PyodideU: Unlocking Python Entirely in a Browser for CS1

Thomas Jefferson  
Stanford University  
tjj@stanford.edu

Chris Gregg  
Stanford University  
cgregg@stanford.edu

Chris Piech  
Stanford University  
piech@cs.stanford.edu

ABSTRACT
In this paper, we present an education-focused Python IDE and runtime library which can run entirely in desktop, laptop, tablet, and mobile device web browsers. Our solution provides features useful for an engaging CS1 course, and eliminates the need for a server-based runtime. We describe a new, open source, methodology for running interactive Python entirely in the browser by solving the “WebAssembly blocking problem,” a core technical challenge to a web-based Python solution.

Because our method enables Python entirely in the browser, it unlocks many new features. For example, students can share their code with others, without incurring extra costs to the instructors or institutions. Other features include line by line code highlighting as a program executes, highly intuitive interactive graphics, mouse and touch integration, and use of a wide selection of Python modules such as Numpy and Pandas. Currently, our IDE has been used in 5 classes, covering more than 10,000 students and teachers, with over 350,000 projects created. We found that students and instructors appreciated the variety of tools and abilities the IDE made possible. We benchmark the performance of running code with our method against other online Python solutions and we discuss the benefits and additional possibilities that our method allows, such as mobile device and/or offline code execution. We provide full free public access to our IDE and open source the core libraries which enable the conversion of student written Python to WebAssembly.

CCS CONCEPTS
• Social and professional topics → CS1: Computer science education; K-12 education; Model curricula; • Software and its engineering → Integrated and visual development environments.

KEYWORDS
Python, IDE, integrated development environment, web browser, web assembly, mobile, CS1

1 INTRODUCTION
Browser based environments offer great benefits for both educators and learners of introductory programming. Often, setting up local integrated development environments (IDEs) can be a difficult task for beginners and, at times, require a more advanced understanding than is available [24]. This problem can be especially pronounced in large CS1 courses, where custom instruction is difficult or not possible. Alternatively, browser based IDEs offer an opportunity to present a uniform, low barrier to entry option to start writing code. Traditionally, web based environments meant for non web native programming languages, such as Python, have faced issues regarding executing and interacting with user programs in a near native way.

In preparation for Code In Place [6, 26, 29], a large scale, online, CS1 course, we wanted to develop a system that allowed users to run their code entirely in the browser. We also wanted to give them access to commonly used Python libraries, an interactive and easy to use graphics library with mouse and touch support, and a debugging tool. It was a goal to design an interface that implemented these features in a way that would be intuitive and comfortable for both beginners and teachers.

1.1 Approaches to Python in a Browser
Currently, several solutions exist for running user written Python from the browser. The two most common include server based execution and transpilation to Javascript. While each of these systems offer a subset of functionality, they each have a series of shortcomings that can negatively affect a user’s programming experience.

A server based solution sends user code to a remote computer for the program to be run, then sends the results back for the user to view. This system allows for running complex programs and offers potential access to native Python features, like libraries. However, using a remote server increases the computational cost of the system, and the difficulty for interactive programs. Some current solutions have integrated highly interactive programs using web infrastructure like web sockets, but these services come at a cost, either to the user or the maintainer. Server reliance can also be a major issue for accessibility, as the performance of any interactive programs, and the ability to run them can be highly dependent on the user’s internet access.

Another option is transpilation to Javascript. With this solution, user code is transpiled from Python, to Javascript, then run directly in the browser. Removing the server is helpful as there is no longer a dependence on continued, uninterrupted internet access and communication. Transpilation, however, does not scale with the Python ecosystem, and limits the user’s ability to use external packages. Additionally, some of the most prominent and performant Python to JS transpilers consistently have issues with blocking synchronous user input. This issue manifests through bugs, like infinite loops, crashing the user’s browser, and inability to use Python’s...
native input function without blocking interaction with the rest of the web page. Work-arounds, like running the user’s program in a web worker[14], exist to solve the infinite loop crashing problem, however these solutions come with trade-offs. For example, running the code in a web worker can negatively impact interactive program performance, and make accepting synchronous user text input impossible.

Finally, a newer option includes Python-to-WebAssembly interpretation. WebAssembly is a low level, byte code instruction set that can be run directly from a browser [22]. With this solution, user written Python can be interpreted and compiled to byte code. This allows for a fast runtime and increasingly scalable solution. However, WebAssembly is known to be unblockable, and has some of the same issues as transpilation, including infinite loop crashing in the browser’s main thread, and no synchronous input. For this reason, it is difficult to use an unaltered Python-to-WebAssembly provider like Pyodide[31] to run student code.

Each of these solutions offer a subset of useful features for Python learning and education, but also come with trade-offs between scalability, cost, performance, and control. With the system we will introduce in this paper, we can run performant Python, while remaining entirely in the browser, and enabling synchronous user interrupts using WebAssembly. This approach offers a new opportunity to provide a more full, native, and accessible experience to students beginning their computer science journeys.

1.2 Main Contributions
In this paper, we present (1) a novel, open source, system designed for running student written Python, entirely in the browser, (2) a sample IDE platform that displays the functionality (accessible at https://ide.stanford.edu), (3) the outcomes of using our system in several CS1 courses that collectively reached over 10,000 students and teachers from 150 countries, and (4) an alternative open source implementation which uses web-workers, that may be a useful approach in case of future changes to the web ecosystem. Our methodology utilizes a Python-to-WebAssembly port library in conjunction with custom package to enable a wide variety of features. These features include:

- High fidelity interactive graphics and interactive mouse / touch input using a custom Python library.
- A line by line ‘replay’ debugger that is compatible with the graphics library.
- Reactive program interruption and synchronous input while running Python in the browser’s main thread.
- Entirely in the browser (serverless) interpretation and compilation to WebAssembly.
- Access to a broad set Python libraries written in C or CPython, including Numpy [8], Pandas [9], and Scipy [12].
- Shareable programs at no server runtime cost.
- A file system that allows for storing diverse file types such as images, text files, and CSVs.

2 RELATED WORK
Currently, many web based IDEs tailored to Computer Science Education exist for public and in course use [23, 34] and many others have been developed for research and private use[32, 38]. A popular example includes the Scratch[27] environment introduced by Maloney et al., which provides a specialized language and interface built specifically for learning programming. While Scratch is a major educational IDE, it, like many other options, doesn’t allow for programming in Python. Additionally, we also do not focus on native educational Python IDEs [16, 25] as these require at least some download and setup. There are also several studies on web based platforms built to run a plethora of languages for different University courses [17, 19, 36, 37]. These studies find that using a web based environment can have positive effects like decreasing dropout rates.

Within educational Python focused IDEs, there are several tools, and IDEs that operate in the browser, either through transpilation or server side execution, including Code Skulptor (used by Coursera) [4], and Strype [18] among others [20, 30, 35]. Some of these implementations may solve some of the problems faced by web Python IDEs, but they each face the natural issues that transpilation or server based systems provide that limit the scope of their functionality.

We will focus on a subset of popular, interactive, browser based Python IDEs that are open to the public, used for education, and have very easy interfaces and libraries that students can quickly grasp the basics of. The most similar to ours include the Carnegie Mellon (CMU) CS Academy IDE [33], Repl.it.com [11], and Online Python Tutor [21]. Each of these solutions include either server-side execution, or transpilation to Javascript.

The Repl.it platform offers a diverse set of languages to utilize. Their Python service runs user programs on a server-side back-end. When running an interactive program that requires user input, such as live mouse location data for example, the application utilizes web channels such as web sockets to convey the data. The platform also caps the compute power of free accounts, and requires a paid account for better performance.

The CMU CS Academy IDE, created by Stehlik et al.[33], may be the most similar work to ours. Their platform offers a fully in browser approach. The application transpiles a students code to Javascript using a version of Brython[2], then runs the code directly in the browser via a web worker. Their IDE supports a set of custom libraries including one that displays graphics. They use an event based system, where the user can provide an onStep function, to provide smooth, interactive, graphics. Their solution for synchronous pausing includes custom sleep function that makes a call to a server to block the programs execution. Additionally, their IDE does not support an input function. This description is based on the public documentation and playground editor. The CMU CS Academy’s goal is to offer several CS curriculum to students and teachers in grades K-12.

Another important prior work includes Philip Guo’s Online Python Tutor [21] platform, which offers users the ability to see each step of execution of their program. The user writes a program, runs it, then gets access to the output, along with a set of frames that they can step through. The user’s program is run on a server and the resultant data is sent back to the user to be viewed. At each point they can view what line was last executed, what is next to be executed, and the current state of the variable stack. The program
Figure 1: PyodideU IDE Screenshot. Left, the IDE running a program, showing the file system with multiple file type support. Right, the IDE in "Replay" debug mode showing scope variables, highlighted current line, graphics and mouse handling.

does allow for user input, but restricts the set of libraries that can be used.

3 SYSTEM DESCRIPTION

In this section, we briefly describe the technical details of how our system works, along with its current implementation in the public web based IDE, and any associated costs.

3.1 PyodideU

3.1.1 Main Thread. Our main system utilizes a combination of Pyodide [31] along with a custom CPython package built off of the Unthrow package, by Joe Marshall [28] to allow for synchronous pausing of the users program, along with access to info about the user’s program while running. All Python is run using a Pyodide client to compile to WebAssembly. The program starts by running setup code, including the user’s program as a nested function. Then, we run the users code within a try-except block, periodically throwing "resumable" exceptions via a custom trace function. When an exception is thrown, the user’s code is exited. The current stack frame of the users code at the time the trace function throws the exception is rolled up and saved as a class variable. At this point, we transition from WebAssembly to Javascript, where we can handle synchronous events like input or interrupts. If no interrupts have occurred, we reenter the users program by running a script that reinstates the stack. We can continue this process using callbacks until the program ends either via finishing successfully, throwing a (non-trace) exception, or being interrupted.

We also use the trace function to save information for the ‘Replay’ debug mode. In this mode, we collect the user’s scope, variable stack, any visual component, and output text state at the execution of every line. With this, we can offer a step by step debugging mode, throwing a (non_trace) exception, or being interrupted.

To implement libraries with a visual component, we created Python packages that would make direct calls to a Javascript client (through WebAssembly callbacks) to update the state of the visual component during the program.

3.1.2 Async Webworker. In addition to our main solution, which requires Pyodide and a customized build of Unthrow [28], we have also created an alternate solution that can be run using the stock Pyodide implementation. This solution uses Python’s ast (Abstract Syntax Tree) module to modify the code to work in an asynchronous environment. It does this by making all user functions and interactive library functions (e.g., Python’s input and time.sleep functions) async and all calls to those functions are modified to use await. Because Pyodide natively supports async and await using the asyncio library[1], this solution solves the WebAssembly blocking issue.

Once the user code has been translated, the Pyodide runtime executes the code such that the interactive functions work as expected. As far as the user is concerned, the code runs as originally written, and when errors occur, the user’s unmodified code is shown to the user in the stack trace.

Both of the above listed methods will be open sourced in library form for anyone to freely use at https://github.com/CodeinPlace/PyodideU.

3.2 Implementation

We use the first (main thread) system to run user code in several places on an online platform built for CS1 courses. Primarily, we built a custom IDE to enable all of the functionality. Within the IDE, there is a terminal pane at the bottom, where the user can run code, see output, send input, and use a Python REPL. On the right side, there is a collapsible pane which can contain unit test previews, a canvas, or other important graphical displays. On the left side, there is a multi-page tool bar, that contains documentation, file information, and assignment instructions. There is also a center pane which shows the currently open file. When the user is in ‘Replay mode’, their variable stack appears in the left pane, the programs output (up to their current step) appears in the terminal, and, if necessary, the current graphics state appears in their right pane. Finally, at the top of the page there is a start/stop button, along with a share button that publishes programs (see figure 1 for details).

In addition to the IDE, we use our runtime system to run code in "Published" pages, which allow users to share the output of custom programs as well as in the forum where users can embed
code in posts. The implementations used for the courses utilized a Firestore database to store all files (including Python, images, text, etc.), however we will release a version that allows for local storing so that no remote database is needed.

3.3 Associated Costs
Using our public IDE or the library associated with it will be completely free. If you choose to use the library in a platform, and need to store user code, solutions are very cost effective. The system we used ran at a rate of 150,000,000 database reads of user code per dollar [13].

4 RESULTS
In our results section, we will cover the usage of our custom IDE and Python system, a comparison of the performance of our features to other, current alternatives, and the responses from students and teachers of the courses that the tools were used in.

4.1 Usage
Our PyodideU execution system, along with the IDE implementation presented in this paper have been used as the primary programming environment across five classes, including Stanford University’s CS106A: Programming Methodology, Code in Place, an online CS1 course with over 9,000 students, and two international CS1 courses including one taught in Spanish [7]. These courses have resulted in a combined 10,000 students and teachers in 150 different countries using the software for assignments, practice, teaching, and personal projects. In total, over 350,000 unique projects have been created to date (including projects made by non-students or teachers), and over 13,000 programs have been published using our sharing functionality. During Code in Place, we registered over 4.1 million programs in the IDE. The course contained several programming assignments written and unit tested with the PyodideU system over six total weeks. The quantity of sustained use throughout several course terms indicates the robustness of the application and the underlying Python to WebAssembly library. Additionally, the global use of the IDE indicates that it is a viable option for creating widely accessible courses, and educational experiences.

With the introduction of our system, we were able to offer custom graphics libraries, in a cost effective and accessible manner. If we had remained on a server based model, and introduced all of the additional functionality our IDE offered, we estimate that our server costs would have ballooned by many thousands of dollars based on the amount of time code was run. Having this quantity, and diversity of users also allowed us to see quickly both the opportunities and limitations of the system we had built. One great example is shown in Figure 2, a project a student published that depicted a 3-D maze game they had made using complex ray-casting algorithms. The program performs surprisingly well, displaying the 3-D world and allowing the player to move at a moderate speed.

4.2 Feature Comparison
In this section, we offer a comparative analysis of the features included in our systems versus the current alternatives that are publicly available today. We will also discuss bench-marking tests we conducted that assert the competitiveness of our systems, in the context of graphical programs and animation.

We compare our two systems (described in section 3), to five additional, currently available online systems. These five include the Pyodide Online Demo[10], the CMU Academy IDE[33], the Brython Online Demo[2], Python Tutor[21], and Replit[11]. Pyodide, as we have discussed, is a leading Python to WebAssembly library, while Brython is a leading Python to Javascript transpiler. It is also possible that any one of these programs offers more functionality in a non public setting, however, we are basing these comparisons on only what is offered publicly.

4.2.1 Serverless Program Execution. Each of our two provided solutions have the ability to run completely offline. The only requirement, if the user is accessing the IDE from the web, is that they can load the web page. Alternatively, our methods could be packaged and run as a local application. Transpiler solutions, such as the CMU Academy IDE, and an unaltered version of Brython could accomplish the same feat, but platforms like Replit and Python Tutor require continued communication with the Server.

4.2.2 Interactive Graphics. Aside from our implementations, CMU Academy, and Replit each offer comparable, easy to grasp, interactive graphics libraries. Because Replit does not rely on running code in the browser, they offer access to Tkinter while our implementation and CMU’s require custom graphics libraries compatible with the HTML Canvas. Python Tutor, along with unaltered Brython and Pyodide do not offer a direct graphics library. Both Brython and Pyodide allow you to call Canvas editing functions. However, due to the blocking nature of the two frameworks, creating animations can be difficult.

4.2.3 Synchronous Interrupts (Allow for User Termination). Replit, the CMU Academy IDE, and both of our systems allow the user to trigger an interrupt during a program. Replit communicates with the server to terminate the program, while CMU and our PyodideU systems stop the code directly in the browser.
with the main thread, affecting things like graphics. This issue also
worker, the WebAssembly module must load which takes around
will hopefully make it easy to embed interactive Python code into
by running in the main
4.2.7 Non-blocking Input function
solutions that require tradeoffs in program performance.

4.2.5 File System. In our IDE, we used the built in Emscripten
[5] file system to interact with Python, and other text based file
types. We also built in additional functionality for our graphics
library to handle images. While libraries like Pyodide and Brython
do offer FileSystem support, their online demos do not. The same is
for Python Tutor and the CMU Academy IDE. Replit, on the other
hand, does allow for a diverse file system which is managed in their
database system.

4.2.6 Main Thread Execution Without Crashing. To the best
of our knowledge, our PyodideU is the only entirely in browser
Python to WebAssembly System that can run on the main thread
without fear of crashing on infinite loops. By running in the main
thread, we have created a system with less configurations, that
will hopefully make it easy to embed interactive Python code into
a web page. In addition to convenience, running code in a main
thread has performance benefits. When using Pyodide in a web
worker, the WebAssembly module must load which takes around
2-5 seconds, and there is additional latency when communicating
with the main thread, affecting things like graphics. This issue also
occurs in transpilation based systems for which there exist some
solutions that require tradeoffs in program performance.

4.2.7 Non-blocking Input function. Blocking occurs when the
browser prevents the users access to all parts of a page except for
a single component. This occurs with native Javascript window
functions like "prompt" and "alert". While both Brython and Pyodide
have input functionality, they use the "prompt" function which
pauses all other code execution. Replit and Python Tutor have non-
blocking input functionality, due to the fact that they communicate
the input via server communication. In both of the systems we
have developed, the user can use Python’s native input function
without blocking interaction with other aspects of their window.

4.2.8 Access to C dependent Libraries. Using Pyodide, we can
offer users access to a selection of packages written partially or
completely in C or CPython without having to manually translate
the library. This is a feature that is distinctly unique from tran-
spilation systems, as there is not currently a robust solution for
transpiling C/CPython into vanilla Javascript. Using Emscripten
[5], however, we can compile the packages to be WebAssembly
compatible. While not all packages currently operate perfectly in
WebAssembly (especially packages with visual, and synchronous
components), this drastically increases the scope of usable libraries
in the browser. Server based systems naturally have the ability to
support most Python libraries.

4.2.9 Benchmarking. To test performance of our general graph-
ics execution speed versus the already available solutions, we wrote
a graphics animation program that scales up in the amount of ob-
jects it tracks on the canvas. The program simulates a collection
of balls bouncing off of the wall. We compared main thread PyodideU,
without debug collection, Replit, the CMU CS Academy IDE, and
native Python and collected the average duration of rendering a
set of circles over 500 renders. We scaled the amount of circles
from 100 to 500 to simulate a fairly complex “bouncing balls” pro-
gram. It is important to note that these programs were written in
slightly different ways to match the specific libraries they used.
Each was meant to be as efficient as possible while still generating
the animation.

The results (Figure 4) show that our solution offers strong perfor-
mance and remains within only several milliseconds per frame of the
alternatives. While it was not as fast as Native Python and Replit,
it tended to be similar or moderately better than transpilation based
systems as the amount of shapes scaled up. It is also important to
note that this graph does not capture any buffering or slowness
that occurred in Replit’s execution due to web web communication.
This speaks to the performance gain that WebAssembly offers over
traditional methods of browser based Python.

Also note that this test was run using the Chrome Browser (for
non-native programs) on a Macbook Pro with an M2 Pro Chip.

4.3 User Reactions
Across the classes, both students and teachers had positive re-
sponses to the IDE, and the tools within it. In Code in Place, many
students posted on the internal forums asking if they would have
continued access to the IDE after the course was complete. Ad-
ditionally, teachers were fans of the non-existent learning curve,
staking on the teachers-only forum “The IDE looks modern and is
easy to use for students and section leaders”. They also suggested
that it was a major improvement from the last time the course was
held stating: “I was thrilled to see the new tools & the updated
IDE.” Overall, reactions were overwhelmingly positive. In an in
person course taught in Colombia, when given the choice between

Figure 3: A matrix breaking down popular, publicly available,
in browser Python editor environments by features. Note Py-
odideU solutions (shaded blue) are introduced in this paper.
teaching through our PyodideU IDE or Pycharm [3] which was preinstalled on all school computers, educators chose to teach through our IDE. They stated that it was easy for students to start writing code immediately, was extremely useful for showing demos, and had powerful tools that were in an intuitive environment.

5 PRACTICAL USE AND POSSIBILITIES
There are a number of benefits to a browser-based Python runtime environment, which we will cover in this section.

5.1 Minimization of Server Costs and Security Issues, and Offline Use
Online IDEs that rely on server-side program execution have to contend with the fact that the server can use considerable resources to run user code. This does not scale for large numbers of users, as the ability to provide high performance for user programs often means that the number of servers must scale linearly with the number of users, and can quickly become expensive. Furthermore, running user code on a server requires a significant amount of security, and it is virtually impossible to plan for all methods of potential server attacks.

Using our PyodideU powered IDE, all user code runs in the user’s browser, and therefore there are zero server costs for actual code execution. This also allows for inexpensive sharing of code, where just the code text is provided, along with a minimal runtime interface. Furthermore, the security issues are minimized (to the point of being virtually nonexistent), as the server never runs any untrusted code.

An added benefit provided by a browser-based runtime is that once the website itself is loaded, users are able to disconnect from the Internet to write and run their code. This could be useful for students who are charged for data, or who have limited bandwidth connections (e.g., in a developing country). It is also possible to package the IDE onto physical media such as a flash drive, so that it can be used without any Internet access (but still with a browser).

5.2 Interactivity
Because the Pyodide runtime is able to interface directly with the Javascript and the browser’s Document Object Model (DOM), any functionality that is available through Javascript is also available through Python. As described in subsection 4.2.2, we have written a graphics library that interfaces with an HTML5 Canvas, and we have written mouse-handling routines, touch routines, keyboard input routines, and sleep routines that enable running traditional interactive programs with excellent performance, particularly in an educational setting. The IDE also has the ability to highlight lines of code in real time, aiding in teaching about program flow. Python-to-WebAssembly allows for directly interfacing with the DOM, as well, meaning that access to HTML buttons and input elements (e.g., checkboxes, drop-down lists, etc.) is possible.

As mentioned in the previous subsection, the ability to share code is significant. It has traditionally been difficult to share interactive, graphical Python programs across platforms. With a web-based solution, students can share their code with friends, relatives, and the general public, and from a pedagogical standpoint, sharing “look-what-I-did!” code can be meaningful and inspiring to beginners.

5.3 Browser WebAssembly Unibiquity and Mobile Device Instruction
WebAssembly is a strongly supported technology in today’s browsers [15], including both desktop and mobile platforms, and nontraditional browser-based operating systems, such as ChromeOS. We have written a version of our IDE to work on tablet and smartphone devices, as well. Indeed, it is possible for a student to learn to code on our IDE completely from a smartphone, including entering and debugging code. This could have significant positive impact for students whose only access to the Internet is through a smartphone or tablet. The addition of an inexpensive Bluetooth keyboard would provide an even better user interface.

6 LIMITATIONS AND FUTURE WORK
While we are encouraged by the quick success of the systems and IDE we have put together, we recognize that there are quite a few areas for expansion and improvement. First, while we have collected a large amount of data on the usage of the IDE, and a many positive, but anecdotal, reactions, we would like to do a more intensive study regarding the preferences of students and teachers.

Further, while we feel this Web Python library and IDE does offer an experience similar to local development, it was not without bugs and limitations. Our system that runs exclusively in the main thread currently requires a trace function even when not in ‘Replay’ mode which negatively impacts a programs runtime. While the speed of the program is still very competitive, and in many cases, superior to alternative options, it could be faster. Further, currently, we run a version of Python 3.9, however, with a package upgrade, we could convert to Pyodide’s newer versions which stay up to date with the most recent Python Update.

ACKNOWLEDGMENTS
This work would not have been possible without the help of the Carina Foundation and the entire Code in Place team. Thank you to Joe Tey who was instrumental in developing much of the Code In Place IDE interface. In addition, thank you to the Tym.so team for aiding the development of the initial prototypes.
REFERENCES


