

¹ Teaching Python package development: A structured course with learning resources and an instructor's guide

³ **Gerit Wagner**  ¹, **Laureen Thurner**  ¹, **Carlo Tang** ¹, and **Stella Ott** ¹

⁴ 1 Otto-Friedrich Universität Bamberg

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright¹¹ and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).¹²

⁵ Summary

Although there are many open online courses aimed at teaching Python programming,¹³ few educational resources focus on the specific skills required for understanding and developing Python packages.¹⁴ Existing materials typically emphasize programming basics,¹⁵ but the development of Python packages, an essential skill for contributing to the open source community requires deeper knowledge of packaging infrastructure, dependency management, development environments, and best coding practices.¹⁶ This gap leaves learners unprepared for the practicalities of structuring packages, managing dependencies,¹⁷ implementing version control, and ensuring code quality.¹⁸

To address these topics, we present a collection of educational resources designed to teach Python package development.¹⁹ Our materials include a course plan, detailed syllabus,²⁰ slides, and practice notebooks that cover introductory sessions on Git and Python, an initial session on identifying project topics, a session on best practices, and a code review session.²¹ Additionally, we provide a detailed playbook for instructors to guide course delivery.²² Informed by iterations and refinements over four semesters, these resources provide a well-structured and engaging learning experience.²³

The course materials were developed as part of the “Open Source Project” offered by the Digital Work Lab at Otto-Friedrich-Universität Bamberg.²⁴ The [course repository](#) consists of a Jekyll-based website, featuring course pages, instructor notes, slides, and practice notebooks.²⁵ A preconfigured development setup is offered for GitHub Codespaces, ensuring that the required development environment is pre-installed.²⁶ The course resources are hosted on GitHub and designed to be fully modular for reuse and adaptation.²⁷

²⁷ Statement of need

A broad range of Python learning resources is accessible online, reflecting its prominence as a programming language across industries.²⁸ Massive Open Online Courses (MOOCs), such as those offered by Coursera and edX, cater to vast audiences with substantial enrollment figures.²⁹ For instance, Harvard’s CS50’s Introduction to Programming with Python and IBM’s Python for Data Science, AI and Development highlight beginner-friendly content,³⁰ emphasizing foundational skills like using libraries.³¹ These courses are frequently structured around paid certificates and follow conventional formats, making them popular among learners seeking basic programming knowledge or career-oriented credentials.³² However, these MOOCs rarely delve into more advanced topics like Python package development,³³ leaving a noticeable gap for learners aiming to contribute to the open source ecosystem.³⁴

In contrast, resources available for teaching Python package development primarily target self-learners.³⁵ Materials like those offered by [PyOpenSci](#) or the book of Beuzen & Timbers (2020)³⁶ provide valuable insights into creating reusable, distributable Python libraries.³⁷

⁴¹ However, these materials often lack the structured, interactive learning experience offered
⁴² by formal courses. Consequently, while existing resources equip individual learners with
⁴³ practical tools for package development, they do not cater to a broader audience. Addressing
⁴⁴ this gap requires tailored educational materials that combine accessibility with the
⁴⁵ depth necessary to teach Python package development.

Table 1: Overview of selected Python courses

Course Title	Provider	Target	Duration	Enrollment	Libraries
Python for Data Science, AI and Development	IBM (via Coursera)	Beginner	25h	37,000	Using libraries
Python for Everybody Specialization	University of Michigan (via Coursera)	Beginner	2 months at 10 hours a week	212,000	
CS50's Introduction to Programming with Python	Harvard University (via edX)	Beginner	10 weeks	1,086,875	Using libraries
Introduction to Computer Science and Programming Using Python	MIT (via edX)	Beginner	9 weeks	1,718,898	
CS50's Introduction to Artificial Intelligence with Python	Harvard University (via edX)	Beginner	7 weeks	1,132,411	Using libraries
Machine Learning with Python: A Practical Introduction	IBM (via edX)	Beginner	5 weeks	178,346	Using libraries
Programming for Everybody (Getting Started with Python)	University of Michigan (via edX)	Beginner	7 weeks	581,247	
Applied Data Science with Python Specialization	University of Michigan (via Coursera)	Intermediate	4 months at 10 hours a week	26,000	Using libraries
CS50's Web Programming with Python and JavaScript	Harvard University (via edX)	Intermediate	12 weeks	1,482,100	

⁴⁶ The overview of selected Python courses in Table 1 illustrates the popularity and scope
⁴⁷ of beginner-friendly MOOCs, and highlights the gap in resources for advanced Python
⁴⁸ package development.

⁴⁹ Generally, Python package development can be helpful for a range of purposes:

- ⁵⁰ **1. Reusability** Writing Python code from scratch is time-consuming and error-prone.
⁵¹ Many tasks, especially in fields like data science, web development, and automation,
⁵² have well-established solutions in existing Python packages. Learning how to develop
⁵³ packages enables students to make existing code available for reuse, and it also
⁵⁴ develops understanding and skills related to the use of existing packages.
- ⁵⁵ **2. Access to specialized functionality** Considering that the Python core only
⁵⁶ includes general-purpose built-in functionality, packages are often required to provide
⁵⁷ specialized functionality. For instance, this includes tasks like machine learning
⁵⁸ (TensorFlow, Scikit-learn), scientific computing (SciPy), or web development (Flask,
⁵⁹ Django). Understanding these packages allows students to access to a wide range of
⁶⁰ tools and resources that extend Python's functionality for specific purposes.
- ⁶¹ **3. Dependency management** Python packages often rely on external libraries that
⁶² are updated over time to introduce new features or address security vulnerabilities.
⁶³ Managing these dependencies effectively is an important skill, as different packages
⁶⁴ may require specific versions of the same library, leading to potential conflicts. Tools
⁶⁵ like pip and virtual environments provide mechanisms for isolating dependencies,
⁶⁶ but ensuring stability and reproducibility requires a more comprehensive approach.
⁶⁷ One element are cross-platform and cross-Python-version testing strategies to verify
⁶⁸ that a Python package functions consistently across different environments.
- ⁶⁹ **4. Version control, collaborative development, and open source contribution**
⁷⁰ Version control systems, such as Git, are used for managing changes in Python

71 package development. They allow developers to track modifications, revert to
72 previous states, and maintain a clear history of their work. Beyond individual use,
73 Git facilitates collaboration by enabling multiple contributors to work on the same
74 package simultaneously while managing conflicts and merging changes. Collaborative
75 workflows, often supported by platforms like GitHub or GitLab, introduce students to
76 essential practices such as pull requests, code reviews, and issue tracking. These tools
77 not only streamline teamwork but also teach students how to contribute effectively to
78 shared projects. By engaging in open source contributions, students gain additional
79 experience in a community-driven environment, where their work can be reused,
80 improved, and expanded by others. Ideally, this exposure may foster an appreciation
81 for collaborative coding and emphasizes the importance of building packages that
82 are maintainable, accessible, and aligned with community standards.

- 83 **5. Scalability and maintainability of projects** As a project grows in complexity,
84 managing code becomes difficult without proper structure. Packages help modularize
85 code, separating it into manageable units, and using continuous integration tools
86 to maintain code quality. Understanding package development ensures that code is
87 scalable and maintainable. This is essential when building large-scale applications
88 where different parts of the software can be independently developed, tested, and
89 maintained.

90 Our target audience is second- to third-year Bachelor students who already have some
91 foundational programming skills.

92 Learning objectives and outline

93 The specific learning objectives for the capstone project are:

- 94 **1. Understand the fundamentals of Python package structure and distribution** Students will learn how to design, organize, and structure a Python package
95 according to best practices, including creating modular code, setting up essential
96 files (e.g., `pyproject.toml`), and distributing the package using PyPI.
97
- 98 **2. Implement version control and dependency management** Students will
99 develop skills in managing package dependencies and versioning, using tools like
100 `virtualenv` and `poetry` for isolated environments, and ensuring compatibility across
101 various project setups. This includes understanding the role of code quality tools,
102 the importance of semantic versioning, and maintaining stable software releases.
103
- 104 **3. Contribute to open source Python packages and collaborate in package**
105 **development** Students will gain hands-on experience contributing to open source
106 Python projects by collaborating on GitHub, creating pull requests, resolving issues,
107 and following community-driven development standards. They will also learn how
108 to write documentation and test their packages to ensure quality and usability.

109 Figure 1 provides an overview of the course timeline, showcasing sessions and group work
110 activities that facilitate a step-by-step progression through Python package development
111 concepts. The timeline emphasizes iterative learning, with early sessions focused on
112 foundational skills, followed by group work phases that foster collaboration and practical
113 application. During this time, students are encouraged to iterate between individual coding,
114 group sessions, and hacking sessions with the instructor to discuss current challenges and
115 next steps.

116 In addition, a *Best Practices* session is offered at the beginning of the group work phase.
117 Toward the end of the course, students will open a pull request with their work and
118 participate in a code review session in which they adopt the perspective of a maintainer
119 and evaluate the code of another group. Code improvements are implemented within a
120 week, and student reflections are discussed at the end. When merging the contributions,
we include students as contributors of the package.

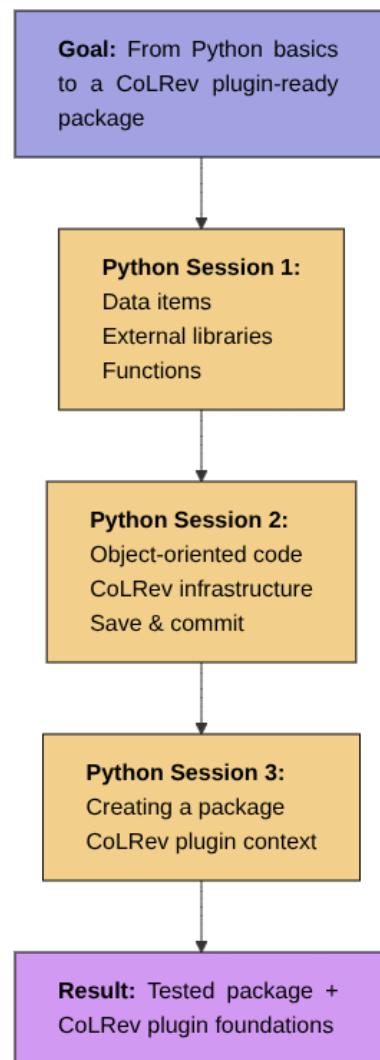


Figure 1: Course timeline with sessions and group work activities

121 **Teaching materials**

122 The delivery of the course is designed to foster active learning through a blend of in-person
 123 sessions, individual group work, and interactive hacking sessions, all facilitated by the
 124 instructor. This approach emphasizes collaboration and hands-on practice, enabling
 125 students to engage deeply with Python package development in a supportive environment.
 126 The materials are structured for complex and integrated learning activities, with a strong
 127 focus on Git-based collaboration. Students work together on shared repositories, navigating
 128 real-world workflows such as branching, merging, and resolving conflicts, which mirror
 129 professional development environments. The Git component of the course builds on the
 130 work of Wagner & Thurner (2025) to ensure a robust foundation in version control while
 131 emphasizing practical applications that enhance both technical skills and teamwork.

Table 2: Materials

Resource	Description and focus
Landing page	Provides an accessible overview of the course, aimed to engage students.
Syllabus	Offers a structured, detailed overview of course objectives, content, and pedagogical approach, complementing the landing page with broader course context.
Slides	Slides in Markdown (Marp) format.
Notebooks	Designed to engage directly with the Python package, using <code>git reset</code> to access solutions, supporting a smooth, practical learning flow.
Teaching notes	The teaching notes contain preparation checklists, scheduled mailings, session readers, and a concept.

¹³² Pedagogical considerations

¹³³ Teaching Python package development requires a structured approach that balances
¹³⁴ simplicity with depth, ensuring students build a solid foundation before progressing to
¹³⁵ more advanced topics. The course design is informed by key pedagogical principles:

- ¹³⁶ **1. Select and Simplify** To reduce cognitive overload, we prioritize simplicity in
¹³⁷ tools and workflows. For example, we use GitHub Codespaces to standardize
¹³⁸ setups, eliminating issues related to different operating systems and environment
¹³⁹ configurations. A focused approach aligns with cognitive load theory (Sweller, 1994),
¹⁴⁰ helping students concentrate on core concepts.
- ¹⁴¹ **2. Gradually Progress in Complexity** Starting with basic Python and Git skills,
¹⁴² the material builds incrementally, introducing concepts like dependency manage-
¹⁴³ ment and package distribution after students have developed sufficient familiarity
¹⁴⁴ with foundational skills. This approach reduces the risk of overwhelming learners
¹⁴⁵ (Anderson et al., 2001).
- ¹⁴⁶ **3. Learn interactively and in groups** Interactive and collaborative learning plays a
¹⁴⁷ crucial role in student engagement and knowledge retention (Guzdial, 1998). The
¹⁴⁸ course incorporates live coding sessions, and group-based exercises to make practices
¹⁴⁹ as accessible as possible and encourage active participation (Vial & Negoita, 2018).
¹⁵⁰ We build on the principles of active learning to promote deeper understanding
¹⁵¹ through hands-on practice and peer collaboration. In particular, group problem-
¹⁵² solving can foster a collaborative environment where learners can exchange ideas,
¹⁵³ learn from one another, and build confidence in their coding skills (Freeman et al.,
¹⁵⁴ 2014).

¹⁵⁵ Development environment

¹⁵⁶ Our recommended setup for Python package development is GitHub Codespaces, a cloud-
¹⁵⁷ based development solution featuring a graphical interface of VisualStudio Code as well
¹⁵⁸ as a full Python environment with pre-installed dependencies and configuration¹. With
¹⁵⁹ Codespaces, students can start their work directly from a browser, where all necessary de-
¹⁶⁰pendencies are automatically configured. GitHub Codespaces offers several key advantages

¹Local VirtualBox images were too slow on most student machines, and resources for self-hosted virtual machines were not available.

161 for Python package development. Offloading all computational tasks to remote servers
162 eliminates the performance issues that often arise when running development environments
163 on local machines. Additionally, the environment is fully standardized, meaning every
164 student works with the same configuration, reducing the variability and potential issues
165 seen in local setups. The Codespaces startup scripts effectively allow us to set up the devel-
166 opment environment automatically and without user interaction. This approach not only
167 saves time but also mirrors the benefits described by Malan (2024), where containerization
168 minimized technical challenges and enhanced the learning experience.

169 For students who prefer a local development setup, we also offer options like Windows
170 Subsystem for Linux (WSL) for Windows users, ensuring they can work with a Linux-like
171 environment while still on their native operating system. This provides flexibility while
172 maintaining the core benefits of a standardized development environment. By enabling
173 students to work in a consistent environment regardless of their operating system, we
174 ensure that everyone has access to the same tools, configurations, and workflows. It also
175 ensures equal opportunities, regardless of students' choice of operating system and enables
176 them to collaborate effectively without technical barriers.

177 **Reuse and modification of materials**

178 The materials provided in this course are designed for easy reuse and modification by other
179 instructors. While the course uses the CoLRev Python package (Wagner & Prester, 2024)
180 as the example context, the materials allow instructors to adapt the content to focus on
181 different Python packages. The learning environment, hosted on GitHub and built with
182 the Just-the-Docs framework, can be cloned, enabling instructors to replicate the entire
183 setup. All instructional content, including slides and practice notebooks, are automatically
184 generated and updated via GitHub Actions, ensuring the materials remain up-to-date.

185 **Story of the project**

186 This project was developed at Otto-Friedrich-Universität of Bamberg, where Gerit Wagner
187 initiated a new team-based course to provide students with hands-on experiences in
188 programming and software development. While other project courses in the program had
189 a stronger focus on management topics, this initiative—launched in 2023—was designed
190 to help students engage with realistic open source software engineering practices. The goal
191 was to move beyond short, standalone Python scripts, and contribute collaboratively to
192 public, functional, maintainable, and open source software packages.

193 Building on Gerit's role as lead developer of the CoLRev open source environment, the
194 course adopted CoLRev as a learning platform. Students developed small extensions or
195 plugins for the system, allowing them to reuse existing functionality (e.g., for loading
196 bibliographic data or interacting with APIs) and extend it by implementing new features
197 such as API wrappers or automation tools. This setup provided an authentic context for
198 learning how to organize projects in teams using Git and GitHub, manage dependencies,
199 write tests, and produce corresponding documentation.

200 Over time, the course evolved substantially through iterative refinement based on student
201 feedback. Early cohorts found the technical scope demanding, prompting the integration of
202 more scaffolding, structured guidance, and formative feedback. Since then, the course has
203 grown into a well-established project with high student satisfaction and recommendation
204 rates. Five cohorts of students have contributed to CoLRev and related open source
205 packages, many continuing their engagement through bachelor's theses on extensions such
206 as the search-query (Eckhardt et al., 2025) or bib-dedupe (Wagner, 2024) packages.

207 References

- 208 Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E.,
209 Pintrich, P. R., Raths, J., & Wittrock, M. C. (2001). *A taxonomy for learning, teaching,
210 and assessing: A revision of bloom's taxonomy of educational objectives, complete
211 edition*. Longman Publishing Group.
- 212 Beuzen, T., & Timbers, T. (2020). *Python packages* (1st ed.). Chapman & Hall/CRC The
213 Python Series. ISBN: 9781138332250
- 214 Eckhardt, P., Ernst, K., Fleischmann, T., Geßler, A., Schnickmann, K., Thurner, L.,
215 & Wagner, G. (2025). Search-query: An open-source python library for academic
216 search queries. *Journal of Open Source Software*. <https://github.com/openjournals/joss-reviews/issues/8775>
- 217
- 218 Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., &
219 Wenderoth, M. P. (2014). Active learning increases student performance in science,
220 engineering, and mathematics. *Proceedings of the National Academy of Sciences*,
221 111(23), 8410–8415. <https://doi.org/10.1073/PNAS.1319030111>
- 222 Guzdial, M. (1998). Use of collaborative multimedia in computer science classes. *ACM
223 SIGCSE Bulletin*, 30(3), 61–64.
- 224 Malan, D. J. (2024). Containerizing CS50: Standardizing students' programming environments.
225 In *Proceedings of the 2024 on innovation and technology in computer science
226 education v. 1* (pp. 534–540). <https://doi.org/10.1145/3649217.3653567>
- 227 Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design.
228 *Learning and Instruction*, 4(4), 295–312. [https://doi.org/10.1016/0959-4752\(94\)90003-5](https://doi.org/10.1016/0959-4752(94)90003-5)
- 229
- 230 Vial, G., & Negoita, B. (2018). Teaching programming to non-programmers - the case
231 of Python and Jupyter Notebooks. *Proceedings of the International Conference on
232 Information Systems*.
- 233 Wagner, G. (2024). Bib-dedupe: An open-source python library for deduplication of
234 bibliographic records. *Journal of Open Source Software*, 9(97), 6318. <https://doi.org/10.21105/joss.06318>
- 235
- 236 Wagner, G., & Prester, J. (2024). *CoLRev: An open-source environment for collaborative
237 reviews* (Version 0.12.3). <https://doi.org/10.5281/ZENODO.11668338>
- 238 Wagner, G., & Thurner, L. (2025). Rethinking how we teach git: Pedagogical recommen-
239 dations and practical strategies for the information systems curriculum. *Journal of
240 Information Systems Education*, 36(1).