© Rzeszów University Press "Journal of Education, Technology and Computer Science"
ISSN 2719-6550 No. 6(36)/2025
e-ISSN 2719-7417 https://journals.ur.edu.pl/jetacomps

Received: 12.04.2025 DOI: 10.15584/jetacomps.2025.6.10
Accepted for printing: 24.11.2025 Scientific
Published: 31.12.2025

BORIS LIČINA[1], **LUKA LIČINA**[2], **SLAVOLJUB HILČENKO**[3], **SANJA NIKOLIĆ**[4]

Enhancing IT Competencies Through Python-Based Computational Thinking Modules for Teachers

[1] ORCID: 0000-0002-5090-1590, Ph.D., College for Vocational Education of Preschool Teachers and Coaches, Subotica, Serbia; email: boris.licina007@gmail.com
[2] ORCID: 0009-0002-0771-3604, B.Sc., University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia; email: licina.luka@gmail.com
[3] ORCID: 0000-0003-2123-6285, Ph.D., College for Vocational Education of Preschool Teachers and Coaches, Subotica, Serbia; email: s.hilcenko@gmail.com
[4] ORCID: 0000-0001-9632-2458, Ph.D., College for Vocational Education of Preschool Teachers and Coaches, Subotica, Serbia; email: drsanjanikolic294@gmail.com

Abstract

This study addresses the growing need for IT competency among educators in higher education by introducing Python-based computational thinking modules. Computational thinking – involving decomposition, pattern recognition, abstraction, and algorithm design – is a critical skill for navigating the digital era. Python, known for its simplicity and versatility, is an ideal language for enhancing these skills. This paper details the design of professional development programs tailored for higher education faculty, focusing on curriculum structures, practical exercises, and case studies of successful implementations. The findings demonstrate how such programs can empower educators to integrate computational thinking into their pedagogy, fostering a culture of innovation and technological fluency in academic environments. Furthermore, this study explores the role of artificial intelligence (AI) in enhancing Python's applicability, highlighting its potential to revolutionize education through tools like machine learning frameworks and intelligent tutoring systems.

Keywords: Computational Thinking, Python Programming, Higher Education, Teacher Training, Professional Development

Introduction

The integration of computational thinking into higher education is essential for preparing both educators and students for the demands of the modern digital world. Computational thinking, as defined by Wing (2006), is a problem-solving

methodology that involves decomposition, pattern recognition, abstraction, and algorithmic thinking. These skills are foundational for navigating not only computer science but also a broad range of academic disciplines and real-world challenges.

Python, a high-level programming language known for its readability and extensive library support, has gained popularity as a teaching tool in higher education. Its versatility makes it an effective medium for introducing computational thinking concepts to educators, allowing them to bridge theoretical knowledge with practical applications. Faculty equipped with Python skills can design innovative teaching strategies, create interdisciplinary connections, and improve student engagement in various courses.

Higher education institutions play a pivotal role in equipping faculty with these competencies. Educators require not only technical proficiency in Python but also pedagogical strategies tailored to adult learners. Professional development programs tailored for faculty members are crucial to achieving this dual goal. Such programs should focus on integrating computational thinking into existing curricula, fostering collaborative learning environments, and supporting continuous professional growth.

This paper outlines the framework for Python-based computational thinking training tailored to higher education faculty. It explores curriculum design principles, practical exercises to enhance IT competence, and case studies demonstrating the transformative impact of these programs. By empowering educators, higher education institutions can advance their mission of fostering innovation and preparing students for a technology-driven future.

Curriculum Design for Teacher Training in Computational Thinking

Learning Objectives:

1. Understanding Computational Thinking: Faculty members are introduced to core components of computational thinking, such as decomposition, pattern recognition, abstraction, and algorithm design. These skills are essential for structuring complex problems into manageable parts (Wing, 2006; Hilčenko, 2023).
2. Solving Real-World Problems: Educators learn to apply Python in modeling and solving problems relevant to their academic disciplines, such as data visualization or automating research workflows (Grover, Pea, 2013; Hilčenko, 2024).
3. Teaching Strategies: Training includes strategies for teaching computational thinking to university students, emphasizing interactive and student-centered approaches (Van Roy, 2009).

Module Structure:

1. Introductory Module: This module provides a foundational understanding of Python syntax, variables, and data types. These basics prepare faculty for more advanced topics and ensure all participants have a common starting point (Python.org., 2025).

2. Intermediate Module: Participants delve deeper into loops, conditionals, and functions. These skills are vital for developing structured and efficient code, which is critical for solving real-world problems (Code.org., 2025).

3. Advanced Module: Faculty work on real-world applications, such as automating repetitive tasks, analyzing datasets, and designing algorithms. This module bridges theoretical learning with practical application, fostering confidence in applying Python in diverse scenarios (Shute, Sun, Asbell-Clarke, 2017).

4. Pedagogical Module: This module equips educators with tools and strategies to integrate computational thinking into their teaching. Techniques such as project-based learning and flipped classrooms are explored to enhance student engagement (Sengupta, Kinnebrew, Basu, Biswas, Clark, 2013).

Resources and Tools:

1. Python IDEs: Tools such as Jupyter Notebook and Thonny are introduced for their user-friendly environments that support coding and visualization (Python.org., 2025).

2. Interactive Platforms: Platforms like Replit and Code.org provide opportunities for additional practice and collaboration, ensuring learning continues beyond structured sessions (Code.org., 2025).

3. Educational Libraries: Python libraries like Turtle Graphics for visualization and pandas for data analysis are integrated into the curriculum to demonstrate practical use cases (Wilson, Guzdial, 2013).

Practical Exercises to Boost IT Competence

Interactive Activities:

1. Programming Puzzles: These exercises are designed to reinforce foundational concepts like loops and conditionals. They encourage hands-on problem solving, helping participants bridge theory and practice (Grover, Pea, 2013).

2. Turtle Graphics Projects: Faculty use Python's Turtle library to create visual representations, such as geometric patterns. These activities foster creativity and logical reasoning, showcasing the versatility of Python in visual problem-solving (Wilson, Guzdial, 2013).

3. Data Manipulation Tasks: Using the pandas library, participants learn to clean and analyze simple datasets. These tasks provide a gateway to advanced data science concepts while remaining accessible to beginners (Resnick et al., 2009).

Collaborative Learning:

1. Pair Programming: Participants work in pairs to write and debug code, which promotes peer learning and fosters collaboration. This approach mirrors real-world programming practices and enhances problem-solving skills (Van Roy,

2009). Additionally, research highlights the importance of choosing beginner--friendly programming languages, such as Python, to ensure an inclusive and accessible learning experience. Python's simple syntax and wide applicability make it particularly effective for engaging educators in collaborative tasks (Viduka, Kraguljac, Ličina, 2021).

2. Group Projects: Teams of educators collaborate on developing small applications or simulations, such as grading systems or classroom management tools. These projects encourage interdisciplinary collaboration and practical application of skills (Lye, Koh, 2014).

Real-World Applications:

1. Automation: Faculty are trained to write Python scripts that automate repetitive tasks like grading assignments or managing datasets. This streamlines administrative workflows and demonstrates Python's practical utility (Shute, Sun, Asbell-Clarke, 2021). Additionally, the integration of Python with Raspberry Pi has proven particularly effective in small-scale production environments, enabling the automation of tasks such as product counting and packaging. Such applications highlight Python's capacity to combine affordability with powerful industrial solutions, as demonstrated in a case study of Raspberry Pi and Python optimizing workflows in manufacturing plants (Ličina, Viduka, Ilić, 2021).

2. Classroom Simulations: Python is used to model educational scenarios, such as simulating statistical experiments or visualizing complex mathematical concepts. These applications enhance teaching efficacy by making abstract concepts more tangible (Sengupta et al., 2013).

Case Studies of Successful Python Integration

Table 1 summarizes the key findings from three case studies that explore the integration of Python into professional development for higher education faculty. Each case study highlights different contexts and activities, showcasing the diversity and effectiveness of Python-based training programs:

Table 1. Three case studies that explore the integration of Python into professional development for higher education faculty

Case Study	Participants	Key Activities	Outcomes
Professional Development Workshops	50 faculty	Python scripting, administrative automation, Turtle Graphics	30% improvement in technical proficiency; increased confidence in pedagogy
Blended Learning Approach	100 faculty	Online lessons, live coding sessions, project design	20% increase in computational thinking integration; enhanced student engagement
Community-Driven Python Learning Circles	30 faculty	Workshops, collaborative projects, knowledge sharing	40% increase in interdisciplinary collaboration; repository of teaching materials

Case Study 1: Professional Development Workshops

A university conducted a series of workshops to introduce faculty members to Python and computational thinking. Each session combined theoretical discussions with practical activities, such as creating scripts to automate administrative workflows and developing visualizations using Turtle Graphics. Participants reported increased confidence in their ability to incorporate Python into teaching and research. Additionally, faculty who participated in the workshops reported a 30% improvement in their technical proficiency, enabling them to implement innovative teaching methods in their courses (Wilson, Guzdial, 2013).

These workshops emphasized hands-on learning, with participants engaging in activities such as scripting and administrative task automation using Python. The result was a significant improvement in technical skills and increased confidence among faculty in integrating computational thinking into their teaching.

Case Study 2: Blended Learning Approach in Teacher Training

An online professional development program was launched to train higher education faculty in computational thinking. The course featured asynchronous video lessons on Python fundamentals and live coding sessions for real-time problem-solving. Faculty were tasked with designing Python-based projects relevant to their disciplines, such as data visualization tools or interactive simulations. Post-course evaluations highlighted a marked improvement in participants' confidence and student engagement. Furthermore, educators reported a 20% increase in the integration of computational thinking activities into their curricula (Van Roy, 2009).

This approach combined asynchronous and synchronous learning to deliver Python training. Participants developed discipline-specific projects, such as data visualization tools, resulting in a marked increase in both student engagement and the adoption of computational thinking methodologies in classrooms.

Case Study 3: Community-Driven Python Learning Circles

In a peer-driven initiative, faculty members from diverse disciplines formed local learning circles to explore Python's applications in education. Monthly workshops included topics such as automating data collection, creating visual aids for lectures, and teaching computational concepts in non-technical fields. These meetings fostered a collaborative learning environment, allowing participants to share resources and best practices. Outcomes included the development of a shared repository of Python-based teaching materials, a 40% increase in interdisciplinary collaboration, and sustained engagement through quarterly hackathons (Resnick et al., 2009).

Faculty from various disciplines collaborated in local learning circles, sharing resources and developing Python-based teaching tools. This peer-driven initiative fostered interdisciplinary collaboration and the creation of a shared repository of educational materials.

Conclusion and recommendations

Python-based computational thinking modules provide an effective framework for enhancing IT competencies among higher education faculty. These programs equip educators with the skills needed to address modern educational challenges, fostering a generation of computationally literate students. By empowering educators with computational thinking skills, institutions can encourage innovation, critical thinking, and problem-solving in academic environments.

To ensure the success of such programs, institutions should:

1. Provide ongoing support through mentoring and resources to help faculty continuously refine their computational thinking skills.
2. Encourage collaboration among educators through communities of practice, promoting the sharing of knowledge, resources, and innovative teaching strategies.
3. Regularly update curriculum content to reflect technological advancements, ensuring that faculty stay ahead of emerging trends and applications.
4. Develop assessment metrics to evaluate the impact of computational thinking programs on teaching efficacy and student engagement.
5. Offer incentives for faculty participation in professional development programs, recognizing the importance of these efforts in advancing institutional goals.
6. Foster partnerships with industry and research organizations to create real-world applications and case studies for training modules.

The integration of artificial intelligence (AI) into Python programming introduces transformative opportunities for both educators and students. AI-powered tools and frameworks, such as TensorFlow and Scikit-learn, allow educators to explore machine learning concepts and their applications in problem--solving and data analysis. These advancements enable higher education faculty to incorporate cutting-edge AI techniques into their curricula, preparing students for careers in emerging technological fields. Furthermore, AI applications in education, including personalized learning platforms and intelligent tutoring systems, can be developed using Python, demonstrating its potential as a tool for innovation in pedagogy. By implementing these recommendations and embracing AI advancements, higher education institutions can establish a culture of continuous learning and technological adaptability. This proactive approach ensures that both educators and students are well-prepared for the demands of a rapidly evolving digital landscape.

References

Code.org (n.d.). *Professional Learning for Educators*. Retrieved from: https://code.org (14.04.2025).

Grover, S., Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43.

Research Review, *22*, 142–158.

Hilčenko, S., Nikolić, S. (2023). CHILD: "I don't understand – we didn't learn that in kindergarten!" *Journal of Education, Technology and Computer Science*, *4*(34), 41–48.

Hilčenko, S., Nikolić, S. (2024). I'm a retiree my brain after the age of 60. *Journal of Education, Technology and Computer Science*, *5*(35), 127–131.

Ličina, B., Viduka, D., Ilić, L. (2020). Application of Raspberry Pi Hardware and Python Programming Language in Small Production Plants. *Quaestus Multidisciplinary Research Journal*, *19*, 305–317.

Lye, S.Y., Koh, J.H.L. (2014). Review on Teaching and Learning of Computational Thinking Through Programming: What is Next for K-12? *Computers in Human Behavior*, *41*, 51–61.

Python.org (n.d.). *Python for Education.* Retrieved from: https://www.python.org (17. 02. 2025).

Resnick, M., Maloney, J., Monroy-Hernández, A. et al. (2009). Scratch: Programming for All. *Communications of the ACM*, *52*(11), 60–67.

Sengupta, P., Kinnebrew, J.S., Basu, S., Biswas, G., Clark, D. (2013). Integrating Computational Thinking with K-12 Science Education Using Agent-Based Computation. *Education and Information Technologies*, *18*(2), 351–380.

Shute, V.J., Sun, C., Asbell-Clarke, J. (2017). Demystifying Computational Thinking. *Educational*

Van Roy, P. (2009). *Programming Paradigms for Dummies: What Every Programmer Should Know.* New Computational Paradigms for Computer Science.

Viduka, D., Kraguljac, V., Ličina, B. (2020). A Comparative Analysis of the Benefits of Python and Java for Beginners. *Quaestus Multidisciplinary Research Journal*, *19*, 318–327.

Wilson, C., Guzdial, M. (2010). How to Make Progress in Computing Education. *Communications of the ACM*, *53*(5), 35–39.

Wing, J.M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33–35.