Smart Learning Environments

RESEARCH

Open Access

Lowering novice programming barriers with Medialib: studies of its effectiveness and transferability

Jingyun Wang[1*], Adam Wynn[1], Andrea Valente[2], Daner Sun[3] and Emanuela Marchetti[4]

*Correspondence:
jingyun.wang@durham.ac.uk

[1] Department of Computer Science, Durham University, Durham, UK
[2] Department of Design, Media and Educational Science, University of Southern Denmark, Odense, Denmark
[3] Department of Mathematics and Information Technology, The Education University of Hong Kong, Hong Kong, China
[4] Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Abstract

Multimedia has been recognised as a powerful domain for contextualising programming concepts. However, the inherent complexity of multimedia applications, particularly their reliance on advanced data structures, often poses significant challenges for novice programmers. To address this issue, we implemented Medialib, a user-friendly Python multimedia library specifically designed for beginners at or above the high school level. Medialib was developed through an iterative process informed by empirical studies involving non-technical university students and their instructors, with the goal of making multimedia programming more accessible to learners without prior technical backgrounds. This paper introduces, for the first time, a simplified multimedia Python library and accompanying pedagogical materials tailored to the cognitive and instructional needs of novice programmers. Medialib enables a pedagogical shift in introductory Python courses from traditional mathematics-oriented exercises to multimedia-focused tasks. To evaluate its effectiveness and transferability, two empirical studies were conducted: a 14-week study in Japan (21 instructional hours, 36 students) and a 2-week study in the UK (12 instructional hours, 84 students). Analyses are done on the study data which includes teacher observational notes, questionnaires, and interviews. Specifically, a comparison of the weekly performance of learners in traditional maths-related exercises and Medialib-related exercises in the first study is discussed. Findings from both studies indicate that learners responded positively to the Medialib materials. Notably, in the Japanese study, students who initially struggled with maths-related programming tasks were able to successfully acquire foundational programming skills through Medialib activities. From week 7 onward, students consistently demonstrated strong performance in both types of exercises, suggesting that Medialib serves not only as an effective entry point for programming education but also as a transferable learning scaffold across contexts.

Keywords: Multimedia programming simplification, Beginner's python course, Programming skill, Notional machine, Python library for beginners

Introduction

Computational Thinking (CT) has the ultimate goal of enabling learners to acquire fundamental skills, such as problem-solving and higher-order thinking skills, to solve real-world problems by means of digital technologies (Lai, 2021). Programming skills

represent core desired competencies in CT and effectively teaching these skills to non-technical novices is one of the hardest challenges in the education field (Barr & Stephenson, 2011). Identifying and solving problems in programming involves two essential skills: abstraction, where irrelevant details are removed to focus on the essential ones, and decomposition where a complex problem is split into simpler subproblems to make the solution easier (Lodi, 2020). However, due to a lack of training in abstraction and decomposition, novice learners normally find text-based programming frustrating.

Although no single pedagogical approach to teach programming is effective for all learners and the same learning outcomes can be achieved through a variety of media, the medium used to teach the content can make a difference (Rich et al., 2022). The analysis results in our previous work (Valente et al., 2020) suggest that most of the novice undergraduate learners struggled with traditional Python exercises that require maths content as a prerequisite as adding programming logic on top of maths logic exacerbates the challenge of understanding the decomposition process for those who already struggle with maths. Therefore, a novel approach that moves away from traditional bottom-up approaches based on maths-related problems and focuses more on programming itself was needed to structure beginner's Python courses.

Current trends in teaching programming to students outside of the Computer Science domain demand for exercises and activities that leverage learners' preunderstanding of common interactive applications other than traditional exercises based on mathematics (Geist, 2016; Peppler & Kafai, 2005). Previous research suggests that multimedia is a motivating domain that helps ground programming concepts for beginners as it allows learners to gain an intuitive understanding of their code by viewing outputs visually whilst exploring how to display and compose images, facilitating abstraction and decomposition (Agbo et al., 2019; Fagerlund et al., 2020). However, existing multimedia-based programming tools have limitations. Block-based tools such as Scratch are less suitable for beginners because students develop undesirable programming habits including incorrect usages of programming structures (Weintrop & Wilensky, 2015) that do not transfer effectively to text-based programming (Weintrop & Wilensky, 2019). Moreover, existing multimedia libraries such as Pygame require complex data structures and asynchronous event handling, making them inaccessible to beginners.

To address these challenges, Medialib, a new beginner-friendly multimedia Python library has been designed to simplify multimedia programming. We aim to create interactive and engaging experiences by incorporating elements from other fields including game implementation to help learners develop a more versatile skill set. Furthermore, a redesign of the entire structure of an introductory Python programming course is proposed, focusing on the learner's need for incremental and logically motivated content, instead of the classic bottom-up approach based on maths-related problems. In this work, we specifically focus on paving the road for learners towards the basics of programming and coding, and in particular reading and writing code to make a program, which is the most challenging issue within programming education (Valente et al., 2020). Our research questions are as follows:

1. How does Medialib impact novice learners' ability to acquire programming skills compared to traditional mathematics-based programming tasks?

2. How do students engage with Medialib across different educational settings, and what factors influence its effectiveness in diverse learning environments?
3. What design considerations are essential for developing a beginner-friendly multimedia programming library and exercises that facilitate effective learning and engagement?

To investigate the above research questions, two successive studies were conducted on two groups of non-technical university students taught by the same teacher (the first author). Based on the feedback of the participants in the first study using Medialib 1.0, Medialib was updated to 2.0, and its exercises were also amended to better support the learning activities in the second study. The learner data of both studies are analysed and discussed in this paper. Meanwhile, Medialib was updated to 3.0 in response to the analysis results of the second study.

The contribution of this paper can be summarized as follows: (1) we analyse existing literature in simplifying programming for non-technical beginners, and explore the notion of notional machines from the perspective of programming education support. (2) By elaborately examining existing teaching materials, especially the two widely adopted Python multimedia libraries (Pygame and Pygame Zero) and their examples, we determine the instruction of Medialib, and then simplify them to eliminate confusion of Python novice programmers when transitioning to working without Medialib. Medialib prioritises programming logic over computing principles and is designed to integrate into any Python IDE alongside other libraries. (3) We design Medialib exercises which facilitate the editing of the presentation of images and sounds, the identification of code that fails, and then the debugging of that code. (4) We update Medialib and its materials iteratively starting from experimental results involving non-technical students and their teacher. In other words, for the first time a cut-down version of a multimedia Python library and its teaching materials are designed to support beginners for pedagogical purposes. We propose a new teaching approach different from traditional approaches which either use standard maths-based exercises that narrowly focus on abstract concepts or use block-based tools that are not suitable for skill transfer when solving real-world problems, and different from existing multimedia libraries and their examples that require complex data structures.

Related work

Block-based and text-based tools in programming education

Block-based tools such as Scratch, Blockly (https://developers.google.com/blockly), and App Inventor (https://appinventor.mit.edu), have been claimed to help students build knowledge about programming concepts, relieving learners from the burden of writing code on their own, a known challenge in the domain of programming education (Fincher et al., 2020). When coding in block-based tools, learners can rely upon the recognition of ready-to-use coding blocks where each type of block has a unique shape to make compatible blocks easier to see and prevent incompatible blocks from being combined, reducing cognitive load by preventing syntax errors, making programming more accessible (Geist, 2016; Weintrop & Wilensky, 2019). However, whilst block-based programming enhances logical reasoning, research suggests that learners struggle to

transfer these skills to text-based languages like Python and Java (Pedersen et al., 2020; Weintrop & Wilensky, 2019).

Due to uncertainty concerning the effectiveness of block-based tools beyond introductory contexts, another approach for supporting beginner programmers is to create text-based tools. Compared to block-based, text-based approaches are more powerful, less verbose and allow the learner to be more creative as they are less limited by the restrictions of using blocks (Weintrop & Wilensky, 2015). However, unlike block-based tools, with text-based tools, learners need to memorise commands and are unable to test out blocks, incorporate them, and see if they work using trial and error (Weintrop & Wilensky, 2019).

To bridge the gap between block-based and text-based tools, bi-directional environments such as BlockPy and Blockly (https://developers.google.com/blockly), allow learners to switch between both views. More specifically, BlockPy is a Python environment designed to help data science students with the transition to textual programming languages and supports Matplotlib (Hunter, 2007). Whilst students valued their experience with BlockPy, only 15% of learners used the dual-functionality feature and most students observed the code as it changed rather than actively writing new textual code (Bart et al., 2017). Alternatively, dual-modality environments combining features of block-based and text-based into a single interface (Weintrop & Wilensky, 2019) aim to provide a scaled-down view of existing programming languages. The advantage of this approach is to enable learners to train, from the beginning, with the same existing languages that they are expected to learn for professional use.

Scaled-down versions of existing programming languages

Beyond block-based and text-based tools lies a third approach: scaled-down programming languages that preserve key concepts from full programming languages while reducing complexity. Systems such as Blockly (https://developers.google.com/blockly) and P5 (https://p5js.org/) offer a cut-down view on JavaScript, targeted specifically at creative programming for graphics, animations and games, similar to block-based tools. Processing (https://processing.org/) also provides cut-down views on Java. The advanced programmer can still use the full power of the respective languages, but beginners can focus on a reduced set of instructions and concepts.

Guzdial (2022) defines "teaspoon languages" as task-specific programming languages that are both highly usable and rapidly learnable (Geist, 2016; Peppler & Kafai, 2005). They are specifically designed to integrate seamlessly into the learning activities of non-CS teachers, providing a manageable "teaspoon" of computational content within other subject areas.

Research gap

While cut-down systems and the teaspoon approach represent a growing trend in programming education, to the best of our knowledge, cut-down versions of multimedia Python libraries for beginners do not exist in the current literature. This absence highlights a significant research gap: Python educators currently lack transitional, text-based multimedia scaffolds that bridge the divide between introductory tools and professional libraries. Furthermore, while cut-down versions exist for other languages (e.g., P5 and

Processing), they often rely on customizable editors. This creates a secondary issue: these custom IDEs can inadvertently expose learners to advanced, confusing error messages from the underlying compiler when hidden features are referenced (Krishnamurthi & Fisler, 2019).

This study extends the current literature by considering these exact design issues in the development of Medialib (discussed in the next section). The rationale for our research questions directly stems from the need to empirically validate this novel approach. Specifically, we investigate how this simplified tool impacts the acquisition of foundational programming skills compared to traditional methods (RQ1). We further examine learner engagement across distinct educational contexts to ensure the findings are robust (RQ2), and we analyse these implementations to extract reusable design principles for future educational technologies (RQ3).

Medialib and its related exercises: design principles and features

Conceptual foundation: notational machines for novice programmers

The design of Medialib is grounded in the concept of notional machines, which serve as simplified mental models to help learners visualise and reason about program execution (Berry & Kölling, 2013; Sorva, 2013). These conceptual models are particularly valuable in novice programming instruction, as they make abstract computational processes more tangible and comprehensible (Duran, 2019). Duran introduced the idea of incremental notional machines, progressively tailoring the level of abstraction and vocabulary to suit different learner groups, from elementary school students to first-year computer science majors, based on their prior knowledge.

Following this approach, Medialib was developed to explicitly embody a notional machine suitable for non-technical novices. It incorporates visual media as a familiar context, reducing cognitive complexity while supporting interactive program execution. The simplified Python environment enables learners to manipulate images and sounds, identify and correct syntax errors, and understand debugging procedures through immediate visual feedback.

Simplifying Python for multimedia learning

Medialib is implemented as a simplified, imperative subset of Python that retains core programming constructs such as numeric and string operations, conditionals, and loops. The notional machine adopted assumes visual media as learners' default mental model and is extended to support multimedia applications while maintaining low complexity (Fig. 1). To determine the appropriate instructional scope, the development process analysed beginner-level multimedia projects built using Pygame and Pygame Zero, two widely used Python multimedia libraries.

An analysis of the Pygame exercises and solutions showed that a beginner programmer would need to understand several complex concepts: (1) importing Python modules, which requires prior knowledge about namespaces and scope of variables across modules; (2) the use of object oriented design including dot notation and dot paths which requires an understanding of how functions can be accessed across modules; (3) data structures such as tuples, objects and classes which are unavoidable even with simple Pygame examples; (4) complexity in managing graphics buffers and surface

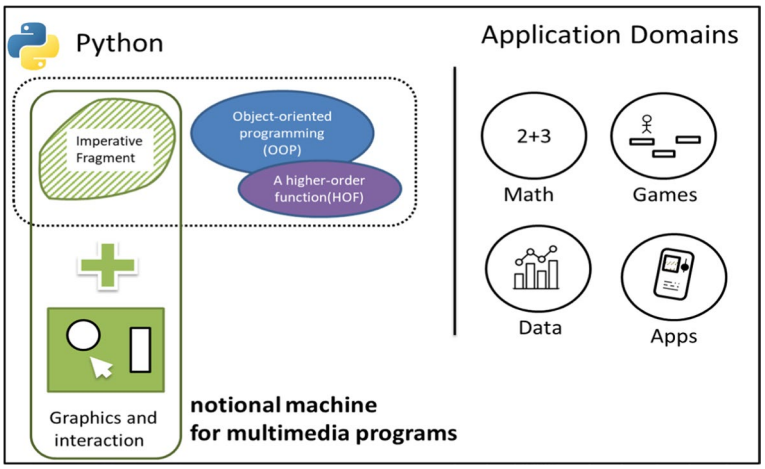

Fig. 1 The relation between Python and our notional machine for multimedia programs

objects which are needed just for drawing images on screen; and (5) event queues and polling which are usually required to understand user input. Although Pygame Zero was designed with pedagogical goals in mind, it still demands familiarity with advanced concepts such as function definitions, functional programming (e.g., callbacks), and concurrency, posing substantial challenges to beginners.

Core design principles of Medialib

Medialib is pedagogically positioned to bridge the gap between block-based environments and professional multimedia libraries. Unlike block-based tools such as Scratch, which as discussed in Related work Section can limit the transfer of skills to text-based languages, Medialib requires learners to engage directly with Python syntax, providing experience with textual coding from the start. Furthermore, unlike Pygame Zero, which requires an event-driven approach involving complex callbacks and concurrency, Medialib employs a strictly sequential execution model. This straightforward design allows learners to comfortably understand core programming logic without being introduced to the cognitive overhead of asynchronous programming.

In other words, in response to the limitations of Scratch and Pygame Zero, Medialib was built with the goal of maximizing usability and learnability for novice users, while preserving sufficient expressive power for meaningful multimedia programming. The following design principles were adopted:

- Sequential execution model, relying on busy-wait and polling rather than asynchronous event handling, to align with the input–process–output logic familiar to beginners (Valente et al., 2020)
- Use of basic data types only, such as integers and strings, avoiding object-oriented structures in early stages

- Exclusion of event-driven programming, allowing learners to postpone the cognitive demands of functions and higher-order logic
- Graceful degradation, or "fail gently" mechanisms to provide a "robust" behaviour for the instructions. For instance, drawing an unavailable image triggers a placeholder display rather than a crash
- Support for a use–modify–create pedagogy, enabling students to begin with functional code examples, understand through execution, and gradually progress toward creative modification and independent coding

In essence, Python is first reduced to a core fragment, keeping just the imperative features of the language, such as sequences, primitive datatypes, choice and iteration, then extend this cut-down portion of Python to multimedia by adding a minimal set of instructions for drawing images and simple shapes, controlling timing, and user interaction. Beginners can start using the Python fragment and Medialib, while more advanced learners can import domain-specific libraries to further expand the expressive power of the imperative fragment.

Functionality overview and progressive exercise design

The current version of Medialib (v3.0) includes 11 void functions (which do not return values) and 8 fruitful functions (which return values). Among them, initialize() and all_done() serve to start and close the Medialib runtime environment, particularly in Jupyter notebooks where standard Pygame initialization may not function properly. The remaining functions are listed in Table 2 of the Appendix. These were iteratively tested and refined over a three-year development cycle, with several newly added functions introduced based on empirical classroom feedback.

Here is an example Medialib exercise designed to practice how to use strings: *"Try to draw a game background using strings as a parameter to a function"*. A figure is provided (as shown in the left of Fig. 2) to show what multimedia effect is required,

Fig. 2 Example of a Medialib exercise illustrating the visual target (left) and the corresponding Python template provided to the learner (right)

facilitating its transformation into programming logic. In addition, the learners are encouraged to use **while** and **if** to program based on a provided py/ipynb file (as shown in the right part of Fig. 2) and some png files. This is different from traditional maths-based exercises which require learners to code towards a maths solution.

To support progressive skill acquisition, Medialib exercises are carefully sequenced. Compared to regular maths-based exercises, Medialib exercises are designed to facilitate basic programming knowledge to be acquired incrementally without requiring in-depth maths knowledge. The first exercise introduces the draw() and clear() functions along with general programming knowledge such as variables and functions. In later exercises, iterative and conditional statements are introduced and then combined with knowledge from previous exercises, with the aim of creating animations. Future exercises can be developed with basic gameplay in mind. For example, users can read data from a file to create a game's background and write the game's performance into log files using Medialib and Pandas (McKinney, 2010).

Case studies

To evaluate the design and pedagogical value of Medialib and its exercises, two empirical studies were conducted across different institutional and cultural contexts. **Study 1** was carried out in a Japanese university with a cohort of 36 international undergraduate students, predominantly from non-technical majors. It aimed to explore how Medialib could support learners with limited programming experience in acquiring fundamental Python skills, especially in comparison to traditional mathematics-based tasks. **Study 2**, conducted at a UK university with 84 Master's students in Business Analysis, built upon the findings of Study 1 by implementing a revised version of Medialib (v2.0) and improved teaching materials. It further examined the system's adaptability and learner engagement in a more intensive, short-term instructional setting. The design of Study 2 was informed by feedback from Study 1, particularly regarding the need for clearer learning objectives, better scaffolding, and improved compatibility with different development environments. Together, the two studies provided complementary insights into the effectiveness, usability, and transferability of Medialib for novice programmers in varied educational settings.

Study 1: evaluating the effectiveness of Medialib in a Japanese University context

Participants

Study 1 was conducted as part of an "Introduction to Computer Science" course at a Japanese university in 2020, targeting a cohort of 36 international undergraduate students (aged 18–21) from a variety of academic disciplines, as shown in Fig. 3. Most students were in their first or second year of undergraduate study. The class included students from diverse national backgrounds, primarily from Asia (Indonesia: 14; China: 7; Thailand: 5; South Korea: 2; India: 1; Japan: 1; Malaysia: 1; Philippines: 1; Vietnam: 1), with two from the United States and one from Guatemala. Over half of the participants were not enrolled in engineering or computer science-related majors.

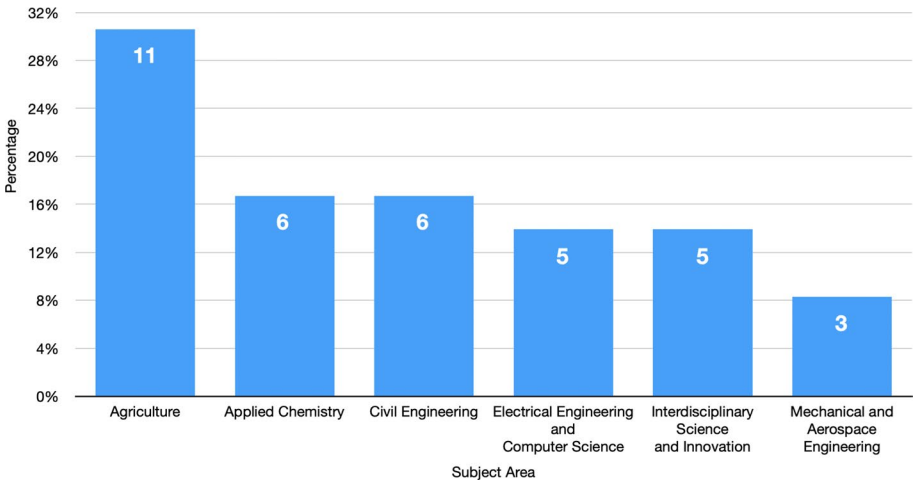

Fig. 3 Distribution of participants' undergraduate majors for Study 1

Data collection and instructional design

The study was conducted over 14 weeks (May to July 2020) and was delivered in a flipped classroom format due to the COVID-19 pandemic. Students engaged with 14 pre-recorded video lectures (each 30–40 min) introducing Python concepts based on the textbook Think Python (Rus, 2016). Topics included (1) Introduction to programming, (2) variables, expressions and statements, (3) functions, (4) comment and debugging, (5) conditionals and recursion, (6) fruitful functions, (7) while loops, (8) strings and for statements, (9) lists, (10) dictionaries, (11) tuples, (12) for loop, (13) files, and (14) matrices.

Weekly 90-minute synchronous sessions were used for coding practice and discussions. All programming exercises were completed using the Mu editor (https://codewith.mu), which includes the Pygame environment by default. At the end of each lecture, students were assigned two sets of exercises: 1) A traditional mathematics-based problem set (with easy and medium difficulty levels), and 2) A Medialib-based task (one per week, adjusted to match the medium difficulty level).

From weeks 5 to 12, students were required to submit both types of exercises weekly. Submissions were graded on a 100-point scale, and formative feedback was provided prior to the next lecture. In addition to these weekly assessments, participants were required to complete a background questionnaire at the first lecture and a learning perception questionnaire at the last lecture. All 36 students completed the first questionnaire, and 34 responded to the second (41% female). Moreover, one male and four female students participated in post-course interviews.

Results

1) Weekly performance comparison between Medialib and traditional exercises

An analysis of students' weekly performance revealed a strong alignment between scores of Medialib-based and traditional mathematics-based programming exercises. As shown in Table 1, the **Pearson correlation coefficients** indicate significant positive

Table 1 The Pearson correlation between the weekly scores for regular and Medialib exercises in Study 1

Week	Target	Content	Function Involved	r (Pearson)	p
5	Function	Rewrite a given code by defining a function to enhance the readability of the code	*rect()*	0.565	3.26e−04*
6	If and while	Control an avatar based on different input	*play(), draw(), wait_key_press()*	0.783	1.61e−08**
7	If, for and string	Display a game background (as shown in Fig.2) based on given strings and given figures	*draw()*	0.739	2.70e−07**
8	For and list	Display the values of each item of a given list in a bar chart	*rect()*	0.784	1.52e−08**
9	If, for and dictionary	Create a contact book with names and photos that allow searching by initial letter of a person	*draw(), wait_key_press()*	0.543	6.28e−04*
10	If, for, tuple, dictionary	Create a contact book with names, photos and telephone number that allow searching by telephone number	*draw()*	0.271	0.110
11	For...range	Write a function to create an animation which draws a rectangle from big to small	*rect(), wait()*	0.690	3.19e−06**
12	open(), read() and write()	Modify based on week 8 code, read from and write to a given file instead of reading from and write to a given list	*rect(), wait_mouse_leftclick()*	0.826	5.57e−10**

*$p < .001$, **$p < .00001$

relationships between the two sets of tasks across most weeks. Notably, in weeks 6, 7, 8, 11, and 12, the correlation was particularly strong, suggesting that Medialib exercises, despite their different instructional design, served as effective alternatives to regular exercises of medium difficulty.

This consistency in performance underscores the pedagogical validity of the Medialib tasks. Designed to scaffold learners' understanding incrementally, the exercises (shown in table 1) encourage learners to integrate a minimal set of Medialib functions (typically one to three), in addition to initialize(), clear(), wait_mouse_leftclick(), and all_done(), alongside standard Python syntax. This approach allowed students to focus on core programming structures without being overwhelmed by complex syntax or abstract logic.

An exception was observed in week 10, where the Medialib task was unintentionally more complex than its traditional counterpart. In this instance, student performance on the Medialib task diverged from the regular task, indicating that task design complexity remains a critical factor in maintaining equivalence across modalities. Nonetheless, the overall alignment of scores across the majority of weeks suggests that Medialib can effectively function as a pedagogically comparable alternative to conventional exercises, particularly when task difficulty is carefully calibrated.

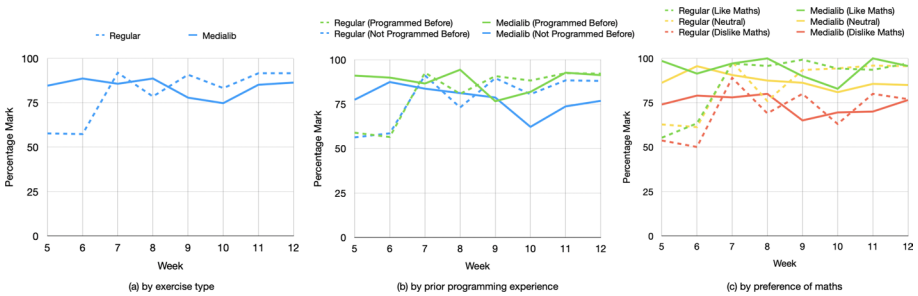

Fig. 4 Comparison of weekly performance scores between regular exercises and Medialib exercises in Study 1

2) Differential performance patterns and learner characteristics

Figure 4 provides a week-by-week comparison of student performance on Medialib and regular mathematics-based programming exercises. As shown in Fig. 4a, students demonstrated consistently stable performance on Medialib tasks throughout the 14-week course. In contrast, their performance on regular exercises was substantially lower during weeks 5 and 6, when abstract programming concepts such as functions, conditionals, and loops were first introduced. Independent t-tests confirmed significant performance differences favouring Medialib exercises in week 5 ($t(36) = -4.682$, $p < .0001$) and week 6 ($t(36) = -6.482$, $p < .00001$), while no significant differences were observed in later weeks. This pattern suggests that Medialib helped learners overcome early conceptual barriers, likely by reducing the cognitive load associated with mathematical abstraction and enabling a smoother entry point into programming logic. Instructor reflections from previous cohorts without Medialib support revealed that early failures on regular tasks often led to decreased motivation and stagnation in learning progression. In contrast, students in this study were able to build programming confidence through the Medialib exercises and subsequently improve their performance on both types of tasks after week 7. This interpretation is reinforced by student feedback, such as: *"My maths logic isn't as good as the others... but I think programming can be fun using Medialib"*, and *"I may be good at programming logic but definitely not good with mathematical concepts... I'm happy I don't need to suffer from this issue in the Medialib exercises."*

Figure 4b further examines performance in relation to prior programming experience. Of the 36 participants, 6 (16%) had previously learned Python, 12 (33%) had experience with other programming languages, and 18 (50%) had no prior exposure to programming. While students with previous experience exhibited slightly higher average scores each week, independent t-tests revealed no significant differences in performance across either exercise type. Similarly, analysis of the final assignment, a mini-game built using Medialib, showed no significant differences between students with and without prior programming experience ($t(18) = -1.164$, $p > .05$), despite a noticeable mean gap (11.9 points). This suggests that Medialib may have supported the learning progression of novices, allowing them to achieve comparable outcomes. The high average score among students with no prior experience (Mean = 72.5)

further supports the accessibility and scaffolding capacity of the Medialib exercise framework.

Student attitudes toward mathematics still strongly influenced learning outcomes, as shown in Fig. 4c. Students who expressed a positive or neutral disposition toward mathematics consistently outperformed those who reported disliking it. In week 6, students with negative attitudes toward math (Regular: Mean = 50, S.D. = 2.71; Medialib: Mean = 79, S.D. = 4.18) scored substantially below the class average (Regular: Mean = 57.2, S.D. = 1.64; Medialib: Mean = 88.6, S.D. = 2.49) in both exercise types. Two students from this group performed especially poorly and ultimately dropped the course without submitting the final assignment. However, for those who persisted, performance improved significantly after week 7, particularly in the Medialib tasks. This suggests that while maths-related anxiety can hinder early programming success, the visual and creative nature of Medialib exercises may help alleviate these challenges, enabling students to gain confidence and progress in their learning.

3) Student perceptions of course design and learning challenges

Feedback from the second questionnaire revealed varied student perceptions of the course design. Approximately half of the participants reported that the course met their expectations, while 12 students (35%) perceived the course as too difficult, and four (12%) felt the pace was too fast. Several students noted challenges in adapting to the flipped learning model with pre-recorded videos, particularly in the context of remote learning during the pandemic. Difficulties in peer interaction and collaborative discussion were also raised. Moreover, some participants expressed frustration with the mathematics-based exercises, citing unclear problem definitions that distracted from programming logic. Although students generally understood Python syntax, many reported difficulties in applying it effectively to solve programming problems, indicating a need for more structured guidance during early stages of learning.

4) Attitudes toward medialib: usefulness, motivation, and cognitive support

Student responses to open-ended questions about Medialib, visualised in keyword frequency in Fig. 5, were overwhelmingly positive. Over 67% of participants

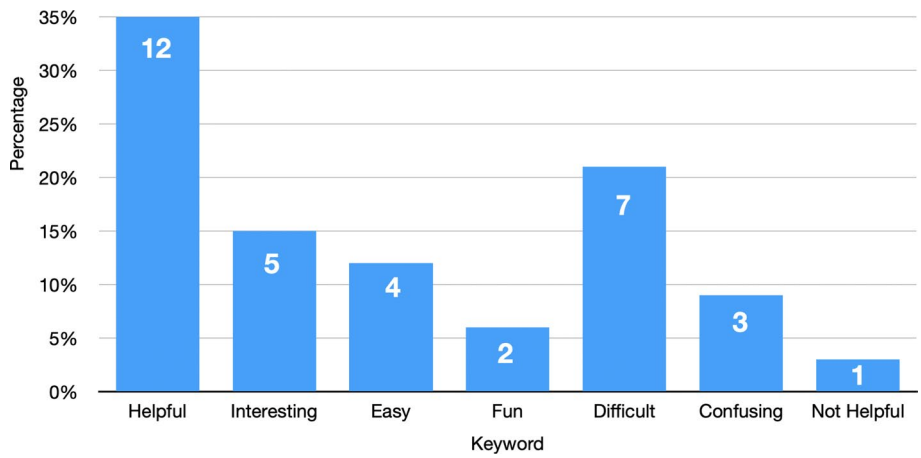

Fig. 5 Frequency of keywords describing how participants felt about Medialib in Study 1

described their experience as positive, and 12 students (35%) explicitly stated that Medialib improved their understanding of programming functions. One student wrote: *"Medialib is really useful because I can just look at the library to see what functions are available if I'm stuck with the exercise. I found the Medialib exercises more fun than the exercises from the textbook. It's nice to see the output of my programming with images and colours rather than just text."* Another commented: *"I think it's pretty interesting learning the basics of how to create a game... The Medialib tasks are usually easier to understand compared to the normal exercises."*.

A smaller group of students (15%) described the Medialib exercises as "interesting" and noted that the visual feedback and simplicity of the multimedia interface made programming tasks feel more intuitive. Some (12%) commented on the ease of use, noting that Medialib made it easier to visualise code during exercises, and a few (6%) indicated that the interactive multimedia outputs increased their enjoyment of learning. However, not all feedback was positive: one participant reported that Medialib was not helpful due to a personal dislike of multimedia-based programming tasks. Additionally, seven students (21%) stated that the exercises were difficult, and three (9%) indicated confusion regarding the purpose or intended learning outcomes of Medialib exercises. This suggests that, while Medialib exercises were generally well-received, further clarification of the educational objectives of each exercise is needed to support all learners.

5) Insights from post-course interviews and recommendations for improvement

Five students participated in semi-structured interviews after the course concluded. While only one of the interviewees was enrolled in a computing-related major (Electrical Engineering and Computer Science), the others represented diverse disciplines, including Agriculture, Applied Chemistry, and Civil Engineering. Most participants indicated that Medialib exercises helped them better understand programming logic because the task expectations, particularly input and output behaviours, were made explicit through instructional videos. Compared to the traditional mathematics-based exercises, Medialib activities were perceived as clearer and more focused on programming reasoning than abstract mathematical formulations. When asked whether Medialib should replace traditional exercises, all five students expressed a preference for a combination of both, emphasising that while maths-based tasks were less engaging, they still played a role in reinforcing precision and algorithmic thinking.

Participant feedback also provided valuable insights for the iterative development of Medialib. A majority of them expressed a desire for clearer articulation of learning objectives at the beginning of each exercise, particularly for those requiring integration of multiple programming concepts. This is a finding consistent with the questionnaire results. Additionally, participants recommended the basic Python syntax be introduced alongside a greater number of examples to help novice learners become familiar with the Medialib functions. Notably, all interviewees (including those who expressed a dislike for mathematics) suggested that incorporating exercises tailored to data processing tasks relevant to their academic majors would enhance their motivation, citing the perceived career relevance of such content. This finding underscores that relevance to students' disciplinary interests serves as a crucial motivational

mechanism, aligning with broader theories of intrinsic motivation and contextual learning (Cordova & Lepper, 1996).

Further requests centred on enhancing opportunities for user interaction within the Medialib environment. For instance, several interviewees suggested extending the draw() function with customisable arguments. One interviewee suggested implementing a function to enable text output on the drawing window. These feedback directly informed the development of Medialib 2.0, which introduced additional parameters (width and height) for draw() to support image resizing, as well as a new text() function to enable on-screen text rendering (see Table 2 in Appendix). These enhancements demonstrate the iterative refinement of Medialib in response to user needs and pedagogical feedback.

Study 2: evaluating the enhanced Medialib 2.0 in a UK master's-level classroom

Participants

Study 2 was conducted at a university in the United Kingdom with two classes across 2 years (2021–2022) consisting of 84 students enrolled in a Master's program in Business Analysis. As illustrated in Fig. 6, most students did not hold an undergraduate degree in engineering or computer science. A background questionnaire designed to gather demographic information and prior programming experience, was provided prior to the course and received 80 responses from participants from diverse national backgrounds (China: 36; Taiwan: 13; Inida:13; Thailand: 12; UK: 2; UAE: 1; Pakistan: 1; Brazil: 1; Australia: 1).

Course design and Medialib integration

The course consisted of three 2-hour lectures and two 3-hour practical sessions delivered over a two-week period. The Python content covered: (1) variables, expressions, and functions; (2) conditionals, strings, and lists; and (3) for and while loops. Medialib 2.0, which resolved previous compatibility issues with Jupyter Notebook, was integrated into the in-class exercises alongside traditional maths-based exercises. Students were free to choose either PyCharm or Jupyter Notebook as their programming environment.

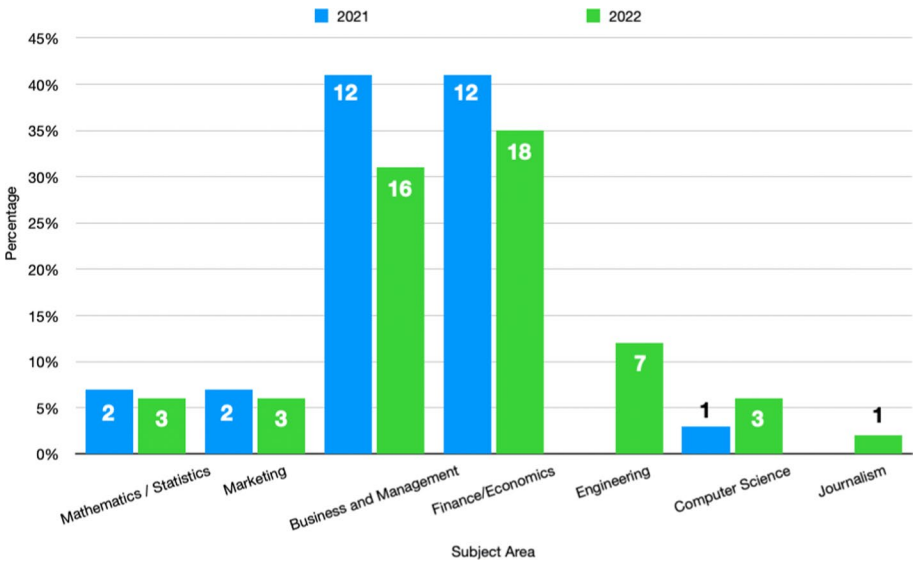

Fig. 6 Distribution of participants' undergraduate majors by cohort year for Study 2

Students were provided with two sets of programming exercises. They were required to practice the same traditional maths-based programming problems as in Study 1 as homework after each lecture. In Study 1, the Medialib tasks were supplementary exercises to go alongside the regular textbook exercises presented at the end of the lecture videos, and the students weren't given enough guidelines. Considering the feedback from Study 1 that the students were confused about the purpose of using Medialib, in Study 2 we clearly stated the learning objectives of each Medialib exercise at the beginning of each practical session. In both practical sessions in Study 2, the students were guided to practice Medialib exercises first and then they could discuss the maths-based questions with the lecturer. Throughout these sessions, the learner not only can learn how to use an external library but also how to combine basic Python syntax and media-based programming.

Data collection

In addition to the background questionnaire, students were invited to complete a post-course questionnaire. This survey included both quantitative and qualitative items to capture their perceptions of the difficulty, usefulness, and instructional sequencing of the two exercise types. The Likert-scale items assessed students' perceptions of task difficulty (on a 5-point scale), preferences for exercise sequencing, and overall course satisfaction. Open-ended items solicited more nuanced reflections on their experience with Medialib and the course more broadly. Besides survey data, observational notes from instructors during practical sessions were collected to capture students' engagement, challenges encountered, and types of questions raised during coding tasks. These multimodal data sources enabled a comprehensive evaluation of both the instructional design and learner outcomes in Study 2.

Results

1) Exercise order and perceptions of task difficulty

The post-questionnaire was completed by 61 students (52% female). When asked about the preferred order of exercises, 66% preferred to do traditional exercises before Medialib exercises, while only seven students preferred the reverse sequence, and 14 reported no preference. As shown in Fig. 7, most participants rated both types of exercises as moderately difficult. An independent t-test shows there are no significant differences ($t(120) = 0.524$, $p>0.05$) between regular (Mean=3.74, S.D.=0.705) and Medialib (Mean=3.67, S.D.=0.676) exercises.

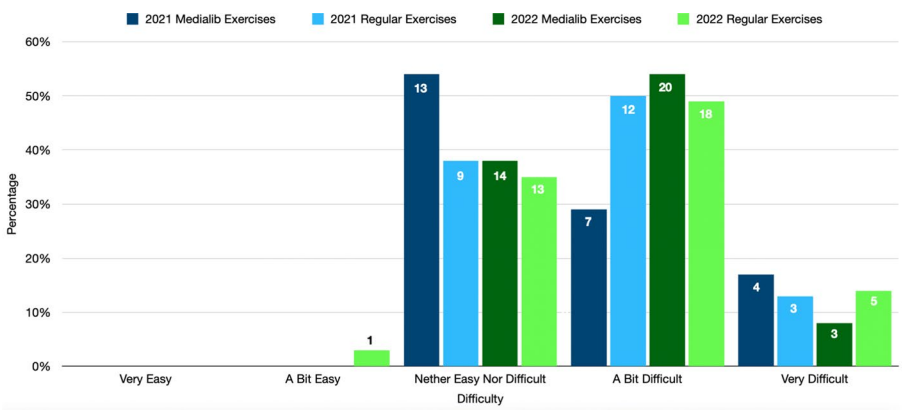

Fig. 7 Comparative perceived difficulty of regular and Medialib exercises for Study 2

Another independent t-test is performed to understand if prior programming experience affected the student's perception of the difficulty of the Medialib and regular exercises. For the Medialib exercises, there is no significant difference in the perceived difficulty ($t(59) = 1.29$, $p>0.05$) despite those with programming experience ($N = 46$, Mean = 3.61, S.D. = 0.714) finding the exercises slightly less difficult than those without ($N = 15$, Mean = 3.87, S.D. = 0.516). Similarly, for the regular exercises, there is also no significant difference in the perceived difficulty ($t(59) = 0.814$, $p>0.05$) despite those with programming experience ($N = 46$, Mean = 3.7, S.D. = 0.662) finding the exercises less difficult than those without ($N = 15$, Mean = 3.87, S.D. = 0.834). However, due to the small sample size of the participants without programming experience in study 2, future studies with larger and more balanced sample sizes should further explore the impact of programming experience on the perceived difficulty of exercises. Furthermore, no participants thought that both exercises were very easy even if they had prior experience suggesting that the exercises were still sufficiently challenging.

2) Student feedback on Medialib

As illustrated in Fig. 8, students' feedback on Medialib was overwhelmingly positive. Among the 61 respondents, 79% reported that Medialib was helpful in reinforcing basic programming concepts and in applying lecture content to practical tasks. One participant noted: *"Medialib is very helpful. It forces us to approach problems in a different way and think outside the box sometimes, which can come in handy in the time of doing analytics work in real life situations. I find Medialib's flexibility to be a great introduction to the programming way of thinking."*.

Five students (8%) found the game design aspects of the exercises particularly engaging, while others (5%) appreciated the multimedia elements, which made it easier to understand and debug their code. One student commented: *"I would like to learn more about Medialib. It would be better if I could use Python to create some sample games."*. Only two participants expressed uncertainty about the usefulness of Medialib; both had no prior programming experience and found the Medialib tasks somewhat challenging.

3) Perceptions of course design and learning outcomes

A large majority of participants (90%) stated that the course met or exceeded their expectations. Several students expressed surprise at the degree of coding involved,

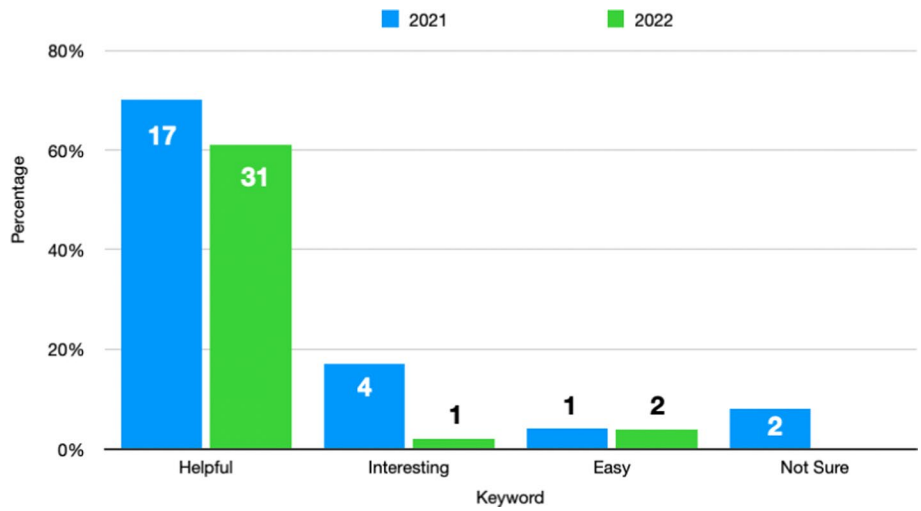

Fig. 8 Frequency of keywords describing how participants felt about Medialib in Study 1

noting that the interactive lab sessions provided a deeper learning experience than anticipated. Some mentioned that even if they previously had programming experience, the course was beneficial in learning the fundamentals of programming. One student remarked: *"The course is beyond my expectation because the lectures not only help me build a framework of basic knowledge about computer programming language but also engage me to learn more extra things. I would describe the course content as being moderately challenging in a creative way, and the team did well in choosing the content and level of difficulty for the course."* This aligns with quantitative data indicating that both sets of exercises were perceived as moderately challenging, regardless of students' prior programming experience.

Discussion: deepening the understanding of Medialib's pedagogical value

This study aimed to investigate the pedagogical utility of a simplified multimedia-based Python library, Medialib, for novice programmers, particularly non-technical students who are often underserved by conventional programming curricula. Drawing on empirical findings from two classroom-based studies, we examined how Medialib exercises influences programming skill acquisition, student engagement across diverse learning contexts, and the pedagogical and technical considerations essential for designing such tools. This section offers a deeper interpretation of these findings in relation to the three research questions.

Impact on programming skill acquisition (RQ1)

The first research question examined how Medialib affects novices' programming development relative to traditional, maths-based tasks. The performance data from Study 1 strongly suggests that Medialib acts as a pedagogical buffer, a cognitive scaffold that smooths the transition from conceptual exposure to practical application. In weeks 5 and 6, when students were introduced to abstract constructs such as functions, conditionals, and loops, performance on regular exercises sharply declined, while performance on Medialib tasks remained stable. This divergence can be interpreted as a function of reduced cognitive load: Medialib's tasks embedded these abstract concepts in a familiar and visually rewarding context, helping learners grasp their functionality without needing to simultaneously decode mathematical logic.

Moreover, the convergence of scores after week 7 suggests a transfer effect, wherein the conceptual gains and confidence built through Medialib translated into improved performance on traditional tasks. Notably, students with no prior programming background still achieved above-passing average scores (Mean=72.5, S.D.= 3.64) on final projects, outcomes rarely observed in standard introductory programming settings. This supports the view that multimedia-supported tools like Medialib, can make computational literacy more accessible by decoupling programming from mathematics, thereby engaging a broader spectrum of learners.

Student engagement across contexts (RQ2)

The second research question explored how students interact with Medialib across different settings. Study 1 took place in an Japanese undergraduate context during the COVID-19 pandemic, while Study 2 involved UK-based postgraduate students from a

business analytics programme. Despite these contextual differences, student engagement was consistently positive, albeit for different reasons.

In Study 1, engagement was largely driven by emotional relief, students who had previously struggled with mathematics found the visual and interactive nature of Medialib tasks less intimidating, which reduced their programming anxiety. Some expressed that they felt programming could finally be "fun," a qualitative shift that may influence long-term attitudes toward computing. In Study 2, engagement was shaped more by perceived relevance and cognitive ownership. Students appreciated how Medialib supported the incremental integration of lecture concepts and how it allowed them to create tangible, even playful outputs (e.g., mini games). This aligns with constructivist theories of learning (Hein, 1991), which emphasise the value of active exploration, personal relevance, and immediate feedback in fostering deep learning.

Crucially, both studies indicate that Medialib's design neutralises prior inequalities in experience, disciplinary background, and mathematical confidence, thereby enhancing inclusivity in programming instruction.

Design considerations for educational programming tools (RQ3)

The third research question focused on what makes multimedia programming libraries like Medialib and its exercises pedagogically effective for beginners. Across both studies, several design features emerged as instrumental:

1. Incremental notional machines: Medialib applied this principle by introducing Python syntax in a logically sequenced, simplified form. This allowed learners to build conceptual fluency before tackling more complex or idiomatic code.
2. Natural language-like function names: The use of intuitive, self-explanatory function names (e.g., draw, clear, wait) minimized entry barriers and enhanced learners' ability to understand and modify code independently.
3. Task design with visual anchoring: Providing clear learning objectives with animations or screenshots of expected outputs helped learners reverse-engineer program logic. This strategy bridged the gap between mental models and syntax execution, a common obstacle in early programming education.
4. Use-modify-create pedagogy: Medialib's exercises were intentionally structured to move learners through increasingly open-ended tasks, encouraging progressive autonomy. This aligns with zone of proximal development (Vygotsky, 1978), in which learners gain competence through supported exploration.

Conclusion

This research aims to explore how Medialib supports the development of foundational programming skills, enhances learner engagement, and facilitates transfer across diverse educational contexts. In addition, the study seeks to identify key design principles for constructing accessible programming exercises that address the specific cognitive and motivational needs of beginners.

The results of this study suggest that a scaled-down, multimedia-oriented Python library like Medialib can serve as an effective pedagogical bridge for novice programmers. Medialib

not only supports conceptual understanding and task performance but also enhances learner motivation and confidence, especially for non-technical students who often find programming inaccessible. Both empirical studies show that learners with little or no programming experience can successfully use Medialib to build foundational knowledge and gradually transition to more complex programming tasks. The approach of combining simplified syntax, visual output, and step-by-step learning objectives appears particularly effective for learners with lower mathematical confidence, aligning with recent educational theories that emphasise personalised, visual, and exploratory learning pathways.

Beyond its initial application as an introductory programming scaffold, Medialib demonstrates potential for scalability and cross-disciplinary integration. Its modular design allows it to scale from beginner multimedia exercises to more complex applications by integrating with other Python libraries such as Matplotlib (Hunter, 2007) and Pandas as presented in Wynn et al. (2024). Medialib abstracts foundational programming logic (such as loops and conditionals) from complex syntax and provides a universal, low-barrier entry point regardless of a student's background. By enabling students to build this cognitive scaffolding through simple multimedia interactions before transitioning to domain-specific computational tasks, Medialib provides a foundation for embedding computational thinking across the broader curriculum. Future work should validate more complex exercises to establish whether the benefits of Medialib exercises extend to long-term programming learning.

Table 2 The functions in Medialib 3.0

Category	Definition	Effect
Graphics	clear(r=None,g=None,b=None)	clears the drawing window
	rect(x,y,w,h,r=None,g=None,b=None)	draws a rectangle
	draw(imgFileName,x,y,width=None,height=None)	loads an image (amended in 2.0, was draw(imgFileName,x,y) in 1.0)
	save_screen(file_name,pos_x=None,pos_y=None, width=None, height=None)	saves an image of the current drawing window
Graphical Text	set_font(file_name)	sets an external file as the new current font
	text(message,x,y,font_size,r=None,g=None,b=None)	writes the message string in the drawing window (added in 2.0)
	get_text_rect(message,x,y,font_size)	returns the width and height of the rectangle containing the text (added in 3.0)
Input	wait_key_press()	returns a character when ESC, a number or a character is pressed. Otherwise, return a code
	is_key_press(keyName)	returns True if the key keyName was pressed
	wait_mouse_leftclick()	pauses the control flow of the program until the mouse's left button is pressed (renamed in 2.0)
	is_mouse_leftclick()	returns True if the left mouse button is pressed (renamed in 2.0)
	get_mouse_pos()	returns the x and y coordinates of the mouse
Others	wait(secs)	pauses the control flow of the program for a certain number of seconds
	play(soundFileName)	loads an audio file
	distance(x1,y1,x2,y2)	returns the distance of two points in pixels
	a_to_b(a,b,tPercentage)	returns a linear interpolation between two values
	point_inside_rect(px,py,pos_x,pos_y,width,height)	returns True if a point (px,py) is inside a rectangle (added in 3.0)

Appendix A

See Table 2.

Author contributions
Jingyun Wang designed the methodology, designed and conducted the experiment, analysed the data and wrote the main manuscript. Adam Wynn supported the conduct of the study 2, the data analysis and the writing of the main manuscript. Jingyun Wang and Andrea Valente developed the Medialib library and its teaching materials. Andrea Valente and Emanuela Marchetti supported the experiment design of study 1, and the writing of related work and methodology part. Daner Sun supported the experiment design of study 2, reviewed and edited the manuscript.

Funding
This research is funded by Durham University Research Impact Fund.

Data availability
The raw learner data generated during the current study are not publicly available due to individual user privacy concerns.

Code availability
The teaching materials and the source code of the proposed Python library Medialib can be accessed from our official website https://medialib.club/

Declarations

Ethics approval and consent to participate
Ethical approval for this research was granted from two committees: Faculty of Information Science Ethics Committee at Kyushu University (Reference: Information Science 2020-11) and the Department of Computer Science Ethics Committee at Durham University (Reference: COMP-2022-09-30T11_44_44-dtpg64).

Competing interests
Not applicable

Received: 12 June 2025 Accepted: 9 March 2026
Published online: 25 March 2026

References

Agbo, F. J., Oyelere, S. S., Suhonen, J., & Adewumi, S. (2019) A systematic review of computational thinking approach for programming education in higher education institutions. In *Proceedings of the 19th Koli Calling international conference on computing education research*. Association for Computing Machinery, New York, NY, USA, Koli Calling '19, https://doi.org/10.1145/3364510.3364521,

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to k-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Bart, A. C., Tibau, J., Tilevich, E., Shaffer, C. A., & Kafura, D. (2017). BlockPy: An open access data-science environment for introductory programmers. *Computer, 50*(5), 18–26. https://doi.org/10.1109/mc.2017.132

Berry, M., & Kölling, M. (2013). The design and implementation of a notional machine for teaching introductory programming. In *Proceedings of the 8th workshop in primary and secondary computing education (WiPSCE '13)* (pp. 25–28). https://doi.org/10.1145/2532748.2532765,

Cordova, D. I., & Lepper, M. R. (1996). Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice. *Journal of Educational Psychology, 88*(4), 715–730. https://doi.org/10.1037/0022-0663.88.4.715

Duran, R. (2019). Notional machines. https://compedonline.school.blog/2019/07/26/notional-machines/, computing Education from the Tropics!

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2020). Computational thinking in programming with scratch in primary schools: A systematic review. *Computer Applications in Engineering Education, 29*(1), 12–28. https://doi.org/10.1002/cae.22255

Fincher, S., Jeuring, J., Miller, C.S., et al (2020). Notional machines in computing education: The education of attention. In *Proceedings of the working group reports on innovation and technology in computer science education*. Association for Computing Machinery, New York, NY, USA, ITiCSE-WGR '20 (pp. 21–50). https://doi.org/10.1145/3437800.3439202,

Geist, E. (2016). Robots, programming and coding, oh my! *Childhood Education, 92*(4), 298–304. https://doi.org/10.1080/00094056.2016.1208008

Guzdial, M. (2022). Teaspoon languages for integrating programming into social studies, language arts, and mathematics secondary courses. In *Proceedings of the 53rd ACM technical symposium on computer science education V. 2*. https://doi.org/10.1145/3478432.3499240,

Hein, G. E. (1991). Constructivist learning theory. Institute for Inquiry 14.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering, 9*(3), 90–95. https://doi.org/10.1109/mcse.2007.55

Krishnamurthi, S., & Fisler, K. (2019). Programming paradigms and beyond. *The Cambridge handbook of computing education research* (pp. 377–413). Cambridge University Press. https://doi.org/10.1017/9781108654555.014

Lai, R. P. Y. (2021). Beyond programming: A computer-based assessment of computational thinking competency. *ACM Transactions on Computing Education, 22*(2), 1–27. https://doi.org/10.1145/3486598
Lodi, M. (2020). Informatical thinking. Olympiads in Informatics pp 113–132. https://doi.org/10.15388/ioi.2020.09,
McKinney, W. (2010). Data structures for statistical computing in python. In *Proceedings of the python in science conference. SciPy, Austin, Texas*. https://doi.org/10.25080/majora-92bf1922-00a,
Pedersen, B. K. M. K., Marchetti, E., Valente, A., & Nielsen, J. (2020). Fabric robotics-lessons learned introducing soft robotics in a computational thinking course for children. *Learning and collaboration technologies. Human and technology ecosystems* (pp. 499–519). Springer International Publishing. https://doi.org/10.1007/978-3-030-50506-6_34
Peppler, K., & Kafai, Y. (2005). Creative coding: Programming for personal expression. In *Proceedings of the 8th international conference on computer supported collaborative learning* (pp. 76–78).
Rich, P. J., Bartholomew, S., Daniel, D., et al. (2022). Trends in tools used to teach computational thinking through elementary coding. *Journal of Research on Technology in Education, 56*(3), 269–290. https://doi.org/10.1080/15391523.2022.2121345
Rus, T. (2016). *Think Python: How to Think Like a Computer Scientist*. CreateSpace Independent Publishing Platform.
Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education, 13*(2), Article 31. https://doi.org/10.1145/2483710.2483713
Valente, A., Marchetti, E., & Wang, J. (2020). Design of an educational multimedia library to teach Python to non-technical university students. In *International conference on learning technologies and learning environments (LTLE) 2020*. https://durham-repository.worktribe.com/output/1139801
Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.
Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question. In *Proceedings of the 14th international conference on interaction design and children*. ACM, New York, NY, USA. https://doi.org/10.1145/2771839.2771860
Weintrop, D., & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers and Education, 142*, Article 103646. https://doi.org/10.1016/j.compedu.2019.103646
Wynn, A., Wang, J., & Valente, A. (2024). Simplifying multimedia programming for novice programmers: MediaLib and its learning materials. In *Proceedings of the 2024 on innovation and technology in computer science education V. 2 (ITiCSE 2024)*:785–786. https://doi.org/10.1145/3649405.3659521

Publisher's Note