Journal of Information Systems Education

Volume 35
Issue 4
Fall 2024

Teaching Tip

Scaffolding in Business Analytics Education: Using Python for Web Scraping

Anand Jeyaraj

Recommended Citation: Jeyaraj, A. (2024). Teaching Tip: Scaffolding in Business Analytics Education: Using Python for Web Scraping. *Journal of Information Systems Education*, 35(4), 438-450. https://doi.org/10.62273/QPEA8450

Article Link: https://jise.org/Volume35/n4/JISE2024v35n4pp438-450.html

Received:	March 31, 2024
First Decision:	June 17, 2024
Accepted:	August 3, 2024
Published:	December 15, 2024

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Journal of Information Systems Education, 35(4), 438-450, Fall 2024
https://doi.org/10.62273/QPEA8450

Teaching Tip Scaffolding in Business Analytics Education: Using Python for Web Scraping

Anand Jeyaraj
Raj Soin College of Business
Wright State University
Dayton, OH 45435, USA
anand.jeyaraj@wright.edu

ABSTRACT

A significant activity in the business analytics process is enrichment, which deals with acquiring and combining data from external sources. While different strategies for enrichment are possible, it can be accomplished more efficiently through automation using Python scripts. Since business students may not be immersed in technology skills and may be new to coding activities, instructional scaffolding may be of considerable importance. This paper describes the use of a mixed scaffolding approach involving piecewise integration and progressive integration to help students learn web scraping using Python in the limited amount of time available. Specifically, piecewise integration enables students to learn different chunks of knowledge separately and selectively integrate them as required. In contrast, progressive integration enables students to begin with the first chunk of knowledge and expand it with related chunks of knowledge. Based on performance in a segment-ending assignment and knowledge transfer to other settings, the scaffolding approach seems effective in imparting the necessary knowledge and skills to students.

Keywords: Business analytics, Scaffolding, Web scraping, Python

1. INTRODUCTION

Recent developments and advances in business analytics have raised the stakes for business professionals in virtually all domains to exploit information systems (IS) to gain greater insights into business operations and data-driven decision-making (Choi et al., 2017; Radovilsky & Hedge, 2022). A variety of IS tools and environments with different capabilities may be used to acquire, prepare, manipulate, analyze, and visualize data to facilitate business analytics activities. A significant consideration in the overall process is data enrichment, which relates to acquiring and combining data from external or third-party sources that may fill gaps in existing data, enable deeper analysis, and generate richer insights (Dahiya et al., 2023). Since enrichment activities using manual methods may be time consuming, it may be useful to develop and deploy Python scripts to automate data acquisition for enrichment (Mazilu, 2022). If equipped to handle such IS capabilities in business analytics, business students may be empowered in their future careers as business professionals.

To gain such knowledge and skills, students need an understanding of business analytics from two perspectives. First, they need to develop a high-level understanding of the business analytics process and how IS tools may be used to enable the process. For instance, students may be shown a) the various stages in the process, such as cleansing, analyzing, and visualizing the data, b) how different tools may support the different stages (e.g., Microsoft Excel can serve as a repository for transformed data vs. Python can help data acquisition and data processing), and c) the relative merits and demerits of IS tools for specific purposes. Students may be shown this high-level perspective through frameworks of the business analytics process, discussions of how the IS tools may be gainfully employed, and successful business cases of IS appropriation (Jaggia et al., 2020; Jeyaraj, 2019; Khan et al., 2019; Zhang et al., 2020).

Second, students need an understanding of the low-level details, including the process steps and how to manipulate IS tools for business decision-making successfully. This may help students understand: a) the specific steps, for instance, in cleansing data, including the types of cleansing that may be necessary for the given dataset (e.g., transforming data into a consistent format, handling missing data), b) the features and capabilities of the IS tools that could be used for specific steps (e.g., splitting an address field into street, city, state, and zip code fields using LEFT, MID, and RIGHT functions in Microsoft Excel, loading data into a Python list and apply string manipulation operators), and c) the ways to automate tasks (e.g., develop Python scripts to handle data). These require experiential learning, learning-by-doing, or hands-on activities that demonstrate how IS tools can be used for specific purposes (Kolb & Kolb, 2005; Malik & Zhu, 2023; Niiranen, 2021).

Within the context of experiential or hands-on activities, however, students may need scaffolds such as instructions, guidelines, and demonstrations for learning (Belland et al., 2022). This is particularly true when students are engaged in learning new concepts they have not encountered before or learning to apply old concepts in new ways. In the context of using Python for business analytics, students first need guidance on the features, statements, and libraries such that they can understand the environment and later need instruction on how to develop scripts for specific needs. These can be best accomplished using scaffolds that enable students to get started with the Python environment and which can be faded or eliminated as they gain greater experience and the ability to transfer their learning to different contexts (Janson et al., 2020; Sharma & Hannafin, 2007). Identifying ways to help students quickly grasp new concepts along with the ability to independently apply such concepts is crucial for their growth and effectiveness in professional settings dealing with business analytics.

This paper presents a scaffolding approach that has helped business students grasp and apply concepts and techniques of web scraping using the Python environment. The remaining sections of the paper provide a review of prior literature on scaffolding, an overview of the instructional scaffolding approach to enable student learning of web scraping, a discussion highlighting student performance including a critical reflection of the scaffolding approach, and a conclusion that motivates technology instruction for business students.

2. PRIOR LITERATURE

Scaffolding is one of the common mechanisms by which students can be given support to facilitate their learning (Memmert et al., 2023). It is generally accepted that scaffolding enables students to solve problems, complete specific tasks, or achieve goals that may be beyond their unassisted efforts (Wood et al., 1976). The scaffolding process relies on knowledgeable experts such as teachers or peers who can provide necessary support for student learning (Kim & Hannafin, 2011; Vygotsky, 1978). Technologies can be used at times in place of experts to facilitate learning (Guthrie, 2010; Memmert et al., 2023). Scaffolding methods have been applied to impart learning in a variety of technology contexts including design science, database, projects, and information search (Bunch, 2009; Guthrie, 2010; Memmert et al., 2023; Raes et al., 2012).

Scaffolds may help communicate and illustrate the concepts, principles, and techniques that enable student learning. Different kinds of scaffolds, such as procedural guidelines, learning materials, checklists, assessment questions, student-teacher interactions, structured mentoring, worked-out examples, progressive practice, collaborative discussions, interim deliverables, and in-class learning to acquire specific skills have been proposed (Anand & Mitchell, 2022; Bunch, 2009; Guthrie, 2010). Scaffolds can be static or dynamic as necessary—static scaffolds may be pre-planned to some extent based on typical obstacles or challenges in the learning process whereas dynamic scaffolds are somewhat emergent based on actual difficulties students encounter during the learning process (Brush & Saye, 2002). A mix of both approaches may be needed since static structured task steps may be appropriate in the context of rudimentary skills acquisition, but dynamic approaches would be necessary in contexts involving high-order cognitive skills since learning needs are emergent (Rosenshine & Meister, 1992; Yelland & Masters, 2007).

Regardless of the different approaches, a significant aspect of scaffolding is to select a suitable learning task that can sustain the learning interests of students and describe how learning activities can be broken up into smaller chunks to enable student learning (Winkler et al., 2021; Yelland & Masters, 2007). Students can be taken through the different chunks to enable their learning and mastery of content. As shown in this paper, it is possible to design chunks and sequence them in at least two ways. In the first approach, termed as *piecewise integration*, students may first learn the different chunks separately and then selectively integrate learning from the separate chunks to accomplish specific tasks. In the second approach, termed *progressive integration*, students could begin with the first chunk and gradually build on it to expand the capabilities by incorporating learning from other related chunks. This paper describes a mixed approach in which both piecewise integration and progressive integration were used to enable student learning of web scraping with Python.

3. INSTRUCTIONAL SCAFFOLDING

3.1 Learning Context

The instructional scaffolding approach was applied in an introductory course on business analytics taken by undergraduate students in a business school at a public university located in the Midwestern United States. Students are typically in their junior or senior years of college when taking the course, which introduces them to various principles and techniques for data acquisition, data preparation and transformation, data analysis, and data visualization (Jeyaraj, 2019; Zhang et al., 2020). In preparing the data for analysis, enrichment can be accomplished using purely manual methods (e.g., using a browser to visit a website and key in the desired data into an Excel worksheet), semi-automated methods (e.g., downloading a public dataset into an Excel workbook and use VLOOKUP or similar function to extract the necessary data into the analysis worksheet), or fully automated methods (e.g., use a Python script to visit a web page, extract the relevant data, and save into an Excel worksheet) (e.g., Mazilu, 2022). Hence, one of the major learning modules in the course is the development and use of Python scripts to gather publicly-available data from third-party websites.

Since the course is open to students from all business disciplines, it is not possible to make assumptions about their existing knowledge of Python programming. IS students entering the course may have been exposed to programming principles and techniques in entry-level programming courses that introduced languages such as VB.NET while finance students may have used specific capabilities (e.g., lookup stock market indices using Yahoo Finance API) using Python. But the vast majority of students from various disciplines including marketing, accounting, management, economics, and supply chain management do not possess prior knowledge of coding and have not used Python. Thus, in the limited time available during a typical semester (since the course also introduces other content related to business analytics such as data analysis and data visualization), students will have to be introduced to the coding fundamentals of Python, the use of different libraries relevant for data handling, and the methods for scraping data from web sites.

3.2 Python Instruction

Table 1 depicts the major Python topics introduced in class. The stated goal on the syllabus was to accomplish Python instruction in 8 to 10 sessions, each lasting approximately an hour and 20 minutes. Most of these sessions included both theoretical discussions and hands-on activities for students to understand the content. For instance, students were introduced to the Python data types (e.g., `int`, `float`) and their relevance, the reserved words or commands (e.g., `import`, `for`) and their purposes, the general-purpose operators (e.g., +, *, !=, <, or) and functions (e.g., `len`, `max`), the operators and methods associated with data types (e.g., [] operator and append method for the list data type), and the Python libraries (e.g., `pandas`, `json`, `numpy`) in the theoretical discussions. Further, students learned about the inner workings of the assignment, conditional, and iterative statements as well as other capabilities such as named (i.e., `def`) and unnamed (i.e., `lambda`) functions, syntax rules (e.g., # for comments, : for command block), and identifying the appropriate and necessary libraries for data handling.

The Python Integrated Development and Learning Environment (IDLE) was adopted for the hands-on activities. Students were shown how and when to use the Shell window and the Editor window. The Shell window was used first to demonstrate Python capabilities including how to initialize variables and data structures (i.e., `list`, `set`, `tuple`, and `dictionary`), apply operators and methods, write to and read from data files, populate data from files into different structures such as lists, use pre-defined and user-defined functions, import libraries and examine methods in such libraries, setup different control structures (i.e., `if`, `for`, `while`), and verify intermediate results or status of variables. For instance, students can begin with a Python list such as: `languages = ['French','English','German']` and learn how to add a new item to the list using: `languages.append ('Spanish')`, arrange the list items in order using: `languages.sort()`, or extract items from the list using: languages[1:2]. Students can define a named function to compute simple interest using: `def si(p,n,r): return p*n*r/100`, create a list of random numbers that represent principal values using: `principals = random.sample(range(0, 10000), 20)` after importing the `random` library, and compute the simple interest for 5 years at 3% annual interest rate through list comprehension using: `interests = list(map(lambda x: si(x, 5, 3), principals)`. The computations can be accomplished with iterative structures using: `for ctr in range(0,len(principals)):print(si(principals[ctr], 5, 3))` as well.

Topic	Specifics	Goals
Data structures	Variables: Number, String, Boolean Collections: `list`, `set`, `tuple`, `dictionary`	▪ Understand the differences between data types ▪ Understand the capabilities of the different collections ▪ Apply data types and collections as appropriate
Operations	Arithmetic, comparison, logical operators Slice, range slice operators Membership operators: `in` Assignment operators String methods: `find`, `split`, `strip`, `upper` List methods: `append`, `count`, `sort` Set methods: `add`, `union`, `intersection` Tuple methods: `count` Dictionary methods: `keys`, `values` Assignment statement	▪ Identify operators for different tasks ▪ Identify methods for different data types and structures ▪ Apply operators and methods as necessary ▪ Specify arithmetic, relational, and logical expressions
Functions	Pre-defined numeric functions: `round`, `range` Pre-defined collection functions: `len`, `sum`, `map`, `zip` Predefined conversion functions: `str`, `list`, `set` Pre-defined input/output functions: `print`, `input` User-defined (named) functions: `def` Unnamed functions: `lambda`	▪ Identify and use pre-defined functions ▪ Develop user-defined functions for reuse ▪ Apply unnamed functions
Control structures	Conditional statements: `if`, `match` Iterative statements: `for`, `while` Special statements: `break`, `continue` Exception statements: `with`, `try`	▪ Alter typical sequence of script execution using conditional and iterative statements
Libraries	Installation of libraries: `pip install` Including libraries in script: `import`, `from` Visiting web sites: `requests` Parsing web page content: `BeautifulSoup` Handling data: `pandas` Introducing delays: `time`	▪ Identify, install, and use libraries ▪ Apply methods available in libraries

Table 1. Python Content for In-Class Instruction

The Editor window was used later to demonstrate how .PY script files with a collection of instructions can be developed, saved and archived for future use, and executed to obtain results. Table 2 describes and the Appendix illustrates the Python scripts shown in class consistent with the mixed approach of piecewise integration and progressive integration introduced earlier. Specifically, script A uses piecewise integration by incorporating Python capabilities introduced using the Shell window to students. These capabilities include reading data from files, defining functions, manipulating data on lists, using arithmetic operations, and writing data to files. Also, scripts B and C follow progressive integration since the functionality for gathering data from one web page was developed first (in script B) and the functionality for gathering data from multiple web pages was developed next (in script C by building on the fundamentals in script B). Script D also followed progressive integration.

Script	Description	Features
(A) Ohio income tax	Goal: Compute the income tax for taxable non-business income in the state of Ohio, USA. Script should include: 1) user-defined function to compute income tax 2) read collection of taxable non-business income from a .CSV file 3) save incomes and taxes to a .CSV file	▪ `def` statement to define function ▪ `return` statement for function to return the computed result ▪ `if…elif…else` statement to check income tiers ▪ arithmetic operators and assignment statement for tax computations ▪ `round` function to handle rounding of computed taxes ▪ `pandas` library (and `read_csv` and `to_csv` methods) for file data handling ▪ `map` function for list comprehension ▪ `lambda` function to specify transformation for each item on list ▪ `zip` function for stitching data across multiple lists
(B) Ohio national parks	Goal: Gather (scrape) names, locations, and types of national parks in Ohio as found in: https://www.nps.gov/index.htm Two separate scripts may be developed: 1) to fetch the web page for Ohio parks and save as a .HTML file to local device, and 2) to scrape the necessary data from the .HTML file and save results to a .CSV file	▪ `import` statement for including libraries ▪ identifying the URL or address of the web page to be fetched ▪ `requests` library to visit web site and get method to fetch web page ▪ saving web page as a text file in HTML format using open and write methods ▪ reading text file in HTML format using open and read methods ▪ `BeautifulSoup` library to parse HTML content ▪ `findAll` method to gather all data in specific tags (h3) or tag classes ("subtitle" class in p tag) ▪ `map` function for list comprehension ▪ `lambda` function to specify transformation for each item on list ▪ `zip` function for stitching data across multiple lists ▪ `pandas` library (and `to_csv` method) to save results in .CSV file
(C) National parks for four states	Goal: Gather (scrape) names, locations, and types of national parks in four states (Utah, New Mexico, Arizona, Colorado) as found in: https://www.nps.gov/index.htm Two separate scripts may be developed: 1) to fetch the web pages for the four different states and save as a .HTML files to local device, and 2) to scrape the necessary data from the .HTML files and save results to a single .CSV file	▪ `import` statement for including libraries ▪ identifying the URL or address of the web pages to be fetched, including patterns that can be used for repetition ▪ `requests` library to visit web site and get method to fetch web page ▪ define `list` of states for repetition ▪ `for` statement to repeat steps for each state ▪ saving web page as a text file in HTML format using `open` and `write` methods ▪ reading text file in HTML format using `open` and `read` methods ▪ `BeautifulSoup` library to parse HTML content ▪ examining Page Source of HTML pages to identify tag and data structures ▪ `findAll` method to gather all data in specific tags (h3) or tag classes ("subtitle" class in p tag) ▪ `map` function for list comprehension ▪ `lambda` function to specify transformation for each item on list ▪ `zip` function for stitching data across multiple lists ▪ define `list` to hold intermediate results and append method to handle new intermediate results ▪ `sum` function to flatten list of intermediate results ▪ pandas library (and `to_csv` method) to save results in .CSV file

(D) Historical stock prices	Goal: Gather (scrape) historical stock prices available at: https://www.marketwatch.com/tools/quotes/historical.asp Three separate scripts may be developed: 1) fetch price for one date (e.g., 2021) for one ticker (e.g., MSFT) 2) fetch prices for multiple dates (e.g., 2019-2021) for one ticker (e.g., MSFT), and 3) fetch prices for multiple dates (e.g., 2019-2021) for multiple tickers (e.g., MSFT, AAPL, ORCL) and save results to .CSV file	▪ `import` statement for including libraries ▪ identifying the URL or address of the web pages to be fetched, including patterns that can be used for repetition ▪ `requests` library to visit web site and get method to fetch web page ▪ define `list` of dates and tickers for repetition ▪ `for` statement to repeat steps for each date and state, including nested for statements ▪ `BeautifulSoup` library to parse HTML content ▪ examining Page Source of HTML pages to identify tag and data structures ▪ `time` library to delay accessing web sites to mimic browsing activity by humans ▪ `findAll` method to gather all data in specific tags (td) ▪ range slice operators to extract specific value from list ▪ define `list` to hold intermediate results and append method to handle new intermediate results ▪ `pandas` library (and `to_csv` method) to save results in .CSV file

Table 2. Python Hands-on Activities in Class

3.3 Scaffolding Elements
Figure 1 shows a framework introduced in class for students to assess their specific needs, identify which Python capabilities are appropriate, and complete the necessary steps by applying various principles and techniques. The discussion included different types of data sources (e.g., internal vs. external) and formats (i.e., structured data in table or row and column format, semi-structured data in JSON and XML, and unstructured data in freeform text). The goal was to be able to acquire data from one or more sources that may exist in different formats and capture them in structured formats typically in Excel. Two significant considerations underlie the data transition from sources to targets: a) the use of Python variables and other structures such as list, set, tuple, and dictionary to handle the data, and b) the use of Python libraries to move data from different sources into Python structures (e.g., `BeautifulSoup` to parse HTML pages and load data into a List) or from Python structures into the different targets (e.g., `pandas` to transform a list into a dataframe to facilitate storage in Excel). Such generalized thinking enables students to easily and effectively transfer knowledge and skills to different settings and requirements.

The framework was used prior to each hands-on scripting activity shown in Table 2. When introducing the scenario before developing Python scripts, students were asked to reflect on the framework and share their thoughts on the libraries and techniques that may be employed in developing the scripts. Since they had already been introduced to libraries as shown in Table 1, students brainstormed and identified appropriate libraries and useful strategies for finishing the scripts. Depending on the need, appropriate libraries such as `requests` and `pandas` were installed and imported into the scripts. Over time, the identification of libraries and methods (such as `read_csv` and `to_csv`) to be applied became the responsibility of students.

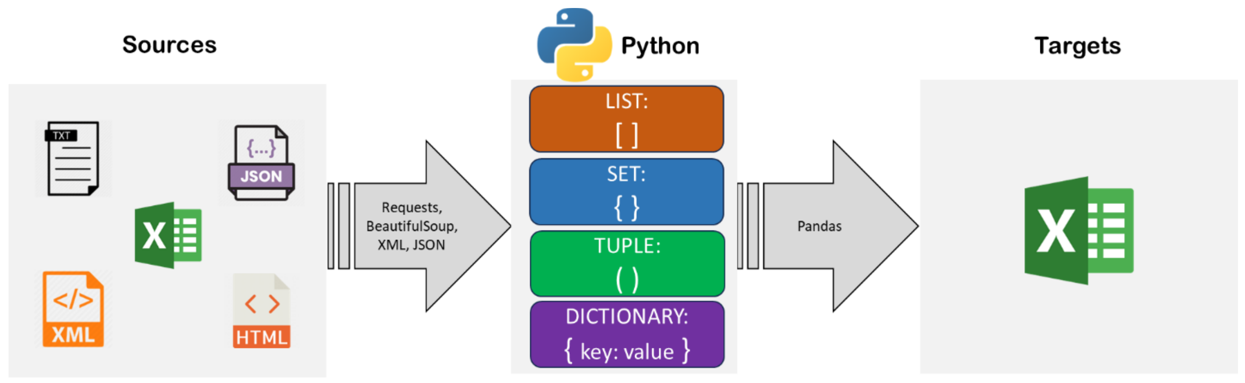


Figure 1. Framework for General Python Thinking

A basic requirement in web scraping is the need to identify the URL (uniform resource locator) or HTTP (hypertext transfer protocol) web address that needs to be accessed for fetching the web page to extract necessary data. In the Ohio parks activity described in Table 2, it was initially introduced to students through the use of a web browser. For instance, using https://www.nps.gov/index.htm to visit the home page of the National Park Service web site and choosing the state of Ohio from the dropdown list changes the URL to: https://www.nps.gov/state/oh/index.htm, which represents the specific page for Ohio parks. When starting work on the parks pages for the four states described in Table 2, students were asked to examine the URLs for the four states and determine patterns that may be helpful in designing the script. Figure 2 presents the URLs for the web pages of parks for four states: Utah, Arizona, New Mexico, and Colorado. Students determined that the URLs were generally similar except for the two-letter state code values that differed between the web addresses. It allowed the opportunity for students to reflect on how the URL can be repeatedly constructed such that it can be included in a repetitive loop structure in the Python script. Students eventually suggested state codes be held in a Python List, the URL be placed inside a `for` loop, and state codes substituted each time the loop is repeated. This resulted in the code block in Appendix (Script C1) as:

```
states = ['UT','AZ','NM','CO']
for s in states:
    url =
'https://www.nps.gov/state/'+s+'/index.htm'
```

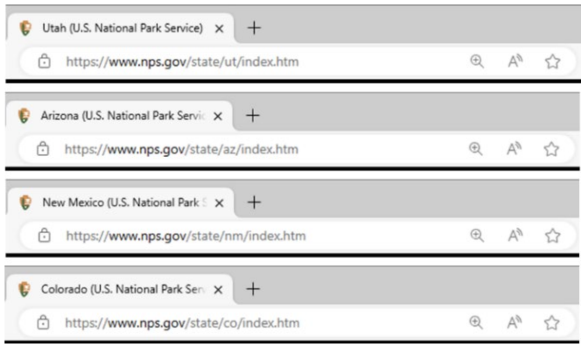

Utah (U.S. National Park Service)
https://www.nps.gov/state/ut/index.htm
Arizona (U.S. National Park Servi
https://www.nps.gov/state/az/index.htm
New Mexico (U.S. National Park S
https://www.nps.gov/state/nm/index.htm
Colorado (U.S. National Park Ser
https://www.nps.gov/state/co/index.htm

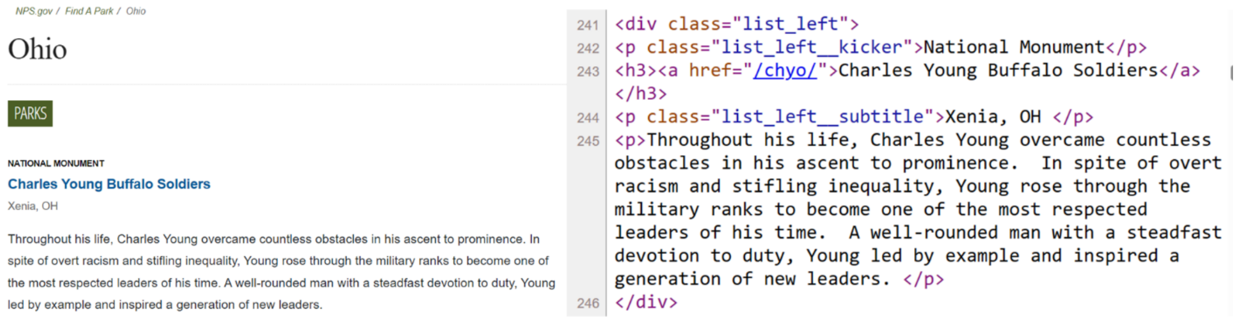
Figure 2. URL Patterns for Park Web Pages Across States

This learning of the combination of the Python list and the `for` structure was crystallized in later scripts C2, C3, D2, and D3 shown in the Appendix. As before, students progressively took greater responsibility for identifying the different elements that needed attention in developing Python scripts.

Another requirement in web scraping is the need to identify how the data displayed on a web page are actually represented in the underlying HTML (hypertext markup language) format. This requires an examination of the page source for the web page. Figure 3 shows the visual display of the first park of the Ohio parks on the left and the corresponding segment of the HTML page source on the right. This analysis shows that the park name (i.e., Charles Young Buffalo Soldiers) is embedded within `<h3>` and `</h3>` tags while the park type and park location are embedded within `<p>` and `</p>` tags. Further, the analysis reveals that different classes of `<p>` tags (i.e., `list_left__kicker` and `list_left__subtitle` classes) are used for embedding data on park type and park location. These observations enabled the use of `findAll` method within scripts B2 and C2 shown in the Appendix.

Students were able to use the learning on HTML tags and patterns as they grappled with subsequent requirements such as scraping stock prices (scripts D1, D2, and D3). Figure 4 shows the web page for MSFT stock price on the left and the corresponding HTML page source on the right. Students gained greater confidence in exploring the HTML page source and identifying the elements that embedded the desired data. For instance, Figure 4 shows that the stock prices were embedded within `<td>` and `</td>` tags, which they used in the `findAll` method.

Moreover, during the hands-on sessions in class, students were allowed time to examine the URLs, web pages, and page source documents and plan out their own solutions. When switching to more advanced scripts (e.g., from script C1 to C2), students were first instructed to develop their own Python scripts by modifying or extending the scripts they had just completed based on in-class instruction. This allowed another level of independence for students to assess their own learning and reapply their learning in different or advanced settings.


NPS.gov / Find A Park / Ohio

Ohio

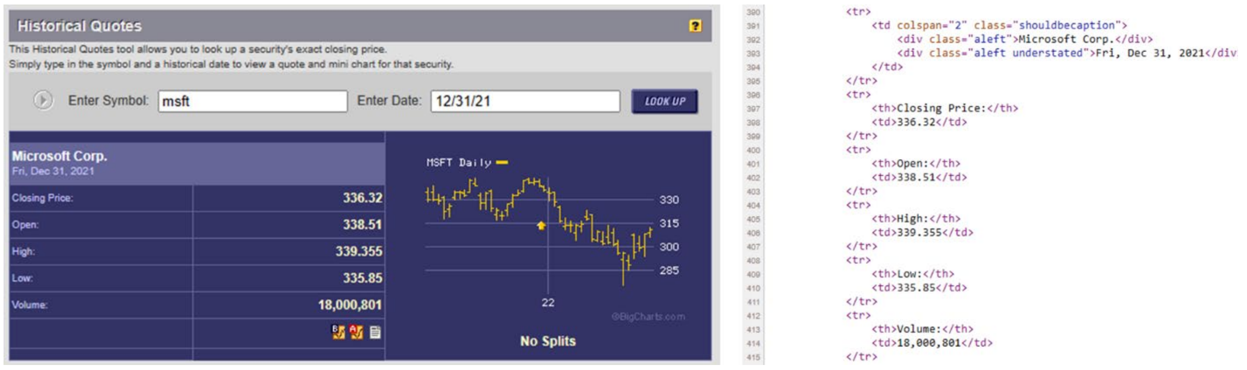
PARKS

NATIONAL MONUMENT
Charles Young Buffalo Soldiers
Xenia, OH

Throughout his life, Charles Young overcame countless obstacles in his ascent to prominence. In spite of overt racism and stifling inequality, Young rose through the military ranks to become one of the most respected leaders of his time. A well-rounded man with a steadfast devotion to duty, Young led by example and inspired a generation of new leaders.

```
<div class="list_left">
<p class="list_left__kicker">National Monument</p>
<h3><a href="/chyo/">Charles Young Buffalo Soldiers</a>
</h3>
<p class="list_left__subtitle">Xenia, OH </p>
<p>Throughout his life, Charles Young overcame countless
obstacles in his ascent to prominence.  In spite of overt
racism and stifling inequality, Young rose through the
military ranks to become one of the most respected
leaders of his time.  A well-rounded man with a steadfast
devotion to duty, Young led by example and inspired a
generation of new leaders. </p>
</div>
```

Figure 3. Ohio Parks Web Page (on Left) and HTML Page Source (on Right)


Historical Quotes

This Historical Quotes tool allows you to look up a security's exact closing price.
Simply type in the symbol and a historical date to view a quote and mini chart for that security.

Enter Symbol: msft   Enter Date: 12/31/21   LOOK UP

Microsoft Corp.
Fri, Dec 31, 2021

Closing Price:	336.32
Open:	338.51
High:	339.355
Low:	335.85
Volume:	18,000,801

```
<tr>
    <td colspan="2" class="shouldbecaption">
        <div class="aleft">Microsoft Corp.</div>
        <div class="aleft understated">Fri, Dec 31, 2021</div
    </td>
</tr>
<tr>
    <th>Closing Price:</th>
    <td>336.32</td>
</tr>
<tr>
    <th>Open:</th>
    <td>338.51</td>
</tr>
<tr>
    <th>High:</th>
    <td>339.355</td>
</tr>
<tr>
    <th>Low:</th>
    <td>335.85</td>
</tr>
<tr>
    <th>Volume:</th>
    <td>18,000,801</td>
</tr>
```

Figure 4. MSFT Stock Price Web Page (on Left) and Page Source (on Right)

4. DISCUSSION

4.1 Student Performance

The course requirements for the semester needed the students to complete an individual assignment on web scraping using Python based on the principles and techniques discussed and shown in class. The assignment had certain requirements such as 100 “cases” (e.g., tickers, cars, cities) and “multiple” attributes (e.g., different years, different characteristics) for each case, but students had considerable latitude in choosing the contexts for scraping depending on their interests, work, or discipline. The assignment also required students to conduct an analysis of the scraped data. The requirements for data analysis are not described here, but it was important for students to successfully develop Python scripts and scrape the data necessary for analysis.

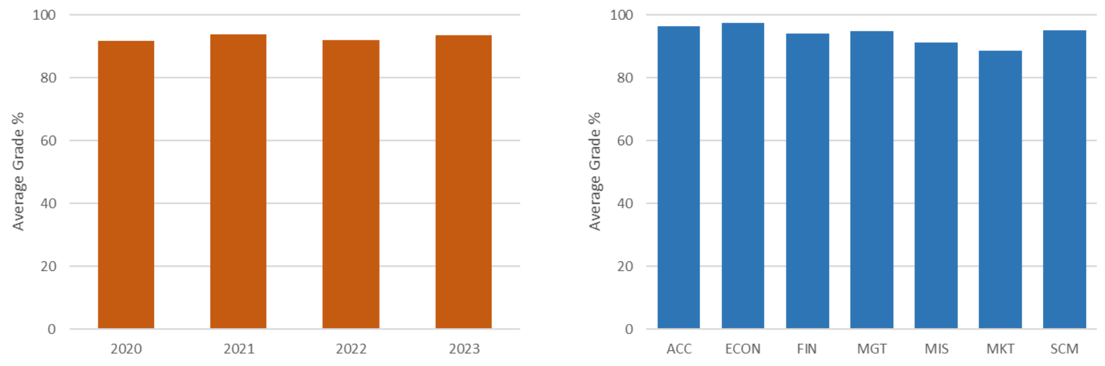
Students earned an average grade percentage of nearly 93% (with a standard deviation slightly below 7%) on the assignment, which represents excellent performance especially since it was their first coding experience, and they had been exposed to it only for a short time during the course. Figure 5 depicts student performance across semesters (represented by years since the course is offered once a year) and the primary disciplines of students. There were no significant differences across semesters or disciplines, which suggests effective learning and instructional scaffolding methods.

4.2 Knowledge Transfer

The framework was used prior to each hands-on scripting activity shown in Table 2. When introducing the scenario before developing Python scripts, students were asked to reflect on the framework and share their thoughts on the libraries and techniques that may be employed in developing the scripts. Since they had already been introduced to libraries as shown in Table 1, students brainstormed and identified appropriate libraries and useful strategies for finishing the scripts.

Students had incorporated new features or combined known features in their Python scripts to extend their learning. For instance, following the Python list for states learned in scripts C2 and C3 and similar extensions for tickers and dates seen in scripts D2 and D3, students created .TXT or .CSV files to hold “cases” (e.g., tickers, cities, movies) and incorporated capabilities in their scripts to read the data from files and populate the relevant Lists. An example of their attempt is shown below:

```
countries=[]
with open('AsianCountries.txt','r') as file:
    countries = file.read().splitlines()
```



Average grade % by semester (on left) and by students’ primary discipline (on right)
ACC: Accountancy; ECON: Economics; FIN: Finance and Financial Services; MGT: Management and International Business; MIS: Management Information Systems; MKT: Marketing; SCM: Supply Chain Management

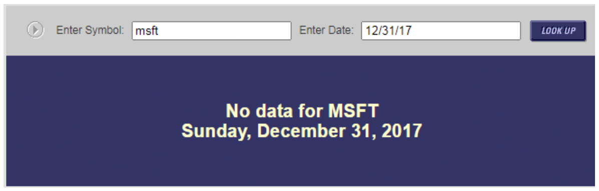
Figure 5. Student Performance in Python Assignment

They also experimented with pandas and datetime libraries (not introduced in class) to generate special dates such as the last business day for different years. Rather than specifying the dates—which would require them to manually check a calendar and pick the last business days—students found ways to automate the identification of such dates. They found ways to load those dates into Python lists to facilitate iterations. This approach also made data files redundant and served as an extension to their learning in scripts D2 and D3. An example of their attempt is shown below:

```
import pandas as pd
import datetime as dt
#obtain last business days of year in a range
lbdates =
pd.date_range('1/1/2010',periods=12,freq='BY')
#extract the dates from the previous output
lbdates = list(lbdates.date)
#convert the dates into mm/dd/yyyy format
lbdates = list(map(lambda x:
dt.date.strftime(x,'%m/%d/%Y'),lbdates))
```

Students became adept at recognizing and handling data issues. For instance, the stock ticker scripts D1, D2, and D3 assume that the dates provided are actual business days in which trading was conducted such that the stock price was actually available and been captured. However, they found that stock prices were not always available (see Figure 6 for an example). After examining the page source and finding that the relevant page did not contain the necessary information, they incorporated exceptions into their scripts. An example of their attempt is shown below:

```
alltd = soup.findAll('td')
if len(alltd) != 0:
    row.append(alltd[3].text)
else:
    row.append('')
```



No data for MSFT
Sunday, December 31, 2017

Figure 6. Non-Availability of Stock Price Data

4.3 Critical Reflection

Based on student performance on the independent Python assignment and the ways in which students transferred their knowledge to settings different from those demonstrated in class, it is possible to conclude that the teaching approach based on scaffolding was largely successful. While student performance and the diversity of contexts in the Python assignments can serve as reasonable indicators, an empirical research study involving an experiment may yield a stronger test of the effectiveness of the scaffolding approaches.

In the limited in-class time available for introducing Python to business students who are typically untrained in technology aspects, especially coding, the scaffolding of Python content demonstrated in the various hands-on activities using the Shell and Editor windows followed by the four scenarios for developing .PY scripts enabled effective student learning.

Piecewise integration was a good choice for script A since it was really the first scenario in which students learned how to bring all the chunks of Python knowledge they had learned using the Shell window. Due to the interactive nature of the Shell window, it was possible to introduce various Python capabilities separately. However, an integration of the disparate capabilities was crucial in enabling student learning because it demonstrated how to effectively combine different capabilities in developing scripts for specific scenarios.

Progressive integration was a good choice for scripts B and C as well as D since it enabled student learning to identify a pattern for completing a specific activity (e.g., fetch web page and extract necessary data from one website) and to build a repetitive structure to repeat the pattern for extending the activity (e.g., fetch web page and extract necessary data from multiple web pages).

The use of these scaffolding approaches needs careful planning to identify relevant chunks of knowledge that can be integrated using piecewise or progressive integration approaches. Consider the requirements for script A in Table 2 and the code shown in the Appendix that incorporate several chunks of knowledge. These activities were planned by considering the competencies to be introduced in class. For instance, students need to learn how to define named functions. Choosing a real-world scenario of computing the state income tax for taxable non-business income automatically provides the opportunity for students to practice assignment statements, arithmetic expressions, and conditional statements. Requiring the actual computation of income tax using the named function introduces students to the complexities of data gathering from files, calling the function using an unnamed function embedded in a list comprehension, and saving data to files. A well-constructed scenario such as script A enables the students to engage in significant learning that can be repeated as necessary in other scenarios.

Additional elements (e.g., the framework in Figure 1, the patterns in Figure 2, and the necessary data in Figures 3 and 4) that offer broad perspectives for students to use as anchors in their learning processes are also helpful in realizing the power of scaffolding activities based on how students were able to transfer their knowledge to other settings.

5. CONCLUSION

This paper contributes to the literature on instructional scaffolding by introducing and applying piecewise integration and progressive integration approaches to facilitate learning. While the approaches were applied in the context of business students learning how to use Python for web scraping activities, they can be extended to other contexts such as applications development, database modeling, and statistical analysis using Python or even other programming languages such as Java, VB.NET, or C#. Moreover, it is likely that these scaffolding approaches may be appropriate in other contexts in which students need to learn and apply new content in a limited amount of time.

6. REFERENCES

Anand, T., & Mitchell, D. (2022). Objectives and Curriculum for a Graduate Business Analytics Capstone: Reflections from Practice. *Decision Sciences Journal of Innovative Education*, 20(4), 235-245. https://doi.org/10.1111/dsji.12272

Belland, B. R., Lee, E., Zhang, A. Y., & Kim, C. (2022). Characterizing the Most Effective Scaffolding Approaches in Engineering and Technology Education: A Clustering Approach. *Computer Applications in Engineering Education*, 30(6), 1795-1812. https://doi.org/10.1002/cae.22556

Brush, T., & Saye, J. W. (2002). A Summary of Research Exploring Hard and Soft Scaffolding for Teachers and Students Using a Multimedia Supported Learning Environment *Journal of Interactive Online Learning*, 1(2), 1-12.

Bunch, J. M. (2009). An Approach to Reducing Cognitive Load in the Teaching of Introductory Database Concepts. *Journal of Information Systems Education*, 20(3), 269-275.

Choi, H. Y., Chun, S. G., & Chung, D. (2017). An Explanatory Study on the Business Analytics Program in the US Universities. *Issues in Information Systems*, 18(2), 1-8.

Dahiya, M., Malik, N., & Rana, S. (2023). Essentials of Data Wrangling. In M. Niranjanmurthy, L. Sheoran, G. Dhand, & P. Kaur (Eds.), *Data Wrangling: Concepts, Applications, and Tools* (pp. 71-90). Scrivener Publishing. https://doi.org/10.1002/9781119879862.ch4

Guthrie, C. (2010). Towards Greater Learner Control: Web Supported Project-Based Learning. *Journal of Information Systems Education*, 21(1), 121-130.

Jaggia, S., Kelly, A., Lertwachara, K., & Chen, L. (2020). Applying the CRISP-DM Framework for Teaching Business Analytics. *Decision Sciences Journal of Innovative Education*, 18(4), 612-634. https://doi.org/10.1111/dsji.12222

Janson, A., Söllner, M., & Leimeister, J. M. (2020). Ladders for Learning: Is Scaffolding the Key to Teaching Problem-Solving in Technology-Mediated Learning Contexts? *Academy of Management Learning and Education*, 19(4), 439-368. https://doi.org/10.5465/amle.2018.0078

Jeyaraj, A. (2019). Pedagogy for Business Analytics Courses. *Journal of Information Systems Education*, 30(2), 67-83.

Khan, R. A., Nadeem, A., & Ali, A. (2019). Business Analytics: A Framework. *International Journal of Computer Technology & Applications*, 10(2), 102-108.

Kim, M. C., & Hannafin, M. J. (2011). Scaffolding Problem Solving in Technology-Enhanced Learning Environments (TELEs): Bridging Research and Theory With Practice. *Computers & Education*, 56(2), 403-417. https://doi.org/10.1016/j.compedu.2010.08.024

Kolb, A. Y., & Kolb, D. A. (2005). Learning Styles and Learning Spaces: Enhancing Experiential Learning in Higher Education. *Academy of Management Learning and Education*, 4(2), 193-212. https://doi.org/10.5465/amle.2005.17268566

Malik, K. M., & Zhu, M. (2023). Do Project-Based Learning, Hands-on Activities, and Flipped Teaching Enhance Student's Learning of Introductory Theoretical Computing Classes? *Education and Information Technologies*, 28, 3581-3604. https://doi.org/10.1007/s10639-022-11350-8

Mazilu, M. C. (2022). Web Scraping and Ethics in Automated Data Collection. In C. Ciurea, C. Boja, P. Pocatilu, & M. Doinea (Eds.), *Education, Research and Business Technologies* (pp. 285-294). Springer Nature.

Memmert, L., Tavanapour, N., & Bittner, E. (2023). Learning by Doing: Educators' Perspective on an Illustrative Tool for AI-Generated Scaffolding for Students in Conceptualizing Design Science Research Studies. *Journal of Information Systems Education*, 34(3), 279-292.

Niiranen, S. (2021). Supporting the Development of Students' Technological Understanding in Craft and Technology Education via the Learning-by-Doing Approach. *International Journal of Technology and Design Education*, 31, 81-93. https://doi.org/10.1007/s10798-019-09546-0

Radovilsky, Z., & Hegde, V. (2022). Contents and Skills of Data Mining Courses in Analytics Programs. *Journal of Information Systems Education*, 33(2), 182-194.

Raes, A., Schellens, T., De Wever, B., & Vanderhoven, E. (2012). Scaffolding Information Problem Solving in Web-Based Collaborative Inquiry Learning. *Computers & Education*, 59, 82-94. https://doi.org/10.1016/j.compedu.2011.11.010

Rosenshine, B., & Meister, C. (1992). The Use of Scaffolds for Teaching Higher-Level Cognitive Strategies. *Educational Leadership*, 49(7), 26-33.

Sharma, P., & Hannafin, M. J. (2007). Scaffolding in Technology-Enhanced Learning Environments. *Interactive Learning Environments*, 15(1), 27-46. https://doi.org/10.1080/10494820600996972

Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.

Winkler, R., Söllner, M., & Leimeister, J. M. (2021). Enhancing Problem-Solving Skills With Smart Personal Assistant Technology. *Computers & Education*, 165, 104148. https://doi.org/10.1016/j.compedu.2021.104148

Wood, D., Bruner, J., & Ross, G. (1976). The Role of Tutoring in Problem Solving. *Journal of Child Psychology and Psychiatry and Allied Disciplines*, 17, 89-100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x

Yelland, N., & Masters, J. (2007). Rethinking Scaffolding in the Information Age. *Computers & Education*, 48, 362-382. https://doi.org/10.1016/j.compedu.2005.01.010

Zhang, L., Chen, F., & Wei, W. (2020). A Foundation Course in Business Analytics: Design and Implementation at Two Universities. *Journal of Information Systems Education*, 31(4), 244-259.

AUTHOR BIOGRAPHY

Anand Jeyaraj is a professor of information systems at the Raj Soin College of Business of Wright State University. He has been an instructor for 20+ years and has taught a variety of information systems courses including programming, applications development, geographic information systems, business analytics, and data visualization. He has employed learner-centered teaching and active learning techniques in his classes and received multiple teaching awards.

APPENDIX

Python Scripts Shown in Class

Script A. Income Tax Computations

```
import pandas as pd

# Named functions
def tax(inc):
    if inc < 26050:
        amt = 0
    elif inc < 100000:
        amt = 360.69 + (inc - 26050) * 0.0275
    elif inc < 115300:
        amt = 2394.32 + (inc - 100000) * 0.03688
    else:
        amt = 2958.58 + (inc - 115300) * 0.0375
    return round(amt)

# get data from CSV file
df = pd.read_csv('incomes.csv')

# move dataframe values into lists
ssns = df['id'].values.tolist()
names = df['last_name'].values.tolist()
incomes = df['nb_income'].values.tolist()

# compute taxes
taxes = list(map(lambda x: tax(x), incomes))

# move data from lists into dataframe
records = list(zip(ssns, names, incomes, taxes))

# transform records into dataframe
df = pd.DataFrame(records, columns=['SSN', 'Name', 'Income', 'Tax'])

# save results to csv file
df.to_csv('results.csv', index=False)
```

Script B1. Fetch Ohio Parks Web Page and Save to Local Device

```
# import library needed for Python to visit web sites
import requests

# visit the web site
url = 'https://www.nps.gov/state/oh/index.htm'
# fetch web page
page = requests.get(url)

# save to local device
with open('oh-parks.html','w') as file:
    file.write(str(page.text))
```

Script B2. Scrape National Parks From the Saved Ohio Parks Page

```
# library to extract content from web page
from bs4 import BeautifulSoup
# library to handle data in Excel files
import pandas as pd

# open OH web page
```

```
with open('oh-parks.html','r') as file:
    page = file.read()
# read page as HTML file
soup = BeautifulSoup(page,'html.parser')

# sift required items from soup
parks = soup.findAll('h3')
ptypes = soup.findAll('p',attrs={'class':'list_left__kicker'})
places = soup.findAll('p',attrs={'class':'list_left__subtitle'})

# clean up the sifted data by dropping tags
parks = list(map(lambda x: x.text, parks))
ptypes = list(map(lambda x: x.text, ptypes))
places = list(map(lambda x: x.text, places))

# stitch data in the three lists
data = list(zip(parks, places, ptypes))

# load combined data into a Pandas dataframe
rows = pd.DataFrame(data, columns=['Park','Place','Type'])

# save the Pandas dataframe to CSV file
rows.to_csv('oh-parks.csv',index=False)
```

Script C1. Fetch Parks Web Pages for Utah, Arizona, New Mexico, and Colorado

```
# import library needed for Python to visit web sites
import requests

# define the states for which pages need to be gathered
states = ['UT','AZ','NM','CO']

# repeat for each state
for s in states:
    # visit the web site
    url = 'https://www.nps.gov/state/'+s+'/index.htm'
    # fetch web page
    page = requests.get(url)
    # save to local device
    with open(s+'-parks.html','w') as file:
        file.write(str(page.text))
```

Script C2. Scrape National Parks for Utah, Arizona, New Mexico, and Colorado

```
# library to extract content from web page
from bs4 import BeautifulSoup
# library to handle data in Excel files
import pandas as pd

# define states for which park info needs to be gathered
states = ['UT','AZ','NM','CO']
# setup a temporary variable to hold data gathered from each web page
alldata = []

# repeat for each state
for s in states:
    # open state web page
    with open(s+'-parks.html','r') as file:
        page = file.read()
    # read page as HTML file
    soup = BeautifulSoup(page,'html.parser')
    # sift required items from soup
    parks = soup.findAll('h3')
    ptypes = soup.findAll('p',attrs={'class':'list_left__kicker'})
    places = soup.findAll('p',attrs={'class':'list_left__subtitle'})
    # clean up the sifted data by dropping tags
    parks = list(map(lambda x: x.text, parks))
    ptypes = list(map(lambda x: x.text, ptypes))
    places = list(map(lambda x: x.text, places))
    # stitch data in the three lists
    data = list(zip(parks, places, ptypes))
    # save data to the temporary variable
    alldata.append(data)

# flatten temporary data to build single list and not list of lists
alldata = sum(alldata, [])

# load combined data into a Pandas dataframe
rows = pd.DataFrame(alldata, columns=['Park','Place','Type'])

# save the Pandas dataframe to CSV file
rows.to_csv('4s-parks.csv',index=False)
```

Script D1. Fetch Stock Price for One Date for One Ticker

```
# Fetch closing price for 1 date for 1 ticker

import requests
from bs4 import BeautifulSoup

# Define URL to fetch data
url = 'https://bigcharts.marketwatch.com/historical/default.asp?symb=msft&closeDate=12/31/21'

# Fetch page and process HTML
page = requests.get(url)
soup = BeautifulSoup(page.text, 'html.parser')

# Search for TD tags on web page
alltd = soup.findAll('td')

# Extract item 3 from the ALLTD list
print(alltd[3].text)
```

Script D2. Fetch Stock Prices for Multiple Dates for One Ticker

```
# Script: Fetch closing price for Multiple dates for 1 ticker

import requests
from bs4 import BeautifulSoup
import time

# Define dates required
ydates = ['12/31/21', '12/31/20', '12/31/19']

# Repeat fetch activity for EACH date
for d in ydates:

    # Define URL to fetch data
    url =
'https://bigcharts.marketwatch.com/historical/default.asp?symb=msft&closeDate='+d
    # Fetch page and process HTML
    page = requests.get(url)
    soup = BeautifulSoup(page.text, 'html.parser')
    # Search for TD tags on web page
    alltd = soup.findAll('td')
    # Extract item 3 from the ALLTD list
    print(alltd[3].text)

    time.sleep(3)
```

Script D3. Fetch Stock Prices for Multiple Dates for Multiple Tickers

```
# Script: Fetch closing price for Multiple dates for Multiple tickers

import requests
from bs4 import BeautifulSoup
import time
import pandas as pd

# Define tickers
tickers = ['MSFT', 'AAPL', 'ORCL']

# Define dates required
ydates = ['12/31/21', '12/31/20', '12/31/19']

# Set up list to hold prices
table = []

# Repeat for EACH ticker
for t in tickers:
    # Set up temporary list to hold prices
    row = [t]

    # Repeat fetch activity for EACH date
    for d in ydates:

        # Define URL to fetch data
        url =
'https://bigcharts.marketwatch.com/historical/default.asp?symb='+t+'&closeDate='+d
        # Fetch page and process HTML
        page = requests.get(url)
        soup = BeautifulSoup(page.text, 'html.parser')
        # Search for TD tags on web page
        alltd = soup.findAll('td')
        # Extract item 3 from the ALLTD list
        row.append(alltd[3].text)

        time.sleep(3)

    # Add data row to table
    table.append(row)

# Build a Pandas Dataframe with the table of results
data = pd.DataFrame(table, columns=['Ticker','2021','2020','2019'])

# Save data to Excel
data.to_csv('prices.csv', index=False)
```

ISSN: 2574-3872 (Online) 1055-3096 (Print)