Installing and running Python: the bare minimum you can get away with

Spoiler: it's more complicated on Linux. Because of course.

APR 03, 2023

Summary

In "Relieving your Python packaging pain", I invited you to not install the latest version of Python, and to use official channels for the installation process, but I didn't explain in detail how.

This post will fulfill that role and explain:

- *How to download, install and run Python on Windows. Mostly go to the right page on python.org, install, reboot.*
- *How to download, install and run Python on Mac. Same, except you need to run the script to install certificates manually at the end. Also, careful with Apple Silicon.*
- *How to download, install and run Python on Ubuntu. Setup deadsnake, then sudo apt install -y python3.X-venv python3.X-distutils python3.X-lib2to3 python3-setuptools python3.X-tk build-essential libssl-dev libffi-dev python3-dev.*

Those are not the only way to do it, but as explained in "Why not tell people to "simply" use pyenv, poetry or anaconda", it's probably your best bet.

The target audience

The formula of this blog is: if the article is published at the beginning of the week, it's not for experts.

Today I even assume you are either a newcomer, or that you agree with the two articles I linked above. So I will not repeat here why I recommend this procedure.

Also, I will state things that seem quite evident. In this particular case, I believe it's useful to state the obvious. First, there is not such thing as a self-evident concept, nor something that is universally obvious to everyone. Second, if you are new to this, while you may be able to figure it out easily, it's nice to hop on the fast track lane.

First, decide on the version of Python you need

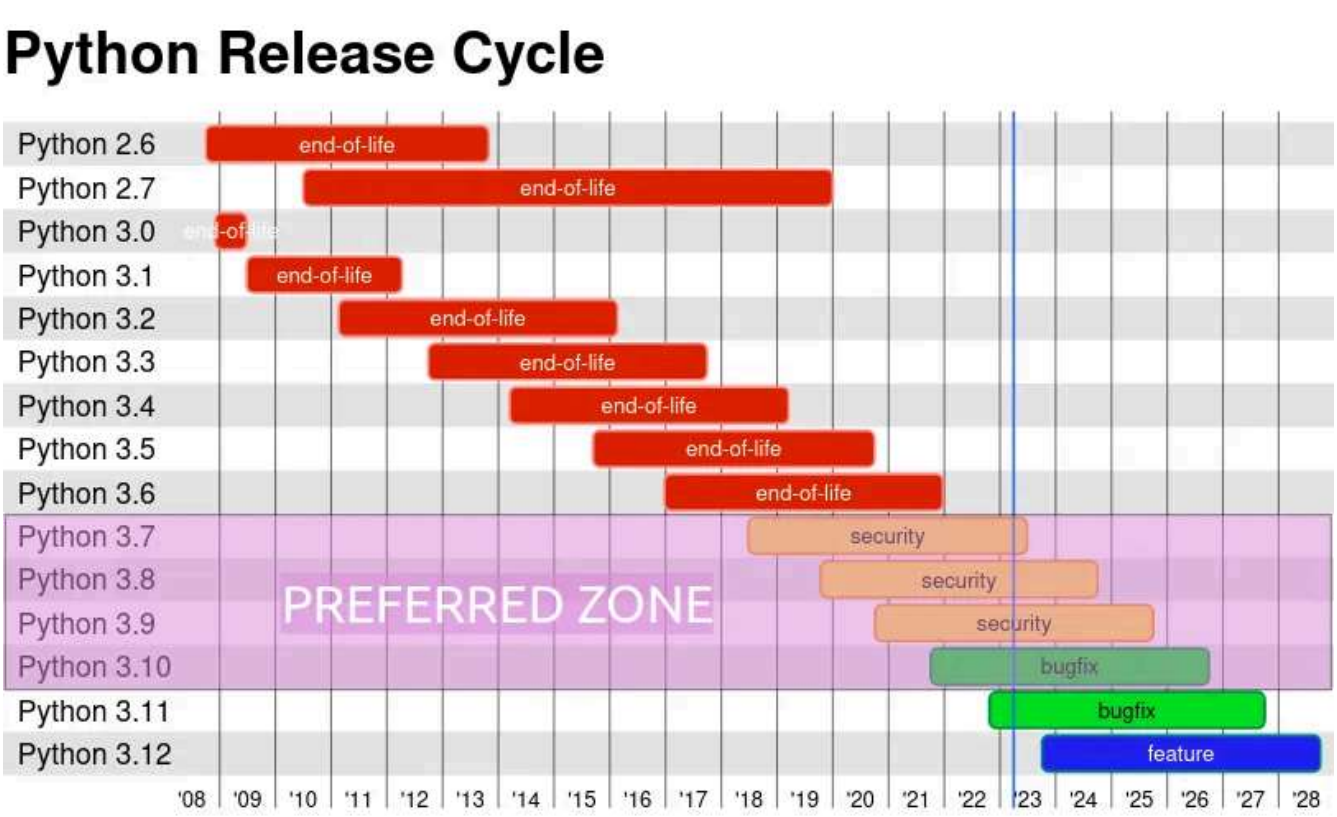
I'll repeat the advice from last week: your life will be much easier if you avoid the latest major version of Python.

For the sake of simplicity, **I will use 3.10 for the examples, but replace it with whatever is good for you** at the time you read this.

You can find this information in the "Status of Python versions". It's the second green one at the bottom of the chart, so for us in April 2023, avoid version 3.11.

It will be even better if you can avoid the versions that have reached "End of Life". So in our case, avoid 3.6 and bellows.

3.7, 3.8, 3.9 and 3.10 should be preferred, with 3.10 being the ideal candidate if you have the choice.



Installing and running Python on Windows

It's not about what to do, but mostly about what not to do.

Of course, as I explained before here:

> Nobody has time for Python
>
> **Relieving your Python packaging pain**
>
> Summary You can protect yourself against a lot of packaging pain by sticking to one recipe for installing and running Python: Don't install the latest major version of Python Use only the python.org installer on Windows and Mac, or official repositories on Linux...
>
> Read more

2 years ago · 2 likes · Nobody has time for Python

Among the things to avoid are using Anaconda, pyenv on WSL, winget, chocolatey, or the Windows Store.

The last one is trickier than it seems.

In the last few years, Microsoft has added a new behavior that seemed like a great idea: if you type "python" in the terminal and you have no "python" command registered, it will open the Windows Store and prompt you to install Python with the click of a button.

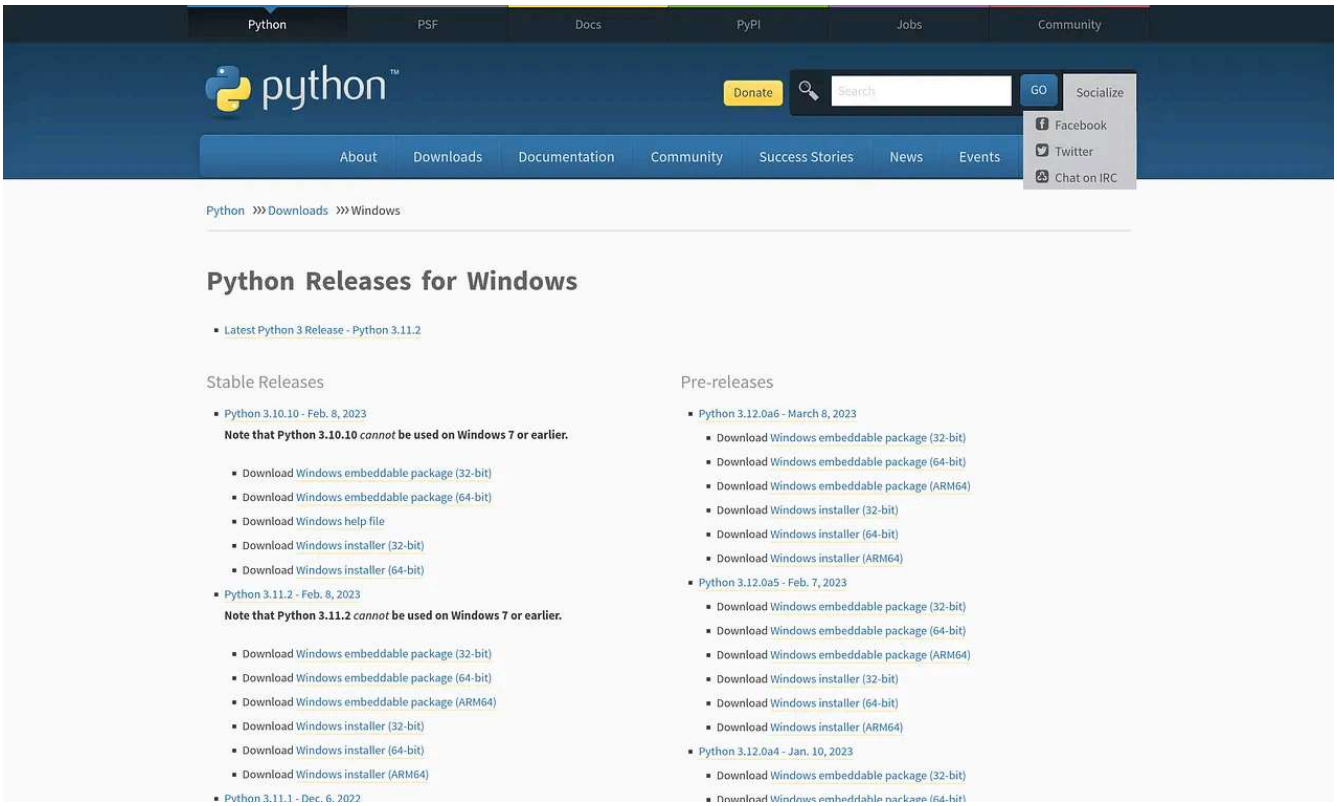
Simple. Tempting.

But I recommended in the other articles to not do it.

Instead, we will download the installer from python.org.

Because you should not download the latest major version of Python, you should skip the landing page, which prompts you to do exactly what you should not. Well, usually. For some reason, today it has a blank square where the button used to be.

What we want is to go straight to the "Python Releases for Windows" page, accessible from the "Downloads > Windows" menu:



Ignore the right column completely. Those are "Pre-releases", and are not stable.

What we want is to find the version of Python that we target, in this example, 3.10.

You will notice that the releases are not in the order of version, they are in the chronological order of release.

This means sometimes, a patch for an older version gets at the top of this list.

As I'm writing this, 3.10.10 is actually listed before 3.11.2.

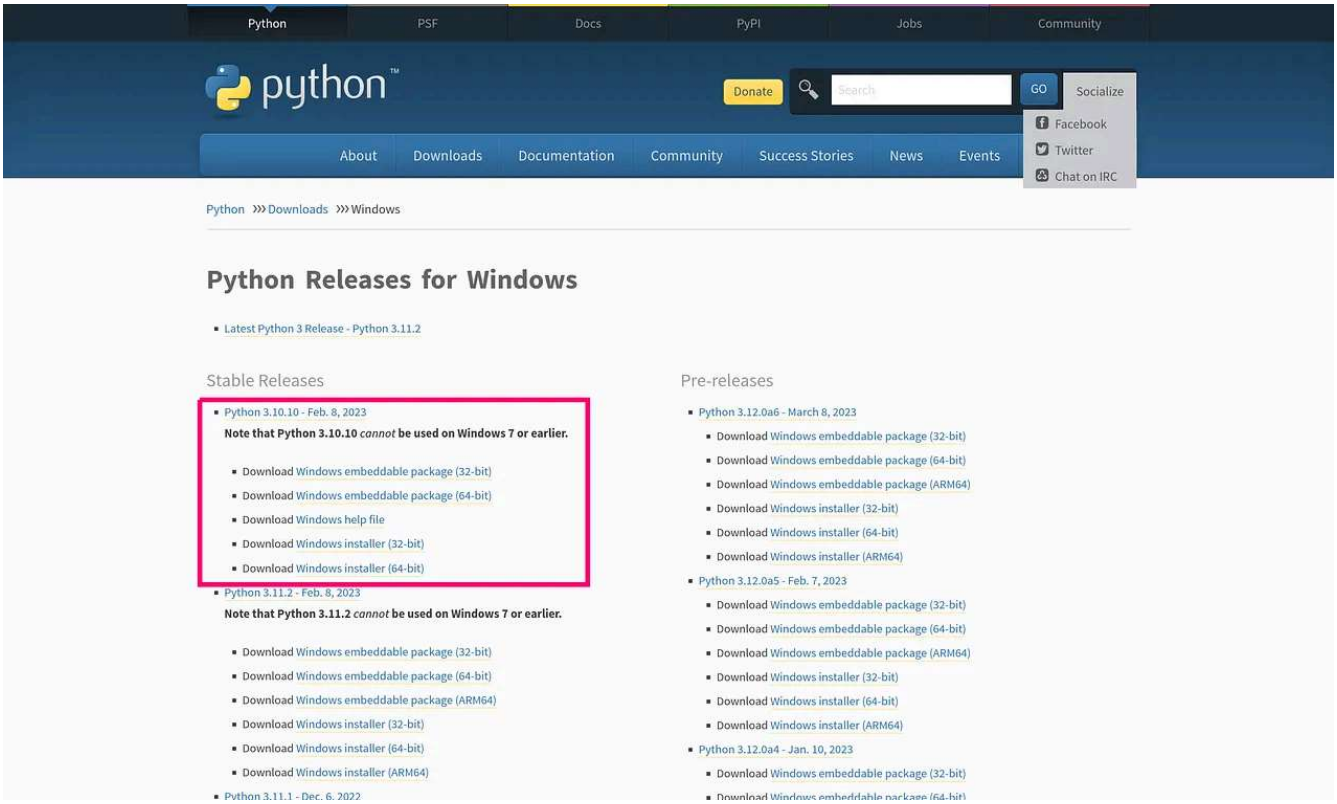
This means you may have to scroll until you find the one you want, but today, the first one is exactly the one we want.

If you have several 3.10 releases, how to choose the right one?

Target the one with the biggest bugfix number possible. This number is the one after the second dot.

E.G.: if you have the choice between 3.10.8, 3.10.9 and 3.10.10, pick the latter.

In our case, we are lucky, it's the first one:



Inside the section dedicated to the release should be listed several download links:

```
Download Windows embeddable package (32-bit)
Download Windows embeddable package (64-bit)
Download Windows help file
Download Windows installer (32-bit)
Download Windows installer (64-bit)
```

If you see "No files for this release" instead of download links, move to the next bugfix release. E.G.: if "3.10.10" doesn't have any download link, go to "3.10.9". Repeat until you find one with download links.

Now, download the "Windows installer (64-bit)"

Ignore the other ones. If you read an article on how to install Python, you are unlikely to need them.

For older release, the links may look like this:

```
Download Windows help file
Download Windows x86-64 embeddable zip file
Download Windows x86-64 executable installer
Download Windows x86-64 web-based installer
Download Windows x86 embeddable zip file
Download Windows x86 executable installer
Download Windows x86 web-based installer
```

In that case **download the "x86-64 executable installer"**.

Once the installer is downloaded, click on it to run it.

Don't run it as administrator.

Don't change any parameters in the installer. Use all the default values.

That's it.

Python is installed.

Now open a new terminal (if opening a terminal is new to you, I'll probably write a tutorial on this at some point).

Don't use one that is already open. In fact, if you use VSCode, Windows terminal, cmder (or any ConEmu based project) or some other fancy terminal container, kill them. Don't just close them, kill them. Honestly, just reboot.

And open a new terminal.

You should be able to use the "py launcher".

Remember, you should to be in a virtual environment to run Python. But on a few occasions, you will be outside of one, like when you create the environment in the first place.

This is where the "py launcher" is useful.

It's a command provided by the official Windows installer that will let you choose which Python to run (if you don't know how to run commands, check this article).

You can list the installed Python versions on your machine by running this command:

```
py --list-paths
```

Which may show you something like this:

```
-3.9        C:\Python39\python.exe
-3.8        C:\Python38\python.exe *
-3.7        C:\Python37\python.exe
```

You can then choose what Python to run doing:

```
py -3.X
```

E.G., to run Python 3.10:

```
py -3.10
>>> print('Hello')
Hello
```

You can even run a script by doing:

```
py -3.10 script_name.py
```

But again, remember that you will usually want to do this in a virtual environment, where the "py launcher" is not necessary.

The "py launcher" is for the rare occasions where you do things outside of a virtual environment.

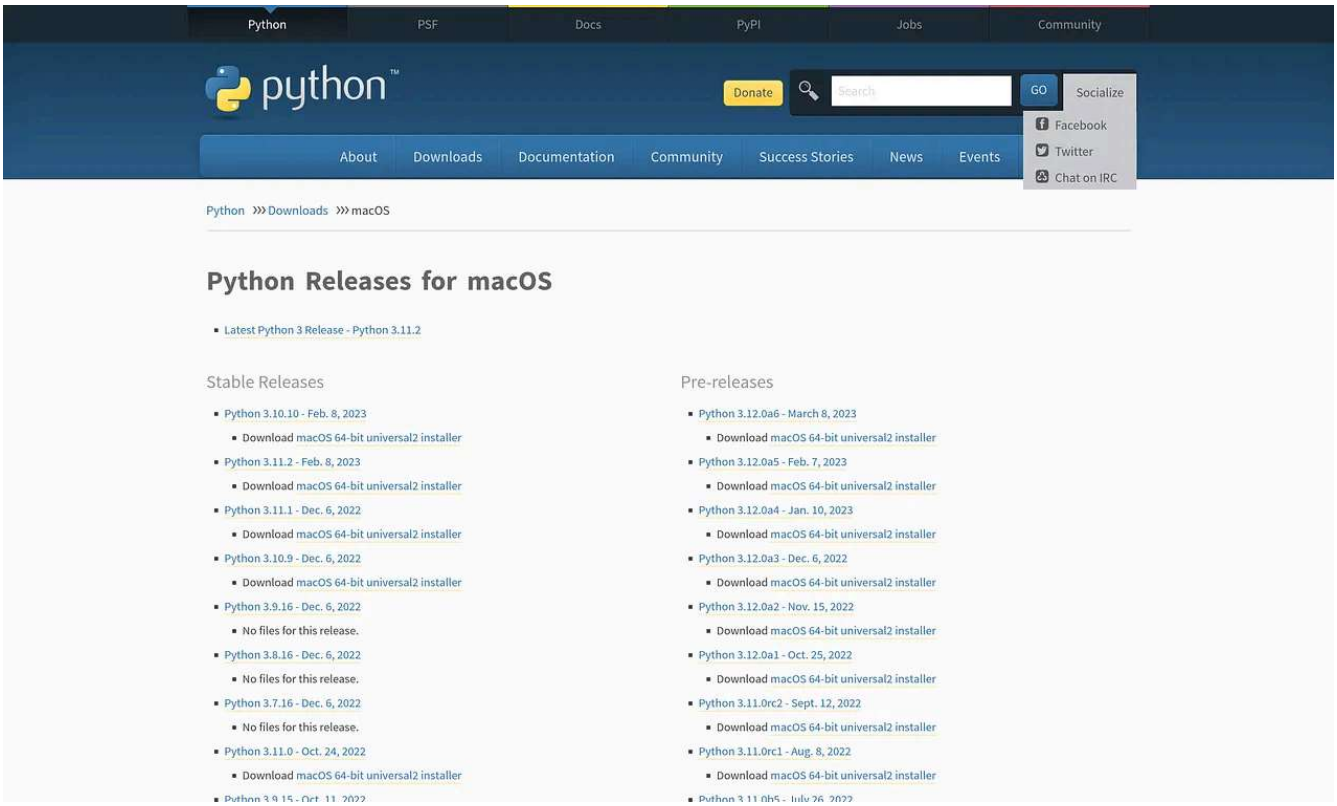
Finally, the "py launcher" is not installed by default on Mac and Linux. There is no need for it on those platforms: we have version suffixes.

Installing and running Python on a Mac

Macs come with Python preinstalled, but I strongly advise that you install one version from python.org and don't use the one that is provided with your OS. Indeed, like the Python on Homebrew, it's not designed to be used by you, but as a system dependency.

All the steps for finding and downloading the Windows section above are pretty much the same, so I will repeat a lot of it in case you jumped right to the Mac section.

As you should not download the latest major version of Python, you should skip the landing page, which prompts you to do exactly what you should not. Go straight to the "Python Releases for Windows" page, accessible from the "Downloads > macOS" menu:



Ignore the right column completely since those are "Pre-releases", and they are not stable.

Now we want is to find the version of Python that we target, in this example 3.10. The releases are not in the order of version, they are in the chronological order of release. This means sometimes, a patch for an older version gets at the top of this list.

As I'm writing this, 3.10.10 is actually listed before 3.11.2.

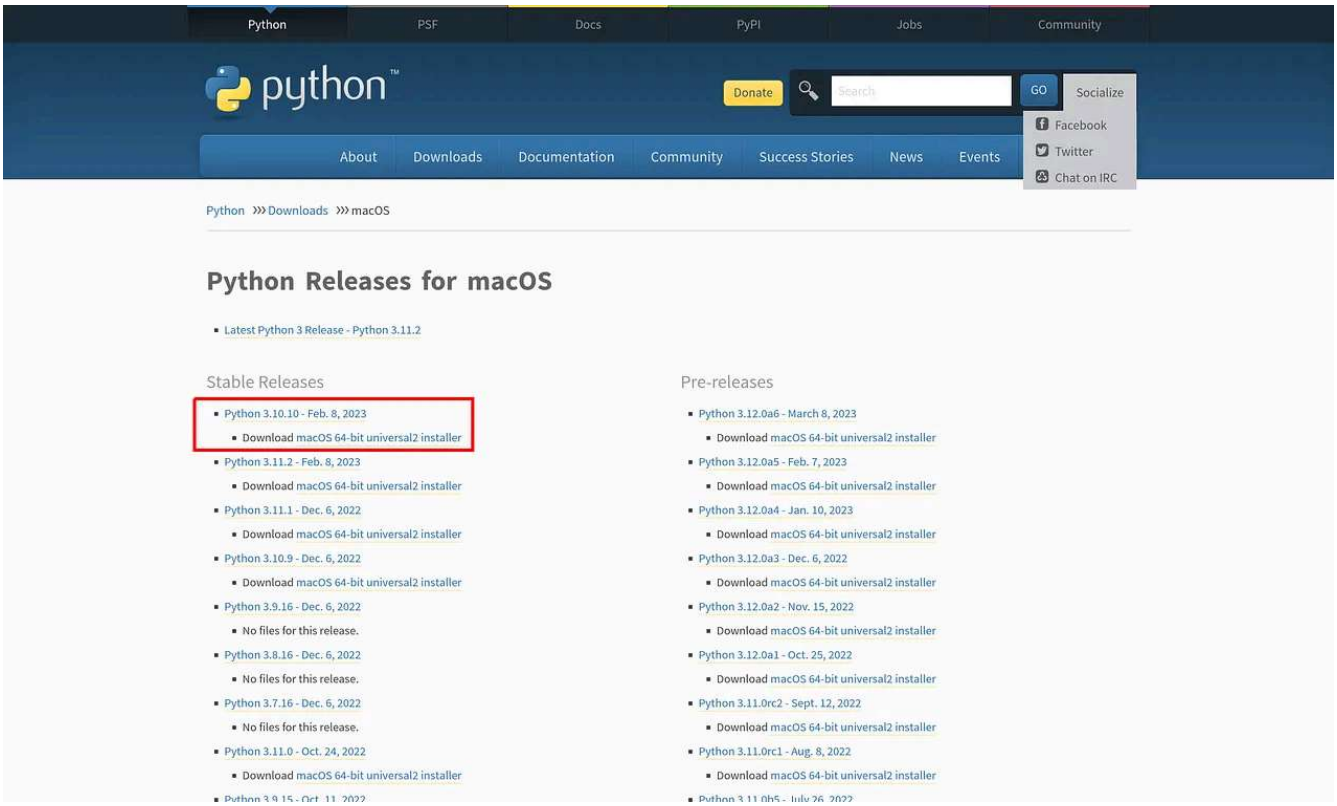
This means you may have to scroll until you find the one you want, but today, the first one is exactly the one we want.

If you have several 3.10 releases, how to choose the right one?

Target the one with the biggest bugfix number possible. This number is the one after the second dot.

E.G.: if you have the choice between 3.10.8, 3.10.9 and 3.10.10, pick the latter.

In our case, we are lucky, it's the first one:



Inside the section dedicated to the release should be listed one or several download links:

```
Download macOS 64-bit universal2 installer
```

or

```
Download macOS 64-bit Intel-only installer
Download macOS 64-bit universal2 installer
```

If "macOS 64-bit universal2 installer" is available, download that.

It's the one that works for all the most possible Mac configurations.

In the event you want to use a version which doesn't provide a "universal2 installer", you need to make sure you don't have Apple Silicon based Mac.

Only the “universal2 installer” works with Apple Silicon.

How do you know which one you have?

Click the Apple Icon in the top left corner and select "About this Mac". If you see the word "Processor", you have an Intel-based Mac. If you see "Chip", it's an Apple Silicon.

Check this tutorial for pictures on how to do this.

If you have an Apple Silicon chip, limit yourself to versions with “universal2 installers”.

Once the installer is downloaded, click on it to run it.

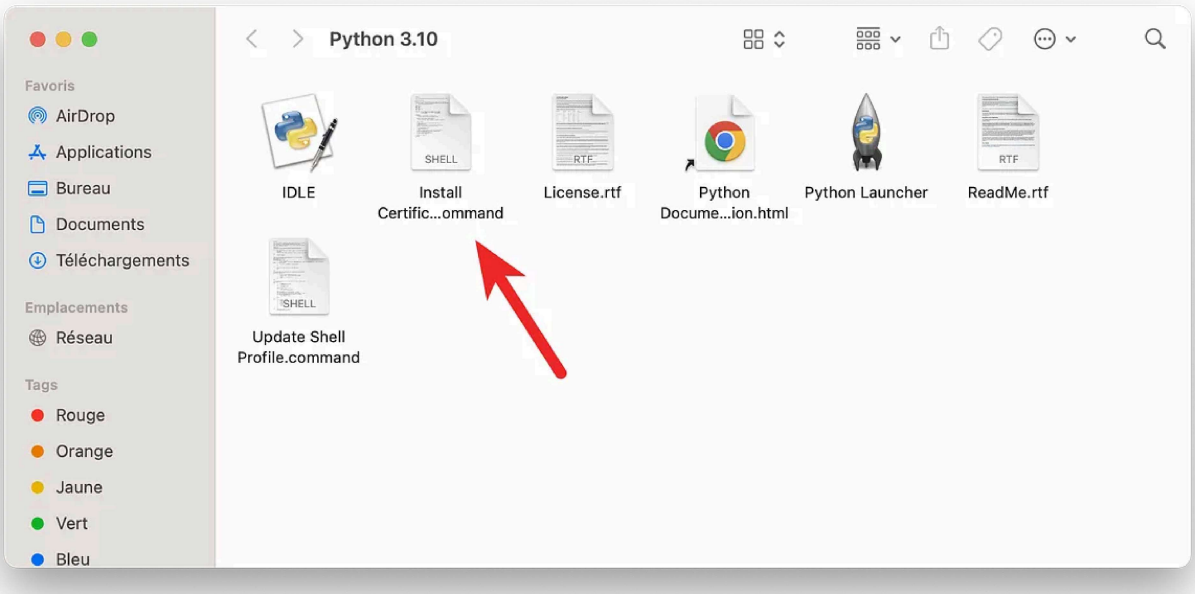
It may ask for your password, which you should provide.

Don't change any parameters in the installer. Use all the default values.

At the end of the installation, the Finder should open automatically to "/Applications/Python 3.10" (if not, go there).

You will find a "Install Certificates.command" file.

Click on it to run it:



Without this, you will get SSLCertVerificationError.

Now Python is ready.

Open a new terminal (if opening a terminal is new to you, I'll probably write a tutorial on this at some point), and type:

```
python3.X
```

to run the Python of the version of your choice.

E.G:

```
python3.10
```

If you want to know what pythons are installed, just type "python" then "tab", it will offer completion and show you all pythons currently available on your system.

Installing and running Python on Linux

The combinations of the number of distro flavors, number of releases and versions of Python make for a gigantic matrix that I don't have the resource to keep track of.

This is one of the reason pyenv is sometimes cited as a solution: it seems a tool that works the same on all Linux distributions. It's not the case, of course, since it requires you to install pyenv correctly, then install the right compilation dependencies, and, of course, make sure the PATH is set correctly, all of that being another matrix entirely.

The steps to install Python are very specific to your system, but know this: the more exotic your OS, the highest probability to have difficulties. Rolling distros are particularly gnarly, since they will regularly break your virtual environments because they update all the things all the time and venv use simlinks.

But if you chose this, I assume you know your way into the Linux world, and you don't need this article: you can autonomously solve the problem.

On Linux, the best is to restrict yourself to what is available in the official repositories.

It's limiting what you can do, but it's also limiting the problems you will have.

Even doing this though, it's not as straight forward as on Windows and Mac, because many distribution maintainers split Python into several packages. And there is no standard ways for the split, it's a bit of the wild west.

What we need is to make sure we have all the following (this list is not specific to Ubuntu):

- python itself
- venv so that you can install stuff safely
- tkinter since many tutorials assume it's there
- distutils, setuptools and lib2to3 for some package compatibility
- python headers, and headers for libffi, libssl if you ever encounter some package that doesn't have a wheel

The irony that this last point is part of what pyenv needs to work correctly is not lost on me.

Now for the fun thing, it will be different packages and commands depending of the distro flavor... but also sometimes the versions of the same distribution.

In the example, I will focus on Ubuntu (and hence Ubuntu-based) distros. It's the default image for WSL, very popular on VPS or the cloud, and a beginner favorite. Since I want to help the most people, it's quite a decent choice.

It's also my main OS, so I'm biased.

I will assume you didn't go bleeding edge and stuck to the Long Term Support releases, currently 20.04 and 22.04. Outside of this, you are choosing to least beaten path. I get it. I've done it. It's fun. But it's adding a lot of entropy to your setup, and it won't solve your python packaging problems.

Now for at least one good news: Ubuntu is special.

It's special because it has the deadsnake PPA, a wonderful community package maintained by the not less wonderful Anthony Sottile.

This means you will be able to safely install a larger spectrum of Python versions.

Open a terminal and type:

```
sudo add-apt-repository universe
sudo apt install software-properties-common -y
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
```

This will add the deadsnake PPA (you'll need to enter your password) to your config, and update the apt sources to let you install Python 3.7 to 3.11 (but really, we'll stick to 3.10 remember ?) on both Ubuntu LTS.

The PPA does more giving you more Python to play it, it normalizes a lot of things, and makes the whole thing work better.

Also note we activate the “universe” ubuntu official repositories first. Without this, “venv” will miss “ensurepip” and install without error, but not work when you run it with the python versions installed with the distro by default.

We will need to install all the right packages, so that eventually you can do this in a Python shell without error:

```
>>> import venv
>>> import distutils
>>> import setuptools
>>> import lib2to3
>>> import tkinter
```

Or the bash one-liner:

```
python3.X -c "import venv, distutils, setuptools, lib2to3, tkinter"
```

With 3.X being the desired Python major version.

Like:

```
python3.10 -c "import venv, distutils, setuptools, lib2to3, tkinter"
```

(you may get a deprecation warning about distutils because it's an old stuff everybody can't wait to see gone, but it's safe)

In Ubuntu, the list of packages to get there can be installed with:

```
sudo apt install -y python3.X-venv python3.X-distutils python3.X-
lib2to3 python3-setuptools python3.X-tk build-essential libssl-dev
libffi-dev python3-dev
```

E.G:

```
sudo apt install -y python3.10-venv python3.10-distutils python3.10-
lib2to3 python3-setuptools python3.10-tk build-essential libssl-dev
libffi-dev python3-dev
```

You may have noticed we don't install anything related to pip, so pip may not be available at the system level. It doesn't matter: you should never use it at the system level and always be in a virtual environment if you call it. And pip is always available in a venv.

Now Python is ready.

Open a new terminal (if opening a terminal is new to you, I'll probably write a tutorial on this at some point), and type:

```
python3.X
```

to run the Python of the version of your choice.

E.G:

```
python3.10
```

If you want to know what pythons are installed, just type "python" then "tab", it will offer completion and show you all Python currently available on your system.

That was a lot

You would think that "just" installing Python would be a short tutorial, but look at the length of this thing!

At the end of the week, I'll write about what a better installation experience could look like. Then next week we will get to the essential things you need to know about virtual environments.

And of course, if this helped you, you can help me by sharing this on a platform you like. Mastodon, Twitter, Reddit, HN... I'll even be glad to appear on your grandma's Facebook timeline.

Hi grandma.

Jim Feb 26 ♥ Liked by Bite Code!

I have a few questions. (beginner so maybe I am asking the wrong questions)

What's the point of setuptools and the other packages on the Ubuntu installation? Are these packages automatically included in the windows installation? I don't see them in my windows python install. Also, why python3-setuptools and not a version specific python3.11-setup tools?

When I look at the docs for something like setuptools they seem to recommend using pip to install it. Is it a bad idea to use pip to install these? Would it be advantageous to have it specific to a virtual environment?

I appreciate your guides. Especially the ones about setting up python virtual environments.

LIKE (1) REPLY SHARE

3 replies by Bite Code! and others

Alexis Jan 31 ♥ Liked by Bite Code!

I appreciate how clearly you've outlined your recommendations here. Could you say a bit more, or have you said more elsewhere, about this point you make: "Macs come with Python preinstalled, but I strongly advise that you install one version from python.org and don't use the one that is provided with your OS. Indeed, like the Python on Homebrew, it's not designed to be used by you, but as a system dependency."

What do you mean it's not designed to be used by the user? If it's not designed to be used by the user, then what processes are supposed to use it and what should the user expect from it?

In particular, I notice that if I do `pip3 list --user` I see a lot of packages. I don't remember if I installed those myself manually. Perhaps I did, and should not have done so according to your advice. But now I'm wondering if it's the case that other, normally functioning applications would be expected to install packages there.

LIKE (1) REPLY SHARE

1 reply by Bite Code!

14 more comments...