Google

Reading: Python’s new versions and features

You've learned that one of Python's great strengths is its dynamism: it is developed and supported by a large community of users. This makes it a language that changes and evolves . In other words, the language and its libraries are constantly growing, improving, and responding to user feedback.

Python's dynamism, while a strength, can introduce complexities. This can lead to challenges when code is intended to be used for long periods of time or when multiple people work on a project together. A script or program developed in one Python version is often tied to that particular environment, and running it in a different version can lead to unexpected behavior. This is because Python and its libraries are updated over time. So, if you developed a project that used, say, Python 2.7.4 in 2013, and you try to run it now in Python 3.12.4, there's a possibility that it won't execute the same way. Similarly, collaborative projects can be hindered by team members using different library versions, resulting in compatibility issues.

Deprecation

Code inevitably evolves, gets phased out, and becomes obsolete. This process is known as deprecation. If you're working in a Jupyter notebook, you'll generally get a warning if something you use is scheduled to be deprecated in a future release. Your code will still run, but it might not continue to do so in future versions, or if it does, its behavior might change. It's best to pay attention to these warnings and change your code to conform with the coming change, because it's easier to do it when you're actively writing the code than to come back to it when it breaks and try to figure out what you were doing at that particular stage of the project. Also, small updates—for example, Python 3.11.8 to 3.11.9—won't be as disruptive as large updates—say, Python 2.X to 3.X, so you'll notice more deprecation when there are major releases.

It's important to note that you can still use older versions of Python and its libraries. You don't always have to update your code every time there is a new release. Indeed, older versions typically continue to receive support long after they've been superseded by more recent releases. This is to allow for people and organizations to plan their code updates over time. While you might be able to simply update your project, it's a lot harder for a large company to update their code base while ensuring that nothing breaks.

Environments

The particular set of conditions and configurations that you use to develop code is known as your environment. Your environment includes the tools, libraries, dependencies, settings, etc. that are used to write and run your code. So, when you work on a project with a group of people, it's important that you all share the same environment so the code runs the same way for everybody. If you're doing this course's labs on Qwiklabs, you generally don't have to worry about the environment because it has

been preconfigured for you. But if you're doing the labs off the Qwiklabs platform, be aware that your code might execute differently than it would on Qwiklabs, depending on your environment.

Environment management can be a complex and technical process, and we won't go into it much in this course because your environment has been preconfigured to allow you to focus on learning other skills. If you're interested in learning more about environment configuration or creating and working in your own development environment, Anaconda Navigator is a great place to get started. Anaconda Navigator is a free, open-source, downloadable graphical user interface that provides a user-friendly tool set to manage environments. It also has a suite of other built-in feature integrations like Jupyter notebooks, PyCharm, VSCode, and other popular tools used by data professionals.

Check which versions you have

We try to keep the code in this course current by using recent stable releases of Python and relevant libraries, but updating everything in all the courses for every new release isn't feasible or necessary. If you're doing the Jupyter notebook activities on your own computer and not on Qwiklabs, you may occasionally find differences in how your code runs compared to how it is demonstrated on Qwiklabs. These differences are likely due to versioning.

To check which version of Python you're running in a Jupyter notebook, execute the following code in a notebook cell:

```
import sys
print(sys.version)
```

To check which version of a library you're using, import the library and then use its **__version__** attribute to access its version information. For example:

```
import numpy as np
import pandas as pd
print(np.__version__)
print(pd.__version__)
```

Key takeaways

Python evolves and grows in response to its users and use cases. This means that code updates can change the behavior of code that was written in older versions. Understanding how to assess, manage, and troubleshoot your coding environment is an important part of coding.
