MILLIMAN BRIEFING NOTE

The Rise of Python

Powerful, robust calculation software

Donal McGinley, FSAI

Milliman

Python has become the fastest-growing major programming language in the world[1]. It has surged in popularity over the last decade, following the development of its industry-leading capabilities in data science, machine learning and artificial intelligence, and it is now one of the most popular and widely used languages in the world[2]. In this briefing note I discuss some of the reasons for its popularity.

Many actuaries and insurers have started using Python for many reasons including:

- It has a vast array of powerful functionality which is available for **free** to anyone who wants to use it.
- It has very strong **data science capabilities**, as it is excellent at manipulating data, such as importing, exploring, checking, modelling, visualising, and summarising data.
- It has strong **modelling capabilities**, and can be used to build actuarial models such as discounted cashflow models and pricing models, or to build less traditional models such as machine learning or artificial intelligence models.
- It is good for **model industrialisation**, and can be used to build full end-to-end processes, such as regular valuation processes and ORSAs.
- It is a general purpose programming language which was designed to be robust enough to be used in production. It is a high-level language (similar to VBA) which was designed to be easier to use and easier to learn than other programming languages.

This briefing note gives an overview of Python and explains why it has grown so rapidly in recent years, with a particular focus on the capabilities relevant for actuaries and insurers.

In future briefing notes, we will discuss other new modelling tools available, including case studies which demonstrate the modelling improvements that can be made.

Follow us on LinkedIn to get notifications of our latest briefing notes and blog posts.

Use Cases

Actuaries are increasingly using Python for many different types of work, including:

- Discounted cashflow modelling (IFRS and Solvency II)
- End-to-end valuation processes
- ORSAs
- SCR aggregation and risk margin calculation
- Reinsurance calculations
- Policyholder asset lookthrough
- Lapse and mortality investigations
- Analysing past policyholder performance
- Data checks
- Output checks
- Machine learning

One of the main benefits of using Python is that it is powerful and robust enough to carry out all of these tasks. In the past we have relied on many different software tools such as Excel, VBA, Access, SQL, and other proprietary software. Now it is possible to perform these types of calculations in one single tool (Python). Python is a powerful, robust tool for performing everyday actuarial work. Also, it contains exciting functionality which isn't typically found in conventional actuarial software, such as machine learning capabilities. Going forward, I expect that actuaries will replace models and processes built in Excel or SQL with Python models and processes, as Python[3] is a powerful, robust tool with strong calculation and data processing capabilities.

History of Python

Python was created in 1990 by Guido Van Rossum, a Dutch programmer, who named it after his favourite comedy troupe, Monty Python's Flying Circus. He had observed that computers were becoming faster and cheaper over time, whereas programmers were generally becoming more expensive due to salary inflation[4]. He expected these trends to continue over time. He set out to create a language which would maximise programmer productivity, by creating a high-level "batteries included" language, which would be easy to learn and easy to use, with ample pre-existing functionality, enabling developers to write code quickly and efficiently. He

[1] https://insights.stackoverflow.com/survey/2019

[2] See Appendix A for some measures of its popularity

[3] Note that, in this document, when I refer to Python, I typically mean the core Python libraries and its widely used data science libraries

[4] https://www2.computerworld.com.au/article/255835/a-z_programming_languages_python/

recognised that this might lead to some undesirable behaviour, such as slower runtimes, but reasoned that it is better to maximise your most scarce resource (employee hours) rather than optimising items like run speeds.

Python had a small user base in the 1990s. It was very popular in certain niches, but was not a mainstream language. Initially, it had limited functionality. However, over time, a virtuous circle developed. Python developers released powerful libraries which anyone could use. Newcomers started learning Python, so that they could avail of these new libraries. This increased the number of people who knew Python. Some of these newcomers started building their own libraries or improving existing libraries, which in turn led to better libraries, which in turn prompted more newcomers to learn Python. Python now has a very large, active community of developers, who can quickly follow any market trends, and who can create their own market trends by building market-leading software.

One library which kick-started the virtuous circle was the release of the Django website-building framework in the mid-2000s, which convinced a lot of web developers to start using Python. Python quickly became one of the go-to languages for building websites.

Python became one of Google's official languages in 2006, which increased its popularity, and led to more rapid development of the Python ecosystem. It is currently used in many large organisations, including Google, Facebook, Amazon, Instagram, Spotify, Netflix, NASA and CERN[5].

Many universities started to use Python in their "Introduction to Computer Programming" courses[6], as it is easier to learn than other programming languages, enabling lecturers to focus on high-level programming concepts, rather than getting bogged down in low-level computer science topics like memory management. This has increased the size of the Python community, and increased the supply of Python developers.

Python was upgraded gradually over time. When new versions were released, users could choose to upgrade or to stay on their existing version. Most new releases were backward compatible so old models would still run in newer versions of Python. However, in 2008, Python 3 was released. This was a major change, and was not completely backward-compatible. Some Python 2 code would not run in Python 3, and existing code bases had to be converted from Python 2 to Python 3. Some users were unhappy with this change, particularly companies with large code bases. This caused a split in the Python community, with some companies continuing to use Python 2 and some moving to Python 3. This episode caused some rancour within the community and slowed the development of Python. However this isn't an issue for new users, who don't have existing Python 2 code bases. New users can just start with the latest version of Python 3.

Python became an industry leader in data science following the development of its data science libraries in the late 2000s and early 2010s[7]. These libraries have powerful data processing capabilities, including the ability to import data from SQL/Excel/CSV/big data systems, data exploration and visualisation, data cleaning and reformatting, modelling, advanced scientific calculations, machine learning and creation of end-to-end processes. Python developers took inspiration from existing data science languages such as R and Matlab, ensuring that the Python libraries would have many of the desirable capabilities found in those languages. The development of these libraries, coupled with the worldwide surge in interest in data science, catapulted Python from being a mid-ranking programming language into one of the top-10 most used languages in the world.

The final development which spurred the growth of Python was the release of artificial intelligence libraries in the mid-2010s. Google released its powerful Tensorflow[8] artificial intelligence library in 2015, and Facebook released its Pytorch[9] artificial intelligence library in 2016. Python users can use Python's data science libraries to prepare the data, and can use Tensorflow or Pytorch to build powerful AI models based on that data. Python has become the defacto industry leader in artificial intelligence modelling. The release of these libraries, coupled with the worldwide surge in interest in artificial intelligence modelling, has led to Python becoming one of the top-3 most used computer programming languages in the world.

Python is now the fastest growing major programming language in the world. See the Appendix for more detail.

The Python Ecosystem

One of the main reasons why Python has become popular is its large ecosystem of modular libraries, each of which contains specialised functionality. Python has a decentralised, modular structure, and a permissive licence. This enables users to build their own libraries, without requiring them to get permission from the core Python

---

5 https://en.wikipedia.org/wiki/Python_%28programming_language%29#Uses

6 https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext

7 https://qz.com/1126615/the-story-of-the-most-important-tool-in-data-science/

8 https://www.tensorflow.org/

9 https://pytorch.org/

developers. As a result, many libraries have been actively developed and improved over time, in many different areas such as data science and web development. Python's abilities can improve rapidly because teams of developers are free to work on whatever projects they wish.

The core Python libraries are the "glue" which hold the whole Python ecosystem of libraries together. The core Python libraries are maintained by the core Python developers and are quite stable, well tested and robust. Core Python is a general purpose programming language, which can do general programming tasks, and wasn't designed to specialise in any particular area.

Specialised functionality can be found in the thousands of non-core libraries in the Python ecosystem. These libraries have been developed by other teams of developers, and they interlink seamlessly with core Python. For example, the following libraries are often used by actuaries and data scientists:

- Numpy (array based calculations)
- Pandas (2-d data tables and SQL-style calculations)
- SciPy (scientific calculations such as solvers)
- Numba (high-speed calculations)
- Matplotlib / Seaborn (static graphs)
- Bokeh / Dash (interactive graphs)
- Scikit-Learn (machine learning)
- Tensorflow / Pytorch (AI / neural networks)
- Pyspark / Dask / SQLite (big data storage)

All of these libraries are maintained by different teams of developers, therefore they can be upgraded and improved simultaneously. This modular system, and large community of developers, has enabled Python to improve rapidly in many different areas.

For example, before 2015, Python was not a leading language for artificial intelligence (deep neural network) modelling. Other languages had better capabilities. In 2015, Google developers released the Tensorflow neural network library, which they had developed internally, and in 2016, Facebook developers released the Pytorch neural network library. These libraries allowed users to build, train and test powerful artificial intelligence models. These libraries were very powerful and relatively easy to use. Python immediately became the leading language for artificial intelligence modelling after these libraries were released to the public. This example highlights how quickly Python's capabilities can improve. In this case the improvements were driven by two tech giants who used Python internally. In other cases the improvements are driven by other less-famous members of the Python community.

What does Python Code Look Like?

Python is a high-level language, similar to VBA. It was designed to be easy to read and easy to understand. The following screenshot shows some basic Python code, including a data structure (a list), a "for" loop, an "if" statement and a print statement. Note that the code has fewer brackets and semicolons than other programming languages, and is structured using indentation (tabs), which results in less clutter and arguably makes it easier to read than other languages.

```
my_list = [1, -10, 3, -5]

for i in my_list:
    if i < 0:
        print(f"The number {i} is negative")
    else:
        print(f"The number {i} is positive")
```

Last executed 2019-11-07 11:58:43 in 7ms

```
The number 1 is positive
The number -10 is negative
The number 3 is positive
The number -5 is negative
```

Python has many external libraries, which contain specialised functionality. For example, actuaries often use the Pandas data science library. The following code loads the library, imports data from a CSV file and prints out the first five rows of the datafile. Note how easy it is to import and use this powerful library.

```
import pandas
my_table = pandas.read_csv("Titanic Dataset.csv")
my_table.head()
```

Last executed 2019-11-10 11:05:03 in 854ms

	PassengerId	Survived	Pclass	Name	Sex	Age
0	1	0	3	Braund, Mr. Owen Harris	male	22.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
4	5	0	3	Allen, Mr. William Henry	male	35.0

Python vs Other Languages

Actuaries use a wide range of software, including traditional software like Excel, VBA and SQL; data science languages like R; and proprietary software such as pricing and discounted cashflow modelling software.

Python is a relative newcomer in the field of actuarial modelling. It has taken inspiration from a lot of existing software, and has implemented a lot of the desirable features from existing languages. Most tasks which can be done in the languages above can also be done in Python.

EXCEL

Anything that can be done in Excel (and VBA) can also be done in Python. Both are general-purpose calculation software, and can be used to build many different types of models. While they can be used for actuarial calculations, they weren't specifically designed for this and don't contain any specialised actuarial capabilities. In my opinion, the main difference between them in terms of design philosophy is that Excel has been designed for maximum flexibility, whereas Python has been designed to be robust in production. Excel is a very good tool for certain tasks. It is easy to use and easy to learn, and is useful for checking results and sharing results. However it is slow, cannot handle large datasets and is quite manual. It lacks many features which would be desirable in production software, such as version control software. In theory, Excel could be used to perform almost any actuarial task, but in practice the inherent weaknesses in Excel make it inappropriate or unusable for many actuarial tasks and as a result many actuaries rely on other pieces of software in addition to Excel.

Python is more difficult to learn, but is much faster, can handle much larger datasets, and is more automated and robust than Excel. Crucially, I think that models built in Python are capable of achieving all of the best practices for actuarial modelling[10]. This makes Python suitable for many types of actuarial work.

The table below compares Python to Excel in certain areas.

Model Attributes	Winner
Easy-to-Use, Easy-to-Learn	Excel
Ad-hoc calculations and prototypes on small datasets (less than 100k rows)	Excel
Ad-hoc calculations and prototypes on larger datasets (more than 100k rows)	Python
Robustness in production	Python
Checking results and sharing results	Excel
Ability to achieve best practices	Python
Automation / amount of manual effort required	Python
Stability	Excel
Expected rate of future improvements	Python
Big data	Python
Speed	Python
Advanced scientific computing capabilities	Python
Machine learning capabilities	Python

While the table above implies that Python and Excel are competitors, in fact they work quite well together. Python libraries such as OpenPyXL and XlWings allow Python users to import data from spreadsheets, manipulate spreadsheets, and save calculation results to Excel. Python users can get the best of both worlds by using Python and Excel together.

For example, in our Python workflows, we typically do most or all of the calculations in Python, and at the end of the run Python will export various summaries of the results to Excel, where they can be manually checked by Excel users. In this way we can get the best of both worlds, as Python is a fast robust calculation engine and can do the calculations in an automated manner, whereas Excel is a good tool for sharing results and performing sense-checks on the run results.

In practice, I think that actuaries will continue to use Excel for the foreseeable future, as it is a good tool, but I think that actuaries will increasingly migrate larger models and processes to Python to take advantage of Python's strong production capabilities.

[10] I am not aware of any definitive list of best practices for actuarial modelling, however I think that "Best Practices for Scientific Computing" by Wilson et al. gives good guidance in this area.

https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745

SQL

SQL is widely used in the actuarial community. SQL is very good at doing what it was designed to do – namely to store and extract data from relational databases. However it is often used for other tasks which it is not good at, such as performing complex calculations on the data. SQL's top-down declarative style of coding enables users to write fast and robust queries, however it can be quite complex and can be difficult to spot-check results at a granular level. This makes it difficult to write complex queries and test them, and it introduces key person risk. Python is better at doing complex calculations than SQL, because users can choose between using a SQL-style of calculation (such as groupby queries and table joins) or an imperative Excel/VBA-style of calculation (such as if statements, for loops and array calculations). It is easier to spot-check and sense-check results in Python because it is easier to dig into results and view intermediate results, whereas SQL typically outputs the end results and can be very difficult to view intermediate results.

Python can connect to SQL servers and can run SQL queries. This enables Python users to get the best of both worlds, as they can have a single workflow which contains both SQL and Python elements. Python can be used for calculations and data analysis, and Python can pull data from the SQL server and save results to the SQL server. I expect to see actuaries migrate more and more calculations and processes from SQL to Python as it is easier to develop and test calculations and processes in Python, and Python interlinks well with companies' existing databases and systems.

R

Python wasn't widely used for data science tasks until the development of its data science libraries over the last decade. However it has experienced phenomenal growth in recent years and more and more actuaries have started using it.

R is a data science tool which is popular within the actuarial community. Like Python, R is an open-source programming language with modular libraries and is being actively developed by the R community. R is a statistical language, built by statisticians for statisticians. Python is a general purpose language which happens to be good at data science. Both tools are excellent at actuarial modelling and data processing due to the large overlap between statistics, data science and actuarial work. They seem to have similar levels of capability at present. R may be better than Python for ad-hoc analyses and once-off statistical investigations. Python may be better than R for repeated analyses and building robust regular processes. On most major surveys, the number of R users has been steady recently, whereas the number of Python users has grown strongly. It can be difficult to do a like-for-like comparison of the popularity of R and Python because R is a niche statistical language whereas Python is a general purpose language and is used for many things, including data science, web development and systems programming. It is safe to say that Python is far more popular and growing more quickly in general usage around the world, but in the particular niche of data science / statistics / actuarial modelling they are both very popular.

Why Actuaries Use Python

There are many reasons why actuaries have started using Python, including:

- It has a fast, powerful calculation engine
- It was designed for high productivity and rapid model development
- It can handle millions of rows of data
- It is an excellent "glue" language, which can be used to combine many individual models into a single end-to-end process
- It is interoperable with other languages, such as Excel, R, Julia, SQL and big data systems
- It can import data from Excel, csv, SQL, big data systems and many other filetypes, do calculations, and export results back to Excel, csv, SQL, PDF, and big data systems
- It contains powerful visualisation software
- It was designed for production and is robust
- It is a high-level language, which is relatively easy to learn and relatively easy to use.
    - Many university and online courses
    - Good online documentation and Q&As
- It is a general purpose programming language which can be used for any type of task
- It is free open-source software, with a permissive license (no procurement issues)
- It is widely used, and still growing in popularity

The Future of Python

Python is a mature, stable language. The core Python libraries are likely to remain quite stable over time, with incremental changes every year, for the foreseeable future. The current roadmap does not include any major changes. This stability is important because it provides a stable base which the rest of the Python ecosystem can be built on.

Python's ecosystem of external libraries should continue to improve in the coming years. The modular structure of Python's ecosystem is one of the main strengths of Python. It enables Python to be an industry leader or a fast follower in multiple industries. Many of the older libraries are quite stable and reliable. Many of the newer libraries contain exciting new functionality and are adding more functionality all the time. If there is a need for some new functionality, the owners of existing libraries can add the new functionality to their existing library, or developers can create a new library which has the required functionality. This enables Python to develop industry-leading capabilities, through continuous improvement of existing libraries or the development of new libraries.

Python is one of the most popular languages in the world and has thousands of developers continuously upgrading its various libraries. In my opinion, the other data science languages will struggle to keep up with Python's pace of improvements because they aren't as popular and don't have as many developers working on them. As long as Python continues to be popular, its capabilities should continue to improve at a fast rate. I think that the future is bright and we should continue to see continuous improvements in the Python ecosystem over time.

How can Milliman help?

Companies that are starting out with Python will face a number of challenges, including modelling challenges (determining the best approach to take and the best libraries to use); IT challenges (getting comfortable with using open-source software, package management, version control systems); personnel challenges (key person risk, ensuring that the Company has sufficient trained staff who know how to use Python); model validation challenges (how to validate Python models in practice) and productionisation challenges (how to ensure that your production models and processes are suitably robust).

Here in Milliman, we can help you to overcome these challenges and to build models and processes in line with best practices. Milliman can assist you with all aspects of your Python needs, including:

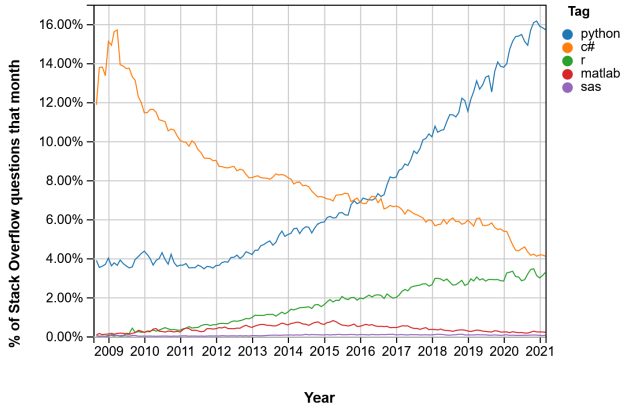
- Training
- Identifying applications
- Identifying suitable libraries and techniques for particular circumstances
- Model development and validation
- Process development and validation
- Constraints and practical challenges

For further information, please contact your usual Milliman consultant or those below.

Appendix A: Popularity

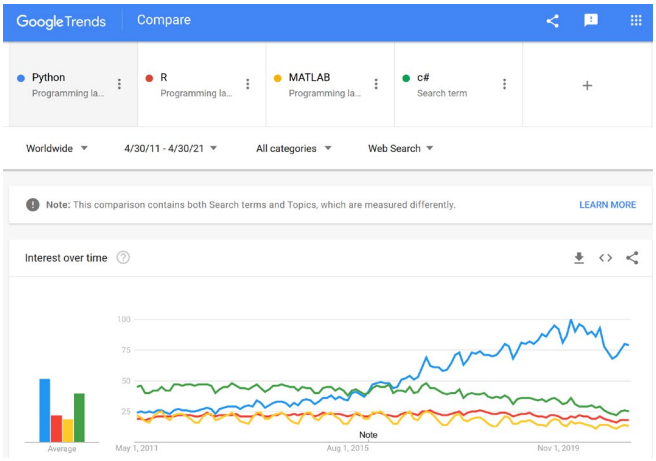
STACK OVERFLOW TRENDS [11]

(Python in blue, R in green)



GOOGLE TRENDS: INTEREST OVER TIME [12]

(Python in blue, R in red)



IEEE SPECTRUM: THE TOP PROGRAMMING LANGUAGES 2020[13]

Rank	Language	Type	Score
1	Python		100.0
2	Java		95.3
3	C		94.6
4	C++		87.0
5	JavaScript		79.5
6	R		78.6
7	Arduino		73.2
8	Go		73.1
9	Swift		70.5
10	Matlab		68.4

TIOBE INDEX FOR MAY 2021[14]

May 2021	May 2020	Change	Programming Language	Ratings	Change
1	1		C	13.38%	-3.68%
2	3	↑	Python	11.87%	+2.75%
3	2	↓	Java	11.74%	-4.54%
4	4		C++	7.81%	+1.69%
5	5		C#	4.41%	+0.12%
6	6		Visual Basic	4.02%	-0.16%
7	7		JavaScript	2.45%	-0.23%
8	14	⇈	Assembly language	2.43%	+1.31%
9	8	↓	PHP	1.86%	-0.63%
10	9	↓	SQL	1.71%	-0.38%
11	15	⇈	Ruby	1.50%	+0.48%
12	17	⇈	Classic Visual Basic	1.41%	+0.53%
13	10	↓	R	1.38%	-0.46%
14	38	⇈	Groovy	1.25%	+0.96%
15	13	↓	MATLAB	1.23%	+0.06%

11 https://insights.stackoverflow.com/trends?tags=python%2Cc%23%2Cr%2Cmatlab%2Csas

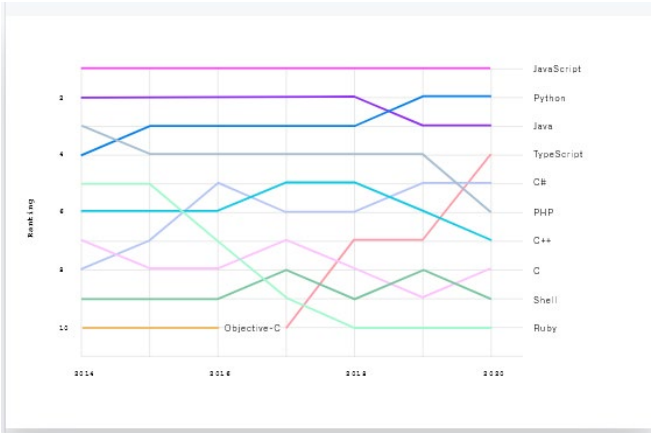
12 https://trends.google.com/trends/explore?date=2011-04-30%202021-04-30&q=%2Fm%2F05z1_,%2Fm%2F0212jm,%2Fm%2F053_x,c%23

13 https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020

14 https://www.tiobe.com/tiobe-index/

GITHUB: THE STATE OF THE OCTOVERSE 2020[15]

Top Languages since 2014 (Python in dark blue):



Milliman

Milliman is among the world's largest providers of actuarial and related products and services. The firm has consulting practices in life insurance and financial services, property & casualty insurance, healthcare, and employee benefits. Founded in 1947, Milliman is an independent firm with offices in major cities around the globe.

ie.milliman.com

CONTACT

Donal McGinley
donal.mcginley@milliman.com

Aisling Barrett
aisling.barrett@milliman.com

[15] https://octoverse.github.com/

© 2021 Milliman, Inc. All Rights Reserved. The materials in this document represent the opinion of the authors and are not representative of the views of Milliman, Inc. Milliman does not certify the information, nor does it guarantee the accuracy and completeness of such information. Use of such information is voluntary and should not be relied upon unless an independent review of its accuracy and completeness has been performed. Materials may not be reproduced without the express consent of Milliman.