Python from Scratch

Peter Harpending <peter.harpending@utah.edu>

2016-08-07

Contents

Chapter 1

Getting set up

Hi there, this is a beginner's guide to Python 3. In this chapter, we're just going to get your computer set up, and run a very simple Python program on it.

You will want to install all of these things:

1. the Python 3 environment;
2. a plain-text editor;
3. if you're on Windows, Cygwin: `https://www.cygwin.com/`.

If you're on a UNIX-like system, such as Linux, OS X, or a BSD system, Python is probably already installed, and a text editor and terminal emulator are almost certainly installed. The only hiccup might be the version of Python.

On Windows, you'll want to download the latest Python 3 release from here: `https://www.python.org/downloads/windows/`. (See fig. 1.1.)

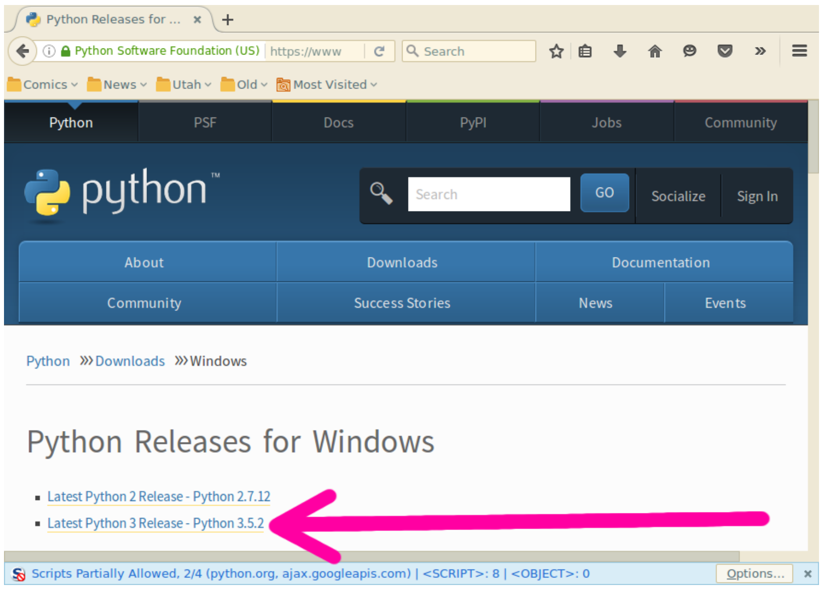
As for a "plain-text editor", that means a program to edit text files that is critically **not a word processor**. There are hundreds of text editors, and experienced programmers are usually rather opinionated on their favorite text editor.

1. For the average new programmer, I'd recommend Atom: `https://atom.io/`. Atom is not a terrible choice for experienced developers, either.

2. If you are willing to learn how to use it, Vim is an excellent terminal-based editor: `http://www.vim.org/`. The poorly-designed website is not reflective of the quality of the text editor. Vim is probably the most popular text editor.

3. If you have several months to spare, and are already an experienced programmer, GNU Emacs is also very popular: `http://www.gnu.org/software/emacs/`. (This is what I use). In this case, the fast, and well-designed website is not reflective of the quality of the text editor.

4. If you are on Windows, I've heard good things about Notepad++: `https://notepad-plus-plus.org/`, although I've never tried it for more than an hour or two.



Python Releases for Windows

- Latest Python 2 Release - Python 2.7.12
- Latest Python 3 Release - Python 3.5.2

Scripts Partially Allowed, 2/4 (python.org, ajax.googleapis.com) | <SCRIPT>: 8 | <OBJECT>: 0

Figure 1.1: Link you should click on for Python 3, iff you're using Windows.

Throughout this book, you'll need to run a lot of commands in the terminal. On UNIX-like systems, you can open an application called "Terminal," "Terminal Emulator," or some variant thereof, and use that. On Windows, use Cygwin. I will prefix commands you should run in the terminal with a dollar sign. Expected output of the program follows in subsequent lines.

```
$ uname -a
Linux centurion 4.4.0-31-generic #50-Ubuntu SMP Wed Jul 13 00:07:12 UTC
    ↪ 2016 x86_64 x86_64 x86_64 GNU/Linux
```

Remark 1.0.1. The red arrow just indicates that the previous line overflows onto the next line. It does not indicate an actual output character.

Remark 1.0.2. It's extremely important that you type in code examples and terminal commands yourself, and observe the results, rather than glancing at them in the book, or copying & pasting.

To test to see if Python 3 is installed on your system, first run `which` ↪ `python` in a terminal.

```
$ which python
/usr/bin/python
```

Then, run `python --version` to make sure it's using Python 3:

```
$ python --version
Python 2.7.12
```

On my system (Ubuntu 16.04), Python 2 is still the default. However, the command `python3` gets me to Python 3.

```
$ python3 --version
Python 3.5.2
```

The command to run Python 3 will vary from one system to another, so therefore I will just say "run the appropriate `python` command," to refer to `python`, `python3`, or whatever it is on your system.

If you run `python3` with no arguments, you'll get Python's **Read/Eval/Print Loop**, or **REPL**.

```
$ python3
Python 3.5.2 (default, Jul  5 2016, 12:43:10)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Remark 1.0.3. The three 'greater-than' signs are the standard Python REPL prompt. Therefore, if I want to indicate that you should type something into the Python REPL, I'll just prefix the line with `>>>`.

At the REPL, you can type in a line of Python code. Python will evaluate it and print the result. Then, under most circumstances, Python will prompt you for another line of code.[1] Examples:

```
>>> print('hello world')
hello world
>>> 2 + 2
4
>>> 3 * 2
6
>>> 17 / 8
2.125
```

Remark 1.0.4. The syntax highlighting is just there to help with readability; it doesn't actually mean anything.

Saving in files

You can type entire programs into the REPL, and it will evaluate them. However, you will usually want to save them in a file, so you can use them later, and not have to type them out every time. Before we do that, you need a small amount of background information.

I'd recommend creating a special directory for the programs in this text. If you can't think of a name, I'd suggest `code`. To make a directory, use the `mkdir` command:

```
$ mkdir code
```

There are a lot of UNIX commands, but here are the commands you'll actually need to know as we move forward:

1. `mkdir` is for "make directory."
2. `cd` is for "change directory."
3. `pwd` is for "print working directory." (Prints the current directory out).
4. `ls` is for "list." (Prints the contents of the current directory)
5. `cp` is for "copy."

[1]The circumstance in which Python won't ask you for a new line of code is if you tell Python to quit via the `exit` command.

6. `mv` is for "move." (Or "rename," if you prefer.)

7. Most importantly, `man` is for "manual." It will print out the manual for any command. I highly recommend glancing through the manual for all of the commands I just mentioned. For instance:

```
$ man pwd         # Shows the manual for the 'pwd' command
$ man man         # Shows the manual for the 'man' command
```

   To exit the manual, press `q`. The `#` indicates that the rest of the line is a **comment**, and is ignored by the shell (program that interprets & executes terminal commands).

   One of the most important options for `man` is `man -k`, which searches for manuals, or, in this case **man pages**.

```
$ man -k python3
2to3 (1)            - Python2 to Python3 converter
2to3-2.7 (1)        - Python2 to Python3 converter
2to3-3.5 (1)        - Python2 to Python3 converter
dh_python3 (1)      - calculates Python dependencies, adds
    ↪ maintainer scripts to byte compile files, etc.
py3versions (1)     - print python3 version information
python3 (1)         - an interpreted, interactive, object-oriented
    ↪ programming language
python3.5 (1)       - an interpreted, interactive, object-oriented
    ↪ programming language
python3.5m (1)      - an interpreted, interactive, object-oriented
    ↪ programming language
python3m (1)        - an interpreted, interactive, object-oriented
    ↪ programming language
```

Your first Python program run from a file. Now that you know the basics of UNIX, we're going to run a very small program from a file.

Python files typically have the `.py` file extension. Here's the `hello world` program in a file:

```python
print('hello world')
```

Use your text editor to write that program to a file called `hello_world.py`, then run this command:

```
$ cd code
$ python3 hello_world.py
hello world
```

Chapter 2

The greatest common divisor

Our goal at the end of this chapter is to write a function that computes the greatest common divisor of two positive integers. I'll try to develop the appropriate math background, and the appropriate Python skills so that you'll understand what's going on at the end of the chapter.

2.1 Mathematical background

First of all, what are the "positive integers"? **Integers** are whole numbers:

$$\mathbb{Z} = \{ 0, \pm 1, \pm 2, \pm 3, \dots \} .$$

The **natural numbers** are the positive integers:

$$\mathbb{N} = \{ 1, 2, 3, \dots \} .$$

Remark 2.1.1. Some people use 0 as the first natural number. It doesn't matter a whole not. In our case, it's more convenient to have 1 as the first natural number.

Sets

Before we go much further, I'd like to introduce the concept of a **set**. A set is a collection of objects, called **elements**, with no notion of order or multiplicity. Finite sets with few elements are typically denoted with braces:

$$\{ 1, 2, 3, 4 \} \text{ is a set.}$$

The things distinguishing sets from **lists** or **vectors** are:

1. in a set, the order in which elements appear is irrelevant;
2. in a set, the number of times that an element appears is irrelevant.

Therefore, all of the following are the same set:

$$\begin{aligned} & \{1,2,3,4\} \\ =& \{1,1,2,2,3,3,4,4\} && \text{(Each element is duplicated).} \\ =& \{1,3,3,4,1,4,2\} && \text{(Different order, and some duplication).} \end{aligned}$$

Given a set A, and an element x, to say

$$x \text{ is an element of } A,$$

we write

$$x \in A.$$

If

$$x \text{ is not an element of } A,$$

then we write

$$x \notin A.$$

Set comprehension notation. This notation is common:

$$\{\,\text{what each element looks like} \;:\; \text{conditions about each element}\,\}$$

In Python, we have list comprehensions, which are similar:

```
>>> [2*x for x in [1,2,3,4]]
[2, 4, 6, 8]
>>> [3*x for x in [1,2,3,4]]
[3, 6, 9, 12]
>>> [x + 7 for x in [1,2,3,4]]
[8, 9, 10, 11]
```

Functions

A **mathematical function** takes elements from one set and maps them to elements in another set. For instance,

$$f : \mathbb{N} \to \mathbb{N}$$
$$f(x) = x + 3$$

Python has things called "functions", but they are slightly different. Here's how we would define that function in Python:

```python
def f(x):
    return x + 3
```

The difference comes down to a property called **referential transparency**:

Axiom 2.1.2. *If $f : A \to B$, and $x, y \in A$, then,*

$$\textit{if } x = y, \textit{ then } f(x) = f(y).$$

Functions in Python do not usually have this property. For instance,

```
>>> import random
>>> random.randint(1, 100)
44
>>> random.randint(1, 100)
10
```

I called the same "function" with the same arguments, and got a different result. That cannot happen in a mathematical function.

The following definitions are useful computations that we'll be using:

Definition 2.1.3. The **Cartesian product** of two sets A and B is the set of ordered pairs where the first value is in A, and the second value is in B. That is,

$$A \times B = \{\, (a, b) \; : \; a \in A, \text{ and } b \in B \,\}.$$

Definition 2.1.4.

Division with remainder

This is the way you learned to divide in elementary school. We want a mathematical function, that, given two positive integers $a, b \in \mathbb{N}$, returns the **quotient** of a and b, plus a **remainder**, which might be 0.

Lemma 2.1.5. *Given two positive integers $a, b \in \mathbb{N}$, with $a \geq b$ we can always write*

$$a = qb + r,$$

*where $q \in \mathbb{N}$ is the **quotient**, and r is the **remainder**. The remainder can be zero, but it must be less than b. Therefore,*

$$r \in \{\, x \in \mathbb{Z} \,:\, 0 \leq x < b \,\}\,.$$