FINCAD®

Risk.net

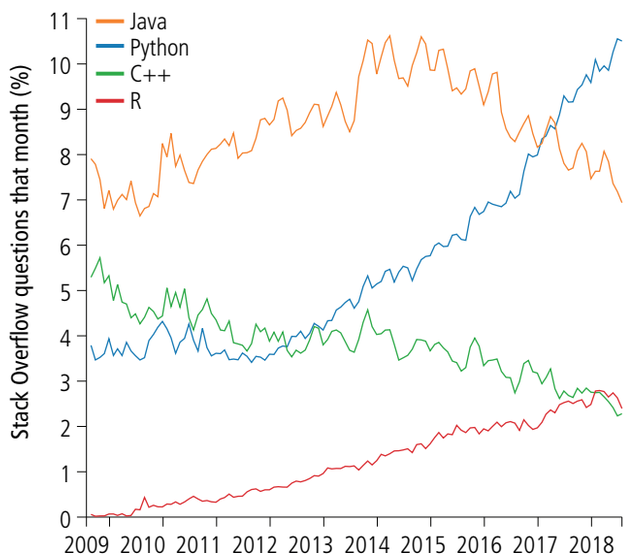
Python – Is the buzz justified?

Python is rapidly becoming the world's most popular programming language and its versatility and ease of use has enabled it to achieve widespread adoption in finance, becoming the multipurpose tool of choice for quantitative analysts and other financial technologists. By Christian Kahl, director and head of client services at Fincad

Data science has become pervasive throughout the financial sector, and Python is almost always a part of the equation. One might argue that the symbiosis of data science and Python has led to significant growth of data analysis in finance, ranging from fraud detection, and market and risk analysis, to investment decision-making and smart beta. However, as with any technological trend, it is important to understand the limitations and risks before jumping on the bandwagon.

So, why is there so much buzz around Python? Is it really justified? And, more importantly, what are the risks to consider when working with this popular programming language?

1 Python versus other programming languages



Source: Stack Overflow

Why has Python become so popular?

Versatile and easy to use

Python's ease of use is no accident. Frustrated by the shortcomings of other programming languages, in late 1989 Guido van Rossum set out to create one that would be easy to read and have maximum flexibility. Python's syntax is so easy to learn that even those who've never coded can follow the logic. This allows users to write code faster and with fewer errors.

Extensive ecosystem with powerful libraries

Python has an extensive selection of libraries, which can save time and shorten the development cycle. Mathematics and statistics libraries such as NumPy and SciPy are very well suited to financial analytics, and when users add tools such as Jupyter notebooks for interactive development, Pandas for managing dataframes and Plotly for user interface (UI) and visualisation, Python becomes a formidable data science and analytics tool. In particular, Jupyter is now becoming a highly productive environment for collaboration and sharing ideas across teams on a web-based platform.

This ecosystem is a significant factor in the huge productivity gains organisations see with Python. Standard libraries and tools enable quants to focus on creating a competitive advantage, rather than spending resources reinventing basic functionality.

Enhances collaboration, efficiency and productivity

The large pool of resources available to those using Python makes getting up and running easy, meaning many can build their own custom analytics and bespoke reports without needing to go through an internal development team or wait for a software vendor's next release. This speed in customising functionality improves agility within the business, and it can also be used to quickly prototype new workflows and reports without the need for costly development projects.

The ease of use and setup means a range of roles within organisations are using Python rather than other programming languages, so it is not only traditional developers that have a say in the development process. With Python, quants, traders and portfolio managers can get involved. This leads to increased collaboration and allows for rapid development timeframes, saving time and cost.

Risks with Python

While there are many good reasons to use Python, it has limitations. Even some of the benefits can lead to risks if not managed carefully.

Easy to get going, but hard to scale

Because Python is so easy to use, individuals will often create applications without first having a proper plan in place for managing key areas. However, without the right plans, technologies and frameworks in place, there is a risk of a Python project collapsing under its own weight. For example, when using Python in a large organisation it can be difficult to maintain control over the code, the different versions of data and models, and who has access to the applications.

Caveat emptor

It is also worth bearing in mind that the libraries available with Python are open source, and users must be careful about which ones they use. Unlike commercial software – which can be expensive – there is no central management of the

2 Example of a Python trade script for defining a targeted accrual redemption note

```python
from f3sdk.services.service_decorators import f3service
from f3sdk.f3.payoff import Max, Min, If
import asyncio

@f3service()
async def fx_tarn(f3,
                  base_fx_index: str,
                  interest_rate_cap_index: str,
                  base_fx_index_initial: float,
                  interest_rate_cap_index_initial: float,
                  cap_rate: float,
                  base_currency: str,
                  notional: str,
                  start_date: str,
                  maturity: str,
                  market_convention: str,
                  pay_rec: str) -> str:

    flow = lambda payoff, roll : f3.CreateSingleCashflowProduct( RollDates = roll,
                                                                 Index = payoff,
                                                                 Notional = notional,
                                                                 Currency = base_currency,
                                                                 PayRec = pay_rec )

    rolls = await f3.evaluate( f3.RollSchedule( StartDate = start_date,
                                                Maturity = maturity,
                                                MarketConvention = market_convention ) )

    total_payoff = 0
    tarn_flows = []
    redemption_flows = []
    for roll in rolls:
        # define payoff in loop over the schedule
        # operators overloaded and functions defined for indices
        this_payoff = roll[1] * 0.01 * Max( [0.0, Min( [base_fx_index(roll) - base_fx_index_initial,
                                                        interest_rate_cap_index(roll) - interest_rate_cap_index_initial] )] )
        prev_cond = total_payoff < cap_rate
        total_payoff += this_payoff
        this_cond = total_payoff < cap_rate
        tarn_flows.append( flow( (total_payoff < cap_rate) * this_payoff, roll ) )
        redemption_flows.append( flow( If( this_cond, 0, prev_cond ) * ( 1 + this_payoff ), roll ) )

    fx_tarn = await f3.evaluate( f3.CreatePortfolioProduct( tarn_flows + redemption_flows ) )
    return fx_tarn
```

Using Python within an established framework can eliminate the need for quants to learn a new proprietary language for trade scripting. If using a centralised risk and valuation system that allows for this functionality, these new trade structures can be quickly distributed to the entire organisation.

code or support, so it is important users choose libraries backed by a robust community of users. In the realm of data science, NumPy and SciPy are examples of reliable libraries and there are many others. Such libraries are vetted, highly regarded and widely used. Thus, there are more eyes on the code and users have the ability to make fixes as needed.

Upgrades

Python is commonly used for extensibility and it can be standard to have lots of different code embedded within workflows for this purpose. Unfortunately, a lack of control when implementating Python code can lead to different libraries and functions being used. This can present problems when it comes to upgrades, leading to breaks in code and subsequent systems failing. For example, problems can occur when using Python 2 versus Python 3 in different parts of an organisation, and then upgrading part of its technology stack when it is incompatible with older versions.

When upgrades are not managed properly, it can lead to downtime and additional costs. However, this risk can be mitigated by having strong processes and controls around the release of new code and how upgrades are managed.

Speed and robustness

There is a common view that Python is not as fast in terms of runtime speed as compiled languages, such as C++. Typically, Python would not be used for the same tasks as C++; it is best used for integrations, extending frameworks or running on-the-fly reporting and calculations where rapid speed is not of primary importance. Equally, the inherent flexibility and openness of Python can be viewed as less robust.

The key here is that Python is easily used in conjunction with other languages, so in cases where users need to use a more structured language, they can still use Python to extend their development efforts. This can be done effectively if users have a robust technology stack and framework, which will enable them to retain the benefits of using C++ for critical code, while scripting around it using the flexibility of Python.

Best practices for using Python

One of the dangers of implementing new technology is not fully understanding where potential risks lie, and adopting Python within an internal framework is no different. It is essential to think holistically about where Python is used and how it is deployed to avoid operational risks. Here are a few key suggestions to mitigate these risks:

- Implement controls for code releases and ensure any code written is using vetted libraries and being used consistently throughout the organisation.
- Manage upgrades carefully to ensure workflows don't break down when released.
- Use Python to extend and enhance larger systems rather than building systems from scratch. This can diminish the margin for error and reduce overall operational risk.
- Use a centralised system with a Python interface that allows those that want to use Python for customisation and analysis to do so, but that can also allow controls to be put in place quickly and easily.
- To maximise collaboration, ensure any development efforts – such as payouts, reporting techniques and scripts – are centralised and available throughout the organisation instead of on staff members' machines.

Conclusion

Python is quick and easy to pick up and therefore has huge benefits in collaboration and time saving. From this perspective, it is very cost-effective. Python allows companies to be more dynamic and agile, eliminating the need to wait for long periods as developers build a new UI or vendors build out workflow. There is, however, a trade-off between the robustness of internal systems and the flexibility of using Python. The open and free libraries have obvious benefits, but can present risks if the versions are not controlled and upgrades are not handled properly.

Using Python can assist developers and quant traders in easily building out custom applications, reports and analysis that drive better investment and risk decisions. It is therefore worth investing in systems that enable Python to be used for extensibility and customisation, and provide centralised modelling, version controls for managing data and instil consistency across the organisation. This approach will free quants and developers from getting bogged down in merely maintaining infrastructure, and give them more time to focus on adding value to their business.

The author

Christian Kahl, Director and head of client services, Fincad

Christian Kahl is responsible for all client-facing quantitative topics and resources globally. He is recognised for his in-depth knowledge of stochastic volatility modelling and high-performance computing, and has more than 10 years' experience implementing models in cross-asset front-office pricing libraries for both sell-side and buy-side institutions. Before joining Fincad, Christian was the deputy head of financial engineering in the equity market and commodity department at Commerzbank.

E: c.kahl@fincad.com