RESEARCH-ARTICLE

# WebTigerPython: A Low-Floor High-Ceiling Python IDE for the Browser

**CLEMENS BACHMANN**, Swiss Federal Institute of Technology, Zurich, Zurich, ZH, Switzerland

**ALEXANDRA MAXIMOVA**, Swiss Federal Institute of Technology, Zurich, Zurich, ZH, Switzerland

**TOBIAS KOHN**, Karlsruhe Institute of Technology, Karlsruhe, Baden-Wurttemberg, Germany

**DENNIS KOMM**, Swiss Federal Institute of Technology, Zurich, Zurich, ZH, Switzerland

**Open Access Support** provided by:

**Swiss Federal Institute of Technology, Zurich**

**Karlsruhe Institute of Technology**

**PDF Download**
**3770762.3772670.pdf**
**19 March 2026**
**Total Citations:** 0
**Total Downloads:** 286

**Published:** 17 February 2026

**Citation in BibTeX format**

SIGCSE TS 2026:The 57th ACM Technical Symposium on Computer Science Education
*February 18 - 21, 2026*
*MO, St. Louis, USA*

**Conference Sponsors:**
SIGCSE

# WebTigerPython

A Low-Floor High-Ceiling Python IDE for the Browser

Clemens Bachmann
clemens.bachmann@inf.ethz.ch
Department of Computer Science
ETH Zurich
Zurich, Switzerland

Alexandra Maximova
alexandra.maximova@inf.ethz.ch
Department of Computer Science
ETH Zurich
Zurich, Switzerland

Tobias Kohn
tobias.kohn@kit.edu
Department of Computer Science
Karlsruhe Institute of Technology
Karlsruhe, Germany

Dennis Komm
dennis.komm@inf.ethz.ch
Department of Computer Science
ETH Zurich
Zurich, Switzerland

## Abstract

The shift to BYOD (bring your own device) policies at schools requires browser-based programming tools that balance accessibility and functionality. We introduce WebTigerPython, a Python IDE combining novice-friendly features (Turtle graphics, robotics, error messages) with advanced capabilities (NumPy, Matplotlib). Its client-side execution and web worker architecture ensure non-blocking interactivity.

Python code is run in WebAssembly, performing only about three times slower than native CPython but significantly faster than other web-based IDEs to which we compared it. Deployed in classrooms with 800+ daily users, WebTigerPython supports offline use, URL-based sharing of code, and aligns with existing curricula—demonstrating how web tools can rival local IDEs without compromising power or accessibility.

## CCS Concepts

• **Social and professional topics → K-12 education**; • **Applied computing → Interactive learning environments**.

## Keywords

CS education; K–12 education; programming education; educational robotics; Python

**ACM Reference Format:**
Clemens Bachmann, Alexandra Maximova, Tobias Kohn, and Dennis Komm. 2026. WebTigerPython: A Low-Floor High-Ceiling Python IDE for the Browser. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1 (SIGCSE TS 2026), February 18–21, 2026, St. Louis, MO, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770762.3772670

## 1 Introduction

A wide range of programming environments provide an engaging entry point for novice programmers. In a K–12 context, however, locally installed software poses challenges—especially as many schools adopt "bring your own device" (BYOD) policies or rely on web-oriented devices. As a result, more educational tools, including programming IDEs, are now browser-based, requiring nothing more than a web browser and an internet connection.

We introduce WebTigerPython, a browser-based Python IDE built on Pyodide [38], a WebAssembly port of CPython that runs Python code entirely in the browser. To keep the interface responsive [6], WebTigerPython executes code in a web worker. Using PixiJS, we implemented a graphics engine supporting both Turtle graphics [8] and Matplotlib. WebTigerPython also supports executing programs on single-board computers such as the micro:bit, Calliope mini, and the corresponding Maqueen and Calli:bot robots via WebUSB API [10].

WebTigerPython showcases the feasibility of providing fully fledged educational programming environments as a web application that may run on virtually any device.

Our main contributions are:

- **Web-based solution.** We translated the full stack of our desktop-based Python libraries to work in the browser—including graphics, animation, audio, and robotics—, allowing for existing textbooks to be used with new (web-oriented) devices such as tablet computers.
- **Low floor.** We took great care to lower the entry barrier as much as possible. With WebTigerPython, you can just open the website and start coding. The user interface is very minimalistic, uncluttered, and beginner friendly. Moreover, WebTigerPython comes with novice-friendly error messages, localisation, a built-in documentation, and many other features. The interface is shown in Figure 1.
- **High ceiling.** We integrated support for more advanced Python libraries such as NumPy and Matplotlib, which raises the ceiling of the IDE and demonstrates that a single implementation can serve a wide range of applications and educational needs. Compared with other tools, our design is very efficient, especially for computationally intensive applications.

WebTigerPython is open source (published under the MPLv2 license) and can be accessed by anyone using any modern web browser via `https://webtp.ethz.ch`. The repository with the sources can be found at `https://gitlab.inf.ethz.ch/public-dkomm/webtp/webtp-core`.

## 2 Related Work

Programming environments with novice K–12 programmers as target audience, such as Greenfoot for Java [27], or block-based approaches like Alice [11], Scratch [42], or Snap [33], have a long history in the educational landscape [24].

In this paper we focus on Python as the programming language. There is a large number of educational Python IDEs that need to be installed locally, e.g., Mu [46], TigerJython [26], or Thonny [1], the latter two of which come also with support for educational robotics. Professional tools like PyCharm [21] and Jupyter Notebooks are also widely used in educational settings [22].

However, with the proliferation of tablet computers in schools and the diversity of devices and platforms caused by BYOD policies, installing such software tools in school settings becomes increasingly hard or even infeasible.

Web applications promise a solution to this dilemma by leveraging the ubiquitous browsers as a unified platform.

Early web applications acted primarily as interfaces to software run on a server—a model still used by some browser-based programming IDEs such as Replit.com [41], Google Collab [13], and the Online Python Tutor [15]. These tools typically execute the entire program on the server, collect the output, and display the result in the browser. In Python Tutor this comes with the added benefit of being able to step forward and backward in time through the program execution. However, real-time user interaction is not feasible in this setup, unless approximated via remote desktops—as long as latency is not an issue. Moreover, server-based tools are expensive [20], require a stable internet connection, and raise concerns about security (in particular in self-hosted setups) or long-term availability (when relying on external services). The latter is exemplified by the recent discontinuation of Replit Teams for Education [40].

In recent years, JavaScript has matured into a platform capable of running complex web applications directly in the browser. WebAssembly [16] has further strengthened this trend towards *client-side* execution through better support for applications written in C and similar languages. However, translating desktop applications to the web remains non-trivial due to JavaScript's single-threaded and blocking execution model, which can freeze the UI [6]. Running code in a web worker mitigates this by isolating execution, but prohibits the application from any unmediated user interaction by restricting access to the DOM (including the UI).

Early web-based Python IDEs such as CodeSkulptor [43], the CMU Code Academy [44], Pythy [35], Trinket [48], Strype [50], and WebTigerJython [47] rely on reimplementations of Python in JavaScript, e.g., Brython [7] or Skulpt [14], where Python code is translated to JavaScript for subsequent execution. Besides performance issues, reimplementations struggle keeping up with the development of the Python language itself and lack support for packages such as, e.g., NumPy.

Heralded by projects such as PyPy.js [25], the second generation of Python web applications relies on cross compilation of existing code bases in system languages like C to WebAssembly. This approach allows to use the actual CPython interpreter compiled to WebAssembly as done by, e.g., Pyodide [38]. A number of programming environments build on top of this approach, e.g., Basthon [5], FutureCoder [17], JupyterLite [23], Papyros [36], PyodideU [20], as well as WebTigerPython, which we present in this paper.

A Python program run with Pyodide in the main thread blocks the user interface due to WebAssembly not supporting asynchronous execution directly. PyodideU [20] offers two possible solutions to this issue. When run inside the browser's main thread, the Python interpreter itself is regularly interrupted by throwing exceptions. Alternatively, the Python code is modified by making functions asynchronous and awaiting them (cf. Baxter et al. [6]). Our approach differs significantly from both of these alternatives.

Finally, some educational tools like MakeCode [4] and the micro:bit Python Editor [30, 31] focus on code execution on external devices, and give the possibility to easily transfer code to them. This feature also finds widespread usage in schools in our area, so WebTigerPython also supports this. More specifically, program code can be downloaded to such a device and executed on it—or the code can be transfered on the single click of a button if the browser supports WebUSB—, borrowing from the micro:bit Python Editor.

## 3 Setting and Features

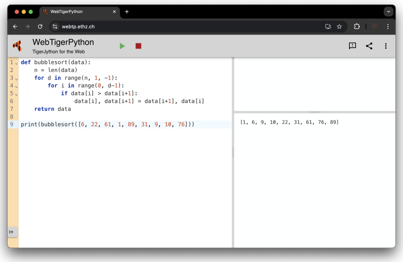
With mandatory CS classes in high school in Switzerland, there is a need for an accessible, novice-friendly programming environment that can be used to teach programming following established curricula.

### 3.1 Starting Point

The authors have a lot of experience with introducing Python programming in high schools across Switzerland. This includes conducting hundreds of training and advanced training courses for (prospective) CS teachers and maintaining close ties with many schools. In addition to developing teaching materials—both free online tutorials and textbooks published by a leading German-speaking educational publisher [3, 12, 18]—one of the authors also developed TigerJython [26], a widely used, locally installed Python IDE that runs in the Java Virtual Machine.

However, recently, an increasing number of teachers reported issues that can be summarized as follows:

(1) With the rise of portable devices and BYOD strategies in schools, more and more students now use tablets—making a web application the only viable solution in the long run.
(2) However, many teachers explicitly stated that they would like to continue using the above-mentioned textbooks. These rely on specific libraries to allow novices to use Turtle graphics, educational robotics, etc.
(3) At the same time, as CS—and programming in particular—is integrated into more and more curricula (as early as primary school), some teachers would like to cover advanced concepts in upper secondary school; say, running computationally intensive programs using CPython libraries like NumPy, which is not possible in the Java-based solution used so far.

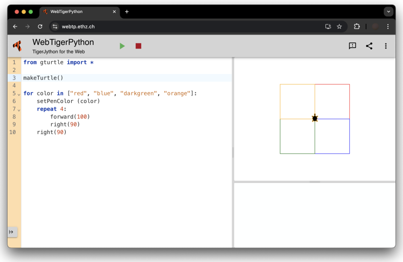


```python
def bubblesort(data):
    n = len(data)
    for d in range(n, 1, -1):
        for i in range(0, d-1):
            if data[i] > data[i+1]:
                data[i], data[i+1] = data[i+1], data[i]
    return data

print(bubblesort([6, 22, 61, 1, 89, 31, 9, 10, 76]))
```

```
[1, 6, 9, 10, 22, 31, 61, 76, 89]
```

**a: A simple implementation of Bubblesort**

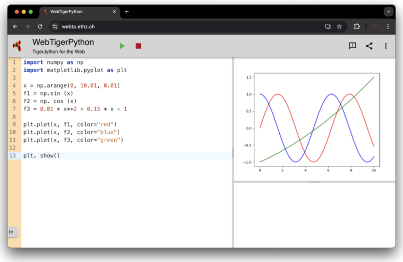


```python
from gturtle import *

makeTurtle()

for color in ["red", "blue", "darkgreen", "orange"]:
    setPenColor (color)
    repeat 4:
        forward(100)
        right(90)
    right(90)
```

**b: Drawing shapes with the Turtle**



```python
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10.01, 0.01)
f1 = np.sin (x)
f2 = np. cos (x)
f3 = 0.01 * x**2 + 0.15 * x - 1

plt.plot(x, f1, color="red")
plt.plot(x, f2, color="blue")
plt.plot(x, f3, color="green")

plt. show()
```

**c: Plotting functions with Matplotlib**

**Figure 1: Different screenshots of WebTigerPython**

## 3.2 Features

The above-mentioned feedback from teachers brought us to a clear catalogue of requirements for WebTigerPython to satisfy all of the following three points.

*(1) Web-based solution.* WebTigerPython runs on virtually any device, such as laptops, desktop and tablet computers. The only requirement on the user's end is a modern web browser.

We included the following features to make the tool easy to use and utilize the advantages of web-based applications.

- *Run out of the box.* There is no necessity to sign up or log in, thereby eliminating technical and privacy issues.
- *Novice-friendly user interface.* Moreover, the uncluttered user interface allows starting right away with coding without any kind of configuration or even creating files.
- *Localisation.* WebTigerPython comes with full language localisation, currently supporting English, French, German, Italian, Slovak, and Spanish.
- *Built-in documentation.* Inspired by block-based IDEs, such as Scratch, and the micro:bit Python Editor [30], we provide documentation for supported commands in a practical side panel, from where exemplary code snippets can be easily dragged and dropped onto the editor.
- *Sharing code.* Programs can be shared in an easy way with others and across devices on a single mouse click (click Figures 1a to 1c for a demonstration).

*(2) Low floor.* We designed WebTigerPython with the same philosophy in mind that drove the development of the original TigerJython, allowing for a low entry barrier and minimizing cognitive load while programming.

- *Continuity.* WebTigerPython works seamlessly with existing textbooks and learning materials that are already in use in Switzerland. Differences between the previously used programming environment and WebTigerPython are kept to a minimum, allowing for a smooth transition.
- *Beginner-friendly Turtle graphics.* An established way to introduce programming to students revolves around Turtle graphics [8]. WebTigerPython offers a tailor-made Turtle library, which enhances Python's standard Turtle. An example is shown in Figure 1b. Nonetheless, the standard turtle is also available. Furthermore, the greatly varying sizes and pixel densities of screens pose a challenge for Turtle graphics, which usually rests on the assumption that a pixel represents a single Turtle step. Depending on the screen, Turtle images may be inappropriately scaled, not fitting into the available space or being too small. WebTigerPython therefore features an "unlimited" canvas that allows the user to zoom in/out or move the canvas around (using finger gestures on tablet devices). This is a clear departure from existing systems that have a fixed window, canvas, or image size to work with.
- *Physical computing.* The IDE also supports programming popular single-board computers such as the micro:bit and Calliope mini via WebUSB API [10], along with custom libraries to program the Maqueen and Calli:bot robots [29], providing hands-on experience with physical computing and educational robotics, as shown in Figure 2.
- *Enhanced feedback.* WebTigerPython provides tailor-made user-friendly, informative error messages that are easy to understand by novices.

*(3) High ceiling.* To complement the low floor, WebTigerPython is equipped with a high ceiling that allows experienced programmers to utilize the full power of advanced CPython libraries and event-handling.

- *Advanced Python programming.* Being built on Pyodide, WebTigerPython includes a fully fledged Python 3.13 environment supporting many advanced C-based libraries such as SciPy, NumPy, pandas, and Matplotlib—the latter being rendered directly in the browser (on the canvas), as can be seen in Figure 1c. This enables advanced students to tackle more complex topics in data analysis, scientific computing, and data visualisation.
- *Computationally intensive applications.* WebTigerPython leverages the compute performance provided by Pyodide, which often is within a factor of 3–5 times of CPython's performance [38, 51], and was in the order of around 14 times faster than a Skulpt-based implementation. This is important when dealing with computationally intensive tasks such as, e.g., computing a Mandelbrot set (Section 5.1).
- *Interactive graphics.* Keyboard and mouse interactions allow students to deal with input and create more engaging applications, such as, e.g., simple games.
- *Advanced Turtle graphics.* Turtle graphics is renowned for its animated drawing movement, allowing students to see the Turtle

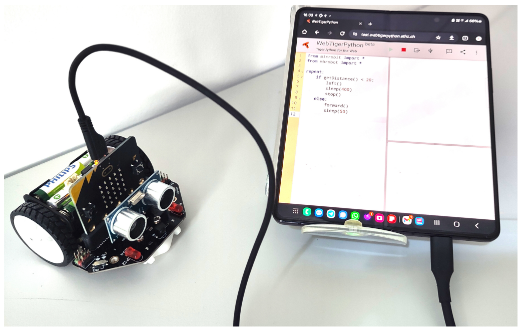


**Figure 2: Programming a Maqueen robot via tablet**

execute commands. However, teaching materials often disable movement to use Turtle graphics for data visualization, custom animations, and simple video games. Consequently, our implementation includes a *direct drawing* mode that eliminates Turtle movement animations, enabling faster performance.

- *Debugger with memory visualisation.* The IDE features a debugger that allows step-by-step execution and the ability to set breakpoints. It displays the current memory state, similar to the Python Online Tutor [15].

## 4 Implementation

We designed a new custom IDE, which is visually inspired by its predecessor WebTigerJython [47]. As mentioned above and shown in Figures 1a to 1c, the layout includes a code editor, a canvas for rendering graphical output, and an output field. We opted for a simple text output instead of a regular console since input is implemented through an interactive dialogue. When designing the user interface we adhered to usability heuristics [34] to minimize the cognitive load of the user; while also trying to make the IDE as touch friendly as possible. It can be used on any device that can run a modern web browser, though we recommend using it with a physical keyboard for the best experience.

Programs can be saved and opened via the File System API. Additionally, code is automatically preserved across page refreshes. To facilitate the sharing of programs, program code can be directly encoded into a URL. This is achieved through a 'share' button in the user interface, which generates a corresponding URL. Users can then send this URL to others or to different devices.

This method eliminates the need for a database backend on the server, reducing costs. However, it is important to note that the size of the programs is limited to the size of URLs the respective browser permits. To maximize the amount of code that can be shared and minimize link length, we employ the Lempel-Ziv algorithm for compression. Teacher feedback suggests that this limit is sufficient.

### 4.1 Python Execution

In Section 3.1, we formulated three issues that were inspired by feedback from teachers and drove the development of the IDE. From a technical point of view, we may add another challenge:

(4) Executing the Python code while not freezing the browser, enabling interactive graphics output, and making sure there are proper input capabilities.

WebTigerPython works with a *main thread* and a *web worker*. The main thread manages the user interface, while the web worker handles Python execution in Pyodide. Communication between the web worker and the main thread occurs via the *postMessage* API. As mentioned above, running Pyodide in a web worker is essential since Python execution in WebAssembly is blocking, and thus executing Python in the main thread would freeze the interface during execution, significantly harming the user experience (cf. Jefferson et al. [20]).

To interrupt or halt Python execution in the blocked web worker, we use a *shared array buffer* [32], a raw binary buffer accessible by both threads. For a specific value in the buffer, the execution pauses. This is necessary for functions such as rendering in the main thread or waiting for user input.

### 4.2 Turtle Library

The original Python Turtle is based on Tkinter, a Python binding of the *Tk GUI toolkit*, which is not usable in the browser. We decided to re-implement the Turtle library from scratch using the web-based graphics library PixiJS [37], a JavaScript library for rendering 2D graphics. It is not straightforward to access PixiJS directly from the web worker, adding to the challenge of communication between the web worker and the main thread.

We implemented most of the Turtle functionality directly in PixiJS. Since PixiJS is a very fast and highly optimized JavaScript rendering library, we decided to create an API that can be called from Python within our web worker. To minimize communication overhead between the two threads, multiple instructions can be bundled together, which is crucial if the Turtle is used to render custom animations or games that require fast drawing with no lag.

### 4.3 Interactivity

Users can interact with their programs either through the `input()` function or through mouse and keyboard events. The former opens a GUI dialogue, prompting the user to enter a string, which is then sent to Python and can be used within the program.

Additionally, we log keyboard events and interactions with the canvas. Events can be queried from Python in both a blocking and a non-blocking way. It is also possible to register callback functions that listen for events after program execution. However, a limitation is that callback functions cannot be executed during Python execution as Python is single-threaded. If events are triggered during the execution of Python code, they are added to an event queue and processed one by one as soon as the execution ends.

### 4.4 Data and Function Plotting

Although Matplotlib comes with an HTML 5 backend [45], we were not able to use it as we run Python in a web worker, which has no access to the DOM. To render plots, we use the AGG (Anti Grain Geometry) backend [2], a graphics toolkit available in Pyodide.

To display the image on the canvas, we first render it within Python and then pass it to the main thread, where it is displayed; a result is shown in Figure 1c. This allows Matplotlib and other libraries that build on it, such as Seaborn and NetworkX, to be used in WebTigerPython. However, animations and interactive features of Matplotlib are not yet supported.

## 4.5 Physical Computing

Single-board computers such as the micro:bit or Calliope mini can also be programmed with WebTigerPython. The Python code can be exported with MicroPython and loaded onto the external computer. Using the WebUSB API [10], it is possible to transfer the program directly using the browser on a single click.

The external computer can communicate with the IDE over a serial connection with the main thread, which is similar to the code execution in the web worker. This can then be used to display error messages and other output that cannot be properly shown on the external computer.

There is also support for extension boards for robots, such as the Maqueen or the Calli:bot, which can be programmed via pins. This process is cumbersome and not suitable for novices. Therefore, we provide additional abstraction libraries [29], which are also flashed to the single-board computer if needed.

We additionally added the possibility to simulate these robots. We used the simulator provided and used by the micro:bit Python editor [30], but we extended it with a playground where the extension boards—like the one shown in Figure 2—can be simulated. This makes robotics accessible to all devices to a certain extent.

## 5 Evaluation

We were able to implement all the functionality necessary to follow the established programming curriculum. Currently, WebTigerPython has an average number of around 800 unique users per day (and up to roughly 2 500), where "unique" is defined as different timestamped browsers that opened our application. We do not have an exact breakdown of the demographics; however, we conducted a survey where we gathered some insights about demographics which we present in Section 5.2.

Overall, we made WebTigerPython as accessible and easy to integrate into current curricula as possible. It is deployed as a progressive web app, allowing users to access it without limitations and install it on their devices without admin rights. It can be embedded as an iframe in web-based teaching materials. In addition, links that contain programs can be embedded in teaching materials as well, enabling teachers to streamline the distribution of code skeletons and examples. Ensuring one-click access to our IDE was crucial to avoid the pitfalls associated with application installations or logins [22, 49].

## 5.1 Benchmarks

To assess the computing power and rendering speed, we ran two benchmarks. We compared WebTigerPython to native CPython execution and two IDEs with client-side code execution: trinket.io, which is Skulpt-based, and PyodideU [20], which is also based on Pyodide. All tests were run on a MacBook Air with an M3 processor. We ran all tests three times and took the best of 3. Variance between all tests was below 1 % except for the ball simulation with trinket.io, where it was below 5 %.

*Compute Benchmarks.* To test the potential for computationally intensive applications, we computed a simple Mandelbrot set for an image of size 200×200 pixels, leading to the computation $z' = z^2 + c$

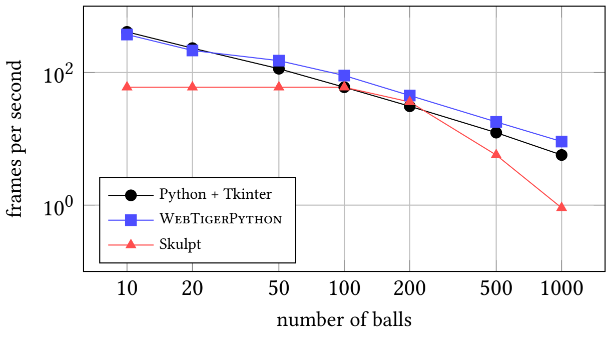


**Figure 3: Benchmark with animated balls**

being executed 4 000 000 times. CPython needed 0.16 s, WebTigerPython 0.38 s, PyodideU 118 s, and trinket.io 43 s. WebTigerPython is within 3 times of the performance of native Python and is more than 100 times faster than both Skulpt-based trinket.io and PyodideU. Particularly noteworthy is PyodideU's rather slow performance, which is in spite of also using Pyodide as the Python interpreter. In contrast to WebTigerPython, PyodideU runs the code inside the main thread. To keep the UI responsive, execution is regularly interrupted and resumed using exceptions that capture the current state [20].

*Graphics Benchmarks.* We additionally ran a graphics benchmark and managed to outperform the CPython implementation as shown in Figure 3. As a graphics benchmark, we ran an animation with a varying number of bouncing balls implemented in the turtle library. The size of the rendering area was 500 × 500 pixels. This benchmark demonstrates that animations still run smoothly in our IDE. Note that all implementations are capped at 60 fps due to hardware constraints. The Skulpt implementation crashed in the simulation with 5 000 balls. We could not include PyodideU in this benchmark, as there is no support for Turtle graphics. The Pyodide team states that Pyodide is 3–5 times slower than native Python [39, 51]; however, since most of the rendering happens in PixiJS, we were confident that it would be possible to achieve a similar rendering performance.

We also did not include the Brython Turtle since the feature scope is very limited. In particular, events and animation control (which is tested in the benchmark) are not supported.

## 5.2 Survey

We surveyed 16 teachers who used WebTigerPython during the last two years. We distributed our survey at a local event and using a mailing list both targeted at CS teachers. Most of them have only used the IDE in a few classes, but this goes up to 4 teachers who employed it with more than 200 students each. The survey showed that WebTigerPython was used mainly within the age group 12 and above with most students being aged 15–17.

Teachers often used WebTigerPython for visual programming like Turtle graphics, but also Matplotlib was used on a regular basis.

Additionally, teachers frequently used and liked the feature of sharing code snippets by URL, which facilitates the integration of our tool in any digital classroom management (CRM) system.

**Table 1: A feature comparison of web-based Python IDEs that support front-end code execution**

| IDE | trinket.io[1] | micro:bit Python Editor | futurecoder | Papyros | Basthon | JupyterLite | pyodideU | WebPython |
|---:|:-:|:-:|:-:|:-:|:-:|:-:|:-:|:-:|
| Python version | 3[2] | 3.4.0 (MicroPython) | 3.12.1 | 3.13.2 | 3.11.2 | 3.12.7 | 3.9.5 | 3.13.2 |
| visual computing | ✓ | 5 × 5 LED Matrix | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Matplotlib | ✓[3] | | | ✓[4,5] | ✓[4] | ✓[4,5] | | ✓[4,5] |
| Turtle | ✓ | | | | ✓[6] | | | ✓ |
| audio | | ✓ | | | | | | ✓ |
| robotics | | ✓[7] | | | | | | ✓[7] |
| robotics accessories libraries | | | | | | | | ✓[7] |
| events | ✓ | ✓ | | | | | ✓ | ✓ |
| debugger | | | ✓ | ✓ | | | ✓ | ✓ |

[1] Version with front-end code execution
[2] Not further specified on their website; the current version of Skulpt supports the grammar of 3.7.3, but many features are not implemented
[3] Limited feature set reimplemented in Skulpt
[4] Matplotlib events or [5]Matplotlib animations are not supported
[6] Limited feature set where events and animations are not supported
[7] Including simulation

Seven teachers stated they embedded WebTigerPython into other tools such as Moodle or OneNote. Four teachers stated that they integrated our tool into other websites via iframe.

Teachers mentioned that they also worked with Thonny [1], trinket.io [48], XLogoOnline [19], and Jupyter Notebooks [23]. When being compared to other educational IDEs, the main benefit of WebTigerPython mentioned was its simplicity. It was noted that it can be used without any prior setup and preparation and has no complicated interface that overloads the students with advanced functionality.

## 6 Limitations and Future Work

While the feature set of WebTigerPython is large, it is not possible to use all features on all devices. For instance, physical computing is merely supported with limitations on some devices since only some browsers support the WebUSB API [10] (Chrome, Edge, etc.) while others do not (Safari, Firefox).

Moreover, since we are executing the Python code inside an isolated web worker without access to the user interface, SDL-based packages like PyGame [9] are not directly compatible, although they are supported by Pyodide. We implemented a prototype of PyGame which runs in the web worker with offscreen rendering; however, this did not find widespread adoption yet.

Work is also ongoing to improve basic functionality and to implement additional Python modules. Although support for multiple files is possible in principle, it is not yet exposed in the current user interface. However, the IDE is mature enough to be used in classrooms and serves as a basis for research in CS education and usability.

The current implementation of WebTigerPython does not provide any way to store data on the server. A future goal is to create cloud and git integration. Students should be able to open and save files from their preferred web storage and work on their code in WebTigerPython.

WebTigerPython does not come with any CRM system; however, we provide detailed information on how WebTigerPython can be included into other CRM tools. The conducted survey indicates that this is already taking place.

The current evaluation demonstrates that existing textbooks in Switzerland can be used with WebTigerPython, and we have received encouraging feedback from teachers. Further evaluation of specific features will be conducted in the following months.

## 7 Conclusion

With WebTigerPython, we were able to fulfil all requirements formulated in Section 3.1 (and Section 4.1), namely to build a *(1) web-based solution* allowing *(4) Python execution while staying responsive* with both a *(2) low floor*—in terms of accessibility, didactic concepts, enhanced error messages, etc.—and a *(3) high ceiling*—in terms of efficiency and support for advanced CPython libraries.

We were able to support a large feature set that is unique in this combination, as demonstrated by Table 1. WebTigerPython is currently actively used in many classrooms in Switzerland and other countries with increasing numbers (currently up to a 2 500 unique visitors a day). Feedback from teachers is very positive with more systematic evaluations to follow.

For our purposes, running Pyodide in a web worker turned out to be surprisingly efficient as is demonstrated by a set of preliminary benchmarks.

There are more and more teaching materials that make use of the tool, and we are therefore confident that it will find a wider adoption by students and teachers.

## Acknowledgements

The authors thank Cédric Donner for help with the initial design choices. We are grateful to Alex Hall, who wrote libraries that simplified the web worker communication, and to everyone who contributed to the code base. Finally, we also thank Julia Bogdan for the initial Turtle prototype, and Daniel Barot for his work on the canvas interactivity.

More on the history of TigerJython and WebTigerPython (and people playing central roles) can be found in a recent blog post [28].

## References

[1] Annamaa, A. Thonny, a Python IDE for learning programming. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (2015), pp. 343–343.

[2] Anti-Grain Geometry (AGG). `https://agg.sourceforge.net/antigrain.com/doc/introduction/introduction.agdoc.html`. Last accessed: 12-November-2025.

[3] Arnold, J., Donner, C., Hauser, U., Hauswirth, M., Hromkovič, J., Kohn, T., Komm, D., Maletinsky, D., and Roth, N. *Informatik: Programmieren und Robotik.* Klett und Balmer Verlag, 2021.

[4] Ball, T., Chatra, A., de Halleux, P., Hodges, S., Moskal, M., and Russell, J. Microsoft MakeCode: embedded programming for education, in blocks and typescript. In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E* (New York, NY, USA, 2019), SPLASH-E 2019, Association for Computing Machinery, pp. 7–12.

[5] Basthon. `https://basthon.sct.pf`. Last accessed: 12-November-2025.

[6] Baxter, S., Nigam, R., Politz, J. G., Krishnamurthi, S., and Guha, A. Putting in all the stops: Execution control for javascript. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2018), pp. 30–45.

[7] Brython. `https://brython.info`. Last accessed: 12-November-2025.

[8] Caspersen, M. E., and Christensen, H. B. Here, there and everywhere – on the recurring use of Turtle graphics in CS 1. In *ACM International Conference Proceeding Series* (2000), vol. 8, pp. 34–40.

[9] Community, P. Pygame Community Edition, 2021. Last accessed: 12-November-2025.

[10] Contributors, M. WebUSB API. `https://developer.mozilla.org/en-US/docs/Web/API/WebUSB_API`, 2023. Last accessed: 12-November-2025.

[11] Daly, T., and Wrigley, E. *Learning Java through Alice 3.* CreateSpace Independent Publishing Platform, 2015.

[12] Gallenbacher, J., Hromkovič, J., Komm, D., Lacher, R., and Pierhöfer, H. *Informatik: Algorithmen und Künstliche Intelligenz.* Klett und Balmer Verlag, 2023.

[13] Google. Google Colaboratory. `https://colab.research.google.com/`, n.d. Last accessed: 12-November-2025.

[14] Graham, S. Skulpt. `https://skulpt.org/`. Last accessed: 12-November-2025.

[15] Guo, P. J. Online Python Tutor: embeddable web-based program visualization for CS education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), SIGCSE '13, Association for Computing Machinery, pp. 579–584.

[16] Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and Bastien, J. Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2017), pp. 185–200.

[17] Hall, A. Futurecoder. `https://futurecoder.io`. Last accessed: 12-November-2025.

[18] Hromkovič, J., and Kohn, T. *Einfach Informatik 7–9: Programmieren.* Klett und Balmer Verlag, 2018.

[19] Hromkovič, J., Serafini, G., and Staub, J. XLogoOnline: a single-page, browser-based programming environment for schools aiming at reducing cognitive load on pupils. In *Informatics in Schools: Focus on Learning Programming: 10th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2017, Helsinki, Finland, November 13-15, 2017, Proceedings 10* (2017), Springer, pp. 219–231.

[20] Jefferson, T., Gregg, C., and Piech, C. PyodideU: Unlocking Python entirely in a browser for CS1. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education, SIGCSE 2024, Volume 1, Portland, OR, USA, March 20-23, 2024* (2024), B. Stephenson, J. A. Stone, L. Battestilli, S. A. Rebelsky, and L. Shoop, Eds., Association for Computing Machinery, pp. 583–589.

[21] JetBrains. PyCharm – The Python IDE for data science and web development, 2024. Last accessed: 12-November-2025.

[22] Johnson, J. W. Benefits and pitfalls of Jupyter notebooks in the classroom. In *Proceedings of the 21st Annual Conference on Information Technology Education* (New York, NY, USA, 2020), SIGITE '20, Association for Computing Machinery, pp. 32–37.

[23] JupyterLite. `https://jupyterlite.readthedocs.io/en/latest/`. Last accessed: 12-November-2025.

[24] Kelleher, C., and Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM computing surveys (CSUR) 37*, 2 (2005), 83–137.

[25] Kelly, R. F. PyPy.js. `https://github.com/pypyjs/pypyjs`, 2013.

[26] Kohn, T., and Manaris, B. Tell me what's wrong: A Python IDE with error messages. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2020), SIGCSE '20, Association for Computing Machinery, pp. 1054–1060.

[27] Kölling, M. The Greenfoot programming environment. *ACM Trans. Comput. Educ. 10*, 4 (2010).

[28] Komm, D. Programming education made in Switzerland: The story of WebTigerPython. `https://svia-ssie-ssii.ch/interface/programming-education-made-in-switzerland-the-story-of-webtigerpython/`, 2025. Last accessed: 12-November-2025.

[29] Komm, D., Regez, A., Hauser, U., Gassner, M., Lütscher, P., Puchegger, R., and Kohn, T. Problem solving and creativity: Complementing programming education with robotics. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2020, Trondheim, Norway, June 15–19, 2020* (2020), M. N. Giannakos, G. Sindre, A. Luxton-Reilly, and M. Divitini, Eds., Association for Computing Machinery, pp. 259–265.

[30] Micro:bit Educational Foundation. Microbit Python Editor. `https://python.microbit.org/v/3`. Last accessed: 12-November-2025.

[31] Micro:bit Educational Foundation and Calliope and Lulububu. Calliope Mini Python Editor. `https://python.calliope.cc/`. Last accessed: 12-November-2025.

[32] Mozilla Corporation. SharedArrayBuffer. `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/`. Last accessed: 12-November-2025.

[33] Mönig, J., and Harvey, B. Snap! – build your own blocks. `https://snap.berkeley.edu/`, 2011.

[34] Nielsen, J., and Molich, R. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1990), CHI '90, Association for Computing Machinery, p. 249–256.

[35] Online Python beta. `https://www.online-python.com`. Accessed: 26-June-2025.

[36] Papyros. `https://github.com/dodona-edu/papyros`. Ghent University. Last accessed: 12-November-2025.

[37] PixiJS. `https://pixijs.com/`. PixiJS team. Last accessed: 12-November-2025.

[38] Pyodide contributors and Mozilla. Pyodide. `https://github.com/pyodide/pyodide`. Last accessed: 12-November-2025.

[39] Pyodide Roadmap. `https://pyodide.org/en/0.26.1/project/roadmap.html`, 2024. Pyodide contributors and Mozilla. Last accessed: 12-November-2025.

[40] Replit. Update on teams for education. `https://blog.replit.com/update-on-teams-for-education`, 2025. Last accessed: 12-November-2025.

[41] Replit. The collaborative browser-based IDE, n.d. Last accessed: 12-November-2025.

[42] Resnick, M., Maloney, J. H., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. S., Silverman, B., and Kafai, Y. B. Scratch: programming for all. *Commun. ACM 52*, 11 (2009), 60–67.

[43] Rixner, S. `https://py3.codeskulptor.org`, 2024. Last accessed: 12-November-2025.

[44] Stehlik, M., Cawley, E., and Kosbie, D. CMU CS Academy: A browser-based, text-based introduction to programming through graphics and animations in python. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2020), SIGCSE '20, Association for Computing Machinery, p. 1420.

[45] Tandon, M. HTML5 <canvas> based renderer for Matplotlib in Pyodide. `https://blog.pyodide.org/posts/canvas-renderer-matplotlib-in-pyodide/`, 2020. Last accessed: 12-November-2025.

[46] Tollervey, N. H. Code with Mu: A simple Python editor for beginner programmers. `https://codewith.mu/`.

[47] Trachsler, N. WebTigerJython – A browser-based programming IDE for education. Master's thesis, ETH Zurich, 2018.

[48] trinket.io. `https://trinket.io/`. Last accessed: 12-November-2025.

[49] Tyrén, M., Carlborg, N., Heath, C., and Eriksson, E. Considerations and Technical Pitfalls for Teaching Computational Thinking with BBC Micro:Bit. In *Proceedings of the Conference on Creativity and Making in Education* (New York, NY, USA, 2018), FabLearn Europe'18, Association for Computing Machinery, pp. 81–86.

[50] Weill-Tessier, P., Kyfonidis, C., Brown, N., and Kölling, M. Strype: Bridging from blocks to Python, with micro:bit support. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2* (New York, NY, USA, 2022), ITiCSE '22, Association for Computing Machinery, pp. 585–586.

[51] Yurchak, R. Pyodide: Benchmark Py3.11 and Py3.10. `https://github.com/pyodide/pyodide/issues/3503`, 2023.