Journal of
open research software

SOFTWARE METAPAPER

]u[ubiquity press

CodeBuddy: A Programming Assignment Management System for Short-Form Exercises

STEPHEN R. PICCOLO
EMME TUFT
P. J. TATLOW
ZACH ELIASON
ASHLIE STEPHENSON

*Author affiliations can be found in the back matter of this article

ABSTRACT

CodeBuddy is a software system for delivering computer-programming assignments to students. It is primarily used for short-form exercises, such as those delivered in introductory-programming courses and informal-learning settings. It provides a Web-based interface, the ability to execute code in a secure environment, support for custom testing logic, near-immediate feedback to students, and support for many programming languages. Other features include support for graphics-based programming exercises, pair programming, the ability for students to review the instructor's solution after solving an exercise, and an intelligent tutor. Upon creating an account, each student is randomly assigned to an "A" or "B" cohort, thus enabling researchers to perform pedagogical research via online controlled experiments. These and other features offer opportunities for instructors to customize the learning experience, in diverse ways, for students learning to program.

CORRESPONDING AUTHOR:
Stephen R. Piccolo
Department of Biology, Brigham Young University, Provo, UT, USA
stephen_piccolo@byu.edu

KEYWORDS:
programming education; automated grading; pair programming; intelligent tutor; web application; automated assessment

TO CITE THIS ARTICLE:
Piccolo SR, Tuft E, Tatlow PJ, Eliason Z, Stephenson A. 2025 CodeBuddy: A Programming Assignment Management System for Short-Form Exercises. *Journal of Open Research Software,* 13: 1. DOI: https://doi.org/10.5334/jors.503

INTRODUCTION

Many software systems have been created to manage computer-programming assignments [1, 2]. Some are commercial applications that require students or institutions to pay fees [3–6]. Others are open source software, thus providing benefits of transparency and extensibility but requiring the institution to host the software. For some types of programming assignments, students submit a solution that consists of multiple source files and perhaps configuration file(s). These solutions may reach thousands of lines of code. Various open-source tools are designed for this type of assignment, including Web-CAT [7], Submitty [8], CodEval [9], and Drop Project [10]. In other settings, programming assignments consist of short-form exercises—typically, those that can be solved in at most 100–200 lines of code within a single file or script. Such exercises are often delivered in introductory computing courses to help students gain experience with basic programming skills. In many cases, it is desirable for students to learn these skills in a setting where they can focus on programming tasks without simultaneously learning to use an integrated development environment (IDE). Existing open-source tools that support this use case include Mooshak [11], CodeRunner [12], CodeWorkout [13], CodeOcean [14], Jutge.org [15], and EDGAR [16]. Features vary across these tools, but they commonly include A) a Web-based interface, B) the ability to execute code and receive semi-immediate feedback, C) validation against expected outputs and/or unit tests, D) support for multiple programming languages, E) code execution in a secure environment, F) the ability to make multiple submissions for a given exercise, G) support for alternative problem types such as multiple-choice questions, and H) support for custom grading. We created CodeBuddy, an open-source tool that supports all of these features and many others. Although originally designed for higher-education courses, CodeBuddy's functionality is also suitable for K-12 and informal learning.

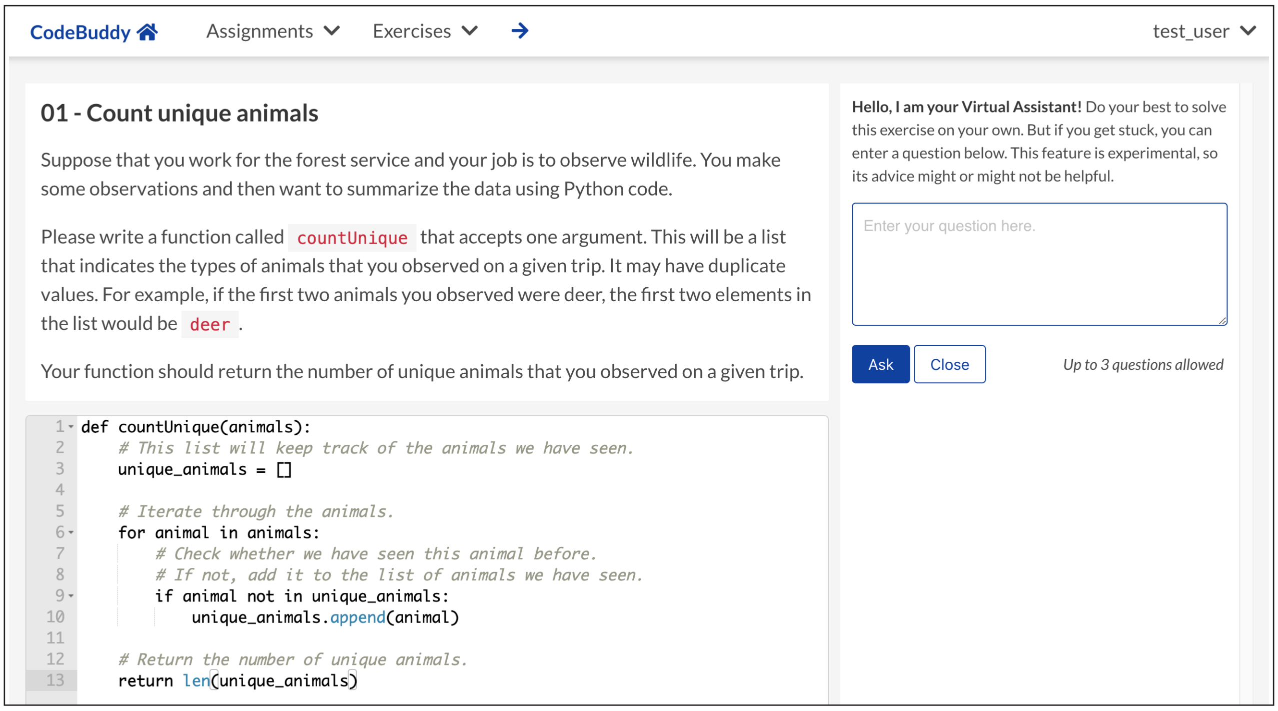
After landing on the CodeBuddy home page, users authenticate via a third-party service (currently, Google is supported). The first user to access a newly created instance is given administrative privileges, which include the ability to create courses and assign other users as instructors. When each user creates an account, CodeBuddy randomly assigns them to either an "A" or "B" cohort. These groups can be used for online controlled experiments [17]. CodeBuddy allows instructors to perform such experiments for some existing features (described below). Because the code is open source, researchers may implement new features and evaluate their effectiveness in this way.

A title and introduction must be specified for each course. Optionally, a passcode may be specified as a means of restricting enrollment. An instructor can indicate whether a given course is visible to students; for example, they might hide a course temporarily while it is being developed. Instructors can create assignments (within courses) and exercises (within assignments).

When an instructor creates an assignment, they specify a title, introduction, and whether to show the "Run" button when students write code. (The "Run" button allows students to test their code before submitting it for scoring.) Optionally, an instructor may specify a start date, due date, and/or time limit. When a due date has been specified, the instructor indicates whether late submissions are allowed and whether students can view the instructor's solution after the due date has passed. If late submissions are allowed, the instructor selects a percentage indicating how many points students can earn with late submissions. When an assignment has a time limit, the instructor indicates whether to treat the assignment as an examination. When this option is selected, students are unable to access other assignments while they are completing the examination. The instructor can specify time-limit exceptions for individual students.

Assignments have three additional options. First, instructors can restrict access to computers with particular IP addresses; this setting would typically be used in examination settings. Second, in some cases, instructors deliver examinations in a controlled environment where students do not have access to external websites; instructors can list URLs that are exceptions. CodeBuddy retrieves content from these URLs, caches a local copy, and allows students to access them during the examination. Third, instructors can indicate whether the "Virtual Assistant" is available for a given assignment. When it is enabled, students can ask questions about their code while completing exercises (Figure 1). CodeBuddy generates a prompt for a large language model (LLM) based on the exercise requirements, the student's code, and the student's question. The prompt requests that the LLM provide suggestion(s) but not code. CodeBuddy sends the prompt to OpenAI's ChatCompletion API [18] and displays the response to the student. Instructors can limit the number of interactions per student per exercise. This feature requires the instructor to specify configuration details for a (paid) OpenAI account. The Virtual Assistant can be made available to A) no students, B) all students, C) students in the "A" cohort, or D) students in the "B" cohort.

For a given exercise, an instructor specifies a title, instructions, and a solution. Exercise instructions are formatted using Markdown syntax [19], which enables flexibility in formatting without the complexity of markup languages like HTML and LaTeX [20]; however, HTML can be used if desired. When providing instructions, an instructor can include placeholders that are replaced with a) a link to the previous exercise, b) a prompt asking students to reflect on their solution;



01 - Count unique animals

Suppose that you work for the forest service and your job is to observe wildlife. You make some observations and then want to summarize the data using Python code.

Please write a function called `countUnique` that accepts one argument. This will be a list that indicates the types of animals that you observed on a given trip. It may have duplicate values. For example, if the first two animals you observed were deer, the first two elements in the list would be `deer`.

Your function should return the number of unique animals that you observed on a given trip.

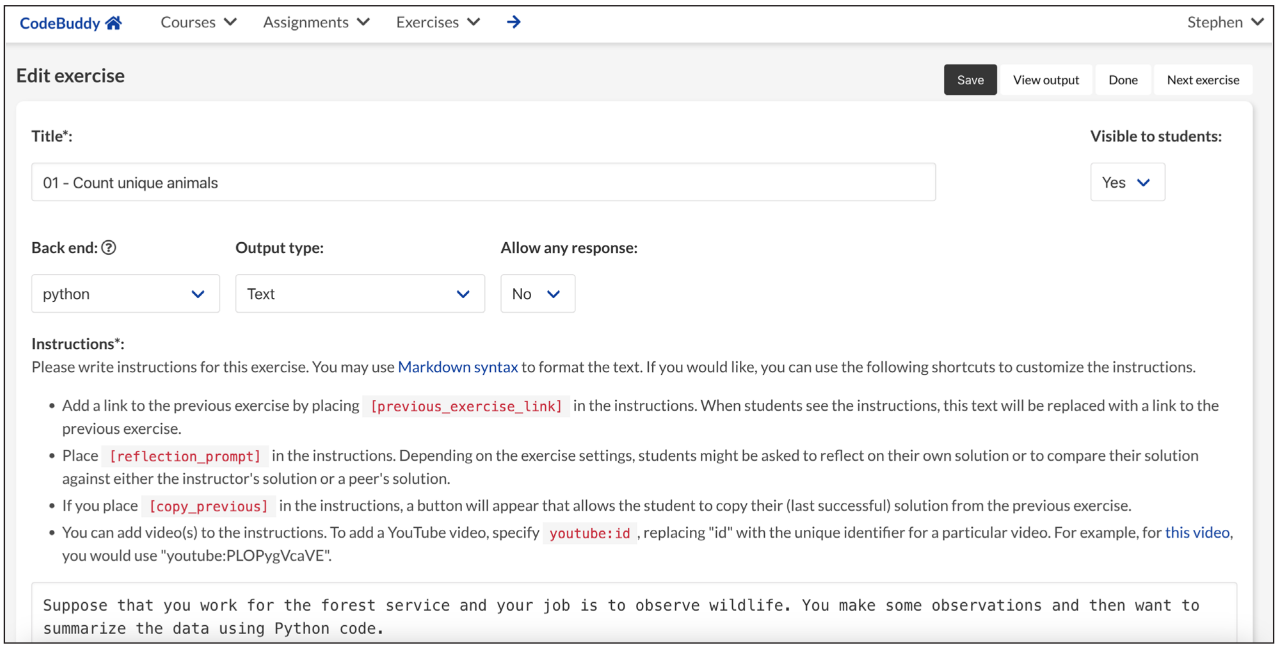
```
def countUnique(animals):
    # This list will keep track of the animals we have seen.
    unique_animals = []

    # Iterate through the animals.
    for animal in animals:
        # Check whether we have seen this animal before.
        # If not, add it to the list of animals we have seen.
        if animal not in unique_animals:
            unique_animals.append(animal)

    # Return the number of unique animals.
    return len(unique_animals)
```

Hello, I am your Virtual Assistant! Do your best to solve this exercise on your own. But if you get stuck, you can enter a question below. This feature is experimental, so its advice might or might not be helpful.

Enter your question here.

Up to 3 questions allowed

Figure 1 An example showing how the exercise page appears to students when the Virtual Assistant is enabled.



Edit exercise

Title*: 01 - Count unique animals

Visible to students: Yes

Back end: python

Output type: Text

Allow any response: No

Instructions*:
Please write instructions for this exercise. You may use Markdown syntax to format the text. If you would like, you can use the following shortcuts to customize the instructions.

- Add a link to the previous exercise by placing `[previous_exercise_link]` in the instructions. When students see the instructions, this text will be replaced with a link to the previous exercise.
- Place `[reflection_prompt]` in the instructions. Depending on the exercise settings, students might be asked to reflect on their own solution or to compare their solution against either the instructor's solution or a peer's solution.
- If you place `[copy_previous]` in the instructions, a button will appear that allows the student to copy their (last successful) solution from the previous exercise.
- You can add video(s) to the instructions. To add a YouTube video, specify `youtube:id`, replacing "id" with the unique identifier for a particular video. For example, for this video, you would use "youtube:PLOPygVcaVE".

```
Suppose that you work for the forest service and your job is to observe wildlife. You make some observations and then want to
summarize the data using Python code.
```

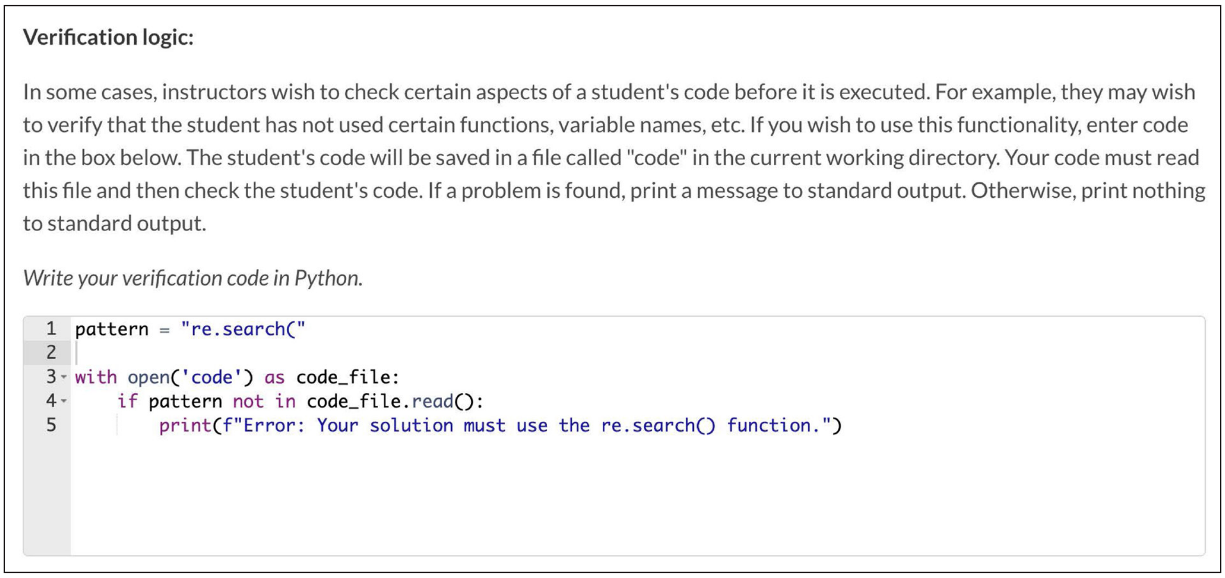
Figure 2 An example of the exercise-settings page.

c) a button that allows the student to copy their (last successful) solution from the previous exercise, thus making it easier to deliver exercise sequences; or d) a YouTube video. A common use case is to embed a video and ask students to post code- or text-based responses.

In the exercise settings, instructors specify a "back end" to be used. Most back ends represent programming languages. The following programming languages are currently supported: bash scripting, C, C++, Java, Javascript, Julia, Python, R, and Rust. One additional back end ("not_code") allows for submissions that are not computer code. For each back end, the instructor specifies an output type. The default is "Text." However, for the Python and R back ends, graphics-based outputs can also be used. For Python, graphics are generated using the *matplotlib* or *seaborn* packages [21, 22]. For R, graphics are generated using the *ggplot2* package [23]. For any back end or output type, the instructor indicates whether students' outputs must match the instructor's outputs (the default). When matching is required, students' responses are graded on a pass/fail basis. When matching is not required, students receive points for any response.

The exercise settings (Figure 2) including the following, additional options:

- Instructors can configure exercises to support *pair programming*, an evidence-based practice in which two students work together at a single computer [24]. When either student submits code for a given exercise, the code (and resulting score) are stored under both students' accounts.
- Each exercise is assigned a weight. Assignment scores are calculated based on whether students pass or fail each exercise and the weights assigned to the exercises in the assignment.
- Instructors can provide data file(s) to be used as inputs for exercises. When a student's code is executed, the data file(s) are saved to the server's file system (within a Docker container) where they can be parsed by the student's code. The maximum size of the data (across all files) is 10,000,000 characters.
- Instructors can write "verification logic" to statically analyze students' code before it is executed. For example, if an instructor asked students to complete an exercise using regular expressions, they could parse the students' code to verify that they had used regular expressions rather than simpler string-manipulation methods (Figure 3).
- Instructors can write tests to verify the functionality of students' code. For exercises with text outputs, CodeBuddy verifies that the output of the students' code matches that of the instructor's solution. For exercises with graphics outputs, CodeBuddy performs a pixel-based comparison between the student's and instructor's images.
- For each test, the instructor can author code that will be executed *before* the student's code. For example, they could declare variables that the student's code must use. Additionally, the instructor can author code that will be executed *after* the student's code. For example, if the instructions ask the student to write a function, the test code can invoke the student's function using a variety of arguments. By providing multiple tests, the instructor can verify that the student does not "hard code" specific values.
- Instructors can create tests with "hidden" inputs and/or outputs. For example, they can ask students to write a function that accepts certain arguments but not tell them what the values are. This prevents students from circumventing test requirements.
- Instructors can provide starter code for students.
- Instructors can provide hints. Students can view the hints after clicking a button.
- Instructors can limit the number of submissions that a student can make for a given exercise.
- Instructors can allow students to see the instructor's solution after they have solved a given exercise (Figure 4). This feature can be made available to A) no students, B) all students, C) students in the "A" cohort, or D) students in the "B" cohort.
- Instructors can allow students to see anonymized solutions from peers after they have solved a given exercise (Figure 5). When a student selects this option, passing solutions from other students are randomly shuffled, and one of those solutions is displayed to the student (if at least three students have solved the exercise). The student can refresh the page to repeat the shuffling process. Instructors can make this feature available to A) no students, B) all students, C) students in the "A" cohort, or D) students in the "B" cohort.
- Instructors can configure an exercise so that students are asked to write a reflection about their solution and how it compares with the instructor's or peer's solutions to the previous exercise.
- Instructors can provide a written description of their solution.



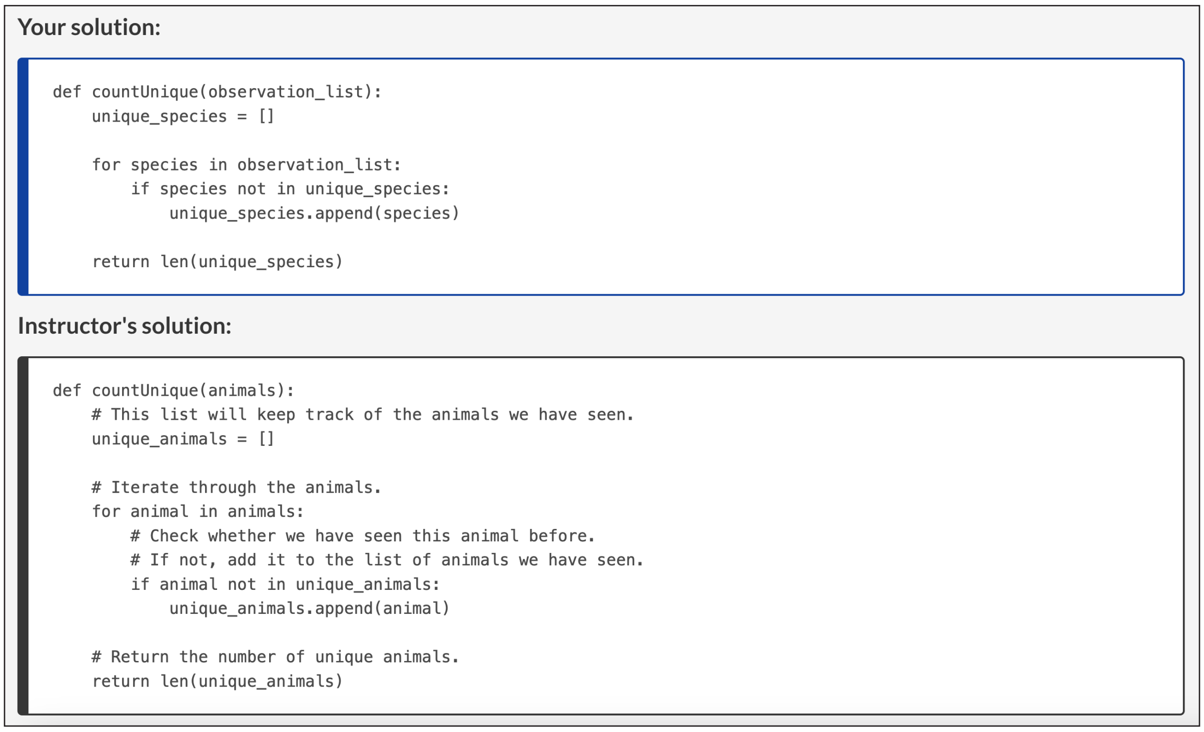
Verification logic:

In some cases, instructors wish to check certain aspects of a student's code before it is executed. For example, they may wish to verify that the student has not used certain functions, variable names, etc. If you wish to use this functionality, enter code in the box below. The student's code will be saved in a file called "code" in the current working directory. Your code must read this file and then check the student's code. If a problem is found, print a message to standard output. Otherwise, print nothing to standard output.

Write your verification code in Python.

```python
pattern = "re.search("

with open('code') as code_file:
    if pattern not in code_file.read():
        print(f"Error: Your solution must use the re.search() function.")
```

Figure 3 An example of using verification logic to statically analyze code. This example is from a Python programming exercise in which the student is asked to use the `re.search()` function to identify a particular string pattern in text. In some cases, students try to implement the logic using alternative means. The verification logic ensures that the student's code uses `re.search()`.

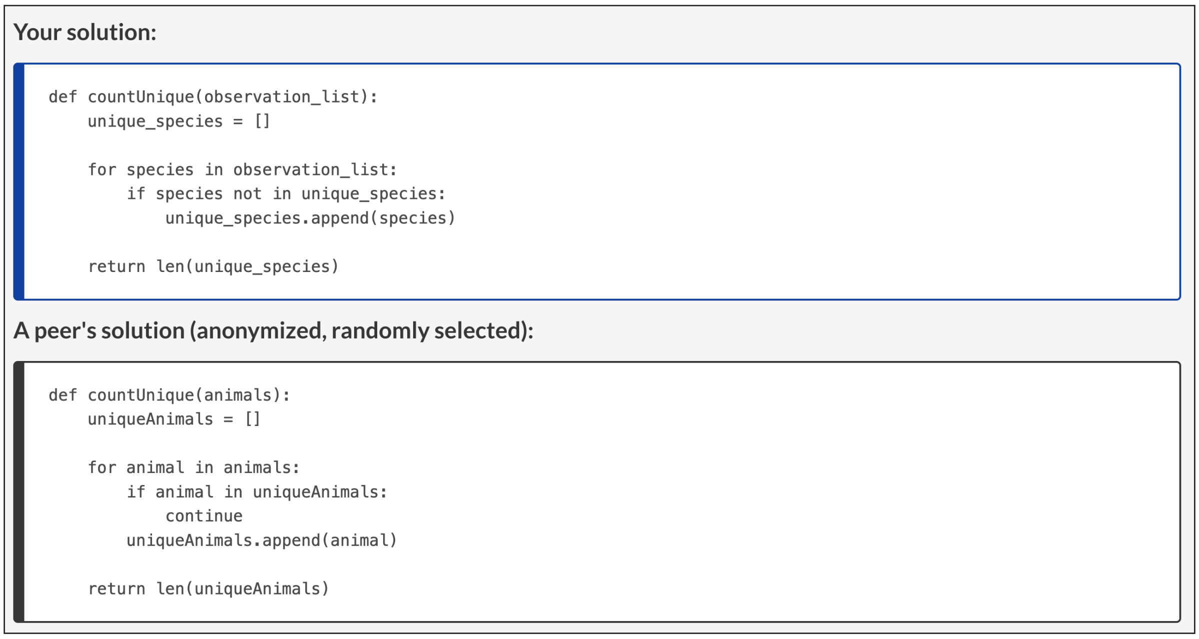

Your solution:

```
def countUnique(observation_list):
    unique_species = []

    for species in observation_list:
        if species not in unique_species:
            unique_species.append(species)

    return len(unique_species)
```

Instructor's solution:

```
def countUnique(animals):
    # This list will keep track of the animals we have seen.
    unique_animals = []

    # Iterate through the animals.
    for animal in animals:
        # Check whether we have seen this animal before.
        # If not, add it to the list of animals we have seen.
        if animal not in unique_animals:
            unique_animals.append(animal)

    # Return the number of unique animals.
    return len(unique_animals)
```

Figure 4 An example showing the ability for students to view the instructor's solution after completing an exercise.


Your solution:

```
def countUnique(observation_list):
    unique_species = []

    for species in observation_list:
        if species not in unique_species:
            unique_species.append(species)

    return len(unique_species)
```

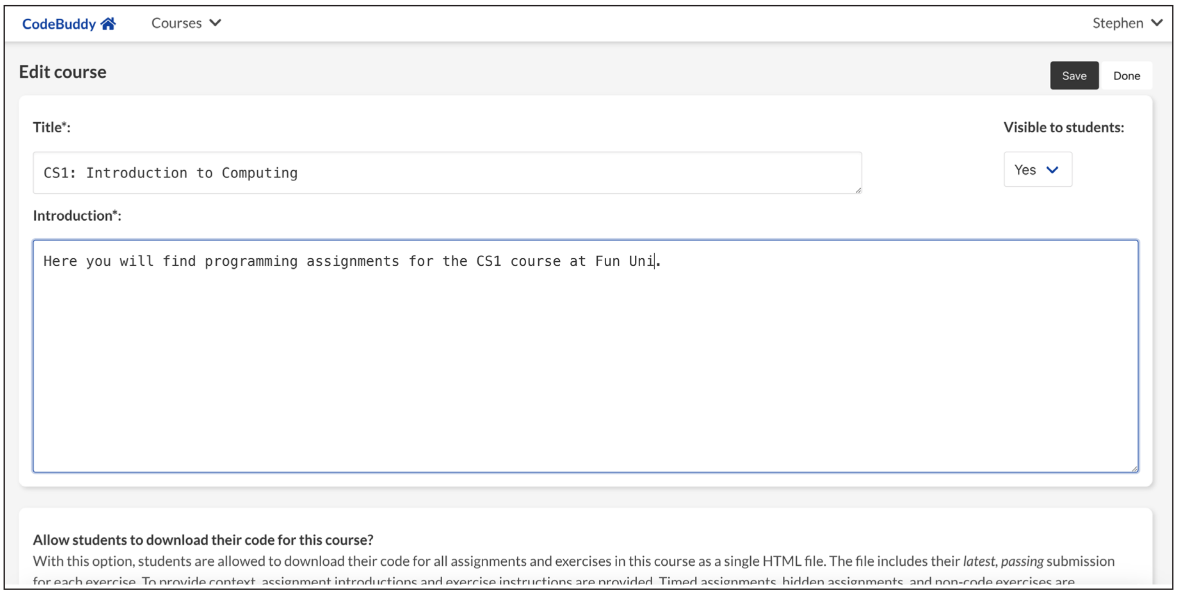
A peer's solution (anonymized, randomly selected):

```
def countUnique(animals):
    uniqueAnimals = []

    for animal in animals:
        if animal in uniqueAnimals:
            continue
        uniqueAnimals.append(animal)

    return len(uniqueAnimals)
```

Figure 5 An example showing the ability for students to view a peer's solution after completing an exercise.

CodeBuddy supports additional features for managing content, users, grading, preferences, and the student experience.

- New administrators can be added.
- Existing administrators can remove their own administrative privileges (but not others' privileges).
- Instructors can import and export assignments.
- Instructors can edit, copy, move, and delete assignments and exercises (Figure 6).
- Instructors can view a table of "at-risk" students who have not made a submission in the past *x* number of hours (or days).
- Instructors can allow students to download all of their latest, passing code submissions. Students often use this feature at the conclusion of a course.
- Instructors can view course- and assignment-level summaries that show the number of completed assignments per course and the number of completed exercises per assignment, respectively; average scores are also shown.
- Instructors can export scores as delimited text files.
- Instructors can review students' submissions and edit scores manually.



Edit course

Title*:

```
CS1: Introduction to Computing
```

Visible to students: Yes

Introduction*:

```
Here you will find programming assignments for the CS1 course at Fun Uni.
```

Allow students to download their code for this course?

With this option, students are allowed to download their code for all assignments and exercises in this course as a single HTML file. The file includes their *latest, passing* submission for each exercise. To provide context, assignment introductions and exercise instructions are provided. Timed assignments, hidden assignments, and non-code exercises are

Figure 6 An example of the course-settings page.

- Instructors can assign users as teaching assistants (TAs). TAs can access the instructor's solutions and students' scores. They are *not* permitted to change some settings.
- All page accesses are stored in log files, which are summarized once per day. Administrators can view and filter these summaries via the Web interface.
- Instructors can configure an assignment so that other assignment(s) must be completed as prerequisite(s). Students are unable to access the assignment(s) until they have completed the prerequisite(s).
- Instructors can require students to have a security code to access an assignment. The instructor can generate a PDF with a separate page for each student. The student would then receive one of these pages upon starting the assignment to ensure that the student takes the assignment in a secure location (i.e., testing center). Optionally, the instructor can configure the assignment so that a confirmation code is generated when the student completes the assignment (or ends it early). They must give this confirmation code to the instructor (or a proxy) to ensure that they completed the assignment in the secure location.
- Instructors can include multiple-choice (or multiple-answer) exercises in assignments.
- An assignment score is calculated as the average of the exercise scores within the assignment. Alternatively, instructors can specify custom scoring logic. For example, suppose an assignment has five exercises. An instructor might specify that if a student successfully completes one exercise (20%), the student receives a passing grade (60%). And/or they might specify that if a student completes all but one of the exercises (80%), the student receives full points (100%) for the assignment.
- Administrators can configure the database to use write-ahead logging. With this option, it is possible to perform automatic backups to remote servers using third-party tools such as Litestream (https://litestream.io).

After logging in, students see an option to register for existing courses. Upon registering for one or more courses, they see a list of registered courses on the main landing page. After clicking on one of these courses, they view a list of assignments for that course, along with icons indicating whether each assignment has been completed and the number of exercises completed. Where applicable, they also see start dates, due dates, and whether the assignment is timed. Upon clicking on one of the assignments, they see a list of exercises. For each exercise, they see the number of submissions they have made, whether it has been completed, and their current score. When pair programming is enabled for a given exercise, an icon indicates this. Upon selecting a given exercise, students see the exercise title, instructions, a text box where code is entered, and a "Submit" button for submitting code. When a student has made multiple submissions, they see buttons enabling them to view previous submissions. Under default settings, a "Run" button is shown on the exercise page; this enables students to execute code and see its output before submitting it for a score. The instructor can optionally remove this button (for example, if they have placed a limit on the number of submissions). When the Virtual Assistant is enabled, students see an additional panel where they can enter questions and see its responses. Finally, a Preferences page enables all users to specify application preferences.

EVALUATION

From January to April 2024, we surveyed students in two separate courses regarding their use of CodeBuddy. In these courses, students learned introductory programming skills in the Python (https://python.org) and R (https://r-project.org) languages. Assignments consisted of short-form exercises in each course. We conducted a short survey to evaluate students' perceptions of the software. This survey was approved by Brigham Young University's Institutional Review Board under exempt status (IRB2023-244). The survey questions included the following:

- How helpful for your learning did you find CodeBuddy overall?
- Which feature(s) did you find most helpful for your learning in CodeBuddy?
- What suggestions, if any, do you have about improving CodeBuddy?

For the first question, students were asked to rate the software using a Likert scale. Allowed responses were "Extremely unhelpful," "Somewhat unhelpful," "Somewhat helpful," or "Extremely helpful." Of the respondents, 88.4% indicated that the software was "Somewhat helpful" or "Extremely helpful" (Figure 7). In response to the second question, students commented positively on A) the ability to make multiple submissions and use the "Run" button, B) support for pair programming, C) the ability to review the instructor's solution, and D) being able to see expected outputs and compare them against their own outputs. Students commented that they would have liked it to function more like an IDE (with debugging capabilities), to have more options for customizing the software's appearance, to provide easier navigation, and not to have tests with hidden inputs/outputs.

IMPLEMENTATION AND ARCHITECTURE

On the client side, CodeBuddy uses HTML, Cascading Style Sheets, and Javascript. These resources are served using Tornado [25] with server-side handlers written in the Python programming language. Data are stored in a relational database [26]. To facilitate deployment, security, and cross-platform compatibility, Docker containerization is used [27].

To facilitate secure code execution, CodeBuddy uses a three-layer system. First, for a given exercise, the front end (Tornado server) identifies which "back end" (associated with a particular programming language) to use. The front end submits a request to a "middle layer" Python server, which uses the FastAPI framework [28]. This request includes the student's (or instructor's) code, test code, verification code, data file(s), configuration settings, and back-end information. The middle layer then instantiates a Docker container; uses a volume to share the code and data files with the container; and executes a command to run the code within the container. To prevent attacks, the container is allowed minimal security privileges; furthermmore, the system administrator can configure the amount of random access memory and number of CPU processes that can be used; these settings can be different for each back end. As the container executes the code, it redirects outputs to file(s), which the middle layer parses. These outputs are returned to the front end, compared against the expected outputs, and returned to the student.

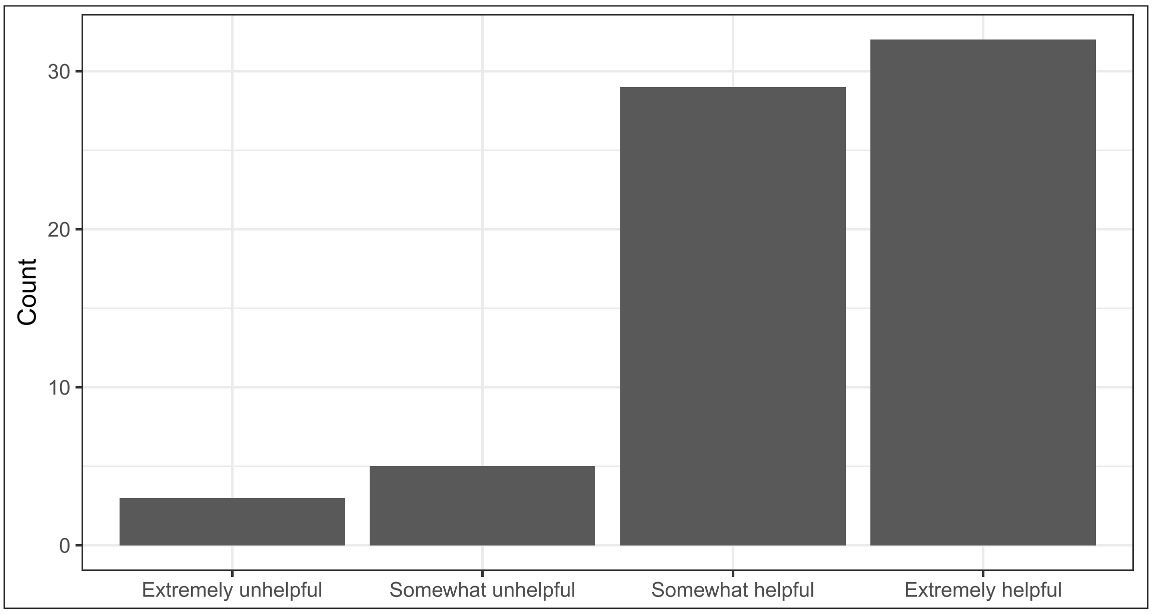


Figure 7 Student survey responses regarding the software's general utility. We surveyed students to assess their perceptions of the software's utility on a Likert scale.

Students edit code using the Ajax.org Cloud9 Editor (Ace) [29]. Ace provides basic code completion and formatting help. When a student's outputs are similar to the expected outputs but do not match exactly, a "diff" summary is generated. For text-based outputs, Ace displays a side-by-side comparison in a separate browser tab. For image-based outputs, the Pillow package [30] generates an image with highlighted differences; this image is displayed in a separate tab.

Additional programming languages can be supported via the following steps. For a given language, the administrator provides a Dockerfile specifying instructions for building a Linux-based, operating-system environment with software to compile (if necessary) and execute code in the respective language. A separate configuration file indicates how much memory usage to allow, whether text- and/or graphics-based outputs are supported, and code examples. Additionally, script(s) are provided for compiling (if necessary) the code, executing the code, and tidying the outputs (e.g., removing common warning or informational messages).

CodeBuddy stores data using *SQLite*–a serverless, embedded database engine. Accordingly, the database can be backed up and shared easily. To date, we have run the system with hundreds of students enrolled simultaneously. We are unsure of its ability to scale to more simultaneous users. Using a client-server database management system could increase its ability to scale. Another option is to run multiple instances of the server, each with a different database file.

Features supported by other tools that are not currently supported in CodeBuddy include A) CPU or memory profiling, B) uploading source files, C) command-line execution, D) integration with source-code repositories, E) automatic checks against code-style guidelines, F) plagiarism detection, G) integration with other learning management systems. The lack of some of these features is by design; other features may be implemented in future versions.

QUALITY CONTROL

The README file in the source-code repository provides instructions for installing and executing the software (https://github.com/srp33/CodeBuddy/blob/master/README.md). Additionally, we provide a data file that can be used to import an example assignment (https://github.com/srp33/CodeBuddy/blob/master/examples/Example_assignment.json). This assignment provides exercises that use diverse back ends, output types, and configuration options.

The source code includes an automated script that performs basic unit testing (https://github.com/srp33/CodeBuddy/blob/master/front_end/tests/run.sh). The script generates a database, starts a web-server instance, and submits HTTP requests to the server. The tests ensure that the most frequently accessed pages are rendered without error and that specific keyphrases are present in the rendered output.

AVAILABILITY

OPERATING SYSTEM

The front-end server is executed within a Docker container; therefore, it can be executed on any operating system that supports the Docker execution engine. The middle-layer server is *not* containerized (to avoid limitations with between-container communication). This server must be executed on a Linux operating system; additionally, the Docker execution engine must be installed on the server because the middle-layer server executes code within Docker containers.

PROGRAMMING LANGUAGE

Server logic is written in Python (version 3.9+). Client-side code is written in Javascript and TypeScript. Shell commands build and run Docker containers.

ADDITIONAL SYSTEM REQUIREMENTS

When running idle, the Python servers use little memory and few CPU cycles. When users access the site and complete tasks other than executing code, the server uses relatively few resources. However, as course content and the number of submissions grows, the database scales accordingly. At our university, the database is currently a few gigabytes in size. The Python servers execute processes in parallel, and the administrator can configure how many processes are allowed. Additionally, the administrator can configure limits on the amount of memory that each Docker container can use when code is being executed; the default is 500 megabytes.

DEPENDENCIES

Docker execution engine.

LIST OF CONTRIBUTORS

All individuals who contributed to the software are listed as authors. Each author made a substantive contribution.

SOFTWARE LOCATION

Archive

Name: Zenodo
Persistent identifier: DOI: 10.5281/zenodo.13250409
License: GNU Affero General Public License v3.0
Publisher: Stephen R. Piccolo
Version published: 55
Date published: 06/08/2024

Code repository

Name: GitHub
Identifier: https://github.com/srp33/CodeBuddy
License: GNU Affero General Public License v3.0
Date published: 06/08/2024

LANGUAGE

English

REUSE POTENTIAL

CodeBuddy has been used at Brigham Young University since 2019. Instructors from other higher-education institutions, primary or secondary schools, industry, and informal settings may find it useful for delivering short-form programming exercises.

An additional use case is the delivery of written or video-based content. In some of our courses, we use CodeBuddy to deliver video-based lectures and ask students to provide responses to each lecture segment. Students have provided informal feedback that this interactivity helps them to remain engaged with lecture material.

Because each student is assigned to an "A" or "B" cohort, CodeBuddy can be used for pedagogical research.

The website footer includes a link to a "Contact us" page, which provides details about support mechanisms. Support is provided through the Issues page on our GitHub site (for developers) and an online forum (administrators running their own instance of the software).

FUNDING INFORMATION

The authors received internal funds from Brigham Young University to pay for a portion of this work.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR CONTRIBUTIONS

All authors contributed to writing the software. SRP wrote the paper.

AUTHOR AFFILIATIONS

Stephen R. Piccolo
Department of Biology, Brigham Young University, Provo, UT, USA

Emme Tuft
Department of Biology, Brigham Young University, Provo, UT, USA

P. J. Tatlow
Department of Biology, Brigham Young University, Provo, UT, USA

Zach Eliason
Department of Biology, Brigham Young University, Provo, UT, USA

Ashlie Stephenson
Department of Biology, Brigham Young University, Provo, UT, USA

REFERENCES

1. **Blanchard J,** *et al.* Stop Reinventing the Wheel! Promoting Community Software in Computing Education. In: *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* 261–292 (Association for Computing Machinery, New York, NY, USA, 2022). DOI: https://doi.org/10.1145/3571785.3574129
2. **Paiva JC, Leal, JP, Figueira Á.** Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.* 2022; 22: 34: 1–34:40. DOI: https://doi.org/10.1145/3513140
3. **Code Autograding Platform Codequiry.** Available from: https://codequiry.com/auto-grading-programming.
4. **CodeGrade – Virtual Assistant for your coding classroom.** Available from: https://www.codegrade.com.
5. **Codio – The Hands-On Platform for Computing & Tech Skills Education.** Available from: https://www.codio.com.
6. **Coding Rooms – Developer training and enablement.** Available from: https://www.codingrooms.com.
7. **Edwards SH, Perez-Quinones MA.** Web-CAT: Automatically grading programming assignments. In: *Proceedings of the 13th annual conference on Innovation and technology in computer science education.* 2008; 328–328. DOI: https://doi.org/10.1145/1384271.1384371
8. **Peveler M, Tyler J, Breese S, Cutler B, Milanova A.** Submitty: An Open Source, Highly-Configurable Platform for Grading of Programming Assignments (Abstract Only). In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* 641 (Association for Computing Machinery, New York, NY, USA, 2017). DOI: https://doi.org/10.1145/3017680.3022384
9. **Agrawal A, Jain A, Reed B.** CodEval: Improving Student Success In Programming Assignments. in 7546–7554; 2022. DOI: https://doi.org/10.21125/edulearn.2022.1767
10. **Cipriano BP, Fachada N, Alves P.** Drop Project: An automatic assessment tool for programming assignments. *SoftwareX.* 2022; 18: 101079. DOI: https://doi.org/10.1016/j.softx.2022.101079
11. **Leal JP, Silva F.** Mooshak: A Web-based multi-site programming contest system. *Software: Practice and Experience.* 2003; 33: 567–581. DOI: https://doi.org/10.1002/spe.522
12. **Wünsche BC,** *et al.* Automatic assessment of OpenGL computer graphics assignments. In: *Proceedings of the 23rd annual ACM conference on innovation and technology in computer science education.* 2018; 81–86. DOI: https://doi.org/10.1145/3197091.3197112
13. **Edwards SH, Murali KP.** CodeWorkout: Short programming exercises with built-in data collection. In: *Proceedings of*

the 2017 ACM conference on innovation and technology in computer science education. 2017; 188–193. DOI: https://doi.org/10.1145/3059009.3059055

14. **Staubitz T, Klement H, Teusner R, Renz J, Meinel C.** CodeOcean-A versatile platform for practical programming exercises in online environments. in *2016 IEEE global engineering education conference (EDUCON)* 314–323 (IEEE, 2016). DOI: https://doi.org/10.1109/EDUCON.2016.7474573
15. **Petit J,** *et al.* Jutge.Org: Characteristics and experiences. *IEEE Transactions on Learning Technologies.* 2017; 11: 321–333. DOI: https://doi.org/10.1109/TLT.2017.2723389
16. **Mekterović I, Brkić L, Milašinović B, Baranović M.** Building a comprehensive automated programming assessment system. *IEEE access.* 2020; 8: 81154–81172. DOI: https://doi.org/10.1109/ACCESS.2020.2990980
17. **Fabijan A, Dmitriev P, Holmstrom Olsson H, Bosch J.** Online Controlled Experimentation at Scale: An Empirical Survey on the Current State of A/B Testing. in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).* 2018; 68–72. DOI: https://doi.org/10.1109/SEAA.2018.00021
18. **OpenAI Platform.** Available from: https://openai.com.
19. **Daring Fireball: Markdown.** Available from: https://daringfireball.net/projects/markdown.
20. **Lamport L.** *I1 (\LaTeX)—A Document.* vol. 410 (pub-AW, 1985).
21. **Waskom ML.** Seaborn: Statistical data visualization. *Journal of Open Source Software.* 2021; 6: 3021. DOI: https://doi.org/10.21105/joss.03021
22. **Hunter JD.** Matplotlib: A 2D graphics environment. *Computing in Science & Engineering.* 2007; 9: 90–95. DOI: https://doi.org/10.1109/MCSE.2007.55
23. **Wickham H,** *et al.* Welcome to the tidyverse. *Journal of Open Source Software.* 2019; 4: 1686. DOI: https://doi.org/10.21105/joss.01686
24. **Hawlitschek A, Berndt S, Schulz S.** Empirical research on pair programming in higher education: A literature review. *Computer Science Education.* 2022; 0: 1–29.
25. Tornado Web Server — Tornado 6.4 documentation. Available from: https://www.tornadoweb.org.
26. **SQLite Home Page.** Available from: https://www.sqlite.org.
27. **Docker.** *Docker*. Available from: https://www.docker.com.
28. **FastAPI.** Available from: https://fastapi.tiangolo.com.
29. **Ace – The High Performance Code Editor for the Web.** Available from: https://ace.c9.io.
30. **Pillow.** PyPI. Available from: https://pypi.org/project/Pillow.

TO CITE THIS ARTICLE:
Piccolo SR, Tuft E, Tatlow PJ, Eliason Z, Stephenson A. 2025 CodeBuddy: A Programming Assignment Management System for Short-Form Exercises. *Journal of Open Research Software,* 13: 1. DOI: https://doi.org/10.5334/jors.503

Submitted: 01 January 2024 **Accepted:** 15 January 2025 **Published:** 27 January 2025