access the numbers in the list, we can *index* the list at the position of interest. If we want to get the i-th number in the list, we use the syntax `z[i]`.

```
z[1]
```

```
9
```

Something strange is happening here... The first number in the list is 3.1, but `z[1]` returns 9, which is the second value in the list. This is because Python starts counting at 0. In other words, if we want to get the first number in the list, we should use `z[0]` instead.

Below we index all values separately.

```
z[0]
```

```
3.1
```

```
z[1]
```

```
9
```

```
z[2]
```

```
1.9
```

```
z[3]
```

```
7
```

💡 Exercise 3.3

Consider the list $a = [1, 4, 2, 5, 6, 3, 7]$.
1. Compute the sum of the numbers at even positions in a (i.e., positions $0, 2, 4$ and 6)
2.
3.

🔥 Solution

1.
2.
3.

3.4 Python function

If we want to compute a certain mathematical expression for many different variables, it is often convenient to use a Python function for this.

For example, consider the quadratic function $f(x) = 3x^2 + 2x^2 - 1$. Say we want to know the values of $f(-3), f(-2.5), f(1)$ and $f(4)$. What we would like to do is to 'automate' the computation of a function value, so we do not have to repeat this everytime.

For this we can use a Python function for this as follows.

```python
def f(x):
    return 3*x**2 + 2*x - 1
```

What does the code above do? First of all the syntax to tell Python we want to define a function called `f` that takes as input a number `x` is `def f(x):`.

We next have to tell Python what the function is supposed to compute. On the second line, with one tab indented, we have the **return** statement. Here we write down the expression that the function should compute, which in our case is the function value $f(x) = 3x^2 + 2x^2 - 1$.

We can now compute the function value $f(x)$ for any value of x. What happens is that Python *calls* the function f with input the chosen value of x, and then returns the function value $f(x)$, i.e., the expression in the **return** statement.

```python
f(-3)
```

```
20
```

```python
f(1)
```

```
4
```

Note that you can also name the function differently, for example we could also have done `def quadratic_function(x):`. The important thing is that the name of the function should be the same as the name you use in the command to compute a function value.

```python
def quadratic_function(x):
    return 3*x**2 + 2*x - 1

quadratic_function(1)
```

```
4
```

3.5 Conditional statements

Here I want to conclude with if/else statements to compute the roots of a function using the 'abc'-formula. There will be a case distinction based on whether the discriminant is positive, negative or zero.

At this point we could zoom out, and talk about that Python can handle much more complex root finding problems (as sort of a teaser).