Step Into the Digital Age with Python

Jacob Albrecht ▪ Sage Bionetworks

The programming language for everybody is for chemical engineers, too.

Chemical engineers work on a diverse range of problems, but address them using similar approaches. For example, one commonality is the way in which we complete routine tasks and office work (*e.g.,* manipulating remote data) using networked computers. Good engineers have access to a wide variety of tools, and they have the skills to know when and how to use them. As the work changes, we need to update our toolbox and become more collaborative. If you are looking to take your work to the next level, or just want to try something new, consider adding Python to your toolbox.

The Python programming language is free, is considered easy to learn, and offers flexibility for nearly any digital task. It is maintained by the nonprofit Python Software Foundation. Python was first released in 1991 as an easy-to-use programming language that would have broad appeal. The language is 30 years old (predating Excel's Visual Basic for Applications), but Python has seen incredible growth in just the past decade, driven by some key application areas, such as data science and machine learning.

A multitude of options are available to meet the technical needs of data analysis. Although spreadsheets and commercial software work well for most needs, situations arise that require a bespoke solution. Programming languages allow unlimited flexibility and are thus well suited to solve unique problems. Whereas other tools focus on data with logic behind it, programming solutions focus on the logic and operate on data in the background.

This article describes the many ways Python can be used for chemical engineering work in the digital age. It is not a tutorial for learning how to program in Python — abundant resources are available online, both free and paid, to learn the language for yourself. Rather, the goal of this article is to present Python's role in the modern chemical engineer's toolbox. To highlight some of the reasons why you should consider Python as your next growth area, let's start with the most compelling.

Python is already widely used by chemical engineers

Python has amassed a large following since its release. It is widely used by major technology companies across many scientific disciplines, and it benefits from the contributions of researchers. Python is considered to be an easy-to-learn programming language, and it is among the first programming languages that many students are taught today. Because more and more students are learning Python, the next generation of engineers will likely be interested in applying it to their professional work. For those out of school, Python is a great programming language to learn independently, with abundant online resources. This time spent learning the language is a smart investment for your career. For example, in the Presidential Lecture given at the 2019 AIChE Annual Meeting

(Nov. 10–15, 2019, Orlando, FL), Matt Sigelman, CEO of Burning Glass Technologies, mentioned Python specifically as a skill that adds a premium to a worker's expected salary.

It is difficult to survey the programming tools that engineers use, but one resource is public code repositories hosted on GitHub — a widely used cloud-based service that helps developers store and manage their code, as well as track and control changes to their code. By scanning GitHub for users who self-describe as chemical engineers, it is possible to analyze what programming languages they share publicly. Figure 1 shows the proportion of more than 1,500 chemical engineers who have shared different types of files in over 16,000 repositories. Python code is the most popular, beating out web technologies such as HTML, CSS, and JavaScript.

Also included in the analysis in Figure 1 are Jupyter notebooks, a rapidly growing open-source web application. Jupyter notebooks from Project Jupyter are web-enabled documents that support dozens of programming languages, with Python being a popular choice. These notebooks are useful for sharing analyses because they allow for a series of "cells" containing formatted text, executable code, and results to exist in one document that can be followed by anyone with a web browser. Behind the scenes, Python runs the code cells and updates the results. Notebooks can be run on your own computer or you can access them on a separate web server.

GitHub trends also reveal how the popularity of these tools has grown over time (Figure 2). The effect of the COVID-19 pandemic is striking: beginning in March 2020, the rate of project creation jumps by about 40% for Python and doubles for Jupyter Notebooks. At the same time, languages like C++ remained flat. This suggests that many engineers took the opportunity to build their Python skills during the lockdown.

The growing popularity of Python among chemical engineers can be attributed to the rapidly maturing capabilities external packages are adding to the language. Users are voting with their time and attention on the best way to do work. They have clearly concluded that Python is a flexible tool that meets the demands of chemical engineering work.

Python adapts to our work

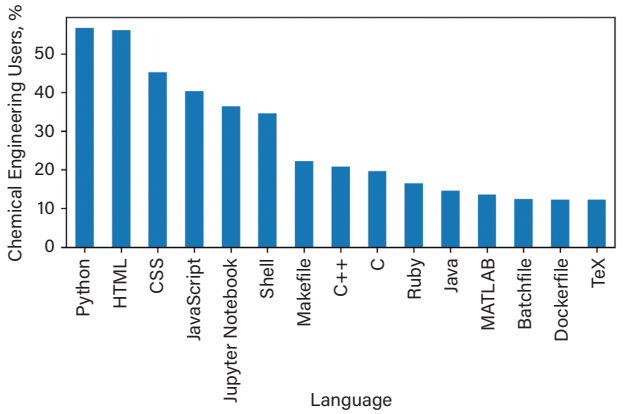
The core capabilities of Python are managed by nonprofits and government laboratories, while a mature ecosystem of packages is available that offers additional capabilities. Python's greatest power is in its flexibility, and without packages, it would not have its breadth of applications. Table 1 highlights some of the most popular enabling packages engineers use to collect and analyze data, perform calculations, and automate tasks.

In addition to these popular packages, Python has an enormous ecosystem of packages; currently, there are more than 300,000 packages on the Python Package Index, and this number continues to grow *(1)*. Because it is easy to share packages, many authors will share their code as part of publishing new work, which allows anyone to reproduce the work. It is easy and free to install most packages using the `pip` and/or `conda` package managers. In considering some common patterns that occur with computer-aided work, four themes are explored: data handling, connecting to remote data, reproducible work, and automating software.

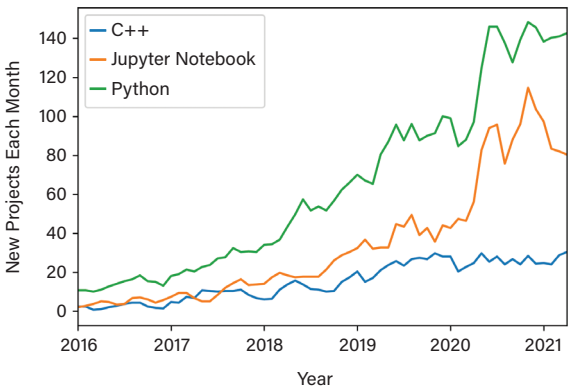
Data cleaning and analysis

As technology continues to improve, facilities are generating an increasing amount of data. This data is now typically stored on public and private networks, rather than a computer we can physically access. Yet, stored data is often organized by how it was collected, not how it will be used; tidy data that is ready for analysis is rare.

Some data is available in organized databases, but often information is only available in tables inside reports, written in Microsoft Word or PDF format. Manual work to extract



▲ **Figure 1.** Python is the most popular programming language or application used by chemical engineers on GitHub.



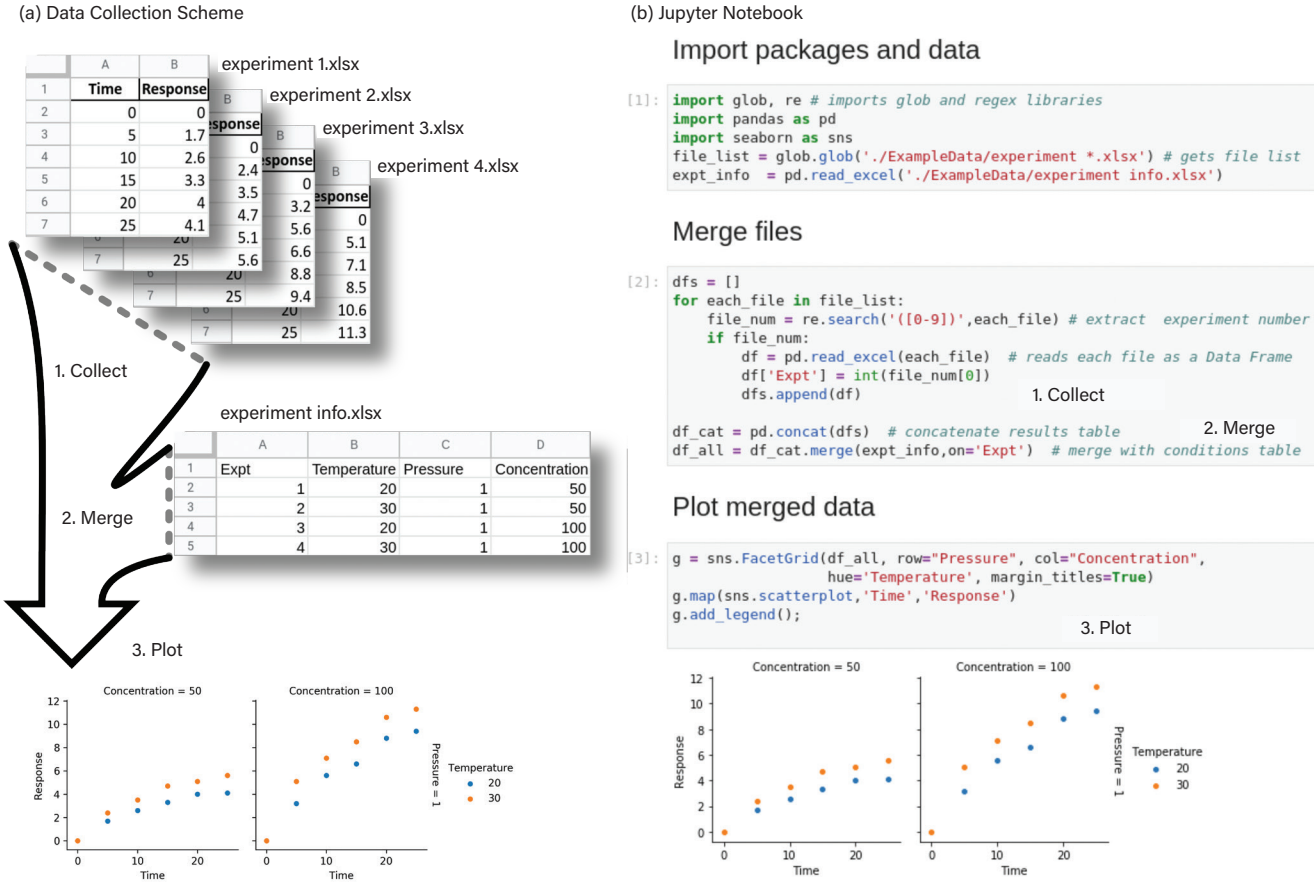
▲ **Figure 2.** The creation of Python and Jupyter Notebook project repositories by chemical engineers in GitHub greatly increased in March 2020.

and process data is appropriate for dozens to hundreds of datapoints, but becomes time consuming and error-prone for larger datasets. This data "wrangling" is a tedious task that Python can expedite. Python can read these file formats to extract the data, and scripts can be created to perform the same actions on any number of files over time. Utilizing data frames from the `pandas` package offers many convenient capabilities for working with data organized in a tabular spreadsheet format.

As a quick example, suppose that a series of experiments is conducted with separate result files generated for each of the experimental conditions. The conditions are stored separately from the data, and files are named for each condition. Figure 3 illustrates the

Table 1. Python has a mature ecosystem of packages that are commonly used for engineering tasks.

Name	Purpose	Homepage
`numpy`	Arrays and basic math functions	https://numpy.org
`scipy`	Scientific and statistical functions	https://scipy.org
`pandas`	Tabular data	https://pandas.pydata.org
`matplotlib`	Plotting	https://matplotlib.org
`selenium`	Automating web browsers	https://selenium.dev
`pyautogui`, `pywinauto`	Automating operating system graphical user interfaces (GUIs)	https://github.com/asweigart/pyautogui https://github.com/pywinauto/pywinauto
`sympy`	Symbolic math	https://sympy.org
`jupyter`	Web notebooks	https://jupyter.org
`scikit-learn`	Machine learning	https://scikit-learn.org
`flask`, `django`	Web frameworks	https://palletsprojects.com/p/flask https://djangoproject.com
`tensorflow`, `pytorch`, `keras`	Deep neural networks	https://tensorflow.org https://pytorch.org https://keras.io



▲ **Figure 3.** Python allows data spread across multiple files to be read, merged, and visualized. (a) This diagram illustrates how a collection of experimental results can be merged with experimental conditions and plotted. (b) The actual code cells to perform these steps is shown here in a Jupyter notebook.

process of collecting and merging the files, along with the actual code in a Jupyter Notebook. With Python, the files can be read and combined with the experimental conditions to create a table containing all of the data. A few more lines of code generate plots of the data. Now we can easily inspect the results and draw conclusions. In Figure 3, we can observe that the concentration factor has the biggest impact on how the response changes over time.

With only a few lines of Python code, we are able to merge the data in five files and create a useful plot. Best of all, doing this with Python code allows you to rerun and extend this analysis easily, and include any number of data files with the same format.

Networked working

Locally available data is convenient, but data shared over a network is becoming the norm. As more and more information is shared freely online, a resourceful person can ask and answer many questions quickly with Python-enabled processes. Commercial data is stored in databases or unstructured data buckets, and is easy to access with packages like `sqlite3` or `boto3`. There is abundant information available through web interfaces, and in case these interfaces are unavailable, Python makes it possible to automatically extract the raw content from webpages.

As an example of working with remote data, the U.S. National Library of Medicine recently shut down TOXMAP, a useful resource to view geographical data from the U.S. Environmental Protection Agency's (EPA's) Toxic Release Inventory (TRI), among other sources. The raw data is still accessible through the EPA's TRI web application programming interface (API), allowing for a custom analysis. Figure 4 is a screenshot of a Jupyter notebook that can download 2019 TRI data, filter it to view the quantity of waste disposed by incineration, then plot the quantities for the contiguous U.S.

The notebook has markdown cells, which add text and format the section headings, as well as four code cells. The first code cell loads `numpy`, `pandas`, a custom interface to the EPA's API, and packages for working with geospatial data (*i.e.,* `geopandas` and `geoplot`). The second cell queries the EPA database for the year 2019. The third cell filters the data for incineration disposal methods, then groups the data by facility and sums the total quantity that was incinerated at each location, keeping quantities greater than 20 m.t. A new column for the log of the quantity is created and added to the data frame. The fourth cell loads a map of the contiguous U.S., then plots the TRI data by latitude and longitude on the map, with the size and color of the marker determined by the log quantity incinerated.

Python makes these types of explorations simple, with endless options for modifying the analysis. When data can be accessed openly, using a commonly available language like Python allows for complete transparency and reproducibility. For complex work, taking the steps to ensure reproducibility is the fastest way for others to trust the veracity of the results.

Reproducible work

Reproducibility is an important consideration for anyone's work. It is even more important if you consider that collaboration occurs not only with your contemporaries, but also with your future self. It can save you time, as valuable work is often followed up with requests to repeat it. For your stakeholders, it is important that the analysis behind a decision can withstand scrutiny. For your collaborators, being able to share the code behind an analysis greatly accelerates the time to learn and maintain methodologies.

Consider the analysis of chemical engineers on GitHub presented in Figures 1 and 2. Developing those figures required accessing the public profiles of GitHub users, analyzing their code repositories, summarizing the data, and then plotting the results. The code to accomplish this task was created initially in 2019 in less than 40 lines of Python. I was able to save time by running the code again in 2021, reusing the original effort. The code and instructions are also provided as a Jupyter notebook in the accompanying repository *(2)*, so that anyone can inspect and recreate the analysis for themselves. Keeping the code in a public repository has other advantages; in this case, the original analysis was included in the 2020 GitHub Arctic Code Vault project, ensuring that it will be available for at least the next millennium.

Automation

Not every activity can (or should) be performed in Python; chemical engineers also rely on several special-purpose software tools. Once software is configured and is used for a task, the work is rarely done after the first use. If the results prove useful, the same process can be expected to be rerun in the future. While the work is done outside of Python, the activity can be automated with Python. Scripting interactions with software saves time and tedium by automating repetitive work. In general, if you can type, see, or click something on your computer display, an automation package can do the same. Python can send mouse clicks and keystrokes to your computer operating system using the `pyautogui` or `pywinauto` packages.

When interacting with specific programs, it is typically faster and more reliable to connect to the program's API if possible. Because Python has a software license that permits integration with closed-source projects, many commercial software include integration with Python (look for it in the software documentation).

Load Libraries

```
import pandas as pd
import numpy as np
from EPA_TRI import TRI_Query
import geopandas as gpd
import geoplot as gplt
```

Get Data

```
df = TRI_Query(year=['=',2019])
```

```
Downloading Records: |################################| 181857/181857 100.0% - 0s
```

Filter and Transform

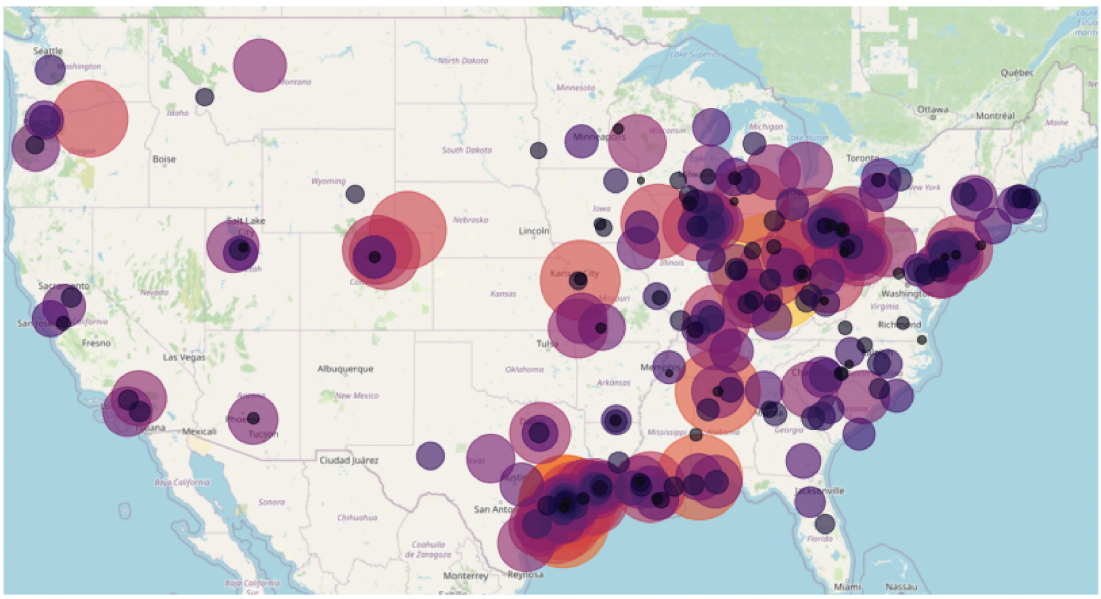
```
#sum incineration quantity by location
grouped_df = df.loc[df['TRI_TRANSFER_QTY.TYPE_OF_WASTE_MANAGEMENT'].str.contains('Incineration'),:].groupby(
    ['TRI_FACILITY.FAC_LONGITUDE','TRI_FACILITY.FAC_LATITUDE'])

subdf = grouped_df['TRI_TRANSFER_QTY.TOTAL_TRANSFER'].agg(sum).reset_index()
#filter data, more than 20 metric tons
subdf = subdf.loc[(subdf['TRI_TRANSFER_QTY.TOTAL_TRANSFER']>44090),:]

#convert to Geo DataFrame
gdf = gpd.GeoDataFrame(subdf,
                       geometry=gpd.points_from_xy(subdf['TRI_FACILITY.FAC_LONGITUDE'].values/-10000,
                                                   subdf['TRI_FACILITY.FAC_LATITUDE'].values/10000))
#add a column for the log of quantity
gdf['Log_Quantity'] = np.log10(gdf['TRI_TRANSFER_QTY.TOTAL_TRANSFER'])
```

Plot Data

```
contiguous_usa = gpd.read_file(gplt.datasets.get_path('contiguous_usa'))
ax = gplt.webmap(contiguous_usa, projection=gplt.crs.WebMercator(),figsize=(15,8))
gplt.pointplot(gdf, ax=ax,hue='Log_Quantity',scale='Log_Quantity',limits=(5,100),legend=False,
               extent=contiguous_usa.total_bounds,cmap='inferno',alpha=0.6);
```



▲ **Figure 4.** This example Jupyter notebook can access and visualize waste disposal data from the U.S. Environmental Protection Agency's (EPA's) Toxic Release Inventory (TRI).

Article continues on next page

The rapid growth in the application of deep neural networks has been facilitated by sharing and benchmarking Python implementations.

As a somewhat trivial example, the `selenium` package is a great way to record macros, then control web browsers directly. The code example in Figure 5 goes to aiche.org and searches the site for Python. In these six lines, Python imports the `selenium` package, opens a new Firefox window, navigates to aiche.org, clicks the new search button, types "Python," then clicks the submit search button. If you find yourself performing routine tasks, like accessing a website, a Python script run as a scheduled task can save tremendous amounts of time. Python has a reputation as a glue language that can connect different applications to facilitate any sort of workflow.

Python has technical computing capabilities

Python wasn't originally designed for technical computing; the built-in capabilities for mathematical functions are very limited. Thanks to its user community, and nonprofit organizations like NumFOCUS (https://numfocus.org), there is a large ecosystem consisting of hundreds of thousands of scientific and engineering packages that can add functionality. In Table 1, the packages `numpy` and `scipy` add fundamental capabilities for scientific calculations in Python. Other packages called `scikits` add functionality to `scipy` for domains like machine learning or image processing. For plotting, `matplotlib` is a standard plotting package that can generate publication-quality plots. The `seaborn` package builds on top of `matplotlib` and adds functionality for exploratory data visualization and statistical plots.

When using Python to solve technical problems, you may be looking for capabilities from classic numerical methods, or you may be interested in applying a promising new approach (*e.g.,* machine learning) to problem-solving. For either scenario, Python is quite capable.

Classic numerical methods

Timeless numerical methods capabilities used by chemical engineers include linear algebra, ordinary differential equation (ODE) solvers, and least squares regression. In Python, these capabilities have been steadily added over the years and are now very mature and well-documented, contained in the `numpy` and `scipy` packages. As a bonus, Python's symbolic mathematics package `sympy` offers excellent capabilities for symbolic algebra and calculus. Refs. 3 and 4 present ten classic chemical engineering problems that employ a variety of numerical methods that can be used to benchmark different software packages. The companion code repository on GitHub has Python solutions to these problems, and can be run at the link provided in the literature cited *(2)*.

```
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.aiche.org/")
driver.find_element_by_id("site-search-toggle").click()
driver.find_element_by_id("edit-query").send_keys("Python")
driver.find_element_by_css_selector(".button--search").click()
```

▲ **Figure 5.** These six lines of code open a web browser and automatically search aiche.org for the word "Python."

Modern machine learning and simulation

The flexibility of Python facilitates the sharing of new algorithms as they are developed. Python can be connected to more efficient code written in C, FORTRAN, or Julia to provide high performance from simple Python functions. For general machine learning, `scikit-learn` is a collection of hundreds of algorithms for classification, regression, clustering, dimensionality reduction, and more. All of these algorithms can be used with a common syntax, facilitating experimentation with different processing and modeling pipelines.

The rapid growth in the application of deep neural networks has been facilitated by sharing and benchmarking Python implementations. For example, `tensorflow` and `pytorch` are currently two of the most popular packages for deep neural networks. Once powerful foundational frameworks are released, the Python community gets to work to develop more and more user-friendly ways to access the capabilities. For example, the `keras` package can make deep learning more accessible.

Many packages have been recently released that offer optimization and simulation capabilities. For example, Pyomo (www.pyomo.org) can define optimization problems and offers flexibility by allowing for different types of input data and available solver algorithms *(5, 6)*. The Atomic Simulation Environment *(7, 8)* works in a similar way by flexibly setting up, running, and analyzing atomistic simulations with a large number of third-party calculators. This type of modularity is one advantage of meta packages in Python, allowing for easy integration of new capabilities as they are created.

Next steps

The aim of this article is to raise awareness of what the Python programming language can do for chemical engineers. Python is capable enough for professional programmers yet simple enough to be taught as an entry-level language. If you are looking to upgrade your skills, consider adding Python to your toolbox. It is free to use, free to learn, and can be used for nearly any digital task.

The next steps, should you choose to take them, include getting access to your own Python. One of the challenges

to getting started is that Python can be used in a dauntingly large variety of formats, either from the OS command line, through a Jupyter notebook, or development environments like PyCharm, IDLE, Spyder, or Visual Studio Code. Identifying resources (people or publications) to stay abreast of the latest capabilities is important; even the mature packages mentioned here may become deprecated in the future. To get started as quickly as possible, Google Colaboratory *(9)* is a fast and free way to work with notebooks online. For individuals looking to get started on their own computer, the Anaconda distribution from anaconda.com is a good place to begin.

The last piece of advice for learning Python is to be persistent; failures and errors are a part of working with programming languages. Stackoverflow.com is a very good resource for finding common questions and getting help. Being motivated is important to get started, but regularly using Python is best to build skills. I hope you find that Python lowers the barriers to what is possible with digital resources, making your own knowledge and expertise even more valuable. CEP

JACOB ALBRECHT, PhD, is currently Director of Benchmarking and Data Challenges at Sage Bionetworks, focused on enabling the evaluation of biomedical tools and algorithms. He received a PhD from the Massachusetts Institute of Technology (MIT) and a BS from the Univ. of Nebraska-Lincoln, both in chemical engineering. Previously, he was an associate director in pharmaceutical development at Bristol Myers Squibb (BMS). He has championed the adoption of modern approaches to data science and machine learning both within BMS and through pharmaceutical industry consortia.

Literature Cited

1. **Python Software Foundation,** "Python Package Index (PyPI)," https://pypi.org (accessed Aug. 3, 2021).
2. **Albrecht, J.,** "Python 4ChEs Code Repository," https://github.com/chepyle/Python4ChEs/tree/CEP2021 (accessed Aug. 3, 2021).
3. **Cutlip, M. B., *et al.,*** "The Use of Mathematical Software Packages in Chemical Engineering," Workshop Material from Session 12, Chemical Engineering Summer School, Snowbird, UT (Aug. 1997).
4. **Albrecht, J.,** "Executable Python Solutions to Chemical Engineering Problems," https://mybinder.org/v2/gh/chepyle/Python4ChEs/master?urlpath=lab (accessed Aug. 3, 2021).
5. **Hart, W. E., *et al.,*** "Pyomo — Optimization Modeling in Python," 2nd Ed., Springer, New York, NY (2017).
6. **Hart, W. E., *et al.,*** "Pyomo: Modeling and Solving Mathematical Programs in Python." *Mathematical Programming Computation,* **3** (3), pp. 219–260 (2011).
7. **Larsen, A. H., *et al.,*** "The Atomic Simulation Environment — A Python Library for Working With Atoms," *Journal of Physics: Condensed Matter,* **29**, #273002 (2017).
8. "Atomic Simulation Environment (ASE)," https://wiki.fysik.dtu.dk/ase (accessed Aug. 3, 2021).
9. **Google Colaboratory,** "Welcome to Colaboratory," https://colab.research.google.com (accessed Aug. 3, 2021).