# Embedding Executable Code in Programming Slideshows: Design Considerations and Field Tests for Interactive Code Playgrounds

Lorenzo Angeli*
lorenzo.angeli@unitn.it
University of Trento
Povo, TN, Italy

Luca De Menego*
lucademenego99@gmail.com
Thema Optical
Domegge di Cadore, BL, Italy

Maurizio Marchese*
maurizio.marchese@unitn.it
University of Trento
Povo, TN, Italy

## Abstract

When students take introductory programming courses in university, they often need to setup a development environment. Development environments enable code execution while also providing a uniform setting for students to compare results and facilitate troubleshooting. At the same time, though, development enviroments often require installing applications or signing up to web services.

In this paper we present an alternative approach, in the form of an in-browser slideshow system called Interactive Code Playgrounds (ICPs). ICPs aim to create a uniform environment while also making programming lectures more approachable facilitating code execution. In ICPs, code is embedded in webpages displayed as slides, enabling code execution and output visualisation directly in the browser through the use of JavaScript and WebAssembly. This paper presents ICPs' design principles, and reports on the outcomes of a mixed-methods validation study that, through a combination of surveys and micro-ethnographies, involved 95 participants at three introductory programming courses at the University of Trento.

Our research suggests that students generally understood ICPs' functionalities, and successfully used ICPs to follow their lectures, with the emergence of patterns of self-pacing and self-tailoring. By introducing an education technology that does not require radical changes in the course's teaching and delivery, we hope to show that education technologies do not necessarily need to disrupt teaching and learning, but can also take an incremental approach while still yielding meaningful pedagogical benefits.

## CCS Concepts

• **Applied computing** → **Interactive learning environments**; • **Social and professional topics** → **Computer science education**; **Computing education**; • **Information systems** → *Web interfaces.*

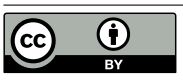
## Keywords

introductory programming, slideshows, webassembly, education technologies

**ACM Reference Format:**

Lorenzo Angeli, Luca De Menego, and Maurizio Marchese. 2025. Embedding Executable Code in Programming Slideshows: Design Considerations and Field Tests for Interactive Code Playgrounds. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701788

*All authors contributed equally to the submission.





*SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA*

ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701788

## 1 Introduction

A long-standing adage in education posits that, if one were to go back in time several centuries, one of the few institutions that they would immediately recognise is a university lecture hall. This adage frames education as a field that is reluctant to innovate itself, resisting change in the face of innovation.

The field of programming education seems to be a particularly relevant case where this paradox persists. Teachers of programming need to teach evergreen concepts, but also face the pressure of changing their course's content to remain up to speed with fast technological change [2], leading to a tension that sees on the one hand techno-solutionism [5, 27], and on the other hand a general reluctance to change teaching practices that sees pedagogical innovators forming "innovation bubbles" [14]. At both sides of this tension, however, lie a missed opportunity: at a previous SIGCSE lightning talk, Fox [6] described that minor changes in the tools used to deliver introductory programming courses may deeply benefit the classroom environment, for example by better including people with visual or hearing impairments [9, 20].

Lecturing remains a common teaching method in university programming courses. In programming lectures, teachers often use slideshows that include code snippets. Students, in turn, need to transfer the code to a development environment before executing it. The different students in a classroom, however, may have different development environments, creating issues in managing the class and assisting students in troubleshooting. As we discuss in Section 2, there are multiple ways to approach this issue, but each of them is a trade-off that mitigates some issues, and exacerbates others.

In this paper, we wish to propose another approach, centered on an education technology that tries to solve some of these challenges while minimising the disruption to the traditional programming lecture's structure. Our goal with this intervention is to try to mitigate two key issues encountered in beginner programming lectures at university level: (1) providing an uniform development environment for students and teachers to play with the execution of code snippets; and (2) reducing the needed steps that students and teachers need to perform to set up a development environment, or to execute code snippets. The tool we present allows code execution and output display on HTML-based slides, and is called Interactive Code Playgrounds (ICPs). In Section 2 we will relate our work to previous academic literature, grounding our intervention in previous research and practice. In Section 3 we will discuss the

methodology of our work, including tool design, the courses in which we validated our tool, and the validation study's research methodology. Section 4 will present the outcomes the validation study itself, which aimed at assessing the tool's design and effectiveness over 10 sessions in 3 different courses involving a total of 95 participants. Finally, in Section 5, we will discuss some limitations of our work, trace future steps, and provide concluding remarks.

## 2 Background

Digitalization has profoundly reshaped the landscape of modern education, presenting both opportunities and challenges [24]. In programming education, the integration of many languages and development environments has however introduced complexity and fragility [3], potentially adding complex setup steps to students' learning experiences. As a response, teachers have been trying to identify simple, platform-independent alternatives that facilitate seamless learning experiences across student setups [16].

Pedagogically, achieving consistency in software behavior across devices, operating systems, and software versions remains a significant challenge [29]. This variety is particularly problematic in blended learning environments or during online lectures, where a degree of uniformity is crucial for effective teaching (i.e., reducing troubleshooting) and providing fair assessment grounds. During the COVID-19 pandemic, researchers observed that socio-technical challenges, such as unequal access to high-speed internet and technology, contribute to educational disparities among students [28]. The rapid evolution of technology also creates challenges for educational institutions striving to keep pace with digital advancements [18], and for less-wealthy students in keeping up with increasing hardware requirements driven by software bloat [31].

The ways in which technology has been integrated in education — and also in programming education — has evolved significantly over the past decades. Early efforts during the 2000s primarily focused on digitizing didactic materials without fundamentally altering teaching paradigms. Clegg *et al.* [4] and Selwyn [25], for example, observed that ICT-based learning often mirrored traditional information delivery methods, such as PowerPoint presentations, rather than innovating pedagogical practices. Prensky [22], however, argued that digital technology's transformative potential in education hinges on adapting pedagogical approaches, shifting from passive information transmission to more interactive and engaging methodologies. In spite of the widespread belief in technology's disruptive impact on education, however, authors such as Teräs [27] cautioned against viewing technology as a panacea for educational challenges, emphasizing the need for critical examination and effective integration into existing educational frameworks. Henderson *et al.* [11] further underscored that digital technologies have not radically transformed university teaching and learning experiences. As a result, they advocated for continuous refinement and development of digital resources to enhance educational quality and accessibility.

In K-12 programming education, teachers often use visual programming languages like Scratch, which significantly enhance learning and maintain student motivation through active engagement [23]. In higher education, instead, change has been slower, and the most prevalent teaching medium are still slideshows with code snippets. Selwyn noted already in 2007 how, at university level, educational technologies seem to be subject to slower advancement [25], perpetuating the problems students encounter when setting up and using development environments.

To address these challenges, teachers attempted to adopt solutions such as virtual machines (VMs) [7, 10, 13], Docker containers [29], and browser-based tools [16, 17]. Each of these solutions offers different trade-offs in simplifying and standardizing development environments: Cloud-based VMs provide a consistent environment but require stable internet and have latency issues [15]; locally-executed VMs offer offline access and full control over resources, but are complex to set up and demand significant computational power; setting up a container-based environment is even more complicated than installing a VM, and the containers metaphor may be somewhat difficult to grasp for students [29].

Browser-based educational tools deserve a deeper look, since they introduce platform-independent solutions that enhance usability, while not requiring the installation of additional programs. Projects such as GitHub Codespaces [16] and online code playgrounds such as Replit [32], Python Tutor [8] or CodeGrade[1] exemplify the integration of development environments within web browsers, facilitating collaborative learning and reducing setup complexities. All of these tools, however, rely on constant internet connectivity, which poses challenges in environments with limited or unreliable internet access, as highlighted by the digital divide aggravated during the Covid-19 pandemic [28]. Additionally, accessibility remains a critical concern, with HTML-based solutions that incorporate semantic elements and ARIA attributes emerging as promising approaches to address diverse learner needs [6, 26]. In this sense, WebAssembly (WASM) seems to hold great promise, through technologies such as JupyterLite[2], RISE[3] or even JSLinux[4].

In summary, the literature recognises that, while programming education is evolving, there still are many challenges in selecting the appropriate supporting education technologies. Effective integration of these technologies requires not only technical innovation, but also pedagogical adaptation and consideration of accessibility needs to ensure equitable educational opportunities.

## 3 Experimental Context

Interactive Code Playgrounds (ICP) is a web-based slides system that uses several web components to enable code editing and execution directly on the presentation. Their primary use case is to support introductory coding lectures, reducing the steps that students need to perform to set up the development environment and execute code snippets, while providing a uniform environment across different types of PCs. By using WebAssembly (WASM), ICPs can execute code in a variety of programming languages and, since web browsers are one of the most ubiquitous PC applications, they are widely compatible and uniform among different setups.

ICPs support a variety of programming languages. The browser's engine can directly execute JavaScript, TypeScript, and Processing (through p5.js), while WASM extends compatibility to Python, Java, C++, and StandardML. Finally, through an in-memory SQLite database, ICPs can also execute SQL queries.

[1] https://www.codegrade.com/ (accessed 2024/07/19)
[2] https://github.com/jupyterlite/jupyterlite (Accessed 2024/11/14)
[3] https://github.com/damianavila/RISE (Accessed 2024/11/14)
[4] https://bellard.org/jslinux/ (Accessed 2024/11/14)

To use ICPs, students open the webpage that contains the class' slideshow. In Figure 1, we provide an example of an ICP slide, featuring some header text and most relevantly the code editor's interface. ICPs' code editor features syntax highlighting, copy/paste functionality, standard keyboard shortcuts for code navigation and editing, full-screen mode, and a light/dark theme switcher. The editor also features a reset button that lets those who use it revert the editor to the original code. The editor additionally supports "scaffolded exercises": instructors can author the slides to only allow specific sections of the code to be editable, guiding learners through tasks. We chose this set of functionalities not to provide feature-completeness or to eliminate the need for development environments, but to reduce some of the main reasons that require switching to the development environment to perform basic operation on code snippets, while still letting learners easily move to an external editor if they feel they need to.

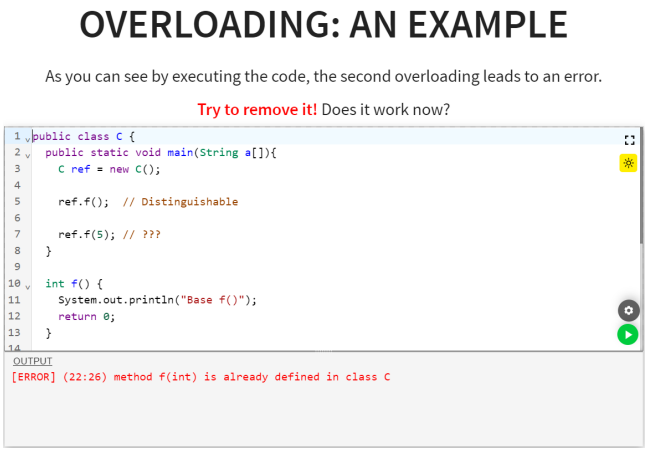



**Figure 1: Example of an ICP slide. The slide features a title, some explanatory text, and the code editor, with the output box below it.**

From a software perspective, ICPs are built on Reveal.js[5], a JavaScript library that facilitates the creation of HTML slide decks and adds a number of visual functionalities. The editor itself is based on CodeMirror[6], and language compatibility is provided through a variety of libraries, compilers, and interpreters. This set of software forms the "ICP Bundle"[7]. Figure 2 summarises the general architecture of ICPs and highlights its main components.

ICP slides can be authored in two ways: either, as it is standard for Reveal.js slides, by directly changing the slides' HTML code[8], or by using a separate web-app, the ICP Editor[9]. While the ICP Editor is still at an early stage of development, it already is a functional authoring environment that also works as a frontend-only web application, in line with the rest of the ICP experience.

Slideshows created with ICPs can be exported and distributed in several ways:

[5]https://revealjs.com/ (accessed 2024/07/19)
[6]https://codemirror.net/ (accessed 2024/07/19)
[7]Available at https://github.com/lucademenego99/icp-bundle (Accessed 2024/11/15)
[8]See https://revealjs.com/markup/
[9]Available at https://github.com/lucademenego99/icp-editor (Accessed 2024/11/14)

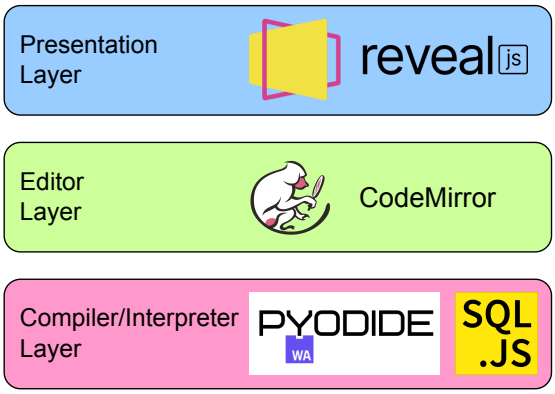



**Figure 2: The architecture of ICP. The compiler/interpreter layer shows some example libraries only.**

(1) as a single HTML file that can be opened by a browser, but that will require internet connectivity to download libraries and assets;
(2) as a ZIP file containing all necessary files for hosting on a web server;
(3) as an executable file compatible with Windows, Linux, and MacOS, automatically starting a local web server (called Redbean[10]) that shows the slides.

We see ICPs as a technology designed to align with principles of "Frugal Computing" as proposed by Vanderbauwhede [30]. We see this action as a response to growing concerns about the environmental impact of computing, such as those discussed in the Computing Within Limits literature [21], meaning that, while taking inevitable trade-offs, we tried to build ICPs to prioritise energy efficiency and software longevity, for example by testing ICPs on 10-years-old computers during development. We also see the choice of using a web browser rather than, for example, an IDE, an Electron-powered editor, or a cloud-based code execution environments, as a choice that goes towards a reduction of the educational environment's computational demands.

## 3.1 Pedagogical Contexts

To make logistics easier, we performed the first tests of ICPs in the University of Trento. We carried out a preliminary round of testing in July 2022 during an intensive Python introductory course at the Department of Sociology and Social Research, with around 20 students who were new to programming.

The main round of testing for which we report our observations, though, was conducted in February–May 2023 on three distinct courses. We discovered the courses using the University of Trento's course index, scanning for introductory computer science or programming courses carried out during the testing period. We eventually identified three courses: (1) "Elements of Programming I", teaching programming with Processing to students in Bachelor's degrees at the Department of Psychology and Cognitive Science; (2) a seminar course titled "Introduction to Python Programming

[10]https://redbean.dev/ (accessed 2024/07/19)

for Social Sciences", again for Bachelor's students; and (3) a course in "Computer Programming 2", teaching Java and Object-Oriented Programming to students of Computer Science.

We summarise the courses that we identified, and the number of participants, in Table 1.

| Course | Discipline | Students |
|---|---|---|
| Intro to Progr. with Python | Sociology | 20 |
| Elements of Programming I | Psychology | 54 |
| Computer Programming 2 | Computer Science | 21 |
| | **Total** | **95** |

**Table 1: General information about the intervention contexts**

We conducted, in total, 10 interventions, of which 2 sessions were held in *Intro to Progr. with Python*, 5 in *Elements of Programming I*, and 3 in *Computer Programming 2*. All the sessions were carried out several weeks after the course had began, meaning that students familiarised themselves with the course's official IDE before interacting with ICPs. In all interventions, the lectures were taught by the course's regular instructors. Our research team, instead, took care of translating the slideshows from their pre-existing format into ICPs, a mostly mechanical task.

## 3.2 Research Methods

For this study, we employed a mixed-methods approach, integrating qualitative and quantitative methods, using as our data sources surveys and a micro-ethnography. Combining micro-ethnography and surveys is a long-standing research method in educational settings [19, 33] and has been characterised by Johnson as a way to collect "relatively objective firsthand information" [12].

The surveys, which included a pre-experience survey and a post-experience one, included both open-ended and closed-ended questions. The pre-experience survey collected demographic information, gauged the students' initial perceptions of programming, and presented ICPs. The post-experience survey followed up on the initial perceptions, assessed the use and quality of ICPs, and gathered additional feedback. All surveys were anonymous, administered in a batch to ensure high response rates and giving space for clarifications on the spot, and were optional.

The micro-ethnography involved participant observation in the classroom, with one researcher engaging directly with the students to document their interactions with ICPs. This method provided rich, contextual data, capturing the nuances of students' experiences that surveys alone might have missed. Observations were systematically recorded, focusing on student activities, behaviors, and classroom dynamics. Students were asked for their informed consent for their participation in the micro-ethnography, and those that did not want to participate received a red paper card to keep on their desk to signal they should not be observed.

For data analysis, we relied on basic descriptive and inferential statistics to analyse close-ended survey questions, plus open coding followed by thematic coding to identify recurring themes from the qualitative data. Quantitative analysis was done on a spreadsheet, while coding was done on Taguette[11]. To improve the qualitative data's reliability, we triangulated data by cross-referencing survey responses with field notes.

[11] https://www.taguette.org/ (accessed 2024/07/19).

# 4 Results and Discussion

Our presentation of results, and their discussion, follows the structure of the surveys, which were divided in several sections. As a summary of our participant population, we refer back to Table 1.

## 4.1 Initial assessment of programming knowledge and student motivation

The initial assessment revealed that 41 out of 74 non-computer science students had prior experience with programming languages such as Stata, Python, Java, and C/C++. Despite this, many students still identified as beginners, indicating a foundational but limited grasp of programming concepts. Computer science students identified as more experienced, though they still perceived themselves as novices. This diversity in prior knowledge provided a broad testing ground for ICPs to test its utility across varying skill levels.

The study found a high level of motivation among students: 84 out of 95 participants provided an open-ended answer to a question asking why they wanted to learn to code, and 82 expressed a desire to learn programming beyond course requirements. This enthusiasm was driven by aspirations to develop projects, enhance future career prospects, and personal interest in the subject. However, field notes indicated a dip in perceived enthusiasm during theory-heavy, non-interactive lectures, suggesting that interactive and engaging teaching methods may help sustaining student interest.

> *Almost everyone has the PDF slides open, but for now, I see few people copying the code onto their IDE to test it. I have to admit that this never happened with ICPs. When the proposed tool was used, as soon as the professor started explaining theoretical concepts, the students would usually go to the first slide containing an example to try running it.*
>
> `Field notes — Elements of Programming I`

## 4.2 Technical Issues

One of the main challenges identified in literature regarding programming classes is the difficulty in setting up a development environment [10, 16]. Before our intervention, 16 out of 95 students declared in the surveys that they encountered difficulties installing the course's prerequisite software, with responses to open-ended questions indicating that version incompatibilities were a frequent issue. ICPs mitigated this issue, since the proposed tool is entirely browser-based and required no installation process, meaning students encountered fewer issues compared to installing the IDE required for the course.

Our aims, however, were not only to reduce issues encountered during installation, but also during interaction with the development environment. When students were asked to rate the amount of problems they encountered while using the course's IDE on a Likert scale of 1 to 5, the average response was 2.1 (95% CI ± 0.2). For ICPs, the score was slightly lower, at 1.9 (95% CI ± 0.2). The responses' mode was different, 2 for the IDE and 1 for ICPs, though

the improvement brought by ICPs looks not to be substantial. A possible explanation that we found by looking at the field notes is that ICPs still occasionally generated errors, especially when compiling and executing Processing, which was not well integrated.

Another limitation of ICPs is their compatibility with browsers other than Google Chrome or Firefox. Students using Safari encountered minor issues, and one student using Brave was unable to execute code snippets from the slides. Although installing a new browser is a quick process, taking only a few minutes, and asking students to switch to Google Chrome resolved the issues, this reliance on a specific browser limits the tool's usability.

## 4.3 UI Evaluation

When showed a static image of an ICP slide in the survey, most students (about 80%) stated they thought they understood how to perform essential tasks like editing, executing code, and finding output; 65% also felt like knowing how to open the settings, enableing full-screen mode and changeing the theme. 4% of students stated they had no idea how to perform any of these tasks. The copy and reset buttons were understood by approximately 80% and 60% of students respectively. Student seeemed to understand less (and we observed less use of) the tabs handling button and the scaffolded code snippets. Post-intervention, 72% of students stated they felt more familiar working with ICPs rather than the course's IDE.

## 4.4 Features of slides systems

Students were also asked to rate what features they found to be more important for a slide system, with the intention to compare PDF slides with ICPs. Students rated device compatibility, copy/paste functionality, and keyword search as the most important features for a slide system (M = 4.0; 95% CI ± 0.2). While compatibility may advantage PDFs, copying code is often cumbersome [6], and ICPs are advantaged on this front. As for search, ICPs only support keyword search in the vertical display, while PDFs generally support search (as long as code is included as text and not as images), but may not properly handle search when words contain special glyphs (like double-'f's) or line breaks. Students also rated offline availability and PDF exports as important (M = 3.7; 95% CI ± 0.2).

Regarding other features of ICPs, Table 2 highlights that students found the ability to stay within the browser and make real-time changes to code snippets to be the most valuable feature.

## 4.5 Accessibility

While we recognize the crucial role of accessibility in educational tools, our assessment of ICPs in this regard is informal. Though we lacked the opportunity to collect rigorous data, particularly concerning the use of ICPs with screen readers, preliminary observations suggest that ICPs offer some accessibility features for students with visual impairments. Zooming functionalities and contrast-based solutions generally operate effectively within the ICP environment.

> *A student with visual impairments increased the contrast to the maximum in order to see the slides better. The only problem resulting from this is that the [on-screen navigation] arrows disappeared. Fortunately, it is possible to navigate the slides using the [keyboard's arrow keys], so I didn't have to assist the student in any way. The playground is also clear enough with this contrast, and all the buttons appear white on a black background, making them fully visible.*
>
> `Field notes — Intro to Progr. with Python`

## 4.6 Usage of ICPs

In our interventions, using ICPs was optional for the students, allowing them to choose between the proposed tool and the original didactic material offered by the instructor. Field observations indicate that, in every intervention, approximately 90–95% of students opted to use ICPs. Students self-reported their own use of ICPs as very high (M = 4.4; 95% CI ± 0.2). This high usage rate may be due to the tool's interactivity: when students were asked to rate whether they preferred interactive or static study material, they showed a clear preference for interactive materials (M = 4.4; 95% CI ± 0.2; on a 1-5 scale where 1 represents static material and 5 represents interactive material). Many students particularly appreciated modifying code snippets directly in the browser, as this hands-on approach facilitated better understanding and retention of programming concepts. The evaluation of slides quality conducted by students yielded consistently high scores across all aspects (M = 4.3; 95% CI ± 0.2), with convenience of use and performance being particularly appreciated, with the absolute majority of respondents rating them 5 out of 5.

# 5 Limitations and conclusions

In this study, we gathered some preliminary data points about the effectiveness and educational impact of ICPs in various contexts. While we aimed for this study to be as comprehensive as possible, there still are some methodological limitations about the study's generalisability and significance. From a significance point of view, though we incorporated descriptive and inferential statistics, our findings should be read as qualitative. So far, this has been a conscious choice: since we wanted to not only test the ICPs' effectiveness, but also to find what elements might have needed a redesign, we tried to create an experimental setting that allowed us to observe as many variables as possible. The qualitative approach we adopted let us achieve our results, observe subtle interactions and contextual elements, and establish a replicable process for observation; at this stage of our work, however, we do not expect our findings to be equally replicable.

Readers should keep two additional caveats in mind when reading our results. First, that all of our interventions were performed in the same location, the University of Trento. While our results are encouraging, part of our interventions' success may be due to our tool fitting our context all too well. Second, all of our interventions were limited in time, and we have not tested what would happen if ICPs were proposed for the duration of a whole course, or as its only slideshow system. In other words, we cannot exclude that our success is not due — at least in part — to a novelty effect. It should be noted, though, that a novelty bias may not be too problematic: since ICPs are intended to support courses in introductory programming, of which every student plausibly only does one in their career. As long as the novelty of ICPs lasts for the duration of a course, the bias would not have a negative impact.

| Feature | Occurrences | Why |
|---|---|---|
| Stay in Browser & on-the-fly changes | 30 | You don't need any other software or IDE<br>It's convenient and it saves time<br>It helps maintaining focus<br>Doesn't require space<br>It's easier to follow the lecture<br>ICPs automatically save my changes<br>It allows you to execute code in other devices<br>On-the-fly changes help understanding the code better |
| Move freely | 17 | Check previous or next exercises<br>I can focus more on unclear concepts |
| Exercises drawing attention to the right concepts | 9 | They allowed me to challenge myself<br>They allowed me to understand which were my knowledge gaps<br>I find it a good way to learn how to program |
| Compare code | 6 | It's easier when using a standardized environment |
| Other answers | 3 | - |

**Table 2: Responses to: "What do you think is the most useful feature of ICPs?"**

We have also observed several inherent limitations of ICP that are due to how the tool has been designed and architected. When students do not use Google Chrome or Firefox as their browser, for example, they are bound to encounter bugs, and the most reasonable solution is to install a supported browser. While this solution helps, it adds an extra technical step, and re-creates the problem that ICP intended to solve, i.e., the installation of an additional application. We see two mitigation strategies: on the one hand, we can work towards expanding browser compatibility; on the other hand, since some of the issues that are found with unsupported browsers are due to not all browsers implementing standard specifications such as SharedWorkers, the simple passing of time while browsers catch up to the standards may passively solve some of the challenges we encountered.

While we also aimed ICPs to improve accessibility compared to PDF slides and we have some observations that seem to support our hypothesis, we have not performed thorough accessibility testing. Future work may investigate this direction more in depth, and is likely to elicit several ideas that would contribute to the improvement of the ICPs' accessibility. In our findings, we also received clear indications that while students generally found the ICPs' UI to be intuitive, we need to improve some of the UI, including tabs handling and scaffolded exercises.

Our immediate avenues for future work go primarily towards further development of ICPs. One of the key areas of intervention that we see is that, so far, we developed ICPs for PC web browsers, while many students use mobile devices such as smartphones or tablets. Since ICPs are HTML slides, getting at least the basic functionalities of the system to work on mobile devices should be trivial, but we foresee many challenges in getting all functionalities to work, and in obtaining reasonable performance. When we shared with colleagues our preliminary findings, including during our presentation of ICPs at the SIGCSE 2024 Demo sessions [1], the feedback we received mainly went in three directions: (1) some colleagues wished for us to add support for more language or functionalities such as debuggers, which is something that we are already working on and that readers and users may already contribute to in our public repositories; (2) some colleagues asked for a system to create slides, or convert existing slides to ICPs, which is also something we already have a simple prototype available for; (3) yet others wished for some sort of a "multiplayer" or "shared editing" mode. We find this last suggestion to be particularly fascinating: since ICPs are designed to run without the need of a separated server, and to keep their "frugal" character, a particularly interesting direction may be to work on peer-to-peer collaborative editing, for example using a system similar to LocalSend or SyncThing that foresees both coordination servers, direct connection, and local discoverability.

In spite of some limitations and need for further development, we still see this first deployment of ICPs as a success. Students seemed to understand well how to interact with ICPs with no training, and they seemed to highly appreciate its functionalities. The flow of the classes, as we observed, was also minimally disrupted, suggesting that we were successful in creating a non-disruptive education technology. Our hope is that with future work, ICPs, or other technologies that follow similar patterns, may contribute to delivering successful lectures without forcing a change of practices, creating instead an incremental — and sustainable — evolution of existing pedagogical practices.

## Acknowledgments

We wish to thank all the teachers that decided to participate in the first field testing of ICPs, namely Radoslaw Niewiadomski, Michele Tizzoni, David Leoni, and Luca Bosotti.

## References

[1] Lorenzo Angeli, Luca De Menego, and Maurizio Marchese. 2024. Embedding Executable Code in Slides for Introductory Programming: The Case of Interactive Code Playgrounds. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1923. https://doi.org/10.1145/3626253.3635426

[2] Lorenzo Angeli, Juan José Jara Laconich, and Maurizio Marchese. 2020. A Constructivist Redesign of a Graduate-level CS Course to Address Content Obsolescence and Student Motivation. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, Portland OR USA, 1255–1261. https://doi.org/10.1145/3328778.3366910

[3] Konstantina Chrysafiadi and Maria Virvou. 2013. Dynamically Personalized E-Training in Computer Programming and the Language C. *IEEE Transactions on Education* 56, 4 (2013), 385–392. https://doi.org/10.1109/TE.2013.2243914

[4] Sue Clegg, Alison Hudson, and John Steel. 2003. The emperor's new clothes: Globalisation and e-learning in higher education. *British journal of sociology of education* 24, 1 (2003), 39–53.

[5] Keri Facer and Neil Selwyn. 2021. Digital Technology and the Futures of Education. https://unesdoc.unesco.org/ark:/48223/pf0000377071

[6] Pamela Fox. 2022. The Benefits of HTML Slides for Programming Lectures. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2* (New York, NY, USA, 2022-03-03) *(SIGCSE 2022)*. Association for Computing Machinery, 1055. https://doi.org/10.1145/3478432.3499246

[7] Thomas F. Griffin and Zack Jourdan. 2012. Educational Use Cases for Virtual Machines. In *Proceedings of the 50th Annual Southeast Regional Conference* (New York, NY, USA, 2012-03-29) *(ACM-SE '12)*. Association for Computing Machinery, 365–366. https://doi.org/10.1145/2184512.2184607

[8] Philip Guo. 2021. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event USA, 2021-10-10). ACM, 1235–1251. https://doi.org/10.1145/3472749.3474819

[9] Alex Hadwen-Bennett, Sue Sentance, and Cecily Morrison. 2018. Making Programming Accessible to Learners with Visual Impairments: A Literature Review. 2, 2 (2018), 3–13. https://doi.org/10.21585/ijcses.v2i2.25

[10] David P Harvie, Jason R Cody, Christopher Morrell, and Tanya T Estes. 2019. Using Virtual Machines to Enhance the Educational Experience in an Introductory Computing Course. In *Proceedings of the 20th Annual SIG Conference on Information Technology Education*. 28–32.

[11] Michael Henderson, Neil Selwyn, and Rachel Aston. 2017. What works and why? Student perceptions of 'useful'digital technology in university teaching and learning. *Studies in higher education* 42, 8 (2017), 1567–1579.

[12] Burke Johnson and Lisa A Turner. 2003. Data collection strategies in mixed methods research. *Handbook of mixed methods in social and behavioral research* 10, 2 (2003), 297–319.

[13] Oren Laadan, Jason Nieh, and Nicolas Viennot. 2010. Teaching operating systems using virtual appliances and distributed version control. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 480–484.

[14] A. Kelly Lane, Jacob D. McAlpin, Brittnee Earl, Stephanie Feola, Jennifer E. Lewis, Karl Mertens, Susan E. Shadle, John Skvoretz, John P. Ziker, Brian A. Couch, Luanna B. Prevost, and Marilyne Stains. 2020. Innovative Teaching Knowledge Stays with Users. 117, 37 (2020), 22665–22667. https://doi.org/10.1073/pnas.2012372117

[15] David J Malan. 2013. From cluster to cloud to appliance. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. 88–92.

[16] David J Malan. 2022. Standardizing Students' Programming Environments with Docker Containers: Using Visual Studio Code in the Cloud with GitHub Codespaces. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2*. 599–600.

[17] David R McIntyre and Francis G Wolff. 1998. An experiment with WWW interactive learning in university education. *Computers & Education* 31, 3 (1998), 255–264.

[18] Fisseha Mikre. 2011. The roles of information communication technologies in education: Review article with emphasis to the computer and internet. *Ethiopian Journal of Education and Sciences* 6, 2 (2011), 109–126.

[19] Missy Morton and David Mills. 2013. Ethnography in education. *Ethnography in Education* (2013), 1–200.

[20] Aboubakar Mountapmbeme, Obianuju Okafor, and Stephanie Ludi. 2022. Addressing Accessibility Barriers in Programming for People with Visual Impairments: A Literature Review. *ACM Transactions on Accessible Computing (TACCESS)* 15, 1 (2022), 1–26.

[21] Bonnie Nardi, Bill Tomlinson, Donald J. Patterson, Jay Chen, Daniel Pargman, Barath Raghavan, and Birgit Penzenstadler. 2018. Computing within Limits. 61, 10 (2018), 86–93. https://doi.org/10.1145/3183582

[22] Marc Prensky. 2008. The role of technology. *Educational Technology* 48, 6 (2008), 1–3.

[23] José-Manuel Sáez-López, Marcos Román-González, and Esteban Vázquez-Cano. 2016. Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education* 97 (2016), 129–141.

[24] Joel T. Schmidt and Min Tang. 2020. *Digitalization in Education: Challenges, Trends and Transformative Potential.* Springer Fachmedien Wiesbaden, Wiesbaden, 287–312. https://doi.org/10.1007/978-3-658-28670-5_16

[25] Neil Selwyn. 2007. The use of computer technology in university teaching and learning: a critical perspective. *Journal of computer assisted learning* 23, 2 (2007), 83–94.

[26] Volker Sorge, Akashdeep Bansal, Neha M Jadhav, Himanshu Garg, Ayushi Verma, and Meenakshi Balakrishnan. 2020. Towards generating web-accessible STEM documents from PDF. In *Proceedings of the 17th International Web for All Conference*. 1–5.

[27] Marko Teräs, Juha Suoranta, Hanna Teräs, and Mark Curcher. 2020. Post-Covid-19 education and education technology 'solutionism': A seller's market. *Postdigital Science and Education* 2, 3 (2020), 863–878.

[28] Marko Teräs, Hanna Teräs, Patricia Arinto, James Brunton, Daryono Daryono, and Thirumeni Subramaniam. 2020. COVID-19 and the push to online learning: Reflections from 5 countries. (2020).

[29] Sander Valstar, William G Griswold, and Leo Porter. 2020. Using devcontainers to standardize student development environments: An experience report. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 377–383.

[30] Wim Vanderbauwhede. 2023. Frugal Computing–On the need for low-carbon and sustainable computing and the path towards zero-carbon computing. *arXiv preprint arXiv:2303.06642* (2023).

[31] Guoqing Xu, Nick Mitchell, Matthew Arnold, Atanas Rountev, and Gary Sevitsky. 2010. Software Bloat Analysis: Finding, Removing, and Preventing Performance Problems in Modern Large-Scale Object-Oriented Applications. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (Santa Fe New Mexico USA, 2010-11-07). ACM, 421–426. https://doi.org/10.1145/1882362.1882448

[32] IS Zinovieva, VO Artemchuk, Anna V Iatsyshyn, OO Popov, VO Kovach, Andrii V Iatsyshyn, YO Romanenko, and OV Radchenko. 2021. The use of online coding platforms as additional distance tools in programming education. In *Journal of physics: Conference series*, Vol. 1840. IOP Publishing, 012029.

[33] Mohammad Zohrabi. 2013. Mixed method research: Instruments, validity, reliability and reporting findings. *Theory and practice in language studies* 3, 2 (2013), 254.