The European Journal of Education
and Applied Psychology 2025, No 1–2
ISSN 2310-5704

PREMIER
Publishing
ppublishing.org

Section 1. Applied psychology

DOI:10.29013/EJEAP-25-1.2-3-12

FROM SPREADSHEETS TO PROGRAMMING: A PRACTICAL LEARNING APPROACH

Aysel Fataliyeva [1]

[1] The methodology of teaching Informatics, Azerbaijan State
Pedagogical University, Baku, Azerbaijan

Cite: *Fataliyeva A. (2025). From Spreadsheets to Programming: a Practical Learning Approach. European Journal of Education and Applied Psychology 2025, No 1–2.* https://doi.org/10.29013/EJEAP-25-1.2-3-12

Abstract

As digital literacy becomes increasingly essential, understanding programming concepts is crucial for students in various disciplines. However, many students perceive programming as complex and challenging. This paper explores an indirect teaching approach using spreadsheet applications to introduce programming concepts in higher education. By leveraging familiar spreadsheet tools, students can develop computational thinking, logical reasoning, and problem-solving skills without directly engaging with complex programming syntax. This approach lowers the barrier to programming education while making learning more engaging and applicable to real-world scenarios. The study specifically examines how this method can be effectively implemented in Azerbaijani universities to enhance students' digital and analytical competencies.

Keywords: *Spreadsheet-Based Learning, Computational, Thinking, Programming Education, Higher Education in Azerbaijan, Digital Literacy*

1. Introduction

In today's rapidly evolving digital landscape, programming skills are becoming an essential competency in various fields, from business and finance to engineering and science The ability to understand and manipulate digital tools through programming enhances problem-solving capabilities and fosters a deeper understanding of technological systems. As a result, many universities worldwide, including those in Azerbaijan, have incorporated programming courses into their curricula to improve students' digital literacy and computational thinking (Kraus, S., Durst, S., Ferreira, J.J., Veiga, P., Kailer, N., & Weinmann, A., 2022; Ghavifekr, S. & Rosdy, W.A.W., 2015; Thelma, C.C., Sain, Z.H., Shogbesan, Y.O., Phiri, E.V., & Akpan, W.M., 2024).

Despite these efforts, traditional programming education presents several challenges. Many students find programming concepts difficult to grasp due to their abstract nature and the need to learn complex syntax and structures. Additionally, programming is often perceived as a skill reserved for computer science students, discouraging individuals from non-technical backgrounds from engaging with it. This has led educators and researchers to explore alternative methods of teaching programming in a way that is accessible, engaging, and applicable to a broader audience. (Medeiros, R. P., Ramalho, G. L., & Falcão, T. P., 2018; Uysal, M. P., 2014; Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., & Szabo, C., 2018; Xinogalos, S., 2012; Toti, G., Lindner, P., Gao, A., Baghban Karimi, O., Engineer, R., Hur, J., & Wicentowski, R., 2025; Sithole, A., Chiyaka, E. T., McCarthy, P., Mupinga, D. M., Bucklein, B. K., & Kibirige, J., 2017).

One promising approach involves integrating programming principles into commonly used software applications, such as spreadsheets. Spreadsheets are widely utilized across various domains for data analysis, financial modeling, and administrative tasks. They provide an intuitive interface that enables users to interact with structured data, apply logical functions, and automate repetitive tasks. By leveraging these capabilities, educators can introduce fundamental programming concepts without requiring students to write traditional code (Argent, R. M., 2004; Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. 2023).

The advantage of using spreadsheets as a learning tool is their familiarity and ease of use. Most students and professionals have encountered spreadsheet software like Microsoft Excel or Google Sheets, making the transition from basic operations to more complex functions relatively smooth. Within spreadsheet environments, users can engage with essential programming concepts such as data types, conditional logic, iteration, and automation through built-in formulas, macros, and scripting tools. This indirect approach to programming education lowers the barrier to entry while still reinforcing key computational thinking skills (Mulle, R., 2023).

Furthermore, teaching programming through spreadsheets aligns with real-world applications, as many industries rely on spreadsheet software for critical business and operational tasks. By integrating computational principles into spreadsheet-based learning, students not only develop problem-solving abilities but also gain skills that are directly applicable to their future careers. This approach also encourages interdisciplinary learning, enabling students from diverse academic backgrounds to harness programming concepts within their respective fields. Unlike spreadsheets, coding in a programming language allows students to work with data structures, loops, and custom functions in a more flexible and scalable manner (Suthar, K. J., Mehta, A., Panda, S. R., Panchal, H., & Sinha, R., 2024; Kennedy, D. B., & Stratopoulos, T. C., 2024; Suthar, K. J., Mehta, A., Panda, S. R., Panchal, H., & Sinha, R., 2024).

2. Teaching Approaches with Spreadsheets

Spreadsheets as a Learning Tool

Spreadsheets are widely used for data analysis, data visualization, financial modeling, and problem-solving across multiple disciplines. They serve as a bridge between basic digital literacy and advanced computational thinking by allowing users to manipulate data in structured ways. Unlike traditional programming environments, spreadsheets provide an intuitive, interactive interface that does not require students to learn complex syntax from the outset (Tedre, M., & Denning, P. J., 2016).

One of the key advantages of using spreadsheets as a learning tool is their accessibility and familiarity. Most students have encountered spreadsheet applications like Microsoft Excel or Google Sheets, making them an excellent entry point for exploring computational concepts. Through working with spreadsheets, students can develop an understanding of fundamental programming ideas such as variables, functions, logic, loops, and data manipulation without the immediate need to write code in a traditional programming language (Nardi, B. A., & Miller, J. R., 1991; Lohani, S. K., 2023).

When students work with formulas, they are essentially engaging with algorithmic problem-solving, identifying patterns, and structuring data processing workflows (Murray, A., 2022).

Another significant benefit of spreadsheets as a learning tool is their application-driven nature. Unlike abstract programming exercises, spreadsheet-based learning is often tied to real-world problems. Students can use spreadsheets to analyze business scenarios, conduct scientific data evaluations, or optimize financial records. This makes learning more relevant and engaging, as students see the immediate impact of their work and understand the practical importance of computational thinking (Evans, J.R., 2000). Although spreadsheets help introduce basic programming logic, they do not provide sufficient exposure to object-oriented programming (OOP), memory management, or modular software design. As a result, students may struggle when transitioning to traditional programming languages.

The shift from merely using spreadsheets for data entry to actively leveraging them for problem-solving fosters deeper comprehension of programming logic. When students learn to design their own spreadsheet models, automate calculations, and create dynamic data visualizations, they are, in effect, learning to think like programmers. Encouraging students to explore these capabilities in a structured way enhances their analytical skills and prepares them for more advanced computational learning (Törley, G., Zsakó, L., & Bernát, P., 2022).

Thus, spreadsheets offer a unique and powerful approach to introducing programming concepts in a low-barrier, high-impact manner. By focusing on logical reasoning, problem decomposition, and structured data manipulation, spreadsheets help bridge the gap between basic computer literacy and formal programming education. Traditional programming languages provide a more structured learning experience by requiring students to understand syntax, debugging techniques, and algorithmic complexity (Csernoch, M., Biró, P., & Máth, J., 2021).

Understanding the Process – Learn Programming!

By treating spreadsheets as programs and exploring their internal logic, students develop problem-solving skills applicable to broader computational tasks.

When students use spreadsheets, they engage in structuring and manipulating data, applying formulas, and using logical statements, all of which align with programming fundamentals. This process enables them to develop an intuitive grasp of computational thinking without the need for direct coding.

Furthermore, spreadsheets help students understand core programming concepts such as variables, data types, conditional statements, and iterative processes. For example, using functions like IF, COUNTIF(), and SUMIF() introduces them to conditional logic, which is an essential concept in programming. Additionally, automating calculations through formulas and macros can be compared to writing simple scripts, reinforcing procedural thinking (Törley, G., Zsakó, L., & Bernát, P., 2022).

Another crucial benefit of using spreadsheets as a programming learning tool is debugging. Just like in traditional programming, spreadsheet users often encounter errors when working with complex formulas. Debugging these formulas requires a methodical approach – identifying the mistake, tracing the logic step by step, and making necessary corrections. This mirrors the problem-solving skills required in programming and provides an accessible way for students to develop resilience in tackling computational challenges.

By gradually shifting from spreadsheets to programming languages, students can build confidence in their problem-solving abilities while developing essential coding skills for their future careers.

They begin to appreciate how computational processes work and how they can be optimized, further strengthening their programming mindset.

Spreadsheets as a Program

Spreadsheets function as interactive programming environments where users can define input values, apply formulas, and automate repetitive tasks. Each cell in a spreadsheet can be seen as a variable, where data is stored and manipulated dynamically. By learning how spreadsheets manage data, students are introduced to the fundamental concepts of variables, expressions, and functions – critical components in any programming language.

These functions allow users to apply conditional statements, similar to how programmers use if-else statements in coding languages. Understanding these logical operations within spreadsheets enables students to develop structured problem-solving skills (Heimlich, S., 2019).

Moreover, spreadsheets include iterative processes, such as drag-down autofill and array formulas, which parallel loops in programming. By utilizing these features, students gain hands-on experience with repetition, automation, and efficiency – important principles in programming and algorithm design. More advanced spreadsheet functions, such as macros and VBA (Visual Basic for Applications), further strengthen the connection between spreadsheets and traditional programming paradigms.

Another crucial aspect of spreadsheets as programs is error handling and debugging. When working with spreadsheets, users frequently encounter issues such as formula errors, incorrect references, or unexpected outputs. Learning to troubleshoot these problems fosters a debugging mindset, a key skill for programming. By examining error messages, tracing dependencies, and modifying formulas, students develop resilience and analytical thinking, which are transferable to coding in other environments.

By integrating these concepts, spreadsheets serve as an ideal gateway for students to grasp computational thinking without the steep learning curve associated with syntax-heavy programming languages. This approach makes programming education more accessible, engaging, and applicable to real-world problem-solving scenarios.

3. Teaching Programming Concepts through Spreadsheets, Python, and Wolfram Alpha: 5 Examples

Example 1: Conditional Logic in Spreadsheets

Scenario: A student is analyzing sales data for a retail store in Azerbaijan to determine which items are eligible for a discount based on sales performance.

Table 1. *Conditional Logic in Spreadsheets*

Product (Məhsul)	Sales (Satış)	Discount Decision (Endirim Haqqında Qərar)
Shirt (Köynək)	150	=IF(B2>=100, “Yes (Bəli)”, “No (Xeyr)”)
Pants (Şalvar)	80	=IF(B3>=100, “Yes (Bəli)”, “No (Xeyr)”)
Jacket (Jaket)	200	=IF(B4>=100, “Yes (Bəli)”, “No (Xeyr)”)

Explanation:

The student uses the `IF` function in spreadsheets to apply conditional logic. The formula checks whether the sales of an item are greater than or equal to 100. If true, a discount (‘Bəli’) is applied; otherwise, it is not (‘Xeyr’). This mirrors if-else statements in programming, introducing students to conditional logic that can be used in programming languages.

Example 2: Using Loops in Spreadsheets

Scenario: A student calculates the total cost of items based on their quantity and unit price for an inventory system used in a business in Azerbaijan.

Table 2. *Using Loops in Spreadsheets*

Product (Məhsul)	Quantity (Sayı)	Price (Qiymət)	Total Price (Ümumi Qiymət)
Shirt (Köynək)	10	20	=B2*C2
Pants (Şalvar)	5	40	=B3*C3
Jacket (Jaket)	7	60	=B4*C4

Explanation:

In this example, the student applies the same formula across multiple rows to calculate the total price of each item (quantity × unit price). By dragging the formula down the column, the student experiences iteration, similar to a loop in programming. This teaches students how to apply the logic of repetition, a key concept in programming.

Example 3: Using Functions for Data Manipulation

Scenario: A student calculates the average score of students in a university in Azerbaijan and categorizes their performance.

Table 3. *Using Functions for Data Manipulation*

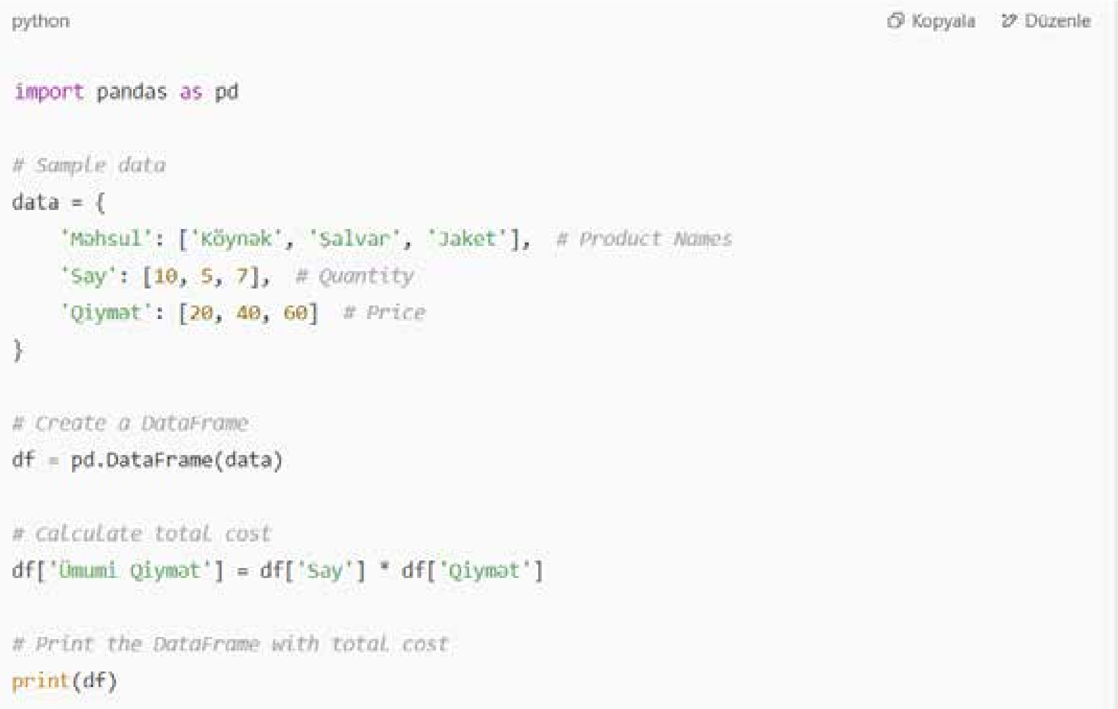
Student Name (Tələbə Adı)	Score (Bal)	Performance (Performans)
Nigar	95	=IF(B2>=90, "Excellent (Əla)", "Not Good (Yaxşı deyil)")
Tural	85	=IF(B3>=90, "Excellent (Əla)", "Not Good (Yaxşı deyil)")
Elvin	70	=IF(B4>=90, "Excellent (Əla)", "Not Good (Yaxşı deyil)")

Explanation:

In this example, the student uses the `IF()` function in a spreadsheet to categorize the performance of students based on their scores. The condition checks if a student's score is 90 or above, and if true, labels them as 'Əla' (Excellent). Otherwise, the result is 'Yaxşı deyil' (Not Good). This exercise introduces the concept of functions in programming, specifically logical functions, to automate decision-making processes based on conditions.

Example 4: Data Automation with Python

Scenario: A student is tasked with automating the calculation of monthly financial reports for a business in Azerbaijan using Python (https://www.online-python.com).



```python
import pandas as pd

# Sample data
data = {
    'Məhsul': ['Köynək', 'Şalvar', 'Jaket'],  # Product Names
    'Say': [10, 5, 7],  # Quantity
    'Qiymət': [20, 40, 60]  # Price
}

# Create a DataFrame
df = pd.DataFrame(data)

# Calculate total cost
df['Ümumi Qiymət'] = df['Say'] * df['Qiymət']

# Print the DataFrame with total cost
print(df)
```

Explanation:

Using Python, the student writes a script that automatically calculates the total cost for each item based on quantity and unit price. This task introduces the concept of loops and functions in programming. By using the `pandas` library, the student can handle data structures (Data Frames) and automate calculations, making it easier to manipulate large data sets in real-world scenarios. To ensure students gain real coding experience, spreadsheet-based learning should be supplemented with hands-on exercises in Python or Java. For example, students can first implement logic in a spreadsheet and then write equivalent Python scripts to reinforce programming concepts.

By gradually shifting from spreadsheets to programming languages, students can build confidence in their problem-solving abilities while developing essential coding skills for their future careers.

Example 5: Data Analysis Using Wolfram Alpha Scenario:

A student is asked to analyze the average growth of a business's monthly revenue over the past year using Wolfram Alpha (https://www.wolframalpha.com/)

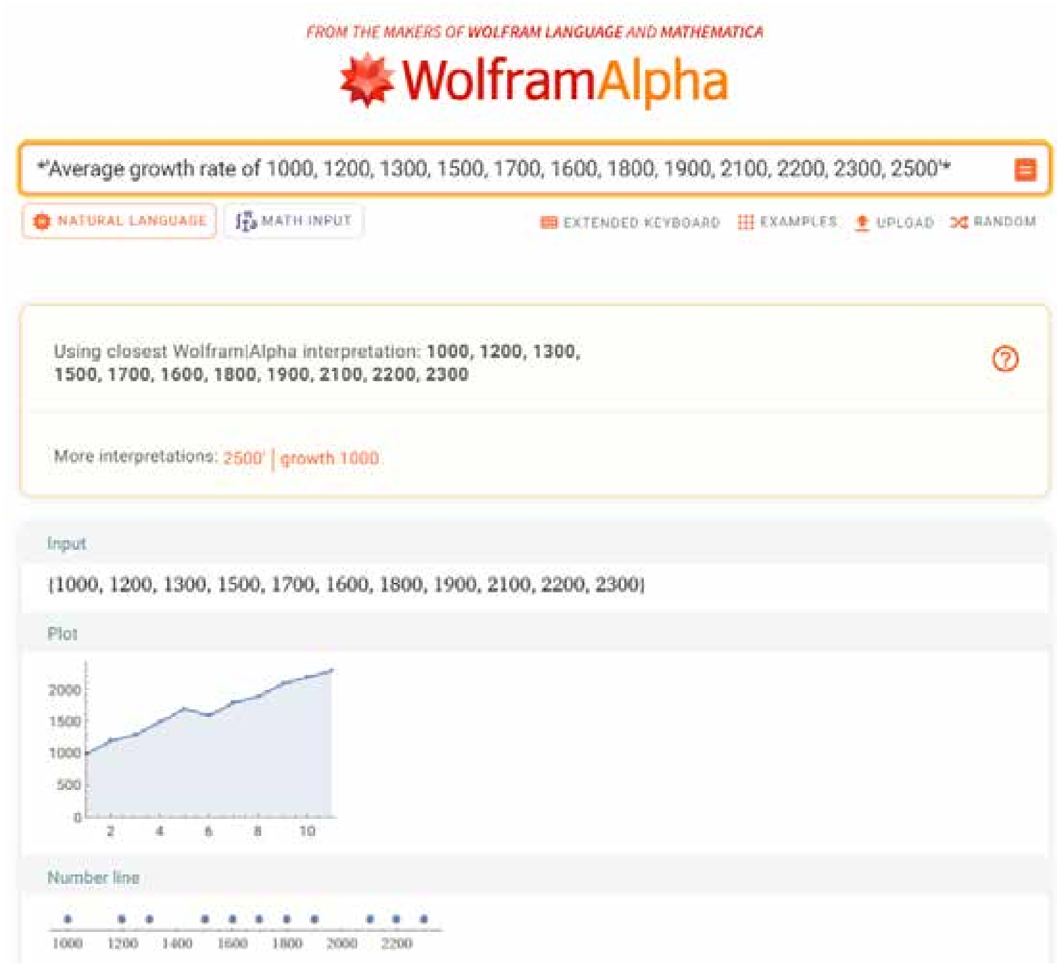
Data:

- Yanvar (January): 1000
- Fevral (February): 1200
- Mart (March): 1300
- Aprel (April): 1500
- May: 1700
- İyun (June): 1600
- İyul (July): 1800
- Avqust (August): 1900
- Sentyabr (September): 2100
- Oktyabr (October): 2200
- Noyabr (November): 2300
- Dekabr (December): 2500

Wolfram Alpha Query: The student enters the query: '1000, 1200, 1300, 1500, 1700, 1600, 1800, 1900, 2100, 2200, 2300, 2500' average growth rate (*Note: While entering data into Wolfram Alpha, it is important to pay attention to how the tool processes the input. Instead of entering the entire dataset directly, the student may request Wolfram Alpha to calculate the percentage growth between each month (for example, using the formula for growth between two points: (current value – previous value) / previous value * 100). This provides more detailed information on the growth rate each month).*

By following these steps, Wolfram Alpha would provide the student with a quick and accurate growth rate analysis. This analysis not only teaches students the fundamentals of data analysis but also enhances their problem-solving and computational thinking skills. These skills are applicable in both academic and real-world business scenarios.

Now, by paying attention to the images, we can observe how the data analysis results are processed and how Wolfram Alpha works.



"Average growth rate of 1000, 1200, 1300, 1500, 1700, 1600, 1800, 1900, 2100, 2200, 2300, 2500'"

Using closest Wolfram|Alpha interpretation: **1000, 1200, 1300, 1500, 1700, 1600, 1800, 1900, 2100, 2200, 2300**

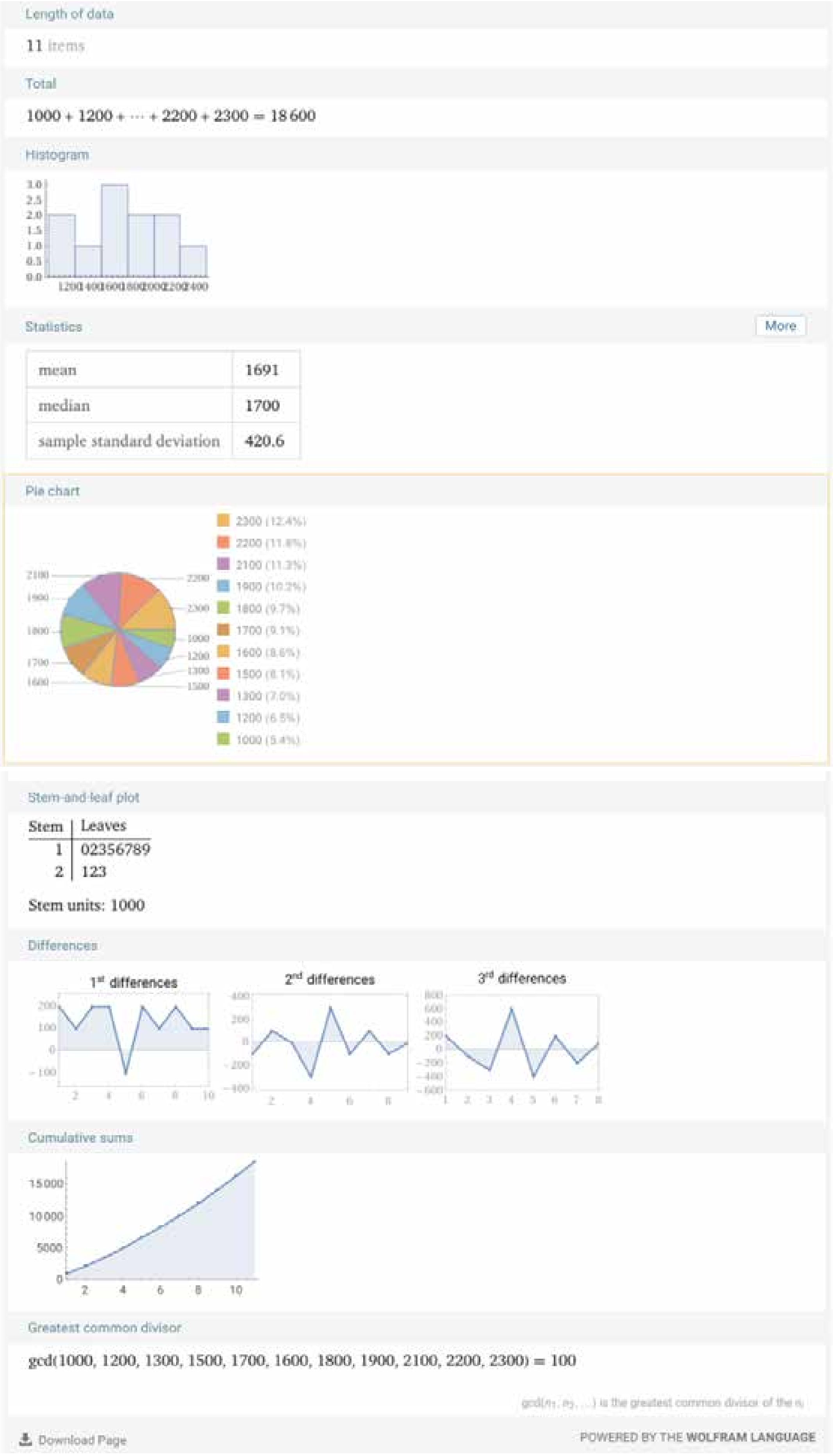
More interpretations: 2500' | growth 1000

Input

{1000, 1200, 1300, 1500, 1700, 1600, 1800, 1900, 2100, 2200, 2300}

Plot

Number line



Length of data

11 items

Total

1000 + 1200 + ⋯ + 2200 + 2300 = 18 600

Histogram

Statistics

mean	1691
median	1700
sample standard deviation	420.6

Pie chart

🟧	2300 (12.4%)
🟥	2200 (11.8%)
🟪	2100 (11.3%)
🟦	1900 (10.2%)
🟩	1800 (9.7%)
🟫	1700 (9.1%)
🟧	1600 (8.6%)
🟥	1500 (8.1%)
🟪	1300 (7.0%)
🟦	1200 (6.5%)
🟩	1000 (5.4%)

Stem-and-leaf plot

Stem	Leaves
1	02356789
2	123

Stem units: 1000

Differences

Cumulative sums

Greatest common divisor

gcd(1000, 1200, 1300, 1500, 1700, 1600, 1800, 1900, 2100, 2200, 2300) = 100

gcd(n_1, n_2, …) is the greatest common divisor of the n_i

Wolfram Alpha quickly computes the average growth rate across the months, helping the student understand how growth is calculated over a period of time. As seen in the images above, Wolfram Alpha provides immediate and detailed calculations for real-world data analysis. This allows students to input data points and derive meaningful insights without manually calculating complex formulas. By using Wolfram Alpha, students can see the practical application of concepts like **average growth rate**, which are crucial in fields such as data science and programming.

This introduces students to the process of analyzing and interpreting data trends. Just like in the examples above, Wolfram Alpha automatically interprets and analyzes the provided data, offering a breakdown of the results. The student can see how each monthly data point contributes to the overall trend and learn how to interpret these results more deeply, enhancing their data literacy and critical thinking skills.

In this case, Wolfram Alpha not only provides the growth rate but also helps students gain a more comprehensive understanding of how numerical data evolves over time. This is a key skill for students pursuing computational studies or those looking to integrate programming concepts into their daily work environments.

4. Benefits in Teaching Programming

Teaching applications as though they are programs lays a solid foundation for teaching programming. For example, every spreadsheet lesson can serve as an opportunity to teach programming through the representation of data and algorithms. Azerbaijani university students, particularly those in fields like business and engineering, can begin learning programming concepts by using such applications, especially as they frequently deal with numerical data in their respective disciplines. As students work with these applications, they are introduced to core programming concepts such as variables, conditional logic, functions, and loops.

Other applications, such as document and image editors, can also help illustrate different data structures and algorithms. For example, document editors involve organizing and processing textual data, helping students understand how data is stored, processed, and retrieved. Image editors, on the other hand, introduce concepts such as pixel manipulation, arrays, and matrix operations, which foster algorithmic thinking and problem-solving skills.

University students in Azerbaijan, particularly those in engineering and economics programs, can use such applications to develop skills in data analysis, financial modeling, and data processing that are essential in the professional world. For instance, when working with financial data in a business scenario, students can analyze data sets, make predictions, and use decision support systems. These activities not only improve their programming skills but also help them tackle real-world challenges.

When teaching applications as programs, data is explored through binary representation, and functions and methods are analyzed as algorithms. This introduces students to the concept of abstraction, as they don't need to understand how data is stored in memory but can focus on how data is manipulated through functions. Furthermore, applications allow students to observe how simple sorting algorithms work, as well as more complex image processing techniques.

This approach ensures that students encounter fundamental programming concepts before they need to use data and algorithms to create their own programs. By using everyday applications as teaching tools, students can establish connections between abstract programming concepts and practical real-world applications. Azerbaijani university students, for example, gain experience with these tools in programming classes, boosting their software development skills. This not only strengthens their computational thinking but also builds their confidence in using programming languages to develop their own software solutions.

Conclusion

To overcome this limitation, educators should implement a structured transition plan that moves students from spreadsheet-based logic to coding exercises in Python or Java.

While spreadsheets can teach fundamental programming concepts, they are not ideal for developing full-scale applications.

Advanced topics such as recursion, multithreading, and algorithm optimization require a programming environment beyond spreadsheets.

The approach presented in this paper significantly contributes to the education of university students in Azerbaijan by combining digital literacy and programming skills. Using applications such as spreadsheets to learn programming concepts enables students to both use technology more efficiently and understand fundamental programming principles. Traditionally, spreadsheet applications focus on data analysis and problem-solving, but this method encourages students to view these applications as programs, allowing them to develop a deeper understanding of algorithms written by developers.

This approach not only helps students acquire basic programming skills through tools like spreadsheets but also fosters computational thinking. For university students in Azerbaijan, this method provides an accessible and motivating entry point into programming. Moreover, students can apply the programming skills they learn to solve real-world problems in their respective fields, such as data processing and problem-solving. In conclusion, this teaching method makes programming education more accessible and equips students with valuable skills to use technology effectively in their future careers.

References

Kraus, S., Durst, S., Ferreira, J. J., Veiga, P., Kailer, N., & Weinmann, A. (2022). Digital transformation in business and management research: An overview of the current status quo. *International journal of information management*, – 63. – P. 102466.

Ghavifekr, S. & Rosdy, W. A. W. (2015). Teaching and learning with technology: Effectiveness of ICT integration in schools. International Journal of Research in Education and Science (IJRES), – 1(2). – P. 175–191.

Thelma, C. C., Sain, Z. H., Shogbesan, Y. O., Phiri, E. V., & Akpan, W. M. (2024). Digital Literacy in Education: Preparing Students for the Future Workforce. *International Journal of Research (IJIR)*, – 11(8). – P. 327–344.

Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2018). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, – 62(2). – P. 77–90.

Uysal, M. P. (2014). Improving first computer programming experiences: The case of adapting a web-supported and well-structured problem-solving method to a traditional course. *Contemporary Educational Technology*, – *5*(3). – P. 198–217.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., & Szabo, C. (2018, July). Introductory programming: a systematic literature review. In *Proceedings companion of the 23rd annual ACM conference on innovation and technology in computer science education* (P. 55–106).

Xinogalos, S. (2012). An evaluation of knowledge transfer from microworld programming to conventional programming. *Journal of Educational Computing Research*, – 47(3). – P. 251–277.

Toti, G., Lindner, P., Gao, A., Baghban Karimi, O., Engineer, R., Hur, J., & Wicentowski, R. (2025). Diversity, Equity, and Inclusion in Computing Science: Culture is the Key, Curriculum Contributes. In *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (P. 175–225).

Sithole, A., Chiyaka, E. T., McCarthy, P., Mupinga, D. M., Bucklein, B. K., & Kibirige, J. (2017). Student attraction, persistence and retention in STEM programs: Successes and continuing challenges. *Higher Education Studies*, – 7(1). – P. 46–59.

Argent, R. M. (2004). An overview of model integration for environmental applications – components, frameworks and semantics. *Environmental Modelling & Software*, – 19(3). – P. 219–234.

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023, March). Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education* – V. 1 (P. 500–506).

Mulle, R. (2023). Spreadsheets Application in Teaching Data Management in Mathematics of the Modern World: Effects on Students' Performance. *Sprin Journal of Arts, Humanities and Social Sciences*, – 2(06). – P. 11–18.

Suthar, K. J., Mehta, A., Panda, S. R., Panchal, H., & Sinha, R. (2024). Practical exercises of computer-aided process synthesis for chemical engineering undergraduates. *Education for Chemical Engineers*, – 48. – P. 31–43.

Kennedy, D. B., & Stratopoulos, T. C. (2024). How to Implement a Data Analytics and Emerging Technologies-Enabled Accounting Curriculum. *Journal of Emerging Technologies in Accounting*, – P. 1–19.

Suthar, K. J., Mehta, A., Panda, S. R., Panchal, H., & Sinha, R. (2024). Practical exercises of computer-aided process synthesis for chemical engineering undergraduates. *Education for Chemical Engineers*, – 48. – P. 31–43.

Tedre, M., & Denning, P. J. (2016, November). The long quest for computational thinking. In *Proceedings of the 16th Koli Calling international conference on computing education research* (P. 120–129).

Nardi, B. A., & Miller, J. R. (1991). Twinkling lights and nested loops: distributed problem solving and spreadsheet development. *International Journal of Man-Machine Studies*, – 34(2). – P. 161–184.

Lohani, S. K. (2023). *Excel for Finance and Accounting: Learn how to optimize Excel formulas and functions for financial analysis (English Edition)*. BPB Publications.

Murray, A. (2022). Advanced Lookup Functions. In *Advanced Excel Formulas: Unleashing Brilliance with Excel Formulas* (P. 541–656). Berkeley, CA: Apress.

Evans, J. R. (2000). Spreadsheets as a tool for teaching simulation. *Informs transactions on education*, – 1(1). – P. 27–37.

Heimlich, S. (2019). *Applying automatic program verification techniques to spreadsheets* (Doctoral dissertation, Macquarie University).

Törley, G., Zsakó, L., & Bernát, P. (2022). Didactic Connection between Spreadsheet and Teaching Programming. *Athens Journal of Technology & Engineering*, – 9(2). – P. 77–94.

Csernoch, M., Biró, P., & Máth, J. (2021). Developing computational thinking skills with algorithm-driven spreadsheeting. *IEEE Access*, – 9. – P. 153943–153959.

URL: https://www.online-python.com

URL: https://www.wolframalpha.com

submitted 14.02.2025;
accepted for publication 28.02.2025;
published 30.07.2025

Contact: marius-85@mail.ru