Choosing a Python library

The essential checklist

You're working on a Python project, and you realize the next thing you need to do is a bit tricky. You don't want to reinvent the wheel if you don't have to. You wonder: *has someone solved this problem before*?

Start with the PSL

The first place to look is the PSL: The Python Standard Library.

One of Python's great strengths is that it *comes with batteries included.* Python includes well-documented, tried and tested libraries which do all sorts of useful things.

Then try GitHub

No luck? Turn to GitHub for help. It usually can! Many of the libraries I use are hosted on GitHub.

Sometimes you'll find one candidate library; sometimes there will be more than one. You'll need to decide if any are fit for purpose, and which looks best.

As we'll see, GitHub can tell you a lot about the quality of the project.

Here's my checklist

Here are the things I like to ask about a library I'm considering. I've illustrated the checklist using the `guizero` project as an example, since I use it a lot and it ticks all the boxes.

Does it have good documentation?

Is the intended use clear, and does it match your requirements? I've sometimes been caught by a library that looks as if it does the job but actually does something different.

And if it says it's doing what you want, do the docs show you how to *install* and *use* the library?

Is the project active and well-supported?

GitHub is your friend here. It's easy to check how "alive" the project is, right from the project's home page. Look at the home page to find out

- When was the last update?

- How many issues are there?

  - How many unresolved issues are there?
  - How long have they been around?
  - Are pull requests dealt with quickly?
- How popular is the project?
  - How many people are watching it?
  - Is it often starred?
  - Has it often been forked?
    - *Forking* is a process for creating a personal copy which you can use for modification, or to suggest fixes or improvements via pull requests.

What is the quality of the code?

Does it support Python 3? Most libraries do these days, but if it doesn't, that is a showstopper.

- Is the code readable? I look for good naming of functions, classes, methods, and variables, and I like to see type hints whenever they help.
- Is the code sufficiently commented? *Is it well-structured: simple and short?
- Is it sufficiently performant? This may not matter, but if it does, it might matter *a lot*.

Does it have appropriate licensing?

In my case, I look for a licence that's compatible with the MIT licence but your choice will depend on your intended use and your context.

If you're a solo developer, you will be able to decide for yourself.

In some situations, you may need to check Corporate Policy, and you may even need to talk to your employer's legal department.

Does it have a large, supportive community?

If the library has a Slack or Discord channel, you may be able to dip in and quickly get the feel of the community. Is it friendly, respectful and helpful?

Is the library easy to learn?

Ideally, you should be able to see how to get going straight from the README. (There is a README, isn't there?) Are there links to Tutorials, Books or Courses?

Is it written by authors you know and trust?

That's not essential, but it's very reassuring.

Does it have a sensible, consistent API?

A good API satisfies the *principle of least surprise*: if you guess how to use it, you'll probably be right.

Does the author specifically encourage pull requests to add features or fix bugs?

Check the language of the documentation (especially the README) to see if the author wants to hear if you find a problem. It often indicates their mindset when writing it - do they intend for it to be used in a collaborative fashion or is it a "one man show"?

Are there automated tests?

I can write elementary test-free code, but as soon as things get complicated, I know I need tests to keep me on track.

If I am using other people's code, plenty of automated tests reassure me that the code is likely to work. They also indicate that I can refactor the code or extend it safely if I need to.

Summary

A few minutes' search on GitHub can tell you a lot about whether you should use a third party Python Library.

I hope you find this checklist helpful, and I welcome constructive feedback.

Thanks to Ben Nuttall, Michael Horne and BrianLinuxing for their helpful suggestions.