APPY INSIGHTS: THE NEW FRONTIER IN PETROPHYSICAL SOFTWARE PLANNING

Mario Martins Ramos, Fernando Vizeu, Rodrigo Dutra, José Augusto Vitorino Dias, João Vitor Alves Estrella, Jordan Jusbig Salas Cuno, Ana Carla dos Santos Pinheiro, Fábio Júnior Damasceno Fernandes, Antonio Fernando Menezes Freire, Wagner Moreira Lupinacci

Copyright 2024, held jointly by the Society of Petrophysicists and Well Log Analysts (SPWLA) and the submitting authors.
This paper was prepared for the SPWLA 65th Annual Logging Symposium held in Rio de Janeiro, RJ, Brazil, May 18–22, 2024.

ABSTRACT

The meteoric rise of the Python programming language did not happen in isolation. Alongside this development, there was a notable amplification in the processing capacity of contemporary computers. Together, these technological advances made it possible for computational operations, which were previously considered expensive and time-consuming, to become more accessible and implementable. This is especially true in the context of academic centers, where research and technological development find fertile ground. Within the vast field of research, petrophysical problems have reaped the benefits of these innovations. With expanded processing capabilities and the versatility of Python, researchers now have the ability to work with massive data sets, especially when it comes to oil wells. Operations ranging from simple data extraction from files in the LAS and DLIS formats, to tasks demanding intense computational power, like machine learning and inversion techniques, have become more efficient and achievable. However, despite these promising advances, one challenge persists. Many of the routines and algorithms that are meticulously developed during academic research are set aside after these activities conclude. These algorithms, rich in potential application, are forgotten, and if there's an interest in reusing them later on, there is often a need to almost redevelop from scratch. It is in this context that the relevance of this work arises. We aim to present practical solutions to this challenge, and one of the pillars of this solution is the software APPy (Petrophysics Evaluation in Python - acronymous in portuguese). This software aims to bridge the gap between researchers, both those with deep knowledge in programming and those unfamiliar with the field. APPy offers a platform where the production, reuse, and optimization of algorithms become more integrated and efficient processes, eliminating the barrier that often prevents the reuse of valuable solutions. In this way, we seek a more connected academia, where knowledge is shared and maximized for the benefit of research and practical application.

INTRODUCTION

Python, over the years, has seen phenomenal growth, and it has become one of the most popular programming languages worldwide. Initially developed in the late 1980s by Guido van Rossum, Python's simplicity, readability, and versatility have made it the first choice for many developers and industries (Python Software Foundation, 2001). Its growth can be attributed to a combination of factors. One significant reason is its wide application in emerging fields, such as data science, artificial intelligence, machine learning, and web development.

Several studies and surveys, including those from Stack Overflow, have highlighted Python's soaring popularity. Stack Overflow, a major platform for developers to learn and share their knowledge, has observed an increasing number of questions related to Python over the years (Stack Overflow, 2017). This is indicative of a growing community of Python developers and enthusiasts. Additionally, the language's extensive libraries and frameworks, coupled with a supportive community, have made Python an attractive choice for both beginners and experienced developers (**Figure 1**).

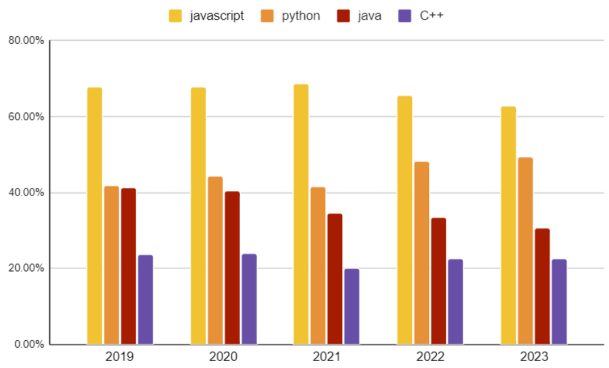

Figure 1: Most popular languages survey (Stack Overflow, 2019, 2022).

Also, based on the most recent data from Stack Overflow's annual developer survey, JavaScript maintains its status as one of the most popular and widely used programming languages in the world (**Figure 1**). For several years running, an overwhelming majority of developers have reported using JavaScript, with its dominance evident across both front-end and back-end development domains (State of JS, 2021). The data suggests that the language's adaptability, coupled with the continual emergence of modern libraries and frameworks, has solidified its importance in the ever-evolving landscape of web technologies. Additionally, the active and expansive community support evident on platforms like Stack Overflow indicates that JavaScript's relevance is not merely a reflection of its historical significance but also a testament to its current vitality and potential for future innovation.

Considering database language, SQL's enduring relevance stems from its ubiquity across most relational database management systems, its standardization by ANSI, the optimized performance of relational databases, its maturity from decades of use, and its persistent demand in various tech job roles (Winand, 2012). However, its prominence might be challenged in the future by the rise of NoSQL databases, potential advancements in newer query languages, and integrated platforms that provide query solutions without the direct need for SQL (Avantika, 2022). Despite these challenges, as of 2023 to 2024, SQL's foundational position in data management suggests it will likely remain significant for years to come (Stack Overflow, 2022).

The rapid growth of popularity of programing languages, results in one last question to answer: Is still valid to develop softwares, or everyone should learn some programing languages? While the production of software remains crucial in our technologically driven world (U.S. Bureau of Labor Statistics, 2023), with a growing demand seen in sectors like machine learning, IoT, and augmented reality, the need for individuals to learn programming is more nuanced. A basic understanding of programming concepts is beneficial for many, akin to how understanding arithmetic doesn't make everyone a mathematician. As digital transformation becomes more pervasive, underscored by reports like the World Economic Forum's "The Future of Jobs Report 2020" (World Economic Forum, 2020) it's clear that a foundation in digital literacy is becoming increasingly important. However, this doesn't necessarily translate to proficiency in specific programming languages for everyone. Instead, understanding the logic and problem-solving approach inherent to programming may be more vital.

Consequently, despite the substantial proliferation of programming languages, there remains a valid need for the development of software applications to cater to the increasing demand for emergent technologies. Furthermore, the employment of the aforementioned technologies (Python, SQL, and Javascript) appears to be of significant relevance in the development of such software applications.

Besides the global trend that drives software development in multiple application fields, the specific scenario of petrophysical software outlines a particularly distinctive niche within the technological spectrum. This software area is often highlighted, especially in the academic context, due to a series of challenges and opportunities that fuel innovation. Firstly, the cost associated with the licenses of these softwares are often high, making it frequently unfeasible given the budget constraints of many academic and research projects. Additionally, the operational opacity and the lack of transparent documentation regarding the functionalities and methods implemented in these software programs limit adaptability and customization. This limitation becomes particularly significant when considering the implementation of new methodologies and approaches, a challenge for both experienced developers and professionals who lack extensive programming experience. Thus, there's a growing demand for the development of more accessible, transparent, and adaptable solutions in the realm of petrophysical software. Therefore, this "environment" was conducive to the emergence of the petrophysical evaluation software APPy.

METHODOLOGICAL PREMISES OF THE APPY SOFTWARE

The foundational principles guiding the development of the APPy software were twofold. First and foremost, it sought to address the primary requirements for software tailored to the academic needs pertaining to petrophysical evaluation within university environments. Secondly, it was imperative for APPy to establish itself as both competitive and dependable in the broader software landscape. Subsequent sections will delve deeper into these specific requirements and the rationale behind their prioritization.

Academic necessities and development environment.

Petrophysics is commonly intertwined with geophysics, engineering, and geology. A significant portion of students in these fields have encountered, or will encounter, Python at some juncture in their academic journey (Guo, 2014). It's thus anticipated that many students will incorporate Python into their research. Even those who don't work with Python directly often possess a foundational understanding of the language, enough to utilize code written by others. This familiarity has made Python a unifying language among students, fostering collaboration and innovation.

Given this context, the foundational design principle for the APPy software is its identity as a 'Python-centric' tool. This entails that the majority of its functionalities are linked to the Python language, ensuring that most students can effortlessly interact with and enhance the software to meet their specific academic requirements or those of their research group.

Adopting Python in academic settings addresses three primary challenges posed by other software:

- Cost-efficiency: Python is free.
- Transparency: As an open-source, interpreted language, Python's workings are accessible and understandable.
- Modularity: Python boasts a plethora of libraries, including those tailored for geosciences and petrophysics.

While these advantages are particularly appreciated by students familiar with Python, they may pose barriers for those who don't regularly engage with the language.

Competitiveness in the broad software landscape.

In today's highly competitive software landscape, it's paramount for software applications to have an edge, even during their developmental phases. To achieve this, it's indispensable for the software to adhere to two primary development methodologies: microservices and incremental development. Microservices architecture allows for flexibility and scalability by breaking down an application into smaller, independent services that communicate with one another. On the other hand, incremental development emphasizes the importance of continuous improvement by building software in small, manageable increments, ensuring that each new feature or enhancement is thoroughly tested and integrated. Adopting these strategies not only fosters adaptability and responsiveness but also positions the software favorably in a rapidly evolving market. (Fowler, 2016; Cohn, 2005; Rola and Kuchta, 2015).

Based on Fowler (2016), microservices are small, individual applications that work together to form a larger system. For a system to be reliable, each microservice must be stable, able to handle increased demand, and recover quickly from failures. Effective monitoring, clear documentation, and secure practices are also essential. In essence microservices can consistently deliver top performance in real-world conditions. In the APPy software, the concept of microservices is primarily implemented through the use of stable and reliable libraries for both Python and JavaScript. The microservices architecture is seamlessly integrated into APPy's core framework, particularly within its command structures. APPy can be operated from the command prompt, executing its primary functions via specific commands. These commands facilitate a range of well-log operations, from project creation and data import to advanced machine learning tasks. An simplified version of this operations are show in the **Figure 2**.

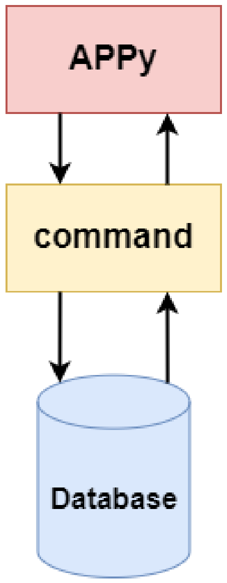

Figure 2: Illustration of how all interactions between users on APPy and the database are facilitated through commands.

The command structure offers a compelling solution to a prevalent problem in many software systems: the absence of modularity. Within APPy, enhancing the platform with new functionalities is made straightforward for users. All they need to do is introduce a new command alongside the existing ones.

Another development methodology prominently utilized in the APPy software is incremental development. This approach ensures that the software remains partially functional even in its early stages of production, permitting the client to engage with the product and offer feedback (Rola and Kuchta, 2015). A classic illustration of incremental development can be found in **Figure 3**.

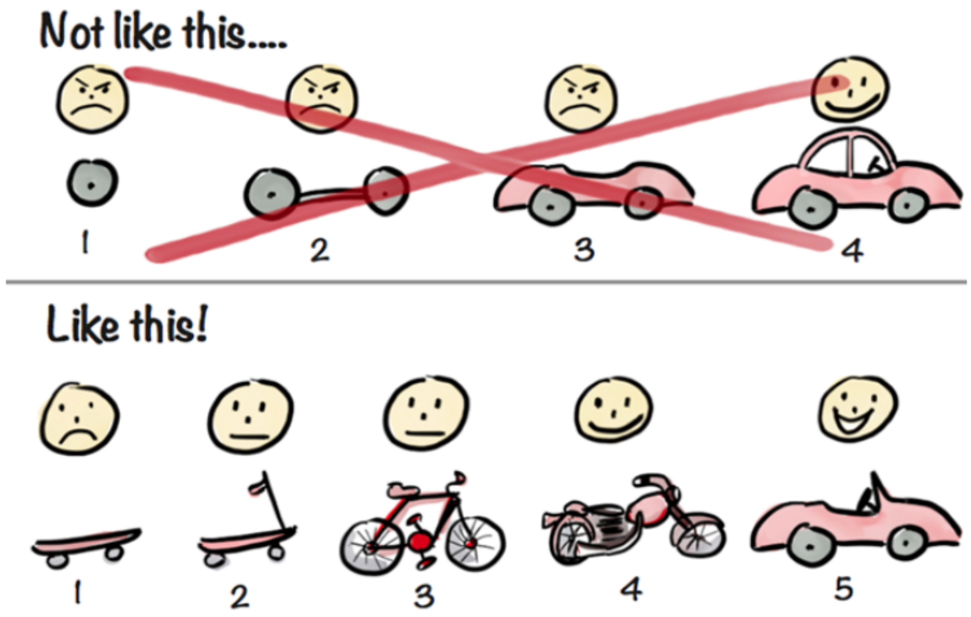

Figure 3: Schematic illustration of incremental development: In the top scenario, the client receives only the final version of the software; in the bottom scenario, the client can use the software at each stage of development (Kniberg, 2015).

Overall APPy structure

To address both academic demands and the competitiveness of the software, the decision was made to divide it into three parts: Stoneforge, APPy-core, and APPy-gui. This division allows separate teams to work on distinct portions of APPy with greater autonomy. By doing so, developers can exercise more creative freedom without compromising the software's overall structure and development process. This separation of components ensures that improvements or changes in one section do not adversely impact the others. Moreover, it provides clear delineation of responsibilities, allowing each team to specialize and hone their expertise in their respective areas, leading to a more efficient development cycle.

Considering the overall operation of APPy, the APPy-core directly accesses the Stoneforge library by importing the library and executing its features. Within the APPy-core's source code, there is an API (Application Programming Interface) that allows for the retrieval of information from the database via the HTTP protocol. Consequently, through this API, the database can be viewed in a browser. This type of access is essential for APPy-gui to access and manipulate well data through an interactive interface. A schematic of this process is presented in **Figure 4**

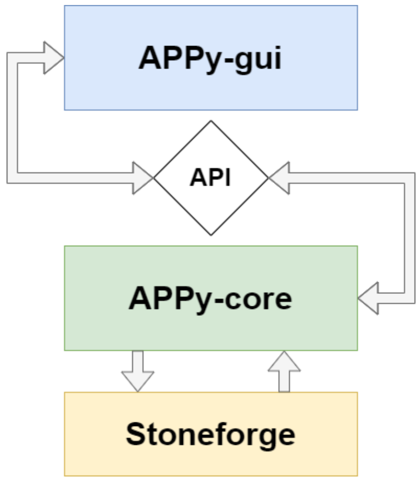

Figure 4: Appy conceptual estructure:

Stoneforge.

Stoneforge is a cutting-edge petrophysical library developed in Python. This versatile library encompasses a spectrum of methodologies, ranging from classic techniques for reservoir property estimation—like porosity, clay content, and saturation—to more advanced modules focused on rock physics, pseudo-wells, and machine learning (GIECAR, 2021).

One of the foundational aspirations of the Stoneforge library is to act as a gateway to the APPy platform. This has been architected with inclusivity in mind, ensuring that Python enthusiasts of varying proficiencies can not only utilize the platform to its fullest but also contribute to its growth by integrating new functionalities.

For the intermediate Python user with a foundational understanding of petrophysics, Stoneforge simplifies many processes. They can easily import well data, preprocess it, and perform various operations—all without needing to have the APPy software installed. Furthermore, Stoneforge's commitment to open-source principles greatly augments the transparency of APPy. Given that Stoneforge lays bare many of APPy's primary features, users and developers can have a clearer understanding and appreciation for the platform's capabilities and functionalities.

APPy-core.

The APPy-core essentially serves as the "heart" of APPy. The entire software was originally designed around this core, with functionalities added later on. Although one can still use the APPy-core natively via command lines, this feature is quite limited. In its original design, the APPy-core acts as a manager for both database and petrophysical projects. When creating a project, an APPy user establishes two databases: one in SQL (sqlite) and another in HDF5.

SQL, a widely recognized database language that has been around since the 1970s (Boyce and Chamberlin, 1973), remains popular to this day (Stack Overflow, 2022). On the other hand, HDF5, another database language, might not be as well-known as SQL. However, it bears some structural similarities to SQL, with the added benefit of being more suited for storing vast amounts of data. These databases house well data, alongside other details like color codes, mnemonic patterns, and curve units.

Furthermore, the APPy-core has an API designed to simplify the development of a separate graphical user interface (GUI) from the main program. This API can be accessed via a browser using a specific HTTP port. Through the API, one can access the database and issue commands.

Adding new functionalities to APPy is done through commands, which consist of two files: the first has a .appy extension and contains a list of arguments, along with a header for GUI access, and the second file contains the desired functionality coded in Python.

APPy-gui.

APPy-gui is a specialized interface designed primarily for the visualization and analysis of petrophysical and well log data. The interface is constructed using the Vue.js framework, a choice made for its component-driven architecture, which enables modular and maintainable code structures. These components facilitate the dynamic rendering of complex graphs and charts, a feature essential for deciphering intricate data patterns typical of petrophysics.

The backend operations of APPy-gui are anchored on a Node.js server. Node.js, with its non-blocking I/O and event-driven model, provides the capability to handle numerous simultaneous client requests, which is especially crucial when dealing with large and multifaceted petrophysical datasets. The interplay between the GUI and its data source is orchestrated through a Python-based API constructed with Flask and Flask RESTful. Flask offers a nimble framework for creating web services, while Flask RESTful extends this capability, providing a structured way to build endpoints for data retrieval, updates, and other CRUD operations. As such, the combined structure ensures efficient data flow between the backend and the APPy-gui, granting users the capability to manipulate and visualize their data seamlessly.

Conclusions.

The development of APPy began in 2017, and it has now matured over a span of more than six years. Throughout this time, a myriad of students have collaborated on its evolution. While some devoted extended periods to its growth, others played a shorter, yet still significant, role. Every contributor, irrespective of their tenure, has left a distinct imprint on the application. The current version of APPy stands as a testament to the collective effort and dedication of these budding scholars. Beyond offering them an avenue to hone their skills in Python, the project also provided invaluable hands-on experience in software development, aiding their academic and professional journey.

ACKNOWLEDGMENTS

First and foremost, The authors would like to extend their gratitude to everyone involved in the software's development, especially Thiago Rebeque, Ana Carolina, Rishi Kumar Ray, João Reis, and Yuri Alves. We would also like to acknowledge the dedicated users of APPy over its six-year development phase: Jeniffer Nobre, Thaís Candido, Luane Erbe, Carolina Ferreira da Silva, Bruna Maia Imbuzeiro, Lídia Waltz, Isadora Dutra, and Juliana. Our appreciation extends to the Department of Geology and Geophysics, with a special mention to professors Rodrigo Bijani, Francisco Romério Abrantes Júnior, and

Victor Matheus Joaquim Salgado Campos. Lastly, our heartfelt thanks go to PETROBRAS for their valuable support, which was instrumental in the development of this software.".

NOMENCLATURE

SQL	Structured Query Language
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
ANSI	American National Standards Institute

REFERENCES

Avantika, M., 2022. Rise of nosql and why it should matter to you. URL `https://www.simplilearn.com/rise-of-nosql-and-why-it-should-matter-to-you-article`. Accessed: 2023-09-23.

Boyce, R. F. and Chamberlin, D. D., 1973. Using a structured english query language as a data definition facility. *Research Report / RJ / IBM / San Jose, California*, **RJ1318**. URL `https://api.semanticscholar.org/CorpusID:45314360`.

Cohn, M., 2005. *Agile Estimating and Planning*. Prentice Hall.

Fowler, S., 2016. *Production-Ready Microservices*. O'Reilly Media, Sebastopol, CA.

GIECAR, 2021. Stoneforge. `https://github.com/giecaruff/stoneforge`.

Guo, P., 2014. Python is now the most popular introductory teaching language at top u.s. universities. URL `https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext`. Accessed: 2023-09-23.

Kniberg, H., 2015. Agile development illustrated. URL `https://www.scrum.org/resources/blog/what-iterative-incremental-delivery-hunt-perfect-examplen`. Accessed: 2023-09-23.

Python Software Foundation, 2001. Python history and versions. URL `https://www.python.org/doc/versions/`. Accessed: 2023-09-23.

Rola, P. and Kuchta, D., 2015. Implementing scrum method in international teams—a case study. *Open Journal of Social Sciences*, **03** (07): 300–305. doi: 10.4236/jss.2015.37043. URL `http://dx.doi.org/10.4236/jss.2015.37043`.

Stack Overflow, 2017. The incredible growth of python. URL `https://stackoverflow.blog/2017/09/06/incredible-growth-python/`. Accessed: 2023-09-23.

Stack Overflow, 2019. Developer survey. URL `https://survey.stackoverflow.co/2019education-ed-level-learn/`. Accessed: 2023-09-23.

Stack Overflow, 2022. Developer survey. URL `https://survey.stackoverflow.co/2022education-ed-level-learn/`. Accessed: 2023-09-23.

State of JS, 2021. The state of javascript 2021 survey. URL `https://stateofjs.com/`. Accessed: 2023-09-23.

U.S. Bureau of Labor Statistics, 2023. Software developers: Occupational outlook handbook. URL `https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm`. Accessed: 2023-09-23.

Winand, M., 2012. *SQL Performance Explained*. Winand, Markus, Vienna, Austria.

World Economic Forum, 2020. The future of jobs report 2020. URL `https://www.weforum.org/reports/the-future-of-jobs-report-2020`. Accessed: 2023-09-23.

ABOUT THE AUTHORS

Mario Martins Ramos, Fernando Vizeu, Rodrigo Dutra, José Augusto Vitorino Dias, João Vitor Alves Estrella, Jordan Jusbig Salas Cuno, Ana Carla dos Santos Pinheiro, Fábio Júnior Damasceno Fernandes Are all members of GIECAR - (Grupo de Interpretação Exploratória e Caracterização de Reservatórios).

Antonio Fernando Menezes Freire, Wagner Moreira Lupinacci Are professors of Universidade Federal FLuminense (UFF), members of GIECAR and also members

of the INCT-GP (Instituto Nacional de Ciência e Tecnologia Geofísica do Petróleo).