An Overview of Why Scientists should Learn to Program in Python

Arvind Kumar Pandey, Assistant Professor

Department of Computer & IT, ARKA JAIN University, Jamshedpur, Jharkhand, India

Email Id- arvind.p@arkajainuniversity.ac.in

ABSTRACT: Software is becoming more important in many fields of scientific study, including powder diffraction. For scientists, knowing how to program a computer is a fundamental and valuable ability. This article discusses the three approaches to programming languages and why non-expert programmers choose scripting languages. Python is a highly efficient programming language for research, and scientists are increasingly using it. Python is also one of the most user-friendly programming languages. The language is explained, as well as some of the numerous add-on packages that may be used to expand its capabilities, such as numerical calculations, scientific visuals, and graphical user interface programming. Python is a powerful programming language that is translated, interactive, object-oriented, and general-purpose. Guido van Rossum built it between 1985 and 1990. Python software is also accessible under the GNU General Public License, much like Perl (GPL). This course provides sufficient knowledge of the Python programming language.

KEYWORDS: Graphics, Numerical Analysis, Python, Programming, Software.

1. INTRODUCTION

Scientists were probably the most enthusiastic early users of technology when they discovered how computers might help them with their job[1]. In the 1950s, 1960s, and 1970s, when computers became more widely accessible, scientists embraced them even more, developing software to conduct long calculations and automate time-consuming data gathering activities. Through initiatives for direct-methods phasing and crystallography, few disciplines have been changed as dramatically as crystallography[2]. Computerized tools and least-squares refinements The idea of capturing measurements on a strip chart recorder sounds as antiquated as it was fifty years ago. Taking a horse-drawn carriage to work. There is currently an abundance of highly specialized software available[3]. Powder diffraction programs and practitioners are also drawn numerous general-purpose tools, such as spreadsheets and word processors programs for word processing When it comes to computers and their usage, Despite the fact that science's power has increased, there is an irony in the fact that much fewer people have access to it. Scientists are learning how to create software[4]. This is a unique opportunity loss.

While current apps are capable of a lot, In science, there are always basic activities that no one has coded in a user-friendly manner. Furthermore, there are always some fresh concepts that should be explored. The question is also a source of worry. Who will be responsible for writing the next generation of scientific software? Even when scientists collaborate with computer experts on a project, When it comes to software development, it's still extremely useful. The programming method is well-understood by scientists. As a result, we suggest that more scientists learn to code. Not only have computers improved, but they've also gotten more affordable not only are they common, but so are the abilities required to master programming have been simplified, at least with the help of certain high-level computer software languages[5].

There are many programming languages to choose from, each with its own set of advantages and disadvantages. A computer language was created to meet an unmet need in its predecessors' capabilities, yet any language in widespread usage has benefits and drawbacks. Some of them will be discussed further down. Professionals are not exempt. Programmers usually specialize in the usage of a computer a limited number of languages, but since scientists have so little time, they usually choose to learn only one[6]. A language that can meet as many requirements as feasible. Multiple kinds of tasks are required in scientific computing be carried out in tandem: It is necessary to conduct numerical or other types of calculations. Usually with the help of a scientific software program, symbolic analysis is performed[7].

In order to prevent recreating the wheel, the findings must be communicated to the user, generally with scientific visuals; the user must interact with the program, commonly via a graphical user interface (GUI or web browser) in contemporary programs; documentation is provided to the user required to explain the

program and teach people how to utilize it to make use of the software Python is a programming language. Ability to do all of this and more, and therefore meets the needs of scientists. In addition, the writers of python is considered to be one of the most user-friendly languages by the authors of this article[8]. Despite being one of the finest programming languages for beginners to learn Automation, for example, is a good example of an environment for scientific reasons.

Numerical analysis and image processing It also includes a users may start with the most basic capabilities and work their way up to more sophisticated features in this environment. As your abilities improve, you'll be able to do things like object-oriented programming. Python may be used in two ways to solve a problem. Scientific computation: "numerical" vs. "symbolic" scientific computation systems[9]. The former necessitates the use of software designed especially for handling numerical data (e.g., Matlab, Octave, R-language, etc.) and other high-level programming languages), while the latter does not manipulates symbolic phrases or equations that have no end. (For example, Mathematica and Maple) Since all of these characteristics are present, it's written in Python and may be used to tackle a variety of problems issues.

Python is simple enough for a beginner to pick up and use. She or he can rapidly convert their thoughts into programs, but if a bigger project requires it, she or he may learn to write object-oriented code, speed up code, and build advanced scientific visualization with complicated graphical user interfaces. Python is also a cross-platform open source software package released under the "Python Software License." The interpreter is distributed under the terms of the "Foundation License," which allows it to be freely distributed. There are no hidden licensing fees for programs built in it. Costs. This makes it suitable for classroom usage as well. Since replication of work is a cornerstone in the laboratory of scientific methodology Many of the terms will be explained in detail below.

Python's features and why it's so useful for almost everyone from quick-and-dirty format data translation through model fitting, instrument automation, and even first-principles theory, all elements of scientific computing are covered. We start with a comparison. Python will be compared to some of the major programming languages frequently used in scientific computing, followed by a brief demonstration. A summary of Python syntax what is Python such a useful tool for scientific computing . Python's syntax is not just beginner-friendly, but it also has a lot of features. Packages that make it possible to automate many typical programming tasks dozens of lines of code rather than hundreds or thousands Hundreds of thousands in other languages There are tens of thousands of them. As a result, we've highlighted a few of them in this article. In order to emphasize what makes Python so useful, as well as to orient newcomers to some of the most useful resources. Finally, since a search for Python resources may be time-consuming, generate a massive amount of data, many of which are useless We offer a section for items that are of little usefulness to a scientist. which includes advice on a variety of topics educational resources to assist scientists in learning to program in Python[10].

1.1 An Overview Of Modern Programming Languages :

Although there are many different programming languages, they can be divided into three groups, and it is essential to grasp the advantages and disadvantages of each. Another big software called a compiler interprets the original computer languages, such as Fortran, Algol, and Cobol, as well as more contemporary languages, such as C and C+ +. Because it includes the actual hardware instructions suitable for the circuitry and operating system of a particular kind of computer, the output of a compiler process is referred to as machine code. A software built for Windows will not run on a Linux computer or a Macintosh (unless in an emulator), and a program compiled for one version of an operating system will not work on earlier or newer versions of the same operating system in certain circumstances.

Code should be easily portable across computer systems after recompilation, however this is not always the case. The primary benefit of utilizing compilers is that they generate the most powerful and efficient programs since they take direct advantage of the computer and operating system architecture, but they come at a higher cost in terms of learning time and slower coding progress for the programmer. As computers grew more ubiquitous, efficiency became less important, interpreted programming languages were created. In this case, a master program known as an interpreter examines the program to be executed, translating each line of code as it is encountered and performing the required actions.

Although recent interpreted languages include Perl, Ruby, Tcl/Tk, and Python. These languages also contain advanced instructions that make it possible to write frequently performed operations in a straightforward way. This tends to decrease the size of programs while also making the authoring process easier. Another

benefit of interpreted languages is that a programmer may directly enter specific instructions into the interpreter and test the output. These two characteristics make it possible to create programs with the least amount of work. Most interpreters are tested to operate in the same way on a variety of computer platforms, which implies that interpreted language program code will generally execute in the same way on a variety of machines. Scripting languages are frequently referred to as such since they were originally used for creating basic and short programs known as scripts.

There are several drawbacks to using interpreted programming languages. One need is that the interpreter be installed on each machine that will execute the application. Furthermore, interpreted programs do not execute as quickly as programs written in compiled languages. Computational speed, on the other hand, is often less of a concern than the time required to develop and debug a program. In the very rare situations when speed is a major issue, only a tiny portion of a program's lines would have an effect, and many interpreted languages have methods in place to alleviate these bottlenecks. The most recent kind of computer language is a cross between compiled and interpreted languages. A compiler is used to transform the program into bytecode, which is a set of basic and generalized instructions. These instructions, unlike the output of a compiler, are not unique to any kind of computer hardware.

A virtual machine takes these hardware-independent bytecode instructions and executes the real operations by translating them, command by command, to the actual hardware and operating system. As a result, bytecode will execute on any computer that has the virtual machine ported. Java is the most well-known example of this kind of language, and it is supported by a broad variety of hardware, including all popular personal computers and an estimated 109 mobile phones across the globe. It's even built into the majority of online browsers. The requirement to translate bytecode to hardware instructions adds an additional layer of cost, lowering speed to somewhere between compiled and unoptimized interpreted languages. The code is just as complicated and time-consuming to create as compiled code, but it has the benefit of being portable, unless machine-specific packages are used to enhance performance, graphics, and other features. Although these languages are appealing for professional programming jobs, they need a higher degree of expertise before they can be used in science, and we cannot suggest them for informal usage.

1.2 Python Applications:

Many apps are built on top of Python or include Python programming capabilities. Sage , for example, is a freestanding mathematical software package that includes capabilities for algebra, combinatorial optimization, numerical mathematics, number theory, and calculus. It makes use of a number of opensource computing programs in order to provide a "viable free open source alternative to Magma, Maple, Mathematica, and Matlab." It is not a Python package that can be imported, but it does provide a useful Python programming interface. Sage may also be used for cloud-based computing, which distributes a job over many machines at different locations. Other examples are Enthought's PyXLL, which enables Python to be integrated into Spread sheets, and Spyder (https://pythonhosted.org/spyder/), a code creation and computing environment.

The IPython software is one of the most useful tools to come from the Python community. The IPython output (notebook file) architecture is based on commonly used protocols such as JSON, ZeroMQ, and WebSockets, as well as CSS for HTML style and display and interoperable (HTML, Markdown, PNG, and JPEG) for content storage. Because many of the features are generic, and because of the use of open standards, other programs can read the notebook files, the IPython software is now being used as an application framework for other languages. The IPython notebook may be used for a variety of tasks, including code development, teamwork, documentation, and debugging. It includes profiling tools, so that if a program is running slowly, the (typically brief) portions of code that are taking up the most time may be easily discovered and changed. It works well with matplotlib and NumPy and may be used to interactively examine data with integrated visuals and computing, much like Matlab. Finally, IPython allows for the execution of Python commands at a distance. A big calculation may be distributed to a supercomputer using this method. The remote computer may have a totally different architecture than the controller since Python commands and intrinsic data objects are portable. The most difficult aspect of learning IPython is mastering all of its features.

1.3 Visualization and image-processing packages in Python:

Python is very good at displaying data graphically, which is an essential component of scientific study. There are two types of graphical displays: 2D representations and pseudo-three-dimensional (3D) representations.

Objects in 3D are shown as being seen from a certain direction, although that orientation may typically be changed interactively. This enables the brain of the observer to interpret the depiction as if it were a real physical item. The interpretation and transformation of digital pictures is a similar computational problem. SciPy has some image processing capabilities, but more advanced capabilities are described farther down.

- *Matplotlib*: Matplotlib is a scientific 2D graphics toolkit that includes a programming interface for creating vector graphics and rendering them for output (Hunter, 2007). The benefit of vector graphics is that the charts produced may be shown at extremely high resolution, as in a journal article or presentation. Matplotlib may also be used interactively to plot findings, similar to how it would be done in a spreadsheet. To quickly create visuals, utilize simple Matplotlib commands. NumPy is used to calculate an array of 500 points across the range of 0.0 to 6, and then a new array is generated containing the sine of each value in the previous array, as an illustration of how easy it is to utilize Matplotlib. Finally, a plot is created, with the x-axis representing the previous plot but in units of, and the y-axis representing the sine value.
- *Chaco*: Chaco (http://code.enthought.com/projects/chaco/) is a scientific graphics package that produces beautiful static 2D charts rather than creating "hard-copy" graphs. Although it may be used for dynamic data visualization and exploration, it lacks the complexity of Matplotlib. However, it is quite simple to use.
- *Bokeh*: (http://bokeh.pydata.org) is an interactive visualization toolkit for Python and other languages that allows you to create dynamic graphical presentations. It provides excellent speed with big datasets and is integrated with Python, with visuals that match the commonly used D3.js JavaScript library. It's compatible with IPython Notebook. Matplotlib and Bokeh are both compatible. It allows programmers to quickly build apps that provide high-performance interactive display of big datasets.
- *Mayavi*: Mayavi is a Python framework for 3D visualization of scientific data that is part of the Enthought Tool Suite (http://code.enthought.com/projects/mayavi/) (ETS). Mayavi data corresponds to 3D objects that may be "rotated" on the computer display, enabling users to better comprehend the 3D nature of the presented information. Mayavi is based on the very advanced VTK (http://www.vtk.org/) technology, which also has a Python interface. OpenGL is a hardware/software combination protocol that is used to create dynamic 3D display visuals. Alternatively, using the PyOpenGL package, extremely high-performance interactive graphics may be done in Python using direct OpenGL calls, although this is much more complex and not recommended for most applications.

2.DISCUSSION

Python is a high-level, general-purpose programming language that is interpreted. The use of extensive indentation in its design philosophy promotes code readability. Its language features and object-oriented approach are aimed at assisting programmers in writing clear, logical code for both small and large-scale projects. Python is garbage-collected and dynamically typed. It supports a variety of programming paradigms, including structured (especially procedural) programming, object-oriented programming, and functional programming. Because of its extensive standard library, it is frequently referred to as a "batteries included" language. Guido van Rossum started working on Python in the late 1980s as a replacement for the ABC programming language, and Python 0.9.0 was published in 1991. In the year 2000, Python 2.0 was released, bringing with it new capabilities like as list comprehensions and a garbage collection mechanism based on reference counting. Python 3.0 was introduced in 2008, and it was a significant upgrade of the language that was not backwards compatible.

3.CONCLUSION

Python is a strong programming language, as we've seen and discussed, but it's also basic enough to be taught in beginning high school classes. It's simple to learn, yet it has a lot of power for professional software development. The vast number of scientific packages available, of which just a handful were shown here, demonstrates Python's great value in the hands of scientists. The authors urge scientists to learn Python and use it into their own research. we would like to emphasize that the basic knowledge of programming is important for a computer scientist. And because of complexity of sophisticated programming languages it seems that learning them is not that simple. First programming language imprinted students' knowledge and willingness to study programming and computer science in full. So it is essential to apply specific and appropriate approach to the process of learning the language. Today several of top institutions either utilize

Python to teach introduction to programming to their students or build and teach their own basic compiler that can be readily understood by students. As a consequence we can conclude that students grasp programming well with the usage of Python. They have command line tool which enables them to test their ideas quickly.

REFERENCES

[1] V. M. Ayer, S. Miguez, and B. H. Toby, "Why scientists should learn to program in Python," in *Powder Diffraction*, 2014.

[2] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, 2011.

[3] K. J. Millman and M. Aivazis, "Python for scientists and engineers," *Computing in Science and Engineering*. 2011.

[4] D. Kuhlman, "A Python Book: Beginning Python, Advanced Python, and Python Exercises," *A Python B.*, 2009.

[5] A. Meurer *et al.*, "SymPy: Symbolic computing in python," *PeerJ Comput. Sci.*, 2017.

[6] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in Python using PyMC3," *PeerJ Comput. Sci.*, 2016.

[7] M. L. Hines, A. P. Davison, and E. Muller, "NEURON and Python," *Front. Neuroinform.*, 2009.

[8] J. Demšar *et al.*, "Orange: Data mining toolbox in python," *J. Mach. Learn. Res.*, 2013.

[9] T. De Smedt and W. Daelemans, "Pattern for python," *J. Mach. Learn. Res.*, 2012.

[10] A. Patil, D. Huard, and C. J. Fonnesbeck, "PyMC: Bayesian Stochastic modelling in Python," *J. Stat. Softw.*, 2010.