Python/R for Data Science 2022 Fall

Xinli Xiao

2022-06-22

Contents

1 About **5**

Part I: Python 9

2 Python Fundamentals **9**

2.1 Why Python? 9

2.2 Hello world! 10

2.3 Built-in Types 12

2.4 Flows 13

2.5 Logic 13

2.6 Exercises 13

2.7 Projects 15

3 more advanced techniques **17**

3.1 List 17

3.2 dict 17

3.3 numpy and pandas 17

3.4 exercises 17

3.5 proj 17

4 Classes/Packages for Python **19**

4.1 functions 19

4.2 packages 19

4.3 classes 19
4.4 virenev 19
4.5 ex 19
4.6 proj 19

5 Python Notebooks **21**
5.1 notebooks 21
5.2 ex 21
5.3 proj 21

Part II: R **25**

6 R Fundamentals **25**
6.1 Hello world for R 25

Chapter 1

About

This is the lecture notes for R part in STAT 2304 during 2022 Fall semester. There are many ways to write content in Jupyter Book. This short section covers a few tips for how to do so.

```
bookdown::serve_book()
```

Part I: Python

Chapter 2

Python Fundamentals

2.1 Why Python?

2.1.1 Python is easy to use

Programmers familiar with traditional languages will find it easy to learn Python. All of the familiar constructs—loops, conditional statements, arrays, and so forth—are included, but many are easier to use in Python. Here are a few of the reasons why: - Types are associated with objects, not variables. A variable can be assigned a value of any type, and a list can contain objects of many types. This also means that type casting usually isn't necessary and that your code isn't locked into the straitjacket of predeclared types. - Python typically operates at a much higher level of abstraction. This is partly the result of the way the language is built and partly the result of an extensive standard code library that comes with the Python distribution. A program to download a web page can be written in two or three lines! - Syntax rules are very simple. Although becoming an expert Pythonista takes time and effort, even beginners can absorb enough Python syntax to write useful code quickly.

Python is well suited for rapid application development. It isn't unusual for coding an application in Python to take one-fifth the time it would in C or Java and to take as little as one-fifth the number of lines of the equivalent C program. This depends on the particular application, of course; for a numerical algorithm performing mostly integer arithmetic in for loops, there would be much less of a productivity gain. For the average application, the productivity gain can be significant.

2.1.2 Python is expressive

2.1.3 Python is readable

Klosterman [2021] Ramalho [2015] Shaw [2017a] Shaw [2017b] Ceder [2018] Youens-Clark [2020]

2.2 Hello world!

We will mainly focus on this code editor mode at the beginning and check our results or do some simple computations in the console.

Notebook is another very popular mode to use Python. We will talk about it later.

2.2.1 Set up Python environment using IDEs

Please follow the following steps to run your first line of Python codes.

We will talk about the relation between Python and Anaconda and more about packages sometime later.

1. Go to Anaconda download page. Download and install Anaconda.
2. There are several IDEs for Python bundled with Anaconda. Pick any one you like. I personally use VS Code. Here we use Spyder as an example for now since it doesn't require any configurations.
3. Here is a screenshot of Spyder 5.1.5.
4. The right lower window is the console. Type the following code, and run. If `Hello world!` is displayed, the Python environment is set up successfully. Now you can start to play with Python!

```
print('Hello world!')
```

```
## Hello world!
```

2.2.2 Code editor

The left window is called *Code Editor*. You can write multiple lines of codes in the code editor and run them all together. The output results might appear in the console.

As shown in the screenshot, when press `F5` to run file, the codes in the code editor will be excuted line by line.

The code in the example is

```
print('Hello world!')
```

```
## Hello world!
```

```
print('Another line')
# Everything after # are comments that won't be excuted.
```

```
## Another line
```

2.2.3 Indentation

One key feature about Python is that its structures (blocks) is determined by **Indentation**.

Let's compare with other languages. Let's take `C` as an example.

```
#/*This is a C function.*/
#int f(int x){return x;}
```

The block is defined by `{}` and lines are separated by `;`. `space` and `newline` are not important when `C` runs the code. It is recommended to write codes in a "beautiful, stylish" format for readibility, as follows. However it is not mandatary.

```
#/*This is a C function.*/
#int f(int x) {
#    return x;
#}
```

In Python, blocks starts from `:` and then are determined by indents. Therefore you won't see a lot of `{}` in Python, and the "beautiful, stylish" format is mandatary.

```
# This is a Python function.
def f(x):
    return x
```

The default value for indentation is 4 spaces, which can be changed by users. We will just use the default value in this course.

2.3 Built-in Types

There are severl built-in data structures in Python. Here is an (incomplete) list: - `None` - Boolean – `True`, `False` - Numeric Types — `int`, `float`, `complex` - Sequence Types — `list` - Text Sequence Type — `str` - Map type - `dict`

We will cover numeric types and strings in this section. The rests are either simple that are self-explained, or not simple that will be discussed later.

You may always use `type(x)` to detect the type of the object `x`.

2.3.1 Numeric types and math expressions

Numeric types are represented by numbers. If there are no confusions, Python will automatically detect the type.

```
x = 1 # x is an int.
y = 2.0 # y is a float.
```

Python can do math just like other programming languages. The basic math operations are listed as follows. - `+`, `-`, `*`, `/`, `>`, `<`, `>=`, `<=` works as normal. - `**` is the power operation. - `%` is the mod operation. - `!=` is `not equal`

2.3.2 Strings

Scalars are represented by numbers and strings are represented by quotes. Example:

```
x = 1        # x is a scalar.
y = 's'      # y is a string with one letter.
z = '0'      # z loos like a number, but it is a string.
w = "Hello" # w is a string with double quotes.
```

Here are some facts. 1. For strings, you can use either single quotes or double quotes. 2. `\` is used to denote escaped words. You may find the list Here. 3. There are several types of scalars, like `int`, `float`, etc.. Usually Python will automatically determine the type of the data, but sometimes you may still want to declare them manually. 4.

There is a very subtle relations between the variable / constant and the name of the variable / constant. We

2.4 Flows

2.5 Logic

2.6 Exercises

2.6.1 Indentation

Please tell the differences between the following codes. If you don't understand `for` don't worry about it. Just focus on the indentation and try to understand how the codes work.

```
for i in range(10):
    print('Hello world!')
```

```
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
```

```
print('Hello world!')
```

```
## Hello world!
```

```
for i in range(10):
    print('Hello world!')
    print('Hello world!')
```

```
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
```

```
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
## Hello world!
```

```
# for i in range(10):
# print('Hello world!')
# print('Hello world!')
```

```
for i in range(10):
    pass
print('Hello world!')
```

```
## Hello world!
```

```
print('Hello world!')
```

```
## Hello world!
```

2.6.2 Play with built-in data types

```
print(True and True)
```

```
## True
```

```
print(True or True)
```

```
## True
```

```
print(False and True)
```

```
## False
```

```
print((1+1>2) or (1-1<1))
```

```
## True
```

2.7 Projects

jupyter notebook

2.7.1 Gnomic data

lists of data

Chapter 3

more advanced techniques

3.1 List

3.2 dict

3.3 numpy and pandas

3.4 exercises

3.5 proj

3.5.1 key

3.5.2 project

- open and parse a CSV file
- do some operations
- decipher (replace letters / frequency analysis)

Chapter 4

Classes/Packages for Python

4.1 functions

4.2 packages

4.3 classes

4.4 virenev

4.5 ex

4.6 proj

- clean commented code, use of functions
- write classes
- folllow-up: geospatial data based analysis in python

Chapter 5

Python Notebooks

5.1 notebooks

5.2 ex

5.3 proj

Part II: R

Chapter 6

R Fundamentals

ddd

6.1 Hello world for R

Bibliography

Naomi R. Ceder. *The Quick Python Book*. MANNING PUBN, June 2018. ISBN 978-1617294037. URL https://www.manning.com/books/the-quick-python-book-third-edition.

Stephen Klosterman. *Data Science Projects with Python*. Packt Publishing, Limited, 2021. ISBN 978-1800564480. URL https://www.packtpub.com/product/data-science-projects-with-python/9781838551025.

Luciano Ramalho. *Fluent Python : Clear, Concise, and Effective Programming*. O'Reilly Media, 2015. ISBN 978-1492056355. URL https://www.oreilly.com/library/view/fluent-python-2nd/9781492056348/.

Zed A. Shaw. *Learn Python 3 the Hard Way*. Addison Wesley, June 2017a. ISBN 0134692888. URL https://www.pearson.com/us/higher-education/program/Shaw-Learn-Python-3-the-Hard-Way-A-Very-Simple-Introduction-to-the-Terrifyingly-Beautiful-World-of-Computers-and-Code/PGM1768668.html.

Zed A. Shaw. *Learn More Python 3 the Hard Way: The Next Step for New Python Programmers*. ADDISON WESLEY PUB CO INC, September 2017b. ISBN 978-0134123486. URL https://www.pearson.com/us/higher-education/program/Shaw-Learn-More-Python-3-the-Hard-Way-The-Next-Step-for-New-Python-Programmers/PGM285143.html.

Ken Youens-Clark. *Tiny Python Projects*. Manning Publications, November 2020. ISBN 9781617297519. URL https://www.manning.com/books/tiny-python-projects.