

In this issue:

- 4. Preparing the XR Workforce: Curricular Patterns of Extended Reality in Higher Education**
Tan Gürpınar, Quinnipiac University
Sana Quadri, Quinnipiac University
Shizhen (Jasper) Jia, Quinnipiac University
Luis SaCouto, Quinnipiac University
Guido Lang, Quinnipiac University
- 16. *Teaching Case***
Our Interconnected, Resilient, Modern World ... is Still Painfully and Remarkably Brittle: A Case Study of the Risks of Technology Failures
Paul Witman, California Lutheran University
Jim Prior, California Lutheran University
- 27. From Excel to Python to AI: A Connectivist Model for Introducing Coding and Prompt Engineering to First-Year IS Students**
Mark Frydenberg, Bentley University
- 44. Exploring Student Experiences with ChatGPT in Data Analytics Education: Gender, Academic Level, and Structural Model Evidence**
Mandy Yan Dang, Northern Arizona University
Yulei Gavin Zhang, Northern Arizona University
Yiyan Stella Li, Northern Arizona University
Susan Williams, Northern Arizona University
Howard Qi, Northern Arizona University
Xihui "Paul" Zhang, University of North Alabama
- 59. *Teaching Case***
Code Together, Apart: Teaching Asynchronous Team Development Workflows with GitHub
Kareem Dana, West Texas A&M University
Abraham Abby Sen, West Texas A&M University
Jeen Mariam Joy, Virginia Commonwealth University
- 75. K-12 AI Curriculum Design: A Review of Frameworks, Approaches, and Evaluation**
Ni Lei, Kennesaw State University
Zhe Zhao, Kennesaw State University
Zhigang Li, Kennesaw State University
Xin Tian, Kennesaw State University

The **Information Systems Education Journal** (ISEDJ) is a double-blind peer-reviewed academic journal published by **ISCAP** (Information Systems and Computing Academic Professionals). Publishing frequency is five times per year. The first year of publication was 2003.

ISEDJ is published online (<https://isedj.org>). Our sister publication, the Proceedings of the ISCAP Conference (<https://iscap.us/proceedings>) features all papers, abstracts, panels, workshops, and presentations from the conference.

The journal acceptance review process involves a minimum of three double-blind peer reviews, where both the reviewer is not aware of the identities of the authors and the authors are not aware of the identities of the reviewers. The initial reviews happen before the ISCAP conference. All papers, whether award-winners or not, are invited to resubmit for journal consideration after applying feedback from the Conference presentation. Award winning papers are assured of a publication slot; however, all re-submitted papers including award winners are subjected to a second round of three blind peer reviews to improve quality and make final accept/reject decisions. Those papers that are deemed of sufficient quality are accepted for publication in the ISEDJ journal. Currently the target acceptance rate for the journal is under 35%.

Information Systems Education Journal is pleased to be listed in the Cabell's Directory of Publishing Opportunities in Educational Technology and Library Science, in both the electronic and printed editions. Questions should be addressed to the editor at editor@isedj.org or the publisher at publisher@isedj.org. Special thanks to volunteer members of ISCAP who perform the editorial and review processes for ISEDJ.

2026 ISCAP Board of Directors

Amy Connolly
James Madison University
President

Michael Smith
Georgia Institute of Technology
Vice President

Jeff Cummings
Univ of NC Wilmington
Past President

David Firth
University of Montana
Director

Mark Frydenberg
Bentley University
Director/Secretary

Leigh Mutchler
James Madison University
Director

RJ Podeschi
Millikin University
Director/Treasurer

Bryan Reinicke
Rochester Institute of
Technology / Director

Jeffry Babb
West Texas A&M University
Director/Curricular Matters

Eric Breimer
Siena University
Director/2026 Conf Chair

Thomas Janicki
Univ of NC Wilmington
Director/Meeting Planner

Xihui "Paul" Zhang
University of North Alabama
Director/JISE Editor

Copyright © 2026 by Information Systems and Computing Academic Professionals (ISCAP). Permission to make digital or hard copies of all or part of this journal for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial use. All copies must bear this notice and full citation. Permission from the Editor is required to post to servers, redistribute to lists, or utilize in a for-profit or commercial use. Permission requests should be sent to Kevin Mentzer, Editor, editor@isedj.org.

INFORMATION SYSTEMS EDUCATION JOURNAL

Editors

Kevin Mentzer
Editor
Nichols College

Ira Goldman
Associate Editor
Siena University

David Yates
Associate Editor
Bentley University

Michelle Louch
Teaching Cases & Exercises
Editor
University of Pittsburgh - Greensburg

Mark Pisano
Teaching Cases & Exercises
Associate Editor
Southern Connecticut
State University

Thomas Janicki
Publisher
Univ of NC Wilmington

David Woods
Assistant Publisher
Miami University
Regionals

Paul Witman
Emeritus Editor
(2021-2026)
California Lutheran
University

Jeffry Babb
Emeritus Editor
(2016-2021)
West Texas A&M
University

Donald Colton
Emeritus Editor
(2003-2010)
Brigham Young University
Hawaii

From Excel to Python to AI: A Connectivist Model for Introducing Coding and Prompt Engineering to First-Year IS Students

Mark Frydenberg
mfrydenberg@bentley.edu
Bentley University
Waltham, MA

Abstract

This study examines the use of connectivist learning principles to teach first-year students about coding with Python in a Fundamentals of Information Systems course. The instructional design integrates tools such as Microsoft Excel, Google Colab, and AI chatbots to support conceptual understanding, promote knowledge exchange among students, and develop problem solving skills. Grounded in a connectivist approach, the module considers the relationship between students, spreadsheet logic, Python coding, and generative AI tools as inter-connected nodes that influence how students construct, transfer, and apply knowledge. Students engaged in collaborative coding activities, progressing from designing and sharing spreadsheet-based solutions to translating logical requirements into Python programs through iterative prompt engineering.

The study addresses four research questions: (1) To what extent does prior experience with Excel support students' understanding of Python programming concepts? (2) How do digital tools and peer networks support student engagement and learning in coding? (3) How do students perceive the value of learning Python for academic and career development? and (4) To what extent are students motivated to continue learning coding independently? Validated survey results indicate that students find benefit in using networked collaboration and learning tools, recognize the value in learning Python, and favor further informal study. These findings also support the use of connectivist learning techniques as an effective framework for presenting coding instruction to first-year information systems students engaged in a learning scenario shaped by personal networks, technology, and AI tools.

Keywords: Connectivism, Spreadsheet logic, Python Pedagogy, Prompt Engineering, AI Pedagogy

Recommended Citation: Frydenberg, M., (2026). From Excel to Python to AI: A Connectivist Model for Introducing Coding and Prompt Engineering to First-Year IS Students. *Information Systems Education Journal*, v24(n2), pp 27-43. DOI# <https://doi.org/10.62273/CACW2093>

From Excel to Python to AI: A Connectivist Model for Introducing Coding and Prompt Engineering to First-Year IS Students

Mark Frydenberg

1. INTRODUCTION

The ability to code, or at least understand code, has become an essential skill for many 21st century professionals (Kivunja, 2014). Approaches to teaching coding concepts to students in introductory information systems survey courses have varied from teaching the vocabulary of coding (sequence, selection, repetition, input, output, variables, functions, classes, objects, methods) by example (Parsons, 2023) to using visual block-based coding tools to implement basic algorithms and procedures (Andone & Frydenberg, 2021; Bozan & Taslidere, 2024). Emphasis is often on concepts rather than creating actual programs.

This paper presents an innovative approach to introducing first-year students in a Fundamentals of Information Systems course to coding through a three-session module. Building on their prior knowledge and proficiency with Microsoft Excel, students learn the basics of Python programming and then use generative AI to create Python code for more complex tasks such as data visualization.

Guided by principles of connectivist learning, which emphasizes knowledge construction through connections between people, tools, and ideas, this study investigates the following research questions:

- **RQ1.** To what extent does prior experience with Excel support students' understanding of coding concepts?
- **RQ2** How do digital tools and peer networks support student learning and engagement in an introductory information systems course?
- **RQ3.** How do students perceive the value of learning Python for their academic and professional goals?
- **RQ4.** To what extent are students motivated to continue learning coding after the course?

2. THEORETICAL FOUNDATIONS AND RELATED WORK

Connectivist learning theory, as introduced by Siemens and Downes (Downes, 2010; Siemens, 2005; Siemens & Downes, 2011) emphasizes these core principles:

- Learning happens by making connections, constructing and interacting with both human and technological networks.
- Learning is not only about understanding concepts, but also making connections with individuals, ideas, and technology.
- Learning from peers and being exposed to a variety of perspectives and solutions is central.
- Making decisions is an ongoing process in a rapidly changing world driven by information.
- Learning also relies on the use of technology to store, process, and generate information. Technology is essential to the connectivist classroom, enabling students to access, share, and create knowledge.

Literature suggests a growing interest in applying connectivist learning approaches in business education, with a few studies specifically focusing on learning to code. Connectivist learning empowers students to learn collaboratively, generate knowledge, interact with different tools and connect with different information sources (Gottipati et al., 2023; Utecht & Keller, 2019).

A second-year programming course at the University of Pretoria integrated connectivist strategies with scaffolding interventions from instructors and found that students valued this approach, engaging with online resources and peers, and as a result, felt more confident in taking on complex coding projects (Matthee & van Deventer, 2022).

In a recent Finnish study investigating the impact of a connectivist learning approach on teaching sustainable business in an online context, results showed that peer interactions and digital tools

were seen as active nodes of learning, allowing students to make use of technology to help strengthen knowledge and learn from different perspectives. (Dziubaniuk et al., 2023) This suggests that a connectivist approach also can be applied to learning to code, especially when learners choose tools and networks that suit their needs and learning styles. While the coding environment (Google Colab) was standardized, students used the AI tools of their own choice refining prompts and debugging code across platforms. This process reflects key connectivist principles: navigating multiple knowledge nodes, applying real-time feedback, and developing personal strategies for problem solving.

Recent research supports a meaningful connection between spreadsheet fluency and learning to code. Lovászová and Hvorecký (2004) describe how spreadsheets can be used to explore foundational algorithmic structures such as sequence, selection, and repetition without writing formal code in a programming language. As they note, "Programming problems can be also solved using spreadsheet calculations. The stage is built around one of the spreadsheet solutions. It is selected and arranged in a way that exhibits and amplifies the considered properties of the algorithm. By experimenting with the spreadsheet solution, the students reveal the properties and extrapolate them to the field of programming" (Lovászová & Hvorecký, 2004, p. 46).

Csernoch and Biró (2019) found that first-year students who approached solving spreadsheet tasks algorithmically demonstrated stronger problem solving and computational thinking skills than those who relied on more basic functions. From a connectivist perspective, these findings suggest that activating prior spreadsheet knowledge can help students build new conceptual connections when they learn to code.

Recent research also highlights the role of generative AI enabled coding environments as important learning nodes within a connectivist framework. Zvi-Girshin (2024) found that students quickly incorporated AI tools into their everyday coding habits, especially for tasks like debugging, adding comments, and looking up information, and became more comfortable with them as the semester went on. The study also noted that students often leaned on these tools too much, which sometimes diminished their learning of problem solving and coding skills.

Gardella et al. (2024), examined how novice programmers performed when using GitHub

Copilot to complete introductory programming tasks. Their findings showed that AI-assisted coding significantly increased programming efficiency and reduced mental workload, although gains in confidence emerged only with repeated practice. These results suggest that AI tools can support beginning coders by providing immediate feedback and reducing frustration, but their ability to develop foundational skills may be limited when students over-rely on them. Together, these studies align with connectivist principles by showing how learners can benefit from multiple technological and informational nodes as part of their learning process.

Figure 1 presents a conceptual model of the Excel / Python / AI progression framed by connectivist learning. Knowledge develops as students traverse technological, peer, and conceptual models while translating their fluency in translating spreadsheet logic to developing Python code and then leveraging AI tools for higher-order tasks. Excel knowledge provides familiar problem structures that students then implement in Python code, while AI supports developing code for more complex tasks such as data visualization and requires students to critically evaluate the output of generated code.

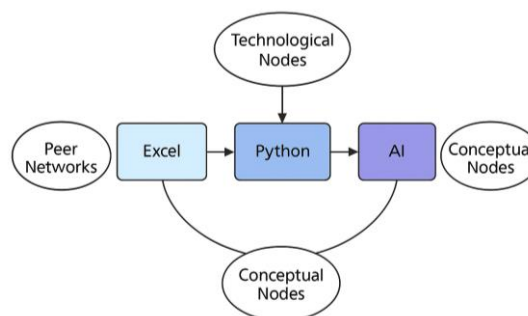


Figure 1. Conceptual model of the progression from Excel to Python to AI framed by connectivist learning.

The pedagogical transition explored in this research follows a clear evolution of digital literacy. As Prather et al. (2024) argue, in the AI era, students are not merely 'writing code'; they are performing high-level problem decomposition through 'Prompt Problems.' This aligns with the progression observed in this study, where students move from writing formulas in Excel to writing code in Python to writing prompts for AI.

The progression from Excel to Python to AI is deliberate, grounded in the idea that students learn coding more effectively when new concepts build on existing knowledge. Many first-year

students arrive to college with some familiarity using spreadsheets (McCarron & Frydenberg, 2023). Excel provides a familiar entry point for exploring basic computational thinking principles such as calculations, data types, logical conditions and problem solving.

By starting with spreadsheet-based coding tasks and then translating those same steps into simple Python programs, students experience coding as an extension of the patterns they have already learned. It also aligns with connectivist learning principles by treating Excel and Python as two different knowledge nodes that students can learn to connect.

The transition from spreadsheets to coding is supported by the shared logical thinking that both tools require. Sarkar et al. (2020) found significant correlations between proficiency at writing formulas and programming expertise, suggesting that writing Excel formulas facilitates the progression to writing more abstract code in traditional programming languages such as Python or Java.

Introducing generative AI after students gain experience writing and running Python code serves a different pedagogical purpose. Once students can recognize and run code, AI becomes a tool for higher-order thinking. In this way, AI becomes a third node in the learning network, encouraging students to apply their Excel vocabulary to create prompts, critique outcomes, and iterate to refine the results.

As studied by Denny et al. (2023), the integration of generative AI into the programming curriculum provides a modern scaffolding mechanism that can reduce the initial cognitive load for novices, bridging the gap between basic data manipulation and complex software development.

The flow from Excel to Python to AI also reflects a workflow students are likely to see in their future academic and professional careers. Analysts build models in spreadsheets, transition to code as tasks become more complex, and use AI tools to generate insights into data. Introducing this progression into the course supports students in their learning and prepares them for their future roles in the digital workplace.

3. IMPLEMENTATION: FROM EXCEL TO PYTHON TO AI

One of the biggest challenges in learning to code is understanding new syntax and abstract

concepts. "The learning of programming is difficult because it involves the simultaneous acquisition of three domains of knowledge: the syntax and semantics of a programming language, the notional machine (an abstract model of the execution process), and problem-solving strategies" (Robins et al., 2003, p. 138). This study describes a three-session module for teaching coding concepts using familiar examples based on students' prior knowledge of Microsoft Excel in CS 100 ("Solving Business Problems with information Technology"), an information systems fundamentals course required of all first-year students at a business-focused university in New England. Course topics include intermediate proficiency in Excel along with basic computing concepts (operating systems, organizing files and data, cybersecurity, the Internet and World Wide Web, security and privacy), and emerging technologies such as virtual reality and generative AI.

The course also includes a coding component which historically had three classes on web development. Students learned to create simple web pages by hand-coding basic HTML tags for headers, paragraphs, images, and links, along with simple formatting for fonts and colors. The original intent was that by writing HTML and viewing it in a browser, students would learn about coding. The Python curriculum described here replaces the HTML unit for those sections that adopted it as an alternative.

Comparing Excel and coding concepts is an effective way to scaffold student learning (Groner, 2023). "At its heart, connectivism is the thesis that knowledge is distributed across a network of connections, and therefore that learning consists of the ability to construct and traverse those networks" (Downes, 2010).

Students used the Google Colab environment for developing and submitting their Python programs; after creating their notebooks, they shared a link to it with their instructor who checked it for grading and completion. Figure 2 shows an example of a student's Google Colab notebook. Students add a hierarchy of headings to make their notebooks easy to navigate and are required to show the program output to facilitate grading. Text blocks describe code, and code blocks contain executable Python statements. Using an online environment eliminates the need to install an IDE and upload solutions to a course learning management system. After creating their notebooks, students share the links with their instructor by completing an online form.

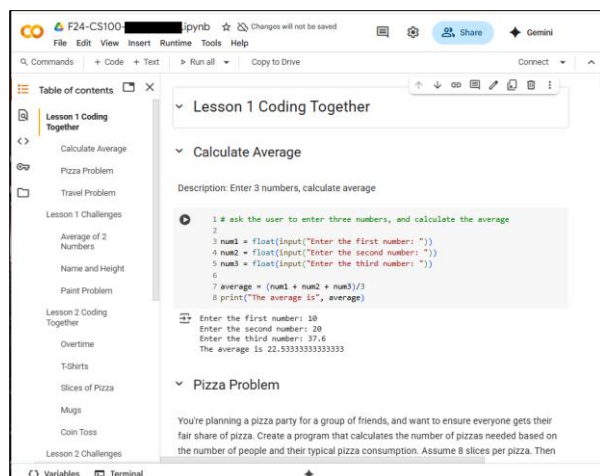


Figure 2. A Google Colab Notebook.

Google Colab can provide AI-generated code. Students were instructed to modify notebooks' settings to disable this feature for the first two lessons when they are learning coding fundamentals, and then to enable it for the third lesson when they will run and evaluate the results of AI-generated code to create charts and graphs.

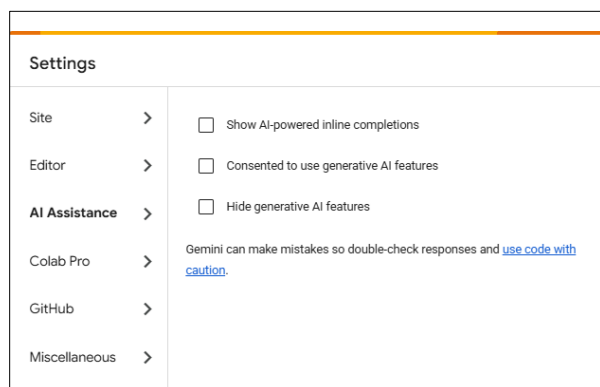


Figure 3. Disabling or Enabling AI Assistance in Google Colab.

Each day's lesson began with "coding together" problems where students solved a problem in small groups using Microsoft Excel. The instructor then showed them how to translate that solution into a Python program. See Figure 4 for a sample spreadsheet and coding solution for a problem of

calculating the number of pizzas required and cost of a pizza party. Note that the spreadsheet solution (at the left of Figure 4) uses a named range for cell B3 as an example of how named ranges can be more descriptive than cell names, encouraging the use of descriptive variable names.

Each Coding Together problem had a similar Coding Challenge for students to work on, usually with a partner, for homework. Students presented their coding challenge solutions at the start of the next class.

This structure reflects connectivist learning by applying its core ideas: students learn through peer collaboration, make use of familiar tools and learn to use new ones, and they engage with technology as an active partner in their learning. This approach moves beyond memorizing concepts and typing code without understanding to develop students as networked learners, capable of adapting and learning through their connections with tools, peers, and knowledge systems. Each day's lesson introduces a coding concept by relating it to a familiar Excel concept. The focus is on what coding can do, as much as it is on writing the code itself. On the third day, by using AI to generate code, students learn that AI can be an effective problem-solving tool, where the solution that AI provides is not the end, but a means to accomplishing a greater task (in this case, visualizing and interpreting data).

Students still need to run the AI-generated code and make sure that it produces the desired results (or modify their prompts or possibly the code) if the results are not as expected. Students also learn that when the AI-generated code does not work (and it often may not), they should try a different AI tool to see if it generates different solutions. The exercise becomes a lesson in prompt engineering, as students must apply the vocabulary of charts and graphs (chart types, legends, titles, axis labels, major and minor axis, etc.) to specify the desired appearance of their charts and then verify that the results are correct.

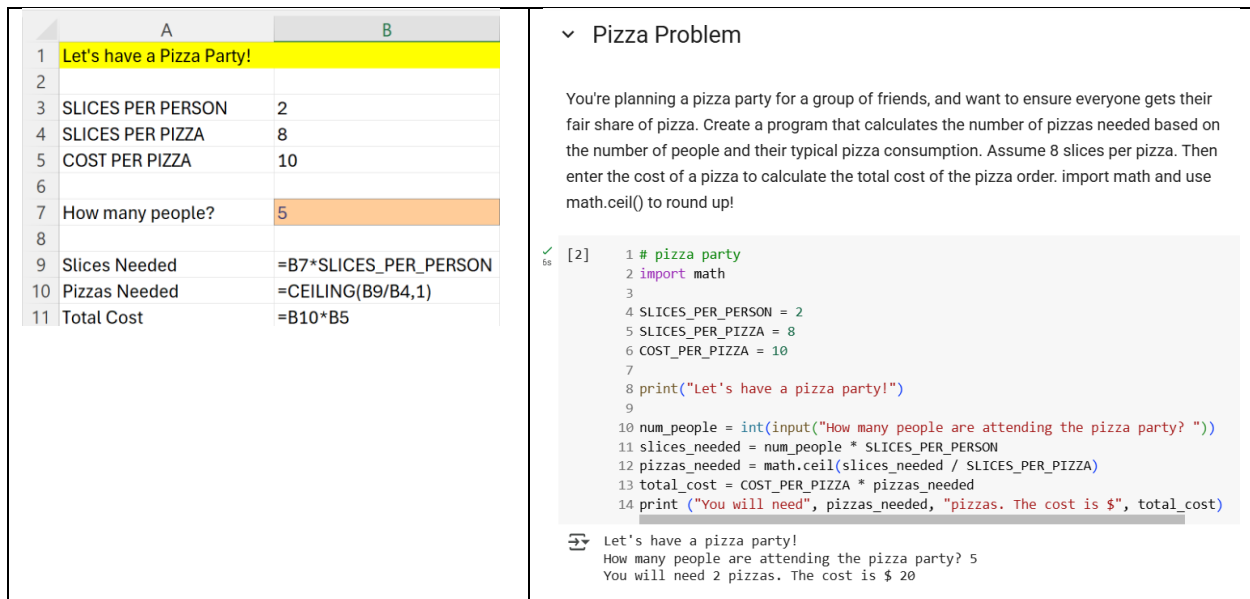


Figure 4. From Excel to Python (Spreadsheet and Python solutions)

Anecdotal, ChatGPT's code seemed more reliable than that generated within Colab for data/charts. Figure 5 shows part of a student's Colab notebook with AI-generated code and the resulting bar chart visualizing temperature data.

Table 1 summarizes the concepts of each lesson. Appendix B shows a summary of the Coding Together and Coding Challenge problems for each lesson.

Lesson	Python Concepts	Excel Concepts	Connections
1	variables, calculations, data types, input and print	Cells, formulas, data types, calculations, entering and displaying values	Calculations are the basic feature of spreadsheets and at the foundation of any algorithm. Topics include data types (int and float), simple math operations (round, ceiling, max, min, average) and their corresponding functions in Excel.
2	if statements, conditions for Loops	=IF(), =AND(), =OR() functions Copying values or formulas down a range of cells	Students are familiar with various forms of the if function in Excel, so writing if statements in Python is a natural extension. The process of copying values down a range of cells in a spreadsheet is similar to iterating over a list of items or values using a for loop.
3	using AI to generate Python code	Charts, graphs	Students are familiar with making pie, bar, line, and other charts in Excel. Coding these in Python by hand is difficult, but an AI tool can often generate the code if given a descriptive prompt.

Table 1. Excel to Python to AI Lesson Topics and Rationale



Figure 5. AI-Generated Code for Charting Weather data, shown in a Colab notebook.

4. METHODOLOGY

The author piloted the three-session module during the Spring 2024 semester with one section of 18 honors students. The following Fall, after making it available to other CS 100 instructors, one instructor adopted the lesson in place of HTML/Web Development in Fall 2024 for the two sections of approximately 35 students that he taught, and three instructors (five sections of approximately 35 students each) adopted the module for Spring 2025. All instructors who taught the Python module participated in two training sessions led by the author, who reviewed both the pedagogy and coding principles for these lessons. Students completed a short survey reflecting on their experience learning about coding after completing the last module.

Survey Data and Reliability

During these three semesters, 217 students (124 males, 92 females, 1 preferred not to say) completed the survey, usually during class time. Given maximum class sizes, at most 299 students would have been enrolled in sections teaching this curriculum, for a response rate of approximately 73%. The majority (60%) had not coded before; 30% had some Java or Python in high school, and the rest had varying degrees of coding experience (self-taught, knew HTML coded in Scratch, did "hour of code"). Most students (86%) were 18 or 19 years old, with 30 students 20 or older. When asked about their intended majors, there were

only two that mentioned Computer Information Systems, while the majority were split between Accounting, Finance, and Corporate Finance and Accounting.

The survey includes 16 questions based on a 5-point Likert scale (1 = strongly disagree, 5 = strongly agree), as shown in Appendix A. For clarity of results, the author organized the questions into broad categories:

- Learning Experience (6 questions)
- Engagement and Interest (4 questions)
- Perceived Career Value (4 questions)
- Prior or Recommended Exposure to Python (2 questions)

These categories were developed to track the pedagogical transition from established spreadsheet logic (Excel) to algorithmic thinking (Python), while incorporating the emerging role of AI tools for those with little or no coding experience.

The internal consistency of the survey instrument was evaluated using Cronbach's alpha (α). This step ensures that the questions within each category reliably measure the same underlying concept. Values were reverse-coded for the item *Learning Python was challenging* to ensure that a higher score (5) consistently represented a more positive perception of the learning process while a lower score (1) represented a perceived barrier.

The items *'Using AI tools to generate Python code was new to me'* and *Other sections of CS 100 should incorporate Python as we did* were excluded from the internal consistency (Cronbach's alpha) testing. While the other items measured perception (learning, engagement, and perceived value) these measured prior technological experience or future course recommendations. The summary of all responses is shown in Fig. 6. The results, shown in Table 2, indicate that all categories exceed the standard academic threshold of 0.70, demonstrating high reliability (Nunnally & Bernstein, 1994).

Category	# Items	Cronbach's α
Learning and Understanding	6	0.788
Engagement and Interest	4	0.883
Perceived Career Value	4	0.847

Table 2. Cronbach's α by Category

Survey Results

Because the Python curriculum was entirely similar across sections, and because all instructors were trained to follow the pedagogy for this module, the results that follow aggregate all results from all sections over three semesters.

The charts in Figure 6 summarize the averages of all responses for each question in each category.

Examining *Student Engagement and Interest*, students found the coding activities to be engaging (3.79) and fun (3.46). They had a moderate interest in learning more about Python independently (3.24), and lower-rated the possibility of taking a college course in Python (2.81). While they see the value in learning Python, fewer are motivated to take a course in it. This is consistent with the selection of majors that students are interested in pursuing, most of which do not require a coding course. One student commented, "I really enjoyed Python, and I think basic coding is an essential skill for extracurricular projects; I have already needed to code for a case competition."

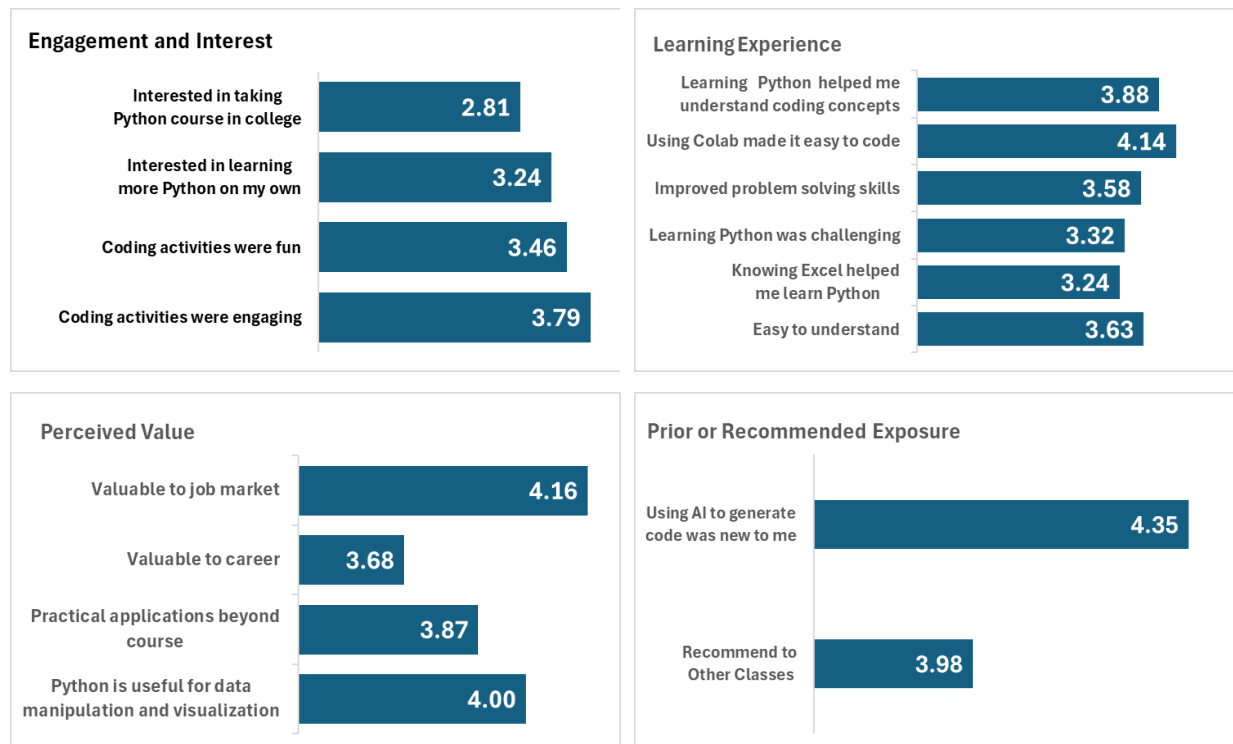


Figure 6. Survey Results

In the *Learning Experience* category, students overwhelmingly rated Google Colab as an easy-to-use tool for coding in Python (4.14). They felt that learning Python helped them understand programming concepts (3.88) and improve their problem-solving skills (3.58). While the examples were easy to understand (3.63), students were challenged to create their own code (3.32).

Said one student, “[writing code] definitely is something that you need to practice at and understand what you're doing, writing code is not always easy and you have to think about every action involved.” Another student commented that they could connect learning Python with previous coding experiences: “I had some prior experience with Java so it was nice to learn a new language and apply some of my prior knowledge.”

Unexpectedly, students rated “knowing Excel helped me to learn Python (3.24)” lowest in this category, suggesting mixed impact of prior Excel experience on learning about coding in Python. Upon sharing this finding with a CIS tutor who helped CS 100 students learning Python, the tutor reflected: “When I would help students in CS 100 with Python, I noticed there was a logic disconnect. Some students struggled with the multi-step process of solving a Python problem, whereas Excel tended to work in smaller, separate chunks.” While formulas would often translate similarly, when writing Python programs, students also needed to think about accepting user input, converting values to numeric data types, printing results, adding comments to code, and navigating Python syntax issues (indentation, colons, etc.) not encountered when creating spreadsheets.

Considering the *Perceived Value* of learning Python, students overwhelmingly felt it was a valuable skill to have in today's job market (4.16) and saw it as valuable to their career (3.68), with practical applications beyond the course (3.87). They found the value in using Python as found Python to be useful for manipulating and visualizing data (4.00).

Under *Prior or Recommended Exposure*, using AI tools to generate Python code was new to most students (4.35), and recommended that other sections should teach it (3.98)

Overall, the data suggests that students found the coding environment and Colab to be helpful; they were engaged with the topic (especially using AI); and they clearly valued the relevance

of knowing Python; but they prefer informal over formal instruction for future learning.

Sentiment Analysis

To better understand students' responses to the coding module, this study implemented a sentiment analysis using Python's TextBlob library (Loria, 2018). TextBlob evaluates sentiment along two dimensions: polarity (-1.0 for strongly negative to +1.0 for strongly positive), and subjectivity (0 for objective to 1.0 for highly personal) (Liu & Pang, 2025). The analysis shown here focuses on polarity, and classifies comments as:

- Positive (polarity > 0.2)
- Neutral (-0.2 <= polarity <= 0.2)
- Negative (polarity < -0.2)

These bounds are based on a commonly used heuristic (Loria, 2018). Although they provide a practical way to filter positive or negative sentiment, they are admittedly not perfect, especially for short comments, and the author acknowledges this as a limitation of this study.

Out of 78 responses, 47 (60%) were positive, 24 (31%) were neutral, and only 7 (9%) were negative. Neutral sentiments were mostly due to blank (“n/a” or “none”) responses. Figure 7 displays a word cloud colored by sentiment polarity. Green words indicate positive sentiment, and red words indicate negative sentiment (See Figure 7).



Figure 7. Sentiment Word Cloud (Generated with wordclouds.com).

Positive themes included student enjoyment when learning Python, recognition of its value for

Comment	Polarity	Sentiment
"Very interesting and enjoyable."	0.58	Positive
"I was never efficient with technological skills so my opinion is biased and against coding however this course was not terrible for someone like myself."	0.50	Positive
"I enjoyed learning about Python and felt that it was an easy concept to understand."	0.47	Positive
"It was a fun and engaging experience. I'm not much for coding, but it was nice to learn some basics nonetheless."	0.30	Positive
"I found the Excel integration a little challenging even though I already kind of know python. I also didn't completely understand its value."	0.25	Neutral
"It was fun but i don't see a ton of practical applications for it in my major."	0.18	Neutral
"At first I didn't understand it completely but by the time we had to do the project I felt ready and it was very simple to do because I knew what I was doing."	0.14	Neutral
"I did not enjoy using AI to code because it gave me error messages and I didn't know how to fix it. Everything else was fun."	0.05	Neutral
"The third python assignment was difficult. Adding the data into the platform didn't work."	-0.25	Negative
"Bad/boring."	-1.00	Negative

Table 3. Comments and Polarity Scores.

future careers or in other courses and appreciating the use of Google Colab and engaging projects. Selected examples are shown in Table 3 to provide additional context. While the results from the survey's Likert scale responses show that students valued the experience overall, this sentiment analysis confirms engagement and enjoyment, even among students who found coding difficult, and suggests their emotions (frustration, reward, curiosity) while completing the module.

6. CONCLUSIONS

This study demonstrates that a connectivist framework provides a powerful lens for understanding how first-year students can learn about basic coding concepts when supported by familiar tools, peers, and AI technologies. By intentionally following in the Excel /Python / AI sequence, students engage with multiple technologies and concepts and find relevant connections between them. Survey results show that while students drew some parallels between spreadsheets and programming, the transition from Excel's cell-based logic to Python's multi-step reasoning and algorithm development required more support and explanation than was originally anticipated.

Students confirmed strong engagement with digital tools and their peers and described positive emotions when learning about Python and using AI tools in a new way. Their responses, which

range from struggling with technology to succeeding in using it, reflect the iterative and exploratory nature of constructivist learning environments, where learners construct meaning by navigating multiple information sources and actively exchanging ideas with peers through collaborative problem solving, shared coding challenges, and comparing AI-generated results.

Many students said they would pursue coding informally rather than through additional course work; this self-directed interest also is aligned with connectivist principles. As digital technologies and AI tools continue to evolve, developing the ability to navigate, evaluate, and connect these tools will be critical to future success. This study shows that connectivist strategies can enhance student understanding and prepare them for the data-driven and AI-enabled tools and processes they will encounter in their future careers.

Limitations and Future Research

This study combines data from eight sections taught by five different instructors during a three-semester span. For all except the original instructor, none had taught this material before. Teaching Excel concepts knowing what the upcoming Python content would be, or having more familiarity with it, might also influence student perception of the value of knowing Excel prior to learning basic Python skills. Future studies could include a larger sample size, introducing this curriculum in a similar course at another university, or examining results based on

instructor or students' anticipated majors or minors to discover learning trends or biases.

Research Questions

Returning to the research questions:

RQ1. To what extent does prior experience with Excel support students' understanding of coding concepts?

While the curriculum aimed to build on students' prior experience with Excel, the impact of that prior knowledge on learning Python was less significant than expected. The survey results suggest this connection was moderate. Students gave lower ratings to the statement "Applying my Excel knowledge to Python made it easier..." (mean = 3.24), indicating mixed perceptions about the strength of that knowledge transfer. From a connectivist learning perspective, not all students were able to make those connections effectively without additional guidance. While the instructor designed the module to create the bridge between Excel and Python, it may require additional scaffolding to make those connections clearer to more students.

RQ2. How do digital tools and peer networks support student learning and engagement in an introductory information systems course, through a connectivist lens?

Web-based tools such as Google Colab assist with student learning, but surprisingly, prior Excel knowledge was less of a benefit in learning Python than expected. Students made connections between spreadsheet and Python concepts, but found the coding exercises to be more challenging, and using AI to generate complex Python code to be most interesting.

RQ3. How do students perceive the value of learning Python for their academic and professional goals?

Students' high ratings of Python's usefulness and relevance suggest that they clearly see the benefit of learning about coding for both academic projects and future employment.

RQ4. To what extent are students motivated to continue learning coding after the course?

Students recognized the value of learning Python, particularly in their continued studies and in their careers. Their interest in pursuing future course work, however, was less than expected. This aligns with connectivist principles, however, which stresses learning as a self-directed process.

One student noted "I really enjoyed Python, and I think basic coding is an essential skill for extracurricular projects; I have already needed to code for a case competition."

Discussion

These results collectively support the use of connectivist learning as a lens through which to evaluate this instructional approach. Students engaged meaningfully with tools such as Google Colab, Excel, and AI chatbots, while also learning through interactions and collaborative problem solving with their peers. Sharing solutions in class reinforced social learning and using technology tools enabled students to draw connections between familiar and new concepts. Overall, the findings suggest that coding education grounded in connectivist principles can result in greater engagement and deepen understanding in learning environments that thrive on technology, collaboration, and real-world applications.

While this study focused on the transition between spreadsheet logic, coding, and generative AI, emerging trends suggest a fundamental shift is taking place in programming pedagogy, one that prioritizes intention over syntax. The direct transition from Excel to AI has been accelerated by the emergence of vibe coding, a practice where "you fully give in to the vibes... and forget that the code even exists... It's not really coding – I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works" (Karpathy, 2025).

Recent shifts in computer science education suggest that the "vibe coding" approach, skipping the coding language entirely in place of natural language prompts that lead to an AI-generated result, shifts the focus from syntax to solution (Pradhan et al., 2025; Sarkar & Drosos, 2025). The world has changed, and vibe coding shifts the student's role from coder to code-reviewer or system architect, allowing especially non-technical users to build technology solutions while leaving the implementation details to an AI large language model. Future research could investigate the implications and possibilities of vibe coding on learning to code, computational thinking, and empowering students to leverage AI for social impact and ethical data-driven decision making.

7. REFERENCES

- Andone, D., & Frydenberg, M. (2021). Co-creating with TalkTech: Developing Attributes through International Digital Collaborative Projects. *2021 IEEE Global*

- Engineering Education Conference (EDUCON)*, 1073–1077. <https://doi.org/10.1109/EDUCON46332.2021.9454119>
- Bozan, I., & Taslidere, E. (2024). The Effect of Digital Game Design-Supported Coding Education on Gifted Students' Scratch Achievement and Self-Efficacy. *International Journal of Contemporary Educational Research*, 11(1), 20–28. <https://doi.org/10.52380/ijcer.2024.11.1.531>
- Csernoch, M., & Biró, P. (2019). *Are digital natives spreadsheet natives?* (arXiv:1909.00865). arXiv. <https://doi.org/10.48550/arXiv.1909.00865>
- Denny, P., Prather, J., Becker, B. A., Finnie-Ansley, J., Hellas, A., Leinonen, J., Luxton-Reilly, A., Reeves, B. N., Santos, E. A., & Sarsa, S. (2023). *Computing Education in the Era of Generative AI* (arXiv:2306.02608). arXiv. <https://doi.org/10.48550/arXiv.2306.02608>
- Downes, S. (2010). New Technology Supporting Informal Learning. *Journal of Emerging Technologies in Web Intelligence*, 2(1). <https://doi.org/10.4304/jetwi.2.1.27-33>
- Dziubaniuk, O., Ivanova-Gongne, M., & Nyholm, M. (2023). Learning and teaching sustainable business in the digital era: A connectivism theory approach. *International Journal of Educational Technology in Higher Education*, 20(1), 20. <https://doi.org/10.1186/s41239-023-00390-w>
- Gardella, N., Pettit, R., & Riggs, S. L. (2024). Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation. *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, 290–296. <https://doi.org/10.1145/3649217.3653615>
- Groner, D. (2023). *Python for Data Analytics: A Business Oriented Approach*. Prospect Press. <https://www.prospectpressvt.com/textbooks/groner-python>
- Karpathy, A. (2025, February 2). *There's a new kind of coding I call "vibe coding"* [Letter]. <https://x.com/karpathy/status/1886192184808149383>
- Kivunja, C. (2014). Do You Want Your Students to Be Job-Ready with 21st Century Skills? Change Pedagogies: A Pedagogical Paradigm Shift from Vygotskyian Social Constructivism to Critical Thinking, Problem Solving and Siemens' Digital Connectivism. *International Journal of Higher Education*, 3(3), 81–91. <http://dx.doi.org/10.5430/ijhe.v3n3p81>
- Liu, C., & Pang, S. (2025). *Empty Vessels Make the Most Noise: Analyst Self-Promotion Behavior and Market Outcomes* (SSRN Scholarly Paper No. 5292117). Social Science Research Network. <https://doi.org/10.2139/ssrn.5292117>
- Loria, S. (2018). *TextBlob: Simplified Text Processing—TextBlob 0.19.0 documentation*. <https://textblob.readthedocs.io/en/dev/>
- Lovászová, G., & Hvorecký, J. (2004). On Programming and Spreadsheet Calculations. *Spreadsheets in Education*, 1(1). <https://sie.scholasticahq.com/article/4510-on-programming-and-spreadsheet-calculations>, <https://sie.scholasticahq.com/article/4510-on-programming-and-spreadsheet-calculations>
- Matthee, M., & van Deventer, P. (2022). Preparing IS Programming Students for Work by Enhancing a Connectivist Teaching Approach by Scaffolding Practices. *Proceedings of the 2022 AIS SIGED International Conference on Information Systems Education and Research*. <https://aisel.aisnet.org/siged2022/11>
- McCarron, E., & Frydenberg, M. (2023). *Digitally Prepared for Success? Technology Skills of Incoming First-Year College Students*. 21(3), 70–90. <http://www.isedj.org/2023-21/n3/ISEDJv21n3p70.html>
- Nunnally, J. . C., & Bernstein, I. H. (1994). *Psychometric Theory* (3rd Edition). McGraw-Hill. <https://archive.org/details/dli.scoerat.1556psychometrictheorysecondedition>
- Parsons, J. (2023). Coding with Python. In *New Perspectives Concepts 2021 and Microsoft Office 2021* (21st ed.). Cengage.
- Pradhan, A., Henriques, J., Pan, S., & Engineer, R. (2025). EXPLORING VIBE CODING PRACTICES IN COMPUTER SCIENCE EDUCATION. *ICERI2025 Proceedings*, 5153–5162. 18th annual International Conference of Education, Research and Innovation. <https://doi.org/10.21125/iceri.2025.1451>
- Prather, J., Denny, P., Leinonen, J., Smith, D. H., Reeves, B. N., MacNeil, S., Becker, B. A., Luxton-Reilly, A., Amarouche, T., & Kimmel,

- B. (2024). *Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models* (Version 1). arXiv. <https://doi.org/10.48550/ARXIV.2401.10759>
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2), 137–172. <https://doi.org/10.1076/csed.13.2.137.14200>
- Sarkar, A., Borghouts, J. W., Iyer, A., Khullar, S., Canton, C., Hermans, F., Gordon, A. D., & Williams, J. (2020). Spreadsheet Use and Programming Experience: An Exploratory Survey. *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, CHI EA '20*, 1–9. <https://doi.org/10.1145/3334480.3382807>
- Sarkar, A., & Drosos, I. (2025). *Vibe coding: Programming through conversation with artificial intelligence* (arXiv:2506.23253). arXiv. <https://doi.org/10.48550/arXiv.2506.23253>
- Siemens, G. (2005). Connectivism: A Learning Theory for the Digital Age. *International Journal of Instructional Technology and Distance Learning*, 2(1), 3–10.
- Siemens, G., & Downes, S. (2011). *Connectivism and Connective Knowledge 2011*. <https://cck11.mooc.ca/>
- Zviel-Girshin, R. (2024). The Good and Bad of AI Tools in Novice Programming Education. *Education Sciences*, 14(10), 1089. <https://doi.org/10.3390/educsci14101089>

APPENDIX A

Survey Questions

Demographics

- Your age
- Gender
- Have you decided what you might major in? If so, what?
- Which section of CS 100 are you in?
- Prior to learning Python in CS 100, what was your coding experience?

Learning Experience*

- The Python programming concepts introduced were easy to understand
- Applying my Excel knowledge to Python concepts made it easier when learning to code in Python
- I found the activities we completed using Python to be challenging
- Using the Google Colab environment made it easy to code in Python
- Working on Python problems improved my problem-solving skills
- Learning to code in Python helped me understand fundamental programming principles

Engagement and Interest*

- I found the activities we completed using Python to be engaging
- I found the activities we completed using Python to be fun
- After completing the Python programming activities in CS 100, I am interested in learning more about Python on my own
- After completing the Python programming activities in CS 100, I am interested in taking a Python course in college

Perceived Career Value

- Python is a useful language for data manipulation and visualization
- Knowing some coding will be a valuable skill in my career
- Having some coding skills are valuable in today's job market
- I can see practical applications for Python beyond this course

Prior or Recommended Exposure

- Using AI tools to generate Python code was new to me
- I would recommend that other sections of CS 100 incorporate learning Python as we did

Open-Ended Feedback

- Other comments about your experience learning Python in CS 100?

***Likert Scale (1 = Strongly Disagree, 5 = Strongly Agree)**

APPENDIX B.

Sample Lessons and Activities

Lesson 1: Calculations and Data Types

Coding Together

- Write a program to calculate and print the sum and product of two numbers that the user enters.
- You're planning a pizza party for a group of friends and want to ensure everyone gets their fair share of pizza. Create a program that calculates the number of pizzas needed based on the number of people and their typical pizza consumption. Assume 8 slices per pizza. Then enter the cost of a pizza to calculate the total cost of the pizza order. `import math` and use `math.ceil()` to round up!
- You're going on a road trip and want to figure out the cost of travel based on user inputs. Write a program to ask the user for the total distance of the trip in miles, the average miles per gallon (MPG) of the vehicle you're driving, the expected price of gasoline per gallon, and the expected average speed of the trip in miles per hour (MPH). Your program should calculate the total amount of fuel needed for the trip, the total cost of fuel, and the estimated travel time. Finally, it should output the total cost of the road trip.

Challenges

- Write a program that takes three numbers as input from the user, calculates their average, and prints the result.
- Write a program to ask the user to enter their name and their height in inches. Print their name followed by their height in feet and inches. Hint: use the `//` operator for integer division and the `%` operator for remainder.
- You want to paint four walls in a rectangular room (don't worry about doors and windows!) Assume a gallon of paint covers 350 square feet, and a painter can paint 150 square feet in an hour. Ask the user to enter the length, height, and width of the room in feet. Also enter the painter's hourly rate, and the cost per gallon of paint. Calculate the number of gallons you'll need, the total square feet needed for coverage, the cost of paint, the cost of labor, and the total cost to paint the room.
Run your program for a 9 x 12 room with 8-foot ceilings, and a painter who makes \$50/hour. Paint costs \$25/gallon.

Lesson 2. If Statements and For Loops

Coding Together

- Enter an employee's name and the number of hours worked. Assume an hourly rate of \$15, with time and a half for all hours over 40.
- A store sells a case of 6 books at a time. Each book costs \$10. Write a program to calculate the cost of 6, 12, ..., 48 books.
- A user enters their t-shirt size (S/M/L) and the program converts that size to words (small, medium, large).
- Write a program to simulate flipping 100 coins, count the number of heads and tails. Use these statements to generate random numbers.

```
import random
number = random.randint(0,1) # 0 for heads, 1 for tails
```

Challenges

- A user enters an integer, the program determines whether the number is odd or even. Hint: If `number % 2` is 0, the number is even. Otherwise it's odd.
- Print a table of Fahrenheit to Celsius temperature conversions for C values between 0 and 100 in multiples of 10 (0, 10, 20, ... 100). Use the formula $F = (C * 9/5) + 32$ to convert from C to F.

- Write a program of your own choosing that shows your understanding of several concepts from today's lesson. Be sure to include a comment at the top of your program that describes what your program is trying to compute.

Lesson 3. Charts, Graphs, and Coding with AI

Coding Together

Add the file temps.csv to your Colab Notebook in the Files area.

Temps.csv

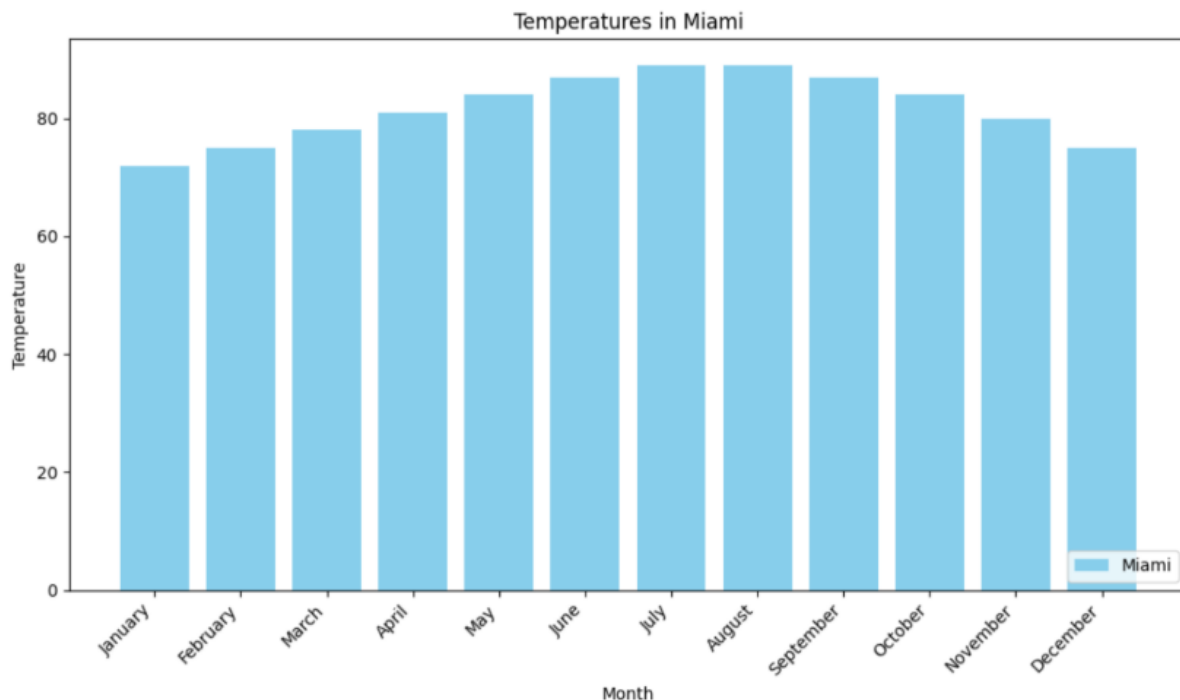
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	City	January	February	March	April	May	June	July	August	September	October	November	December
2	New York City	32	41	51	61	71	81	88	86	78	67	55	42
3	Los Angeles	57	59	62	65	69	73	77	79	77	73	67	59
4	Chicago	26	34	45	56	67	77	81	80	72	61	48	35
5	Houston	55	61	69	77	85	92	97	96	89	80	70	60
6	Miami	72	75	78	81	84	87	89	89	87	84	80	75

```
City, January, February, March, April, May, June, July, August, September, October, November, December
New York City, 32, 41, 51, 61, 71, 81, 88, 86, 78, 67, 55, 42
Los Angeles, 57, 59, 62, 65, 69, 73, 77, 79, 77, 73, 67, 59
Chicago, 26, 34, 45, 56, 67, 77, 81, 80, 72, 61, 48, 35
Houston, 55, 61, 69, 77, 85, 92, 97, 96, 89, 80, 70, 60
Miami, 72, 75, 78, 81, 84, 87, 89, 89, 87, 84, 80, 75
```

In Colab, create a code block, click "generate" to use AI, and enter this prompt:

Using temps.csv, write code to generate a bar chart showing temps from Miami for each month. use the month names as x axis labels, place a legend in lower right

You should see a chart that looks like this when you run the Python code:



Now, create a new text block and this time use the prompt:

Using temps.csv generate a bar chart showing average temperature for each city make each bar a different color

Run the code. If you get an error, it may be because Colab is assuming what the data looks like. In fact there is no "AvgTemperature" column in the data. Try to refine the prompt to give Colab more information about the data:

Challenges

Download the data file: sales.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Product	January	February	March	April	May	June	July	August	September	October	November	December
2	Laptops	245	112	331	315	451	573	561	812	553	884	590	965
3	Tablets	904	317	790	747	402	720	867	733	965	267	418	955
4	Phones	854	221	769	386	396	298	546	198	377	497	313	836

Product, January, February, March, April, May, June, July, August, September, October, November, December
Laptops, 245, 112, 331, 315, 451, 573, 561, 812, 553, 884, 590, 965
Tablets, 904, 317, 790, 747, 402, 720, 867, 733, 965, 267, 418, 955
Phones, 854, 221, 769, 386, 396, 298, 546, 198, 377, 497, 313, 836

Your challenge is to ask Colab (or ChatGPT, Gemini, CoPilot, or another tool) to create four charts that you design, that show many different features of charts that you learned about in Excel. Try making pie charts, bar charts, horizontal bar charts, or line charts. Use what you know about Charts in Excel to make your prompts as detailed as possible.

If the code that Colab generates doesn't work (as sometimes happened for me), you'll see some Python code with error messages. When that happens, try another tool. Upload the data file, and re-enter your prompt. If that tool doesn't run the code, copy and paste it back into a code block in Google Colab and run it there. See if the results are any better. You may need to edit the line of code `df = pd.read_csv('temps.csv')` to use the name of your data file if that line shows a different file name other than the one you are using.

Ask Colab, CoPilot, Gemini, or ChatGPT to generate code to display this data as a line chart, pie chart or bar chart. If the chart does not appear when you run the code in Colab, you may need to add a line such as `chart.show()` at the bottom of the code to instruct Python to display the chart.

How specific do your prompts have to be to get the charts you desire? Some AI tools work better than others with this data when generating code, so if you get error messages with one tool, try using a different one. What problems do you encounter?