Teaching Python in the Bachelor: Experiences and Recommendations

Rainer Telesko, FHNW University of Applied Sciences and Arts Northwestern Switzerland, Switzerland

The European Conference on Education 2025
Official Conference Proceedings

Abstract

This paper presents practical insights and pedagogical innovations from teaching Python in the Bachelor of Science in the Business Information Technology (BIT) program at the FHNW University of Applied Sciences and Arts Northwestern Switzerland. Addressing the unique challenges of introducing programming for undergraduate students in a curriculum that merges business and IT, we critically evaluate traditional instructional approaches and propose a shift toward a more modern, student-centered methodology by actively promoting modern technologies. Key obstacles identified include cognitive overload, the abstract nature of programming, and a mismatch between academic content and industry needs. In response, the paper explores the integration of emerging tools such as Large Language Models (LLMs) and the Vibe Coding paradigm to enhance learning engagement, accessibility, and outcomes. LLMs act as code co-pilots and explainers, supporting both students and instructors, while Vibe Coding promotes a design-first, iterative approach to problem-solving through natural language interfaces. Building on these trends, the paper proposes a gamified, AI-supported teaching concept currently under development using Python's pygame library. Features include a modular "Python Islands" learning journey, point-based rewards, daily coding prompts, and collaborative contests. The overarching goal is to foster deeper understanding, active learning, and adaptive thinking in programming education. Initial implementation and feedback are anticipated in 2026.

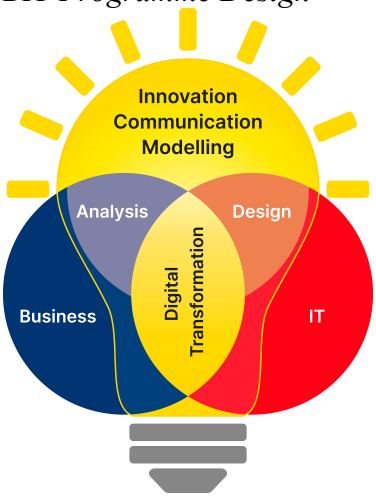
Keywords: Python, teaching programming, LLM, Vibe Coding

iafor
The International Academic Forum
www.iafor.org

Introduction

The Business Information Technology (BIT) Bachelor programme at FHNW blends computer science and management topics to prepare students for roles at the intersection of business and IT. Typical job profiles for students finishing this Bachelor programme comprise IT Project Manager, Business Analyst, Application Manager and Consultant for managing and building IT systems.

Figure 1
BIT Programme Design



Source: www.fhnw.ch

The design of the BIT programme is depicted in Figure 1. The two circles "Business" and "IT" in Figure 1 represent the pillars business administration and computer science. Business administration comprises selected modules from management science like supply chain management, accounting, finance, marketing etc. which are of particular interest for BIT students because of the rising importance of IT. Computer science is an analogy on the IT side with modules like programming, databases, internet technologies etc.

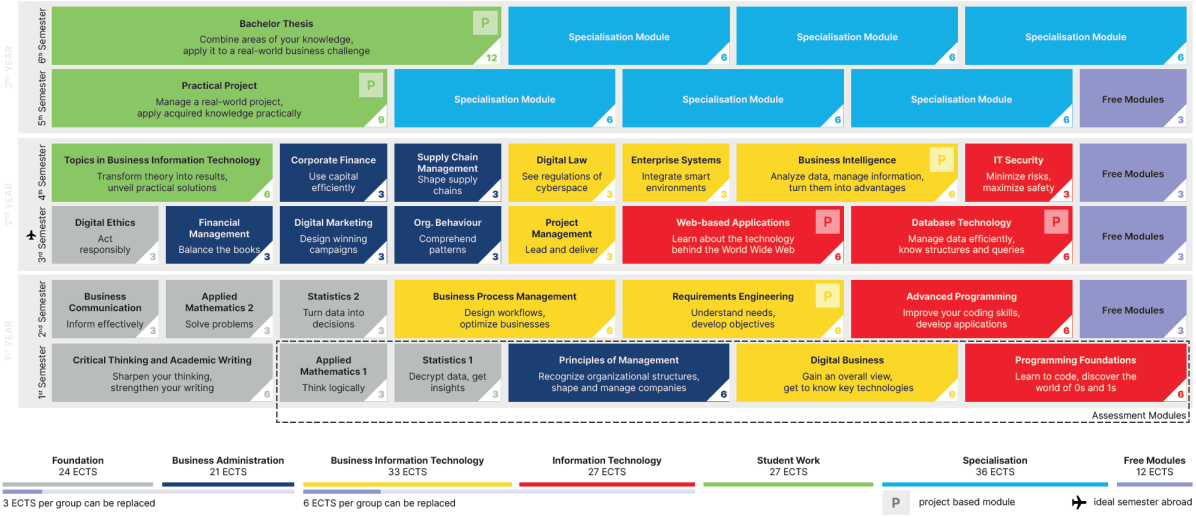
The intersection in the figure "Digitalization" can be considered as the core part of BIT. In courses like Business Process Management, Digital Business, Business Intelligence etc. students learn how IT can be used to bring the business forward. On top of the diagram innovation, communication and modelling highlight areas where BIT seeks to put a particular focus also with regards to competing universities in Switzerland. IT is rapidly evolving, the trade-off to keep the programme stable and nevertheless to offer the latest insights is mastered by offering free / elective modules like Internet of Things, Mobile Computing etc.

Figure 2 shows the organization of the BIT programme into different module groups. The module groups are:

- Foundations: give insight into the necessary basics like mathematics, business communication etc.
- Business Administration: basics of management science relevant for a BIT student
- Business Information Technology (BIT): areas where the interplay between business administration and IT is vital
- Information Technology: give insight into the core IT areas

- Specialization: allows to put the focus on a specific area of interest in the last two terms of the study (offered are Data Science, Digital Business Management and Digital Trust)
- Free modules (electives): takes into account the rapid development of IT and related fields

Figure 2
BIT Module Overview



Source: www.fhnw.ch

Challenges in Teaching Programming

Python is the core programming language in BIT and introduced in two modules. The first module "Programming foundations" is dedicated to the core principles of programming with code organization, variables, controls structures and data types. The second module "Advanced programming" contains advanced content like object-orientation, graphical user interfaces and persistence of data. The specializations build upon the Python basics by introducing advanced libraries like NumPy, Pandas, TensorFlow etc.

As outlined before, BIT is not a "pure hard core" IT programme, therefore teaching programming – widely considered as the "king" discipline in IT – poses a tricky challenge in balancing necessary skills on the one side and not overwhelming undergraduate students (Caspersen et al., 2008; Or-Bach & Lavy, 2004). Based on interviews with our students and teachers in 2021 the following main challenges have been identified (Telesko et al., 2023):

- The starting point of programming is the development of an algorithm, which demands strong analytical skills. It is widely acknowledged among researchers that the ability to abstract is essential for learning to program. However, these skills such as grasping sorting algorithms or organizing concepts into a hierarchy are often lacking at the outset and differ significantly from the kind of thinking students are used to in subjects like mathematics (e.g. solving equations or understanding assignment operations).
- In introductory programming courses, instructors often face the choice of whether to adopt an "OO-first" (object-oriented first) approach as the primary teaching model. Introducing OO concepts from the beginning enables students to develop this mindset early through extensive examples, but it also confronts them with unfamiliar and potentially confusing ideas like inheritance, late binding, overloading, and overriding

in the initial weeks. Conversely, beginning with procedural programming – focusing on data types, control structures, etc. – makes the initial learning curve gentler. However, when OO is introduced later, students frequently question which paradigm to follow and may shy away from OO entirely due to uncertainty. Interviews with students revealed that despite the promise of an "OO-first" curriculum, the understanding of OO principles remained weak by the end of the first term.

- Programming entails juggling multiple tasks at once, such as developing ideas, learning syntax, writing or modifying code, and navigating an IDE. This multitasking often leads to cognitive overload and stress, as students quickly realize time constraints. Moreover, early programming efforts usually yield limited visible results (e.g. basic console output), which can further reduce their motivation.
- Creating software is inherently cross-disciplinary, drawing from areas like infrastructure and project management, requirements analysis, databases, and more. Yet, university curricula are often divided into separate modules, neglecting the interconnected nature of these topics. As a result, programming tends to focus on abstract or simplified "toy" problems, which stands in contrast to the practical and integrated objectives of a Business-IT program.
- Syntax frequently becomes a stumbling block, prompting students to ask "why" certain elements are required. For instance, comparing a basic print command in Python with Java, students often find the boilerplate code in Java difficult to justify or comprehend. Industry practices generally don't align well with academic programming education. While university teaching often emphasizes syntax and algorithmic depth, industry places greater importance on efficient division of labor (such as frontend versus backend) and productivity tools (e.g. DevOps). Additionally, academic staples like recursion are often viewed critically in professional settings. In general, "programming in the large" – relevant to real-world projects and involving multi-layer systems, object-relational mapping, and legacy integration – differs markedly from the "programming in the small" typically taught in beginner-level courses, which usually center on simple desktop apps or basic web frontends.

Important Trends

Large Language Models (LLM)

Large Language Models (LLMs), such as GitHub Copilot OpenAI GPT-4, Anthropic's Claude, Mistral AI, Google Gemini etc., are revolutionizing how programming is taught and practiced. These models can generate code, explain syntax, offer documentation-style answers, and even perform debugging based on natural language prompts. Their impact on teaching software development is profound, with growing academic interest in their integration into curricula. Discussions about the impact of using LLMs on productivity in the professional software industry are ongoing with almost equal positive and negative statements.

From an educational standpoint, LLMs reduce the entry barrier for beginners. Studies by Ahmad et al. (2023) and Finnie-Ansley et al. (2022) highlight that students using LLMs in programming assignments gain confidence and learn faster by receiving immediate, context-aware feedback. However, these benefits come with caveats – overreliance may lead to shallow learning and misconceptions if not scaffolded properly by instructors. In professional settings, Chen et al. (2023) categorize LLM usage into "co-pilot" and "agent" roles, where the former assists the developer while the latter generates complete solutions. A third role –

the "explainer" – may be added which can be regarded as coach to explain the focused code snippets and the underlying theory. This is a task which also helps to support teachers in classes with a big number of students.

While useful for rapid prototyping, concerns persist about code security, maintainability, and explainability. This has led to emerging best practices such as "human-in-the-loop review", LLM code linters, and ethical AI use guidelines (Sridhar et al., 2023).

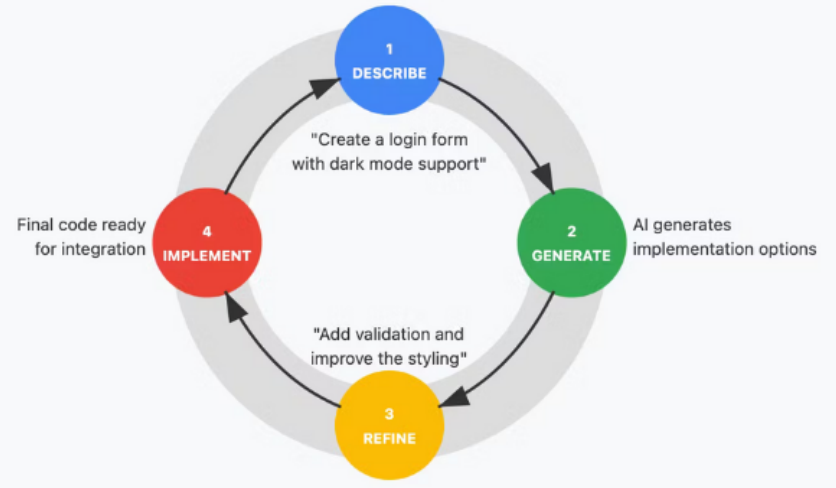
Vibe Coding

Vibe Coding is an innovative pedagogical and professional programming approach that integrates natural-language guidance and AI assistance into the coding process. Introduced by Andrej Karpathy – a software engineer working for Tesla and OpenAI – Vibe Coding encourages intuitive, design-first coding using LLMs. Key advantages include accessibility, rapid prototyping, and enhanced creativity. Karpathy coined the term "Vibe Coding" to express the state of "flow" by guiding AI models (such as LLMs) to generate functional code while focusing on conceptualization, design, and iterative testing.

The Vibe Coding workflow as shown in Figure 3 explains the paradigm of the four step approach (Gehlot, 2025).

- Describe: Here the programming task is described in plain language, akin to explaining concepts to a colleague. This can be regarded as a user requirements specification.
- Generate: AI creates an initial solution which in most cases does not meet the users' expectations.
- Refine: Via prompts the user seeks to push the tool to optimize the existing solution ("try and error").
- Implement: AI-generated code can be difficult to debug due to its dynamic, iterative way of creation and lack of clear architectural structure. Vibe Coding allows testers to leverage AI to generate and execute test cases using natural language thus realizing the next stage called "Vibe Testing".

Figure 3
Vibe Coding Workflow



Source: https://ssojet.com/blog/mastering-vibe-coding-essential-skills/

Vibe Coding can therefore be regarded as an approach where the software engineer acts as "conductor", by letting generate all relevant software engineering artefacts, i.e. code, test cases, comments etc. via natural language prompts.

From a scientific perspective, studies demonstrate Vibe Coding's utility in education and accessibility. Chow and Ng (2025) report improved inclusivity and engagement in clinical teaching through AI-assisted application development. Sapkota et al. (2025) distinguish Vibe Coding from the more rigorous "agentic coding", emphasizing its benefits in early-stage design and conceptual modeling. Chen et al. (2025) show that visually impaired programmers benefit from AI-supported coding workflows, provided interface challenges are addressed. These findings support the use of Vibe Coding in project-based learning, bootcamps, and first-year university programming courses.

However, some experts caution against relying solely on Vibe Coding in professional settings due to potential quality and security concerns (Willison, 2025). This underscores the importance of combining Vibe Coding with foundational knowledge in algorithms, system design, and code review for long-term skill development. It seems to be best suited for rapid, disposable projects rather than work for production-ready environments. This is also emphasized by the fact that recently more and more posts on software engineering platforms ask for troubleshooting or refactoring Vibe Coding solutions.

Enhanced Learning Concept for Python

Based on the results from the exam gradings, interviews with students and teachers and incorporating the paradigm change how programming will likely be done in future we aim to promote a shift from the existing traditional approach to a modern style of teaching programming.

Traditional Approach to Teaching

The traditional method involves teaching via slides, textbooks, and code snippets. While highly structured and predictable (e.g. by the number of slides shown in a unit), it lacks engagement, practical relevance, and fails to stimulate curiosity. Mostly, students are passive and have difficulties moving to an active style because of the complexity of tasks they have to accomplish within a short period of time. These tasks comprise setting up an integrated development environment like Eclipse, IntelliJ or PyCharm and continuously managing the infrastructure (e.g. working with Git and GitHub), finding a suitable algorithm for the problem and finally coding it using the correct syntax. Due to the high effort, we can observe in most lectures that lecturers skip the highly necessary task of writing test cases and performing unit tests.

Summarizing, the classical approach follows the process steps: explaining theory with slides/textbook → showing code examples → coding exercises, where the last step in most cases does not really work as intended. Usually, the lecturers tend to overestimate the velocity of students' coding capabilities. The author observed that the velocity planned – in terms of slides or coding examples – normally is around one third more than what is realized.

The Modern Approach of Teaching

The modern approach now aims at emphasizing actively AI integration and problem-solving via prompt engineering to stimulate the participation of students and to lower mental barriers. The main idea is to reward not only writing but also reading code by taking into account that future software engineers act like "dispatchers" when implementing new features. For the assessment the decisive point is that students can explain code – whoever the author is – down to every single source code line. Discussions about gaining productivity using LLM's are obsolete in our context because coding at universities is done for mere problem solving and not measured by predefined key performance indicators.

Summarizing, the aimed pedagogical shift reflects a new problem-solving process expressed by the following steps: Prompt → LLM Output → Test/Iterate → Final solution.

Realizing the Concept

The new teaching concept will integrate gamification- and AI tools and is currently being implemented in a Bachelor thesis by using the Python "pygame" library.

The new concept is based on the following principles:

- "Showing code, no slides": Students will be encouraged within the guided self-study to get acquainted with the Python "theory." The new curriculum C25+ will reserve more time for this task.
- Active use of AI tools for problem solving, explanation and reflection.

The Bachelor thesis will implement selected gamification features:

- "Python Islands" model: each island covers a well-defined topic (e.g. Strings, Lists, Dictionaries etc.) with quests and paths.
- Reward systems: Points, badges, and themed challenges will stimulate interest and strengthen the competition character among students.
- Daily reminders: Reminders will enable to keep the students active and not to postpone programming tasks for a longer period.
- Coding contests: Good results have been achieved in the past years by introducing coding contests. Usually, a team of 2-3 persons solve a predefined, bigger problem which encompasses the topics covered in the lecture. For this feature, normally the usage of LLMs is forbidden to strengthen the cognitive capabilities.

Conclusion

Teaching programming at the undergraduate level must change. Over the past years, interviews with students / teachers, analysis of module grades and UQM (University Quality Management) results – a survey where students assess a module from their viewpoint – have shown that programming modules need a thorough redesign. The main pain points are excessive demands regarding content and velocity, not suitable learning material and style like mainly using books and slides and not taking into account new technologies like LLMs which drastically will change job profiles in the IT.

Based on these pain points, a Python prototype supporting gamification is currently being implemented. This prototype will be extended in the future using student projects. The prototype will encompass a navigation concept ("Python islands") for structuring learning

topics, a reward system for answering questions and mastering programming challenges, a daily reminder to keep the students active and interested and a coding contest to simulate the "Hackathon" feeling. It is expected to have the first feedback ready by students at the end of the autumn term, i.e. February 2026.

References

Ahmad, W. U., Chakraborty, S., & Chang, K. (2023). Can large language models help students learn programming? A study on feedback and code quality. *arXiv*. https://arxiv.org/abs/2304.12345

Caspersen, M. E., Kölling, M., & Whalley, J. L. (Eds.). (2008). *Proceedings of the Fourth International Workshop on Computing Education Research: 2008, Sydney, Australia, September 6–7, 2008*. ACM Press.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., ... & Zaremba, W. (2023). Evaluating large language models trained on code. *arXiv*. https://arxiv.org/abs/2107.03374

Chen, N., Qiu, L. K., Wang, A. Z., Wang, Z., & Yang, Y. (2025). Screen reader users in the vibe coding era: Adaptation, empowerment, and new accessibility landscape. *arXiv*. https://arxiv.org/abs/2506.13270

Chow, M., & Ng, O. (2025). From technology adopters to creators: Leveraging AI-assisted vibe coding to transform clinical teaching and learning. *Medical Teacher*. https://doi.org/10.1080/0142159X.2025.2488353

Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The robots are coming: Exploring the implications of OpenAI Codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference* (pp. 10–19). ACM. https://doi.org/10.1145/3511861.3511863

Gehlot, G. (2025, March 25). Mastering vibe coding: Essential skills for the future of tech. *SSOJet*. https://ssojet.com/blog/mastering-vibe-coding-essential-skills/

Mohamed, A., Assi, M., & Guizani, M. (2025). The impact of LLM-assistants on software developer productivity: A systematic literature review. *arXiv*. https://arxiv.org/abs/2507.03156

Or-Bach, R., & Lavy, I. (2004). Cognitive activities of abstraction in object orientation. *ACM SIGCSE Bulletin*. https://doi.org/10.1145/1024338.1024378

Sapkota, R., Roumeliotis, K. I., & Karkee, M. (2025). Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic AI. *arXiv*. https://arxiv.org/abs/2505.19443

Sridhar, D., Jang, J., & Ilyas, A. (2023). Trustworthy AI programming assistants: Challenges and design considerations. *Communications of the ACM, 66*(6), pp. 60–68. https://doi.org/10.1145/3594493

Telesko, R., Spahic-Bogdanovic, M., Hinkelmann, K., & Pande, C. (2023, April). A new approach for teaching programming: Model-based Agile Programming (MBAD). In *ACM International Conference Proceeding Series* (pp. 13–18). Association for Computing Machinery. https://doi.org/10.1145/3594441.3594445

Willison, S. (2025). Concerns about code quality in AI-generated software. *Ars Technica*. https://arstechnica.com/?p=123456

Contact email: rainer.telesko@fhnw.ch